Date: 20th Aug, 2013

Report

Topic of discussion: Top Down Parser and Introduction to Predictive Parser

Top Down Parser:

To design a Top Down Parser, the Grammar should be Non ambiguous and Non-left recursive.

Consider the Grammar
E-> E+E| E*E | E | id

The string id+id*id can be derived in two ways using the above Grammar as follows:

E-> E+E
->E+E*E
->id + id*id
OR
E->E*E
->E+E*E
->id+id*id

Hence the above Grammar is ambiguous.
On disambiguating the above grammar, we have
E->E+T|T
T->E*F|F
F->(E)|id

However, the above grammar is left recursive as it contains productions of the form
A->Aa|B

Replace these productions by
A->BA'
A'->aA'|epsilon

Thus we get
E->TE'
E'->+TE' |epsilon
T->FT'
T'->*FT' |epsilon
F->(E) |id

The modified grammar is both non ambiguous as well as non left recursive.

In the designing of Top Down Parser, it is desirable that the grammar is left factored i.e.
Consider the grammar
A->aB1 | aB2

Its left factored version is
A->aA'
A'->B1|B2

The advantage is that after a has been processed, we may have sufficient information to choose between the productions A'->B1 or A'->B2

Predictive Parser:

It can be designed only for LL(1) grammars.
We use two functions First() and Follow() to generate the Predictive parser table.

1. First(X)

i) If X->Terminal a
A belongs to First(X)

ii) If X->Non Terminal
X->epsilon, then
epsilon belongs to First(X)

iii) If X->Y1 Y2 Y3... Yn
i.e. X produces non terminals, then if Y1->epsilon i.e. Y1 somehow derives epsilon, then
First(X)<-First(Y2)
Similarly so on

Using above results for the modified grammar in Top Down Parser,
First(F)= {c,id}
First(T)=First(F)={c,id}
First(E)=First(T)
First(E')={+,epsilon}

2. Follow(A)

Follow(A)={set of terminals} where A is a non terminal.
S->aAcB
S is start state and the production is some sentential form.
Follow(A)= {set of terminals that follow A in sentential forms}

(More about Follow(), LL(1) type grammars and other concepts of Predictive Parser were continued in the next class)