The Cut Operator in Prolog

Prof. Deepak Khemani

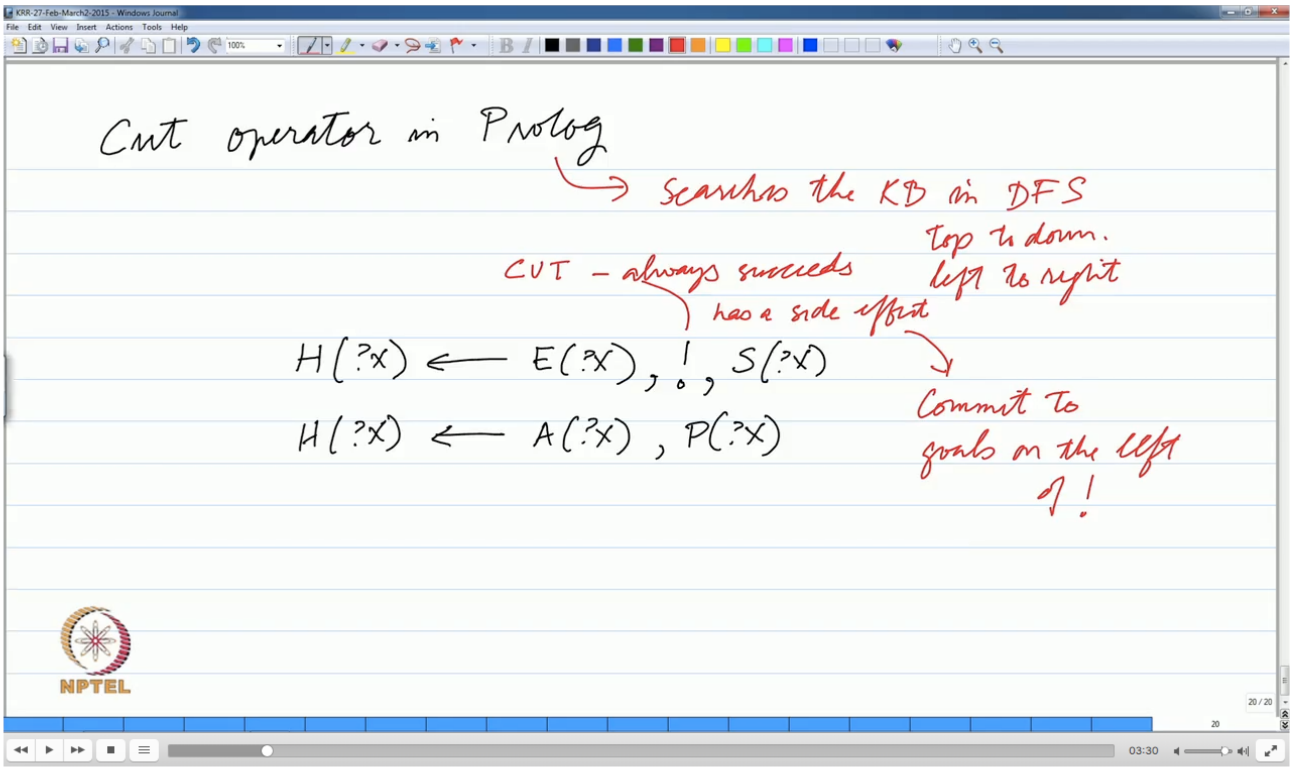
Department of Computer Science and Engineering

Indian Institute of Technology, Madras

Module – 07

Lecture - 37

We are looking at logic programming and we are looking at prolog and in particular we are looking at cut operator in Prolog. Now what the cut operator does is the kind of exploits the fact that the prolog search is the knowledge base in depth first fashion. By this we mean in top to down in the clauses we have written. If we have set of goals to solve, then left to right. So typically a prolog cut operator is used to improve the efficiency. So we may a program like this, if we say if you remember, some. We have tried to define happiness at some time. X is happy , if x has eaten well. E stand for eaten well. Lets put a cut operator there, If x has slept well. The other clause we have that x is happy if x has finished his assignment and has played something in the evening and



Cut operator in Prolog

→ Searches the KB in DFS

top to down.

left to right

CUT – always succeeds

has a side effect

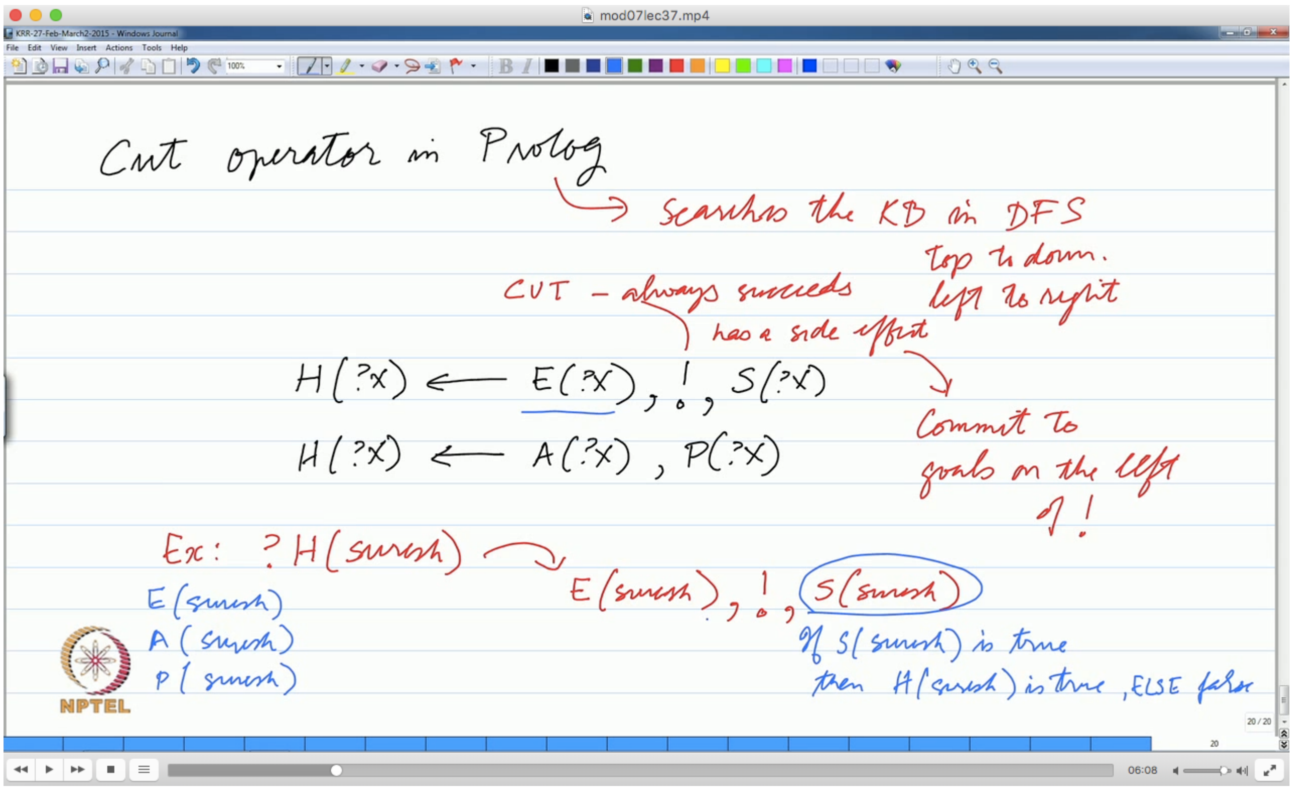
$H(?x) \leftarrow E(?x), !, S(?x)$

$H(?x) \leftarrow A(?x), P(?x)$

→ Commit to goals on the left of !

something like that. If we have these two sentences, only difference is that this time we are carting the first sentence, the cut operator which is denoted by this !. The first thing about cut is that it always succeeds as a predicate .It always become true. When it becomes true, it has a side effect. And what is the side effect, that you commits to goals on the left of this. [Time 03:22]

What does it mean in this context, For example, if we have a query it say. Lets says put a question mark to denote a query, just like prolog does. Lets say we are talking about an individual. Lets say suresh [Time 0357]. We are asking whether suresh is happy, whether this goal is true and then it calls these three goals. Has suresh eaten well? Supposing this goal becomes true. Then it will


mod07lec37.mp4
KRR-27-Feb-March2-2015 - Windows Journal
File Edit View Insert Actions Tools Help
NPTEL
20 / 20
06:08

Cut operator in Prolog

→ Searches the KB in DFS

Top to down.

left to right

CUT – always succeeds

has a side effect → Commit to goals on the left of !

$H(?x) \leftarrow \underline{E(?x)}, !, S(?x)$

$H(?x) \leftarrow A(?x), P(?x)$

Ex: $?H(suresh) \rightarrow E(suresh), !, \boxed{S(suresh)}$

E(suresh)

A(suresh)

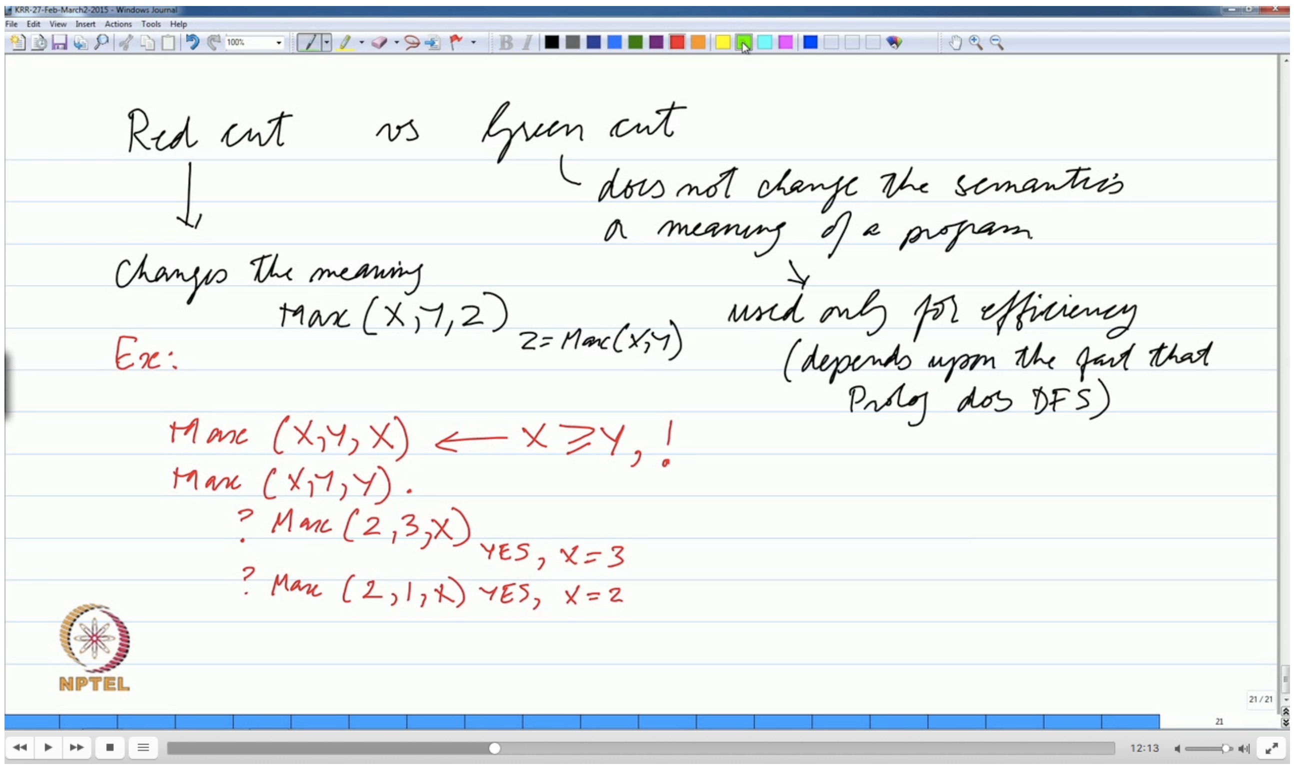
P(suresh)

If S(suresh) is true then H(suresh) is true, ELSE false

move to the cut , which will always becomes true. Then it moves to slept well. So once you have crossed the cut, there is no other way to show that suresh is happy. only if you can show that suresh has slept well. [Time 0458] At this point, If suresh is true, then happy suresh is true else it is false. Even if we had in our knowledge base these facts that eaten suresh and finish assignment suresh and late suresh. Even if we had these three facts in the knowledge base and asks the question that is suresh happy. Because the fact that eaten suresh becomes true, you get the program commit to showing, only way to show happy suresh is to now show that the slept suresh is true. If this is not true you return false. You do not try the next way to show happy suresh is true. This is the impact of cut essentially.

Lets look at some examples. It might have occurred to you that in this program this is not what you intended. You were trying to show that there were two ways that suresh could be happy. Either it could be that he has eaten well and slept well or it could be that he finish the assignment and played the game. This clause is saying that if he has eaten well, then you can not use the other clause in which you have said that you have finished the assignment and slept well essentially. One often distinguish between two kind of cuts. One is called a Red cut and other is called a green cut. Now a green cut does not change the semantics or meaning of a program. where as a red cut changes the meaning. So ideally ou should use only green cut. It is used only for efficiency purposes. And basically it is dependent on the search strategy. Lets look at the example of slight variation in the program One program uses the green cut and other program uses the red cut.
Lets say you want to write a program to write maximum of two numbers. You have been given two numbers and you want to decide which one of them is maximum. And the schema is that x y z means z equal to max x y. This is what you want to compute. You are defining a predicate of three arguments where third argument should match maximum of two. So for example if you give max three five five it will return true. If you give max three five seven then it will return false and so on.

Only when z is equal to either of x and y but it should be maximum of these two elements, then it return true. So we are defining the predicate to return max. So one way to do is and it employ a red cut is as follows: I could say max x y x. Remember that these are variables. If x is greater than equal to y and I am saying i am putting a cut there. Essentially what I am saying with the cut is that if you know x is greater than equal to y then x must be greater of two, you have done your job and do not try anything else. That is what you are saying with that cut. And then I am making this statement max x y y. I am writing these two statement. It relies on the fact that prolog searches in some fashion. It looks as first statement and then it looks at the second statement. So on the surface there is no flaw with program. For example, if you give a query like max two three three, it will


KRR-27-Feb-March2-2015 - Windows Journal
File Edit View Insert Actions Tools Help
NPTEL
21 / 21
12:13

Red cut vs Green cut

$\llcorner$ does not change the semantics or meaning of a program

$\downarrow$

Changes the meaning

Max (X, Y, Z) Z = Max(X, Y)

$\searrow$ used only for efficiency (depends upon the fact that Prolog does DFS)

Ex:

Max (X, Y, X) $\leftarrow$ X $\geq$ Y, !

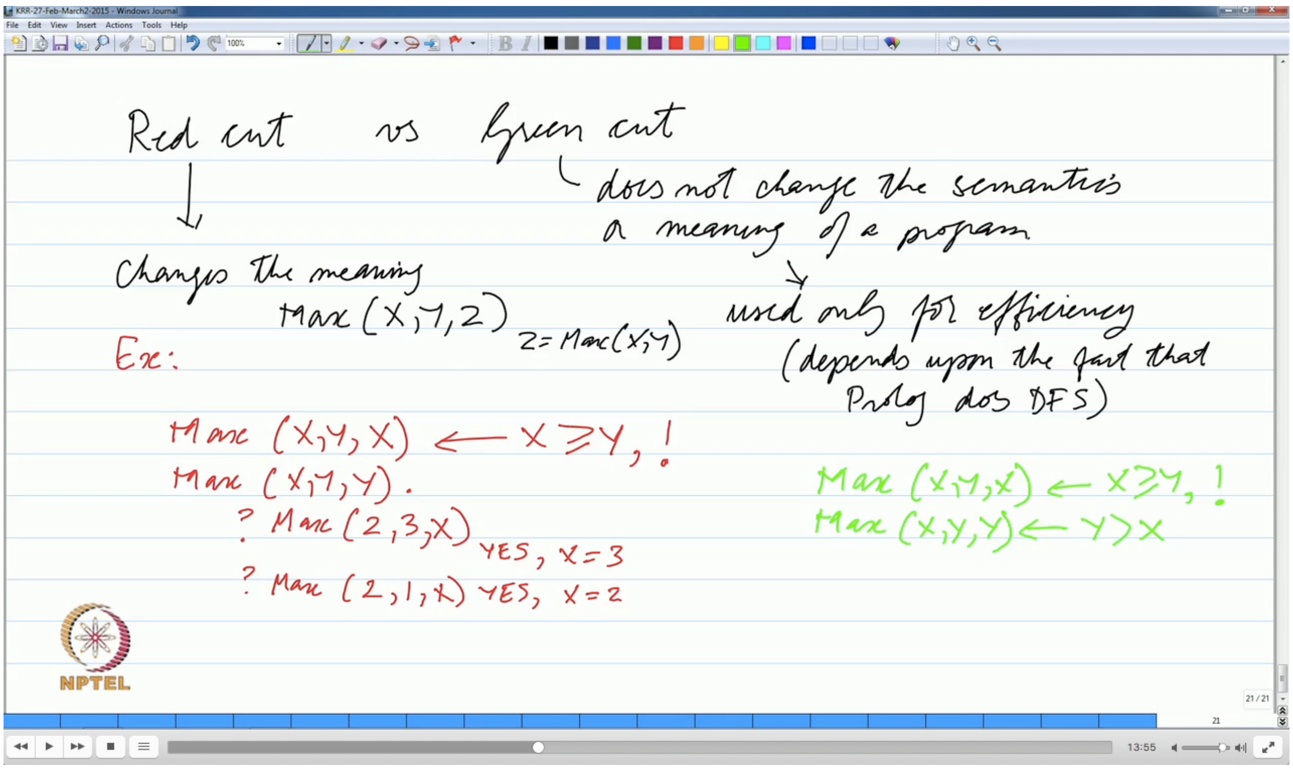
Max (X, Y, Y).

? Max (2, 3, X) YES, X = 3

? Max (2, 1, X) YES, X = 2

say yes and it will say three is the max. It will return the value or something. assuming it will return the value. If you say max two one, sorry I wanted to give a variable, It is x. x equal 3. It will again yes because the statement is true. It will again say that x equal two. In the both cases it will return a correct value to you which is maximum of these. So on the surface it lookalike this program is working correctly.

As oppose to this I am making a claim that this version of max which I will be writing in green color, green cut. It also uses a cut and similar statement.So either it is x or it is y. So first statement will be true if x is the max. Second is true if y is the max as before. But this time I have put a condition again that this time i explicitly ask that y should be greater x [Time 1252]. So the first statement is saying that x is max of x y if x is greater than equal to y. and then do not do anything else. This do not do anything is for the prolog operator as it searches for left to right and top to bottom. Essentially you are sayngin that you are done by showing max and do not do anything. Second statement says that y is the maximum of x and y if y is greater than x. As you can see I do not need to write equal to as this case has already been taken care of. So it look on the surface the both these program are identical, except that program in the green do extra test , it will only say y is max if it know y is greater than x.


KRR-27-Feb-March2-2015 - Windows Journal
File Edit View Insert Actions Tools Help
NPTEL
21 / 21
13:55

Red cut vs Green cut

does not change the semantics a meaning of a program

Changes the meaning

Max(X,Y,Z)   Z = Max(X,Y)

used only for efficiency (depends upon the fact that Prolog does DFS)

Ex:

$Max(X,Y,X) \leftarrow X \geq Y, !$

$Max(X,Y,Y).$

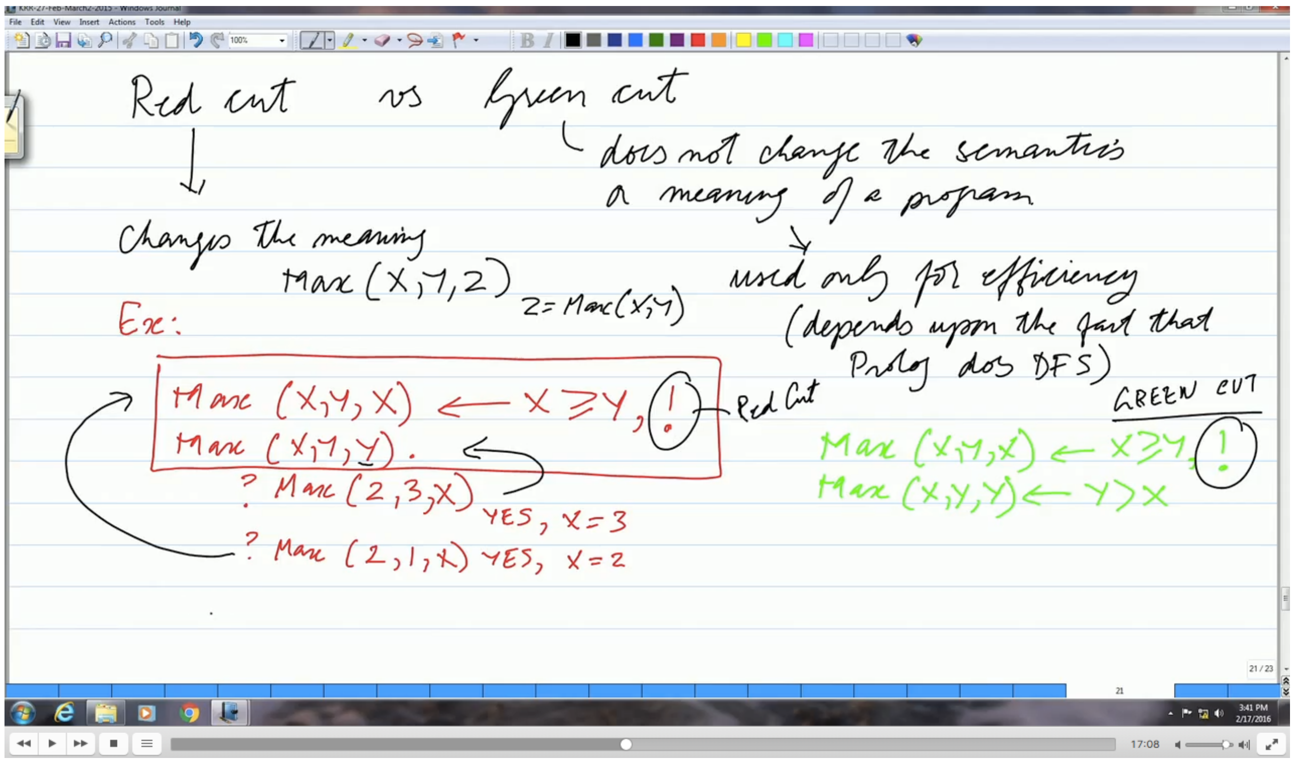
? $Max(2,3,X)$ YES, $X=3$

? $Max(2,1,X)$ YES, $X=2$

$Max(X,Y,X) \leftarrow X \geq Y, !$

$Max(X,Y,Y) \leftarrow Y > X$

The program in the red will say y is max if x is not greater than max essentially in some sense. So how does that change. So one thing you must try to realise is that it dose not change semantics or the meaning of the program is that we are using cut in conjunction with the depth first strategy, then it does not change the meaning of the program as in pure logic programming the answer should not depend in the order in which you look at the clauses essentially. It is purely a logical consequence we are talking about. So logical consequence should not depend on the order in which different rules are tried. But prolog because it does things in a particular fix strategy, we are trying to exploit the fact that we know the order in which it will look at things, you try to make things



Red cut vs Green cut

Green cut: does not change the semantics / a meaning of a program → used only for efficiency (depends upon the fact that Prolog does DFS)

Red cut: Changes the meaning

max(X,Y,Z)   Z = Max(X,Y)

Ex:

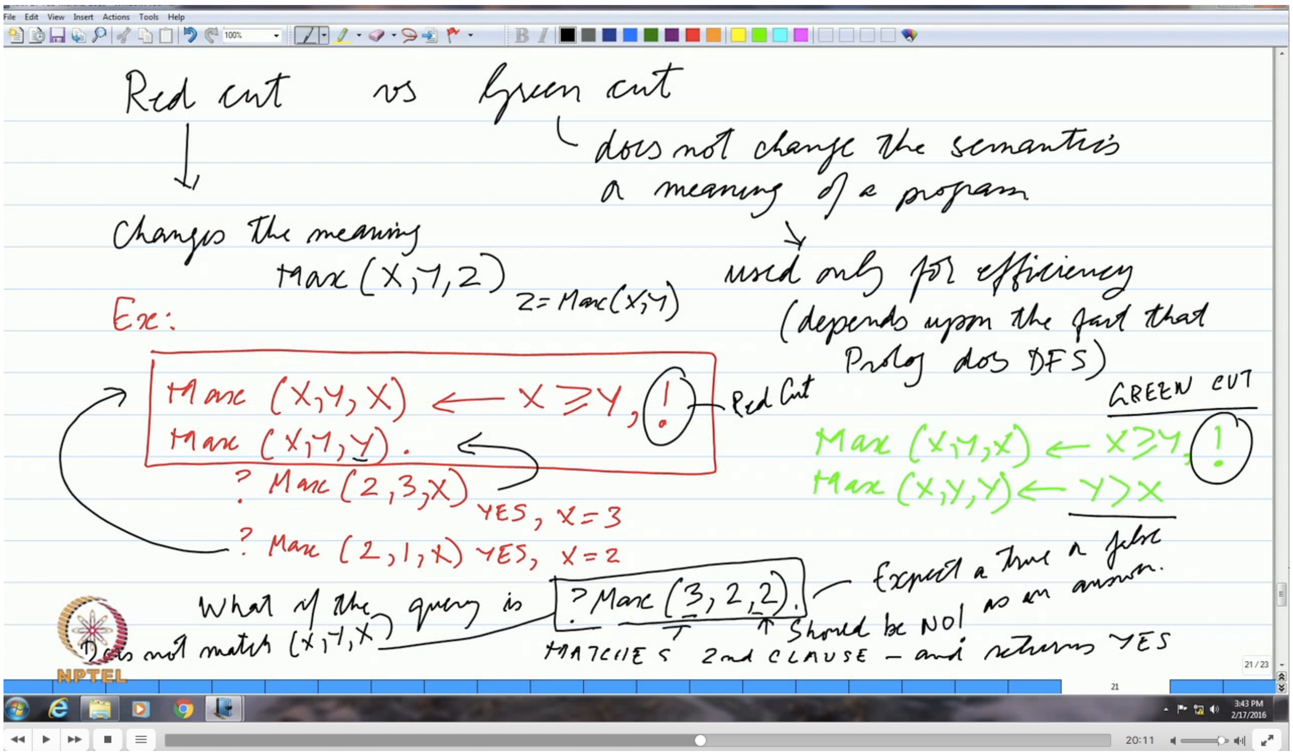
```
Max(X,Y,X) ← X ≥ Y, !     Red Cut
Max(X,Y,Y).
? Max(2,3,X)   YES, X=3
? Max(2,1,X)   YES, X=2
```

GREEN CUT

```
Max(X,Y,X) ← X ≥ Y, !
Max(X,Y,Y) ← Y > X
```

more efficiently and if you do that then it is a green cut. If you somehow change the program then it is red cut.

If x greater than equal to y, it returns false? how it will return false?
When will x greater than y will be false? When y is max. That is why second statement is confidently saying that y is max. But I am still saying that it is a red cut as it can still change meaning of the program. Ok. We have looked at these two algorithms. One in green which I said is green cut. And this one in red which I called red cut. For the example we have been looking at, which is where query has a variable x, there is no problem. The first query max two three x returns yes and x equals three is the answer. So 3 is the maximum of two and three. Second query, a different order is given, two one and x. Again it finds this thing. Now second query gave an answer because of first clause. x matched two, y matched one, so x match two because it has to be same value there. The first query gave the answer because it matches second clause. [Time 16.59].
The y is max essentially. So x y y matches two three three and three will be the answer essentailly. So what if, I gave a proportional query. What is the maximum of three two two. When I give an proportional query, i expect a true false answer. What is more i expect is correct answer. This is what we mean by saying is that red cut changes the meaning of the program essentially. Now, if you looked at pure logic programming, two is maximum of three and two then the answer should be no. An emphatic no infact. What will the program will do. The first clause will not match because these two values are different essentially [Time 18:40]. For it to match these two values must have been same. But it matches the second clause and answers yes, Which of course is not correct answers. It is because second clause is not explicitly checking that y is greater than x. Whereas in green cut version, we have an explicit check that y is greater than x. only then y is maximum of x and y. Red is simply saying that if x is not the maximum of x and y, then y must be the maximum of x and y. And he does not really care that y is maximum of x and y. This essentially proves that a red cut can result in some cases changing the meaning of the program as is the case here. So for this query a red program will return yes [Time 1957], which is wrong. Green program will return no, which is correct essentially.



Red cut vs Green cut

does not change the semantics a meaning of a program

Changes the meaning

Max(X,Y,Z) Z= Max(X,Y)

used only for efficiency (depends upon the fact that Prolog does DFS)

Ex:

Max(X,Y,X) ← X ≥ Y, ! Red Cut
Max(X,Y,Y).

? Max(2,3,X) YES, X=3

? Max(2,1,X) YES, X=2

GREEN CUT

Max(X,Y,X) ← X ≥ Y, !
Max(X,Y,Y) ← Y > X

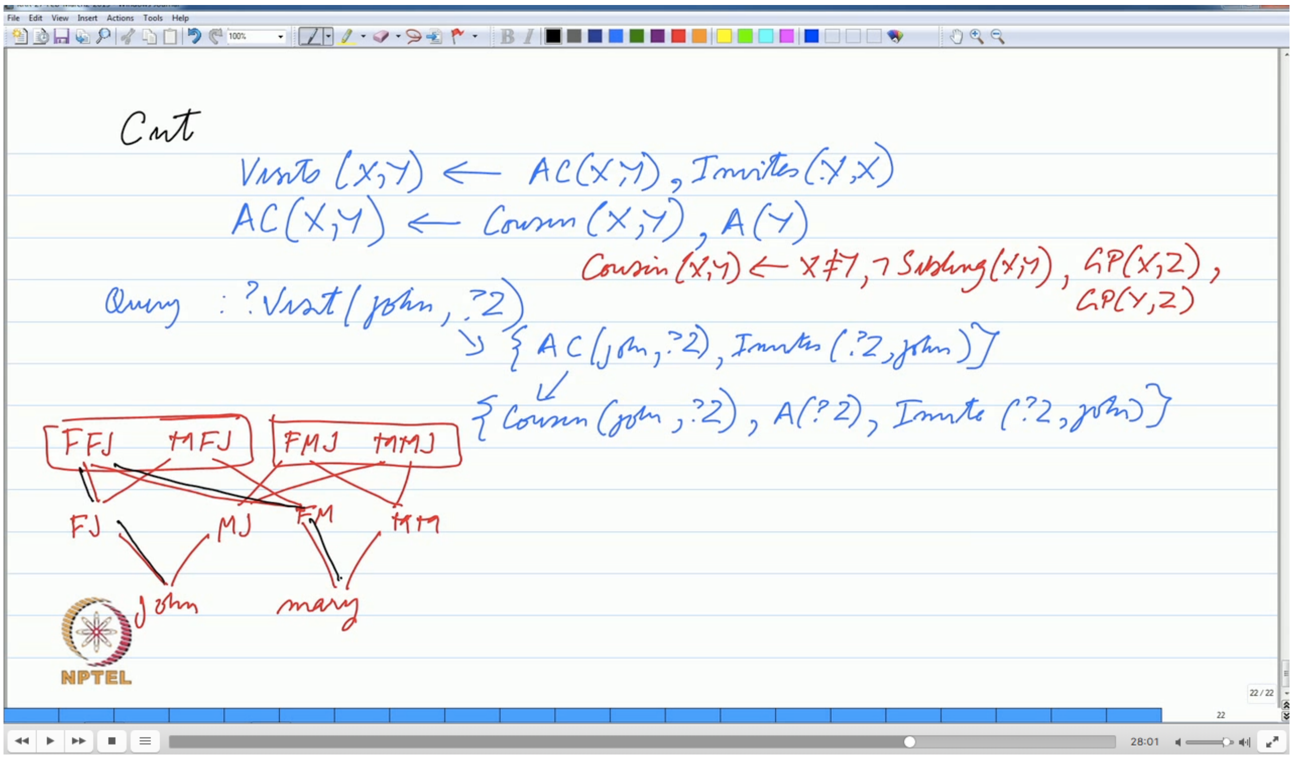
What if the query is ? Max(3,2,2). Expect a true or false as an answer.

Does not match (X,Y,X)

Should be NO!

MATCHES 2nd CLAUSE – and returns YES

Lets look at an example where it really helps. So remember that we have written a program to define a american cousin. Let us say this is the definition that AC stands for american cousin. X is american cousin of Y if Y is cousin of X and Y is american essentially. Let us say this itself is called by another clause which say that visit X Y if Y is american cousin of X and invites Y X. Some program which call american cousin. Lets see what prolog is going to do here. Let us say we have a query that does visit John. Let me use a question mark again. Does john visit somebody? Then this will spawn tow goals. That does John has an american cousin who invites him. The first one will intern spawn these two goals. Does John has a cousin and is the cousin american and did the cousin invite John. Just imagine this is our knowledge base. We have John and let us say Mary is a cousin of the John which means they have a common grand parent. Lets say this is the definition of cousin. First of all X must not be equal to Y, they must not be siblings, and then there must be a grand parent of X who is also grand parent of Y. Assuming that meaning of that GP X Z is that Z is grand parent of X. Ok so. We are making a query about John. We are first going to check whether Mary is cousin of John. Test that we will need to look at their family tree. Lets say that this person is father of John. this person is mother of John and this person is father of Mary . This person is mother of Mary and lets say this person is father of father of John. [Time 25:50] and this person is mother of mother of, sorry mother of father of john. This is the relation between them. So I should had drawn it little , let me draw it again. Father of father of john, mother of father of john. Father of mother of John and mother of mother of John. These are parent relationships. And lets say as it can happen sometimes this is the relationship between them. [Time 26:51] . The parents of John and Mary are siblings essentially. I wonder whether I am doing something wrong here. Okay both mothers sibling of this couple and both the fathers are sibling of this couple [Time 27:25]. Lets say this is a closely knit family. To show than John is a cousin of Mary, you need to show that they have a common parent and you can see that this path will work. The father of father of John is also the father of father of Mary. [Time 27:54]. The path in black in work, so we know John and Mary are cousins.



Cut

$Visits(X,Y) \leftarrow AC(X,Y), Invites(Y,X)$

$AC(X,Y) \leftarrow Cousin(X,Y), A(Y)$

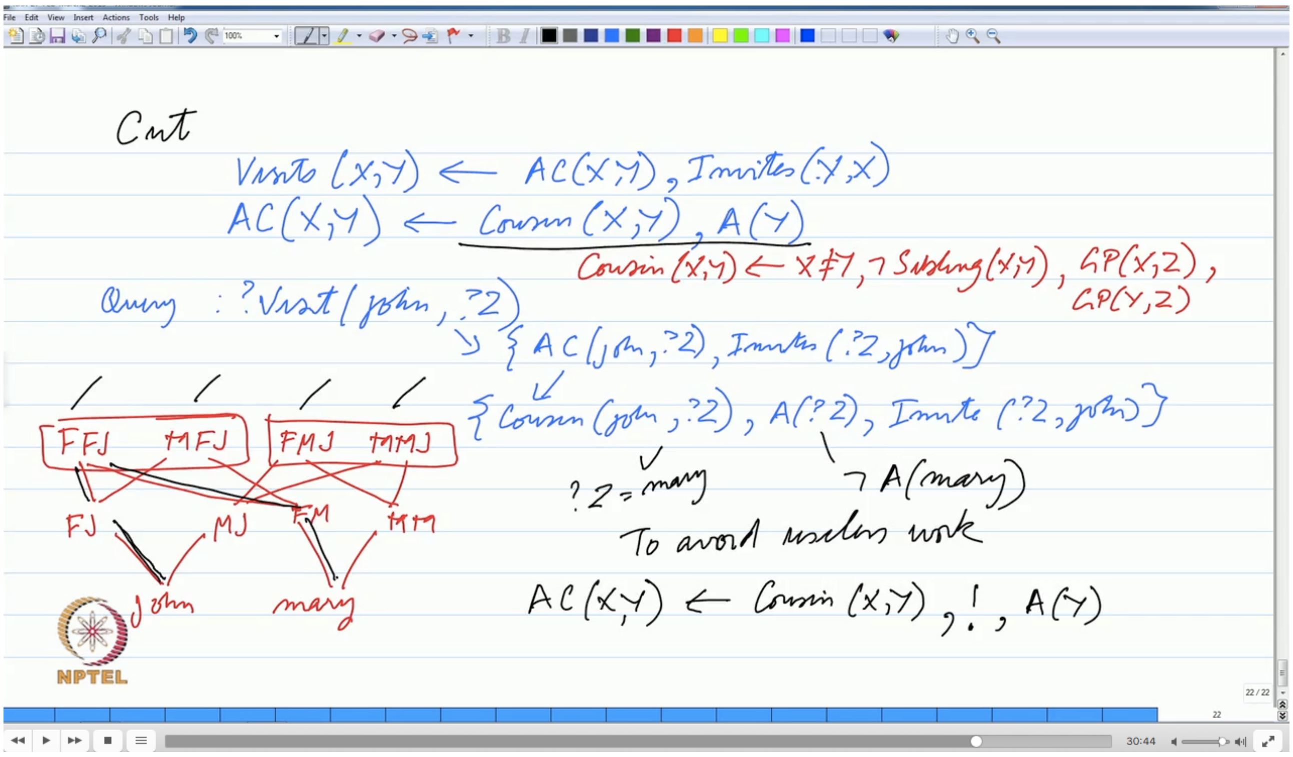
$Cousin(X,Y) \leftarrow X \neq Y, \neg Sibling(X,Y), GP(X,Z), GP(Y,Z)$

Query : $?Visit(john, ?Z)$

$\{AC(john,?Z), Invites(?Z,john)\}$

$\{Cousin(john,?Z), A(?Z), Invite(?Z,john)\}$

This will become true with Z equals Mary and let us say Mary is not American. When it is trying to look at these two goals here [Time 28:44], it is trying to show that Mary is an american cousin of John. It shows that as we have along this black line as both their fathers are sons of one person so they are cousins succeeds but american Mary fails. Lets say Mary is not american .What prolog do , it comes back to cousin X Y and tries to show that cousin John Mary is true. And it can do that as it can see that there are four different ways to show cousin John Mary is true because they have four common grand parents. All four grand parents are same for both of them. So it is going to be useless work essentially. So first it has tried with this, then it will try with this and then it will try with



Cut

$Visits(X,Y) \leftarrow AC(X,Y), Invites(Y,X)$

$AC(X,Y) \leftarrow \underline{Cousin(X,Y), A(Y)}$

$Cousin(X,Y) \leftarrow X \neq Y, \neg Sibling(X,Y), GP(X,Z), GP(Y,Z)$

Query : $?Visit(john, ?Z)$

$\searrow \{AC(john,?Z), Invites(?Z,john)\}$

$\{Cousin(john,?Z), A(?Z), Invite(?Z,john)\}$

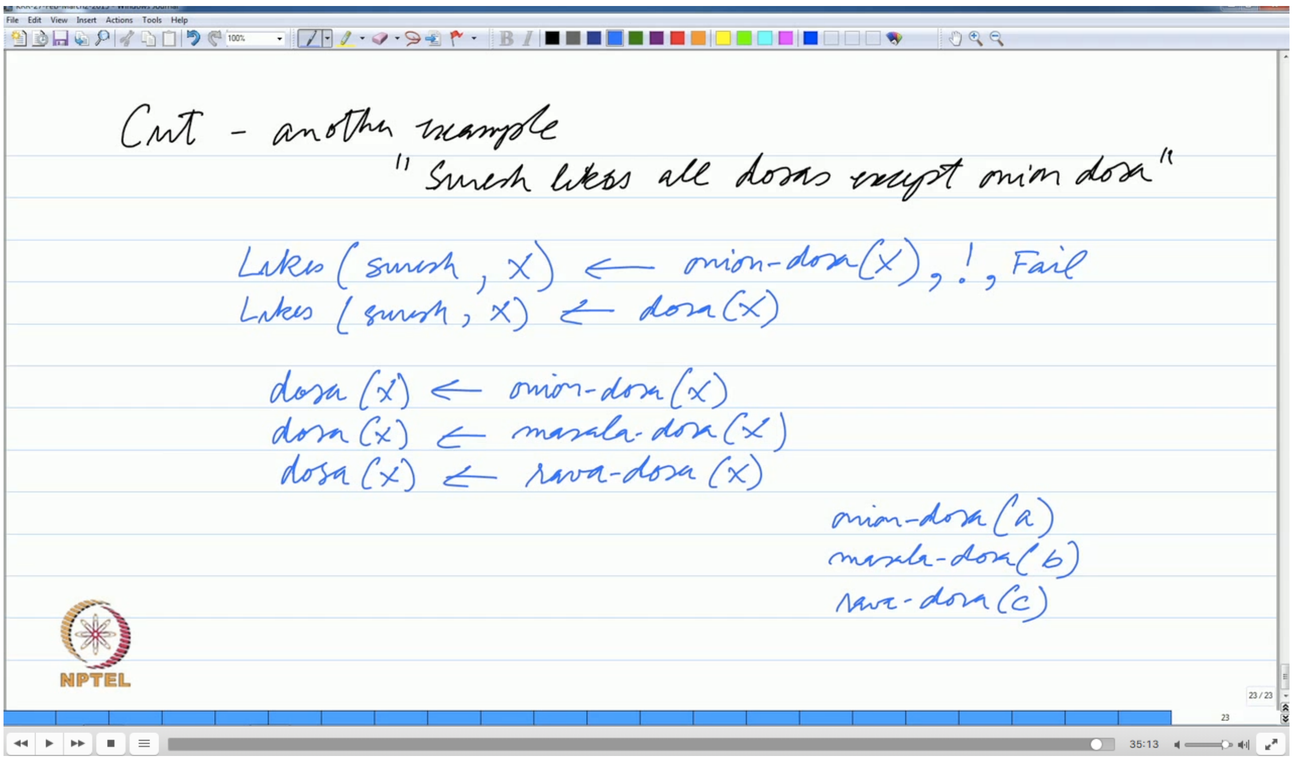
$?Z = mary$ ✓ $\neg A(mary)$

To avoid useless work

$AC(X,Y) \leftarrow Cousin(X,Y), !, A(Y)$

this and then it will try with this [Time 29:45]. In all four times because we know that american is not american, second goal will fail anyway. It would had done some useless work and then it will go and try to find somebody else whether John has another cousin who is americana and so on and so forth. To avoid this repetition, we can rephrase the program as that american cousin of X i.e. Y is an american cousin of X if Y is a cousin of X. If you already know that Y is a cousin of X, you are committing to this part of the proof since you already know that Y is a cousin of X. You put a cut here. Then you test whether Y is american or not. This means once you found that John and Mary are cousins, you are not going to try to find any other way to show that John and Mary are american cousin. Either Mary is american, in which case it will return true. Or if she is not american, it will return false essentially. Here it is a case where it is useful. In last class, when we had introduced cut, we see it is another interesting use of cut.

So lets look at another example. We want to express the fact that Suresh likes all dosas except lets say, onion dosa, if there is such a thing. We can do it like this. Likes Suresh X. Lets deal with exception first. If X is onion dosa then return fail. Remember that once you have crossed the cut, you are committed to the second part. Then you can say likes Suresh X, as long as X is dosa Suresh Likes it. Prolog look at the first clause. If X happen to be onion dosa, Sures do not like it. Then you can give definition of dosa. A dosa is. An onion dosa is a dosa or masala dosa is a dosa. Or rava dosa is a dosa. Or may be you have other kind dosa . If you have knowledge base like this, you give different name to dosas, something like,   A . You define a small knowledge base in which you say onion dosa a, masala dosa b , c. Then you ask query does Suresh like anything? With these set of rules, it will return first masala dosa, in the sense it will not be able to show that a is true but for b and c it will return answer yes because first call would had failed. b is not onion dosa, so it would had tried second clause. Then it will test whether b is a dosa, and to test that it will see that because it is a masala dosa so it is a dosa. So it like all dosa except onion dosa. I will stop here.



Cut - another example

"Suresh likes all dosas except onion dosa"

Likes (suresh , X) ← onion-dosa(X) , ! , Fail
Likes (suresh , X) ← dosa(X)

dosa (X) ← onion-dosa (X)
dosa (X) ← masala-dosa (X)
dosa (X) ← rava-dosa (X)

onion-dosa (a)
masala-dosa (b)
rava-dosa (c)

We will start looking at backward chaining here. In the next class we will see drawback of backward chaining, which is also drawback of forward chaining and then move towards general solution.ss`