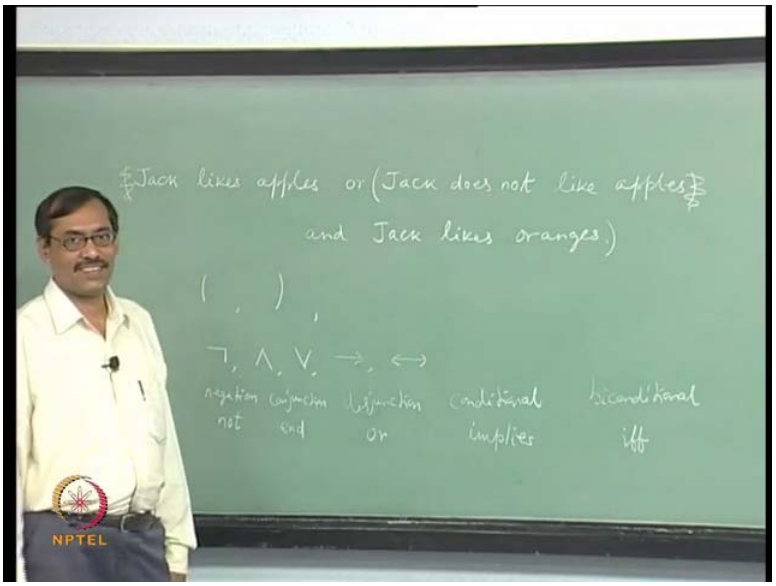
Mathematical Logic
Prof. Arindama Singh
Department of Mathematics
Indian Institute of Technology, Madras

Lecture - 2
Syntax of Propositional Logic (PL)

Basically in logic we will be concerned with how a proposition follows from a set of other propositions. We have to formalize what is a proposition, what is the meaning of `follows from' and so on. All these things we have to formalize. We will start with what is a proposition, we say a proposition, we are not going to analyze it.

Further, we are not going to be looking for what is in a proposition, like a grammatical thing: subjects, predicates, so on. We are not going to do that. We will be starting with the propositions, Well, be it so, very simple logical propositions as usual. Something is true something is false, so on. What is the need for formalization? Why you need to precisely formalize it? Let us see an example.

(Refer Slide Time: 01:01)



Say, Jack likes apples or Jack does not like apples and Jack likes oranges. Consider this proposition. From this what do you conclude about Jack's liking for fruits. He likes apples, oranges? You may have to decide. Well, suppose I take up this, now I can say: yes, Jack likes oranges because the first one within two parentheses, I am not bothered

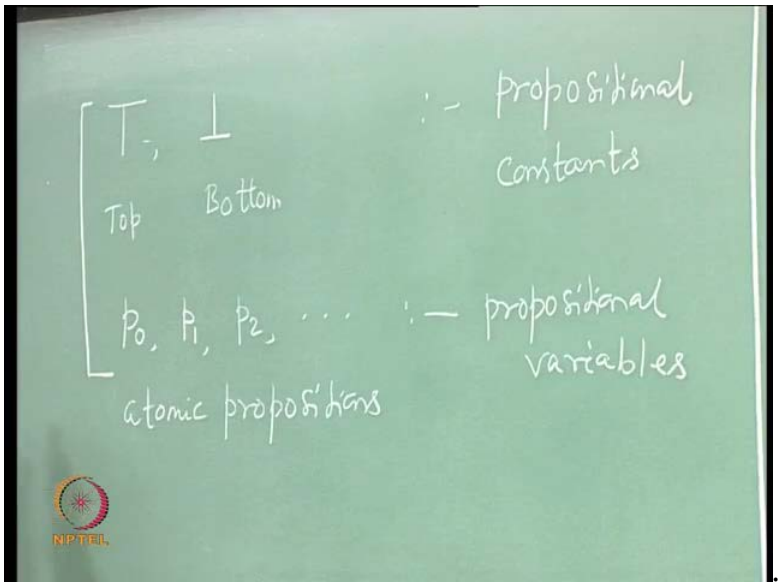
about it, it is anyway true, it does not matter whatever it is. But, suppose instead of this, I do it this one, is it definite that he likes oranges? We need precise formulation.

We will use some set of symbols, some punctuation marks, like parenthesis, and we will not always take all these propositions, the basic ones which are involved, in detail we will also symbolize. We will start with the symbols in this form, we need the punctuation marks: left parenthesis, right parenthesis; and you may have observed some connectives like your or, and, and some other. Let us have some connectives, you can have some other connectives in our languages, but I am not telling what they mean.

We do not know what are they; you are just introducing some symbols; you can read them, but will say how to read them. The reading we should have, now otherwise it is difficult to write something and not read it; it is difficult talk about. Let us give some names: this is called negation, first symbol is called negation, which you may read it as `not', its meaning only something like `not', but later. Then this symbol which is a V, is called a Vee, also we will read it as `or'. Conjunction, reverse V, will have a name as conjunction, we will read it as `and', then this symbol is, we write as V upside down.

We will give a name to Vee as disjunction and we read it as usual. Similarly this one side arrow, right arrow, we will give a name, we call it `conditional', we can read it as implies, arrow, or `if … then', somewhere. But, nothing is so precise here, conditionality is best read as `implies'. Last one which is double arrow, we will give the name `biconditional' and we can read it as `if and only if' or `iff'. There was something like you can say it is as `fif' it is if this way and the fi this way, so fif. But we will use `iff', this is traditional to right as `iff' not `fif'. These are the symbols we are introducing slowly, there are some other symbols.

(Refer Slide Time: 06:02)



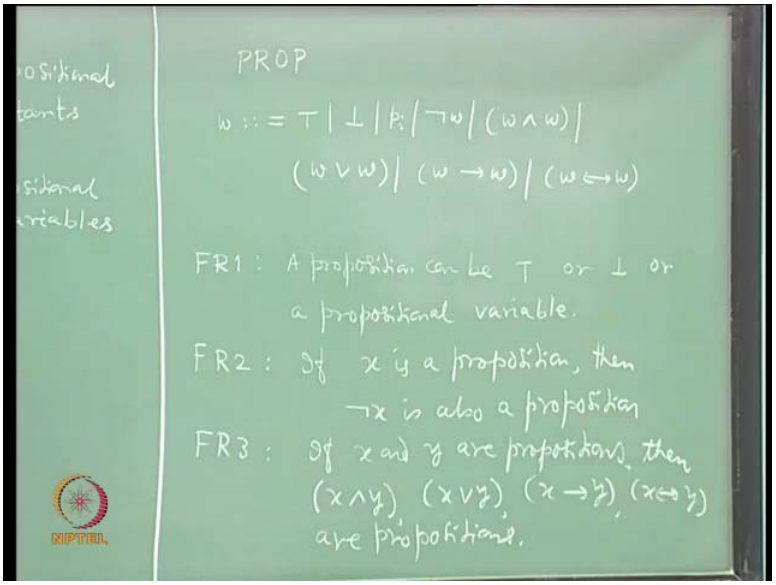
Like, which will be propositions: top, bottom. We read them as top, bottom. Next we will have some list of infinitely many symbols, p0, p1, p2 and so on; these are called atomic propositions. But, we will give a different name, we will say these things as propositional variables. The propositional variables and both these (top and bottom) together we call as atomic propositions. We are just varying a little bit, but it will be very helpful for us, in that sense these two symbols are called propositional constants.

It is something like telling Jack is a name then you say good is an adverb or adjective and so on, you do not know what is the meaning of noun, proper name or adjective we are just giving some names. We are going to define the grammar, but how it will be precisely formulated, so that all the symbol we are going to use are given. Here, they will be punctuation marks, left parenthesis, right parenthesis, connectives, these five connectives, and then propositional constants, top and bottom, then propositional variables p0, p1, p2 and so on.

All these together is the alphabet of our language. Just like in English language the alphabet, which contains the symbols a, b, c, d and so on, every string is not a word, there are some strings that we say, they will be found in the dictionary, or they are not. Here, the same way. But, there is a difference: your languages grow to include something, here nothing will be included; static, once defined that is it. Now, we are going to define words: those particular strings are, where you are interested, and we have

to take care of that example of malformation; it should be well formed. That we can assign meaning later. How to declare the grammar? It will be done in a very criptic way for example, the Backus Nour form.

(Refer Slide Time: 09:18)



We can say all the propositions are generated this way, if you look at the grammar, it is simple, as this only one line gives the grammar. But to read it will be more difficult that is the reason we had written it very criptically. But, once you get acquainted it will be easier to write it in this form, here what happens is first three things, top, bottom and pi they are from the alphabet.

There is nothing to worry about; we will know what they are. The problem comes when you read this symbol `not w' as `w can be not w'; that is the cryptic thing. In fact, every instance of this w can be different. That is what we have to remember. We can read it the other way: let w stand for any proposition, now start reading, any proposition can be top or bottom or a propositional variable or negation followed by another proposition (not necessarily same w) or left parenthesis, a proposition and one more proposition, right parenthesis, or, . . . Now, we can read it, that is what this means. Write it in terms of formation rules, you may have to write in four or five sentences.

But, let us write it to make it clear, it says something like this: a proposition can be top or bottom or a propositional variable. In another way we can tell it we can just declare it right: every atomic proposition is a proposition that is another way of writing the same

thing, because top bottom and propositional variables, all of them are called as atomic propositions. Next we have to take care of the connectives if x is a proposition then not x is also a proposition. You can see translation in the grammar: w can be not w that is how you have to write it in English.

Then all the other things we can write: if x and y are propositions then x and y, x or y, x implies y, x iff y are propositions. Then there is one more rule, which I am not writing here. But, you must remember that is the FR4, formation rule 4, which says `nothing is a proposition unles it has been generated by application of one of these 3 rules'. We just close there; nothing else is a proposition. Unless you declare it, we must say, yes something else is also a proposition there. As once a lecture going on in logic, one person wanted to convince the students that there is something called non-monotonicity of arguments.

Monotonicity means if you have some assumptions, concluded something from it, now if you take or define assumptions along with the earlier assumptions, the earlier conclusion still is a conclusion. That is monotonicity. But, that is violated in some cases. To show it, he presented some arguments and then proved that a cow has 2 legs and it is aukeard because you are not following monotonicity. We are monotonically all going, that is why this is happening: a cow has 2 legs. One biology student there, he just answered: cow has 2 legs, is it not? Now, the intension is, it has 4 legs, so it has also 2 legs. It has also 1 leg.

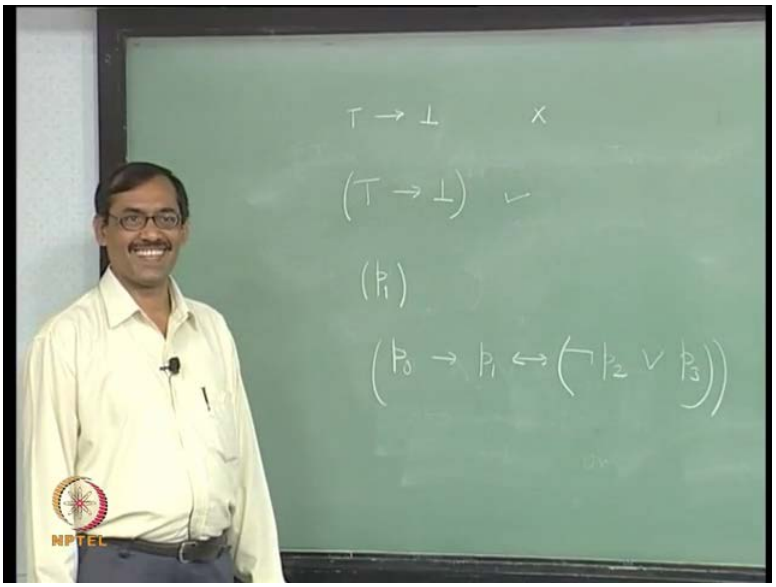
We do not want such things to happen. That is how we needed closure rule: nothing else is a proposition unless . . . It has been generated by applications of one of these rules, we are not writing it, but it was there implicitly. Now, I think you have understood what is the meaning, how to not knowing exactly, how to generate a proposition, starting from an alphabet, not every string is a proposition. But, strings which are in these forms alone will be considered as propositions. But the definition is in essence records here like your definition of factorial form not as product of from 1 to n. But, n factorial equal to n times n minus 1 factorial because it uses what proposition I can write.

It uses the symbol PROP, where will you read x and y are propositions, if x and y belong to PROP, then x and y with parentheses belong to PROP. This symbol has been overused, it is recursive. But, we see that it is well defined it is recursive, not `well defined'? Well, what we want is not only this is well defined, as you want that, there is

no ambiguity because finally we have to decide whether it is this bracket or it is this bracket. But, we have to re-assign, that means, the bracket itself the parenthesis themselves, which one we have taken, we see that up to inserting the new parenthesis.

It may not be ambiguous. That is our aim, that is, go slowly towards them once you do that then that syntax will be complete. We can go for giving meanings, we can connect to the reality this is our thought. Now, it is not in the reality, first created the symbols, now we have to play with the symbols so that to a certain extent, where there is no ambiguity, that is our first concern. Let us see how a proposition is generated whether some thing is string, is a proposition, or not.

(Refer Slide Time: 17:31)



I start with top implies bottom, is it a proposition? Yes? It is not a proposition. If it is a proposition, tell me in which way we have generated it? Top is a proposition, yes? Bottom is a proposition, yes. But it is not a proposition. I can generate a proposition with the top or bottom with parentheses. Without it, I cannot generate by following the rules and the grammar; then this is a proposition. But, this is not right. I can say p1 is a proposition. What about this? It is not a proposition, once there is parentheses, there should be some connective, not the negation, the other connectives.

Let us give a name. We call negation as negation or unary connective; others are binary connectives because they always take two things. We need a binary connective whenever there are parenthesis there is no connective, here we mean binary connective; this is not a

proposition. Is it a proposition? Well, if it is a proposition then my next question: will you show how you have generated it? If not a proposition, then what should be my next question?

Student: How do you show it is not a proposition? That is a tricky thing to decide; if we can show that it can be ambiguous, that can be?

You have to decide something on ambiguity when do you say it is ambiguous? There is no meanings at all?

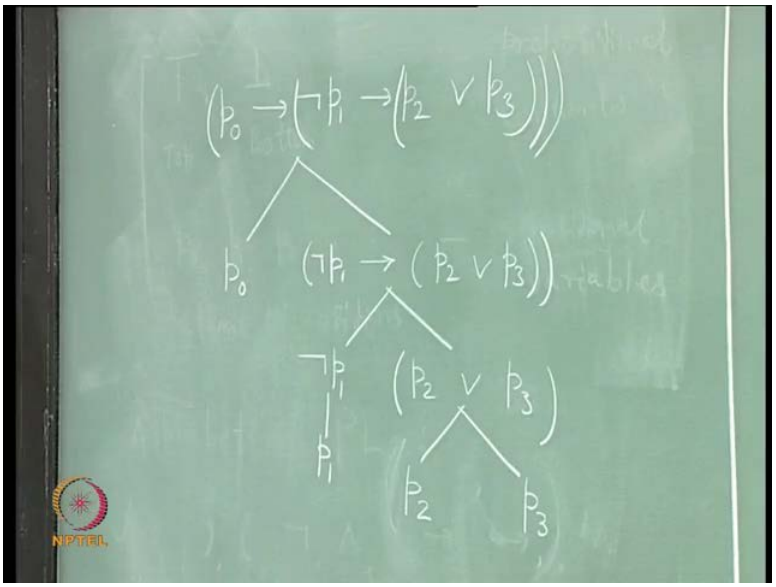
Student: We can show it to be ambiguous then value for propositional variables if the overall proposition can take.

That is giving a meaning.

Student: If we can add parentheses and make a proposition then this is not a proposition.

It was not a proposition earlier, well that is looks like, but where to add, there can be many ways of adding, any one will do? It is a non-deterministic algorithm, that is a complicated thing, well let us start with a proposition and see how does it go.

(Refer Slide Time: 20:37)



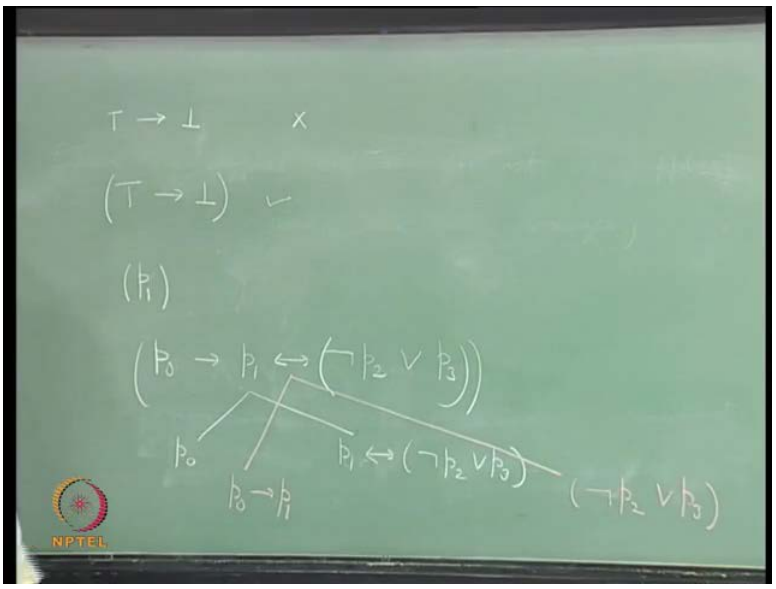
This looks like to be a proposition. Let us try to find out why do you say it is a proposition. I could have started this way say p2 is a proposition by FR1, again, p3 is a proposition, again by FR1, then p2 or p3 with parenthesis is a proposition; why? FR3.

Let us write this on the top of it, I see p2 or p3 from these two propositions, I have got these propositions. Apply formation rule 3, is that next what I do, I take p1 as a proposition, therefore not p1 is a proposition. Since these two are propositions, I can use imply symbol and get the proposition not p1 implies (p2 or p3), this is a proposition.

Now, I know that p0 is a proposition, therefore the original one given is a proposition, now you can see it as a tree, this is the parse tree of the string, it corresponds to the proposition I have. Given a string you can parse like this. You say that this is the only way of parsing it there is no other way you could have formed it. But, if it is not a proposition what is happening? Let us come back to what you told. It is not a proposition what will happen? Here, if we try to parse it, you must see what is happening in parsing to proceed.

Let us again go back try to see what is happening in parsing. To show that this is a proposition, you must find that the left one that is p0, is a proposition, the right one is also a proposition, the left right of connectives implies. It is really unfolding the formation rules slowly when you form it, you form in this way starting from the atomic propositions. Here, p0, p1, p2, p3, but when you parse it you unfold the other way and that makes it difficult in the parsing, it is a tricky affair, but you have to first find out which one was applied last that is to be unfolded.

(Refer Slide Time: 23:59)



Let us try to find out which could have been the last application of a formation rule in order that it is a proposition. It has to match in one of those forms. It will be top to bottom and not of some proposition and so on, which, on it can match starting with the parentheses. FR3 must have been used at the end, now FR3 has been used what could be the connective? Now, you can see if you start form left you get this as the connective, if you start from right it can be this considering this parenthesis, but that is also tricky it is recursive.

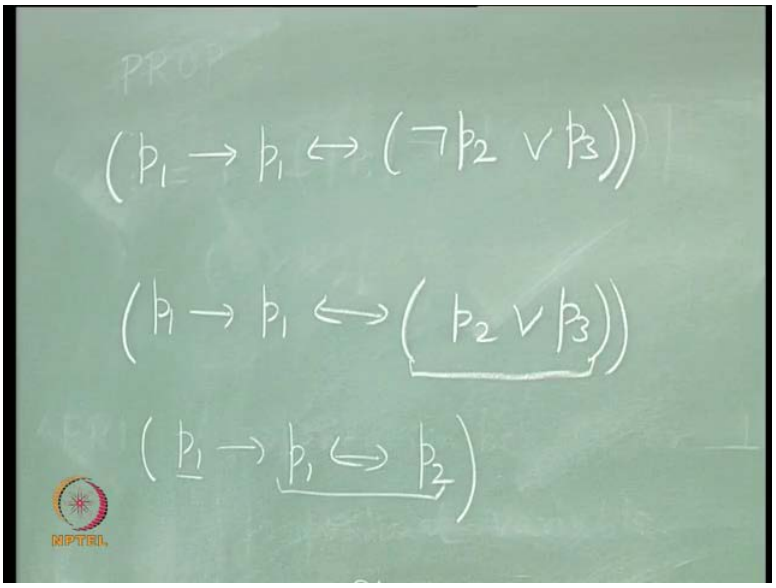
First you have to show this itself can be taken as a unit then only you can come to this, keeping that to heuristics let us try to see it from the left. If you see it from the left you can parse it with this as the connective, p0 and the other one is p1 if not p2 or p3. This is an atomic proposition, it ends there, now you come to this place, here I cannot parse it further because this is neither an atomic proposition nor starting with a left parenthesis, nothing can be done to rest of them.

Now, if you have taken this one as the connective you would have got a different tree, you should start from this way, for one will be p0 implies p1, the other one is with parenthesis not p2 or p3. There are really two parse trees, here none of them is giving a proposition at its leaves, all the leaves must be propositions in order that parsing is correct and there might be ambiguity in parsing. Even if you parse it, it can have a leaf which cannot be proposition, then you say it is not a proposition; you can write an algorithm.

Now, to determine, given any expression from our alphabet, given any string over the alphabet, whether it is a proposition or not. Yeah? Can we write? See, all that you have to do is identify which one is a proposition? Inside it, somewhere, it could have started identifying this. So, not p2 is a proposition because p2 is atomic, now this not p2, I can forget, write some p100 instead, and push it, because all that we do in parsing is unfolding the formation rules.

Once not p2 has been accepted as a proposition p2 should have been accepted as a proposition; p2 is, because it is atomic. You can start the other way: see p2 atomic identify not p2, then think of that as a proposition when it implies not p2 by something else say p2 does not matter, you do not have to invent symbols, if you do that what happens for this?

(Refer Slide Time: 28:05)



In the other bracket one more parenthesis, what we identify is one of the substance of these in the form not p2, this not p2, I can replace with p2. Now, I get this after the replacement, now there are connectives and seeing I can find out which connective, I should tell. That it will become, it will become unified with one of our rules that is important, I can take this one as atomic and the other one is also there. But, it may or may not be unified to absorb it the other one is also a proposition, I will not proceed that way, I proceed with a substring which matches with a proposition.

I will identify this one which is a proposition because p2 implies p3 is p2, (or p3) with parentheses. When I replace that with p2, now my algorithm stops there; because this whole expression cannot be identified with any of the propositions. If I take p1, here implies, here then the all things should have be in a proposition, but it is not to make the algorithm better not depending on finding out whether this is a proposition or not what you will be doing we will not stop.

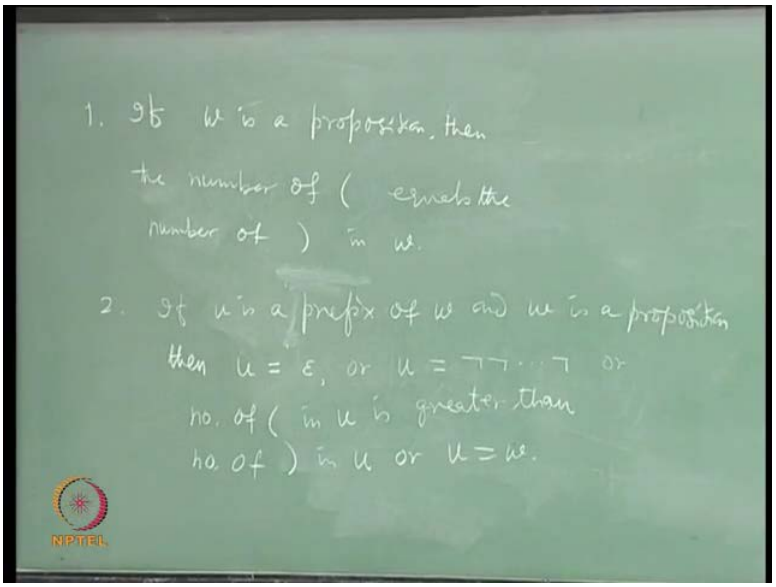
Here, we will proceed thinking that this is also a proposition starting from the left to right, we will be continuing this way say p1 is one this is another. This should have been some x, but it is not a proposition that is why we cannot get that way we have to stop, here in, adopt that one rule will be applied, each one of these two components would have been propositions; this is a propositions. But, this is not, now algorithm stops there.

It looks that this procedure can be used to determine for the any given string is a proposition or not, just go on replacing; we are following the formation rules.

But, then what really allowed us to proceed this far, there are some properties in a proposition which, allowed this, otherwise you will not come up to this case. The first property there is left and right parentheses are matching in any proposition. That is easy to see, is it? Can you see that? Take any proposition, number of left parentheses is equal to number of right parentheses; yes? Do you need a proof for this?

Take an example; first see whether it is alright. Now proof will not be difficult because a proposition has been formed by using the formation rules and in each of the formation rules you see matching of the parentheses is there. Therefore, whatever you get from recursively will have the matching of the parentheses. The proof is by induction; the formal proof is by induction again, because every step you are increasing the length of the proposition of the string, we leave it here, but you note it down as a property of propositions.

(Refer Slide Time: 32:14)



If w is a proposition then the number of left parentheses equals the number of right parentheses in w; it is an easier property. Next property is a bit difficult. But, you can still find it out, you consider a proposition. Look at the example we have already worked out read it from the left, take any prefix of it; do you see something to be happening there? Well, tell me about parentheses.

Student: It is matched, total is matched.

Totally, it is matched that is fine, I am asking about prefix of a proposition.

Student: Number of left always equal to number of right.

Can we say that is a strict?

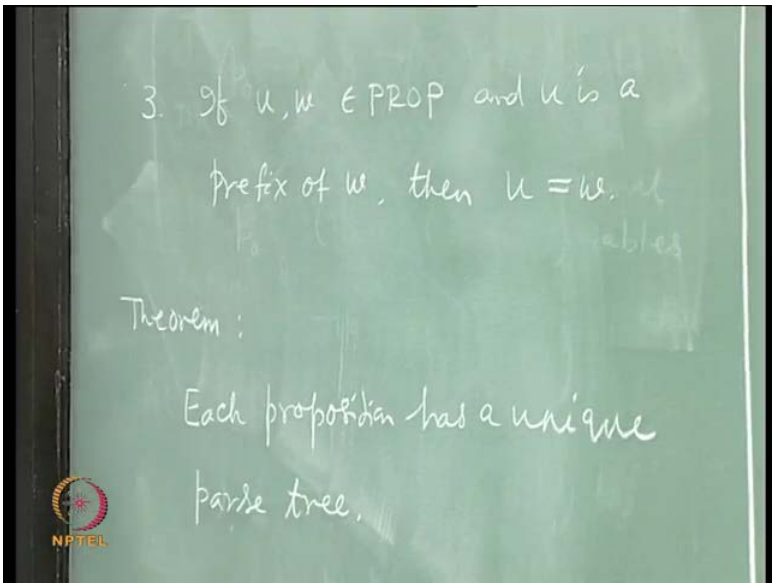
Student: Greater than equal to.

Greater than equal to is fine, is it strict? Is it greater than always? If it is a proper prefix? Right, it can be the prefix of a string, also prefix means you are reading it from the left side stopping at somewhere. If you stop only at the end you get the whole, then you have the number same, but before that number of left parentheses must be bigger than number of right parentheses provided there are parentheses. Else, you can have only negation symbols and pi; is it right? Take an example of a proposition like not p0 there is no parenthesis. It had any prefix? Of it we get only not, is any proper prefix? Is that clear?

Now, how do you form it? Second one? It is about the prefix of a proposition, yes. If u is a prefix of w, and w is a proposition, then for the second case, u can be, what you are talking of, a proper prefix. Let us take u is a proper prefix then what happens? That means the u can be empty string that is one triviality. You have to see: u can be a sequence of not, in a sequence of not, is number of left parenthesis in u is strictly greater than number of right parenthesis in u? Yes, greater than.

Now, if you do not take a proper prefix, then there is the other possibility that u can be equal to w, let us remove this and make it proper; by removing this proper or u can be equal to w, in fact this last property can be formulated in a different way. You can just say that if u and w are both propositions and u is a prefix of w then u has to be equal to w it is slightly cryptic.

(Refer Slide Time: 37:15)



If you take both u and w as propositions and u is prefix of w then u has to be equal to w. No proper prefix of a proposition can be a proposition this is what it says. You can utilize these properties to prove that the grammar is unambiguous. Now, what is that we are going to prove? Lets formulate it first; grammar is unambiguous; that means if a proposition has been formed then its parse tree is unique; it cannot have a different parse tree than what you have got already. This means if a proposition has been formed then the why it has been formed is the only way it could have been formed. If a proposition is already formed there is only one way you can read it.

All these things tell the same thing. That is why the theorem you are going to prove is called unique formation, unique parsing theorem or unique readability theorem. Whatever way you want to look at it. And, let us take the theorem first; it says each proposition is uniquely parsed. You take any proposition, proposition means it is a string over the alphabet having possibly, some parentheses, some connectives, some propositional constants, and some propositional variables; it is finite in length.

If we count the symbols, it will be a finite number, because it is a string; now you say it is a proposition means all the rules have been applied. Not exactly all the rules, only some of the rules could have been applied. But, only from those rules we have applied some and after application, we have got another string as a proposition, now your concentration is in that particular string what you have obtained. Now, you say why you

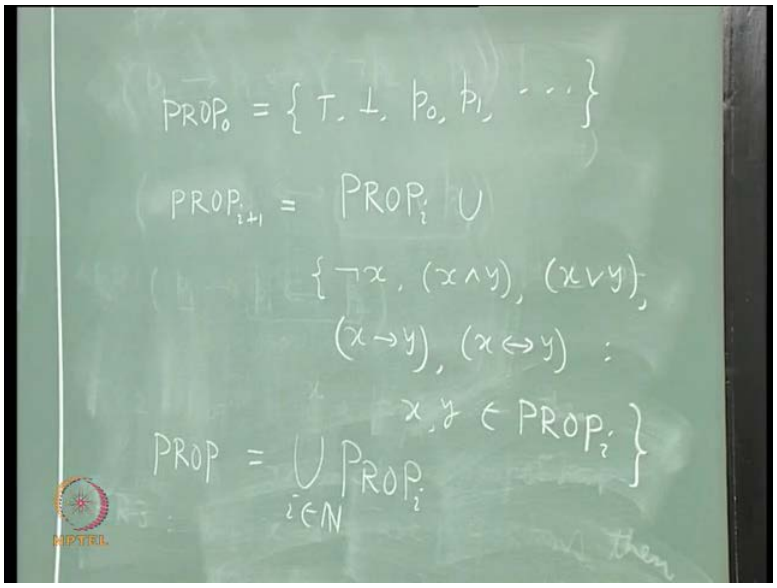
have obtained it is the only way you could have got it; no other way the same thing could have been arrived.

What is the meaning of this way, the way you have obtained? It means, what it means, we have identified from the atomic propositions, which one to choose, top is there, bottom is there, p0 is there or p100 is there, how many p100s are there and so on. First stage that you have chosen the atomic propositions, after choosing the atomic propositions you identify their occurrences, this is first occurrence of p0, this is second occurrence of p0 and so on; in an order it has been written.

Then you combine which or with what, using which connective, that combination is done. After the combination, you go to the second step, you combine the newly updated ones with the old ones or all the new ones and so on. Leave and continue, that is the way you have obtained, now you are going to say that, that is the only way any one could have obtained this proposition, nobody else can obtain it in any other way.

Here, you define a set of propositions in a different way. It tells you the meaning of `this way of obtaining', you could have defined the propositions in alternate different ways instead of the formation rules. Formation rule 1 says that all atomic propositions are by definition propositions, write that as PROP0, without any connectives. Then you go to PROP1, which will have at best 1 connective, then PROP2 including PROP0 and PROP1, you put another connective, you can really generate that way, let us see it that way.

(Refer Slide Time: 42:16)



See, I can define PROP subscript 0, let us say this is equal to top, bottom p0, p1 and so on; then I can say if PROPi has been defined we can define PROPi plus 1 equal to PROPi union? Something will be generated basing on elements of PROPi: not x or you can say x and y, or you can say x or y, or x implies y, or x iff y, such that x, y are in PROPi. You are generating it level by level, this i, here in this u, is the depth in that parse tree. Can you see the leaf level is covered by PROP0, go to the next level that will be covered by PROP1 and so on?

That gives you the depth, then finally you say that PROP is a set of all propositions, it is equal to the union of all these. Recall N is the set of natural numbers including 0. This is what you are doing when you do parsing; you are doing it level by level in the trees, this is also another alternative definition of PROP.

Then it is required to be proved again, proof will be by induction that, what you generate, here is a proposition, and then from the other side every proposition is there in some level because it is finite that also can be done. Now, what about the proof of unique parsing? You have to think of it, see to proceed with this. What we had done till now? Yes? One sentence I want. We have simply defined what PROP is, we have understood what unique parsing is.