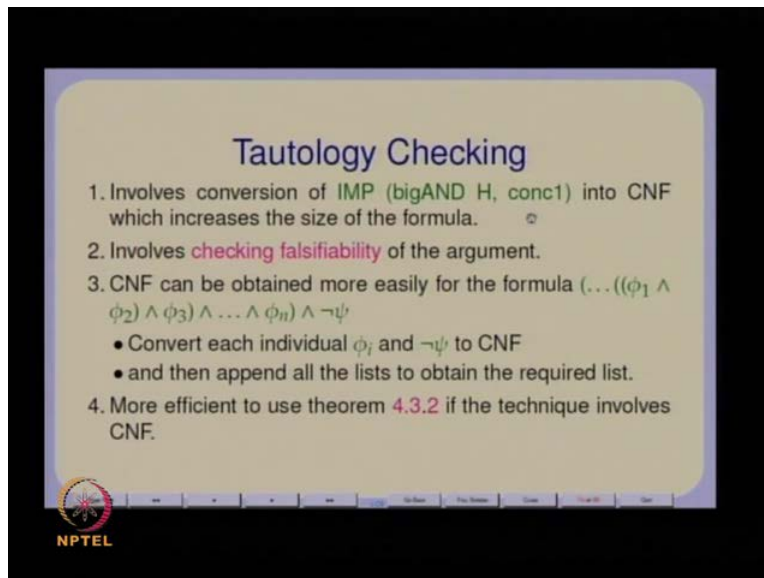


**Logic for C S**  
**Prof. Dr. S. Arun Kumar**  
**Department of Computer Science**  
**Indian Institute of Technology, Delhi**

**Lecture - 07**  
**Propositional Unsatisfiability**

So, let us look at some techniques Propositional Unsatisfiability. So, the simplest and the probably the its not actually the oldest but for some reason it is important because it is used in all logic programming systems.

(Refer Slide Time: 00:54)



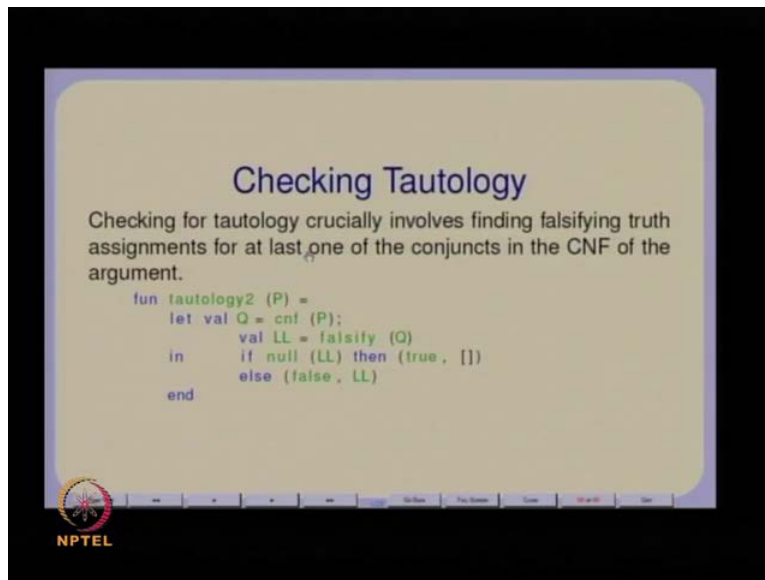
**Tautology Checking**

1. Involves conversion of IMP (bigAND H, conc1) into CNF which increases the size of the formula. ☹
2. Involves **checking falsifiability** of the argument.
3. CNF can be obtained more easily for the formula  $(\dots((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \wedge \neg\psi$ 
  - Convert each individual  $\phi_i$  and  $\neg\psi$  to CNF
  - and then append all the lists to obtain the required list.
4. More efficient to use theorem 4.3.2 if the technique involves CNF.

NPTEL

But, so we looked at a tautology checking and one of the important things of the tautology checker is that at some point you had to compute the conjunctive normal form of essentially a big consumption of formulae implying some other conjunction. So, this implication rewriting of this implication into or essentially means at your conjunctive normal form has to take this big AND and covert it all into all write by Demorgan's law. So, in that sense this is actually quarter bit of effort for one thing and it of course increases the size of the formula tremendously.

(Refer Slide Time: 01:50)



**Checking Tautology**

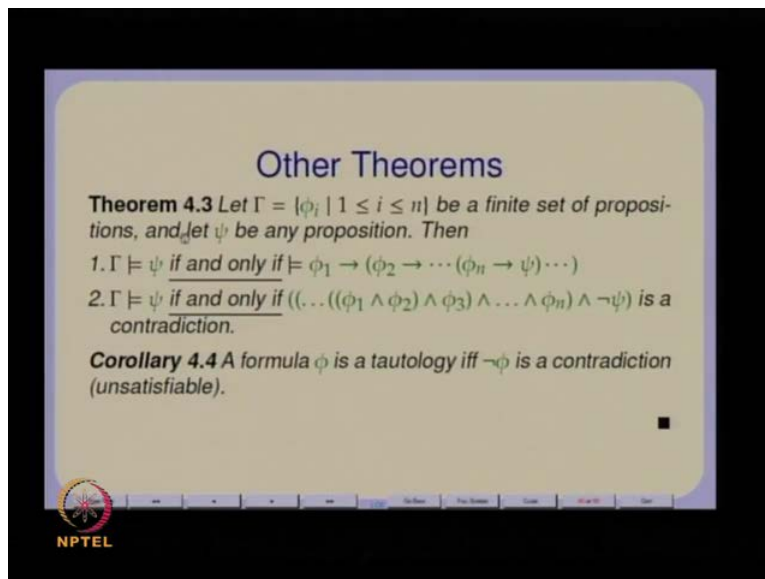
Checking for tautology crucially involves finding falsifying truth assignments for at least one of the conjuncts in the CNF of the argument.

```
fun tautology2 (P) =  
  let val Q = cnf (P);  
      val LL = falsify (Q)  
  in   if null (LL) then (true, [])  
      else (false, LL)  
  end
```

NPTEL

So, and any way finally when you looked at a tautology checker it finally involve falsifying the argument falsifiers and then based on that deciding whether it is a tautology or not. So, the falsifiability of the argument was an important aspect of that tautology of the checker design.

(Refer Slide Time: 02:14)



**Other Theorems**

**Theorem 4.3** Let  $\Gamma = \{\phi_i \mid 1 \leq i \leq n\}$  be a finite set of propositions, and let  $\psi$  be any proposition. Then

- $\Gamma \models \psi$  if and only if  $\models \phi_1 \rightarrow (\phi_2 \rightarrow \dots (\phi_n \rightarrow \psi) \dots)$
- $\Gamma \models \psi$  if and only if  $((\dots ((\phi_1 \wedge \phi_2) \wedge \phi_3) \wedge \dots \wedge \phi_n) \wedge \neg \psi)$  is a contradiction.

**Corollary 4.4** A formula  $\phi$  is a tautology iff  $\neg \phi$  is a contradiction (unsatisfiable). ■


NPTEL

But, if you look at something like this theorem its here you have this take the second part. The second part is just a huge conjunction of formulas. So, if you have all the hypothesis you take the

hypothesis of an argument  $\phi_1, \phi_2, \phi_3$  etcetera up to  $\phi_n$ . And you take the negation of the conclusion that is not  $\psi$  and this  $\psi$  is logically follows from the set  $\phi_1$  to  $\phi_n$  if and only if this huge conjunction is actually a contradiction or it is unsatisfiability. So, since we had falsifying falsifiability as the converse of the tautology checker. Then, actually I think to do is to look for unsatisfiability of this and this huge conjunction has a certain advantage. And that is that if I am anyway going to use a conjunctive normal form. Then, having a conjunction at the top of a large number of formulae means that I can now apply a sort of a divide and conquer technique I can just convert each individual conjunct into a conjunctive normal form.

So, which means I deal with smaller size formulae convert theorem into CNFs and then the huge we got a list of list of literals in our tautology checker then that just means appending all the lists that we get together. So, which means we just append all this so you convert each individual conjunct into a CNF that is a smaller problem you essentially think of it as small problems and then you have to append all the lists to obtain the required list of lists of literals. So, in the certain sense if you are committed to using a conjunctive normal form. The use of this theorem is likely to therefore, be more time efficient than the use of a tautology checker design. As we did before so, and that is in fact what actually resolution theorem proving that right.

(Refer Slide Time: 04:51)



**Propositional Resolution**

To show  $\Gamma \models \psi$  we show that  $\bigwedge \Gamma \wedge \neg\psi$  is false by first transforming  $\bigwedge \Gamma \wedge \neg\psi$  to a formula in CNF.

This CNF is represented as a *set of sets of literals*. Let  $\Delta$  be the set of sets of literals.

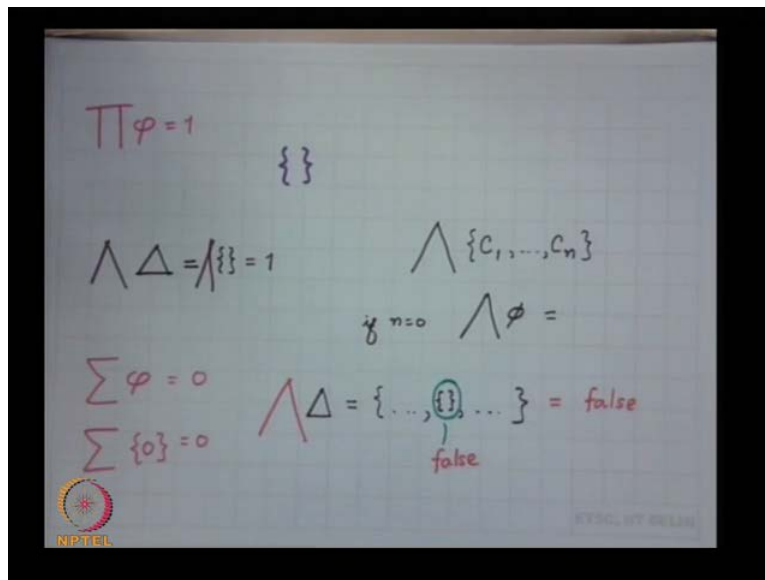
1. Each  $C \in \Delta$  is called a **clause**.
2. Each clause in  $\Delta$  represents a disjunction of literals.
3. The empty clause represents a contradiction.

NPTEL

So, let us quickly look at it Propositional Resolution. And so essentially what we are saying is therefore, to show that some finite set  $\gamma$  of to show that formula  $\psi$  follows from a finite set  $\gamma$  of propositions. We just show that the big AND of  $\gamma$  and not  $\psi$  is false is unsatisfiable. By first transforming this huge formula big AND not  $\psi$  into a formula in conjunctive normal form. So, this will of course represent this conjunctive normal form as a set of sets of literals. Normally in form while, you are doing programming it is usually as a list of list of literals. But, I will look at it as a set of sets of literals because essentially because, of the importance of both conjunction and disjunction. Duplicate occurrences do not matter or actually better thing to in order to reduce the size of the formula is to remove duplicate occurrences.

So, supposing think lets capital delta be the set of sets of such literals. Where, each set of literals is called a clause and, each clause in delta essentially represents at disjunction of literals. And, the empty clause represents a contradiction junction of literals. What is a empty clause I mean why is it represent the contradiction? Supposing, I have an empty set of literals. So, what should that actually be.

(Refer Slide Time: 06:57)



If, now that I gone from the language it self to just literals and sets of literals and sets of sets of literals. Then, the question of interpreting what happens to an empty set in each case is important. So, what is a correct interpretation for the empty set I mean that is an important

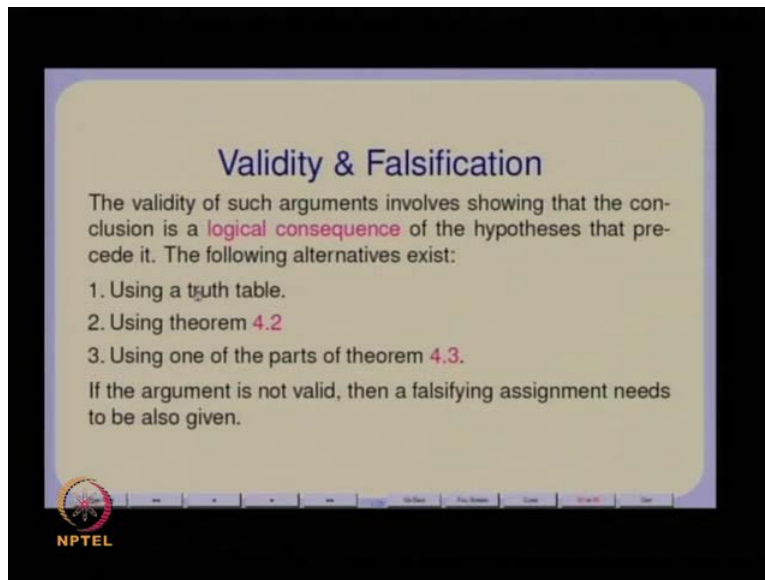
question. So, suppose think let us look at this problem this way. Supposing in  $\Delta$  is empty that means there are no clauses in  $\Delta$ . What does this empty set represents? This is the standard thing in algebra is to take the identity element. So,  $\Delta$  actually represents a big conjunction of a collection of clauses  $C_1$  to  $C_n$ . And, what we are saying now if this big conjunction if  $n$  equals to 0 then, essentially you have a big conjunction of essentially an empty set of clauses. So, you obviously the identity element of conjunction is going to be the answer right this is the standard right.

Let us go to the arithmetic the starting point of let us see the summation and product the summation is the entity element. If, I doing a summation over on empty set of numbers then the answer has to be 0. The whole point is this if I take any set of numbers and I add 0 to that set and I do not say a summation in anyway. Because, 0 is the entity element for the sum so if I just take the empty set just like having the set containing 0 so the answer is going to be 0. So, if I want to take the empty product all the product computations are actually starts with the entity element of multiplication so if i take the empty product this going to be equal to 1. What is the identity element of conjunction? The identity element of conjunction is true. Or if, you like the brown one so which he said if  $\Delta$  were empty then this conjunction of  $\Delta$  this conjunction actually gives you what. However, supposing in  $\Delta$  actually content supposing, if  $\Delta$  was non empty. And it contain some elements along with it it contained an empty set. So, this is the empty set of literals is also a member of  $\Delta$ . First thing as any set of  $\Delta$  inside it represents a disjunction of literals. So which we set this empty set which, means this empty set actually represents the identity element of disjunction which is false. So this has to be false. So if, this is false this  $\Delta$  represents a conjunction of a set of clauses which means you are conjuncting with false and false is 0 for conjunction. And therefore, then this entire conjunction then becomes false is that is fine. So, that is what if the set does contain the empty clauses in a element then you already reached a contradiction it does not matter what other clauses are there in this.

So, this is what actually resolution uses the fact that if you can somehow derive the empty clause as one of the clauses in your set of clauses. Then, you have already found a contradiction which means all of the clauses then become irrelevant. So, this is what so let us look at Propositional Resolution. And we have it is a good idea to look at all these algorithms or a propositional view point so, that the extension to first order and other kinds of logics then becomes easy. Because,

all of the logics have propositional logic as a sub languages always so these are the basic connectives right and or and not so this is a propositional resolution. So, the important thing is so the techniques that you are looking at are I mean if you at this if you look at the truth table technique.

(Refer Slide Time: 14:08)



**Validity & Falsification**

The validity of such arguments involves showing that the conclusion is a **logical consequence** of the hypotheses that precede it. The following alternatives exist:

1. Using a truth table.
2. Using theorem 4.2
3. Using one of the parts of theorem 4.3.

If the argument is not valid, then a falsifying assignment needs to be also given.

NPTEL

For example, so the main problem about the truth table is that it is not discriminating enough I mean you construct regardless of whether a certain atom is relevant or not relevant for the validity i mean.

(Refer Slide Time: 14:22)

**Arguments**

A typical informally stated argument might go as follows:

If prices rise, then the poor and the salaried class will be unhappy.

If taxes are increased then the businessmen will be unhappy.

If the poor and the salaried class or the businessmen are unhappy, the Government will not be re-elected.

Inflation will rise if Government expenditure exceeds its revenue.

Government expenditure will exceed its revenue unless taxes are increased or the Government resorts to deficit financing or takes a loan from the IMF to cover the deficit.

If the Government resorts to deficit financing then inflation will rise.

If inflation rises, the prices will also rise.

The Government will get reelected.

Therefore the Government will take a loan from the IMF.

**NPTEL**

Supposing, even if this argument were valid I can add a whole lot of nonsensical sentences to this argument which you do not have any argument with the atoms of this argument and still the argument will be valued. Now, if I add all those nonsensical sentences also with there own atoms then the number of atoms increases, then the size of truth table increases and the construction of the truth table does not discriminate between what is a relevant atom and what is not a relevant atom for the validity of the argument. Where as what we would like to do in order to reduce the sizes of the sets of formulae and in order to speed up proving is reduce sizes as reduce the number of atoms possible or look at only those sets of atoms. So, the falseifiability in the tautology checker was the way of looking that only the relevant atom right which could provide a clue as to the validity or the invalidity of the argument. Similarly, in resolution we are actually going to so in resolution similarly this is what is going to happen. So, it provides a directed way of looking for contradictions use using literals.

And contradictions are easily obtained by looking at positive and negative literals. So, if there is an some atom which occurs throughout the argument only in either positive form or in only in negative form. Then, that atom is irrelevant to irrelevant to both tautology or it is a contradiction that atom can atmost make the entire formula. That you create a contention formula rather than which means depending on the truth of the atom may be it might be true or false you know so that atom becomes completely irrelevant. So, all are efforts are actually directed towards trying

to firstly reduce sizes of formulae, secondly using some directed approach which will focus on proving a contradiction or proving a tautology. And for that we require to essentially derive the empty class.

(Refer Slide Time: 16:53)

**Clean-up**

Let  $\Delta$  be a finite set of clauses.

1. For all clauses  $C, C'$ , if  $C \subseteq C'$ , then  $C'$  may be *deleted* from  $\Delta$  without affecting logical equivalence.
2. Any clause containing *complementary pairs* of literals, may be deleted from  $\Delta$  without affecting logical equivalence.
3. From any clause, *duplicate* occurrences of a literal may be *deleted* without affecting logical equivalence.

The resulting clause set  $\Delta'$  is said to be *clean*.

$$\bigwedge_{C \in \Delta} \bigvee_{L \in C} L \Leftrightarrow \bigwedge_{C' \in \Delta'} \bigvee_{L' \in C'} L'$$

NPTEL

So, essentially the what of course in general for any kind of propositional resolution it is not particularly true of logic programs. But, it is if you can also use resolution for improving technique for any kind of argument that is right. So, one thing is we let us look at this finite set of clauses so let us assume that the delta is the final set of finite set of clauses. And, we have to just find out whether this set is unsatisfied that is what the problem of resolution theorem 4.3.2 gives you I can just look upon a set of formula and, ask whether they unsatisfied you there this is a contradiction. So, one thing of course is clear so I have a number of clauses and I might have two clauses  $C$  and  $C$  prime such that one is the subset of the other in this set delta.

Now, each clause represents a disjunction of literals. Which means,  $C$  prime if  $C$  is a subset of  $C$  prime then  $C$  prime represents a larger disjunction than  $C$ . Which, means  $C$  prime is irrelevant to proving whether it is a contradiction right so the entire class  $C$  prime can be deleted. Because, whenever  $C$  is true  $C$  prime is guaranteed to be true. So, since we are looking at this as since these are two members within a delta its taking  $C$  and  $C$  prime. And,  $C$  and  $C$  prime where,  $C$  is a



disjunction of literals where  $C'$  is a superset of  $C$  is equivalent to just  $C$ . So, it is not just preserves truth it also preserves false.

So,  $\Delta$  with this  $C'$  removed is still logically equivalent to the original  $\Delta$  which had the  $C'$  right. So, the first thing is that all such clauses may be deleted from  $\Delta$  and of course we are looking at sets if you are looking at lists instead. Then, as I said then you have to put in an artificial ordering and in ordering because checking permutation I mean checking whether one list is a sublist of another. Without sorting them in some reason in some way means it is essentially determining whether some sub list of the other one is the permutation of the first list. And, determining permutations is quite complicated right. So, what I mean if I look at the problem of finding the median of a list of elements. I can do that in less than  $n \log n$  but if I look at the larger context in which I have to do such things such a operations also then its simpler to just sort the list. And essentially hammers the cost that extra factor that sorting gives over all the other operations that I might have.

So, for example the quality is very easy to check if they are sorted I do not need to consider permutation list the quality checking is just linear. So, that itself is where as permutations are actually  $n^2$  or minimum of  $n^2$ . So, that is a problem so I can just delete such clauses. The second thing is suppose in any clause  $C$  contains a complementary layers of course it is a literal so they could be both  $p$  and  $\neg p$  in  $C$ . Since, if  $C$  represents a disjunction of literals therefore the existence of all of the  $p$  or  $\neg p$  makes all of the literals are relevant or irrelevant. And, this clause  $C$  becomes true becomes logically equivalent to true if this clause  $C$  becomes logically equivalent to true its presence a  $\Delta$  does not affect whether the clause exists or not. So, any clause which contains a complementary pair of literals which can be deleted from  $\Delta$  without affecting the logical literals. So, it is this is not something most resolution mechanism tell you to do. But if you do not do this clean up very often we will slip up in your algorithm and you will not get a contradiction.

Or, you will ask you something a contradiction which is not because this complementary pair will allow you to derive something like an empty set. Whether that is an empty clause or an empty set of clauses is a question that you have to ask because that is it is the new difference that tells true and false right. So, we need to do this clean up the most works and resolutions do not even refer to it. The third thing of course from the idiom ports of and and or its clear that the

duplicate occurrences of a literal may be deleted from each clause. This also includes duplicate occurrences of clauses because, there is a subset or equals so you can delete duplicate occurrences of a literal from any clause without affecting logical equivalence you might. So, this clean up gives you a new set of prime which hopefully is smaller than delta. And of course what it satisfies this logically equivalence. So, what we have done so far is that we have actually done a cleanup which preserves logically equivalent to do so. It is not usually true was derivations do not preserve logically equivalence usually they preserves truth or satisfiability. But, this clean up operation alone look that in isolation actually preserves logically.

Student: (Refer Time: 24:00)

each of them is or of literals

Student: How could we delete the supper set

Because, what you are saying is you have so this what we are saying is you have C.

(Refer Slide Time: 24:31)

$$C = \{l_1, \dots, l_m\} \in \Delta$$

$$C' = \{l_1, \dots, l_m, \dots, l_n\} \in \Delta$$

$$(\vee C) \wedge (\vee C') \Leftrightarrow l_1 \vee \dots \vee l_m \Leftrightarrow \vee C$$

Which, say some literals say  $l_1$  to  $l_m$ . And you have C prime which is lets say  $l_1$  to  $l_m$  and let us say after some  $l_n$  then, what you have the fact that C and C prime both belong to delta means that you are looking at or of C, or of the literals in C and or of the literals in C prime. So, if you are looking at this then what happens this set this is just logically equivalent to  $l_1$  or  $l_m$ . So, that is

just equivalent to or of  $c$  so I can delete the prime. So, the reason that I am doing all this because, when I actually started doing some resolution proof by hand. I was suddenly found that I was suddenly structured at various some points because, these books do not mention that you know there has to be clean up done. And there has to be this otherwise you will be leading irrelevant clauses and literals you will not be able to derive an empty clause.

Because, you did not delete the irrelevant literals. So, here actually in this particular case the entire clause  $C$  prime is irrelevant. If, it continued to exist if you did your resolution and you may not get an empty set at all if you happen to do the resolution with  $C$  prime. And so you will be start you will think actually that you could not derive an empty clause. And therefore, the argument is invalid which is a wrong assumption. So, this clean up is actually essential in actually programming it many people just I mean it has to be just like nobody discuss sorting algorithm inside or a tautology checker really. This clean up is also never discuss really in any work that does the resolution. But, this clean up needs to be done other wise it actually get wrong business.

(Refer Slide Time: 27:28)

**The Resolution Method**

For any clean set  $\Delta$  and an atom  $p$  let  $\Lambda = \{C \in \Delta \mid p \in C\}$  and  $\bar{\Lambda} = \{\bar{C} \in \Delta \mid \neg p \in \bar{C}\}$   
 Since  $\Delta$  is a clean set

1.  $\Lambda \cap \bar{\Lambda} = \emptyset$ .
2. **However  $C$  and  $\bar{C}$  may not be disjoint.**
3. For each  $C \in \Lambda$  and  $\bar{C} \in \bar{\Lambda}$ ,  $p \notin \bar{C}$  and  $\neg p \notin C$ .

After resolution on the pair  $(p, \neg p)$  the new set of clauses

$$\text{resolve}(\Delta, p) \stackrel{df}{=} (\Delta - (\Lambda \cup \bar{\Lambda})) \cup \{D \mid D = (C - \{p\}) \cup (\bar{C} - \{\neg p\}), C \in \Lambda, \bar{C} \in \bar{\Lambda}\}$$

NPTEL

Then, the actual Resolution Method is I start with a clean set by the way this word clean is introduced because, I had to introduce the word. So, I start with a clean set  $\Delta$  and so which means that. And I look and I choose any atom so relevant atom is one which some is an atom

such that there is a naught  $p$  also somewhere in the delta. So, let us look at all the clauses which contain  $p$  and let us call them  $\Lambda$ . And  $\bar{\Lambda}$  be the set of all clauses which contains naught  $p$ . So, now you all focus you had already brighten your focus only on the relevant atoms only those atoms which have complementary pairs somewhere both positive and negative form somewhere in the delta are relevant all other atoms are irrelevant. So, any atom which occurs only in one form either positive or negative is irrelevant resolution. So, there straight away goes off the so again sorting the atom, sorting the relevant so on makes sense in choosing such an thing.

For example if, you use an sorting algorithm. Or if you use the total ordering in which the negation of an atom immediately precedes the positive version or immediately succeeds a positive version. Then, just looking through the list of atoms you know the which complimentary price exists the entire sets. Since, delta is the clean set first thing I take these thing  $\Lambda$  or  $\bar{\Lambda}$  are the subsets of delta. The first thing of course is that they have prejoint sets because, they have  $p$  because delta have  $p$ . They have to be disjoint because any clause which contains  $p$  cannot contain naught  $p$ . So, the set of clauses which contain  $p$  is completely different from the set of clauses which contain naught  $p$ . So, this delta and delta bar are disjoint but how are individual clauses  $C$  and  $\bar{C}$  may not be is joint they have some common atoms.

For example or else they might have some common literals. The other thing of course is that  $C$  and  $\bar{C}$  are mutually exclusive as far as  $p$  is concerned. One of them contains  $p$  at the other contains naught  $p$ . The one that contains  $p$  does not contain naught  $p$ . The one that contain does not contain  $p$ . So, they are exclusive with respect to  $p$ . Once you have done that so we do what is known as an operation called resolution on this complementary pair set  $p$  naught  $p$ . And that is a function so what we do is we start with delta. So, essentially we take these sets these two sets  $\Lambda$  and  $\bar{\Lambda}$  and our resolution has to resolve on every  $C$   $\Lambda$  with every  $\bar{C}$  in  $\bar{\Lambda}$ . And what does it do and then what do I do I take the unions of the sets after removing  $p$  and  $\bar{p}$  from the two from each pair of clauses. So, from each pair of clauses  $C$  and  $\bar{C}$  I remove  $p$  and naught  $p$  and take the union. I do this for each pair  $C$  and  $\bar{C}$  in  $\Lambda$  and  $\bar{\Lambda}$ . So, what is the net result net effect? I get a new set delta prime but, this set delta prime is not necessarily equivalent to the original delta that is the first thing. What it does preserve is it preserves a satisfiability condition.

Delta is satisfiable if and only if delta prime is satisfiable and vice versa. Actually since, we are looking at a resolution method which is suppose to give you a contradiction we are saying essentially that delta is unsatisfiable if delta prime is unsatisfied. So, we are actually moving away from logical equivalence. So, this is resulting delta prime if you think of it as a conjunction of disjunctions of sets of literals is not logically equivalent to the original data. And but, what it does preserve is that if the original delta were unsatisfiable this one would also be unsatisfiable. This union tends to increase the sizes of clauses and if for example delta had if Lambda has k clauses and Lambda bar has m clauses. The fact that your taking all pair C and C bar and performing the resolution this is. So, this last two C belonging to Lambda and C bar belonging to Lambda bar means that you are actually taking you might actually be increasing the size number of clauses in the set delta prime. So, if you look at the cardinality of delta prime. Delta prime may not necessarily be a smaller than delta in terms of number of clauses and in terms of sizes of clauses that are common that are there in the two. But, what are the properties does this have.

Student: (Refer Time: 34:42)


That is right what this does is it removes one atom completely from delta. So, if you look at since p and naught p this atom p occurs only in Lambda and Lambda bar. This operation I am removing p and naught p from all elements C and C bar even though, you are taking the union and you are taking all possible pairs of C and C bar. And therefore, you might be increasing the sizes of the clauses even though you are doing all that the number of atoms the total number of atoms in a delta decreases by 1.

(Refer Slide Time: 35:29)

**The Algorithm**

**Require:**  $\Delta$  a clean set of clauses

- 1: **while** ( ${} \notin \Delta$ )  $\wedge \exists (p, \neg p) \in \Delta$  **do**
- 2:  $\Delta' := \text{resolve}(\Delta, p)$
- 3:  $\Delta := \text{Clean-up} \Delta'$
- 4: **end while** ©

 NPTEL

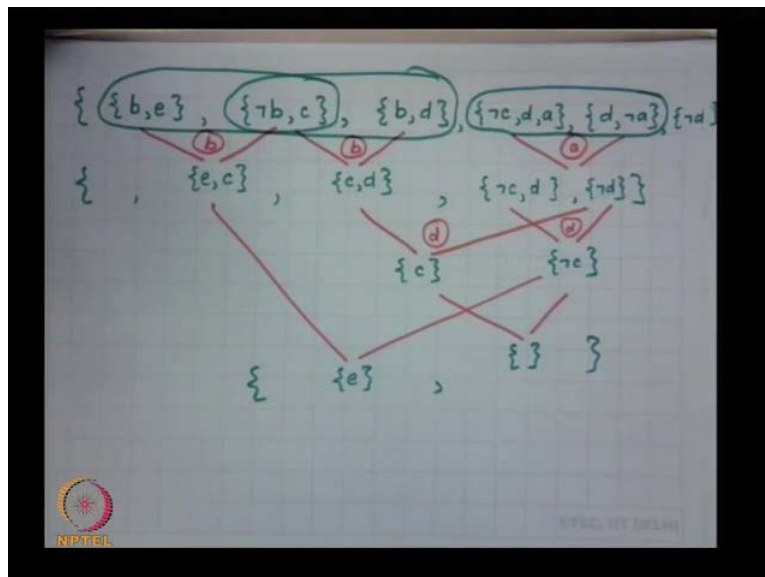
So, that I mean that is crucial in order to show that this algorithm terminates. So, in fact what you do is the clean set of clauses and while I have used the empty list here this could be the empty set. While the empty set does not belong to delta if the empty set belongs to delta you got a contradiction already here. If, the empty set does not belong to delta and there is a complimentary pair  $p, p'$  and  $\Delta$  a  $p$  naught  $p'$  in  $\Delta$ . Then, what you do is you create a delta prime which resolves delta with respect to this atom  $p$ . The result of that is to give you the delta prime which, is possibly larger but it reduces the number of atoms by 1. And what you do is you clean up delta prime and call this delta and give this rate on this file. So, the fact that one atom gets removed in each iteration of this loop guarantees termination.

So, what are the possibilities and terminations? Either an empty clause appears that some point in which case you have proved that your original argument is valid. Or the empty clause does not appear but there is no complimentary pair left you cannot do any more resolution right. If, that happens what is it mean it means that since this new delta that you have got it is actually satisfiable. Since, there are no complimentary price I can give some truth assignments to the individual atoms and somehow make the whole of this latest delta true. Which, means the same assignments can be applied to the original arguments the atoms in the original argument. And of course in the process I have done various things to the various complimentary pairs but now it

does not matter assign any truth value to those atoms which got eliminated. And you have a satisfying assignment problem original argument.

Which, proves that the original argument is invalid because you had negated the conclusion right so right. So, this is what resolution does it is important that so this is the only so this is the actually the only proof of termination would be that you are there are only a finite number of atoms in delta. And therefore, this has to terminate somehow if in the process of resolution you remove one atom every time in every iteration of the resolution you have to terminate. So, when you terminate you either terminate with an empty clause as an element of delta. Or, you terminate with no complimentary pairs of elements in which case those that also provides you the satisfying truth assignment to show the invalidity of the original argument. So, basically you do not care what truth assignments are given to the atoms which were eliminated as complimentary price only the atoms that are left are unresolved a truth assignments to them will also gives you a truth assignment to the original to prove the original invalidity of the atom. Let us do quickly an example by hand. So, here is some data logic program which I thought I will just do. So here is.

(Refer Slide Time: 39:40)



So, I will write it as a set of sets of literals. So, here is something so what we are essentially doing is I am going to take this is too simple so I will take some complicated one. Let us do this

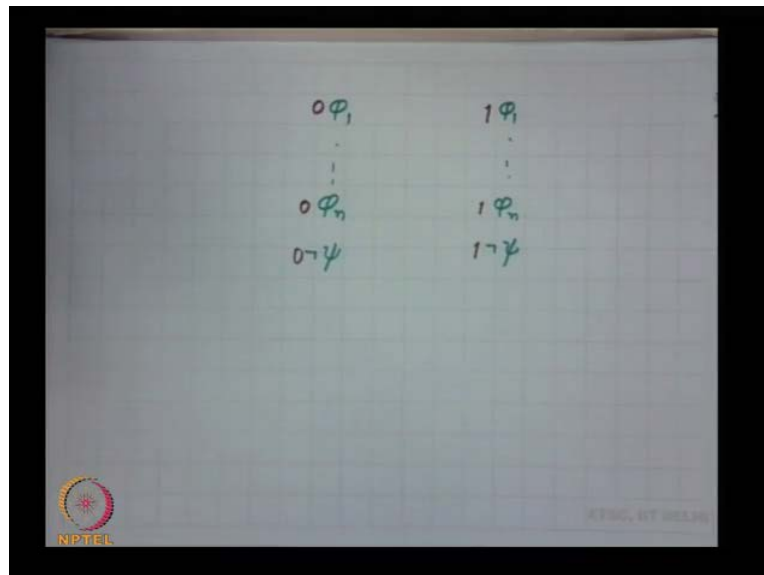
let us take this I have this set of sets I am using small letters for atoms b, e naught b, c, b, d, naught c, d, a, d naught a and then naught. You are able to all of this so I have this so essentially what I do is. I choose some any complimentary pair so one possibility is to choose this a and naught a. And do a resolution a nothing has a in it not a so this is the single resolution which will just give me this. So, essentially what it will do is it will replace the set I have to take this union without so I get naught c and d right. That is what I get. So, notice that d is common to both clauses and of course this is a cleaned up version right this is I am looking at sets. Of course now, I can so essentially this here in this particular case the number of clauses is reduced but and actually the size of the clauses is also reduced. But that is not necessarily true if I consider this pair. So, if I look at this I can do a resolution on b here and then I can do another resolution on b here. So, what this will give me is I set e, c and what this will give me is set c, d. So, now well so essentially e and d are irrelevant these are all these are clean set of clauses so I basically I have replaced these two sign I am actually working to its termination in a very significant fashion. So, I have only these three clauses right now. Which, of course and there is a naught d also. Now, actually I have one possibility is of course treat use d itself as to resolve in this case I have to do two resolutions like this. So, both of these are on d and this resolution gives me essentially naught c here. And this resolution gives me c here and of course strictly speaking so I actually have now three clauses. And strictly speaking I have to do resolution with of both of these clauses this clause naught c. But however, so if you want to do that so essentially what you get is this derives a empty clause.

And this derives e right. So, you are left with essentially just two clauses of which one of the clauses is empty the fact that you have derived the empty clause actually shows at this conjunction or disjunctions is or contradiction. And so it is important to realize that because of I mean the empty set has to be an element of the delta so the delta cannot be empty. If, delta were empty ever then of course it is automatically true right. So, that is not something that is possible if you started with an empty set of literals. You would actually have to derive an empty clause and this is an empty clause. So, this empty clause essentially shows that it is a contradiction so unsatisfied of this so notice that if you were to take this formula. And you were to take this what is this I mean this is just false. It is true that they both they lead to a contradiction but in general a resolution process does not preserve logically there is something that is happening there.



But, of course the main problem in resolution still is the fact that you have to compute a conjunctive normal form and that can be actually quite expensive. And what we would like to do is look at methods which do not necessarily involve doing such conversions. So, these conversions especially using the distributive laws are very expensive so something that does not necessarily use these conversions equivalences which block sizes of formulae. Here, is something that we require is what happens with what is known as a Tabular method. So, the interesting thing about the tabular method is that it is a very symmetric sort of method which and it also proves the unsatisfiability. But, without actually taking the formula taking a huge conjunction trying to compute conjunctive normal forms nothing have to sort. So, it just takes a formula.

(Refer Slide Time: 47:45)



So, essentially what we are saying is that we start with the formulae  $\phi_1$  to  $\phi_n$  and we have  $\neg\psi$ . We have rules the interesting thing about the tabular method is that the rules are symmetric with respect to the underlines semantics that we have provided. So, you take the Boolean algebra it contains actually any Boolean algebra is also complete lattice. And therefore the lattices can be inverted the principle of duality holds and therefore you exploit that in order to be symmetric. So, the essentially whether when you are looking at tautology and contradiction tautology and contradictions are dual concepts. There is nothing about either so instead of focusing on truth or just focusing on false you actually use both truth and false both in

i a way to derive a unsatisfiability. So, what actually this system has is that it has rules so you take this language you take the language of tautology.

And you create a new better language you create a language on top of that. And that language on top of it underling language and and writes 0s or 1s in front of each of these. So, I am treating both so this better languages consists of one prefix of 0 or 1 before each formula. So, I am treating truth and false symmetrically and what I am going to do now is essentially we are going to look at conditions under which a formula is true and conditions under which a formula is false. And we equal to create a tree remember this resolution methods are also used to create a tree. This creates a branching structures and creates a final node so this is like a tree and tabular method is also a tree except that the tabular method has just like the resolution method it has a notion of consumption of a formula. Here, look at this resolution this clause not be your  $c$  is being used twice. That is what the resolution method wants to do all possible pairs  $C$  and  $C$  bar have to be resolved on a complimentary pair. The tabular method is most efficient in the sense that a formula is used exactly ones. There is a notion of consumption in which ones you have read the formula and applied the rules for that formula you does not use the formula again. So, this fact this is what makes tabular somewhat more efficient than resolution the fact that no formula can be used more than ones in the tabular method that is an important aspect of a tabular method.

So, what will do is I will second important aspect of the tabular method is that it treats both truth and false symmetrically. And it also creates a tree but it has a notion of consumption of formulae and ones a formula has been consumed it is never used again whereas most of the proof systems actually might use a formula more than ones. We will do that when we prove the axiomatic proof systems we will do that also. But, the tabular method is very interesting is that because it has its beautiful structure which is just which uses both truth and false with equal priority. And comes up with what are known as close tabular so will just will define the conversion of a close tabular and completely close tabular is what gives you an unsatisfied tabularity result.