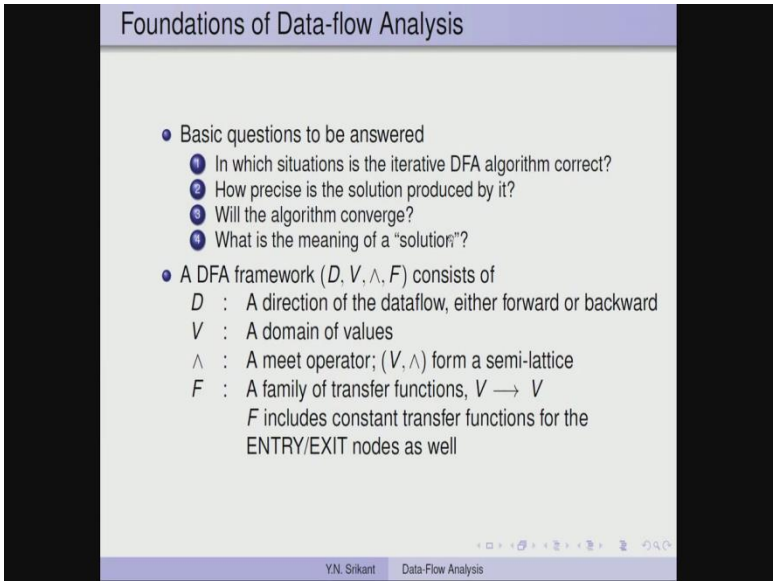
Principles of Compiler Design
Prof. Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Lecture - 34
Introduction to Machine-Independent Optimizations Part – 4

Welcome to part four of the lecture on Machine Independent Optimizations, today we will continue our discussion on data flow analysis it is theoretical foundations, and then move on to control flow analysis.

(Refer Slide Time: 00:34)

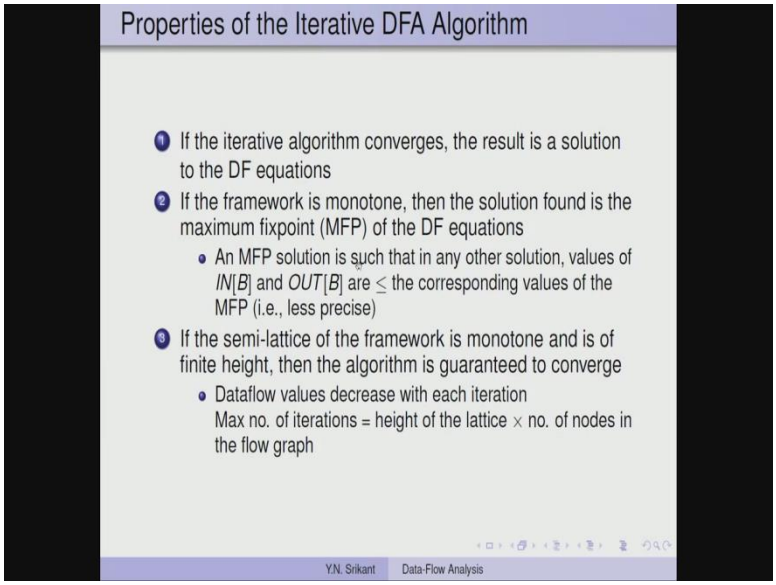


So, in the last part we begin our discussion on the theoretical foundations of data flow analysis. And a mention that there are some basic questions that we can answer using the frame work. So for example, when will I know iterative DFA algorithm be correct, how precise is the solution will the algorithm converge at all and what exactly is the meaning of a solution, these are the questions basic to answer using the frame work. So, and a frame work consist of a direction of data flow either forward or backward, a domain of values which is actually a semi lattice.

A meet operator, so V and the meet operator form a semi lattice and a family of for transfer functions. So, there is going to be one transfer function of each statement of the basic block and error at the whole program, we compose the functions of each of the

statements in a basic block and form a transfer function for the whole basic block. So, we also have these constant or identity transfer functions for the entry and exit nodes.

(Refer Slide Time: 01:51)



So, we discuss the frame work, the semi lattice and you know the iterative algorithm in the last part. So, now what exactly are the properties of the iterative DFA algorithm, a iterative DFA algorithm iterates over the statements of the program, applying the IN and OUT a constraints or equations. Until the solution converges to a fix point, so the first property of the iterative algorithm is, if the iterative algorithm converges we still have not given conditions on it is converges.

So, if the algorithm converges the result is a solution to the DF Data Flow equations, this is quite straight forward to understand. Because, if the algorithm converges then, you know the algorithm make sure that there is no change in the solution, rather in the value produced by the algorithm. And therefore, it must satisfies the data flow equations, if the framework is monotone, then the solution found is what is known as the maximum fix point of the data flow equations.

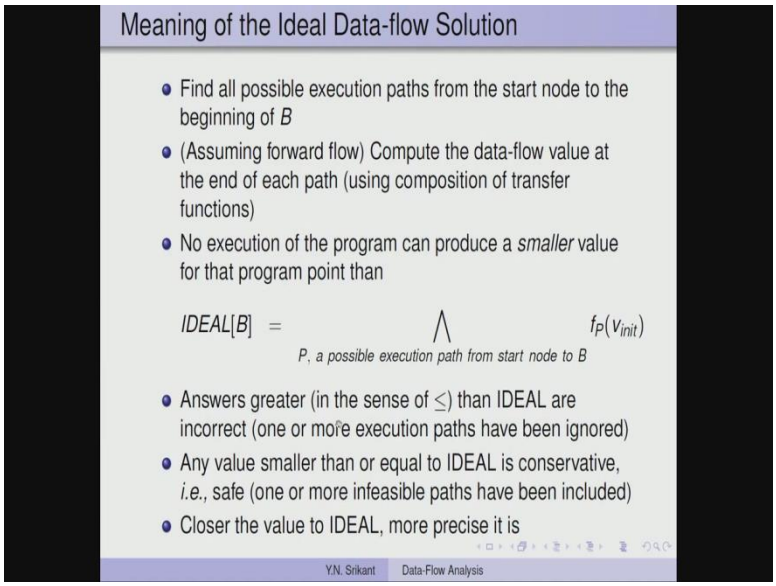
So, this is a very important condition the frame work to be monotone and what exactly is MFP. So, the Maximum Fix Point solution is such that, in any other solution the values of IN and OUT or less than or equal to that is less precise, than the corresponding values of the MFP. So, the MFP solution is the best that you can get by the iterative algorithm

and any other solution that we get would only be less precise, so this is the advantage of getting a maximum fix point solution using the iterative algorithm.

So, this is guaranteed by the frame work again, but the frame work must be monotone in this case, if the semi lattice of the frame work is monotone. So, this is a condition again and of course, it is of finite height most of our rather all of our semi lattices or of finite height, then the algorithm is guaranteed to converge. So, termination is guaranteed, so again intuitively the we can see that the data flow values decrease with each iteration, maximum number of iterations possible would be height of the lattice into the number of nodes in the flow graph.

So, why is this true the height of the lattice is the number of distinct values possible in the lattice one for each level. So, we start with the top and then we can go up to the bottom, we cannot go below that and as we already discussed the values actually you know decrease with each iteration. So, at some point they do not decrease any further, the rather the lowest solution that can be obtained is the bottom of the lattice. So, that would be the, so we start from the top and go up to the bottom that would be the height of the lattice and every time we are going to look at all the nodes in the flow graph, so this is the maximum number of iterations possible for the algorithm.

(Refer Slide Time: 05:30)



So, let us now discuss other possible solutions to the data flow problem and then understand the ideal solution, the you know maximum fix point solution and the MOP

you know solution as we call it. So, MOP solution would be the meet over all paths solution, so what would be the ideal data flow solution to a particular problem, suppose we can find all the possible execution paths from the start node to the beginning of a basic block B or a statement B.

So, this may or may not be possible in all cases, but let us assume that we can find all the execution paths from the start node to the beginning of B. So, if we assume forward flow then you know we can apply composition of the transfer functions to the data flow value at the end of each path. So, what we do is it consider every point on this each path, then compose the data flow values using the transfer functions at those particular nodes and at the end of the path, that is the beginning of B we would have actually got the composed value of the data flow from the start node to the beginning of B.

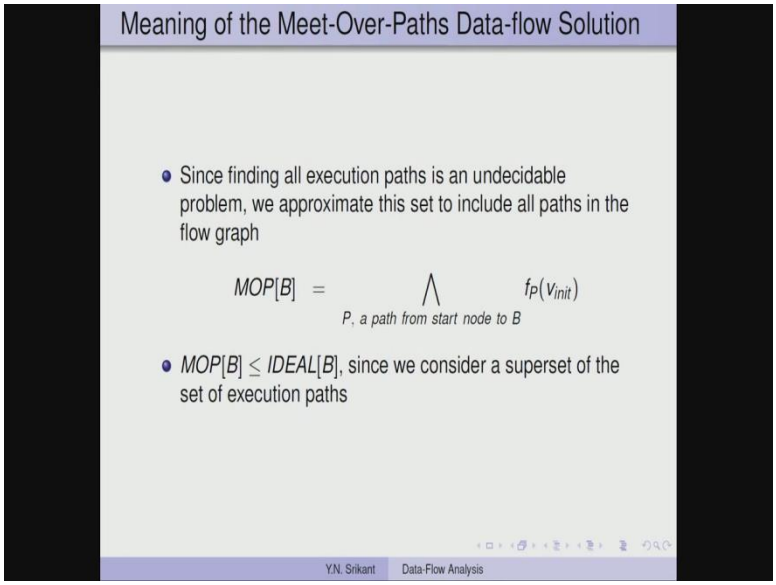
So, we have one such value for each of the paths which are converging at B, now no execution of the program can produces smaller value for that particular program point, than this equation what this equation can produce. So, IDEAL value of B that is possible is the meet over all the execution paths from start node to B of f p of v init. So, v init is the initialized value of the data flow entity, so we f p is the composed transfer function over the path p. So, we apply the trans compose transfer function to the initial value, so you get some value that is what we talk about here.

Then we take the meet of all these values for each one of the paths, we get the ideal data flow solution rather value, why is this ideal, we are considering the execution paths. So, and the data flow value that can actually accrue at execution time, so we really cannot get anything better. So, answers greater than IDEAL, so in the sense of the partial order relation or actually incorrect, so the point is we get a greater value only if we leave out some execution paths valid execution paths.

And any value which is smaller again in the sense of the partial order relation or equal to ideal is of course, fine, but smaller than the ideal solution is conservative. So, in other words it is safe, but it has infeasible paths have been included, so this makes it you know less precise. So, closer the value is to the ideal value more precise it would be, so if you include less and less number of infeasible paths, then you know the value becomes better and better. So, this is the meaning of an ideal solution, so you consider all the execution paths to a particular point. And then take the meet over all those execution paths of the

data flow value using the composition of the various transfer functions and the meet operator.

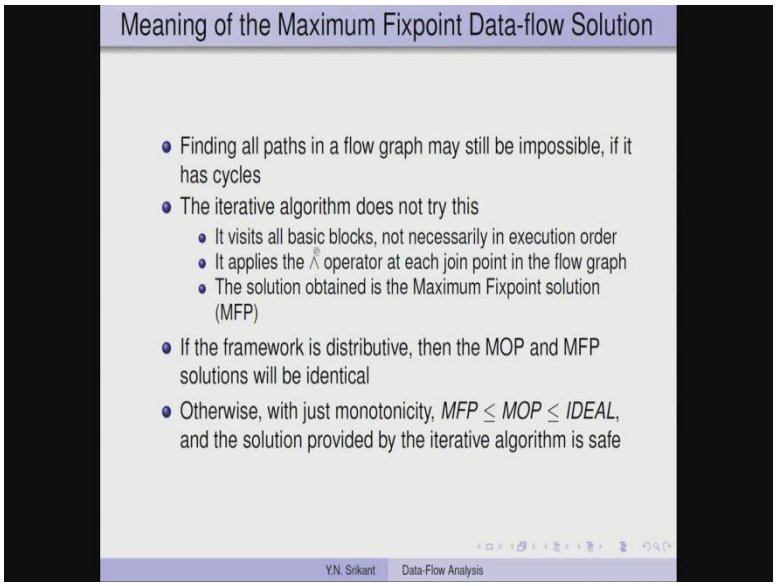
(Refer Slide Time: 09:42)



So, the next type of data flow solution is the Meet Over all Paths solution MOP as it is called. So, finding all execution paths is an undecidable problem, this is a very well known difficulty, so we do the next best that may be possible sometimes, we approximate this set to include all the paths in the flow graph itself. So, if we do that then it is called a meet over all paths solution, so again we consider a path from start node to B. So, it is not necessarily an execution paths some of these paths may not be executable also.

Again we apply composition and then of the transfer functions that could be f p over the initial value and we apply the meet operator. So, now, MOP will be less than or equal to the ideal value because, we considering a superset of the execution paths some of these paths may be infeasible. So, we have included those as well and; that means, we are approximating the ideal solution by the MOP solution and MOP is bound B less precise than the ideal solution.

(Refer Slide Time: 10:58)



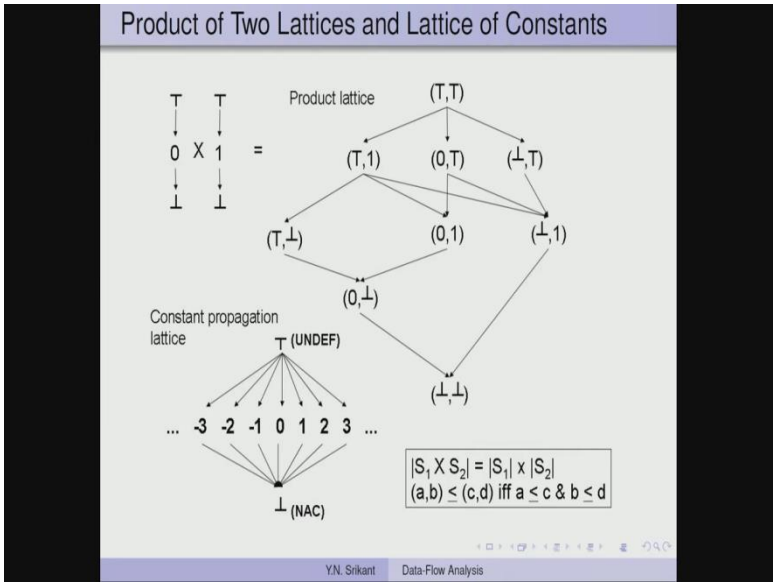
So, there is one more difficulty suppose there is a while loop in the program, then finding all paths in a flow graph may still be impossible. Because, we do not know the number of times a while loop would execute, we may not be able to say this is the number of paths in the flow graph from point to another. So, the next best is to apply the iterative algorithm, the iterative algorithm does not try to find the you know paths in the flow graph.

What it does is it visits all the basic blocks, not necessarily even in execution order in any random. It applies the meet operator to each join point in the flow graph, not necessarily trying to take the paths and then applied at the joint point, we just take the joint point and apply the meet operator, that we know already. The solution obtained is called as the maximum fix point solution, so this would be a little more approximate than the MOP.

But, there is a theorem which we are not going to prove, it is says that if the frame work is distributive, then the MOP and the MFP solutions will be identical. If the frame work is just monotone, but it is not distributive, then the MOP is superior to the MFP solution. Fortunately the reaching distributions problem, the available expressions problem and the live variable analysis problem, they are all distributive in nature therefore, for these problems the MOP and the MFP solutions will be identical.

We just monotonicity we are going to have this inequality MFP is a safe, but the least precise solution, MOP is next safe, but less precise than IDEAL solution and the IDEAL is best, but which cannot a actually achieve. So, the solution provided by the iterative algorithm is always safe that is the MFP solution, so we have answered a couple of questions, so for. We have talked about the quality of the data flow solution provided by the iterative algorithm, we have talked about the convergence of the data flow algorithm and we have also related the you know MFP to the iterative algorithm.

(Refer Slide Time: 13:39)



So, let us continue with our discussion on the theory of data flow analysis by considering a special type of algorithm for constant propagation. So, before discussing the constant propagation algorithm I must give you a little bit of background using the lattice diagram that is shown in the picture. So, here is the constant propagation lattice, so these are all the for each variable, which is a constant in the program we are going to have one lattice of this type.

So, these are all the values that the variable can take if it is a constant and if it is not a constant, then it would be given a value of bottom and if it is a undefined value then it would be given a value top. So, these are the only 3 abstract values possible, the top are an actual constant value or not a constant value bottom, so this is the lattice for each variable and what we want to do is to take a product of the lattices of all the variables.

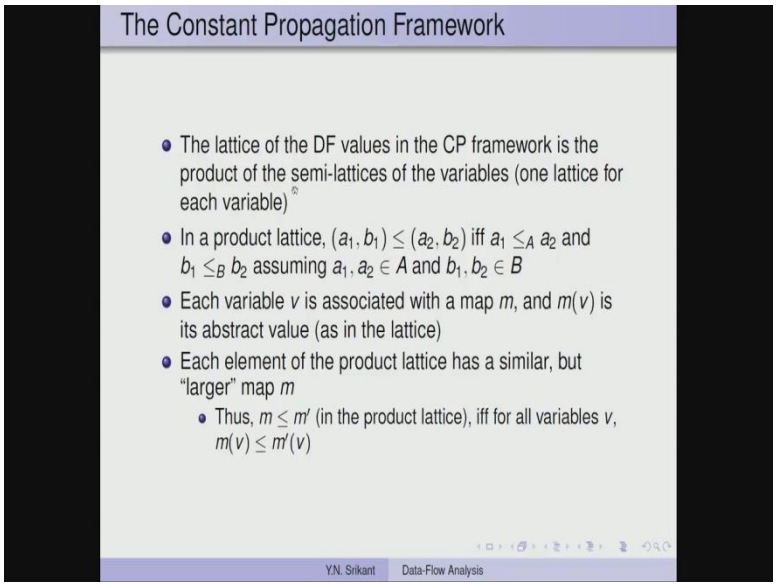
So, if there are 10 variables declared in the program, then we want to determine which of these 10 variables good indeed be a constant.

So, we will have to take the product of all the lattices of these 10 variables, this would be two last to show. So, let me show you a very simple product lattice, so this product lattice has a 0 here a top and a bottom, the other product lattice has a 1 a top and a bottom. So, it does not even have 0 and 1 in both of them. So, if you take the product lattice, the product lattice is defined as S 1 cross S 2, the size of the product lattice would be size of S 1 cross size of S 2.

And an element in the product lattice would be a pair a comma b, so that is shown here and the partial order would be a comma b is less than or equal to c comma d, if and only if both the components a and b are less than equal to c and d respectively. So, a less than or equal to c and b less than or equal to d, so the same holds here you know, so this constant value here. For example, 0 is less than top, then bottom is less than 0, 1 is less than, top and bottom is less than 1, so these are the partial order relations.

So, using this definition and all the possible values for the product lattice, we have just considered exhaustively all the possible values from top top to bottom bottom. And we draw this lattice, these are the various partial orders which are possible, so if there are you know a large number of such lattices taking a product of these we cannot really show it in pictorial form, but believe me that is the lattice we want to consider when we do constant propagation.

(Refer Slide Time: 17:13)



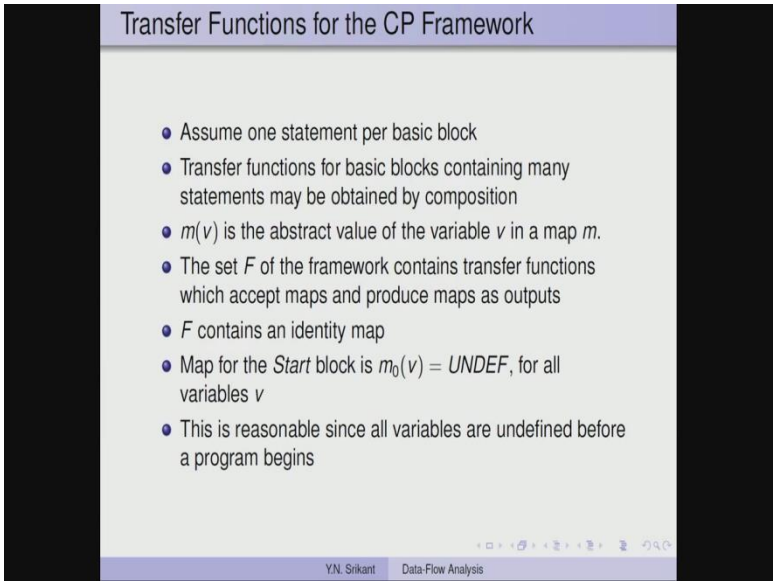
So, let us understand the constant propagation frame work, the reason we want to do it is the constant propagation frame work happens to be a monotone frame work, but it is not a distributive frame work. So, in other words when we apply the iterative algorithm for constant propagation, we get an approximate value it cannot catch all the constants in the program. So, there would be some which are left out, we will show you examples of such a situation very soon.

The lattice of data flow values in the constant propagation work is the product of the semi lattices of the variables one lattice for each variable I already mentioned this. In the product lattice, so even this has been mention, if you take 2 variables we have a 1 comma b 1. If you have 10 variables, then we would have 10 components in this entity a 1, b 1, c 1, d 1, etcetera. So, a 1, b 1 is less than or equal to a 2, b 2 if and only if a 1 is less than equal a 2 according to the a partial order and b 1 is less than or equal to b 2 in the b partial order.

So, each of these semi lattices for the variables may have a different partial order that is the most general case. Assuming a 1, a 2 in a and b 1, b 2 in b, so each variable v is associated with a map m and m v is the abstract value of the variable as in the lattice, so what is the abstract value, top the constant value or bottom, these are the three abstract values possible. So, there is going to be one function m v for each of these, you know variables rather there is a map m and m v is defined for each variable.

So, it could be defined different leaf for each variable, each element of the product lattice has a similar, but larger map m. So, this map m is in the product lattice, where has this map m is for each one of the variables separately, now in the product lattice when can we say a map m is less than or equal to the map m prime. So, we can say, so provided for all the variables v, we have m v less than or equal to m prime v, so this must hold for each variable m prime is a different map m is a different map. So, for these two maps if we take m v and m prime v, then the less than or equal to relation as in the map for the variables must hold. So, if this is possible for all variables, then the map m is less than or equal to the map m prime.

(Refer Slide Time: 20:29)

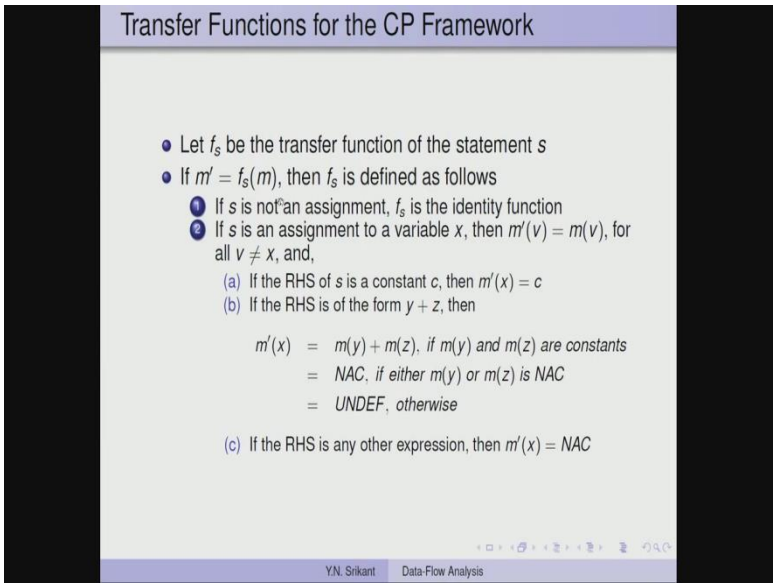


Why did we define the map that is necessary, because our transfer function would actually depend on these maps. So, to define the transfer function for the constant propagation frame work, let us assume that there is one statement per basic block and the transfer function for basic blocks containing many statements may be obtained by composition, this is something we already know m v is the abstract value of the variable v in a map m. So, and the set F of the framework contains transfer functions which accept maps and produce maps.

So, this F we are going to define the one sample transfer function of F very soon, so that would take m as a you know as an input and produce some other m prime as the output. So, F also contains an identity map which is required for the frame work and for the start

block this m naught v would be UNDEF for all the variables, this is reasonable because, all variables are undefined before a program begins, we assign values to the variables and thereby they get their values.

(Refer Slide Time: 21:45)



So, now let us define the sample transfer function of the set of transfer functions, so let f s be the transfer function of the statement s. So, for every statement s we are going to have such a transfer function and capital F will be the set of all such transfer functions. So, again as I said f s takes a map and produces a map, so m prime equal to f s m then f s is defined as follows, the first case is s is not an assignment statement at all. So, if it is not an assignment statement it could be a branch statement or an ordinary go to statement conditional go to these are all possible.

And then f s is the identity function, if it is not an assignment there is no side effect therefore, it is an identity function, if s is an assignment to a variable x. So, then m prime v is m v for all v not equal to x, so the assignment is being made to the variable x, so for other variables the map does not change that is what these says m prime v equal to m v for all v not equal to x. And for v equal to x it changes, again there are 3 cases here, if the R H S of s is a constant.

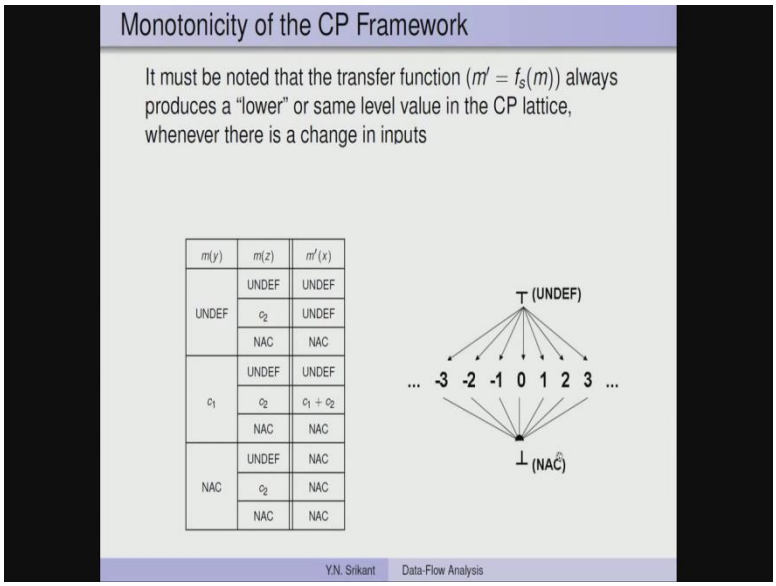
So, we have an assignment to x the right side is a constant, so we have a statement of the form x equal to c. Then m prime x will be; obviously, a constant, so in other words the initial map of x would be something different, but after the assignment x equal to c the

map of x would become c. So, from now on whenever you apply the map for x you get the constant value c, if the R H S is of the form y plus z, so in other words the assignment statement is x equal to y plus z.

So, if we apply m to y and we apply m to z, so and if m y and m z happen to be constants. So, in other words y and z have been assign constant values earlier that is when m y and m z can become constants just like this, so in such a case we add the two constants and that would become m prime x that is quite logical and intuitive. Because, if there are two constants, the new constant value would become the map of x, so if either m y or m z is determined not to be a constant, then the new map of x will also be n a c.

So, if neither of them is N A C and y and z are not constants, then we say that the new map of x would become UNDEF. So, these are the three possibilities and we have covered all of them, so both are constant, so we add the constants, one or both are not constant. So, that is a very strong statement, then x would become not constant and in other cases it would become UNDEF. If the RHS is any other expression, it is not an assignment of the form y plus z, it is not a constant, it is something else. In such a case m prime x would be NAC, so we do not want to take any chances we just make it not a constant.

(Refer Slide Time: 25:36)



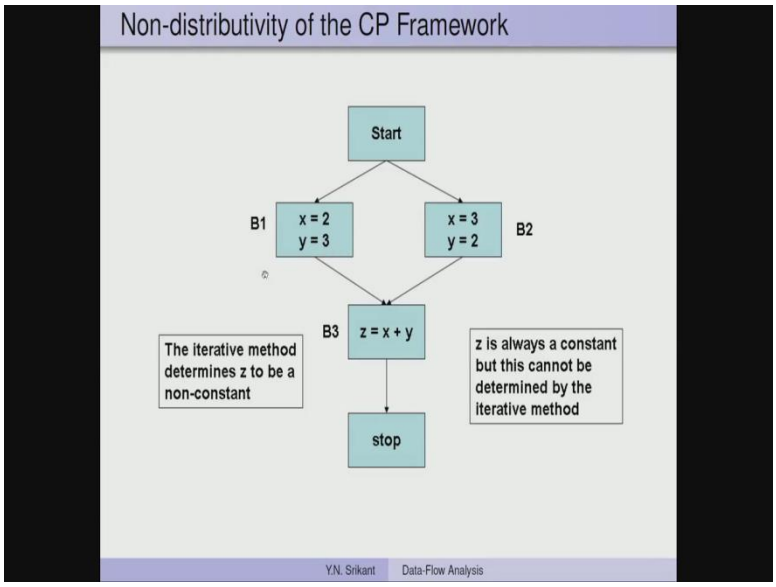
So, the constant propagation frame work is set to be monotone, in other words ((Refer Time: 25:46)) the transfer function that we discussed here always produces only you

know a is monotone. And therefore, the values produced by it will always produces a lower or same level value in the CP lattice, whenever there is a change in inputs. So, let us check it out, so let us say why changes from, so this is UNDEF, this is constant and this is not a constant.

So, if the input for y changes from UNDEF to c 1 all others being constants, then for each of these combinations check this UNDEF is UNDEF here, UNDEF is UNDEF here. So, it remains at the same level c 2 you know UNDEF and c 2 was producing UNDEF here, now we have change the input to c 1, so c 1 and c 2 will become c 1 plus c 2, so we have actually come down from UNDEF to c 1 plus c 2. So, lower value, third one whenever it is UNDEF and NAC it always be NAC, so with this NAC combination this part would be NAC is the lowest level and we get NAC here also.

The same is true from c 1 to NAC as well, so c 1 and UNDEF gets you UNDEF, NAC and UNDEF will get you NAC. So, we have come down from the UNDEF to NAC really c 1 and c 2 gets you c 1 plus c 2, since one of them is NAC and other is c 2 we get NAC, so that is lower and NAC and NAC will always get you NAC. So, in other words this statement is true, the transfer function m prime equal to f s of m always produces a lower or same level value in the constant propagation lattice, whenever there is a change in inputs. So, because of this the you know we have shown informally that the transfer functions of the CP frame work are indeed monotone.

(Refer Slide Time: 27:53)



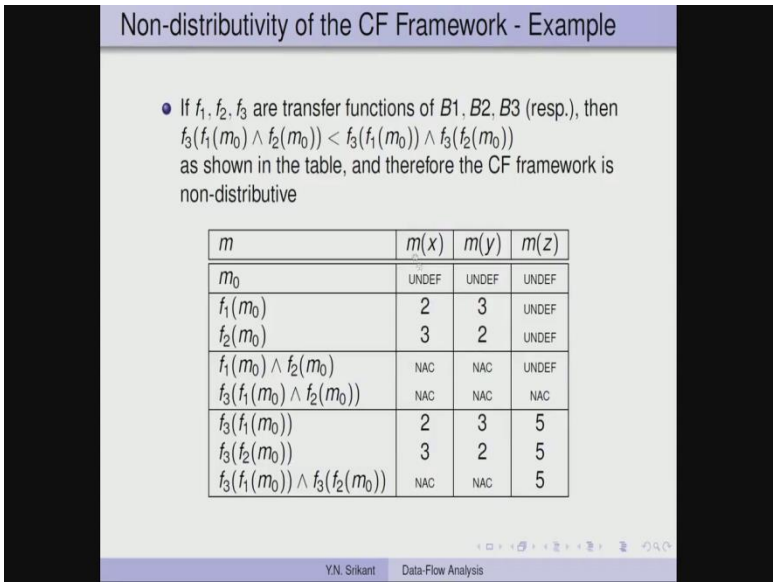
Now, let me give you an example of the non distributive of the constant propagation frame work. So, here is a very simple flow diagram, start this is not even you know it does not even have loops, so on this side we have basic block B 1 with x equal 2 and y equal to 3 and in B 3 we have z equal to x plus y. So, if we take this path x plus y indeed adds to 5 and on this path we have x equal to 3 and y equal to 2 just the inter change version of this and x plus y still adds up to 5.

But, unfortunately when we apply the iterative data flow analysis to this type of you know control flow graph for the constant propagation frame work. At this point we are actually going to take the join for the 2 x values which reach here for the 2 y values which reach here. So, since x carried a value of 2 here and x carried a value of 3 here, the x value here can only be set to be not a constant because, both are different constants, at this again similarly y equal to 3 here and y equal to 2 here.

So, the meet operation of 3 and 2 will produce not a constant value here NAC, so with x and y being NAC at this point z will; obviously, produce NAC. So, the iterative method determines z to be non constant, but z is always a constant and, but this cannot be determine by the iterative method, if you had taken the MOP solution. So, we would have taken this path separately and this path separately, so if we do that then when we take this path separately, we take the value of x value of y and then the value of z is computed that is 5.

And on this path we again take the value of x value of y and another value of z will be computed as 5. And now we take the meet of this two values at this point, the z 1 and z 2 both of which are 5 will; obviously, yield a meet of 5 here, but this was obtained using the MOP solution, but the iterative solution gives us not a constant NAC value for z. So, this is the problem with the constant propagation frame work, it is not distributive and therefore, the iterative algorithm for solving the data flow equations will not catch all the constants available in the program.

(Refer Slide Time: 30:56)



m	$m(x)$	$m(y)$	$m(z)$
m_0	UNDEF	UNDEF	UNDEF
$f_1(m_0)$	2	3	UNDEF
$f_2(m_0)$	3	2	UNDEF
$f_1(m_0) \wedge f_2(m_0)$	NAC	NAC	UNDEF
$f_3(f_1(m_0) \wedge f_2(m_0))$	NAC	NAC	NAC
$f_3(f_1(m_0))$	2	3	5
$f_3(f_2(m_0))$	3	2	5
$f_3(f_1(m_0)) \wedge f_3(f_2(m_0))$	NAC	NAC	5

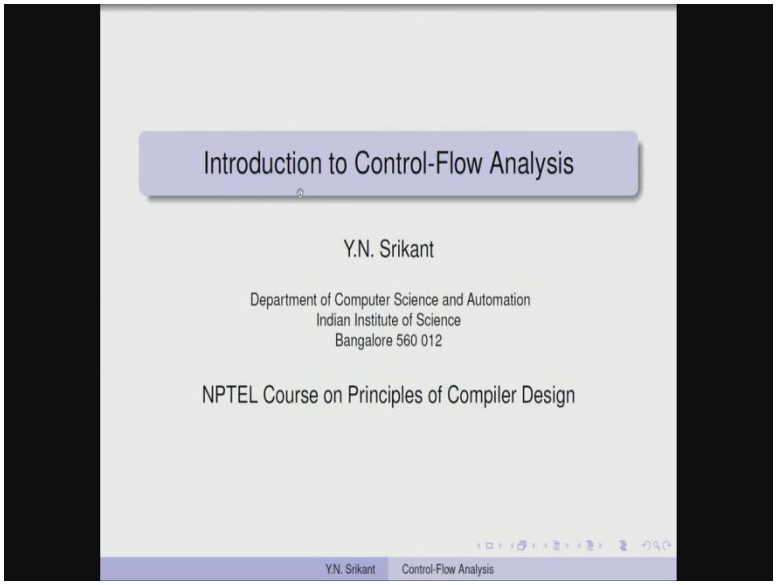
Here is a formal statement of the same property, if f 1, f 2 and f 3 are the transfer functions of B 1, B 2 and B 3 respectively. So, we apply f 1 of m 0 meet it with f 2 of m 0 and then apply f 3 to it, this is one possibility, the other one says take f 1 of m 0 apply f 3 on it take f 2 of m 0 apply f 3 on it and then take the meet operation. So, these two will produce different results, they should have produce the same result if the frame work was distributive, since it is not it produces different results.

So, we can check it out here m naught to begin with is UNDEF for all the three variables x y and z, when you apply f 1 then x gets a value 2, then in the y gets a value 3 and z has not been assigned anything. So, it is still has the value UNDEF, ((Refer Time: 32:13)) f 2 corresponding to this basic blocks, f 1 is this basic block, f 2 is this basic block, f 3 is this basic block. So, f 2 gets a value three for x, 2 for y and UNDEF for z, now if you take the meet of the two values f 1 m 0 and f 2 m 0, these two value being different this will be a NAC, the same is true here these two values are different.

So, this is a NAC and this two are UNDEF, so this is still UNDEF, now apply f 3 on the meet of f 1 and f 2, it gives you nothing better it is gives you NAC here, NAC here. And; obviously, f 3 apply to UNDEF will give you NAC here as well, where as if we had taken f 1 m 0 and then applied f 3 to it we would have got the z value as 5 of course, this two will remain 2 and 3. If we had applied f 3 to f 2 m 0 then we would have got to the z value as 5 here as well this two remain 3 and 2.

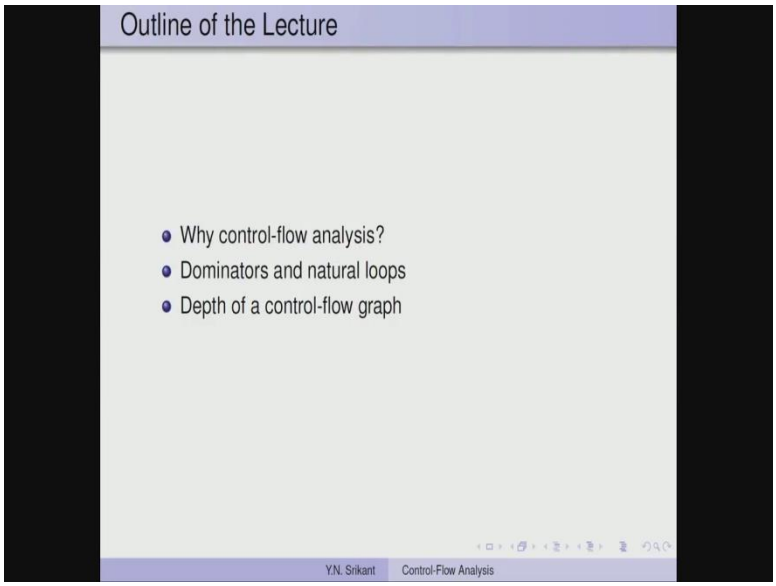
Now, take the meet of these two, so whatever we have done here, so these two; obviously, will produce NAC, these two will also produce NAC, but the z value being 5 and 5 would have produce 5. So, the value being produce by this method is different from the value being produce by this method and therefore, the frame work is not distributive, so this is a counter example to show that this is not distributive.

(Refer Slide Time: 33:52)



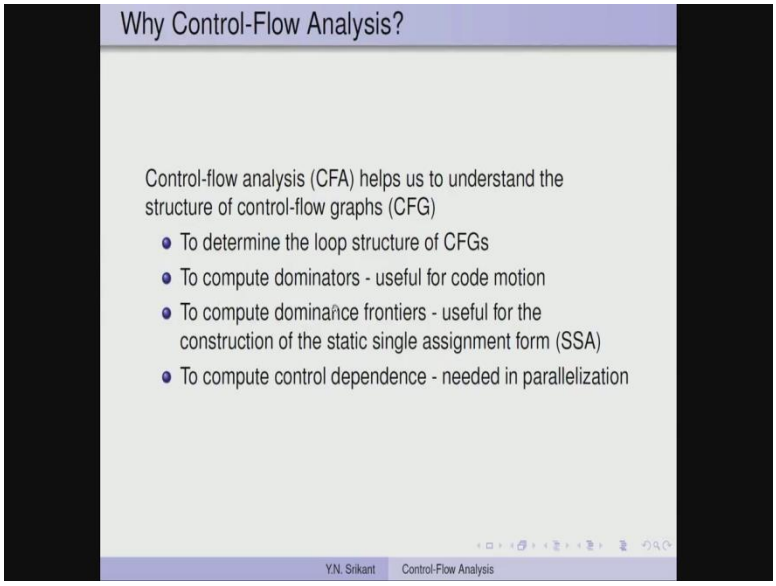
So, that completes our discussion on the theory of data flow analysis, so let us move on and consider the control flow analysis, which is also required before we study the optimization algorithms.

(Refer Slide Time: 34:08)



So, in this lecture we are going to understand why and what is control flow analysis and we are going to study the dominators and there use in producing natural loops. And we will also look at the depth of a control flow graph, which let us know how many iteration are needed for the iterative data flow analysis algorithm to terminate.

(Refer Slide Time: 34:36)



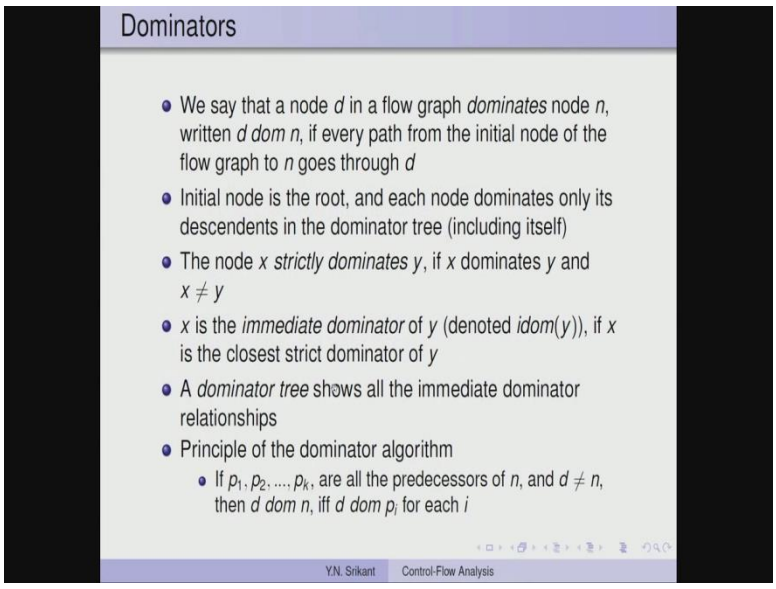
So, why do we require control flow analysis, so control flow analysis help us to understand the structure of control flow graphs. So, the first use of this would be to determine the loops structure of control flow graphs, so far we have only talked about

loops in the graphical sense. Whenever I show you a control flow graph we see a loop in the graphical sense, it goes back to one of the nodes in the you know of in the flow graph.

But, given a data structure for a control flow graph, how do we determine a loop, the reason we requires such a loops structure to be determine is that the optimization requires us to determine the nodes in a loop structure. For example, even the register allocation algorithm which we discussed, you know the usage counts based register allocation algorithm. It requires the loop structure to be known, we must know various basic blocks which are present in the loop.

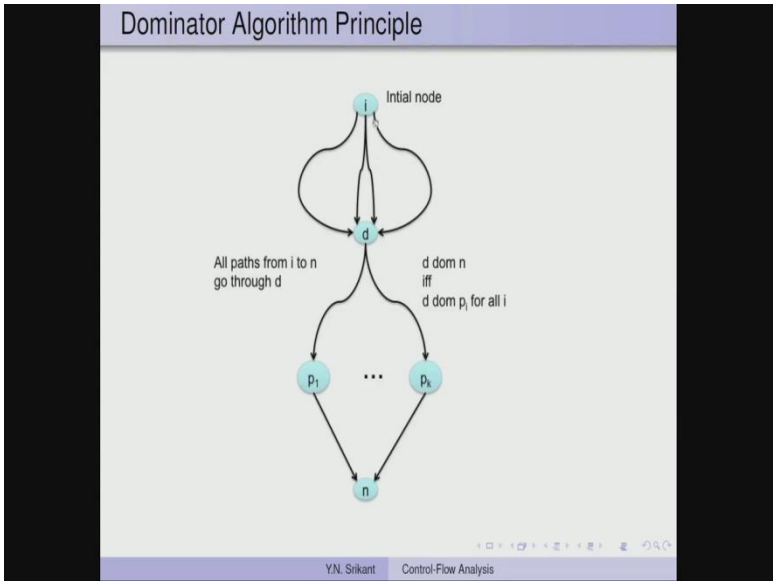
So, then the dominator relationship has to be defined, ((Refer Time: 35:55)) you know analysis allows us to determine dominators. And these dominators are very useful for loop invariant code motion, the conditions on code motion will all be defined using the dominator principle. We also require the control flow analysis the dominators and something more to compute what are known as dominance frontiers, this are useful in the construction of the static single assignment form, which is a very useful intermediate form, which permits many more optimizations. We are going to consider an examples of a SSA later and in parallelization we require the concept of a control dependence to state whether something can be parallelized or cannot be parallelized.

(Refer Slide Time: 36:49)



So, let us begin with the discussion on dominators, so let me show you a picture and then we will come back to the text.

(Refer Slide Time: 37:01)



So, here is a you know I show you a lot of paths in a control flow graph, here is an initial node the initial node rather. And there is a node d here, forget this p 1 and p 2 for the present and there is a node n here, what we want to define is the dominator relationship. So, we say that the node d dominates node n, if and only if all the paths from the initial node to the node n go through d, so for example, there are lots of pass which are emanates from the initial nodes, so many of them.

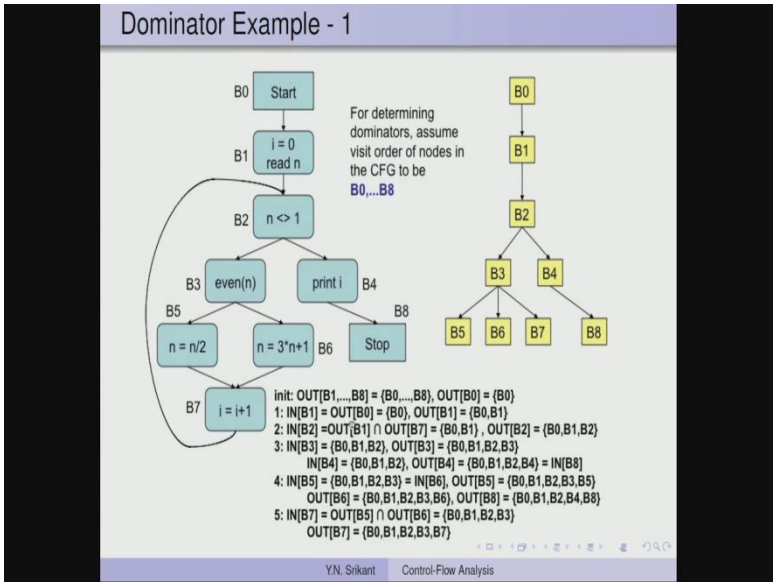
But, they will all converge a d and then all the you know from d to n there are many paths of course, we are not bothered about how they actually diverge or converge. Our important point is whenever we start from i to reach n we must definitely pass through d that is the only requirement as for as the dominator relation is concern. So, d dom n provided all pass from the initial node to n go through d, so this is the definition of the dominator relationship, which is actually termed as a dom ((Refer Time: 38:21)).

Now, the initial node is the root and each node dominates only it is descendents in the dominator tree, I have not yet shown you dominator tree, we will do that after this definitions are over. So, assume that there is dominator tree in which a know dominates all it is decedents and; obviously, the initial node will be the root of such a dominator

tree. The node x strictly dominates another node y, if x dominates y and x is not equal to y, so this is strictly dominates relationship x is not allow to be same node y.

And then we have the immediate dominator relationship x is the immediate dominator of y denoted i dom y, if x is the closest dominator of y. So, I will show you example of this very soon and a dominator tree shows all the immediate dominator relationships, it does not show the transitive relationships, it will shows only the immediate dominator relationship.

(Refer Slide Time: 39:46)



So for example, here this is the dominator tree for this particular control flow graph, so suppose we consider the initial node B naught it; obviously, dominates all the nodes in the flow graph. Because, every path starting from the initial node to any other node must actually passed through the start node, so that is why B naught is the root of this dominator tree. So, let us consider B 6, so B 6 is a leaf node here, so if and the immediate dominator of B 6 is B 3, so this is what is shown here.

Immediate dominator of B 3 is B 2, immediate dominator of B 2 is B 1 and that of B 1 is B 0. So, in other words the dominator of B 6 are B 3, B 2, B 1 and B 0, let us verify that here, B 0 is very trivial and B 1 is also trivial because, there is only one path from start to B 1. So, all path starting from start to B 6 will defiantly have pass through B 1, then we have B 2 again there is only one path from B 1 to B 2, so starting from B 0 to go to B 6 will also have to pass through B 2.

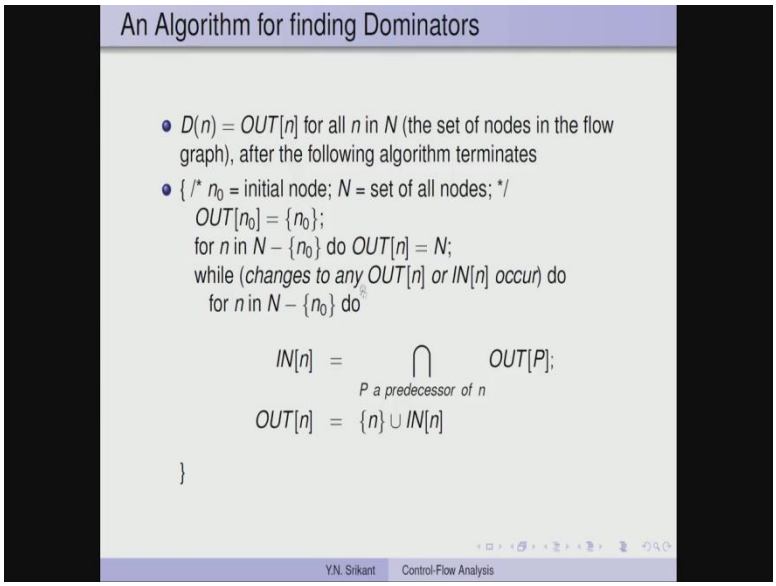
So, that is, so far now from B 2 how do we reach B 6, so; obviously, if we go this way we cannot reach B 6. So, we will have to go through B 3; that means, again B 3 will be a dominator of B 6, then if we go through you know we start from B 0, then B 1, then B 2, then B 3. Let us say we go to B 5, B 7 and then we go back to B 2 and then again B 3 and B 6, so this is not a compulsory path, you know going via these nodes is not compulsory.

Therefore, since every path from the start node to B 6 does not contain B 5 and B 7, B 5 and B 7 do not dominate B 6 only B 3, B 2, B 1 and B 0 dominate B 6. So, this is very clear, but you know if you consider B 7 again all paths from B 0 to B 7 do not go through either B 5 or B 6. But, they definitely go through B 3, B 2, B 1 and B 0, so again those are the ones which are dominators of B 5 and B 7 ((Refer Time: 42:43)) now let us understand a how to compute the dominator.

So, the principal of the dominator algorithm is very simple if p 1 to p k are all the predecessors of n node n and d is not equal to n then d dom n if and only if d dom p i for each i. So, let me show you a picture, this picture and then explain what the statement means, ((Refer Time: 43:08)) so we have d here, we have n here. So, from the initial node any way the pass go through d, but the algorithm cannot determine all the paths and then determine the dominators.

So, it has to be incremental in some way, so what is does is it considers n and all the predecessors of n. So, if there are k predecessors p 1 to p k then, so d dominates the node n if and only if d dominates every predecessors of the node n, so if one of the predecessors is actually not dominated by d, then you know we could have taken that path from i to the predecessors without going through d and then go on to n. Therefore, the d would not have dominated n in that case, this shows that there can be no predecessors of n, you know which is not dominated by d it must dominate all of them, so this is the basic principle of the dominator algorithm.

(Refer Slide Time: 44:25)



So, let us you know, so after this algorithm that we have provided here terminates, the set of dominators of the node n would be the outset of node n. So, this capital N is the set of all the nodes in the graph and for every node d n equal to OUT n for all n in n, so this algorithm looks very bright. So, this is actually just like a data flow analysis algorithms, in fact, it is a data flow analysis the algorithms, so let us identify the components of the data flow analysis algorithms.

So, we have out of course, and we have in then we have initializations for the out node, so out of n naught is n naught permanently. And initialization for n in N minus n naught would be capital N, so capital N is very similar to the universal set, so this is initialize to the universal set. Now, there is a while loop, while changes to any OUT n or IN n any occurs, so this is the loop of the iterative algorithm and we have OUT n equal to n union IN n.

So, in other words this is the OUT which is a function of IN, so this is again you know it is a forward flow problem because, we are computing OUT in terms of IN. And in is considered as the intersection of the predecessors of n, so out of predecessors of n, so the confluence operator happens to be intersection. So, this is very similar to our available expressions problem, in which it is a forward flow with intersection and that is the reason why because of this you know the confluence operator being intersections, we have initialized the OUT n into the universal set N.

So, the OUT of n node n is the self node union the in node there is nothing to be killed, kill is really phi here. Dominator algorithm does not require a kill, it only requires gen which is the self nodes, so this takes care of every node dominates itself and IN is the incoming set of nodes, the rather the set of nodes which dominate the incoming point of the node n. So, whatever dominates at that point will also the actually transmitted to the out part, so this is why and we add n to it which takes care of the reflexive nature.

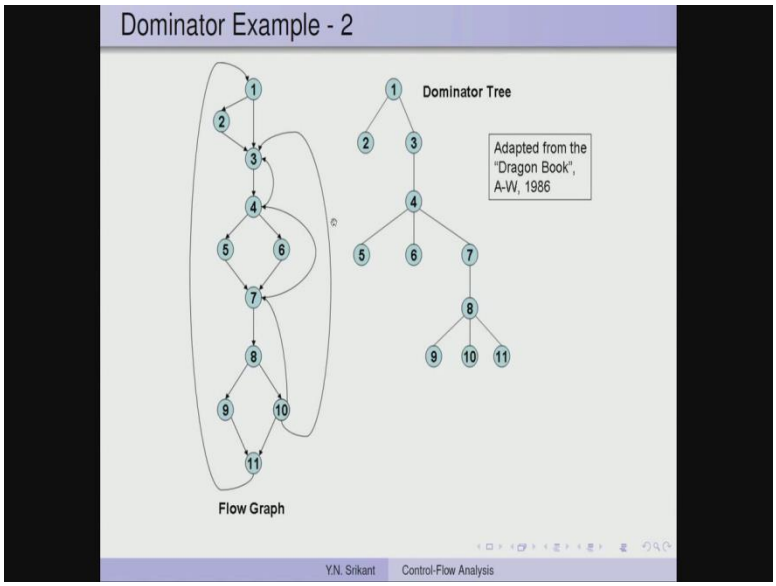
And IN of n would be the intersection of OUT P, so again OUT P is a predecessors, so we go back to this.((Refer Time: 47:46)) So, we actually have OUT here in these places and this says if you know whatever elements are common among the dominators of all this. So, the nodes which dominate of all of them is obtain by intersecting this out sets and that is why the in part is intersection of the out sets, so this is how the data flow analysis algorithm computes the dominators, so let me show you an example ((Refer Time: 48:20)).

So, here initialization would be the OUT of all the elements B 1 to B 8 would be set to B naught to B 8 OUT of B naught will be set as B naught permanently. So, this is our universal set B naught to B 8, the order in which we visit the nodes let us assume that it is B 0, B 1, B 2, B 3, B 4, B 5, B 6, B 7 and B 8, so in that order. Now, when we, so the dominators of this OUT B 0 would b set of B 0 nothing more to do the IN of B 1 is set of OUT of B naught, so that would be B naught.

And the OUT of B 1 would be the IN union with B 1, so we get B naught comma B 1, similarly IN of B 2 we have actually two incoming arcs here, one from B 1 the other coming from B 7. So, we must take the out sets of B 1 and B 7 and intersect them, so that gives us a you know this set is the universal set that B naught to B 7 B 8 rather, so we really get B naught B 1 and as the IN set, OUT set we add B 2 to do it and we get B naught B 1, B 2, so this you know is very simple to compute.

So, let us take B 7 for example, IN of B 7 again we have two of them coming in, so we must take the intersection of this two. So, OUT of B 5 intersection OUT of B 56, so that gives us B naught B 1, B 2, B 3 and then the OUT of B 7 we add B 7 to this, so that gives us B naught B 1, B 2, B 7, B 3, B 7. So, one more iteration would be required in order to stabilize these values and then the values do not change any more, so this is the dominator tree that we get from the computation in this case.

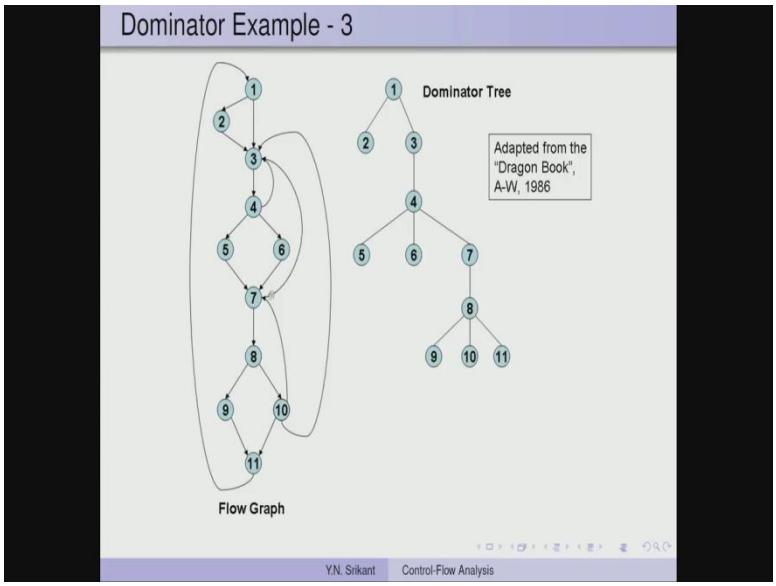
(Refer Slide Time: 50:41)



Another example, this is an example which is adapted from a ((Refer Time: 50:47)) book, so here this is a you know it has many loops and, so on and, so forth. We have still not defined loops, but that does not matter to us, the node 1 is the initial node and that dominates all other nodes in flow graph that is very trivial. So, let us consider node 9, so the dominators of the node 9 or 8, 7, 4, 3 and 1 this are all the dominators, so we are here.

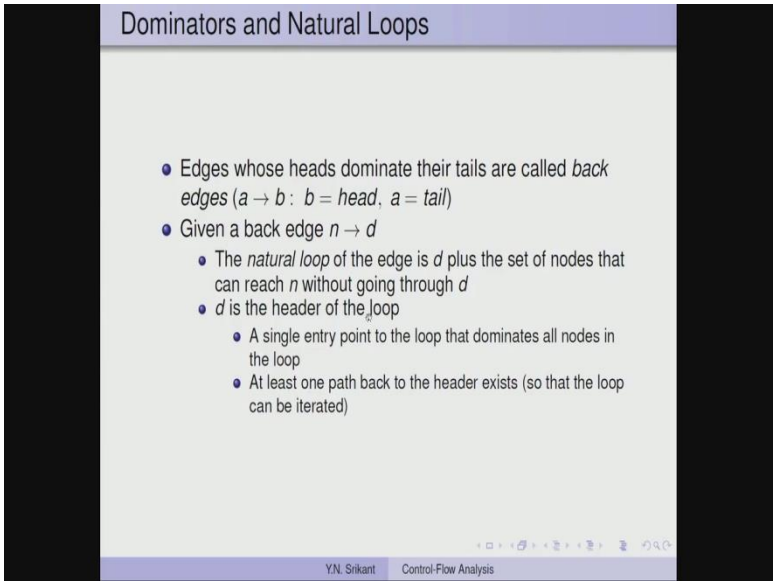
So, it is very obvious that we will have to go through 8 in order to each 9 that is very simple to understand and to get to 8 we need to go through 7. So, that is also easy, but to get to 7 we do not have to go through either 5 or 6, we must definitely go through 4, so, 4 is also a dominator to reach 4 we must go through 3 and to reach 3 we do not have to go through 2. But, we definitely have to go start from initial node, so that is the reason why 1, 3, 4, 7 and 8 are the dominators of node number 9.

(Refer Slide Time: 52:06)



So, remember that this there are two loop structure here 1 and 2, suppose the loop structure change. So, this changes this is remain as it is, so observe that a only this was change, this change to this, there is absolutely no change in the dominator relationship, so the loop did not contribute anything to the dominator relationship.

(Refer Slide Time: 52:29)



So, having known what exactly is a dominator, now we are ready to define the loops structures it is called as a natural loop. So, to define a natural loop we must first define what is known as a back edges, so edges whose heads dominate their tails are called back

edges. So, for example, if there is an edge from a to b then this b is called as the head and a is called as the tail, so the head must dominate the tail that is called as a back edge.

So, let us understand the back edge here, ((Refer Time: 53:13)) so let us look at the dominator tree, so here is an edge from 4 to 3 and 3 dominates 4. Therefore, 4 to 3 is a back edge, but 3 to 4 is definitely not a back edge even though you know, so that tail 3 dominates 4. So, this is not what we wanted, where as in the case of 4 to 3, it is 3 which is the head which dominate 4 the tail, so similarly if you consider the edge from 10 to 7, so we have 7 here it dominate node number 10 which is the tail, so 10 to 7 is also another back edge.

And if you look at the edge from 7 to 3, so 3 dominates 7 therefore, 7 to 3 is yet another back edge. If you look at this edge from 10 to 3 all together 3 dominates 10, so 3 is here 10 is here, so 10 to 3 also another back edge and; obviously, 11 to 1 would be a back edge because, 1 dominates all the nodes in the graph 11 inclusive. So, if you are given a back edge, now let us define the natural loop of the edge, so given a back edge n to d, the natural loop of the edge is the head d plus the set of nodes that can reach n, without going through the node d.

So, in other words in some sense the loop hangs from the node d, so all the nodes which are hanging from d and in some way related to it will be the natural loops. So, the set of nodes that can reach n without going through the node d, so the node d is called as the header of the loop. And the significance of the header is as follows, it is a single entry point to the loop that dominates all the nodes in the loop, so this is a first property of the header.

And there is at least one path back to the header, so that the loop can be iterated, ((Refer Time: 55:39)). So, informally when you look at it you know, so if you consider this node from 7 to rather this back edge from 7 to 3, the loop consist of 3 and a couple of other nodes through which we can reach 7. So, we are going to look at the details of there are many other node which need to be looked at, we are going to consider the algorithm for computing the natural loop in the next part of the lecture, so this is the end of today's part.

Thank you.