Aquery to Q Compiler: Shift/Reduce Issues Remaining

Jose Cambronero

02/05/2015

1 Introduction

The below 2 examples highlight the remaining shift-reduce conflicts in the current grammar. In each we show the current output from bison -v and an example with 2 possible parse trees. We highlight that we may skip intermediate non-terminals in rules that are solely there for precedence encoding. When we do so, we represent them as ellipses in our trees.

2 Shift/Reduce Conflicts

2.1 Parenthesized Value Expression vs Singleton Value Expression List

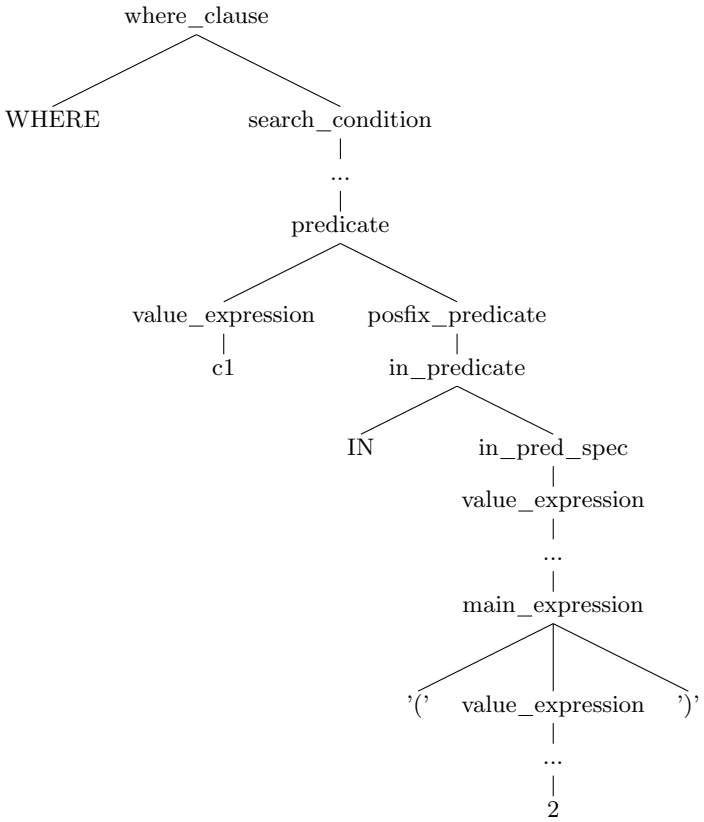
There is an inherent ambiguity in the *in_spec* non-terminal involved in the *in_predicate* rule. It can expand into a value expression, which could be parenthesized, or a list of value expressions. A singleton value expression list is not really distinguishable from a parenthesized value expression. Bison currently favors the latter by shifting the right parenthesis.

2.1.1 Bison DFA output

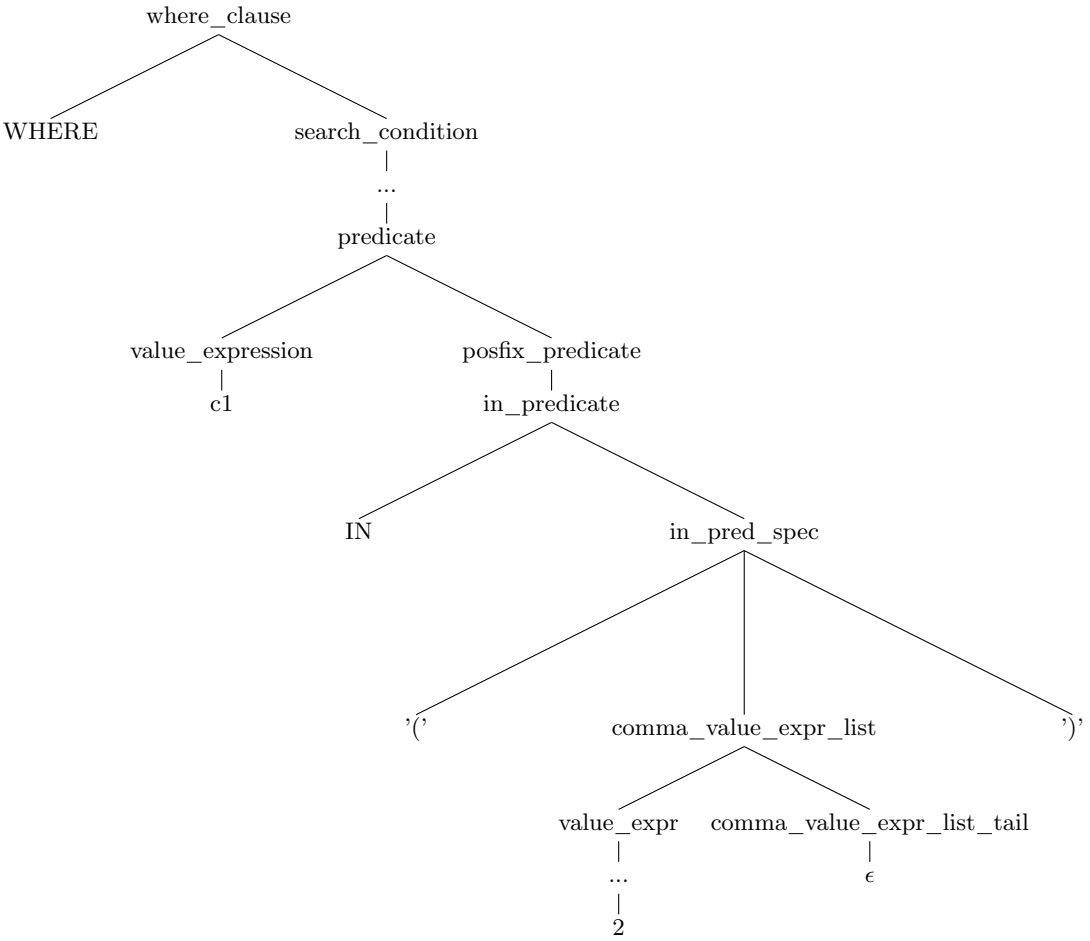
```
state 349

  158 main_expression: '(' value_expression . ')'
  214 comma_value_expression_list: value_expression . comma_value_expression_list_tail

    ')'  shift, and go to state 217
    ','  shift, and go to state 287

    ')'  [reduce using rule 216 (comma_value_expression_list_tail)]

    comma_value_expression_list_tail  go to state 288
```

2.1.2 Example and Trees

`SELECT * FROM table WHERE c1 in (2)`
First tree shows (2) as a parenthesized value expression.



Second possible tree shows (2) as a singleton list.



2.2 Parenthesized search condition vs stand-alone parenthesized value expression as predicate

Our search condition non-terminal requires a boolean expression. This can be achieved using some of the built-in posfix predicates (between, is null, like etc), or can be produced by a value expression without a following predicate. In this case, the value expression must create the boolean vector on its own, e.g. a function that returns boolean, or an identifier associated with a boolean-typed column. Given that we can parenthesize a search condition, it can be unclear whether a parenthesized non-terminal corresponds to a parenthesized value expression followed by an empty posfix predicate, or whether it is a parenthesized search condition that consists of a value expression and an empty posfix predicate. Bison currently favors the former by shifting the parenthesis.

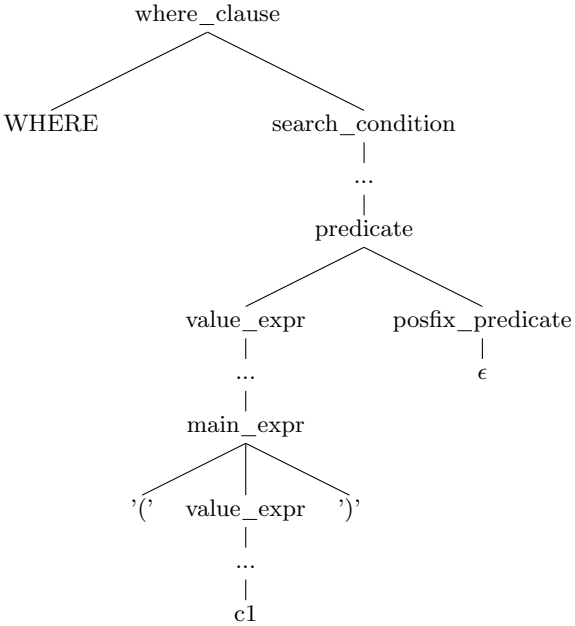
2.2.1 Bison DFA output

```
state 266

   51 predicate: value_expression . postfix_predicate
   74 range_value_expression: '(' value_expression . ',' value_expression ')'
  158 main_expression: '(' value_expression . ')'
```

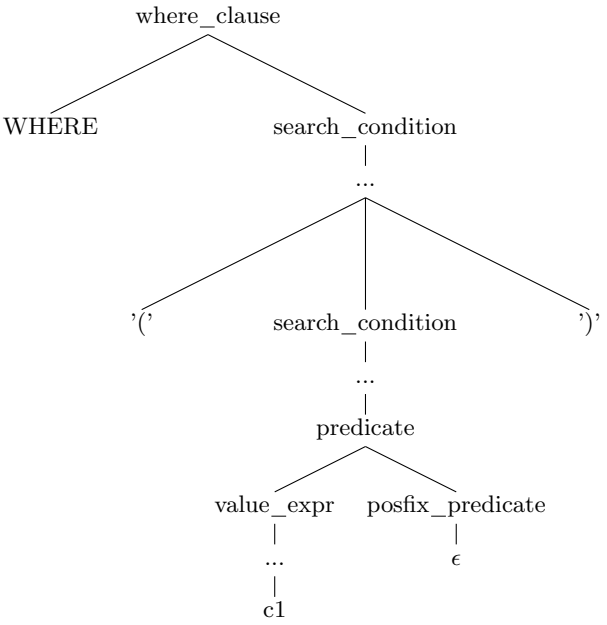
```
IS       shift, and go to state 270
NOT      shift, and go to state 271
BETWEEN  shift, and go to state 272
IN       shift, and go to state 273
LIKE     shift, and go to state 274
')'      shift, and go to state 217
','      shift, and go to state 309

')'       [reduce using rule 58 (postfix_predicate)]
$default  reduce using rule 58 (postfix_predicate)

postfix_predicate  go to state 275
between_predicate  go to state 276
in_predicate       go to state 277
like_predicate     go to state 278
null_predicate     go to state 279
is_predicate       go to state 280
```

2.2.2 Example and Trees

`SELECT * FROM table WHERE (c1)`
First tree shows the parenthesis associated with parenthesized value expression, followed by an empty posfix predicate.



The second tree shows the parenthesis associated with a search condition, which in turn consists of a value expression and an empty posfix predicate.


where_clause
WHERE
search_condition
...
'('
search_condition
')'
...
predicate
value_expr
posfix_predicate
...
ϵ
c1