Logic for CS

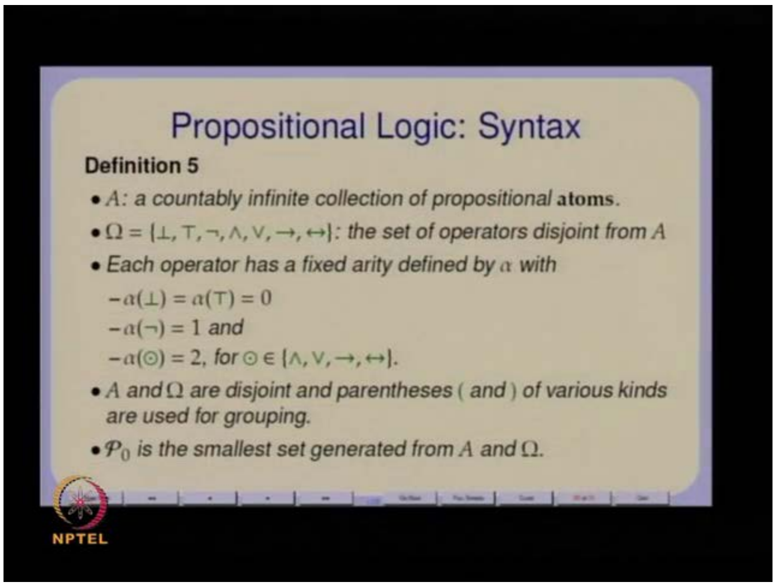
Prof. Dr. S. Arun Kumar
Department of Computer Science
Indian Institute of Technology, Delhi

Lecture - 03
Semantics of Propositional Logic

So, before we will talk about the semantics of propositional logic, as I said you know the initial ideas to be as boring as possible. So, that we fix all the notation in a certain way, in a certain way peculiar to me peculiar to logic, peculiar to distinguishing between various things, which are normally not distinguish in various books easily. So, they create some kind of problems, so we will go about it slowly and we will start coming up with logical concepts from proposition logic, which can be applied to first order logic later.

So, one of the first things in any language that you have to do is to define its semantics, the meaning of the constructs of the language. And so before I get on to that actual semantics of propositional logic, let me just briefly go to the previous lecture and recap some of the things there. So, we have this Boolean so I have a colour coding, brown is because it is down to earth then that is what things are supposed to mean. And of course, and we are talking about a logic a language, when we are talking about a logic, we are talking about a language and the sentences of this language. We are talking about a formal language whose sentences are essentially trees like abstract syntax trees and so there is a colour coding of green, for essentially objects belonging to this object language.

(Refer Slide Time: 02:11)



Propositional Logic: Syntax

Definition 5

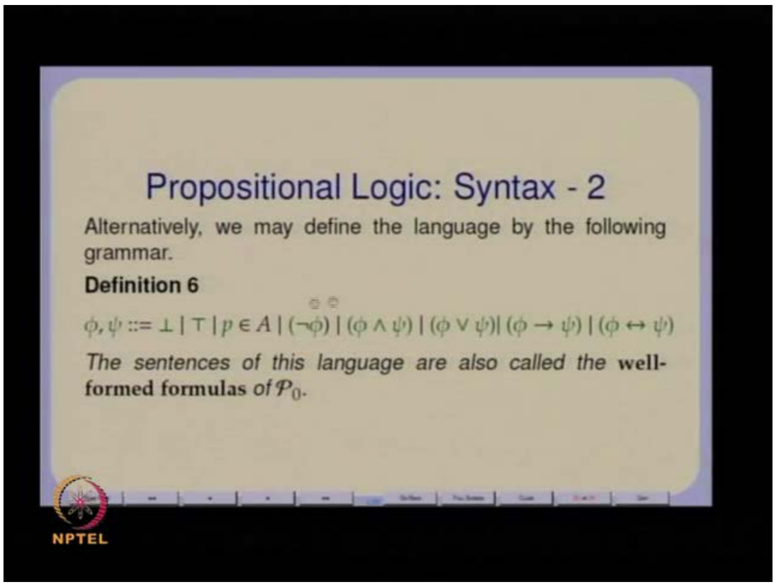
- A: a countably infinite collection of propositional **atoms**.
- $\Omega = \{\bot, \top, \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$: the set of operators disjoint from A
- Each operator has a fixed arity defined by α with
  - $\alpha(\bot) = \alpha(\top) = 0$
  - $\alpha(\neg) = 1$ and
  - $\alpha(\odot) = 2$, 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.

So here in our case the object language is propositional logic, and when we talk about it of course, we talk about it in a mixture of mathematics and English. And that is the meta language of the instructions, so there are already two different languages. So, there is an object language, which is the target language of our discussion and in order to talk about the target language, we need a meta language. And that meta language since its is mathematical logic it is English with concepts of a mathematics since we are going to treat logic as a sub discipline of mathematics, so it essentially is a mixture of that.

So, our meta language is always going to be essentially in black colour, right? So right now we have two colours and even here the brown colour for the semantics is just a distinguishing colour, it is not a separate language. So, we have we will keep worrying about the fact that when we are talking about various relations, whether they belong to the object language or do they belong to the meta language.

So, unless I made serious mistakes the object language will always be green, so the relations in the object language will be green. And the relations in the meta language will be either black or some other colour depending on in the case of for example, in the case of equality of Boolean algebra that is it is that is a brown colour right, it is a relation on that algebra. So, I have made that algebra brown in colour so, we have to and we will we will already come across various kinds of equalities subtle differences, which we need to distinguish.

(Refer Slide Time: 04:06)



Propositional Logic: Syntax - 2

Alternatively, we may define the language by the following grammar.

Definition 6

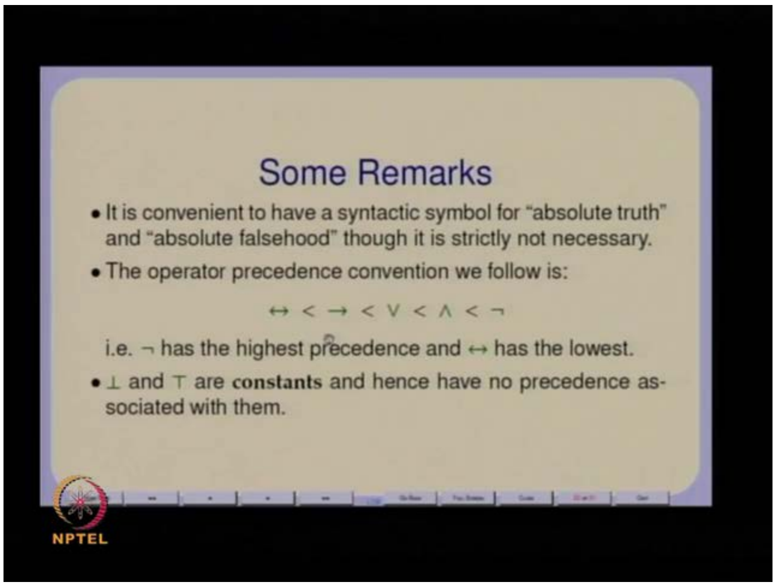
$\phi, \psi ::= \bot \mid \top \mid p \in A \mid (\neg\phi) \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \rightarrow \psi) \mid (\phi \leftrightarrow \psi)$

The sentences of this language are also called the **well-formed formulas** *of* $\mathcal{P}_0$.

So, this is the this is the language syntax and in the grand old tradition of programming languages, we express it in some form which is called similar to the which is a sort of glorified form ((Refer Time: 04:19)). I have added two symbols these are called bottom and top of course, one of the things is one of the nice things that mathematicians like to do is to create one to one correspondences. Since our mod or language of the model namely Boolean algebra, has this two elements 0 and 1 it is a good idea to give them also a linguistic representation say essentially this bottom will refer to absolute falsehood not relative.

As I said truth is can be relative and falsehood can also be relative so this bottom will represent essentially, it is essential the link in the object language it is a representation of a absolute falsehood. And top refers to absolute truth, anyway that will become clear when we give define the semantics. So, this is our language p is a general term is a general symbol p q r small p q r or things we will use for propositions, atomic propositions. And for compound propositions we will use these Greek letters like phi, si, ki and theta and so on. And of course, all of them will be objects and the sentences of the object language will be in green in colour.

(Refer Slide Time: 05:49)



Some Remarks

- It is convenient to have a syntactic symbol for "absolute truth" and "absolute falsehood" though it is strictly not necessary.
- The operator precedence convention we follow is:

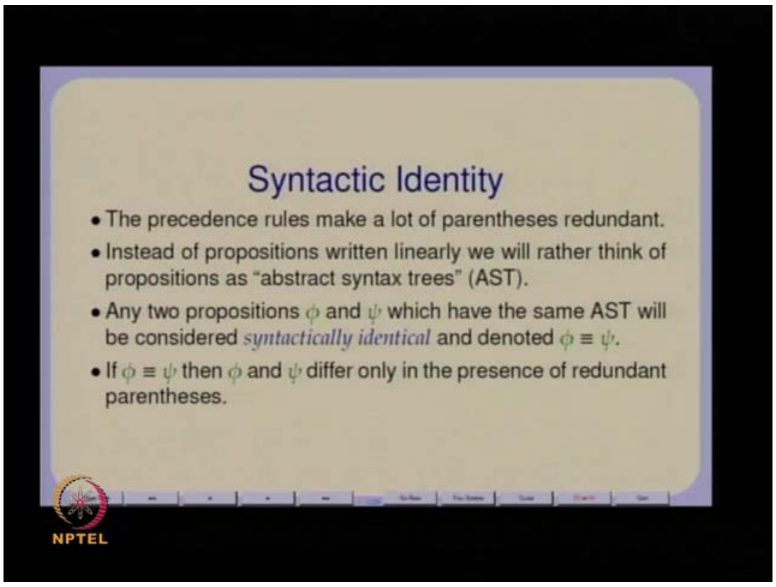
$$\leftrightarrow \; < \; \rightarrow \; < \; \vee \; < \; \wedge \; < \; \neg$$

i.e. $\neg$ has the highest precedence and $\leftrightarrow$ has the lowest.

- $\bot$ and $\top$ are **constants** and hence have no precedence associated with them.

So that is the thing so this we are when we are talking about additives of operators. So, the addition of this bottom and top is essentially as constants, in the algebra right so they have an arity of 0. So, it is not it is not there is no reason to talk about its precedence, so bottom and top are constants, and have they have no precedence associated with them. Or you can think of them as having a higher precedence than even negation 0 value operators.

(Refer Slide Time: 06:37)

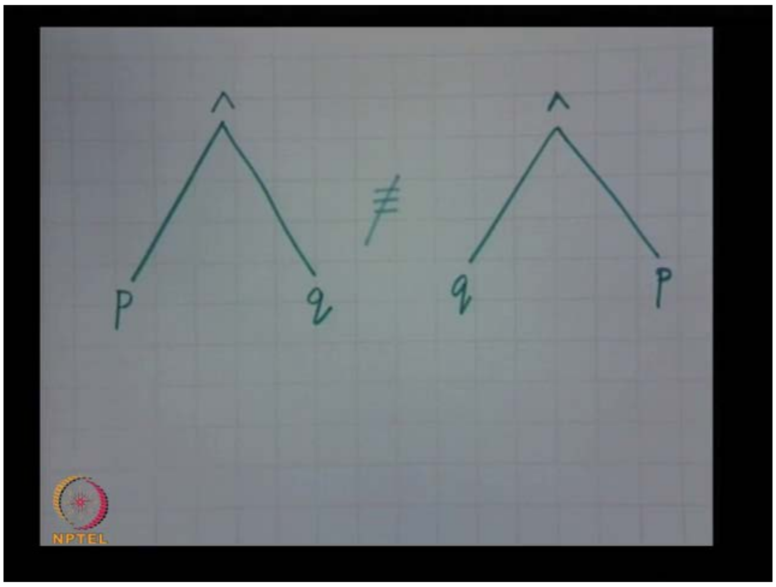


Syntactic Identity

- The precedence rules make a lot of parentheses redundant.
- Instead of propositions written linearly we will rather think of propositions as "abstract syntax trees" (AST).
- Any two propositions ϕ and ψ which have the same AST will be considered *syntactically identical* and denoted $\phi \equiv \psi$.
- If $\phi \equiv \psi$ then ϕ and ψ differ only in the presence of redundant parentheses.

As I said our the most important thing is we will be looking at these sentences essentially as abstract syntax trees because parentheses with precedence and so on. Things can look syntactically different so we will essentially say that any two propositions phi and si which have the same abstract syntax tree will be considered syntactically identical. And so when I say that I also mean that for example, this is not…

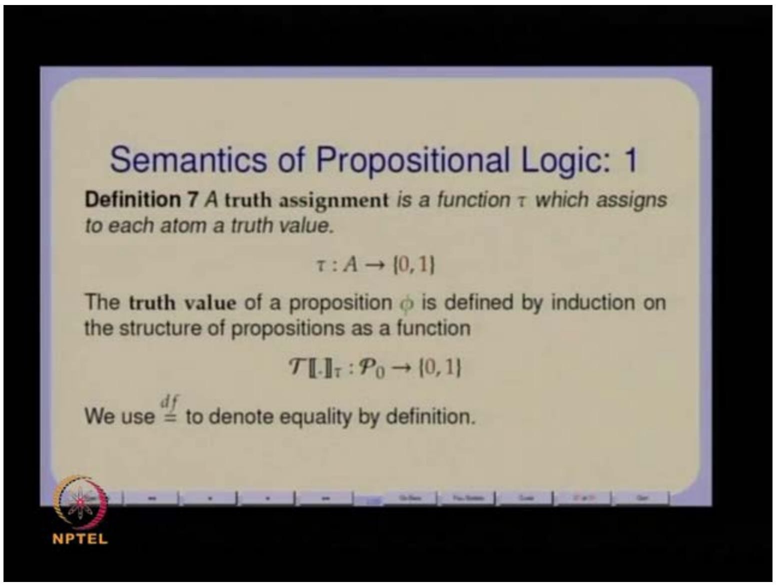
(Refer Slide Time: 07:05)



Is not syntactically identical with something like this, it is not syntactically identical with this. Let us make it clear that anything like the commutative law and so on, so forth is a derived law, before we have defined the semantics these two tress are different. So, these two are not syntactically identical right, so we cannot presume the existence of commutativity or associativity of any operators, till we have defined these. So, these two are not syntactically identical, and we will use this total segment to denote syntactic identity. Notice that it is black in colour so we are talking about comparison of two objects, in our meta language so that so, whatever is green is in the object language. This syntactic identity is a relation, we are talking about the objects so it is in our meta language therefore, its black or it is atleast not green.

So, this is this is something that is going to be important, especially if we reach if we reach a stage in this course, where we will start talking about incompleteness so on so forth then it will become important to distinguish, what exactly belongs to the object language and what belongs to the meta language. So then formally as I said we will take a completely algebraic view point, and an completely algebraic view point is essentially this.

(Refer Slide Time: 09:00)



Semantics of Propositional Logic: 1

Definition 7 *A* **truth assignment** *is a function τ which assigns to each atom a truth value.*

$$\tau : A \to \{0,1\}$$

The **truth value** of a proposition ϕ is defined by induction on the structure of propositions as a function

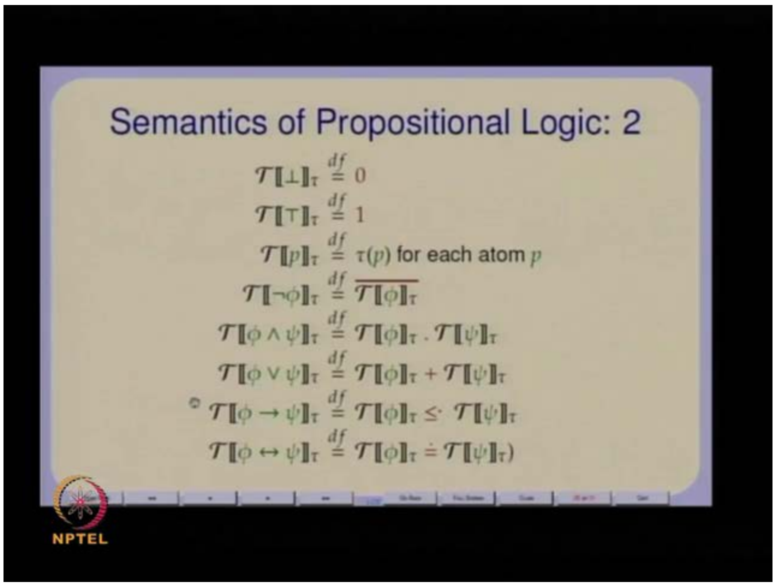
$$\mathcal{T}[\![.]\!]_{\tau} : \mathcal{P}_0 \to \{0,1\}$$

We use $\overset{df}{=}$ to denote equality by definition.

So, we talked about a truth assignment, a truth assignment is just a tau is just a function tau which assigns to every atom a truth value from that brown set 0, 1. So, basically there is the truth assignment just tells us, for each atomic proposition whether it in under the truth assignment, whether it is true or false. The truth value of proposition five is defined by induction on this structure of propositions and function, which I am calling t, and that t actually depends on the truth assignment tau. So, I am putting this subscript tau and this peculiar double brackets that you see is often in printed books is often a standard way of separating out syntax of the object language, syntactical elements of the object language from the meta language. But we will continue to use this because if you if you are going to print these things out in black and white, then you need some distinction at least here I have colour, but other ways normally.

These are in the community of programming language here it is often called semantic brackets, but actually they enclose syntax. So, the other thing is that they actually defining this function t, so we have actually a new kind of equality, a new kind of equality in the meta language which is definitional equality. So, there are there are things which are equal by definition and that is what we will use. So, we have to so at a certain level, it may not matter whether a function whether a certain statement is equal to another statement, whether a certain expression is equal to another expression either by a definition or by derivation. But at least and in the initial stages, we have to be able to distinguish between all of them. So, we will use this to essentially define the function.

(Refer Slide Time: 11:30)



Semantics of Propositional Logic: 2

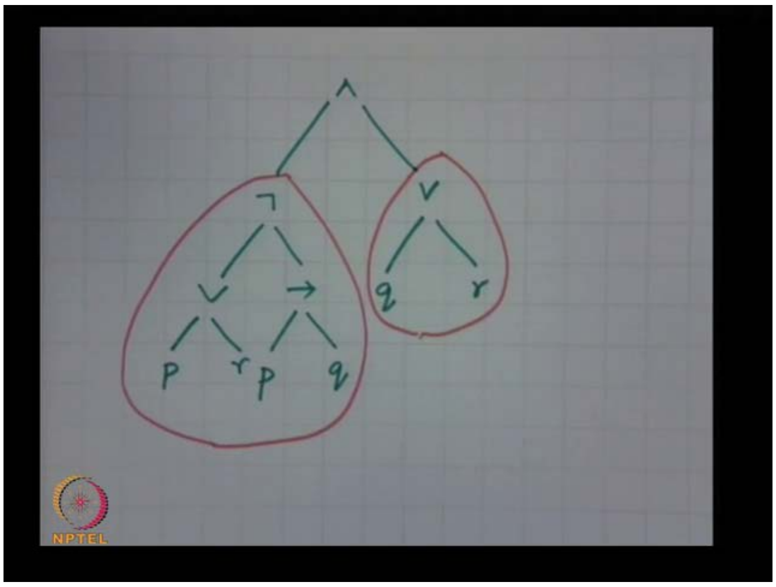
$$\mathcal{T}[\![\bot]\!]_\tau \overset{df}{=} 0$$
$$\mathcal{T}[\![\top]\!]_\tau \overset{df}{=} 1$$
$$\mathcal{T}[\![p]\!]_\tau \overset{df}{=} \tau(p) \text{ for each atom } p$$
$$\mathcal{T}[\![\neg\phi]\!]_\tau \overset{df}{=} \overline{\mathcal{T}[\![\phi]\!]_\tau}$$
$$\mathcal{T}[\![\phi \wedge \psi]\!]_\tau \overset{df}{=} \mathcal{T}[\![\phi]\!]_\tau . \mathcal{T}[\![\psi]\!]_\tau$$
$$\mathcal{T}[\![\phi \vee \psi]\!]_\tau \overset{df}{=} \mathcal{T}[\![\phi]\!]_\tau + \mathcal{T}[\![\psi]\!]_\tau$$
$$\mathcal{T}[\![\phi \rightarrow \psi]\!]_\tau \overset{df}{=} \mathcal{T}[\![\phi]\!]_\tau \leq\!\cdot\; \mathcal{T}[\![\psi]\!]_\tau$$
$$\mathcal{T}[\![\phi \leftrightarrow \psi]\!]_\tau \overset{df}{=} \mathcal{T}[\![\phi]\!]_\tau \doteq \mathcal{T}[\![\psi]\!]_\tau)$$

So, here is the definition, so now what we are saying is bottom the meaning of bottom, or here in our case in most programming languages, we talk about the meaning of the language usually as if it is an imperative language, change of state, if it is a functional language of the value of an expression. In the case of logic the notion of meaning is restricted to whether it is true or false right? That is the notion of meaning and so this function t essentially assigns regardless of what term might be, it actually gives a value it defines this bottom as absolute falsehood, and it defines top as absolute truth.

And for any propositional atom p of course, you have whatever is assigned by the function tau is the truth value of that atomic proposition. And then by induction given the truth values of propositions of smaller depth, a proposition of a depth not phi is defined as the bar, this bar is brown colour. I hope you can make out the brown colour it is a inverse whatever is the truth value of phi in the truth assignment tau. And similarly, in the case of and it is just a product operation, and in the case of or it is a sum operation, in the case of the condition it is the less than or equal to dot operation. Remember that less than or equal to dot is an operation right, which we defined and in the case of the bi conditional it is essentially the equal to dot operation.

So, this is how we formally specify and this is this is the style of specification, which is also fairly popular in programming languages, though it is quite possible that you did not do it in this style. But if you do if you have to do a course on this semantics of programming languages in which denotational semantics is used, then you will actually find this style of definition. So, basically what we were saying is syntax is used, in order to represent some semantics. So, semantics is always a function and which given certain semantical notions, you want representations. So, your object language tries to represent them right and so that is your so your semantics hangs on the structure of your syntax. So, in a certain sense the semantics is by structural induction because essentially what you are saying is…

(Refer Slide Time: 14:41)



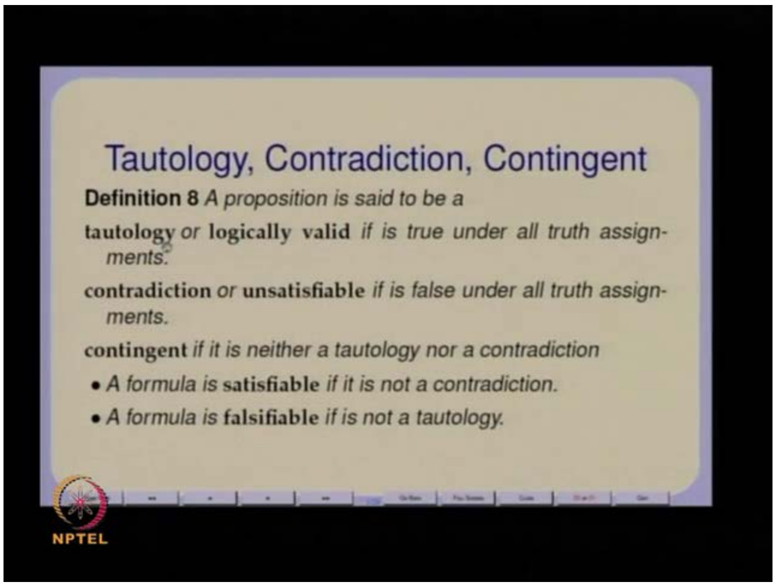
If I were to if I have to evaluate a compound proposition like this, then essentially I am doing an induction based on the structure of the surface. So, the meaning of so the meaning of this portion, so the meaning of the whole proposition is dependent on the individual meanings of these two sub trees. And the meaning of the individual sub tree by induction is dependent on the sub trees under them, the meanings of the sub trees under them. And that is that is how structural induction work and structural induction of course, by now you must have studied structural induction in programming languages, and you will you must also remember that structural induction is actually exactly equivalent to the principle of the mathematical induction. I hope that is clear to everybody, structural induction is has the same power as the principal of mathematical induction.

So, almost so any proof by structural induction can also be rewritten as a proof by the principal of mathematical induction and vice versa. So, I say, but what structural induction gives you just is a sort of convenient case analysis that is what so it is a principal of mathematical induction, where in the induction hypothesis you have a number of cases right, that is that is really what structural induction is. So, there is absolutely no difference in power between the principal of mathematical induction and the structural induction.

As in the case of mathematical induction you can have definitions by induction and you can have proofs by induction. So, in the case of structural induction too you can have definitions by structural induction and proofs by structural induction. And this is the case of a definition by structural induction. So, given any syntax, the best way to cover all the syntactical elements in the language that is generated by the syntax is to define it by structural induction, is to define the syntax semantics by structural induction and that will cover all of it.

So, now this is as far as the pure algebraic aspects of theory are concerned and now we have to come and this is all like more or less like normal mathematics, or whatever. What is it that distinguishes logic as a sub discipline of mathematics or some other logical concepts, which we will begin to look at? So, the basic fundamentals have been set you have an object language and you have a semantics given in a known structure namely the namely the Boolean algebra.

(Refer Slide Time: 18:06)



Tautology, Contradiction, Contingent

Definition 8 *A proposition is said to be a*

tautology *or* **logically valid** *if is true under all truth assignments.*

contradiction *or* **unsatisfiable** *if is false under all truth assignments.*

contingent *if it is neither a tautology nor a contradiction*

- *A formula is* **satisfiable** *if it is not a contradiction.*
- *A formula is* **falsifiable** *if is not a tautology.*

So, a proposition is said to be a tautology or logically valid I mean at this point at the level of propositional logic, there is no difference between the two terms tautology and logically valid, but when we go into first order logic I am going to distinguish between the two. If it is true under all truth assignments it is a contradiction or it is also called unsatisfiable, if it is false under all truth assignments. Otherwise if it is neither a tautology nor a contradiction it is said to be a contingent stage.

Student: ((Refer Time: 18:53))

Nobody will define the whole point about mathematics is that it is platonic, and what does platonic mean? It means that they exist there exist an uncountable number of truth assignments tau, right? You agree with that because your are taking the set of all possible total functions from the set at of atoms A to the Boolean set 0, 1 and that there are an uncountable number of truth assignments and they exist that is it. There is nothing algorithmic about it, it is nothing constructive about it the fact that they exist itself is all that you require and that is the platonic idea.

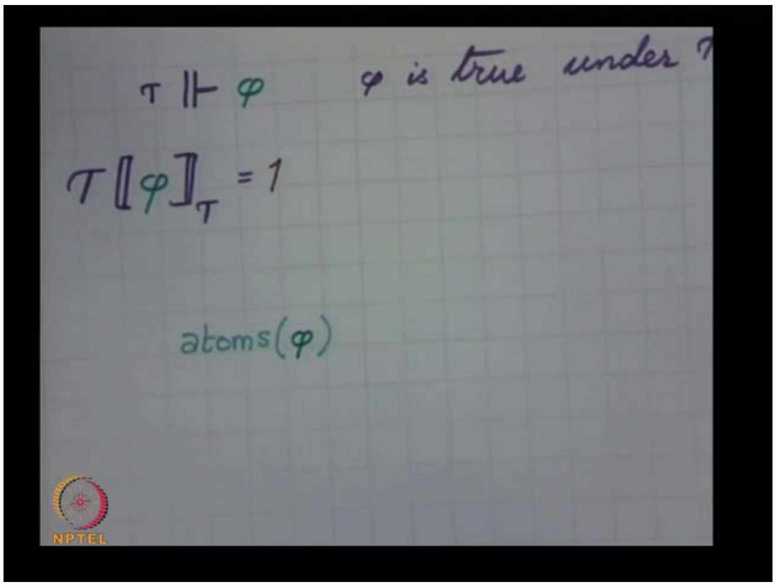
So, a proposition is said to be logically valid and we will look at logical validity more importantly. In the case of in the case of propositional logic it is just as same as being a tautology, if it is true under all truth assignments and it is contingent, and it is a contradiction or it is said to be unsatisfiable, if it is false under truth assignments. And it is contingent, if it is neither a tautology nor a contradiction. So, which means that there are certain truth assignments tau, in which the proposition might be true and some other truth assignments tau prime in which the proposition might be false, which comes from the structural inductive definition.

So, a formula is satisfiable if it is not a contradiction, which means it could be a tautology or it could be contingent, but what this says is that it is not the case that it is false under all truth assignments, which means that there exists at least one truth assignment, in which this formula is true. So, here in you are looking at these kinds of statements, when I say which means and so on and so forth.

I am already actually using a principles of reasoning which are common in mathematics, but which are actually codified in first order logic, when I say something like if it is not a contradiction then what you are saying is that it is not false under all truth assignments. Which means that there exists a truth assignment in which it is true, and all this is actually like first order reasoning. But, it is standard mathematical reasoning and so this kind of this kind of definition consequence mechanism is what codifies first order logic and we will use that quite frequently.

A formula is falsifiable if it is not a tautology, which means there exist at least one truth assignment, out of that uncountable number of truth assignments in which the formula can be shown to be false. Certain other standard things like which we will so we will also look at for example, aswe will look at the notions.

(Refer Slide Time: 23:03)



$\tau \Vdash \varphi$ φ is true under τ

$\mathcal{T}[\![\varphi]\!]_{\tau} = 1$

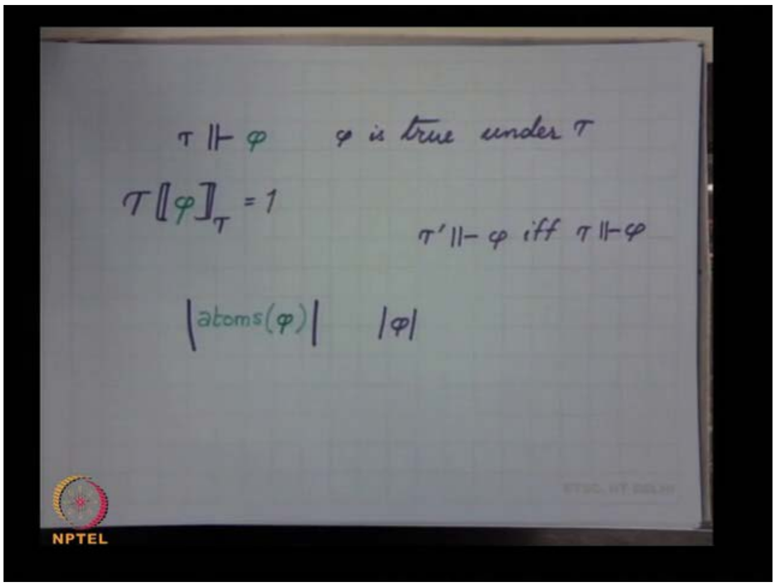
atoms(φ)

Certain other notions which are important from a purely logical point of view is one is that we will actually think of, we will say for any tau for any truth assignment tau. If you have a formula phi, this indicates that phi is true under tau. So, phi is true under tau, that means we are essentially saying that if you look at this, this is equal to 1. So, certain so this is so in the area of algorithms of course, there is a standard thing about proving satisfiability, I mean I do not know whether you have heard about it. One of the things that you want to a problem is so the problem of three sat is empty complete, what is a three sat problem?

Basically, there is problem sat is a empty complete which essentially means given a Boolean formula, which is essentially a propositional logic formula. Finding whether it has a satisfying assignment whether there exists a tau, for which the formula is true that problem is essentially empty complete. Notice that what we what we also have is that if you have any, any formula phi I can also think about by structural induction, I can define atoms of phi. When I define atoms of phi, I can define this by structural induction.

Basically a phi is just an atom if it is true or if it is bottom or top then of course, atoms are five or empty less. For convenience some people include bottom and top also as an atom, but let us keep them separate and if phi is in atomic proposition then of course, that itself is a atom of phi so it is a singleton set. For any formula like phi and si you look at all the atoms of phi and look at all the atoms of si and you take the union of the two sets and you get the set of all atoms of phi and si. And by structural induction that is what happens with the other binary operators. In the case of not phi it is just an atoms of phi.

(Refer Slide Time: 26:55)



$\tau \Vdash \varphi$ φ is true under τ

$\mathcal{T}[\![\varphi]\!]_\tau = 1$

$\tau' \Vdash \varphi$ iff $\tau \Vdash \varphi$

$|atoms(\varphi)|$ $|\varphi|$

So, we can define this so it is clear that if I take two truth assignments tau and tau prime, which for the atoms of phi have exactly the same assignment. And they differ only for those atoms which do not occur in phi, then clearly tau prime satisfies phi, if and only if tau satisfies phi. So, here in lies the fundamental thing about the truth table, so the atoms of phi is any way a finite set. The atoms of phi is a finite set so it allows us to define, so it allows us to define truth tables, which are essentially of size, two raise to the cardinality of the size of phi.

Basically what we are saying is under this condition, it is it is possible to collapse from the point of view of phi, it is possible to collapse all the truth assignments, uncountable set of truth assignments, possible truth assignments to sets of to essentially a finite number of classes. Such that tau and tau prime are the same or belong to the same class. So, which means that the problem becomes finite tree it is only necessary if not necessary to know the whole of a truth value truth assignment tau, it is only necessary to know the assignment of tau for the particular atoms in phi that is it. And that is where we construct that is how we construct truth tables. So, we construct truth tables only based on the truth values assigned to the atoms of phi. And that those are finite always which essentially what we are saying therefore, is that now.

Now, the problem of computing of satisfiablity works out to essentially saying can you find a satisfactory assignment of truth values, only to the atoms of phi. And then I do not care about all other atoms and that is the finite set, so which means that that is something that algorithmically. Firstly if it is infinite it is not algorithmically possible at all, but now it actually becomes something is it possible algorithmically that is the question that becomes feasible. And algorithmically it is possible because we can construct truth tables what is the what is the size of a truth table typically? You have it is it is essentially exponential in the size of the formula, when we say size of the formula we are including not only the number of atoms, but also all the number of operators occurrences of operators in the formula.

So, it is essentially that so the truth table will have order of the number of atoms columns, rows no it will have actually it will have no two raised to the number of atoms rows. And as many columns as the length of the formula basically right, for each operator I will require a new column that is. So, the truth table will have a size that two raise to the number of atoms multiplied by the number size of the form. So, in terms of possibility therefore, there is an algorithmic possibility, but in this algorithmic possibility it is exponential in the size of the form.

And normally like good data structures and algorithms people we would like to find non exponential algorithms, because if formula size is can become really large. In such a case finding non exponential sizes of non exponential time algorithms becomes a challenge. But what we will see in most of this course is a practically anything interesting that you want to do algorithmically is going to be exponential in size. So, which means we do calculate, we will calculate complexity in our algorithms, but they will except for the most trivial problems, all the problems will be exponential at least will be exponential. Some of them will be doubly exponential, some of them will be even more, so you they will go as exponential tower you know you have seen the exponential tower two raise to two raise to two raise to.

So, the most efficient ones will just be exponential so in a certain sense this is where I first I wanted to make the first statement, regarding how we cannot therefore, just dismiss of anything just because it is exponential, because if you are going to do theorem proving you have to live with it. The only alternative therefore, is to come up with heuristic methods for reducing the size somehow, you will never get down, you will never be able to get it down to less than exponential any way. But you might want you might require you might have good efficient data structuring mechanisms by which at least the space also is not exponential, may be it will remain in free space, you know. Time might still be exponential, but then what you are saying is that exponential when you look at analysis of algorithms when you say something is exponential, basically what you are saying is it is in the asymptotic case, for large values of the input.

So, many of our algorithms might actually be dependent on the size of this on the size of this object atoms of phi or length of phi, in which case then if they are all exponential, it does not matter what we want to do to therefore, is to somehow prove maybe the space or you want to factor out computations. So, that you do not do duplicate computations for the same thing and there by speed up. And since this exponential are only in the asymptotic case for sizes less than that threshold, things are still possible on a standard machine. And therefore, we should be able to we should be designing algorithms for our currently available technology.

So, we do not dismiss off algorithms just because they are exponential basically we live with the idea that all interesting problems have only exponential algorithms. But in practice it is possible to still get results, for a large subset of the problems that we are interested. That is so that is important so as we can see truth table construction itself is exponential. So, one of the first problems in empty completeness defined by Steve Cook was to was to show that there is no polynomial way of essentially finding and satisfying truth assignment to a Boolean formula. That means to a propositional logic formula.

The other important thing that you show in empty completeness was that every problem which is inherently exponential can be reduced to the Boolean satisfiability problem. So, they the empty completeness forms is a class of problems, such that at this reduction process and there is a polynomial time reduction of all the empty complete problems to each other, basically. So, you take some representation of the problem I can show that there in a polynomial term I can reduce to satisfiablity Boolean satisfiability problem. So, Boolean satisfiability is basically because everything is you are assuming finite domain finite sets finite everything is said to be finite right, for it to be algorithmically possible.

So, first thing most of our algorithms are going to be exponential our question is do we get do you have some nice ideas to make it make them feasible, and run in reasonable amount of time, and that is the challenge for theorem proving. There is of course, a more complicated subject of heuristic analysis and so on and so forth, but we will not get into that because we are primarily interested in logical problems. Logical problems from the point of view of theorem proving from the point of view of satisfiablity, unsatisfiablity is another thing. And we are also interested in the logical concepts.