Universidade Tiradentes
Ciência da Computação

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 moderna. Congestionamentos, acidentes, atrasos em rotas de emergência e eventos inesperados impactam diretamente a mobilidade e a qualidade de vida da população.

Com os avanços em sistemas inteligentes e na Internet das Coisas (IoT), tornou-se possível coletar dados em tempo real por meio de sensores urbanos, como detectores de fluxo, câmeras, dispositivos de segurança viária e sensores de prioridade. Entretanto, a utilização efetiva dessas informações depende da existência de um mecanismo capaz de interpretar e reagir automaticamente aos eventos monitorados.

Nesse contexto, surge a *TrafficLang*, uma linguagem de domínio específico (*Domain-Specific Language* — DSL) desenvolvida para expressar de forma clara, estruturada e interpretável regras de controle de tráfego. A linguagem foi projetada para permitir a definição de comportamentos automáticos envolvendo sensores, semáforos, rotas e condições lógicas.

O presente trabalho descreve o desenvolvimento completo do compilador da TrafficLang, implementado em Python e estruturado de acordo com as etapas clássicas de um compilador: análise léxica, análise sintática, análise semântica, geração de código intermediário (IR) e execução/simulação.

O compilador foi construído de forma modular, integrando recursos como suporte a operações lógicas (`AND`, `OR`), interpretação de sensores carregados via arquivo JSON, geração automática de código intermediário em formato JSON e execução das regras em um simulador integrado. O sistema serve como um demonstrativo prático da aplicação de conceitos centrais de compiladores em um problema real de automação urbana, fornecendo uma base sólida para futuras extensões, como integração com sistemas inteligentes distribuídos e controle adaptativo 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 da linguagem *TrafficLang* seguiu o modelo clássico de construção de compiladores, dividido em etapas que vão desde a análise léxica até a execução das regras simuladas. Cada módulo foi implementado em Python, contemplando princípios fundamentais da disciplina e integrando-os em um ambiente funcional e interativo. A seguir, são detalhadas as fases da metodologia aplicada.

4.1 Análise Léxica

A análise léxica foi realizada por meio do módulo `Lexer`, responsável por converter o código-fonte em uma sequência de tokens. Foram definidos tipos de tokens como: identificadores, números, strings, operadores relacionais, operadores lógicos (`AND`, `OR`) e delimitadores.

O lexer também trata comentários (iniciados com `#`) e realiza a normalização de literais booleanos, convertendo expressões como `true`, `True` e `TRUE` para um valor padrão. Além disso, erros de caracteres inválidos são capturados e relatados ao usuário com indicação de linha e coluna.

4.2 Análise Sintática

A segunda etapa consistiu na construção do `Parser`, implementado utilizando descida recursiva. Ele é responsável por reconhecer estruturas da linguagem, como:

- instruções condicionais com `IF` e blocos;
- atribuições utilizando `SET`;
- ações associadas a semáforos;
- definição de prioridades de rotas com condições;
- expressões com precedência entre operadores lógicos e relacionais.

O parser converte o fluxo de tokens em uma *árvore sintática abstrata* (AST), representada por classes específicas para cada tipo de expressão e comando.

4.3 Análise Semântica

A etapa semântica foi implementada no módulo `SemanticAnalyzer`, ampliado para suportar:

- **tabela de símbolos** para variáveis, sensores, semáforos e rotas;
- inferência de tipos (número, booleano ou string);
- verificação de coerência em comparações e operadores lógicos;
- reconhecimento de sensores carregados via JSON.

Cada símbolo é registrado na `SymbolTable`, permitindo rastrear seu tipo e utilização ao longo do programa. A análise semântica identifica incoerências como comparações de tipos incompatíveis, operadores aplicados incorretamente ou uso de variáveis não declaradas.

4.4 Geração de Código Intermediário (IR)

A representação intermediária (IR) foi projetada em formato JSON, tornando possível sua visualização, exportação e integração com outras ferramentas. Cada comando da AST é convertido em uma estrutura que descreve:

- o tipo da operação;
- os argumentos e valores envolvidos;
- expressões condicionais estruturadas;
- variáveis ou sensores utilizados.

O compilador salva automaticamente o IR no diretório `ir_output`, permitindo a persistência da lógica trafegada para análises externas e depuração.

4.5 Execução e Simulação

A etapa final consiste no módulo `Simulator`, que interpreta o IR e simula o comportamento das regras definidas pelo usuário. O simulador processa:

- atualização de semáforos;
- configuração de rotas com prioridade;
- ações como `MOVE` e `STOP`;

- atribuição de variáveis com `SET`;
- avaliação lógica e relacional de expressões com sensores reais.

O ambiente final exibe logs detalhados de execução e os estados resultantes de semáforos, rotas e variáveis.

4.6 Menu Interativo e Testes

Foi implementado um menu interativo em linha de comando permitindo:

- entrada manual de código TrafficLang;
- carregamento de sensores via arquivo JSON;
- compilação completa (tokens, AST, semântica e IR);
- execução da simulação;
- processamento em lote de até 50 testes automáticos.

Além disso, foram elaborados exemplos de validação, cobrindo desde ações simples de semáforos até estruturas condicionais complexas.

5 RESULTADOS E DISCUSSÕES

Os resultados obtidos com o desenvolvimento do compilador da linguagem *TrafficLang* demonstram a eficácia da solução proposta e a aderência às etapas clássicas de construção de compiladores. O sistema foi testado com múltiplos cenários, incluindo regras simples, condições compostas, uso combinado de sensores e manipulação de semáforos e rotas, o que permitiu validar sua robustez e flexibilidade.

5.1 Funcionamento Geral do Compilador

Ao final do processo, o compilador mostrou-se capaz de executar integralmente o fluxo:

1. análise léxica com identificação de tokens válidos e rejeição de caracteres inválidos;
2. análise sintática com construção de árvores sintáticas compatíveis com estruturas de controle, blocos e ações do domínio do tráfego;
3. análise semântica utilizando tabela de símbolos, permitindo registro automático de variáveis, sensores e elementos do domínio (semáforos e rotas);
4. geração de código intermediário (*Intermediate Representation* — IR) em formato JSON;
5. execução simulada de regras com avaliação de condições, atualização de estados e emissão de logs detalhados.

5.2 Testes com Sensores Reais e Regras Complexas

Sensores carregados via arquivos JSON permitiram testar cenários dinâmicos, como:

- variação no fluxo de veículos;
- presença de emergências;
- mudanças bruscas em parâmetros ambientais.

Regras contendo operadores lógicos `AND` e `OR` foram interpretadas corretamente, demonstrando que o compilador é capaz de modelar logicamente cenários urbanos mais complexos.

5.3 Simulação de Semáforos e Rotas

O módulo de simulação apresentou resultados consistentes ao:

- acionar semáforos com estados *abrir*, *fechar* e *intermitente*;
- atribuir prioridades de rotas com base em condições avaliadas em tempo de execução;
- registrar logs detalhados sobre cada decisão tomada pelo simulador.

Ao final de cada simulação, o sistema retornou um relatório contendo:

- estado final dos semáforos;
- rotas priorizadas;
- valores finais das variáveis internas;
- histórico completo de ações executadas.

5.4 Exportação do Código Intermediário

Um dos resultados mais relevantes foi a geração automática do IR em formato JSON. Essa funcionalidade permite:

- auditoria e rastreamento das regras;
- integração com outras ferramentas externas;
- armazenamento e reuso de regras já compiladas.

O IR é salvo em diretório próprio, facilitando organização e análise posterior.

5.5 Desempenho e Estabilidade

Durante os testes, o compilador foi capaz de processar até 50 regras em lote, mantendo desempenho estável e boa responsividade. Nenhum erro crítico ou falha de execução foi identificado após a validação final.

Esses resultados reforçam a viabilidade da *TrafficLang* como DSL aplicada à gestão inteligente do tráfego urbano.

6 CONSIDERAÇÕES FINAIS

O desenvolvimento do compilador da linguagem *TrafficLang* permitiu demonstrar, na prática, a aplicação dos principais conceitos de compiladores em um contexto real e relevante: o controle inteligente do tráfego urbano. Todas as etapas fundamentais — análise léxica, sintática, semântica, geração de código intermediário e execução — foram implementadas com sucesso e integradas em um ambiente funcional e interativo.

A criação de uma DSL específica para regras de tráfego mostrou-se uma abordagem eficaz para simplificar o processo de definição e execução de comportamentos automáticos. Ao permitir que regras sejam escritas em alto nível, a *TrafficLang* reduz a complexidade operacional e facilita a adaptação do sistema a cenários reais.

A adição de funcionalidades, como a leitura de sensores via JSON, o suporte a operadores lógicos `AND` e `OR`, a geração automática de IR e a simulação com registro de ações, ampliou significativamente as capacidades do compilador, tornando-o uma base sólida para futuras evoluções.

Conclui-se que o projeto atingiu plenamente seus objetivos pedagógicos e funcionais, fornecendo:

- uma linguagem clara e expressiva para modelagem de regras de tráfego;
- um compilador completo e modular;
- um simulador fiel ao domínio do problema;
- um ambiente interativo que facilita testes e validações.

Como trabalhos futuros, destacam-se possíveis melhorias, tais como:

- integração com sistemas reais de sensoriamento urbano;
- visualização gráfica dos estados simulados;
- otimização do IR para execução distribuída;
- extensão da linguagem para abranger novos atores do sistema de mobilidade.

Dessa forma, o compilador *TrafficLang* não apenas cumpre seu propósito acadêmico, mas também constitui um alicerce promissor para aplicações avançadas em cidades inteligentes.

7 REFERÊNCIAS

7.1 REFERÊNCIAS

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. *Compiladores: Princípios, Técnicas e Ferramentas*. 2. ed. São Paulo: Pearson, 2008.

ALVES, E.; GONÇALVES, R. *Teoria de Linguagens e Compiladores*. 3. ed. Rio de Janeiro: LTC, 2019.

GRUNE, D.; BAL, H. E.; JACOBS, C.; LANGENDOEN, K. *Modern Compiler Design*. 2. ed. New York: Springer, 2012.

SEBESTA, R. W. *Conceitos de Linguagens de Programação*. 10. ed. Porto Alegre: Bookman, 2018.

PARR, T. *The Definitive ANTLR 4 Reference*. 2. ed. Dallas: Pragmatic Bookshelf, 2013.

FOWLER, M. *Domain-Specific Languages*. 1. ed. Boston: Addison-Wesley, 2010.

LEAL, M. C.; BRAGA, J. *Linguagens Formais, Autômatos e Computabilidade*. 2. ed. Rio de Janeiro: LTC, 2020.

TANENBAUM, A. S. *Structured Computer Organization*. 6. ed. Pearson, 2013.

BRAGA, R.; BATISTA, T. *DSLs na Prática: Modelagem e Implementação*. São Paulo: Novatec, 2017.

IEEE. *Standards for Intelligent Transportation Systems*. Disponível em: `https://standards.ieee.org`. Acesso em: 10 nov. 2025.

W3C. *JSON Data Interchange Format*. 2017. Disponível em: `https://www.json.org`. Acesso em: 10 nov. 2025.

APOSTILA: Compiladores – Unidade II. Material fornecido pelo professor (2024).