Experiment No.5

Title: - LR(1) Parser

Aim:- Design and implement LR(1) parser.(LR(1) automaton)

The SLR(1) approach restricts the availability of reduce actions: they are allowed only if the next input symbol is in the Follow set of the LHS of the reducing production. However, in a given parsing context (a given state), the full follow set of a nonterminal is not always available (see lecture 8 slides for an example). An LR(1) parser keeps track of which terminals are actual permitted followers of a given symbol in each given parsing state. It thus recognises fewer valid reduce operations, and thus some shift-reduce and reduce-reduce conflicts are avoided.

1. Change to the Augmented Rule

In LR(0) and SLR(1), we add a rule, called the augmented rule, for recognition of the Start symbol of the grammar:

S' :- S $

2. Closures with Lookahead sets

When determining the closures for a LR(0) grammar, some extra information is indicated with each item: the set of possible terminals which could follow the item's LHS. This set of items is called the lookahead set for the item. For instance, given the normal closure for S0 :

E':- . E
E :- . T
E :- . E + T
T :- . id
T :- . (E)

The first item, the augmented rule, must be followed by $ (meaning that the recognition of the start symbol should consume all the input, leaving $ as the next symbol in the input). Now, the next 2 items are expansions of the 'E' in the first item, and since that E is at the end of the rule, then the lookahead set for the two E items is the same as for E' i.e. { $ }. I.e.,

E':- . E { $ }
E :- . T { $ }
E :- . E + T { $ }

The final two productions are expansions of the T in the second production, and T is the final symbol of the item, and thus the T can only be followed by $. The lookahead set of the final items is thus also { $ }. The LR(1) closure for S0 is thus:

E':- . E { $ }
E :- . T { $ }
E :- . E + T { $ }
T :- . id { $ }
T :- . (E) { $ }

Note however that the E on the RHS of the 3rd rule is normally ignored, since we previously dealt with E at a higher level. However, for an LR(1) closure, we need to expand this E as well. The E can be followed by a '+', so we add two items to the closure:
E :- . T { + }
E :- . E + T { + }
We then need to consider expanding the nonterminals after the dots in these new rules. In the case of the second, the added items would be the same as these two shown here (rules for E with lookahead symbol '+'), so recursion can stop. In the case of the first item, we now need to consider expanding the T with a lookahead set of { + }:
T :- . id { + }
T :- . (E) { + }
So, our final LR(1) closure for S0 is as follows:
E':- . E $ { }
E :- . T { $ }
E :- . E + T { $ }
T :- . id { $ }
T :- . (E) { $ }
E :- . T { + }
E :- . E + T { + }
T :- . id { + }
T :- . (E) { + }
We can simplify by merging rules with the same item but different lookahead list:

E':- . E $ { }
E :- . T { $, + }
E :- . E + T { $, + }
T :- . id { $, +}
T :- . (E)
{ $, + }

1.3 Constructing the LR(1) DFA

In the SLR(1) case, we could unify two states if they shared the same closure. The same rule applies here, but now two states can only be merged if the items and their lookahead sets are identical. This produces a parse table with far more states.

1.4 Constructing the Parse Table

In an SLR parser, when determining which imput symbols allow a reduce, we use the FOLLOW set for the rule's LHS. In an LR(1) parser, we use instead the lookahead set. This set is a subset of the full FOLLOW set of the item's LHS. In the case of items with a dot after the final symbol, the LR(1) parser is thus more selective as to which next input symbols can cause a reduce action.

The process for constructing the parse table from the DFA is much the same as for SLR(1), except:
•

In deciding which columns to put reduce actions in, while previously the FOLLOW set for a rule was used, here we only place a reduce action under each of the terminals in the lookahead set for the production.

1.5 Grammar Limitations

A LR(1) grammar is one where the construction of an LR(1) parse table does not require two action (shift-reduce or reduce-reduce) in any one cell. Many conflicts in SLR(1) parse tables are avoided if the LR(1) parse approach is used, because the latter approach is more restrictive on where it allows reduce operations. An SLR(1) parse table may allow reduces where the next input token should not allow such.

Input Specifications:- Students are supposed to take input the augmented grammar productions in any of the languages C, C++, Java or Python.

Output Specifications:- LR(1) automaton should be produced as an output.

Exercise:-

Q1. Construct LR(1) parser table for the following grammar:

S -> Aa / dAb / dca / cb

A -> c

Answer :

Q2. Merging states with a common core may produce _____ conflicts but does not produce _____ conflicts in an LALR parser.

A. reduce- reduce ; shift-reduce B. shift-reduce ; reduce- reduce

C. shift-reduce ; shift-reduce D. None of these

Conclusion: