Projeto de Compiladores 2024/25

Compilador para a linguagem deiGO

Trabalho realizado por:
Miguel Castela uc2022212972
Nuno Batista uc2022212972

1 Introdução

Este relatório descreve o desenvolvimento de um compilador para a linguagem deiGO, realizado no âmbito do projeto da disciplina de Compiladores do ano letivo 2024/25. O objetivo deste projeto é aplicar os conhecimentos adquiridos ao longo do curso na construção de um compilador funcional.

2 Gramática

The DeiGO language grammar is organized based on syntax rules that enable bottom-up parsing while respecting the deiGO language specifications. Our implementation focuses on resolving ambiguities by explicitly defining operator precedence and associativity rules.
Non-termials where simplified by introducing auxiliary nodes to handle the optional elements and repititions.

- **Simplification of Non-terminals with Aux nodes**:
  In our grammar, we use auxiliary(AUX) nodes to simplify the structure of non-terminals. This approach helps in managing optional elements and repetitions more effectively. These nodes help in managing repeated elements by acting as a container for multiple instances of a particular non-terminal. For example, in the VarSpec rule we use an AUX node to handle mutiple vairable declarations. This function has the main goal of maintaining a clear and organized AST. By grouping related nodes under an AUX node, we can ensure that the tree structure is easy to traverse and understand.

- **Handling of Optional and Repeated Elements**:
  With the help of AUX nodes, optional and repeated elements are handled using specific grammar rules that provide alternatives for their presence or absences. For example, OptFuncParams can either be a FuncParams node or an AUX node representing the absence of function parameters. By defining optional elements explicitly, we also ensure that the parser can handle cases where certain elements are missing. This is crucial for maintaining the correctness of the AST. (For instance, the Declarations rule manages multiple functions and variable declarations). By using AUX and optional nodes, we can easily append

new declarations to the existing list, ensuring that the AST remains well structured and easy to transverse.

- **Optimization and Readibility** :
  Our grammar rules are desgined to be clear and readable, making it easier to understand the parsing process. We define operator precedence and associativity explicitly to resolve ambiguities. This ensures that expressioons are parsed correctly according to the DeiGO language specifications. Additionally, meaningful names for non-terminals and auxiliary nodes are used to enhance the readability of the grammar of the grammar.
- **Error Handling**:
- **AST Generation**: (block, clearAUX, etccccccc)
- **Lexical Analysis Integration**

3 Algoritmos e estruturas de dados da AST e da tabela de simbolos

4 Algoritmos e Estruturas de Dados

O compilador desenvolvido é composto por várias fases, cadaaaaa uma responsável por uma parte específica do processo de compilação. As principais fases são:

- Análise Léxica
- Análise Sintática
- Análise Semântica
- Geração de Código Intermediário
- Otimização
- Geração de Código Final

5 Geração de Código