Universidade Tiradentes
NOME DO CURSO

Gladiston Teles de Meneses Filho
Gabriel Moura Feitosa de Melo
Guilherme Araújo Chaves

Sistema de Monitoramento de Tráfego Urbano

Aracaju - SE
ANO

Gladiston Teles de Meneses Filho
Gabriel Moura Feitosa de Melo
Guilherme Araújo Chaves

Sistema de Monitoramento de Tráfego Urbano

ATIVIDADE sobre xxxx apresentado como requisito parcial da avaliação da disciplina F115363 - Compiladores - E01, ministrada pela Prof. Layse Santos, no 2º semestre de 2025.

Aracaju - SE
2025

Sumário

1 INTRODUÇÃO

Nas grandes cidades, o tráfego de veículos representa um dos principais desafios da gestão urbana. Congestionamentos, acidentes e atrasos em rotas de emergência afetam diretamente a mobilidade e a qualidade de vida da população.

Com o avanço das tecnologias de sensoriamento e automação, tornou-se possível coletar dados em tempo real sobre o fluxo de veículos, detecção de acidentes e presença de veículos prioritários. No entanto, o uso eficiente dessas informações depende da existência de um sistema capaz de interpretar e reagir automaticamente a esses eventos.

Nesse contexto, surge o **TrafficLang**, uma linguagem de domínio específico (DSL — *Domain Specific Language*) desenvolvida para expressar de forma estruturada e legível regras de controle de tráfego urbano. A linguagem foi projetada para permitir a definição de comportamentos automáticos de semáforos, rotas e sensores, promovendo maior previsibilidade e eficiência no trânsito.

O presente trabalho descreve o desenvolvimento completo do compilador **TrafficLang**, implementado em Python e estruturado conforme as etapas clássicas de compilação: análise léxica, análise sintática, análise semântica, geração de código intermediário e execução/simulação.

O objetivo é demonstrar o funcionamento de um compilador modular, orientado a objetos e aplicável a um contexto real de automação urbana, servindo como base para futuras extensões, como integração com sistemas de IoT e controle distribuído de tráfego.

2 JUSTIFICATIVA

O desenvolvimento da linguagem **TrafficLang** surge da necessidade de expressar, de forma clara e estruturada, regras de controle de tráfego que possam ser processadas automaticamente por sistemas inteligentes. Nas grandes cidades, o gerenciamento do tráfego depende cada vez mais de dados coletados por sensores urbanos, como medidores de fluxo de veículos, detectores de acidentes e sensores de emergência. A existência de uma linguagem voltada a esse domínio permite a integração eficiente entre os dispositivos de coleta de dados e os sistemas de controle, otimizando a mobilidade e reduzindo ocorrências de congestionamentos.

A criação do compilador **TrafficLang** se justifica pela importância de demonstrar, de forma prática, como os conceitos de compiladores podem ser aplicados a problemas reais. Por meio de sua arquitetura modular — composta por analisador léxico, sintático, semântico, gerador de código intermediário e simulador —, o projeto possibilita compreender o fluxo completo de tradução de uma linguagem até sua execução final.

O projeto também se alinha às metas de aplicação de linguagens de domínio específico (DSLs) no contexto da Engenharia de Software e de Sistemas Inteligentes. Essas linguagens favorecem a expressividade, a manutenibilidade e a precisão de regras de automação, especialmente em cenários complexos como o tráfego urbano.

O desenvolvimento do compilador foi planejado em etapas semanais, visando à entrega gradual e à validação contínua de cada fase:

- **Semana 01 — Análise Semântica e Tabela de Símbolos:** Implementação do analisador semântico responsável por identificar variáveis, tipos e escopos. Nesta etapa, foi criada uma tabela de símbolos para armazenar os elementos da linguagem (*sensores*, *semáforos*, *rotas* e *condições*), permitindo a validação semântica e a detecção de erros de declaração e uso.

- **Semana 02 — Geração de Código Intermediário:** Desenvolvimento da tradução das regras **TrafficLang** para uma representação intermediária (IR), estruturada em formato JSON. Essa etapa foi essencial para permitir a visualização e depuração das regras processadas, garantindo a correta conversão entre a sintaxe da linguagem e sua representação executável.

- **Semana 03 — Execução e Simulação das Regras:** Implementação do módulo de execução, responsável por interpretar o código intermediário e simular o comportamento das regras de tráfego. Essa fase consolidou todas as anteriores, permitindo a observação do funcionamento completo do compilador, desde a análise léxica até a execução das ações de controle de semáforos e rotas.

A conclusão das três primeiras etapas representa um marco fundamental no desenvolvimento do compilador, garantindo sua funcionalidade básica e servindo como base sólida para a documentação e aprimoramentos futuros.

3 OBJETIVOS

3.1 Objetivo Geral

Desenvolver um compilador completo para a linguagem de domínio específico **TrafficLang**, capaz de interpretar e executar regras de controle de tráfego urbano com base em sensores e sistemas inteligentes. O compilador deve permitir a análise, validação e simulação de comportamentos como mudança de estados de semáforos e priorização de rotas, contribuindo para a automação e eficiência da mobilidade em centros urbanos.

3.2 Objetivos Específicos

1. Implementar um **analisador léxico** que identifique corretamente os tokens da linguagem, incluindo palavras-chave, operadores, identificadores e comentários;
2. Desenvolver um **analisador sintático** baseado em descida recursiva, capaz de validar a estrutura das regras definidas em **TrafficLang**;
3. Criar um **analisador semântico** que verifique a consistência dos elementos da linguagem, utilizando uma tabela de símbolos para controle de tipos e identificadores;
4. Implementar a **geração de código intermediário** em formato JSON, representando as ações e condições de forma estruturada e independente da execução;
5. Desenvolver um **módulo de execução e simulação** capaz de interpretar o código intermediário e demonstrar o comportamento das regras de tráfego, como a abertura e fechamento de semáforos ou a ativação de rotas prioritárias;
6. Estruturar o compilador de forma **modular e orientada a objetos**, favorecendo a manutenção, a clareza e a expansão futura do sistema.

4 METODOLOGIA

A metodologia adotada para o desenvolvimento do compilador **TrafficLang** foi baseada em uma abordagem incremental e orientada a objetos, garantindo modularidade, clareza e fácil manutenção. Cada etapa do processo de compilação foi tratada como um módulo independente, permitindo testes isolados e integração progressiva das funcionalidades.

O projeto foi dividido em quatro fases principais, seguindo o planejamento proposto para as semanas de implementação:

4.1 Fase 1 — Análise Léxica e Sintática

Na primeira fase, foram implementados os módulos **Lexer** e **Parser**. O **Lexer** é responsável por percorrer o código-fonte e gerar uma lista de tokens, reconhecendo palavras-chave, identificadores, operadores, números e delimitadores. O **Parser**, por sua vez, aplica uma estratégia de *descida recursiva* para validar a estrutura gramatical das instruções e construir a árvore sintática abstrata (AST), que servirá de base para as etapas seguintes.

4.2 Fase 2 — Análise Semântica

A segunda fase consistiu na criação do módulo **SemanticAnalyzer**, responsável por verificar a coerência lógica do código. Essa etapa implementa uma **tabela de símbolos** que armazena informações sobre os elementos da linguagem, como sensores, semáforos, rotas e variáveis. Com base nessas informações, o compilador é capaz de detectar erros como declarações duplicadas, uso de variáveis não declaradas ou inconsistências de tipos.

4.3 Fase 3 — Geração de Código Intermediário

A terceira fase foi dedicada à implementação do módulo **CodeGenerator**, encarregado de traduzir a AST em uma representação intermediária (IR) em formato JSON. Esse formato foi escolhido por sua legibilidade e facilidade de integração com outros sistemas. A estrutura intermediária permite a visualização das regras processadas, facilitando o entendimento e depuração do comportamento do compilador.

4.4 Fase 4 — Execução e Simulação

Por fim, a quarta fase consistiu na criação do módulo **Simulator**, o qual interpreta o código intermediário gerado e executa as ações simuladas. Durante a simulação, o compilador avalia condições lógicas envolvendo sensores e aplica as ações correspondentes, como alterar o estado de um semáforo ou ativar uma rota de emergência. Os resultados são exibidos no console, demonstrando o funcionamento prático das regras definidas em **TrafficLang**.

4.5 Ferramentas Utilizadas

O desenvolvimento foi realizado em **Python**, devido à sua simplicidade e ampla disponibilidade de bibliotecas. A documentação técnica e científica foi elaborada em LaTeX, seguindo o modelo ABNT, e o código foi testado em ambiente **Google Colab** e **VS Code**. As representações intermediárias e simulações foram validadas com exemplos extraídos e adaptados do documento de referência do projeto.

4.6 Estrutura Modular

A modularização do compilador foi planejada para garantir independência entre as etapas. Cada componente principal (*Lexer*, *Parser*, *SemanticAnalyzer*, *CodeGenerator*, *Simulator*) foi implementado como uma classe isolada, seguindo os princípios de encapsulamento e responsabilidade única. Essa estrutura facilita a substituição ou expansão de partes do sistema, permitindo, por exemplo, a adição de novos tipos de sensores ou condições lógicas mais complexas (como operadores AND e OR).

4.7 Integração das Fases

Após o desenvolvimento de cada módulo, as etapas foram integradas para formar o fluxo completo do compilador. A sequência de execução do sistema segue o ciclo:

$$\text{Fonte} \rightarrow \text{Lexer} \rightarrow \text{Parser} \rightarrow \text{SemanticAnalyzer} \rightarrow \text{CodeGenerator} \rightarrow \text{Simulator}$$

Esse fluxo garante que o código fonte seja analisado, validado, convertido em uma representação intermediária e, por fim, executado de forma simulada, permitindo verificar o funcionamento das regras de tráfego em diferentes cenários.

5 RESULTADOS E DISCUSSÕES

AAAAA

6 CONSIDERAÇÕES FINAIS

AAAAAAA

1. aaaaaa;
2. aaaaaa;
3. aaaaaa;
4. aaaaaa.

7 REFERÊNCIAS

8 ANEXOS

Figuras 1 xxxx

Figura 1 – LEGENDA