Intermediate Code Generation

Sudakshina Dutta

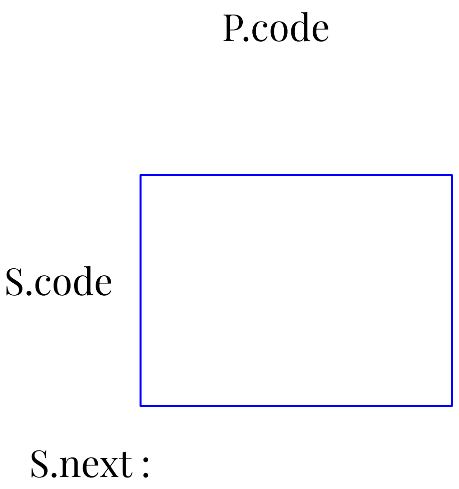
IR Generation

```
a = 5;
```

```
    a = 5;
L :
```

Non-terminal	Semantic rules

P -> S	S.next = new label()
	P.code = S.code || S.next



P.code

S.code

S.next :

Types of statement

- Assignment
- If-statement
- If-else statement
- While statement
- Sequence of statements

IR Generation

a = 5;

a = 5;

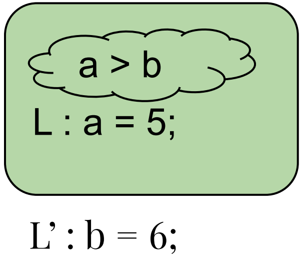
S.code

Non-terminal	Semantic rules
S -> assign	S.code = assign.code

a = b;

IR Generation

```
if (a > b)
   a = 5;
b = 6;
```



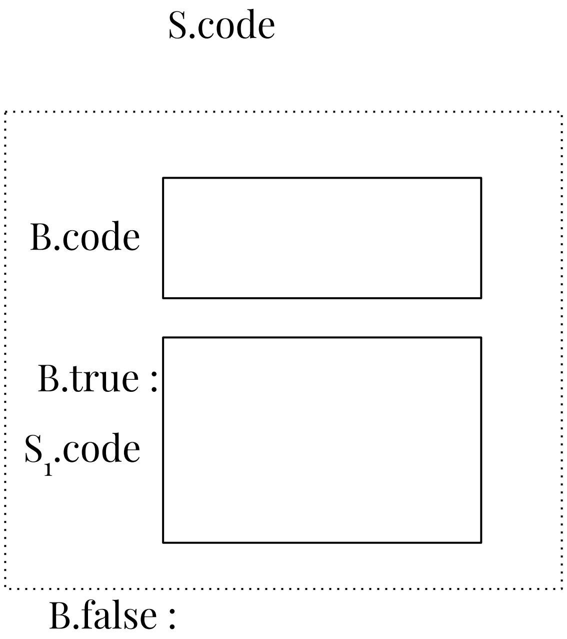
a > b

L : a = 5;

L' : b = 6;

Non-terminal	Semantic rules

S -> if (B) S_1	B.true = new label();
	B.false = S_1.next = S.next
	S.code = B.code || label(B.true) || S_1.code



S.code

B.code

B.true :

S_1.code

B.false :

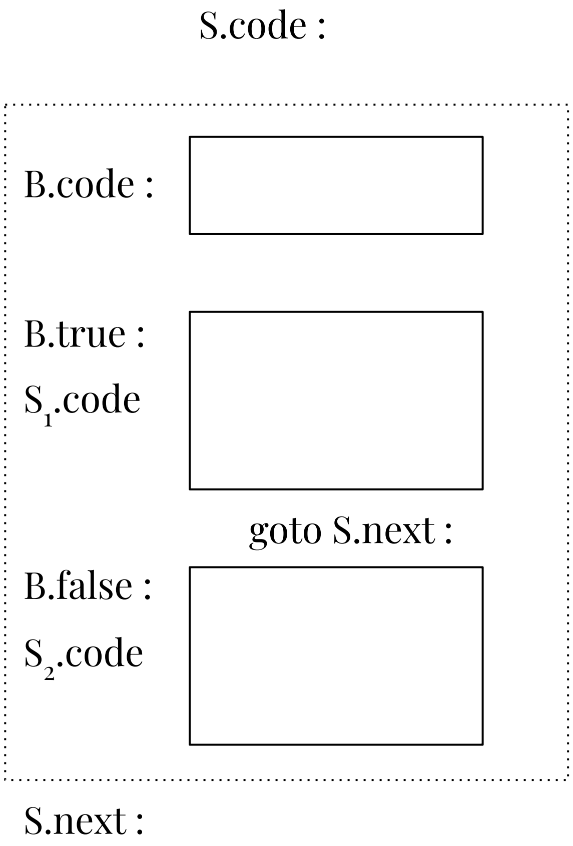
IR Generation

```
if (a > b)
   a = 5;
else
   a = 6;
```

```
   a > b
L : a = 5;
     goto L”;
L’ : a = 6;
```

L” :

Non-terminal	Semantic rules
S -> 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 || label(B.true) || S_1.code || gen(‘goto’ S.next) || label(B.false) || S_2.code



IR Generation

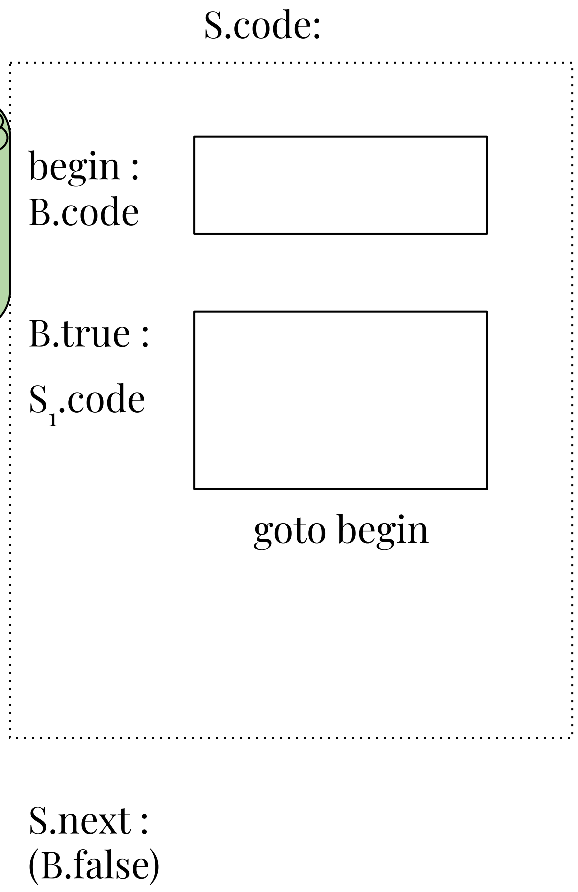
```
while (a < b)
  a = a + 5;
```

```
begin : a < b

L : a = a + 5
    goto begin
```

L’ :

Non-terminal	Semantic rules

S -> while (B) S_1	begin = new label()
	B.true = new label()
	B.false = S.next
	S_1.next = begin
	S.code = label(begin) || B.code || label(B.true) || S_1.code || gen(‘goto’ begin)



S.code:

begin :
B.code

B.true :

S_1.code

goto begin

S.next :
(B.false)

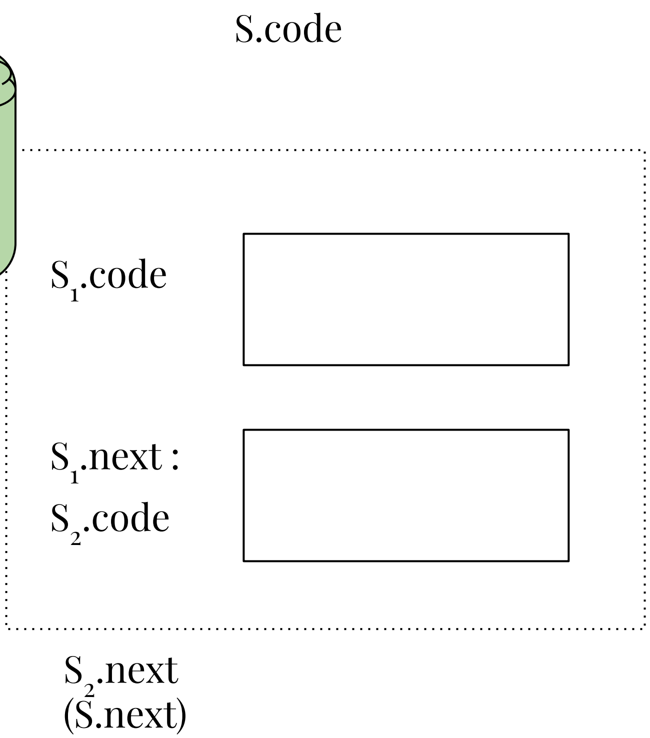
IR Generation

```
while (a < b)
   a = a + 5;
a = a + 7
```

```
begin : a < b

L : a = a + 5
     goto begin
```

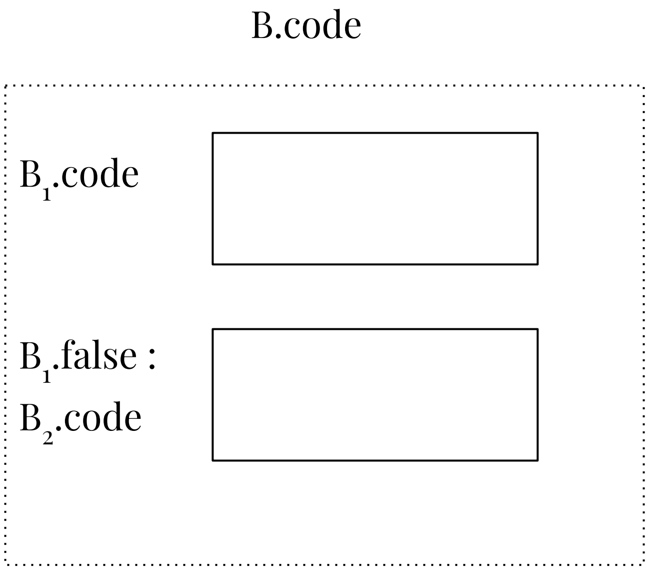
L' : a = a + 7
L" :

Non-terminal	Semantic rules
S -> S_1S_2	S_1.next = new label() S_2.next = S.next S.code = S_1.code || label(S_1.next) || S_2.code



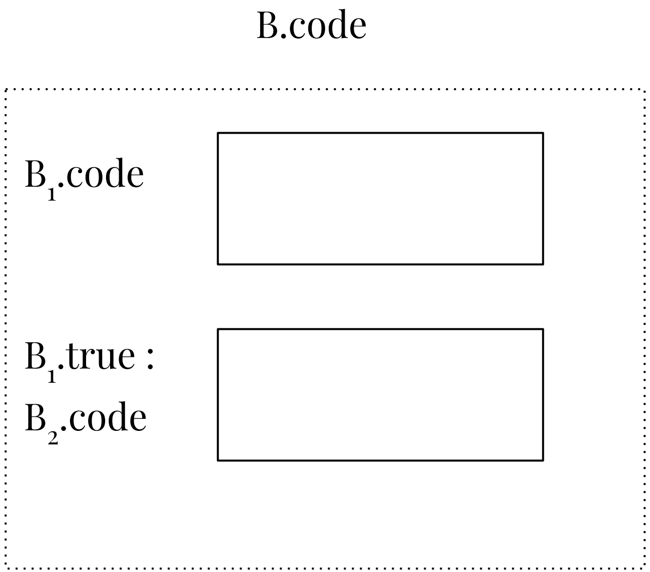
IR Generation for boolean expression

Non-terminal	Semantic rules
B -> B_1|| 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 || label(B_1.false) || B_2.code



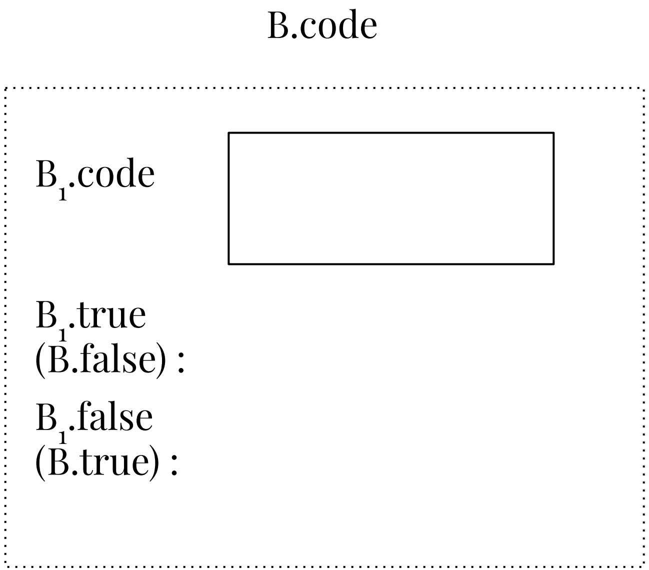
IR Generation for boolean expression

Non-terminal	Semantic rules
B -> 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



IR Generation for boolean expression

Non-terminal	Semantic rules
B -> ! B_1	B_1.true = B.false B_1.false = B.true B.code = B_1.code



B.code

B_1.code

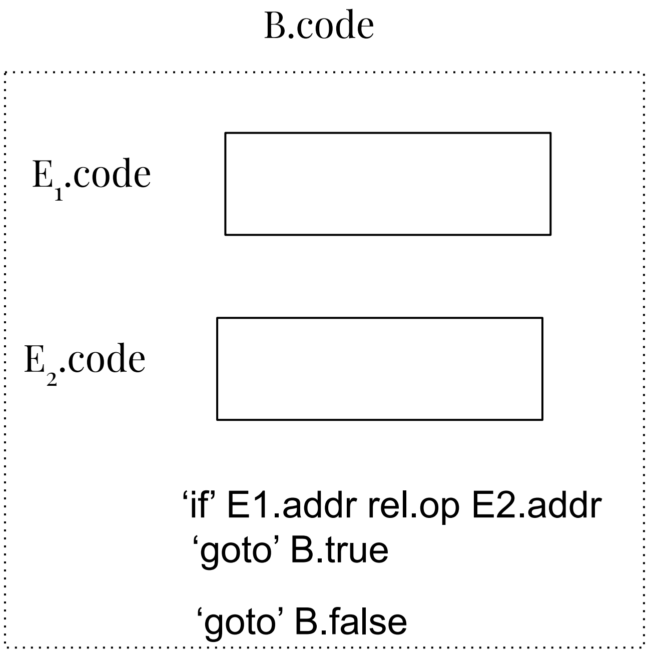
B_1.true (B.false) :

B_1.false (B.true) :

IR Generation for boolean expression

Non-terminal	Semantic rules

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)



IR Generation for boolean expression

Non-terminal	Semantic rules
B -> true	B.code = gen(‘goto’ B.true)

B.code

‘goto’ B.true

IR Generation for boolean expression

Non-terminal	Semantic rules
B -> false	B.code = gen(‘goto’ B.false)

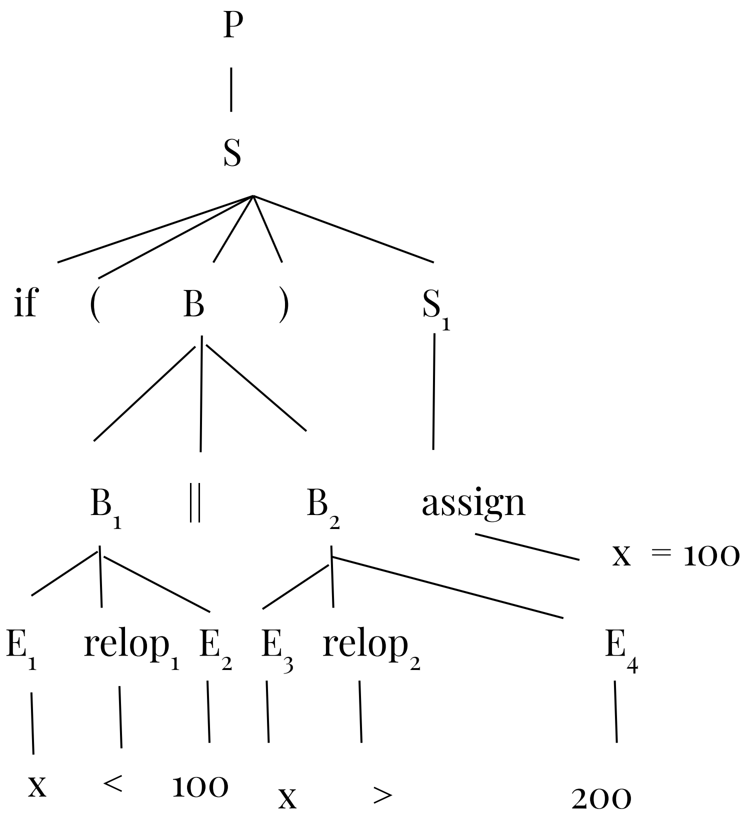
B.code

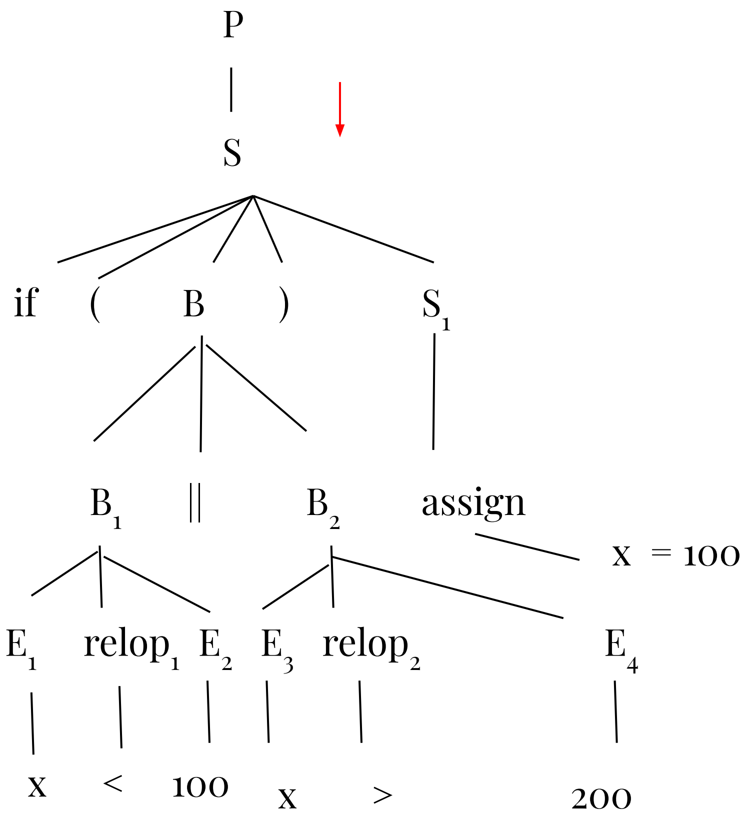
‘goto’ B.false

Example

Consider the following string

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

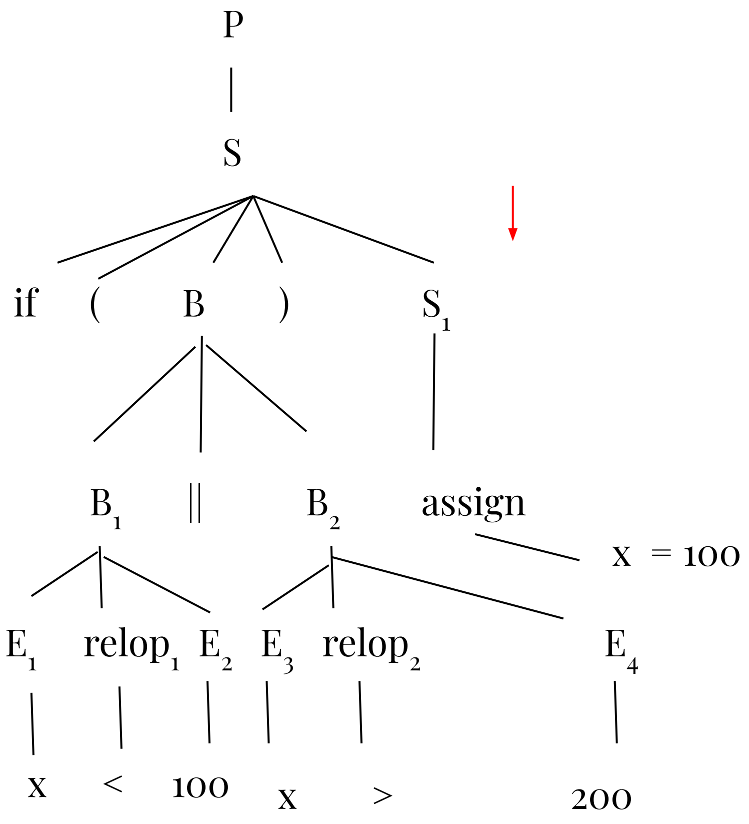

P
S
if
(
B
)
S_1
B_1
||
B_2
assign
x = 100
E_1
$relop_1$
E_2
E_3
$relop_2$
E_4
x
<
100
x
>
200



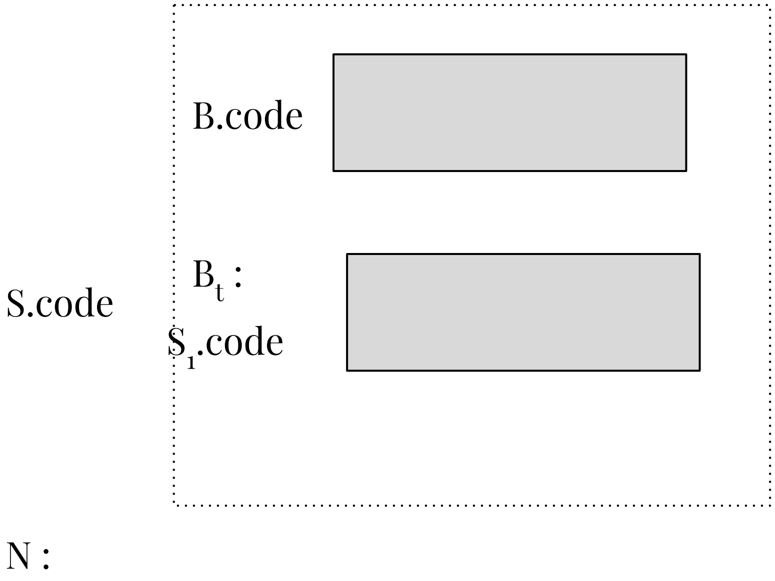
P -> S
{ S.next = newlabel() **= N**
P.code = S.code || label(S.next) }

S.code

N :



S -> if (B) S_1
{ B.true = newlabel() **= B_t**
B.false = S.next **= N**
S.code = B.code || label(B.true) ||
S_1.code}



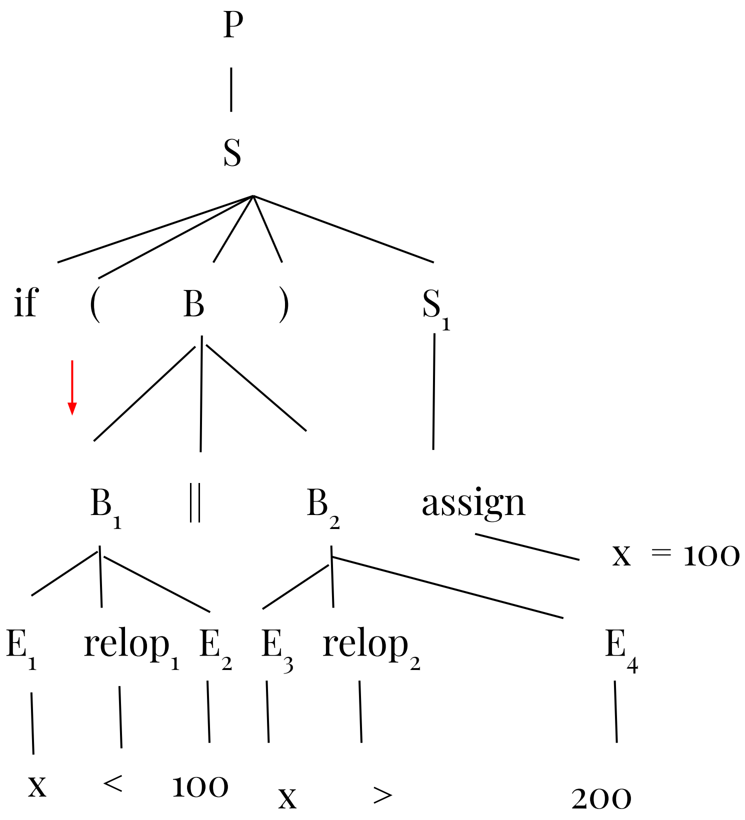
B.code

S.code

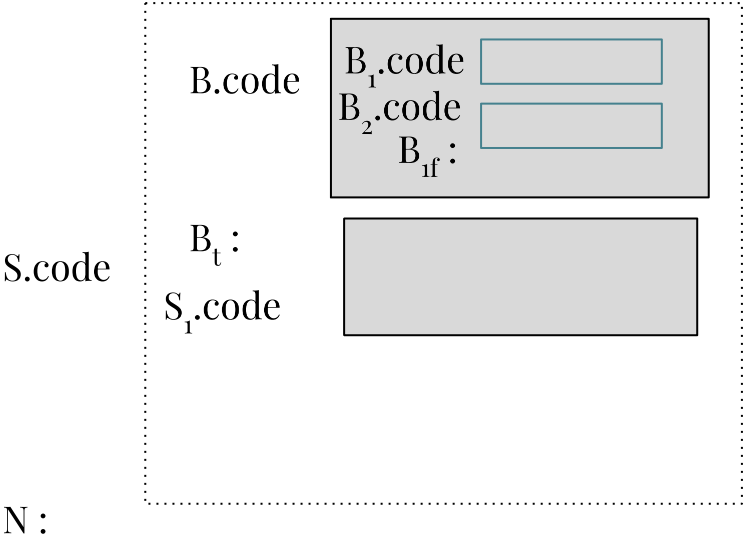
B_t :

S_1.code

N :



B -> B_1 || B_2
{ B_1.true = B.true **= B_t**
B_1.false = newlabel() **= B_{1f}**
B_2.true = B.true **= B_t**
B_2.false = B.false **= B_f**
B.code = B_1.code ||
label(B_1.false) || B_2.code}



B.code

B_1.code

B_2.code

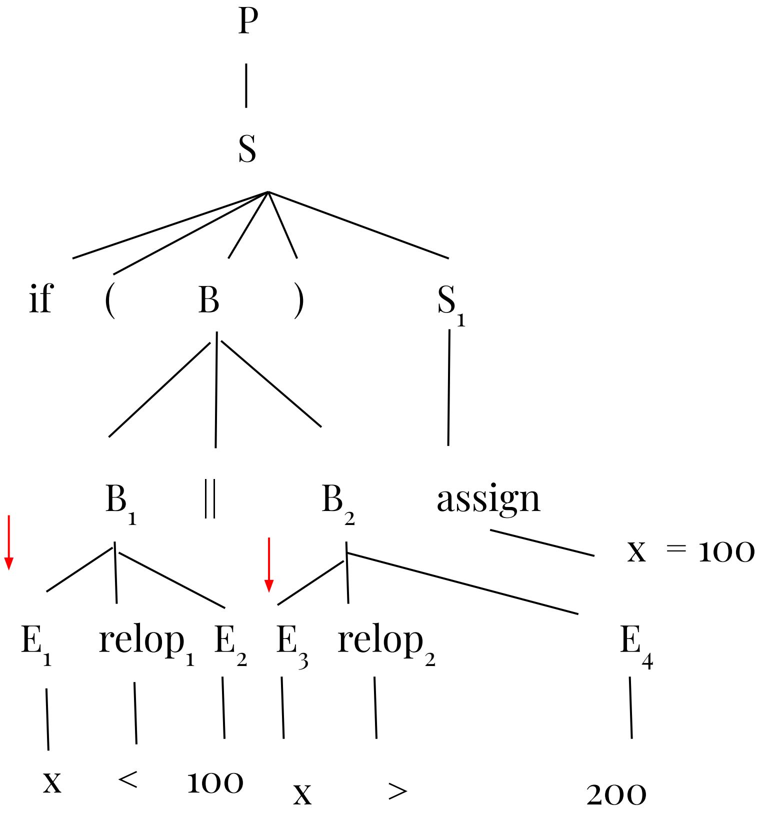
B_{1f} :

S.code

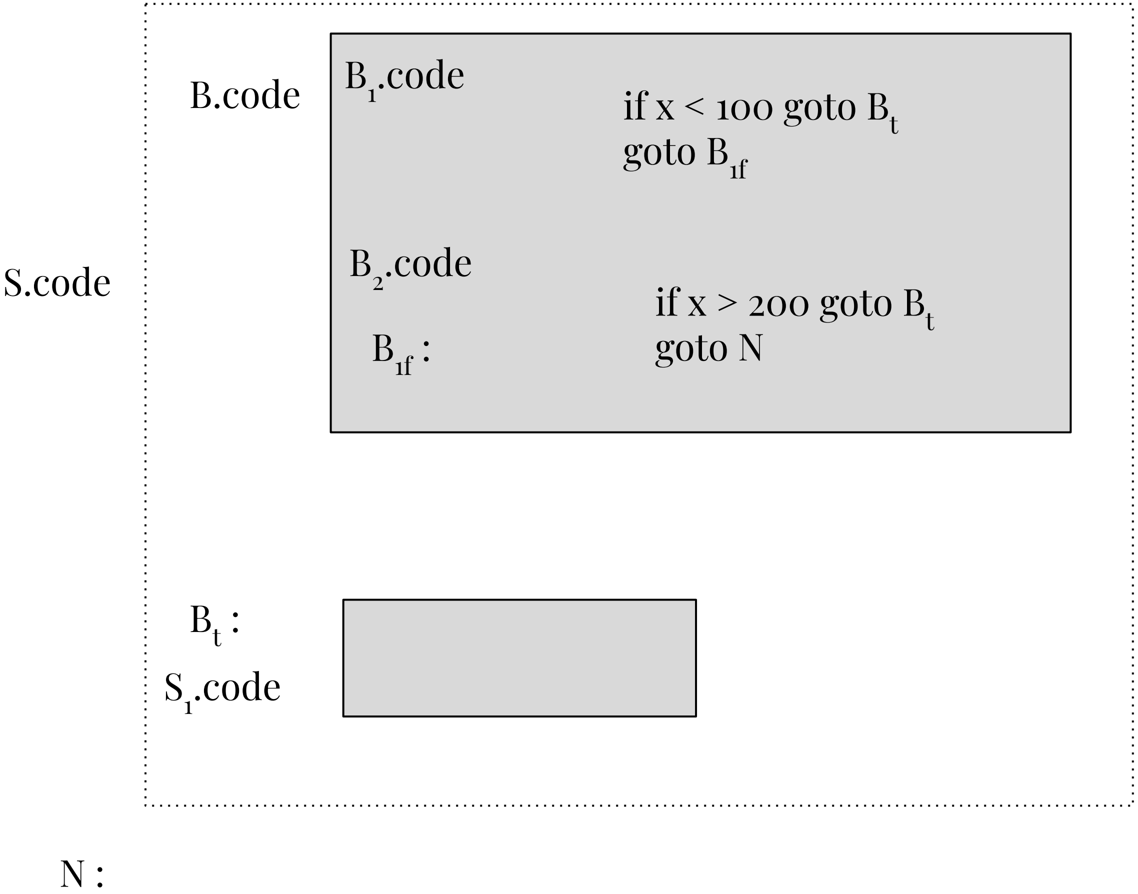
B_t :

S_1.code

N :



$B \to E_1$ relop E_2
{ B.code = E_1.code || E_2.code ||
gen('if' E_1.addr relop E_2.addr
'goto' B.true)}
|| gen('goto' B.false)}



B.code

B_1.code

if x < 100 goto B_t

goto B_{1f}

S.code

B_2.code

B_{1f} :

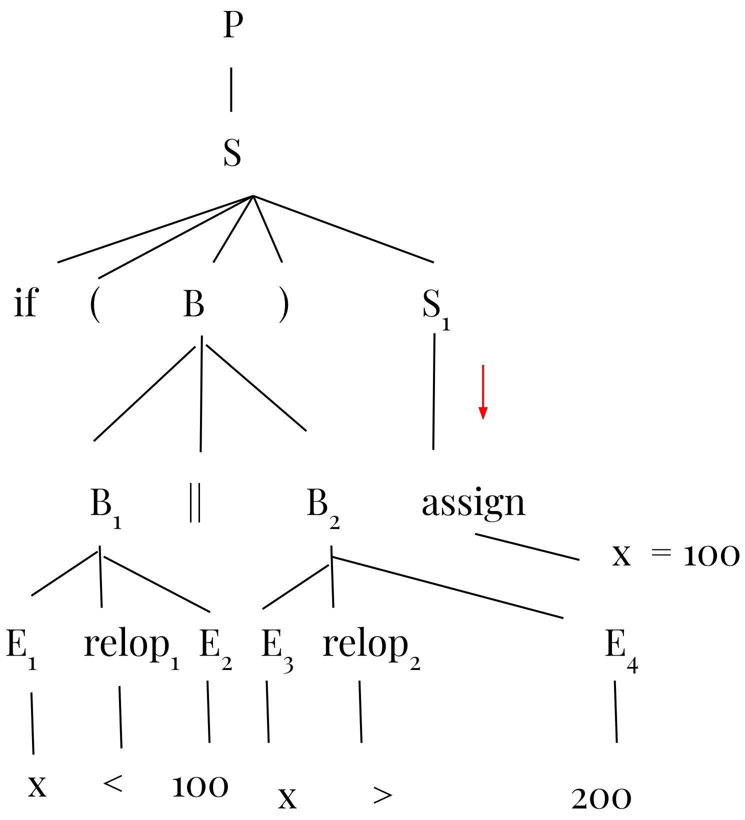
if x > 200 goto B_t

goto N

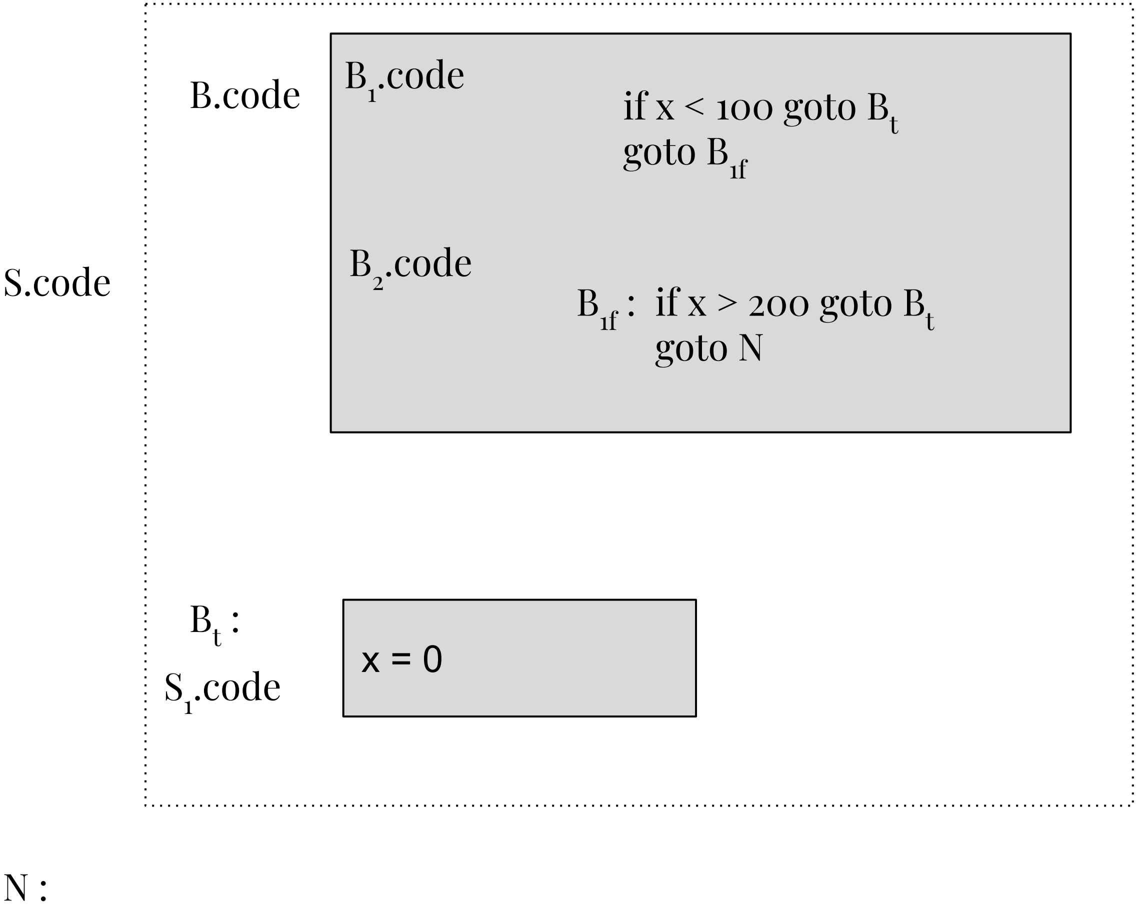
B_t :

S_1.code

N :



S -> assign
{S.code = assign.code}



S.code

B.code

B_1.code

if x < 100 goto B_t

goto B_{1f}

B_2.code

B_{1f} : if x > 200 goto B_t

goto N

B_t :

S_1.code

x = 0

N :

```
       if x < 100 goto $B_t$
       goto $B_{1f}$
$B_{1f}$ : if x > 200 goto $B_t$
       goto N
$B_t$ : x = 0
N :
```

IR Generation using backpatching method

Non-terminal	Semantic rules

B -> B_1|| M B_2	{backpatch(B_1.falselist, M.instr);
	B.truelist = merge(B_1.truelist, B_2.truelist);
	B.falselist = B_2.falselist;}

IR Generation using backpatching method

Non-terminal	Semantic rules

B -> B_1&& M B_2	{backpatch(B_1.truelist, M.instr);
	B.truelist = B_2.truelist;
	B.falselist = merge(B_1.falselist, B_2.falselist);}

IR Generation using backpatching method

Non-terminal	Semantic rules

B -> ! B_1	{B.truelist = B_1.falselist;
	B.falselist = B_1.truelist;}

IR Generation using backpatching method

Non-terminal	Semantic rules

B→(B_1)	{B.truelist=B_1.truelist;
	B.falselist=B_1.falselist;}

IR Generation using backpatching method

Non-terminal	Semantic rules

B→E_1 rel E_2	{B.truelist = makelist(nextinstr);
	B.falselist = makelist(nextinstr+ 1);
	gen(if E_1.addr rel.op E_2.addr goto _);
	gen(goto _);}

IR Generation using backpatching method

Non-terminal	Semantic rules

B→true	{B.truelist = makelist(nextinstr);
	gen(goto _);}

IR Generation using backpatching method

Non-terminal	Semantic rules

B→false	{B.falselist = makelist(nextinstr);
	gen(goto _);}

IR Generation using backpatching method

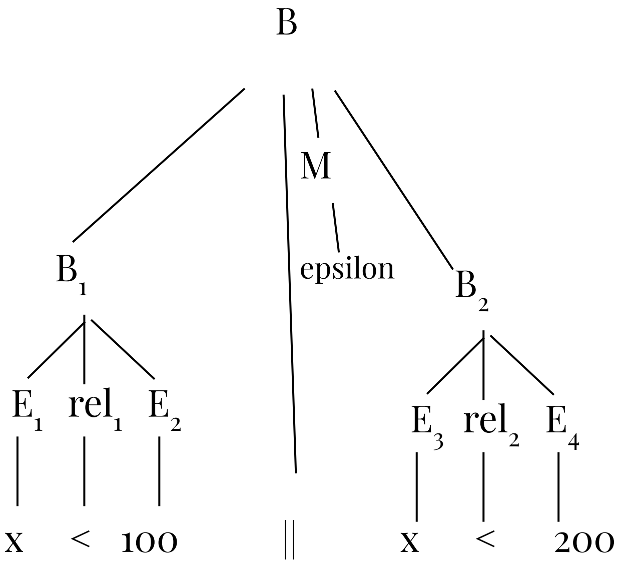
Non-terminal	Semantic rules
M -> epsilon	{M.instr = nextstr;}

Example

Consider the following boolean expression

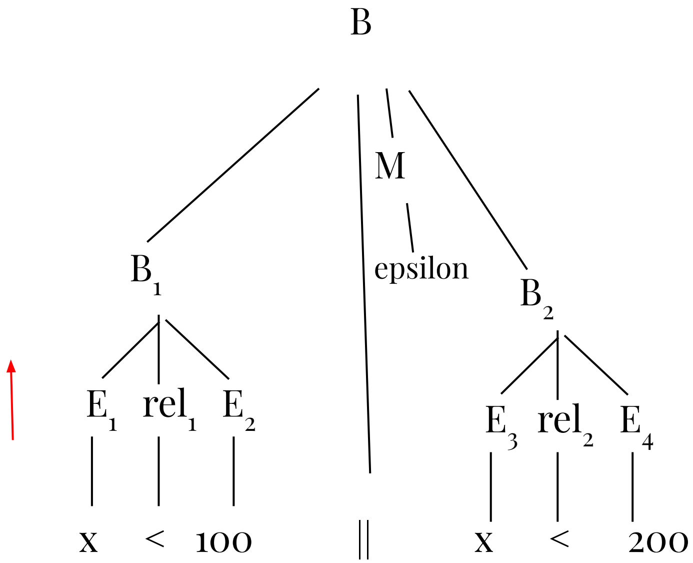
- x < 100 || x < 200

Parse Tree



IR generation

B_1 -> E_1 rel_1 E_2
{B_1.truelist = makelist(nextinstr) **= {100}**;
B_1.falselist = makelist(nextinstr+ 1) **= {101};**
gen(if E_1.addr rel.op E_2.addr goto _);
gen(goto _);}

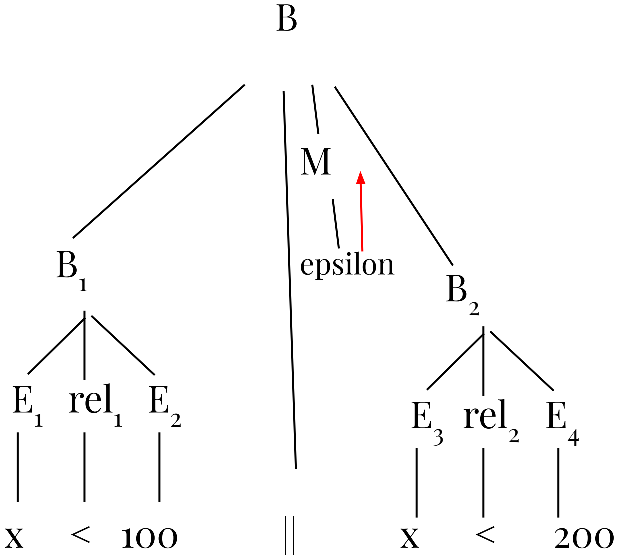


100 : if x < 100 goto _

101 : goto _

IR generation

M -> epsilon
{M.instr = nextinstr **= {102}**; }

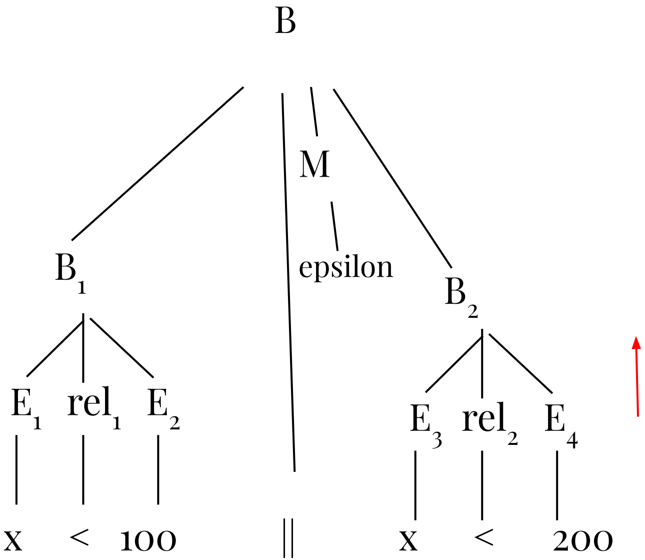


100 : if x < 100 goto _

101 : goto _

IR generation

B_2 -> E_3 rel_2 E_4
{B_2.truelist = makelist(nextinstr) **= {102}**;
B_2.falselist = makelist(nextinstr+ 1) **= {103};**
gen(if E_3.addr rel.op E_4.addr goto _);
gen(goto _);}



100 : if x < 100 goto _

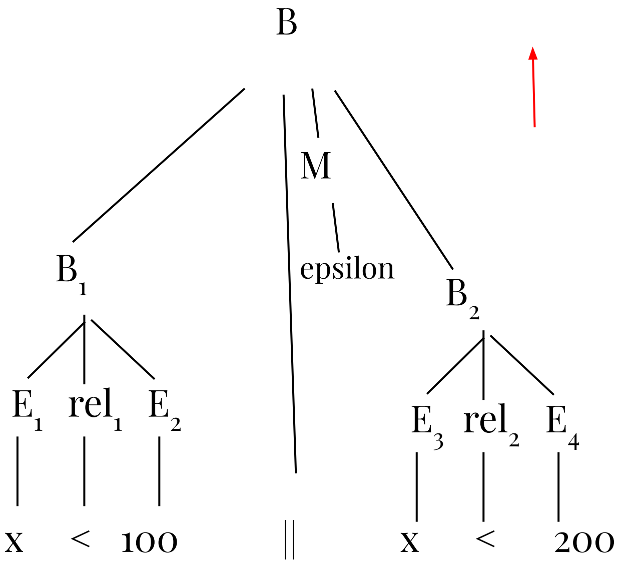
101 : goto _

102 : if x < 200 goto _

103 : goto _

IR generation

B -> B_1 || M B_2
{backpatch(B_1.falselist, M.instr);
B.truelist = merge(B_1.truelist, B_2.truelist) **= {100, 102}**;
B.falselist = B_2.falselist **= {103}**;}



100 : if x < 100 goto _

101 : goto 102

102 : if x < 100 goto _

103 : goto _

IR Generation using backpatching method

Non-terminal	Semantic rules

S→if(B)M S_1	{backpatch(B.truelist, M.instr);
	S.nextlist = merge(B.falselist, S_1.nextlist);}

IR Generation using backpatching method

Non-terminal	Semantic rules

S→if(B) M_1 S_1N else M_2 S_2	{backpatch(B.truelist, M_1.instr);
	backpatch(B.falselist, M_2.instr);
	temp = merge(S_1.nextlist, N.nextlist);
	S.nextlist = merge(temp, S_2.nextlist);}

IR Generation using backpatching method

Non-terminal	Semantic rules

S→while M_1(B)$M_2$$S_1$	{backpatch(S_1.nextlist, M_1.instr);
	backpatch(B.truelist, M_2.instr);
	S.nextlist = B.falselist;
	gen(goto M_1.instr)}

IR Generation using backpatching method

Non-terminal	Semantic rules
S→{L}	{S.nextlist = L.nextlist;}

IR Generation using backpatching method

Non-terminal	Semantic rules
S→A	{S.nextlist = null;}

IR Generation using backpatching method

Non-terminal	Semantic rules
M -> epsilon	{M.instr = nextinstr;}

IR Generation using backpatching method

Non-terminal	Semantic rules

N -> epsilon	{N.nextlist=makelist(nextinstr);
	gen(goto _);}

IR Generation using backpatching method

Non-terminal	Semantic rules

L→L_1M S	{backpatch(L_1.nextlist,M.instr);
	L.nextlist = S.nextlist;}

IR Generation using backpatching method

Non-terminal	Semantic rules
L→S	{L.nextlist = S.nextlist;}

Input string

```
if(a > b || b > c)
{
   b = 5;
}
else
{
   b = 6;
}
a = 7;
```

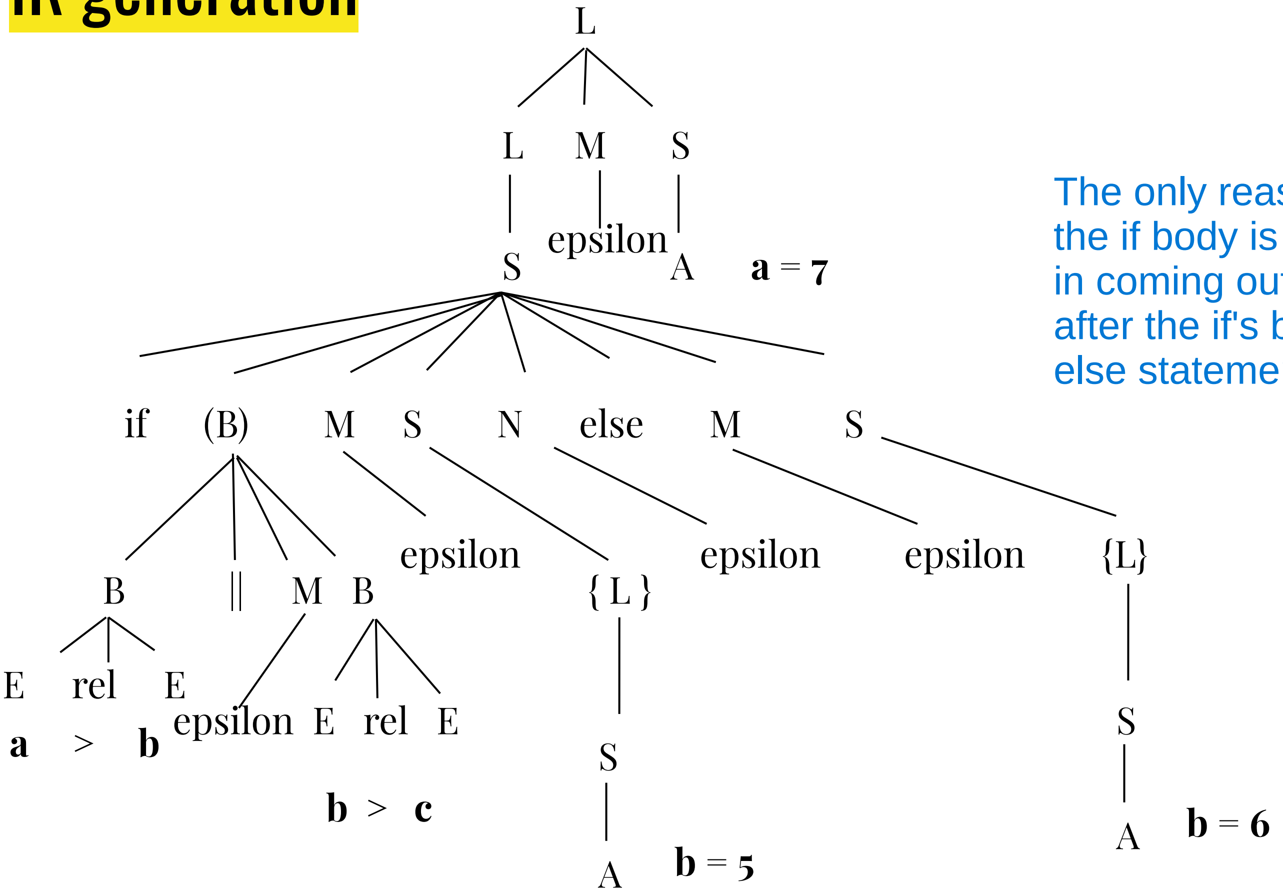
Boolean expressions have TRUE & FALSE lists.
Statements have NEXT lists.

We are using backpatching operations to fill the unknown jump targets using known information.

We do bottom up parsing.

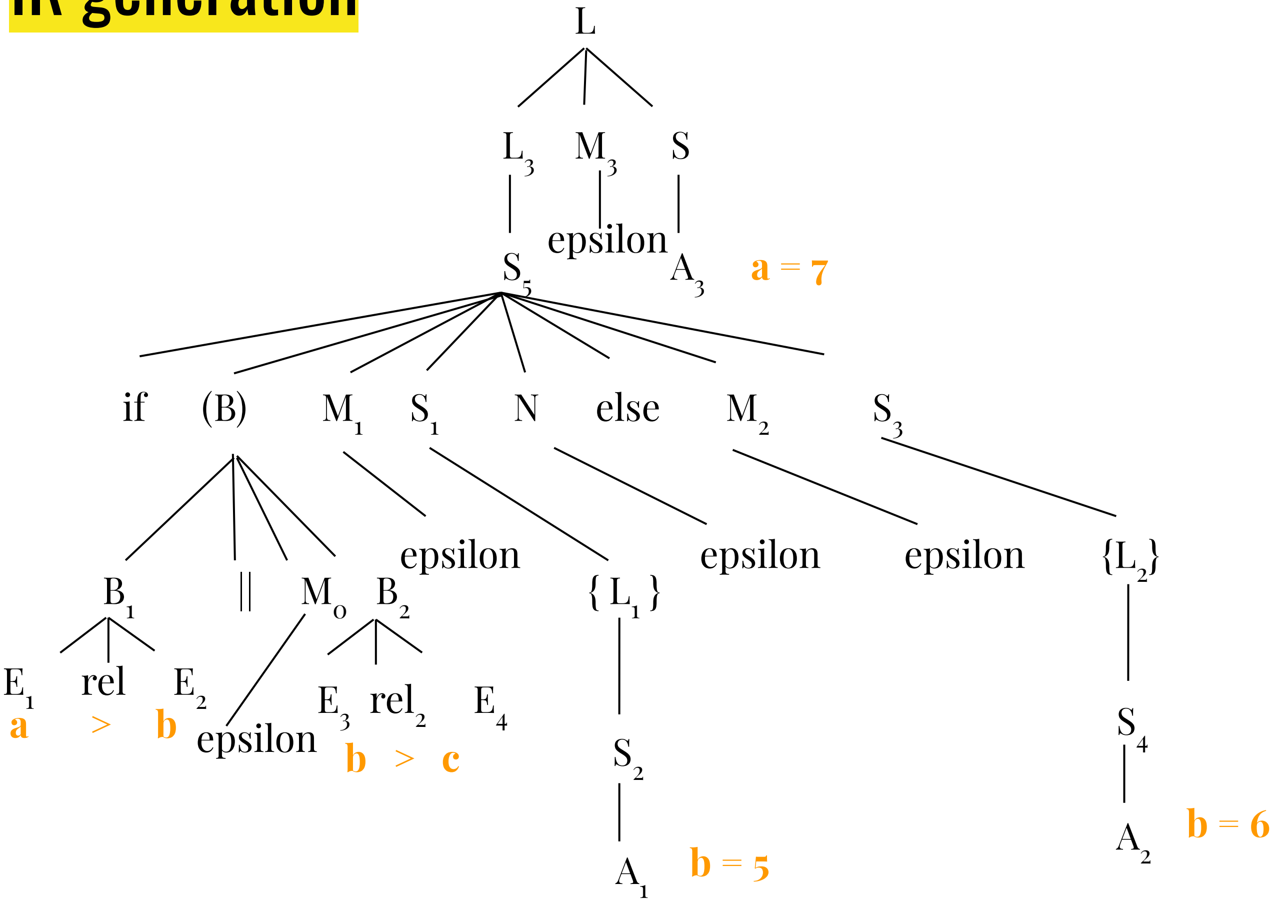
We start with left most & bottom & parse the things & group up &move a bit up if possible, etc..

IR generation



The only reason for putting the "N" after the if body is to create a GOTO that helps in coming out of the if statement directly after the if's body execution by skipping the else statement following it.

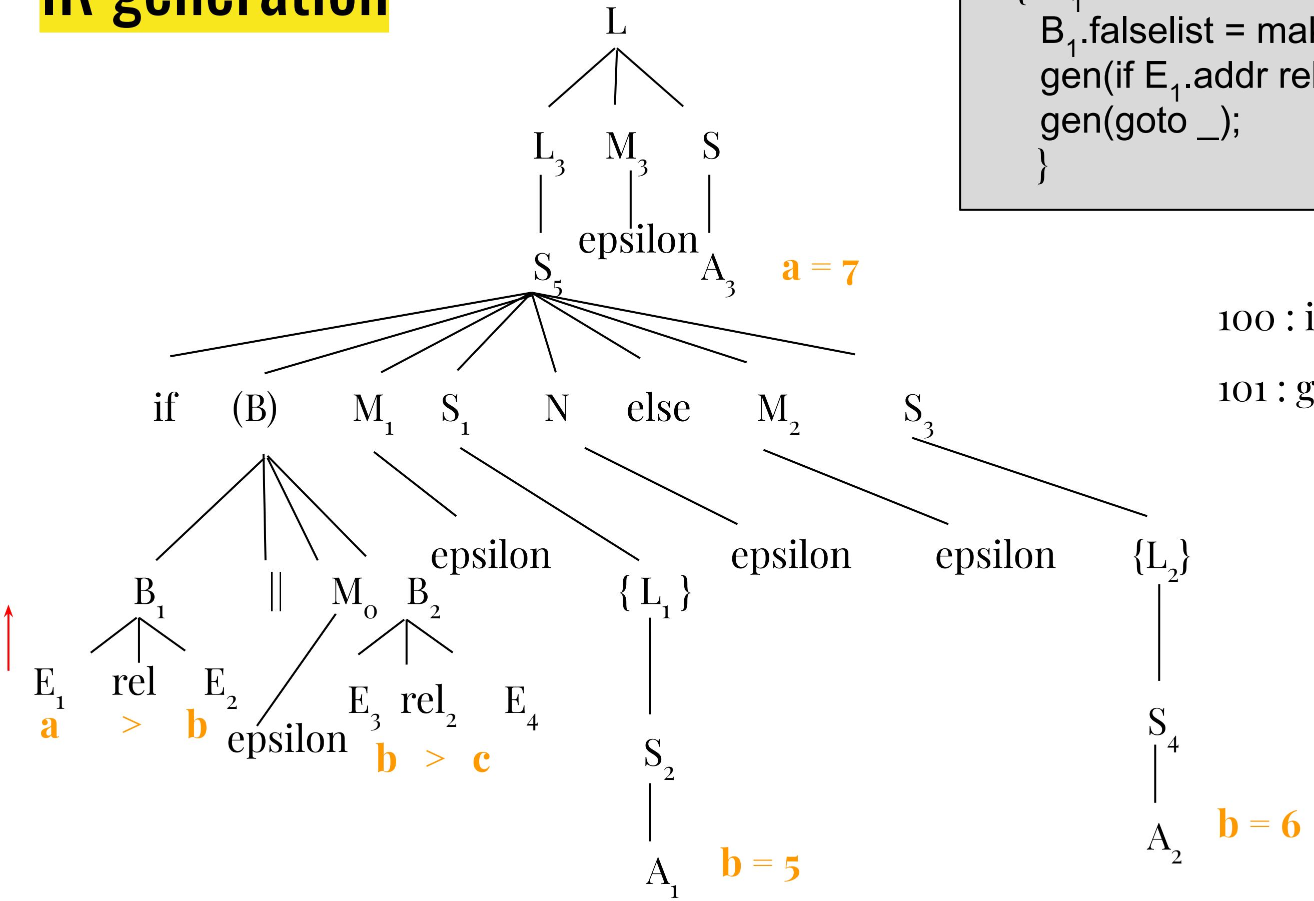
IR generation



Note that all Ls, all Ms,
all Ss are indiviually same.

But to make it precise we are indexing them like:
S1, S2, S3,...
M1, M2, M3...

IR generation



$B_1 \twoheadrightarrow E_1$ rel E_2

{ B_1.truelist = makelist(nextinstr); **= {100}**

B_1.falselist = makelist(nextinstr+ 1); **= {101}**

gen(if E_1.addr rel.op E_2.addr goto _);

gen(goto _);

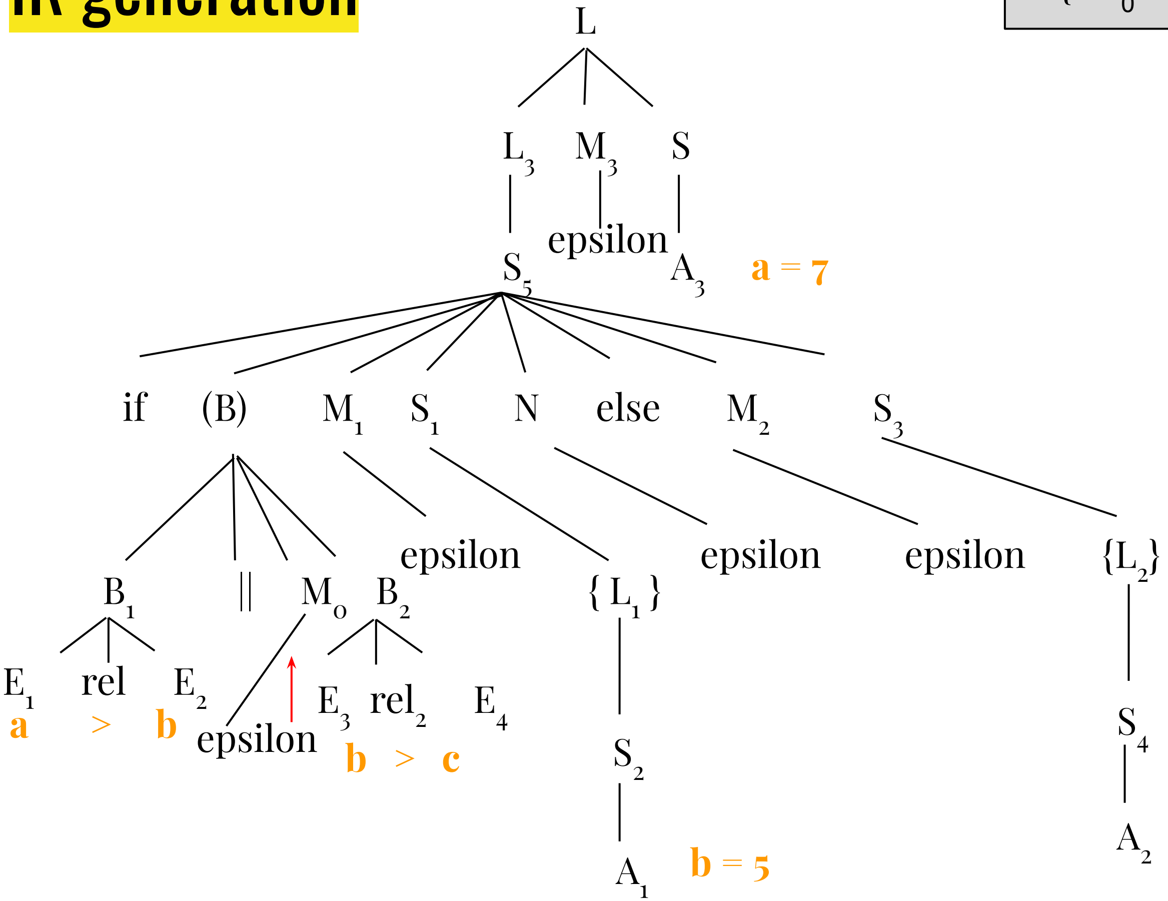
}

100 : if a > b goto _

101 : goto _

IR generation

$M_0 \twoheadrightarrow$ epsilon
{ M_0.instr = nextinstr = **{102}** }



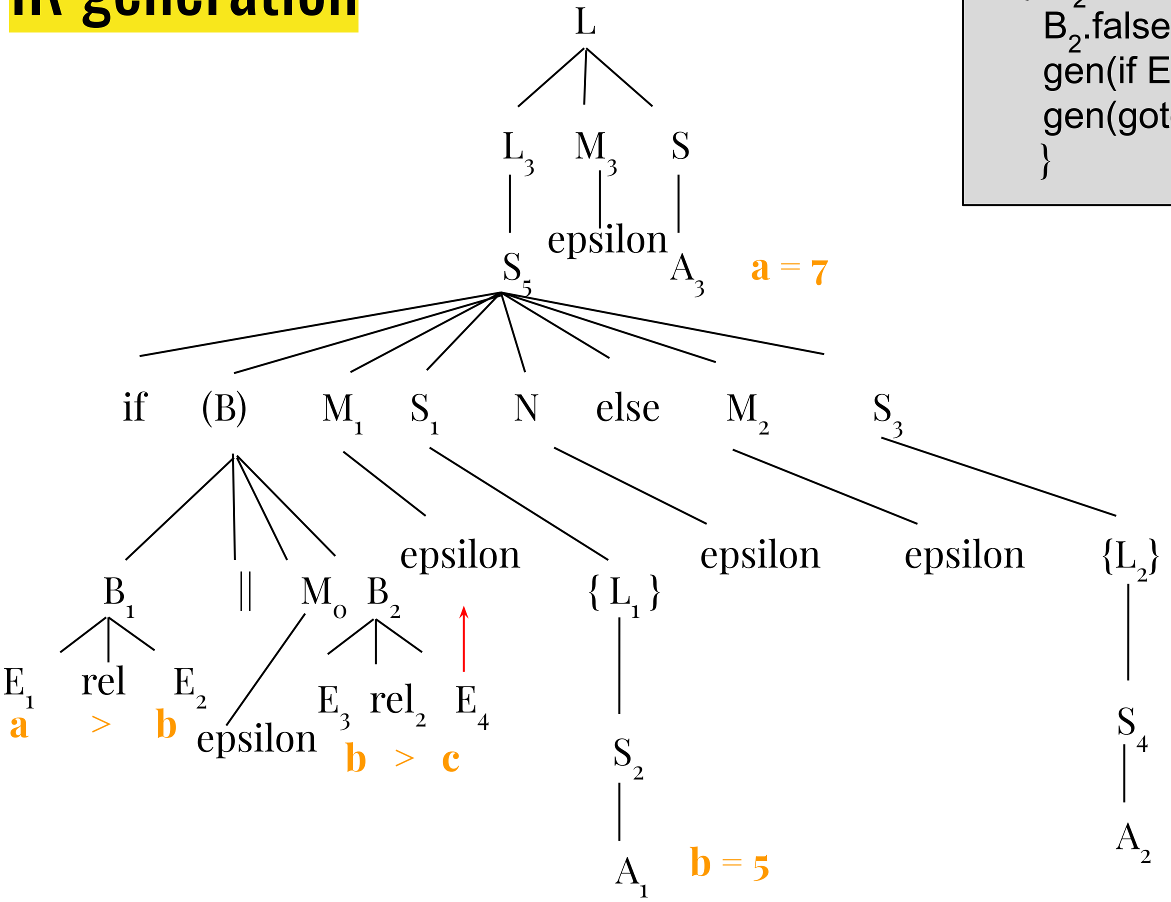
100 : if a > b goto _

101 : goto _

IR generation

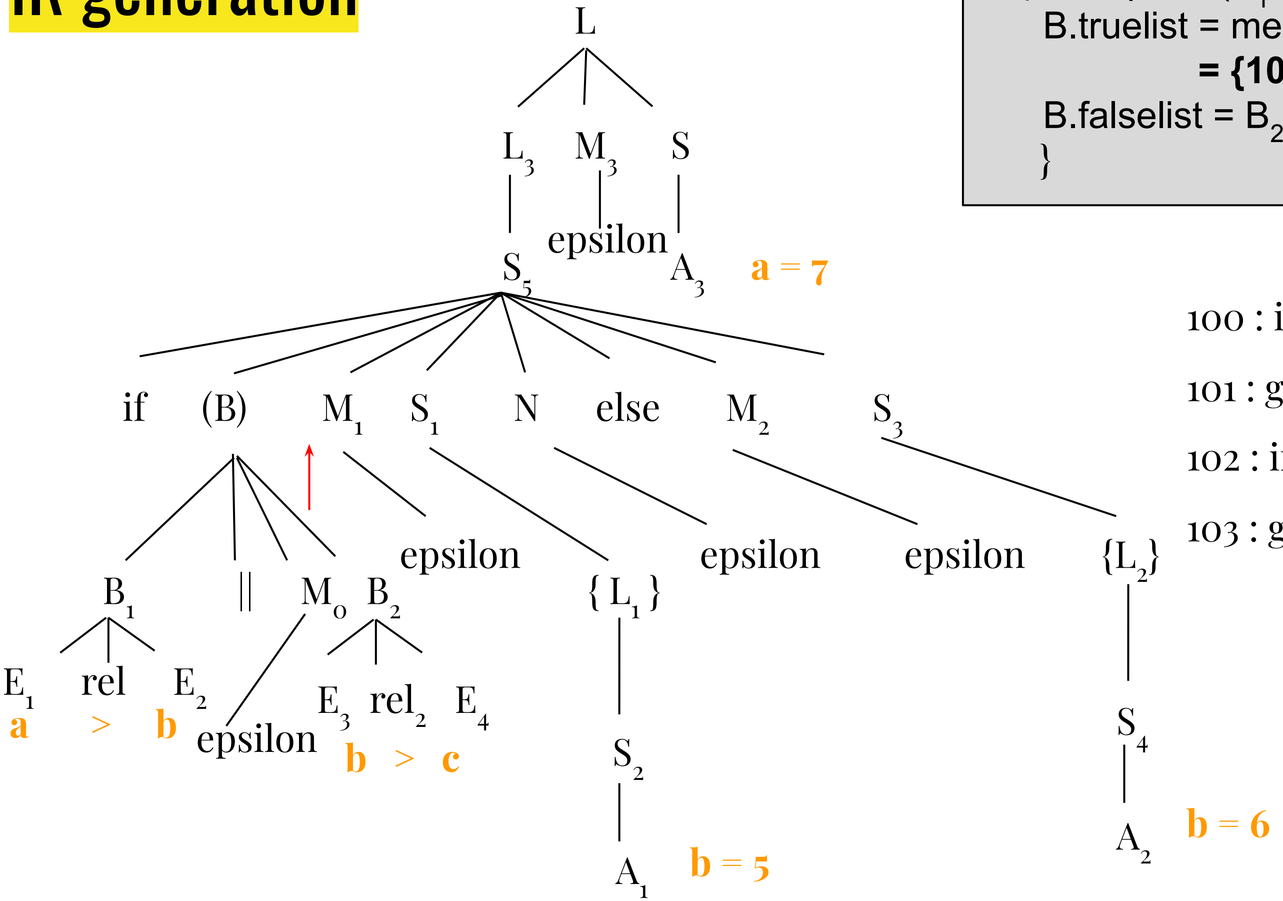
$B_2 \twoheadrightarrow E_3$ rel E_4

```
{ B_2.truelist = makelist(nextinstr); = {102}
  B_2.falselist = makelist(nextinstr+ 1); = {103}
  gen(if E_3.addr rel.op E_4.addr goto _);
  gen(goto _);
}
```



```
100 : if a > b goto _

101 : goto _

102 : if b > c goto _

103 : goto _
```

IR generation



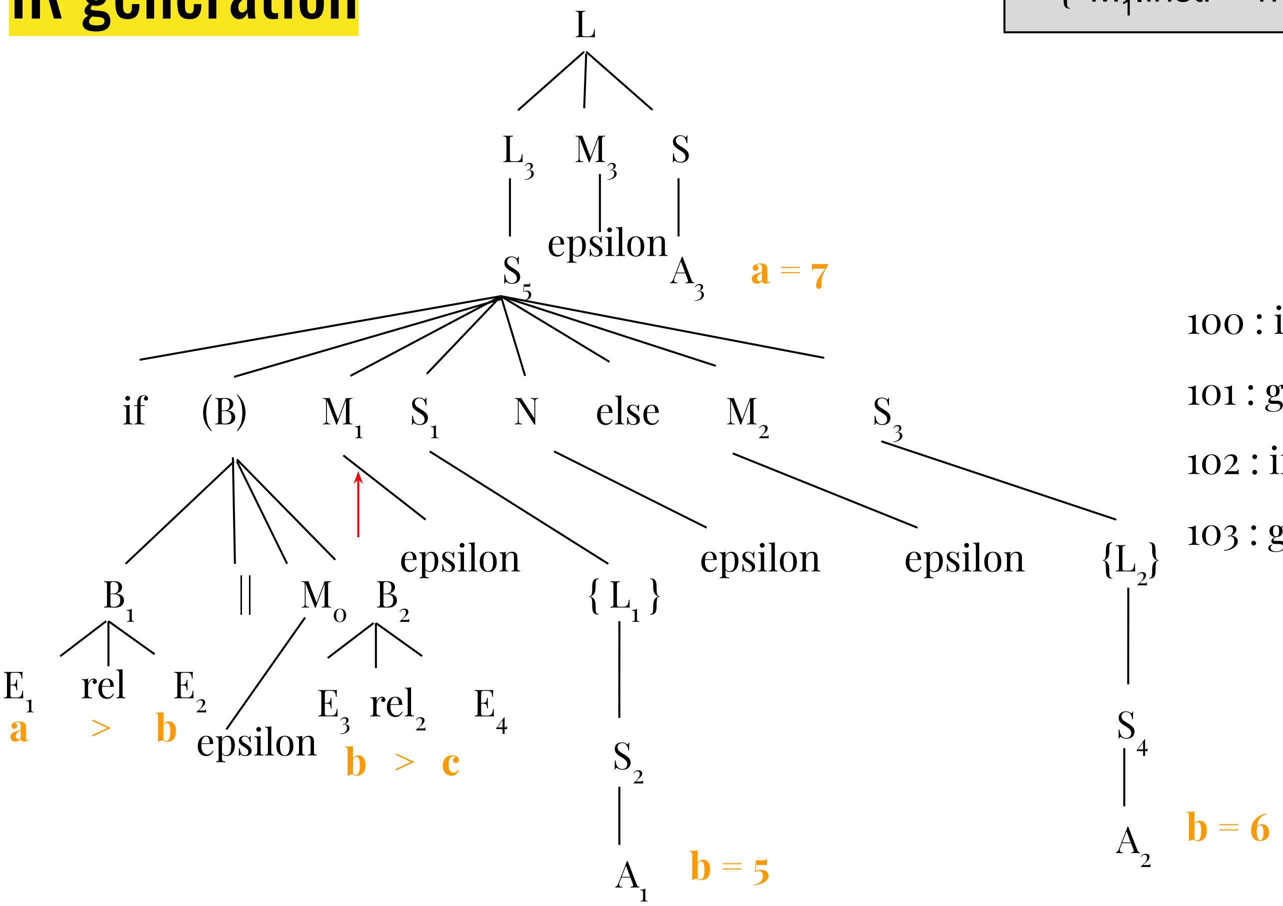
$B \twoheadrightarrow B_1 \mid\mid M_0\ B_2$
{ backpatch(B_1.falselist, M_0.instr);
B.truelist = merge(B_1.truelist, B_2.truelist)
= {100, 102};
B.falselist = B_2.falselist **= {103}**;
}

```
100 : if a > b goto _
101 : goto 102
102 : if b > c goto _
103 : goto _
```

IR generation

$M_1 \twoheadrightarrow$ epsilon

{ M_1.instr = nextinstr = **{104}** }



100 : if a > b goto _

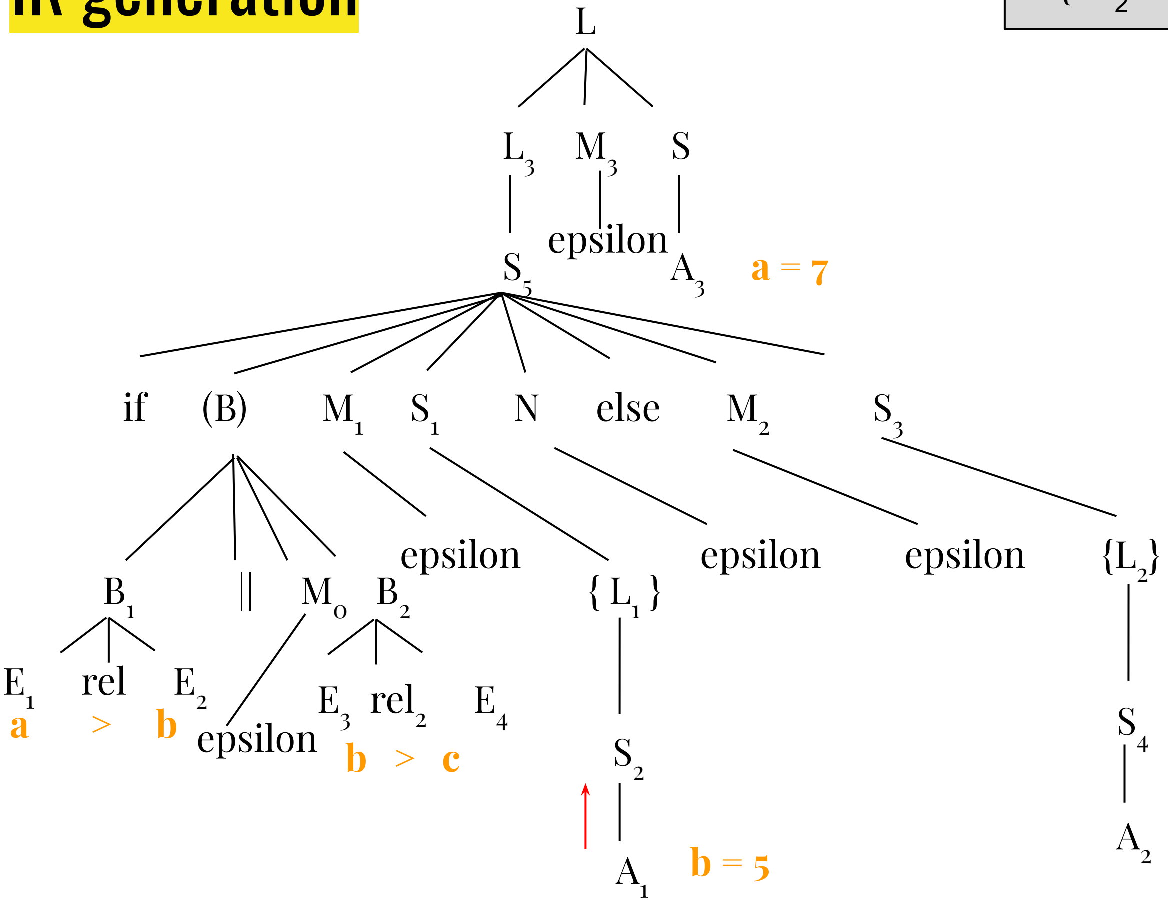
101 : goto 102

102 : if b > c goto _

103 : goto _

IR generation

$S_2 \twoheadrightarrow A_1$
{ S_2.nextlist = null; }



100 : if a > b goto _

101 : goto 102

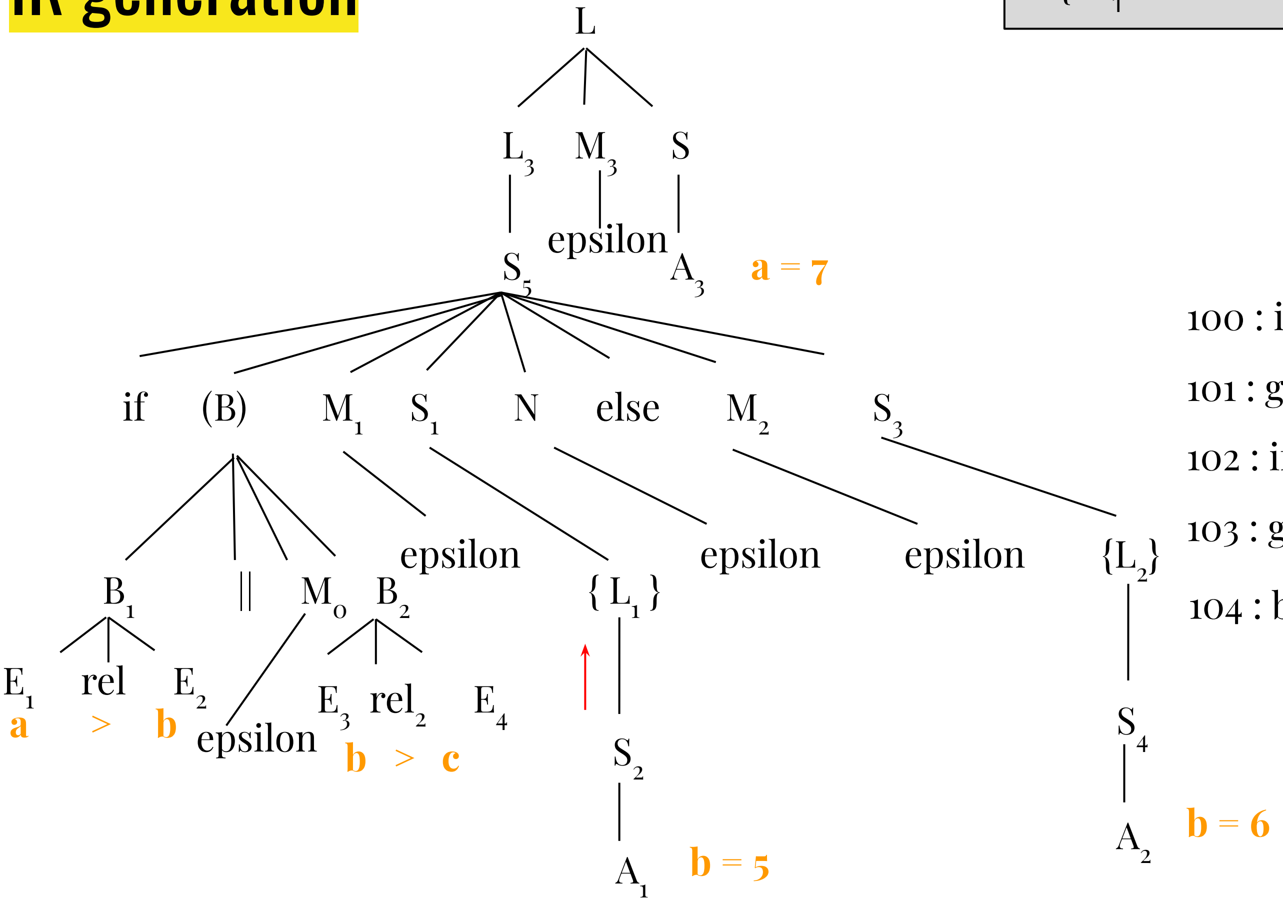
102 : if b > c goto _

103 : goto _

104 : b = 5

IR generation

$L_1 \twoheadrightarrow S_2$
{ L_1.nextlist = S_2.nextlist; }



100 : if a > b goto _

101 : goto 102

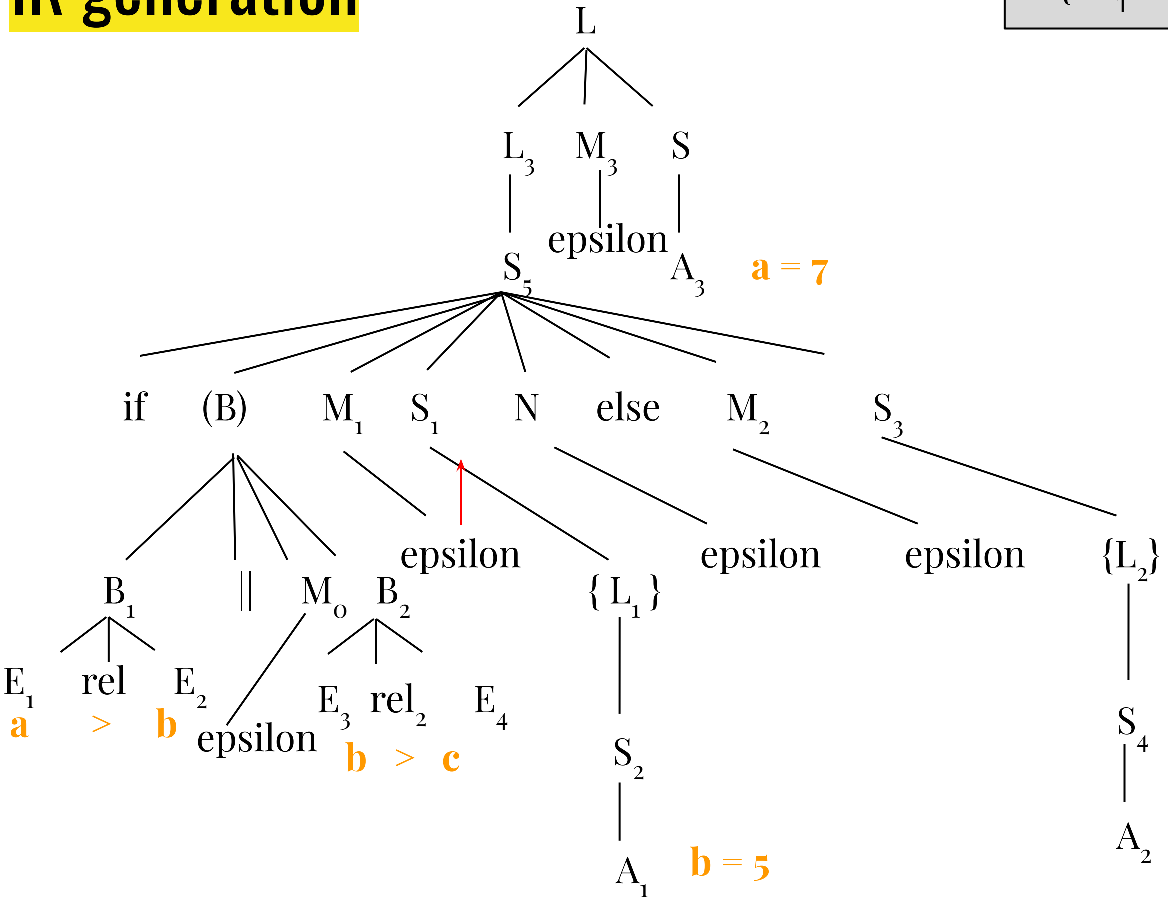
102 : if b > c goto _

103 : goto _

104 : b = 5

IR generation

$S_1 \twoheadrightarrow L_1$
{ S_1.nextlist = L_1.nextlist; }



100 : if a > b goto _

101 : goto 102

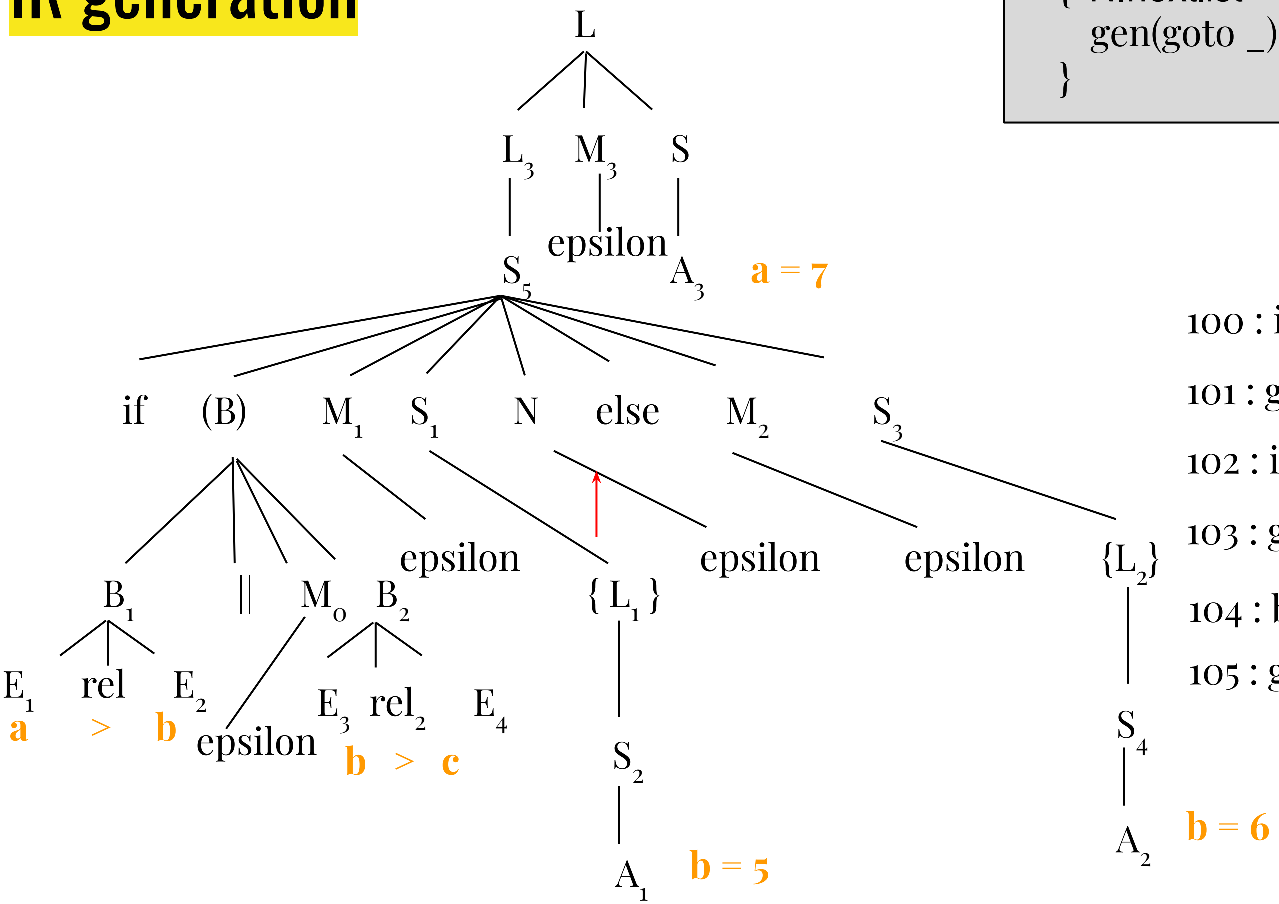
102 : if b > c goto _

103 : goto _

104 : b = 5

IR generation

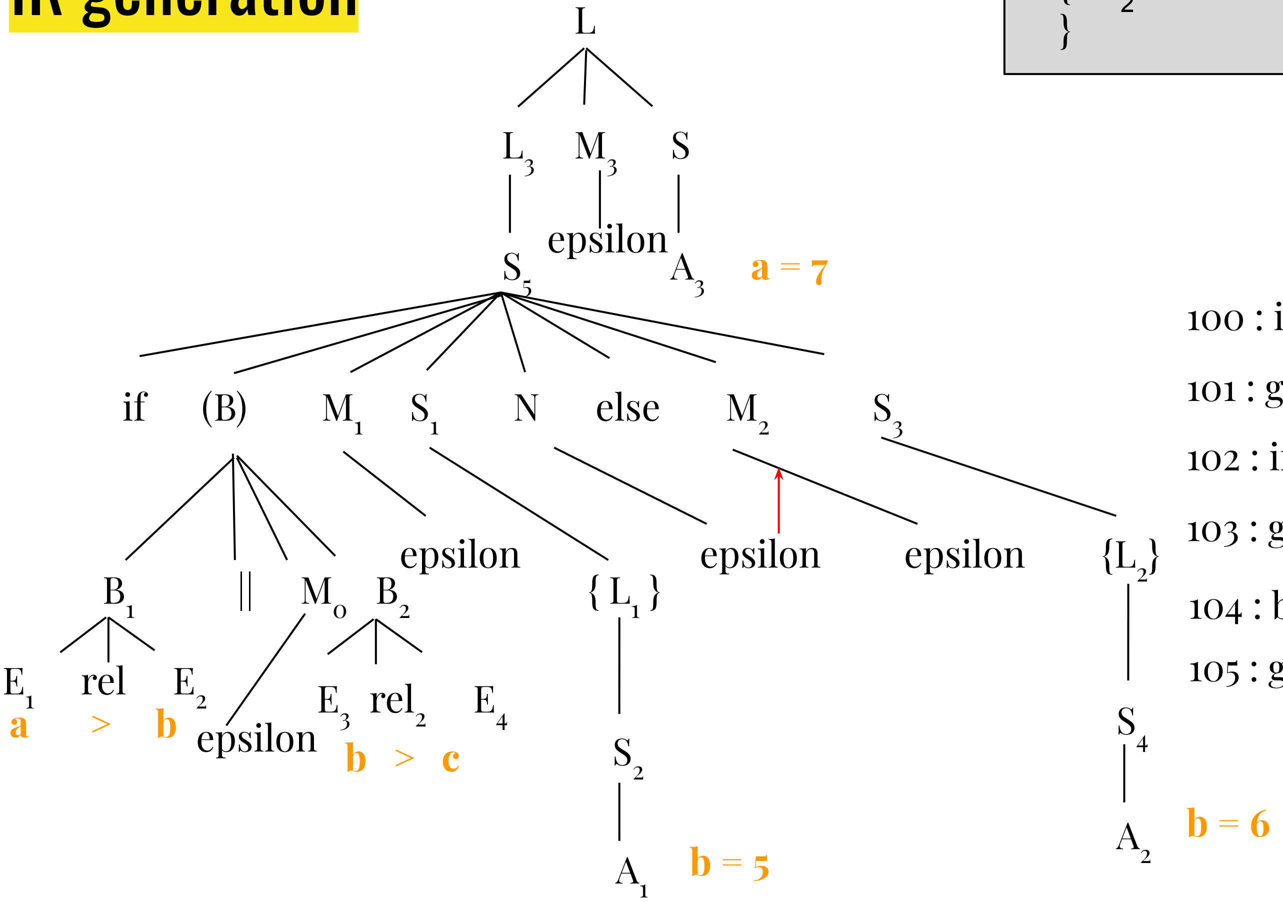
```
N ↠ epsilon
  { N.nextlist = makelist(nextinstr) = 105;
    gen(goto _);
  }
```



```
100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

103 : goto _

104 : b = 5

105 : goto _
```

IR generation

M_2 ↠ epsilon
{ M_2.instr = nextinstr = **106**;
}



100 : if a > b goto _

101 : goto _

102 : if b > c goto _

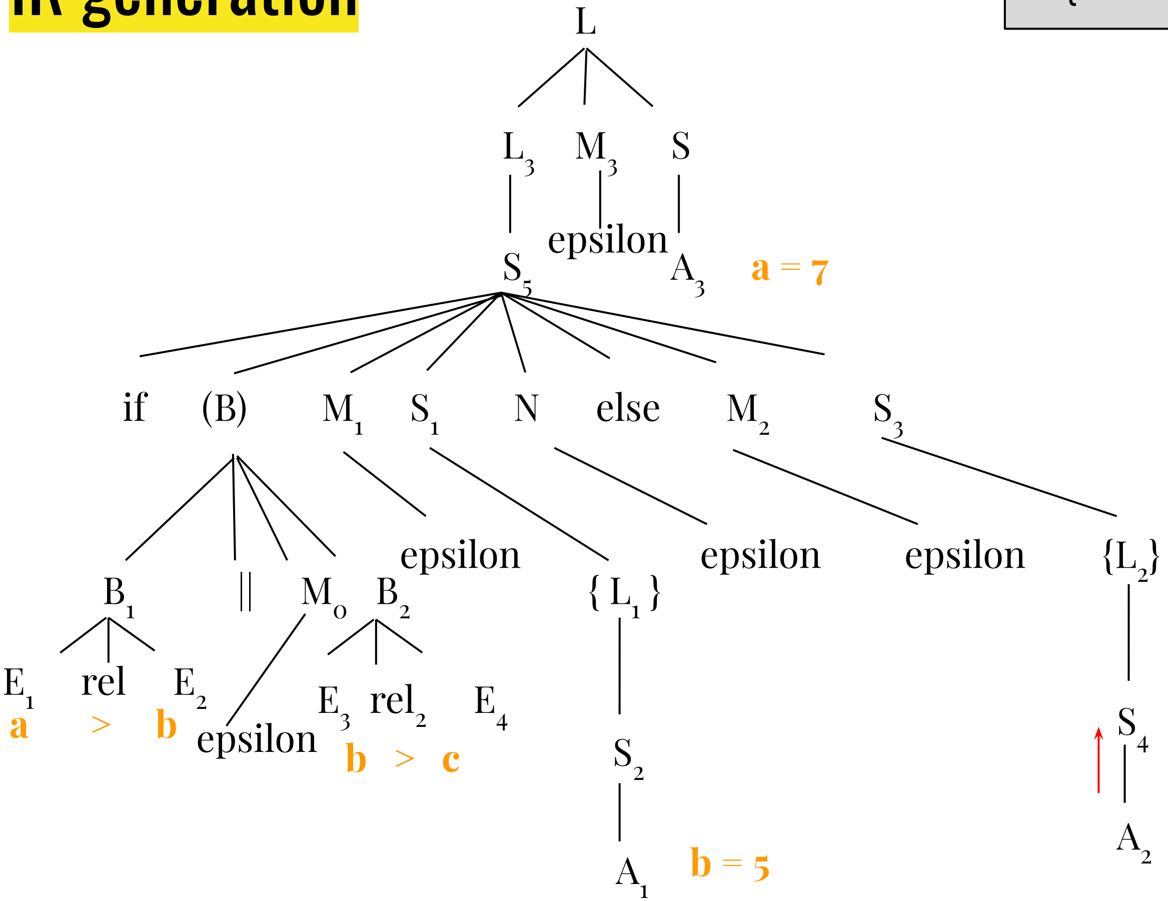
103 : goto _

104 : b = 5

105 : goto _

IR generation

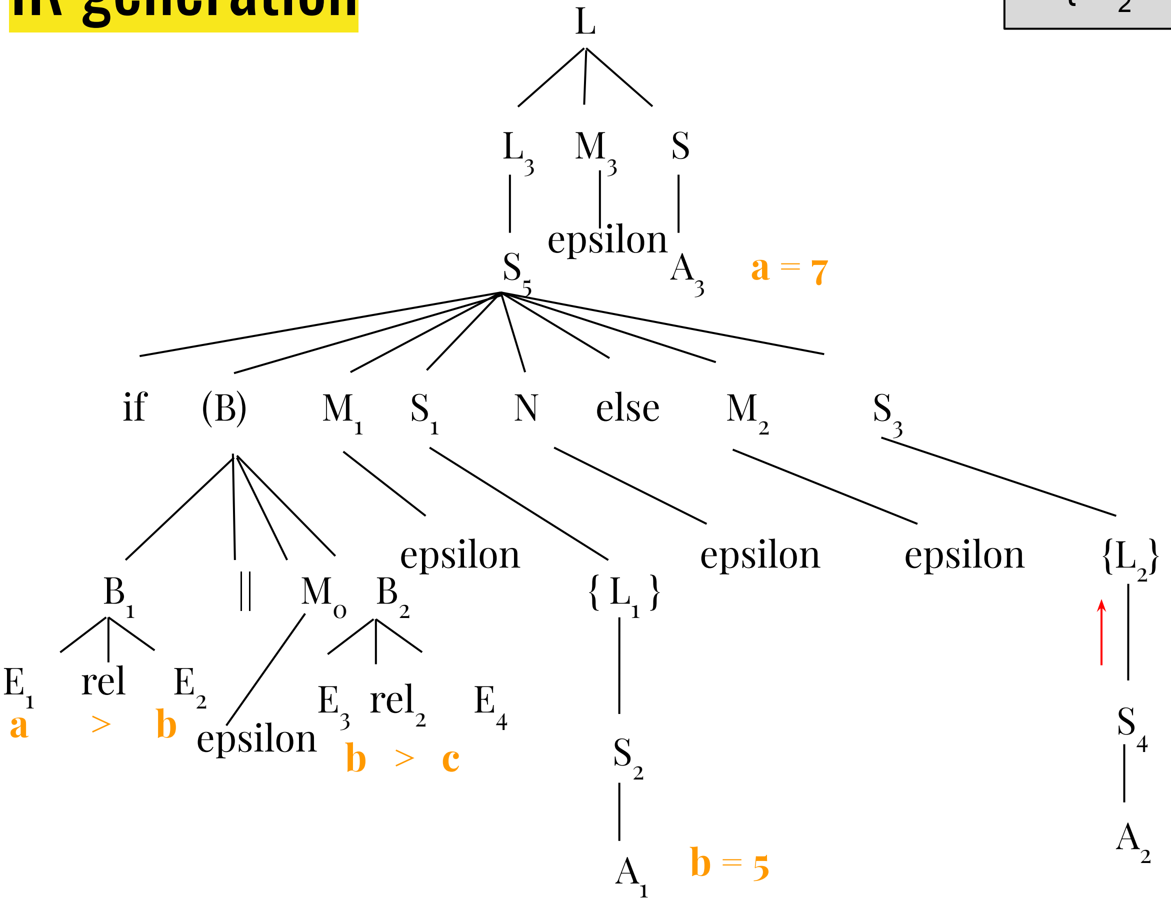
S_4 -> A_2
{ S4.nextlist = null; }



```
100 : if a > b goto _
101 : goto 102
102 : if b > c goto _
103 : goto _
104 : b = 5
105 : goto _
106 : b = 6
```

IR generation

L_2 -> S_4
{ L_2.nextlist = S_4.nextlist = **null** ; }

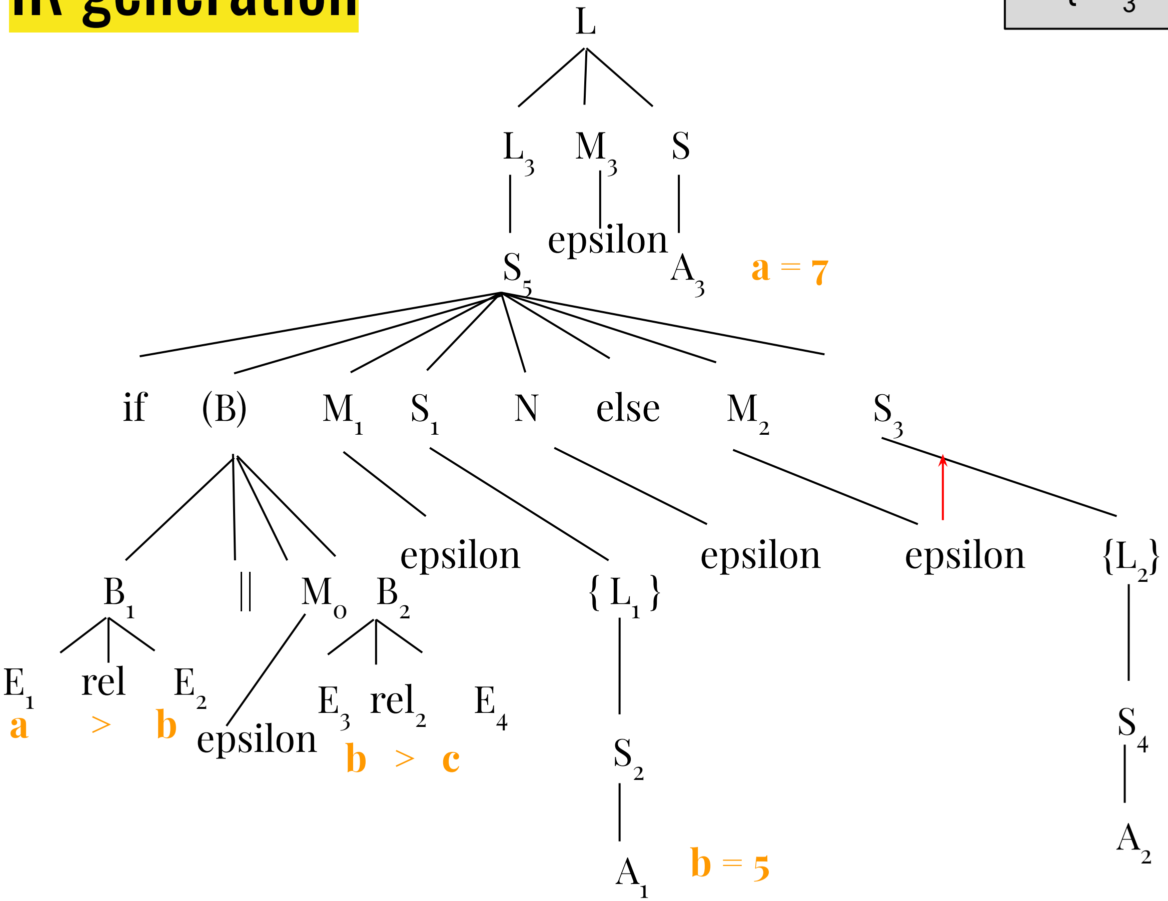


```
100 : if a > b goto _
101 : goto 102
102 : if b > c goto _
103 : goto _
104 : b = 5
105 : goto _
106 : b = 6
```

b = 6

IR generation

S_3 -> { L_2}
{ S_3.nextlist = L_2.nextlist **= null** ; }



100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

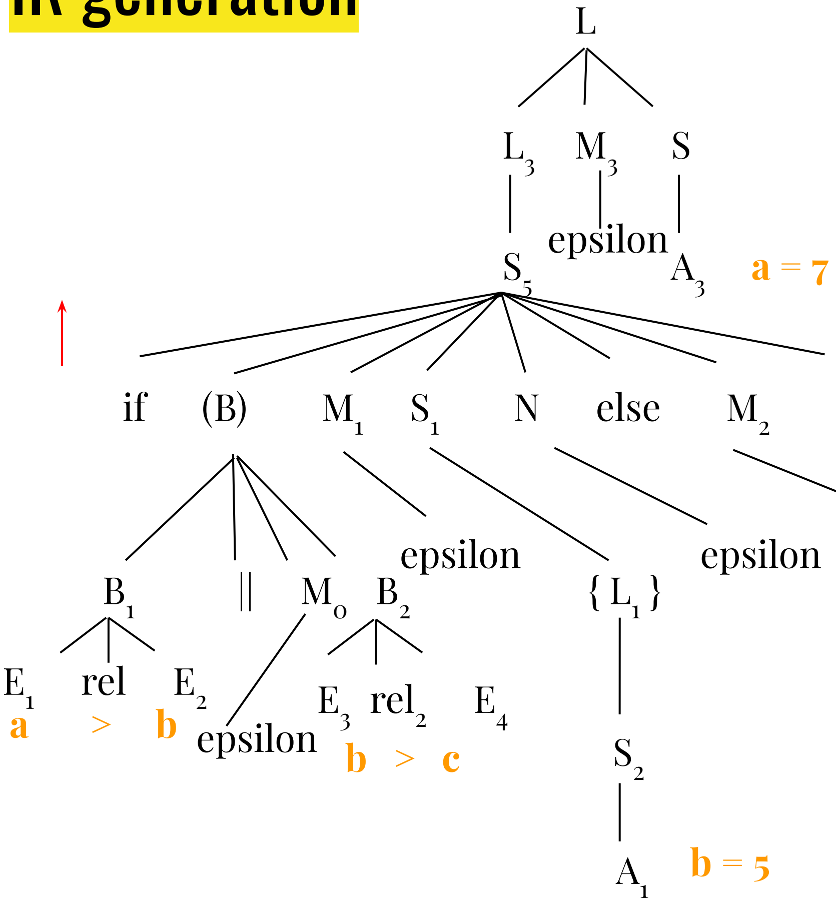
103 : goto _

104 : b = 5

105 : goto _

106 : b = 6

IR generation



```
S5 -> if (B) M1 S1 N else M2 S3
  { backpatch(B.truelist, M1.instr);
    backpatch(B.falselist, M2.instr);
    temp = merge(S1.nextlist, N.nextlist);
    S5.nextlist = merge(temp, S2.nextlist)
    = {105};
}
```

100 : if a > b goto 104

101 : goto 102

102 : if b > c goto 104

103 : goto 106

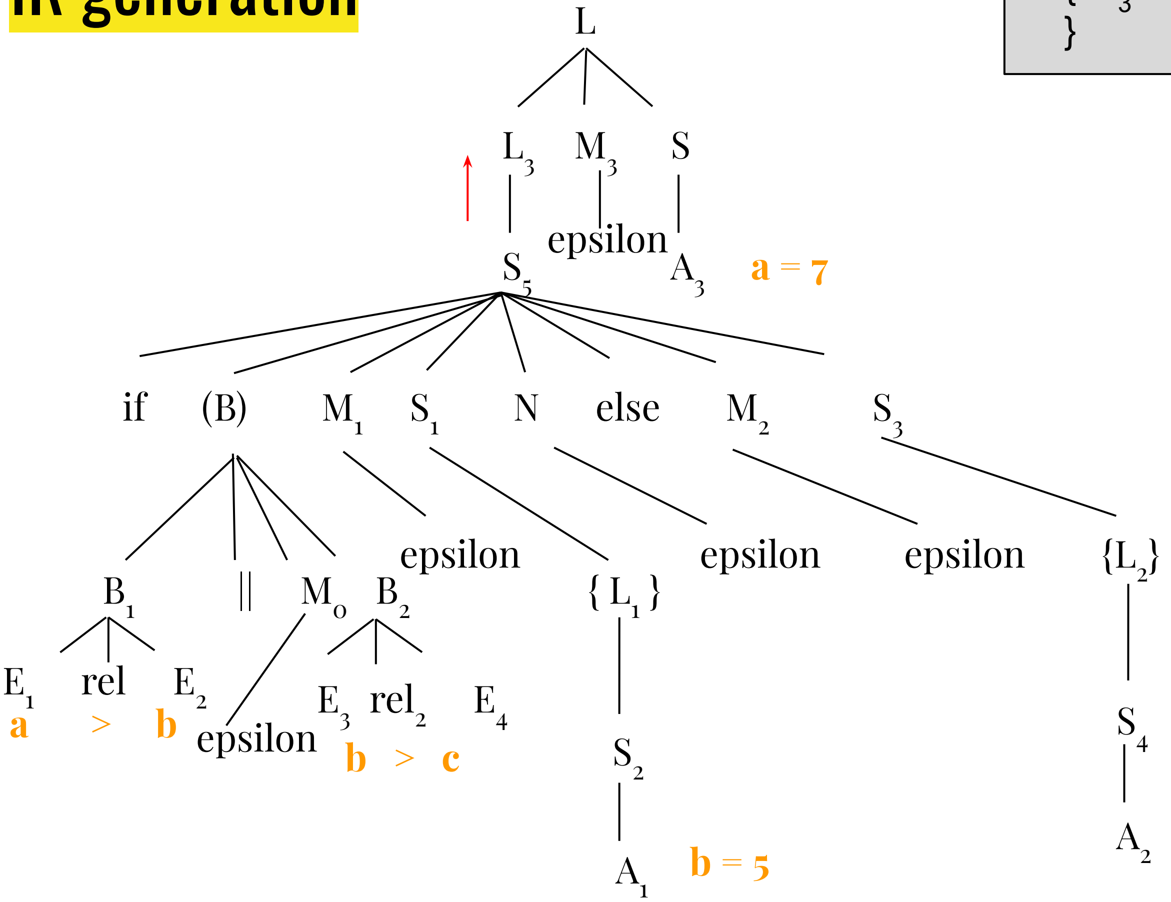
104 : b = 5

105 : goto _

106 : b = 6

IR generation

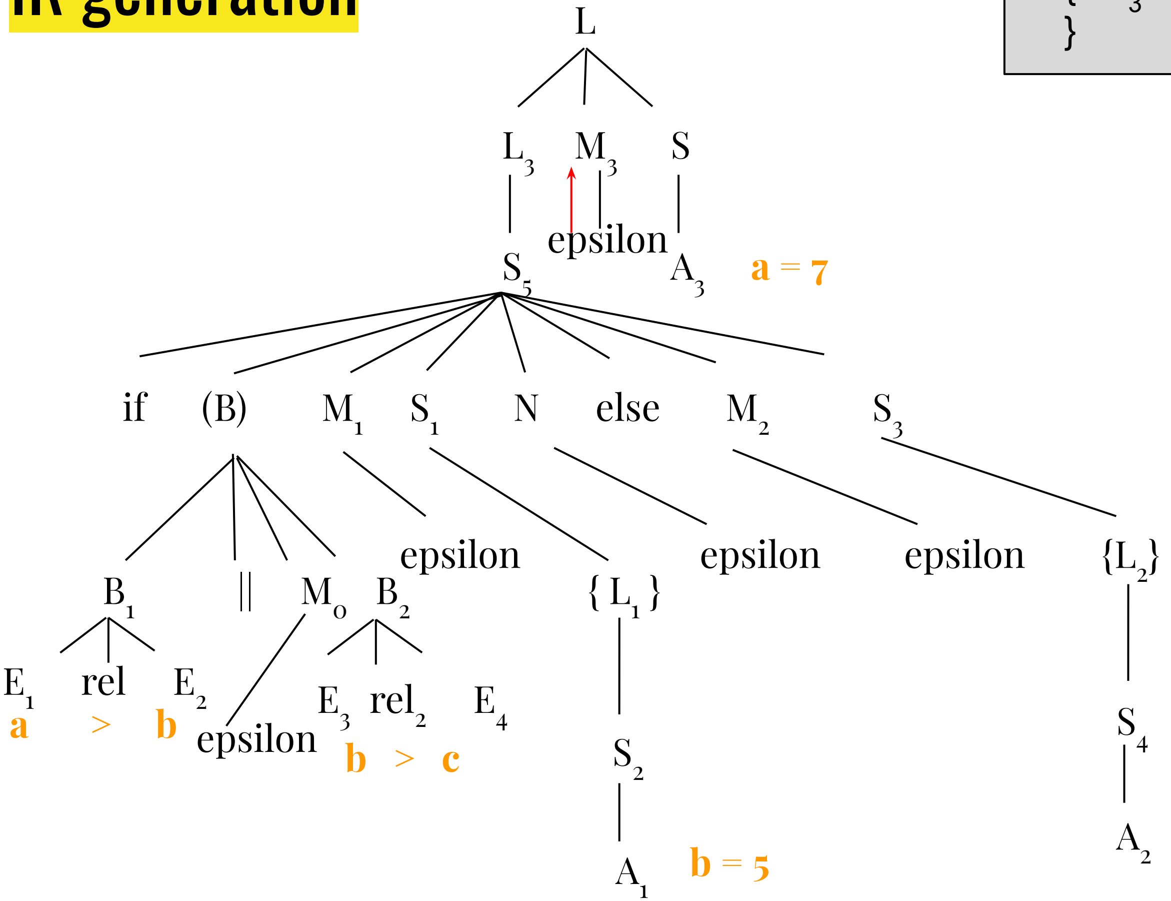
L_3 -> S_5
{ L_3.nextlist = S_5.nextlist = **{105}**;
}



```
100 : if a > b goto 104
101 : goto 102
102 : if b > c goto 104
103 : goto 106
104 : b = 5
105 : goto _
106 : b = 6
```

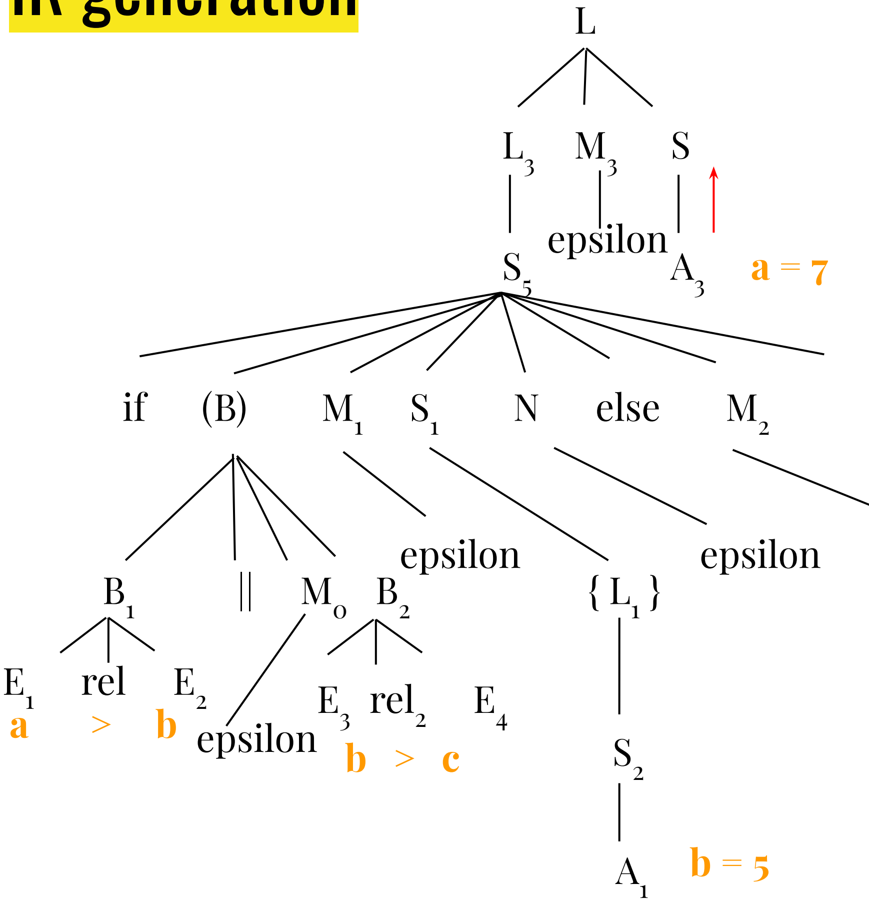
IR generation

M_3 -> epsilon
{ M_3.instr = nextinstr = **{107}**;
}



```
100 : if a > b goto 104
101 : goto 102
102 : if b > c goto 104
103 : goto 106
104 : b = 5
105 : goto _
106 : b = 6
```

IR generation

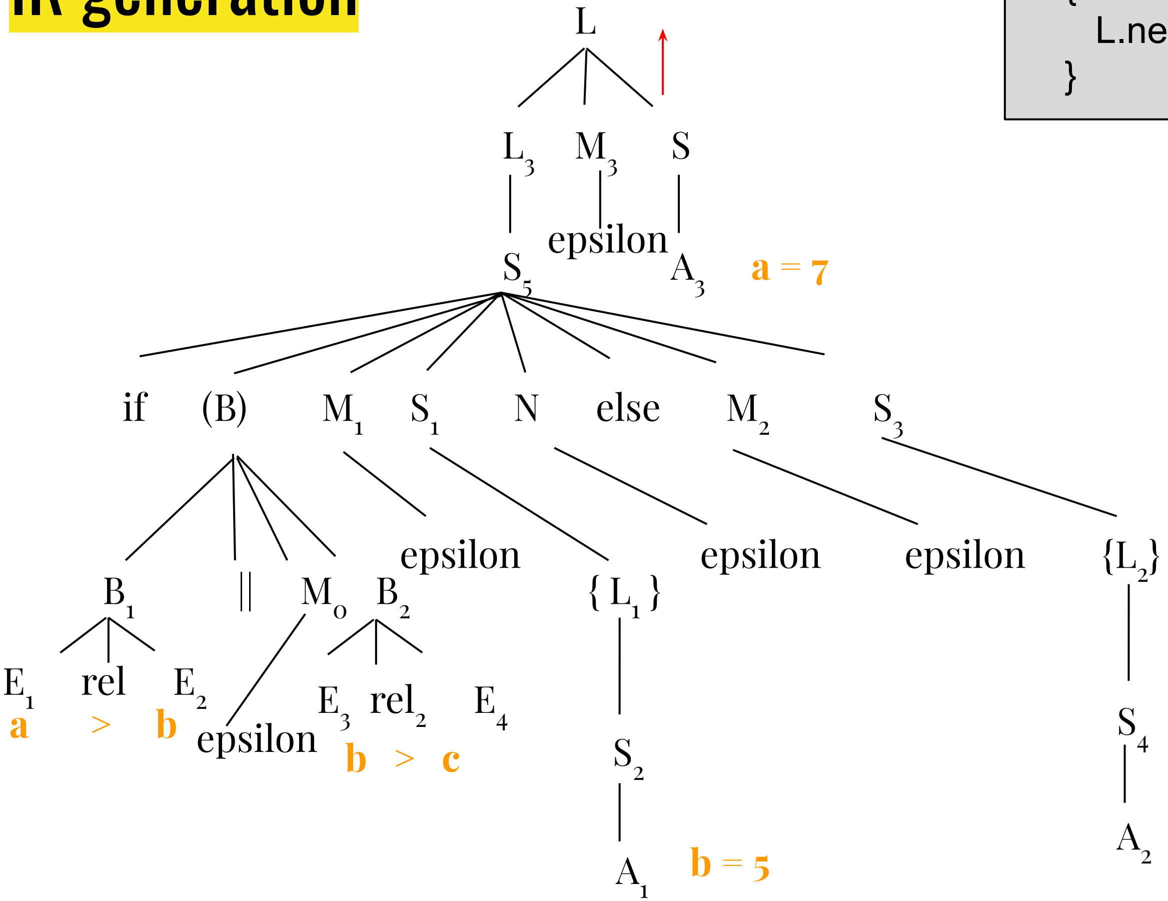


```
S -> A3
    {  S.nextlist = null;
    }
```

```
100 : if a > b goto 104
101 : goto 102
102 : if b > c goto 104
103 : goto 106
104 : b = 5
105 : goto _
106 : b = 6
107 : a = 7
```

IR generation

```
L -> L₃ M₃ S
   {  backpatch(L3.nextlist, M3.instr);
      L.nextlist = S.nextlist;
   }
```



100 : if a > b goto 104

101 : goto 102

102 : if b > c goto 104

103 : goto 106

104 : b = 5

105 : goto 107

106 : b = 6

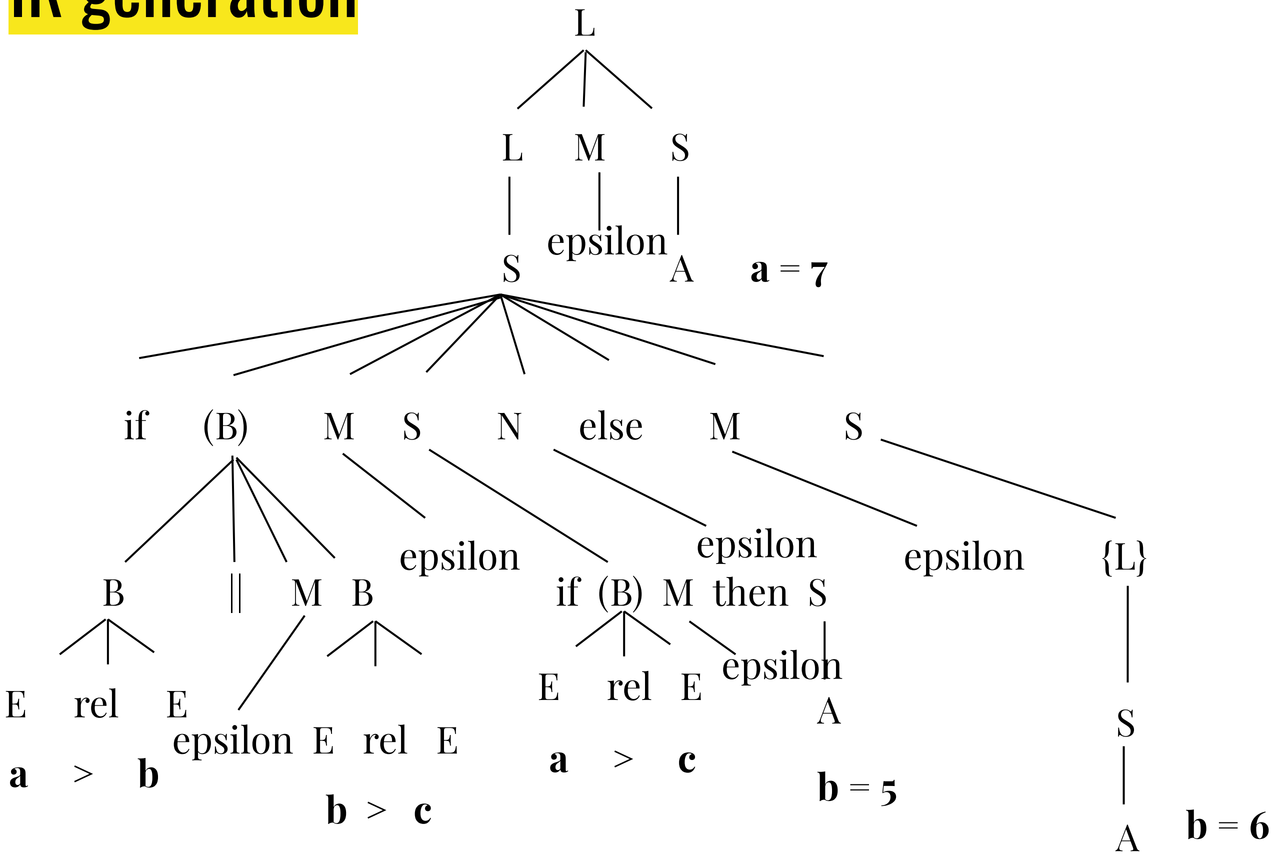
107 : a = 7

Tutorial - 6

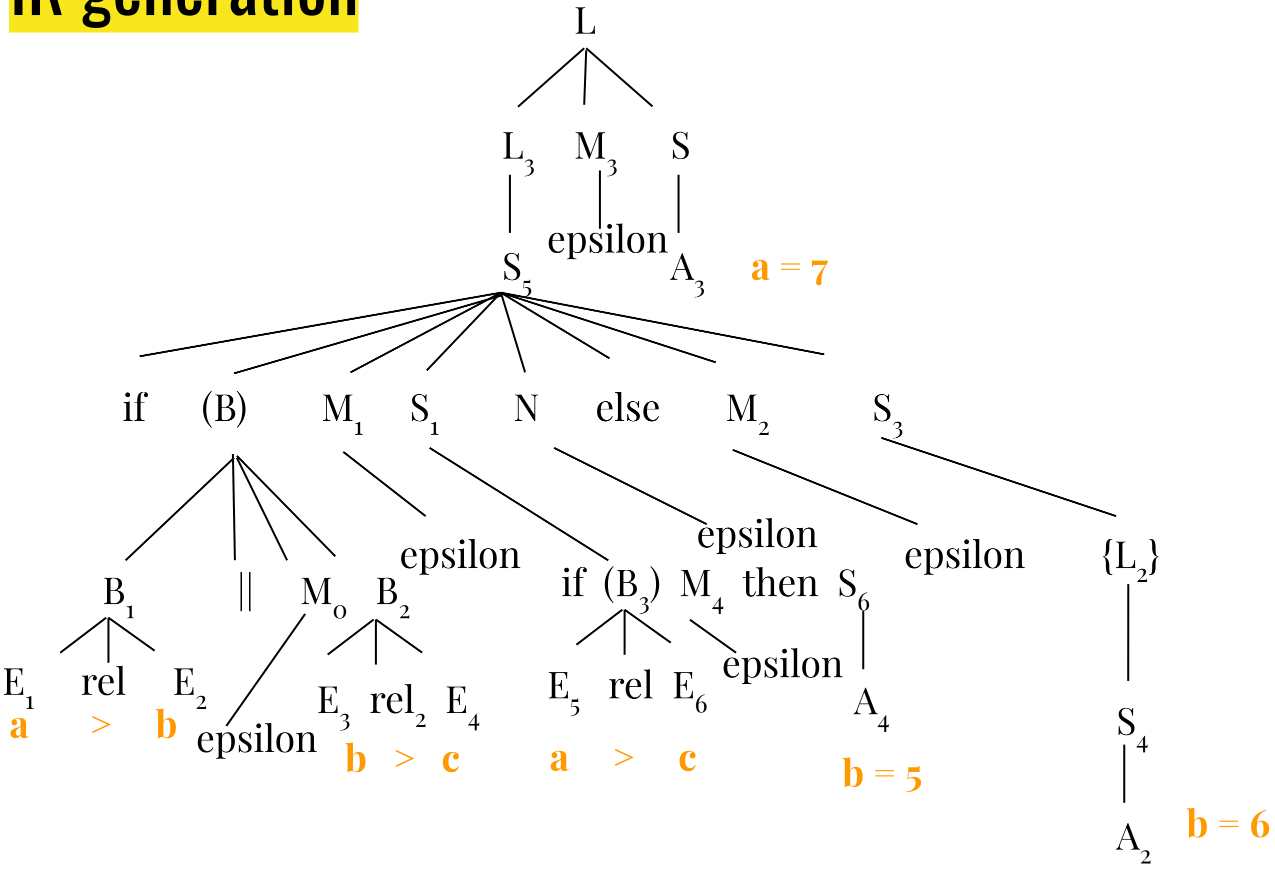
```
if(a > b || b > c)
{
   if(a > c)
  {
    b = 5;
  }
}
else
{
    b = 6;
}
a = 7;
```

```
100 : if a > b goto 104
101 : goto 102
102 : if b > c goto 104
103 : goto 108
104 : if a > c goto 106
105 : goto 109
106 : b = 5
107 : goto 109
108 : b = 6
109 : a = 7
```

IR generation

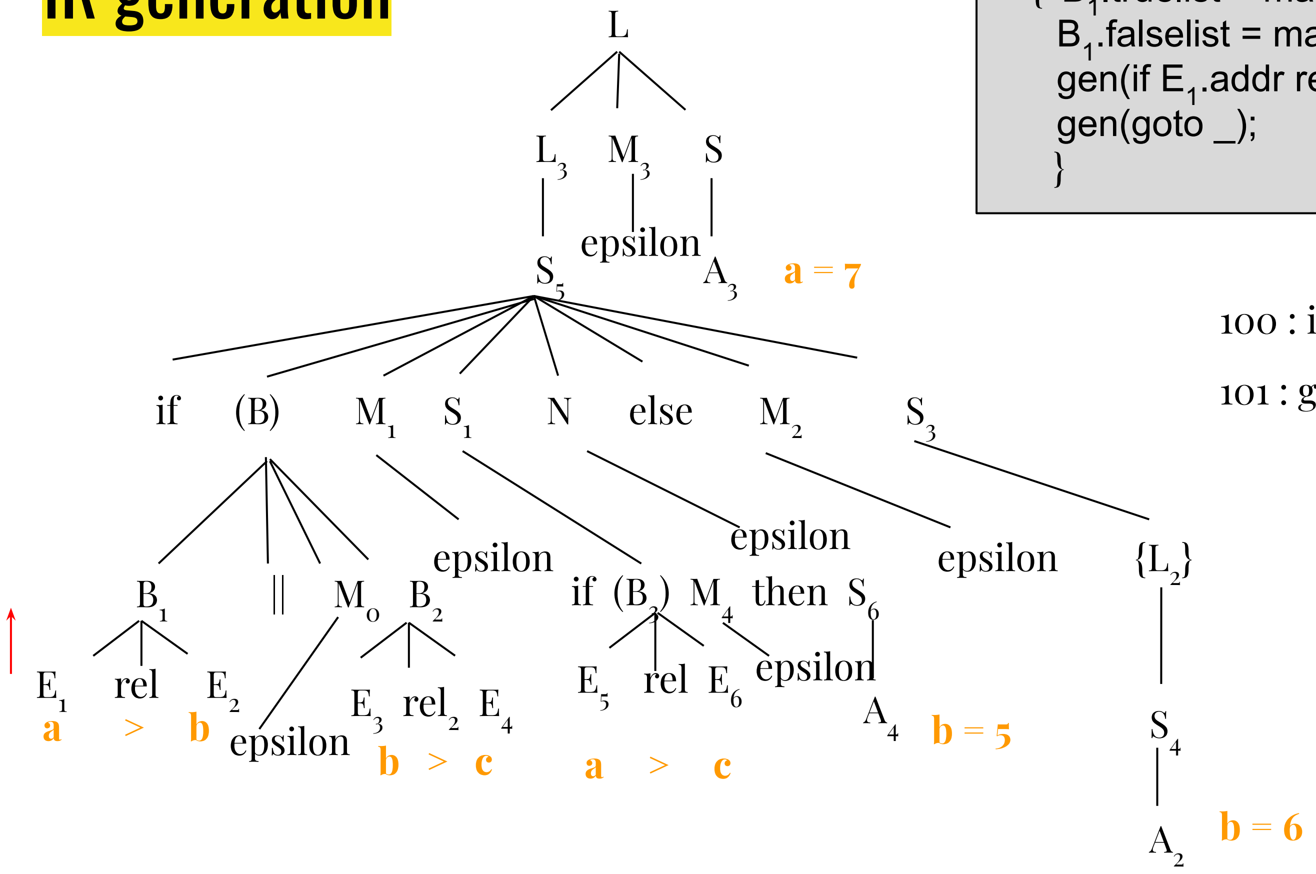


IR generation



IR generation

```
$B_1 \twoheadrightarrow E_1$ rel $E_2$
  { $B_1$.truelist = makelist(nextinstr); **= {100}**
    $B_1$.falselist = makelist(nextinstr+ 1); **= {101}**
    gen(if $E_1$.addr rel.op $E_2$.addr goto _);
    gen(goto _);
  }
```

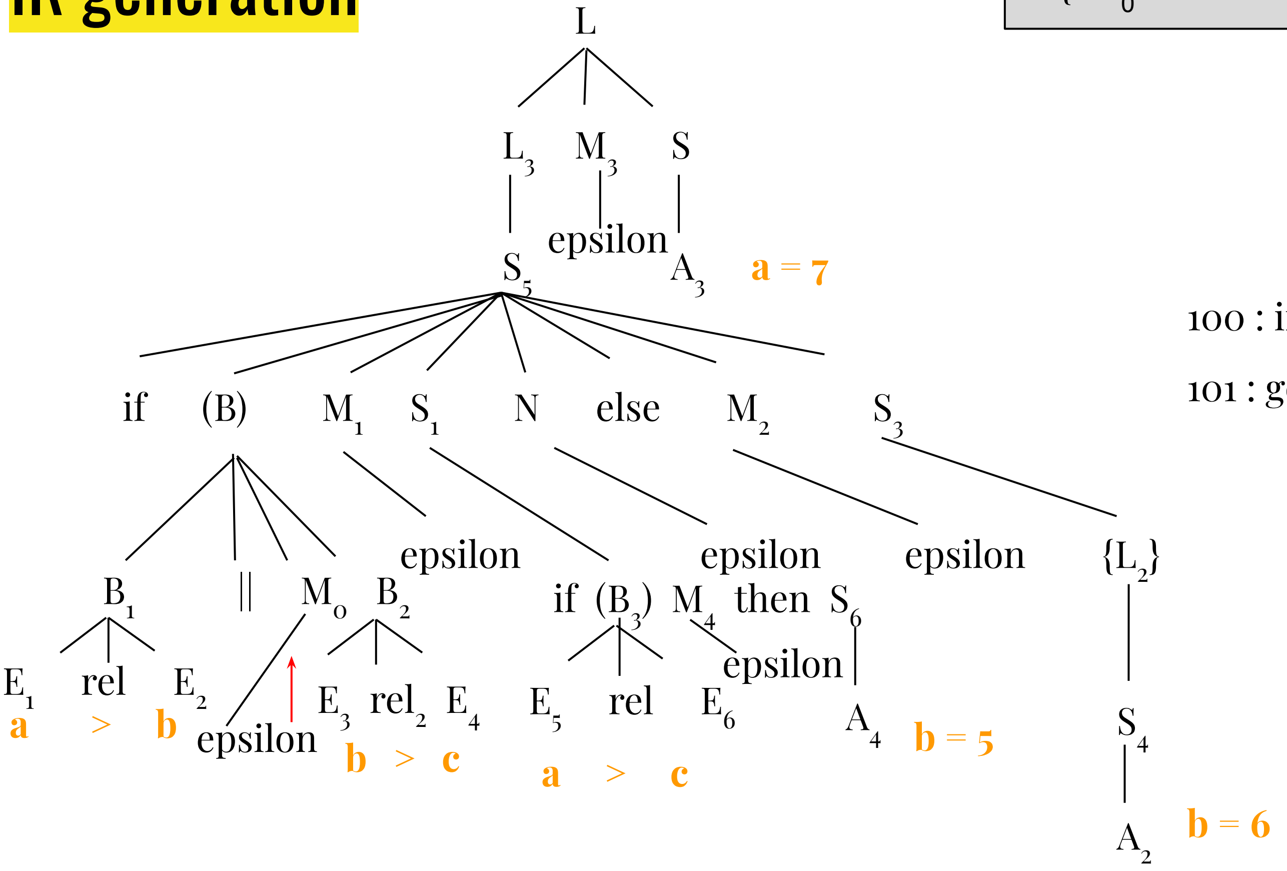


100 : if a > b goto _

101 : goto _

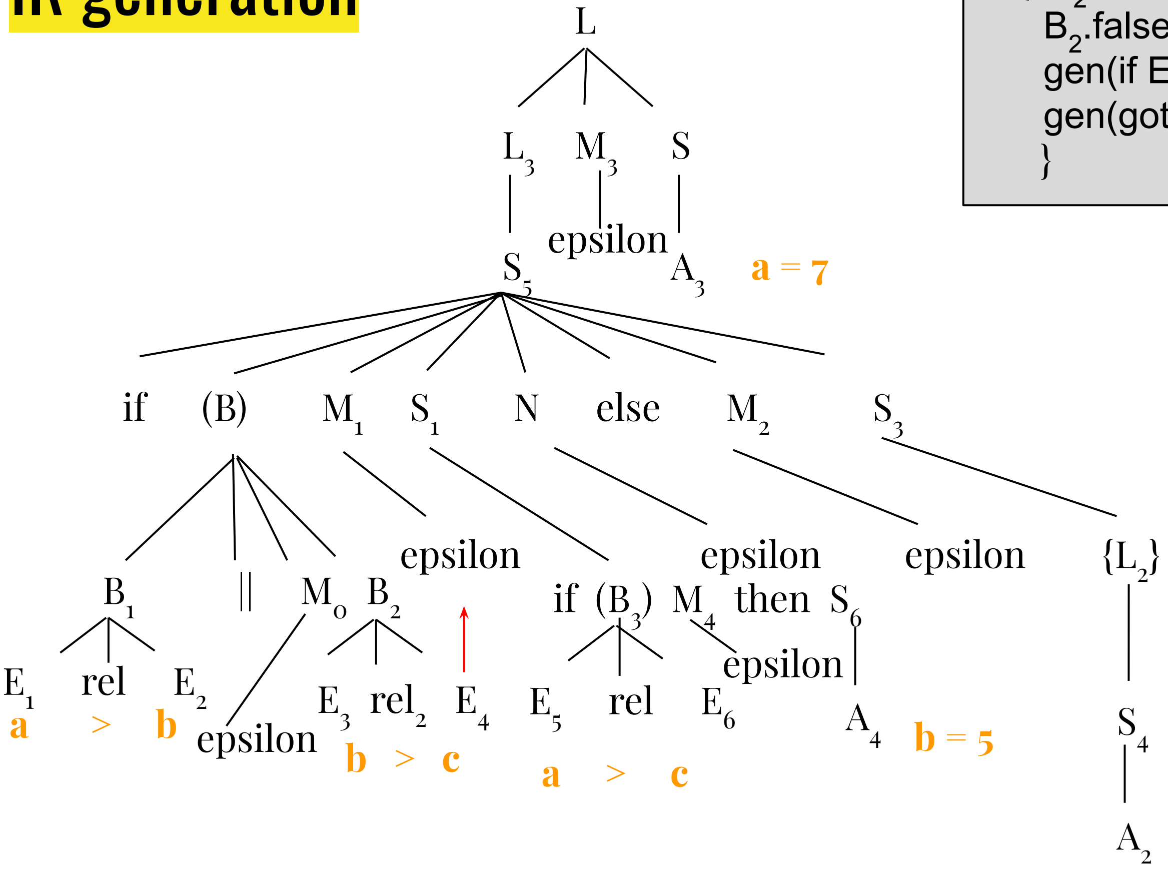
IR generation

$M_0 \twoheadrightarrow$ epsilon
{ M_0.instr = nextinstr = **{102}** }



```
100 : if a > b goto _

101 : goto _
```

IR generation



```
$B_2 \rightarrowtail E_3$ rel $E_4$
  { $B_2$.truelist = makelist(nextinstr); **= {102}**
    $B_2$.falselist = makelist(nextinstr+ 1); **= {103}**
    gen(if $E_3$.addr rel.op $E_4$.addr goto _);
    gen(goto _);
  }
```

100 : if a > b goto _

101 : goto _

102 : if b > c goto _

103 : goto _

IR generation

$B \twoheadrightarrow B_1 \,||\, M_0\ B_2$

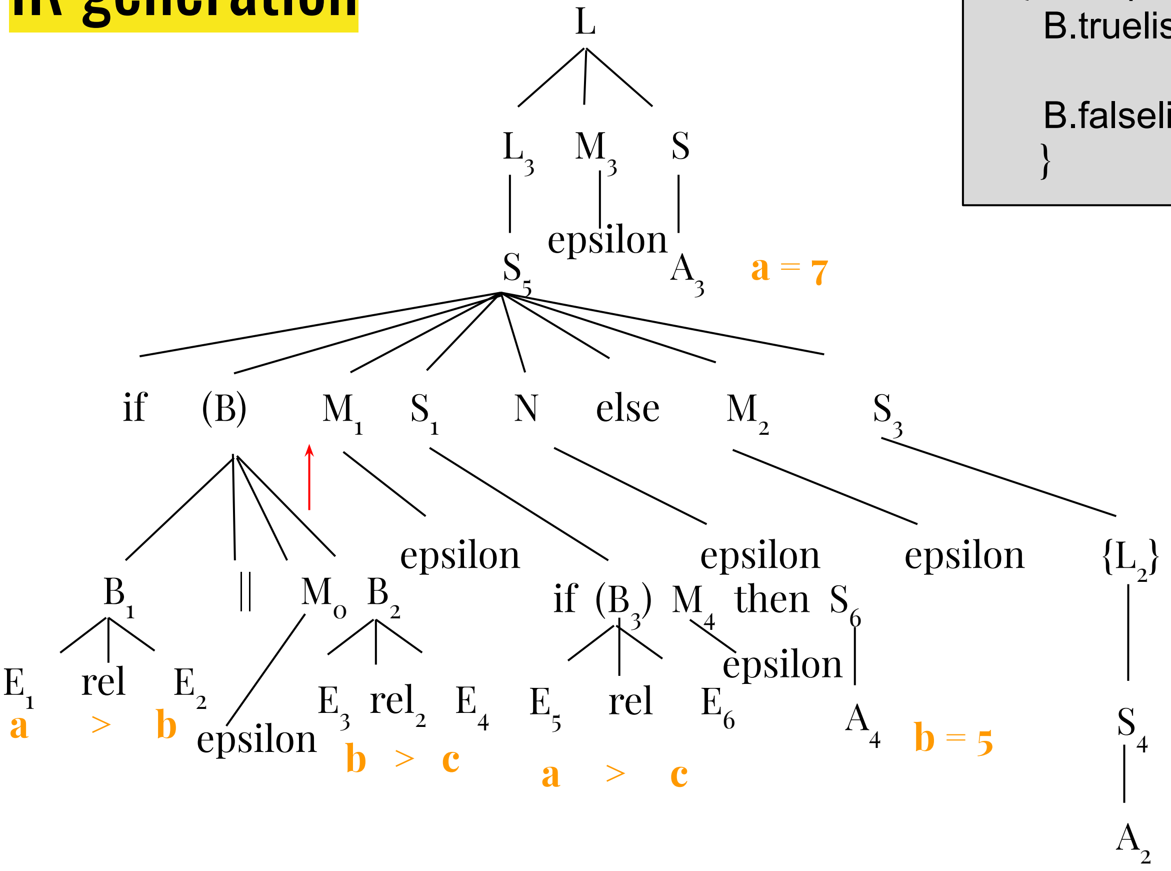
{ backpatch(B_1.falselist, M_0.instr);

B.truelist = merge(B_1.truelist, B_2.truelist)

= {100, 102};

B.falselist = B_2.falselist **= {103}**;

}



100 : if a > b goto _

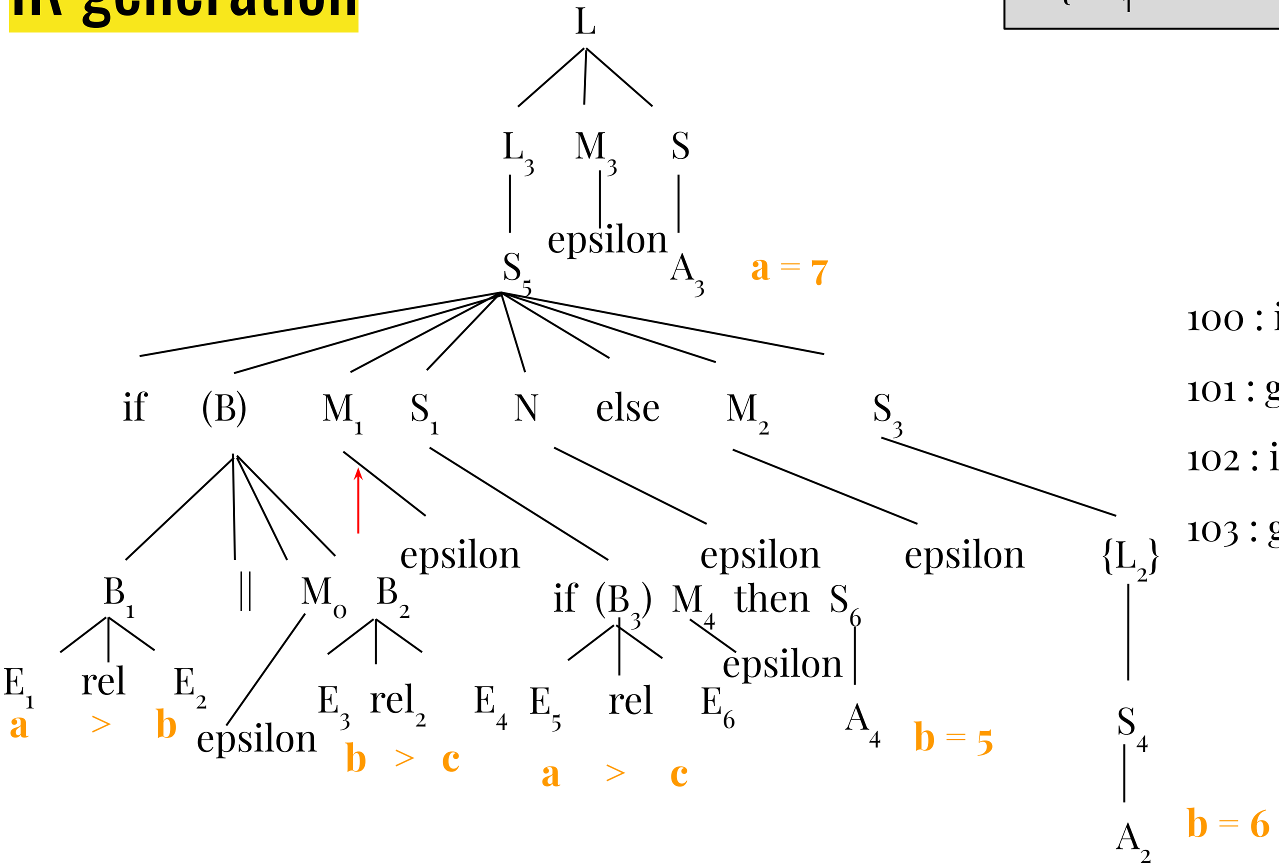
101 : goto 102

102 : if b > c goto _

103 : goto _

IR generation

$M_1 \twoheadrightarrow$ epsilon
{ M_1.instr = nextinstr = **{104}** }



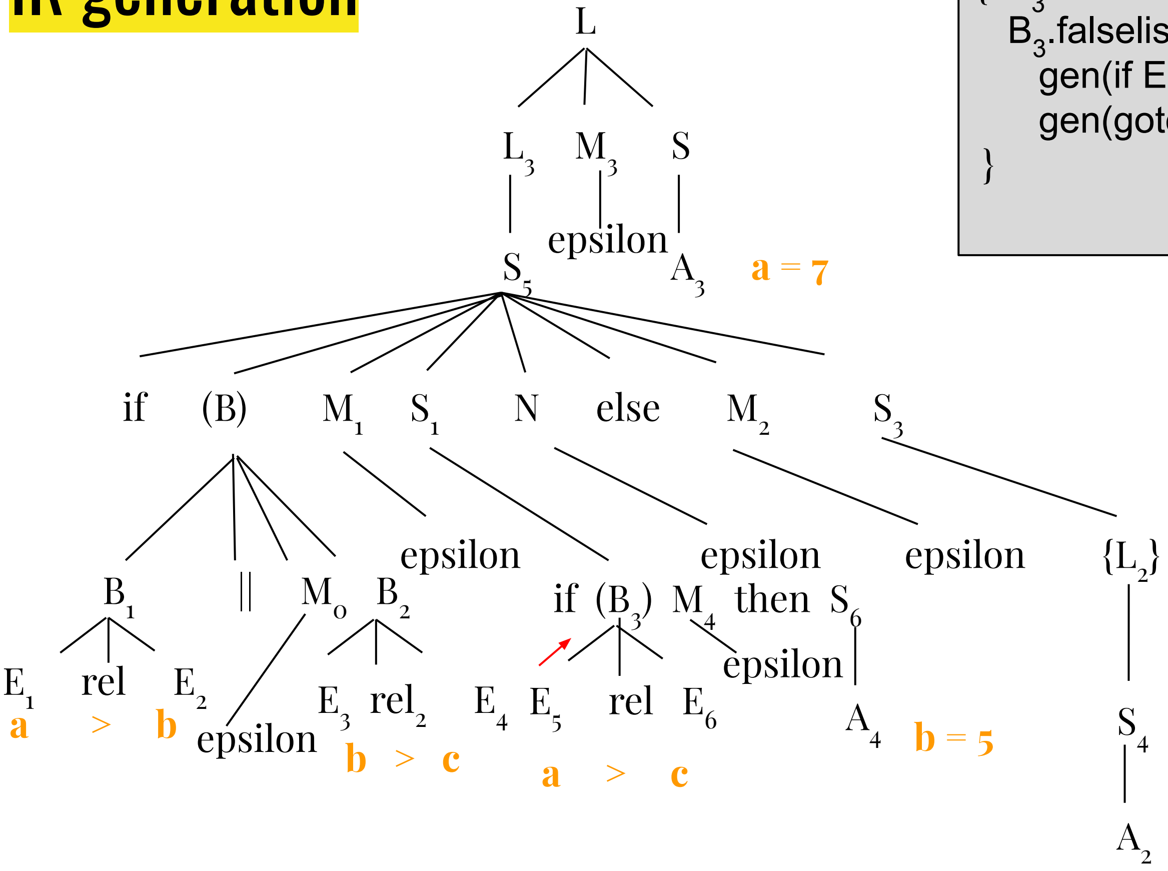
100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

103 : goto _

IR generation



B_3 -> E_5 rel E_6

{ B_3.truelist = makelist(nextinstr); **= {104}**

B_3.falselist = makelist(nextinstr+ 1); **= {105}**

gen(if E_5.addr rel.op E_6.addr goto _);

gen(goto _);

}

100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

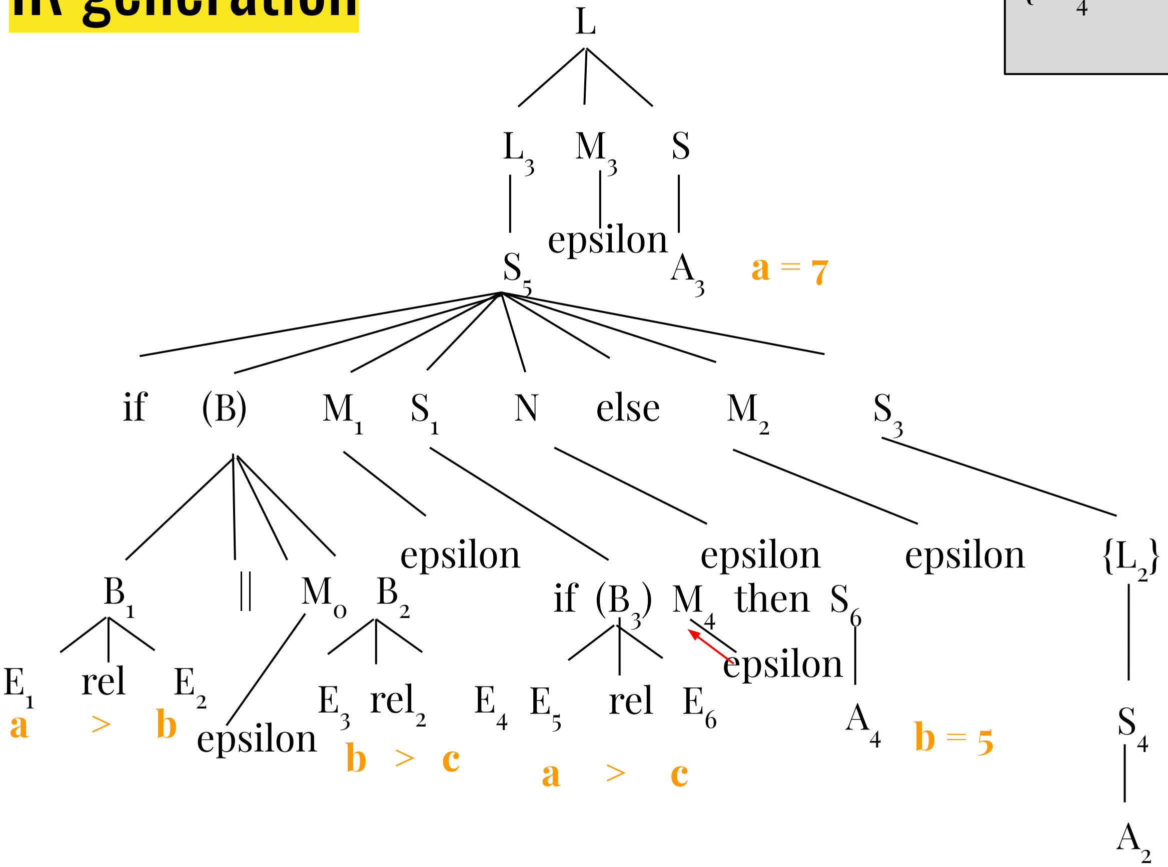
103 : goto _

104 : if a > c goto _

105 : goto _

IR generation

M_4 -> epsilon
{ M_4.instr = nextinstr = **{106}**; }



100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

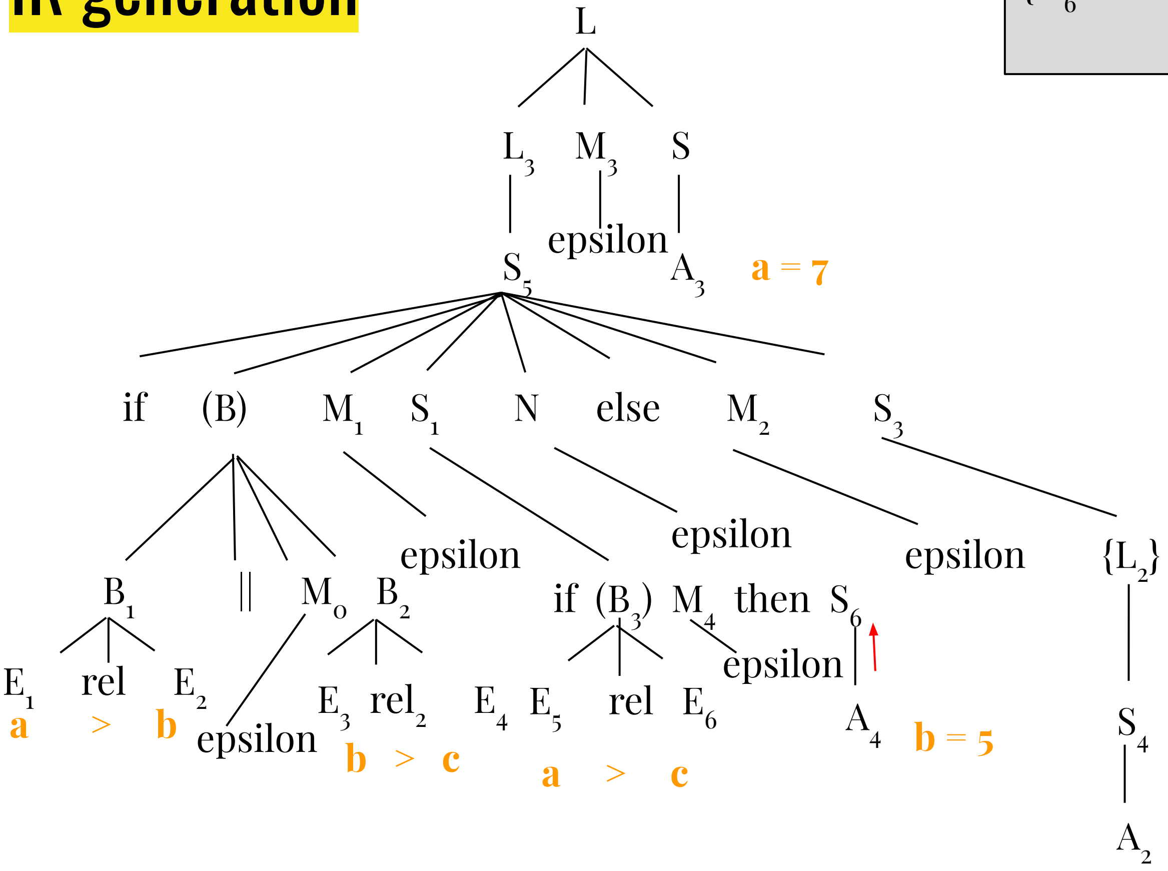
103 : goto _

104 : if a > c goto _

105 : goto _

IR generation

S_6 -> A_4
{ S_6.nextlist = null; }



100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

103 : goto _

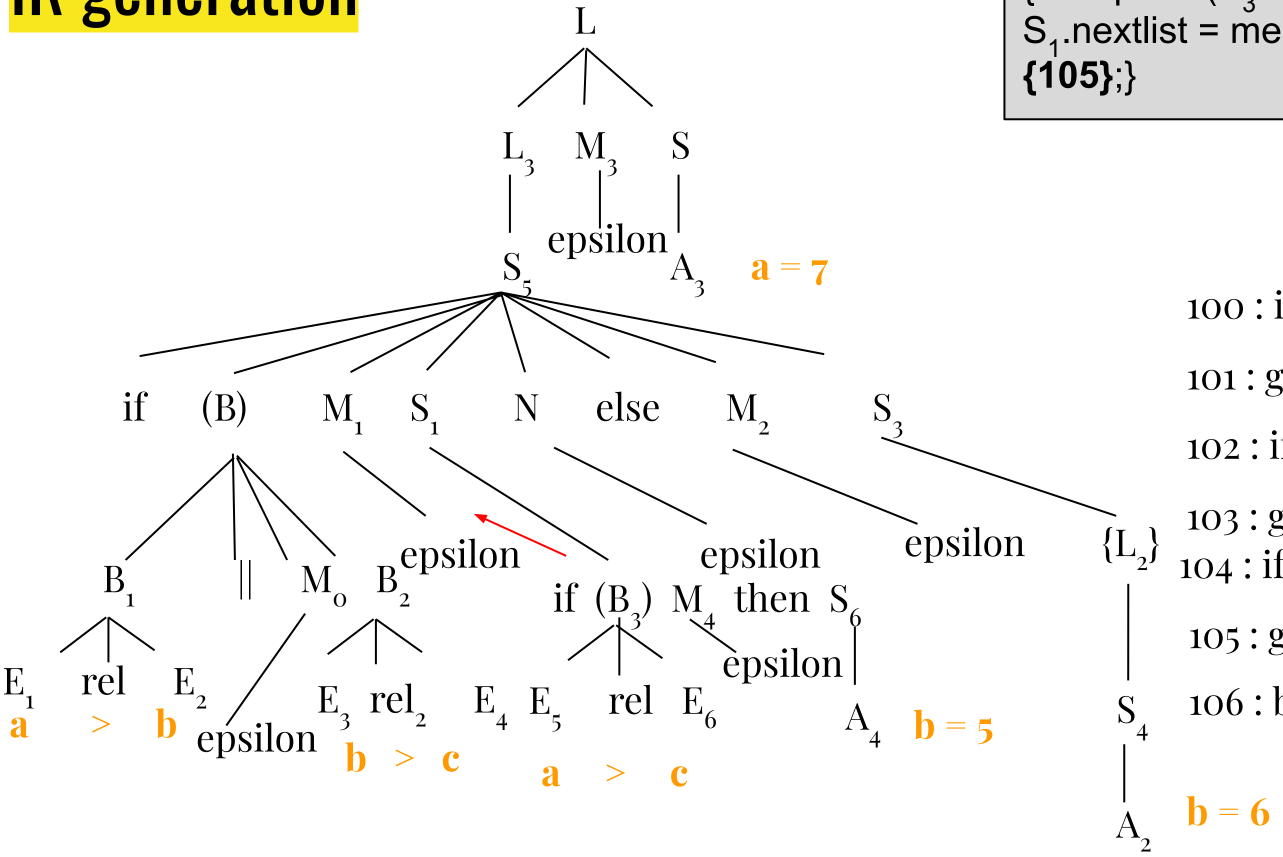
104 : if a > c goto _

105 : goto _

106 : b = 5

IR generation

S_1 -> if (B_3) M_4 then S_6
{backpatch(B_3.truelist, M_4.instr);
S_1.nextlist = merge(B_3.falselist, S_6.nextlist) = **{105}**;}



100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

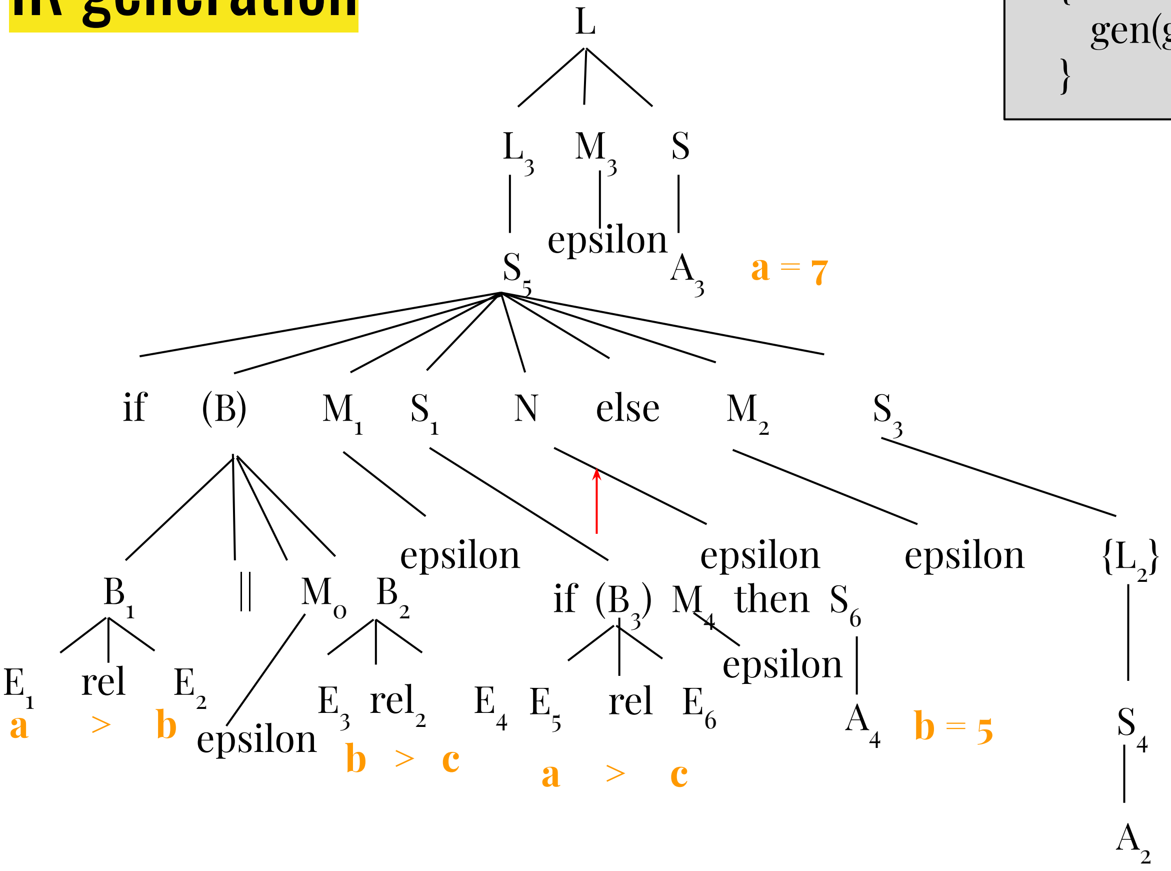
103 : goto _

104 : if a > c goto 106

105 : goto _

106 : b = 5

IR generation



N ↣ epsilon
  { N.nextlist = makelist(nextinstr) = 107;
    gen(goto _);
  }

100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

103 : goto _

104 : if a > c goto 106

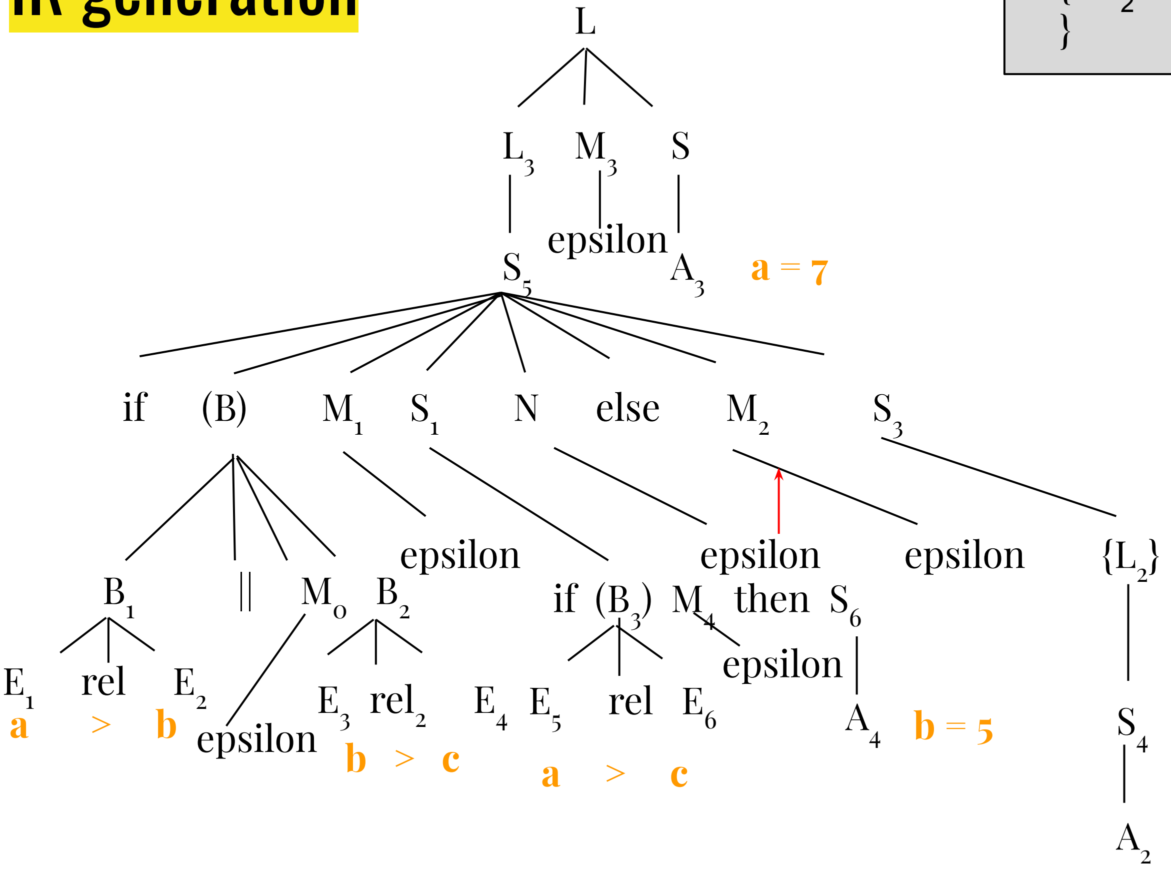
105 : goto _

106 : b = 5

107 : goto _

IR generation

$M_2 \twoheadrightarrow$ epsilon
{ M_2.instr = nextinstr = 108;
}



100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

103 : goto _

104 : if a > c goto 106

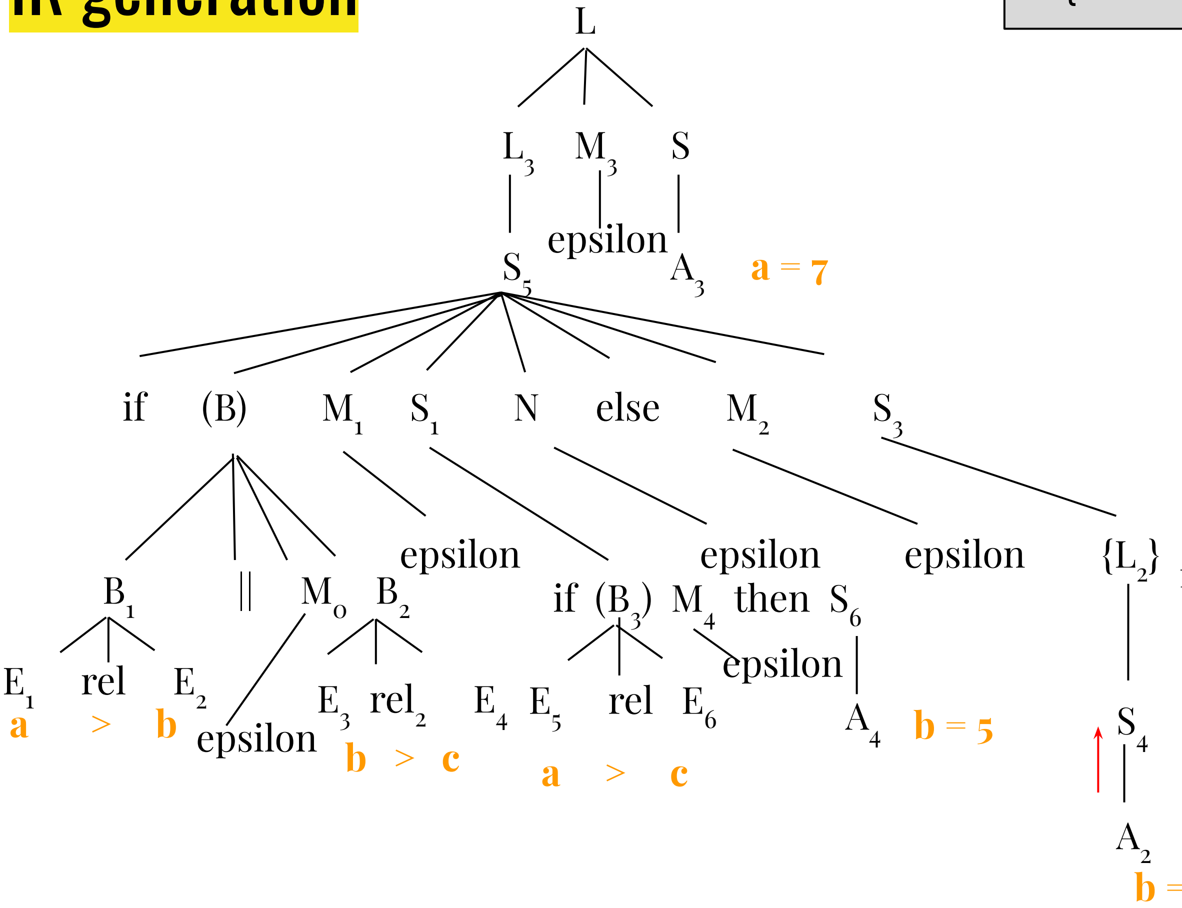
105 : goto _

106 : b = 5

107 : goto _

IR generation

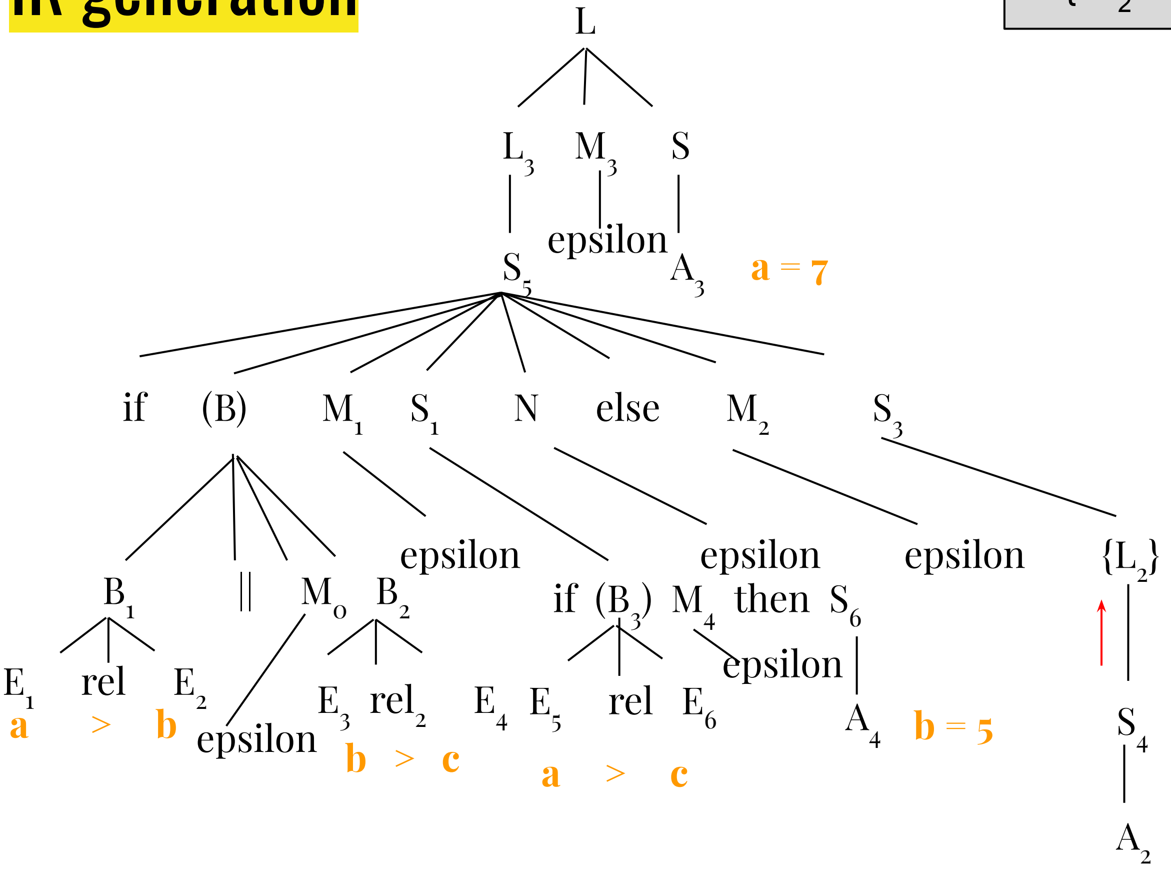
S_4 -> A_2
{ S4.nextlist = null; }



```
100 : if a > b goto _
101 : goto 102
102 : if b > c goto _
103 : goto _
104 : if a > c goto 106
105 : goto _
106 : b = 5
107 : goto _
108 : b = 6
```

IR generation

L_2 -> S_4
{ L_2.nextlist = S_4.nextlist = **null** ; }



100 : if a > b goto _

101 : goto 102

102 : if b > c goto _

103 : goto _

104 : if a > c goto 106

105 : goto _

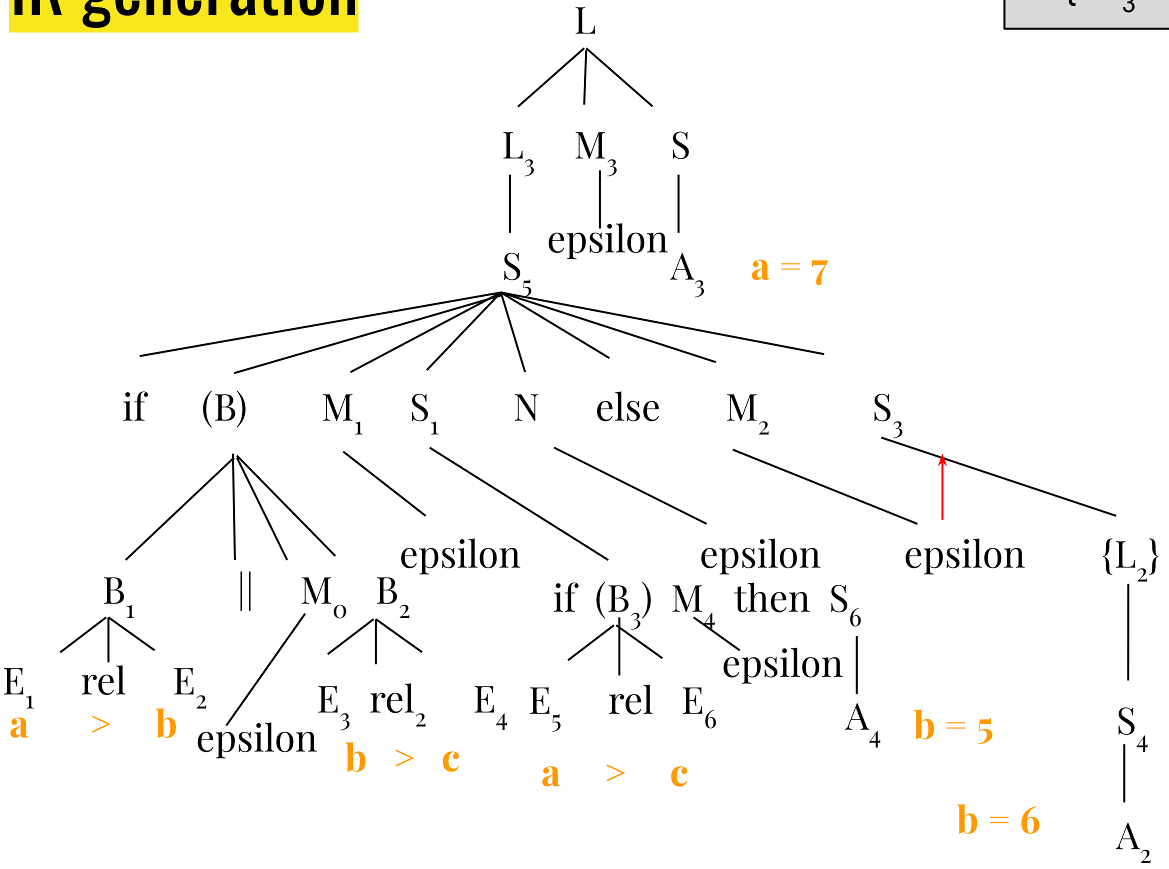
106 : b = 5

107 : goto _

108 : b = 6

IR generation

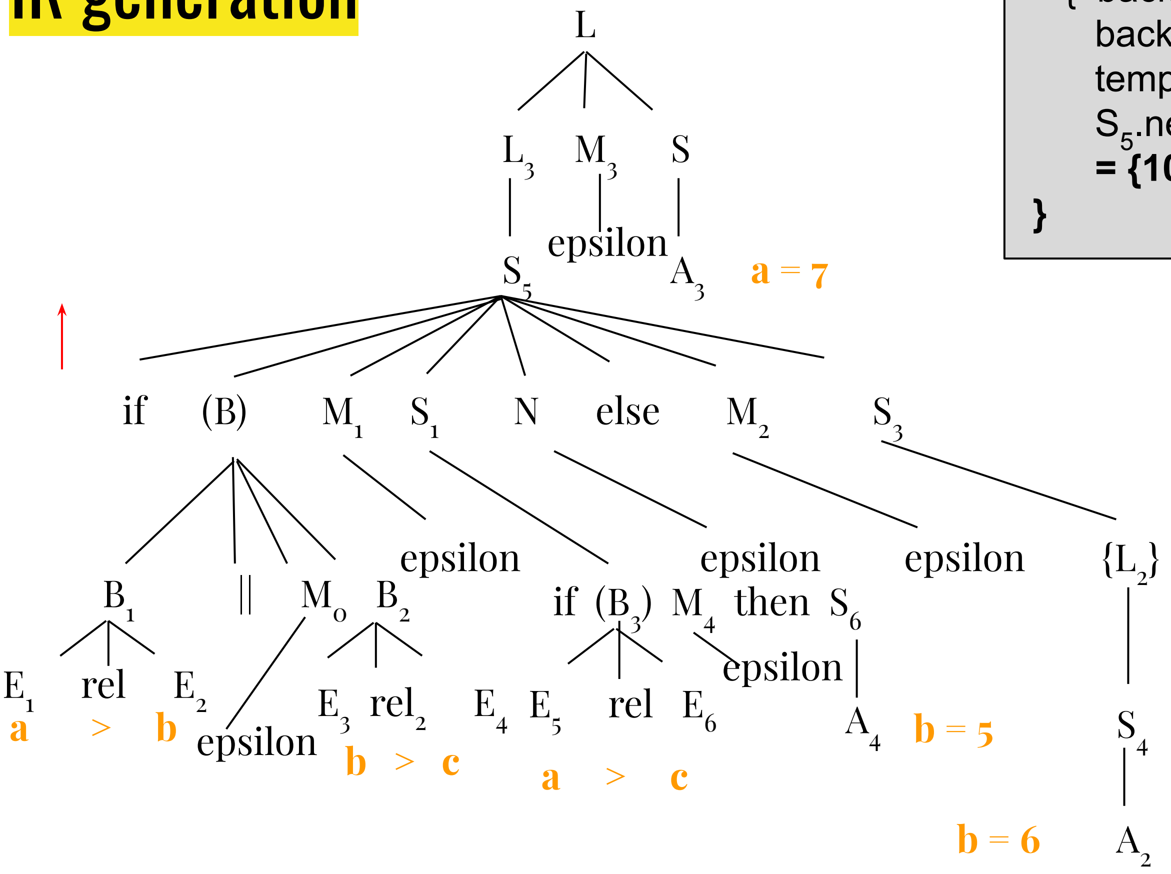
S_3 -> { L_2}
{ S_3.nextlist = L_2.nextlist **= null** ; }



```
100 : if a > b goto _
101 : goto 102
102 : if b > c goto _
103 : goto _
104 : if a > c goto 106
105 : goto _
106 : b = 5
107 : goto _
108 : b = 6
```

IR generation

S_5 -> if (B) M_1 S_1 N else M_2 S_3
{ backpatch(B.truelist, M_1.instr);
backpatch(B.falselist, M_2.instr);
temp = merge(S_1.nextlist, N.nextlist);
S_5.nextlist = merge(temp, S_3.nextlist)
= {105, 107};
}



100 : if a > b goto 104

101 : goto 102

102 : if b > c goto 104

103 : goto 108

104 : if a > c goto 106

105 : goto _

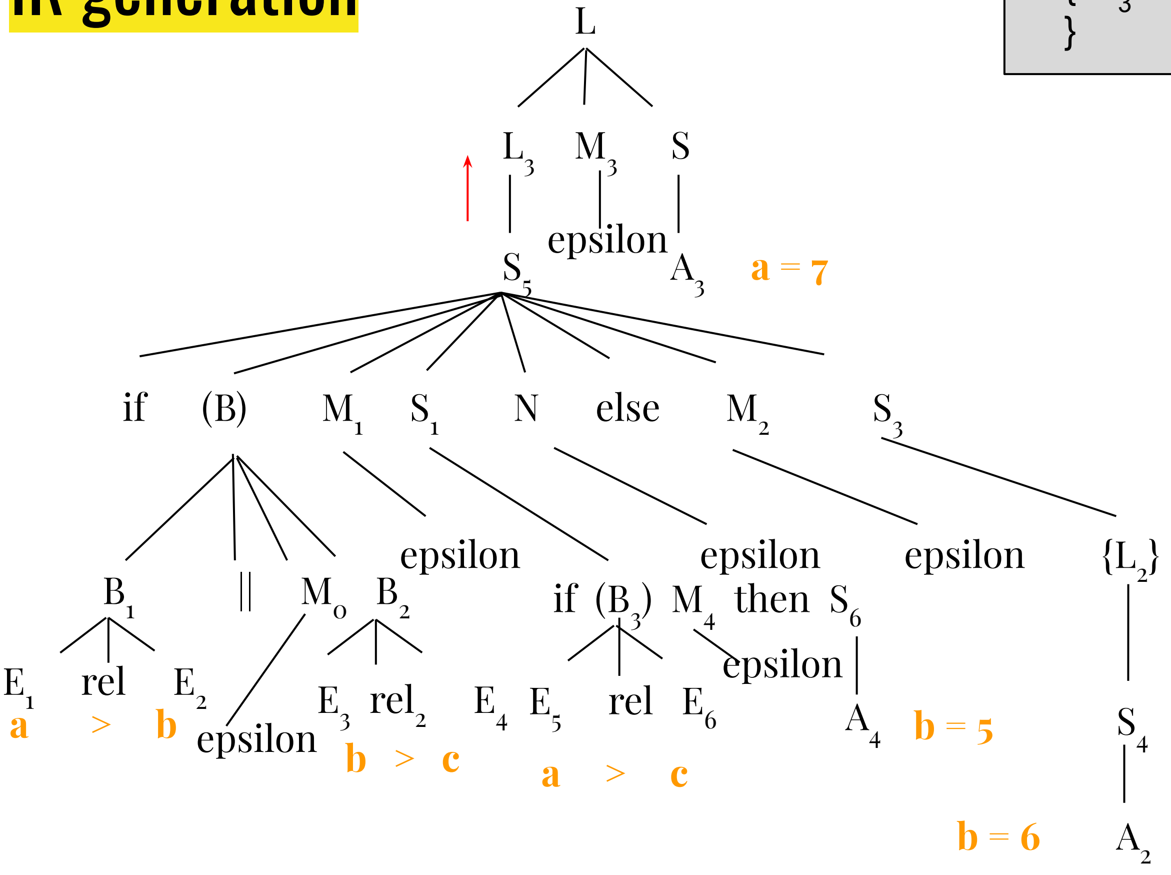
106 : b = 5

107 : goto _

108 : b = 6

IR generation

```
L3 -> S5
  {  L3.nextlist = S5.nextlist = {105, 107};
  }
```



100 : if a > b goto 104

101 : goto 102

102 : if b > c goto 104

103 : goto 108

104 : if a > c goto 106

105 : goto _

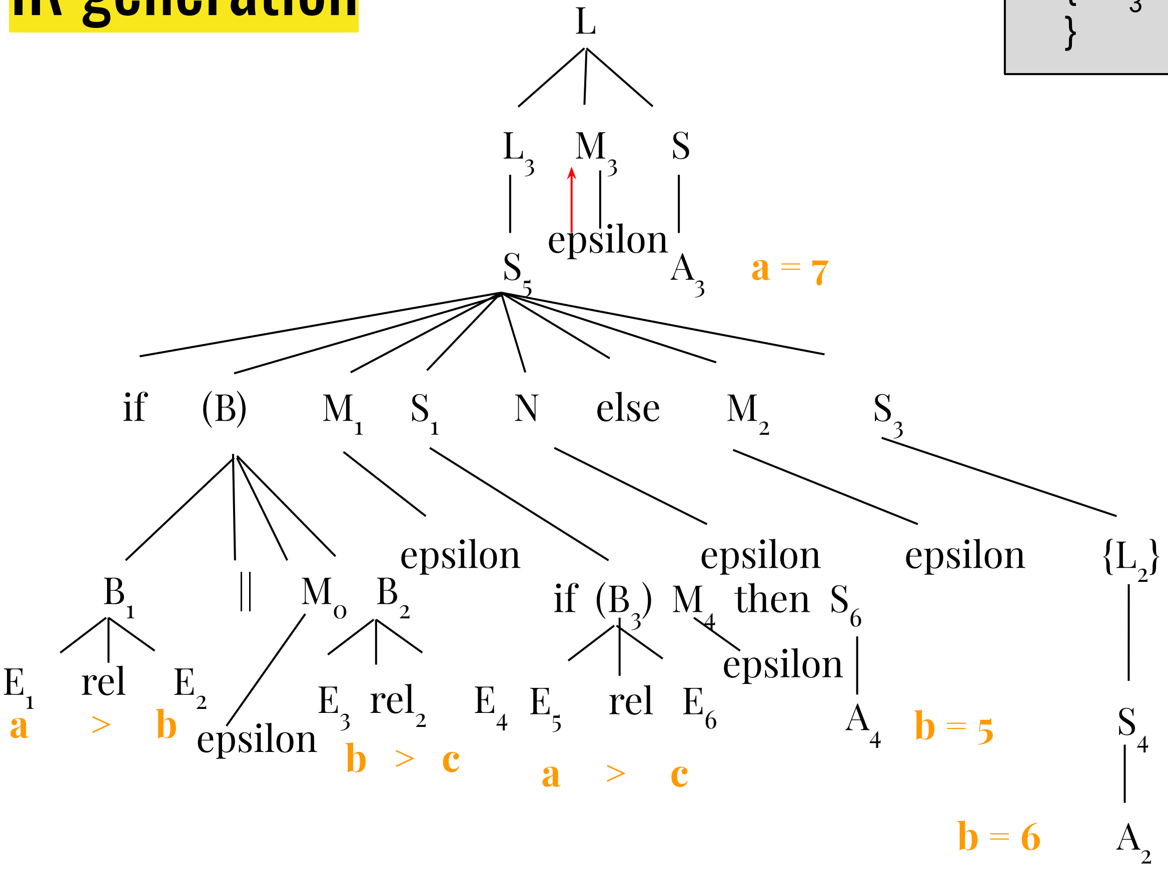
106 : b = 5

107 : goto _

108 : b = 6

IR generation

M_3 -> epsilon
{ M_3.instr = nextinstr **= {109}**;
}



100 : if a > b goto 104

101 : goto 102

102 : if b > c goto 104

103 : goto 108

104 : if a > c goto 106

105 : goto _

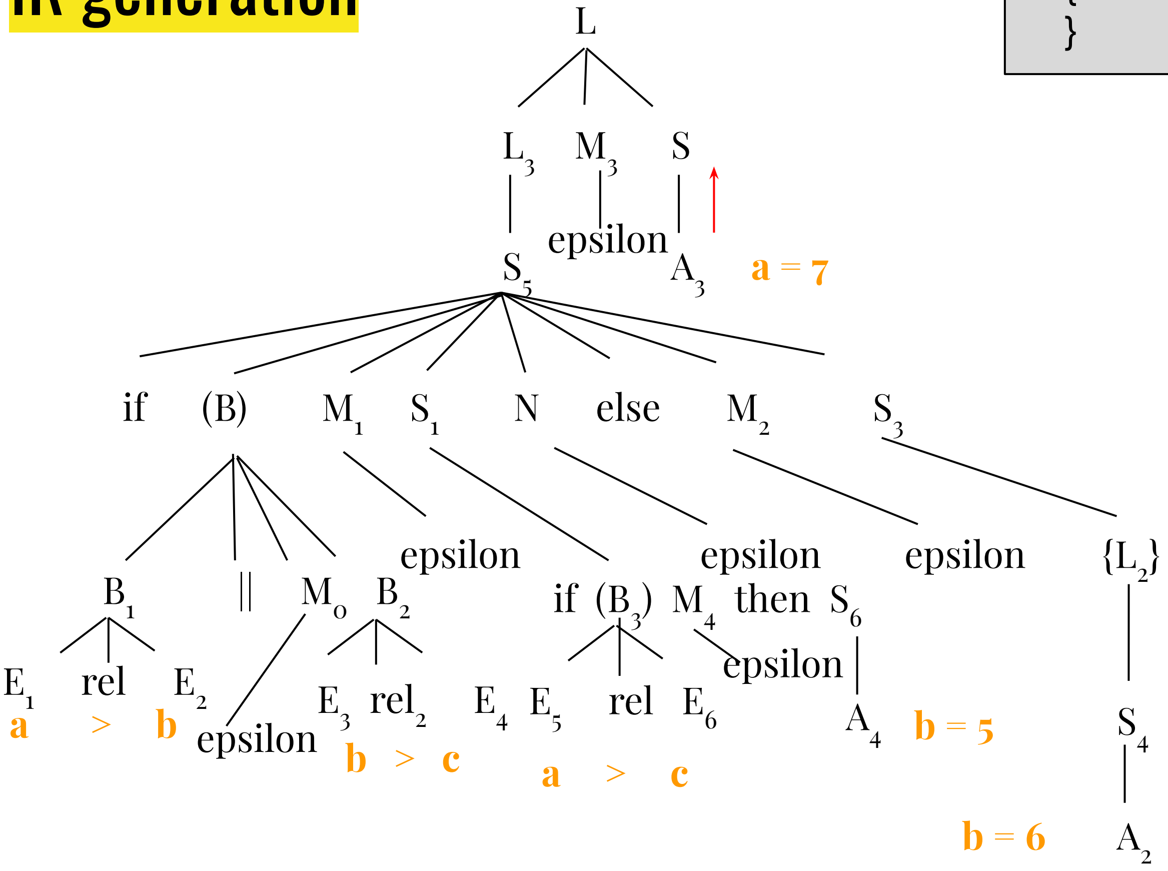
106 : b = 5

107 : goto _

108 : b = 6

IR generation

```
S -> A3
   {  S.nextlist = null;
   }
```



100 : if a > b goto 104

101 : goto 102

102 : if b > c goto 104

103 : goto 108

104 : if a > c goto 106

105 : goto _

106 : b = 5

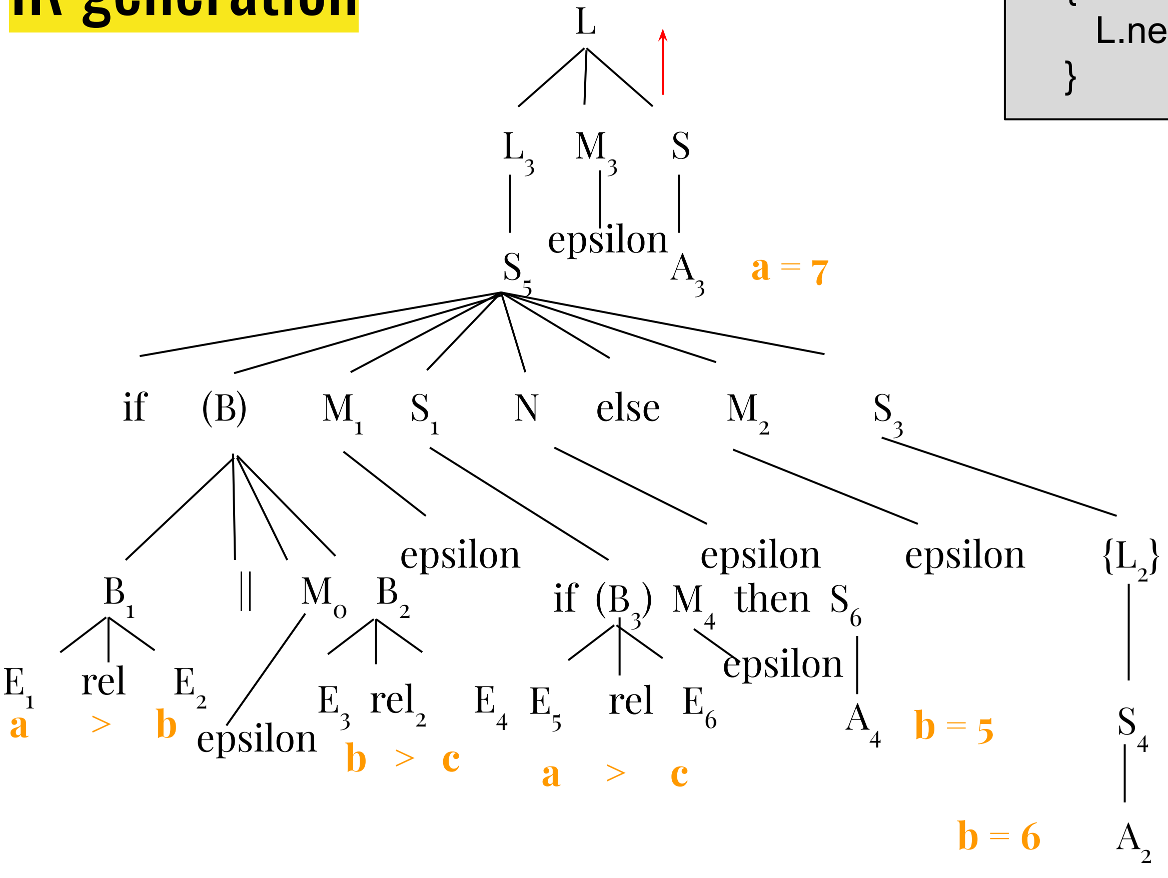
107 : goto _

108 : b = 6

109 : a = 7

IR generation

```
L -> L3 M3 S
  {  backpatch(L3.nextlist, M3.instr);
     L.nextlist = S.nextlist;
  }
```



```
100 : if a > b goto 104
101 : goto 102
102 : if b > c goto 104
103 : goto 108
104 : if a > c goto 106
105 : goto 109
106 : b = 5
107 : goto 109
108 : b = 6
109 : a = 7
```