Lecture 07

Intermediate Code Generation

III: TAC generation for control statements

Outline

- Intermediate code generation
  - Intermediate Code for Code Generation
  - Basic Code Generation Techniques
  - Code Generation of Control Statements and Logical Expressions

Code Generation of Boolean Expressions

- Logical Expressions (or Boolean Expressions) are composed of the Boolean operators (and,or,not) applied to elements that are Boolean variables or relational expressions
  - Relational expressions are of the form "E1 relop E2", where E1 and E2 are arithmetic expressions , relop is a comparison operator such as < , <=, ==, !=, > , >=
- Simple Boolean expressions generated by the following grammar

  E -> E or E | E and E | not E | (E)

  | id relop id | true | false

  or and and are left-associative, and or has lowest precedence, then and ,then not

Code Generation of Boolean Expressions

- Logical expressions have two primary purpose
  - If they are used to compute logical values, Boolean expressions are translated in a manner similar to arithmetic expressions

    For example : The translation for "a or b and not c" is the three-address sequence

    t1= not c

    t2=b and t1

    t3=a or t2
  - If they are used as test in the context of control statements, such as if-then or while-do, the value of Boolean expression is not saved in a temporary but represented by a position reached in a program

    Grammar for control statements

    S-> if E then S1 | if E then S1 else S2

    | while E do S1

    E is the Boolean expression

Translation of control statements

- Attribute grammar for translating control statements
  - Functions used in code generation
    - newlabel() returns a new label each time it is called
  - Attributes
    - E.true(E.false) is the label to which control flows if E is true(false)
    - S.next is a label that is attached to the first three-address instruction to be executed after the code for S
    - S.begin is a label that is attached to the first instruction of generated code for S

TAC generation for control statements

- Code pattern example for "if (E) S1 else S2"

<code to evaluate E to t1 >
if t1 goto E.true
goto E.false
Label E.true
<code for S1>
goto S.next
label E.false
<code for S2>
label S.next

if (not a<b) S1 else S2

if a<b goto E.false
goto E.true
Label E.true
<code for S1>
goto S.next
label E.false
<code for S2>
label S.next

Translation of Boolean expressions in the context of control statements

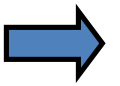
- TAC (E.code) for Boolean expressions E in the context of control statements is a sequence of conditional and unconditional jumps to one of two location: E.true and E.false
  - E is of the form id relop id

    E → id1 relop id2

    - Semantic Rules

      E.code = gen("if" id1.name relop id2.name "goto" E.true)
      || gen("goto" E.false)

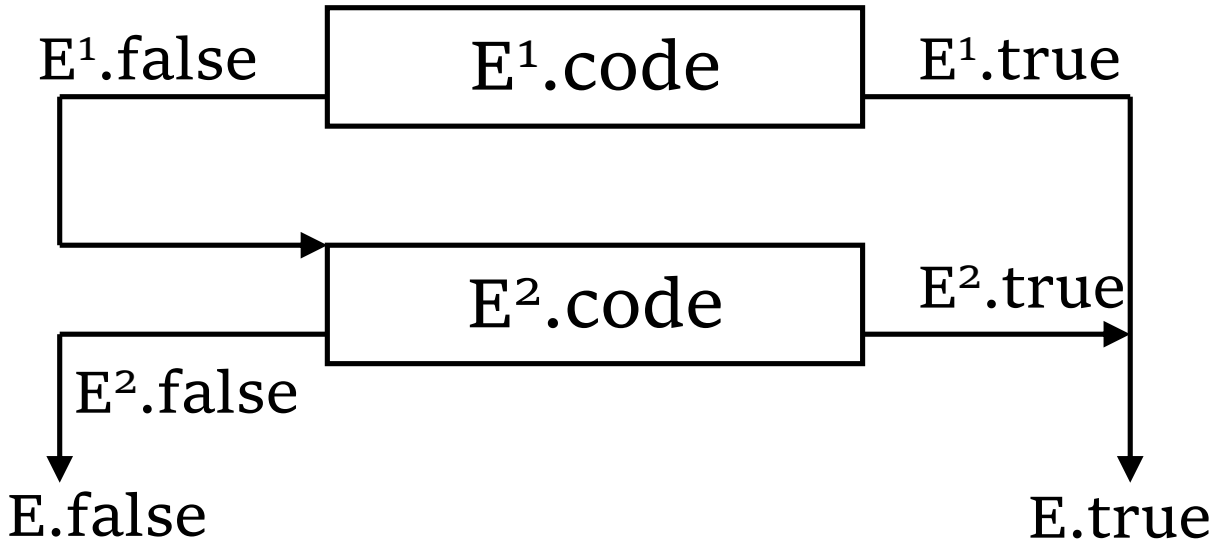
if a > b goto E.true
goto E.false



  - E is of the form E1 or/and/not E2

Translation of Boolean expressions in the context of control statements

– E is of the form E1 or E2 i.e. E -> E1 or E2



•if E1 is true, then E is true, so **E1.true** = E.true

• if E1 is false, then E2 must be evaluated, so **E1.false** is the first statement in the code for E2

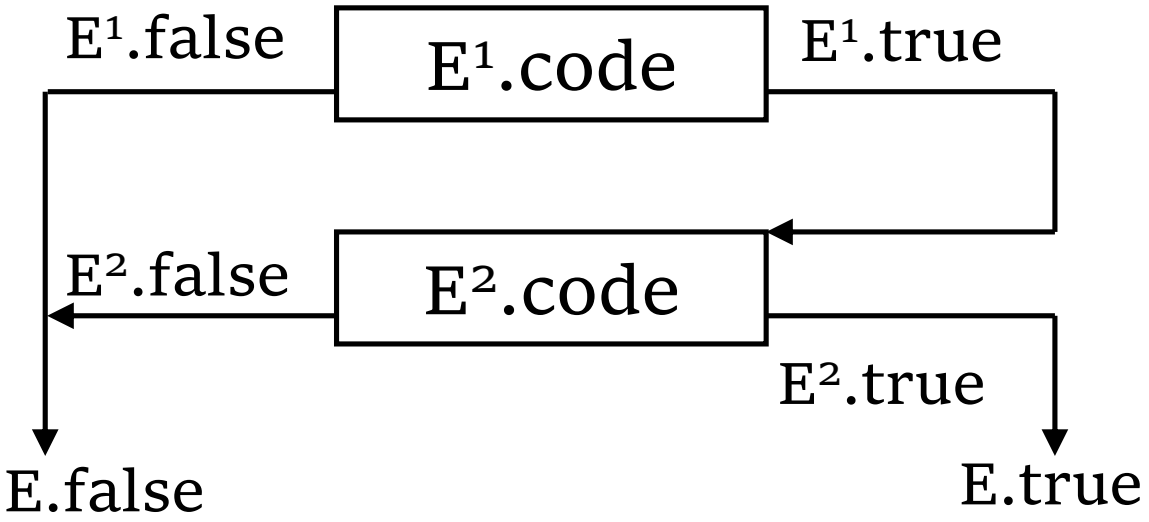
• The true and false exits of E2 is the same as E respectively

- **Semantic Rules** for "**E -> E1 or E2**":

```
E1.true = E.true;
E1.false = newlabel();
E2.true = E.true;
E2.false = E.false;
E.code = E1.code ||
    Label E1.false || E2.code
```

Translation of Boolean expressions in the context of control statements

– E is of the form E1 and E2 i.e. E -> E1 and E2



Semantic Rules:

E1.true = newlabel();

E1.false = E.false;

E2.true = E.true;

E2.false = E.false;

E.code = E1.code || Label E1.true || E2.code

Translation of Boolean expressions in the context of control statements

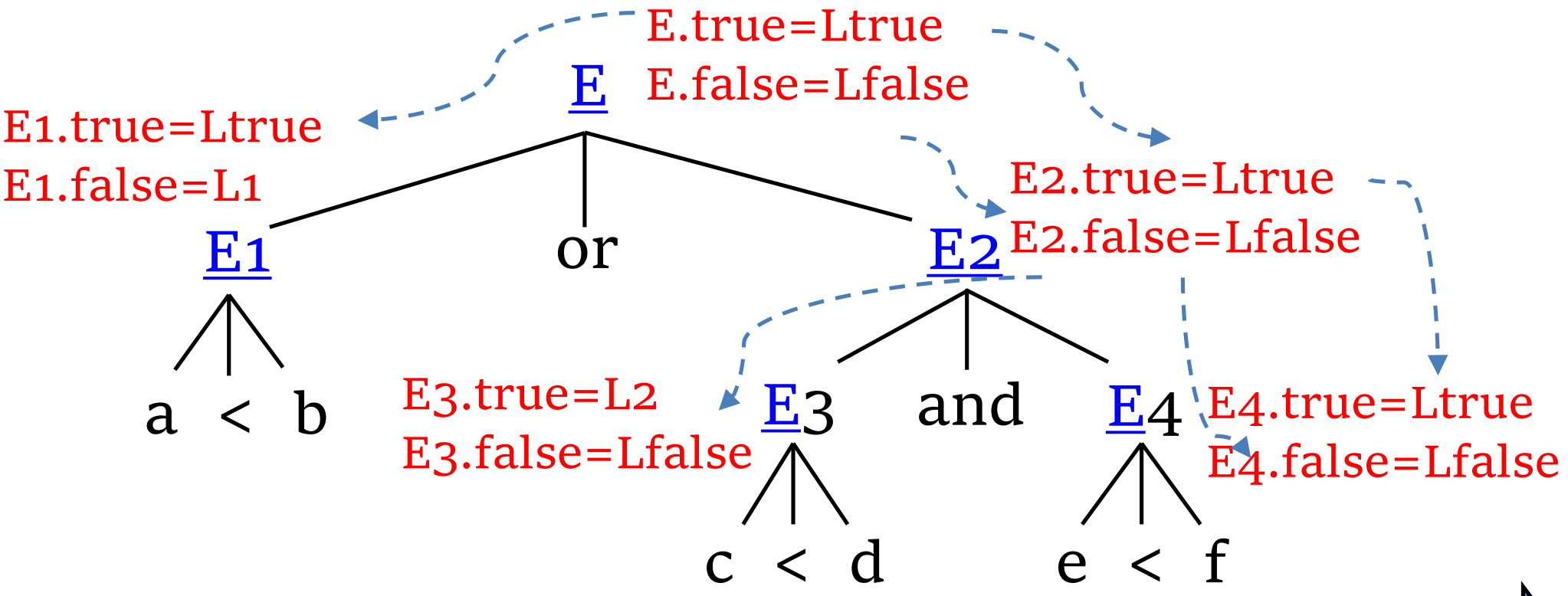
- E is of the form not E1 i.e. E -> not E1
- Just interchange the true and false exits of E to get the true and false exits of E1

  - Semantic Rules:

    E1.true = E.false;
    E1.false = E.true;
    E.code = E1.code;

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

Suppose the true and false exits for the entire expression have been set to Ltrue and Lfalse



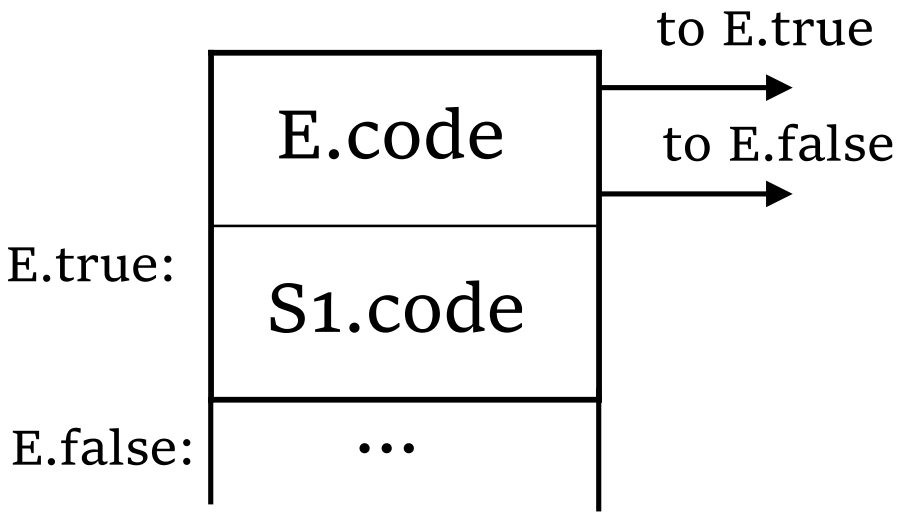
if a<b goto Ltrue
goto L1
Label L1

if c<d goto L2
goto Lfalse
label L2

if e<f goto Ltrue
goto Lfalse

Translation of control statements

- "S → if E then S1"



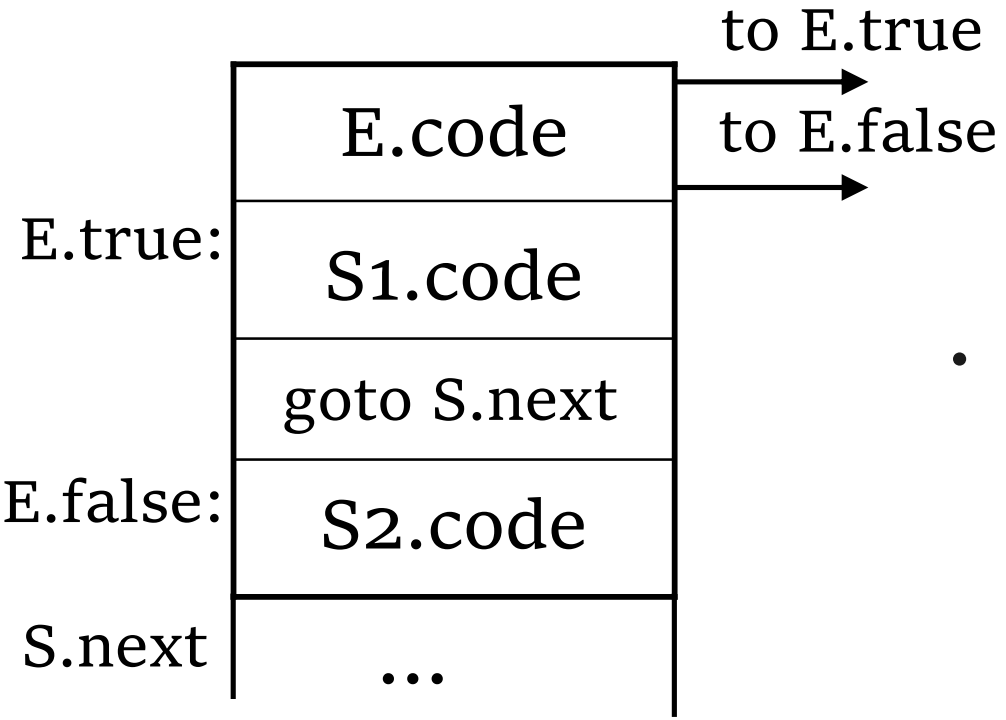
• E.true is attached to the first instruction generated of the code for S1

• E.false is attached to the first instruction to be executed after the code for S

E.true = newlabel();
E.false = S.next ;
S1.next = S.next ;
S.code = E.code || Label E.true || S1.code

TAC generation for control flow structure

- "S → if E then S1 else S2"



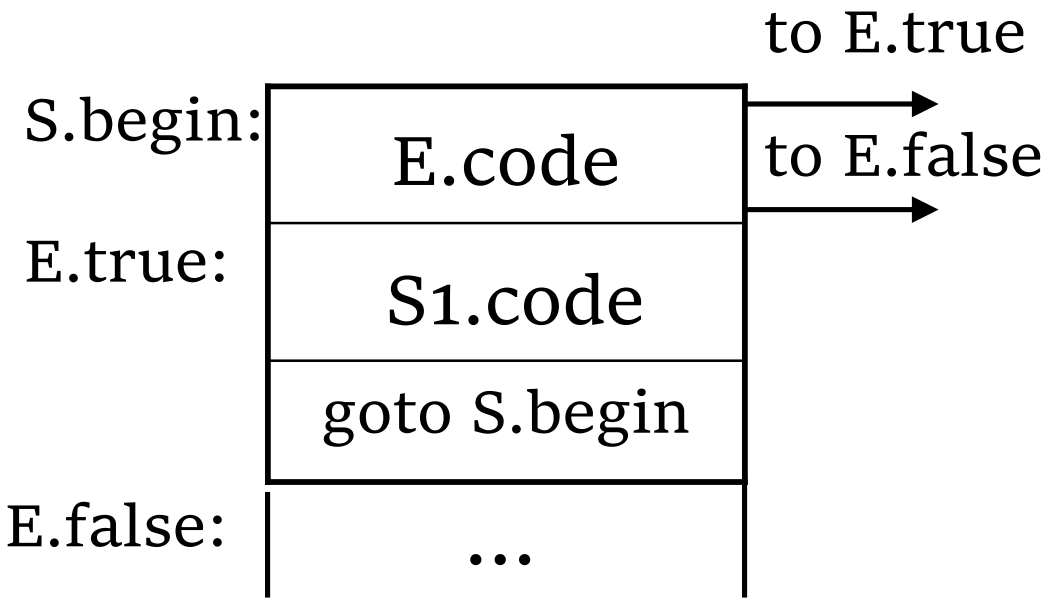
- E.true is attached to the first instruction of the code for S1
- E.false is attached to the first instruction of the code for S2
- Code pattern for "if (E) S1 else S2"

```
<code to evaluate E to t1 >
if t1 goto E.true
goto E.false
Label E.true
<code for S1>
goto S.next
label E.false
<code for S2>
label S.next
```

E.true = newlabel();
E.false = newlabel();
S1.next = S.next; S2.next = S.next;
S.code = E.code || Label E.true || S1.code ||
gen("goto" S.next)|| Label E.false || S2.code

TAC generation for control flow structure

- "S → while E do S1"

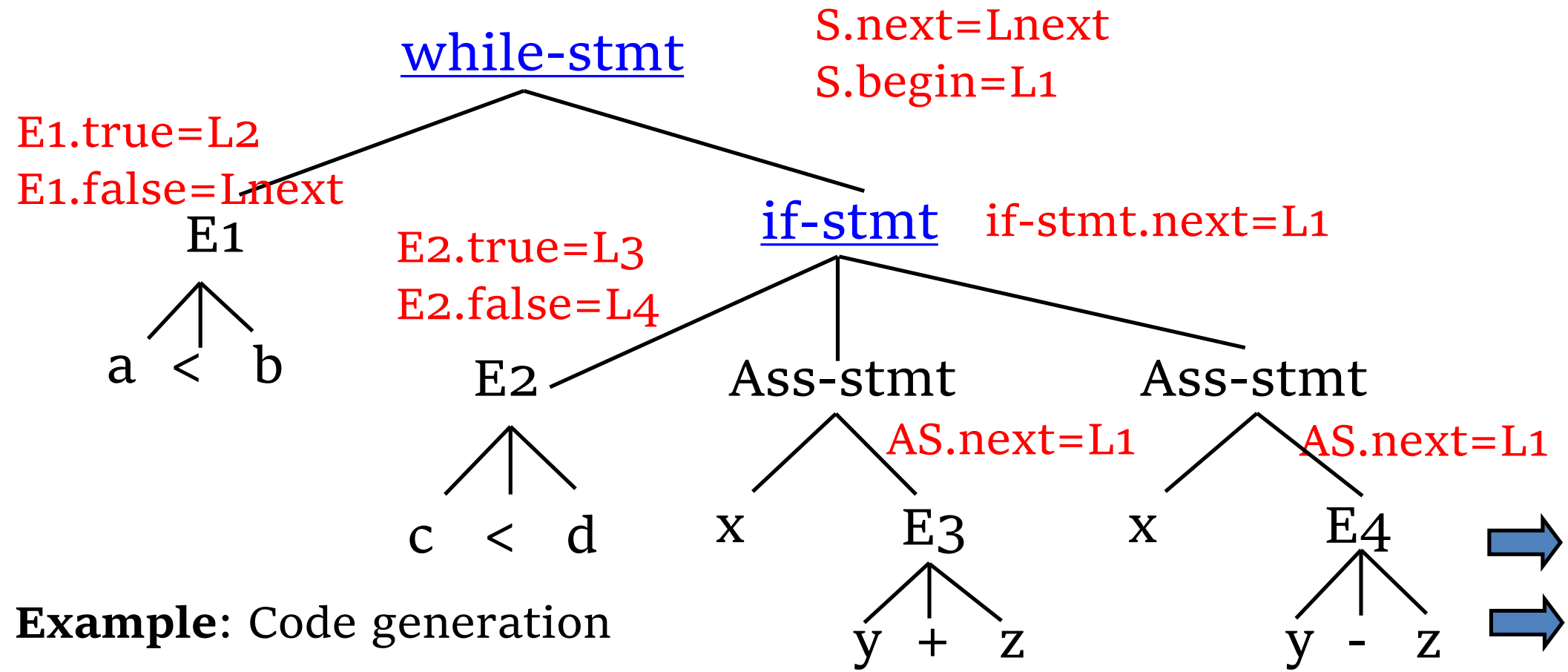


- E.true is attached to the first instruction of the code for S1
- E.false is attached to the first instruction to be executed after the code for S

- Code pattern for "while (E) S1"

label S.begin
<code to evaluate E to t1>
if t1 goto E.true
goto E.false
Label E.true
<code for S>
goto S.begin
label S.next

S.begin = newlabel();
E.true = newlabel();
E.false = S.next;
S1.next = S.begin;
S.code = Label S.begin || E.code ||
Label E.true|| S1.code|| gen("goto" S.begin)



Example: Code generation for the control statement

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

```
Label L1
if a<b goto L2
goto Lnext
Label L2
```

```
if c<d goto L3
goto L4
Label L3
t1=y+z
x=t1
```

```
goto L1
Label L4
t2=y-z
x=t2
goto L1
```