Engenharia Gramatical - Analisador de Código Fonte

Pedro Azevedo, Jorge Teixeira, and Rui Pinto

Universidade do Minho, Rua da Universidade, 4710-057 Braga, Portugal

Introdução

Este projeto foi realizado no âmbito da unidade curricular de Engenharia Gramatical e teve como objetivo desenvolver um **analisador de código fonte** para a linguagem LPI (Linguagem de Programação Imperativa), definida pelos alunos.

A ferramenta foi construída em **Python**, utilizando a biblioteca **Lark** para parsing e travessia da árvore sintática, e gera automaticamente um **relatório em HTML** com os resultados da análise.

Objetivos e Funcionalidades Implementadas

Foram implementadas todas as funcionalidades previstas no enunciado do TP2, utilizando a biblioteca `Lark` para a construção da árvore sintática abstrata e para a travessia com `Visitors`. Cada um dos cinco objetivos foi resolvido através de componentes dedicadas, descritas a seguir:

1. **Análise de variáveis:**
   A gestão de variáveis é realizada através da classe `SymbolTable`, que armazena instâncias da classe `Symbol`, contendo informações como tipo, escopo, linha/coluna, se foi inicializada, usada ou redeclarada.
   Durante a travessia da árvore:
   - Variáveis declaradas são adicionadas à tabela.
   - Usos de variáveis são verificados: se a variável não existir na tabela, é marcada como *não declarada*.
   - Atribuições marcam a variável como *inicializada*.
   - No final, são recolhidas listas de variáveis *não usadas*, *usadas sem inicialização* e *redeclarações*.
2. **Contagem de variáveis por tipo:**
   A contagem por tipo é feita diretamente na tabela de símbolos. Ao final da análise, é gerado um dicionário com os totais por tipo (ex: `Int`, `Set`, `Tuple`, etc.) usando a função `get_type_counts()`.
3. **Contagem de instruções por tipo:**
   Cada tipo de instrução (atribuição, leitura/escrita, estruturas condicionais e cíclicas) é contabilizado no `Visitor` principal, `SymbolTableBuilder`, através de contadores dedicados:

  - `assignment_count`
  - `read_write_count`
  - `conditional_count`
  - `cyclic_count`
  - `declaration_count`

  Estes são incrementados ao visitar os respetivos nós: `attribution`, `read_stmt`, `if_stmt`, `while_stmt`, `for_stmt`, etc.
4. **Deteção de estruturas de controlo aninhadas:**
   Foi implementado um `Visitor` adicional, chamado `NestingCounter`, que utiliza uma `stack` interna para identificar quando uma estrutura de controlo ocorre dentro de outra.
   A lógica é:
   - Ao visitar uma estrutura como `if`, `while`, `for`, etc., adiciona-se o tipo à `stack`.
   - Se a `stack` já tiver elementos, significa que estamos dentro de uma outra estrutura — contabilizando como um *aninhamento*.
   - Ao sair do bloco, o tipo é removido da `stack`.

   O total de aninhamentos é registado em `self.aninhamentos`.
5. **Deteção de `ifs` aninhados simplificáveis:**
   Implementado através do `Visitor IfsSimples`, que percorre a árvore à procura de blocos do tipo:

```
IF cond1 THEN
  IF cond2 THEN
    ...
  ENDIF
ENDIF
```

   A simplificação só é considerada válida se:
   - O bloco `THEN` do `if` externo tiver apenas uma instrução.
   - Essa instrução for um novo `if`, sem outras instruções intermédias.
   - Nenhum dos `if`s tiver parte `ELSE`.

   O total de casos detetados é acumulado em `self.otimizavel`.

Como funciona o analisador

- A gramática foi definida em `grammar.lark`.
- O ficheiro `fase2.py` carrega o código, gera a árvore sintática com `Lark`, e percorre a árvore com múltiplos `Visitors`.
- A informação é recolhida em estruturas como a `SymbolTable`, contadores e stacks.
- O relatório HTML é gerado automaticamente com `generate_html.py`.

Execução

A execução é feita com:

```
python3 fase2.py <ficheiro.lpi>
```

Será gerado o ficheiro `relatorio.html` com a análise detalhada do código.

Exemplos

Foram desenvolvidos cinco ficheiros de teste (`exemplo1.lpi` a `exemplo5.lpi`) para validar cada funcionalidade individualmente, bem como um exemplo geral (`exemplo_geral.lpi`) com todos os casos combinados.

Todos estes ficheiros produzem os respetivos relatórios `.html`.

Esses ficheiros `.html` também já foram previamente gerados, e a primeira secção dos mesmos é o código fonte usado quando foram gerados, respetivamente.

Conclusão

O projeto desenvolvido permitiu construir uma ferramenta funcional e robusta para a análise de programas escritos na linguagem LPI. Ao longo da implementação, foram aplicados conceitos fundamentais de Engenharia Gramatical, nomeadamente a definição de gramáticas formais, a construção de árvores sintáticas abstratas e a sua travessia recorrendo ao padrão `Visitor`.

A ferramenta cumpre integralmente os objetivos propostos no enunciado do trabalho prático. Foi possível detetar e reportar com precisão situações como: redeclaração de variáveis, uso indevido sem declaração ou sem inicialização, bem como a identificação de variáveis não utilizadas. Além disso, foram contabilizadas instruções por tipo e analisadas estruturas de controlo aninhadas ou redundantes (como `ifs` aninhados que poderiam ser simplificados).

A modularidade do código facilita não só a leitura e manutenção, como também a sua extensibilidade. Por exemplo, a adição de novos tipos de instruções ou regras de análise poderá ser feita facilmente adicionando novos `Visitors` especializados, sem alterar a lógica existente.

Destaca-se também a geração automática de relatórios em formato HTML, com visualização clara e realce de sintaxe. Esta funcionalidade permite interpretar rapidamente os resultados e validar o comportamento do analisador com diferentes exemplos.

Em suma, a solução apresentada é completa, reutilizável e adaptável, constituindo um bom exemplo da aplicação prática dos conhecimentos adquiridos na unidade curricular. A metodologia adotada revelou-se eficaz tanto na organização do código como na garantia de que todos os requisitos foram testados e validados de forma sistemática.

Anexos

Nesta secção incluem-se capturas dos relatórios HTML gerados para cada um dos cinco exemplos utilizados no desenvolvimento e validação do analisador.

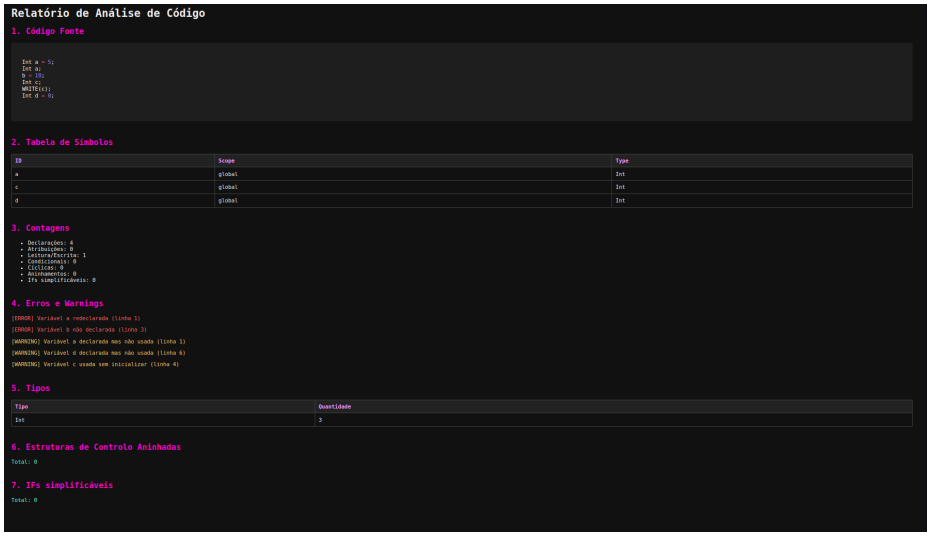

Fig. 1. Relatório HTML - Exemplo 1: Análise de variáveis

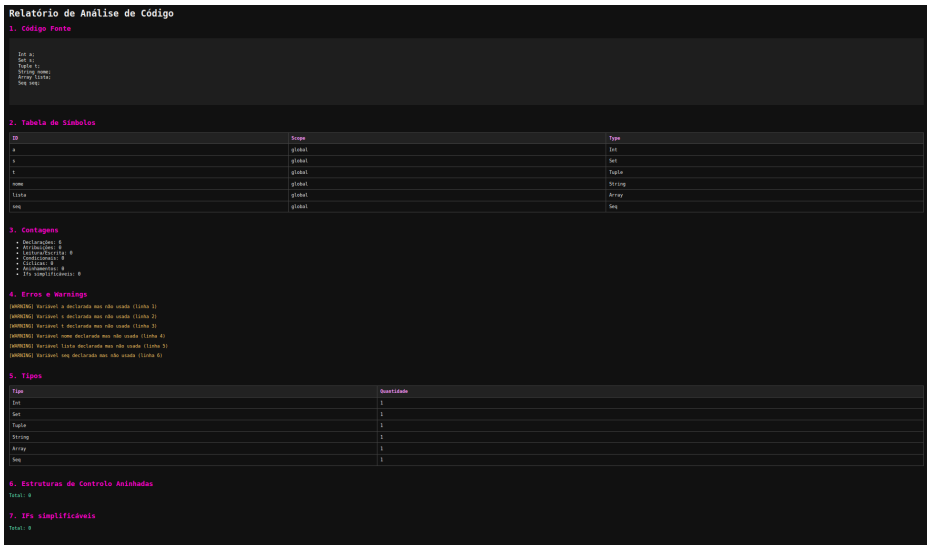

Fig. 2. Relatório HTML - Exemplo 2: Contagem por tipo de variáveis

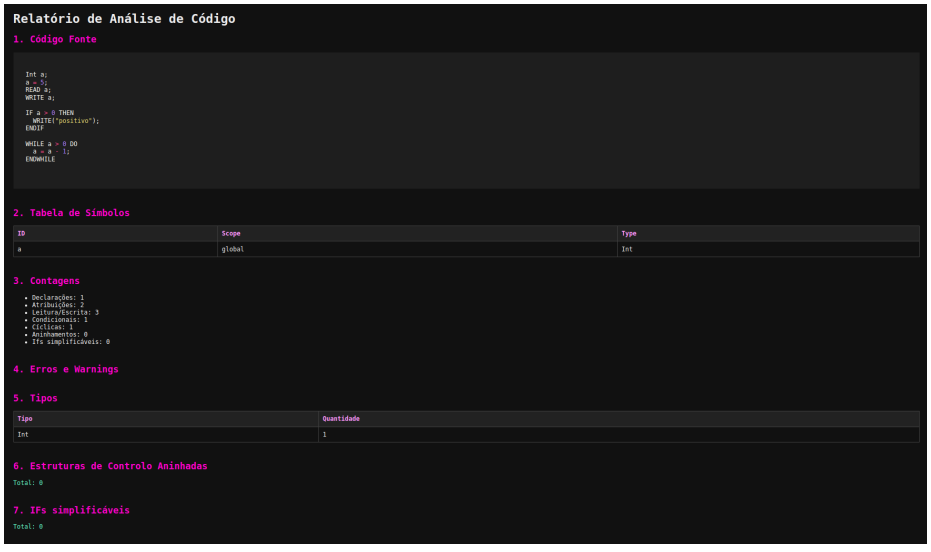

Fig. 3. Relatório HTML - Exemplo 3: Contagem de instruções por tipo

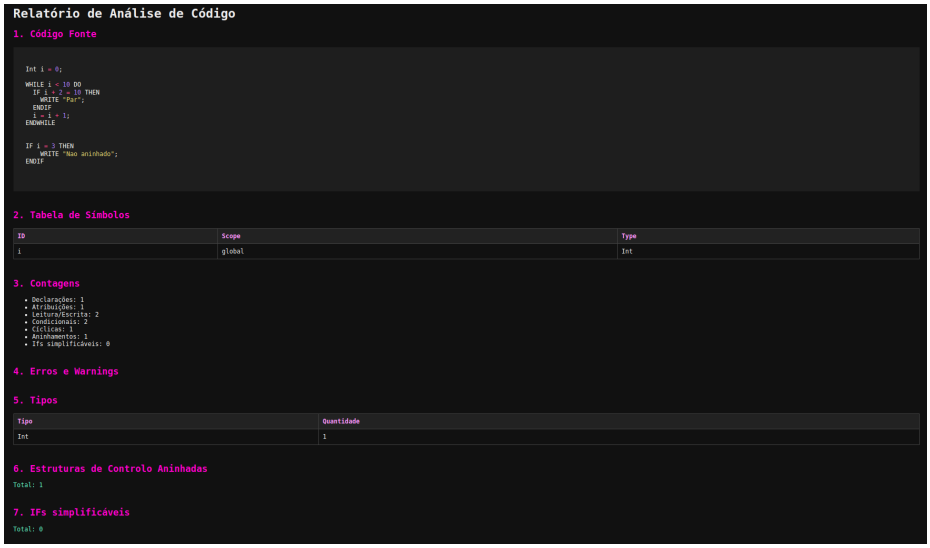

Fig. 4. Relatório HTML - Exemplo 4: Estruturas de controlo aninhadas

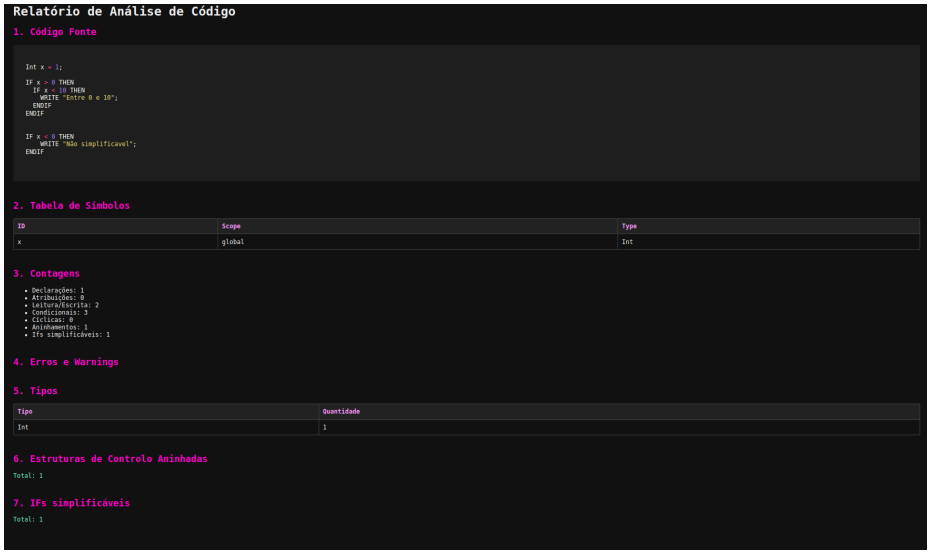

Fig. 5. Relatório HTML - Exemplo 5: Ifs simplificáveis