Boolean Expressions

Boolean Expressions

- Used in
  - conditional stmts that alter flow of control i,e. **while** and **if**
  - used to compute logical values
- Composed of Boolean operators (**and, or, not**)
- Can be applied to Boolean variables or relational expressions
- Relational expressions are of the form
  - E1 relop E2, where E1 and E2 are arithmetic expressions.
- Typical use of Boolean expressions
  - Conditional expressions that alter the flow of control
    - While , if-then
  - Compute logical values

Methods of Translating Boolean Expressions

- There are two principal methods for translating Boolean expression
  1. **Numeric Representation**-Encode true and false numerically and to evaluate a Boolean expression analogously to an arithmetic expression
     - 1 for true and 0 for false .
     - Non-zero for true, zero for false.
     - Nonnegative for true and negative for false.
  2. **Flow of Control**- representing the value of Boolean expression by a position reached in a program.
     - Represent the value by a position in the three-address code sequence.
     - Very convenient for if-then-else, while-do

Methods of Translating Boolean Expressions

- In addition to the three-address statements used previously branching statements will also be used here.

  goto L

  If A goto L

  If A relop B goto L
- Where A and B are simple variables or constants
- L is a quadruple label,
- relop is any of $<, \leq, \geq, >, =, \neq$

Numerical Representation

- The translation for the Example **A or B and C** is the three address sequence

  T_1:=B and C

  T_2:=A or T_1

- **A < B** is equivalent to **if A <B then 1 else 0**

  (1) if A<B goto (4)

  (2) T_1:=0

  (3) goto (5)

  (4) T_1:=1

  (5)

- **A<B or C**
  - **(5) T2:=T_1 or C**

Numerical Representation

Production	Semantic Rule
$E \rightarrow E(1)$ or $E(2)$	{T:=NEWTEMP(); E.PLACE=T; GEN(E.PLACE:=$E^{(1)}$.PLACE or $E^{(2)}$.PLACE);}
$E \rightarrow id^{(1)}$ relop $id^{(2)}$	{T1 = NEWTEMP(); E.PLACE=T1; GEN(if $id^{(1)}$.PLACE relop $id^{(2)}$.PLACE goto NEXTQUAD + 3); GEN(T_1: = 0); GEN(goto NEXTQUAD+2); GEN(T_1: = 1)}

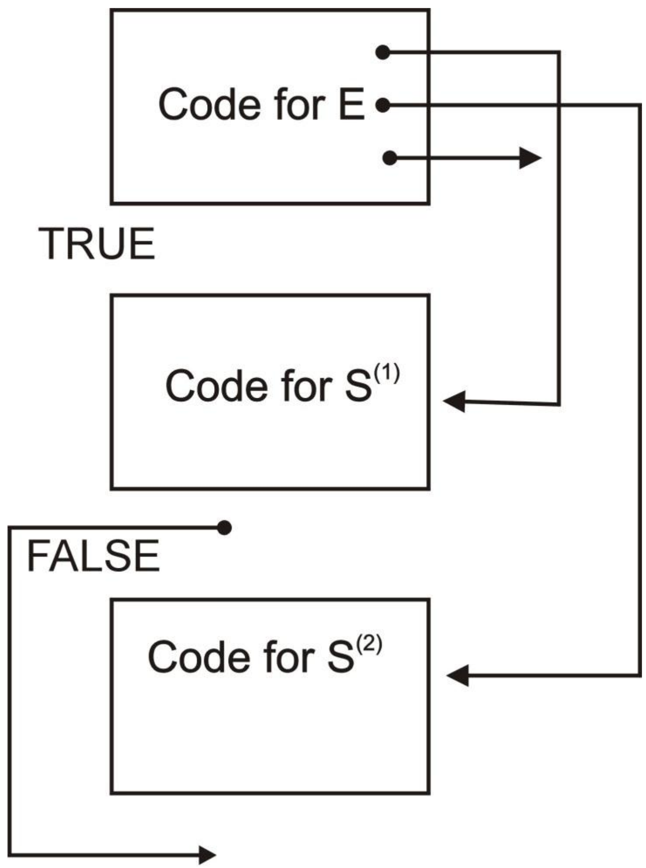
Control Flow Representation

- Represent the value of a Boolean expression by a position in the three-address code sequence.
- For example, if we point to the statement labeled $L1$, then the value of the expression is true (1); whereas if we point to the statement labeled $L2$, then the value of the expression is false (0).
- Use of temporary variable becomes unnecessary.
- If we evaluate expressions by program position, we may be able to avoid evaluating the whole expression
  - Example **A or B** //if **A is true we need not evaluate B**
  - however if there are side effects then an unexpected answer may be obtained

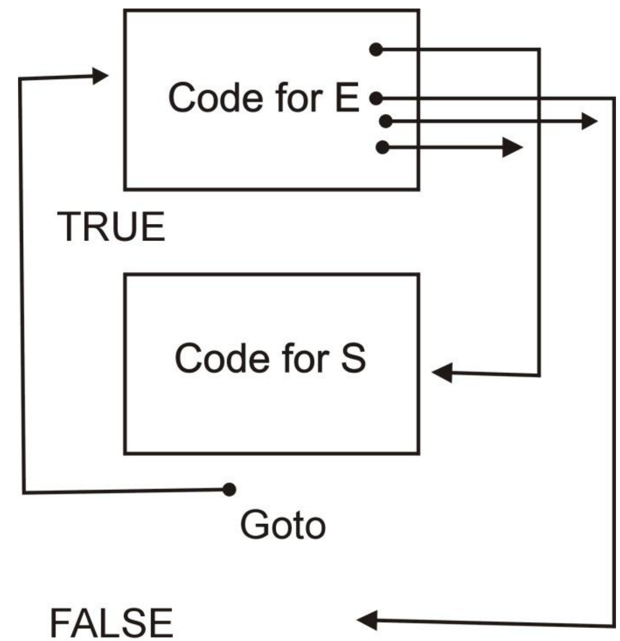
Control Flow Representation cont…

- the translation of Boolean expressions in the context of conditional statements such as
  - **if E then S(1) else S(2)**
  - **while E do S**
- In these contexts we can associate two kinds of exits with the Boolean expression *E*,
  - a **true** exit to statement TRUE
  - and a **false** exit to statement FALSE

Form of code for constructs using Boolean expression



If-statement



while-statement

Control Flow Representation cont…

- An expression E will be translated into a sequence of three-address statements that "evaluate E"
  - seq1
    seq2
    ….
    seqn
  - this *translation* is a sequence of **conditional** and **unconditional** jumps to one of two locations (i.e location TRUE and location FALSE)

Control Flow Representation cont…

- Consider a FORTRAN statement

IF (A.LT. B.OR. C .LT. D) X=Y+Z

- The three address code sequence:

1) If A < B goto (4)
2) If C < D goto (4)
3) Goto (6)
4) T:=Y+Z
5) X:=T
6)

Here (4) is the true exit of the boolean expression and (6) is the false exit.

Control Flow Representation cont…

- Consider an expression of the form $E^{(1)}$ **or** $E^{(2)}$
- $E^{(1)}$ has TRUE and FALSE location and the same applies to $E^{(2)}$
- If $E^{(1)}$ is true then E is true then location TRUE for $E^{(1)}$ is the same as TRUE for E
- If $E^{(1)}$ is false then we must evaluate $E^{(2)}$,
  - so we make FALSE for $E^{(1)}$ to be the first statement in the code of $E^{(2)}$
  - the true and false exit of $E^{(2)}$ can be made the same as for E

SDT scheme for Boolean expressions :Generates quadruples

- Problem
  - We may not have generated the actual quadruples to which the jumps are to be made at the time the jumps statements are generated
- the code we generate, is a series of branching statements with the targets of the jumps temporarily left unspecified
- Each such quadruple will be on one or another list of quadruples to be filled in when the proper location is found (subsequent filling in of quadruples is called **backpatching**)

SDT scheme for Boolean expressions :Generates quadruples

- **MAKELIST(*i*):** creates a new list containing only I, an index into the array of quadruples being generated
  - Returns a pointer to the list it has made
- **MERGE(**p1,p2**):** takes the lists pointed to by p1 and p2, concatenates them into one list and returns a pointer to the concatenated list.
- **BACKPATCH(***p,i***):** makes each of the quadruples on the list pointed tp by p take quadruple I as a target.

SDT scheme for Boolean expressions :Generates quadruples

- Consider the production $E \rightarrow E^{(1)}$ **and** $E^{(2)}$
  - If $E^{(1)}$ is false then E is false
  - so the quadruples on list $E^{(1)}$.FALSE can eventually be filled in with the location which follows the larger expression *E* in the case that *E* is false
  - i.e we have to make $E^{(1)}$.FALSE part of the list E.FALSE

SDT scheme for Boolean expressions :Generates quadruples

- If $E^{(1)}$ is true then we have to evaluate $E^{(2)}$
- so the **target** for the quadruples on list $E^{(1)}$.TRUE must be the beginning of the code generated for $E^{(2)}$
- but we cannot do this backpatching when we reduce $E^{(1)}$ **and** $E^{(2)}$ to E , it will be too late
- **Solution**
  - Is to ***factor*** the production $E \rightarrow E^{(1)}$ **and** $E^{(2)}$
  - backpatching of $E^{(1)}$.TRUE is done immediately after the code of $E^{(1)}$ is generated
  - NEXTQUAD holds the number of the first quadruple to follow and it therefore gives the proper value to substitute into quadruples in $E^{(1)}$.TRUE

SDT scheme for Boolean expressions :Generates quadruples

- *Solution1 : replace* $E \to E$ *and* E *by the two productions*
  - $E \to \text{EAND}\ \ E$
  - $\text{EAND} \to E$ **and** {BACKPATCH(E(1).TRUE,NEXTQUAD)
- *Solution2: create a marker nonterminal M and one translation M.QUAD with semantic routine*
  - $M \to \in$ { *M*.QUAD:=NEXTQUAD}

- *The new grammar is*

$$E \rightarrow E^{(1)} \text{ or } M\ E^{(2)}$$

$$\mid E^{(1)} \text{ and } M\ E^{(2)}$$

$$\mid \text{not } E^{(1)}$$

$$\mid (E^{(1)})$$

$$\mid \text{id}$$

$$\mid \text{id}^{(1)} \text{ relop id}^{(2)}$$

$$M \rightarrow \epsilon$$

SDT scheme

- $E \rightarrow E^{(1)}$ or M $E^{(2)}$ {

  BACKPATCH($E^{(1)}$.FALSE, M.QUAD);

  E.TRUE=MERGE($E^{(1)}$.TRUE, $E^{(2)}$.TRUE);

  E.FALSE= $E^{(2)}$.FALSE;}

- $E \rightarrow E^{(1)}$ and M $E^{(2)}$ {

  BACKPATCH($E^{(1)}$.TRUE, M.QUAD);

  E.TRUE= $E^{(2)}$.TRUE;

  E.FALSE=MERGE($E^{(1)}$.FALSE, $E^{(2)}$.FALSE);

  }

- $E \rightarrow \text{not } E^{(1)}$ {
  E.TRUE=$E^{(1)}$.FALSE);
  E.FALSE= $E^{(1)}$.TRUE;
  }
- $E \rightarrow (E^{(1)})$ {
  E.TRUE= $E^{(1)}$.TRUE;
  E.FALSE=$E^{(1)}$.FALSE;
  }

- E→**id** {
  E.TRUE=MAKELIST(NEXTQUAD);
  E.FALSE= MAKELIST(NEXTQUAD+1);
  GEN(if id.PLACE goto__);
  GEN(goto __);
  }
- E→**id**$^{(1)}$ relop **id** $^{(2)}$ {
  E.TRUE=MAKELIST(NEXTQUAD);
  E.FALSE= MAKELIST(NEXTQUAD+1);
  GEN(if id$^{(1)}$.PLACE relop id$^{(2)}$.PLACEgoto__);
  GEN(goto __);
  }
- M→ε {
  M.QUAD:=NEXTQUAD;
  }

- Now consider the expression

$P<Q$ or $R<S$ and $T<U$

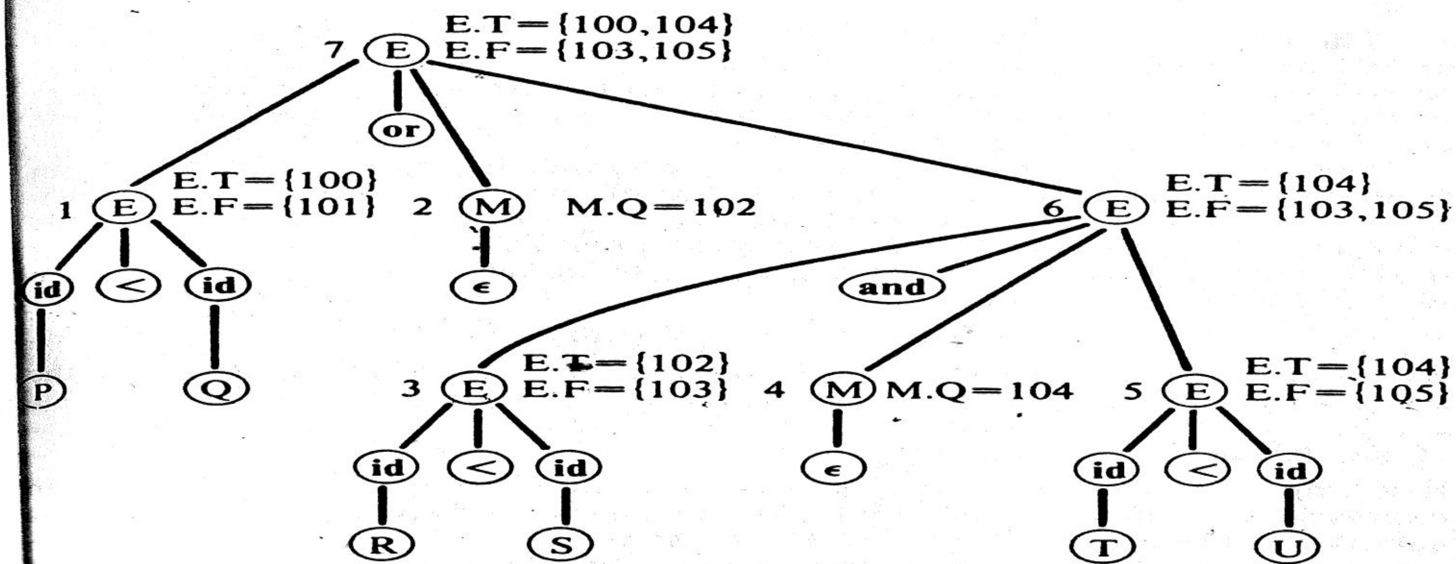


Fig. 7.25. Parse tree for P<Q or R<S and T<U

- In response to the reduction corresponding to node 1, the two quadruples are generated

  100: if P < Q goto _

  101:goto _
- Corresponding to node 3

  102: if R<S goto _

  103:goto _
- Corresponding to node 5

  102: ifT<U goto _

  103:goto _

- Node 6 corresponds to a reduction by

$E \rightarrow E^{(1)}$ and M $E^{(2)}$

  - semantic action BACKPATCH({102},104) fills in 104 in quadruple 102

100: if P < Q goto _

101:goto _

102: if R<S goto 104

103:goto _

104: if T<U goto _

105:goto

- Node 7 corresponds to a reduction by

$E \rightarrow E^{(1)}$ or $M\ E^{(2)}$

  - semantic action BACKPATCH({101},102) fills in 102 in quadruple 101

100: if P < Q goto _

101:goto 102

102: if R<S goto 104

103:goto _

104: if T<U goto _

105:goto

- The entire expression is true if and only if the **goto**'s of quadruples 100 or 104 are reached.
- And is false if and only if the **goto**'s of quadruples 103 or 105 are reached.