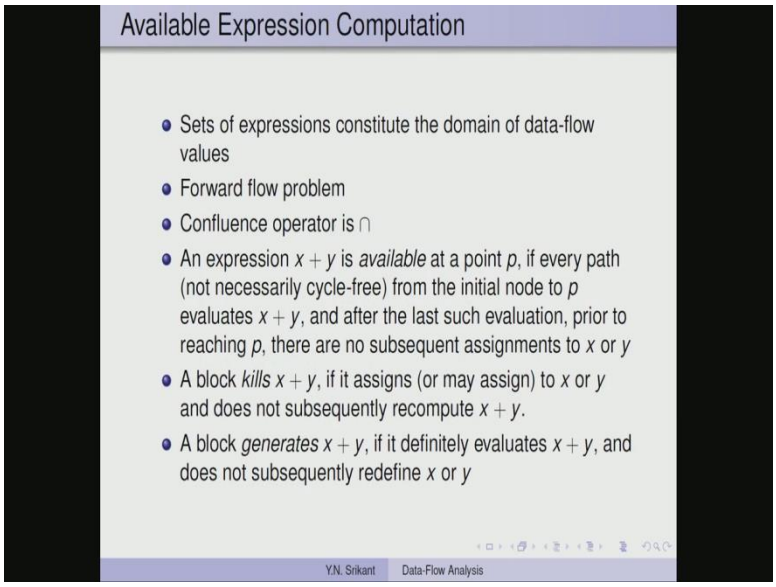
Principles of Compiler Design
Prof. Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Lecture - 33
Introduction to Machine-Independent Optimizations Part – 3

Welcome to part three of the lecture on Machine Independent Optimizations. Today we will continue with our discussion on data flow analysis.

(Refer Slide Time 00:29)



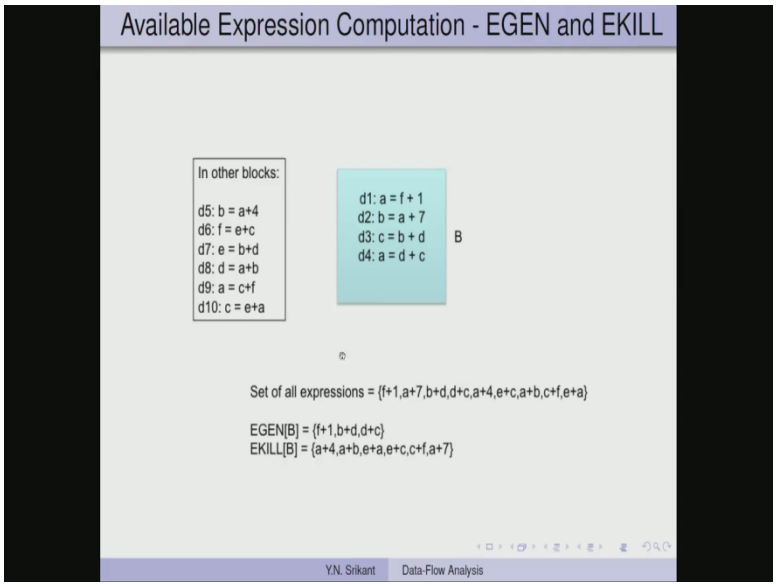
We were looking at the available expression computation, so in this problem you know an expression x plus y is available at a point p, if every path from the initial node to p evaluates x plus y. So, this is very important, because we should not be considering expressions which are on some infeasible path, so we want to make sure that the path is beginning at the initial point and terminates at p and that path should evaluate x plus y.

And further it is also important that the value of x plus y does not change after the evaluation. And such a change can happens only if x and or y are assign values, after the evaluation and again x plus y is not reevaluated, so we want to prohibit such subsequent assignments to x and y after the last such evaluation. So, the domain of data flow values; obviously, would be the sets of expressions, this is also a forward flow problem, in other words our equations will indicate that out of b will be a function of in of b.

The confluence operator is intersection I will explain why this is essential in the next few slides. Before we look at the equations, let us understand what exactly we mean by you know killing and generating expressions, so when can we say a block kills an expression x plus y well. Obviously, if we assign a value to either x and or y in the block and we do not subsequently re compute x plus y, then you know we can say that the block kills the expression x plus y.

Similarly, a block generates x plus y if it definitely evaluates x plus y that is no pointer problems here, we want to make sure that there is a definite evaluation of x plus y and there is no subsequent redefinition of either x or y. So, here the generates part we want to evaluate and then we do not want to redefine x or y, in the kill part you know a block kills x plus y, if it assign to either x or y and does not re compute x plus y.

(Refer Slide Time 03:34)



So, here is a very simple example, it is an extension of the example I gave you for the reaching definitions problem. So, you can see the you know 4 statements, the first one has evaluation of f plus 1, second one has a plus 7, third one has b plus d and the fourth one has d plus c. But, what is important here is that there are other expressions in other blocks, so the set of all expressions would be you know f plus 1, a plus 7, b plus d, d plus c and then a plus 4, e plus c etcetera b plus d is not to be repeated again. So, we leave that because, it has already been covered. So, we go to a plus b, c plus f and e plus a.

So, all these are the expressions which subsets of this set form the domain of data flow, you know the actually data flow values and all the subsets of these expressions form the domain of data flow values, what about the generation and kill. So, let us start looking at each one of these quadruples one at a time, so now, let us compute gen first, so we have a equal to f plus 1 and then we do not have any redefinition of f here, so f plus 1 is definitely generated by this block.

The second one is b equal to a plus 7, so in this case a is redefined later in d 4 and a plus 7 is not recomputed after words. So, definitely a plus 7 is not generated by this block, c equal to b plus d, so we have not assigned anything to either b or d you know in this block. So, b plus d is definitely generated by the block, then a equal to d plus c, so again we have not assigned either to d or c in this block, so d plus c is definitely generated by this block.

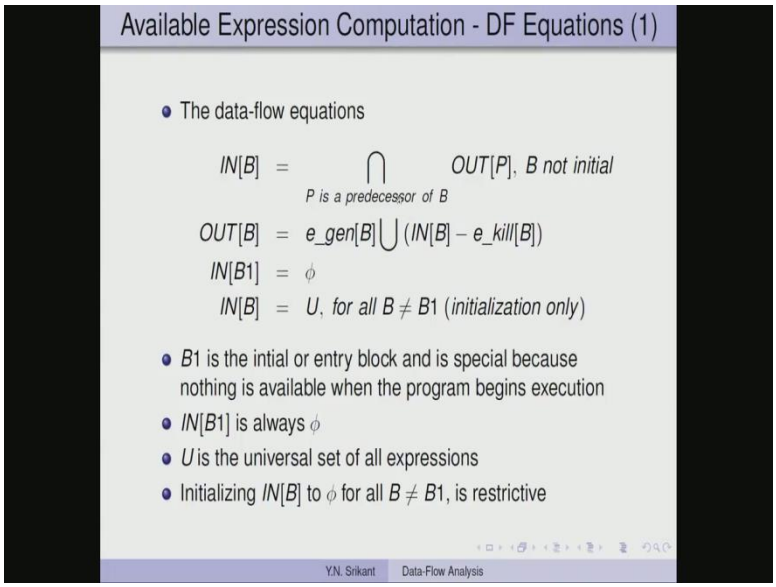
So, when it is generated then the point is all these expressions will be available at the end of the basic block that is the understanding that we have here, this is a forward flow problem. So, what is generated here is visible at this point that is the understanding, how about kill, kill is again similar inspirit to the kill that we had in the reaching definitions problem. So, this assigns to a, so a equal to f plus 1, so all the expressions, which actually involve a will be killed by this particular statement.

So, I must hasten to add that you know this a plus 7 should not be killed because, it is after the definition. So, we must consider only those which are prior to this particular definition or in other blocks, so here we have a plus 4, then we have a plus b and then we have e plus a. So, all these involve a, so these three expressions are definitely killed by the basic block, then we have b equal to a plus 7 again b plus d is not to be taken as kill simply because, b is assigned value in this block it is not, so because, b plus d is being evaluated after the assignment to b.

So, we again consider the expressions which involve b, so b plus d is not to be included, but a plus b has already been included. So, and that is it, the third one is c equal to b plus d, so we have again d plus c being evaluated after c, so that is not killed where as there are many others that is e plus c here. And then we have c plus f here, so both these are killed by the basic block finally, we have a equal to d plus c. So, this kills, this a plus 7 as well because, a plus 7 appears before the assignment to a. So, a plus 7 and others have

already been included you know a plus 4 a plus b and e plus a have already been included. So, the kills set does not include them again, so this is the kill set and this is how it is computed using the quadruples in the basic block.

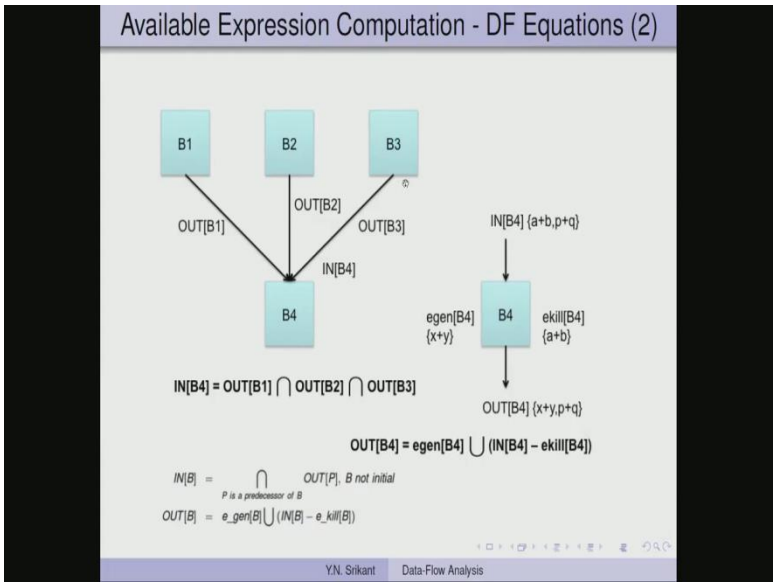
(Refer Slide Time 08:37)



So, let us look at the data flow equations, we have an equation for IN B and another equation for OUT B IN of B 1 has been permanently assigned phi. So, this is something very important, so the set of expressions coming into the basic block is a sign phi and that remains as phi in all our. ((Refer Time: 09:10)) The second is IN of B is equal to the universal set of all expressions for all B not equal to B ONE, in the case of reaching definitions, we had set this as phi, but for that had you know it is a different problem all together.

So, in this case we have as U, so this is something very important, whenever we have the confluence operator as intersection, we must initialize to U and whenever we have the confluence operator as union we initialize to phi. So, this is the general rule and reasons for this are beyond the scope of this lecture, so in the case of OUT B this is a forward flow problem. So, OUT B is a function of IN B it is very similar to the equations that we had for the reaching definitions, we have e gen of B union IN B minus kill B. So, it is very similar whatever you generated by the block (( )) with whatever is coming from the top with e kill removed, I will give an example of this very soon IN B is an intersection of OUT B being a predecessor of B.

(Refer Slide Time 10:41)



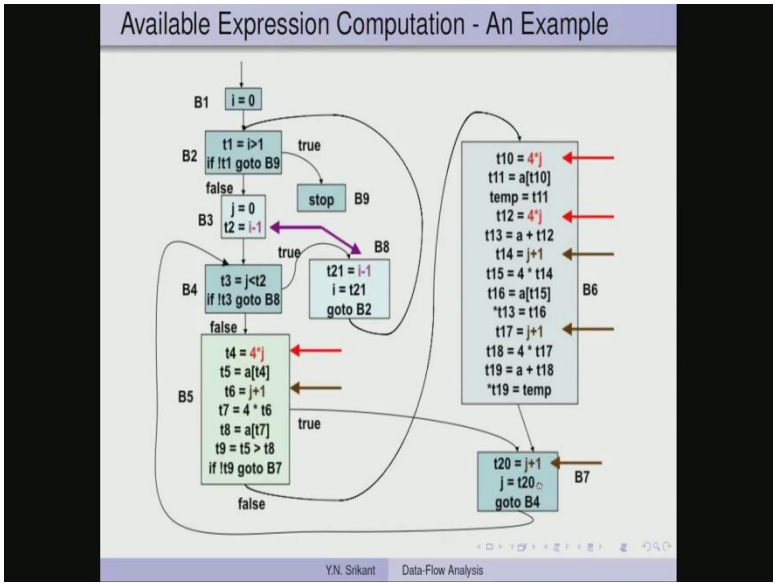
So, let me go to the example here, so we want to compute the inset for the block B 4 there are three predecessors. So, B 1, B 2 and B 3, they already have the sets available OUT B 1, OUT B 2 and OUT B 3 and at this point we want find out the set of expressions which are available. Since the major application of the available expression problem is for common sub expression elimination and as I already explained we would like the expression to be evaluated along all paths reaching the block.

It is only fair that in the available expressions problem also we want to make sure that the expression is available along all the 3 paths and this is the reason for taking the intersection of these 3. So, if you take the intersection, then the expression which is available here and here and here will be available here, but if an expression is available on at 1 or 2 of these points, but not the third will not be available here. So, therefore, common sub expression elimination will be done, only if all the 3 predecessors have computations of the expression.

So, this is what we wanted, so it is driven by our application, so this is how in is computed, now coming to the out computation. So, whatever is generated by the basic block which is e gen, so in this case it is the expression x plus y that will be in the out of B 4 definitely according to this equation. And that is fair as I explained whatever is generated here, will definitely be visible at this point. Then there are some expressions coming into the basic block via in.

So, let us say they are a plus b and b plus q, so out of this, this block is killing a plus b, so possibly it is assigning to a and b. So, a plus b will not pass through the block a and b are being assigned here, so a plus b will be killed, therefore only p plus q will be passing through the block and that will be included in the outset. So, to repeat whatever is generated by e gen and then whatever transparently passes through the block will be included in the outset that is the reason why we have e gen union in minus kill as the equation for computing the out.

(Refer Slide Time 13:22)

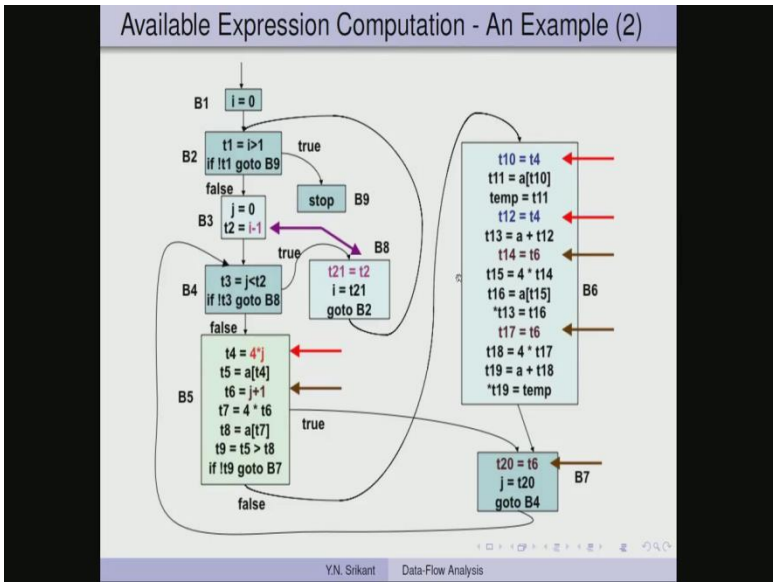


So, let us take the you know bubble sort example again, so in this example there are many places where common you know the available expressions are computed and common sub expression elimination is performed. So, i minus 1 is computed here i minus 1 is being computed here as well and if you observe the flow i minus 1 actually goes like this and it will be definitely, this is the only path to this block from here. So, right from the top we can actually go to this particular thing like this.

So, i minus 1 will be available at this point, this i minus 1 will be available at this point, so this will become redundant. Similarly, 4 star j is being computed here, here and here, so if we consider the availability of course, right from the top, you know 4 star j will actually be available along this path, this is the only path to this block. So, this will be available we can make sure of that by applying the, you know computation algorithm, but it is of it is easy to see that it will become available here.

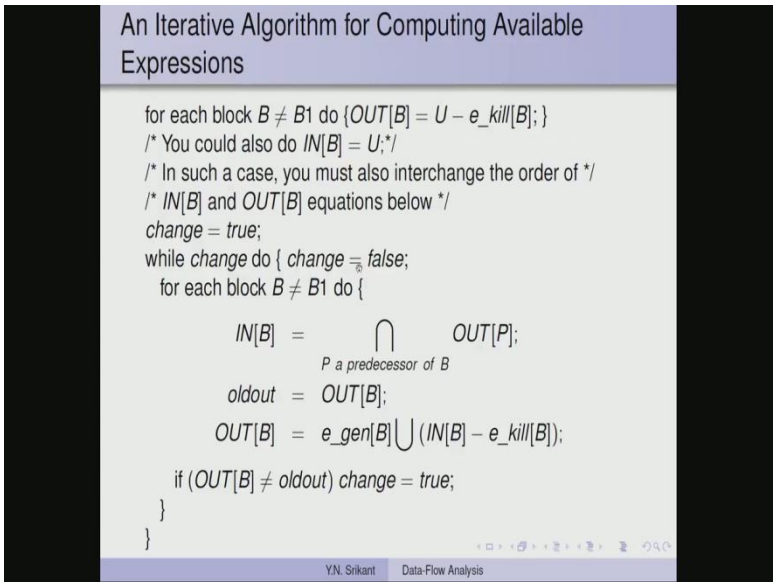
So, this and this become redundant computations, similarly we have j plus 1 which will also become available at the beginning of the block and it is not reassign. So, this also become redundant there is a j plus 1 here as well, so j plus 1 very trivially is available along this path this j plus 1 is available and this j plus 1 is available trivially along this path. So, again along both the paths a plus 1 is available and therefore, this becomes a redundant expression.

(Refer Slide Time 15:25)



So, if once the redundant expressions are removed we can say replace this by t 4 and this by t 6 and so on and so forth this will of course, get removed and replaced by t 2.

(Refer Slide Time 15:37)



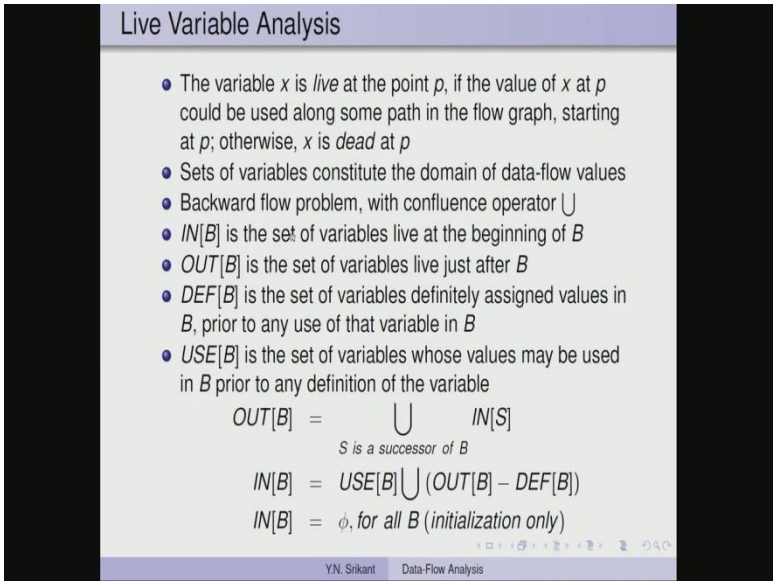
So, let us look at the iterative algorithm for computing the available expressions, so this has the same format as the reaching definitions algorithm. The initializations are different though for each block B not equal to B 1 we initialize OUT B to U minus e kill of course, this is very simply taken from this. So, OUT B will be IN B minus e gen union IN B minus e kill. So, if IN B is initialized to you then you know this can be regarded as a e kill as a constant, so IN B minus e kill will become U minus e kill e gen does not contribute anything once we have the universal set here.

So, we could have instead of doing this we could have said OUT B equal to you know we could also have said IN B equal to U and not initialize OUT B at all. But, in that case we must reverse the order of these two equations, interchange them and put OUT B first. So, that OUT B get's computed because, we are not initializing it IN B could be initialize to U as I told you before, the reason why we want to initialize IN B to U is to make sure to increase the precision of the solutions.

So, if IN B is actually initialize to phi then the solution is not incorrect, but it becomes less precise then is possible with you. So, as usual we have the flag change which is said to true and we have the same while loop with you know changed becoming false and then for each block B not equal to B 1. Because, we do not want recompute anything for B 1 it is already been computed, we compute IN and then OUT and we keep the old value of OUT and we keep the old value of OUT in old out. So, if OUT B not equal to

old out change equal to true, so this keep happening until change becomes false and then the equations would have reached their fix point solutions.

(Refer Slide Time 18:02)



Now, we move on to the third problem, which is the live variable analysis problem, we have used live variable information extensively in the code generation and register allocation algorithms. So, let us understand live variable analysis in some detail, so the variable x is said to be live at the point p, if the value of x at p could be used along some path in the flow graph, starting at p; otherwise x is dead at p. So, this is the basic idea, if the value of x at p whatever is available at p could be used along some path in the flow graph starting at p.

So, that is a very important thing, so is there a use later on is the question, so if p a you know x is live at p; otherwise x is dead at p. The domain of data flow values sets of variables again, so if you take a single data flow value it would be a set of variables, the reason is we are talking about variables being live. So, it is very obvious that the domain of data flow values must be sets of variables, for a change this is a backward flow problem with the confluence operator union.

The reaching definitions was a forward flow problem with the confluence operator union, the available expression problem was also a forward flow problem with the confluence operator intersection. This is a backward flow problem with the confluence operator union, we are not going to consider the last combination, backward flow with

confluence operator intersection that would be required for the computation of anticipated expressions useful in partial redundancy elimination.

Then before we define the equations for OUT and IN, let us understand what they mean, IN B is the set of variables live at the beginning of the basic block B. So, in other words at the beginning of the basic block, the variables which have uses later on either IN B or later will be will form the set IN B. Similarly, set OUT B consist of set of variables live just after B, again there must be a usage after the basic block B for everyone of the variables which is listed in OUT B. To compute OUT and IN we are going to have the equivalents of gen and kill.

So, DFE B is the kill counterpart and USE B is the gen counterpart, so let us look at the equation and then come back to the definition of DEF and USE. So, the equation says initialization of IN B to phi as I told you, if the confluence operator is union, then the initialization will be to phi, this is the backward flow problem. So, we have an equation for IN B in terms of OUT B, so we have use B union OUT B minus DEF B for the reaching definitions problem we had gen union you know in minus kill.

So, whereas, here we have IN on the left hand side, so we have use, so this is equal into gen union out minus kill. So, kill is DEF and gen is use, the other equation is OUT B equal to, so we have OUT B as the union of all the in sets of the successors of B, so because, this is a problem, which is backward flow. Once, we are computing IN, in terms of OUT it is only you know correct to compute the OUT of a set IN sets of the successors, so it goes backwards.

Let us understand what DEF and USE are, so let us begin with use which is the gen set, so USE B is the set of variables, whose values may be used in B prior to any definition of the variable. In other words we want only those variables, which have been used before definition, so the definition would be later on, but the usages would be prior to that in the basic block. It does not mean that we are using undefined variables, the definition of these variables would have occurred in a basic block prior to B and that value would flow to this particular basic block.

The intuition behind this is the variables which are used before a definition occurs will all be live at the input point of the basic block. That is because, at the beginning of the basic block or the input of the basic block, these uses which we are considering will all

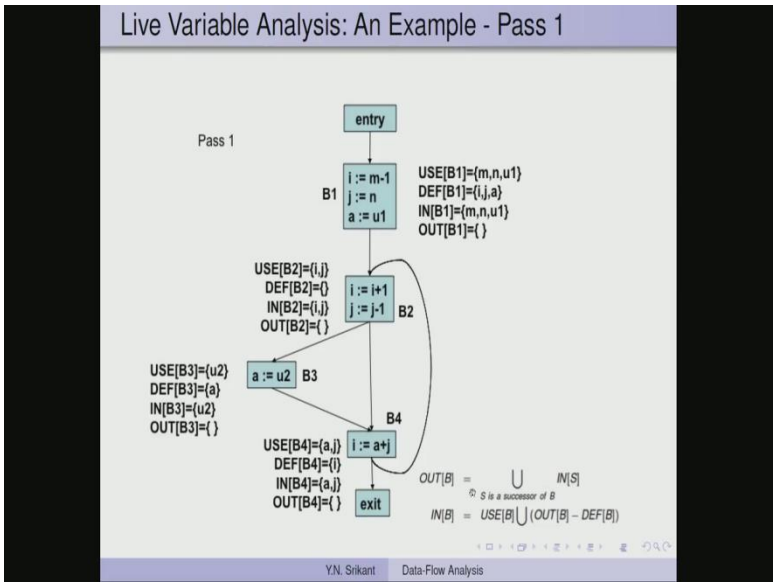
be visible. And therefore, we can say that these variables which have uses inside the basic block prior to any definition will be a live variables, so there we put them into the set use.

So, this is in some sense generation of live variables from the basic block, the second one is the kill set or the DEF set. So, here we look at the you know in some compliment of this, set of variables definitely assigned values in B prior to any use of that variable in B. So, definitely assigned means we do not want any point assignments through pointers which are not definite, so here we used variables which have uses before definition, here we are looking at variables which are definitely defined before any use of that variable.

So, the again the intuition is if a variable is defined and then used, it cannot be live at the input point of the basic block. Because, definition is not considered as a use, so that is the difference between these, so now, coming back to these equations, so IN B the set of variables which are live at the beginning of the basic block would be; obviously, one part would be use B, which are the variables which are used before definition, so they are visible at the input point of the basic block.

Then you know there are some variables which are used after the basic block, so if they pass through the basic block then in a transparent manner, then they will also be visible at the input point of the basic block and hence live. But, some variables which are defined in the basic block will; obviously, seize to be visible at the input point of the basic block and they must be removed from OUT B.

(Refer Slide Time 26:02)

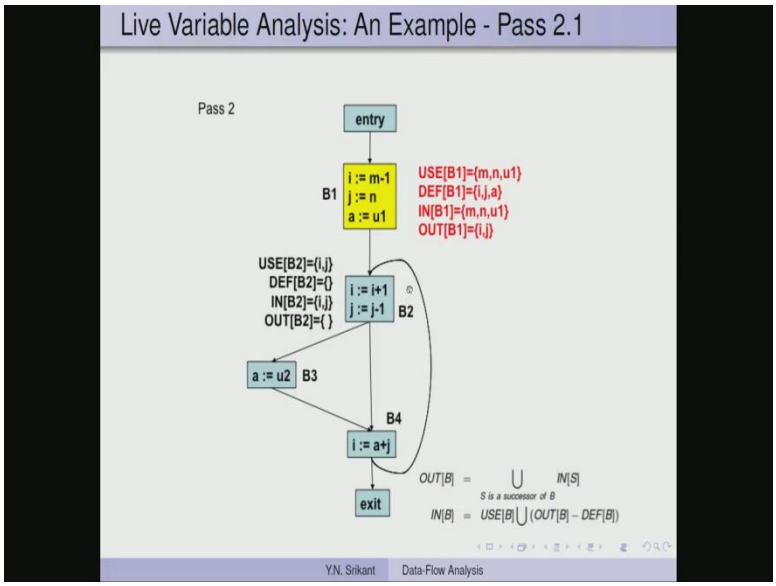


So, let us work out this example to understand the liveness computation, here is block B 1, so in this block we are defining i j and you a and there are no uses of i j and a before the definitions. So, all the 3 variables will be in the DEF set, we have m n and u 1 being used in this block and they are not at all defined in the block, either before or after these statement. So, all these three variables will be in the use set of the basic block, then for the block B 2 we have the use set as i and j because, here i and j are being used and then defined on the left hand side.

So, because we said usage occurs first and then the definition, the DEF set becomes phi there is nothing you know these are definitions which have occurred after the usages. For the third basic block we have a equal to u 2, so u 2 is in the USE set and a is in the DEF set, for the fourth basic block we have i equal to a plus j. So, a and j are in the USE set and i is in the DEF set, so this is the simple computation of USE and DEF for this particular control flow graph.

So, then the you know we could simply say OUT B equal to phi as the initialization and IN B 1, you know we could simply put USE B into it. So, that is what we did for the reaching definitions also, all those which are in the used set; obviously, this particular part will be in the IN. So, practically we could initialize IN B 2 USE B 1, here it is a USE B 2, here it is USE B 3 and here it would be USE B 4.

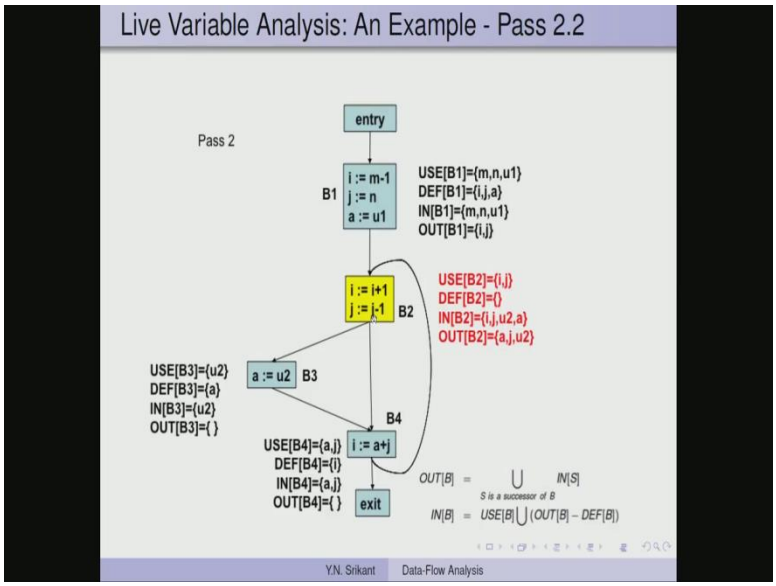
(Refer Slide Time 28:25)



Now, that is the first parse, the second parse we compute using these equations, so we must compute the outset here, the inset of course, remains as use because, you know we have the same equations. So, look at these equations, ((Refer Time: 28:50)) so we have outset and then the inset computation, so we let us compute the outset, the outset happens to be nothing, but the inset of B 2. So, the inset of B 2 is i j, so the outset of B 1 becomes i j.

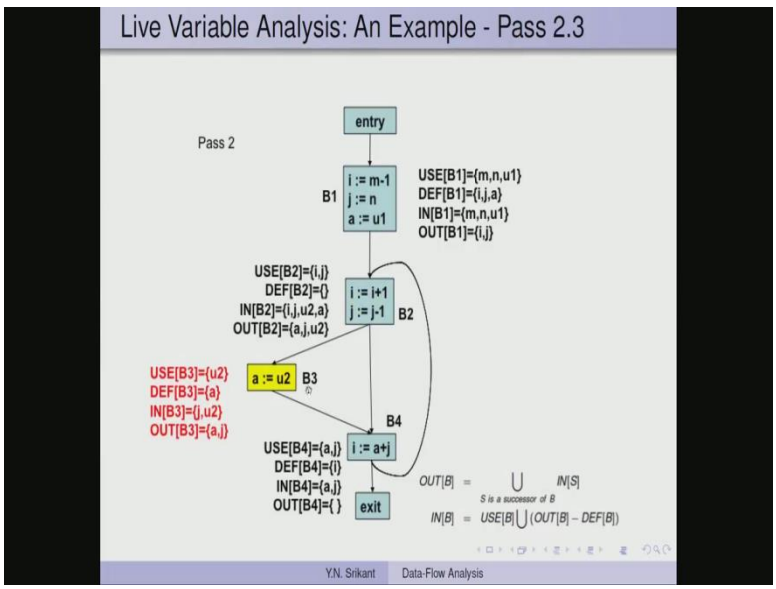
Then we compute the IN of B 1, so that would be whatever is in the use m n u 1, then we have whatever is in OUT B 1 minus whatever is in the DEF i and j both are in DEF set, so this goes out. So, we have m n u 1 as the inset of this basic block that is quite understandable because, at this point i j and a are not live, you know they are defined, so they are not live. Whereas m n and u 1 are being used, so these are used after this point, so they are all live.

(Refer Slide Time 29:53)



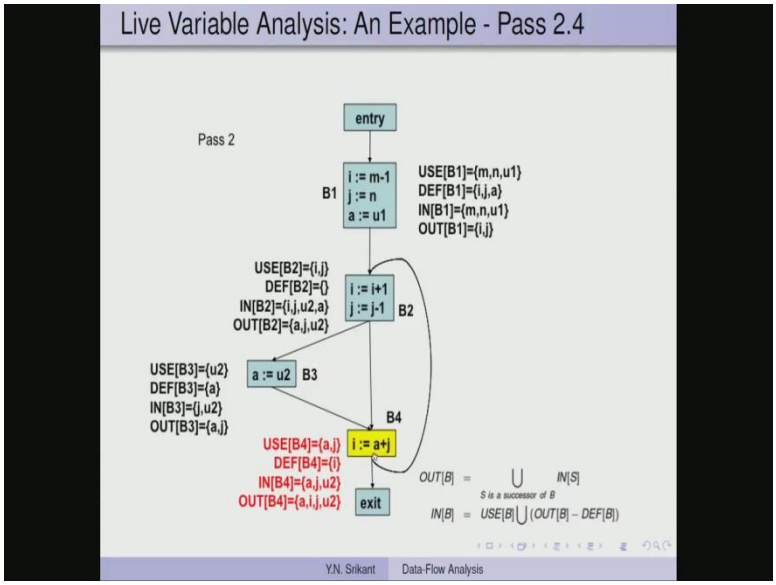
For B 2 again let us compute OUT of B 2, so this is the out point, so there are 2 insets possible here, one of B 3 and one of B 4. So, in of B 3 is u two and IN of B 4 is a comma j, so OUT will be a, j, u 2, so that is very easy union of these two, now what about the inset we have i j of course, from the USE set. Then we have a j and u 2 and DEF set is phi, so a, j, u 2 and also i, j will all be included in IN of B 2, so i, j, u 2 and a will all go into the IN of this particular set. So, that is understandable again at this point i and j are being used, so that is easy to see and then u 2 is being used, so that is easy to see and we also have a being used, so that is also easy to see.

(Refer Slide Time 30:56)



The third would be block number B 3, so in block number B 3 we have again the outset is nothing, but the inset of B 4. So, which is a comma j, so a comma j here, the inset would be USE set that is u 2 and then a comma j minus the DEF set a, so that leaves us with j. So, j and u 2 would form the inset of this particular basic block, so here whatever j and u 2 are here, so u 2 is the usage immediately and j is the usage here at this point.

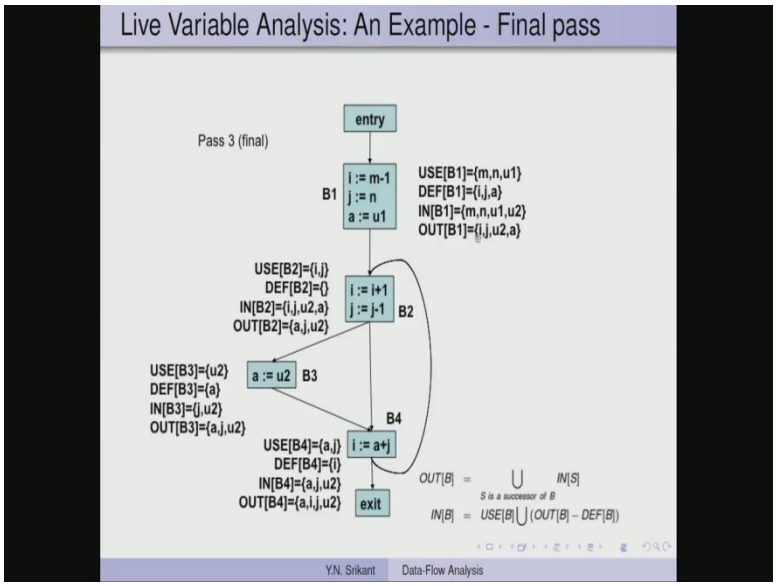
(Refer Slide Time 31:35)



The last block is B 4, so for this block the outset would be the inset of B 2, so we have a inset of B 2 which is nothing, but you know i, j, u 2, a. So, that part would be here and then we also have you know, so this is one of them, so then we have the other one which is going out, so that is not there at all. So, then we have I think the, so it should have been in of this, so that would be i, j, u 2 and a, so a, i, j, u 2 the order is the different, but the set is the same.

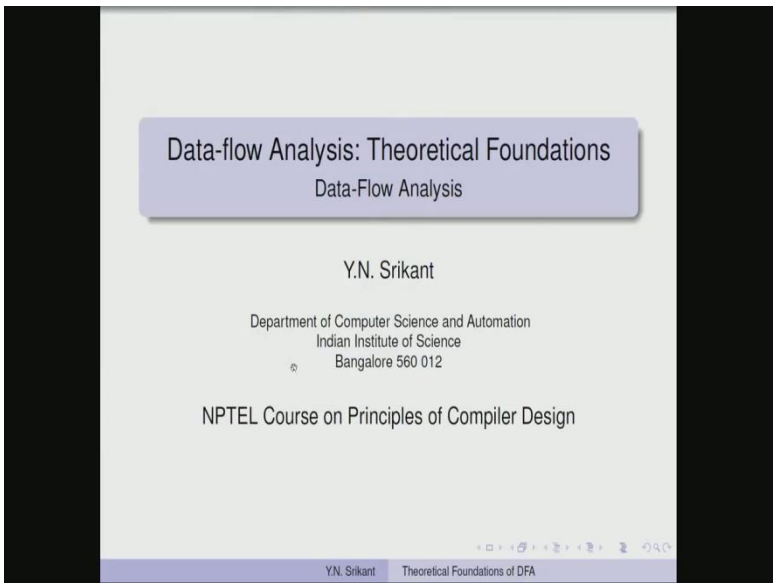
Then we have a computation for in, which would be use set of B 4, so which is a j and then the outset a, i, j, u 2 with i removed. So, that would be a, j, u 2, so again we have a, j, u 2 as the inset of the block before, so for the in here we have a and j of course, and then the u 2 part comes because of this path, so that is the usage after that. So, this is how we compute the blocks, rather the IN and OUT sets for the various blocks, so one more pass will be required, where this you know this changes.

(Refer Slide Time 33:16)



So, the OUT here has changed and the IN also changes, similarly for this also there is a change. So, I would leave you know computation of the IN and OUT set for this iteration as an exercise, so after the third pass the values stabilize and there would be no more changes to any of these sets. So, this is how the computation of IN and OUT values take place in live variable analysis.

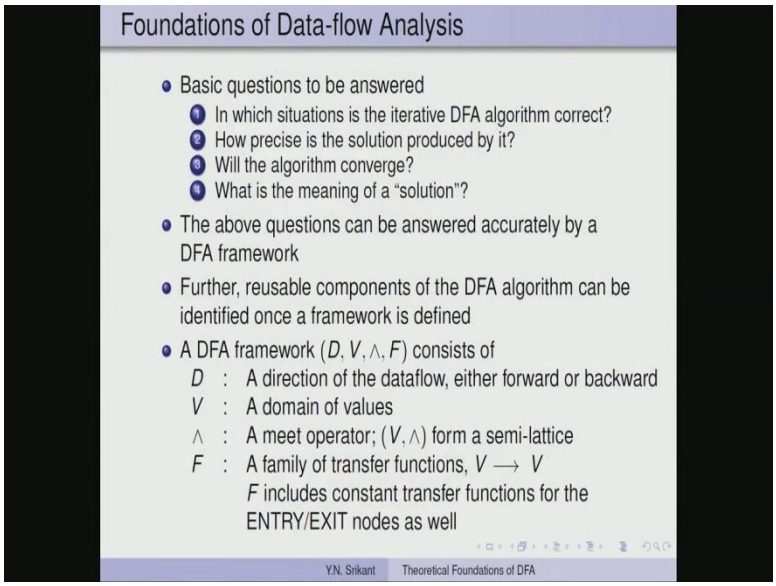
(Refer Slide Time 33:47)



So, let us now understand some of the theoretical foundations of data flow analysis, what we have seen, so far are the algorithms, which can be directly implemented as programs.

But, we did not understand what makes the you know sets, you know take on confluence operators which are union or intersection, why should it be a forward flow problem, why should it be a backward flow problem these were not really understood properly, so let us understand the theoretical foundations of the analysis now.

(Refer Slide Time 34:28)



The basic questions which we want to answer they are actually listed here, so in which situations is the iterative data flow analysis algorithm correct. So, I you know, so we have not yet answered this question, how precise is the solution produced by it; that means, we must define something which is ideal and then comparative. Will the algorithm converge and what exactly is the meaning of a solution to the data flow analysis problem.

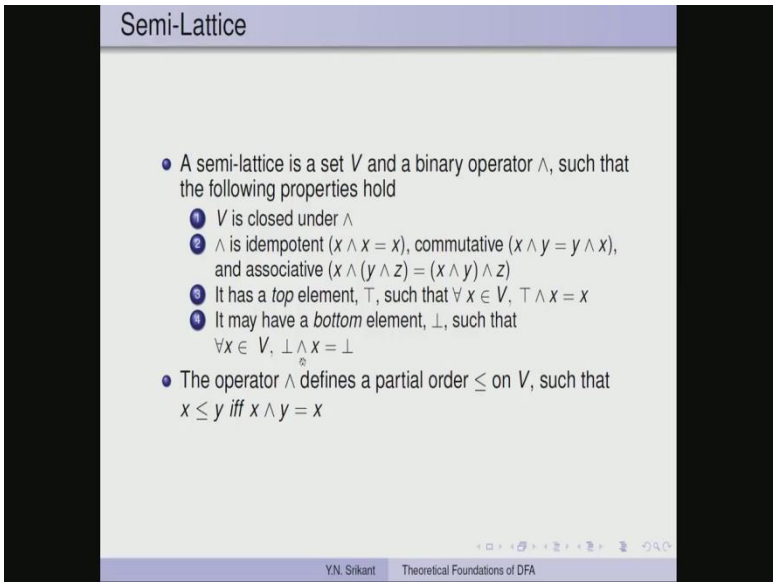
So, to answer these questions we need to define a formal framework for the data flow analysis. And once we do that actually turns out that some of the reusable components of the algorithm can be identified and we could build something like LEX and YACC, which would take as inputs a description of the domain of values, the confluence operator, etcetera and the direction of flow. And finally, you know the various sets functions and, so on and, so forth, it can actually generate a dataflow analysis program.

So, such generators are possible, once we identify the reusable components of the framework. So, let us begin with the framework a definition, so as 4 components D V the meet operator and F, D is a direction of the dataflow, either forward or backward as we

have understood. Then we have V which is the domain of values, we are yet to define the domain formally, it is not going to be a set in all cases it is going to be a semi lattice.

Meet operator is indicated as this inverted V and V comma meet operator forms an algebraic structure called as a semi lattice. I will define a semi lattice and give you an example as well, F is a family of transfer functions V to V, so we will define this also properly F includes constant transfer functions for the entry and exit nodes as well.

(Refer Slide Time 37:05)

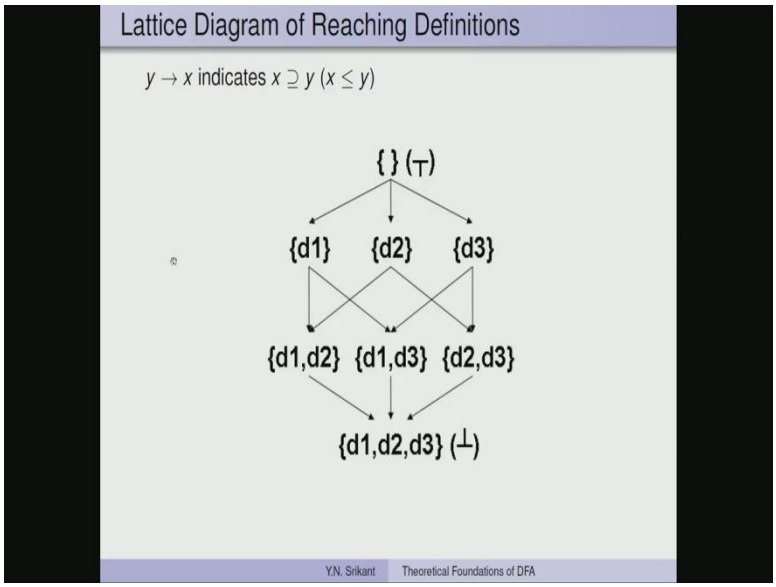


So, to begin with we must understand the structure of a domain as I told you the domain is a semi lattice. So, let us understand the structure of a semi lattice, a semi lattice is a set V and a binary operator meet, such that the following properties hold, so the first important property is V is closed under the meet operation, whatever is defined as. Then the meet operator is idempotent, so x meet x is x it is commutative, so x meet y is equal to y meet x it is associative, so x meet y meet z is same as x meet y meet z.

It has what is known as a top element T such that for all x in V top meet x will always be x, in other words top element is the top most element of this domain. So, you cannot go above that in some sense, it may also have a bottom element bottom such that for all x bottom meet x is bottom. So, in other words you cannot go below the bottom element, in fact, a semi lattice requires only the first three that is the closure, then the top element you know and the idem potency, commutativity and associativity, so if there is a bottom element as well that is will be an extra.

So, the meet operator as a side effect defines a partial order on the elements of the you know set V. Such that x is less than or equal to y if and only if x meet y is x, so in some sense x is lower than y in the partial order.

(Refer Slide Time 39:12)



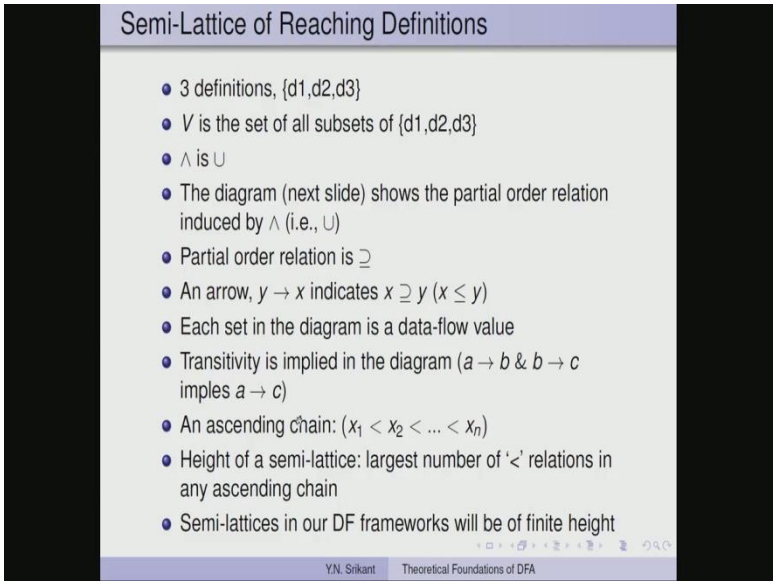
So, let me give you an example and then read through this text, so here is a lattice diagram of the domain of reaching definitions. So, as we see the domain of reaching definitions consist of only the definitions, so the various sets in the domain are of course, we have the null set, then we have the singleton sets d 1, d 2, d 3. Then the paired elements d 1, d 2, d 1, d 3, d 2, d 3 and finally, the entire set of definitions d 1, d 2, d 3.

So, the null set is the top element and the entire set d 1, d 2, d 3 is the bottom element, so in some sense, if we say that null set is the set of reaching definitions at a particular point, it is a very strong statement, which says that no definitions reach this point. So, as we go down we say some definitions reach may be 1 here, there would be two definitions if we take one of these as the values. And if we take d 1, d 2, d 3; that means, all the definitions reach a particular point, this is the weakest statement that we can make in the system.

So, here the arrow y arrow x, so there is an arrow from here to here and then here to here, etcetera. That indicates x superset y and that superset operation is the you know relational operator x less than or equal to y in this particular example, so that is easy to

see because, we have d 1, d 2 as the arrow from d 1 to d 1, d 2 and this is a superset of d 1, so this is y and this is x.

(Refer Slide Time 41:10)



So, there are three definitions in this lattice d 1, d 2, d 3, V the is the set of all subsets of d 1, d 2, d 3, the meet operator is the set union operator and the partial order we have already defined that. So, each set in the diagram is a data flow value and transitivity is implied in the diagram a to b and b to c implies a to c, so an ascending chain is x 1 is less than x 2 etcetera, etcetera. ((Refer Time: 41:45)) So, in this case we have you know y then this is you know x less than equal to y, then z less than equal to x and, so on and, so forth.

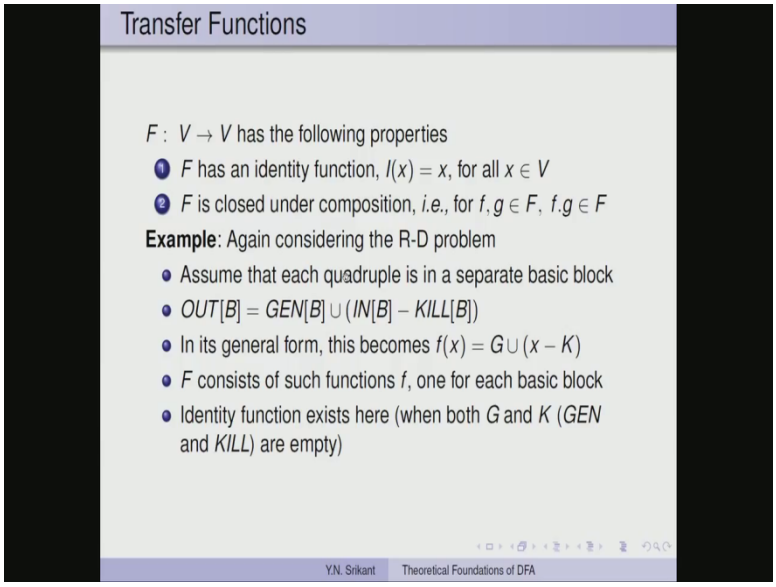
So, this is the less actually chain of these transitive relations, so this is strictly less than and the height of the lattice is defined as the, you know largest number of less than relations in any ascending chain. So, this is a strictly less than operator, ((Refer Time: 42:23)) so if you include the same set, then it becomes d 1 less than equal to d 1, but if we simply say you know this is y and this is x. So, d 1 d 2 less than equal to d 1 are superset d 1, than you know we can instead of less than equal to we can use the relation less than.

Semi lattice is in our data flow framework will always be of finite height and this is a big contributor to the termination of the algorithms. ((Refer Time: 42:55)) So, this is the you know structure of the reaching definitions lattice, so if we actually take the sets of

expressions, a similar lattice will hold here as well. So, we could have sets of expressions as the values in the various lattice points, only thing is the this null and this universal set will not be the same, you know we cannot have a this as the top and this as the bottom.

The lattice will be inverted in some sense I would encourage you to write that lattice, it would be the inverted lattice. Because, if all the expressions are available at a particular point that would be a very strong statement to make, where as saying no expression is available at a point is you know would be a very week statement to you know, if we say that expression no is available that should be fine for the common sub expression elimination. So, it could be a very weak statement, where as this would be a very strong statement. So, the lattice for the available expressions problem would be in some sense the inverted lattice of this. But, the values at the various point will still be sets of expressions, so I would encourage you to write the lattice as an example.

(Refer Slide Time 44:24)



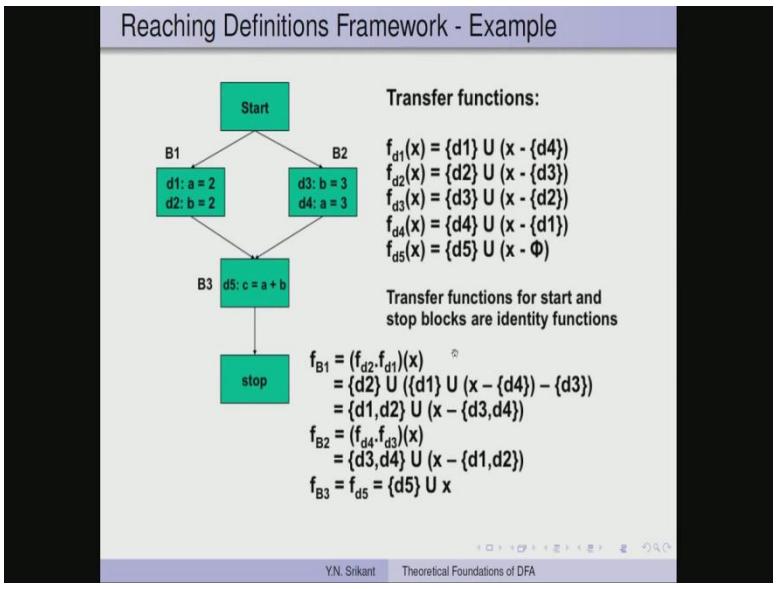
Then, so that defines the lattice and the domain of various data flow values, so now, we define the transfer functions on these domains of values. So, F is a set of functions from V to V, so we can define a number of them, so one for each statement of the program in real in fact,. So, F would have an identity function, so I x equal to x for all x in V straight forward and F is closed under other words for F and g in F f dot g will also be a function in the same family.

So, to give you a clue as to what the structure of these functions and the F would be, let us again take the reaching definitions problem. So, to begin with let us assume that each quadruple is in a separate basic block, so we know our famous equation OUT B equal to GEN B union IN B minus KILL B. Now, remember that each block is a single statement., so if we write the general form of this equation using our F notation, we would simply write f x equal to G union x minus K.

So, where f x is the output value, input value is IN B that is x GEN and KILL are the two constants, so G and K. So, each one of the functions that we define for every statement or quadruple in the program would be of this form and these are called the transfer functions. So, they are the once which define how the you know IN and OUT computations takes place. So, F would consist of such functions f one for each basic block or one for each quadruple.

So, this is our family of functions F and these are the individual functions one for each statement. The identity function exist here, very simple you know make G and K as phi, so we have f x equal to x, so that is the identity of functions, so these two properties are all satisfied. Because, this function you know it can be shown that the composition of these functions will also be a function of the same form, so I will give you an example of this.

(Refer Slide Time 47:10)



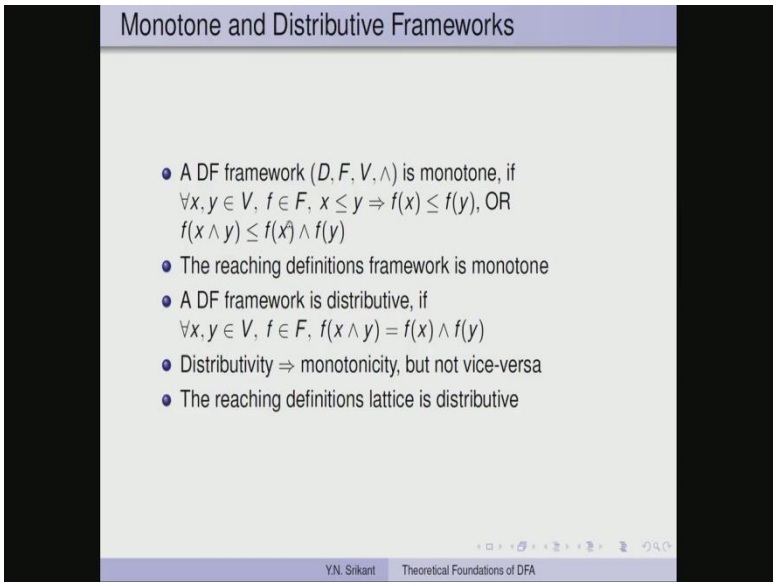
So, now here is a very simple basic block for the start and stop we have identity functions and for the basic block B 1 we have two statements. So, for the transfer function for d 1 let us call it as f d 1 x would be the definition d 1 which is generated by the block union x minus whatever is killed by the block. So, it involves variable a, so and the variable a is involved in d, so minus d 4, so remember x is a set, so the notation is proper x is not a single variable or number or something like that it is a set, so set minus set, so which is correct.

Similarly, for d 2 we have f d 2 of x which is d 2 union x minus d 3 because, this defines b again. Then, similarly for d 3 we have f d 3 union x minus d 2 and finally, f d 4 would be d 4 union x minus d 1 because, that involves a and finally, f d 5 would be d 5 union x minus phi because, there is no statement with x anywhere in the program. Now, to compute the transfer function for the entire basic block B 1 we do a composition, so f d 2 dot f d 1 on x. So, let us apply to b 1 it would be d 2, so we take f d 2, so d 2 union x minus d 3 for x we must you know replace it by f d 1.

So, we really have f d 1 as d 1 union x minus d 4, so d 2 union this is the entire you know x f d 1 union which is f d 1. So, d 1 union x minus d 4 and then the minus d 3 corresponding to f d 2, so simplifying this we get d 1 union d 2 union x minus d 3 comma d 4. So, this is of the same form as the original function and therefore, it is closed under composition, so similarly for the basic block B 2 we get d 3 comma d 4 union x minus d 1 comma d 2 and for f b 3 we get f d 5 equal to d 5 union x, so that is from this right.

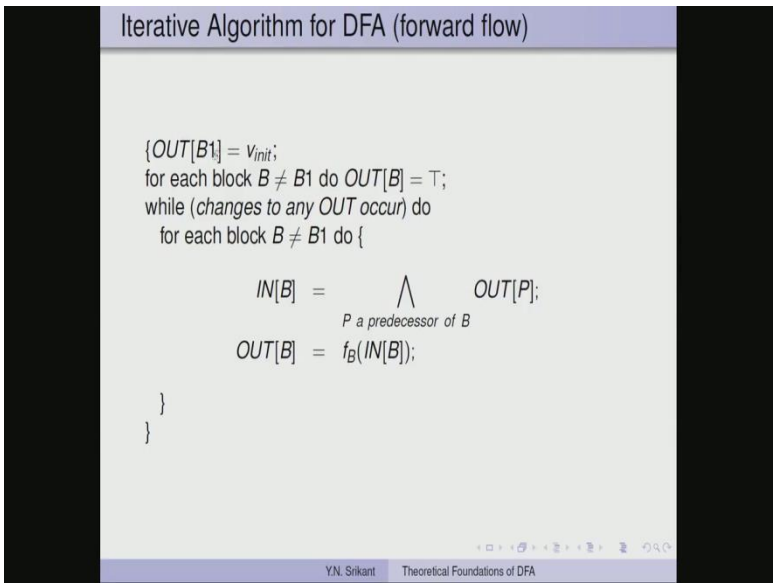
So, f d 4 dot f d 3 x it can be computed exactly the same way we have d 4 union and, so on and, so forth. So, transfer functions are defined for each one of the statements in the basic blocks, rather in the program and then for computing the transfer function of the basic block we make a composition of them. So, finally, we get one transfer function for each basic block, so this is now available to us.

(Refer Slide Time 50:31)



Now, what needs to be done is to apply them to the various data flow analysis problem, so before we understand what to do, we will have to define monotone and distributive frameworks. So, a data flow we have understood the domain, we have understood the transfer function, so very soon, we will also see let us look at that and then may be go back.

(Refer Slide Time 50:56)



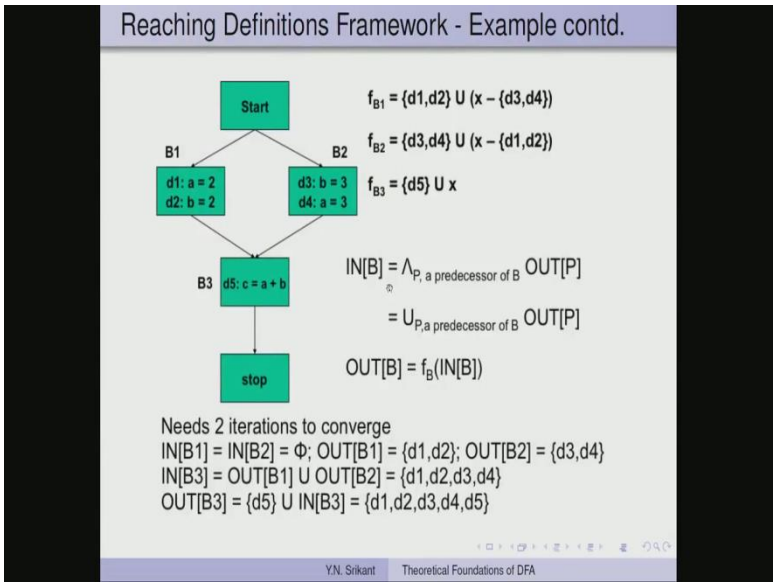
So, let us look at the algorithm for data flow analysis assuming that there is a forward flow. So, forward flow implies you know we have the out in terms of the IN, so instead

of saying gen kill and all that we have given a transfer function for each basic block, so let us define it as a f b of IN B. So, this f b automatically incorporates gen and kill as we have seen here, the iterative algorithm is very simple, it is very similar to what we have already understood. So, OUT B 1 is initialized to some value and then you know for the others we initialize it to the top, this is a forward flow remember.

And then while changes to any OUT occur, we keep doing for each block B not equal to B 1 do compute IN and OUT. So, we did not specifically write down change equal to true false, but this statement captures everything, so while changes to any OUT occur, we must compute IN and OUT repeatedly and make sure that they do not change anymore. So, we are computing in you know as a meet of the OUT sets and we are computing OUT as a function of the IN set.

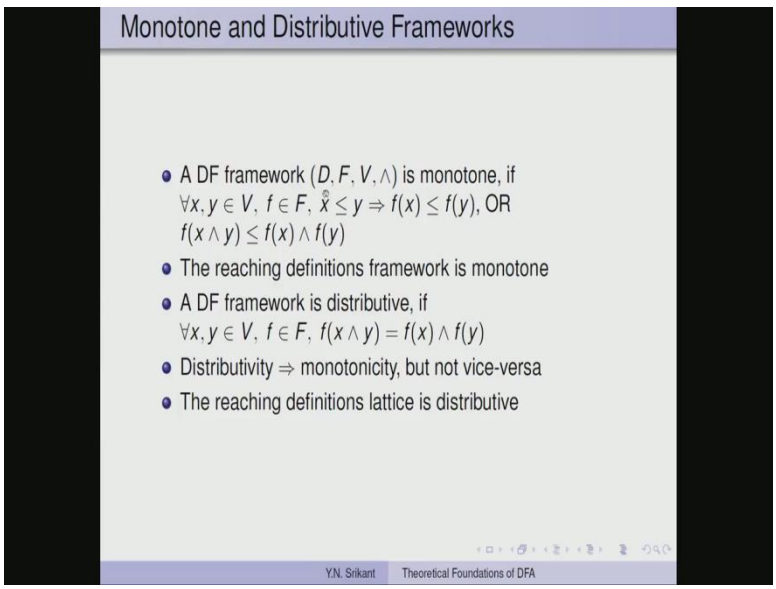
So, we had used you know union and intersection in this case, so this is a forward flow problem, so we would have used union here. And if it was a backward flow problem then the initializations would all be different, so that is the point, so here the forward flow problem and the backward flow problem will relate to the transfer function here. So, we would have changed the initialization and also the order of these equations as necessary, where as the confluence operator would have been union or intersection. So, if we change the confluence operator then also the initialization would be very different. So, these are to be kept in mind, when we actually implement these things, so here this value V init etcetera has to be appropriately defined based on the problem itself.

(Refer Slide Time 53:32)



So, here now we one minute, so let me also show you the example of the same control flow graph. So, we had defined f B 1 f B 2 and f B 3, so now, we have the IN equation for which we have union and the OUT equation for which we have the transfer function. So, I will not go into iterations you know exhaustively because, it is a fairly straight forward thing to apply, so IN B 1 and IN B 2 are initialized to phi and they remains, so B 1 becomes d 1, d 2, OUT B 2 becomes d 3, d 4 from these two equations. And then IN B 3 is the union of these two and OUT B 3 would be got by applying this d 5 union x, so this is how we compute the various values using these data flow equations.

(Refer Slide Time 53:32)



So, let us understand what we mean by monotone framework and a distributive framework. So, a data flow framework is monotone, if we have for all x y in V and for all f in F x lesser than equal to y implies f x less than or equal to f y, so if x is a smaller value than y in you know in this using the same meaning as this less then operator. Then the f x and f y values must also respect the same relationship that is what this really says.

So, if the reaching definitions problem of course, is a monotone framework that will be very easy to see. So, if once we consider that lattice, so x is less than or equal to y and our equations transfer functions will always define it such that it is f x less than or equal to f y. And a framework is distributive if f of x meet y is f x meet f y, so otherwise monotonicity is simply f of x meet by less than or equal to f x meet f y, where as a distributivity implies equal to...

So, distributivity is a stronger property and distributivity implies monotonicity, but not vice versa. In our case the reaching definitions lattice is also distributive, the important thing is a monotonicity is very important along with the finite height of a lattice, it ensures that our algorithms indeed terminate with a fix point. Whereas, distributivity gives even stronger properties, which we are going to see later, so let us stop at this point and continue with rest of the theoretical foundations in the next lecture.

Thank you.