

Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 11
A First Course in Artificial Intelligence
Lecture – 86
Deduction as Search
Depth First Search on Goal Trees

(Refer Slide Time: 00:14)

Backward Chaining (Propositional Logic)



Alice likes mathematics (P) and she likes stories (Q). If she likes mathematics (P) she likes algebra (R). If she likes algebra (R) and likes physics (S) she will go to college (T). She does not like stories (Q) or she likes physics (S). She does not like chemistry (U) and history (V).

Then the given facts are, $(P \wedge Q), (P \supset R), ((R \wedge S) \supset T), (\sim Q \vee S), (\sim U \wedge \sim V)$

Equivalently

1. P
2. Q
3. $(P \supset R)$
4. $((R \wedge S) \supset T)$
5. $(Q \supset S)$
6. $\sim U$
7. $\sim V$

Goal Set

- $\{T\}$ Given goal
- $\{R, S\}$ from ④
- $\{P, S\}$ from ③
- $\{S\}$ matches ①
- $\{Q\}$ from 5
- $\{\}$ matches 2, success

"Is T true?"
 We answer this by backward chaining.



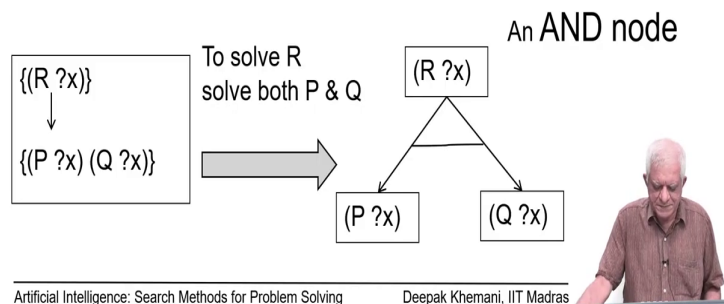
So, we are back. And we have been looking at the process of backward chaining or backward reasoning in logic. We just saw in the last session how a backward chaining proof in propositional logic can be found.

(Refer Slide Time: 00:40)

Backward Chaining with Conjunctive Antecedents

A goal $(R \text{ ?}x)$ and a rule $(\text{if } (\text{and } (P \text{ ?}x) (Q \text{ ?}x)) (R \text{ ?}x))$

A goal which matches the consequent of a rule reduces to the antecedents in the rule.



Now, we want to extend this notion to first order logic, and see how that works essentially. In particular, we are interested with conjunctive antecedents. This formula or this rule if you want to call it written in the list notation that we talked about (Refer Time: 00:57) introduce it says that for all x so remember that this there is an implicit for all x sitting somewhere there because x has a question mark before that.

For all x if $P \ x$ is true and if $Q \ x$ is true, then $R \ x$ is true. In this notation that we have in our logic notation, this would have been $P \ x$ and $Q \ x$ implies $R \ x$ that is it. But this list notation I keep introducing it because I am hoping that some of you would go and try to write programs for doing that then you do not have to worry about all those mathematical symbols and things like that.

Whereas, this here everything is a list, and it is kind of easy to process anyway, coming back to backward chaining. So, given this rule and given a goal and by now we have got used to idea of existential goals, we are saying is there something x which is an R or is $R(x)$ true for some x , or is the formula that exists in $\exists x R(x)$ true.

Now, that of course, matches this I have said x here, I could have very well said z or I could have instead of said a constant a , it does not matter. As long as we have the algorithm for unification, we know we have briefly talked about unification how two formulize the unified together; as long as we can find the substitution we can go through this process.

So, a goal in our case $R(x)$, it matches the consequent of a rule reduces to the antecedents in the rule, so that is a process that we also saw in the last end of the last class that we reduce goals to sub goals essentially. So, let us look at that process again. You want to show that $R(x)$ is true. And we have a rule which says that if $P(x)$ is true and $Q(x)$ is true then $R(x)$ is true essentially. How does backward chaining work? So, let us look at this process here you want to show that $R(x)$ is true.

To solve R , solve P and Q . Why, because we are doing backward chaining over the rule essentially. So, we replace goal $R(x)$ which is two goals $P(x)$ and $Q(x)$. So, you have to solve for $P(x)$ and you have to solve for $Q(x)$.

And now this is our old friend the AND node which we talked about when we are talking about goal trees, AND, OR trees. So, essentially this process can be captured. This rule can be represented as an AND node in some representation, in some graph representation.

(Refer Slide Time: 03:57)

Goal Trees



Consider the following KB in skolemized list notation, and the goal (niceToy ?z)

Rule1: (if (and (green ?x) (circle ?x)) (niceToy ?x))

Rule2: (if (and (red ?x) (square ?x)) (niceToy ?x))

(green A)

(green B)

(circle C)

(red C)

(red D)

(square D)

(circle E)

*base - green circle
base - red square*



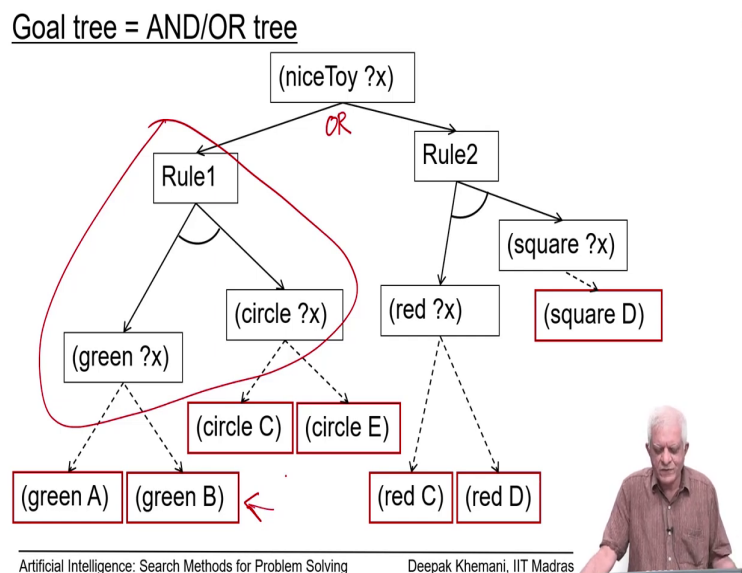
So, let us assume that we have some knowledge base also or database or knowledge base whatever you want to call it. There are some blocks in this world. So, there is A, B, C and D and E – 5 blocks are there. And we have some properties about some blocks. We can say that A is green. This is again in the list notation that we have been looking at. That B is green, C is a circle, C is red, D is red, D is a square, E is a circle. So, we have some statements about five blocks here. And we have two rules ok. There is some child who prefers these kind of toys.

And one rule says that for all x, remember again there is a implicit for all x sitting here. If x is green and x is a circle, then it is a nice toy let us say we are talking about the base of a block or something like that is one rule if it must be green and it must be circular essentially. But there is another rule which says that if it is red and if it is a square, then also it is a nice

square. So, there are two kinds of nice toys. One is a green circle, and the other one is a red square.

So, let us say we are talking about the base of some block or something like that. If either of this case happens then we are happy with the toy. And given this knowledge base, we have we have how many statements 3 plus 3 – 6, 7, 7 statements about 5 blocks. We are asking a question that is there are nice toy in our database or knowledge base essentially? How does that look? When we look at it from the backward chaining perspective?

(Refer Slide Time: 06:00)



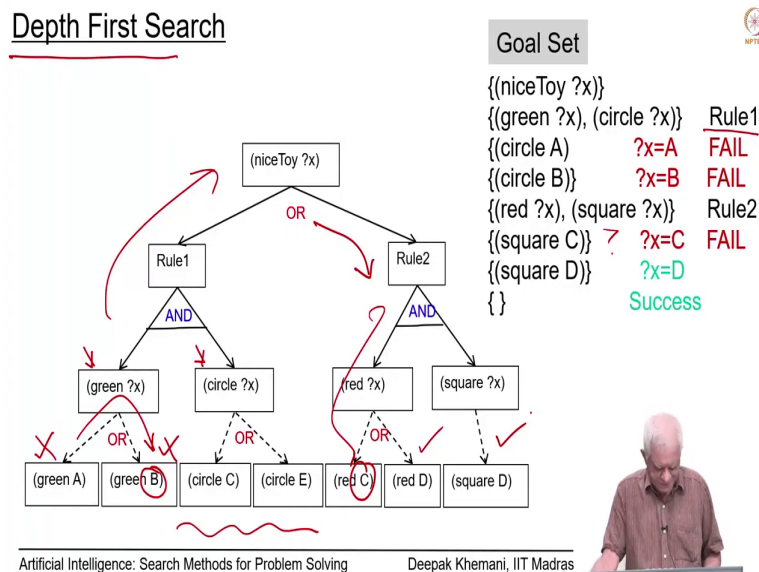
It looks like a goal tree or AND OR tree that we have studied so far essentially. So, an AND OR tree has two kinds of nodes. One is an OR node and the other one is an AND node. As you can see here there is the OR choice here. Either you can apply Rule1 or you can apply Rule2 essentially. But if you apply a Rule1, then Rule1 says this whole thing represents

Rule1. It says that look for a green object which is also a circle. So, this x and this x must match the same object.

And Rule2 says look for a red object which matches a square. And the red, triangle, red, the red outline rectangles at the bottom here all these they are the facts in some sense. Facts are equivalent to saying that that goal is solved.

There is no real difference between facts, and rules, they are all statements, and they are all accepted to be true. Rules allow you to change from facts to other facts, whereas facts are true by default. So, this is the goal tree that our algorithm will search essentially. And we have seen this process that the way that it goes about doing that is to maintain a set of goals.

(Refer Slide Time: 07:30)



So, what is the goal that we talked about that is there a nice toy in our knowledge base and this is the same goal tree which is drawn again here. We start off with the one goal in the goal set which is nice toy x, and it is reduced using Rule1 to 2 goals which says that look for a green x which is also a circle essentially. Then again as we said we always pick the first element in our goal set. We are going to look for something which is green essentially. And of course, this is found here which is green of A.

And our goal now reduces to saying that yes A is green, now go and check whether A is a circle or not essentially. But when you look at the circle part of our database, there are only two objects which are circles, one is C and one is E.

So, the goal circle A fails, and our algorithm says this does not work, so it goes back and tries the next goal. So, it is doing one of the first algorithms we studied which is depth first search. In fact, that first search is the way that a prologue search engine works essentially or a prolog interpreter works essentially.

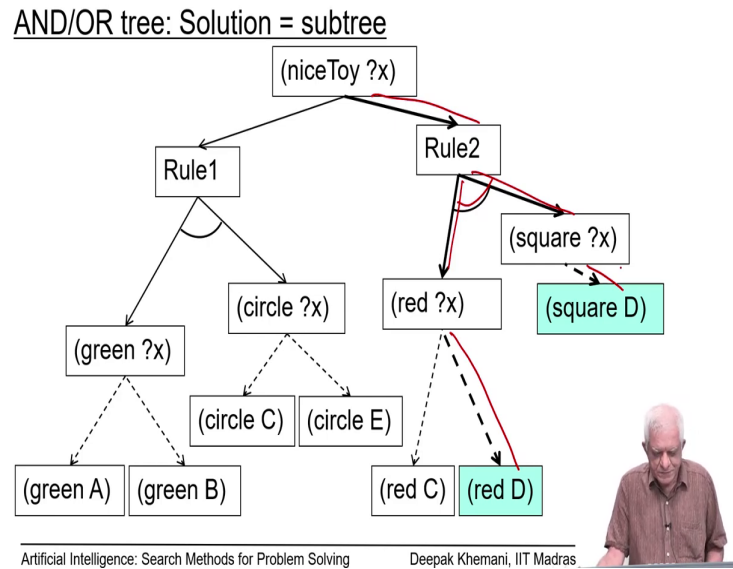
So, it goes and tries this says, yes, I have another green object here. Now, go and check whether that object which is B as you can see from here is B a circle? And then again this fails; B is not a circle. And then we find that there are no more green objects in our knowledge base. So, our program now backtracks all the way to this, and starts searching on the 2nd rule essentially. So, it says ok, let us look for something which is red and which is a square.

It comes down here and finds that, yes, C is red and says is C a square then, but no C is not a square, the only square we have is D. So, it backtracks. And again comes back, and says yes D is the circle, sorry D is a red, D is red. And says go and check whether D is a square, and yes, indeed D is a square. So, both the goals got solved. And we are left with an empty goal set which is a sign of success in backward chaining.

So, the important thing to observe here is that we constructed a goal tree or an AND OR tree, and we searched over it in a depth first fashion, and that is what prolog does with its program.

We will see an example of a prolog program though we will not look at too much of syntax in the details.

(Refer Slide Time: 10:42)



So, if you remember the solution of an AND OR tree is a subtree. And the subtree is shown here highlighted with thicker arrows. The rule 2 is the one which works. It has an AND node. So, you have to solve both of them. And red x works with red D, and square x also works with square D. So, this subtree is the solution to the goal tree essentially.

(Refer Slide Time: 11:20)

A Prolog KB (program)


$\xrightarrow{\text{CONSEQUENT}} \text{outingPlan}(X,Y,Z) :- \text{eveningPlan}(X), \text{moviePlan}(Y), \text{dinnerPlan}(Z).$
 $\xrightarrow{\text{ANTECEDENTS}} \text{eveningPlan}(X) :- \text{outing}(X), \text{likes}(\text{friend}, X).$
 $\text{moviePlan}(X) :- \text{movie}(X), \text{likes}(\text{friend}, X).$
 $\rightarrow \text{dinnerPlan}(X) :- \text{restaurant}(X), \text{likes}(\text{friend}, X).$

$\xrightarrow{\text{LOWER CASE}} \text{outing}(\text{mall}).$
 $\text{outing}(\text{beach}).$
 $\text{movie}(\text{theMatrix}).$
 $\text{movie}(\text{artificialIntelligence}).$
 $\text{movie}(\text{bhuvanShome}).$
 $\text{movie}(\text{sevenSamurai}).$
 $\text{restaurant}(\text{pizzaHut}).$
 $\text{restaurant}(\text{saravanaBhavan}).$
 $\text{likes}(\text{friend}, \text{beach}).$
 $\text{likes}(\text{friend}, \text{theMatrix}).$
 $\text{likes}(\text{friend}, \text{bhuvanShome}).$
 $\text{likes}(\text{friend}, \text{saravanaBhavan}).$

$(\text{if} (\text{and} (\text{restaurant } ?x) (\text{likes friend } ?x)) (\text{dinnerPlan } ?x))$
 $\forall x [(r(x) \wedge L(f, x)) \supset d(x)]$
 $(A_1 \wedge A_2 \wedge \dots \wedge A_n) \supset C$
HORN CLAUSE

UPPER CASE = VARIABLE

RULES



So, as I have been saying prolog is essentially first order logic, and it does backward chaining. And it does backward chaining in a depth first fashion. So, if you take sentences in logic, prolog is restricted to a subset of logic sentences, it does not work with full first order logic. So, if you take an example this rule that we are talking about, now this is an example that we had looked at when we are talking about AND OR trees as well.

So, you know already that the solution to this problem is a subset of the goal tree. But when you look at it from the perspective of prolog the same knowledge base is expressed as rules and facts in the program. Everything that you see here is a fact and a fact in prolog notation is simply a predicate followed by a full stop. And it says that yes this is a fact which means it is true essentially.

That Mall is an outing, that beach is an outing, that theMatrix is a movie, artificial Intelligence is a movie, you know bhuvanShome is a film, sevenSamurai is a film, pizzaHut is a restaurant, saravanaBhavan is a restaurant. And some facts about which friend, what your friend likes? The friend likes the beach, the Matrix, the bhuvanShomes, saravanaBhavan and so on.

The rest of the things are expressed as rules here. So, these are rules. And if you look at the fourth rule, prolog writes the consequent on the left hand side. So, the consequent is here and the antecedents are here.

Prolog uses a convention that upper case is a variable and lower case is a constant, so that is why you can see that all these things are lower case. We said that we will use a question mark to remark a variable. Prolog says no let us use uppercases uppercase symbols for variables, and lowercase ones for constants. It is just a way of distinguishing between variables and constants.

Prolog inverts a rule and writes a consequent on the left hand side and the antecedence on the right hand side. So, the way to need it is the if version of a rule essentially we are not looked at logic too much in detail, but essentially we are reading it as saying that the consequent is true if the antecedents are true. So, you are moving from left to right and reading it as saying that consequent is true if the antecedents are true.

So, if you look at the fourth rule here which is the dinner plan rule. It says X is a dinner plan, remember X is a variable. If X is a restaurant and your friend likes X. If you were to write this in logic, you would write it as a statement like this for all x restaurant x, and friend and likes friend x implies dinner plan.

So, let me write it again here. For all x, let me just use r for restaurant, and likes L for likes, f for friend and x for the variable implies dinner plan x. Remember in first order logic variables, we had some conventions that x, y, z are variables and things like that. And of course, in this case it is a quantified variable.

Now, I said that prolog works with a subset of first order logic and the only subset it works with is where the facts are of course simple atomic formulas, but rules are only of the kind that you have an antecedent and another antecedent, and some n antecedents are there joined together with AND, and they imply one consequent C. This particular form restriction on sentences in first order logic is called Horn clause.

So, prolog uses Horn clause logic on Alfred Horn was the one of the persons who worked on this aspect of logic, and so they are named after him. So, prolog has two kinds of statements either they have Horn clauses which are statements like this as you can see four statements on the top here, or they are facts which are also called horn clauses, because the left hand side can be empty essentially.

The consequence can be empty, so which is also rational for writing the consequent followed by the antecedent because you are going down this list here, and trying to ask this question, is this true, is this true, is this true and so on essentially.

So, the way prolog does that first search is that it starts by asking a question that is there an outing plan x, y, z, where you know x is an evening plan, and so on and so forth. You must remember that each of these statements are universally quantified.

So, here for example, it says for all x, and here it says for all x. So, if we have x here and if we have x here, prolog is not going to get confused. It simply says that it is a new variable in a new statement ok. So, what are the three rules?

The three rule says that if you have an evening plan, if you have a movie plan, and if you have a dinner plan, then you have an outing plan which prolog needs are saying that you have an outing plan, if you have an evening plan and if you have a movie plan and if you have a dinner plan, then for each of those things there are again rules.

The second rule says if you have an evening plan that sorry the second rule says that you have an evening plan if you have an outing x and your friend likes x. Likewise you have a movie

plan if there is a movie which your friend likes and you have a dinner plan if there is a restaurant which your friend likes.

So, you are always asking about the left hand side and moving towards the right hand side that is the process of backward chaining. Remember you are going from consequence to antecedence in backward chaining and that is what prolog does.

So, it just it tries from top to bottom, and it searches from the in the sub goals it takes a left one for synthesizing. And this process this strategy of searching which we called lexical when we were talking about conflict resolution strategies in rule based systems.

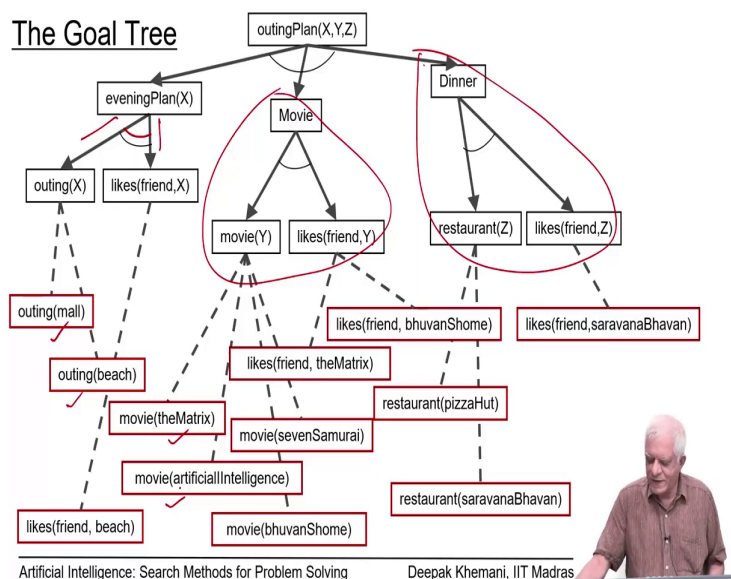
It is lexical in the sense that the order in which the user writes the program is the order in which the search engine will look at that which translates to depth first search essentially. So, in this particular example, if you are querying that if there is a outing plan, then the sub query is there an evening plan that is a first sub query prolog takes up the first sub query and goes into that.

Then it says that you have an evening plan if you have an outing x and if your friends likes x, then it goes to the outing what is the first outing, it looks for is mall then it says whether your friends like the mall and so on.

You can imagine that there is the same kind of a subtree that we saw in the previous on this thing. A similar subtree exist for the program to determine whether you have an outing, and essentially prolog does depth first search here.

So, prolog does reasoning in first order logic, or to be more precise it does reasoning with a subset of first order logic which is the Horn clause subset. And it does backward chaining. And it implements backward chaining the search strategy is depth first search essentially in the goal tree.

(Refer Slide Time: 21:00)



So, let us again see this whole thing here this is a search tree that you would construct it is an AND OR tree very similar to what we saw for the toy example. You have an outing plan you must have three things – an evening plan, a movie plan, and a dinner plan. You have a movie, evening plan, if you have two things that if you have an outing and if you have if your friends like the outing.

And likewise you should have a movie and your friends should like the film, and you should have a dinner plan, and your friends should like the dinner and everything below which is in red rectangles are the facts that are given to us. That mall is an outing, beach is an outing, Matrix is a film, artificial intelligence is a film and so on all that is those are facts that are given to that.

Can you choose the right combination of these things to construct an outing plan is a question? And prolog will look at this three and search it in a depth first fashion. And we have seen that happens by a process of adding maintaining a goal set. So, the top level goal set would be the outing plan, is there are outing plan then we will keep adding new things to that essentially.

(Refer Slide Time: 22:23)

Backward Chaining: Depth First Search

<u>{outingPlan(X,Y,Z)}</u>	theta = {}
<u>{eveningPlan(X), moviePlan(Y), dinnerPlan(Z)}</u>	theta = {}
<u>{outing(X), likes(friend, X), moviePlan(Y), dinnerPlan(Z)}</u>	theta = {}
<u>{likes(friend, mall), moviePlan(Y), dinnerPlan(Z)}</u>	theta = {X=mall}
<u>"fail", moviePlan(Y), dinnerPlan(Z)}</u>	theta = {X=mall}
{outing(X), likes(friend, X), moviePlan(Y), dinnerPlan(Z)}	theta = {} backtrack
<u>{likes(friend, beach), moviePlan(Y), dinnerPlan(Z)}</u>	theta = {X=beach}
<u>{moviePlan(Y), dinnerPlan(Z)}</u>	theta = {X=beach}
{movie(Y), likes(friend, Y), dinnerPlan(Z)}	theta = {X=beach}
{likes(friend, theMatrix), dinnerPlan(Z)}	theta = {X=beach, Y=theMatrix}
{dinnerPlan(Z)}	theta = {X=beach, Y=theMatrix}
{restaurant(Z), likes(friend,Z)}	theta = {X=beach, Y=theMatrix}
{likes(friend, pizzaHut)}	theta = {X=beach, Y=theMatrix, Z=pizzaHu}
"fail"	theta = {X=beach, Y=theMatrix, Z=pizzaH}
{restaurant(Z), likes(friend,Z)}	theta = {X=beach, Y=theMatrix} backtrack
{likes(friend, saravanaBhavan)}	theta = {X=beach, Y=theMatrix, Z= saravanaBhavan }
{}	theta = {X=beach, Y=theMatrix, Z= saravanaBhavan }

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, I will ask you to try and go over this in a depth first fashion and verify that this is the process that is going to be happening here. So, theta is what? Theta is the unifier or the substitution which will make something true. So, we have three variables here X, Y and Z, and somehow you have to unify them with three these things. The fact that we use X in many places should not confuse you too much I think. And prolog does not get confused either.

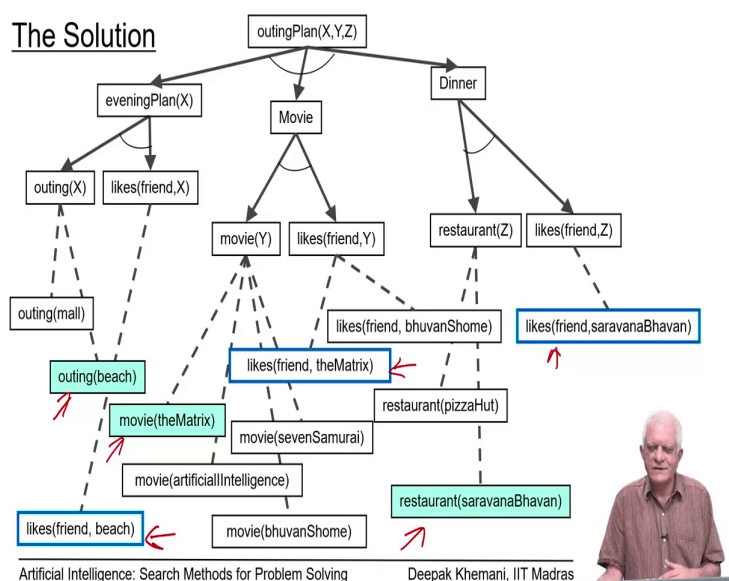
So, we start off with the top level goal which is says that the outing plan is there, and so far there is nothing there. And it says to solve the outing plan you go to the second step which says that there are three sub goals – evening plan, movie plan, and dinner plan, so far nothing. Then it says that for an evening plan, the first sub goal here you reduce it to two sub goals which says that you must have an outing and your friend must like the outing.

So far still again no substitution is there. Then you find that in the set of outings that we had, we had something called mall. So, X equal to mall that is an outing. And now you are asking the question whether your friend likes the mall. It turns out that thing fails. So, prolog will backtrack, and try the next outing. The next outing turns out to be the beach, so there you are X equal to beach.

And then you ask the question does your friend like the beach, and it turns out that your friend does like the beach, so that goal of evening plan is now solved essentially. So, one part you have solved.

Now, you are left with the two remaining sub goals which is a movie plan and a dinner plan. And in a similar fashion prolog will search in depth first fashion over the search tree and come up with the solution which by now you know is a subtree of the goal tree.

(Refer Slide Time: 24:22)



So, this particular solution is that the three cyan colored boxes are the three components of the solution which says that the outing is the beach, the movie is theMatrix and the restaurant is saravanaBhavan. And the three other blocks here with cyan edges simply say that all these three things your friend is agreeable to go along with essentially. So, this very brief introduction to prolog the language. So, just like ops five was a language which works with forward reasoning.

Prolog is also a language programming language which works with logic as a representation and backward chaining as the process of solving problems. In particular, it constructs a AND OR search tree like the one that you can see here, and searches it in a depth first fashion.

As we had said sometime when we were talking about conflict resolution because we have frozen the search strategies to be a blind search strategy or a depth first strategy, the owners of

writing an efficient program now shifts more to the user. So, the user has to choose rules carefully. And not only that the rules have to be ordered carefully.

So, especially if you are writing a recursive program, the base clause must always come first and then the recursive clause because prologue will first check the base clause and then go to the recursive clause if the base clause fails.

If you had written the recursive clause first and the base clause afterwards, it could just get locked into an infinite loop in the recursive clause itself essentially. You know keep saying that this is true, if this is true, that is true, the next thing is true and so on, never get never reach the base clause.

So, this kind of concludes our study of deduction as search. So, to remember that deduction is the task of finding out whether a given statement α is true once you have a knowledge base given to us. We cannot talk about truth values because you know they are not represented in our first order logic.

So, the program does not know what it is doing, it is just manipulating symbols. There are efforts to create representations of meaning. So, a first order sentence, first order logic sentence can be interpreted over a domain, and the domain has relations on it essentially.

So, first order logic is closely tied to relations in a domain. So, you can try and have a formal notion of the semantics of first order sentences, but that is all work in kind of progress. And there is much that needs to be done in the case of semantics. Semantics of first order logic is not so hard, all you have to do is to construct a domain and an interpretation.

Semantics of other logics is more complex essentially, in particular we have been looking at something called epistemic logics which is one of the logics that we had mentioned here in which there is uncertainty and that uncertainty the semantics of the uncertainty is captured by something called a possible world semantics which are also known as kripke structures.

So, you imagine many possible worlds. And if you do not know which of those worlds you are in then or if you do not know what is true in those possible worlds, then some things you do not know and there is uncertainty about it. And this life gets more complicated if there are other people also involved other agents also involved essentially, then you do not know whether the other agent knows something or not essentially. So, that leads to interesting possibilities.

But that is all work to be done in the future. We will stop here for forward chaining and backward chaining. And we have one more session in logic which will talk about the limitations of these two approaches to search.

So, we will do that after a short break.