Three-Address Code using Semantic Analysis

KRISHNA

17CS01008

Grammar considered for Evaluating expressions is

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

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

F -> (E) | INT

Consider a dictionary D = {} (Map), val = 0, TA = stack()

E -> E + T { D[T+val] = E.value + T.value; val ++;}

E -> E – T { D[T+val] = E.value - T.value; val ++;}

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

T -> T * F { D[T+val] = E.value * T.value; val ++;}

T -> T / F { D[T+val] = E.value / T.value; val ++;}

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

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

F -> INT {TA.push(INT); F.value = INT;}

We use SAG for writing the 3 address code for the given Grammar. We combine this with LR parsing to compute the 3-address code of a given Expression.

T+val is used as a key for the expression on the R.H.S which contains only 3 variables.

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 of the result simultaneously while parsing.