Indian Institute of Technology
Kanpur
NP-TEL
National Programme
on
Technology Enhanced Learning
Course Title
Compiler Design
Lecture-13

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

So let me start looking at situation where I will not even do a wrong thing on the parser an on that scaling take care of the situation and order is in the order of first and if I say c*d which is not in the language the parser declares an error where I will not do a wrong shift in the errors canonical LR parser never makes a wrong shift or reduce move and it immediately that is an error this is the most powerful parser method problem was that parser table has a large number of States.

Then we start looking at the LALR Parse table now LLR word comes from Lucre headily but this is the name which was given to it historically this is what we follow this has not been explained in literature then what I the difference here o what we do here is we look at similar looking states and when we say similar what that means is the states which have the same kernel but different look ahead in the set of LR(1) item and then we try to merge them.

So for example when we had a I_4 and I_7 where the kernel was the same which said C goes to d. and look the look ahead in this case was c d and in this case of was $ and if we merge it we are going to replace both 4 and 7 by let us say a new state I_{47} what does it consist of now it consists of same kernel and with the head of c d. And similarly we say that I_3 and I_6 and similarly I_8 and I_9 form pairs and if we merge all the LR(1) items only the bundle will be the same but look ahead will be different .

And then we start at a discussion saying that if I now construct LALR parse table what will be the size of this and size is going to be same as the LR table so the question that came was will it be having the same power as the LR or will it be more power that is what we will be discussing and turns out to be if e can do it by construction of languages and the languages which are not in SLR which are in LAL turns out to be more powerful than SLAR so when we construct LAL parse table is that.

So this is a longer step that first being construct all sets of LR(1) this is the step we went find that all sets having the same core and replace these sets by their union and we I do that then now I am going to get a set LR(1) items which will be of the form which says $J_0=J_m$ after the system of merging say that for each core and once we do this we construct the parse table a earlier there is no difference so the only thing happened now is that earlier you had multiple rows for the common kernel .

But different look ahead and now I have the same rule so what is the implication on this so j each of these J's I actually a union of verification of knowledge that I have compact parse tables now at the same time I have little less power o in fact we can again find examples again which are which is in canonical LR and which are not in LAL and say something which is in canonical LR and not in LALR what does that mean in terms of the parse table .

And so when we first look at the SLR table we said that a grammar is going to be essential SLR and what was the condition if it does not have in the parse table and then we went to canonical LR and if there is a language is in Canonical LR and is not in LALR after this merger what does it means in terms of conflict it can have multiple in the table but kind of nds these will be will it be shift reduce or reduce can I have shift use .

And conflict LALR if the grammar was in Canonical LR okay so why is that so the only thing that can happen is that when I merge these table or states it cannot give rise to a shift and the argument is very simple if there is a shift reduce conflict in LALR then that conflict must have conflict in canonical LR that means the grammar was not itself in the canonical LR so if the grammar is in canonical LR then the shift it can give rise to only reduce conflict okay so definitely there are language like this .

So since they have the same quote so go to of GX the X values are grammar symbol will also have a same rule so this is coming from construction so this is how we construct and this is how LALR parse table is going to be so what I have done is I have replaced states 3 and 6 by a state symbol called 36 and 4 and 7 by the state symbol 47 and 8 and 9 by state symbol as 89 so it has 3 states this is exactly the same number of states as far as SLR and this is exactly the exactly the same number of states as far as SLR a parse table but they actually will be different because we have more entries .

So let us not even compare unless we construct the method of parser only thing we can assert at this point of time is that the number of states are going to be exactly the same so again looking at some of the properties of LALR parser if I now look at this reduction what is going to happen

now that I have now C * D C * D is my language and suppose I came only please start what will happen in case of LALR parse table .

Suppose by equal to a let LALR parse table is C * D there will attach there so if you keep on saying that C *d will get reduce to capital C and then you can see a dollar but since this is the first capital C you will still not be able to accept it because I still have a rule which says as close to C or s goes to a so it is not accept but it will now move additional shifts but before for the reduction and we kept it there .

So this is what happens here so in general core is a set of LR(0) items and LR (1) grammar will produce more than one set of items for the same core and merging this never produces shift reduce conflicts but may produce reduce or reduce conflicts and SLR and LALR parser have the same number of states continuing on this discussion so merging the result into conflicts in the LALR and these conflicts are reduced reduce conflicts.

So here is a small argument which says that why I cannot have I should reduce conflict so shift reduce conflict will come only if I have LALR parser with item which say of this form it says X goes to α .and this says y goes to α β. b with a look ahead of me what is happening then that in this case I am ready for reduction and in this case I am saying that I can shift now if I have such an alarm on item in LALR parse table then they must have in some state from where I arrived in this state.

So similar conflict must have existed in the earlier state and before and therefore I do not have any kind of shift reduce conflict but if I look at the situation and I had states like X α. on a look ahead of A and y goes to α. look ahead of A and after merger I am going to produce a state like this and this state has a reduce conflict earlier this conflict did not exist this happens when I construct LALR parse table .

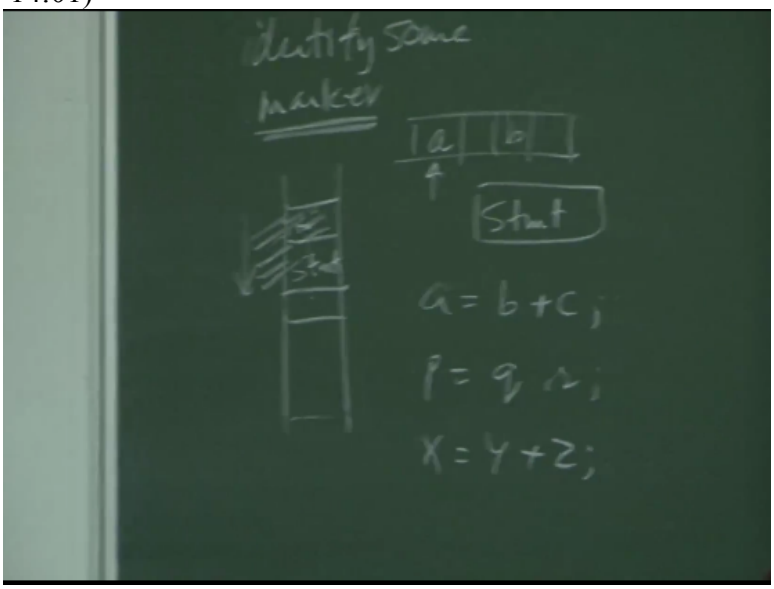
The only thing is that these methods are more complex and they are direct, complicated and they are efficient algorithms to develop LALR parsers and normally a tool which constructs a parse table and if I look at relative powers then SLR (1) is less powerful than LALR(1) and that is still less powerful than LR(1) and if I look ahead if I have a look ahead of K then the same thing applies that for a fixed set of SLR (1)is still going to be less powerful than canonical LALR .

So languages which can be passed by this particular parser cannot be passed by this and similarly LL (K) is less powerful than LR(k) and why LL(K) is less powerful than LR(K) compared to a lot left but we move so left recursion is no longer an issue so think about it in general argument is that if I am looking at LL parse table or I am trying to do countdown parsing then what information do I have a symbol which I want to expand and I have table effects based on that I

take a decision what do I have in case of LR and in LR I have the full stack information my state information captures whatever I have seen which in case of LLR is not possible in case of LL what we are saying is that this the grammar symbol I want to expand here so which in case of LL is not possible we are saying whatever I have seen that extra information I have so that I more power in an LLR and in general programming languages most programming languages actually fall in this class LALR .

So when you look at tool like parser what you are generating is LALR parser and not a canonical parser and there are parser generators which are available but for canonical LLR for all practical problems as LLR is sufficient as far as programming language is concerned and again how do I do in generally LLR parsers so think about it .

(Refer Slide Time: 14:01)



So let me draw the figure for you let me think about this that suppose this is my stack and I have now state s and I have a look ahead let us say a and now when I refer to my parse table that find that s a is error free how will i recover from this so how can I continue parsing suppose I reach this configuration so if I say that I skip some tokens here and reach let us say here a symbol B on which I can find an action here what happens in that case cannot be contacted because what you are saying is valid because if I look at by parse table.

I am saying that from this point onwards I can continue important okay so let me give you an example let me write a piece of program suppose I a=b+c then p=qr and x=y+z so what is missing here is the plus symbol so I reach some state where I have seen this I have seen this part then I am expecting to see an operator but what I see is an identifier okay so I say I could have done continued parsing on an operator right .

So I say skip this skip this skip and I reach here and I have skipped all this does that make sense yeah very good so how do I do that so solution that has been provided is that is being proposed is that if I skip everything up to semicolon okay and then continue parsing from here then I will be good right but how do I now do that in terms of the parser so assuming that this is a correct solution now I explain that in term of the parse table okay so let me try to articulate what you are saying that imagine that there is no error here but suppose there is no error here then this would have reduced to same statement so there is some state here followed by a statement then this whole thing would have reduced.

So what we try to do is try to create a situation where as we say let us skip this till the end and suppose I say that statement is the symbol I want to skip till the end of this then I say that whatever is the symbol now so I have to now do two manipulations one that I keep going down into the state so first I identify certain markers and what are these certain markers so for example steps into the market and I say that in case I am parsing a statement.

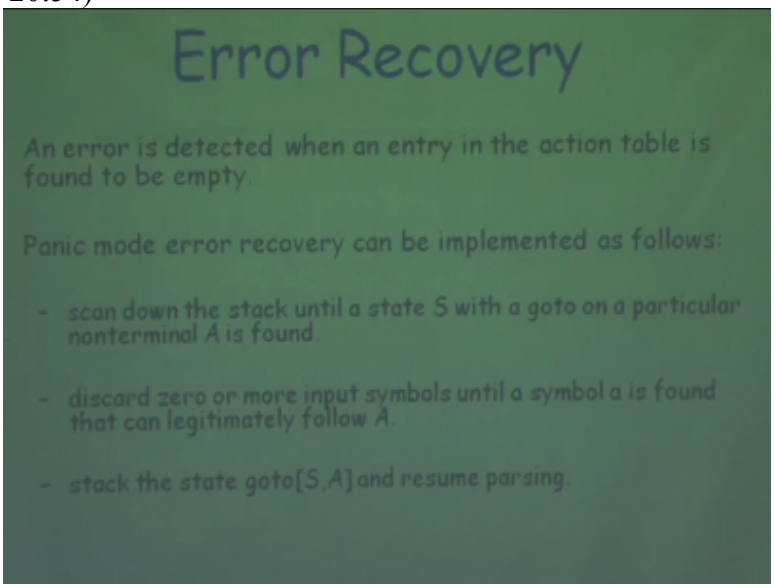
And I find when I just keep everything up to the end of the statement yes that means I basically saying that when I skip everything up to the end of the statement okay I will assume that I have known all these errors and as if this error never existed. so that means first I have to go down and find state corresponding to this okay and then once I have discarded all these symbols change then in this state then I push statement.

And then that will give me a very good right then I will be able to find in my parse table and once I have reached this then what do I do I say skip everything up to the end of the statement that means spot scanning my input skip everything up to semicolon and once I reach here so what we do is that another parser always defines what are the symbols if you want to synchronize and on which you want to do error and so you can say that when it comes to end of statement it is one thing that .

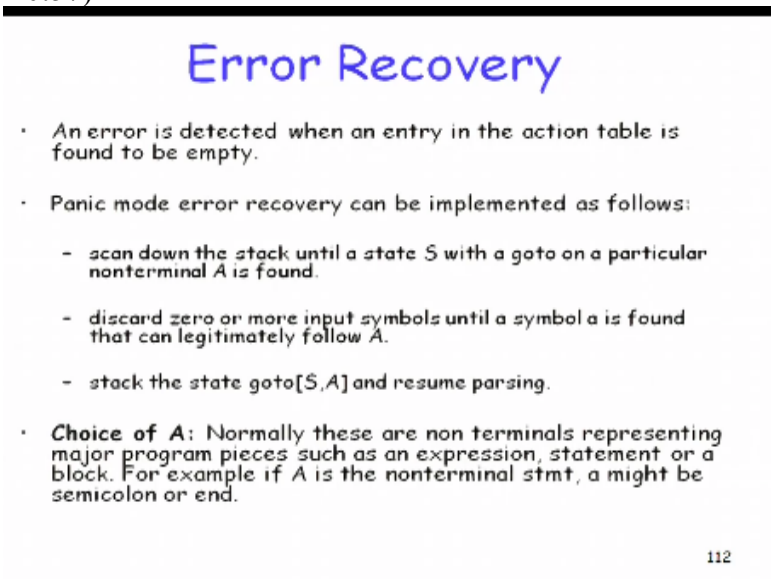
You can define block and of function choice is up to the parser builder this is going to be the granularity so if you want to be through the level of some expression that can become very complex so normally what is done is we try to find certain blocks where we say that if an error occurs then I will skip up to end of that block and if I skip up to end of that block in terms of parser I say that on stacks find a state.

Which has a valid go to on that particular symbol then pop everything up to that state push this symbol do a go to skipping input till you have reached end of that statement and continue parsing right so this is what will happen that if I say that there is an error I say skip this whole thing skip it up to semicolon and therefore, I will start scanning mind discard all these symbols then I try to

continue parsing from here on the step I will pop symbols which will be taking me to this state then I push the non-terminal is can sat that whenever a function I start everything on the function in so normally this is capture at the level of a statement okay.
So aratibacally is if you detect an error when the action table is found to be empty so this for panic mode .I will double you scan down the stack until you have a state with a go to one a particular non terminal.
(Refer Slide Time: 20:54)

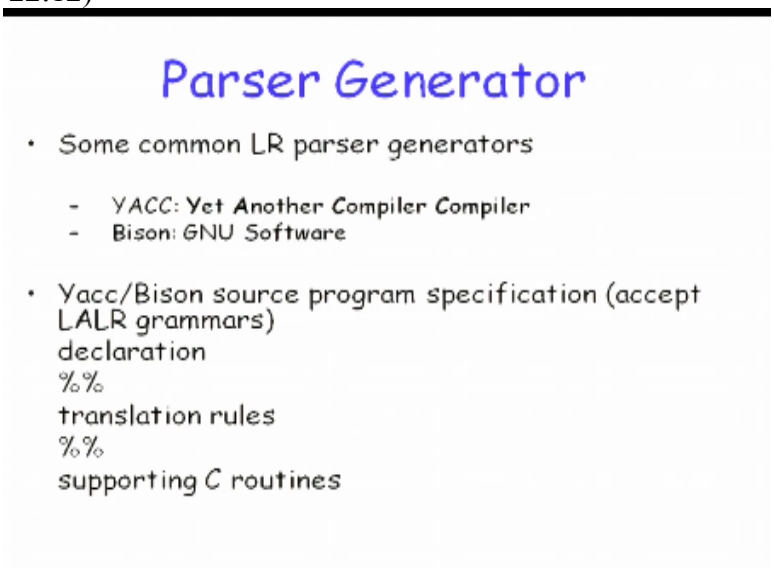


A and then the start 0.
(Refer Slide Time: 20:57)

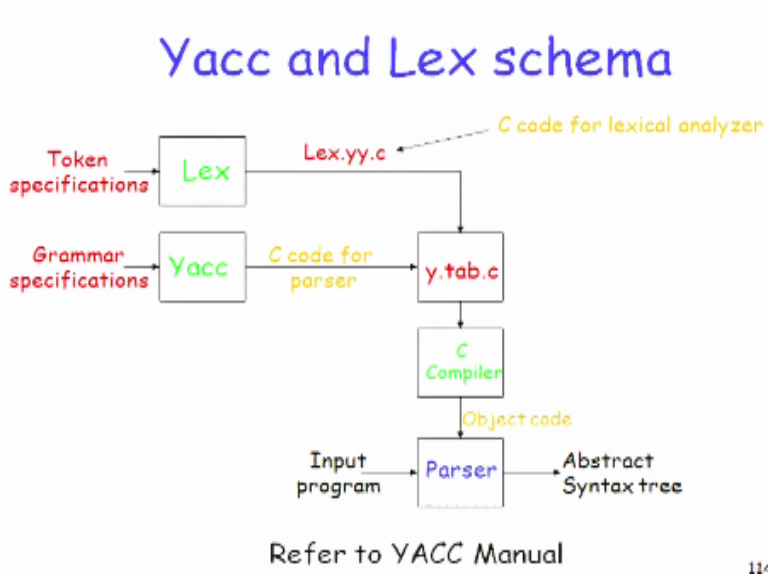


Error input symbol until you find a symbol which is which form of a and then stack the state this state obviously by pushing A symbol and then part okay what this is and one is the choice of a normally these are some non terminals which represented major programs so for example it

could be stacks it could be a block it could be a function ,so you have to take a procedure as parser to parts of the what the error become which are going to do.
So most parsers will say that I want to go every recovery at the level of the state then you write this part everything here both in terms of concept as well understand input okay, so now we have some parser generators we do not really write parsers by hand so the interest and parsers well generally and what we have our parts of the knitted yak is one bison is a little which is really common and these are really.
(Refer Slide Time: 22:12)



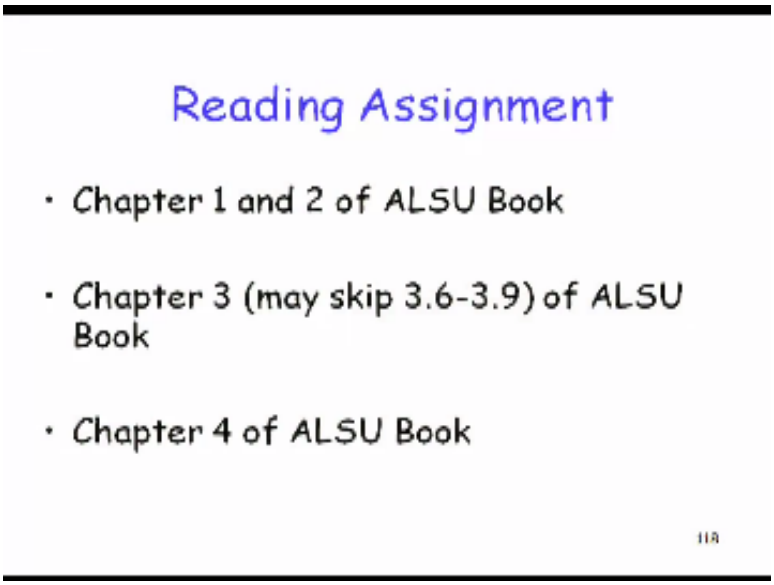
Source specifications are LALR grammars you will write grammars in LALR then we will be able to generate 10 parsers for this and the format just to refresh your memories and we have declarations ordered by two special symbols ordered by all the transition rules and followed by all the series this is how the structure of specification.
(Refer Slide Time: 22:36)



And if I look at block diagram block diagram goes now you can see that I am using Lex and get together YACC is generating this file called Y.N.C in which I have to do X.YY.C which has been generating form of and then when I use the native C compiler to compile this 5 it is me an executable which can take any program and we mean and parsing expect syntax tree in every 10 sec syntax tree you will have to write certain actions but in general it will be able to say whether this thing and also one important thing to remember here is from his limitation point of view that is a viral brail Y.C okay, it is a fine variable it will just again this 5 we will that YY0 you turn that as to one that you find that and you try to generate parcels and γ an error maybe give you a lot of people.

So this variable can used way we can turn on the debugging and can get all the sad information that means every time you see a symbol what is being pushed what is being taught what is being reduced all the different information is not coming oh that literally can also help you in understanding how your parser is working and if case something is going wrong so this is where we will close about discussion.

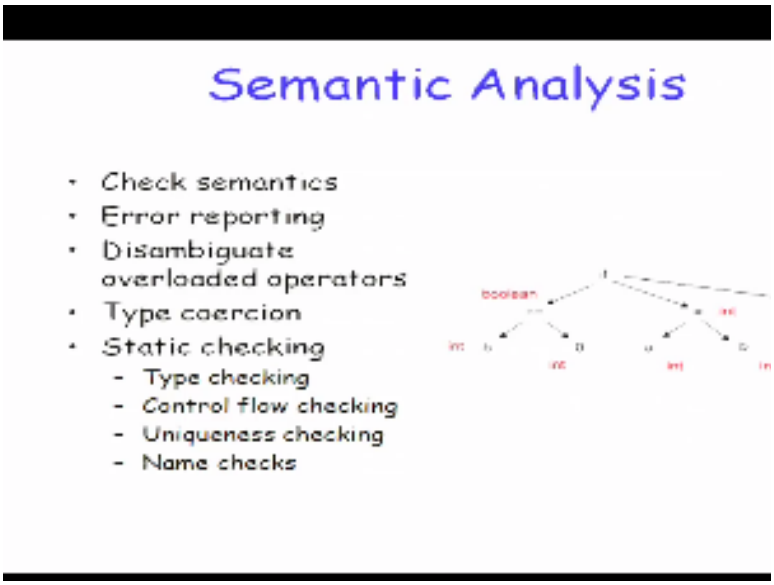
(Refer Slide Time: 24:10)



On R sin and there is a reading assignment and finish this assignment before you are mixing exam it will be helpful basically look at book by a whole and safety Pullman chapters 1 to 4 but you can skip sections 3.6 to 3.9 which basically say although I can generate the finite state machine of regular expressions how do I generates non-deterministic finite automata how you do that I might inform which I am sure we can go back next are taking into your business 0 but it will be helpful if you all this before you make sense.

So this is where my close discussion on parsing and you move on to that any questions anything down bottom whatever questions comments you may have can you move on back in timing checking yeah so LALR 2 is more powerful we can go to any number of state one that we do not have tools second that most languages require they force LALR what is sufficient most of the program there may be languages it cannot be parsing so when we go into those who natural language processing they look at all those techniques as far as programming language processing is concerned in a rush anything else. So then let us move on to spell checking so we also here is for example.

(Refer Slide Time: 27:05)

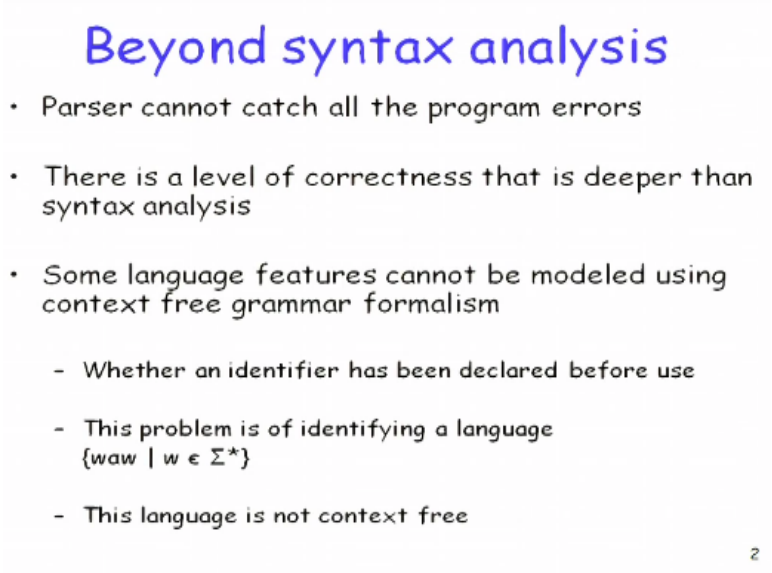


What is time to do in this semantic analysis this coil is again coming from what we had in the introduction okay so now we want to check the meaning of the program and also we want to report on the errors we also want to listen operator okay so here is question example I do not have example of overloading but we discuss overloading we found that that will be operator which depending upon or different meaning we want to exactly find out what are the overloading operators what are the overloaded functions.

Because some languages in the overloaded functions than it want to do type in case your expression can terminate different types we want to type the and going to checking control checking is says that ,I cannot jump into middle control zone so I want uniqueness step in saying that all my variables context are going to have a unique meaning or unique name a name check so sometimes drops may require that the begin block and block we have it for me and that name has to be same and so on.

If but this list in no way is exhaustive ID is actually language dependent so depending on the language what I know I taken you want to write list of types so this is something which is beyond the syntax analysis and obviously what we are trying to do here is it was not possible something looking at the context free grammar formalism.

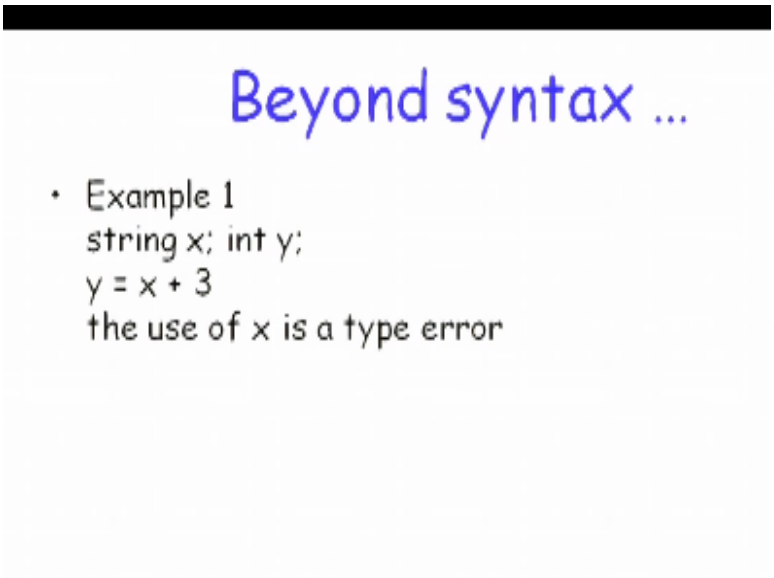
(Refer Slide Time: 28:50)



So syntax analysis was all based on context-free grammars and we want to know look at some errors ,which are deeper than for syntax analysis so this when I look at programs and I look at programming languages and we talk about you I am going to body part. I am still into the language definition there are some issues which are deeper than the syntax so I never get something in syntax and compact syntax analyzer always accepts a superset of the language what we are trying to do now is we are trying to narrow it down to saying that whether it is sticking to the semantics of the language are defined okay.

So some language features cannot be modeled using context-free grammar formulas okay, and one of the very common features will find out if I want to check whether an identifier has been declared before use . I cannot model context-free grammar will look something like this ww-Σ stack where and this is something which is not check.

(Refer Slide Time: 29:54)



So beyond syntax here are now complete examples so suppose we say that I have a string X and string Y and then I write an expression which says X is sine Y is a sin X + B now clearly you can see that there is a type error here if I do not permit this addition on string a name okay but ,when I parse this if this is not an error because all it is saying is as far as parsing rules are concerned that right hand side is an expression that fin side is available that should define okay.

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 Lal

an IIT Kanpur Production