LR PARSER

Construction of Items from a Production

Let there be a production of the form **A->XYZ**

We can construct 4 items from the production which are of the form

1. A->**.** XYZ
2. A->X**.**YZ
3. A->XY**.**Z
4. A->XYZ**.**

Each of these items tell us how much of the input string has been recognized. For example,

A->X.YZ implies that prefix of the string being parsed has been reduced to the non terminal X and a portion of the rest of the string may be matched with the non terminals YZ.

A->XYZ. tells us that prefix of the string to be parsed has been matched with XYZ and probably it is the time to reduce.

A->. XYZ tells us that we have not yet started matching prefix of the remaining string to be parsed with the production of A.

Construction of LR(0) Automaton

In case of the bottom up parsers there is an extra information module which tells the parser whether to shift or reduce. In case of LR parser it is the parsing table. In order to construct the parsing table we have to first construct the LR automaton. Each **state** in this automaton is nothing but a set of items. In order to construct states , 2 functions are defined namely **CLOSURE() , GOTO()** .CLOSURE helps to find states of the automaton and GOTO will help in constructing transitions of the automaton.

In construction of the states of the LR(0) automaton from the grammar G , some steps are followed .

STEP 1: Add the production **S'->S** to the grammar G to form augmented grammar G' where S is the start symbol of G and S' will be the start symbol in the augmented grammar G'.

STEP 2: For any LR(0) automaton the initial state contains the item **S'->.S.** Now in order to construct the state we need to find the closure of this item. Closure() takes set of items as input and generates closure of the set of items as output. Now CLOSURE(**S'->.S**) will give us the start state i.e S_0 for any LR automaton.

CLOSURE(I_0)

1. Put all items present in I_0 in CLOSURE I_0

2. If there is a production of the form A->B.CD in I_0 and there is a production of the form B->E in G' then add the item B->.E to the CLOSURE I_0

Let there be a grammar G' of the form

E'->E

E->E+T|T

T->T*F|F

F->id

To find State S_0 add the item E'->.E to the state .Now for all productions in G' with E as the head like E->E+T|T we have to add the items E->**.**E+T and E->**.**T to the state S_0 .Again we get an item with T appearing to the right of the dot. So we have to add items corresponding to the production where T appears as the head. So we get the items T->.T*F and T->.F . Similarly we get the item F->.id . Therefore **S_0** is equivalent to the set of items

E'->.E   E->.E+T   E->.T   T->.T*F   T->.F   F->.id

Now CLOSURE gives us the start state . In order to find the next state we have to use the **GOTO** functions. GOTO functions takes a state and a terminal/ non-terminal symbol as input and gives another set of items i.e state as the output.

GOTO(I_i , X) -> I_j where I_i -> input set of items X -> terminal/non-terminal symbol I_j -> output set of items.

Now if there is an item of the form A->B.XD in Ii then add the item A->BX.D to I_j and then compute the CLOSURE of the item. GOTO function moves the dot one symbol forward.

Let the string be of the form $x_1 \ldots. x_m x_{m+1} \ldots x_s x_{s+1} \ldots$

$x_1 \ldots. x_m$ has already been matched with B. $x_{m+1} \ldots x_s$ can be now matched with X . So our next item after marching X must be of the form A->BX.D where we have already matched a portion of the input string with BX and can probably match a prefix of the remaining string with D.

Let I_0 be the set of items corresponding to the state S_0 for the above given example. Now we want to find **GOTO(I_0,E), GOTO(I_0,T), GOTO(I_0,F), GOTO(I_0,id)**

GOTO(I_0,E) :I_1

Item E'->.E   E->**.**E+T are available . So we add E'->E.   E->E.+T to I_1 and take the closure of items.

So we get **I_1: E'->E.   E->E.+T**

GOTO(I_0,T) :I_2

Item E->.T   T->**.**T*F are available . So we add E->T.   T->T.*F to I_2 and take the closure of items.

So we get **I_2: E->T.   T->T.*F**

GOTO(I_0,F) :I_3

Item T->.F is available . So we add T->F. and take the closure of item.

So we get **I_3: T->F.**

GOTO(I_0,id) :I_4

Item F->.id is available . So we add F->id. and take the closure of item.

So we get **I_4: F->id.**

Each of the set of items I_4, I_1 , I_2, I_3 correspond to a new state 4,1,2,3 respectively of the LR(0) automaton . For each of these states we again use the GOTO function for those terminals and non-terminals which appear to the right of the dot in at least one of the items present in that state. For each **GOTO(I_i , X) -> I_j** call we make if we get a new set of items **I_j** we add a new state to the automaton . Corresponding to I_j we add the state j to the automaton and add the transition from state i to state j in the automaton labeled with the grammar symbol X.