Abstract Syntax Tree using Semantic Analysis

KRISHNA

17CS01008

Grammar considered for calculator is

E -> E + T | E - T | T

T -> T * F | T / F | F

F -> (E) | INT

Consider Two Functions:

nodeInt(INT): Node consists only of leaf nodes, i.e. Integers.

nodeOp(Op, L, R): Node consists of the operation that is to performed(Op), expression on the Left side of the operator(L) and expression on the right side of the operator(R).

E -> E + T {E.value = nodeOp(+, E.value, T.value)}

E -> E – T { E.value = nodeOp(-, E.value, T.value)}

E -> T { E.value = T.value}

T -> T * F { T.value = nodeOp(*, T.value, F.value)}

T -> T / F { T.value = nodeOp(/, T.value, F.value)}

T -> F { T.value = F.value}

F -> (E) {F.value = E.value}

F -> INT {F.value = nodeInt(INT)}

This is a Syntax Attributed Grammar (SAG). We compute the attributes in a bottom up fashion. So we use this along with LR Parsing.

As the leaf nodes must be integers and we evaluate them in a Bottom up parsing. If we encounter an Integer we push them into a separate stack which consists only of values. The values from this stack are used for computation.