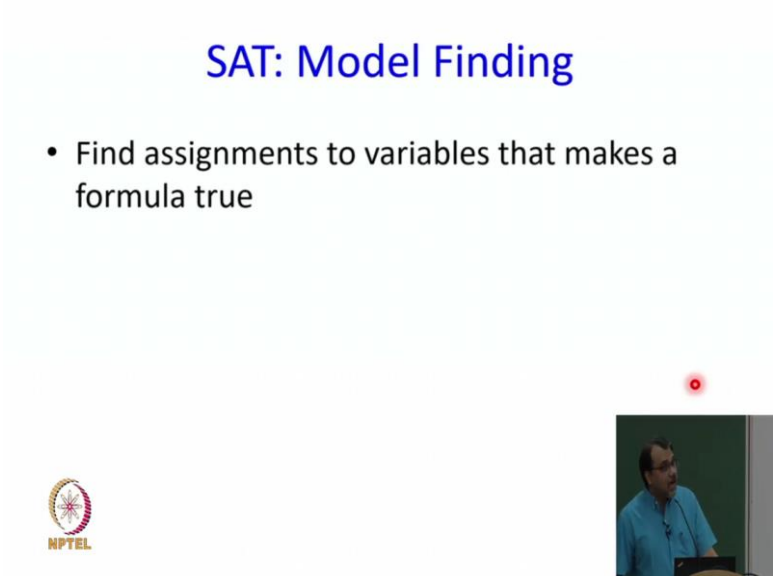
Artificial Intelligence
Prof. Mausam
Department of Computer Science and Engineering,
Indian Institute of Technology Delhi

Lecture-48
Logic in Artificial Intelligence: SAT Solvers: DPLL Algorithm, Part-7

(Refer Slide Time: 00:17)



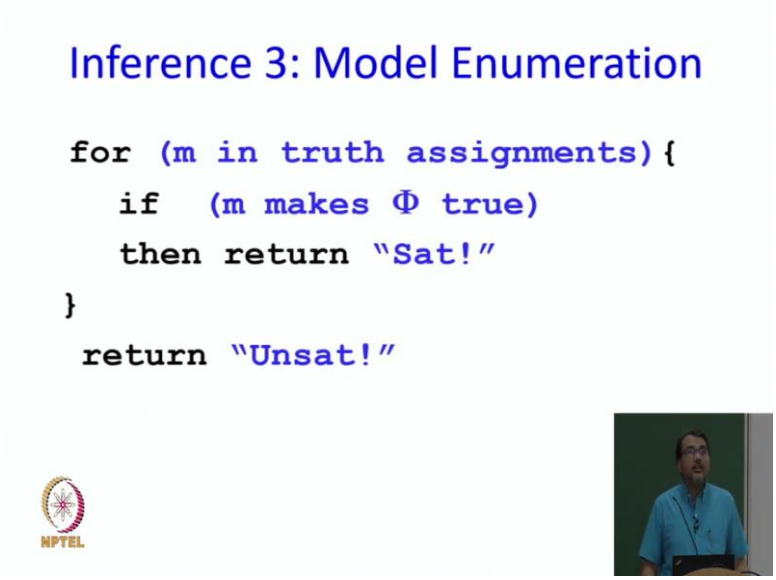
So where are we? We have been talking about satisfiability problems. And in the last clause, we discussed why satisfiability problems are an important problems to solve. And the reason it is an important problem to solve is because it is a canonical NP complete problem, which basically means you give me a hard problem if it is in the clause NP. Which a lot of the real world practical problems.

Are then I can convert it into a SAT problem and I can solve it using SAT and if has solved it using SAT this solution of SAT will give me the solution of the original problem. What is the SAT problem? The SAT problem is a problem where I have some variables, logical variables, and I have a set of clauses that are given to me, or a formula given to me in a conjunctive normal form. And my goal is to find assignments to variables, such that all the clauses in my CNF are satisfied.

And now suppose I want to solve this problem, I want to find a solution in let us say, I am starting with a very brute force approach is a very brute force approach. Would we start with all possible models? By the way, this is also called model finding. Why is it called model

finding because one complete assignment of variables is a model in the language of logic, we have discussed this earlier.

(Refer Slide Time: 01:34)



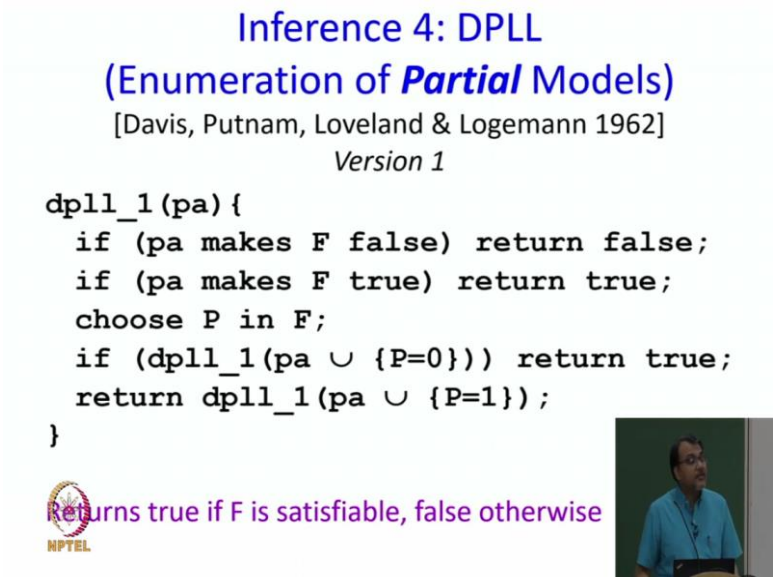
So if I wanted to do a complete brute force approach, my brute force approach would be to go over all possible truth assignments. And let us see if any one of them makes the formula true. And if any one of them makes the formula true, you return satisfiable and you return that particular assignment. And otherwise you return unsatisfiable. If I wanted to implement this if I have n variables in my SAT formula, how much time will this take? 2 to the power what? 2 to the power n?

Why? Rows wait for n variables; they will be 2 to the power n different rows, what rows are we talking about? Truth table but in our language, what will we call them? Assignments, completed assignments all models so there will be 2 to the power n complete assignments, and then why would it take another explanation on that? It would not so there are 2 to the power n assignments and for every assignment, it takes me some linear time to check whether the formula is satisfiable or not.

So of course, we do not want exponential to the power n is not very good. We do not want an algorithm which is so bad we want to do better than brute force. And the first observation that we have probably is this SAT formula is nothing but an instance of another problem, we have studied earlier. Constraint satisfaction, it is a CSP problem. Why? Because my clauses, my constraints, my clauses are my constraints. And my variables have truth true false values. So

it is basically a series. So we can do search. First SAT formula as we did search for CSP backtracking search, and if we did this, we will call it the Davis Putnam algorithm.

(Refer Slide Time: 03:55)



So it is a new name that I am introducing to Davis Putnam or later it became dpll algorithm. dpll stands for the 4 different inventors who worked on this particular algorithm. Davis Putnam, Loveland and Logemann and this particular algorithm came in 1960s. Now I use the word invented. And you know, we will take a very let us take a brief diversion and just think about our algorithms invented or discovered.

This actually a very good question it is a very hard question. It is a philosophical question. I am not going to answer this question for you. I do not know. But you should ask this question. You should say this Dykstra algorithm, or this depth first search algorithm. Is this a discovery? Or it is an invention. Now we know revolver is an invention. And we know that India was a discovery or USA was a discovery.

What is so different between a revolver and as place? For example, in the context of finding it for the first time it already existed now, revolver did not already exist. Did Dexter's algorithm already exist? Did Dexter's algorithm already exist in the way different from the way revolver existed? What did Dexter said, if I put together my computational instructions in a very specific way. Then I have an algorithm that works.

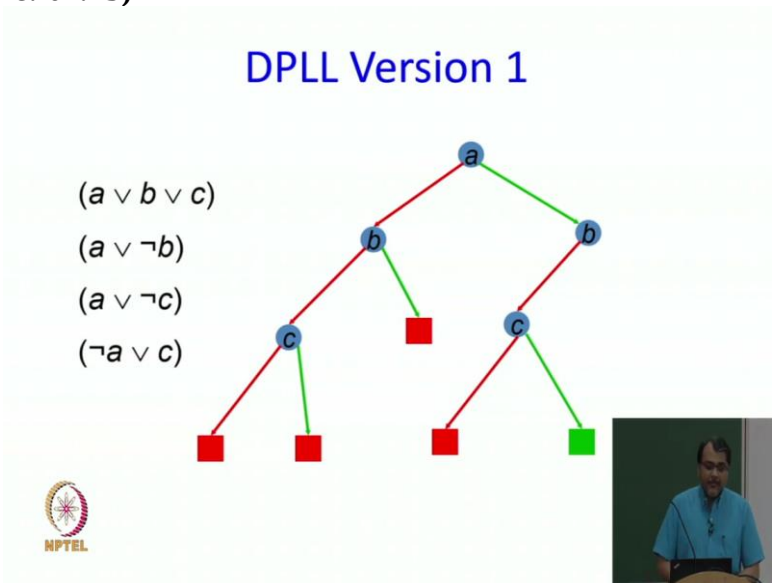
And what did hold in the revolver he said if I put together this metal and this spring and does everything in a very specific way, it makes this revolver I do not know the answer to this

question, you know, I keep going back and forth, some algorithm seems so innovative, that they feel like inventions. And some algorithms are so obvious that you know you give it give that problem and you will get to that algorithm.

It feels more like a discovery. It is a feeling I am not making a scientific argument here. So anyway, I just said that they invented the dpll algorithm, but you can say discovered that is, it is your personal prerogative at this point. But you should ask these questions. You know, you are computer scientists or budding computer scientists. These are fundamental philosophical questions about computer science anyway, coming back dpll.

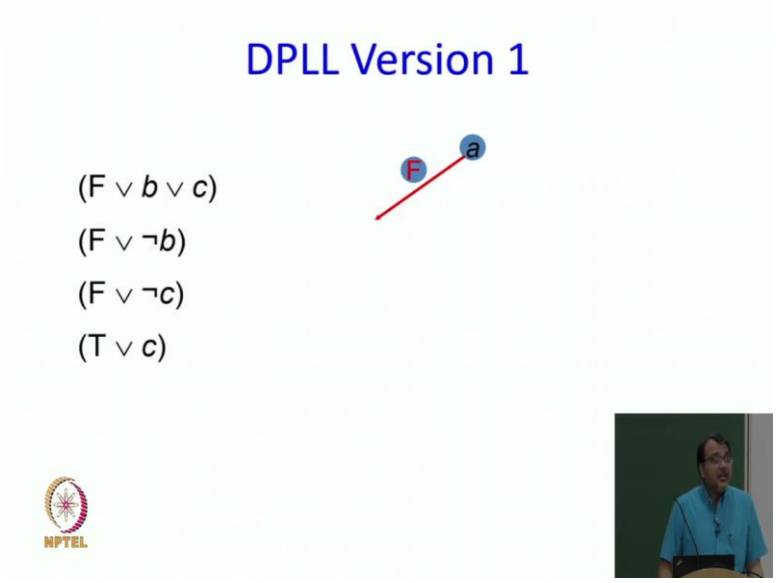
So dpll essentially says let us do search. lets you take a partial assignment if it makes the formula false return false if the make form it makes the formula true return true, or choose a variable in the formula and then you search for that variable equal to 0 and that variable equal to 1. So it is basically branching factor is always going to be 2.

(Refer Slide Time: 07:15)



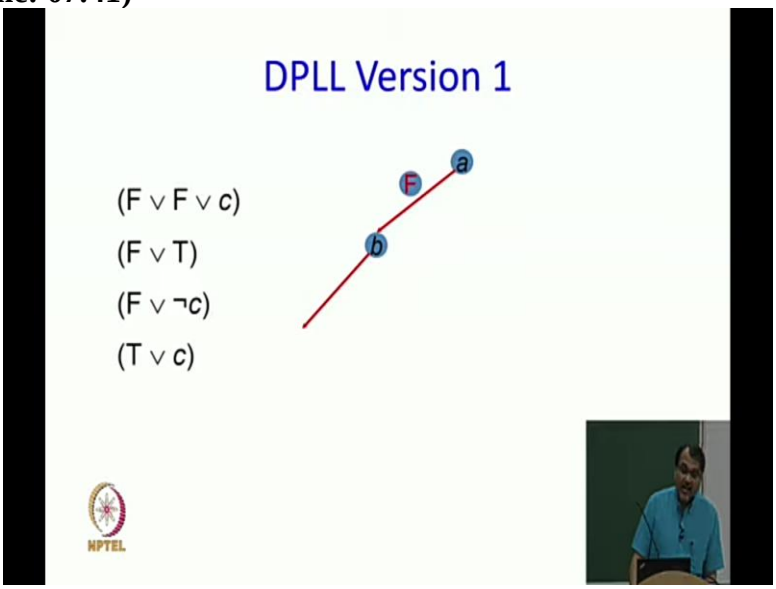
And let us say we run this very basic dpll algorithm. So this is a very, very basic algorithm. So what it does is it does a nice formula A or B, or C or not B or not C, not A, or C, I start doing the dpll search, let us say I pick variable a first I decide to go for false first.

(Refer Slide Time: 07:30)



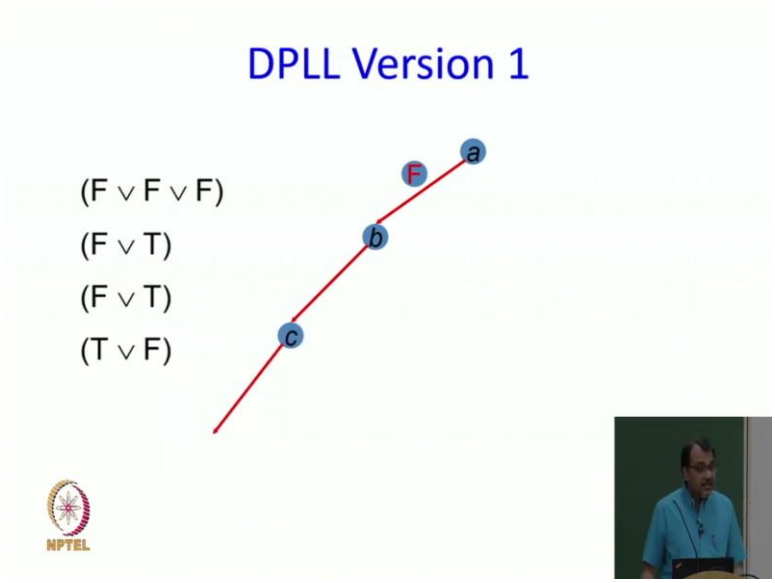
Then if I simply run the algorithm that has been given to me, the question is does this make formula true or false? No, it does not make anything right now.

(Refer Slide Time: 07:41)



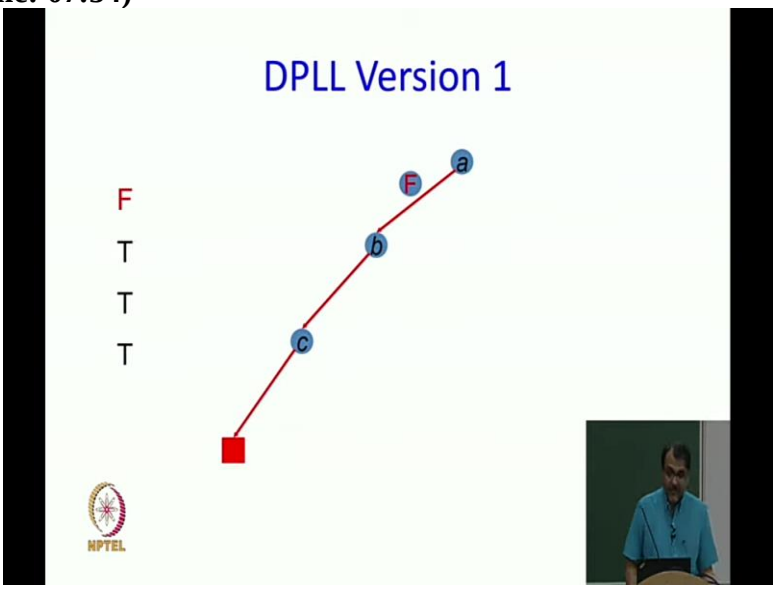
So I go b I branch and b false. Then I could ask the question, does it make the formula false or true so far? No, it does not.

(Refer Slide Time: 07:48)



I keep doing I branch and see then I ask the question, does it make the formula true or false, it makes the formula false. I backtrack.

(Refer Slide Time: 07:54)

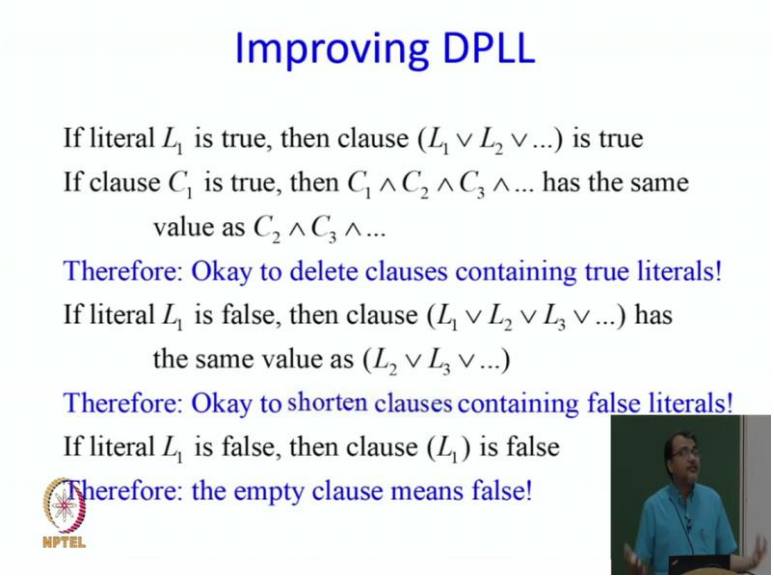


Now, when I backtrack, I go back to c and I try the value true. Then I try the value true b and I keep doing this. And this is how the algorithm will work. And finally, I will get the first success and that is where I will start. So now, of course, this is a backtracking search algorithm because at any point we are only branching on one variable.

(Refer Slide Time: 08:19)

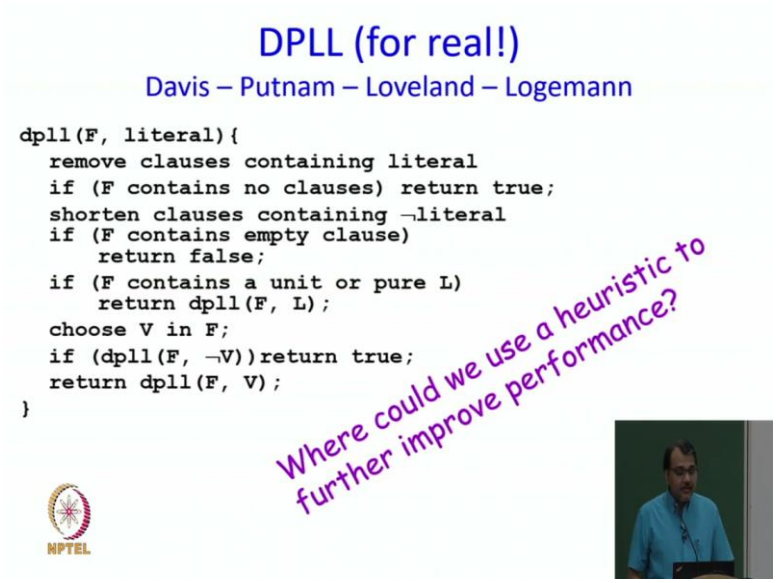
This is the search space is the space of partial assignments just like in a CSP. And it is a depth first search style algorithm, in particular backtracking search. So far, so good but of course, this does not work because it is too simple and takes a long time.

(Refer Slide Time: 08:38)



So, people started improving dpll and of course, there are very some very obvious things that you can do to improve dpll you must have seen it already. So if a clause has already been assigned a true value, you can remove the clause. If a variable has been assigned a value which is negated in the clause you can remove that variable from the clause. So let us shorten the clauses and remove the clauses whenever appropriate, an empty clause means false, of clause. So suppose I had a or b, I set a to false. So then I have, I am left with b, I said b to false, then I am left with an empty clause, empty clauses false, I backtrack.

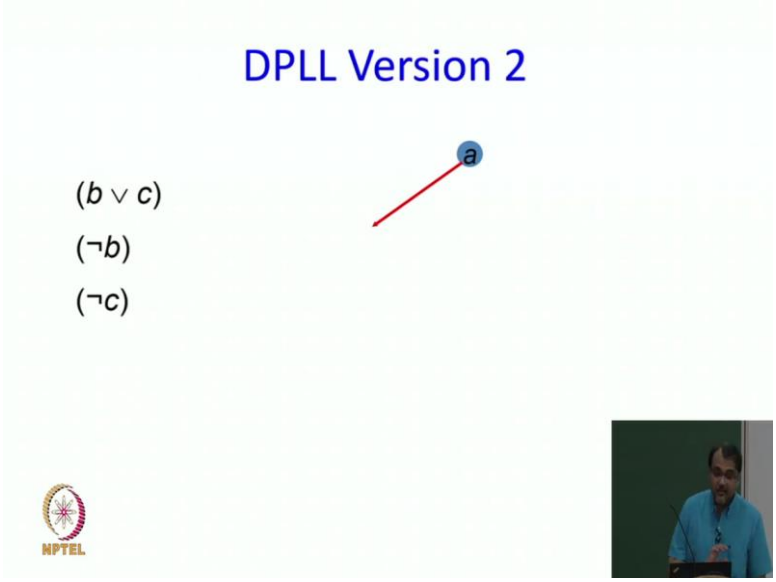
(Refer Slide Time: 09:17)



So the change in the algorithm would be removed clauses that contain a literal that has already been set. And if formula contains no clauses that return true. If a formula contains an empty clause return false. And you shorten clauses containing negativity so let us take an example. If we started with setting a to false now as soon as we set a to false, the first clause becomes this is where I want to start having you put your thinking caps back on.

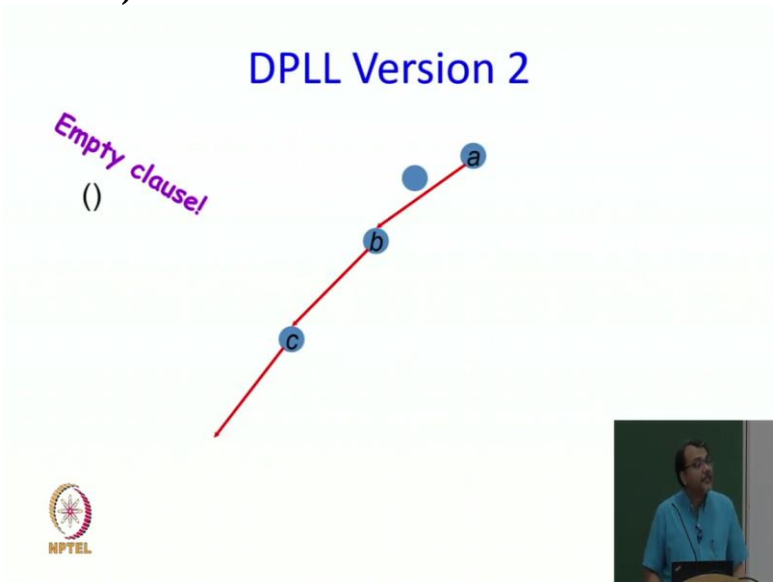
Because I am going to ask you some questions in the next slide so as soon as I put a to false I set a to false the first clause becomes BRC the second clause becomes not B and so on, so forth. Fourth clause becomes true. So you can remove it.

(Refer Slide Time: 10:07)



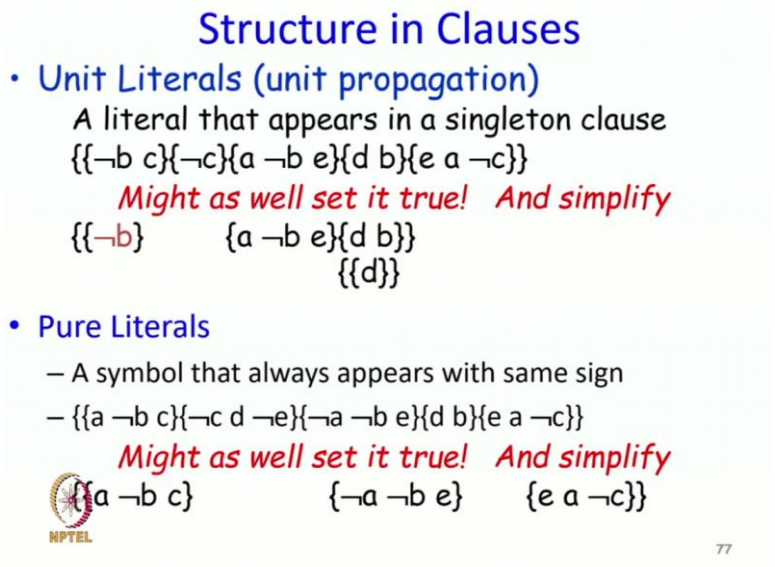
So this is what we do. This is the formula we are left with, then we branch on b, the branch and b, set b to false, then we are left with c and not c, the middle for formula goes away, and then we set c to false. And this is what we are left with.

(Refer Slide Time: 10:23)



And as soon as we got an empty clause, we say this is a failure. Let us backtrack. So, now, this is where we start from, and now we ask the question can we do better? And because it is a CSP, you sort of know what does better mean? Better means what is the model with a notion of better in the context of backtracking search, how can we do better? Detect failures early exactly and what else? Choose which variable to branch on. So those are the 2 important levers we have right now choose which value to branch on is also appropriate and we will see an example so in order to get your intuitions going.

(Refer Slide Time: 11:14)



Let us look at this particular formula. Now, if you were looking at this formula, which variable would you pick first think about it, think about it, I will ask a random person, let us say you have to figure out which how to make this formula true formula true means each individual clause should be set true, and each individual closer disjunction of variables. So, how, let us say I asked the person in the green what is your name? In the second row? Yes Miyan.

So in this formula, what do you what can you obviously save looking at this formula if c is false, then not a c will be true? I agree with this statement. But in the context of this formula, can you say that c has to be false? If there is a satisfying assignment you cannot say that good question. So claim one that for this formula to be true, every assignment should have c false claim to that is not true. How many believe in claim 1? How many believe in claim 2, nobody good so you figured this out.

You figured this, that look there is a formula we need to make it true every clause has to be made true. There is a clause with just one variable. If I had the clause just one variable, what is the only way to make it true? By choose the variable value says that literally becomes true. In this case, c should become false. So now there is no point in branching on a branching on b you know exactly some value of a variable.

Let us set it right away and this is called unit literals and this process is called a unit propagation, unit propagation is like forward checking in forward checking, what should we say? We would say what are the remaining values? Let us figure out what are the remaining values and now, we are sort of figured out that c has only one remaining value which is false. So, we just pick that and move on.

So, now that we have set c to be false, we can always shorten the clauses. So, now we are left with not b, a not b e and d b, in fact 2 clauses got satisfied. Now, what can we do? Set b to false as soon as we said b to false we have got to other clauses satisfied we are left with d. Now, what can we do? Set d to true? So what did we do? We did not do any search. We only did inference. We said and the final answer is d false c false, d true a and b can be anything, it does not matter.

So notice that if I did this, this would actually make my search much better because at some points I will not be searching at all. I will simply be setting the variables with much which must be set and move on and must be set a specific value. So not only am I choosing the variable, I am only also choosing the value I need to set I do not have to even try the other value. Now, similar to this, can you think of some other suggestion?

This is the harder one but you know let us see if you can think about it went through in this particular formula. Can you think of some other way to branch or not branch but set a value to us variable so let us ask each on Eshan. Yes Eshan we choose a variable that occur in many number of clauses, but you are asking, responding to which variable should I say? But you are still doing search? You are trying both sides, my question is, is there something else we can do where we do not have to even to search?

We can just sort of say that let us set this variable, this value. And we do not even have to backtrack from there. It is a good decision. Like we had a decision where we said c has to be false. There is just no way around it. Is there something else? Let me give you a slightly difficult example then that might help you move. So let us say this is my formula. Think about it. If you were doing this again, you should always think about if you were doing this, what would you do?

And very often you will have an insight that you can put in the AI system, at least in the free neural world. In the neutral world, this thing is hard. In The deep learning world, this thing is hard, but at least until the pre neural world you can say that if you have an insight, you can put it in the algorithm one way or the other. Neural world also allows it but there are levels of indirection which make it harder, but anyway, we will come back to that.

So let us say this is the formula I gave you. There is no unit literal here. So you cannot pick unit literals. Now can you pick some other variable and a specific value and said this variable this value should give me a solution and I do not even need to try the other side so let us see. Yes, what is your name new person? Abbay yes so, if there is a literal which is a same polarity of a using terms which are not using the right logic language.

But you should say if a literal appear in only that form and not its negated form, then we assign that literal first we assign literal true. So, Abbay makes a very good observation for

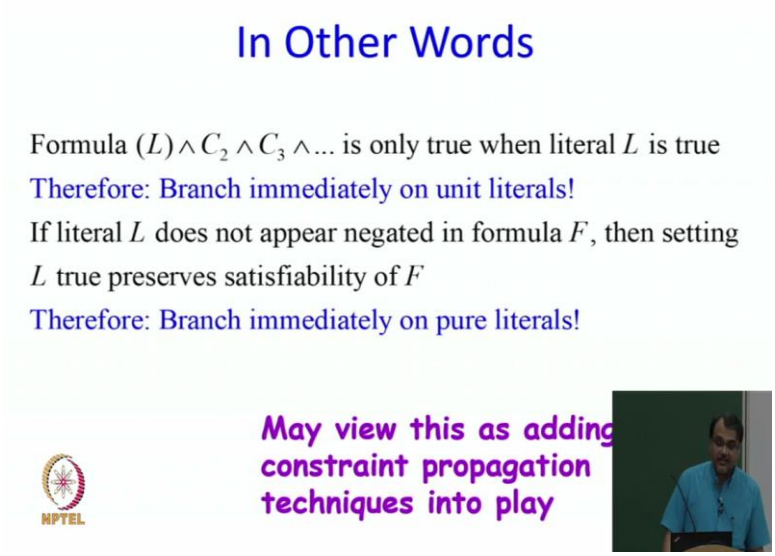
example, in this case, we can assign d true so, you know as Abbay something very interesting he says look, think about d, I have d in the second clause I have d in the fourth clause in both of these clauses d is in the positive form.

Do I have not d anywhere I do not have not d anywhere. I just need to find one satisfying assignment. I do not have to find all satisfying assignments. So, if I said d to be true I am only helping the clause and not hurting the formula at all. I am helping all those clauses where d shows up, but I am not hurting anything. If I am not hurting anything, there is no reason to try the other value. They may be solutions there. They may be solutions with d set to false. I am not saying that there are no solutions, but we do not care.

We just care for one solution. Are you able to see this? So because d only appears in the positive form, we can set it to true. Now as soon as you set it to true, 2 formulas get satisfied. Of course, they do not get simplified and nothing else gets simplified, because they did not even appear in other clauses. What else can you do now? We can set b to false because b also has the same characteristic.

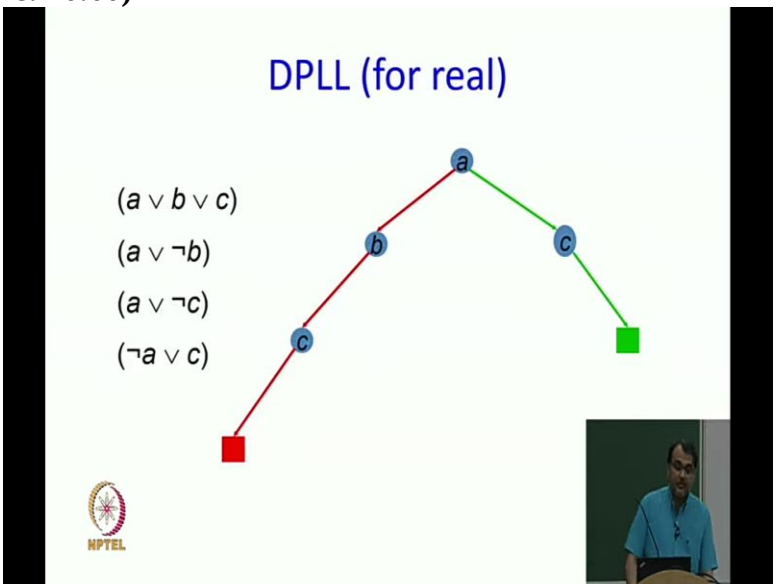
These are called pure literals. By the way, if he said b to false, we will be left with e a and not c and they will all so we pure you can set anything. So your 4 year solution would be b false, d true. And let us say a true these are called pure letters or symbols that always appears with the same sign and that really helps us in not branching on some variable values and making faster progress in search.

(Refer Slide Time: 19:25)



So, see if a formula is only true when literal L is true branch immediately on that literal those are called unit literals. If l does not appear negated in a formula f then setting l true preserves satisfiability affects branch immediately on to literal. Again, these can be viewed as adding constraint propagation techniques into play. And so my final Davis Putnam Loveland and Logemann algorithm becomes everything that I did earlier plus if F contains a unit literal or a pure literal, just set it and do not even try the other side.

(Refer Slide Time: 20:00)



And so if you actually do the dpll for real on this particular formula if you look at this formula carefully, are there any unit literals? No, there is no unit literals because there is no clause of size one. Are they any pure literals no, a and not a b and not b c and not c all of them are present. At this point I cannot set anything obviously. So I have to set a variable true and set a variable false and you know, do search.

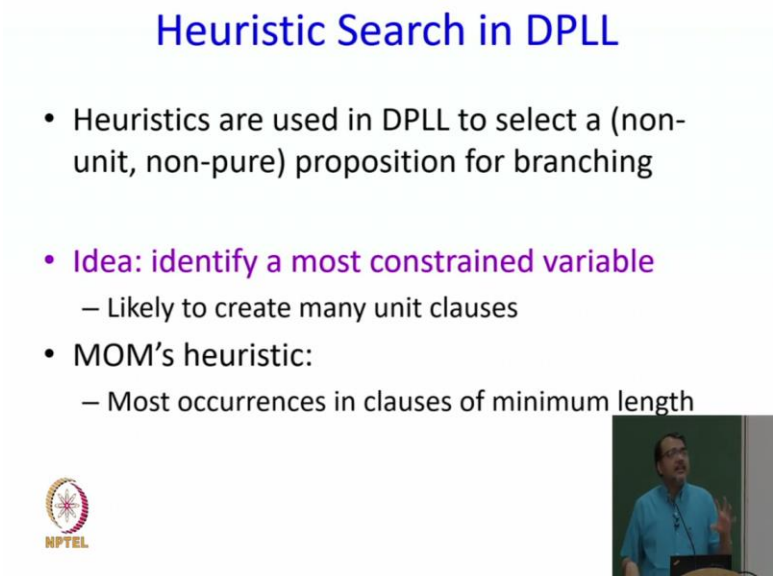
So I do that search let us say set a false but then as soon as I set a false the last formula get satisfied second and third formula have become unit. As soon as they have become unit you said b false c false. When you said b false c false you get in failure you backtrack but you do not have a try see true. You do not have to try b true. You go to a and now you try a true as soon as you try a true all the 3 formulas. First 3 formulas get satisfied. You are left with a unit clause c you set it and you are done. Your solution is a true c true.

So, this is my dpll algorithm for solving satisfiability problems. But now we have to ask the question is there someplace where a heuristic could further improve performance? And you know the answer to that, where is that heuristic? When there is no unit clause a unit and when

there is no pure literal, what do we have to decide which variable to do search over? Let us go back in CSPs. We said that which variable do we pick?

Do you remember? Most constraint and our definition of last constraint was number of remaining values for that variable should be the highest how many variable values do we have for any variable here? 2 so, that maximum remaining values to the stick is obviously not going to work yet everything will have 2. So, let us think about it again, what do we want we can use that degree heuristic the degree heuristic says a variable appearing in more constraints in this case are variable appearing in both clauses.

(Refer Slide Time: 22:43)



However, there is a secondary insight here. And the secondary insight is that, you know in typical CSPs the domains are large and so we do not get there. But in logic the domains are always a size 2 and we know all the tricks about you unit literals, etc. And so we really want to get some clauses become as short as possible quickly so that we can set them to those values and not we do not have to do that much search.

So, a particular branch gets explored extremely fast. So we sort of intuitively want that variable which is likely to create many unit clauses. That is sort of the intuition of the branching heuristic can you come up with something like this? So, we can take for every variable, check it out how many clauses it appears in correct and what else and count the number of other variables appearing in that clause.

And for the one that has the least number of the other variable, so is that are these numbers suggestion and you know when you try it, you might find that there is some inconsistency or something you need to fix. But the final answer is not very different. The answer is sort of what is called the MOM's heuristic the most occurrences, which is like degree in clauses of minimum length.

So if I have clauses of length 2, then take a variable that has that can reduce the clauses of length 2, if I have clauses only of length 3 or more, then I pick all the clauses of length 3, and then find the most occurrences in those clauses. So the extremely large clauses we do not worry about initially, because there are too many ways to set them to true a clause is a disjunction at the end of the race.

If you have a large clause, then there are many, many different ways to typically satisfy it. And so, we sort of are interested in the small clauses and then we say, let us pick that variable that appears in more of the smallest clause. So this is the finally the dpll algorithm satisfiability