

**Compiler Design**  
**Prof. Santanu Chattopadhyay**  
**Department of E & Ec Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 56**  
**Intermediate Code Generation (Contd.)**

So, in our last class we were discussing on translation of Boolean expressions.

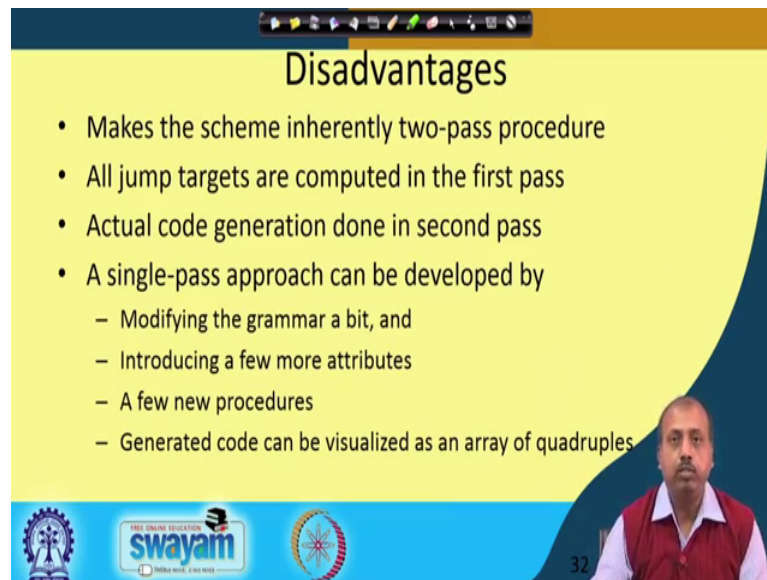
(Refer Slide Time: 00:21)

**Translation of Boolean Expressions**

$B \rightarrow B1 \text{ and } B2$ { B1.true = newlabel() B1.false = B.false B2.true = B.true B2.false = B.false B.code = B1.code    gen(B1.true, ':')    B2.code }	$B \rightarrow \text{not } B1$ { B1.true = B.false B1.false = B.true B.code = B1.code }	$B \rightarrow (B1)$ { B1.true = B.true B1.false = B.false B.code = B1.code }
$B \rightarrow \text{id1 relop id2}$ { B.code = gen('if' id1.place relop id2.place 'goto' B.true)    gen('goto' B.false)}		
$B \rightarrow \text{true}$ { B.code = gen('goto' B.true) }	$B \rightarrow \text{false}$ { B.code = gen('goto' B.false) }	<p>Handwritten notes: A control flow graph with a start node, a decision node, and two paths labeled 'true' and 'false' leading to 'S1' and 'S2' respectively. The text 'if x1 then S1 else S2' is written next to it.</p>

And in that process we have seen a first one grammar where it is that the Boolean expression grammar that we know from our previous discussions on the passed chapter. So, rule B may non terminal B can produce B1 and B2 or B1 or B2 or say B1 naught of B1 or within bracket B1 id1 relop id2 like that.

(Refer Slide Time: 00:53)



### Disadvantages

- Makes the scheme inherently two-pass procedure
- All jump targets are computed in the first pass
- Actual code generation done in second pass
- A single-pass approach can be developed by
  - Modifying the grammar a bit, and
  - Introducing a few more attributes
  - A few new procedures
  - Generated code can be visualized as an array of quadruples

32

So, the main difficulty with this type of grammar was that it is difficult to have a single pass through the source code and generate the intermediary code. Why do I say so is because of this reason that we have to have this goto's like say. So, at places we have written that, so, this particularly this rule where it is so, id1 relop id2.

So, if id1 dot place is say a relational operators say it is greater than. So, if it is greater than id2 dot place then goto B dot true. So, this B dot true may not be available when we are looking doing this particular reduction a typical example is done if then else statement. So, if some Boolean expression say x greater than y then we have got the statement block S1 else the statement block S 2.

So, in the parts tree, so, it will be like this the if statement. So, the if statement it will have the keyword if then the Boolean condition B and then I will have the then part then S 1 block and then else S2 block. Now this B has got this x greater than y. So, this has got that this particular parts tree so, x greater than y.

Now, the point is when this particular reduction is being made like when I am doing this reduction x greater than y to B. So, at that time so, this S 1 code has not been generated the code for S1 has not been generated code for S2 has also not been generated. So, we really do not know what is the where the control should jump if this condition is true or if this condition is false. So, so, that way we need a 2 pass procedure; in the first pass we

can just note down the, that entity start addresses of all this code blocks and in the second pass we fill up all these gotos like this goto this B dot true.

So, B dot true will get defined only when S1 code has been generated. So, if this is the target 5 this is the intermediary code file where you have generating the code may be in this part I have got the code corresponding to S1 and in this part I have got the code corresponding to S2.

So, B dot true should be filled up with this particular offset and B 1 dot true sorry B dot true and B dot false should be filled up with this particular. So, these two addresses are offsets within the intermediary code file is not known, when I am doing this reduction. So, I have to make it a 2 pass procedure. So, to get to resolve this issue, so, we can make the intermediary code generation in a single pass by having the by modifying the grammar a bit and that modification is given by this grammar.

(Refer Slide Time: 04:01)

**Modified Grammar**

$B \rightarrow B \text{ or } MB$   
 $| B \text{ and } MB$   
 $| \text{not } B$   
 $| (B)$   
 $| \text{id rel op id}$   
 $| \text{true}$   
 $| \text{false}$   
 $M \rightarrow \epsilon$

$M$  is a dummy nonterminal with attribute  $M.\text{quad}$ , that can hold index of a quadruple

Consider the rule  $B \rightarrow B1 \text{ or } MB2$ :  
 Before the reduction of  $B2$  starts, reduction  $M \rightarrow \epsilon$  has already taken place. Hence,  $M.\text{quad}$  points to the index of the first quadruple of  $B2$

The slide includes a diagram showing a nonterminal  $B$  branching into  $B1$  and  $MB2$ , with  $M$  producing  $\epsilon$ . There are also logos for 'swayam' and a person's video feed in the bottom right corner.

So, where we have introduced in another non terminal B another nonterminal M; so, this M produces epsilon. So, if you replace this particular grammar in this particular grammar. So, if you replace this M by epsilon, so, you see that this M does not have any effect ok. So, there will it is the standard Boolean grammar that we have.

However, this M producing epsilon, so, this is this can be very much useful, because when I am doing this particular reduction reproducing B 1 or MB 2. So, the reduction is

like this B producing. So, here I have got the parts tree for B1 then or then M and then the parts tree for B2 and then this M produces epsilon.

Now when this; when this particular reduction will be done, so, when this when this B2 code will be generated. So, before that this M producing epsilon, so, this reduction will be done and M producing epsilon, so, it does not have any anything more. So, only thing.

So, at this time of reduction, so, if you look into the code the code that is generated, so, in that code file I will already have the code for B1. So, code for B1 is already there, now the code for B 2 should come after this the code for B 2 should come. So, in between, so, I need to know, what is this particular address this particular offset. And that can be found by the M producing epsilon production because once this is done. So, B 1 code has been generated. So, if I have something like say next quadruple number or next quadruple address that next quadruple address if you look if you query for that. So, would be getting this particular address from where the code for B 2 will be generated.

So, at this time of reduction, so, we will know where the next code will be generated. So, in that way we will know what is the code what is the start address for this the code B2. So, that will be utilised for generating the three address code in a single pass fashion. So, this is the thing that before reduction of B to starts the reduction of M producing epsilon has already taken place.

So, we have got an attribute M dot quad, so that can hold the index of a quadruple. So, that index is known; so, M dot quad points to the index of the first quadruple of B2. So, because I am so, if I ask you for the, if I somehow make the system. So, that it can return me the next quadruple index then that can be assigned to this M dot quad attribute and accordingly we can do something, so, that this code generation is proper. So, we will see how is it done.

(Refer Slide Time: 07:01)

### Translation Rules

**B → B1 or MB2**  
 { backpatch(B1.falselist, M.quad)  
 B.truelist = mergelist(B1.truelist, B2.truelist)  
 B.falselist = B2.falselist  
 }

**B → B1 and MB2**  
 { backpatch(B1.truelist, M.quad)  
 B.truelist = B2.falselist  
 B.falselist = mergelist(B1.falselist, B2.falselist)  
 }

**B → not B1**  
 { B.truelist = B1.falselist  
 B.falselist = B1.truelist  
 }

**B → (B1)**  
 { B.truelist = B1.truelist  
 B.falselist = B1.falselist  
 }

**B → true**  
 { B.truelist = makelist(nextquad())  
 emit('goto ...')  
 }

**B → false**  
 { B.falselist = makelist(nextquad())  
 emit('goto ...')  
 }

**B → id1 relop id2**  
 { B.truelist = nextquad()  
 B.falselist = nextquad()  
 emit('if id1.relop id2.place 'goto' ...)  
 emit('goto ...')  
 }

**M → ε**  
 { M.quad = nextquad() }

*Handwritten notes on slide:*  
 - In B → B1 and MB2, B2.falselist is crossed out and replaced with B2.truelist.  
 - In B → true, emit('goto ...') is annotated with (00) goto ...  
 - In B → false, emit('goto ...') is annotated with (00) goto ...

So, this is the modification of the grammar and the associated rule that we have got. So, first rule is for the or part so, this B producing B1 or MB2. So, as you know that by the short circuit because of you know that if B1 is true I do not have to evaluate B2, because this is an or but if B1 is false, so, I have to evaluate B2. So, what we do in B1s false list. So, if you remember, so, we said that this false list it has got this false list it has got the list of locations within the code of the B at which B is definitely false.

(Refer Slide Time: 07:41)

### Attributes

- **B.truelist**
  - List of locations within the generated code for B, at which B definitely true
  - Once defined, all these points should transfer control to B.true
- **B.falselist**
  - List of locations within the generated code for B, at which B definitely false
  - Once defined, all these points should transfer control to B.falselist

So, so, this way it is basically, so, this B dot false list is the points at which B1 is definitely false. So, that that place so, we will be back patching with this M dot quad. So, M dot quad has got the start address of B2 so, they will be filled up with the start address of B2. So, that will be the backpack and for getting the true list that is true list is the attribute that has got all the points for the code for B at which B is definitely true.

So, B dot true list is it is created by merging B1 true list and B2 true list because these are all the points at which the expression B is definitely true and B dot false list is equal to B2 dot false list because these are the positions at which the overall expression B becomes false.

So, that we can modify the three address code generation statements the syntax directed translation mechanism so, that the locations are corrected. Similarly, if you look into this B producing B1 and MB2, so, there by the short circuit principal we know that if B1 is false we do not have to do anything, but if B1 is true we need to evaluate B2 also and B2 is start address or the start quadruple index is available in M dot quad.

So, that way it is doing this thing that back patch B1 true list with M dot quad. So, that way so, all that points at which B1 is true, so, there this M dot quad is put. So, that we can we can jump to the evaluation of B2. Bs true list is equal to be true false list. So, so, if sorry this is a mistake. So, this should be true list not false list so, should be true list. So, B true list equal to B2 true list because for the overall expression B.

So, this is true this true list will be wherever B2 is true. So, that that way it is going to be; going to be true that because at this point both B1 and B2 are truth. And Bs false list is either B1 is false or B2 is false. So, it is a merge merger of these two list B1s false list and B2s false list. So, by merging them we get the false list for B.

Now, not of B1 so, B2 list equal to B1 false list and B false list equal to B1 true list. So, it is just swap the locations at which B1 was true with the locations at which B1 was false to get the false and true locations for B. B within bracket B 1 so, this is nothing, but the true list will be copied to B true list and false list B1s false list should be copied to Bs false list.

Now B producing true, so, this is I need to generate a code because there I have to generate a goto, this goto has to be generated because at this point this definition is this

Boolean expression B is definitely true so it generates a goto at some point ok. So, this value is unknown at present so, it will be filled up later. But this index, so, this three address code are being generated, so, what is the index of this particular line?

So, that is available by calling this function next quad ok. So, before generating this codes. So, before emitting this code or generating this code we are calling the function next quad. So, naturally next quad will return me the index of this particular quadruple and that is passed to the function make list. So, if this number is say 100, then it will make a list with 100 as 1 as an one entry and B dot true list is made to point to this. So, that will identify that if I am looking for B dot true list. So, this is the location where B is definitely true.

Similarly, so, this id1 relop id2; so, this is interesting. So, this is we B dot true list is next quad B dot false list is next quad then we generate the, if then else for the true and false part. So, it is relational operator may be any of the greater than less than those symbols.

(Refer Slide Time: 12:03)

**Translation Rules**

- $B \rightarrow B1 \text{ or } B2$ 

```
{ backpatch(B1.falselist, M.quad)
  B.truelist = mergelist(B1.truelist, B2.truelist)
  B.falselist = B2.falselist
}
```
- $B \rightarrow B1 \text{ and } B2$ 

```
{ backpatch(B1.truelist, M.quad)
  B.truelist = B2.truelist
  B.falselist = mergelist(B1.falselist, B2.falselist)
}
```
- $B \rightarrow \text{not } B1$ 

```
{ B.truelist = B1.falselist
  B.falselist = B1.truelist
}
```
- $B \rightarrow (B1)$ 

```
{ B.truelist = B1.truelist
  B.falselist = B1.falselist
}
```
- $B \rightarrow \text{true}$ 

```
{ B.truelist = makelist(nextquad())
  emit('goto' ...)
}
```
- $B \rightarrow \text{false}$ 

```
{ B.falselist = makelist(nextquad())
  emit('goto' ...)
}
```
- $M \rightarrow \epsilon$ 

```
{ M.quad = nextquad() }
```
- $B \rightarrow id1 \text{ relop } id2$ 

```
{ B.truelist = nextquad()
  B.falselist = nextquad()
  emit('if id1 place relop id2 place 'goto' ...)
  emit('goto' ...)
}
```

Handwritten notes on the slide include '100' and '101' pointing to 'goto' labels, and a diagram showing a list structure.

So, if id1 dot place is of relational operator id2 dot place. So, if this expression goes to be happens to be true ok, if id1 dot place is say relop is greater is greater than id dot place then go to. So, this part is not known, so, this part is there and then it generous this. So, before doing this so, you see that we have called is next quad function twice once to get the quadruple index for true list and ones to get the quadruple list for false list.

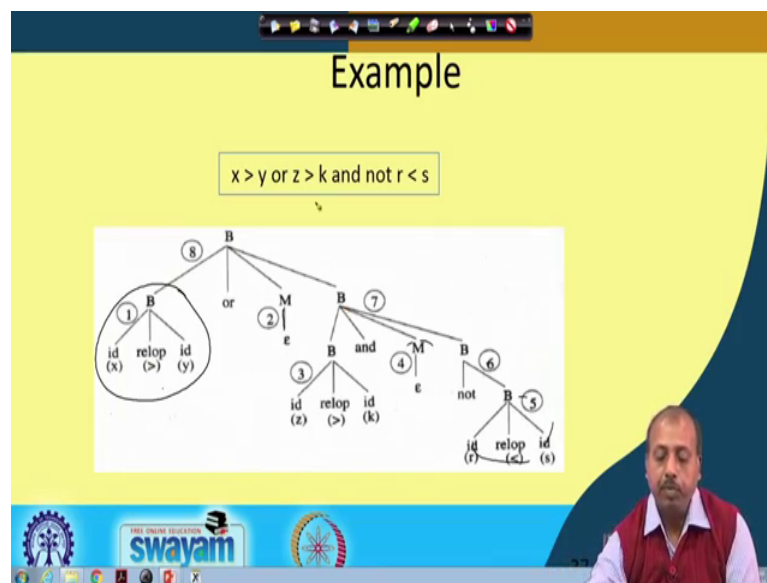
So, at present suppose I am at suppose I have generated code up to index say 99 up to this much has been generated. So, at this point if you call next quad, so, this B dot true list is becoming equal to 100 and B dot false list is become equal to 101 fine. Now it generates the code at 100 it generates the code that if etcetera etcetera that goto that part is generated and then at 101 it generates goto. So, this is nothing, but the if then else execution.

So, if id1 dot place is relationally operator having the relational operator id2 dot place then it will generate then it will goto this thing. So, this is the true list. So, this 100 happens to be the true list of B and 101 happens to be the false list of B. Later on, when this Boolean expressions target will be known like if it is a part of this if then else statement or while statement like that so, target will be known at that time we will be filling up this goto this positions with some backpatching procedure. So, this will be clear when we goto some statement like if then else or while loop like that.

Similarly, this B dot false list is just the replica of B dot B producing false is a replica of B producing true and so only thing is that instead of true list. So, we do a false list here and B dot false list is make list because this B is always false ok, so, there is no true parts. So, there is no true list true list is null and false list is equal to this, then M producing epsilon so, M dot quad equal to next quad. So, these are the functions that we have in the translation rules. Now how are you going to use this translation rules for generating some code so, that we will see.



(Refer Slide Time: 14:35)



So, suppose this is the; this is the Boolean expression that we have a  $x$  greater than  $y$  or  $z$  greater than  $k$  and not of  $r$  less than  $s$ . So, this is the parts tree, that is produced and if we number the reduction. So, first this reduction will be made then this  $M$  producing epsilon will be done, then it can do this reduction. So, that is number 3, then this can be done that is number 4, then this can be done number 5, then the 6, then this 7. So, you see that before this 7th reduction is made by this time is  $M$  producing epsilon has already taken place.

So, and for this part for this part of the Boolean expression the code has already been generated. So, after this the code for evaluating this second part will be there and this  $M$  dot quad will definitely have the corresponding quadruple index for the second part.

So, let us see how this code is being generated ok. So, initially so,  $B$  dot true list so at reduction number 1 so,  $id$  relop  $id$ .

(Refer Slide Time: 15:35)

Translation Example (Contd.)		
Reduction	Action	Code generated
1	B.trueList = {1} B.falseList = {2}	1: if x > y goto ... 2: goto ...
2	M.quad = 3	
3	B.trueList = {3}, B.falseList = {4}	3: if z > k goto ... 4: goto ...
4	M.quad = 5	
5	B.trueList = {5} B.falseList = {6}	5: if r > s goto ... 6: goto ...
6	B.trueList = {6}, B.falseList = {5}	
7	Backpatches list {3} with 5	3: if z > k goto 5
8	Backpatches list {2} with 3 B.trueList = {1, 6}, B.falseList = {4, 5}	2: goto 3

**Full Code:**  
 1: if x > y goto ...  
 2: goto 3  
 3: if z > k goto 5  
 4: goto ...  
 5: if r < s goto ...  
 6: goto ...

1, 6 true exit,  
4, 5 false exit

So, if you look into this group. So, first it gets to next quad and that that are assigned to B dot true list and B dot false list. So, that is what is exactly done here. So, B dot true list is it is by calling the function next quad it will get 1 and B dot false list by getting the calling the function next quad we will get 2 and it will generate 2 lines of code as it is said by this emit statement. So, if id1 dot place etcetera. So, it generates this code that if x greater than y goto this is not known now and this is goto not known now and true list and false list for the B is kept as 1 and 2.

Now, for reduction number 2, so, this M producing epsilon. So, M dot quad equal to next quad that is the semantic action and M dot quad is equal to next quad. So, already we have generated 2 quadruple, so, next quad is at 3 fine. Now comes the reduction number 4. So, reduction number 4 is sorry reduction number 3. So, that is again another if there is id relop id. So, it will generate true list and false list next quad.

So, B dot true list equal to 3 dot false list equal to 4 and it will generate these two three address code like if z greater than k goto and 4 at 4 goto. So, this part is generated then this M dot quad equal to then this redetection number 4. So, M producing epsilon, so, M dot quad gets 5, because that is the next quadruple index. Then it will do reduction number 5. So, at this reduction number 5, so, again this is an if then else. So, 2 true list and false list will be created by calling the function next quad twice.

So, that is done here. So, you get the true list as 5 and the false list as 6. So, at it generates the code at offset 5 and 6 like if r greater than Sgoto and at 6 goto. And then at reduction number 6 not of B so, you know that true list and false list they will get interchange. So, no code will be generated, but this true list becomes equal to B1s false list so, that is 6 and these false list is equal to B1s true list that is equal to 5. Now I know that line now at reduction number 7 ok, reduction number 7 is an and operations. So, for this and you see first it says backpatched B1 true list with M dot quad ok.

So, B1 true list so, this at whatever at production number 3 whatever was the true list. So, that will be backpatched with the reduction number force M dot quad. So, so, backpatches list 3 with 5, so, this is the reduction number 4. So, this is the true list is having 3. So, at location 3, it will back patch with the value 5. So, it will be correcting this.

So, this value was previously unknown and now this value will be corrected and you will get a 5 here ok, so, that will happen. And when it says that so, that is so, in B1 and so, that so, that was a reduction number 7 you are looking into.

So, and so, and there is a. So, B dot true list is merger of B1 true list and B2 true list. So, that way this B dot true list is made B1 true list and B 2 true list. So, those true list will be merged and that will be the true list of B. So, then it will then it comes to reduction number 7 so, that B1, so, with this B and MB2 and after that reduction number 8. So, in reduction number 8 so, it says backpatch list 2 with 3 and B dot true list is 1 6 and B dot false list is 4 5. So, this is created by merging the thing like at 1 and 5 whatever were the false list.

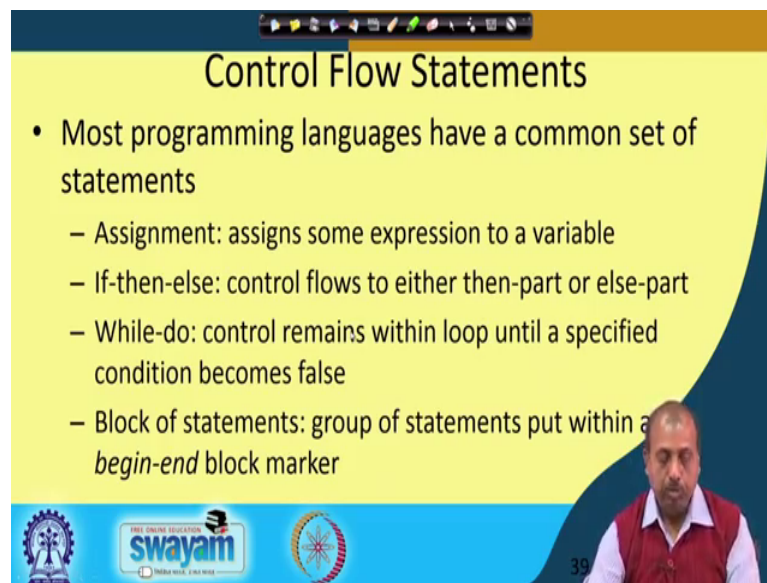
So, whatever that 1 and 6 this true list. So, they are merged getting 1 6 and the false list is equal to this 4 5 actually this merger is not shown here, this B and MB2 that at that at reduction number 7 there is a merger. So, so, this B dot true list. So, this and B, B dot true list is B2 for true list and B false list is merger list of 2 false list.

So, for this it will be 5 3 and 3 6, so, these 2 true lists are to be merged 3 and 6. So, these 3 is 3 is this 1 and 6 is this one. So, 3 and 6 so, they will be merged. So, that a result this true list will be 3 6 and false list will be B2 is false list that is 5 ok. So, in the next step, it will be doing the final reduction and it will be generating this code.

So, at the end this one and so, this is the code that is generated by backpatching say this location and these two backpatches have taken place. And then it the final true list will be 1 6 will be the true exit for the whole expression B and 4 5 false exit for the whole expression.

So, we will be taking more examples and it will be clear further. So, you just also practice something. So, we will also do some more exercises that will make it more clear.

(Refer Slide Time: 21:53)



The slide is titled "Control Flow Statements" in a large, bold, black font. Below the title, there is a bulleted list of common programming statements. The first bullet point is "Most programming languages have a common set of statements". The subsequent four bullet points are: "Assignment: assigns some expression to a variable", "If-then-else: control flows to either then-part or else-part", "While-do: control remains within loop until a specified condition becomes false", and "Block of statements: group of statements put within a *begin-end* block marker". The slide has a yellow background with a blue border on the right side. At the bottom, there are logos for "swayam" and "MOOCs" (Massive Open Online Courses). A small video inset in the bottom right corner shows a man with a beard and a red vest speaking.

- Most programming languages have a common set of statements
  - Assignment: assigns some expression to a variable
  - If-then-else: control flows to either then-part or else-part
  - While-do: control remains within loop until a specified condition becomes false
  - Block of statements: group of statements put within a *begin-end* block marker

So, next we will be looking into the control flow statements like how do you have this control flows like if then else when while block then assignments. So, these statements how for them how are you going to generate the three address code. So, most programming languages they have a common set of statements like assignment, if then else while do and block of statements if they many a time some time will have got the begin and marker block marker sometime we have open brace close brace markers.

So, like that so, we have got different block markers. So, based on that the blocks are made. So, how to generate code for this type of statement?

(Refer Slide Time: 22:33)

**Grammar**

$S \rightarrow \text{if } B \text{ then } M S$   
|  $\text{if } B \text{ then } M S N \text{ else } M S$   
|  $\text{while } M B \text{ do } M S$   
|  $\text{begin } L \text{ end}$   
|  $A \text{ /* for assignment */}$   
 $L \rightarrow L M S$   
|  $S$   
 $M \rightarrow \epsilon$   
 $N \rightarrow \epsilon$

**Attributes:**

- $S.\text{nextlist}$ : list of quadruples containing jumps to the quadruple following  $S$
- $L.\text{nextlist}$ : Same as  $S.\text{nextlist}$  for a group of statements

Nonterminal  $N$  enables to generate a jump after the *then*-part of *if-then-else* statement.  
 $N.\text{nextlist}$  holds the quadruple number for this statement

*Diagram:* A control flow graph showing a box labeled  $S_1$  with a 'goto' arrow pointing to a box labeled  $S_2$ . There is also a self-loop arrow on  $S_1$ .

So, the grammar that we will be considering is something like this that the statement it can be an if then statement if then else statement while statement a block of statement or it can be an assignment ok. So, it is like this that statement is if Boolean expression then  $M$ . So, this  $M$  is introduced for the same purpose that we have seen previously for as a placeholder so, that we can generate the next quadruple index.

So, here in this, when so, when this reduction is being made so, by that time. So, we will be knowing the address for the starting quadruple address of  $S$  as results in  $B$  dot true list, so, we can put that particular quadruple number. So, it will be clear as we look into some example.

Similarly, if  $B$  then  $MSN$  else  $M S$ . So, this is so, if some Boolean expression  $B$  is true then it will come to this statement execution and that quadruple can be found by the  $M$  dot quad. And this so, this is introduced to have a goto after the then part. So, basically if this is the code for the then part that is the  $S_1$  and after this I am putting the code for  $S_2$  so, in between there should be a goto statement.

So, that should be goto otherwise what will happen if the condition is true. So, it will start executed at this point it will execute  $S_1$  then it will go into execution of  $S_2$ , but you do not want that. So, if  $S_1$  is executed after that the control should come out and it should come to this point how is it should jump over the  $S_2$ . So, for ensuring that so,

this N non terminal is introduced and we will see that it will generate a next it will generate a goto statement accordingly.

So, if B then M S N else M S similarly the while loop is modified while M B do M S and then for the block of statements so, this is begin and end. So, these are the block markers and this L is the list of statements and A is a simple assignment statement. So, and L for L that block of statement, so, it can be a single statement or it can be a null sequence of statements. So, this particular rule this two rules. So, they will be capturing that situation.

So, L producing L M S or S and the standard reduction like M producing epsilon, an N producing epsilon. So, they are there. So, we have got the attributes like S dot next list. So, that is that will have the list of quadruples containing jump to the quadruple following S ok. So, it will have the. So, jumps to the, so, it will have the list of quadruples where I have to put the next value of S after executing S what next where the control should go next.

So, at all the places so, in the block of S so, you may need to (Refer Time: 25:37) there may be a various points at which you need to tell like where what to do next. So, what will be the next quadruple entries? So, that so, those are all these locations they will be kept in S dot next list as and when that target will get define what we will do we will backtracks this value at this places so, that the generated code will be corrected.

And this L dot next list. So, this is again same as S dot next list expecting that this is for a group of statements so, we want to do that. So, that is L dot list and this nonterminal N as I discussed. So, it enables to generate a jump after the then part of the if then else statement and N dot next list it holds the quadruple number for this particular statement ok.

So, for this statement that is the goto statement. So, we will see the rules for generating the code.

(Refer Slide Time: 26:35)

**Translation Rules**

```
S → if B then M S1
{
  backpatch(B.truelist, M.quad)
  S.nextlist = mergelist(B.falselist, S1.nextlist)
}
```

```
S → if B then M1 S1 N else M2 S2
{
  backpatch(B.truelist, M1.quad)
  backpatch(B.falselist, M2.quad)
  S.nextlist = mergelist(S1.nextlist,
    mergelist(N.nextlist, S2.nextlist))
}
```

```
S → while M1 B do M2 S1
{
  backpatch(S1.nextlist, M1.quad)
  backpatch(B.truelist, M2.quad)
  S.nextlist = B.falselist
  emit('goto' M1.quad)
}
```

Handwritten annotations on the right side of the slide include a control flow graph with nodes labeled 'B', 'M1', 'M2', and 'S1'. Arrows indicate the flow of execution. A box labeled 'B' contains 'B.truelist' and 'B.falselist'. A box labeled 'M1' contains 'M1.quad'. A box labeled 'M2' contains 'M2.quad'. A box labeled 'S1' contains 'S1.nextlist'. The graph shows a loop structure where the 'B' node branches to 'M1' if true and to 'M2' if false. 'M1' branches back to 'B', and 'M2' branches to 'S1'. 'S1' branches back to 'B'. Handwritten notes include 'B.truelist', 'B.falselist', 'M1.quad', and 'M2.quad'.

So, first one; so, if the if then statement, so, if B then MS1. So, what we will have to do is that. So, so, in this particular reduction is being made you see that I have got this situation if B then M and S1. So, by this time this B has already been reduced and we have got with associated with this B we have got B dot true list and B dot false list.

As we have seen in the Boolean expression conversion. So, whenever we have got have done this reduction, so, B dot true list is having all those places where the expression B is definitely true and B dot false list are all those places where the expression B is definitely false. So, this is already available. Now M producing epsilon so, that was there. So, we have only already we already know what is the next address of this after this where the code of S 1 will be there.

So, and then this S1 has already been the code for S1 has already been generated and that start of set is available in M dot quad ok. So, in the file I have got the code for B I have got the code for S1 these are available and then and M dot quad actually having this particular index M dot quad is having this particular index.

So, what I can do is that at all the places where B is true. So, they can be connected with they can be backpatched with his M dot quad. So, there we had the goto part and goto left it the target I will leave the target as blank, now all those targets can be filled up with this M dot quad which is the start address of this S 1. So, we have goto somewhere.

So all these goto that were there in the code of the B where B is definitely true by consulting this B dot true list so, we can backpatch everything of this B dot true list due to M dot quad. Similarly now for false of course, I want to skip over this S1 fine.

So, for that for this statement S I do not know what is the next statement. So, so, when that next statement will get defined then I have to rectify all the places. So, what are the places I need to rectify? So, wherever B is false then I will be skipping over this S1. So, if somehow this address becomes known this offset becomes known then wherever B is false. So, there should be filled up with this particular quadruple index.

And similarly at within S1 also there maybe places where I need to generate that I want to the next statement. So, there also I will be filling up with this star. So, this is S dot next list is merge list of B false list and S 1 next list. So, we will continue with this discussion in the next class.