Control flow statements and Boolean expression

Boolean Expression

- Computes logical values for flow of control statements
- Boolean operator
  - and, or, not
- Relational operators
  - < , >, <=, >=, ==, !=

Syntax Directed Translation

Production	Semantic Rule
E → E1 or E2	E.place := newtemp emit(E.place ‘:=‘ E1.place ‘or’ E2.place)
E → E1 and E2	E.place := newtemp emit(E.place ‘:=‘ E1.place ‘and’ E2.place)
E → not E1	E.place := newtemp emit(E.place ‘:=‘ ‘not’ E1.place)
E → (E1)	E.place = E1.place

Production	Semantic Rule
E → id1 relop id2	E.place := newtemp emit (if id1.place relop.op id2.place, goto nextstat + 3) emit (E.place := 0) emit (goto nextstat+2) emit (E.place :=1)
E → true	E.place = newtemp emit(E.place ‘:=‘ 1)
E → false	E.place = newtemp emit(E.place ‘:=‘ 0)

Example

E_1 E_2 E_3

- a < b or c < d and e < f
- E => E1 or E2 => E1 or E2 and E3

Three address code

- 100 if a < b goto 103
- 101 t1 := 0
- 102 goto 104
- 103 t1 := 1
- 104 if c < d goto 107
- 105 t2: = 0

Three addresses code

- 106 goto 108
- 107 t2:= 1
- 108 if e < f goto 111
- 109 t3 := 0
- 110 goto 112
- 111 t3 := 1
- 112 t4: = t2 and t3
- 113 t5 := t1 or t4

Short Circuit code

- Boolean expression – 3 address code without generating code for boolean operators
- Need not evaluate the full expression

E_1 or E_2

①

E_1 and E_2

0

Flow of Control Statements

- Boolean expressions are typically used along with if-then, if-then-else, while-do statement constructs
- S → if E then S1 | if E then S1 else S2 |

  while E do S1 | do S1 while E

Flow of control statements

- All these statements “E” corresponds to a Boolean expression evaluation
- This expression E should be converted to three address code as already discussed
- This is then integrated in the context of control statements

if-then statement

S → if E then S1

Label	Control flow
	E.code
E.true:	S1.code
E.false:	

Syntax directed definition for if-then

Production	Semantic Rules
S → if E then S1	E.true:= newlabel E.false := S.next S1.next := S.next S.code := E.code || gen (E.true':') || S1.code

if-then-else statement

Label	Control flow
	E.code
E.true:	S1.code
	goto S.next
E.false:	S2.code
S.next	

SDD for if-then - else

Production	Semantic Rules
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

While-do statement

Label	Control flow
S.begin	E.code
E.true:	S1.code
	goto S.begin
E.false:	

SDD for while - do

Production	Semantic Rules
S → while E do S1	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)

Do-while statement

Label	Control flow
E.true: / S.begin	S1.code
	E.code
	goto S.begin
S.next / E.false:	

Production	Semantic Rules
S → do S1 while E	S.begin := newlabel E.true := S.begin E.false := S.next S.code := S1.code || E.code || gen (E.true ‘:’) || gen (‘goto’ S.begin)

For loop

- A ‘for’ loop construct may be considered as a ‘while’ construct to generate 3-address code

Control flow with boolean expression

- If the expression E of the control flow statement is boolean, code can be generated by incorporating short circuit information
- Short circuit avoids computing the full expression involving logical operators

Example

- E – a > b
- Then code is
  - if a > b goto E.true
    goto E.false
- This is different from creating a temporary variable and assigning it 0 / 1

Short circuit information

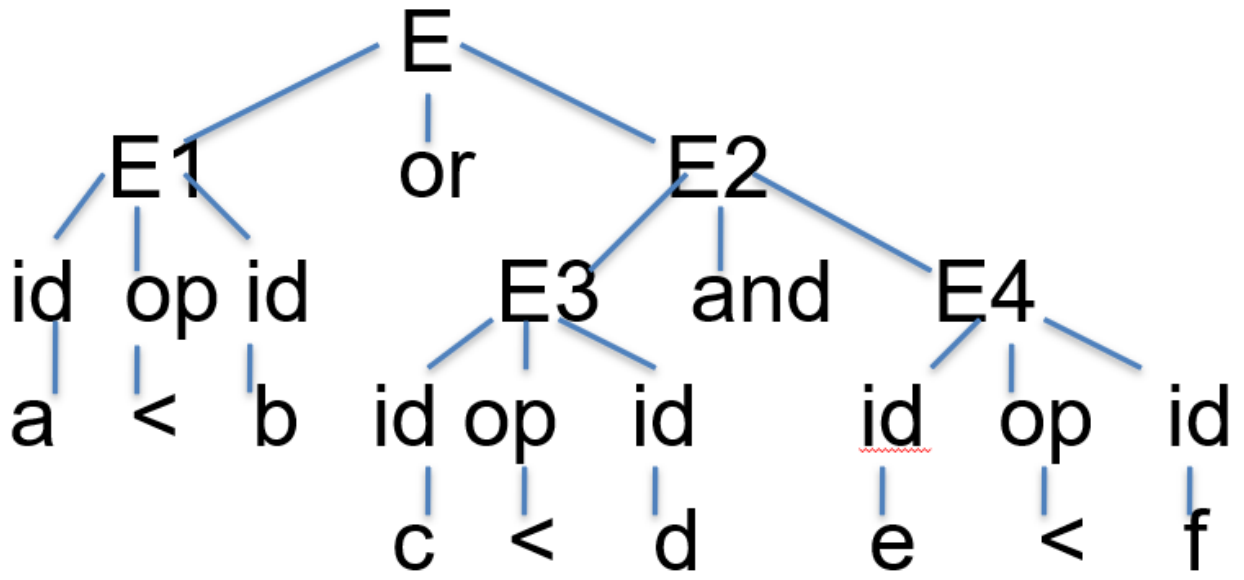
- If E is of the for E1 or E2, then if E1 is true, then this can directly be associated to E.true.
- If E1 is false then E2 need to be evaluated

SDD for 3-address code for boolean

Production	Semantic Rule	Inference
E → E1 or E2	E1.true : = E. true E1.false := newlabel E2.true := E.true E2.false := E.false E.code := E1.code || gen (E1.false ‘:’) ||E2.code	Instead of creating newlabel, we directly assign it to E.true
E → E1 and E2	E1.true : = newlabel E1.false := E.false E2.true := E.true E2.false := E.false E.code := E1.code || gen (E1.true ‘:’) || E2.code	

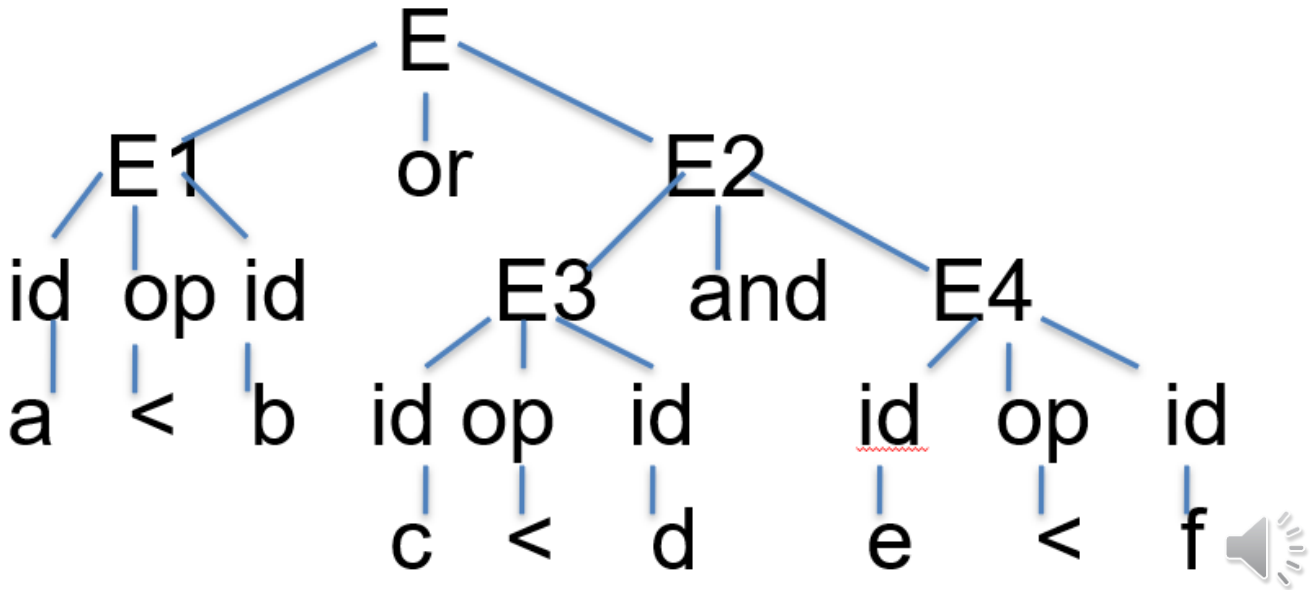
Production	Semantic Rule	Inference
E → not E1	E1.true : = E. false E1.false := E.true E.code := E1.code	
E → (E1)	E1.true : = E. true E1.false := E.false E.code := E1.code	
E → id1 relop id2	E.code := gen (‘if’ id1.place relop.op id2.place ‘goto’ E.true) || gen (‘goto’ E.false)	
E→ true	E.code := gen (‘goto’ E.true)	
E → false	E.code := gen (‘goto’ E.false)	

Example - a < b or c < d and e < f



Interpretation - a < b or c < d and e < f

- E1.true = E.true
- false label for E1
- true label for E3
- E3’s false is E2’s false and E’s false
- E4’s true is true for E
- E4’s false is E’s false



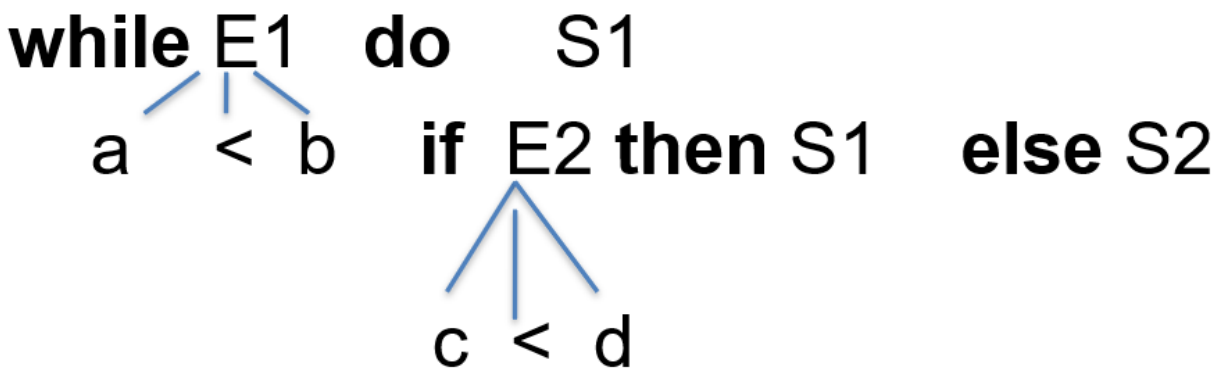
Code- a < b or c < d and e < f

```
      if a < b goto Ltrue
         goto L1
L1: if c < d goto L2
    goto Lfalse
L2: if e < f goto Ltrue
     goto Lfalse
```

Example while

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

Derivation




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

Explanation

- E1.code – if a < b goto true label
  goto false label
- True label is the body of the while
- False label is the S.next
- E2.code – if c < d goto true label
  goto false label

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

Explanation

- This should be done in the context of while and if-else
- If then else

  S.code := E.code || gen (E.true':') || S1.code || gen ('goto' S.next) || gen (E.false :') || S2.code
- While do

  E.false = S.next

  S1.next := S.begin

  S.code := gen (S.begin ':') || E.code ||
  gen (E.true':') || S1.code ||
  gen ('goto' S.begin)

Code

```
L1: if a < b goto L2
    goto Lnext
L2: if c < d goto L3
    goto L4
L3: t1 : = y+ z
    x := t1
    goto L1
L4: t2 := y – z
      x := t2
      goto L1
Lnext:
```

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

Summary

- Control flow statements
  - If-then, if-then-else, do-while
- Boolean expressions with control flow
- Incorporating short circuit code in generating 3-address code