Atividade Compiladores 2021.1

Aluno: Victor Ferreira Teixeira da Silva
Matrícula: 114031056
data: 23/07/2021

Objetivo

Mostre na prática que a tabela de parsing SLR é ambígua e que a LALR não é utilizando o gerador de analisador sintático PLY, uma implementação em Python da ferramenta Yacc. Dada a gramática:

```
S -> L = R | R
L -> * R | id
R -> L
```

Iniciamos com a implementação do lexer que é necessária para realizar a interpretação dos tokens:

```
1  from ply.lex import lex
2  from ply.yacc import yacc
3
4  tokens = ('ID', 'MULT', 'EQ')
5  t_ID = r'id'
6  t_EQ = r'='
7  t_MULT = r'\*'
8  t_ignore = ' \t'
9
10
11 def t_error(t):
12     print("Caractere '%s' inválido" % t.value[0])
13     t.lexer.skip(1)
14
15 lexer = lex()
```

o formato das variáveis utilizadas para denotar cada token é segue o formato padrão da biblioteca PLY, sendo este ***t_NOMETOKEN***

As funções a seguir definem a representam a gramática inicialmente definida pela linguagem, também seguindo o formato padrão PLY de nomeclatura.

```
16
17    def p_expression_L_EQ_R(p):
18        """
19        S : L EQ R
20        """
21
22    def p_expression_R(p):
23        """
24        S : R
25        """
26
27    def p_L_MULT_R(p):
28        """
29        L : MULT R
30        """
31
32    def p_L_ID(p):
33        """
34        L : ID
35        """
36
37    def p_R_L(p):
38        """
39        R : L
40        """
41
42    def p_error(p):
43        print(p)
44
```

por fim os parsers são gerados, um aplicando SLR e outro LALR permitindo validar evidenciar a ambiguidade da no parser SLR.

```
45
46    if __name__ == '__main__':
47        slr_parser = yacc(method='SLR', debugfile='slr_parser.out', tabmodule='slr_parsetab')
48        lalr_parser = yacc(method='LALR', debugfile='lalr_parser.out', tabmodule='lalr_parsetab')
```

O output ao executar o programa é

```
❯ python3 slrvslalr.py
Generating SLR tables
WARNING: 1 shift/reduce conflict
Generating LALR tables
```

Este conflito ***shift reduce*** evidencia o problema ao aplicar o parser SLR, ao inspecionar a saída gerada no arquivo **slr_parser.out** temos:

```
48
49  state 2
50
51      (1) S -> L . EQ R
52      (5) R -> L .
53
54    ! shift/reduce conflict for EQ resolved as shift
55      EQ              shift and go to state 6
56      $end            reduce using rule 5 (R -> L .)
57
58    ! EQ              [ reduce using rule 5 (R -> L .) ]
59
60
```

Como a possibilidade de executar o ***shift*** leva a um estado, e o executar o ***reduce*** leva a outro temos a evidência do conflito. Verificando o este mesmo estado, porém no parser LALR ***(arquivo lalr_parser.out)*** temos:

```
state 2

    (1) S -> L . EQ R
    (5) R -> L .

    EQ              shift and go to state 6
    $end            reduce using rule 5 (R -> L .)
```

Aqui a ambiguidade oriunda de caminhos distintos ao executar o **shift** ou **reduce** não está presente. Deste modo podemos evidenciar que a utilização do parser SLR apresenta ambiguidade enquanto esta gramática é interpretada corretamente pelo parser LALR.