TRANSLATION SCHEME FOR FLOW OF CONTROL STATEMENTS

Translation of Boolean expression into the sequence of three address code is considered in the context of flow of control statements such as if-then, if-then-else, while-do statements.

Consider the following grammar

S -> if E then S1
| if E then S1 else S2
| while E do S1
E is the Boolean expression

Terms used in the translation scheme are

newlabel – returns a new symbolic label each time when it is called
true and false are the **two inherited attributes** associated with the non terminal E which help us to control the flow of the program.

Value of S.next is a label that is attached to first three address instruction to be executed after the code for S

SNo	Productions	Semantic action
1	E->E1orE2	E1.true =E.true E1.false=newlabel E2.true =E.true E2.false=E.false E.code=E1.code || gen(E1.false:) || E2.code
2	E->E1andE2	E1.true = newlabel E1.false=E.false E2.true =E.true E2.false=E.false E.code=E1.code || gen(E1.true:) || E2.code
3	E->not E1	E1.true=E.false E1.false=E.true E.code=E1.code
4	E->(E1)	E1.true=E.true E1.false=E.false E.code=E1.code
5	E->id1 relop id2	E.code= gen(if id1.place relop id2.place goto E.true) || gen(goto E.false)
6	E->true	E.code=gen(goto E.true)
7	E->false	E.code=gen(goto E.false)

Fig. 1 Syntax directed definition to produce three address code for Booleans

SNo	Productions	Semantic action
1	S->if E then S1	E.true =newlabel E.false=S.next S1.next=S.next S.code=E.code || gen(E.true:) || S.code
2	S->if E then S1 else S2	E.true =newlabel E.false=newlabel S1.next=S.next S2.next=S.next S.code=E.code || gen(E.true:) || S1.code || gen(goto S.next) || gen(E.false:) || S2.code
3	S->while E do S	S.begin=newlabel E.true =newlabel E.false=S.next S1.next=S.begin S.code=gen(S.begin:) || E.code || gen(E.true:) || S1.code || gen(goto S.begin)

Fig. 2 Syntax directed translation of flow of control statements

Example

Consider the statement

```
while a<b do
  if c<d then
    x=y+z
  else
    x=y-z
```

This statement can be viewed like the following

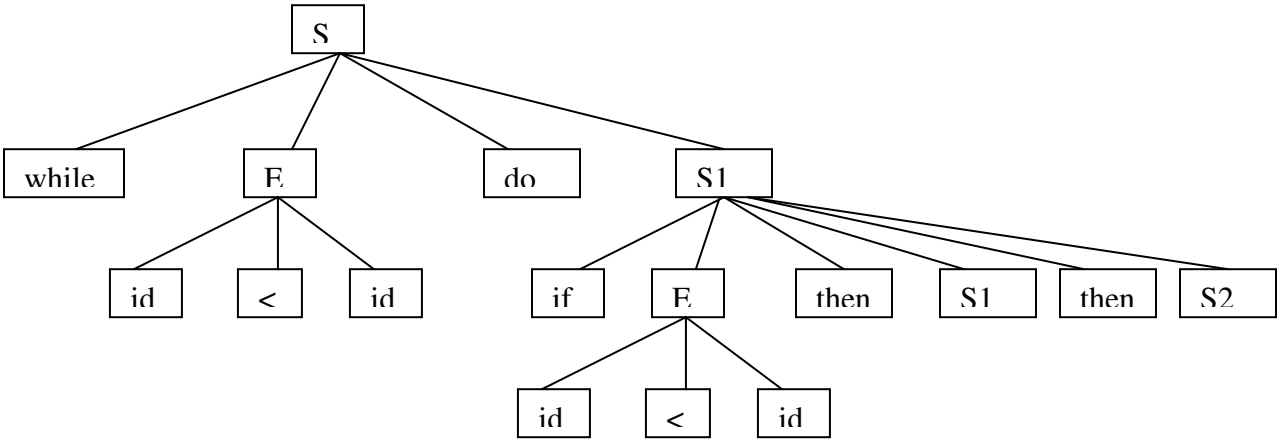

Fig. 3 Parse tree

Explanation:

Since there are inherited attributes associated with the nonterminals we need to apply the semantic rules from the parent level itself. For example the above example statement **begins with the start symbol of the grammar S.** S derives while statement from the above parse tree. So the semantic rule for while statement will be applicable to start with the generation of three address code.

Semantic rule for while – do statement is

S.begin=newlabel - L1
E.true =newlabel - L2
E.false=S.next

S1.next=S.begin - L1
S.code=gen(S.begin:) || E.code || gen(E.true:) || S1.code || gen(goto S.begin)

S.code
L1: if a<b then L2 - this code we got by referring fig. 1
goto S.next
L2: S1.code – for this take the semantic rule for if-then-else since S1 derives if-then-else
goto L1

Considering the semantic rule for if-then-else we get the TAC like the following

E.true =newlabel - L3
E.false=newlabel - L4
S1.next=S.next - L1
S2.next=S.next - L1
S.code=E.code || gen(E.true:) || S1.code || gen(goto S.next) || gen(E.false:) || S2.code

S1.code
if a<b then L3 - this code we got by referring fig. 1
goto L4
L3: t1=y+z
x=t1
goto L1
L4: t2=y-z
x=t2

Finally we get the TAC for the above example statement will be like the following

L1: if a<b then L2
goto S.next
L2: S1.code
if a<b then L3
goto L4
L3: t1=y+z
x=t1

```
        goto L1
L4: t2=y-z
        x=t2
        goto L1
S.next:
```