UNIT

Universidade Tiradentes

Alan Reis Anjos, Anthony Ramos dos Santos, José Henrique Oliveira de Carvalho.

Analisador Semântico

Aracaju, Outubro de 2025

Aracaju, Outubro de 2025

Relatório realizado na disciplina de Compiladores orientado pela professora Layse como complemento para a nota da 2ª Unidade.

1. Introdução

A compilação possui 4 etapas: A análise léxica, análise sintática, análise semântica e geração de código. Ao longo da disciplina de compiladores, aprendemos os processos da compilação e seus algoritmos. Neste trabalho, implementamos o analisador semântico.

2.1. Objetivo geral

Demonstrar o funcionamento do processo de análise semântica de um compilador.

2.2. Objetivos Específicos

✔ Relatar o processo de desenvolvimento;

3. Referencial Teórico

3.1 Análise Semântica

A análise semântica é a fase responsável por verificar o "significado" do código, assegurando que ele seja logicamente coerente com as regras da linguagem. Esta fase, implementada em Python, utiliza o padrão de projeto Visitor (QuizLangVisitor) para percorrer a árvore sintática gerada pelo ANTLR.

As principais verificações semânticas implementadas incluem:

- Gerenciamento de Escopo e Símbolos: Uso de uma Tabela de Símbolos para registrar todos os identificadores (IDs de quizzes, seções e questões), juntamente com suas informações associadas, como tipo e escopo.
- Verificação de Redefinição: Garante que não haja identificadores duplicados no mesmo escopo (por exemplo, duas questões não podem usar o mesmo ID dentro da mesma seção).
- Validação de Questões de Múltipla Escolha (mcq): A resposta correta deve estar contida na lista de opções; as opções não podem ser duplicadas; e a questão deve ter um mínimo de duas opções.
- Validação de Questões Numéricas (numerica): O valor inicial do intervalo não pode ser maior que o valor final.
- Validação de Questões Discursivas (discursiva): O limite de palavras deve ser um número positivo.
- Avisos: Emissão de um aviso se uma seção for declarada, mas não contiver nenhuma questão.

3.2 Tabela de Símbolos

A Tabela de Símbolos (SymbolTable) é uma estrutura de dados fundamental para a análise semântica. Ela foi implementada em Python:

- Estrutura: É um dicionário onde a chave é uma tupla composta por (escopo, nome_do_simbolo), permitindo que o mesmo nome seja usado em diferentes escopos (e.g., Q1 em Secao1 e Q1 em Secao2).
- Gerenciamento de Escopo: A tabela rastreia o escopo atual usando uma pilha de escopos (scope_stack). Ao entrar em uma nova estrutura (como um quiz ou seção), um novo escopo é empilhado, e ao sair, é desempilhado.
- Operações: As operações básicas incluem define (adicionar um novo símbolo ao escopo atual, verificando duplicatas) e lookup (procurar um símbolo, começando pelo

escopo atual e subindo na hierarquia).

4. Materiais e métodos

Para realizar a implementação do analisador semântico, escolhemos a linguagem Python, juntamente com a biblioteca ANTLR4, que nos permite implementar um compilador de forma mais simplificada, fornecendo os nossos tokens e as nossas gramáticas.

4. 1. Procedimento experimental

Implementação do Visitor: A classe QuizVisitor, que herda de QuizLangVisitor, foi implementada para percorrer a árvore sintática e extrair as informações do quiz de forma estruturada.

1. Visita ao Quiz: A função visitQuiz extrai o ID do quiz e visita os nós filhos (metadados e secoes).
2. Visita aos Metadados: A função visitMetadados extrai o título (STRING) e o tempo (NUMBER).
3. Visita à Seção: A função visitSecao extrai o nome da seção e itera sobre cada item (questão) dentro dela.
4. Visita a Questão Múltipla Escolha (visitMcq): Extrai o ID da questão, a pergunta, a lista de opções e a resposta correta.
5. Visita a Questão Discursiva (visitDiscursiva): Extrai o ID da questão, a pergunta e o limite de palavras.
6. Visita a Questão Numérica (visitNumerica): Extrai o ID da questão, a pergunta e os dois números que definem o intervalo (ctx.NUMBER(0) e ctx.NUMBER(1)).

4. 2 Teste

Foi criado um arquivo de teste (teste.txt) contendo uma linguagem QuizLang para validação.

10. Resultados e discussão

Neste estudo, o foco principal foi o desenvolvimento e a implementação robusta do analisador semântico para a QuizLang. Essa fase foi implementada em Python, utilizando o padrão de projeto Visitor (QuizLangVisitor) para percorrer a árvore sintática gerada pelo ANTLR.

10. 1 Implementação da Análise Semântica e Validações:

A análise semântica foi implementada com sucesso para verificar a coerência lógica do código e garantir que ele respeite as regras de significado da QuizLang.

- Gerenciamento de Escopo e Símbolos: O analisador utiliza uma Tabela de Símbolos (SymbolTable), que é uma estrutura de dados central para esta fase. A tabela armazena informações sobre cada identificador (ID) e seu contexto.
- Sua estrutura é um dicionário onde a chave é definida como uma tupla (escopo, nome_do_simbolo), o que permite que o mesmo nome seja reutilizado em escopos distintos (como seções diferentes)
- O gerenciamento de contexto é feito através de uma pilha de escopos (scope_stack), que é manipulada ao entrar (empilhar) ou sair (desempilhar) de estruturas como quizzes ou seções;
- Verificação de Redefinição: Foi garantido que não ocorressem identificadores duplicados dentro do mesmo escopo. As operações de define na Tabela de Símbolos incluem essa verificação, impedindo que duas questões dentro da mesma seção possuam o mesmo ID.
- Validações Específicas da QuizLang: Regras semânticas foram aplicadas a cada tipo de questão, garantindo a validade dos dados:

    1. Múltipla Escolha (mcq): Foi verificado que a resposta correta estivesse presente na lista de opções, que as opções não fossem duplicadas, e que a questão possuísse no mínimo duas opções.
    2. Questão Numérica (numerica): Assegurou-se que o valor inicial do intervalo não fosse maior que o valor final.
    3. Questão Discursiva (discursiva): Validou-se que o limite de palavras fornecido fosse um número positivo.
    4. Avisos: O analisador semântico também foi configurado para emitir avisos (warnings), por exemplo, se uma seção fosse declarada, mas não contivesse nenhuma questão.

10. 2 Principais Dificuldades e Fundamentação:

O sucesso da análise semântica dependeu da correta estruturação da gramática. A principal dificuldade encontrada durante a fase de definição da gramática para o parser foi identificar e remover as ambiguidades. Além disso, a definição precisa das regras do analisador léxico também se mostrou desafiadora, visto que as regras influenciam diretamente o resultado final da análise sintática que serve de base para o Visitor semântico. No entanto, a conclusão bem-sucedida destas etapas preliminares permitiu a implementação robusta e detalhada das verificações semânticas em Python.

11. Conclusão

A implementação dos analisadores léxico e sintático forneceu uma visão prática e aprofundada dos processos essenciais na construção de um compilador. Durante o desenvolvimento, conseguimos:

1. **Desenvolver e Testar o Analisador Léxico**: O analisador léxico foi projetado para identificar e classificar tokens a partir de uma sequência de caracteres de entrada. Implementamos expressões regulares e lidamos com a identificação de tokens e erros léxicos. Os testes confirmaram que o analisador é capaz de processar diferentes tipos de tokens e gerar erros apropriados quando o texto não está conforme as especificações da linguagem.
2. **Construir e Validar o Analisador Sintático**: O analisador sintático foi desenvolvido para organizar os tokens fornecidos pelo analisador léxico de acordo com uma gramática formal, resultando na construção de uma árvore sintática. Esse processo

ajudou a garantir que a sequência de tokens esteja em conformidade com a sintaxe da linguagem. A validação da árvore sintática e a identificação de erros sintáticos foram realizados com sucesso, comprovando a eficácia do parser na análise de entradas válidas e inválidas.

3. **Lidar com Problemas de Ambiguidade**: Durante o desenvolvimento, enfrentamos e solucionamos problemas de ambiguidade na gramática. Esse processo foi crucial para garantir que o analisador sintático funcione corretamente e produza uma análise precisa da entrada.
4. **Aplicar o ANTLR4 e Python na Implementação**: A utilização da biblioteca ANTLR4 em conjunto com Python facilitou a implementação do compilador, fornecendo ferramentas e suporte para a definição de tokens e gramáticas. A capacidade de inserir código Python diretamente na gramática ANTLR4 permitiu a personalização e a integração dos analisadores com o código Python gerado.

11. Referências Bibliográficas

AHO, Alfred V.; LAM, Monica S.; SETHI, Ravi; ULLMAN, Jeffrey D. *Compiladores: princípios, técnicas e ferramentas.* 2. ed. São Paulo: Pearson, 2009.