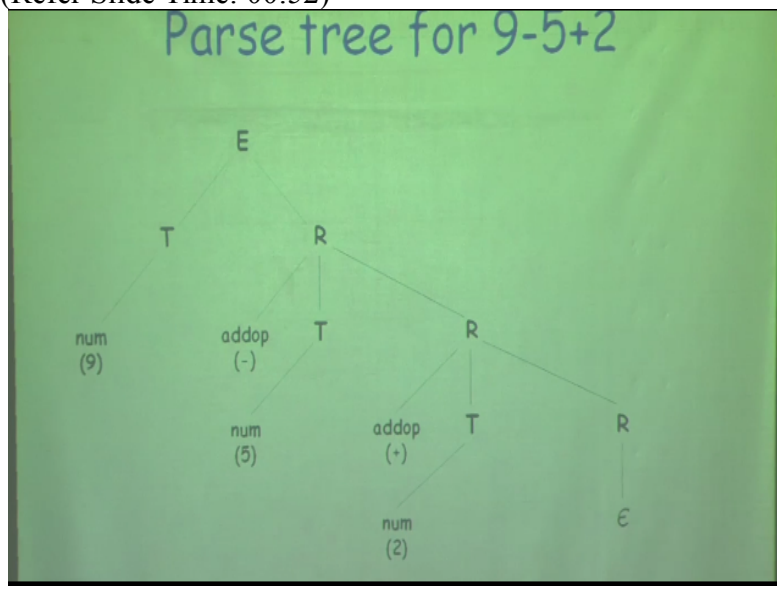
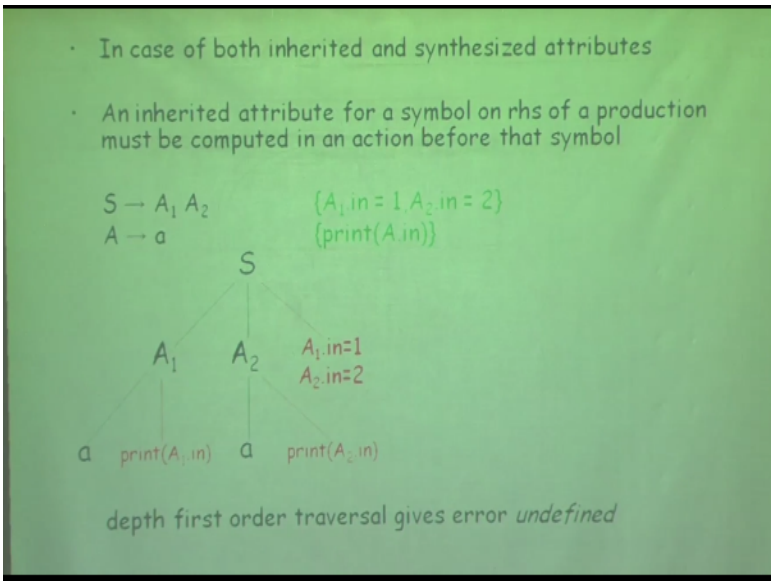
Indian Institute of Technology
Kanpur
NP-TEL
National Programme
on
Technology Enhanced Learning
Course Title
Compiler Design
Lecture-17

by…
Prof. S.K. Aggarwal.
Dept. of Computer Science and Engineering.

Good afternoon so quickly to recapture what we were discussing towards the previous class we started looking at L attributed definitions and once we introduced L attributed definitions we also introduced translation schemes where we said that the code need not be the end of any rule and we can embed the code anywhere in the right-hand side okay and then we looked at positioning of the port and we said that this port should be evaluated in a position.
(Refer Slide Time: 00:52)

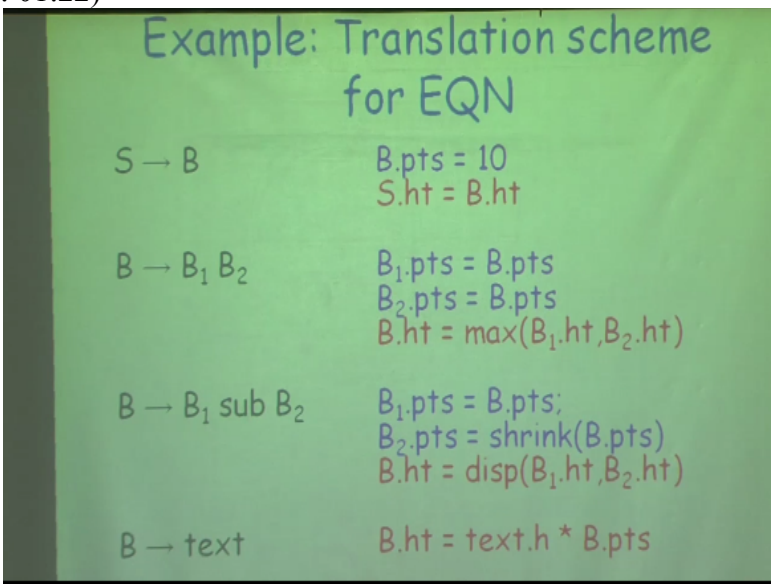


So that we have all the attributes which are required to evaluate it and a general position we found was that for inherited attributes the code must be evaluated in the immediately of the left of the grammar symbol and for synthesized attributes the code can be evaluated at the end of the rule this is what we were discussing and then we took an example this example came from so these were examples.
(Refer Slide Time: 01:14)



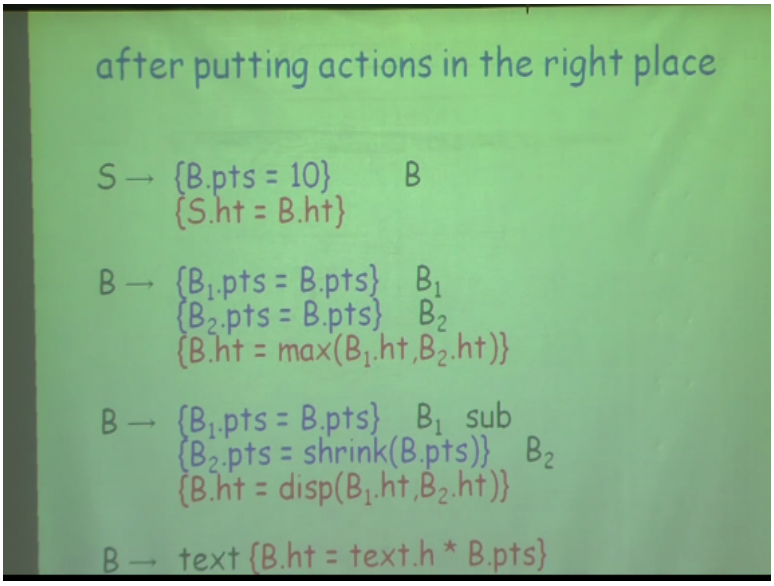
We were discussing for the placement of attribute rules and then we took an example from Equation typesetter.

(Refer Slide Time: 01:22)



And introducing the rules here we were trying to position these rules so that value of these attributes get evaluated in the right place and then we looked at the placement of these and this turned out to be the placement.

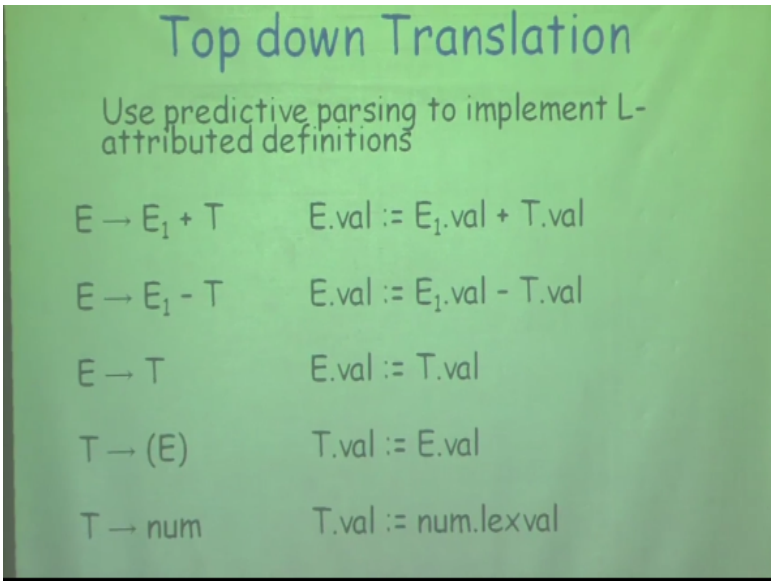
(Refer Slide Time: 01:31)



Where we said that each of the inherited activity is getting evaluated on the immediately left off the grammar symbol .So this is what we were doing and once again just to set the tone right for the discussion we had two parsing methods and two kind of attributes synthesized and inherited attributes and two parsing methods which are top-down and bottom-up and we saw that at least for bottom up parser and synthesized attribute we could always evaluate all the attributes while we were building the parse tree or doing the parsing and we also found that as far as L attributed definitions were concerned .

We could also use a top-down parser and could finish off evaluation of all the L attributed definitions right two things which we have not managed so far that how do we evaluate translation schemes and the most critical problem was that how do we use a top-down parser along with synthesized attributes and how do you use a bottom up parser along with the inherited attributes these two problems were still not clear so everyone is sort of is able to recall what we were doing is that the same footing we can start over discussion from here okay.

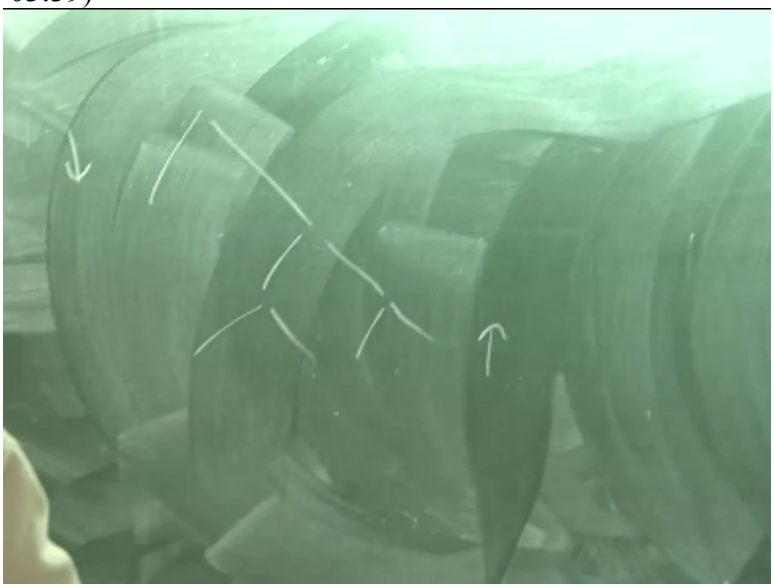
(Refer Slide Time: 02:52)



So first problem that comes is that when I have a top-down parser and I have some embedded rules rules could be embedded anywhere or they could be on the right-hand side also so first thing I need to do is in a top-down parser that I need to worry about whether I have problem of left recursion or not okay so here is an example where I have a grammar I have some attributes which are synthesized attributes and I want to write a top-down parser for this so there is a first problem I am taking okay .

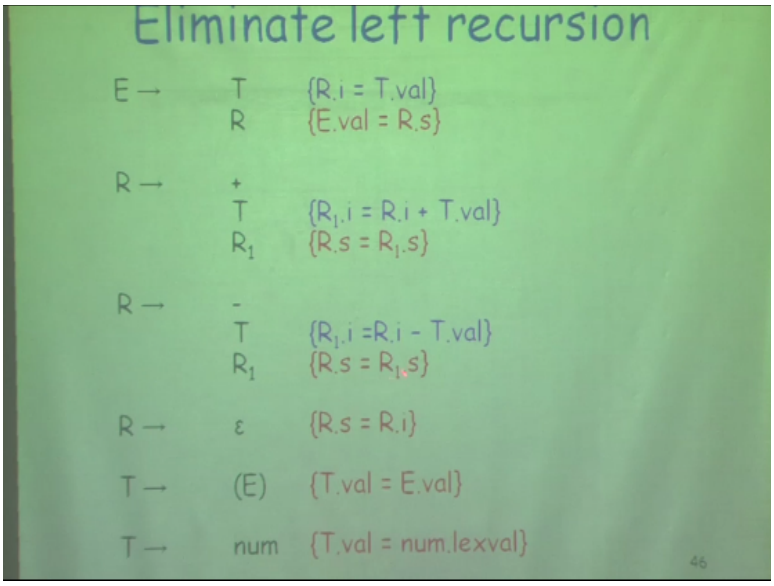
That we have a top-down parser and we have synthesized attributes so first problem that will come in top-down parser is that I start building my tree from top to bottom.

(Refer Slide Time: 03:39)



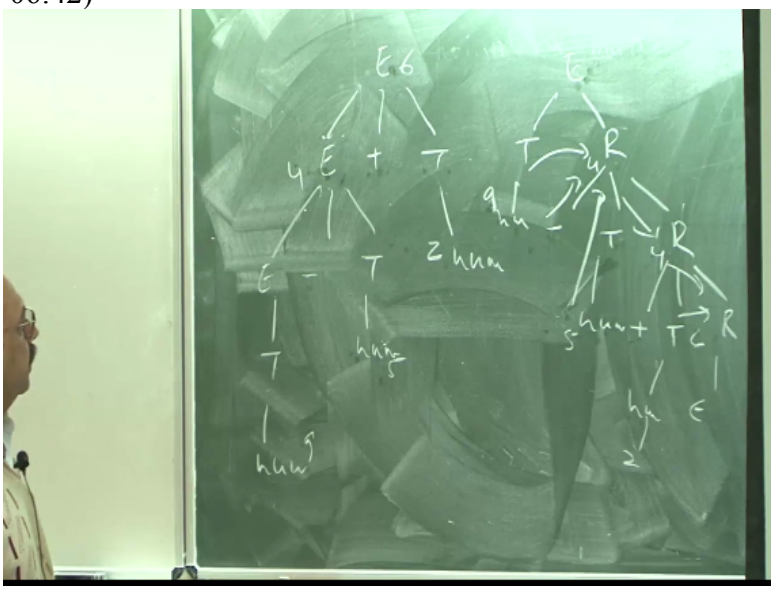
I will see that somehow if you are talking of synthesized attributes synthesized attributes will go in this direction and my parser will go in this direction how do I handle this situation okay so

here is a typical situation where you have a rule set of rules which are left recursive and then you have a set of synthesized attribute equations or s attributed equations right.
So how do I solve this problem first any ideas anything or can this problem be solved first let me ask this question and this problem is solved but the problem is just not soluble in therefore there is no point attempting even to try to find the solution anything now we just start thinking aloud so again what should come to your mind is L attributed definitions okay now suppose I say that I am trying to build this parse tree.
I am saying that I am using synthesized attributes now when I build this part of the parse tree okay remember that when I do top-down parsing this part will be built before this part is built that means it is possible at least that if I evaluate all the attributes in this part and I want to evaluate certain attribute equations then I can take the value on the right hand side okay so basically what is happening is look at it this way that because of the limitation we have put on the attributes now we only want to deal with L attributed definition.
So L attributed definitions will have both inherited as well as synthesized attributes so if somehow I change my synthesized definition to L attributed definitions then I can do a top-down parsing right suppose all these synthesized attributes get translated into L attributed definitions then I can use a top-down parser to crack this problem okay now the way I want to do this translation is I will first take an example.
And then I will give you a general rule basically conceptually this is saying that since I have built this part of the parse tree first evaluate all the attributes here and then somehow take them on the right hand side so first problem let us focus on the grammar and forget about the attribute part and then we will see how to change all these attribute equations from s attributed definition to L attributed definitions so it is a reverse transformation right .
So here is a grammar if I remove first left recursion from this grammar because this says e goes to e + D and E goes to e - T I cannot handle left recursive grammars as far as top-down parsers are concerned so I need to remove left recursion and if I remove left recursion this is how the grammar will look.
(Refer Slide Time: 06:18)



I have introduced now a new non terminal R and this e goes to TR and R goes to + TR and R goes to -TR and R goes to Epsilon okay and these two rules remain as it is because there is no left recursion there .Now if I compare the two grammars okay how will my parse tree look okay so let us look at this grammar this grammar will say that.

(Refer Slide Time: 06:42)



I will go down into something like this thing that e goes to e plus T and then e goes to e -T and then e can go to T and T can go to num and okay standard technique for removal of the left recursion is I now say that e goes to TR and then R goes to let us say - TR and then this e goes to num and this goes to +TR and this is how the parse tree will look okay now if I say that I want to use certain definitions to find out this value which are synthesized attributes it is equivalent to saying that if the same value becomes available here .

And then if I now use instead of only synthesized attributes L attributed definitions then I can solve this problem but how do I convert that now I say that since I am evaluating from left to right that means when I say that I want to know the value of this e I must know this value and this value now similarly if I say that if I am at this point I must know this I must know the operator and I must know this value then.

I can find out what is the value corresponding to this part of the tree and once I know this then knowing this value and this operator I can find out this value which I am saying that knowing this value this operator in this value I can find out the same thing making sense what I am saying or not at all just not sinking in explain what asked the question because that is a very generic statement explain it again I can do it ten times but that still does not solve the problem.

So look at it this way yeah let us suppose I have this value so let me just put some numbers here okay so suppose this number is 9 this number is 5 this number is 2 and I want what I want is I want value of this expression to be available at this point right so what will be the value you will take this value this operator and this value evaluate it and we will say this value is 4 and therefore this value becomes 4and then you will say I know this value I know this operator I know this value and therefore this value becomes six right this how you will do it okay.

So it is basically a left to right evaluation nothing else okay now I can do the same left to right evaluation here if I say this value is nine and this value is five I know the operator here somehow if I say that if I carry this value on the right hand side and then use this operator and use this value and associate that value at let us say this R okay so what will the value I will get for 9-5 somehow if I take this value here I am taking this value here and then using this information here and this information here I can do this computation okay I will get the same value somehow.

So let us not worry about how do I change my equations that I will show you systematically do but basically you are saying that if I know this and this I can compute something here then I take it down and then say now I know this and this and therefore I can recomputed this value there right so if I say that I have nine minus five which gives me four and then I take four down here and now I say I know four here I know plus and I know of what is the number here 2 then I can recomputed this value here and somehow take this value .

Now you can see that what I am using here is RL attributed definitions here so this value can then become fixed here only problem is value is not available to me still at this point value becomes available at the rightmost node rightmost bottom node right now somehow I need to carry it back okay so now if I say that I have both synthesized and inherited attribute if I just can

make a copy here and then I can take this value back okay and this is precisely what I want to do okay.

So let me first take an example and then see what I am doing here so let us look at the rules here these rules nothing is changing because I have no left recursion here I just copy them as it is but now what I am doing here is now I am saying that the new non-terminal I have introduced corresponding to that I have an inherited attribute which says R inherited is T value I am also putting the right places for this so basically now I am saying that this node which is the new one which has been introduced this will now copy into its inherited attribute a value which is value of T value right.

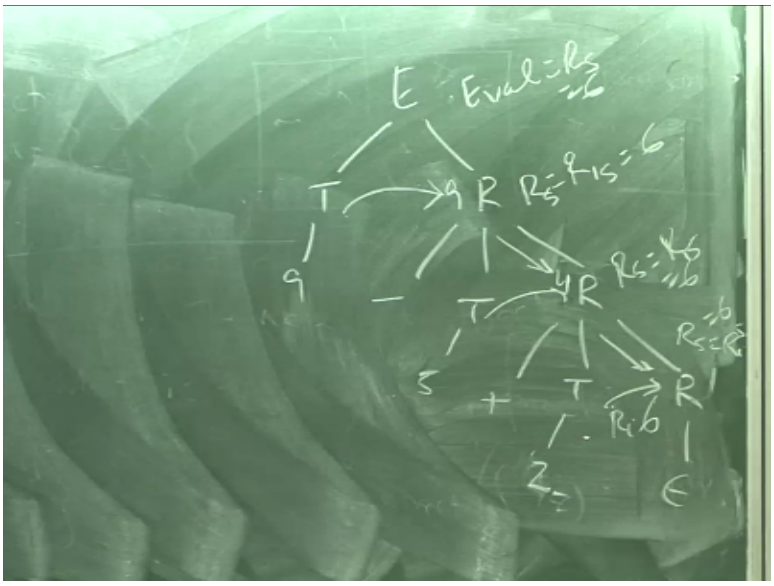
So this value gets copied from the left sub-tree to the right sub tree and then at this node now I say when R goes to + TR then corresponding to this R I am now computing an attribute which is in terms of this and this now you can see that what I am doing here is that if I now look at this information if I want to compute a value here okay how do i compute this value I am looking at the operator in the left sibling and I am looking at a value of the parent and using this information .

So basically my left siblings there are this number this operator and this value this is all coming from my L attributed definitions I can compute this value so if I now say R1 inherited which is this R1 gets a value which is R inherited +T value now + you have to take as an operator okay in this case it happens to be plus it could be anything okay now if I evaluate this what will happen now we will say that this value which was 9 here which I copied 9 -5 gives me 4 here and then if I look at this equation this equation says R1 inherited is R inherited - T value okay .

So in this case yeah so actually it is 9-5 so okay so let me have both these rules so this was - this was 9 5 this gives me 4 and now I say that I am looking at R inherited here this says I am taking this value and I am taking value from the left child and this gives me 2and 4 and this gives me value 6 right only except issue remains that this value 6 is available here and not here I want e value to be 6 and not R inherited to be 6 okay so how do I take this value up now I can actually do a very simple transformation .

Because now I am saying that R is going to epsilon and R can have both inherited and synthesized attributes now I say that R synthesized is nothing but a copy of R inherited okay so if I do that what will happen then I will say that can you see this part from the back you cannot okay why don't you point it out.

(Refer Slide Time: 14:03)



I let me draw this here this is all your trees right can you see this from this corner okay. So now when I take this value first this value just gets copied here okay and now I say that I want to compute inherited attribute in terms of this and this attribute is coming from the parent this attribute is coming from the Left sibling and therefore I get a value which is 4 here and then I say that now I am computing a value which is coming from the parent and a value which is coming from the sibling and I get a value which is 9-5 what is the equation I was using here 2 so this should be 2 here.

So this gives me 6 okay now I say that R synthesized is nothing but a copy of R inherited yeah so if I do that then I can say that this was value of RI and now R s here becomes RI and what will be the value of RS 6 right and now I can say this R synthesized is the copy of this R synthesized so this gives me R 1 s which becomes 6 and then I can use the same rule here which says R synthesized is R1 synthesized which is 6 and then I can say e value is R synthesized which is 6 and that is what I wanted okay.

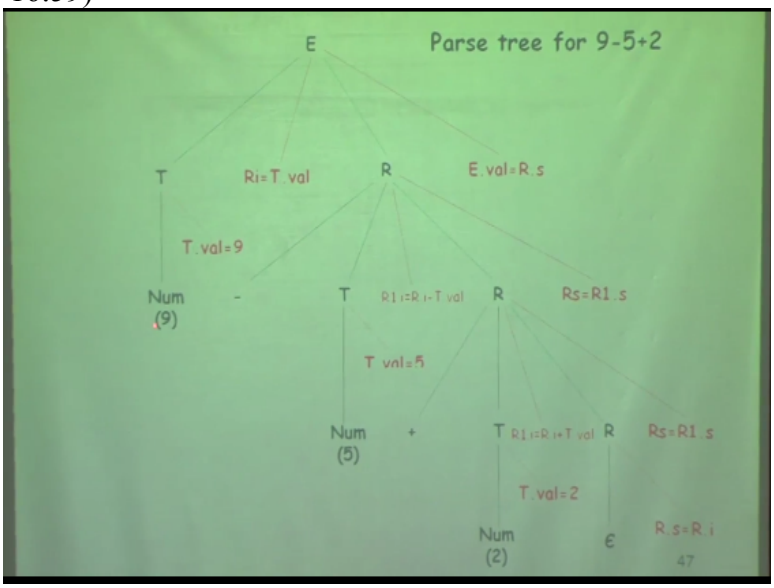
So this is what we do here that we say that whenever I do this at silent reduction then I am saying R synthesized is nothing but a copy of R inherited and here I say that R synthesized is a copy of R 1 synthesized and the same rule I will use here which says R synthesized is a copy of R1 synthesized and e value is nothing but RS okay now when I finish parsing this and simultaneously I evaluate all the attribute equations.

What will be the e value the e value will be the correct value which is 6 okay and this is what I wanted right now you can see that what I have is I have taken a parser which only had s attributed definitions converted that into L attributed definitions so I can use now a top-down

parser this is doable with top-down parser how will I use the top-down parser for this okay you can see that all my attribute equations are in the right place.

So I can do the computation without violating any dependencies okay and how will the parse tree look for this is how the parse tree will look without the rules and if I never introduce nodes corresponding to all the rules this is how the nodes will look right.

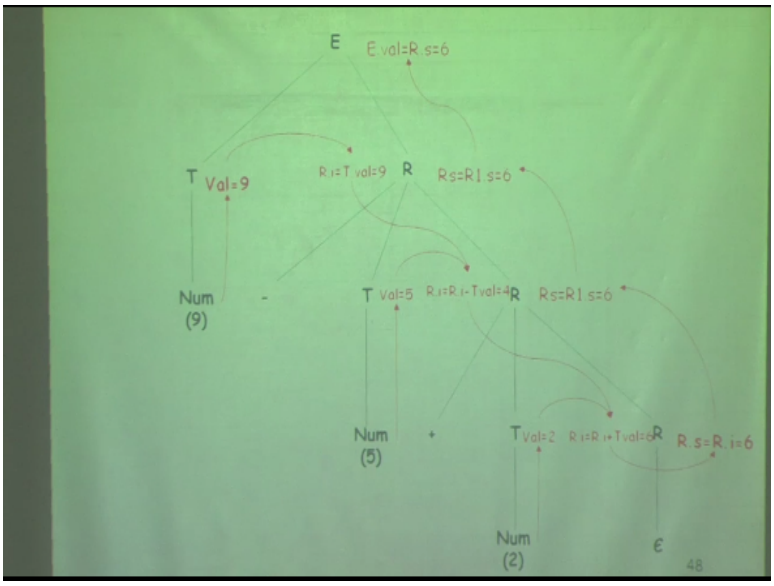
(Refer Slide Time: 16:59)



So e goes to num followed by T value basically this is what you have to remember T goes to num followed by T value and just put rules at the right places so R goes to - T this rule r1 followed by this rule okay so this is what it is R goes to - T R and this rule okay so this is how I put all these values and now if I evaluate all these in the depth-first traversal order whenever I encounter a rule I just execute it what will be the last rule to be evaluated this will be the last tool to be evaluated and what will be the e value the right value whatever is the right value of this expression okay .

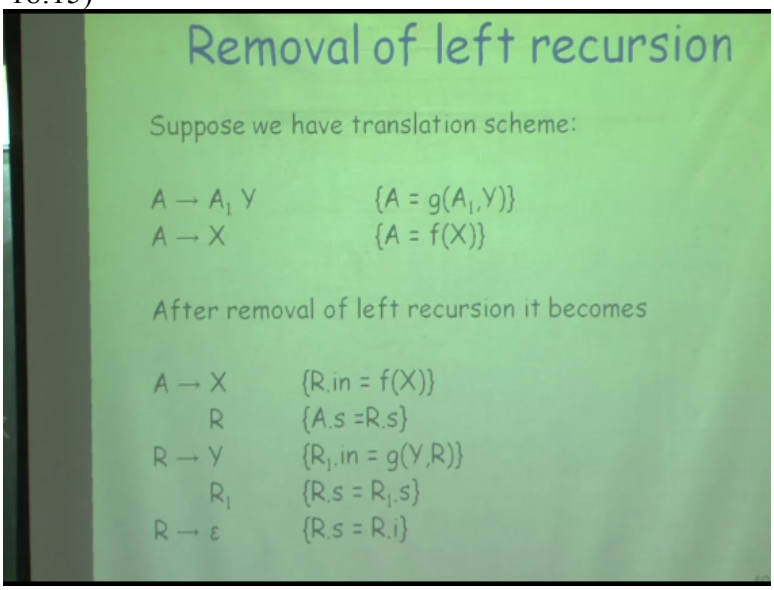
So you can see that if I am given synthesized attributes I need a method so I did not still give you a systematic method of converting all the S attributed definitions into an L attributed definition okay but it is doable at least by this example we are able to see that we can do it yeah now let us look at a more okay .

(Refer Slide Time: 18:05)



So this is just giving you execution place of the same thing okay this is how it is traversing so finally I get value of synthesized attribute here and then I am just carrying it to the top okay.

(Refer Slide Time: 18:15)



But this is really what happens okay just focus on this rule okay where you have a left recursion and you have synthesized definitions and what I want to do is I want to convert this into L attributed definitions so when I put a L attributed definitions first thing that will happen is that L attributed definitions will introduce now a new symbol and then I am saying that R first must-carry R inherited must be a function of whatever is on the left hand side write A synthesized in this case is nothing but R synthesized .

And whenever you have a rule like this all we are saying is that R1 is a function of whatever is G and followed by R here and then R must go to epsilon in when R goes to Epsilon I am just

copying inherited attribute into making a copy of that into synthesized attribute so that gives you a general scheme of taking s attributed definitions and converting them into L attributed definitions okay.

So basically this is exactly what I used here that now if you see that what I have done here is I am saying that this R is going to have an inherited attribute which is a function of whatever is on the left hand side tree and this is nothing but a copy and if I see this says R1 inherited as a function of the parent and the left sibling this is what I have computed here and then whenever I have this rule this says that synthesized attribute is nothing but a copy of the inherited item.

So systematically I can take s attributed definitions and convert them into L attributed definitions and once I convert them into L attributed definitions then I know that I can use a top-down parser make sense this clear questions anything G of you mean whether this function is commutative or not is that what you are saying you have to be slightly careful in writing those arguments no no careful removing left recursion does not give us L attributed definitions removing left recursion is a recondition for using a top-down parser .

What we are saying here is that synthesized attributes can be converted into L attributed definitions in this manner where we say that since I am constructing a tree starting in a bottom-up traversal manner that means left-hand side of the parse tree will be computed first before I compute the right-hand side of the parse tree I just use that logic and say that anything to be computed in the right-hand side must be a function of either the parent or the left-hand side but do these two things go concurrently.

So now you can see that what I did in this example precisely this is what I did okay that in first case I was looking only at synthesized attribute where I was saying that A depends on the attributes of A 1 and Y and this A depends upon attributes of X which I systematically converted into L attributed definition saying that the right hand side of the attribute of the right-hand side of the tree depends upon the attribute of the left hand side or attribute of any node depends upon the left sibling and the parent right okay .

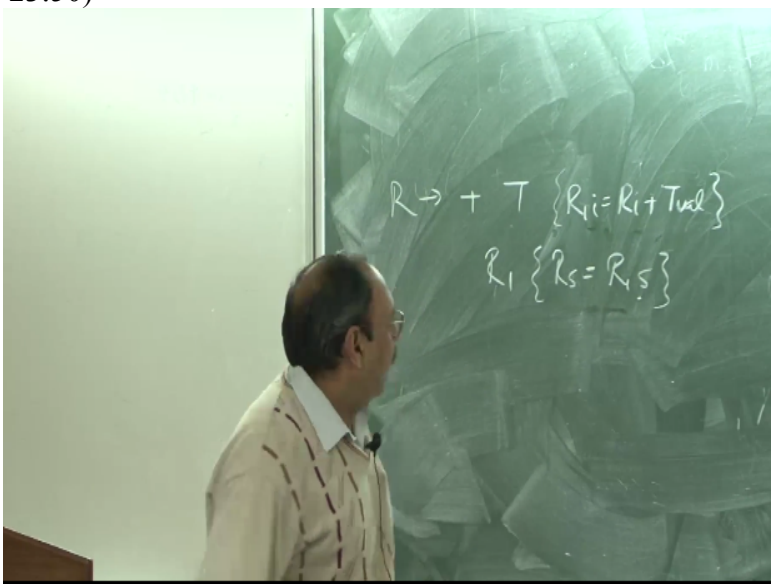
So now let us move on with implementation okay because now you also want to deal with saying that I have a top-down I have a bottom of parser and I want to deal with inherited attributes so how do I deal with inherited attributes okay so let us keep this as the running example which is this one let me go back to the same grammar this one okay now suppose let me reverse the problem yeah and now I say that instead of a top-down parser I want to use a bottom-up parser .

Now first thing you are saying that in a bottom-up parser I can deal with synthesized attribute if everything is s attributed then life is easy but if something is inherited then I do not know what to

do with it right so that is where I am going to use the translation schemes now how do I first convert an L attributed definition which has inherited attributes in through a synthesized attribute okay so let us take this one okay.

Let me just pick up one of the rules here and do you remember that how do we evaluate all the synthesized attributes using a bottom up parser what did we do there we created a value stack on which I was able to keep all the values right this everyone remembers how we use the value stack okay so let us just pick up a rule from here okay.

(Refer Slide Time: 23:50)



So rule I am using here is let us say R goes to +E and R and then they are two definitions on the right-hand side so let me also write the embedded definitions here and embedded definitions here are in this case it is R1 inherited is a function of R inherited +T value followed by R followed by R synthesized =R1 synthesized okay and this I am calling as all this is what I have and now I want to use a bottom up parser okay only problem with bottom up parser is going to be that I am dealing with inherited attributes and I do not know how to compute inherited attributes okay.

So can you think of a scheme to now convert all L attributed definitions into s attributed definitions right we are looking at two kind of transformation I want to convert s attribute definitions into L attributed definition when I am using a top-down parser and I want to convert all the L attributed definitions only into synthesized attributes if I am using a bottom of parser so what is the reverse transformation.

We just focus on the line I have written here so one way to look at this is that if I say that I have only synthesized attributes then what is the right place to compute all the synthesized attributes where do I put all the synthesized attribute computation at the end right so how do I take this

attribute equation rather from this place so one two three four and five symbols now this is already at the end so I don't worry about it but this one is somewhere embedded.

How do I take it to the end without changing my computation order so again we will have to think of a general method we might deal with examples to understand it but finally we have to come up with a general algorithm see all the time we have been introducing these non terminals to remove left recursion we introduce a non terminal so let me play this trick now and say that I want to introduce new non terminals okay and we say that for corresponding to every embedded rule you introduce a new non terminal okay.

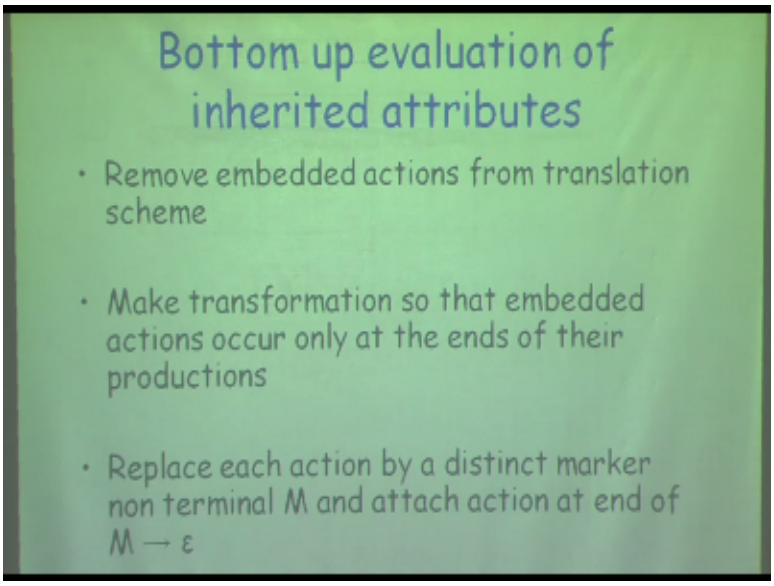
And if I now suppose I do now a transformation and say that R goes to plus T and corresponding to this I introduce now a new non terminal M and then I say R1 and then I have our synthesize equal to R1 synthesized okay and then I say M goes to at fine okay so at least if I forget about this rule okay am i dealing with the same language yeah and now I now say that now M is going to have certain attributes because now it is a new tone terminal a new non will have inherited attributes it can have synthesized attributes.

But we will deal with data deal with that suppose now I say that I am goes to silent and this is where I introduce R 1 inherited is R1 plus female when it give me the same parse tree so what is the past tree I will get from here I was getting are going to plus TD section R1 and this section right so I was getting a parse tree which was of this form plus T let me call this as action 1 let me call this as action 2 I was getting action 1 and then R1 and action 2 and now if I look at past tree corresponding to this, this gives me R goes to plus.

T M R1 action to this M goes to silent and action what if I just do a scan of all the leaf notes is it going to give me the same strings yes no so all I need to do is whenever I have embedded rules introduce a marker and move rule to the right hand side now if you look at this all the rules are on the right hand side if all the rules are on the right hand side can I use now bottom of parcel because everything looks like now a synthesized attribute yes no what are the two actions leading to so I have multiple symbols which can go to FL remember the taxiing will be different in each case so in general this is how it will work.

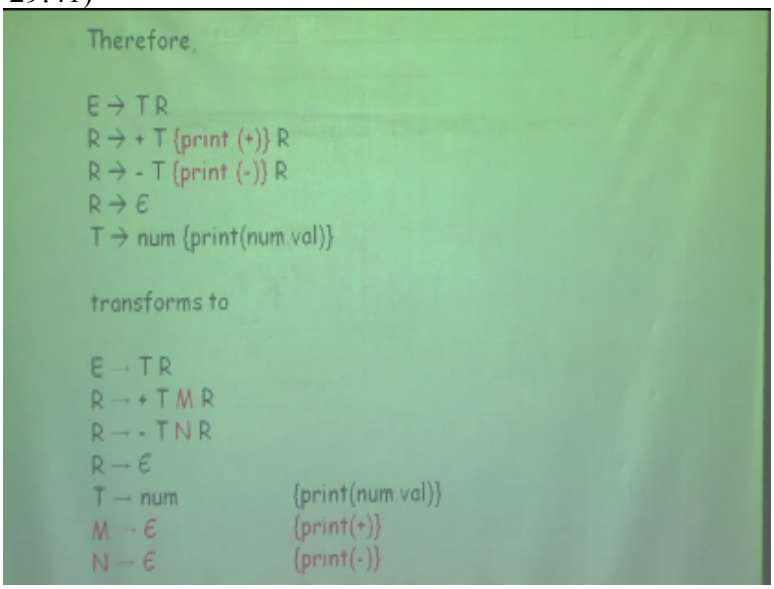
.

(Refer Slide Time: 29:20)



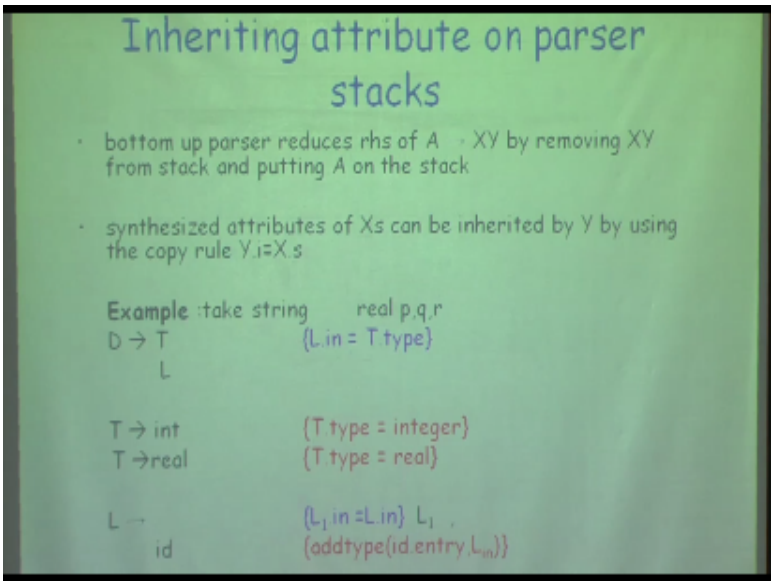
That whenever you have okay so this part we have done you are okay so bottom up evaluation of inherited attribute you want to just remove embedded action from the translation scheme and how do I remove my deduction lake transformation so that embedded action occurs only at the end of their production and replace each action by a distinct marker and this distinct marker is the non terminal which says this marker goes to Epsilon so it does not change my language and this is what happens okay.

(Refer Slide Time: 29:41)



That now if I say so this is an embedded action this was the first example we took as far as the example of translation schemes were concerned where we said we want to take an infix and convert that into a postfix okay so this is what we will do we will say that I am going to

corresponding to this action I am putting a marker M corresponding to this action I am putting a marker and M and N both go to silent and the action corresponding to that occurs here.
But I still need to do more I mean this is just not sufficient because I can have attributes now corresponding to not just all the non terminals in the original grammar but I can have now attributes corresponding to the new non terminals I have introduced and each non terminal can have both inherited at least one inherited and one synthesized attribute it can have more but we can deal with at least one and then we will see how to generalize it okay.
Now if I look at this grammar can I use a bottom up parser for this okay so I started with a grammar where I had all these eligibility definitions here basically if you see placement why I mean we are dealing with relative boot definitions we are saying that I need to place my attribute equation at the right place so that it does not violate any of the dependencies when I start evaluating it so I have done a very simple syntactic transformation.
Without changing the language all I am saying is that introduce some redundancies and then compute all the attribute equations into the new places I have created so basically corresponding to these non terminals what will happen you will now have a place on the stack where I can compute it and shortly I am going to argue that actually you do not have any kind of inherited attribute in the system all attributes are always synthesized attribute inherited attributes are only a convenient way of writing attribute equations there is nothing like inherited attribute and if you are dealing only with synthesized attribute I can always use a bottom part yeah.
So I will come to that I will come to that, that is why I am saying that so far I have not introduced attributes corresponding to eminent so once I introduce attributes values of MN n then it will become clear but right now if you see this example this is lot simpler where I am not I am just taking some action I am not computing anything then at least this will work about this you are convinced okay so once I introduce attributes corresponding to M and N even this question will be answered is it clear okay.
First step is what looks like is prime of a see that if somehow I can remove all the embedded actions and move them to the right hand rightmost position then I can get grammars by introduction of new non terminals where all the actions occur only at the rightmost position and therefore they are equivalent to or they look like synthesized attributes only and therefore I can use a bottom of parser to evaluate this but I still have to generalize it is this part clear to everyone okay so let us move on then okay and let us see that how do I deal with inherited attributes on the parcel stack okay.
(Refer Slide Time: 33:32



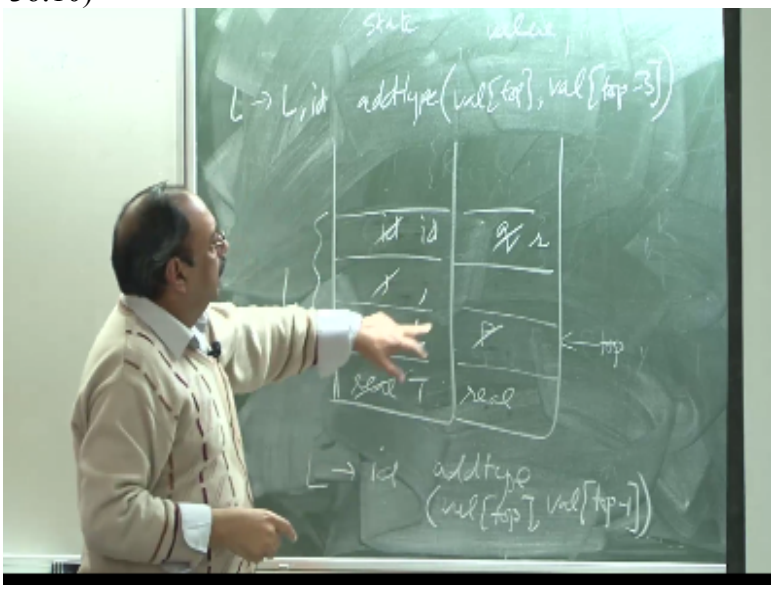
So what we are doing here is that when we deal with bottom up parser remember that this is what we did when we were having only synthesized attributes that if I have a rule which says a goes to X Y then I already had the attributes of synthesized attributes of X and Y on the stack and before reduction I computed value of A or X and Y and then push the A and in the value stack I pushed the synthesized attribute of a little what I did okay so synthesized attribute of X can be inherited by Y many times which are really copy rules okay.

So when I say that I have says that a goes to XY basically we are saying that now if I use only L attributed definitions how can value of Y I be defined Y I can be a function only of AI and X I and XS right that is the only way I can compute Y I there is no other argument which is required to compute Y I okay so many times what happens is y is a copy of X but as I said I am only giving you examples I am building up so that I can come up with a general algorithm which will solve this problem for all grammars okay.

So here is an example first yeah you remember this example types okay so now I had this DY into DL which said L inherited is T type but at that point of time I just wrote this equation at the rightmost position but now I am computing it in there because L was an inherited attribute and therefore I want to compute it only on the left of it right so I will first say T followed by the section followed by L and embedding actions in the right place now and Here I am saying that T goes to into and T goes to real.

So there is no embedded action Here I am computing this in the rightmost position and then I said L goes to L comma I D so I said L 1 inherited is L inherited then I have L 1 and here I was saying that I am making an entry in the ED type and when I had L going to ID then I was saying that I am only making this entry in the positive way through the grammar I had okay now let us

do one thing let us try to parse the string and he actually trying to observe a phenomenon and see what happens.
And this thing I am trying to parse is this one real PQR right and I am going to use a bottom of parcel okay because that is the intent that I have L attributed definitions but I want to use the bottom of parcel right now if I use a bottom of parcel let us just deal with the stack there is nothing else which is really required what will be the first symbol I see on the stack first I am going to.
(Refer Slide Time: 36:10)

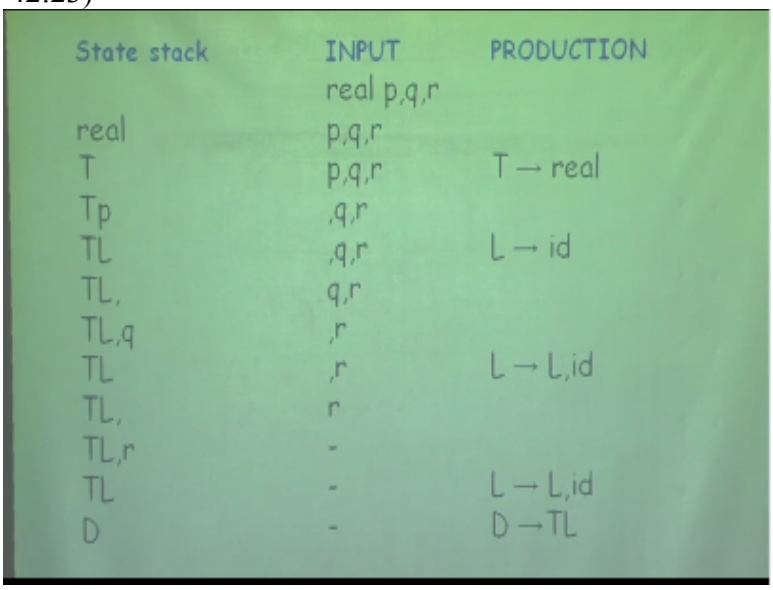


So let us say that I have this value so this is my state stack this is my values back and first I will get this name which says I have real here and the value stack may contain real and this real will get is basically nothing but tight yeah so T type is nothing but real and that value is available here then what will happen then I will push P on the stack okay so P will come here okay and corresponding to P okay.
What will be the symbol here look at the grammar here P is nothing but an ID and therefore this is nothing but an L okay, and now when I do a reduction by this rule which says L goes to I D what is it that I want to do so remember that I have T here now what will happen is that I will first take P on the stack so let us see how it grows okay I will take P on the stack then I will make corresponding to this will make an entry in the symbol table.
And I will push comma on the stack then I push Q on the stack and then corresponding to Q I will make an entry in the past table and then I say oh sorry I will not reduce that Q so I will have L comma ID and then when I reduce L comma ID to L that point of time I will make an entry corresponding to Q in the symbol table and then I will push comma on the stack and I will push

R on the stack and then I will say L goes to L comma ID which is L comma R and that point of time I will make an entry corresponding to R in the symbol table right.
These are the three entries I was making in the symbol table so when this reduction takes place okay that this was ID and L goes to ID I want to take the section and what is the action here I want to make an entry in the symbol table saying that type of P is what is the type of P real now where is that on real on the stack with respect to the grammar symbol so if I say this lexeme if I say stack this is my stack pointer this is top okay.
Where is the value of the type available on the stack value stack top minus one so can I say which role which says L goes to ID which is saying at type ID and tree L inherited instead of that can I write a rule that whenever I have this reduction I am saying as type and where is my ID IDs at the top so I say value at the top and then I say type information so what is L inherited there L inherited is nothing but type information.
So there I say that this is value at minus one so basically I am looking for this information what is the type of T and that is on the stack I could find it at top minus one if I now go further and this has been reduced to L then I push comma here and then I will push you here and I will have ID now when I do this then this whole thing is going to get reduced well now when I have this rule which says L goes to L comma ID what is it that ultimately.
I am trying to do see remember that all these rules this rule and this rule which is a copy rule this is not contributing anything this is just taking value in one part of the tree to another part of the tree ultimately these are the only two actions I am interested in I want to make this entries in the symbol table okay so now if I say that I am reducing by this rule okay can I say at the same time that whenever L goes to L comma ID basically what I am trying to do is and where is value of I what is the offset for value of type.
So this is my top which contains like seeing this is top minus one top minus 2 end top minus three right this is where the value of attribute is and once I do this reduction what will happen now that this will go this will go this will go this, this will go and it will become L okay and then I will again push comma on the stack and I will again push now ID on the stack which is going to be R and ID and I will again do this reduction.
And when I say now I want to make an entry corresponding to R okay what is the rule I will use I will say that type of R is available at top minus 3 which is here okay so do I need to use any of this rule or this rule if I know the position of the type in the stack so two things you must remember here is that corresponding to this rule and this rule basically I am saying that lexeme is always at the top and type is either when I do the reduction by this rule type is a top minus one

and when I do this reduction type is at top minus three it is always a fixed position on the stack okay.

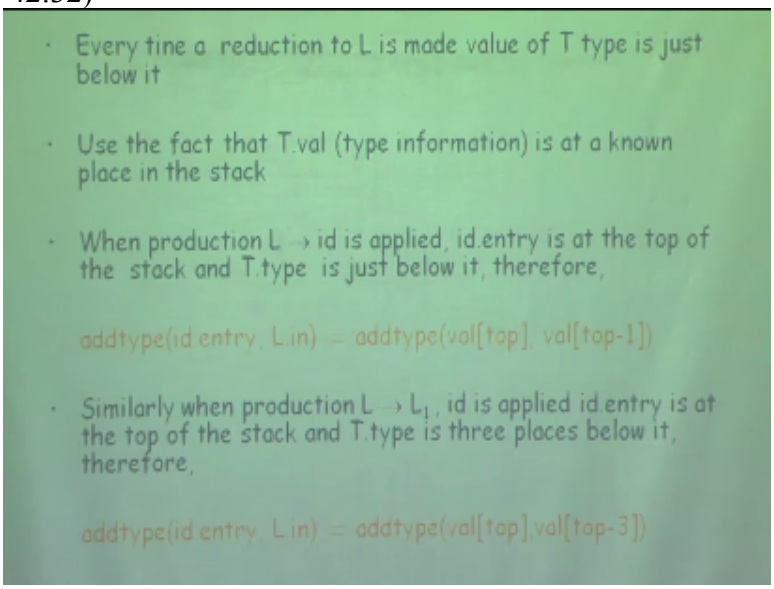
(Refer Slide Time: 42:23)



State stack	INPUT	PRODUCTION
	real p,q,r	
real	p,q,r	
T	p,q,r	T → real
Tp	,q,r	
TL	,q,r	L → id
TL,	q,r	
TL,q	,r	
TL	,r	L → L,id
TL,	r	
TL,r	-	
TL	-	L → L,id
D	-	D → TL

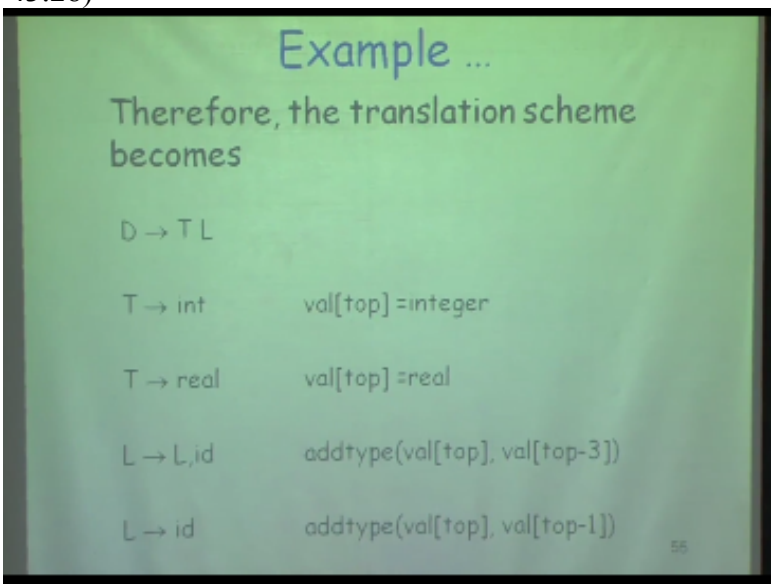
So this is what happens that if I keep on doing this reduction every time a string is reduced well T well is just below it on the stack every time I say that whole thing is getting reduced T well is just below this.

(Refer Slide Time: 42:32)



And therefore now I can say that every time reduction to L is made use this information that it is in a known place and therefore when I say L going to ID is applied then we say that T type is nothing but ad type ID and trees nothing but at type top and top minus one okay which is here

and if I say L goes to L comma ID is applied then this is basically saying that this is equivalent to at type top and top minus three.
So I could always find this value in a fixed location on the stack I was fortunate this time it may not happen in general but in this case it happened as far as this glamour was concerned and therefore now if I replace all these actions by this how will my grammar look my grammar will look something like this.
(Refer Slide Time: 43:28)



I do not have to have any inherited attributes I do not have to carry these values of L inherited being L1 inherited and T type being L inherited and so on okay I can just write these two rules and all those inherited attributes will just go away because they were anyway carrying information from one part of the tree to another part of the tree so I can use a bottom up parser can see that I started with the grammar which was Ella T booted definitions and then I was all I was always able to find attributes.
Somewhere on the stack and I was able to then use it questions if you do not ask me I am going to immediately give you a question which may have in the quiz form which have been edited from the parent so what about them now what attributes can you inherit from the parent can you inherit synthesized attribute or only inherited attribute give answered your question right actually if you start looking at carefully there is nothing like an inherited attribute in the system they are only synthesized attributes look at it this way.
So let me in general write something okay some rule and let us say that this is the first rule of my any grammar I so suppose I have a rule which says a goes to X Y in red and this happens to be the first rule of my grammar where a is the start symbol okay and right in the beginning we said

that what are the kind of definitions we will have we will say that as far as lexemes are concerned the tokens are concerned they can only have synthesized attribute right now if I look at A.

What kind of attributes a can have what is inherited attribute of A and what is synthesized attribute of A what is inherited attribute of A cannot be defined is the start symbol it has no left sibling it has no payment right so this is under what about synthesized attribute of a it can be defined as a function of let us say let me call this function of XS YS and ZS right now what about X inherited how can you define how can I define X inherited what does it depend upon what does it depend upon inherited but inherited is not defined it is undefined.

And what about its left sibling does they have XLF sibling no so this is also undefined what about X synthesized what does it depend upon what does that synthesize depend upon so there is going to be some rule corresponding to X somewhere and it will depend upon the symbols on the right hand side so XS can be defined okay now what about Y inherited so let us try to compute Y inherited so X is now a function okay corresponding to some symbols okay.

But it can be defined using the rule of X what about Y inherited what does it depend upon it can only depend upon inherited attribute of the parent which is undefined and then it can depend upon attributes of its left in left sibling but X I is undefined so it can only be a function of X synthesized cannot be anything else what about Y synthesized Y synthesized is some function of its symbols which will occur in the rule corresponding to Y where Y occurs on the left hand side and symbols on the right hand side.

Similarly I can argue about Z inherited so can you see that do I need any inherited attributes in my system if I am dealing only with synthesized attributes yeah then I can always find them in the stack I just need to know the right location like in this case I was able to know the right location saying that whenever this reduction takes place then the right reduction is finding the value in top minus three whenever this takes place right places top minus random and I am done right.

So I do not have to worry about inherited attributes at all inherited attributes are on notation which is easier to express but really is not required I can only deal with synthesized attributes if you are dealing with the electrocuted definitions if you have cycles then that is a different issue and we will see how to deal with cycles okay so let us take a break here today and we will continue our discussion in the next class from this point on this.

Acknowledgment

Ministry of Human Resources & Development

Prof. Phalguni Gupta

Co-ordinator, NPTEL IIT Kanpur
Satyaki Roy
Co Co-ordinator, NPTEL IIT Kanpur
Camera
Ram Chandra
Dilip Tripathi
Padam Shukla
Manoj Shrivastava
Sanjay Mishtra
Editing
Ashish Singh
Badal Pradhan
Tapobrata Das
Shuubham Rawat
Shikha Gupta
Pradeep Kumar
K.K Mishra
Jai Singh
Sweety Kanaujia
Aradhana Singh
Sweta
Preeti Sachan
Ashutosh Gairola
Dilip Katiyar
Ashutosh Kumar
Light& Sound
Sharwan
Hari Ram
Production Crew
Bhadra Rao
Puneet Kumar Bajpai
Priyanka Singh
Office
Lalty Dutta
Ajay Kanaujia
Shivendra Kumar Tiwari
Saurabh Shukla
Direction
Sanjay Pal
Production Manager
Bharat Lals
an IIT Kanpur Production