Análise Sintática da linguagem TPP

Henrique S. Marcuzzo[1]

[1]Departamento de Ciência da Computação
– Universidade Tecnológica Federal do Paraná (UTFPR)

henriquemarcuzzo@alunos.utfpr.edu.br

Abstract. *This article describes the steps taken to perform the syntactic analysis work, with a brief explanation of the syntactics adopted, details of how the code was produced, and the experience of working with the technologies used.*

Resumo. *Este artigo descreve os passos tomadas para a execução do trabalho de análise sintática, com um breve explicação sobre sintática adotada, detalhes de como foi produzido o código e a experiência de se trabalhar com as tecnologias usadas.*

1. Introdução

Este trabalho foi realizado com a linguagem de programação *Python* juntamente com a biblioteca *PLY Yacc*, *Anytree* e *Graphviz*, para a Análise sintática dos códigos `.tpp`, durante o processo foi desenvolvido métodos para tratamento de erros sintáticos e aprimorado alguns métodos existentes no código de início, para que atendessem as necessidade descritas na documentação da linguagem de programação fictícia da disciplina **TPP**.

2. Descrição da gramática TPP

A gramática *Tpp*, possui, segundo o padrão *BNF*, as seguintes regras gramaticais sintáticas: A linguagem *Tpp*, foi construída na disciplina de compiladores, com intuito didático, para proporcionar aos alunos uma noção de como funciona a construção de um compilador, portanto a linguagem estudada oferece apenas as estruturas mais simples de uma linguagem convencional, que são:

- programa
  - lista_declaracoes
- lista_declaracoes
  - lista_declaracoes declaracao
  - declaracao
- declaracao
  - declaracao_variaveis
  - inicializacao_variaveis
  - declaracao_funcao
- declaracao_variaveis
  - tipo DOIS_PONTOS lista_variaveis
- inicializacao_variaveis
  - atribuicao
- lista_variaveis
  - lista_variaveis VIRGULA var

    - var
- var
    - ID
    - ID indice
- indice
    - indice ABRE_COLCHETE expressao FECHA_COLCHETE
    - ABRE_COLCHETE expressao FECHA_COLCHETE
- tipo
    - INTEIRO
    - FLUTUANTE
- declaracao_funcao
    - tipo cabecalho
    - cabecalho
- cabecalho
    - ID ABRE_PARENTESE lista_parametros FECHA_PARENTESE
    - corpo FIM
- lista_parametros
    - lista_parametros VIRGULA parametro
    - parametro
    - vazio
- parametro
    - tipo DOIS_PONTOS ID
    - parametro ABRE_COLCHETE FECHA_COLCHETE
- corpo
    - corpo acao
    - vazio
- acao
    - expressao
    - declaracao_variaveis
    - se
    - repita leia
    - escreva
    - retorna
    - erro
- se
    - SE expressao ENTAO corpo FIM
    - SE expressao ENTAO corpo SENAO corpo FIM
- repita
    - REPITA corpo ATE expressao
    - SE expressao ENTAO corpo SENAO corpo FIM
- atribuicao
    - var ATRIBUICAO expressao
- leia
    - LEIA ABRE_PARENTESE var FECHA_PARENTESE
- escreva
    - ESCREVA ABRE_PARENTESE expressao FECHA_PARENTESE
- retorna

    - RETORNA ABRE_PARENTESE expressao FECHA_PARENTESE
- expressao
- expressao$_{l}ogicaatribuicao$
- expressao_logica
    - expressao_simples
    - expressao_logica operador_logico expressao_simples
- expressao_simples
    - expressao_aditiva
    - expressao_simples operador_relacional expressao_aditiva
- expressao_aditiva
    - expressao_multiplicativa
    - expressao_aditiva operador_soma expressao_multiplicativa
- expressao_multiplicativa
    - expressao_unaria
    - expressao_multiplicativa operador_multiplicacao
    - expressao_unaria
- expressao_unaria
    - fator
    - operador_soma fator
    - operador_negacao fator
- operador_relacional
    - MENOR
    - MAIOR
    - IGUAL
    - DIFERENTE
    - MENOR_IGUAL
    - MAIOR_IGUAL
- operador_soma
    - MAIS
    - MENOS
- operador_logico
    - E
    - OU
- operador_negacao
    - NAO
- operador_multiplicacao
    - VEZES
    - DIVIDE
- fator
    - ABRE_PARENTESE expressao FECHA_PARENTESE
    - var
    - chamada_funcao
    - numero
- numero
    - NUM_INTEIRO
    - NUM_PONTO_FLUTUANTE
    - NUM_NOTACAO_CIENTIFICA

- chamada_funcao
  - ID ABRE_PARENTESE lista_argumentos FECHA_PARENTESE
- lista_argumentos
  - lista_argumentos VIRGULA expressao
  - expressao
  - vazio

3. Formato na Análise Sintática

Para o desenvolvimento deste trabalho, foi utilizado a árvore sintática *LALR(1)*, sendo mais eficaz que modo *SLR(1)*, e menos eficaz que o *LR(1)*, pois gera menos estados da árvore sintática do que *LR(1)*, mas diminui o tamanho da tabela final, portanto, por questões de desempenho e eficácia o modo *LALR(1)* foi o escolhido.

Mesmo que exista linguagens que apenas podem ser interpretadas por *LR(1)*, a linguagem tpp não se aplica a esta regra, podendo ser interpretado por ambos os métodos, portanto foi priorizado o desempenho e a redução de uso de memória. Outro fator decisivo para esta escolha se da devido ao fato de que a maioria dos compiladores aceitam bem esse tipo de árvore sintática, *LALR(1)*.

4. Ferramenta *Yacc*

O *Yacc (yer another compiler-compiler)* fornece uma ferramenta geral para impor uma estrutura para a inicialização de um programa de computador. O *Yacc* irá preparar uma especificação do processo de entrada. isto inclui regras que descrevem a estrutura de entrada, código para ser invocado quando estas regras são reconhecidas, e uma rotina de baixo nível para fazer a entrada básica. O *Yacc* gera então uma função para controlar o processo de entrada de dados. Esta função, chamada parser, chama a rotina de entrada de baixo nível fornecida pelo analisador léxico para captar os tokens do fluxo de entrada. Estes tokens são organizados de acordo com as regras gramaticais. Quando uma destas regras for reconhecida, o código do utilizador fornecido para esta regra, entra em ação e é invocado pelo *Yacc*. As ações têm a capacidade de devolver valores e de fazer uso dos valores de outras ações.

Portando para a execução deste trabalho foi escolhido a linguagem de programação *Python* e a biblioteca *PLY Yacc*, assim toda implementação de tratamento para construção da árvore seguiu a lógica determinada pelo *Yacc*, descrita anteriormente, que mostrou-se prática e de fácil entendimento.

A base de implementação do analisador sintático foi fornecida pelo professor da disciplina, e ficando a cargo dos alunos aprimorar o que já foi implementado e produzir o código para os tratamentos de eventuais erros sintáticos que pudessem acontecer durante a análise.

O analisador léxico também utilizado pelo *Yacc* foi desenvolvido na atividade passada com apenas algumas alterações nos nomes dos tokens, visto que ouve uma mudança entre a atividade léxica e sintática.

5. Árvore Sintática

Para a construção da *Árvore Sintática*, foi utilizado o código fornecido pelo professor *mytree.py* e das bibliotecas *anytree* e *graphviz*.

No final do processo, duas árvores são geradas, uma com todos os valores únicos para cada ramificação da árvore e a outra visualização se faz com todos os valores únicos sem repetição nas ramificações.

Podemos ver essas duas árvores geradas, executando o código abaixo, que apesar de simples, permite demonstrar as árvores geradas, na imagem 1 vemos a árvore com todos os valores terminais do código, porém sem repetição para cada ramificação, e na imagem 2 vemos a árvore com os valores únicos para cada ramificação:

```
inteiro principal()
    inteiro: i
    i := 1
fim
```

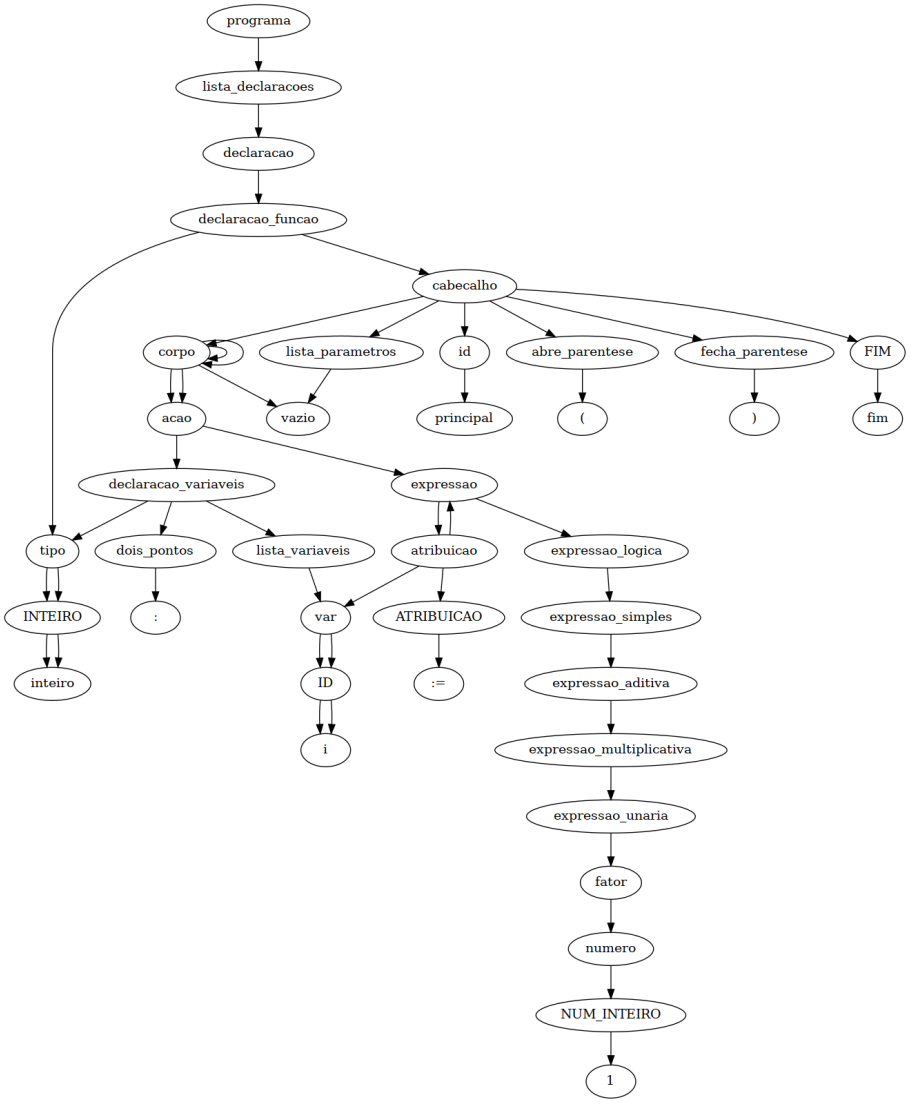

Figura 1. Árvore sem os valores únicos para cada ramificação

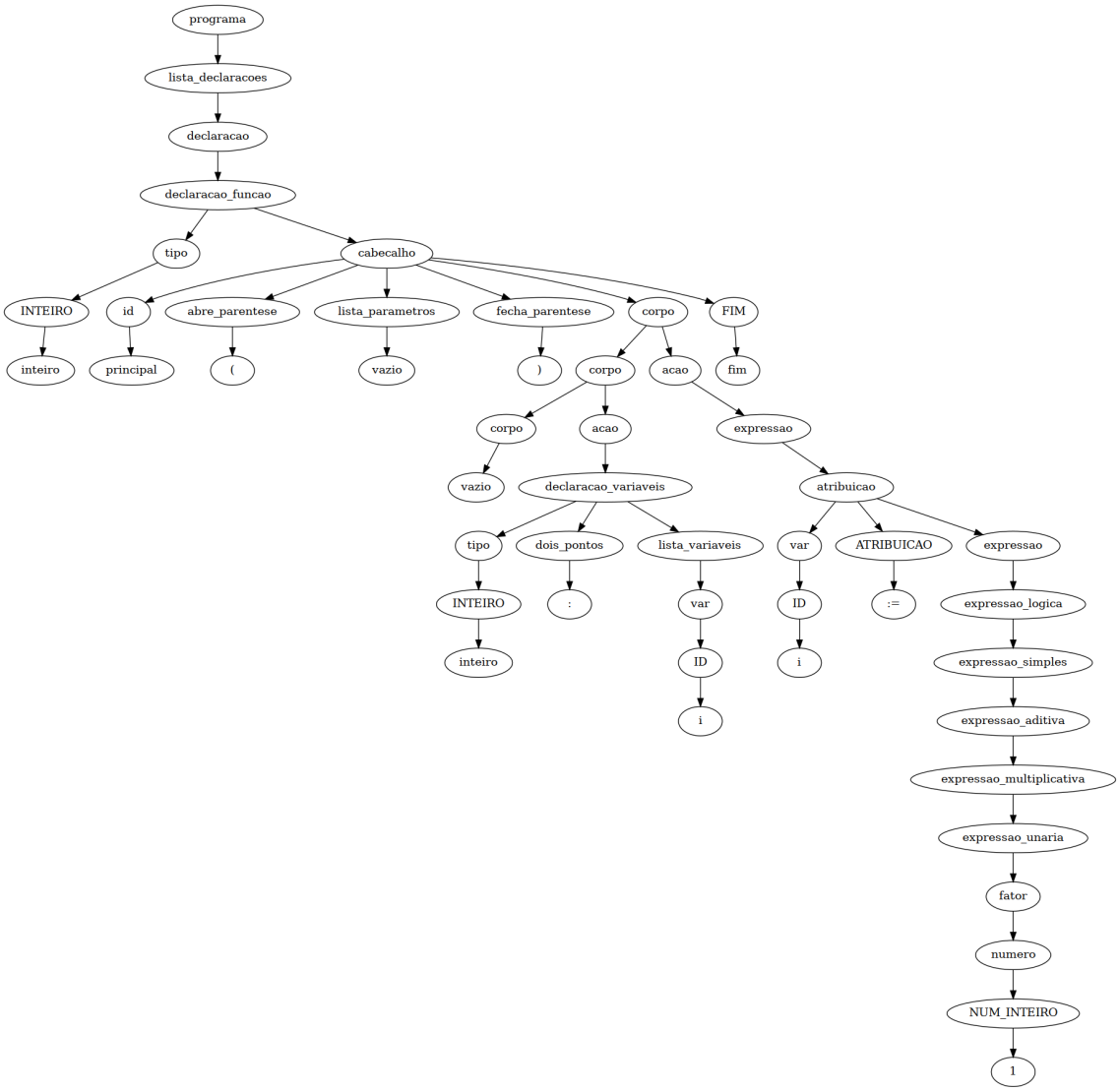

Figura 2. Árvore com os valores únicos para cada ramificação

6. Referências

Moodle. Gramática da Linguagem T++ com TOKENS (para a lista de tokens). 2021. Disponível em: *https://docs.google.com/document/d/1e7_M-bD1RUbJAnyR8rZyJ35vKbYEN6KQG4l5L8FQ7_I/edit*. Acesso em: 01 abr. 2021.

Informatik. Yacc: Yet Another Compiler-Compiler. 2009. Disponível em: *https://web.archive.org/web/20090306123410/http://www2.informatik.uni-erlangen.de/Lehre/WS200304/Compilerbau/Uebungen/yacc.pdf*. Acesso em: 05 abr. 2021.

Wikipedia. LALR parser. 2021. Disponível em: *https://en.wikipedia.org/w/index.php?title=LALR_parseraction=history*. Acesso em: 05 abr. 2021.