Análise Semântica na implementação de um compilador para a Linguagem T++

Guilherme Vasco da Silva

Departamento de Ciências da Computação
Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão, PR – Brazil

guilhermevasco@alunos.utfpr.edu.br

Resumo. *Este artigo é um trabalho com o intuito de documentar um programa/ferramenta para realização da análise semântica de códigos na linguagem de programação T++.*

1. Linguagem T++

A linguagem a ser lida pelo analisador sintático será a T++, que suporta dados inteiros e flutuantes e necessidade da especificação de tipos das variáveis locais e globais, porém as funções podem ter tipo omitido. A linguagem possui também operações de adição, multiplicação, subtração, divisão e também operadores lógicos E, OU e NÃO.

2. Análise Semântica

A terceira parte a ser desenvolvida do compilador é a Análise Semântica, onde são verificados os erros semânticos que vieram da primeira e segunda parte do projeto, também sendo coletadas as informações necessárias para a próxima fase da compilação. No compilador proposto para a linguagem T++, será feita uma "segunda passada" na AST, para a anotação de atributos. Também nessa fase será realizada a poda da árvore.

3. Regras Semânticas

Para a análise semântica, como citado anteriormente, a AST será percorrida novamente analisando as regras semânticas e gerando uma tabela de símbolos.

As regras semânticas para a análise podem ser separadas em oito, que serão apresentadas a seguir.

3.1. Funções e Procedimentos

O identificador de função (nome e quantidade de parâmetros formais), além de que os parâmetros formais devem ter um apontamento para o identificador de variáveis.

3.2. Função Principal

Todo programa escrito em TPP deve ter uma função principal declarada.

3.3. Funções e Procedimentos

A quantidade de parâmetros reais de uma chamada de função/procedimento “func” deve ser igual a quantidade de parâmetros formais da sua definição.

3.4. Compatibilidade do retorno

Uma função deve retornar um valor de tipo compatível com o tipo de retorno declarado.

3.5. Variáveis

O identificador de variáveis locais e globais: nome, tipo e escopo devem ser armazenados na Tabela de Símbolos. Variáveis devem ser declaradas, inicializadas e antes de serem utilizadas (leitura).

3.6. Atribuição

Na atribuição devem ser verificados se os tipos são compatíveis.

3.7. Coerções implícitas

Warnings deverão ser mostrados quando ocorrer uma coerção implícita de tipos.

3.8. Arranjos

Na linguagem T++ é possível declarar arranjos, pela sintaxe da linguagem o índice de um arranjo é inteiro. Na tabela de símbolos é necessário armazenar se uma variável declarada tem um tipo, se é uma variável escalar ou um vetor ou uma matriz.

4. Tabela de símbolos

Uma tabela de símbolos deve ser gerada para a realização da análise dos erros citados nas regras semânticas, a construção dessa tabela pode ser iniciada desde a primeira etapa do compilador (análise léxica). No caso da ferramenta proposta, a construção se inicia na segunda etapa (análise sintática), com uma verificação das regras sendo feita pelos nós.

São geradas duas tabelas no código: a Tabela de Funções e a Tabela de Variáveis.

A tabela de funções é composta por lexema, tipo, número de parâmetros, parâmetros, um indicador se ela é inicializada (init), linha inicial e linha final, já a tabela de variáveis é composta por lexema, tipo, dimensões, tamanho das dimensões, escopo e linha.

5. Árvores Sintáticas

A análise sintática permite a criação de uma árvore sintática abstrata anotada (ASTO), mas após a verificação das regras semânticas é possível se gerar uma poda na árvore, o que facilita a geração de código e, consequentemente, facilitará a próxima fase do compilador. A ferramenta implementada para análise semântica gera a árvore sintática abstrata, bem como a árvore após a poda.

6. Execução

Como as outras etapas do compilador, a análise semântica também foi escrita em Python, a pasta desta etapa também é organizada da mesma forma das anteriores, com o código podendo ser localizado na pasta *impl* e uma pasta com códigos escritos em T++ para teste na pasta *semantica-testes*. O arquivo responsável pela etapa é chamado *tppsemantic.py* e pode ser executado pelo comando *python tppsemantic.py <arquivo tpp>*.

7. Testes

Através dos arquivos fornecidos para testes, pode-se realizar uma verificação do funcionamento das regras semânticas.

O primeiro teste a ser executado será com o arquivo *sema-001.tpp* através do comando *python tppsemantic.py .\semantica-testes\sema-001.tpp* (formato de pastas do Windows).

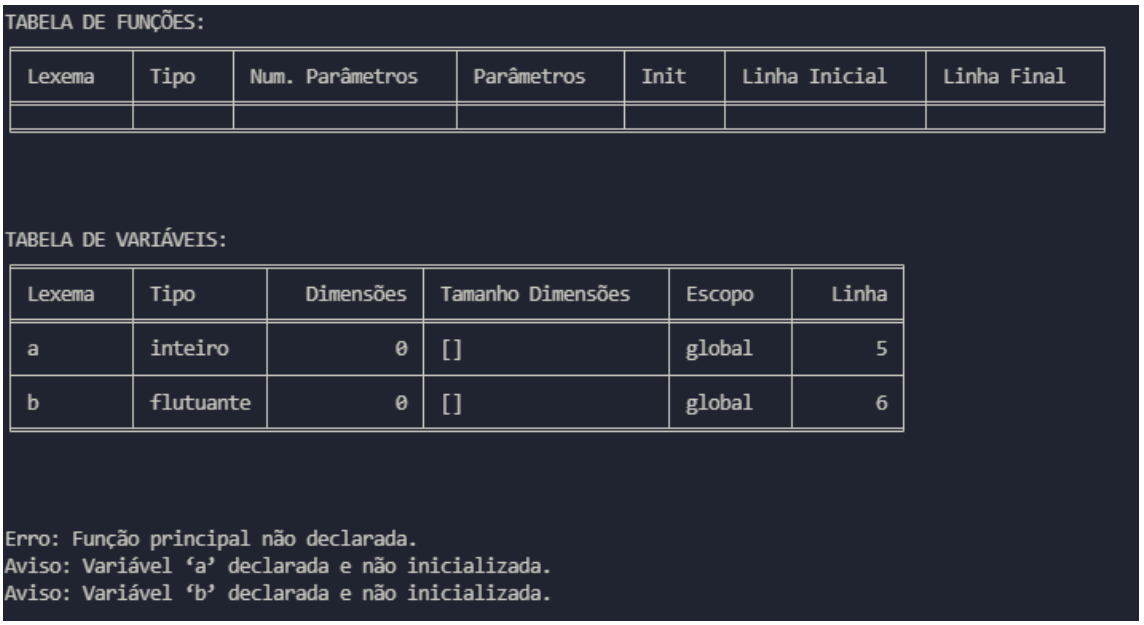

Figura 1. Saída da análise semântica do arquivo sema-001.tpp.

As duas tabelas foram geradas conforme o esperado, assim como as imagens com as árvores abstrata e podada, mas é possível notar que a tabela de funções está em branco, isso se dá pelo código de exemplo não ter funções, isso pode ser visto ao final também onde está presente o erro referente à segunda regra: Função principal não declarada. Junto a isso, estão dois avisos referentes à quinta regra.

O próximo teste a ser executado será com o arquivo *sema-005.tpp*, com o comando *python tppsemantic.py .\semantica-testes\sema-001.tpp*.

TABELA DE FUNÇÕES:

Lexema	Tipo	Num. Parâmetros	Parâmetros	Init	Linha Inicial	Linha Final
func	inteiro	2	['x', 'y']	True	6	8
principal	inteiro	0	[]	True	10	18

TABELA DE VARIÁVEIS:

Lexema	Tipo	Dimensões	Tamanho Dimensões	Escopo	Linha
x	inteiro	0	[]	func	6
y	inteiro	0	[]	func	6
a	flutuante	0	[]	principal	11
c	flutuante	0	[]	principal	12
b	inteiro	0	[]	principal	13

```
Aviso: Atribuição de tipos distintos ‘b’ inteiro e ‘expressão’ flutuante
Erro: Função principal deveria retornar inteiro, mas retorna vazio.
Aviso: Atribuição de tipos distintos ‘a’ flutuante e ‘expressão’ inteiro
Aviso: Coerção implícita do valor de ‘func’.
Aviso: Coerção implícita do valor de ‘c’.
```

Figura 2. Saída da análise semântica do arquivo sema-005.tpp.

Nesse retorno é possível perceber ambas as tabelas preenchidas e existe a função principal, então o erro que aparece ao fim já é outro, mas ainda é referente à segunda regra: Função principal deveria retornar inteiro, mas retorna vazio.

Além disso, são geradas imagens das árvores, que podem ser localizadas na pasta semantica-testes.

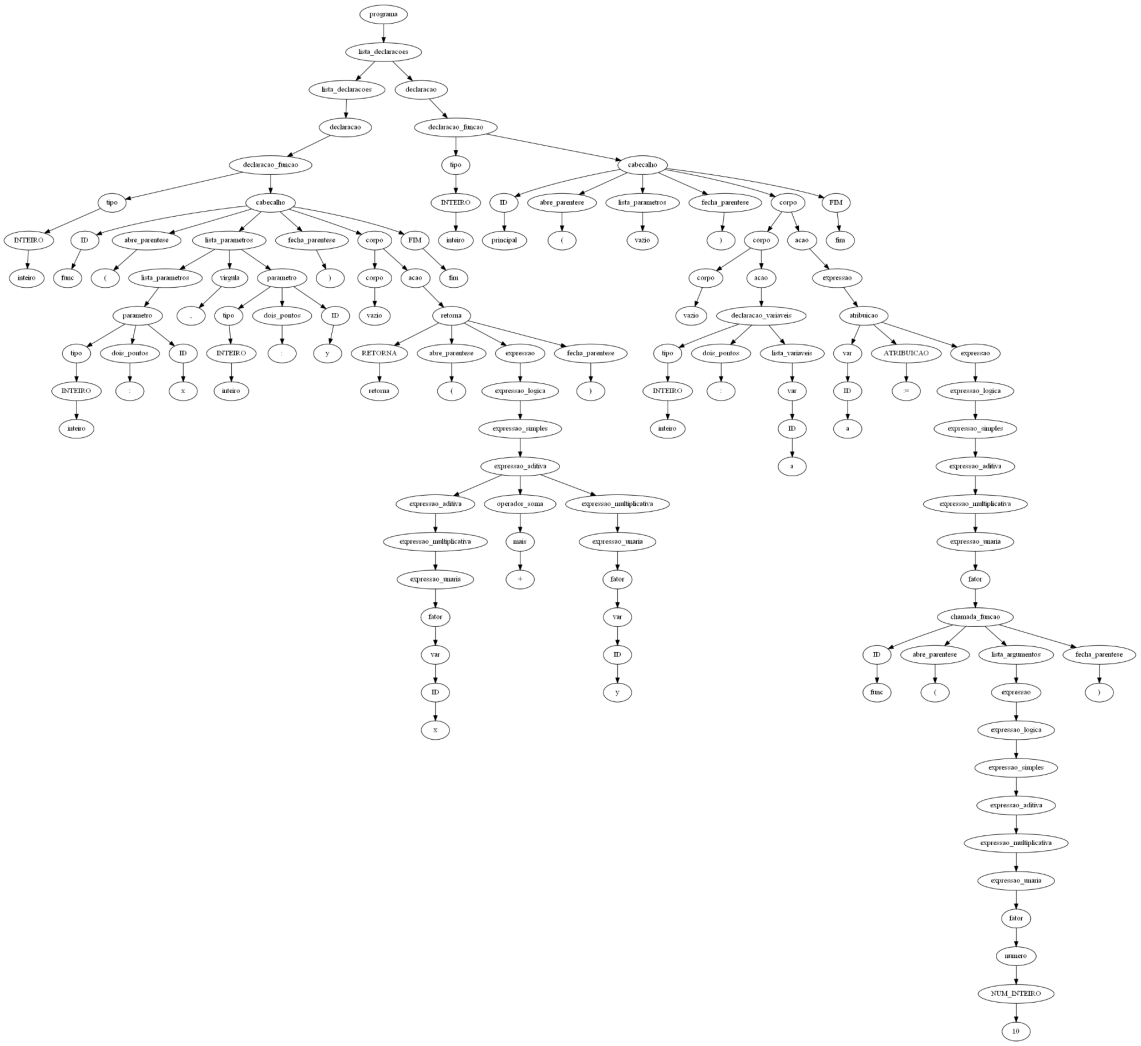

Figura 3. Árvore sintática abstrata resultante (sema-005.tpp.unique.ast.png).

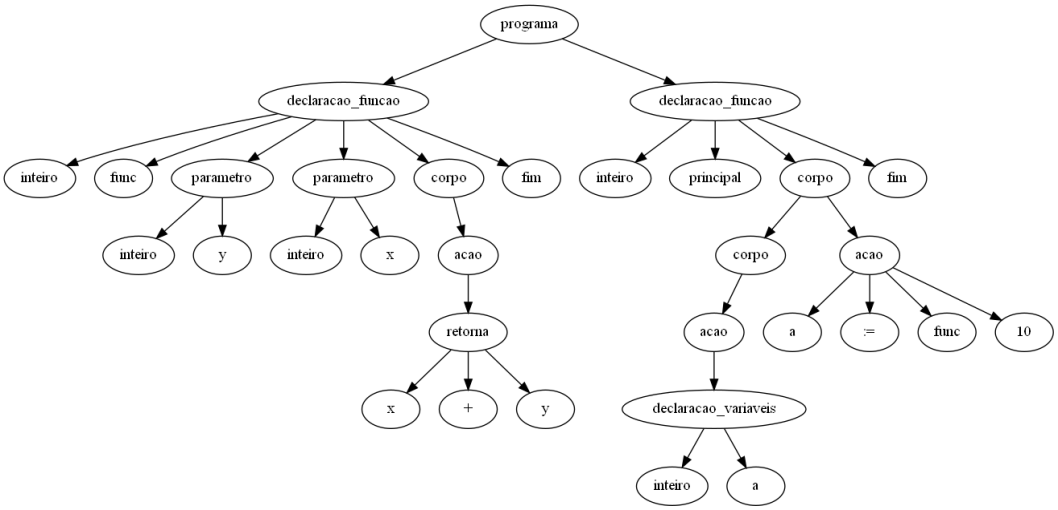

Figura 4. Árvore sintática podada resultante (sema-005.tpp.cut.unique.ast.png).

8. Conclusões

A terceira etapa do compilador funciona com taxa de sucesso aceitável. As regras estão sendo aplicadas e possibilitam a informação de erros de contexto que passaram pelas outras duas etapas anteriores. As tabelas geradas permitem uma visualização boa das variáveis e funções presentes e, juntamente com a árvore podada gerada ao final, serão essenciais na geração de código objeto.

Referências

Gonçalves, R. A. (2017) "Documentação online da Gramática da TPP", https://docs.google.com/document/d/1oYX-5ipzL_izj_hO8s7axuo2OyA279YEhnAItgXzXAQ, Novembro.

Ricarte, I. L. M. (2003) "Análise semântica", https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node71.html, Novembro.

Gonçalves, R. A. (2021) "Projeto de Implementação de um Compilador para a Linguagem T++", https://moodle.utfpr.edu.br/pluginfile.php/183223/mod_resource/content/16/trabalho-03.md.notes.pdf, Novembro.

Gonçalves, R. A. (2021) "Visão Geral e Regras Semânticas para TPP", https://moodle.utfpr.edu.br/pluginfile.php/1373326/mod_resource/content/0/aula-16-analise-semantica-regras-semanticas-tpp.md.notes.pdf, Novembro.