NEW YORK UNIVERSITY

CSCI-GA.2130-001

Compiler Construction

Lecture 10: Intermediate-Code Generation II

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu

Type Checking

- Language comes with type system
  - Set of rules
  - What types are there?
  - Where do they appear?
- Compiler's job
  - Assign type expression to each component
  - Determine that these type expressions conform to the type system

Type Checking: Dynamic and Static

- Type checking can be done dynamically for any language (i.e at run-time)
  - compiler generates code to do the checks at run-time
- Better to do it statically (i.e. at compile-time)
- A sound type system eliminates the need for dynamic checking.
- A language is strongly typed if compiler guarantees that program it accepts will run without type error.
  - Examples of strongly typed: Java, C#, Pascal, Ruby, Python

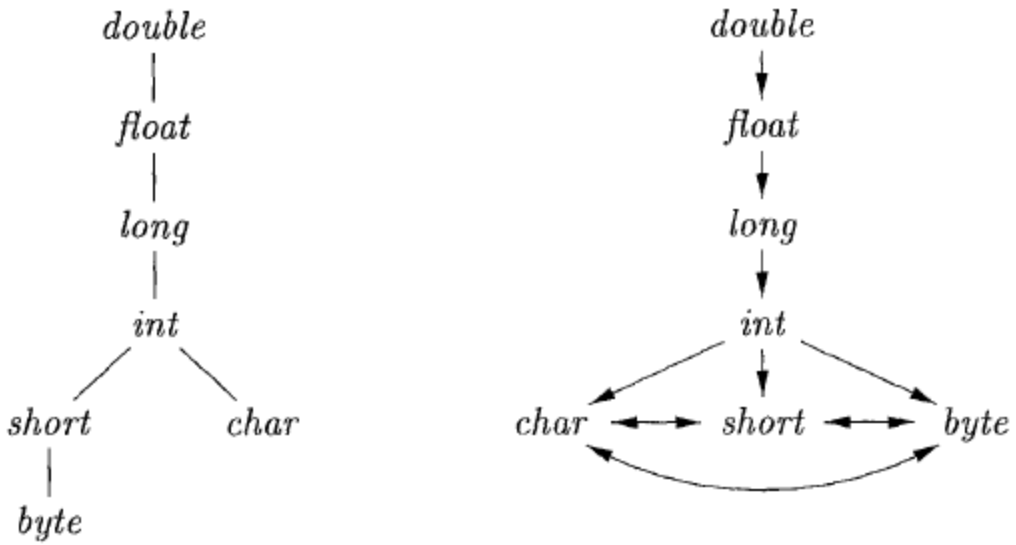
Rules for Type Checking

- Type Synthesis
  - Builds the type of an expression from the types of its subexpressions
  - Requires names to be declared before usage
- Type inference
  - determines the type of a construct from the way it is used

Type Conversions

- Type conversion rules vary from language to language
- Explicit type conversion
  - Must be done by the programmer
  - Called cast
- Implicit type conversion
  - Done automatically by the compiler
  - Called coercions
  - Widening (used most often) vs narrowing

Type Conversions: Example



(a) Widening conversions (b) Narrowing conversions

Conversions between primitive types in Java

Type Conversions: Example

$$
\begin{array}{lll}
E \rightarrow E_1 + E_2 & \{ & E.type = max(E_1.type, E_2.type); \\
 & & a_1 = widen(E_1.addr, E_1.type, E.type); \\
 & & a_2 = widen(E_2.addr, E_2.type, E.type); \\
 & & E.addr = \textbf{new } Temp\,(); \\
 & & gen(E.addr\ '='\ a_1\ '+'\ a_2); \ \}
\end{array}
$$

$max(t_1, t_2)$ takes two types t_1 and t_2 and returns the maximum (or least upper bound) of the two types in the widening hierarchy. It declares an error if either t_1 or t_2 is not in the hierarchy; e.g., if either type is an array or a pointer type.

```
Addr widen(Addr a, Type t, Type w)
        if ( t = w ) return a;
        else if ( t = integer and w = float ) {
                temp = new Temp();
                gen(temp '=' '(float)' a);
                return temp;
        }
        else error;
}
```

Type Conversions: Example

$$
\begin{aligned}
E \rightarrow E_1 + E_2 \quad \{\ & E.type = max(E_1.type, E_2.type); \\
& a_1 = widen(E_1.addr, E_1.type, E.type); \\
& a_2 = widen(E_2.addr, E_2.type, E.type); \\
& E.addr = \textbf{new}\ Temp\,(); \\
& gen(E.addr\ '='\ a_1\ '+'\ a_2); \}
\end{aligned}
$$

1.Determine the type of the result

2. Convert both operands to the result type (may involve generating an extra instruction.

3. Perform the operation on the new type.

Control Flow

- Boolean expressions
- Alter the flow of control

$$B \rightarrow B \,||\, B \mid B \,\&\&\, B \mid !\, B \mid (\, B\,) \mid E \,\mathbf{rel}\, E \mid \mathbf{true} \mid \mathbf{false}$$

Are left associative

Has higher precedence

<, <=, =, !=, >, >=

Control Flow: Short-Circuit

- The operators of the Boolean expression do not appear in the code

Example:

```
if ( x < 100 || x > 200 && x != y ) x = 0;
```

Control Flow: Short-Circuit

- The operators of the Boolean expression do not appear in the code

Example:

```
if ( x < 100 || x > 200 && x != y ) x = 0;
```

```
        if x < 100 goto L2
        ifFalse x > 200 goto L1
        ifFalse x != y goto L1
L2:     x = 0
L1:
```

Do you see
|| or && in the code?

Syntax-Directed Definitions for Flow of Control

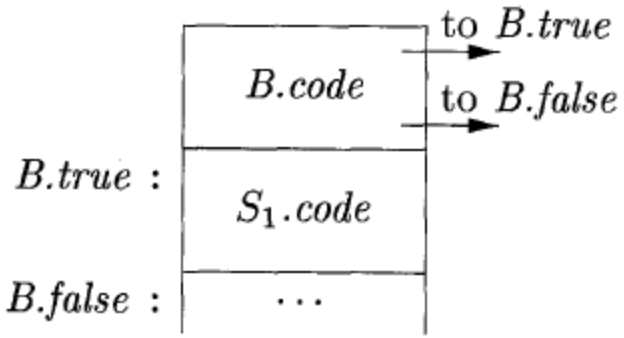
PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen(\texttt{'goto'}\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen(\texttt{'goto'}\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

newlabel() creates a new label each time it is called

label(L) attaches label L to the next three-address instruction to be generated.

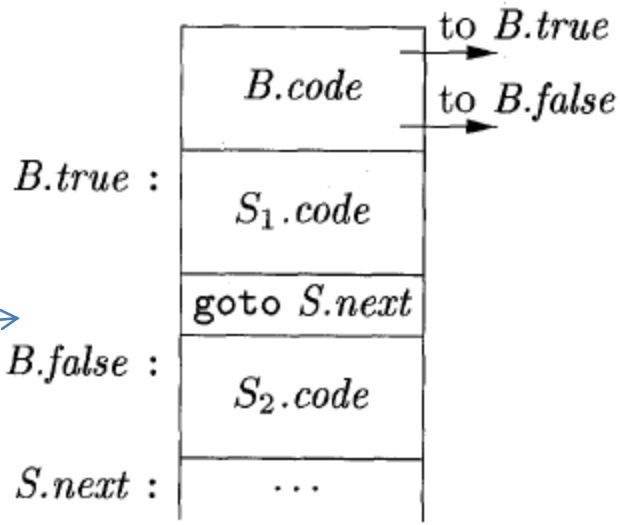
Syntax-Directed Definitions for Flow of Control

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$



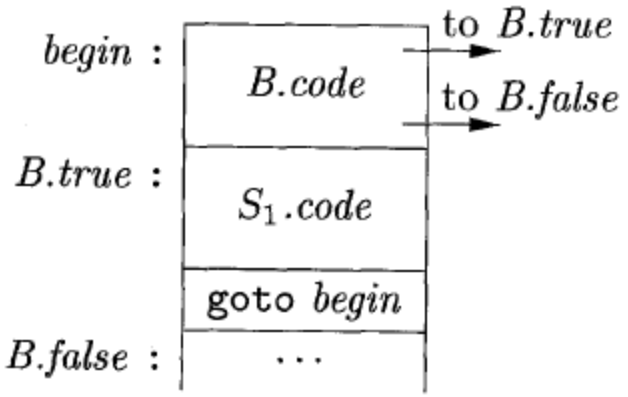
Syntax-Directed Definitions for Flow of Control

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$



Syntax-Directed Definitions for Flow of Control

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$



Control-Flow Translation of Boolean Expressions

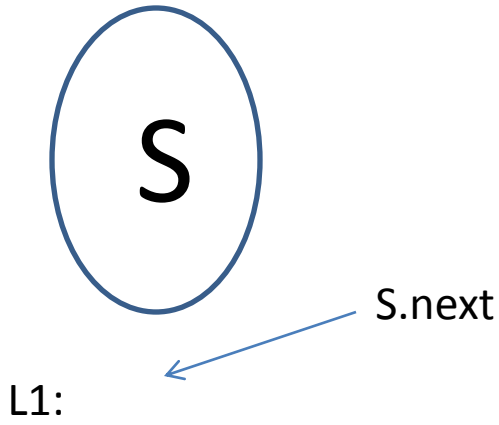
PRODUCTION	SEMANTIC RULES
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \,\&\&\, B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$
$B \rightarrow ! B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$ $\mid\mid gen('\texttt{goto}'\ B.false)$
$B \rightarrow$ **true**	$B.code = gen('\texttt{goto}'\ B.true)$
$B \rightarrow$ **false**	$B.code = gen('\texttt{goto}'\ B.false)$

If B.condition goto B.true
goto B.false

S

```
if( x < 100 || x > 200 && x != y ) x = 0;
```

$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$



B S_1

```
if( x < 100 || x > 200 && x != y ) x = 0;
```

$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$

if B.condition goto B.true

goto B.false

L2: x = 0 ← S_1.code

L1:

B_1 B_2

```
if( x < 100 || x > 200 && x != y ) x = 0;
```

$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$

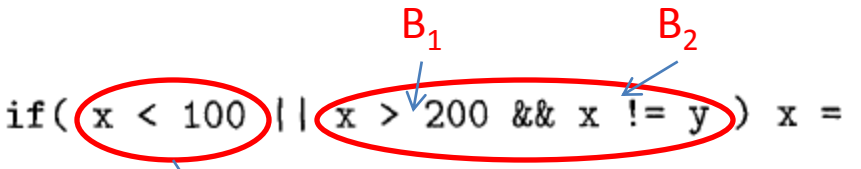
if B1.condition goto B1.true
goto B1.false
L3: if B2.condition goto B2.true
goto B2.false

L2: x = 0

L1:

B.true

B.false



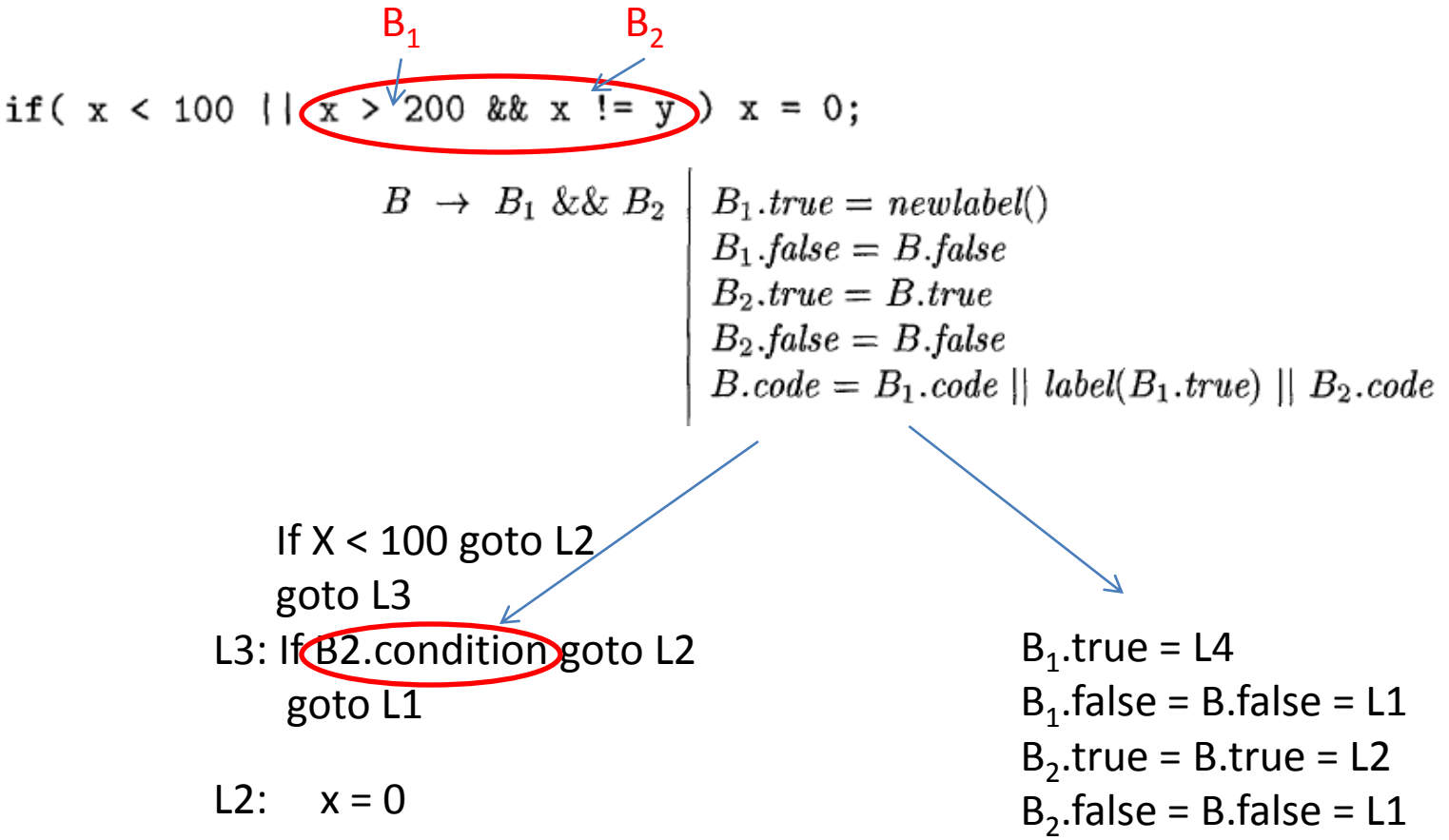
$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \;\|\|\; label(B_1.true) \;\|\|\; B_2.code$

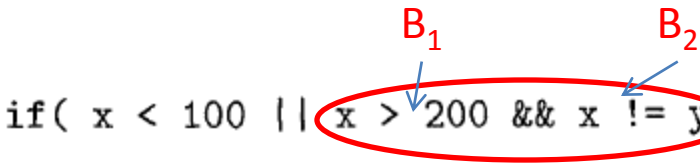
If X1.condition goto L2
goto L3

L3: If B2.condition goto L2
goto L1

L2: x = 0

L1:


B1
B2
if(x < 100 || x > 200 && x != y) x = 0;
B → B1 && B2
B1.true = newlabel()
B1.false = B.false
B2.true = B.true
B2.false = B.false
B.code = B1.code || label(B1.true) || B2.code
If X < 100 goto L2
goto L3
L3: If B2.condition goto L2
goto L1
L2: x = 0
L1:
B1.true = L4
B1.false = B.false = L1
B2.true = B.true = L2
B2.false = B.false = L1



$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \;\|\|\; label(B_1.true) \;\|\|\; B_2.code$

```
      if x < 100 goto L2
      goto L3
L3:   if x > 200 goto L4
      goto L1
L4:   if x != y goto L2
      goto L1
L2:   x = 0
L1:
```

B_1.true = L4
B_1.false = B.false = L1
B_2.true = B.true = L2
B_2.false = B.false = L1

If B1.condition goto L4
goto L1
L4: If B2.condition goto L2
goto L1

```
if( x < 100 || x > 200 && x != y ) x = 0;
```

```
        if x < 100 goto L2
        goto L3
L3:     if x > 200 goto L4
        goto L1
L4:     if x != y goto L2
        goto L1
L2:     x = 0
L1:
```

So…

- Skim: 6.5.3, 6.5.4, 6.5.5, 6.6.5, 6.7,6.8, 6.9
- Read rest of 6.5->6.9