Aluno: Gustavo Muller Moreira - 23/07/2021

O problema é que o parser SLR é menos poderoso que o LALR, fazendo com que em determinadas circunstâncias o primeiro gere tabelas que contenham conflitos do tipo shift/reduce, isto é, não é possível, através da tabela de parsing, decidir se a próxima ação dado o símbolo da pilha e o estado atual deva ser shift ou reduce.

Isso acontece porque SLR é um parser que utiliza o cálculo do FOLLOW para tomar decisões sobre reduce, já o LALR leva em consideração um símbolo *lookahead* presente nos itens que compõem os estados, o qual carrega mais informações contextuais e auxilia a resolução de conflitos do tipo shift/reduce.

Gramática:

```
Grammar

Rule 0     S' -> S
Rule 1     S -> L EQUAL R
Rule 2     S -> R
Rule 3     L -> STAR R
Rule 4     L -> ID
Rule 5     R -> L
```

SLR:

Uma mensagem de *warning* é exibida no terminal assim que o script é executado:

```
gustavo@gustavo-dell:~/code/compiladores/slr-vs-lalr-GMMULLER$ python3 slrvslalr.py
Generating SLR tables
WARNING: 1 shift/reduce conflict
```

E no fim do arquivo “parser.out” vemos a seguinte mensagem de *warning*:

```
WARNING:
WARNING: Conflicts:
WARNING:
WARNING: shift/reduce conflict for EQUAL in state 2 resolved as shift
```

O que indica que houve um conflito entre as ações shift e reduce por causa do símbolo ‘=’ no estado 2. Essa implementação do parser decidiu resolver o conflito escolhendo pela ação shift.

No mesmo arquivo, na descrição do estado 2, temos a seguinte saída:

```
state 2

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

  ! shift/reduce conflict for EQUAL resolved as shift
    EQUAL           shift and go to state 6
    $end            reduce using rule 5 (R -> L .)

  ! EQUAL           [ reduce using rule 5 (R -> L .) ]
```

O conflito surge por causa da seguinte situação:

Por causa do item (1) devemos calcular GOTO(2,EQUAL) = 6, então ACTION(2,EQUAL) = shift 6.

Por outro lado, por causa do item (2), devemos calcular o FOLLOW(R) = {$,EQUAL}, dessa forma ACTION(2,EQUAL) = reduce usando ‘R -> L’. Essa possibilidade foi descartada e isso é sinalizado pelo símbolo ‘!’ na última linha da imagem.

LALR:

```
state 2

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

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

Podemos ver que o mesmo conflito não acontece usando o parser LALR, isto porque o símbolo *lookahead* de (5) não é EQUAL, eliminando a ambiguidade da tabela.