

Artificial Intelligence:
The Resolution Refutation method of First Order Logic
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module - 07
Lecture - 02

Okay so we have started looking at the resolution method of first order logic. And to do that we were looking at how to convert a formula into clause form. Lets look at an example and lets say this is the formula given to us.

Lets say you want to show that this formula is unsatisfiable. Lets not worry about how we got it. But to show that it is unsatisfiable we want to first convert it into clause form. and use resolution method to show whether we can derive a null clause or not. So if we remember the steps the first thing that we do is we find that there is a free variable here y . so we add a quantifier here, there exists a y . that's the first step so we now have no free variable. And we need to now convert this into clause form.

so the first step is to remove the connectives. So we will get there exists y for all z , now this equivalence we will break it up into two parts which is bi implication, both ways. So one side we will say that $P z y$ implies.. and the other is the other way round which is .So we handled this one now. Next we handle this and this. So we get. That's the first part. The second part. Because we are removing the not this will go away. And appropriate number of brackets.

Then we push the NOT inside, that's only left hand side so let me rewrite. So when I push the NOT inside, I will do both the things together, I will push the quantifier into that AND so that AND will become OR. And that quantifier will become a universal quantifier. And the right hand side remains same so I will not copy it here.


(Refer Slide Time: 5:44)

Clause form - an example. $\exists y \forall z [P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z))]$

$\exists y \forall z [(P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z))) \wedge (\neg \exists x (P(z,x) \wedge P(x,z)) \Leftrightarrow P(z,y))]$

$\exists y \forall z [\neg P(z,y) \vee \neg \exists x (P(z,x) \wedge P(x,z)) \wedge \exists x (P(z,x) \wedge P(x,z)) \vee P(z,y)]$

$\exists y \forall z [\neg P(z,y) \vee \forall x (\neg P(z,x) \vee \neg P(x,z))] \wedge$



Then we need to eliminate the existential quantifier, there exists y and there exists x. in the first case where there exists y, is not in the scope of any universal quantifier so we will be left only with the universal quantifier, for all z. and for this formula where there was y, we will replace it with some skolem function. Lets call it sk1, skolem constant. And the remaining part of this formula will remain the same. Whereas on the and side, this existential variable x is in the scope of universal variable z essentially so it must become a function of z. it will become P z, so lets call this skolem function also sk function, sk2 z and P sk2 of z , x and whole thing P z sk1. So we have eliminated the existential quantifier. We can bring the universal quantifier out here and so we don't have any quantifier inside.

And inside we have only AND OR and NOT. So all we need to do is distribute next. So this for all x will come here so once we distribute we will get for all x for all z. now the first of this conjunct is already in clause form, you don't really need to do much. Let me just write it again. In second part you need to distribute the AND over OR. So we get P z sk2 of x OR P z sk1 so this becomes another clause. P sk2 of x comma z OR P z sk1 which is third clause. Now we are left with three clauses. Now we can throw away the quantifier symbols and we can rename the variables in each of the three clauses. So which are variables which are common, its z so lets say this is clause1, this is clause 2 and clause 3 so we can rewrite the three clauses as a set. So the first clause is this one. Not P z, lets call it z1 sk1 comma not P z1 x1 comma not P x1 z1. We just call them with the subscript 1. Or suffix 1 to say that it's the first clause.

(Refer Slide Time: 10:32)

Clause form - an example. $\exists y \forall z [P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z))]$

$\exists y \forall z [(P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z))) \wedge (\neg \exists x (P(z,x) \wedge P(x,z)) \Leftrightarrow P(z,y))]$

$\exists y \forall z [\neg P(z,y) \vee \neg \exists x (P(z,x) \wedge P(x,z)) \wedge \exists x (P(z,x) \wedge P(x,z)) \vee P(z,y)]$


$\exists y \forall z [\neg P(z,y) \vee \forall x (\neg P(z,x) \vee \neg P(x,z))] \wedge$

$\forall x \forall z [\neg P(z,sk1) \vee \dots] \wedge (P(z,sk2(z)) \wedge P(sk2(z),z) \vee P(z,sk1))$

$\forall x \forall z [\neg P(z,sk1) \vee \neg P(z,x) \vee \neg P(x,z)] \wedge (P(z,sk2(z)) \vee P(z,sk1)) \wedge (P(sk2(z),z) \vee P(z,sk1))$

1. $\neg P(z1,sk1), \neg P(z1,x1), \neg P(z1,z1)$

2. P.



Likewise the second clause would be $P(z2)$ let's call it. And $sk2$ of $z2$ comma P of $z2$ comma $sk1$. The third clause in the similar fashion is P of $sk2$ the function will remain the same but argument has become different $z3$ comma $z3$. $P(z3)$ comma $sk1$.

(Refer Slide Time: 11:14)

Clause form - an example. $\exists y \forall z [P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z))]$

$\exists y \forall z [(P(z,y) \Leftrightarrow \neg \exists x (P(z,x) \wedge P(x,z))) \wedge (\neg \exists x (P(z,x) \wedge P(x,z)) \Leftrightarrow P(z,y))]$

$\exists y \forall z [\neg P(z,y) \vee \neg \exists x (P(z,x) \wedge P(x,z)) \wedge \exists x (P(z,x) \wedge P(x,z)) \vee P(z,y)]$

$\exists y \forall z [\neg P(z,y) \vee \forall x (\neg P(z,x) \vee \neg P(x,z))] \wedge$


$\forall x \forall z [\neg P(z,sk1) \vee \dots] \wedge (P(z,sk2(z)) \wedge P(sk2(z),z)) \vee P(z,sk1)$

$\forall x \forall z [\neg P(z,sk1) \vee \neg P(z,x) \vee \neg P(x,z)] \wedge (P(z,sk2(z)) \vee P(z,sk1)) \wedge (P(sk2(z),z) \vee P(z,sk1))$

1. $\neg P(z1,sk1), \neg P(z1,x1), \neg P(z1,z1)$

2. $P(z2,sk2(z2)), P(z2,sk1)$

3. $P(sk2(z3),z3), P(z3,sk1)$



So we started with this formula which is in first order logic and we came up with a set of three clauses which use universally quantified variables $z1$ $x1$ $y2$ and $y3$. So this illustrates how one can convert it into clause form.

Now lets go back to the resolution rule. The resolution rule says as follows. If you have two clauses, $C1$ is made up of some formulae, lets call it a set S UNION set lets call it P . so by this I mean that let me rewrite it. And by this I mean a set of formula using predicate P . so this is just informal, semiformal way of trying to describe it.

And we have another clause $C2$ which is $R1$ $R2$ and a set of formulas of the type not P in the similar fashion. And from this we will derive the set $R1$ $R2$ UNION $S1$ Sn if all Ps can be unified with θ . And then supply this θ .

(Refer Slide Time: 15:05)

Resolution Rule


$$C_1 = \{S_1, \dots, S_m\} \cup \{P\}$$

a set of formulas may produce P

$$C_2 = \{R_1, \dots, R_n\} \cup \{\neg P\}$$

$$\{R_1, \dots, R_n\} \cup \{S_1, \dots, S_m\}$$

if all P's can be unified with Q



So it is similar to what we did in propositional logic that we will remove certain instances of a formula and remaining we will keep essentially. So let me take an example of this from the last thing we just showed. So we derived these three clauses so let me look at one of them. So NOT P z1 sk1 NOT P z1 x1. its the same I am just rewriting it. NOT P x1 z1. That was clause 1. And the second clause was P z2 sk2 of z2, P z2 y2. So from this what can we do. Supposing we were to take the following substitution. X1 becomes equal to sk1, that's allowed because it's a constant, then z1 is also sk1. So our goal is that because this z1, this x1 and this sk1 we want to unify they will all become the same. So in the first part the three parts will become identical.

(Refer Slide Time: 17:34)

Resolution Rule

$$C_1 = \{S_1, \dots, S_m\} \vee \{P\}$$

a set of formulas may produce P


$$C_2 = \{R_1, \dots, R_n\} \vee \{\neg P\}$$

$$(\{R_1, \dots, R_n\} \vee \{S_1, \dots, S_m\}) \text{ if all } P_s \text{ can be unified with } Q$$

Ex: $\neg P(z_1, \underline{sk_1}), \neg P(\underline{z_1}, z_1), \neg P(z_1, \underline{z_1})$

$P(\underline{z_2}, sk_2(z_2)), P(\underline{z_2}, z_2)$

$z_1 = sk_1$
 $z_2 = sk_1$



So they will all become NOT P, sk1, sk1. So if you look at this z1 will get replaced by sk1, this z1 also so on and so forth. So the goal is to convert all to that. And the second clause supposing we choose z2 is also sk1. Then all these underlined things will become sk1. And this set of three clauses will cancel this one clause, one literal from the second clause. And what we will be left is only this second one clause which will say P sk1 because we have substituted that sk2 z2 sk1. So this comes from here and this whole thing comes from here. So its similar to what we did in propositional logic that we cancel positive and negative clauses. But for the sake of generality we have to state this rule in a more general form. in a sense that in propositional resolution we will cancel only one positive literal and one negative literal from the two clauses. In the case of first order logic as this example shows from the first clause all three vanished, from the second clause one of the two vanished and we are left with only one.

In general it could apply to whole set like this. So lets see that the resolution method is a generalization of both forward chaining and backward chaining. So lets take our favorite example which is this Socratic example. He said that Man Socrates and implicit quantifier from Man X implies Mortal X. show that Mortal Socrates. That's the example we started out with. This example with forward chaining how does it work? That you have essentially match left hand side. So you have Man X implies Mortal X. and then you have Man Socrates. So you unify this and produce and infer Mortal Socrates.

This was in forward chaining. In backward chaining we said from Goal Mortal Socrates and Rule: Man X implies Mortal X. you infer the goal Man Socrates. And this matches the fact.

(Refer Slide Time: 22:44)

The image shows a whiteboard with handwritten notes in red ink. The title is "Resolution Method".

FC (Forward Chaining):

- Match LHS
- $Man(?x) \supset Mortal(x)$
- $Man(Socrates)$
- UNIFY
- infer $Mortal(Socrates)$

BC (Backward Chaining):

- Goal: $Mortal(Socrates)$
- Rule: $Man(?x) \supset Mortal(?x)$
- Goal: $Man(Socrates)$
- matches fact.

On the right side of the whiteboard, there is a resolution rule written in blue ink:

$$\frac{Man(Socrates) \quad Man(?x) \supset Mortal(?x)}{Mortal(Socrates)}$$

The NPTEL logo is visible in the bottom left corner of the whiteboard.

Now let's just rewrite this whole thing in clause form. So we have Man Socrates and other clause we will have it NOT Man X or Mortal X. This is this clause created in the clause form. Now you can see that for example forward chaining is a straight forward instance here of resolution. You can resolve this and you will get Mortal Socrates. So forward chaining is just one particular way of doing resolution. What I have applied on the right hand side in blue is the resolution rule and I have come up with something which is equivalent to this. So this is equivalent to this. So I have not done refutation here. I have just shown that you can derive Mortal Socrates. But in refutation you add this the negation of a goal. You add this to a set of clauses that you already have which is so let me just take the rule part here. Man X or Mortal X. again I can apply resolution to this and I will get NOT Man X. so if you look at this part and this part you can see that something very identical is happening, only if you think of a goal as negation. If in the left hand side instead of goal which is a kind of marker for saying that it's a goal I replace it with a negation sign, which is what I have done. I replaced it with this negation sign here and what I get is a new formula which has a negation sign which I can always interpret as a goal. So you can see by adding the goal with a negation sign I am essentially doing what backward chaining was trying to do.

(Refer Slide Time: 26:05)

Resolution Method

FC

Match LHS

$$\frac{\text{Man}(?x) \supset \text{Mortal}(?x)}{\text{Man}(\text{Socrates})} \text{ UNIFY } \text{Mortal}(\text{Socrates})$$

BC

Goal: $\text{Mortal}(\text{Socrates})$

Rule: $\text{Man}(?x) \supset \text{Mortal}(?x)$

Goal: $\text{Man}(\text{Socrates})$ → matches fact.

$\text{Man}(\text{Socrates})$
 $\text{Man}(?x) \supset \text{Mortal}(?x)$
 $\text{Mortal}(\text{Socrates})$

=

$\text{Man}(\text{Socrates}) \neg \text{Man}(?x) \vee \text{Mortal}(?x)$
 $\swarrow \searrow$
 $\text{Mortal}(\text{Socrates})$

Resolution

Add

$\neg \text{Man}(?x) \vee \text{Mortal}(?x)$

$\swarrow \searrow$

$\neg \text{Mortal}(\text{Socrates})$

$\neg \text{Man}(?x)$

The same inference I am making that from show that Socrates is Mortal and the rule that all men are mortal, I reduced it to a goal called show that Socrates is a Man essentially. Somebody should have pointed this out, this is Socrates. The subgoal that I have, I have to apply the substitution, so I have reduced it to this goal. So this negation is little bit like show saying its like goal. And to this if I add the real fact which is Man Socrates then I said here that this matches a fact but you can see that it amounts to saying that I can take that fact and this goal, think of Not Man Socrates as a goal. We match it and what you really get is a null clause. So this matching fact is kind of equivalent to this.

So I have three clauses. I have Socrates is a man as a clause. I have All men are mortals as a clause. And then taking the negation of the goal which is that Not Mortal Socrates, I get a third clause. If I as you can see on the top, if I resolve Man Socrates with the rule then its like doing forward chaining. If I resolve Not Mortal Socrates with the same rule its like doing backward chaining. The backward chaining you are doing the same thing, you can chaining with the same rule. You are going from goal to subgoal or you are going from fact to new fact essentially. But we can see if we think about it a little bit that when you are doing resolution you are subsuming both activities and it really depends upon what is the two clauses you pick to resolve. If you pick first two its like forward chaining if you pick the last two its like backward chaining.

For the sake of completeness and we will come to this in the next class, it was shown by Alan Robinson who invented this method in 1985 that the method is

complete if you want to show that some formula is unsatisfiable. We have seen example we just saw those two examples of green blocks and we said we cant do it with forward chaining or backward chaining and then we atleast one of the examples we showed that we can do it with resolution method. In the second example hopefully you will do as an exercise and show that it can be done with resolution method. So we saw that something we could not do with forward chaining or backward chaining we could do with resolution. And in fact Robinson formally proved that resolution method is complete whenever you want to derive the null clause. It means that if you start with a set of formulae or if you start with a formula which is unsatisfiable then you can derive them or you can always derive the null clause. So in this example it shows that if you have these three clauses Socrates is a man and All men are mortals and Socrates is not mortal which is a third clause that we add which is a negation of the goal then these three clauses are unsatisfiable.

(Refer Slide Time: 30:11)

Resolution Method - Robinson

FC
 Match LHS
 $Man(?x) \supset Mortal(x)$
 $Man(Socrates)$
 UNIFY
 $Man(Socrates)$

BC
 Goal: $Mortal(Socrates)$
 Rule: $Man(?x) \supset Mortal(?x)$
 Goal: $Man(Socrates)$
 matches fact.

Resolution
 $Man(Socrates)$
 $Man(?x) \supset Mortal(?x)$
 $Mortal(Socrates)$
 $\neg Man(?x) \vee Mortal(?x)$
 $\neg Mortal(Socrates)$
 $\neg Man(Socrates)$
 VN

And we have gone through this argument, why do we call them unsatisfiable? Because we can derive the null clause essentially. Okay so we have seen an introduction to the resolution method. In the next class we will try to look at some of the intricacies of resolution method and we will talk about complexity a little bit and try to see how can we make things more efficient.

