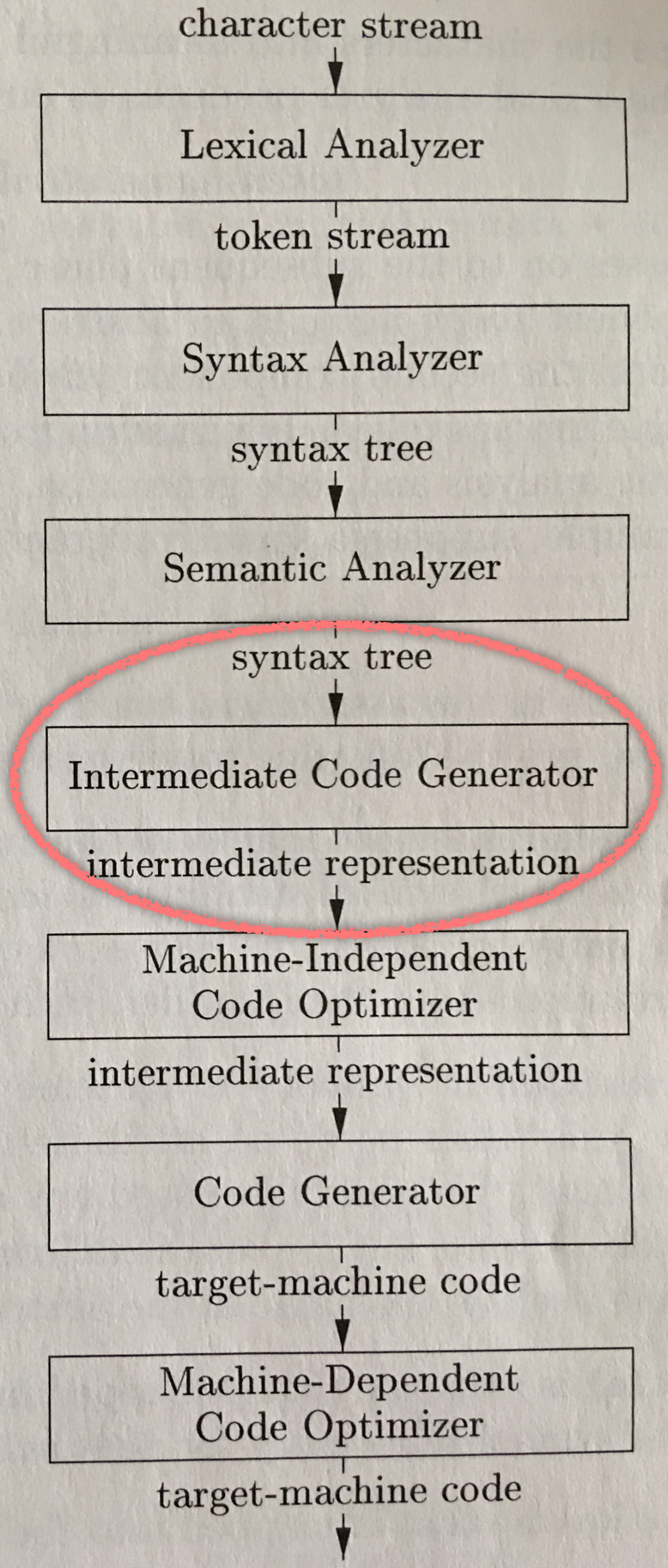
CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

Phases of a compiler

Intermediate Representation (IR): specification and generation

Figure 1.6, page 5 of text



Boolean expressions

- A concrete exercise - how is this translated?
- if (r < s | (r = s & 0 < s)) then { A } else { B }

Here's a summary of the Intermediate Representation (IR) that we'll be using.

Three address code instructions
(see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Boolean expressions

- A concrete exercise - how is this translated?
- if (r < s | (r = s & 0 < s)) then { A } else { B }

Exercise: try to come up with a suitable translation.

Three address code instructions
(see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Boolean expressions

- A concrete exercise - how is this translated?
- if (r < s | (r = s & 0 < s)) then { A } else { B }

This has the same form as our example:

if (X | (Y & Z)) then { A } else { B }
is translated as

```
      if X goto LA
      ifFalse Y goto LB
      ifFalse Z goto LB
LA:   A
      goto END
LB:   B
END: (next instruction)
```

Three address code instructions
(see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Boolean expressions

- A concrete exercise - how is this translated?

- if (r < s | (r = s & 0 < s)) then { A } else { B }
- if (x | (y & z)) then { A } else { B }

This has the same form as our example:

if (X | (Y & Z)) then { A } else { B }
is translated as

```
      if X goto LA
      ifFalse Y goto LB
      ifFalse Z goto LB
LA:   A
      goto END
LB:   B
END: (next instruction)
```

Three address code instructions
(see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Boolean expressions

- A concrete exercise - how is this translated?

- if (r < s | (r = s & 0 < s)) then { A } else { B }
- if (x | (y & z)) then { A } else { B }

This has the same form as our example:

if (r<s | (Y & Z)) then { A } else { B } is translated as

```
      if r<s goto LA
      ifFalse Y goto LB
      ifFalse Z goto LB
LA:   A
      goto END
LB:   B
END:  (next instruction)
```

Three address code instructions (see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Boolean expressions

- A concrete exercise - how is this translated?

- if (r < s | (r = s & 0 < s)) then { A } else { B }
- if (x | (y & z)) then { A } else { B }

This has the same form as our example:

if (r<s | (r=s & Z)) then { A } else { B }

is translated as

```
      if r<s goto LA
      ifFalse r=s goto LB
      ifFalse Z goto LB
LA:   A
      goto END
LB:   B
END: (next instruction)
```

Three address code instructions
(see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Boolean expressions

- A concrete exercise - how is this translated?

- if (r < s | (r = s & 0 < s)) then { A } else { B }
- if (x | (y & z)) then { A } else { B }

This has the same form as our example:

if (r<s | (r=s & 0<s)) then { A } else { B }

is translated as

```
      if r<s goto LA
      ifFalse r=s goto LB
      ifFalse 0<s goto LB
LA:   A
      goto END
LB:   B
END: (next instruction)
```

Three address code instructions
(see 6.2.1, pages 364-5)

1. x = y op z
2. x = op y (treat i2r and r2i as unary ops)
3. x = y
4. goto L
5. if x goto L / ifFalse x goto L
6. if x relop y goto L
7. function calls:
   - param x
   - call p, n
   - y = call p
   - return y
8. x = y[i] and x[i] = y
9. x = &y, x = *y, *x = y

We'll start with these.

We'll spend significant time on function calls later.

We'll explore these as needed later on.

Backpatching

Allows jump targets to be filled in during a one-pass parse.

When (forward) jumps are needed, keep a list of where the addresses need to be inserted.

Once address is known, go back and fill in the address ("backpatching").

6.7 Backpatching

"For specificity, we generate instructions into an instruction array, and labels will be indices into this array." [p. 410]

We have an instruction pointer (the first available index in the array), called nextinstr.

6.7 Backpatching

page 410

makelist(i) creates a new list containing only i, an index into the array of instructions; makelist returns a pointer to the newly created list.

merge(p1,p2) concatenates the lists pointed to by p1 and p2, and returns a pointer to the concatenated list.

backpatch(p,i) inserts i as the target label for each of the instructions on the list pointed to by p

6.7.1 Backpatching for Boolean Expressions

B -> B1 || B2	B1.true = B.true B1.false = newlabel() B2.true = B.true B2.false = B.false B.code = B1.code || label(B1.false) || B2.code
B -> B1 || M B2	backpatch(B1.falselist, M.instr) B.truelist = merge(B1.truelist, B2.truelist) B.falselist = B2.falselist

6.7.1 Backpatching for Boolean Expressions

B -> B1 || B2	B1.true = B.true B1.false = newlabel() B2.true = B.true B2.false = B.false B.code = B1.code || label(B1.false) || B2.code
B -> B1 ||	

Assume that B1 and B2 generate code stored as attributes of the parse tree nodes.

Actions happen at reduction step of rule B -> B1 || B2

6.7.1 Backpatching for Boolean Expressions

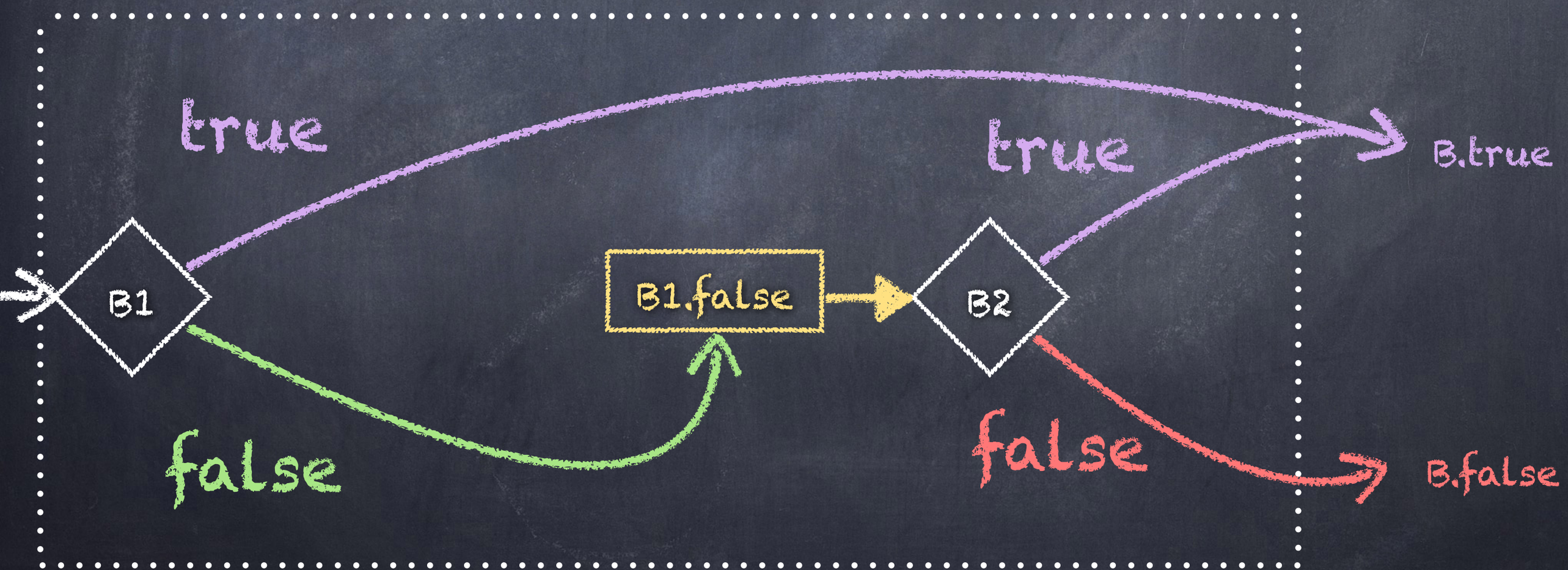
Assume that B1 and B2 generate code directly, rather than storing it in the tree.

Actions happen at reduction step of rule
B -> B1 || M B2

B -> B1	...code
B -> B1 || M B2	backpatch(B1.falselist, M.instr) B.truelist = merge(B1.truelist, B2.truelist) B.falselist = B2.falselist

6.6.4 Control-flow translation of Boolean Expressions

B -> B1 || M B2	backpatch(B1.falselist, M.instr) B.truelist = merge(B1.truelist, B2.truelist) B.falselist = B2.falselist

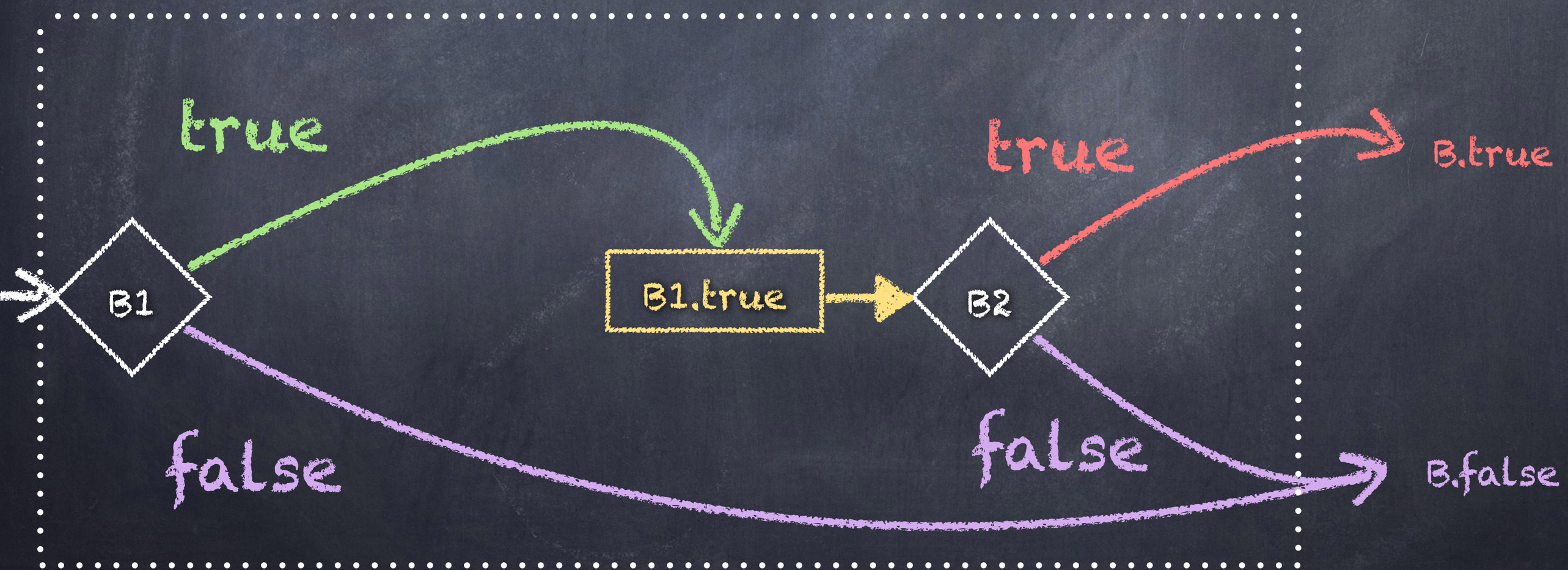


6.7.1 Backpatching for Boolean Expressions

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
B -> B1 && M B2	backpatch(B1.truelist, M.instr) B.truelist = B2.truelist B.falselist = merge(B1.falselist, B2.falselist)

6.6.4 Control-flow translation of Boolean Expressions

B -> B1 && M B2	backpatch(B1.truelist, M.instr) B.truelist = B2.truelist B.falselist = merge(B1.falselist, B2.falselist)

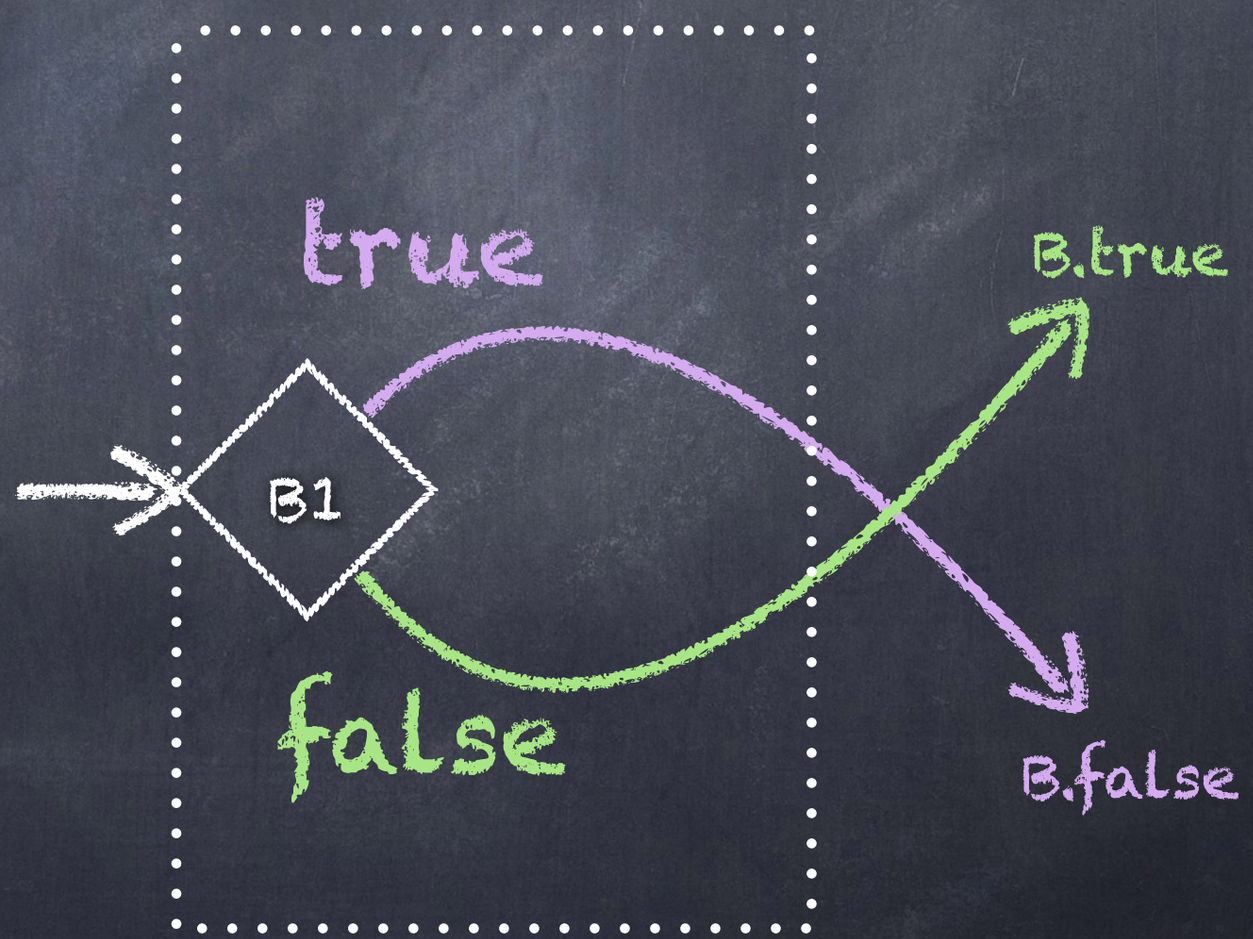


6.7.1 Backpatching for Boolean Expressions

B -> ! B1	B1.true = B.false B1.false = B.true B.code = B1.code
B -> ! B1	B.truelist = B1.falselist B.falselist = B1.truelist

6.6.4 Control-flow translation of Boolean Expressions

B -> ! B1	B.truelist = B1.falselist B.falselist = B1.truelist



6.7.1 Backpatching for Boolean Expressions

B -> (B1)	B1.true = B.true B1.false = B.false B.code = B1.code
B -> (B1)	B.truelist = B1.truelist B.falselist = B1.falselist

6.7.1 Backpatching for Boolean Expressions

B -> E1 rel E2	B.code = E1.code || E2.code || gen ('if' E1.addr rel.op E2.addr 'goto B.true') || gen('goto' B.false)
B -> E1 rel E2	B.truelist = makelist(nextinstr) B.falselist = makelist(nextinstr + 1) gen('if' E1.addr rel.op E2.addr 'goto _') gen('goto _')

6.7.1 Backpatching for Boolean Expressions

B -> true	B.code = gen('goto' B.true)
B -> false	B.code = gen('goto' B.false)
B -> true	B.truelist = makelist(nextinstr) gen('goto _')
B -> false	B.falselist = makelist(nextinstr) gen('goto _')

6.7.1 Backpatching for Boolean Expressions

M -> ε	M.instr = nextinstr

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

100:

Let's assume nextinstr has value 100.

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}

Where does this come from?

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ____ 101: goto ____	truelist = {100} falselist = {101}

B -> E1 rel E2	B.truelist = makelist(nextinstr) B.falselist = makelist(nextinstr + 1) gen('if' E1.addr rel.op E2.addr 'goto _') gen('goto _')

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}
M	M.instr = 102	

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}
M	M.instr = 102	
x > 200	102: if x > 200 goto ___ 103: goto ___	truelist = {102} falselist = {103}

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}
M	M.instr = 102	
x > 200	102: if x > 200 goto ___ 103: goto ___	truelist = {102} falselist = {103}
M	M.instr = 104	

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}
M	M.instr = 102	
x > 200	102: if x > 200 goto ___ 103: goto ___	truelist = {102} falselist = {103}
M	M.instr = 104	
x != y	104: if x != y goto ___ 105: goto ___	truelist = {104} falselist = {105}

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}
M	M.instr = 102	
x > 200	102: if x > 200 goto ___ 103: goto ___	truelist = {102} falselist = {103}
M	M.instr = 104	
x != y	104: if x != y goto ___ 105: goto ___	truelist = {104} falselist = {105}
x>200 && x!=y	backpatch({102},104) 102: if x > 200 goto 104 103: goto ___	truelist = {104} falselist = {103,105}

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

B -> B1 && B2

B -> x > 200 && x != y

x < 100	100: 101:	
M	M.instr = 102	
x > 200	102: if x > 200 goto ____ 103: goto ____	truelist = {102} falselist = {103}
M	M.instr = 104	
x != y	104: if x != y goto ____ 105: goto ____	truelist = {104} falselist = {105}
x>200 && x!=y	backpatch({102},104) 102: if x > 200 goto 104 103: goto ____	truelist = {104} falselist = {103,105}

backpatch(B1.truelist, M.instr)

B.truelist = B2.truelist

B.falselist = merge(B1.falselist, B2.falselist)

Example 6.24

x < 100 || x > 200 && x != y

B -> B1 || M B2 and B -> B1 && M B2

x < 100	100: if x < 100 goto ___ 101: goto ___	truelist = {100} falselist = {101}
M	M.instr = 102	
x > 200	102: if x > 200 goto ___ 103: goto ___	truelist = {102} falselist = {103}
M	M.instr = 104	
x != y	104: if x != y goto ___ 105: goto ___	truelist = {104} falselist = {105}
x>200 && x!=y	backpatch({102},104) 102: if x > 200 goto 104 103: goto ___	truelist = {104} falselist = {103,105}
x < 100 || x > 200 && x != y	backpatch({101},102) 100: if x < 100 goto ___ 101: goto 102	truelist = {100,104} falselist = {103,105}

How does this play out during parse in array

Example 6.24

x < 100 || x > 200 && x != y

100: if x < 100 goto ____

101: goto ____

truelist = {100}

falselist = {101}

Example 6.24

$x < 100 \;||\; x > 200 \;\&\&\; x \mathrel{!}= y$

```
100: if x < 100 goto ____
101: goto ____
102: if x > 200 goto ____
103: goto ____
```

truelist = {100}
falselist = {101}
truelist = {102}
falselist = {103}

Example 6.24

x < 100 || x > 200 && x != y

```
100: if x < 100 goto ____
101: goto ____
102: if x > 200 goto ____
103: goto ____
104: if x != y goto ____
105: goto ____
```

truelist = {100}
falselist = {101}
truelist = {102}
falselist = {103}
truelist = {104}
falselist = {105}

Example 6.24

$x < 100 \;||\; x > 200 \;\&\&\; x \mathrel{!}= y$

```
100: if x < 100 goto ____
101: goto ____
102: if x > 200 goto 104
103: goto ____
104: if x != y goto ____
105: goto ____
```

truelist = {100}
falselist = {101}
truelist = {104}
falselist = {103,105}

Example 6.24

x < 100 || x > 200 && x != y

```
100: if x < 100 goto ____
101: goto 102
102: if x > 200 goto 104
103: goto ____
104: if x != y goto ____
105: goto ____
```

truelist = {100,104}
falselist = {103,105}

Example 6.24

x < 100 || x > 200 && x != y

```
100: if x < 100 goto ____
101: goto 102
102: if x > 200 goto 104
103: goto ____
104: if x != y goto ____
105: goto ____
```

truelist = {100,104}
falselist = {103,105}

The remaining open jumps will be backpatched by other instructions, outside this expression