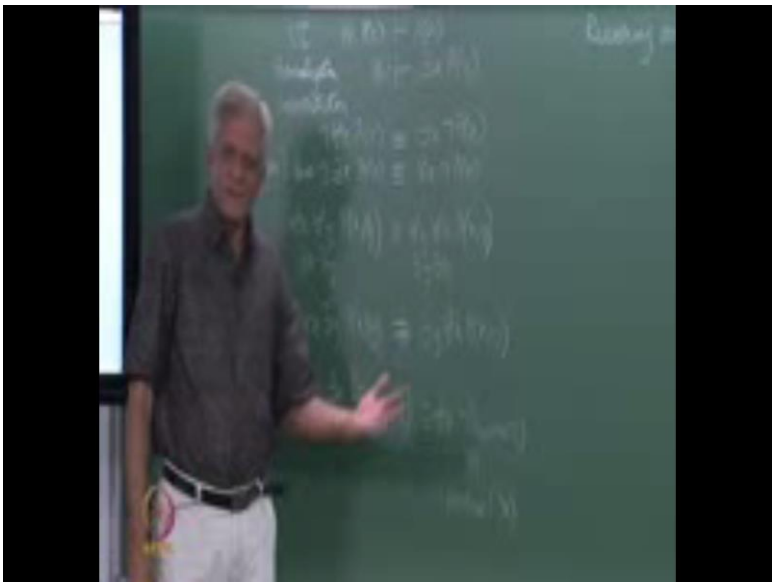
Artificial Intelligence
Backward Chaining
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 47
Backward Chaining

(Refer Slide Time: 00:15)



So, we are looking at FOL and in particularly we are looking at reasoning aspect. We shall not have the time to look in to the knowledge representation aspect too much here. So, we will assume some simple representation schema which means the choice of the functions at we are talking about a set of predicates a set of functions and the set up business.

Now, we saw 2 rules of infants; 1 was a universal in sensation we said that a from for all x and this is an instance of that. You can p of a and then we saw generalization it says that from p of a you can reduce the x p x. So, in case you have a query of this kind we saw an example of that earlier. Now, in addition to this there are also some rules of substitution. So, for example, you can replace for all x p x with there exist x naught p x and likewise it exist naught exist x p x equivalent to for all x naught p x. So, these are kind of commonsense rules. So, if you see what we are saying here we are saying that if

it is naught the case that for every x from property p or some predicate p is to it means there must be some x for which p x is naught true essentially.

So, if you move the naught across a quantifier it changes the nature of the quantifier. If you move a naught a across a universal quantifier it becomes an existential quantifier. The rest of the expression does naught change it is like you are moving a naught inside, but changing this likewise if you change the move are naught across the existential quantifier it becomes a universal quantifier. So, again if you look at this; this is like saying that for example, if p x stood for something which is both even and odd. So, this left side is saying that there does naught exist an x which is both even and odd which is equivalent to saying that for all x which the case at they are naught even and odd. So, you can move them across on both sides. So, these are rules of substitutions which I quite useful in some situations we will see some of them. And you might be familiar with them they are known as de Morgan's laws. Also something which are use some times that is that for example, if you have for all x for all by p x y there p some predicate this is equivalent to for all y for all x p x y.

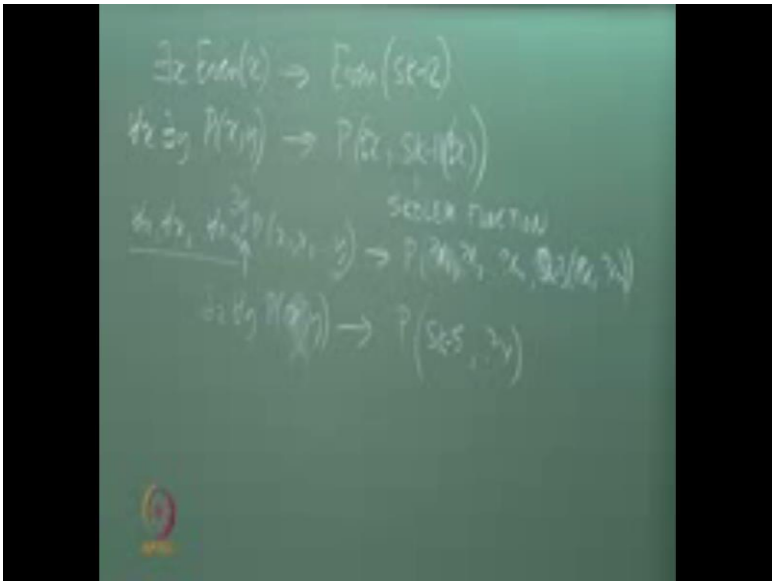
So, the quantifier of the same kind you can change without changing the meaning of the sentence. Likewise for there exist there exists y there exist y; there exist x they would be similar, but if the quantifier of different kind then you cannot do that. So, if you say for all x there exists the y p x y this is not equivalent to saying that there exist a y thus that for all x p x y. So, you can try and think of a counter example to show that this is not the case. So, for example, for every number x there exist a number y which is bigger than x. So, p stands for bigger than or greater than then this statement is true whereas this statement say that there exist a y which is greater than every number. So, which is obviously not true essentially So, you cannot switch 2 quantifies of different kind if you do switched and the meaning change as the truth value also will change. And you are talking about something totally different here and something totally different here. Whereas, here there is no difference if they have the same kind then you can switch the quantifiers and it does not change the meaning.

Here we are saying that if you if you have a negation sign you can move it inside inwards towards the expression and what it does? It changes the sign of the quantifier

changes the nature of the quantifier. Universal quantifier becomes existential and existential becomes universal, which is why I had mention in the last class at it is not very easy to identify what is really the nature of the quantifier. So, for example, if I make a statement a it does not let say we are talking about people and this statement is saying that they does not exist a x who is divine. What is the nature of this variable x? Is it a existential variable or is it a universal variable? And this becomes important, because if you remember the implicit quantifier for that we discussed in the last class you can replace a universally quantified variable with a question mark and then it become simplistic.

So, is this a universally quantified variable or existentially quantified variable? So, the way to look your answer is right your way the way to understand that is to push the negation sign as much is possible. And then look at what is outer most quantifier essentially if you push a negation sign inside it would become equivalent to saying that for all x naught divine x. So, we are saying that everyone is naught divine which is like saying that no 1 is divine essentially. So, this is the universally quantified variable. So, we can replace at he want to put at in implicit quantifier form by saying naught divine x I putting a question mark that is a cementation we are adopting between our sense. So, that we do not have to write the quantifier we do not have to process the quantifier when we are liking a program to do that we can just keep those kind of variables as universal variables. So, just a word about what you do with existentially quantified variables when you are talking about implicit quantifier from.

(Refer Slide Time: 08:18)



Let us forget a sentence like this there exist x even x. So, it is saying that there exit some number which is an even number essentially. The way to convert is into implicit quantifier form and the process is called skolemization after legation called skolem. So, well solemn who for us introduce these ideas by replacing it with a constant even s k 12. So, conventionally we may say we use a name s k this is us again between us it does naught impact the meaning of what we are writing it is a constant. So, we so what are we done? We have remove the quantifier and replace it we replace x by something called s k 12. So, s k again is the sin honors or skolem you could have use any constant as you as far as you remember that it is a constant which is been introduce in this process and it must naught be a constant which is being use anywhere else. So, you cannot say a for example, 0 if it is a constant or something like that you must use some unnamed some new name and treated as a constant after that.

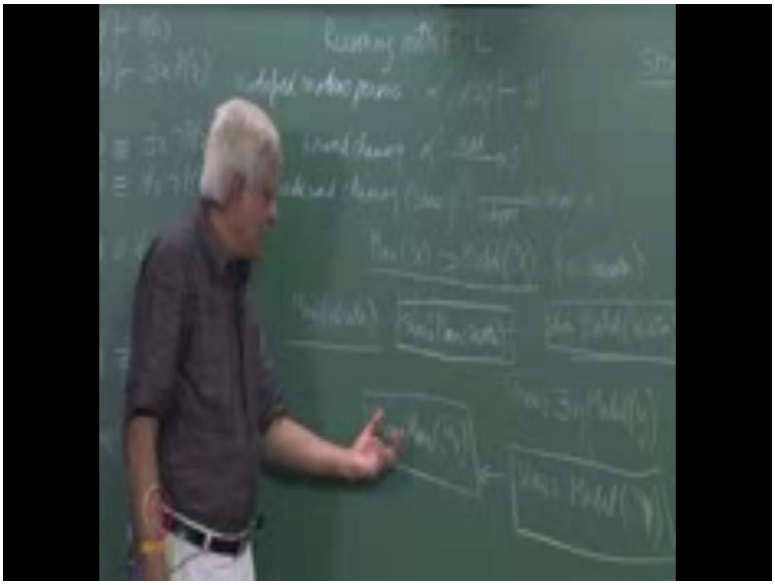
Because if you look at the meaning of the there is some number which is even and all you have do saying is here that there some numbers which I am calling s k 12. And that is even essentially, because you may naught really know what that number is, but you can do that. If you have something like this or statement like this for all x there exists y p x y they if you look at the what is the sentences saying the sentences saying you should always read the quantifier from left to right that for every x there exist a y. So that p x y

is true essentially. So, for example, for every x there exist search y which is greater than x essentially. So, that is an example that we mentioned in the last class we can this into an implicit quantifier form by writing it as p x s k eleven x. So, what are we saying here when we have a existentially quantified variable inside scope of a universally quantified variable then we are replacing that variable with the function of the other variable and the function is the skolem function.

So, this is the this is the skolem constant of x So, what is this function we do not know what that function is it is some function and what we intent by the usage of the word function here is that the value that the variable can take is dependent upon the value that x can take. So, again if p stands for greater than and the implication and the meaning of this is at y is greater than x because the meaning is always determine by us. It depends upon the relation that you are talking about essentially then what we are saying is that for every x you can choose a y. So, is that y is greater than x so that y that it choose depends upon x and therefore, we can think of it as a function of x. We do not know what that function is, but it some function as for as reasoning in concerned will treated as a function and translate this into this statement.

So, this so in general off course if you have for all x 1 for all x 2 for all x n p x 1 x 2 y then you will translated to p of x 1. So, I should put a question mark here because x is the universally quantified variable x 1 so this y which is in the scope. So, there exist the y here is in the scope of all these universal quantifier for x 1 to x n. So, this y become to skolem function of x 1 to x n. So, in general you just look at what are the quantifiers on the on this side universal quantifiers on this side and make that variable of function of that. If I had something like this there exists x for all for all y p y p x y then I would replace it with p s k 5 y. So, this entire process of converting a sentence into implicit quantifier from now it is we call it as skolemization after this suggestions skolem essentially.

(Refer Slide Time: 14:55)



So, in the last class, we had started looking at forward chaining we had looked at the role of modified modus ponens mindset that if you have alpha prime and alpha implies beta then you can produce beta prime from there. And if you move from alpha prime to beta prime the process is called forward chaining. So, in forward chaining you have alpha prime and then you can add beta prime in backward chaining you would not to show beta prime. So, we use it is connection show beta prime so we have sub goal now show alpha prime. So, backward chaining we just started looking at in the last class it works with goals by goals. We mean something that we want to show to be true a formula that we want to show to be through. And looks for implications of this kind alpha implies beta so basically it is using mod modus ponens in a slightly different form. Or if you think a little bit about this you can see that what backward chaining is the doing is this kind of modus may be that will become clearer as we move forward that is a different rule of infonants.

So, if we have our original Socratic argument which said that man x implies mortal x and in our database. So, this is there in the database and this is there in the database which is man Socrates. And off course, you may have as a date of birth less ignores that for the movement and you want it to show that mortal Socrates is true. Then forward chaining would apply this modus modified modus ponens rules and in this form and this form this

thing. Whereas, backward chaining we are not written the it in this form, but it says that which show beta prime man with alpha implies beta you can reduce it to show alpha prime essentially. So, show mortal Socrates how does it is works if you recall we match this with this with a unifier x is equal to Socrates we will look at unification a little later. So, we apply this unifier to this and according to this backward chaining process we reduce this to show man Socrates. So, let us put wholes inside boxes so that we can distinguish between goels and fact. So, this is a fact this is a fact this is a goel; this is a goel and so on. So, what this off course, it severe way to simplify to solve the goal and so see with that is present in the data base. So, in this example, when you have this goal of show that that man Socrates is true then you simply look up the database.

And you will find that it true essentially will in the movement we will look at how the language prolog is basically doing this which is it is doing backward chaining. And if it if a fact is present in the database or in your program prolog program it is really true essentially. But the nice thing about reasoning with logic is that we can ask a question like is this formula true that it does there exist the y such that mortal y is true this is what is given to us this is a database. And this is a database even this database is that satement true that there exist the y so that mortal y is true. So, I intentionally use a different variable here y and here an x here it is naught really necessary forming to do that. Because when you look at this statement then it is saying that for there exist some y such that pus that y is mortal. I could have Jolivel said there exist some x such that x is mortal, but because we want to set off keep above formally spleen and a part and is a practice that is necessary as we will see.

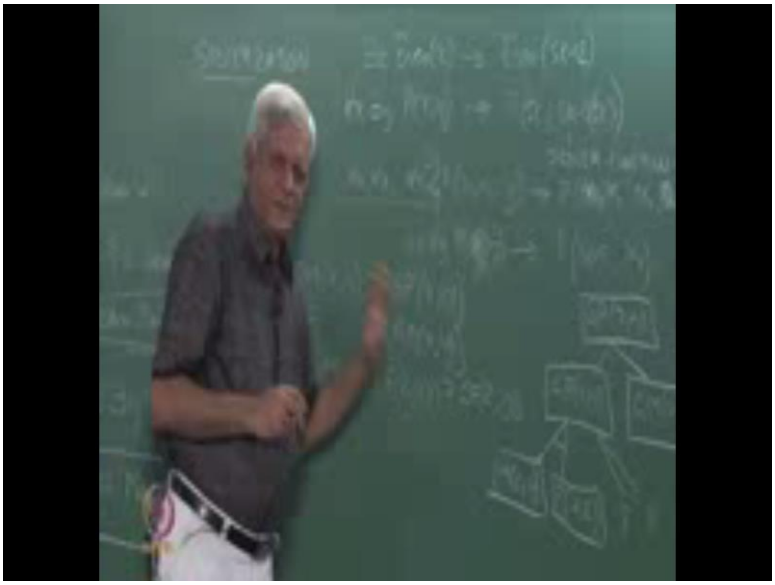
We use a different variable name which is a is a goal noise know if I if this is my goel that show that this is a case. Then we invert that is skolemization convention when you are doing forward chaining a formula like this a a universally a formula which has a universally quantify variable is put with a question mark here. Whereas, in backward chaining an existential variable is denoted by a question mark that is specifically in backward chaining when you are talking about show when you are talking about goels sorry for goels the convention is reverse. So, I will this as mortal y so again let be that when I write show mortal y in this form this notation denotes and existentially contified variable that is true only for variables which occur inside goels. So, inside goels a

convention is reverse essentially and the next class we will see why that makes sense but you can try and guess.

So, I can ask a query existentially query so this is now becoming like little bit like a database essentially activity. So, you have some set of acts available to you and you are asking a query is there some entity with satisfy certain properties. So, if you are saying somebody like something like is there in employee he wants more than 10000 rupees and something and you get some results out of that essentially. The difference between something like RDBA BMS and using login is that logic can make inferences on the way essentially to retrieve answers for you. So, when you asking a question that is there somebody whose mortal in the knowledge base or data base whatever you want to call it. The knowledge base has only 2 statements in our small knowledge base that Socrates is the man and that all women are mortal a knowledge base does not say that any one is mortal any specific individual is mortal.

But backward chaining in particular and reasoning in login in general allow you to ask existential queries like this. And answer those queries after a process of doing some inferences making some direction. So, what do we do? We match this show mortal x with the right hand side of our statement implication mortal x. So, we say x equal to y is the substitution you want so this becomes mortal y. So, this case translated to show man y so our query is about. So, it is still an existential query it is a easier someone whose mortal gets translated into a sub query or a subgoel which is easier some whose the man essentially now that can be answer by the database yes Socrates is the man. So, we can communicate the query written the answer to this query by saying yes y equal to Socrates is the answer to your question. So, in the last class, we had briefly mentioned that you could define. So, let say you want to define grandfather or let say you want to define grand pa grandparent.

(Refer Slide Time: 25:08)



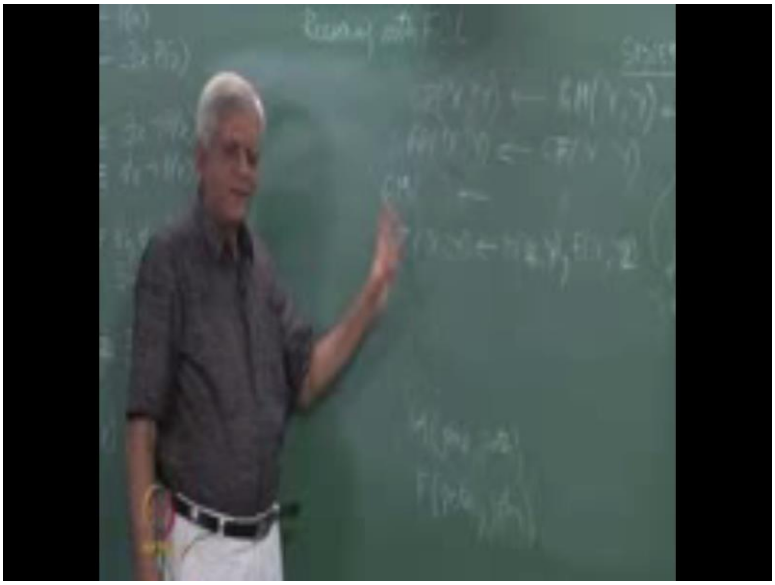
So, you might want a statement like this a for all x for all y grand. So, let say g m stands for grand ma x y and let say this means that x is the grandmother of y implies. Let say g p stands for grandparent let say for some reason we want to we want to this put this rule into our knowledge base or database, what you want to call it? So, what you are saying here is a grandmothers are grandparents then you could have a role like for all x for all y g f and the same thing. Then you could have a role which says a for all x for all y and for all z mother x y and father z x implies grandfather z y. I could have a role of this kind which says that for all x for all y for all z if x is the mother of y and z is the father of x then z is the grandfather of y. So, you could imagine that you have billing a database of relationships where you are defining how what is relationship means what is grandfather abouts a grandmother abouts a grandparent and so on and so forth. And then you should be able to ask a query you should give us database of acts. So, let say the basic database only contents a mother child or let say parent child and the gender of each person. So, I could say a Jane is a parent of tom and Jane is female and Tom is male.

So, I could have this kind of database. So, there is only 1 relation parent child relationship and general relationship then you can define a mother at saying that x is the mother of y if x is the parent of y and x is female. So, you could do all that kind of staff we will naught get into the details, but you could have a knowledge base of this kind.

And then you could ask a question how is Peter related to Jane? For example, you know then the system should find whether what is the relation between them? Off course, you cannot ask this very generic question is to how it is related you can ask something like whose Janes paternal uncle. For example, you could ask such questions essentially let say we in talking about grandparents whose grandparent. And that is about to basic query about then you can see that grandparent x y can be solve in 2 is that either you are a grand pa x is the grandfather of y how x is the grandmother of y? Then you could say x is the grandfather of y if you could use this as 1 rule which says that now you have to a bit careful here a father x z. So, I am just using different name such to be consistent here and mother z y. And you can imagine that there is a another rule which uses father and father in both the places so that 2 possibilities here.

So, you can see what is happening here that is the space in which backward chaining operates if you ask a query about something is Jane the grandfather of Peter then this system will apply. So, in backward chaining you match with right hand side of an implication and c is the left hand side can become a subgoel or you can match either with grandparent So, you can match either with this rule or with this rule in both cases grandparent will match. So, you could either use r 1 or 2 if you called 1 rule will take you here another rule will take you to grandmother. So, either x is a grandfather of y or x is a grandmother of y then grandfather if x is a grandfather of y. It could be that x is the father of z who is the mother of y or it could be that x is the father of z who is the father of y both are possible. So, you have all these question possibilities and the space that backward chaining we have to search is that is an ando trees essentially and what prolog dose is backward it does depth first search on that tree. So, let we rewrite this you know I am skipping in the step of skolemization which in this cases simple because we are only universally quantified variables. So, you can just replace everything with a question mark before that if you were to write this in prolog you would write it as something like this g p.

(Refer Slide Time: 32:10)



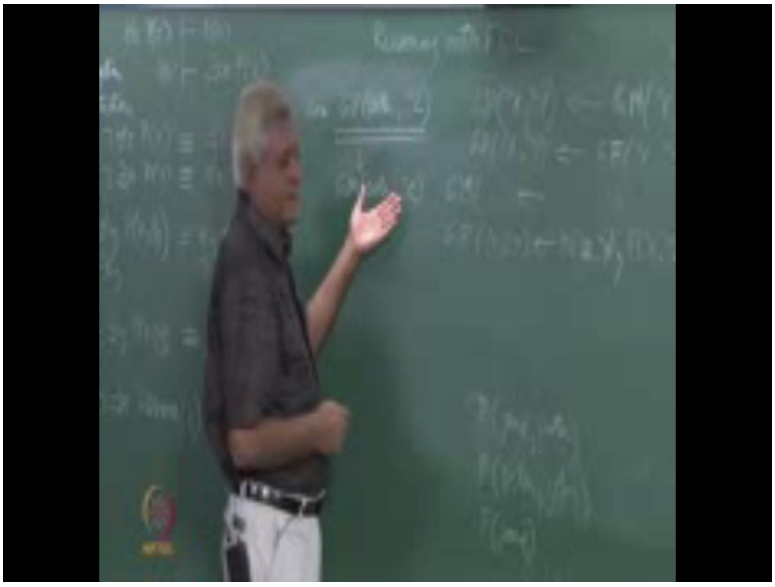
So, prolog is this the different convention pro see we are using a convention that question mark stands for the variable and something without a question mark stands for a constant. It should be a constant or existentially quantified variable which is the skolem constant or something like that it does not matter. But that is a convention way using prolog is this the different convention prolog uses the case that when you have x and y uppercase letters then it is a variable. So, those of you why use prolog would know this so let us stick to our convention which is to use a question mark. So, we write the consequent first and then we write the antecedent. So, this same rule let us write it like this so what is happened I have taken this rule and rewritten it like this are you I have away the quantifiers I have convert it to an implicit quantifier form. And I have the order in which you are writing in the normal rule you write there antecedents on the left hand side and the consequent on the right hand side in this notation. I am write in the consequent on the left hand side and the antecedent on the right hand side and I have change the direction of that.

So, have use an arrow instead of that sign here. So, again those of you use prolog would know that prolog this is something like this instead of the arrow sign, but it means a same thing that is only a matter of convention. This is the easier for us to understand that ther direction of implication is from right to left. Then I would write the second

statement as g p x y g m x y then g f x y then somewhere I would write a statement for g m then g f which is what I have written here which is that g m x y if a mother x y prolog is it is a coma instead of. And so we will also use a comma here the keep in mind that this coma basically stands for in and so the same set of statement to either writing in a different. We are not changing anything we have into implicit quantifier from and we are writing it consequent on the left hand side and antecedent on the right hand side.

And we are replacing ands by comas and this kind of staff, but the statements are still the same they are still the same universally quantified statements in logic. Somewhere down there I would have let us say a mother Jane Peter sorry oh ya a x is the grandfather of y. So, y is the mother of no sorry z is the mother of y and x is the grandfather of x is the father of z thanks. Somewhere I would have a statement saying a mother Jane Peter for example, and father Peter Jane and may be other fact also, you recognize this as a prolog programme. I hope prolog programme a statement of these kinds now this is a restricted form of logic, but we do not go into that the restriction here is that the consequent can only be one predicate. You cannot have more than you cannot have alls inside here and so and so forth, but you can recognizes as the prolog programme what does prolog do?

(Refer Slide Time: 36:55)



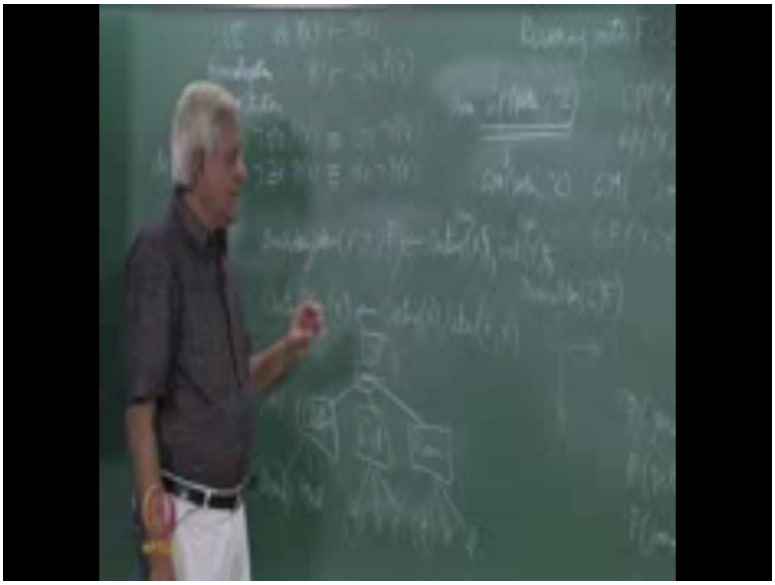
If I ask a query like is there or show grandparent x. So, a let us say Peter z or something

like that. The reason for writing things in this invited form is, because it makes a task of matching simpler you always match with what is on the left hand side in you have a programme and the right hand side is the step of making that inference. So, if I ask a query like that is there someone whose peters grandson or is there someone to be more precise whose grandfather is Peter. Then prolog starts looking from top to down trying to match this with the things on the left hand side in this example very conveniently. It matches the first element it is in so when it matches this it poses this as the goel.

So, at last whether Peter is a grandmother of some z essentially So, it will translate that gole into the subgoel. So, backward chaining we said was moving from right to left in this notation. So, in this notation it is moving from left to right. So, it basically goes from goels to subgoel ask that as a new query and as you can imagine I should have something like I have a I could use parent here parent here. And then a female Jane and so on and so both I mean that is a data that I originally said we have the Jane is female Peter is male and Jane is male and that kind of staff. This will get translated to g m grandmother Peter z which intern will so essentially here going down. So, because I written this rule first this grandmother rule first it is like having this rule on this side. So, it is going down that path so you can visualize what prolog is doing as backward chaining and with a particular strategy which is depth first. And the way it is implemented here is that the first rule matches the that.

So, it will it is it tried this so just imagine that that tree was lift from left to right. So, it would be going down the left side first. And then it would eventually with naught be able to show that Peter is a grandmother of z it would back track. And then try this second rule essentially which saying we go all the way down and then try the other branch. Exactly as we had when we looked at goel trees with that first search first essentially. So, let us look at another a example that we have considered earlier which is that of planning and outing. So, if you recall when you are looking at a goel trees the we had this task of planning and outing with a friend and outing consisted of some 3 things that 1 evening out 1 a entertainment and followed by dinner. So, eventually you know you have to finds values for these which your friend would be happy with.

(Refer Slide Time: 40:55)



So, if you are remember it was something like this that let say a let say let us call with a birthday plan I do not remember what we had said that time. So, you have a birthday plan made up of x y and z if you have an outing plane of called x and entertainment plan called y in a dinner plan for z. And then you could say an outing plan is a valid plan if it is an outing and likes that is a f stands for friend x likewise a other 2 plans entertainment plan and a dinner pans. So, you could write it as a prolog programme you could say that this is how. So, you could add f as a parameters so x comma f here comma f here comma f here. So, let say f is f if stands for your friend and let say your query is a essentially that. So, let me write the query here a what is the good birthday plan x y z and let say your friends name is Peter and this is my query.

So, that question mark after this what backward chaining will do is exactly the same thing that we did here earlier it would. So, this is birthday plan then it tries to find an outing and entertainment and dinner or may be this is have entertainment. We are said movie in the last class it does naught matter and below this would be ando tree outing let say beach or moll and things like that. And entertainment could be some in movie a or movie b or movie c and restaurant could be some restaurant d or e or f and below that. off course, you have the facts like whether your friend f likes going to the beach some statement like beach Peter it must be present in your in your knowledge base. Because

that is how will instead of asking in a friend you are saying that is there a statement like that in my database already.

So, that attention that I want to draw to you here is to the same problem that we encounter. Now, you must recall what does happening there you are searching this ando tree and well initially we had pouse it as a simple odd tree. But what were saying is that let say that you decide that you want to go to a go to the beach you ask your friend shell we go to the beach and then you are doing a depth first search. So, you try all these options 1 by 1 you say shell we go to this movie a or shell we go to this movie b or shell we go to this movie c and then try all combinations of this. So, for example, a beach and a and b then beach and a and e then beach and a and f and let say all fail. So, what is happening here in backward chaining is that you are doing this subgoel promulgation. And so the way prolog does is there it goes from top to down and left to right so given trees of goel outing and entertainment and dinner. It will first try outing try to find a value for that that say beach then entertainment.

And then try to find a value for that let say a and then dinner and try to find a value for that d and at that point let us say it fails and say that no this is naught a good birthday plan. So, it will back track and try e here so it is still be in this goal, but it would try a different value. So, whenever back tracking happens it happens in these same directions. So, when it is back tracking here it goes up 1 step when it is back tracking here it goes up. Well, where is back tracking here goes down 1 step and it is back tracking here it comes f 1 step. So, it tries the first option for dinner and the second option for dinner and the third option for dinner. And everything fails you do naught want to do repeat this same search again for a different option of b if you can figure out somehow that beach is the. So, we had mention the term dependency directed back tracking there. So, in some systems like in satisfaction system that is kind of done automatically the system keeps track of what is the dependency? But in a system like this in logic or the implementation of logic call prolog the language gives the user ability to control back tracking.

So, instead of writing a statement like this what you can do in prolog is to instead of this you write the statement as follows that a birthday plan. So, let me just use this b x y z f if outing plan then you use a special symbol cut then and entertainment plan any user

special symbol and then a dinner plan. So, instead of 3 subgoels you have added 2 extra subgoels these are which special subgoels is prolog allows you. And so those you have use it could know the this is the cut operator as it is called. And what it does is there is basically a device given to the user or the programmer. If you want to say to control back tracking and what it basically saying is that if you are going to back track from this to this side. Then do not try a new value for this really jump back to the original goel. And you have without going to the details I just want to point out that the cut feature of prolog is basically use to control this huge amount of back tracking that one doing in an unconstraint search.

So, what prolog does forward backward chaining is allows you to do is to allows it to ask existential queries you can ask some query about is Peter the grandfather of somebody or who is Janes grandfather or you could defined. And maybe that is a good exercise for you to define the ancestor of so when is an x in ancestor of y. Then you could ask a query whether you know Jane is an ancestor of someone or things like that and the system will search in your database. It may it will make inferences by jumping across such implication signs and eventually dig out an answer for you. This process is often called as which is more than what a database system give you database system gives you retrieval in an efficient faction. But it does not do deductions on the way prolog allows it you deductive retrieval and in that sense it is more powerful than the RDBMS system.

And in fact prolog is the complete programming language anything you can do in java. For example, you can do in prolog essentially and logic programming in general is also a complete programming paradigm essentially it had it is limitations. So, I had hinted about that when I said that there is only a limitation in the format in which you can write prolog program it works with only a subset of logical statements which are known as a which a which basically say that there can be only 1 consequent in any implication statement. So, it is not complete I refer you back to this problem of 3 blocks that we had mentioned in the last class, we are said that a is on b and b is on c and a is green and c is naught green. And we are said that is it true or does there exist 2 blocks x and y such that x is on y and x is green and y is not green in we cannot solve that problem using in the forward chaining or backward chaining.

So, in that sense forward and backward chaining are naught complete and remember that when you talk about logical systems we are interested in sound and complete system. So, everything that we are doing here is sound, because it is based on sound rules of inference it is only the queue completeness which is the question. So, in the next class which is the last class of our course, we will look at this use of resolution method which we saw for proportional logic and apply to in ((Refer Time: 52:09)) logic. And see that that particular problem which I will discuss again in the next class can be solved using resolution method essentially. So, on the way we will need to just have a quick look at the unification algorithm which is this algorithm which is used for matching a to terms like this which is necessary in the implicit quantifier form that we are using here. So, I will stop here and will meet in the next class for the last time in this course.