# Análise Semântica da Linguagem TPP

**Guilherme B. Del Ro**[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão, PR – BRASIL

guilhermerio@alunos.utfpr.edu.br

***Abstract.*** *This document describes the implementation of a semantic analyzer for the TPP language, through the creation of symbol tables for functions and variables used together with the tree created in the parsing, as a result it was possible to identify the semantic errors and show them in the terminal , along with the tables*

***Resumo.*** *Este documento descreve a implementação de um analisador semântico para a linguagem TPP, através da criação de tabelas de símbolos para funções e variáveis utilizadas em conjunto com a árvore criada na análise sintática, como resultado foi possível identificar os erros semânticos e mostrá-los no terminal, juntamente com as tabelas.*

## 1. Introdução

Este trabalho realiza a análise semântica da linguagem TPP ou T++, através da criação de uma tabela de símbolos tanto para as variáveis quanto para as funções, tal tabela contém os atributos de ambos. A identificação dos erros começa a ser feita na análise sintática, onde os erros de escritas são computados, computando também os erros semânticos e os enviando para tratamento no arquivo que ocorre a análise semântica.

## 2. Regras Semânticas e Tabelas de Símbolos

### 2.1. Tabela de Símbolos

As regras semânticas são definidas através de duas tabelas, sendo elas uma tabela para as variáveis e uma tabela para as funções do código de entrada. Em cada tabela há os parâmetros utilizados para comparação, sendo estes campos lexema, tipo, linha inicial e final, entre outros.

### 2.2. Regras Semânticas

As regras semânticas foram definidas no documento de especificações e servem para a identificação dos erros e avisos, sendo estas compostos por:

- Erro: Função principal não declarada.
- Erro: Função principal deveria retornar inteiro, mas retorna vazio.
- Erro: Chamada à função ‘func’ com número de parâmetros menor que o declarado
- Erro: Função principal deveria retornar inteiro, mas retorna vazio.
- Erro: Chamada a função ‘func’ que não foi declarada.
- Erro: Chamada para a função principal não permitida.
- Aviso: Função ‘func’ declarada, mas não utilizada.
- Aviso: Chamada recursiva para principal.

- Aviso: Variável ‘a’ declarada e não utilizada.
- Erro: Variável ‘a’ não declarada.
- Aviso: Variável ‘a’ declarada e não inicializada.
- Aviso: Variável ‘a’ já declarada anteriormente.
- Aviso: Atribuição de tipos distintos ‘a’ inteiro e ‘expressão’ flutuante.
- Aviso: Atribuição de tipos distintos ‘a’ flutuante e ‘func’ retorna inteiro.
- Aviso: Coerção implícita do valor de ‘x’.
- Aviso: Coerção implícita do valor retornado por ‘func’.
- Erro: Índice de array ‘X’ não inteiro.
- Erro: índice de array ‘A’ fora do intervalo (out of range).

## 3. Árvore Sintática Abstrata

Com a árvore gerada na sessão anterior (sintática) apenas é necessário retirar da árvore os nós que não queremos, estes nós são compostos em sua maioria por expressões e indicadores de operadores de expressão.

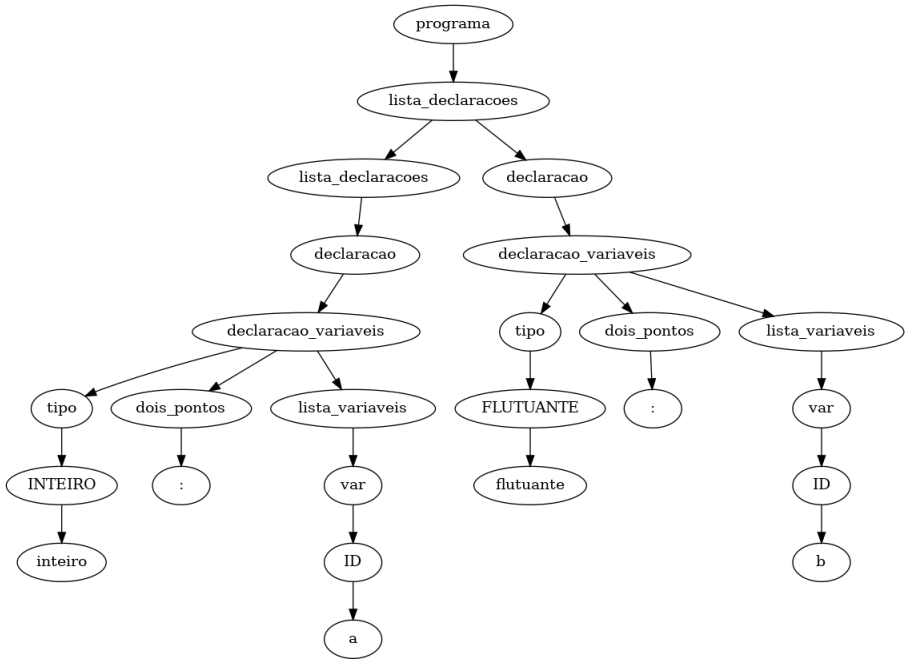


**Figura 1. Árvore antes da realização da poda**

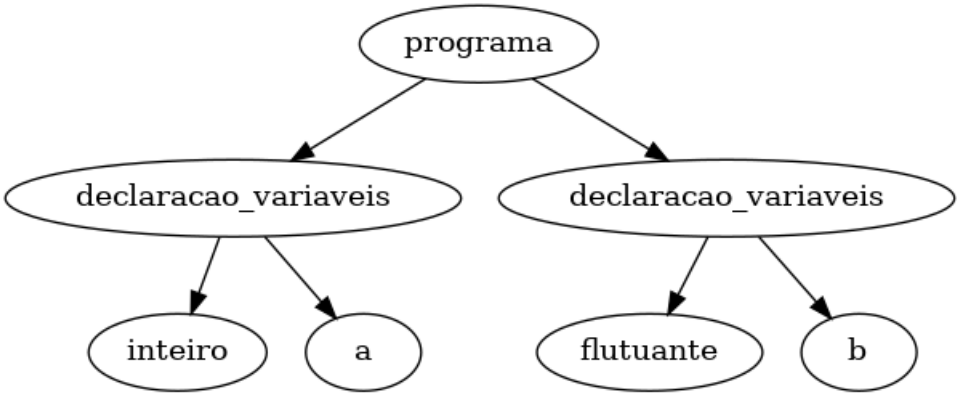


**Figura 2. Árvore após a realização da poda**

## 4. Especificação Formal dos Autômatos

As expressões regulares são formadas por símbolos, sendo eles de operação, atribuição e entre outros, por operadores lógicos e operadores relacionais além das expressões do ID, números inteiros, números com ponto flutuante, comentários no código e notação cientifica que são mais complexos.

## 5. Detalhes da Implementação

Após importar os dados necessários do arquivo da análise sintática é verificado se as variáveis estão no escopo correto, isto é, se ela foi declarada dentro do escopo da função, caso não, adiciona a variável ao escopo global. Então é necessário fazer a criação das tabelas de símbolos através da função *gera_tabelas*, onde será feito a criação das tabelas através da biblioteca python *tabulate* que então serão preenchidas através de uma contagem de variáveis e funções, e suas propriedades e então mostradas no terminal.

Após este processo é chamada uma função de controle, que contém todas as funções de checagem semântica, que verificam os erros citados na seção de regras semânticas, colocando todos os erros em uma lista de mensagens, que serão mostradas no terminal através de um laço de repetição simples.

Para a poda da árvore há outra função de controle chamada *arruma_arvore* que contém um variável com todas as expressões indesejadas. Além de outras duas funções chamadas *poda_arvore* que através de uma chamada recursiva obtém a declaração, o corpo e o final do código de entrada, e a função *ajusta_arvore* que trata os operadores lógicos e reorganiza a arvore.

## 6. Exemplos de Entrada e Saída

Na entrada do código enviamos um arquivo com a extensão .tpp através do comando:

- python3 tppSemantic.py ./semantica-testes/sema-001.tpp

Onde o algoritmo utilizado como exemplo foi *sema-001.tpp* obtido através da pasta de testes chamada *semantica-testes* (ver Figura 3).

```
{Erro: Função principal não declarada}
{Aviso: Variável 'a' declarada e não utilizada}
{Aviso: Variável 'b' declarada e não utilizada}

inteiro: a
flutuante: b

```

**Figura 3. Exemplo de código de entrada**

Observa-se no código de exemplo os comentários esperados pelo analisador semântico referente aos erros encontrados, tais erros são mostrados na saída juntamente com a tabela e a árvore podada (Figura 2). Além da árvore gerada na fase anterior (sintática).

```
TABELA DE FUNÇÕES:
```

| Lexema | Tipo | Número de Parâmetros | Parâmetros | Init | Linha Inicial | Linha Final |
|---|---|---|---|---|---|---|
| | | | | | | |

```
TABELA DE VARIÁVEIS:
```

| Lexema | Tipo | Dimensões | Tamanho Dimensões | Escopo | Linha |
|---|---|---|---|---|---|
| a | inteiro | 0 | [] | global | 5 |
| b | flutuante | 0 | [] | global | 6 |

```
Aviso: Variável ‘a’ declarada e não utilizada.
Erro: Função principal não declarada.
Aviso: Variável ‘b’ declarada e não utilizada.
```

**Figura 4. Saída esperada utilizando o código de exemplo**

## 7. Conclusão

Através deste trabalho foi possível concluir a importância de um analisador semântico bem feito para uma linguagem, pois ele nos permite refatorações e correções de forma mais assertiva.

## Referências


Gonçalves, R. A. (2021a). Análise semântica (trabalho – 3ª parte).

Gonçalves, R. A. (2021b). Aula 016 – análise semântica.