Compiladores - Análise Semântica

Alisson da S. Vieira

[1]Ciência da Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87.301-899 – Campo Mourão – PR – Brasil

alissonv@alunos.utfpr.edu.br

Abstract. *For this work, we will implement the Semantic Analysis, which is the analysis after the Syntactic Analysis, done together with the tree generated from the parsing. With this tree, we will perform the semantic analysis, and at the end perform the tree pruning. And we'll discuss development processes throughout the article.*

Resumo. *Para esse trabalho, iremos implementar a Análise Semântica, esta que é a análise posterior a Análise Sintática, feita juntamente com a árvore gerada a partir da análise sintática. Com essa árvore, iremos realizar as análises semânticas, e ao final realizar a poda da árvore. E vamos discutir sobre os processos de desenvolvimento com o decorrer do artigo.*

1. Introdução

Análise semântica é um dos processos do desenvolvimento de um compilador. Nessa etapa são verificados os erros semânticos no código fonte, esses que são definidos pelo próprio projetista da linguagem. A análise dos erros semânticos não podem ser executados nas etapas anteriores, pois é necessário analisar se o código está correto sintaticamente primeiro, e após, com base nas regras definidas pelo projetista, verificar se semanticamente também está correto. Logo, a análise semântica é um processo que visa, a partir da árvore gerada na análise sintática, implementar um analisador semântico e gerar uma árvore podada. Essa análise foi implementada com a linguagem de programação `Python`, as bibliotecas: `Anytree`, `ply` e `pandas`.

2. Tabela de Símbolos

Para auxiliar no processo da análise semântica, foi feito uma tabela de símbolos. Essa tabela foi criada nas etapas anteriores (análise léxica e sintática), e é dividida nas: tabelas de funções e tabelas de variáveis. Para ver as tabelas na saída do código, é necessário executá-lo com o seguinte parâmetro: "sta". Pegando o Código 1 de exemplo, podemos ver que esse é um código simples, cujo propósito é apenas somar dois números.

Código 1. Código exemplo em T++

```
1 inteiro: resultado
2
3 principal()
4     inteiro: n1, n2
5
6     leia(n1)
7     leia(n2)
8
```

```
9     resultado := somaNumeros(n1, n2)
10    escreva(resultado)
11
12 fim
13
14 inteiro somaNumeros(inteiro: n1, inteiro: n2)
15    retorna(n1 + n2)
16 fim
```

Para esse exemplo, ao rodar o analisador passando o argumento “sta”, são esperados as tabelas de variáveis e de a tabela de função. Vemos na Figura 1 o resultado com as duas tabelas.

```
TABELA DE FUNÇÕES:
      tipo         nome                       parametros  linha_inicio  linha_fim
0    VAZIO    principal                               []             3         12
1  INTEIRO  somaNumeros  [[inteiro, n1], [inteiro, n2]]            14         16

TABELA DE VARIÁVEIS:
      tipo       nome       escope  linha  using dimensoes
0  INTEIRO  resultado       global      1   True        []
1  INTEIRO         n1    principal      4   True        []
2  INTEIRO         n2    principal      4   True        []
3  INTEIRO         n1  somaNumeros     14   True        []
4  INTEIRO         n2  somaNumeros     14   True        []
```

Figura 1. Saída com as tabelas, após executar o analisador.

As tabelas se diferem entre si em seus atributos, onde cada um deles possuem significados diferentes dependendo se for uma função ou uma variável. Os atributos da tabela de função são mais detalhados na Tabela 1.

Atributo	Definição
Tipo	*Esse atributo diz qual é o tipo de retorno da função.*
Nome	*Esse atributo nos diz qual é o nome da função.*
Parâmetros	*Esse atributo revela informações sobre os atributos da função. Caso a função não tenha nenhum atributo, ela será representada por uma lista vazia. Porém, caso a função tenha parâmetros, esse atributo irá, para cada um, dizer qual o seu tipo, e qual seu nome.*
Linha_inicio	*Esse atributo nos diz qual é a linha de início da função.*
Linha_fim	*Esse atributo nos diz qual é a linha final da função.*

Tabela 1. Detalhamento dos atributos da tabela de funções.

Já os atributos da Tabela de função, são mais detalhados na Tabela 2.

Atributo	Definição
Tipo	*Esse atributo diz qual é o tipo declarado da variável.*
Nome	*Esse atributo nos diz qual é o nome da variável.*
Escope	*Esse atributo revela qual é o escopo em que a variável foi declarada.*
Linha	*Esse é o atributo que contém informação sobre a linha em que a variável foi declarada.*
Using	*Esse é o atributo que apenas diz se a variável está sendo utilizada, ou se apenas foi declarada e não utilizada.*
Dimensoes	*Esse atributo mostra se a variável possui mais de uma dimensão. Caso ela não possuir, é representado com uma lista vazia, caso possua é representado pelo tamanho das dimensões.*

Tabela 2. Detalhamento dos atributos da tabela de variáveis.

3. Regras Semânticas

Com o auxilio da tabelas de variáveis e a tabela de função, podemos realizar a verificação das regras semânticas. As regras semânticas foram definidas pelo professor da disciplina, e elas são mostradas pela Tabela 3.

Tabela 3: Regras semânticas da linguagem T++.

Regras para:	Regras:
Funções e Procedimentos	*1. Verificar a existência de uma função principal que inicializa a execução do código.* *2. Uma função deve retornar um valor de tipo compatível com o tipo de retorno declarado.* *3. Funções precisam ser declaradas antes de serem chamadas.* *4. Uma função qualquer não pode fazer uma chamada à função principal.* *5. Uma função pode ser declarada e não utilizada.* *6. Avisar caso uma função faça uma chamada para ela mesmo.*
Variáveis	*1. Variáveis devem ser declaradas e inicializadas e antes de serem utilizadas.* *2. Não deve ser permitido uma tentativa de leitura ou escrita de qualquer variável não declarada.* *3. Não deve ser permitido uma tentativa de leitura de uma variável que foi declarada, mas não foi inicializada.* *4. Não deve ser permitido uma variável declarada duas vezes no mesmo escopo.*

Atribuição	*1. Na atribuição devem ser verificados se os tipos são compatíveis.* *2. Devem ser verificados se a atribuição de um retorno de uma função for compatível com o tipo da variável que está recebendo.*
Coerções implícitas	*1. Se houver uma coerção implícita de tipos (inteiro<->flutuante).*
Arranjos	*1. Índice de um arranjo deve ser do tipo inteiro.* *2. Se o acesso ao elemento do arranjo estiver fora de sua definição.*

4. Arvore Sintática Abstrata

Após criar as tabelas de variáveis e de funções, é feito todo o processo de verificação das regras semânticas. Para esse processo foram feitas 3 funções principais, que são mostradas no Código 2.

Código 2. Funções que realizam a análise semântica.

```
def verifyFunctions(dataPD, functionsPD, variablesPD, errors):
    verifyFunctionsLine(dataPD, functionsPD, errors)
    verifyFunctionsRepeat(dataPD, functionsPD, errors)
    verifyFunctionReturn(dataPD, functionsPD, variablesPD, errors)

def verifyVariables(dataPD, functionsPD, variablesPD, errors):
    verifyVariableUse(dataPD, variablesPD, errors)

def verifyAssignment(dataPD, functionsPD, variablesPD, errors):
    verifyAssignmentValues(dataPD, functionsPD, variablesPD, errors)
    verifyFunctionAssignmentValues(dataPD, functionsPD, variablesPD,
        errors)
```

Essas funções são as responsáveis por realizar a maioria das verificações das regras semânticas listadas acima, na Tabela 3. Após todo o processo, é gerado uma árvore sintática abstrata que assim como a árvore da análise sintática, é uma estrutura de dados em árvore que representa estruturas sintáticas de cadeias, de acordo com a gramática formal definida na análise anterior. A diferença entre as duas, é que os nós da arvore sintática abstrata são diretamente valorados em seus símbolos terminais, não havendo portanto a representação das derivações por meio dos símbolos não terminais. Para criar essa arvore sintática abstrata, é necessário fazer a poda da arvore sintática gerada na análise sintática, para isso foi criado as funções mostradas no Código 3.

Código 3. Funções que realizam a poda na árvore.

```
def remove(no):

    # pego o pai do no
    pai = no.parent

    # var auxiliar
    aux = []
```

```
8
9       # para cada um dos filhos do pai
10      for i in range(len(pai.children)):
11
12          # se o filho for o no que quero remover
13          if (pai.children[i].name == no.name):
14              # concatenos os filhos
15              aux += no.children
16
17          else:
18              # adiciono na lista auxiliar
19              aux.append(pai.children[i])
20
21      # adiciono a lista auxiliar ao pai
22      pai.children = aux
23
24  def podaAux(root):
25
26      # para cada um dos nos
27      for no in root.children:
28          podaAux(no)
29
30      # remove o no
31      if root.name in nosRemove:
32          remove(root)
33
34      # remove caso nao tenha nenhum filho
35      if root.name == 'corpo' or root.name == 'retorna' or root.name == '
            escreva' or root.name == 'se' or root.name == 'repita' or root.
            name == 'ate' or root.name == 'leia':
36          if len(root.children) == 0:
37              remove(root)
```

A função `podaAux(root)` passa por todos os nós da árvore, e caso encontre um nó que tenha que ser removido (definido na lista "nosRemove"), ele realiza a função de remove-lo. Essa função, chamada `remove(no)` adiciona todos os filhos em uma lista, e faz o pai do nó receber essa lista, removendo assim esse nó. E dessa forma, ao final do processo é gerado uma arvore sintática abstrata.

5. Exemplos

Para testar a implementação do analisador semântico, iremos utilizar como exemplo o Código 4 de entrada ao analisador semântico.

Código 4. Função de entrada para o analisador.

```
1 inteiro: a
2 inteiro: b
3
4 inteiro principal()
5     a := c + 1
6 fim
```

Ao usar esse código no analisador, deverá retornar 2 erros e 1 aviso. Eles são: erro pela função principal retornar vazio quando deveria retornar inteiro, erro pela variável

“c”não ter sido declarada e estar sendo usada, e por fim, 1 aviso sobre a variável b não ser utilizada. Conseguimos ver a saída após rodar o analisador na Figura 2

```
als-v@DESKTOP-NRMU9ML:~/Compiladores/2. Compilador/3. Análise Semântica$ python3 code.py code.tpp
Generating LALR tables
WARNING: 54 shift/reduce conflicts

Erro: Função “principal” deveria retornar inteiro, mas retorna vazio
Erro: Variável “c” não declarada

Aviso: Variável “b” declarada, mas não utilizada
```

Figura 2. Erros e avisos de saída do analisador.

Ao passar os parâmetros “sta”, podemos ver as tabelas de variáveis e as de função. Vemos a saída na Figura 3.

```
TABELA DE FUNÇÕES:
      tipo       nome parametros  linha_inicio  linha_fim
0  INTEIRO  principal         []             4          6

TABELA DE VARIÁVEIS:
      tipo nome  escope  linha  using dimensoes
0  INTEIRO    a  global      1   True        []
1  INTEIRO    b  global      2  False        []
```

Figura 3. Saída com as tabelas.

E ao final do processo, é gerado a arvore sintática abstrata, e podemos ver ela na Figura 4

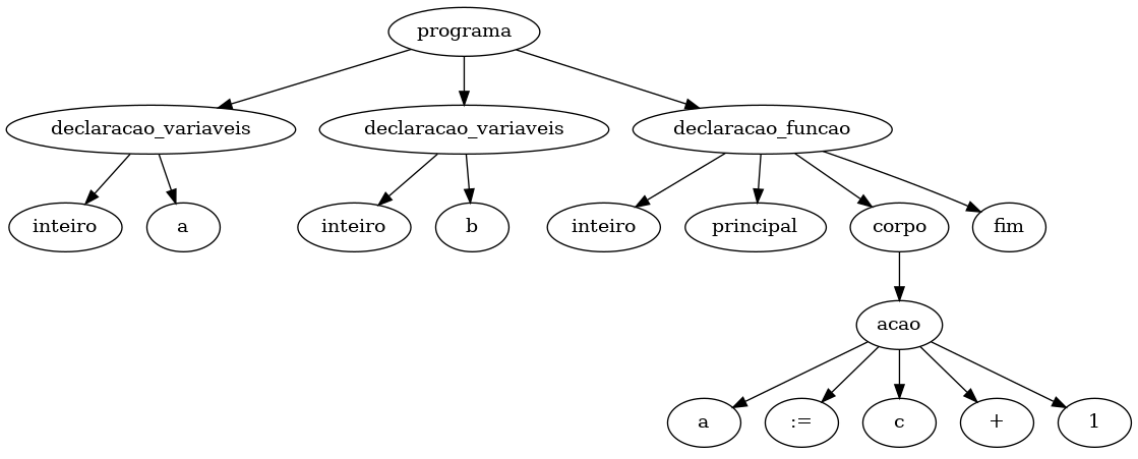

Figura 4. Arvore Sintática Abstrata.

Referências

Gonçalves, R. A. Conjunto de testes. ttps://moodle.utfpr.edu.br/mod/resource/view.pp?id=136961.

Gonçalves, R. A. Slides - análise semântica. ttps://moodle.utfpr.edu.br/pluginfile.pp/1373325/mod$_r esource/content/0/aula - 16 - analise - semantica - regras - semanticas - tpp.md.slides.pdf$.

Wikipedia. Arvore sintatica abstrata. ttps://pt.wikipedia.org/wiki/%C3%81rvore_sint%C3%A1tica_abstrata.\end{tebibliography