Projeto de compiladores : Análise sintática

Edson Lemes da Silva[1]

[1]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – 89.802-112 – Chapecó – SC – Brasil

edson_lemes@live.com

Abstract. *This paper describes some theorists for a syntactic analysis in the compilation process. Conversely,it presents the definition of a hypothetical language and the syntactic analysis for the recognition of allowed structures. In this way, the development details and their particularities are presented.*

Resumo. *Este trabalho descreve alguns aspectos teóricos para a análise sintática no processo de compilação. Por outro lado, apresenta a definição de uma linguagem hipotética e a análise sintática para o reconhecimento das estruturas permitidas. Desta forma, são apresentados os detalhes de desenvolvimento e suas particularidades.*

1. Introdução

Apos a realização da análise léxica no processo anterior. A próxima ação é fazer a análise sintática do código-fonte de entrada. Nesta fase, são verificados se os tokens lidos formam estruturas permitidas e descritas pela gramática livre de contexto (GLC) da linguagem. Esse processo utiliza uma tabela de *Parsing* (ou sintática) construída a partir da GLC de entrada.

A aplicação descrita aqui atende a leitura da tabela, leitura dos tokens identificados, posteriormente é feito a análise sintática, verificando as estruturas lidas. A linguagem de programação usada é o Python, juntamente com algumas classes que já estão implementadas.

2. Análise sintática

A análise sintática é a segunda etapa do processo de compilação, ela verifica se as estruturas descritas pela linguagem estão sendo respeitadas. Uma estrutura sintática, pode ser representada através da construção de uma GLC.

Para o funcionamento deste procedimento, o analisador sintático recebe uma cadeia de tokens reconhecidos pelo analisador léxico, e verifica se a mesma pode ser gerada pela gramática da linguagem [Aho et al. 2008, p. 72].

Os principais métodos de analisadores sintáticos são classificados como: ascendentes e descendentes. O primeiro tipo constrói a árvore de derivação a partir das folhas até a raiz; o tipo descendente analisa a partir da raiz. Ambos trabalham com a leitura de símbolos da esquerda para direita.

Os analisadores descendentes descrevem a árvore de derivação por um método chamado de analisadores sintáticos preditivos. Esse tipo de análise busca encontrar uma

derivação mais à esquerda. Desta maneira, pode haver a recursividade quando a gramática descrita é recursiva à esquerda, levando a um laço infinito.

Em muitos casos, quando não há recursividade à esquerda, a gramática pode ser processada por um analisador sintático preditivo. O fluxo deste método consegue detectar os símbolos distintos de cada parcial da gramática. Este método necessita da construção da tabela sintática preditiva, através da definição dos conjuntos *first* e *follow*.

Por outro lado, estão os analisadores ascendentes, também conhecidos como análise empilhar e reduzir. Um método geral para este caso é descrito como analisadores LR. Assim, constrói a derivação a partir dos símbolos identificados, até obter a regra inicial da gramática (redução).

Os analisadores LR podem ser usados para decompor uma ampla classe de GLCs. A varredura é feito da esquerda para direita, construindo uma derivação mais à direita e definindo um valor de *lookahead*. Esses analisadores também necessitam da construção de uma tabela sintática. Existem alguns métodos para isto, entre eles estão o: LR simples (SLR) e LR *lookahead* (LALR).

A construção da tabela sintática possui algumas ações possíveis, entre elas estão: empilhar (*shift*), reduzir (*reduce*) e desvio (*goto*). A primeira coloca na pilha o próxima token sendo analisado. A segunda reduz conforme as regras da gramática e a última faz a transição para um estado específico. Estas ações são utilizadas para a implementação da tabela de SLR e LALR.

O método SLR primeiramente constrói o conjunto de itens válidos para obter as operações de empilhamento. Em seguida, a partir dos conjuntos *first* e *follow*, são descritos na tabela SLR as operações de redução.

A construção da tabela LALR permite tratar algumas situações que o SLR não consegue. Entre elas: pode-se fazer o agrupamento de estados que se diferem apenas no parâmetro de *lookahead*. A construção desta tabela é baseado no analisador LR canônicos como forma de permitir a compactação de estados.

3. A aplicação

Este projeto implementa a análise sintática de uma linguagem hipotética a partir de um conjunto de tokens definidos. Neste trabalho, é feito o uso do algoritmo de análise léxica implementado no projeto 1, com a leitura do autômato gerado pela ferramento Gold Parser.

A aplicação lê a fita de saída da análise léxica, para fins de análise sintática. Então, a partir do arquivo construído pelo Gold Parser, é feito a instanciação da tabela de *Parsing* gerada pela gramática livre de contexto (GLC) da linguagem definida.

A saída do programa é o reconhecimento ou não da estrutura sintática descrita no código-fonte.

4. Uma linguagem hipotética

A linguagem utilizada para a análise léxica e para as etapas seguintes, possui os seguintes tokens definidos: while, if, then, else, begin, end, continue, break, true e false. Os identi-

ficadores são descritos apenas pelas letras: a,e,i,o e u. Assim, para as variáveis apenas a combinação destas cinco letras são permitidas.

O escopo de um código-fonte esta delimitado pelos tokens: “begin e “end”. Deste modo, todos os outros tokens devem estar entre estas duas palavras reservadas.

A linguagem permite utilizar estrutura de repetição, de condição e atribuição de variáveis. A palavra “then” é usada exclusivamente com a estrutura de condição. Para a estrutura de repetição, as palavras “continue” e “break” podem ser utilizadas após uma operação qualquer.

A declaração de variável não requer a definição de tipo. Assim, a declaração requer apenas que seja feito a atribuição de algum valor.

```
1 begin
    a = 77
3   while a > 4

5     if a < 80 then a = 2
      else e = a + 1
7

9 end
```

Exemplo 1. Estrutura permitida

5. A implementação

Inicialmente, é preciso descrever o ambiente de programação. A linguagem utilizada para a aplicação, foi o Python; utilizando-se de alguns módulos (classes) já implementados para esta linguagem.

Como forma de compatibilidade, esta aplicação foi desenvolvida sob a versão 2.7.12 do python. Os módulos necessários para a execução são: `collections`, `copy`, `itertools e`. Além destes, também é necessário importar o módulo que implementa a análise léxica : `lex`, assim como a classe *Pilha*.

Na implementação, a análise sintática esta descrita por uma classe, chamada de `styxAlg`. Uma instância desta classe, recebe dois parâmetros do tipo `String`, que representam a gramática da linguagem e o código-fonte para verificação, respectivamente.

A função principal que executa a análise sintática é chamada de *syntaxAnalysis*.Ela recebe como parâmetro um valor booleano que representa a ativação ou não do *debug* na análise, a partir da tabela já carregada na memória.

Primeiramente, a função *readGOLDParserFile* lê o arquivo gerado a partir da GLC pelo GoldParser. Para agilizar o processo de construção do projeto, o autômato finito utilizado pela análise léxica nesta etapa está descrito nos estados lidos do arquivo de saída da ferramenta, e não mais pela implementação utilizada no projeto 1.

A tabela carregada pela aplicação é descrita por algumas estruturas de *lists*. Assim, cada estado da tabela é composto por duas listas: a primeira representa os símbolos do alfabeto da linguagem, e a segunda os símbolos da pilha. As demais informações lidas do arquivo de entrada, tais como: produções da gramática, autômato finito, terminais e não-terminais são armazenados em estruturas do tipo: *dict*.

Para a definição da pilha de controle do analisador, a aplicação utiliza a classe *Pilha*, ela implementa uma lista de elementos que representam o funcionamento de uma pilha. Deste modo, cada componente é descrito por duas informações: o símbolo acrescentado na pilha e o seu estado respectivo (na tabela de Parsing).

A função *syntaxAnalysis* inicializa a pilha contendo um único elemento, descrevendo o estado zero, que não corresponde a nenhum símbolo específico. Então, a partir dos tokens da fita oriunda da análise léxica, é feito a verificação da tabela para os tokens lidos. Após a identificação da próxima operação para um elemento qualquer, inicia-se os processos : empilhar, reduzir, aceitar ou tratamento de erro.

A operação de empilhar (*shift*) é identifica pela letra "s" na tabela, cujo o propósito é fazer a inclusão de um símbolo na pilha.

Quando há uma operação de redução (*reduce*), primeiramente remove-se da pilha a quantidade de elementos igual ao tamanho da produção. Posteriormente inclui-se novamente na pilha, o símbolo apontado pela tabela do item no topo da pilha.

O estado que descreve a aceitação da cadeia de tokens, é descrito na tabela pela letra "a", assim quando isso acontece, significa que a estrutura esta de acordo com que a gramática permite.

O tratamento de erro neste caso, aponta o local e a linha onde ocorreu este erro. Deste modo, é utilizado o símbolos que cada estado permite para indicar os tokens esperados.

Como forma de depuração, a aplicação permite imprimir na tela todos os elementos que estão na pilha em tempo de execução. A saída do analisador sintático é a aceitação da cadeia, ou o apontamento de erros encontrados.

5.1. Execução do algoritmo

Um exemplo de código-fonte está escrito no arquivo *test.txt*. A gramática da linguagem no *gramatica.txt*, e o arquivo gerado pelo GoldParser está descrito em: *table.txt*.

A execução do algoritmo pode ser feita a partir do interpretador do Python. Primeiramente, deve-se importar o arquivos que contém as classes implementadas, que estão nos arquivos "ap.py" e no "scc.py". O próximo passo é criar uma instância para a classe *styxAlg*, passando como parâmetro a gramática e o código-fonte.

O construtor da classe se encarrega de instanciar e executar o algoritmo de análise léxica, assim como a leitura e construção das tabelas e controles necessários.

A saída do programa apresenta eventuais mensagens de erro.

```
import sys
from collections import defaultdict
from lex import *
from ap import Pilha

test = styxAlg('table.txt','test.txt')
test.syntaxAnalysis(True)
```

Exemplo 2. Execução do algoritmo.

6. Conclusão

No trabalho foi demonstrado uma implementação de um analisador sintático gerado pela análise de uma GLC que representam identificadores, delimitadores ou palavras reservadas de uma linguagem hipotética.

A aplicação desenvolvida utiliza uma modificação do analisador léxico implementado na primeira parte do trabalho. A construção da tabela de *parsing* foi feito com o uso da ferramento GoldParser. Toda a implementação foi desenvolvida na linguagem Python.

A aplicação construída nesta etapa do projeto permite fazer o reconhecimento sintático a partir da GLC descrita, porém não faz recuperações parciais das estruturas durante o processo de análise, isto é, identifica-se apenas erros sintáticos individuais, mas não todos eles.

A documentação sobre a implementação descreveu as funções principais de forma detalhada, assim como no código-fonte, onde estão descritas de forma resumida as funcionalidades de cada uma delas.

Referências

Aho, A., Sethi, R., and Lam, S. (2008). *Compiladores: princípios, técnicas e ferramentas*. Addison-Wesley.