Artificial Intelligence:

Prolog: Depth First search and Efficiency Issues

Prof. Deepak Khemani

Department of Computer Science and Engineering

Indian Institute of Technology, Madras

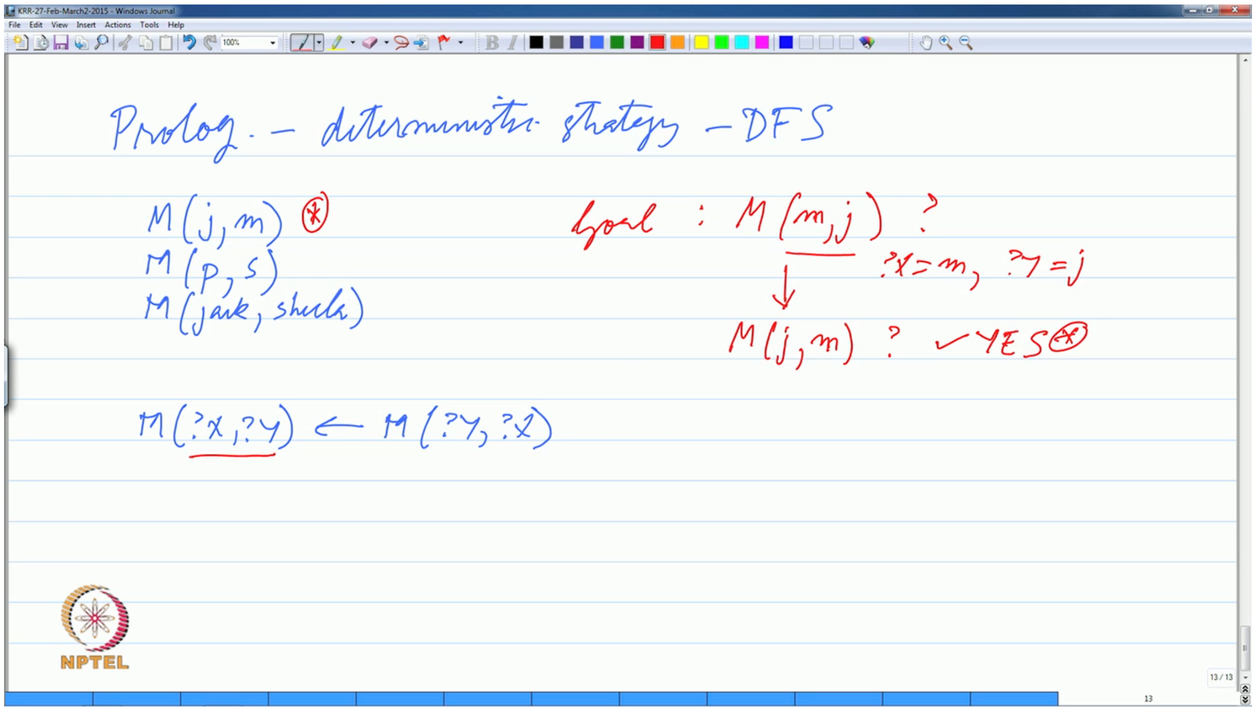
Module – 06

Lecture - 04

Okay so we are looking at Prolog and it has a deterministic strategy which is depth first search over the goal G. now there are certain things which are kind of pitfalls in a language like prolog. Supposing I gave you a database of people, lets see people who are married to each other and lets say M stands for married and lets say John is married to Mary and lets say Peter is married to Sally. Lets say Jack is married to Sheela, anyway some names. So we have such knowledge base and then we have one rule of inference which says if married x y then married y x. now obviously this is something that we take without blinking an eye in logical framework because all you are saying is that marriage is a symmetric relation. If x is married to y then y is also married to x. but stating that in a logic program or rather stating that in a prolog program is sort of fraud pit danger. Now imagine what would happen. Lets say for argument sake that we have only these three statements, it’s a very small world we are talking about there are only 6 people and these three statements that’s given to us. And if I gave a goal as follows Is Mary married to John?

Then you can see that it does the depth first search. Remember we said top to down and left to right. There is no left to right here but atleast there is top to down. So it will first try the first clause which says that John is married to Mary, that will not match that. The second one will not match the third one will not match. The fourth one will match. This m j will match x y we will say that x is equal to m and y is equal to j. and this will reduce to a new goal which is Is John married to Mary? This is backward chaining simple backward chaining here. So the question of is Mary married to John we have now reduced to a subgoal is John married to Mary? Because we have this symmetric relationship as a part of this knowledge base. And this will be answered yes because it will match this fact and our program will terminate.

(Refer Slide Time: 3:58)



Prolog. – deterministic strategy – DFS

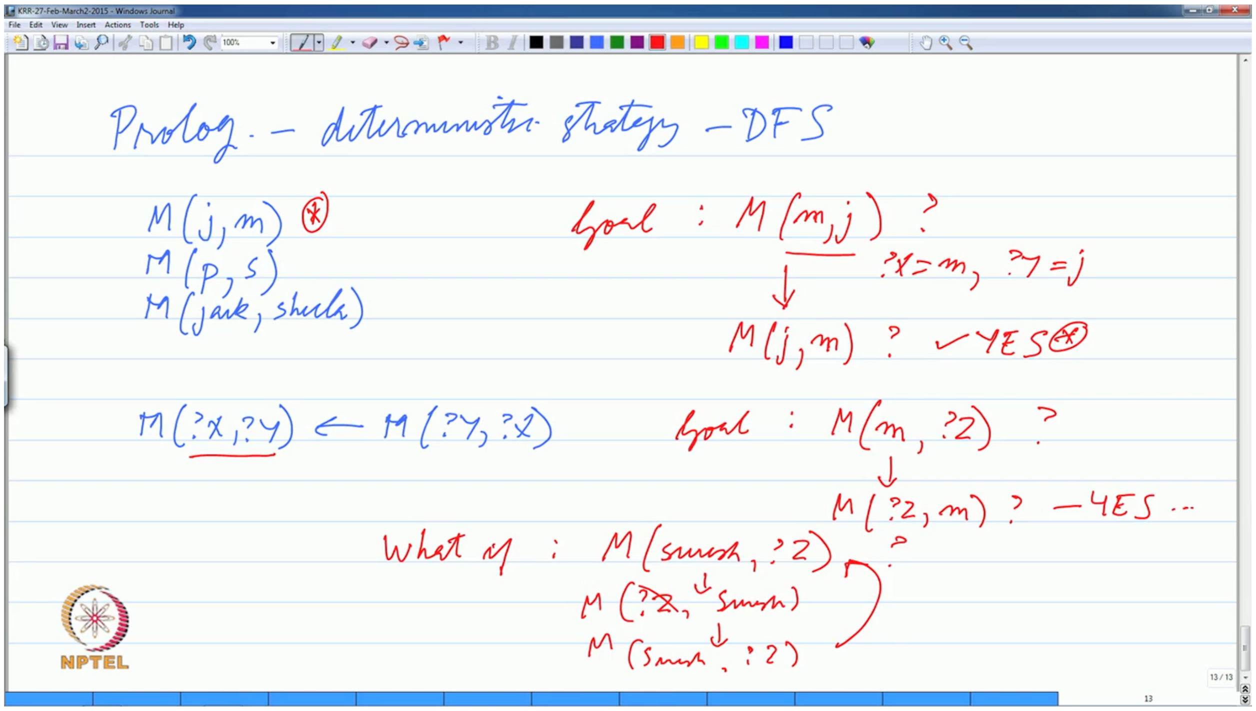
$M(j, m)$ ⊛
$M(p, s)$
$M(jack, sheila)$

$M(?x, ?y) \leftarrow M(?y, ?x)$

Goal : $M(m, j)$?
$?x = m,\ ?y = j$
$\downarrow$
$M(j, m)$? ✓ YES ⊛

Likewise if I had given a goal as follows Is Mary married to anyone? Then a similar thing would happen, it would translate to Is anyone married to Mary because it tries the first second and third clause and then it comes to the fourth one which is a recursive clause and that would translate it into z and again the answer would be yes. But what if I give a goal of this kind. Some person about whom we have no information what will happen with this goal? The first three clauses will never match and it will keep going through the fourth clause again and again and again so it will replace this by saying is z married to Suresh and again it will match the same clause and replace it with Suresh married to z and you can see that it has basically gone into a loop. It will just go into a loop and It will never come out of the loop

(Refer Slide Time: 5:40)



Prolog - deterministic strategy - DFS

$M(j, m)$ $\circledast$

$M(p, s)$

$M(jack, sheela)$

$M(\underline{?x, ?y}) \leftarrow M(?y, ?x)$

Goal : $M(\underline{m, j})$?

$?x = m, \; ?y = j$

$\downarrow$

$M(j, m)$? ✓ YES $\circledast$

Goal : $M(m, ?z)$?

$\downarrow$

$M(?z, m)$? — YES ...

What if : $M(suresh, ?z)$?

$M(?z, suresh)$

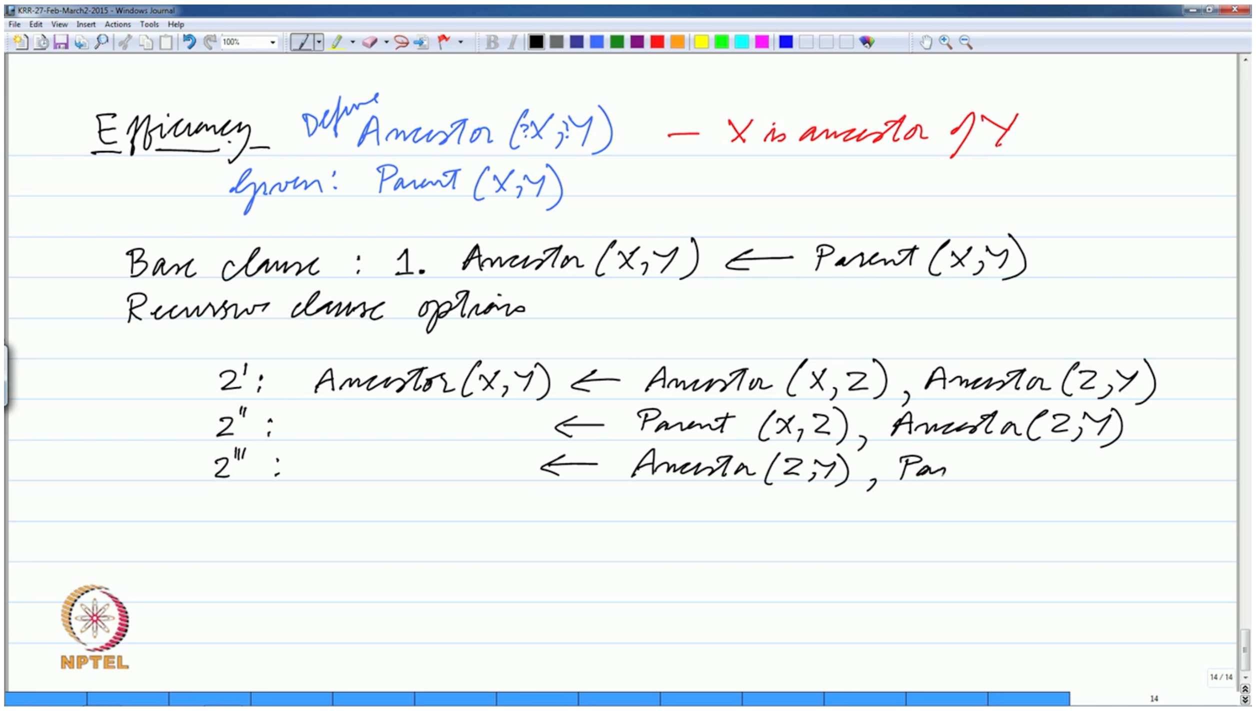
$M(suresh, ?z)$

So that’s a danger that is there in a language like prolog that you have to be careful that you don’t write clauses that end up in a loop. Now this is a simple clause about marriage but you could have a set of clauses for example you could have a clause which says that Child x y if Parent y x. and then you could say Parent x y if Child y x. you can see that these two clauses between them have the danger that you could go into an infinite loop that you could for example if you ask a question Is Suresh anybody’s parent? You know then from parent it will go to Is is true that Parent Suresh somebody. It will translate to the question is it true that Suresh is somebody’s child and then this child will go to parent and parent will go to child. Between the two of them they will keep cycling indefinitely. So one has to be quite careful with that essentially. But nevertheless let us look at some things which will address this question of efficiency. Assuming that we have avoided writing clauses which will go into infinite loop there is still question of efficiency. So lets look at this example and try to analyze this little bit.

So we want to define Ancestor X Y I am not writing question marks here but ideally you should, well lets write them. And the meaning of this is X is ancestor of Y. and remember we had looked at this problem a little bit earlier and want to basically define this. And what are we given. We are given let us say just one predicate which is Parent. So given the Parent relation we want to define Ancestor relation. We had looked at this problem a little bit. Lets look at what are the options available to us. So one is the base clause which simply says that X is ancestor of Y if X is parent of y. so we start off by defining that parents are ancestors. And then we have define grandfather or grandparent earlier, parent of parent is grandparent. But now we want to define ancestor.

Now recursive clauses and we want to look at options. Let me state 3 options here or lets a couple of more than 3. So one is, so when I say options so lets write it like this. Lets call this 1 and lets call this 2 prime. So one way of defining this is that x is the ancestor of y if x is a ancestor of z and z is the ancestor of y. clearly it's a true statement. Lets look at another option. So I will just write the right hand sides here. Parent x z this one says that x is an ancestor of y if x is a parent of z and z is an ancestor of y. so it is also a true statement. Then I can think of a third option which is I will reverse the order of these statements.

(Refer Slide Time: 11:18)


Efficiency Define Ancestor (?X ; ?Y) — X is ancestor of Y

Given: Parent (X,Y)

Base clause : 1. Ancestor (X,Y) ← Parent (X,Y)

Recursive clause options

2': Ancestor (X,Y) ← Ancestor (X,Z), Ancestor (Z,Y)

2'' : ← Parent (X,Z), Ancestor (Z,Y)

2''' : ← Ancestor (Z,Y), Par

Logically ofcourse it should not matter because there is just an and connected between these two subgoals and we know that and is a commutative relation. So I can say Ancestor z y first and then Parent x z or I can say Parent x z first and then Ancestor z y. and lets just separate these, fourth one says Parent z y x y. it looks a little bit like the second one except that the variables are different. And lastly we could exchange these two and that would give us the fifth one. So lets say we have these five options of defining the second of the two clauses. We will need two clauses to define the ancestor relationship. One is the base clause which says if you are a parent you are an ancestor. But if you have removed more than a parent then you will have to use one of these clauses. My question is which is them is good definition and which of them is not a good definition.

Any lets look at one of them. So the first clause is always there. So let me just use A here. Ancestor x y if Parent x z and Ancestor z y. what is this doing? So we are asking a question is this x related to this y by ancestor relationship. And ofcourse what we are given only is parent relationships. So we might have for example x may be some parent of lets say 3 people. Lets call them Z1 Z2 Z3 and essentially what this definition is saying that look for a child of X and see recursively if that child is ancestor of Y. in this example since there are three children then the Prolog remember it does depth first search left to right so it will first try with Z1 then it will try with Z2 and then it will try with Z3.

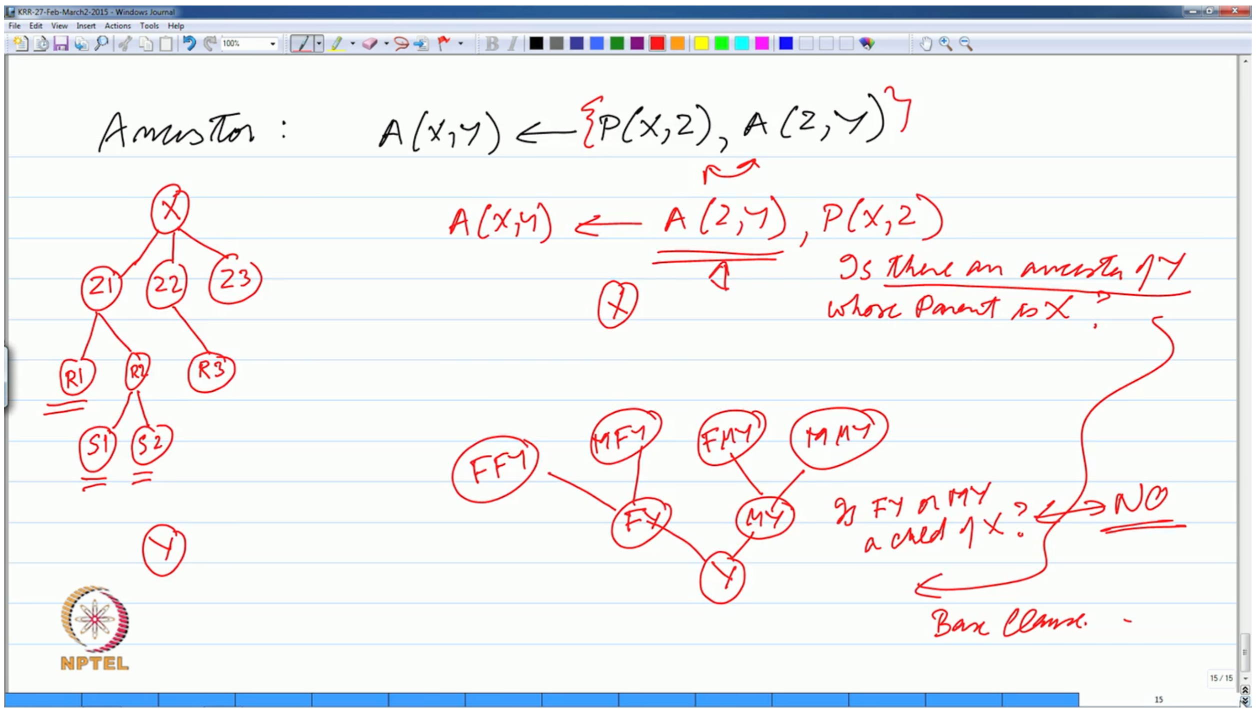
So it will start searching from the ancestor down towards the child. What will happen if we reverse the order here? So observe that for example if it is doing Z1 then it will lets say look for 2 children and then it will lets say R1 and R2. And lets say R2 has no children so it will backtrack from R2 it will come to, from R1 so lets say it has no children so it will come to R2 and then it will try this so lets say S1 and S2. And lets say they have no children so it will backtrack all the way and try a child of Z2. Lets say Z2 has only one child which is lets call it R3 then it will look at whether R3. So you must try to imagine whats happening here. Everytime we are going to put this in a goalset. We will tackle the first goal in the goalset then try the second one or if necessary we will backtrack. What if I had reversed the order?

How will this program work? So what this will now do is it will address the second goal first which means you noticed its focusing on y then it will say is there an ancestor of z, sorry is there an ancestor of y who is a child of x? that's the question it is asking. Let me write it here. Is there an ancestor of y whose parent is x. that's the question it is asking. So the first part is there an ancestor of y which is this underlined part here. The recursive clause will try lets say the father of y and the mother of y, not the recursive clause, base clause. So you must have to imagine whats happening here. We are trying to show whether there is an ancestor of y who we are calling z. so either father of y will be z or mother of y will be z depending on what we are doing. It's a variable anyway. And then we are asking a question is that a child of x. so x is sitting somewhere here and you can see that atleast in the diagram I have drawn they are not children of z actually so you will backtrack.

So the question that we will reduce this to is FY or MY where FY is father of y and MY is mother of y a child of x and the answer to that is No. so we will backtrack from the base clause and go to the recursive clause. So recursive clause will say okay this ancestor is not a child of x is there some other ancestor who is a child of x? what are the other ancestors other ancestors are father of father of y and mother of father of y, parents of father of y. likewise father of mother of y and the mother of father of y. ofcourse prolog will not do this in the breadth first fashion I am drawing this in. it will do it in depth first fashion. But you can imagine whats really happening essentially.

So its trying to find some ancestor of y which is a child of x. whereas the first one is focusing on the child of x and trying to see whether that's an ancestor of y. so you can see that simply by changing the order in which you are writing clauses you have changed the way that the program will search. The program is searching its doing depth first search over the space of this relation trying to establish the fact that there is a path from x to y. that's all its trying to do. And its trying to do this in a recursive fashion. But the behavior of the program is so critically dependent upon the order in which you write these statements.

(Refer Slide Time 20:13)

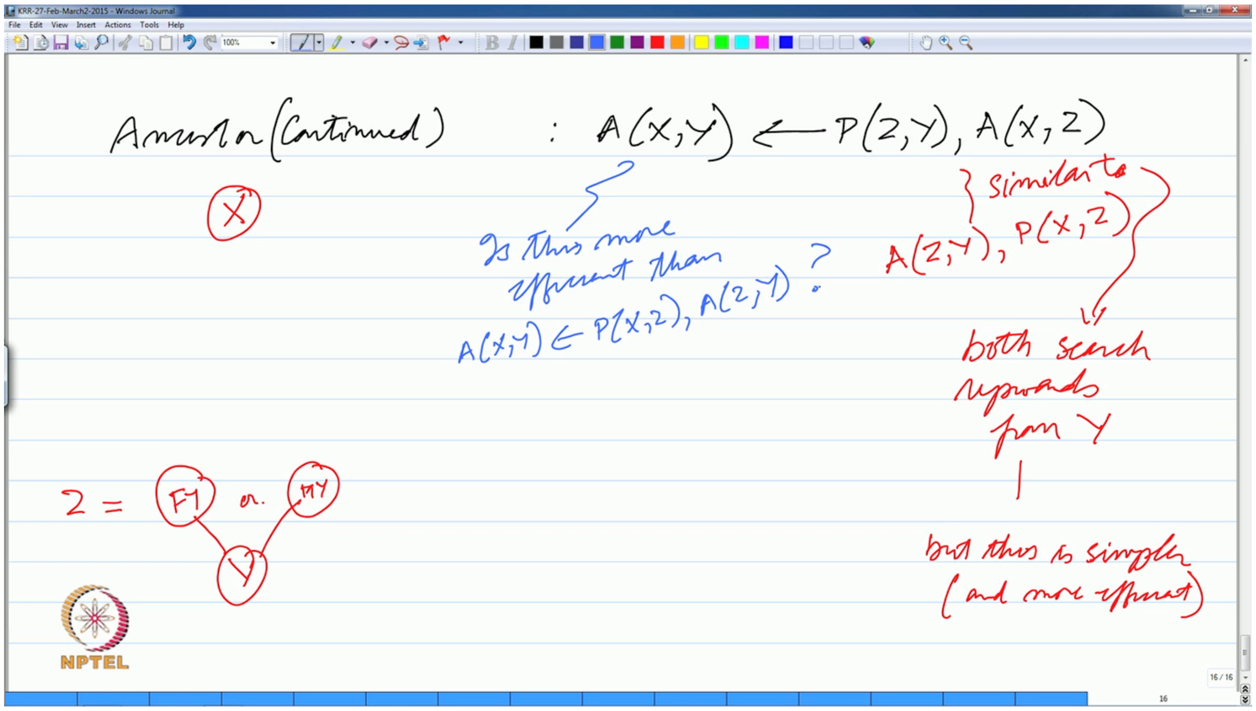

Ancestor :
A(X,Y) ← {P(X,Z), A(Z,Y)}
A(X,Y) ← A(Z,Y), P(X,Z)
Is there an ancestor of Y whose Parent is X ?
X
Z1
Z2
Z3
R1
R2
R3
S1
S2
Y
FFY
MFY
FMY
MMY
FY
MY
Is FY or MY a child of X ? ↔ NO
Base Clause.
NPTEL

Lets look at one more of this situation. So lets focus on this one which is also talking about looking at the parent clause. So lets say we have between us figured out and you should go back and think about this a little bit that having the parent clause first kind of narrows down the search space a little but because you are only looking at parents and then trying to see whether they satisfy the relation or not. so instead of looking for a parent relation with X the other option is to look for parent relation with Y and ask if Z is a parent of Y then is X an ancestor of Z. and you will observe that this is similar to the second of the two options we saw in the last one which is if ancestor Z Y and parent X Z. why did I say that it is similar? Both search upwards from Y. because this is first looking for a parent for Y and then trying to see whether that's an ancestor of Z whether X is an ancestor of Z. but they are not identical. The searches that the two will do is not identical but they are similar in the sense that they are starting the search from below but I will argue and you should think about this. But this is atleast simpler and in fact it is more efficient. So what is this doing? This is saying that if you are asking a question that if X is an ancestor of Y you are explicitly now asking whether Z there is a Z who is the parent of Y and who is the ancestor of Z.

And we saw when we were doing last example that there are two options. Z could take one of these values either the father of Y or mother of Y. so Z could be either that or this. So lets look at this earlier one. Earlier one said that look for a Z who is a child of X and as you can see on the left hand side of this figure. You start searching from the top and you will eventually try to find a path from X to Y in a depth first fashion. If you look for as we are doing here Z which is a parent of Y then you will start searching from the bottom and try to find a path which goes from Y to Z.

So my question to you is which is more efficient? Is the one we let me state the question. Is this more efficient than A X Y parent X Z A Z Y, that’s the question

(Refer Slide Time: 24:48)



Ancestor (Continued) : $A(X,Y) \leftarrow P(Z,Y), A(X,Z)$

Is this more efficient than $A(X,Y) \leftarrow P(X,Z), A(Z,Y)$?

similar to $A(Z,Y), P(X,Z)$

both search upwards from Y

but this is simpler (and more efficient)

$Z = \boxed{FY}$ or $\boxed{MY}$

Both of these clauses use parent as the first clause and ancestor as the second clause but one of them looks at the parent relation of the proposed ancestor which is X and the second one talks about parent of the proposed descendent which is Y. is it better to start searching from X or is it better to start searching from Y that’s the question.

So it depends on which country you live in which is actually a correct answer. So the answer to that is it depends on branches. As in if you live in a country where there is very strong population control then may be you should start searching from ancestor to descendent otherwise its better to search from descendent to ancestor. Okay I think I will stop with this example here and in the next class we will continue to look at what are the things which can make a prolog program more efficient.