CS251: Introduction to Language Processing

Intermediate Code Generation

Vishwesh Jatala

Department of CSE

Indian Institute of Technology Bhilai

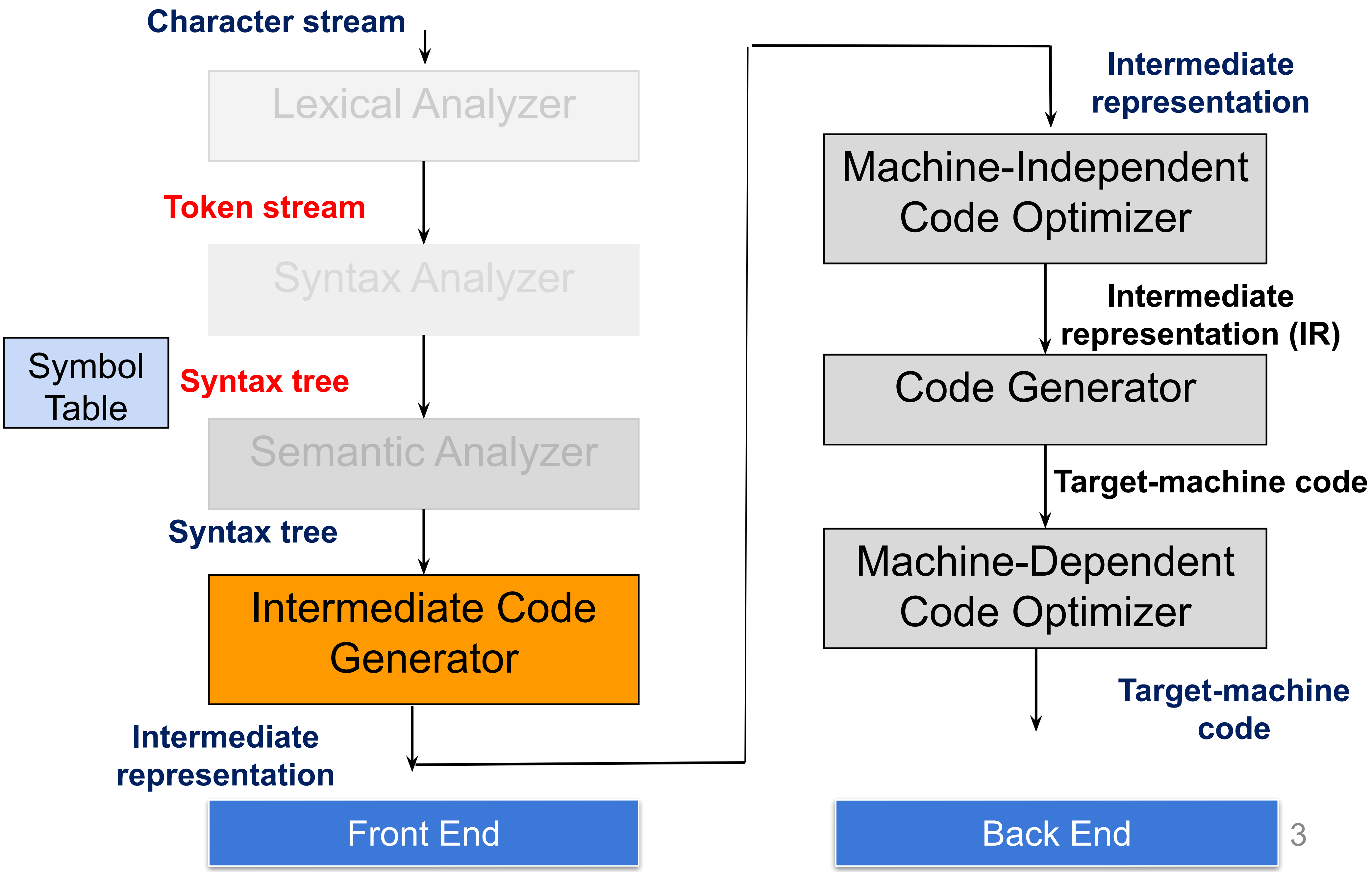
vishwesh@iitbhilai.ac.in

2023-24-M

Acknowledgement

- References for today's slides
  - *Lecture notes of Prof. Amey Karkare (IIT Kanpur) and Late Prof. Sanjeev K Aggarwal (IIT Kanpur)*
  - *IIT Madras (Prof. Rupesh Nasre)*
    - *http://www.cse.iitm.ac.in/~rupesh/teaching/compiler/aug15/schedule/4-sdt.pdf*
  - *Course textbook*
  - *Stanford University:*
    - *https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/*

Next…



Character stream
Lexical Analyzer
Token stream
Syntax Analyzer
Symbol Table
Syntax tree
Semantic Analyzer
Syntax tree
Intermediate Code Generator
Intermediate representation
Front End
Intermediate representation
Machine-Independent Code Optimizer
Intermediate representation (IR)
Code Generator
Target-machine code
Machine-Dependent Code Optimizer
Target-machine code
Back End

Recap

- Semantic Analysis
- Intermediate code generation
  - Expressions
    - Arithmetic
    - Boolean

Boolean Expressions

```
E →
   |    E relop E
   |    E or E
   |    E and E
   |    not E
   |    true
   |    false
```

Methods of translation

- Evaluate similar to arithmetic expressions
  - Normally use 1 for true and 0 for false

- Implement by flow of control using short circuiting
  - given expression E_1 or E_2
    if E_1 evaluates to true
    then E_1 or E_2 evaluates to true
    without evaluating E_2

Numerical representation

- relational expression a < b is equivalent to if a < b then 1 else 0

```
 1.if a < b goto 4.
 2.t = 0
3. goto 5
4. t = 1
5.
```

Syntax directed translation of boolean expressions

```
E → E1 < E2
          E.place := newtmp
          emit(if E1.place < E2.place goto nextstat+3)
          emit(E.place = 0)
          emit(goto nextstat+2)
          emit(E.place = 1)
```

"nextstat" is a global variable; a pointer to the statement to be emitted. emit also updates the nextstat as a side-effect.

Syntax directed translation of boolean expressions

$E \longrightarrow E_1$ or E_2

```
        E.place := newtmp
        gen(E.place ':=' E₁.place 'or' E₂.place)
```

$E \longrightarrow E_1$ and E_2

```
        E.place:= newtmp
        gen(E.place ':=' E₁.place 'and' E₂.place)
```

$E \longrightarrow$ not E_1

```
        E.place := newtmp
        gen(E.place ':=' 'not' E₁.place)
```

Syntax directed translation of boolean expressions

```
E → true
        E.place := newtmp
        emit(E.place = '1')

E → false
        E.place := newtmp
        emit(E.place = '0')
```

Exercise

Generate TAC for

a < b or c < d and e < f

Operator	**Meaning**	**Associativity**
<	Relational less than	left-to-right
and	Logical AND	left-to-right
or	Logical OR	left-to-right

Precedence and Associativity Symbol. Top row as highest precedence.

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

```
100: if a < b goto 103
101: $t_l$ = 0
102: goto 104
103: $t_l$ = 1
104:
        if c < d goto 107
105: $t_2$= 0
106: goto 108
107: $t_2$ = 1
108:
```

```
if e < f goto 111
109: $t_3$ = 0
110: goto 112
111: $t_3$ = 1
112:
        $t_4$ = $t_2$ and $t_3$
113: $t_5$ = $t_l$ or $t_4$
```

Short Circuit Evaluation of boolean expressions

- Translate boolean expressions without:
  - generating code for storing the boolean result explicitly
  - evaluating the entire expression

- Flow of control

  statements S $\longrightarrow$ if E then

  S_1

  | if E then S_1 else S_2

  while E do S_1

Short Circuiting

E1 E2

if (x < 100 || x > 200) x =0 ;

100: if x < 100 goto 108

101: t_l = 0

102: goto 104

103: t_l = 1

104: if x> 200 goto 108

105: t2 = 0

106: goto 109

107: t2=1

108: x=0

109:

100: if x< 100 goto 108

102: goto 104

104: if x> 200 goto 108

106: goto 109

108: x=0

109:

Boolean Expression

E: x < 100

```
100: if  x< 100  goto _
102: goto _
```

```
100: if  x< 100  goto E.true
102: goto E.false
```

Syntax directed translation of boolean expressions

if E is of the form: $a < b$

then code is of the form:

```
if a < b goto E.true
goto E.false
```

Syntax directed translation of boolean expressions

$E \rightarrow E_1 \text{ relop } E_2$

$E.code = gen(\text{ if } E_1 \text{ relop } E_2 \text{ goto } E.true) \mid\mid$
$gen(\text{goto } E.false)$

Each Boolean expression E has two attributes, **true** and **false**. These attributes hold the label of the **target stmt** to jump to.

Control flow translation of boolean expression

$E \longrightarrow E_1$ and E_2

E_1.true := newlabel
E_1 false := E.false
E_2.true := E.true
E_2 false := E.false
E.code := E_1.code || gen(E_1.true) || E_2.code

Control flow translation of boolean expression

$E \rightarrow E_1$ or E_2

E_1.true := E.true
E_1.false := newlabel
E_2.true := E.true
E_2.false := E.false
E.code := E_1.code || gen(E_1.false) || E_2.code

Control flow translation of boolean expression …

$E \rightarrow \text{not } E_1$

$E_1.\text{true} := E.\text{false}$

$E_1.\text{false} := E.\text{true}$

$E.\text{code} := E_1.\text{code}$

Control flow translation of boolean expression …

E $\longrightarrow$ true   E.code = gen(goto E.true)

E $\longrightarrow$ false   E.code = gen(goto E.false)

Example

Code for a < b or (c < d and e < f)

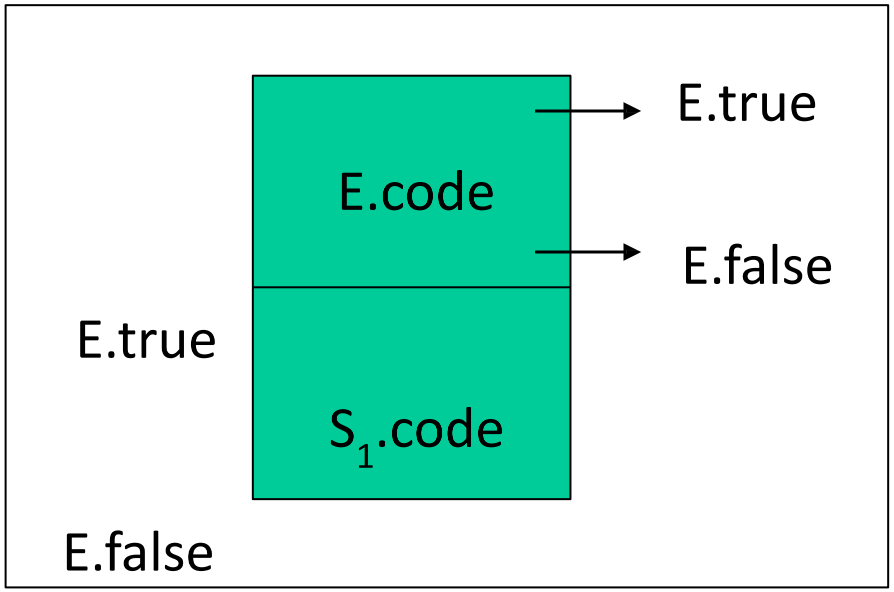
```
        if a < b goto Ltrue
        goto L1
L1:     if c < d goto L2
        goto Lfalse
L2:     if e < f goto Ltrue
        goto Lfalse

Ltrue:
Lfalse:
```

Control Flow Statements

- Flow of control statements $S \longrightarrow$ if E then S_1

  | if E then S_1 else S_2

  | while E do S_1



$S \longrightarrow$ if E then S_1

E.true = newlabel

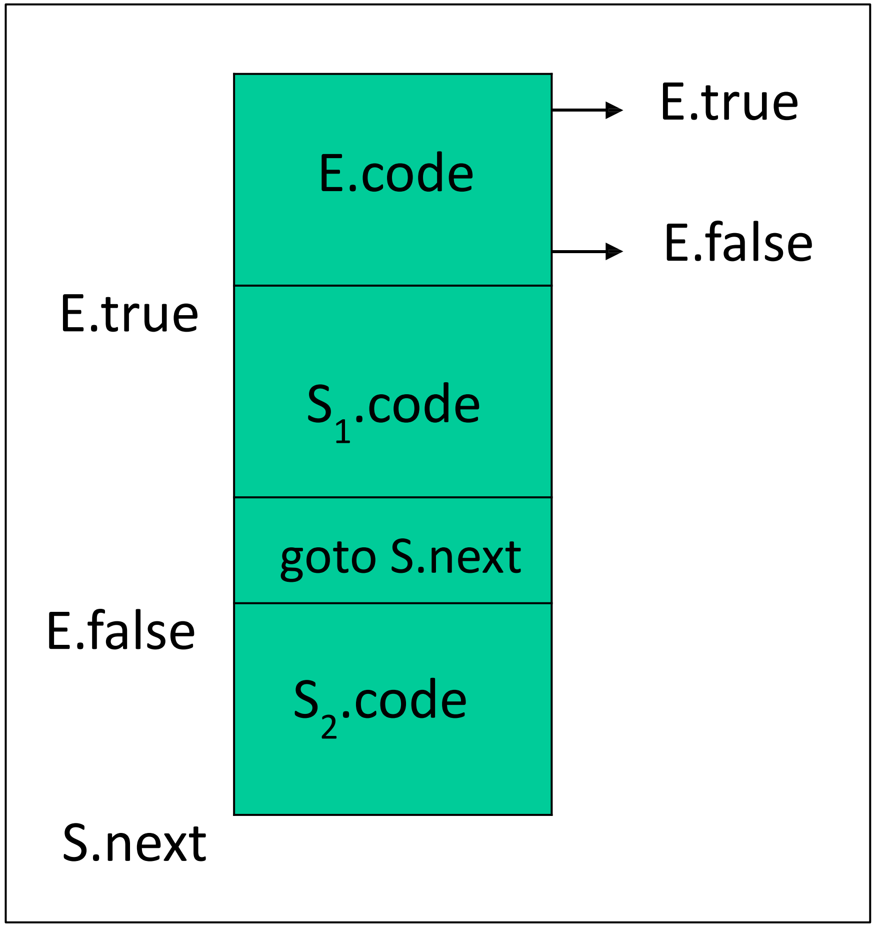
E.false = S.next

S_1.next = S.next

S.code = E.code ||

gen(E.true ':') ||

S_1.code



$S \longrightarrow$ if E then S_1 else S_2

E.true = newlabel

E.false = newlabel

S_1.next = S.next

S_2.next = S.next

S.code = E.code ||

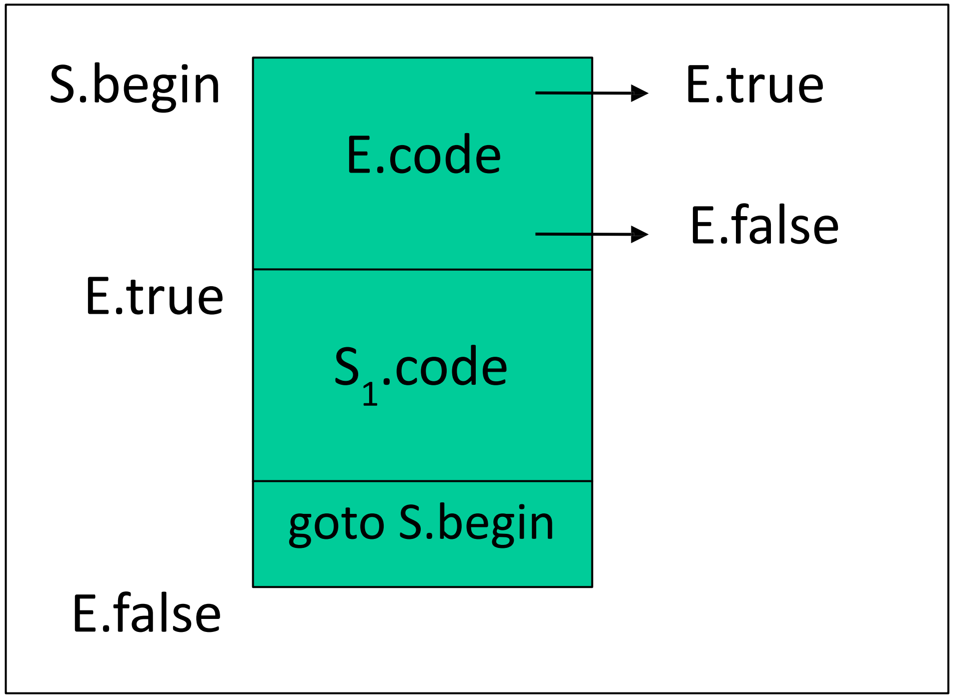
gen(E.true ':') ||

S_1.code ||

gen(goto S.next) ||

gen(E.false ':') ||

S_2.code



$S \rightarrow$ while E do S_1

S.begin = newlabel

E.true = newlabel

E.false = S.next

S_1.next = S.begin

S.code = gen(S.begin ':') ||
E.code ||
gen(E.true ':') ||
S_1.code ||
gen(goto S.begin)

Example …

Code for

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

```
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:     t1 = Y - Z
        X= t1
        goto L1

Lnext:
```