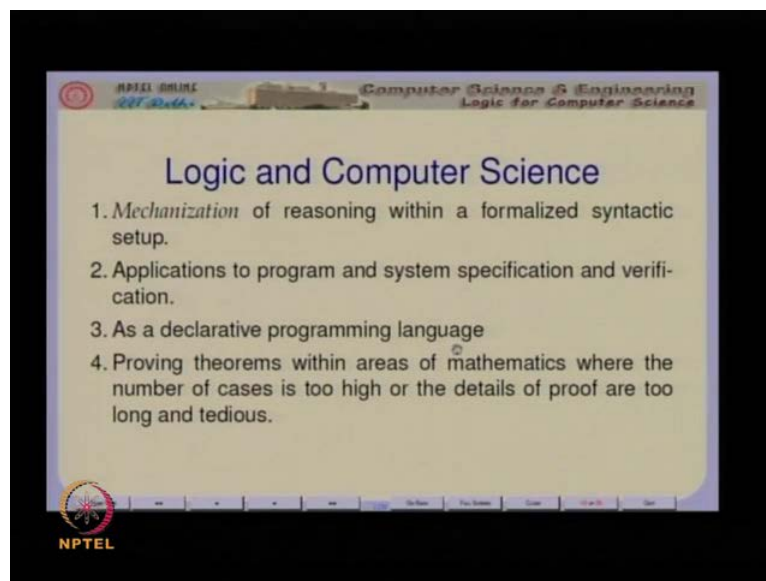


Logic For C S
Prof. Dr. S. Arun Kumar
Department of Computer Science
Indian Institute of Technology, IIT Delhi

Lecture - 02
Propositional Logic Syntax

So, let us just briefly recap. So, I just say the whole point about the logic and computer science is that we have somehow interested in mechanization.

(Refer Slide Time: 00:41)



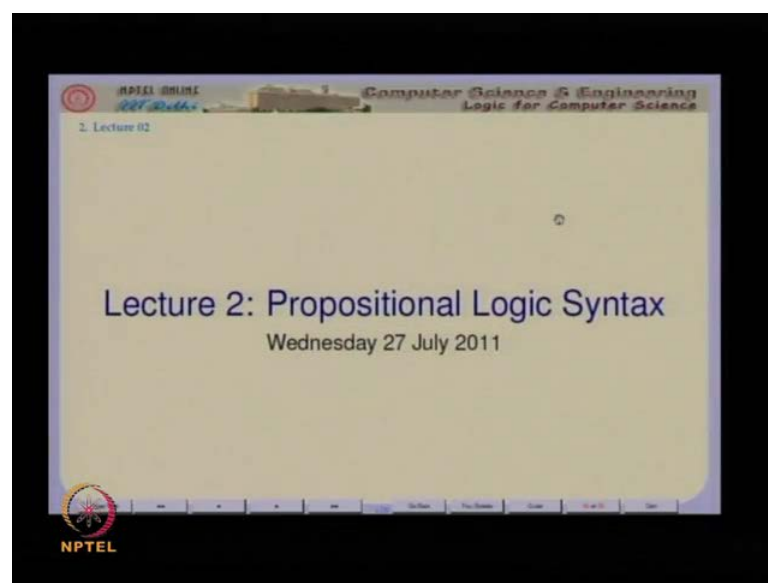
It is sort of fundamental, can be used for checking getting clarity about the nature of proofs, in mathematics and all mathematic aligned subjects. And of course, within computer science logic has a large role to play I mean it has applications to program and system specification and verification. Logic is also used as a declarative programming language many of you must have studied in studied prolog in programming languages. It is there equation logic is widely used in constraint programming.

And of course, in this there this hot and important area of the improving with in areas of mathematics and outside. So, let us start with formal logic will let us start every body of course, knows propositional logic, but it is necessary for us to establish notation and notation and conventions all and so far. So, what I will do is I will weekly go through propositional logic and then we will start with we will spend a lot more time on logic and

some other logics. But propositional logic forms a basis for all other logics therefore, it is important to sort of locate it.

So, it is important to establish notation especially because I have peculiar believes and notation that is one reason probably the most important reason. And therefore, much of my notation may not be found in normal text books so, but however we live with my notation because is obviously better than all other notation. So, and also we need what I will do is I will take a distinctly algebraic view of the whole subject.

(Refer Slide Time: 02:58)



In fact logic is essentially an algeberzation of reason, and no book actually tells you that, but it is implied. And therefore, I will take a distinctly algebraic view point. The whole idea of algeberzation as it was inspired, essentially through Euclidian geometry was that it was much harder to deal with the nature of proofs and there correctness, with in a geometrical reasoning system. However if you algebrised it somehow things become easier to at least verify whether, whether certain proofs are correct firstly.

Secondly algeberzation means also to it was never specified by anybody even logicians. Algeberzation also means using a formal language and programming languages for example, and all formal languages are actually inherently algebraic in nature. So, there are two primary viewpoints in mathematics, one is algebra and the other is geometry. And a large part of the effort of formularization of mathematics and these two forms

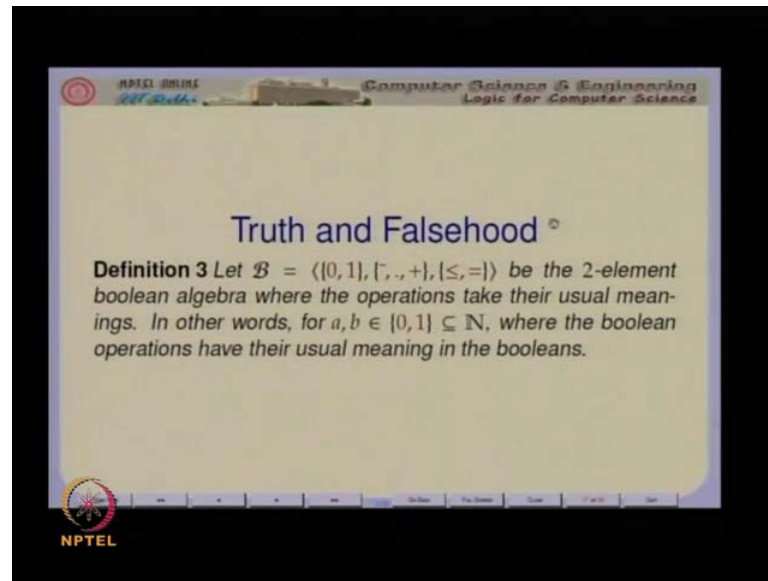
have actually kept on. So, you can think of you can think of in this way when coordinate geometry came that was essentially in algebraization geometry.

So, you could I mean so, the notions like lines and lines and points and so on could essentially be specified in some equations. So, as algebraic equations so, you so, the relationship between algebra and geometry has always been I mean there are looked different the nature of proofs is different. But they always interacted with each other so the notion so but notion by logic is algebraic in that sense that is a formal language which basically this is by know all of you know that all programming languages are essentially some kinds of term algebras, right?

All formal languages are some kinds of algebras you can think of any programming language as essentially as set of basic constructs with some operators. The programming language constructs are essential with operators. So, the moment and then you can think about equation reasoning, you can think about specifying things as identity equations is a general term for also identities, it is not just for solving equations. On the other hand the concept of recursion and programming language is it is itself is actually, the concept of solving an equation in an unknown. Namely that name that is being specified recursively.

So, there is any formal language is actually algebraic in nature what logic gives you from geometry is really the notion of the assimilation essentially inspired by Euclid's elements. Essentially the idea that there are few basic axioms of postulates and everything else should be derived as a theorem from this basic postulates and axioms. So, that is the interactions between algebra and geometry and since where essentially dealing with it as a formalized system we look at it algebraic mostly algebraic to set the stage for the algebra.

(Refer Slide Time: 07:08)



The first thing of course, as I said yesterday was that we will talk about truth and falsehood. Truth and falsehood as far we have concerned, the underlined model of the truth and falsehood is just two values right belonging to a set 0 and 1. And I am using brown color because it is very down to earth. So, 0 and 1 are the Boolean set you can regard it as a being embedded within the natural numbers, or you can regard it as a being separate from the natural numbers. But either way it is essentially a semantics will all be specified in terms of this Boolean algebra. And this Boolean algebra I have also defined it in some peculiar pattern.

Well that this is inverse operation and there is this product operation and there is this summation operation and then there are these two relations less than are equal to. So, any algebraic system, as you know consist of a carrier set a collection of operations and a collection of relations and that is what our Boolean algebra. And Boolean algebra of course, is specified as set equations, set identities and remember that we are going to we are doing mathematical logic in which we are using the tools of mathematics itself to specified logic, also to deal with the notions of reasoning in logic.

So, we have the two element Boolean algebra whose, where the operations take there usually mean. So, you can regard this dot either as a product operation or else the minimum, minimum operation minimum numbers taken from this set. And this plus of course, is Boolean summation and not addition, so there are operators with in the

Boolean algebra. So, we just as take for granted this algebraic system and all its properties.

And this algebraic system and all its properties they are going to form the basis of the notions of truth and falsehood, on which the language of logic is going to be built. So, the language of logic and the meanings, there is syntax and there is semantics. So, if it is syntax we have to it the semantics the meanings will be expressed in terms of these notions of truth and falsehood. And this is the most concise way I could think of representing truth and falsehood.

Now, so you have you all studied these Boolean algebras and their properties and various things. So, if for example, you could think of I mean there is there is also another kind of duality which goes on what that is that a model construction, and in a certain way the hardware circuit that you have studied about are really like models of this Boolean algebra. But when you study hardware circuit as models of this Boolean algebra there you are looking at this Boolean algebra itself as a language, the expressions of this Boolean algebra as a language whose meanings are given through the working of those circuit and gates or gates whatever not gates so on.

So, on the other hand what we are going to do is we are going to take this algebra itself as well understood and we are going to build language, whose meanings we are going to express in terms of this algebra. So, the question of so in mathematical logic therefore, depending on the view point, certain things can be models or they can be object of study.

(Refer Slide Time: 11:43)

The slide is titled "Truth and Falsehood" and is part of an NPTEL course on "Computer Science & Engineering Logic for Computer Science". It lists several mathematical operations and relations:

- $\bar{a} = 1 - a$ is the unary boolean inverse operation,
- $a + b = \max(a, b)$ is the binary summation operation which yields the maximum of two boolean values regarded as natural numbers,
- $a \cdot b = \min(a, b)$ is the binary product operation which yields the minimum of two boolean values regarded as natural numbers,
- $a \leq b$, and $a = b$ denote the usual binary relations "less-than-or-equal-to" and "equals" on natural numbers restricted to the set $\{0, 1\}$.

The slide also features the NPTEL logo at the bottom left and a navigation bar at the bottom.

So, these operators I am going to define them in terms of in terms of the natural numbers as far if I think of them as essentially the subsets of the naturals. I have to just ensure however in any algebraic system, I have to ensure that the operators are closed on the carriers right. We should not we should not have to go out outside the carrier set. So, these operators I will just think of them as this is the most it is a convenient way to think of them. So, this minus that you see is not a really a part of the Boolean algebra, but this is actually a part the natural numbers the subtraction operation, and the natural numbers max and mean are also operations on natural numbers.

And then of course, the notion the notions of less than or equal to an equal to an Boolean algebra or well understood, though equality is actually a very (()) object certain relation and we will have the need to define various kinds of equality. So, for example, even in the case of programming languages, when you actually copy a program straight of let us say from one file A to another file B, A and B are equal in certain senses.

A and B are equal in the sense that they are both synthetically identical, that is the kind of equality there A and B are A and B are also equal because, since they are identical copies the meanings of the two programs are also the same. So, they are so they are equal as programs, they are equal in terms of their program behavior they are asymptotically identical, there also equal in their programming equation. How are the two files A and B

are not the same file. So, in terms of are A and B the same object they are not they are basically different objects and therefore, they are not equal in terms of their identity.

A simple distinguishing feature of A and B is that they would have different I nodes in the operating system, it is they are not identity in the same element. So, equality has very supple meanings, depending on how you look at. So, there is identity of elements, there is asymptotic identity, but between different elements. There is a equality that comes just from the fact that I am calling the same object by two different names. So, there is copy is one thing soft link is another kind of equality, hard link is the third kind of equality and they all have certain differences. Most of you also done a course on operating system and you already use you use soft links and hard links.

So, you can see the each of them has actually has a difference, so aliases is so two names aliases for the same object, in which case that is the neat quality that goes even higher than asymptotic identity, they are not only asymptotically identical they are also the same object. At some point these differences between the various kinds of equality will also become important. Unfortunately in mathematics truth equality is not greatly distinguished for all it is supple variations. So, when we look at this Boolean algebras model, we are looking at that equality within with in this Boolean algebra, whatever Boolean algebra says are identical there identical and that is why I have this brown.

So, my color coding is also going to ah sort of distinguish this certain differences. So, this equality is the mathematical equality defined on the Boolean algebra. So, I hope the colors are clearly visible right now does anybody have problem with the colors. So, now you have these two less than or equal you have this equality on the Boolean algebra, and you less than or equal to where this less than or equal to comes actually from the natural numbers. So, 0 is less than or equal to 1 and 1 is less than or equal to 1 or 0 is less than or equal to 0 that is of the set.

However the Boolean algebra is a peculiar thing because I am using it as a representation of truth and falsehood, it is possible to construct these consider these relations less than or equal to and equal to, it is possible to look upon them also as operators, And I do not mean as operators in the same sense in which any relation in a program is regarded as a Boolean function. I do not mean in it the same sense it is because it is Boolean that this relation has a Boolean is also an operation.

(Refer Slide Time: 17:46)

Computer Science & Engineering
Logic for Computer Science

Extending the Boolean Algebra

While \leq and $=$ are binary relations, it is possible to define corresponding binary operations as shown below.

Definition 4 For $a, b \in \{0, 1\}$ define

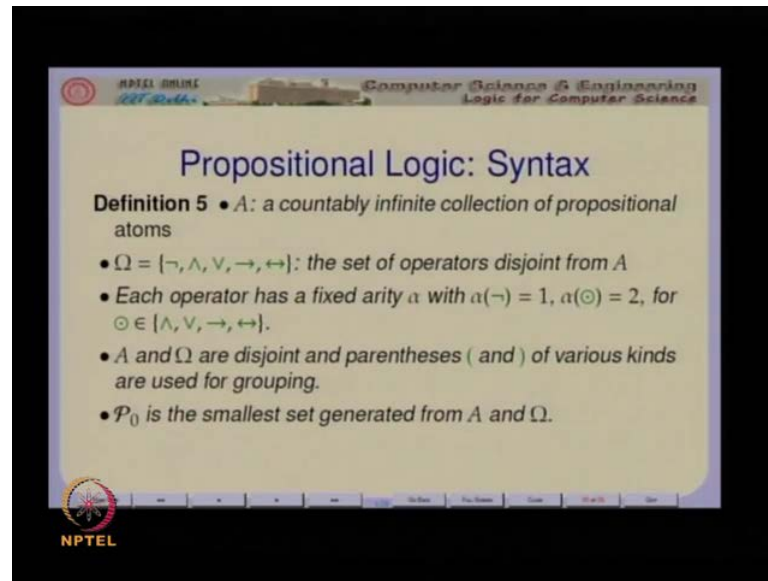
$$a \leq \cdot b = \begin{cases} 1 & \text{if } a \leq b \\ 0 & \text{otherwise} \end{cases} \quad a \doteq \cdot b = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

NPTEL

So, we will use a dot in front of these relations to regard them as operations, this is the problem and this is one problem with me then this sense that these are not the things that most books which specify. Whereas I am actually what I am trying to say now is that is these less than or equal to dot and equal to dot. I have essentially and algebraic system, in which there are five operators instead of that, instead of three operators and two relations I can think of them as five operators whenever I want to that is the convenience that I will that is flexibility between that I will exercise, whenever I need.

So, this is the interesting thing, so $A \leq \cdot B$ where less than or equal to dot is regarded as binary operation and it is equal to 1 and if A is actually less than or equal to B and 0 otherwise and similarly, $A \doteq \cdot B$ is 1 if A is equal to B in the Boolean algebra otherwise it is 0. So, that is like actually it is since they are talk they are notions of truth and falsehood with 0 representing false and 1 representing truth. We can also look at the truth and falsehood of this relations themselves with in the Boolean algebra of operators.

(Refer Slide Time: 19:27)



The slide is titled "Propositional Logic: Syntax" and is part of an NPTEL course. It contains the following text:

Definition 5 • A : a countably infinite collection of propositional atoms

- $\Omega = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$: the set of operators disjoint from A
- Each operator has a fixed arity α with $\alpha(\neg) = 1$, $\alpha(\odot) = 2$, for $\odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.
- A and Ω are disjoint and parentheses (and) of various kinds are used for grouping.
- \mathcal{P}_0 is the smallest set generated from A and Ω .

The slide also features the NPTEL logo in the bottom left corner and a navigation bar at the bottom.

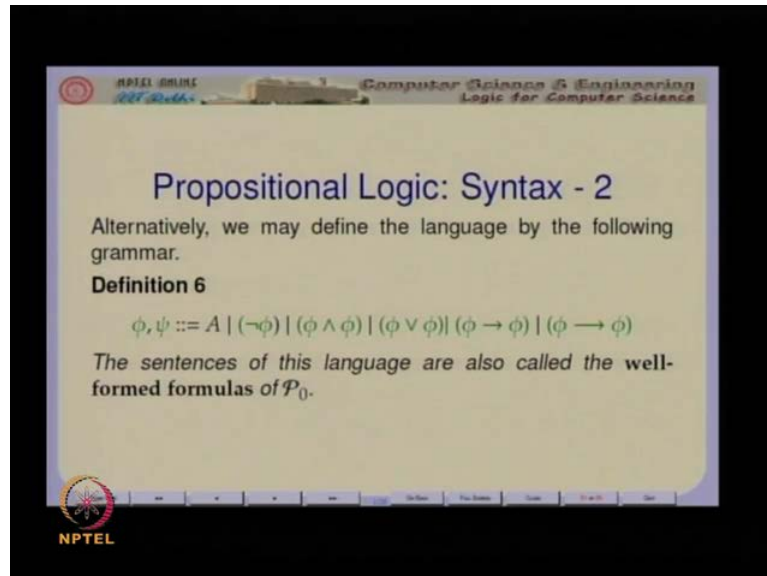
So, the actual so we will start with propositional logic and establish your notation, we start with a set A which is a countably infinite collection of propositional atoms, if you like there is another word for it. So, we have so we are essentially defining a language, and we are going to define it algebra. So, you have a collection A of atoms accountably infinite collection everybody understand count ability infinity and so on. So, it is count ably infinite so it is not it is not unaccountably infinite for example, and it is not finite it is not finite because every time I will require a fresh atom, I should be able to get on.

And then I have a collection of operators Ω and I am going to represent these propositional logic elements in green. So, this is the notation I am going to use for the various operators, the first one is not, the second one is and, the third one is or the fourth one is a conditional if then, and the fifth one is the by condition. So, these are the set of operators and they are disjoint from A . And each operator has a fixed clarity α with a negation being a unary operator and all other operators being binary.

And of course, in addition to this of course, we will use various kinds of parenthesis as grouping for grouping, and for what we say is that \mathcal{P}_0 is the smallest set generated from A and Ω , there anybody who does not understand this last sentence. You do not understand that. So, what we will do is we will give an alternative definition in terms of ((Refer Time: 21:50)) that is standard practice in programming languages. So, we will

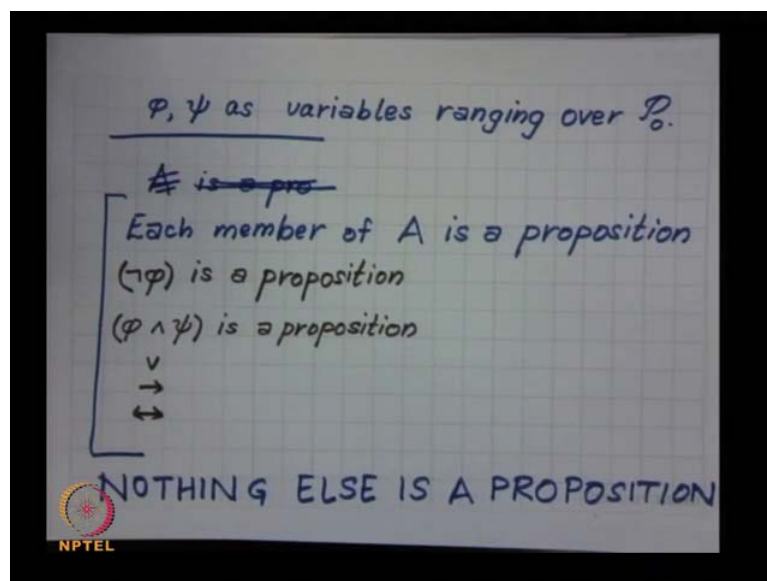
follow that right so then I will come back to what this means by the small set generated from A and omega.

(Refer Slide Time: 22:06)



So you are familiar with the ((Refer Time: 22:08)) for defining the syntax of the programming languages, no? It's like this listen carefully to what I am saying, so this one first thing says that I am going to use.

(Refer Slide Time: 22:30)



Phi and psi as what are known as variables ranging over p naught. So, phi and psi represent typical elements of this language of propositional logic. So, it is like let phi be

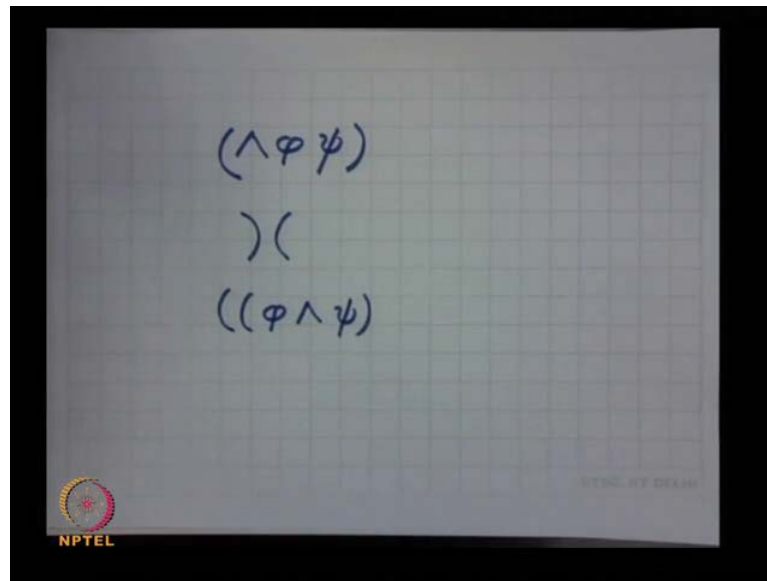
an element of language of proposition logic, let us ψ be another element of propositional logic then. So, the ϕ and the ψ occur on the left hand side to essentially specify how I am going to construct propositions from the basic elements. So, A which I wrote in green on this slide, A is a proposition I am sorry each member of A is proposition.

And if you look here the next clause essentially says so this A here essentially says that every member of A is a proposition. So, that means we started with the collection of propositional atoms and that is itself a subset of the set of propositions set of elements of this language. The next class says that I take any proposition and let ϕ be any proposition and if I put a negation, then this is also a proposition. Now, for the binary operators, essentially what I am saying is if ϕ and ψ are propositions then this is a proposition.

So, similarly, I can say for or, arrow and so, what are we doing? Starting from the countably infinite set of atoms A , they are constructing proposition compound propositions from the atomic propositions by using these operators. And what we are also saying and this needs to emphasize, be emphasized is that besides these nothing else is a proposition. So, what we are saying is that you cannot take any junk elements from outside and combined them with the operators of the propositional logic in order to get any proposition, is it clear? So, this is how programming languages are defined by this is known as the form I have actually modified the form modified for the purpose of just defining the language in a brief in a concise function.

So, this so this notion smallest is that suppose if you don't restrict yourself to A , you took some set larger than A and started generating compound propositions, then this smallest world's smallest ensures that all propositions which contain all elements, outside the operators outside the set A are all thrown out, even within the set A and the operators if you construct something like this.

(Refer Slide Time: 27:42)



This is also thrown out, this is also not a this is something that can be generated by this right. So, essentially what we are saying is that this the language has to be defined in such a way that the compiler can parse any compiler for this language can parse the language meaningfully, it is like saying that it should be compilable. So, this last sentence nothing else is a proposition and this phrase is smallest set generated, all essentially say that it should be compilable and there should be no parsing errors. So, this is so these are known as sentences, so in fact propositional logic is also called sentential logic. And their main motivation is that somehow since we are interested in truth and falsehood.

We are interested in statement about which, it can be stated whether they are true or they are false. So, it does not include statements, even from the natural language which are not of this form. So for example, it does not statements which are like questions, it does not include statements which are like commands. You know go and do this home work I mean that is a command to which normally you cannot assign a value of true or false. So, it refers to essentially comes from the linguistic philosophy in school, it refers to that subset of natural language consisting of those full sentences, for which a truth value may be assigned, for which it is at least theoretically it is possible to say, whether they are true or false.

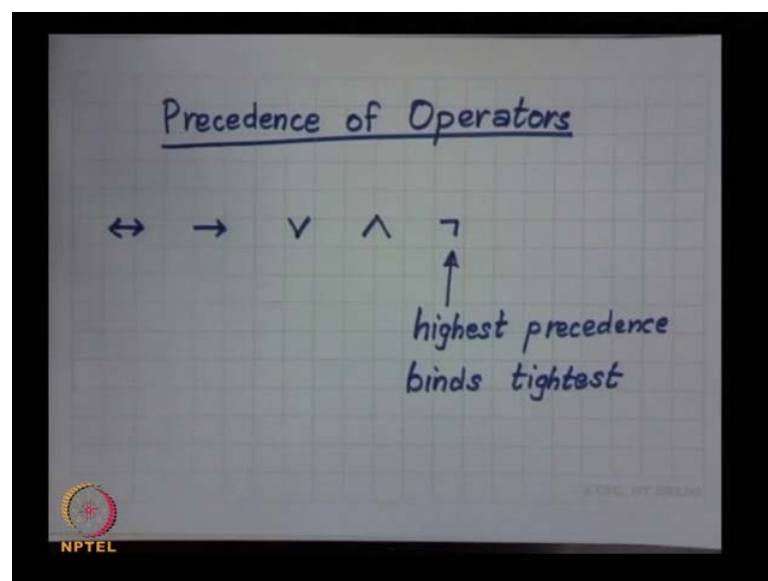
So, that is what our syntax is going to, are there any questions? Do you have any questions?

Student: ((Refer Time: 30:30))

A sentences let us say a string of the language, however a sentence is a string of the language, but of course, the strings of the language contain in addition to elements from the set of atoms and the operators. They also contain these parenthesis so, for example. So, this so just as this I mean you cannot have sentences like this for example, ya or and you cannot have sentences like this for example. I mean the parenthesis have to match this and that on the usual the usual things that are associated with the formal language.

One of the reasons for using parenthesis is to is to remove ambiguity, the safest thing to do is to use a fully parenthesized language and what this BNF this BNF you are all familiar with the BNF and it stands with the backus naur form and it is a standard thing that is used for design defining the syntax of all programming languages. So, this BNF essentially gives completely parenthesized notation for proposition logic r. And in general of course, a parenthesis, a completely parenthesis notation is safe, but in general when you when we actually write we would not like to put parenthesis. So, the normal thing is to define. I will just proceed ahead, I will just proceed ahead with the normal thing is to define a precedence of operators.

(Refer Slide Time: 32:45)

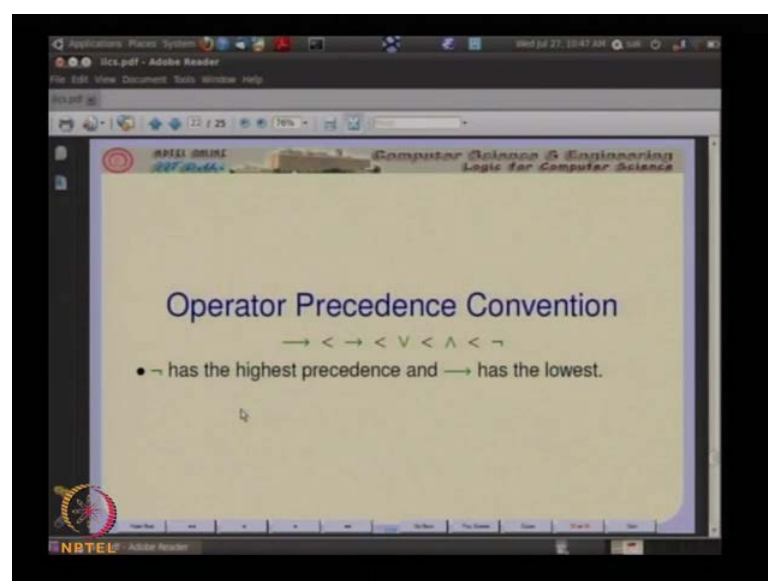


So, let us look at the precedence, this is true starting from your school mathematics. So, what we are essentially saying is that this by conditional has the low, so we will follow the convention that this by conditional has the lowest precedence, followed by the conditional, followed by or, followed by and, and the negation has the highest. So, it is often stated that this is the highest precedence, it is also it is also used in the form that this binds tightest. I am most of you who have done a course on programming languages would be familiar with this thing right?

Going back to your school mathematics there is there is this precedence rule, precedence rules also apply for arithmetic expressions. In the sense that additions has lower precedence than multiplication and the negative bind tightest. So, you can so that is of course, a unary negation, there is a subtraction binds like addition, but you have to and you use grouping of you use brackets and parenthesis, only when you want to over write the precedence, right?

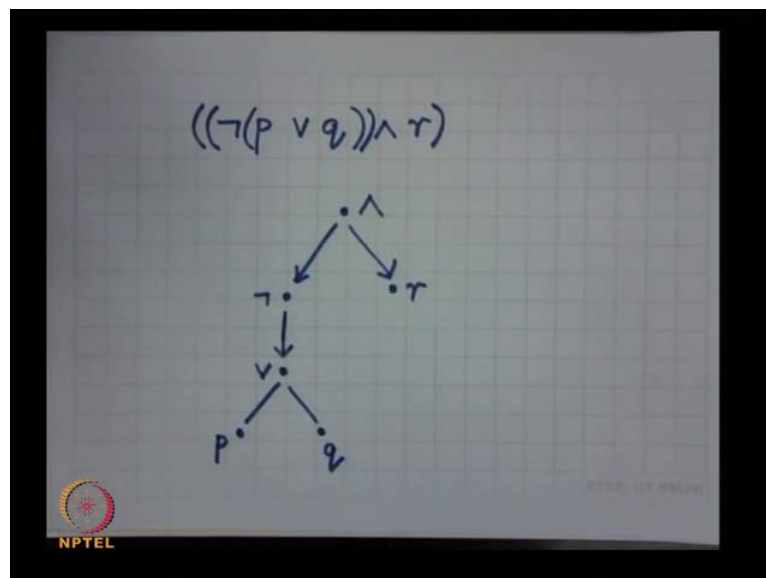
All that is of course, purely matter of dealing with strings, but we do not want to actually look at this way, we want to look this film you want to look at all these things essentially instead of looking at the strings and breaking your heads about parenthesis and so on and so forth. The simplest thing to do is to look up on them as essentially trees. So, you can think of since all these well formed, this are this are what are what are known as well formed sentences.

(Refer Slide Time: 35:18)



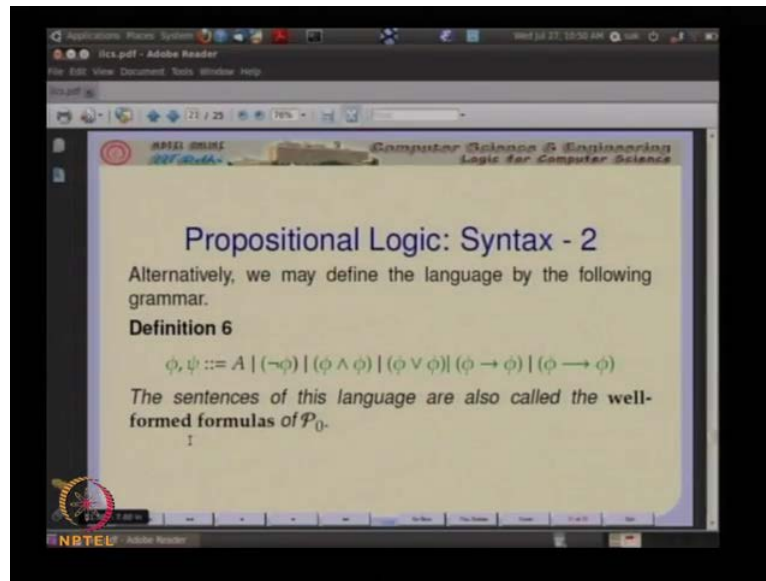
So, here the operator precedence convention, which essentially says that negation has the highest precedence and this should be the I am sorry, this is wrong by conditional which have written here has the lowest precedence here. So but more importantly as I said this kind of thing is dealing with strings and parenthesis and so on so forth is completely is a completely wasteful activity, which can be taken care of by any machine. On the other hand what we would like to think of these well formed formulate, these are known as the well formed formulate. This well formed formulate we would like to view them as trees, essentially abstracts syntax tress. So, you all of you have studied in programming languages, the notion of an abstract syntax tree. So, the when you, when I construct a well formed formula like.

(Refer Slide Time: 36:24)



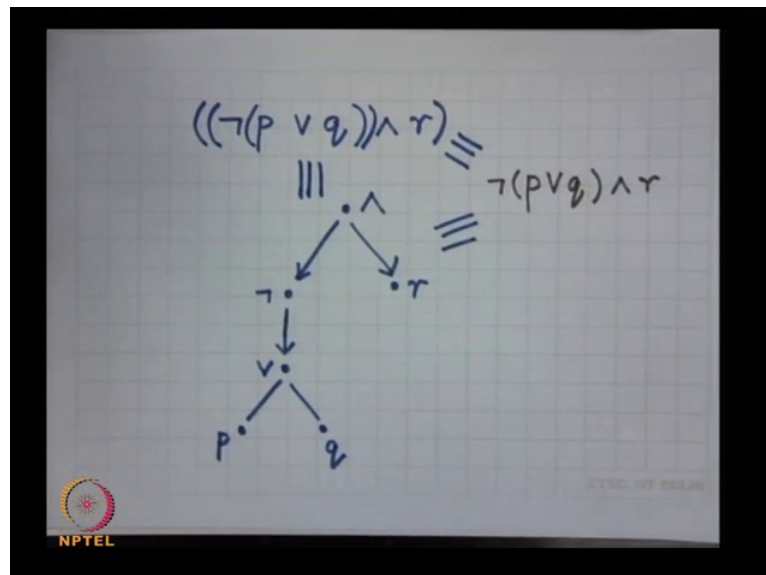
Let us assume that small p is an atomic proposition, small q is an atomic proposition and r is an atomic proposition. Now, this is essentially a well formed formula, actually this is a completely parenthesized formula, and what this represents is a tree is an abstract syntax tree, in which I have as a root node the and operator. And as which has two sub trees, the right sub tree is essentially just a leaf node consisting of the proposition r, the left sub tree is a negation under which there is a sub tree, whose roots is or and of two atomic propositions p and q. If you view, essentially we can think of parenthesis...

(Refer Slide Time: 37:56)



As just being used to render linearly these trees are meaning are these trees, I mean what we mean are these obstacles syntax trees, but in order to able to write in a linear form we use parenthesis and typically with this precedence convention, we would write this linearly essentially just this.

(Refer Slide Time: 38:25)



Naught p or q and r and this therefore, parenthesis are not necessary not really necessary, except when you want to ensure grouping in a particular order. So, these two well formed formulas are going to be considered, syntactically I equal because the only

difference is in the because the only difference is in the use of in the removal of reddened parenthesis, right? According to the precedence convention, so we will think of these as essentially being syntactically identical.

So, we will what we will do now is shall this is going to be a basic tool set. And whenever I mention a proposition it is up to syntactic identity. Basically think of this way, two propositions written differently are syntactically identical if they have the same abstracts syntax tree, is that fine? So, as long they have the same abstracts syntax tree in which no parenthesis are used, we consider them to be syntactically identical, right? I will stop here today.