CSL302: Compiler Design

Intermediate Code Generation

Vishwesh Jatala

Assistant Professor

Department of CSE

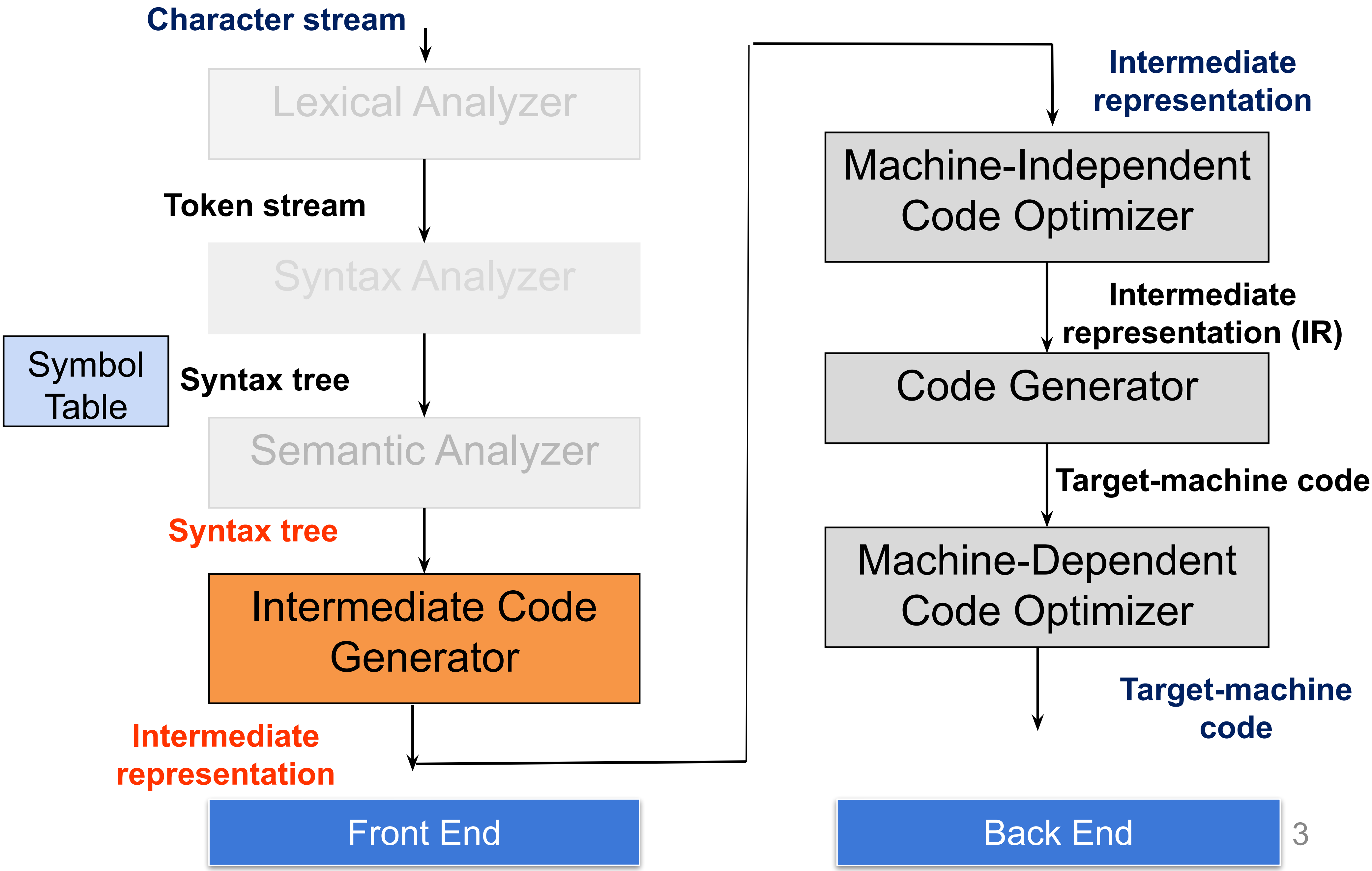
Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

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/*

Compiler Design



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

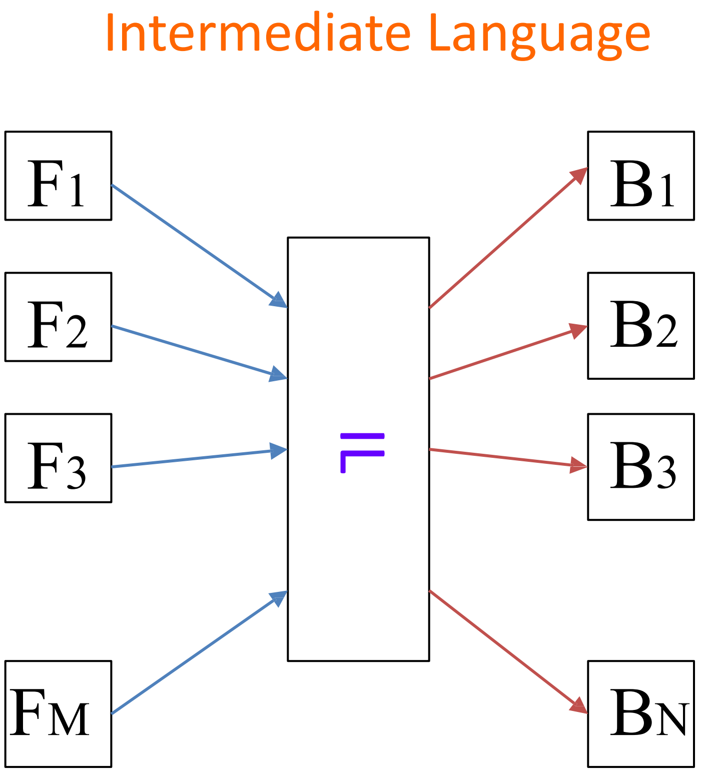
Target-machine code

Machine-Dependent Code Optimizer

Target-machine code

Back End

Recap: Why Intermediate Code?



Requires M front ends
And N back ends

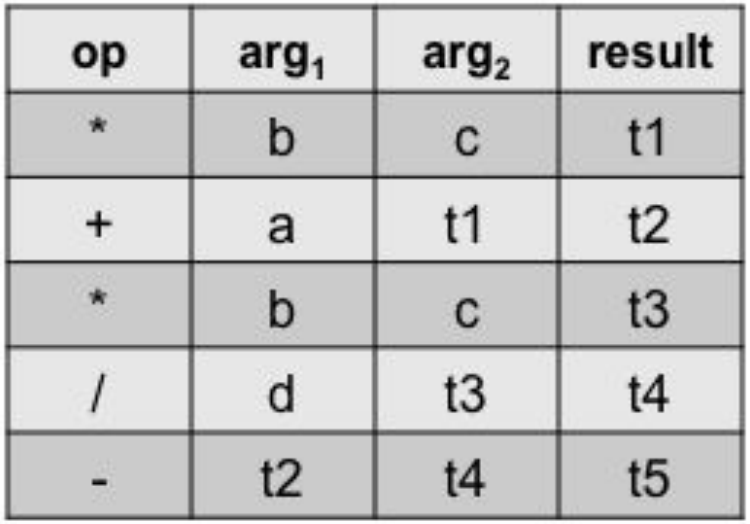
- M front ends, N back ends
- Facilitates machine independent code optimizers

Intermediate Code Representations

- Three address code (TAC)
  - Instructions are very simple
  - Maximum three addresses in an instruction
  - LHS is the target
  - RHS has at most two sources and one operator
  - address:
    - Name: programmer defined
    - Constant
    - Temporary variables

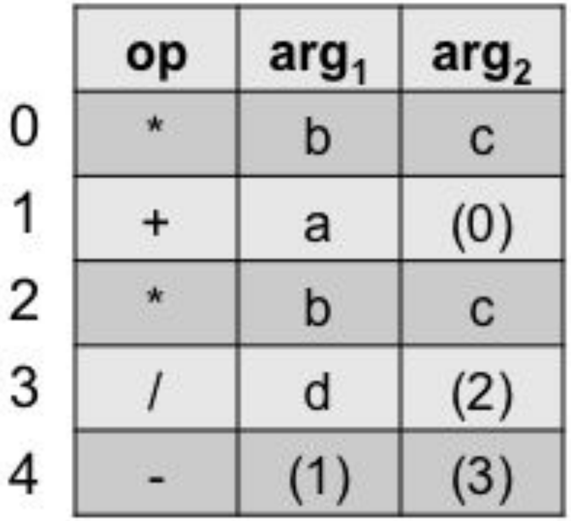
```
t = a + 5
p = t * b
q = p  - c
p = q
p = -e
q = p + q
```

Implementations of TAC



op	arg$_1$	arg$_2$	result
*	b	c	t1
+	a	t1	t2
*	b	c	t3
/	d	t3	t4
-	t2	t4	t5

Quadruples



	op	arg$_1$	arg$_2$
0	*	b	c
1	+	a	(0)
2	*	b	c
3	/	d	(2)
4	-	(1)	(3)

Triples

Three address code

- Assignment
  - x = y op z
  - x = op y
  - x = y
- Jump
  - goto L
  - if x relop y goto L
- Indexed assignment
  - x = y[i]
  - x[i] = y
- Function
  - param x
  - call p,n
  - return y
- Pointer
  - x = &y
  - x = *y
  - *x = y

Intermediate Code Generation

- Expressions
- Statements
  - Simple statements
  - Conditional statements
  - Control flow statements
    - if, if-else, while.
  - Declarations
  - Arrays
  - Functions

Intermediate Code Generation

- Expressions
- Statements
  - Simple statements
  - Conditional statements
  - Control flow statements
    - if, If-else, while
  - Declarations
  - Arrays
  - Functions

Syntax directed translation of expression into 3-address code

Expression: a + b * c

Three-address code:

t1 = b * c

t2 = a + t1

Syntax directed translation of expression into 3-address code

- newtmp() -> creates a new temporary variable
- **gen(…):** produce sequence of three address statements
  - The statements themselves are kept in some data structure, e.g. list
  - SDD operations described using pseudo code

Syntax directed translation of expression into 3-address code

- Attribute:
  - ***E.place***, a name that will hold the value of E

Syntax directed translation of expression into 3-address code

$E \longrightarrow E_1 + E_2$

E.place:= newtmp()

gen(E.place := E_1.place + E_2.place)

Syntax directed translation of expression into 3-address code

$E \rightarrow E_1 * E_2$

E.place:= newtmp()

gen(E.place := E_1.place * E_2.place)

Syntax directed translation of expression into 3-address code

S → id := E

S.code := gen(id.place:= E.place)

Syntax directed translation of expression …

$E \rightarrow -E_1$

E.place := newtmp()

gen(E.place := - E_1.place)

$E \rightarrow (E_1)$

E.place := E_1.place

$E \rightarrow$ id

E.place := id.place

Exercise

Generate the Intermediate representation for

a = b * -c + b * c

Exercise

Expression: a = b * -c + b * c

Generated code:

t_1 = -c

t_2 = b * t_1

t_3 = b *c

t_4 = t_2 + t_3

a = t_4

Boolean Expressions

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

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
        gen(if E1.place < E2.place goto nextstat+3)
        gen(E.place = 0)
        gen(goto nextstat+2)
        gen(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 ':=' E_1.place 'or' E_2.place)

$E \longrightarrow E_1$ and E_2

E.place:= newtmp

gen(E.place ':=' E_1.place 'and' E_2.place)

$E \longrightarrow$ not E_1

E.place := newtmp

gen(E.place ':=' 'not' E_1.place)

Syntax directed translation of boolean expressions

```
E ⟶ true
        E.place := newtmp
        gen(E.place = '1')

E ⟶ false
        E.place := newtmp
        gen(E.place = '0')
```

Boolean Expressions

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

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