Análise Sintática

Wesley Franco Ferreira

[1]Universidade Tecnológica Federal do Paraná (UTFPR)

wesley2ff@gmail.com

Abstract. *The text described in the next sections will explain one of the steps in the process of running a compiler, the syntax analysis for the T ++ language. It will clarify basic concepts and the implementation of the analyzer, in sequence results and tests performed.*

Resumo. *O texto descrito nas próximas seções irá abordar uma das etapas do processo de funcionamento de um compilador denominado análise sintática para a linguagem T++. Será esclarecido conceitos básicos e a implementação do analisador, em sequência resultados e testes realizados.*

1. Introdução

A Análise Sintática é uma etapa fundamental da construção de um compilador é a análise sintática, no qual possui o principal objetivo verificar se, para uma determinada gramática livre de contexto, uma cadeia qualquer pertença à linguagem da gramática, utilizando os *tokens* da análise léxica, é possível construir uma estrutura de árvore com base no código de entrada, o qual abordaremos com mais detalhes na seção 2, juntamente com conceitos mais detalhados para o melhor entendimento do tema.

2. Gramática Livre de Contexto

Gramática livre de contexto é composta por um conjunto de símbolos terminais, denotado por letra minúscula, um conjunto de símbolos não terminais, denotado por letras maiúsculas, além de um não terminal inicial, um conjunto de produções e a própria linguagem denotada pela gramática.

Para gerar uma cadeia é iniciado com o simbolo não-terminal inicial, em sequência, é substituído por um símbolo não-terminal presente na cadeia por uma de suas regras. Gramáticas livres de contexto são utilizadas devido a diversas possibilidades de construção de cadeias, diferente das expressões regulares, onde existem certas limitações, tal como precedência de operadores. Um problema que deve ser dado atenção é a ambiguidade de uma gramática livre de contexto, caso exista alguma cadeia para qual ela tem mais de uma árvore sintática, a mesma deve ser modificada, gerando assim uma inconsistência entre diferentes compiladores.

3. Análise Sintática

O principal objetivo da análise sintática é encontrar os erros sintáticos através do uso de uma gramática livre de contexto, verificando se a entrada está de acordo com a especificação da linguagem. Sua função é analisar o código de entrada e criar uma estrutura de árvore sintática para facilitar a verificação de erros no código-fonte.

Existem duas formas de realizar a análise sintática tomando base na árvore gerada. O método *top-down* utiliza a raiz da árvore é derivada até as folhas serem alcançadas, já o método *bottom-up* começa das folhas e vai até a raiz. A técnica de *shift reduce* utiliza o método de *bottom-up* sendo ilustrado na Figura 1.

Step	Symbol Stack	Input Tokens	Action
1		3 + 5 * (10 - 20)$	Shift 3
2	3	+ 5 * (10 - 20)$	Reduce factor : NUMBER
3	factor	+ 5 * (10 - 20)$	Reduce term : factor
4	term	+ 5 * (10 - 20)$	Reduce expr : term
5	expr	+ 5 * (10 - 20)$	Shift +
6	expr +	5 * (10 - 20)$	Shift 5
7	expr + 5	* (10 - 20)$	Reduce factor : NUMBER
8	expr + factor	* (10 - 20)$	Reduce term : factor
9	expr + term	* (10 - 20)$	Shift *
10	expr + term *	(10 - 20)$	Shift (
11	expr + term * (	10 - 20)$	Shift 10
12	expr + term * (10	- 20)$	Reduce factor : NUMBER
13	expr + term * (factor	- 20)$	Reduce term : factor
14	expr + term * (term	- 20)$	Reduce expr : term
15	expr + term * (expr	- 20)$	Shift -
16	expr + term * (expr -	20)$	Shift 20
17	expr + term * (expr - 20	)$	Reduce factor : NUMBER
18	expr + term * (expr - factor	)$	Reduce term : factor
19	expr + term * (expr - term	)$	Reduce expr : expr - term
20	expr + term * (expr	)$	Shift)
21	expr + term * (expr)	$	Reduce factor : (expr)
22	expr + term * factor	$	Reduce term : term * factor
23	expr + term	$	Reduce expr : expr + term
24	expr	$	Reduce expr
25		$	Success!

Figure 1. Tecnica de Shift reduce.

A API emprega o método Look ahead left-to-right (LALR) para otimização do uso a árvore sintática, melhorando assim seu desempenho, sendo um variante do left-to-right (LR) expandindo os nós prioritariamente a direita e depois da esquerda. Para a geração das estruturas, foi utilizado uma tabela EBNF especificada anteriormente, o qual basicamente é descrita pela gramática livre de contexto.

4. Materiais

Python foi a linguagem determinada para a implementação por conter certas bibliotecas que ajudariam a implementação ser mais simples, como por exemplo, a biblioteca Ply, no qual foi utilizado para a detecção de *tokens* através de expressões regulares, além da api *YACC* e *Graphviz* para representação gráfica da arvore construída.

5. Implementação

Uma estrutura para a árvore sintática é definida, o qual armazenaria o tipo do nó, sendo definido com base na gramática da linguagem, um vetor de filhos e por final um valor, tomando base para o funcionamento do algoritmo. Para construir a árvore sintática, foi implementado diversas funções obedecendo a tabela EBNF da linguagem T++ obedecendo o padrão determinado pelo *Ply*. A definição da estrutura da árvore é representada na Figura 2.

```python
class Tree:
    def __init__(self, type_node='', child=[], value='', lineno=''):
        self.type = type_node
        self.child = child
        self.value = value
        self.lineno= lineno
```

Figure 2. Definição da estrutura.

```
1   inteiro principal()
2       inteiro: a
3       a : 5
4   fim
```

Figure 3. Exemplo de código com erro sintático.

6. Resultados e Conclusão

Por fim, após realizar testes com diferentes arquivos contendo diferentes erros sintáticos, tais quais, comentários incompletos, escopos de funções e inicialização e declaração de variáveis feitas de forma errada sendo representados a seguir, resultando em uma saıda a qual indicava o erro. Caso erros sejam encontrados erros a execução é terminada e o erro identificado. Um exemplo de saída em caso de erro é ilustrado na Figura 3 e a saída sendo ilustrado na Figura 4.

```
Erro Sintatico na linha 3
Erro na lista de variáveis
```

Figure 4. Exemplo de saída em caso de erro sintático.

Testes com arquivos sem erros sintáticos foram realizados, obtendo como saıda uma visualização gráfica da árvore gerada demonstrada na Figura 5.

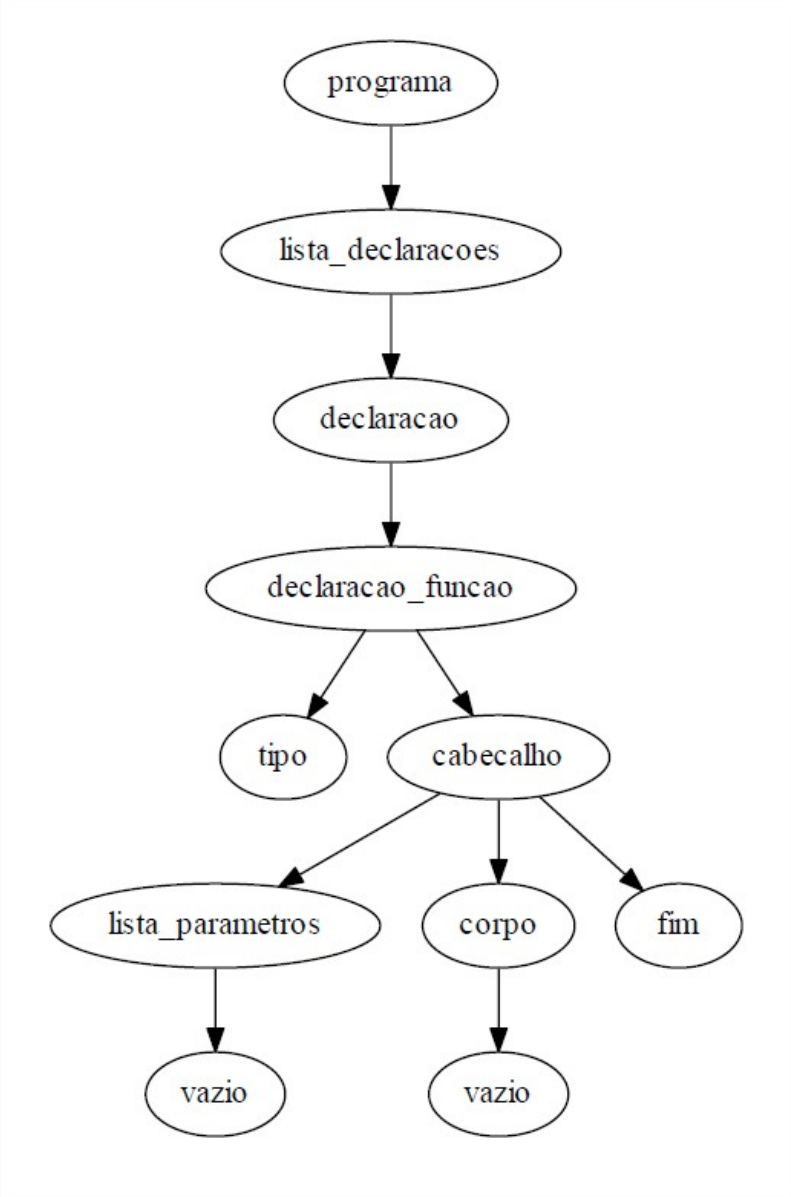

Figure 5. Exemplo de árvore de saída em caso código sem erro sintático.