Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology Delhi

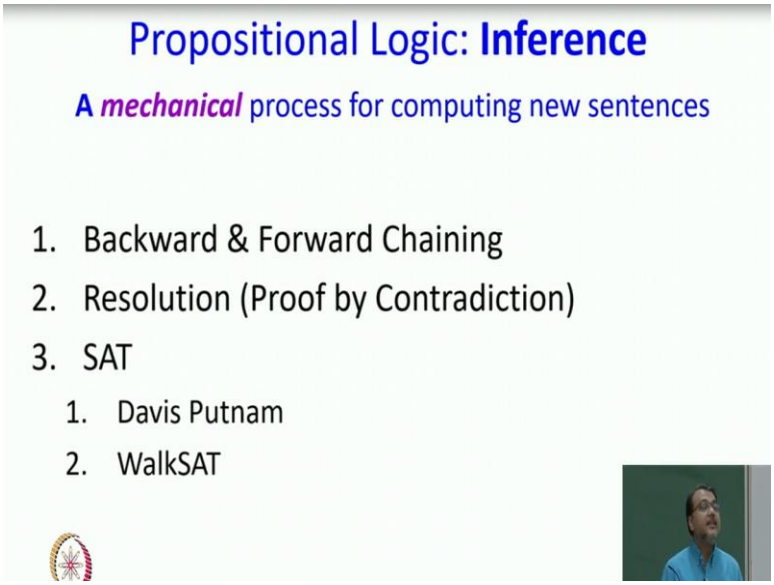
Lecture-46
Logic in Artificial Intelligence: Resolution, part- 5

Let us get started. So, we have been talking about the importance of logic in the field of artificial intelligence. And we have said that logic is one of the most important ways in which you communicate with an AI agent. You express what you know about the world in a logical representation. You therefore, enable the machine to make logical inferences. And you can then query the machine for new inferences or properties of the domain, which it might be able to use the current knowledge to respond to.

In that context, we talked about you know, what is the logical representation? What is the syntax what is the semantics. And we then started talking about inference algorithms and we did one inference algorithm which is forward chaining. And forward chaining is this procedure where it typically uses horn clauses in the system and it uses modus ponens rules to make inferences. So, whenever the rule says that you know P 1 and P 2 and P 3 to P n implies q and we know that P 1 is true, we know that P 2 is true.

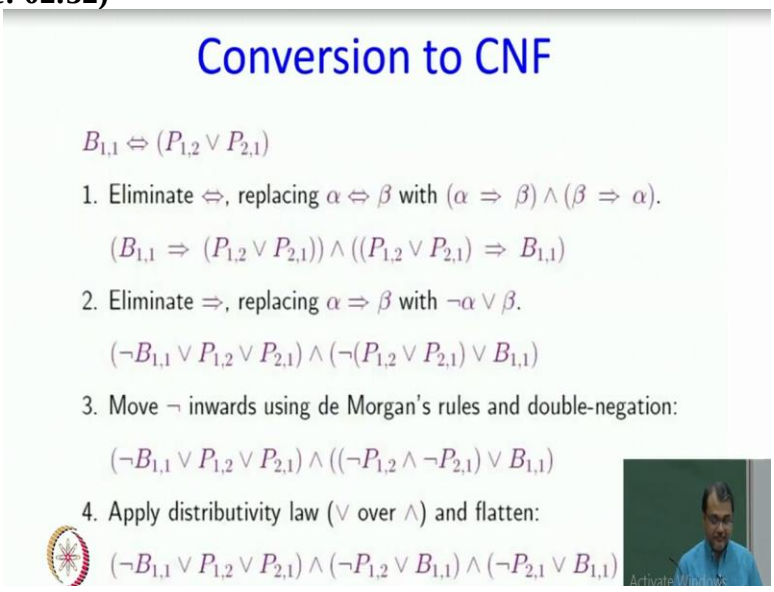
We know that P n is true, then we can prove that q is true. And we said that, is it a sound algorithm? Yes. Is it a complete algorithm? No. But if the only kinds of rules in our knowledge base are horn clauses, and the query is a single literal or a conjunction of literals, then it is complete. This is very well.

(Refer Slide Time: 01:54)



And then we very briefly mentioned that there is also something called backward chaining in which you sort of start from what you want to prove and then try to create more support for it by saying, I have to prove this what should be true in my domain and then I have to prove this what is should be true in my domain, I keep doing this. And eventually I get to what was originally true in the knowledge base that would be search in the backward direction. So, today our goal is to talk about resolution proof systems. This is equivalent to proof by contradiction. And for that, we will first talk about converting everything into CNF.

(Refer Slide Time: 02:32)



Now, I want to mention that the reason you convert everything into CNF is because it is a canonical representation, any logical formula can be converted into a CNF formula and so, we

work on CNF formulas because we assume that if you have a different kind of formula you will first do a basic conversion, give me a CNF formula and then I will start working on it. So you always have to start with some representation for an inference procedures and so they decided that they will use the representation of CNF.

They could have used DNF, they could have used other representations, but they said they will use CNF. So first so very briefly, how do we convert any formula, let us say my formula is B 1 1 bidirectionally, equivalent to P 1 2, or P 2 1. So now what we do is we basically start applying rules in this order first if it is a bidirectional implication, we convert it into a unidirectional implications so A if and only if B can be equivalently said as if A than B and if B than A.

So B 1 1 bidirectional equivalent to this is equal to saying B 1 1 implies this and this implies B 1 1. So that is the first step that we use, we will eliminate bidirectional implication and convert it into 2 unidirectional implications, and a conjunction of those. So now you get B 1 1 implies P 1 2 or P 2 1. And P 1 2 or P 2 1 implies B 1 1 the next step is to convert implication into a disjunction. Every alpha implies beta is equivalent to not alpha or beta.

So we apply that rule. And so we get not B 1 or the disjunction, so that becomes like a longer disjunction. And then we have this conjunction here and it becomes not of this antecedent or B 1 1. Now, at every step we asked the question, is it in CNF form? And CNF form is a conjunction of disjunctions is it in CNF form right now is the first part a clause. Remember what was a clause? A clause is a disjunction.

So, is the first part of clause yes is the second part of clause only disjunctions are allowed, but there is a not in a in a parenthesis. So therefore, it is not a clause. So, whenever I have not A or B I can bring the not inside using which rule De Morgan's. So, not alpha or beta becomes not alpha and not beta. So, that is what happens. You get not P 1 2 and not P 2 1 and you do not have to touch the first part where there is already a clause.

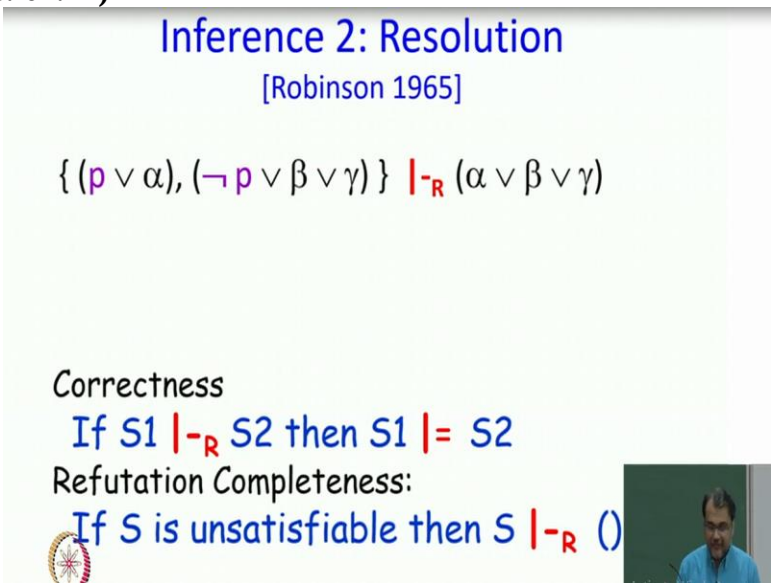
Now, is it a clause on the right? It is still not a clause because we have a conjunction in the parentheses and then a disjunction outside. We only need conjunctions outside and all

disjunctions inside. So when this happens, you apply distributive law and so in this distributive law what would happen is that you will make one conjunction which is not P 1 2 or B 1 1 and you will make one conjunction which is not P 2 1 or B 1 1.

And now you have got a conjunction of disjunctions, you got it in the CNF form. It is the conjunction of clauses. So, you are done any questions on this? So, always start removing the syntactic sugars, the bio directional and the unidirectional. Once you do that, everything has only not ended or then you apply the De Morgan's and the distributive law appropriately to finally get to a conjunction of disjunctions a CNF.

So now for the rest of this lecture slides, we are going to assume that anything that is given to us is in the CNF form. We will not worry about anything else because we know that anything can be converted to CNF. So now let us talk about resolution. Now, the basic tenet of resolution is this rule.

(Refer Slide Time: 07:12)



If I have a clause pure alpha, and another clause not pure beta, or in this case not pure beta gamma, then we can prove that alpha or beta or gamma. Let us think about this. We know that either P will be true or P will be false we may not know its value, but if P is true, what has to be true beta and gamma has to be true because we have a clause that says not P or beta and gamma not P is false.

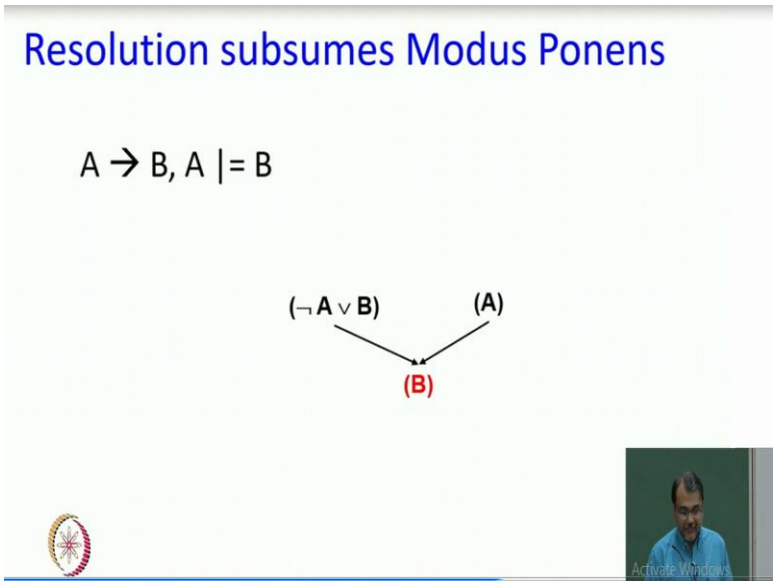
So, therefore, beta and gamma have to be true. Similarly, if P is false alpha has to be true it is like saying if this happens, then you have to do A if this does not happen, you have to B. So, that first you know that you either have to do A or you have to do B this is what this rule is saying. So, this is the basic resolution rule. And now we have to ask the question is it sound as a rule like modus ponens was sound as a rule is this sound as a rule? Yes.

So, whenever the resolution says S1 proves S2 then you know that S1 entails S2 in the world what is even more beautiful is that it has one kind of completeness, it is not complete. In fact, there are theorems which say that girdles incompleteness theorem, you know about that, which basically says that you cannot have a proof system which is sufficiently expressive and also complete. So, you cannot build something which will prove everything, if the problem is any interesting.

So actually, when Girdle proves this, everybody was like, wow, we cannot do anything, this is so sad because we can never create a complete proof system. But you know, over time AI, researchers realized that all these incompleteness business good for theory, we will just make systems that work and then they will make different kinds of proof systems. So if you have that kind of problem users, and if you have another kind of use that it needs to work in real world, it does not have to solve every problem in the like universe.

So anyway, so this is called a refutation, completeness refutation completeness means that I may not be able to prove everything. But if it is unsatisfiable that means that if it is not satisfiable at all, if there is no model for the formula, if there is no word which is consistent with this set of assertions, then it will be able to prove false. And how would it be able to prove false by creating an empty clause? Remember, we talked about the fact that empty clauses are equivalent to false and you will see it very soon in the as an example.

(Refer Slide Time: 10:25)

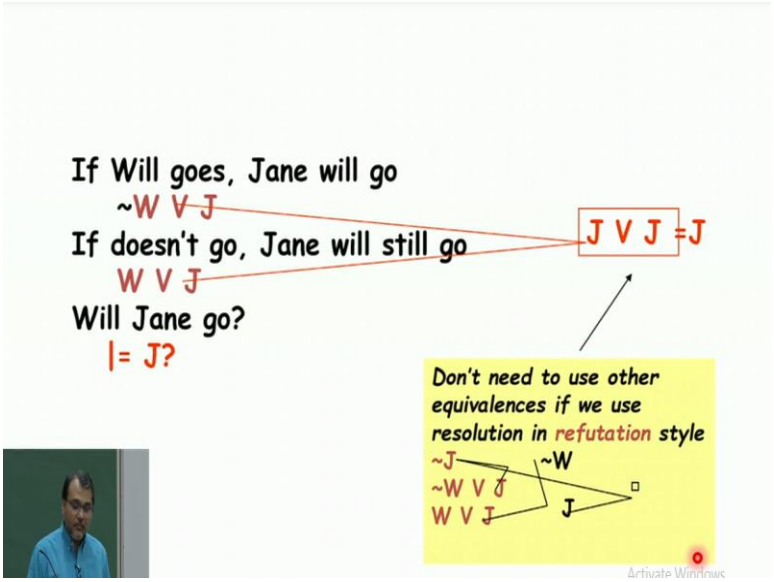


so, very first point that I want to make is that resolution already knows about Modus Ponens anything that Modus Ponens can do resolution can also do and to show that I have to show you that if A implies B, then A entails B. So, if I know that A implies B and I know A then I will entail B. So, let us write this out. So, we will first create clauses. So, if A implies B becomes not A or B, A is A and then let us apply resolution here what would resolution say what is a P here.

In the previous there was a P in the previous slide, what is the equivalent of P here A we are looking for that thing which when is true you do something and when is false you do something. So, A is that thing is A somewhere in the clause and there is not A somewhere in the clause and when you resolve them then you add everything else that was the disjunction and create a new clause for it. So, in this case, we said pure alpha not pure beta gamma.

So, the solution leads to alpha beta gamma so, similarly in this case not A or B. A that leads to B get rid of A and not A and we are left with B which is exactly what Modus Ponens says so, the first thing is that resolution is stronger than Modus Ponens. It knows more. So, you can think of it as simple pushing you are just pushing symbols down, by creating new formula. And the rule is very simple. Now, I also want to show you with a very simple example, that it cannot prove a basic.

(Refer Slide Time: 12:16)



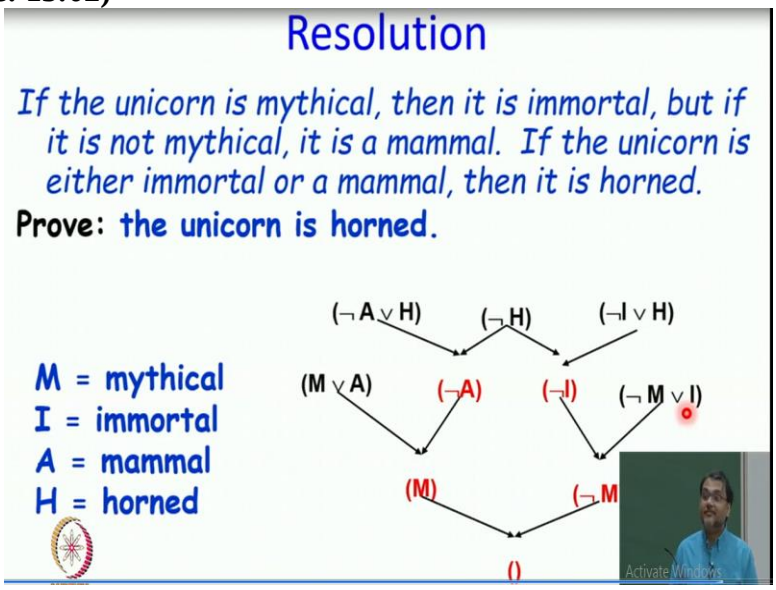
And here is an example. It is a terrible example, but it makes the point. If will goes Jane goes, if will does not go Jane goes, will Jane go? Answer? Yes, should resolution be able to prove this? You think so. And I will tell you why it is not able to. If you resolve these 2 you get J or J. Do you get J or J.? Now, does it no J or J is J it does not know J or J it is so stupid that it absolutely has no idea that J or J is J just by applying resolution, you cannot prove the simple thing. But that is okay.

Do not worry about it, because it is a refutation complete. What does that mean? That is something is unsatisfiable you will definitely build to prove false. So, how do we prove these facts? Now, what we do is we use proof by contradiction. So, we said this is what we know what we know is not W or J and W or J and what do we want to prove J? Let us suppose J is false, this is what we do in proof by contradiction, let us suppose what do you want to prove is false.

So, we assume not J and then try to come to a contradiction in this case or false in this case, an empty clause. And here is a simple way in which you do it. You resolve not J and not W or J. You get not W we mean there are many ways to do this. And you resolve not W and W or J you get J and then you resolve not J and J and you get empty clause. If this example is not clear, do not worry, I will show you another example and we will do it slowly.

But here noticed that J and not J resolved to an empty clause if empty clause was true, which is what we assume last time, this will go completely mad. When J is true and not J is true that means, this formula is not satisfiable cannot be true. So, when they get resolved and you get nothing that nothing should mean false. And again you go back to how we discuss what the 0 of the disjunction operation is. So, we are going to work on this particular example.

(Refer Slide Time: 15:02)



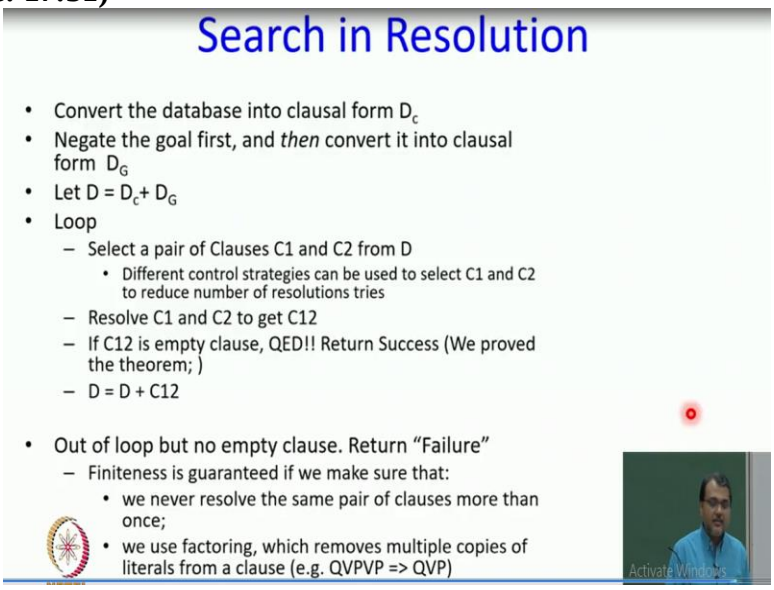
If the unicorn is mythical, then it is immortal if the unicorn is not mythical, it is a mammal. If the unicorn is either immortal or a mammal, then it is horned to that unicorn is horned. And I am sure I mean you can go back to your Boolean logic and prove it one or the other that will prove it using resolution. So, we will define symbols so let us say our diff symbols are M for mythical I for immortal A for mammal and H for horned. And then we define formula or clauses.

So for the first statement, what is a clause M implies I and we will write it as a clause not M or I. For the second, it would be M or A, and so. So these are my clauses for the input. Now, we will always prove it by contradiction. So what do we assume? Not H we assumed not H and now our goal is to apply the solution repeatedly to get to empty clause and there are many paths so we will just look at one possible path. So give me one resolution that you can do.

For example, let us start with not H and not H we get not A similarly not H and not I or H we get not I so we get these 2 by resolving. Now let us keep doing this. We resolve M or A and not A we get M. We have not I or not M or I, we get not M. We have M and not M, we get emptiness.

So at least in this simple example, we were able to prove empty clause, and therefore, this is falsified. It is a contradiction, and therefore, the unicorn is horned. Now we know. Everybody with me any questions on resolution? Now, I am not going into the detail of how do you actually implement a resolution very much. of course, you do search.

(Refer Slide Time: 17:31)



There is just no way around it. So you have a database of clauses. And you negate the goal and convert it into clausal form. And you add that to the database, and now you have a full database of clauses and the negated goal. And then you loop you basically select which 2 clauses you need to use in resolution on and generate a new thing and you put it in the database and you keep doing this. And in order to figure out you know, which pair of clauses you put in there. Their usage control strategies here it says some branching heuristic.

And so at some point when you add empty clause, you have got success you have proved it. And then if you are out of loop but you do not get an empty clause, you cannot get something new but you do not even get empty clause you return failure saying that I cannot prove this. And if we have to make sure that we never resolve the pair of clauses more than once and if multiple copies of something happened like Q or P or P, we just converted into Q or p. So we add that knowledge also in the system. So this is called resolution.