Fundamentals of Artificial Intelligence

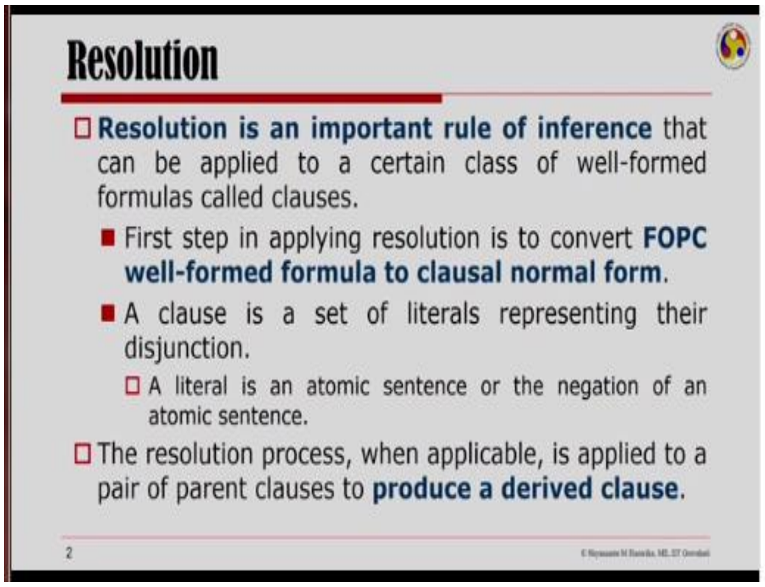
Prof. Shyamanta M Hazarika
Department of Mechanical Engineering
Indian Institute of Technology- Guwahati

Lecture-15
Inference in FOL–Part II

Welcome to fundamentals of artificial intelligence we continue our discussion on knowledge representation and reasoning. So far we have examined first order logic and looked at how first order logic could be use for representation of knowledge in simple application domains. We have looked at how logical reasoning could derive facts implicit in the knowledge base particularly we have looked at inference a process to derive new sentences from already existing sentences.

In the last lecture, we had introduced resolution and looked at 2 processes, 1, the process of converting first order predicate calculus statements to clausal normal form and another the process of unification. Resolution was introduced in quite informal terms today we would introduce more formally a general principle of resolution and particularly look at how a given sentence can be proved in a knowledge base.

(Refer Slide Time: 02:30)



Resolution

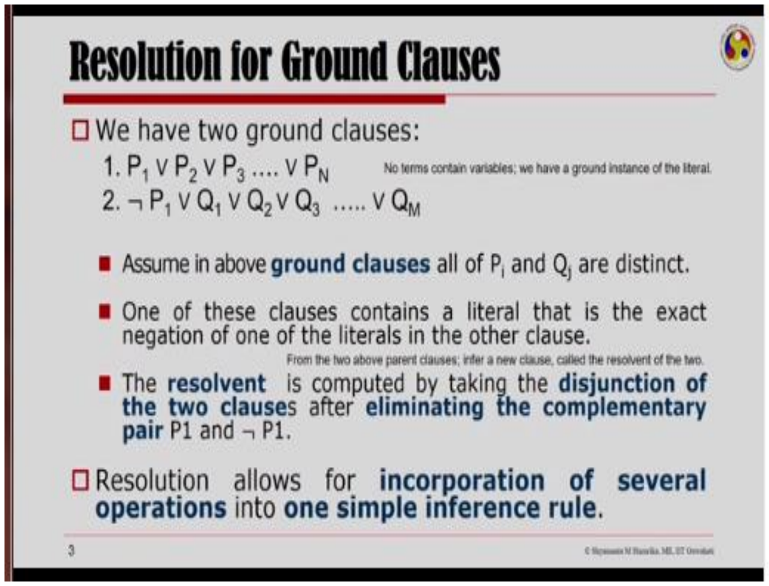
- Resolution is an important rule of inference that can be applied to a certain class of well-formed formulas called clauses.
  - First step in applying resolution is to convert FOPC well-formed formula to clausal normal form.
  - A clause is a set of literals representing their disjunction.
    - A literal is an atomic sentence or the negation of an atomic sentence.
- The resolution process, when applicable, is applied to a pair of parent clauses to produce a derived clause.

2

Resolution as we have already discussed is an important rule of inference and can be applied to only certain class of well-formed formulas which are called clauses. Therefore, the first step in applying resolution is to get to what is called the clausal normal form and we have looked at this in our last lecture, how to get to the clausal normal form from first order predicate calculus well-formed formula.

A clause is a set of literals representing their disjunction and a literal is an atomic sentence or the negation of an atomic sentence. In resolution we are picking up a pair of parent clauses and from there we produce a derive clause.

(Refer Slide Time: 03:33)



Resolution for Ground Clauses

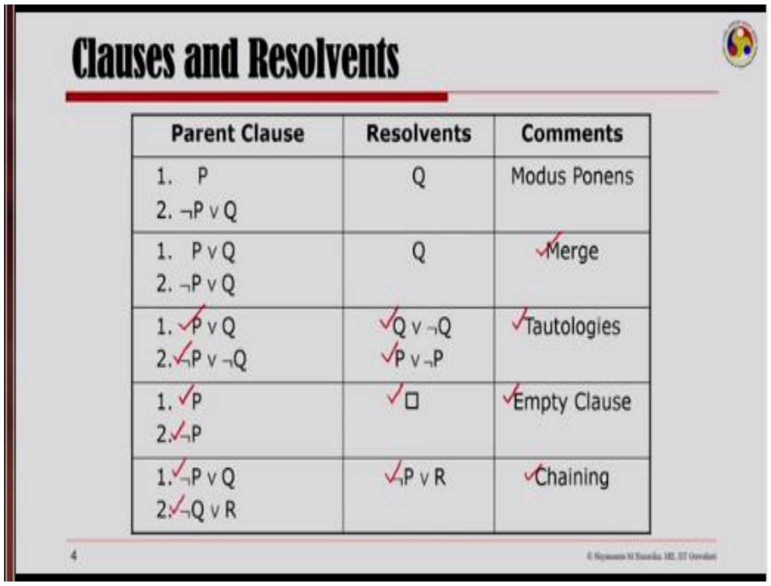
- We have two ground clauses:
  1. $P_1 \vee P_2 \vee P_3 \ldots \vee P_N$ No terms contain variables; we have a ground instance of the literal.
  2. $\neg P_1 \vee Q_1 \vee Q_2 \vee Q_3 \ldots \vee Q_M$
  - Assume in above **ground clauses** all of P_i and Q_j are distinct.
  - One of these clauses contains a literal that is the exact negation of one of the literals in the other clause.

    From the two above parent clauses; infer a new clause, called the resolvent of the two.
  - The **resolvent** is computed by taking the **disjunction of the two clauses** after **eliminating the complementary pair** P1 and ¬ P1.
- Resolution allows for **incorporation of several operations** into **one simple inference rule.**

3

Resolution for ground clauses is what we will look at first. Now ground clauses are those where no term contains variables, that is we have ground instance of the literals. So if we have 2 ground clauses P1 or P2 or P3 so on and so forth up to PN. So that is the disjunction of Ps and another ground clause which is at the disjunction of the Qs with the first literal being the negation of P1 then in above ground clauses, if all of Pis and Qjs are distinct and I want to resolve these clauses, clause 1 and clause 2 the resolvent of this clauses is computed by taking the disjunction of the 2 clauses. However, I would eliminate the complimentary pair that is P1 and negation of P1, resolution as we could see earlier in our discussion, allows incorporation of several operations into one simple inference rule, like earlier I had showed you how Modus Tolens and Modus Ponens could be seen as resolution.

(Refer Slide Time: 05:17)



Clauses and Resolvents

Parent Clause	Resolvents	Comments
1. P 2. ¬P ∨ Q	Q	Modus Ponens
1. P ∨ Q 2. ¬P ∨ Q	Q	✓Merge
1. ✓P ∨ Q 2.✓¬P ∨ ¬Q	✓Q ∨ ¬Q ✓P ∨ ¬P	✓Tautologies
1. ✓P 2.✓¬P	✓□	✓Empty Clause
1.✓¬P ∨ Q 2.✓¬Q ∨ R	✓¬P ∨ R	✓Chaining

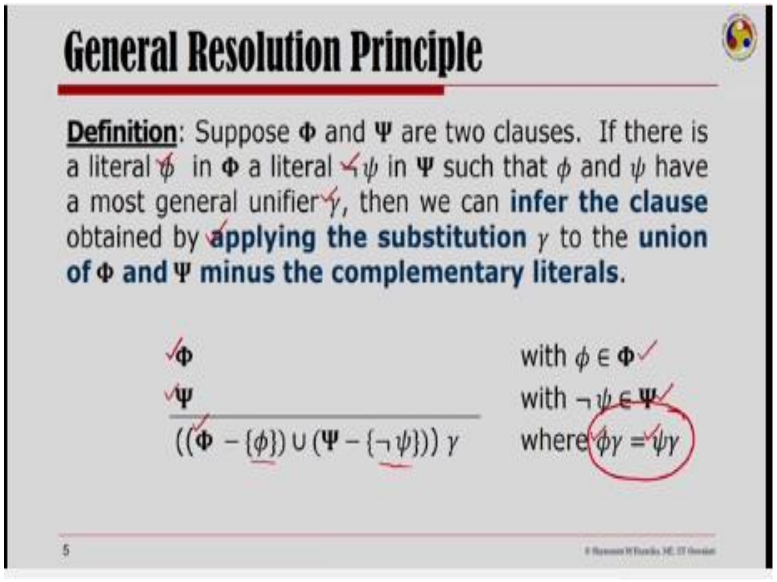
4

Let us look at a couple of more examples on the same lines, here are the parent clauses P and P implies Q which could be written as not P or Q. And therefore using Modus ponens I could write this is Q, the same thing I could do by resolving these 2 parent clauses and I could have a resolvent Q. Similarly, if I have 2 parent clauses P or Q and not P or Q, I could do a resolution of this to arrive at Q which is nothing but the merge operation arriving at tautologies can also be looked as resolution of 2 clauses.

If I have 2 clause P or Q and not P or not Q as my parent clauses, I could arrive at Q or not Q or I could arrive at P or not P which are tautologies. Here I have 2 clauses now P and not P if I resolve them I end up having the empty clause. Now this is written by an empty box that is the representation that we will use here for an empty clause. And if you take a moment to reflect having P and not P is about having a contradiction in the knowledge base, a contradiction when we resolve we end up having an empty clause.

The other example here is about chaining which is if I have P implies Q and I have Q implies R. So these implications are now written in clausal form as disjunctions of not P or Q and not Q or R and if I take these 2 clauses as parent clause and resolve them I have not P or R which is P implies R and this is the inference rule of chaining. So what we have seen here is that I can see different inference rules as simple resolution. So resolution is a very powerful rule of inference.

(Refer Slide Time: 08:35)



General Resolution Principle

Definition: Suppose Φ and Ψ are two clauses. If there is a literal ϕ in Φ a literal $\neg\psi$ in Ψ such that ϕ and ψ have a most general unifier γ, then we can **infer the clause** obtained by **applying the substitution** γ to the **union of Φ and Ψ minus the complementary literals.**

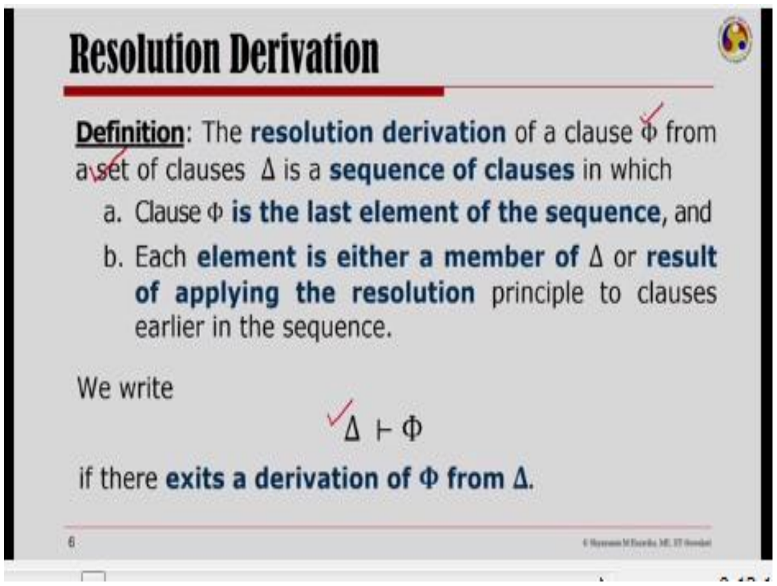
$$\frac{\begin{matrix}\Phi \\ \Psi\end{matrix}}{((\Phi - \{\phi\}) \cup (\Psi - \{\neg\psi\}))\,\gamma} \qquad \begin{matrix}\text{with } \phi \in \Phi \\ \text{with } \neg\psi \in \Psi \\ \text{where } \boxed{\phi\gamma = \psi\gamma}\end{matrix}$$

5

Having looked at this let us now try and give a general resolution principle, in formal terms the general resolution principle could be define as follows. Suppose I have 2 clauses phi and psi and there is a literal phi in my first clause and a literal not psi in my second clause such that these 2 literals phi and psi have a most general unifier gamma, what that means is that if I take that unifier and operate on phi and operate on psi these 2 will be equivalent.

So I have a clause phi and I have a clause psi, phi is a literal in the first clause and not psi is a literal in the second clause and they have a general unifier gamma. Then we can infer the clause obtained by applying substitution gamma to the union of phi and psi minus the complimentary literals. So basically it would mean that if I have a clause phi which has phi in it and if I have a clause psi which have a literal not psi in it such that if I operate with a unifier I have this equivalence of the given literals. Then the resolution is defined as the union of phi and psi minus the complimentary literals.

(Refer Slide Time: 10:51)



Resolution Derivation

Definition: The **resolution derivation** of a clause Φ from a set of clauses Δ is a **sequence of clauses** in which

a. Clause Φ **is the last element of the sequence**, and

b. Each **element is either a member of Δ** or **result of applying the resolution** principle to clauses earlier in the sequence.

We write

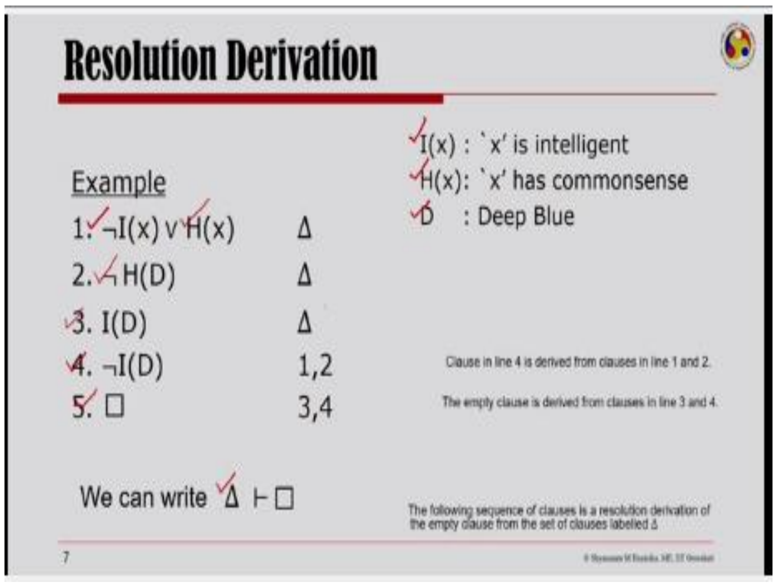
$$\Delta \vdash \Phi$$

if there **exits a derivation of Φ from Δ.**

So given such a definition of resolution, the resolution derivation of a clause phi from a set of clauses delta is a sequence of clauses in which phi is the last element of the sequence. And each element that I get in the sequence is either a member of delta or the result of applying the resolution principle to clauses earlier in the sequence. And if that happens, I can write that delta derives phi, so that means there exists a derivation of phi from delta.

So what I am saying here is the following, I start with a set of clauses delta and apply a resolution to this set of clauses as I apply resolution to this set of clauses, I generate new clauses and this proceeds until I generate the clause phi itself. Then I would say that I have a resolution derivation of clause phi.

(Refer Slide Time: 12:19)



Resolution Derivation

Example

1.	$\neg I(x) \vee H(x)$	Δ
2.	$\neg H(D)$	Δ
3.	$I(D)$	Δ
4.	$\neg I(D)$	1,2
5.	$\square$	3,4

We can write $\Delta \vdash \square$

I(x) : `x' is intelligent

H(x): `x' has commonsense

D : Deep Blue

Clause in line 4 is derived from clauses in line 1 and 2.

The empty clause is derived from clauses in line 3 and 4.

The following sequence of clauses is a resolution derivation of the empty clause from the set of clauses labelled Δ

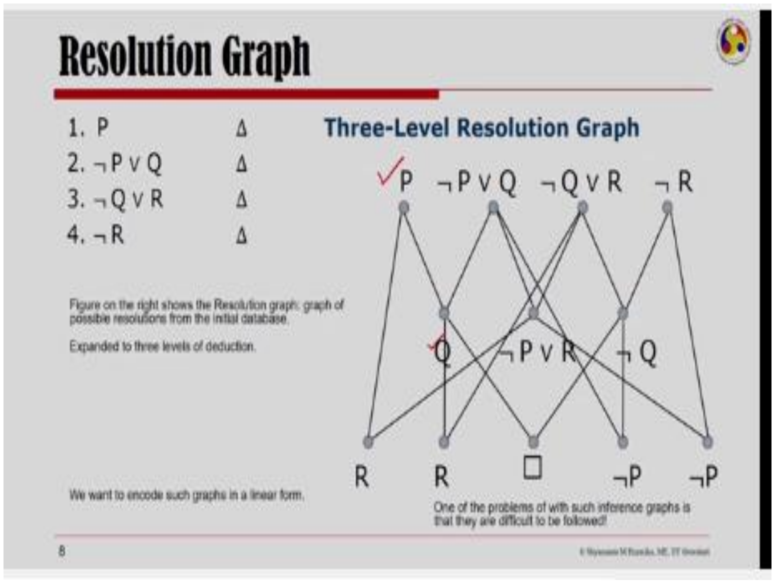
7

Let us try to understand this with this following example, I have a predicate here Ix to mean that x is intelligent, Hx to show that x has common sense. Let us take a constant D to mean the deep blue So the first statement is saying if something is intelligent, it has common sense. So I am writing Ix implies Hx I could write that as in clausal form as not of Ix or Hx and then the second statement is saying that deep blue does not have common sense.

And here I have a statement which is saying deep blue is intelligent, so this set of clauses delta I can now derive a new clause using 1 and 2. So, if I replace x for D this my H of D and the negative H of D will go away they are the complimentary literals. I will be left out with not of I of D and then I can look at the third clause together with the fourth clause which is I of D and not I of D, so that will lead me to the empty clause.

This sequence of clauses that I am getting starting from a set of clauses delta, finally arriving at the empty clause here is a resolution derivation and I can write that delta derives the empty clause or delta proves the empty clause. Very close to this concept of resolution derivation is the concept of a resolution graph.

(Refer Slide Time: 14:36)



Resolution Graph

1. P Δ
2. $\neg P \vee Q$ Δ
3. $\neg Q \vee R$ Δ
4. $\neg R$ Δ

Figure on the right shows the Resolution graph: graph of possible resolutions from the initial database.

Expanded to three levels of deduction.

We want to encode such graphs in a linear form.

Three-Level Resolution Graph

One of the problems of with such inference graphs is that they are difficult to be followed!

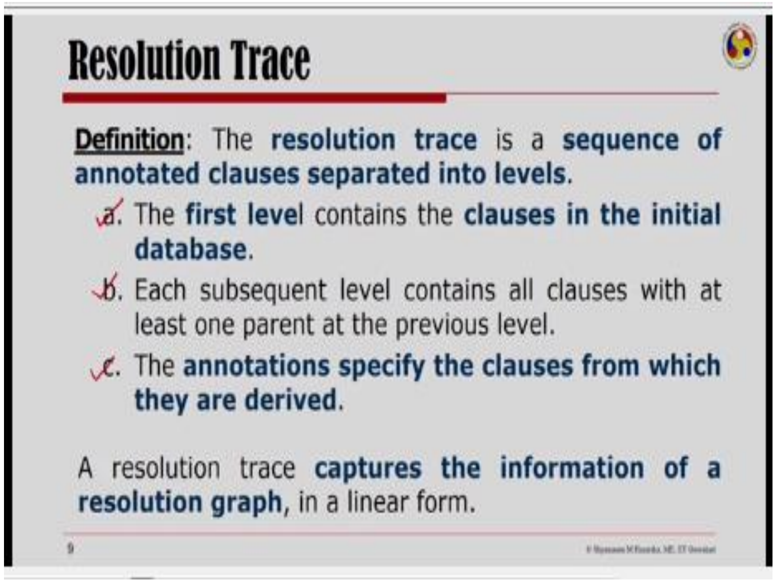
8

So if I have a set of clauses delta as shown on the left of your screen for ease of simplification of writing them, I have just taken propositional clauses. But it equally remains true for clauses that would have variables in them, so here I have P not P or Q not Q or R and not R that is the set of clauses delta. And on the right of your screen is the resolution graph that I could generate by resolving clauses in delta, so here are my clauses from delta.

I could resolve between P and not P or Q to get the Q, I could resolve not P or Q and not Q or R to give me not P or R so on and so forth. So here is a resolution graph up to the third level that is I have expanded to 3 levels of deduction. Now one of the problems with such inference graphs if you could see is that they are very difficult to follow and we would love something to be in a linear form.

So if we want to encode such graphs in a linear form, we would rather love that we keep the sequence of the resolution and that is done by something called a resolution trace.

(Refer Slide Time: 16:32)



Resolution Trace

Definition: The **resolution trace** is a **sequence of annotated clauses separated into levels.**

a. The **first level** contains the **clauses in the initial database.**

b. Each subsequent level contains all clauses with at least one parent at the previous level.

c. The **annotations specify the clauses from which they are derived.**

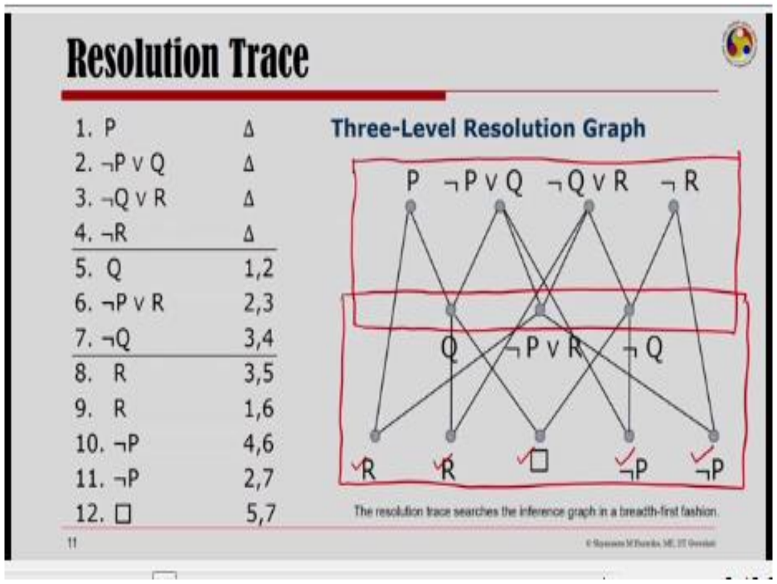
A resolution trace **captures the information of a resolution graph**, in a linear form.

9

So we will now try to define what is a resolution trace, the resolution trace is a sequence of annotated clauses, which are separated into levels. In the first level, I have clauses in the initial database and then each subsequent level contains clauses with at least one parent at the previous level. Now the annotations here specify the clauses from which they are derived it is important for us to realize that the resolution trace actually captures the information of a resolution graph.

But in a linear form, so that readability is improved and I can see how the resolution is proceeding.

(Refer Slide Time: 17:27)



Resolution Trace

1.	P	Δ
2.	$\neg P \vee Q$	Δ
3.	$\neg Q \vee R$	Δ
4.	$\neg R$	Δ
5.	Q	1,2
6.	$\neg P \vee R$	2,3
7.	$\neg Q$	3,4
8.	R	3,5
9.	R	1,6
10.	$\neg P$	4,6
11.	$\neg P$	2,7
12.	$\square$	5,7

Three-Level Resolution Graph

The resolution trace searches the inference graph in a breadth-first fashion.

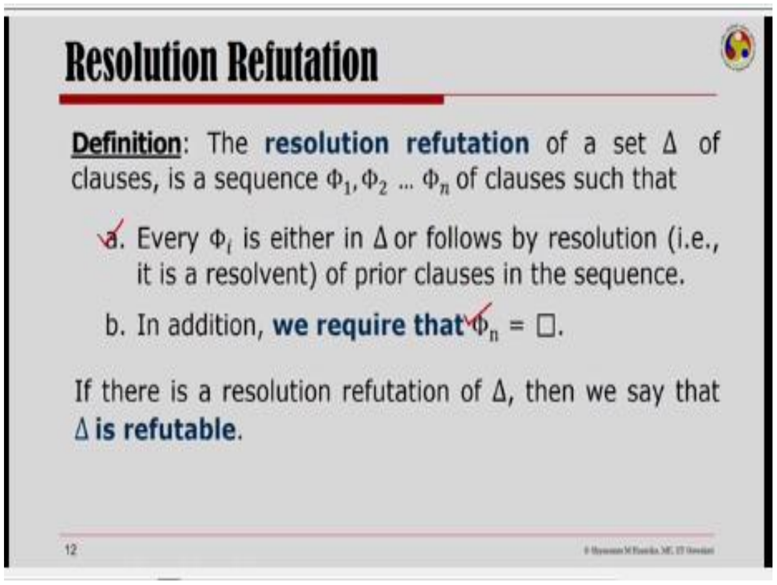
11

So here on the left of your screens is the resolution trace of the resolution graph that we have seen little while ago. At the first level you could see that I have all the clauses from the initial set of clauses delta and from them I could derive 3 clause Q not P or R and not Q. These 3 clauses form the second level of the resolution trace, the resolution trace as you can see captures the information of the resolution graph in a linear form.

And finally I can have the third level of the resolution trace which is the clauses that I derive at the third level of the resolution graph. Now a little reflection on the resolution trace, vis-a-vis the resolution graph is important. If you look at the resolution graph and the resolution tree, it should be obvious that when I am doing the first level of resolution, I am restricted to taking things from the parent clause and generating clauses at this level.

Next I am generating clauses at the second level here. So the resolution trace actually is searching the inference graph in a breadth-first fashion.

(Refer Slide Time: 19:27)



Resolution Refutation

Definition: The **resolution refutation** of a set Δ of clauses, is a sequence $\Phi_1, \Phi_2 \ldots \Phi_n$ of clauses such that

a. Every Φ_i is either in Δ or follows by resolution (i.e., it is a resolvent) of prior clauses in the sequence.

b. In addition, **we require that** $\Phi_n = \square$.

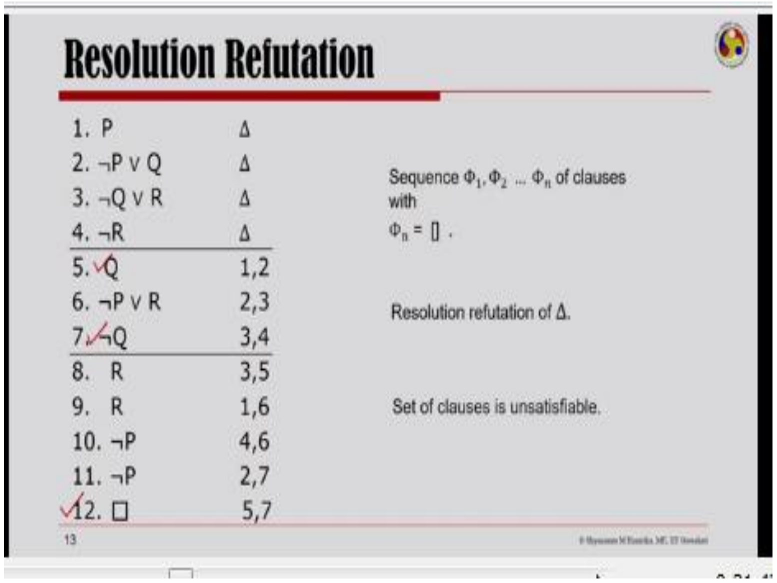
If there is a resolution refutation of Δ, then we say that **Δ is refutable.**

12

Now having introduced the concept of a derivation and a trace I would introduce what is resolution refutation. So the resolution refutation of a set delta of clauses is a resolution derivation that is a sequence of phi 1s, phi 2 up to phi n of clauses such that every phi i is either in delta or follows by resolution that is, it is a resolvent of prior clauses in the sequence. And in addition we require that the final clause in the resolution derivation delta n is an empty clause.

So a resolution refutation of a set delta of clauses to put simply is a resolution derivation that leads us to the empty clause. If there is a resolution refutation of delta, then we say that delta the set of clauses that I started with is refutable.

(Refer Slide Time: 20:43)



Resolution Refutation

1. P	Δ
2. $\neg P \vee Q$	Δ
3. $\neg Q \vee R$	Δ
4. $\neg R$	Δ
5. Q	1,2
6. $\neg P \vee R$	2,3
7. $\neg Q$	3,4
8. R	3,5
9. R	1,6
10. $\neg P$	4,6
11. $\neg P$	2,7
12. $\square$	5,7

Sequence $\Phi_1, \Phi_2 \ldots \Phi_n$ of clauses with $\Phi_n = []$.

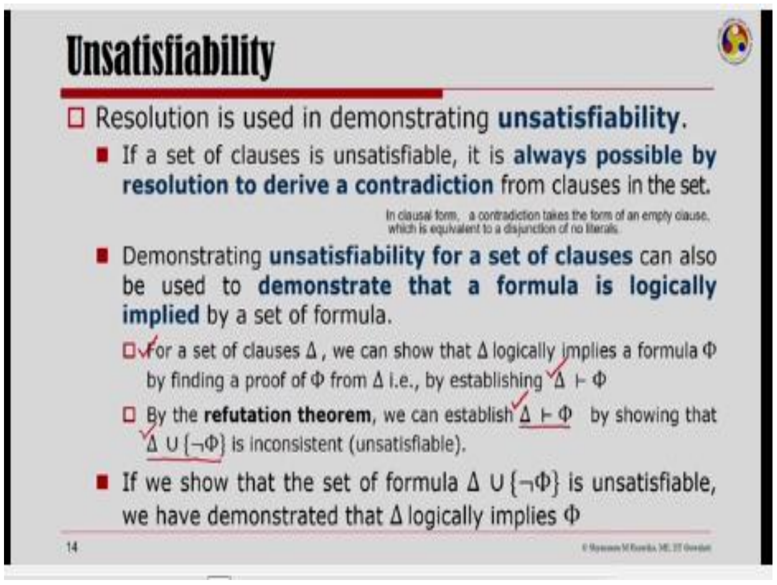
Resolution refutation of Δ.

Set of clauses is unsatisfiable.

13

So here is the resolution refutation of the set of clause delta because as you can see I have a resolution trace and in that resolution trace you can see that finally I have the empty clause. So the sequence of delta 1, delta 2 to delta n finally comes back with delta n as the empty clause and that is the resolution refutation of delta. One interesting thing to note about this is that the set of clauses if you look here, the set of clauses which had lead to a empty clause that means, which had prove to be refutable, it is actually unsatisfiable because you could see in the very first level I could have things like Q and not Q.

(Refer Slide Time: 21:42)



Unsatisfiability

- Resolution is used in demonstrating **unsatisfiability**.
  - If a set of clauses is unsatisfiable, it is **always possible by resolution to derive a contradiction** from clauses in the set.

    In clausal form, a contradiction takes the form of an empty clause, which is equivalent to a disjunction of no literals.
  - Demonstrating **unsatisfiability for a set of clauses** can also be used to **demonstrate that a formula is logically implied** by a set of formula.
    - ✓For a set of clauses Δ, we can show that Δ logically implies a formula Φ by finding a proof of Φ from Δ i.e., by establishing ✓$\Delta \vdash \Phi$
    - By the **refutation theorem**, we can establish ✓$\underline{\Delta \vdash \Phi}$ by showing that ✓$\underline{\Delta \cup \{\neg\Phi\}}$ is inconsistent (unsatisfiable).
  - If we show that the set of formula $\Delta \cup \{\neg\Phi\}$ is unsatisfiable, we have demonstrated that Δ logically implies Φ

14

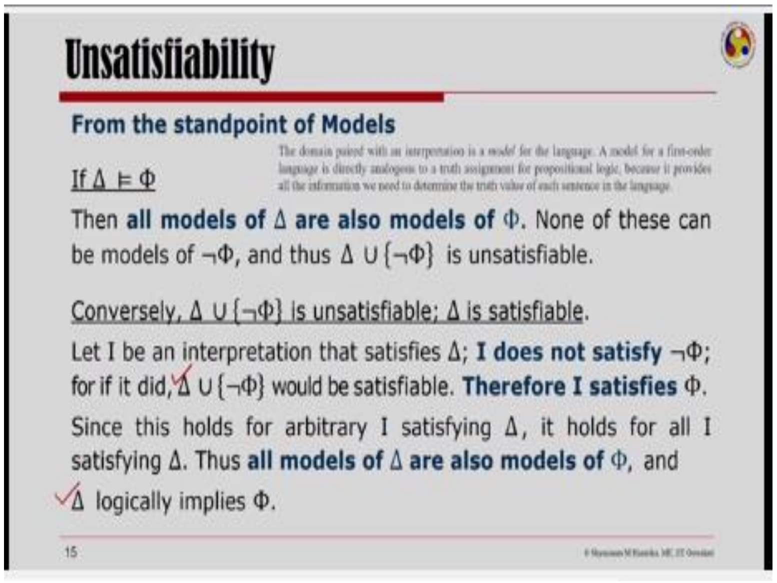
Now let us try and understand what do we mean by unsatisfiability and when can I have unsatisfiability in a set of clauses. Resolution is used in demonstrating unsatisfiability, so if a set of clauses is unsatisfiable it is always possible by resolution to derive a contradiction from clauses in the set. In clausal form a contradiction takes the form of an empty clause which is actually a disjunction of no literals.

So whenever we arrive at an empty clause in resolution we arrive at a contradiction, now demonstrating unsatisfiability for a set of clauses can also be used to demonstrate that a formula is logically implied by the set of formula. This is very important for us to realize because this leads us to the concept of how we arrive at the proof of a given statement from a knowledge base. So for a set of clauses delta we can show that delta logically implies a formula phi by finding a proof of phi from delta that is if we could apply some resolution and we could establish that delta derives phi. Then we could show that delta logically implies phi. Now by the refutation theorem we establish that delta derives phi by showing that the union of delta and not phi is inconsistent. So it is important for us to reflect what we mean by this statement here.

Here we are trying to say that we can establish delta proves phi, for that we will not derive phi from delta, what we will rather do is we will take the union of not of phi and delta and prove it to be inconsistent. Let me give you the intuitive feeling of why this would be true, very loosely you can think of this as following: When I have delta and added up to not of phi, if that becomes inconsistent or unsatisfiable looking in this set of clauses somewhere I already have phi that is why adding not of phi brings in a contradiction.

And that is why the refutation theorem, we can show that delta derives phi by showing the union of delta and not phi is inconsistent. So if we show that the set of formula delta union not phi is unsatisfiable, we have demonstrated that delta logically implies phi.

(Refer Slide Time: 25:15)



Unsatisfiability

From the standpoint of Models

The domain paired with an interpretation is a *model* for the language. A model for a first-order language is directly analogous to a truth assignment for propositional logic, because it provides all the information we need to determine the truth value of each sentence in the language.

If $\Delta \models \Phi$

Then **all models of Δ are also models of Φ**. None of these can be models of $\neg\Phi$, and thus $\Delta \cup \{\neg\Phi\}$ is unsatisfiable.

Conversely, $\Delta \cup \{\neg\Phi\}$ is unsatisfiable; Δ is satisfiable.

Let I be an interpretation that satisfies Δ; **I does not satisfy** $\neg\Phi$; for if it did, ✓$\Delta \cup \{\neg\Phi\}$ would be satisfiable. **Therefore I satisfies** Φ.

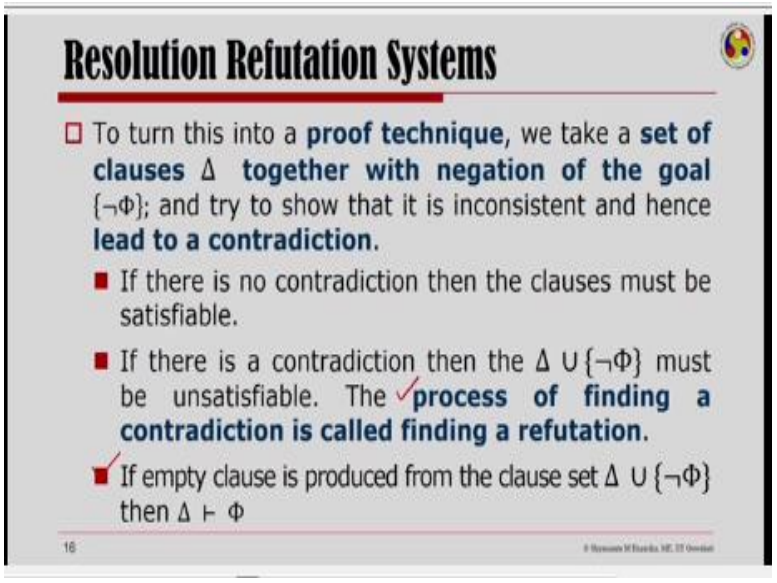
Since this holds for arbitrary I satisfying Δ, it holds for all I satisfying Δ. Thus **all models of Δ are also models of Φ**, and ✓Δ logically implies Φ.

15 © Shyamanta M Hazarika, ME, IIT Guwahati

So let us try to understand the same from the standpoint of models or just to remind you a model is the domain paired with an interpretation. So a model for a first order language is directly analogous to a truth assignment for propositional logic. Because it provides all the information we need to demonstrate the truth value of each sentence in the language. So if delta entails phi that means all models of delta are also models of phi then none of these can be models of not phi and thus delta union not phi is unsatisfiable.

On the contrary conversely if we have delta union not phi is unsatisfiable that is delta is satisfiable, let us assume an interpretation I that satisfies delta. So as I satisfies delta I does not satisfy not phi, for if it did, then it would make delta union not phi satisfiable. Therefore, interpretation I only satisfies phi since this holds for any arbitrary interpretation satisfying delta it holds for all interpretations satisfying delta that is, all models of delta are also models of phi and we can say that delta logically implies phi.

(Refer Slide Time: 27:06)



Resolution Refutation Systems

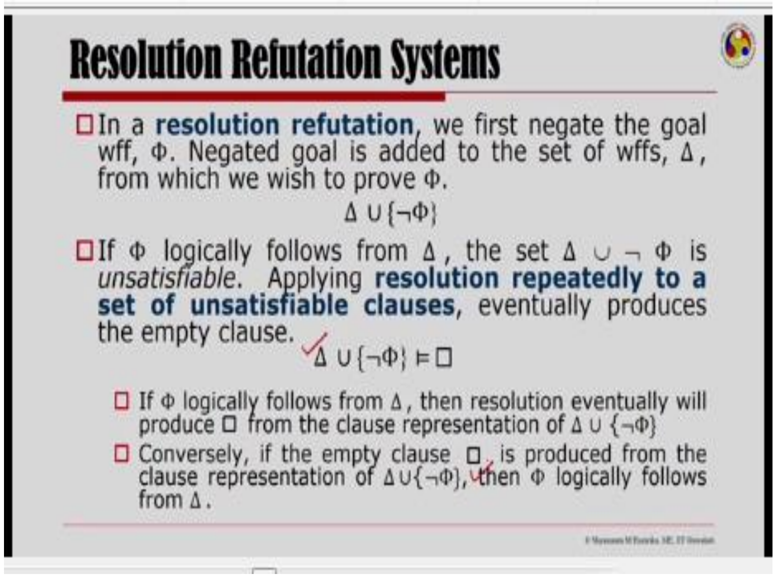
- To turn this into a **proof technique**, we take a **set of clauses Δ together with negation of the goal** {¬Φ}; and try to show that it is inconsistent and hence **lead to a contradiction.**
  - If there is no contradiction then the clauses must be satisfiable.
  - If there is a contradiction then the Δ ∪ {¬Φ} must be unsatisfiable. The ✓**process of finding a contradiction is called finding a refutation.**
  - ✓If empty clause is produced from the clause set Δ ∪ {¬Φ} then Δ ⊢ Φ

16

Now to turn this into a proof technique this is what we have to do, we take a set of clauses delta and together with that, we take the negation of the goal not of phi and try to show that it is consistent and lead to a contradiction. If there is no contradiction, then the clauses must be satisfiable. If there is a contradiction, then delta union not phi must be unsatisfiable and the process of finding a contradiction is actually called finding a refutation.

If empty clauses produced from the clauses delta union not phi then we have already argued we can say that delta proves phi. Now these types of systems are called resolution refutation systems precisely because the process of finding a contradiction is called finding a refutation.

(Refer Slide Time: 28:16)



Resolution Refutation Systems

- In a **resolution refutation**, we first negate the goal wff, Φ. Negated goal is added to the set of wffs, Δ, from which we wish to prove Φ.

$$\Delta \cup \{\neg\Phi\}$$

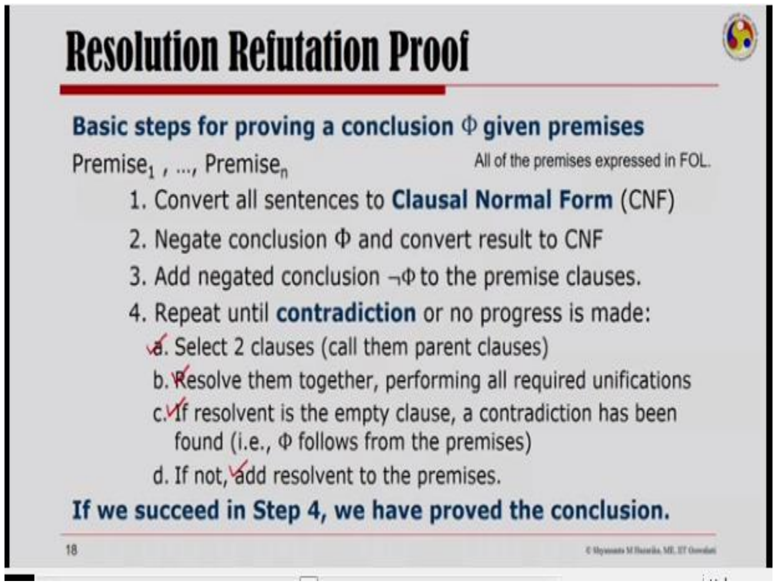
- If Φ logically follows from Δ, the set $\Delta \cup \neg \Phi$ is *unsatisfiable*. Applying **resolution repeatedly to a set of unsatisfiable clauses**, eventually produces the empty clause.

$$\Delta \cup \{\neg\Phi\} \models \square$$

  - If Φ logically follows from Δ, then resolution eventually will produce $\square$ from the clause representation of $\Delta \cup \{\neg\Phi\}$
  - Conversely, if the empty clause $\square$ is produced from the clause representation of $\Delta \cup \{\neg\Phi\}$, then Φ logically follows from Δ.

So in a resolution refutation we first negate the goal, negated goal is added to the set of clauses so I have delta union not phi. If phi logically follows from delta, the set delta union not phi is unsatisfiable. And then we have seen that, if I have an unsatisfiable set of clauses applying resolution repeatedly would eventually produce the empty clause. So I would have delta union not phi entailing the empty clause and then we have already argued that if I arrive at an empty clause then actually I know that phi logically follows from delta.

(Refer Slide Time: 29:09)



Resolution Refutation Proof

Basic steps for proving a conclusion Φ given premises

Premise_1, ..., Premise_n All of the premises expressed in FOL.

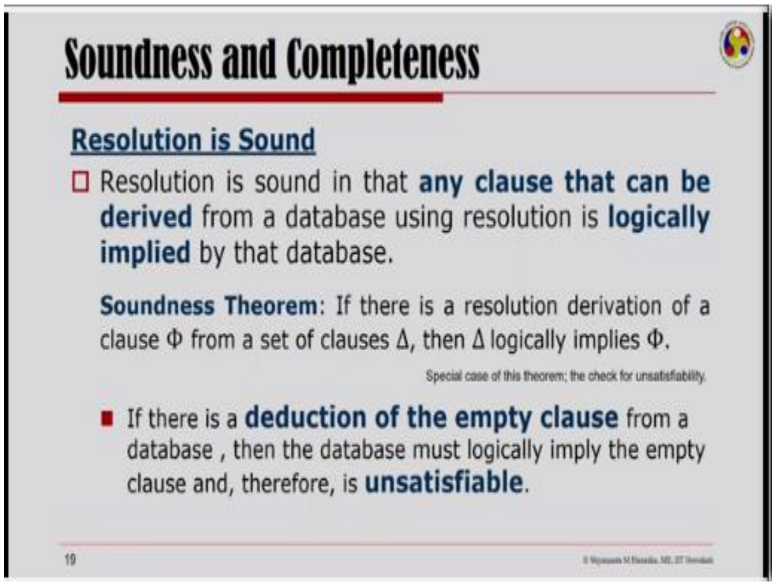
1. Convert all sentences to **Clausal Normal Form** (CNF)
2. Negate conclusion Φ and convert result to CNF
3. Add negated conclusion $\neg\Phi$ to the premise clauses.
4. Repeat until **contradiction** or no progress is made:
   a. Select 2 clauses (call them parent clauses)
   b. Resolve them together, performing all required unifications
   c. If resolvent is the empty clause, a contradiction has been found (i.e., Φ follows from the premises)
   d. If not, add resolvent to the premises.

If we succeed in Step 4, we have proved the conclusion.

18

So here is the resolution refutation and algorithm the basic steps for proving a conclusion phi given the premises. So let us assume we have a set of premises and all of these are expression first order logic. So as I was emphasizing the first step here would be to convert all sentences to clausal normal form. Then we negate the conclusion phi and convert the result into clausal normal form at the negated conclusion to the premise clauses and then repeat the process of resolution we repeat until contradiction or until no progress is made the following: we select 2 clauses we call them the parent clause and then we resolve them together performing all required unification. Now if the resolvent is an empty clause we have already found the contradiction and we can conclude that phi follows from the premises. If not we add the resolvent that we have arrived in the step c here to the premises and repeat the step 4, now if we succeed in step 4 we have proved the conclusion.

(Refer Slide Time: 30:51)



Soundness and Completeness

Resolution is Sound

- Resolution is sound in that **any clause that can be derived** from a database using resolution is **logically implied** by that database.

  Soundness Theorem: If there is a resolution derivation of a clause Φ from a set of clauses Δ, then Δ logically implies Φ.

  Special case of this theorem; the check for unsatisfiability.

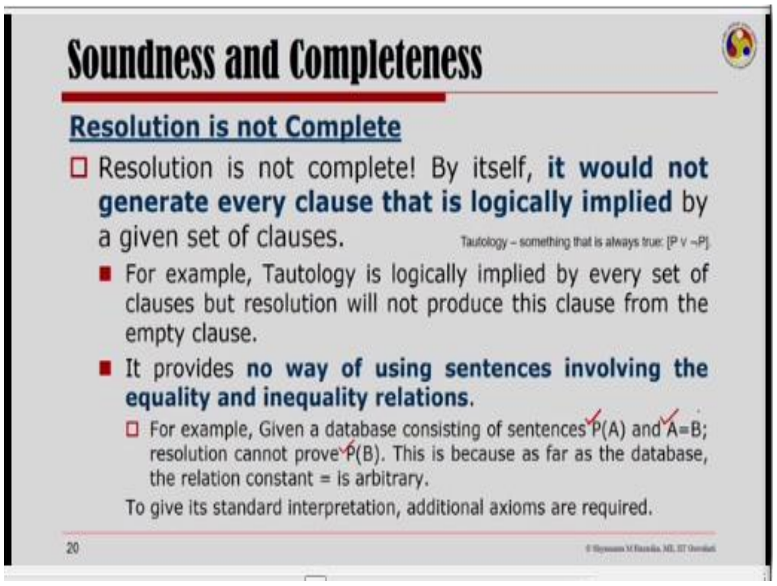
  - If there is a **deduction of the empty clause** from a database , then the database must logically imply the empty clause and, therefore, is **unsatisfiable**.

19

Let us look at now a couple of properties of resolution, the first one states it is soundness, so resolution is sound, what it means is that any clause that can be derive from a database using resolution is logically implied by the database. Here is the soundness theorem, so if there is a resolution derivation of a clause phi from a set of clauses delta then delta logically implies phi. So the special case of this theorem is the check for unsatisfiability for us.

And the deduction of the empty clause from a database would mean that the database must logically imply the empty clause and therefore is unsatisfiable.

(Refer Slide Time: 31:48)



Soundness and Completeness

Resolution is not Complete

- Resolution is not complete! By itself, **it would not generate every clause that is logically implied** by a given set of clauses. Tautology – something that is always true: [P ∨ ¬P].
  - For example, Tautology is logically implied by every set of clauses but resolution will not produce this clause from the empty clause.
  - It provides **no way of using sentences involving the equality and inequality relations.**
    - For example, Given a database consisting of sentences P(A) and A=B; resolution cannot prove P(B). This is because as far as the database, the relation constant = is arbitrary.

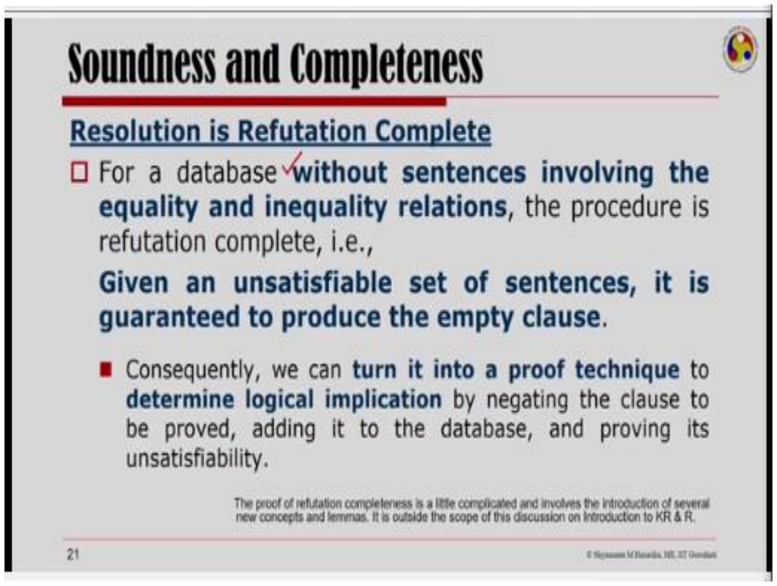
    To give its standard interpretation, additional axioms are required.

20

The second property of resolution is something that we need to be very clear about when we are doing resolution refutation proofs, resolution is not complete, what I mean by it is that by itself, it would not generate every clause that is logically implied by a given set of clauses. For example, tautology is logically implied by every set of clauses but resolution will not produce this clause from the empty clause.

Now resolution provides no way of using sentences involving equality and inequality relations. For example, if in a data base I have sentences PA and somewhere an equality statement A = B now resolution cannot prove P of B which is very obvious to the reader because I can see that I have PA and I know P = B and therefore I should be able to get PB. Resolution cannot derive that because as far as the database, the relation constant equal is arbitrary now to give it standard interpretation additional axioms are required.

(Refer Slide Time: 33:28)



Soundness and Completeness

Resolution is Refutation Complete

- For a database **without sentences involving the equality and inequality relations**, the procedure is refutation complete, i.e.,

  Given an unsatisfiable set of sentences, it is guaranteed to produce the empty clause.

  - Consequently, we can **turn it into a proof technique** to **determine logical implication** by negating the clause to be proved, adding it to the database, and proving its unsatisfiability.

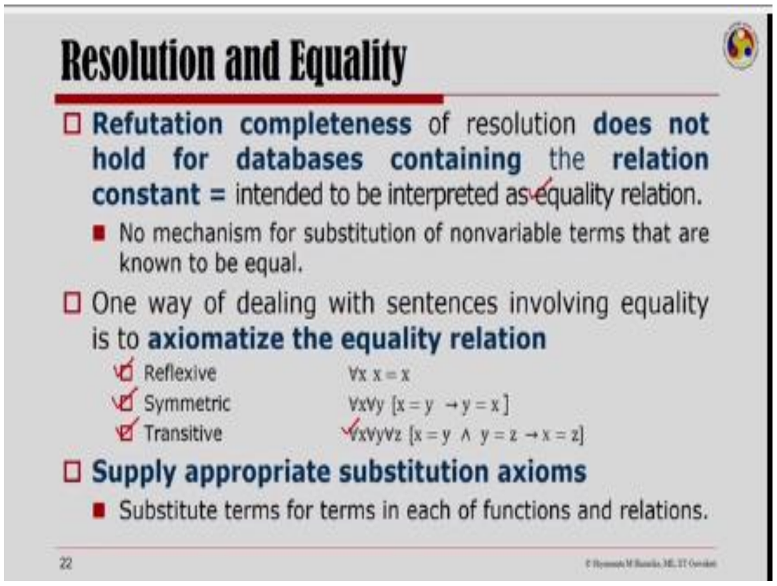
The proof of refutation completeness is a little complicated and involves the introduction of several new concepts and lemmas. It is outside the scope of this discussion on Introduction to KR & R.

21 © Shyamanta M Hazarika, ME, IIT Guwahati

We will look at this as part of a lecture today, however resolution is refutation complete that is if I have a database without sentences that involve equality and inequality then given an unsatisfiable set of sentences, it is guaranteed to produce the empty clause. It is this property of resolution that we exploit to get of rules. So consequently we have a prove technique to determine logical implication by negating the clause to be proof as we have seen adding it to the database and proving it is unsatisfiability.

This is precisely possible because resolution is refutation complete the proof of refutation completeness is a little complicated and involves introduction of several new concepts and lemmas. I feel it is outside the scope of this discussion on a fundamental introduction to knowledge representation and reasoning we will not cover it here.

(Refer Slide Time: 34:45)



Resolution and Equality

- Refutation completeness of resolution does not hold for databases containing the relation constant = intended to be interpreted as equality relation.
  - No mechanism for substitution of nonvariable terms that are known to be equal.
- One way of dealing with sentences involving equality is to **axiomatize the equality relation**

✓ Reflexive	$\forall x\ x = x$
✓ Symmetric	$\forall x \forall y\ [x = y \rightarrow y = x]$
✓ Transitive	$\forall x \forall y \forall z\ [x = y \wedge y = z \rightarrow x = z]$

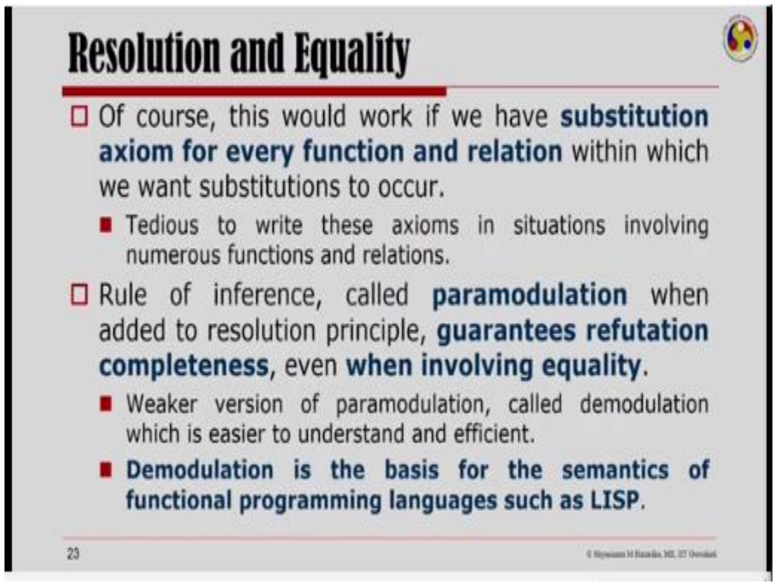
- **Supply appropriate substitution axioms**
  - Substitute terms for terms in each of functions and relations.

22 [illegible]

Now let us take some time to understand the link between resolution and equality. Refutation completeness of resolution as we have stated before does not hold databases contain the relation constant of equality which is intended to be interpreted as a equality relation. Now there is no mechanism for substitution of non variable terms that are known to be equal. One way of dealing with sentences involving equality is to axiomatize the equality relation.

Now we know equality is reflexive, symmetric and transitive, so we could introduce into a set of clauses, axioms to state equality being reflexive, symmetric and transitive. Here are the 3 axioms for all x, x = x is trying to state that the equality relation is reflexive. I could have symmetric for all x, for all y, x = y would imply y = x and the transitive, with t of equality saying for 3 variables x, y and z. If x = y, and y = z this would imply that x = z, another way to have refutation completeness of databases containing the equality relation is to supply appropriate substitution axioms. So this could substitute terms for terms in each of functions and relations.

(Refer Slide Time: 36:47)



Resolution and Equality

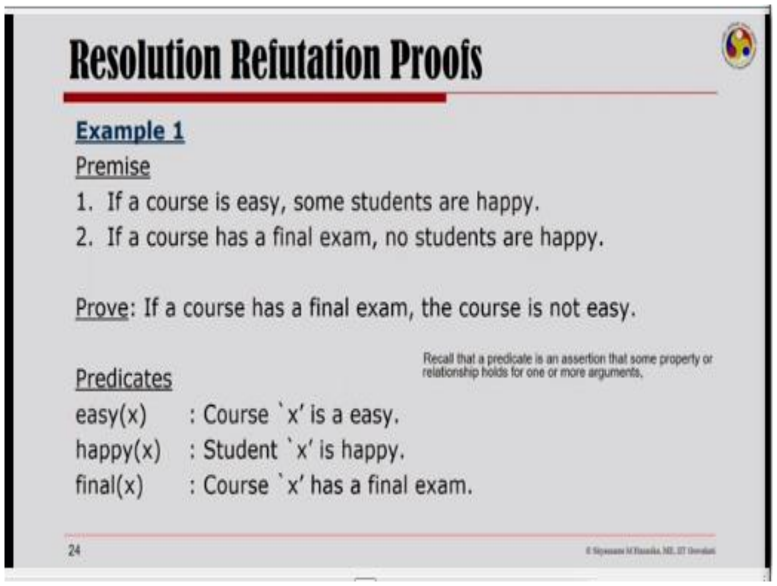
- Of course, this would work if we have **substitution axiom for every function and relation** within which we want substitutions to occur.
  - Tedious to write these axioms in situations involving numerous functions and relations.
- Rule of inference, called **paramodulation** when added to resolution principle, **guarantees refutation completeness**, even **when involving equality**.
  - Weaker version of paramodulation, called demodulation which is easier to understand and efficient.
  - **Demodulation is the basis for the semantics of functional programming languages such as LISP.**

23

Now this would work if we have substitution axioms for every function and relation within which we want substitutions to occur. Now this would be tedious to write this axioms there are other rules of inference called paramodulation which when added to resolution principle guarantee refutation completeness even when involving equality. There are weaker versions of paramodulation called demodulation.

And demodulation is the basis for the semantics of functional programming languages like LISP, however for our discussion here, we would not cover these rules of inferences.

(Refer Slide Time: 37:33)



Resolution Refutation Proofs

Example 1

Premise

1. If a course is easy, some students are happy.
2. If a course has a final exam, no students are happy.

Prove: If a course has a final exam, the course is not easy.

Recall that a predicate is an assertion that some property or relationship holds for one or more arguments.

Predicates

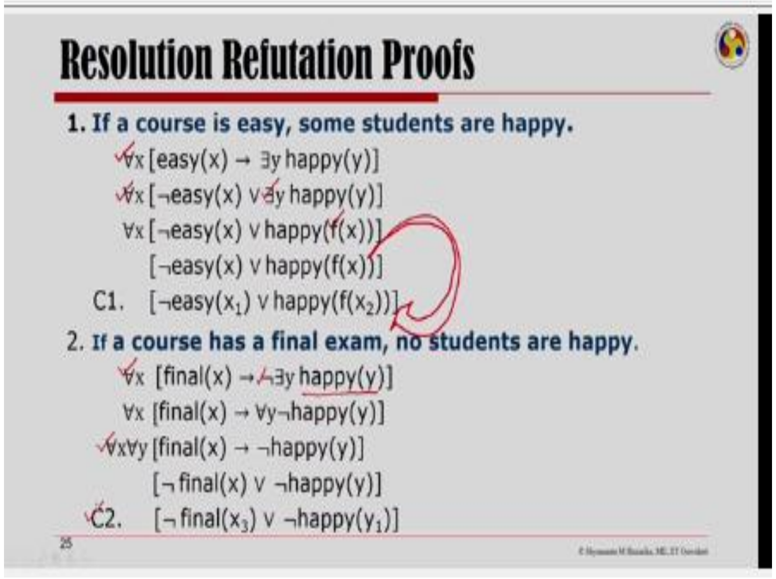
easy(x)	: Course `x' is a easy.
happy(x)	: Student `x' is happy.
final(x)	: Course `x' has a final exam.

24

Let us now take a couple of examples to understand resolution refutation proofs so here is a first example, here is the premise. The first statement is saying if a course is easy some students are happy the second one is stating if a course has a final exam, no students are happy. The idea is to prove now that if a course has a final exam the course is not easy. Let us take the following predicates now recall that a predicate is an assertion that some property or relationship for one or more arguments hold.

So we take a predicate called easy x to say course x is easy, we have another predicate happy x to means student x is happy. We introduce a third predicate final x to mean course x has a final exam.

(Refer Slide Time: 38:39)



Resolution Refutation Proofs

1. **If a course is easy, some students are happy.**

   $\forall x\,[\text{easy}(x) \rightarrow \exists y\, \text{happy}(y)]$

   $\forall x\,[\neg\text{easy}(x) \vee \exists y\, \text{happy}(y)]$

   $\forall x\,[\neg\text{easy}(x) \vee \text{happy}(f(x))]$

   $[\neg\text{easy}(x) \vee \text{happy}(f(x))]$

   C1. $[\neg\text{easy}(x_1) \vee \text{happy}(f(x_2))]$

2. **If a course has a final exam, no students are happy.**

   $\forall x\,[\text{final}(x) \rightarrow \neg\exists y\, \underline{\text{happy}(y)}]$

   $\forall x\,[\text{final}(x) \rightarrow \forall y \neg\text{happy}(y)]$

   $\forall x \forall y\,[\text{final}(x) \rightarrow \neg\text{happy}(y)]$

   $[\neg\,\text{final}(x) \vee \neg\text{happy}(y)]$

   C2. $[\neg\,\text{final}(x_3) \vee \neg\text{happy}(y_1)]$

25

Now let us try and convert the premises to causal normal forms. If a course is easy, some students are happy. We can write this in first order saying for all x, x is easy, so course is easy there exists some students who are happy. So here is the first order predicate calculus formula for the above statement now in order to apply resolution this has to be converted into a clause. So the first thing that is to be done is to replace the implication.

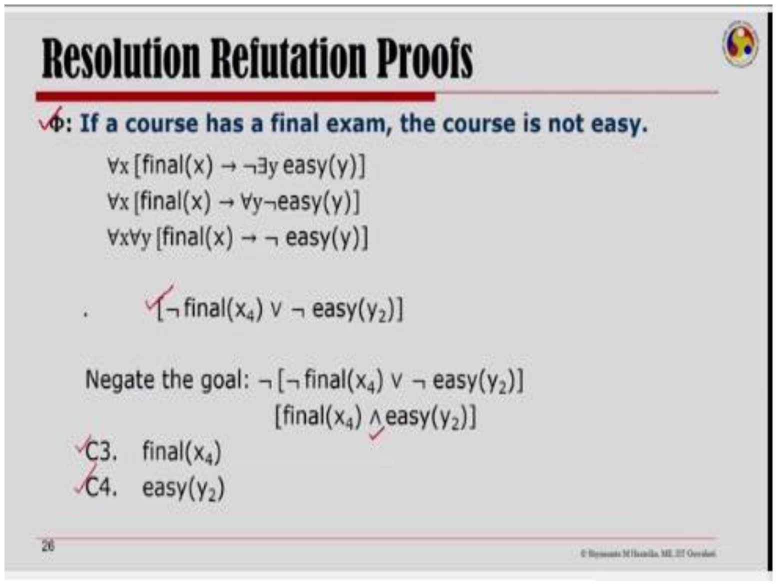
So that implication P implies Q is replaced as not P or Q and now we will try to remove the existential quantifier this is very important to remember from our last discussion that here this y is dependent on the universal quantifier x. And therefore we would introduce what is called a

Skolem function f of x to replace the existential quantifier there exist y. Now we are in a position to move it to prenex form.

And we have our first clause, which says not easy x1 or happy f of x2. Now this is important for us to realize that while coming from this to this clausal form here, we have replaced every variable with a new variable that is not occurring anywhere else. Let us now take our second statement. If a course has a final axiom no students are happy, so this again involves an implication but then when I am saying no students are happy, I am looking for a negation.

So here is the first order predicate calculus statement, which is saying for all x which have a final exam, there does not exist any y such that happy y is true. So no students are happy, now when you convert this to a clausal form this negation needs to be first pushed in to apply to only this portion or the atomic portion and we push that inside. So not there exist y becomes for all y not of happy y. Thereafter we could eliminate the implication to write a statement saying not of final x or not of happy y and all of the universal quantifiers are at the front. So from this I generate my second clause which is not final x3 or not happy y1. Now remember I have now introduce new variables, while writing clause 2, what I wanted to prove is the statement that if a course has a final exam the course is not easy that is my phi.

(Refer Slide Time: 42:25)



Resolution Refutation Proofs

Φ: If a course has a final exam, the course is not easy.

$\forall x\,[final(x) \rightarrow \neg\exists y\, easy(y)]$

$\forall x\,[final(x) \rightarrow \forall y \neg easy(y)]$

$\forall x \forall y\,[final(x) \rightarrow \neg\, easy(y)]$

$[\neg\, final(x_4) \vee \neg\, easy(y_2)]$

Negate the goal: $\neg\,[\neg\, final(x_4) \vee \neg\, easy(y_2)]$

$[final(x_4) \wedge easy(y_2)]$

C3. $final(x_4)$

C4. $easy(y_2)$

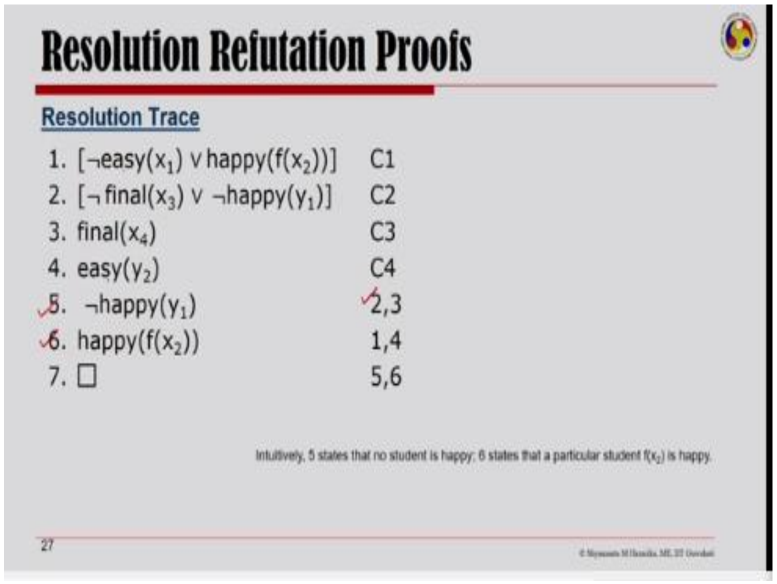
26

© Shyamanta M Hazarika, ME, IIT Guwahati

So we have to first convert it into clausal normal form and thereafter the negation has to be added to delta. So here is the statement for all x final x if the course has a final exam then the course is not easy. So that is what the statement is saying, so as before you push the negation inside and replace it by for all y there exist has to be replace now for all y and the negation will go inside and then you will have the implication replaced.

So once I replace the implication I have the following statement which is saying not of final x4 or not of easy y2. So that is the clausal normal form of phi now in order to take this proof forward I must negate the goal, so, here is the negation of that statement and this is like not of P or Q which would be like P and Q. So I end up having final x4 and easy y2. Now this being a conjunction here, I can write this as 2 separate statements final x4 that is my third clause and easy y2 that is my fourth clause, so the negation of the goal lead to C3 and C4.

(Refer Slide Time: 44:25)



Resolution Refutation Proofs

Resolution Trace

1.	$[\neg easy(x_1) \vee happy(f(x_2))]$	C1
2.	$[\neg final(x_3) \vee \neg happy(y_1)]$	C2
3.	$final(x_4)$	C3
4.	$easy(y_2)$	C4
✓5.	$\neg happy(y_1)$	✓2,3
✓6.	$happy(f(x_2))$	1,4
7.	$\Box$	5,6

Intuitively, 5 states that no student is happy; 6 states that a particular student $f(x_2)$ is happy.

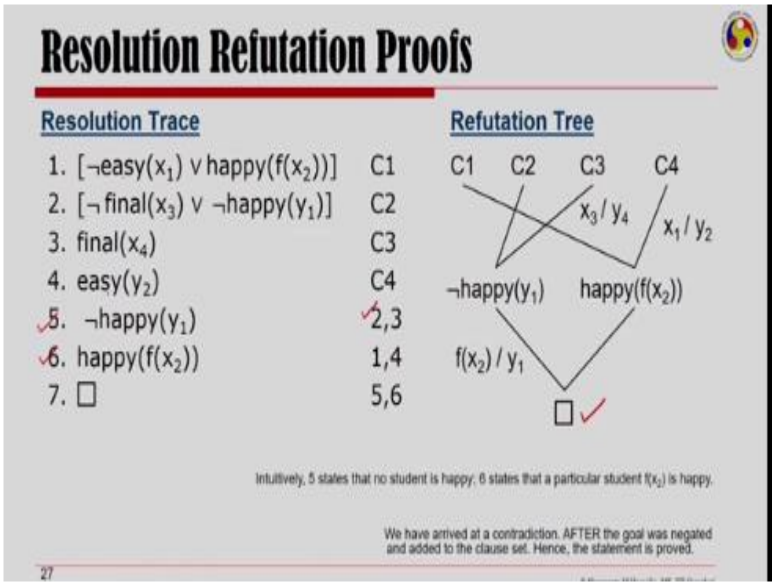
27

© Shyamanta M Hazarika, ME, IIT Guwahati

I now have the resolution trace of these example, here is the set of initial clauses that I start with and I derive not happy y1 by taking clause 2 and clause 3. So taking 2 and 3, this final and not final goes away, so I have not happy y1. From 1 and 4, I could remove this complimentary clauses easy and not easy to have happy f x2. And then I have a contradiction here from 5 and 6 now how could I get a contradiction from 5 and 6 intuitively the clause 5 is saying that no student is happy.

The clause 6 is saying that a particular student f of x2 is happy and therefore I have a contradiction.

(Refer Slide Time: 45:41)



C1
C2
C3
C4
x_3 / y_4
x_1 / y_2
$\neg happy(y_1)$
$happy(f(x_2))$
$f(x_2) / y_1$
$\square$ ✓

Resolution Refutation Proofs

Resolution Trace

1.	$[\neg easy(x_1) \vee happy(f(x_2))]$	C1
2.	$[\neg final(x_3) \vee \neg happy(y_1)]$	C2
3.	$final(x_4)$	C3
4.	$easy(y_2)$	C4
✓5.	$\neg happy(y_1)$	✓2,3
✓6.	$happy(f(x_2))$	1,4
7.	$\square$	5,6

Refutation Tree

Intuitively, 5 states that no student is happy; 6 states that a particular student $f(x_2)$ is happy.

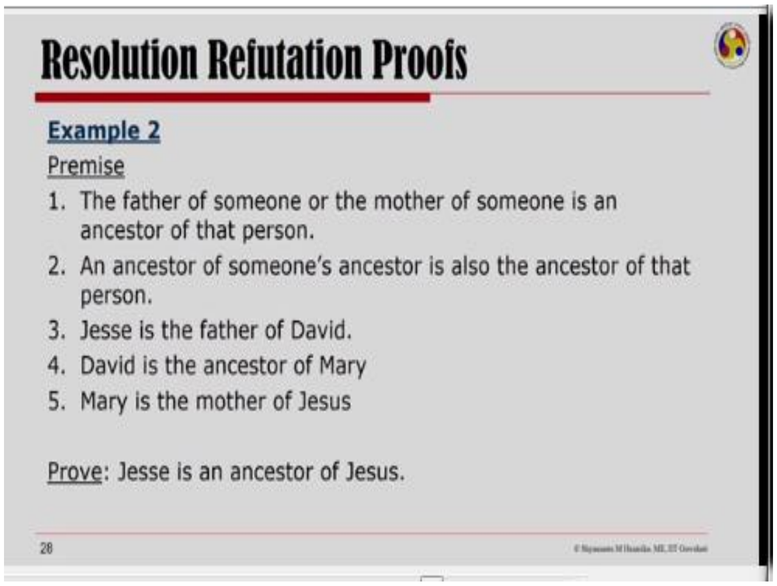
We have arrived at a contradiction. AFTER the goal was negated and added to the clause set. Hence, the statement is proved.

27

Now for the same resolution trace I could create the refutation tree and here is my refutation tree. So if you now see I had my first derive clause f1 which is between C2 and C3 I had f1 and the substitution that was used while getting not happy y1 from 2 and 3 was that x3 need to be like here, I had the substitution for 2 and 3 as x3 for x4. And then I could have happy fx y2 from 3 and 4, so here is the refutation tree for the resolution and trace on your right.

And we could see that we have arrived that a empty clause, a contradiction after the goal was negated and added to the clause set hence the statement is proved.

(Refer Slide Time: 46:55)



Resolution Refutation Proofs

Example 2

Premise

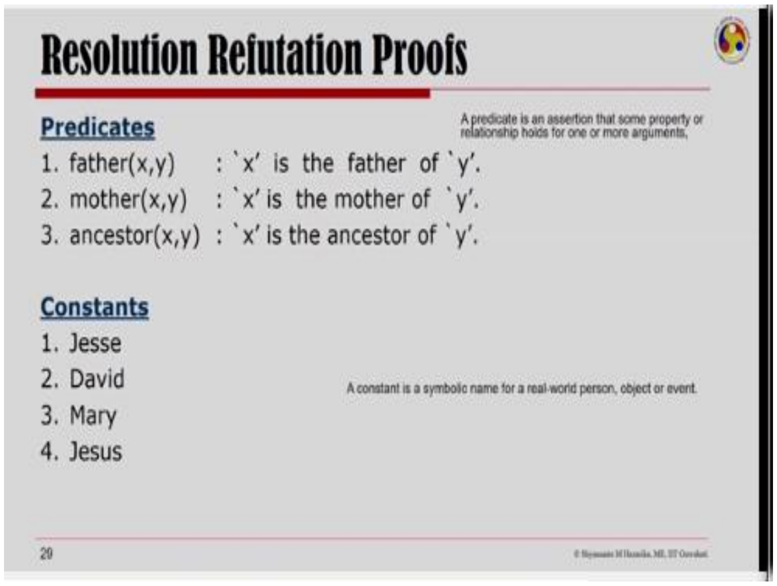
1. The father of someone or the mother of someone is an ancestor of that person.
2. An ancestor of someone's ancestor is also the ancestor of that person.
3. Jesse is the father of David.
4. David is the ancestor of Mary
5. Mary is the mother of Jesus

Prove: Jesse is an ancestor of Jesus.

28

Let us take another example we have an example here which says the father of someone or the mother of someone is an ancestor of that person. And ancestor of someone's ancestor is also the ancestor of that person. The third statement is saying Jesse is the father of David, the fourth statement is saying David is the ancestor of Mary and the fifth statement is saying Mary is the mother of Jesus now we have to prove that Jesse is an ancestor of Jesus.

(Refer Slide Time: 47:33)



Resolution Refutation Proofs

Predicates

A predicate is an assertion that some property or relationship holds for one or more arguments.

1. father(x,y) : `x' is the father of `y'.
2. mother(x,y) : `x' is the mother of `y'.
3. ancestor(x,y) : `x' is the ancestor of `y'.

Constants

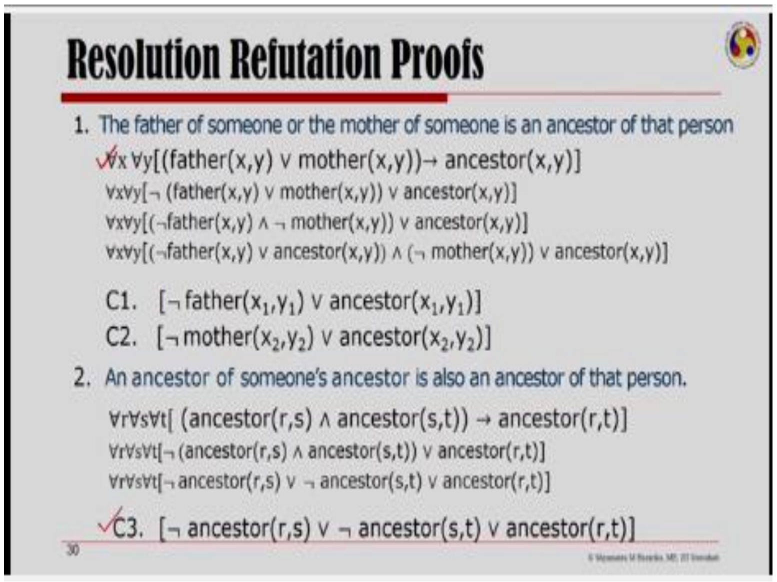
1. Jesse
2. David
3. Mary
4. Jesus

A constant is a symbolic name for a real-world person, object or event.

29

First we look at the predicates we have father (x, y), x is the father of y, Mother (x, y), x is the mother of y ancestor x is the ancestor of y and then we have to use a couple of constants. Now recall that a constant is a symbolic name for a real world person of object or event, so we have Constants here Jesse, David, Mary and Jesus.

(Refer Slide Time: 48:08)



Resolution Refutation Proofs

1. The father of someone or the mother of someone is an ancestor of that person

   ✓$\forall x \forall y[(\text{father}(x,y) \vee \text{mother}(x,y)) \rightarrow \text{ancestor}(x,y)]$

   $\forall x \forall y[\neg (\text{father}(x,y) \vee \text{mother}(x,y)) \vee \text{ancestor}(x,y)]$

   $\forall x \forall y[(\neg \text{father}(x,y) \wedge \neg \text{mother}(x,y)) \vee \text{ancestor}(x,y)]$

   $\forall x \forall y[(\neg \text{father}(x,y) \vee \text{ancestor}(x,y)) \wedge (\neg \text{mother}(x,y)) \vee \text{ancestor}(x,y)]$

   C1. $[\neg \text{father}(x_1,y_1) \vee \text{ancestor}(x_1,y_1)]$

   C2. $[\neg \text{mother}(x_2,y_2) \vee \text{ancestor}(x_2,y_2)]$

2. An ancestor of someone's ancestor is also an ancestor of that person.

   $\forall r \forall s \forall t[(\text{ancestor}(r,s) \wedge \text{ancestor}(s,t)) \rightarrow \text{ancestor}(r,t)]$

   $\forall r \forall s \forall t[\neg (\text{ancestor}(r,s) \wedge \text{ancestor}(s,t)) \vee \text{ancestor}(r,t)]$

   $\forall r \forall s \forall t[\neg \text{ancestor}(r,s) \vee \neg \text{ancestor}(s,t) \vee \text{ancestor}(r,t)]$

   ✓C3. $[\neg \text{ancestor}(r,s) \vee \neg \text{ancestor}(s,t) \vee \text{ancestor}(r,t)]$

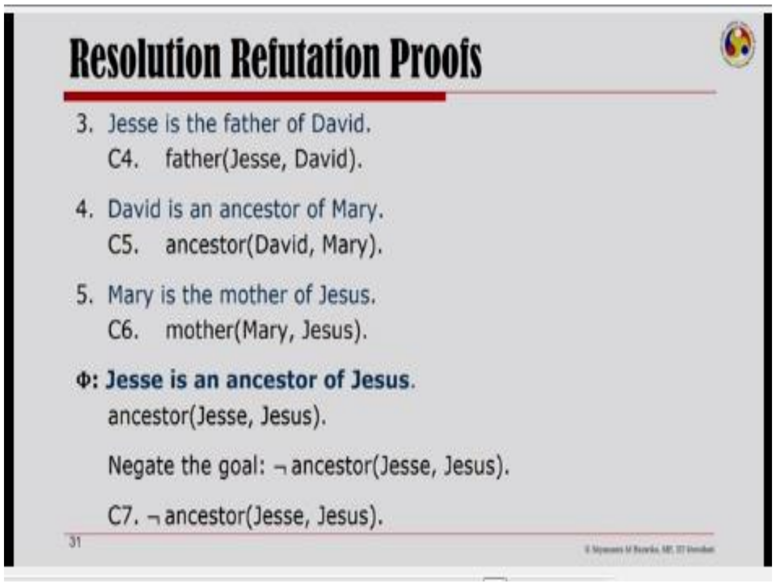
30

And let us look at getting to the clausal normal form for these statements the first statement the father of someone or the mother of someone is an ancestor of that person. Here is the statement which says for all xy, x is the father of y or x is the mother of y would imply that x is an ancestor of y. So in order to convert this to clausal normal form, you have to eliminate the implication, so we replace the implication by IP implies QS not P or Q.

And then you take this negation inside and distribute over the or, so you end up having an and not a father and not of mother and this is retained this side. Now you have an and and an or and all we have is father or ancestor and mother or ancestor. So that leads us to 2 clauses for this first statement, because here is an implication and I can break it up into the 2 following clauses number 1, not father x1, y1 or ancestor x1, y1 and the second not mother x2, y2 or ancestor x2, y2.

The second statement is here an ancestor of someone's ancestor is also an ancestor of that person. So we could write for all r for all s for all t, r of s ancestor r, s, an s is an ancestor of t. So we could write r is an ancestor of s, and s is an ancestor of t implies r is an ancestor of t. Now you replace the implication first then you have negation and an end you rewrite it using the DeMorgan's to not of P or not of Q and this portion is already not of ancestor r, t. So, we have one clause coming out of our second statement, and this is the clause.

(Refer Slide Time: 50:46)



Resolution Refutation Proofs

3. Jesse is the father of David.
   C4. father(Jesse, David).
4. David is an ancestor of Mary.
   C5. ancestor(David, Mary).
5. Mary is the mother of Jesus.
   C6. mother(Mary, Jesus).

Φ: Jesse is an ancestor of Jesus.
ancestor(Jesse, Jesus).

Negate the goal: $\neg$ ancestor(Jesse, Jesus).

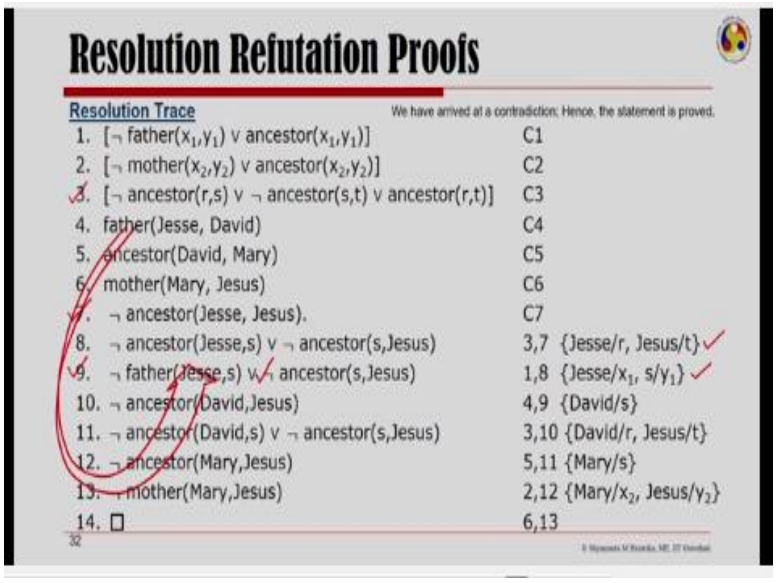
C7. $\neg$ ancestor(Jesse, Jesus).

31

Now the third statement is pretty simple, it is saying Jesse is the father of David, so we have a ground statement Father Jesse, David no variables in there. So it is already in clausal normal form we have David is an ancestor of Mary, so we have ancestor David, Mary already in clausal normal form Mary is the mother of Jesus. So mother Mary, Jesus and now the statement to be prove, Jesse is an ancestor of Jesus.

So here is the statement ancestor Jesse, Jesus and you negate this to get our seventh clause not ancestor Jesse, Jesus.

(Refer Slide Time: 51:38)



Resolution Refutation Proofs

Resolution Trace

We have arrived at a contradiction; Hence, the statement is proved.

1.	[$\neg$ father(x_1,y_1) v ancestor(x_1,y_1)]	C1
2.	[$\neg$ mother(x_2,y_2) v ancestor(x_2,y_2)]	C2
✓3.	[$\neg$ ancestor(r,s) v $\neg$ ancestor(s,t) v ancestor(r,t)]	C3
4.	father(Jesse, David)	C4
5.	ancestor(David, Mary)	C5
6.	mother(Mary, Jesus)	C6
✓7.	$\neg$ ancestor(Jesse, Jesus).	C7
8.	$\neg$ ancestor(Jesse,s) v $\neg$ ancestor(s,Jesus)	3,7 {Jesse/r, Jesus/t} ✓
✓9.	$\neg$ father(Jesse,s) v $\neg$ ancestor(s,Jesus)	1,8 {Jesse/x_1, s/y_1} ✓
10.	$\neg$ ancestor(David,Jesus)	4,9 {David/s}
11.	$\neg$ ancestor(David,s) v $\neg$ ancestor(s,Jesus)	3,10 {David/r, Jesus/t}
12.	$\neg$ ancestor(Mary,Jesus)	5,11 {Mary/s}
13.	$\neg$ mother(Mary,Jesus)	2,12 {Mary/x_2, Jesus/y_2}
14.	□	6,13

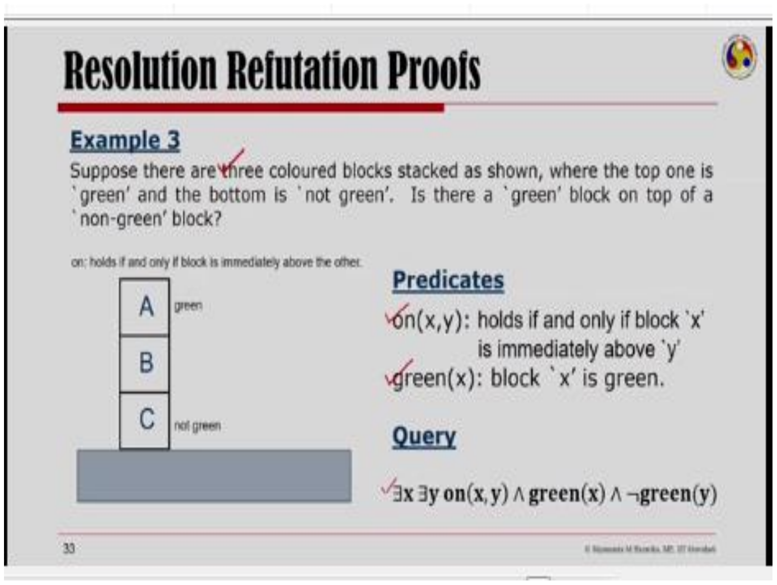
32

So this is our initial set of clauses delta and we now look for the empty clause, so I take clause 3 and clause 7 and resolve them. But now kindly note a new introduction of the substitution that is being used written in front of the annotation. So I have Jesse for r and Jesus for t in clause 3 and clause 7, so that leads to having Jesse, s here, from this first literal and s, Jesus from here and this and this are the complimentary literal pairs that will go away.

The next is about taking 1 and 8, so in 1 and 8 you could see that we are talking of a substitution which is Jessie for x1 and s for y1. So that substitution will give us the number 9 clause we then have 4 and 9 resolve and here once we resolve 4 and 9 this father and father Jesse, David these are the complimentary pair. And all I it will be left out with is not of ancestor David, Jesus next I have 3 and 10 to generate my 11 clause in the resolution trace.

I go on forward like that and I finally arrived at the empty clause, so arriving at the empty clause means that the statement that I wanted to prove is actually derived from the initial database or initial set of clauses delta.

(Refer Slide Time: 54:05)



Resolution Refutation Proofs

Example 3

Suppose there are three coloured blocks stacked as shown, where the top one is 'green' and the bottom is 'not green'. Is there a 'green' block on top of a 'non-green' block?

on: holds if and only if block is immediately above the other.

Predicates

on(x,y): holds if and only if block 'x' is immediately above 'y'

green(x): block 'x' is green.

Query

$\exists x \exists y \; on(x,y) \wedge green(x) \wedge \neg green(y)$

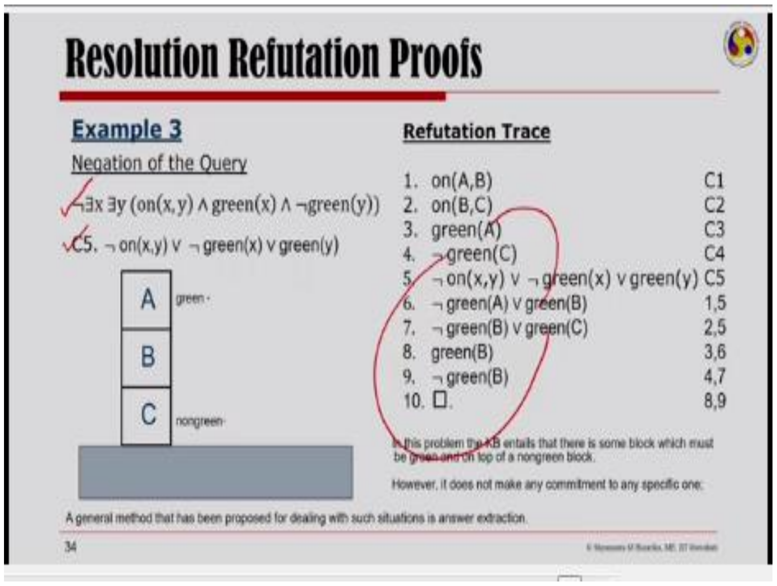
33

Now to finally wind up this lecture today let us quickly look at an example from the blocks world problem that we have been discussing. So here is a slightly different version of the blocks world example, suppose there are 3 coloured blocks now and they are stacked as shown, where the top one is green and the bottom is not green. Now the question is, is there a green block on top of a non green block.

Now intuitively you can very well see that the answer to this question is yes, because if B is green then I have the statement being true because green is on top of non green. Now on the other hand, if B is non green then also the statement is true because I have A which is a green block on top of the non green block, how do you look at this in first order predicate calculus. In order to do this I introduce 2 predicates on x, y which holds if and only if block x is immediately above block y and green x which is saying that block x is green.

And is there a green block on top of a non green block reduces to a query there exist an x there exists an y on x, y and green x and not green y.

(Refer Slide Time: 55:57)



Resolution Refutation Proofs

Example 3

Negation of the Query

$\neg\exists x\, \exists y\, (on(x,y) \wedge green(x) \wedge \neg green(y))$

C5. $\neg on(x,y) \vee \neg green(x) \vee green(y)$

A
green
B
C
nongreen

Refutation Trace

1.	$on(A,B)$	C1
2.	$on(B,C)$	C2
3.	$green(A)$	C3
4.	$\neg green(C)$	C4
5.	$\neg on(x,y) \vee \neg green(x) \vee green(y)$	C5
6.	$\neg green(A) \vee green(B)$	1,5
7.	$\neg green(B) \vee green(C)$	2,5
8.	$green(B)$	3,6
9.	$\neg green(B)$	4,7
10.	$\square$.	8,9

In this problem the KB entails that there is some block which must be green and on top of a nongreen block.

However, it does not make any commitment to any specific one.

A general method that has been proposed for dealing with such situations is answer extraction.

34

So these clauses are already in clausal normal form except the query, so I convert the query to a clausal normal form. This would require that I first convert this as for all x, for all y not of the statement and take it inside. So I generate the clause for the negation of the query and add it up to my set of clauses, so all I am told here is that A is on B and second B is on C and I am told that A is green and I am told that C is not green.

So these are clauses C1, C2, C3 and C4 and I have the fifth clause which is negation of the query. Now from 1 and 5, I could derive that x can be replaced by an A and y could be replaced by a B to give me not green A or green B and from 2 and 5. I could derive a statement not green B or green C for when I am doing 2 and 5 I would replace x by B and Y by C. So I could have this statement then I resolve this statement 7 and 6 with 3 and 4 to give me 2 statements.

So I can take 6 and resolve it 3 that will give me green B and I can take 4 and resolve it 7 that will give me not green B and that could lead to a contradiction. Now when I have a contradiction in this problem, the knowledge base entails that there is some block which must be green and on top of a non green block. However it does not make any commitment to any specific one, specifically the whole resolution that I have perform does entail this existential statement.

But in no way entails what is x or what is y, so a general method for dealing with such scenarios has been propose and this method is called the process of answer extraction. We will look at answer extraction in our next lecture, thank you.