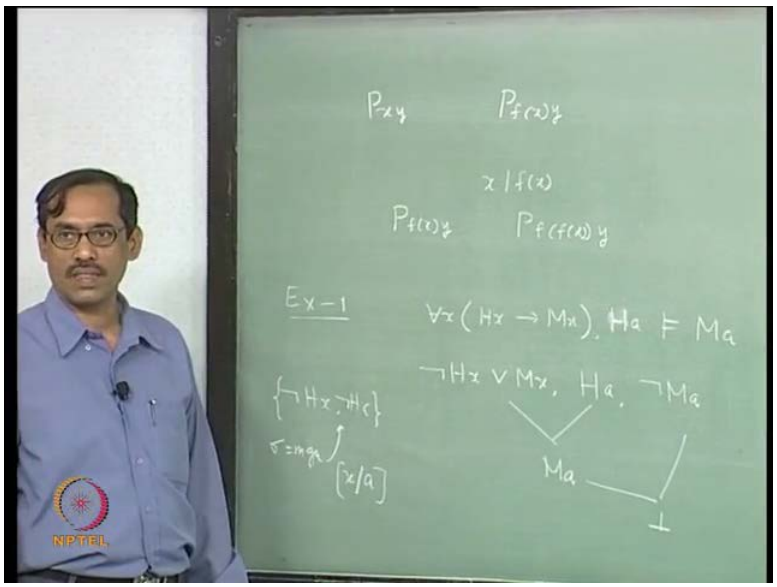
Mathematical Logic
Prof. Arindama Singh
Department of Mathematics
Indian Institute of Technology, Madras

Lecture - 37
Resolution Rules

If some literals can be unified, the same algorithm should be able to unify the terms also. Can you see that or not? See, you have a clause, which is a set of literals. Now, let us take only two literals. There, you want to unify those two; so you start matching from the left to right. The same way suppose there are two terms, say, f of something, g of something, and so on. Now, if their first symbols match then only you can unify. So, after they match you start finding out the same way. You can also see the terms can be unified. Anyway, they are just strings. You are unifying the strings by choosing one variable to be substituted by a term, but there is one danger in finding out that one variable is x, the other variable is a term where x occurs. Suppose mismatch occurs at these two points. Then how will you unify it?

(Refer Slide Time: 01:18)



Let us see an example. Suppose, you have P; one literal is P x y, another literal is P f of x y. Now, when the unification algorithm works, it finds out both of them are P, so this is unifiable; it goes for the next step: x and f of x. So, you should give x here f of x as the term, x as the variable; so it goes for the substitution x by f of x. Then it updates the formulas; updates the literals to f of x y P f of f of x y. It does not mark that place to omit

it; and then goes for the next mismatch. It again starts from the beginning. So, it sees that P is there, it matches now f is there f is there, f matches; parenthesis it matches. Now, x, f of x; you see the danger. It will go on looping over there; it will be never able to come out. So, all it says is that, this also should be a condition for telling that in such cases the literals are not unifiable; the same thing happens for terms. Instead of P, you will have some g of something; so same thing can also happen for the terms.

So, what are the cases when the literals will not be unifiable or the terms are not unifiable? First thing is, they should start, if they start, any one of them starts with the negation symbol, all of them start with the negation symbol. In other words we can say their first symbols are the same whether it is a term or literal, starting with the negation symbol or not. All their first symbols should be same; either it is a not symbol or it is a predicate symbol or it is a function symbol. Once this happens then only it goes to the next step. Then it tries to find the mismatch. Once mismatch is found, one of them should be a variable, the other should be terms; at least one of them should be a variable. If both are variables then it takes the first one as the variable, next one as the term. If they are terms, then again it would have matched the symbols, function symbols at that instant. If the function symbols do not match, it will say it is not unifiable. Ultimately, it should come to variable and a term.

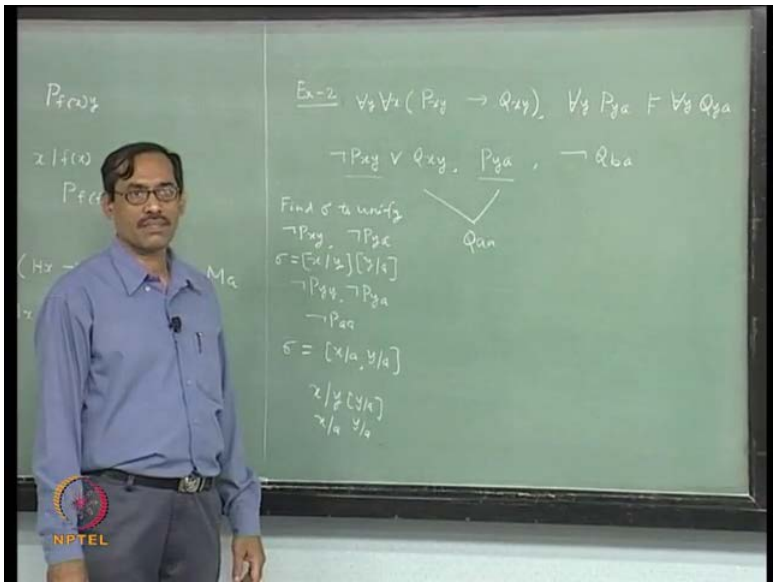
Once a variable comes, others term comes, then it checks whether this variable occurs in that term or not. If it occurs, then it is not unifiable. If does not occur, then it tries to construct a substitution x by t. So, this substitution. It now updates the two literals. After updating it again goes to the first step; starts from the matching and it continues. This is how it goes to compute the most general unifier.

Now, our aim was to use the most general unifier for resolution. Let us see one example. We will take a very simple example. Suppose we have for each x H x implies M x and M of a, entails say, H of a, entails M a. In resolution, what we will do is, try to convert all these things to scnf, where you will get the disjunctive clauses; then we should take the resolution on the disjunctive clauses as in PL; that is what we are doing. But somewhere more general unifier can be helpful. When converting to a clause set, we would get from this as: H x implies M x, for all x is simply omitted. That is same thing as not H x or M x. This is a clause. Manually when you write, we will write with or, but a machine would take it as a set of those two literals. Then, we have H a and we bring it to the other side

for using reductio ad absurdum. We want to see that this gives you bottom, by taking the resolvents. Now, when you take resolvent of these two, it sees one is not H x, another is H a. Imagine this is not H a. It would have gone for the conclusion as M x. If it is not H a, but it is not exactly H a, it is x; here is the role of the most general unifier. It tries to unify with not H x, with not H a, putting another not here, because unification needs both of them same. What it does, it unifies not H x with not H a. This set of literals. It computes sigma as the mgu of this set; with one not symbol added. Then you see that x divided by or x by a is the substitution. Now, it applies the substitution here, and then it takes resolvent. So once it applies this substitution, this is not H a, M a and H a, so the resolution gives M a. All this is done in one step; apply the most general unifier and then resolve. That whole thing is called resolution.

Now, it is just like proportional resolution; but before it, we should apply the most general unifier. The most general unifier of what? Of not H x not H a. One not symbol is added here. Then it gets x by a as the substitution. So, it corresponds to a universal specification. This would have been for all x not H x or M x, from which you conclude not H a or M a. Then, not H a and H a resolve to give you M a. Then next step would be, of course, not M a and M a to give bottom.

(Refer Slide Time: 08:04)



Let us see one more example, before giving the resolution rule. I just change it a bit. Take for each x for each y, say, P x y implies Q x y; and I have another, which says each

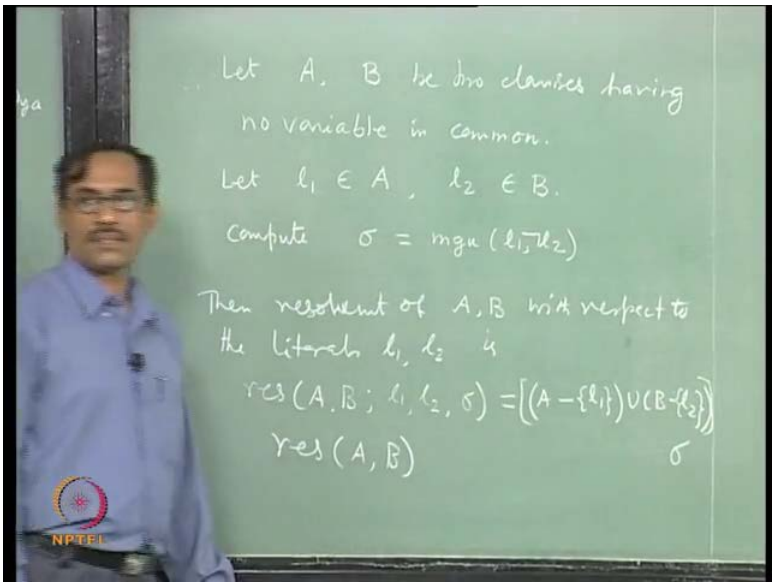
y P y a. Therefore, I should get for each y Q y a. It looks alright semantically. Let us try to see what happens. When I take, use RAA, it will come as not of this, then I have to convert to scnf; so first one gives not P x y or Q x y, next one gives P y a, next, not of this. That becomes there is y not Q y a. Then you have to chose a Skolem function for y; it does not entail Q b y. But now you are choosing a Skolem function, because you want scnf. Skolem function gives y replaced by some constant; it depends on nothing; this becomes simply not Q b a. Am I right? Well, let us apply resolution.

Here it takes not P x y as one literal and not P y a as another literal. You want to unify those two. So, the problem is: find sigma to unify not P x y not P y a. How does it find? It matches them; it gets the substitution x by y. So, it gets x by y; after applying x by y, it gets not P y y, not P y a. Again it chooses another substitution, it is y by a. It composes them to get a sigma. Once y by a is applied, they are unified to P a a. Then what happens, not P x y now becomes not P a a. This one becomes P a a. These two unify and then resolution is applied. Both x and y have become a now. Because this composition is simply x by a, y by a; because in x by y, you will apply y by a first; that x theta s or t theta. So, that way this x becomes a, y also becomes a; that is the composition; is that ok? You get the composition; let us see. We will be starting with this set x by a, y by a. Remember, if this set; this is x by a, then add to it the other one y by a; if y and x are not same; and then delete x by x forms; so nothing is to be deleted here; that is the composition.

Now, with this composition we get, with this most general unifier x is a, y is a; so not P a a, P a a gives you Q a a. Now, what happens, Q a a and not Q b a cannot be resolved upon. No unification is possible; because a is constant and b is also a constant. So, the method fails, right? What is the reason? Why the method fails? See, even in scnf, you have seen it while converting to scnf of a formula; what is the first step? It is the rectification. You rectify. Otherwise, your prenex form will be wrong. What we have done here is, each formula, each sentence has been rectified; but as a formula totally, it is not rectified. While you are taking the most general unifier, these variables are shared in two clauses. y is a variable present here, y is also a variable present here. That has been shared; that has been captured by this most general unifier. So, most general unifier is incapable of producing that sharing of variables. Its generality is lost. Because of this sharing of variables, while converting to scnf, you must take care that the variables are

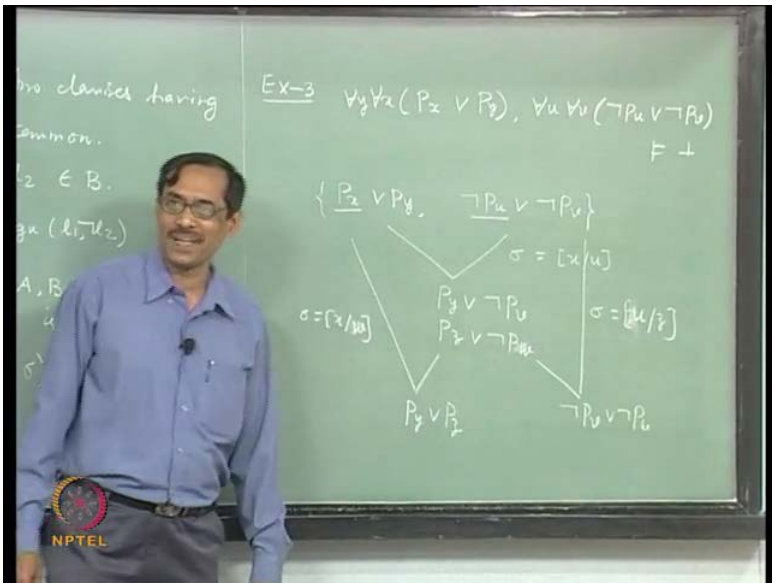
not repeated in the clauses. So, that can be done by naming or using a substitution. This is a variant, which makes them different not sharing the variables. While doing this our first assumption is, before resolution, that all the clauses are using separate variables; different clauses using different variables; no share of variables is there; then only you can go for mgu and so on.

(Refer Slide Time: 13:55)



Now, you can formulate the resolution rule. Suppose A is a clause. Let us take two clauses; two clauses having no variables in common; then you go for the formulation of the resolution rule. So, let l 1 be a literal in A and l 2 be a literal in B; then compute the most general unifier of these literals l 1, l 2 with a not sign, that is what we have seen, l 1 and not l 2. Then, what happens, the resolvent of of A, B with respect to these two literals l 1, l 2 is sometimes, we also write what is the mgu being used there or simply, we write sometimes resolution of A, B even all these symbolism are used. This will be equal to A minus this l 1 then B minus l 2, and then apply sigma, that is what we are doing here; you forget from this clause P x y not P x y forget from this clause P y a, and take whatever is there; apply your most general unifier on that. So, Q x y, x by a, y by a; that gives Q a a. Is that ok? This is the resolvent. Resolution rule says that from two clauses deduce its resolvent. Whatever way you get that resolvent does not matter.

(Refer Slide Time: 16:45)

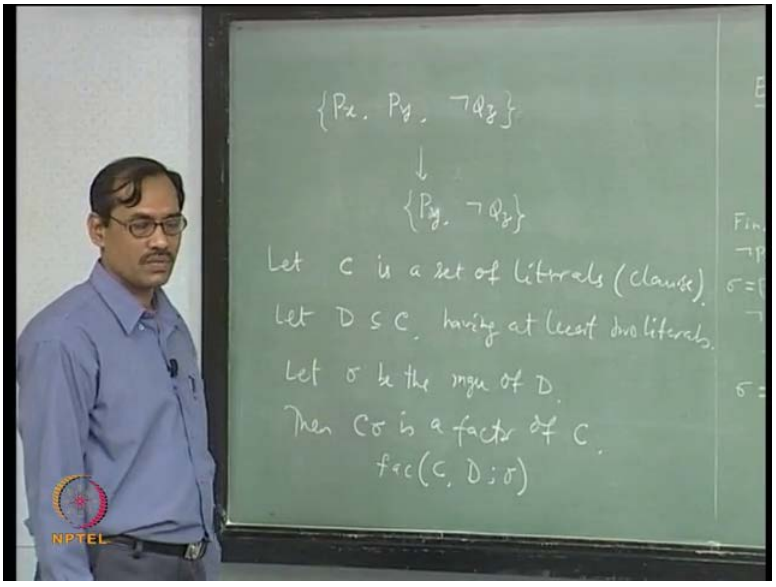


We will take one more example. The fist one is equivalent to for each x P x. Similarly, the second one is equivalent to for each u not P u. So, clearly they should produce bottom; they are unsatisfiable. Let us apply resolution on that. Now, they do not have any variables in common. First one in scnf, we write as P x or P y; second one as not P u or not P v. Let us choose one as P x, another as not P y. We will take negation of this and try to unify it. Our convention is, if double negation comes that is omitted in the literals. When you take the literal you say not of that literal means, if double negation, that is omitted. That means, we want to unify P x with P u. So x by u. This x by u, what do you get? You get P y or not P v. Now, sigma is equal to x by u. When you take the further resolvent, say P x or P y and P y or not P v. First thing is, they are sharing the variable y; that should be renamed in one of the clauses. So, that means you may write this as say, P z or not P v; then only we can go for resolution. The first two, if, I see, say, not P v with P x. So, sigma, I would choose as x by v; and that gives P y or P z. If I do with this, resolution, I would have chosen this as w. Now, it is having no common variable. With this even, no common variable with this, any way I can resolve. When I resolve these two, say with which one, P z with not P u, so z by u or say u by z. These two go and these two remain; w and P v, whatever way you go, either this way or that way; resolving this with first one, resolving this with the second one. One of the approaches will be ok; then next we will try to resolve this, but this is in the same form as earlier.

So, it looks it will go on; a never ending process. Do you see what is happening? We have one danger here, in the first example; second example; it says if you share the variables, you will get into problems. You will not get the conclusion that you want. The other one is telling, if you do not share the variables, you will not get the conclusions what you want. So, second one anyway we have agreed, because always we can rename. But this one is a major hurdle, because always you can rename. So, there is no harm in renaming. Then why don't you rename? If you rename, there is a problem; it is going to be unending.

This means resolution principle as we have formulated is not complete. Probably there is another rule, which you require to handle this kind of things; which, the resolution or taking resolvents is not enough. Try to see what is happening here. The main thing is, the equivalence we wrote here for each y for each x P x or P y entails P x, that we are not able to get from the resolution process, because they are not sharing the variables. Had they shared the variables P y or P v, now, you can resolve P x with not P v and get P y directly. This is the main problem here. Then you have a remedy. What do we do is, even if they do not share the variables, we can do something like that.

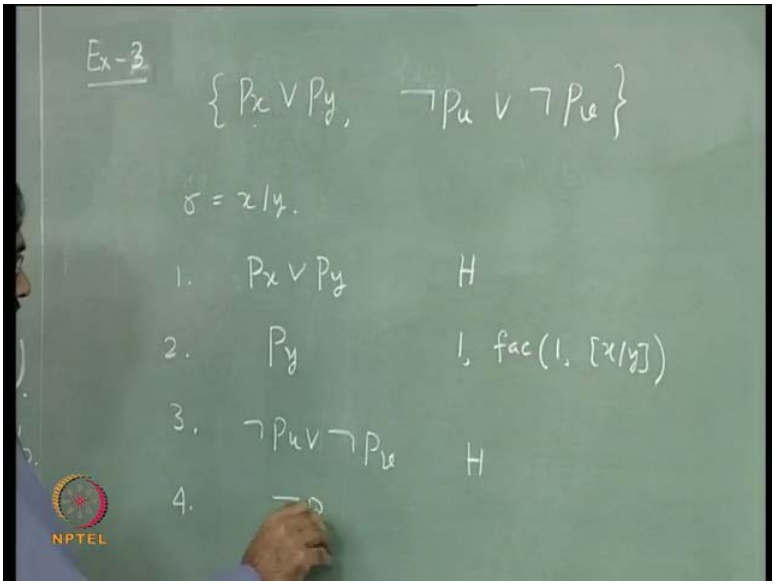
(Refer Slide Time: 22:00)



The main thing is, we have a clause P x, P y say not Q z. From this clause we would like to, in fact, P x, not Q z or P y, not Q z. This is what we want: P x or P y should give as just P y. Which says that subset of the set of literals can be brought down to one literal

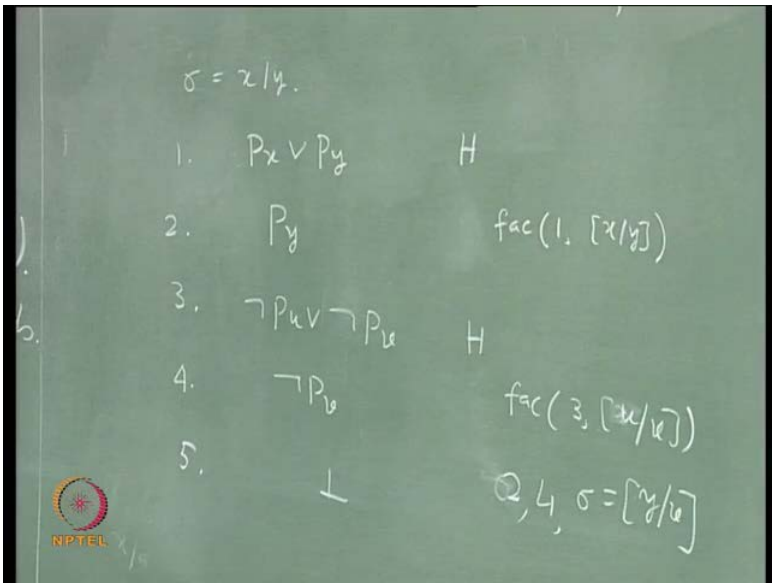
by applying some substitution; that is what it says. So, let us formulate; it is called a factor of this clause P y. not Q z is a hector of P x, P y, not Q z. So, let us define the factor. Suppose C is a clause, is a set of literals. Let D be a subset of C having at least two literals. Then you can unify them. Let sigma be, a unifier, or you can take the most general unifier directly, be the most general unifier of D. That means, once you take D sigma, D sigma becomes a singleton like your P x, P y becomes P y; then keep all the others, apply also sigma on that; that is the factor. Then, C sigma is the factor of C. This factor, sometimes we write as factor of C with respect to this clause set D. That is enough because mgu will be fixed by D itself; if they are unified. Sometimes we also keep that C. What this suggests is, not only deduce the resolution or resolvent of two clauses, two literals, but along with that you have another rule: deduce also the factor. If there is a set of literals, deduce a factor of it.

(Refer Slide Time: 24:46)



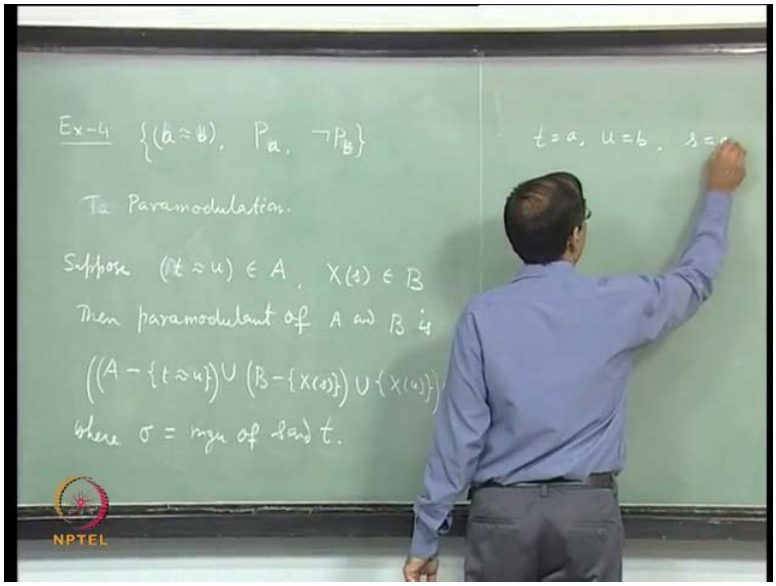
So, let us redo that. Here, we have the clause sets P x or P y, not P u or not P v. Now, how to chose a factor? You just take this set of literals P x or P y. Now, P x is a literal, P y is a literal; that is my D. There are two literals; in D, either this or that. So, C is P x or P y. D also I take equal to C; P x or P y. Now, most general unifier is x by y, so sigma equal to x by y; then this says, if I start by deduction, P x or P y as a hypothesis, I would go, in the next step, P y; my justification will be 1 and factor of 1 with sigma equal to x by y. That is how it will go. Next step? I would do, third one, which is my hypothesis, not P u or not P v, so that is the way, hypothesis.

(Refer Slide Time: 26:10)



Fourth one, I would go for, factor again. Two, sorry, 3, hector of x by this is, u by v. So, these line numbers can be omitted anyway; it is there. What are the formulas? Fine. Next, I would say, I resolve Py and not P v; that would give you bottom directly because here sigma equal to? I will write the line numbers 2, 4 sigma equal to y by v; that is the most general unifier of P y and not not P v, that is all. So, along with taking resolvents we should have factors; that is what it says.

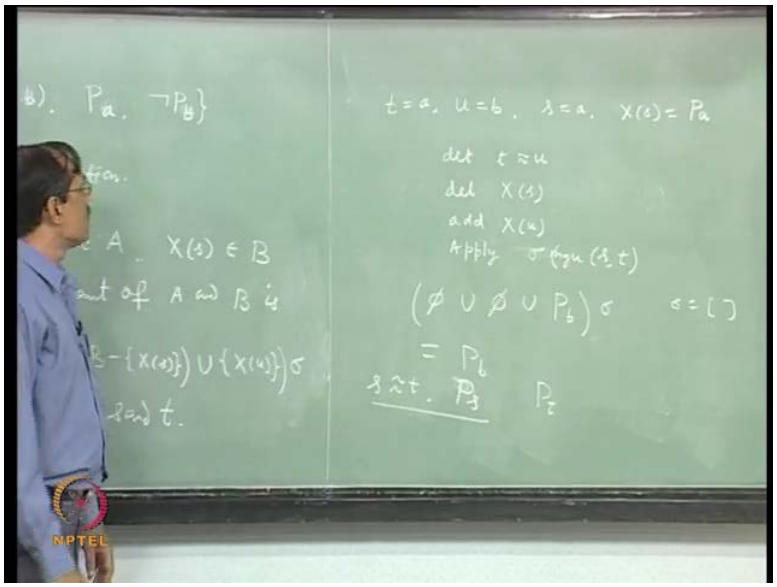
(Refer Slide Time: 27:25)



Let us see one more example. What about this clause set? It is unsatisfiable. Yeah? But how do you proceed? Instead of this, t is, suppose it is a variable, u is a variable; then directly you can give. Because these two will give resolution as bottom, with sigma equal to t by u. Now, let us change it slightly. What happens, if you change them to constants, let us say. Now? We cannot take resolvent of these two. At least one variable is there; then you can do, but there is no variable. So, P a, not P b that is all. Something has to do with this equality, because they are equal you can think of P a as P b also; that is why it is giving contradiction.

Now, as such it does not give any contradiction by resolution method, not doing by factors, because factors also need variables to be substituted upon. You need something else here. That means resolution along with factor is not a complete method; it does not handle equality. For this, we have another rule, which is called taking paramodulants; and we just say paramodulation. It is a bit complicated. Lots of symbolism are there; but we will see how to proceed.

(Refer Slide Time: 32:08)



Suppose A is a clause having, say, t equal to u belongs to A. Here, I am taking t and u as general terms, not only constant or variables, general terms. A is a clause; t equal to u belongs to this; and then, let us say, X of s belongs to B. So, here we write X of s. It means X is a literal, s is a term which occurs in X; X of s means that here. This is the notation. Then, paramodulant of A and B is A minus t equal to u, then B minus X of s;

should take their union, join to that X of u. Then, use the most general unifier of s and t. Here, we need most general unifier of terms. Then, the rule of para modulation says that from two clauses A and B, deduce their paramodulant.

This is my t, this is my u. Let us say t equal to a, u equal to b. Then we should have another literal in B, another clause, let us say, in B here, so s equal to a and X s equal to P a. We have two clauses. Then, this says, first delete t equal to u from A. We can write this one as delete t equal to u, delete X s, add X of u, apply sigma, most general unifier of s, t. This is the paramodulation process.

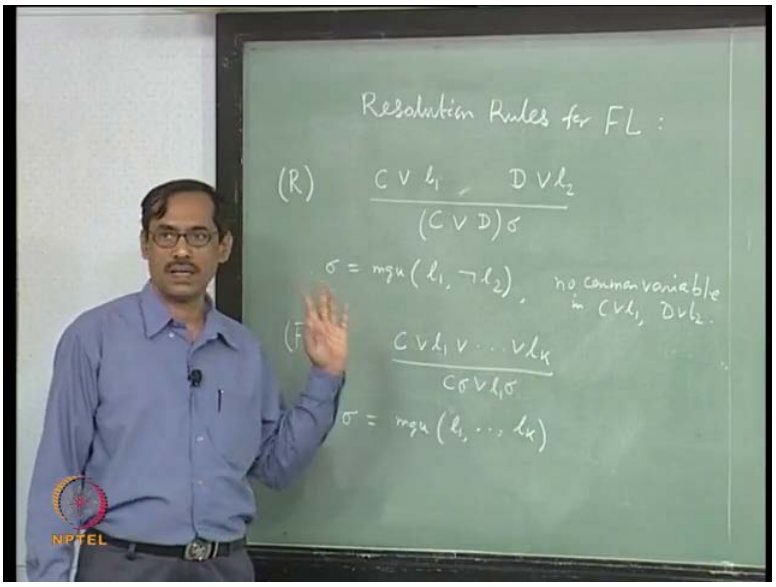
Let us do that now. If I delete t equal to u, there is nothing in A. I delete X of s from B, there is nothing there. Then I add X of u; it is P of u, u is b. Then, apply sigma, which is the most general unifier of s and t; s is a and t is a, so sigma is empty; there is no variable in it; let us call it empty substitution, nothing is applied. It is kept as it is. It is the identity of the composition of substitutions. That means, this becomes P b. This is what we required. From t, a equal to b, and P a, you are able to infer now P b, because of para modulation. Usually, this is the form you will be using most of the time. Fine.

Suppose you have s equal to t, and some P s. Then, you should be able to infer from this P t; this is what it says. But there, it is a bit more complicated; it can handle still bigger things, not only this much; that is why this rule. This example can be done now. Because from these two, you would get P b by paramodulation. Then, take resolvent of P b and not P b; that gives the answer.

So, you have the third rule of paramodulation. Is it enough or we are going to give more? Because equality, it looks we can handle. But there is one problem. In all of these three rules you need at least two literals to apply anything. Can you see that? In taking the resolvents you need at least two literals on which you apply the resolvents, then in taking the factors you need at least two literals in D so that they unify to become one literal, fine? In para modulation you need at least two: s equal to t and X of s or X of u. In fact, three are there, some of them can be same. So, at least two will be required. Now how do you infer t equal to t, which is valid? This is valid, right? Now, how resolution method will give t equal to t? It cannot. Because everyone needs two. You can repeat it. You can repeat it as two. Suppose you repeat. Can you still get so that this is valid? If you repeat, that means, what will happen, Phi entails t equal to t; that is what you have to prove; not,

t equal to t entails t equal to t. So, it seems nothing; still it gives nothing. What we do is, this will be our fourth one, anywhere you can deduce t equal to t; because that is what we are not able to get. So, all together there will be four rules of resolution. One is taking resolution, another taking factors, third one is taking paramodulant, fourth is the equality. It looks like an axiom. Anywhere, you can introduce it, you can deduce it from anything.

(Refer Slide Time: 37:25)



So, let us write the rules. First rule is R: taking resolvents. This says, if you have a clause C along with some literal say l 1, and then another which is D, another literal say l 2. Then, from these you conclude C or D sigma. What are the conditions here? Sigma is equal to the most general unifier of l 1, not l 2, and then C or l 1, D or l 1, they do not have any common variables; that is also required. So, no common variables. We will write just this way. These are the conditions. Then from these two you can deduce this.

Next, we have the factors. Which say, if you have C or l 1 or l k, then from these you can conclude C sigma where sigma is equal to the most general unifier of all these variables or literals l 1 to l k. Finally, this one will come to C sigma and one of this say l 1 sigma, because once you apply this sigma, l 1 sigma is equal to l 2 sigma is equal to l k sigma; all of them become same. So, that remains, and all the others with sigma, that is a factor here.

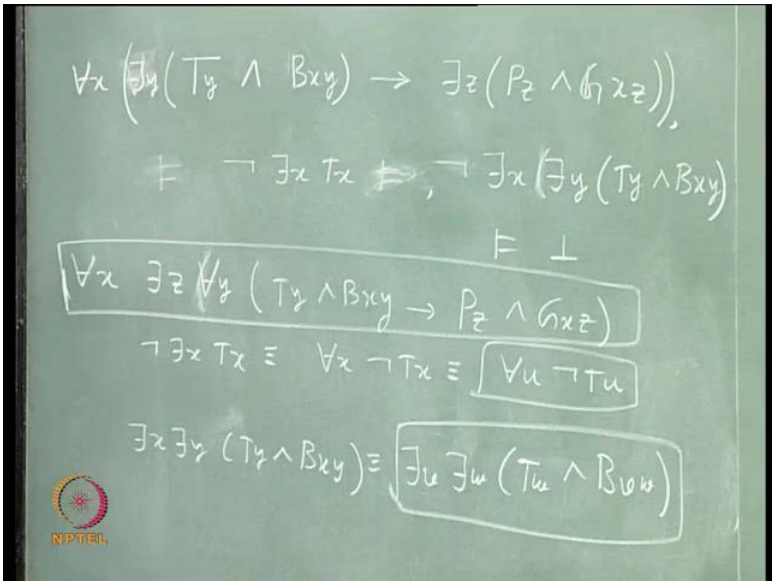
So, next one is taking paramodulants or paramodulation. There, we have say, some clause along with t equal to u. There is some other clause with X depending on s. From

these two we conclude C or D or C of u and then sigma, fine, where sigma is mgu of, is one mgu of s and t. So, there we are writing in the notation C minus something; now that C is this total thing; that earlier you have seen. It will be easier to apply this way.

Next rule is our equality. It looks like an axiom. Wherever it is, you do not need anything here. You can infer t equal to t. That is all. These are the rules of resolution. This is really a complete procedure. Now, we will not prove this.

We will see some examples; how it is applied. And these two rules of resolution and factor; they are really combined together. That is called the generalized resolution or sometimes they call this one as binary resolution. Their combination is the resolution, where what happens is, you just modify the resolution rule. In the resolution rule you are taking l 1 as the literal l 2 as the literal then taking their unification. Right? What happens in the general case, instead of l 1 you take a subset of literals. It is a set of literals; this is also a set of literals; then they are unified. Once unified, omit all of them; and get this directly. If you do that way then that will be the generalized resolution. So that you do not need the factor rule. But, while using manually this will be easier than using a subset and going on doing it.

(Refer Slide Time: 39:51)



So, what are the first steps you will be doing? Yeah? Reductio ad absurdum? So, first we will take, this, use deduction theorem; because it is implies. You take this to the other side Then RAA; that gives this entails bottom. You want a resolution refutation, always

it ends with bottom. These are the premises from which we have to start. They might be sharing variables; that we have to take care; they should not share variables. Now, what is the next one? Is it rectified? Yes? For all x there is y there is z, they are all different variables; first one is rectified, but then you have to pull the quantifiers and taking it to scnf. So, I have to pull the quantifiers; z does not occur anywhere here, so this z can be brought to there is z. Similarly, there is y, but there is y will become for all y; it is before the implication sign. So T y and B x y implies P z and G x z. What about this? Not there is x T x. This, we will write as for each x not T x, which you would like to change x to something else, say, we will write as for P g u not T u. You want different variables in different clauses. But that may not guarantee; after only converting to this you can know what are the clauses; that will be in scnf. It is a conjunction of disjunctions. In each conjunct you should use different variables; that will see, if needed we will do that. Now, what about the last one? That is, there is x there is y T y and B x y, as it is; only we have to change the variables, say, there is v there is w T w and B v w.

We will leave it here. Then, next job we have to do is, convert this one to scnf; convert this one to scnf which already is; all this three have to be converted. Then, get the clause sets, apply the resolution. This is what we will be doing next. We stop here. All that we have done is just formulated the resolution rules.