Análise Sintática

Mateus Tomoo Yonemoto Peixoto

DACOM – Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 271 – 87301-899 – Campo Mourão – PR – Brazil
{mateustomoo}@gmail.com

Abstract. This paper describes the development of the sintactic analysis for a compiler being designed for the T++ programming language. In the future, it will so also approach the development of the semantic analysis.

Resumo. Este artigo descreve o desenvolvimento da análise sintática para um compilador projetado para a linguagem de programação T++. No futuro, também abordará o desenvolvimento da análise semântica.

1. Introdução

Um compilador exige alguns passos até que um código final seja obtido. Esses passos são divididos em análise léxica, análise sintática, análise semântica e geração de código.

A Análise Sintática é responsável por determinar a estrutura sintática de um programa a partir dos tokens separados na primeira etapa (análise léxica). Ou seja, irá verificar se as sentenças são válidas para a linguagem de programação, no caso T++.

Para o desenvolvimento deste trabalho, foi utilizado a linguagem de programação Python (versão 3.6.2), onde possui a biblioteca chamada PLY, que contém ferramentas léxicas e sintáticas. Também foi utilizado o Python YACC.

2. A linguagem

A linguagem T++ possui algumas características, dentre elas são:

- Tipos básicos de dados suportado: inteiro e flutuante;
- Suporte a arranjos uni e bidimensionais (arrays);
- Variáveis globais e locais devem ter um dos tipos especificados;
- Linguagem quase fortemente tipificada: nem todos os erros são especificados, mas sempre deve ocorrer avisos.

3. Análise Sintática

A gramática no padrão BNF utilizada foi:

programa ::=	lista_declaracoes
lista_declaracoes ::=	lista_declaracoes declaracao | declaracao
declaracao ::=	declaracao_variaveis | inicializacao_variaveis | declaracao_funcao
declaracao_variaveis ::=	tipo ":" lista_variaveis
inicializacao_variaveis ::=	atribuicao
lista_variaveis ::=	lista_variaveis "," var ~~| lista_variaveis "," atribuicao~~ ~~| atribuicao~~ | var
var ::=	ID | ID indice
indice ::=	indice "[" expressao "]" | "[" expressao "]"

tipo ::=	INTEIRO | FLUTUANTE
declaracao_funcao ::=	tipo cabecalho | cabecalho
cabecalho ::=	ID "(" lista_parametros ")" corpo FIM
lista_parametros ::=	lista_parametros "," parametro | parametro | vazio
parametro ::=	tipo ":" ID | parametro "[" "]"
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
atribuicao ::=	var ":=" expressao
leia ::=	LEIA "(" ID ")"
escreva ::=	ESCREVA "(" expressao ")"
retorna ::=	RETORNA "(" expressao ")"
expressao ::=	expressao_simples | atribuicao
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_relacional ::=	"<" | ">" | "=" | "<>" | "<=" | ">="
operador_soma ::=	"+" | "-"
operador_multiplicacao ::=	"*" | "/"
fator ::=	"(" expressao ")" | var | chamada_funcao | numero
numero ::=	NUM_INTEIRO | NUM_PONTO_FLUTUANTE | NUM_NOTACAO_CIENTIFICA
chamada_funcao ::=	ID "(" lista_argumentos ")"
lista_argumentos ::=	lista_argumentos "," expressao | expressao

	| vazio

3.1. Formato da Análise Sintática

O formato na Análise Sintática realizado pela ferramenta é a LR.

O formato LR é do tipo bottom-up, onde o L significa que o analisador lê o texto de entrada da esquerda para a direita e o R significa que o analisador produz uma derivação mais a direita em sentido inverso, ou seja, faz uma análise bottom-up. O padrão LR normalmente é seguido por um número, como por exemplo LR(1) e muitas vezes é precedido por outros qualificadores, como o LALR.

O padrão LR é determinístico, ou seja, produz uma única análise sem adivinhações. Isso ocorre pois ele aguarda até ter visto uma instância completa de alguma gramática antes de se comprometer com o que encontrou, diferente de um LL que deve decidir ou adivinhar o que está vendo muito mais cedo.

Outra característica do LR é que lidam com uma gama maior de linguagens e gramáticas em relação à por exemplo o LL (top-down).

3.2. YACC

A ferramenta YACC é uma ferramenta que a biblioteca PLY nos permite utilizar como analisador sintático. Para implementação de tal, foi utilizado os seguintes comandos:

```
parser = yacc.yacc(debug=False, module=self, optimize=False)
self.ast = parser.parse(code)
```

A primeira linha em questão se remete a invocação da ferramenta YACC, gerando o "parser" e armazenando na variável parser. A segunda linha utiliza a função parse da variável parser, passando como parâmetro um código, onde esse código será analisado sintaticamente e armazenado.

3.3. Árvore Sintática

Após a utilização da ferramenta YACC e implementar a gramática da linguagem, uma árvore sintática é gerada. Essa árvore possui tipos de nós, os filhos e seus valores. Para a impressão da árvore, foi criado uma função print_tree, onde por parâmetro é passado os nós dessa árvore, no caso a análise sintática feita pelo parser mostrado na sessão anterior.

3.4. Saída

A seguir é mostrado um exemplo de código em T++ e sua respectiva saída sintática.

```
1   inteiro: a[1024]
2
3   inteiro func(inteiro: a[])
4       retorna(0)
5   fim
6
7   inteiro principal()
8       inteiro: i,b
9
10      i:= 1
11
12      a[i]:= b := 0
13
14      b := b + func(a)
15
16      b := +b + i
17
18      retorna(0)
19  fim
```

```
- programa
-- lista_declaracoes
--- lista_declaracoes
---- lista_declaracoes
----- declaracao
------ declaracao_variaveis
------- tipo
------- lista_variaveis
-------- var a
--------- indice
---------- expressao
----------- expressao_simples
------------ expressao_aditiva
------------- expressao_multiplicativa
-------------- expressao_unaria
--------------- fator
---------------- numero 1024
---- declaracao
----- declaracao_funcao
------ tipo
------ cabecalho func
------- lista_parametros
-------- parametro
--------- parametro a
---------- tipo
```

```
-------- corpo
-------- acao
--------- retorna
---------- expressao
----------- expressao_simples
------------ expressao_aditiva
------------- expressao_multiplicativa
-------------- expressao_unaria
--------------- fator
---------------- numero 0
--- declaracao
---- declaracao_funcao
----- tipo
----- cabecalho principal
------ lista_parametros
------ corpo
------- corpo
-------- corpo
--------- corpo
---------- corpo
----------- corpo
------------ corpo
------------ acao
------------- declaracao_variaveis
-------------- tipo
```

```
-------------- tipo
--------------- lista_variaveis
---------------- lista_variaveis
----------------- var i
---------------- var b
----------- acao
------------ expressao
------------- atribuicao
-------------- var i
-------------- expressao
--------------- expressao_simples
---------------- expressao_aditiva
----------------- expressao_multiplicativa
------------------ expressao_unaria
------------------- fator
-------------------- numero 1
---------- acao
----------- expressao
------------ atribuicao
------------- var a
-------------- indice
--------------- expressao
---------------- expressao_simples
----------------- expressao_aditiva
------------------ expressao_multiplicativa
------------------- expressao_unaria
-------------------- fator
--------------------- var i
------------- expressao
-------------- atribuicao
--------------- var b
```

4. Referências

LOUDEN, Kenneth C. Compiladores: princípios e práticas. São Paulo, SP: Thomson, c2004. xiv, 569 p. ISBN 8522104220.

http://hackingoff.com/compilers/regular-expression-to-nfa-dfa

PLY (Python Lex-Yacc). Disponível em: <http://www.dabeaz.com/ply/>.