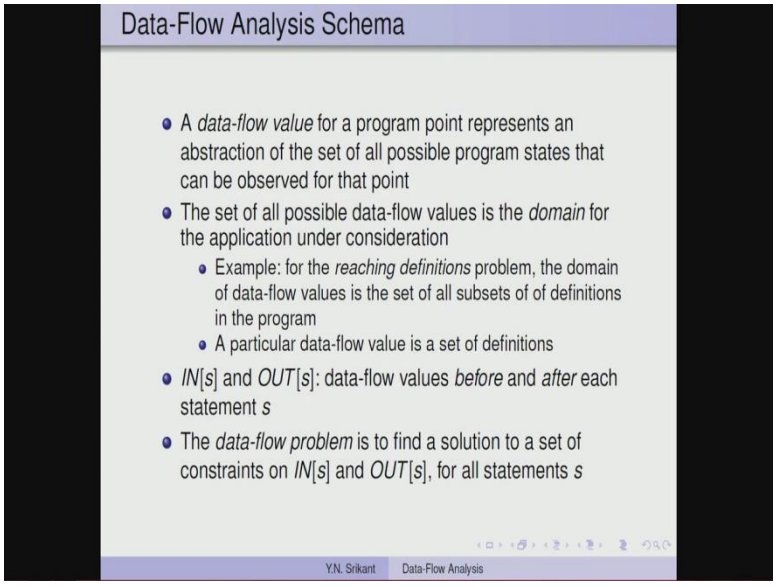
Principle of Complier Design
Prof. Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Lecture - 32
Introduction to Machine Independent Optimizations Part-2

Welcome to part two of the lecture on Machine Independent Optimizations, today we will continue our discussion on data flow analysis, so we covered illustrations of code optimizations in the last part.

(Refer Slide Time 00:37)



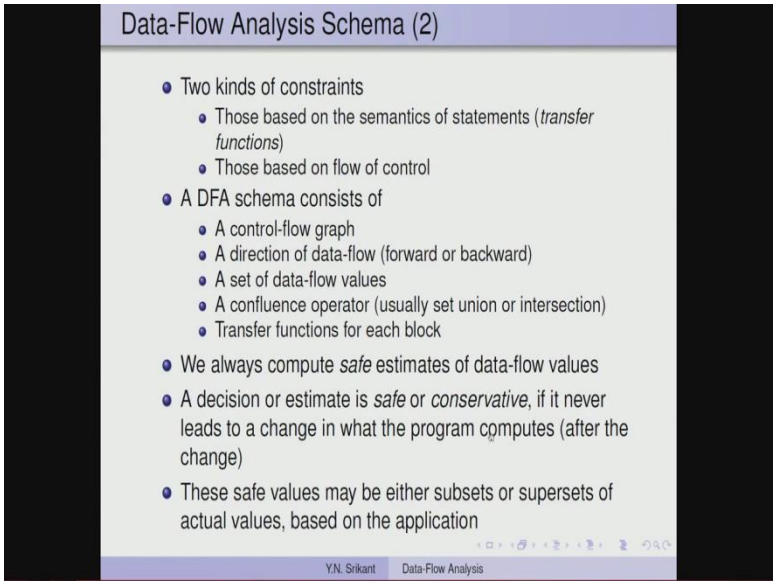
To do a bit of recap a data flow value for a program point, represents an abstraction of the set of all possible program states that can be observed for that point. So, the issue is when the program is executing at every point in the program, there may be several possible program states and when we do a data flow analysis, we are statically examining the program. So, we compute an abstraction of this set, so that is an approximation of this set.

In other words, it is possible that this approximation includes certain program states, which can never be reached at that point in an actual execution, but because of our static analysis this extra information will be included in the abstraction. The second important point is the set of all possible data flow values is the domain for the application under

consideration. So, for each one of the applications this domain is going to be different, for the reaching definitions for example, it would be sets of all subsets of definitions.

And for the available expressions it would be sets of all subsets of expressions, for the live variables problem it would be the set of all subsets of variables etcetera. We also have two quantities IN and OUT, so these are the data flow values before and after each statement S. And as I mentioned we are going to be interested in a larger grain size that is basic block rather than just a single statement, the data flow problem is to find a solution to a set of constraints on in and S sets.

(Refer Slide Time 02:43)



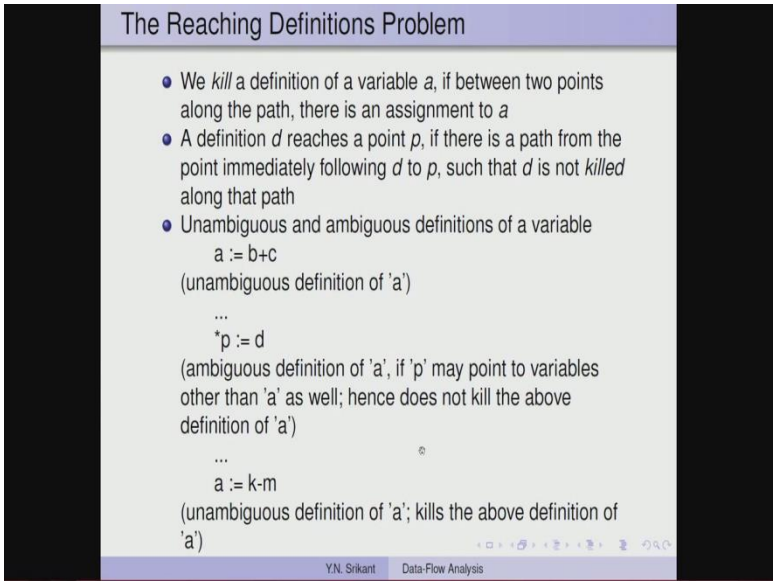
So, what kind of constraints can we provide, we provide two types of constraints, the constraints based on the semantics of statements are called transfer functions. And constraints based on the flow of control are also going to be provided, that is if there is a join of different paths in a program, then what do we do about the computation of IN and OUT, that is going to be the question to be answered.

So, a data flow schema consists of five components, one is the control flow graph itself, then a direction of data flow forward or backward has to be indicated, a set of data flow values is to be provided. A confluence operator usually set union or set intersection should be given, and the transfer functions for each basic block, as I said instead of statement we are going to provide the transfer functions for each basic block, will also be provided.

When we compute the estimates of data flow values, that is the approximations or abstractions of data flow values, we want to make sure that these estimates are safe, so again safety is with respect to a particular application. So, in our case we want to perform several optimizations, so safety is with respect to these optimizations, so an estimate is safe or conservative, if it never leads to a change in what the program computes after the change is carried OUT in the program by a transformation.

So, in other words, if we perform an optimization then the optimized program should provide the same output as the unoptimized version, so that is precisely what we mean by safety, so in doing the optimization we would have used lot of data flow information. So, and we would have made decisions on transformations using data flow information, so this type of a decision is said to be safe, if there is no change in what the program computes. So, this will become clear as we go on and take up more examples, these safe values may be either subsets or supersets of actual values based on the application.

(Refer Slide Time: 05:23)



So, to get into the problem of reaching definition analysis, so we need to define two quantities, one is what we mean by generation of a definition and what we mean by killing a definition. So, we KILL a definition of variable a if between two points along the path there is assignment to a, so let us take up this example. So, here we have a equal to b plus c and later in the program in the same program, we have another definition a equal to k minus m.

Obviously, a equal to k minus m and a equal to b plus c are both definitions of a and it is possible, suppose these two are in the same basic block. So, in such a case it is never possible for a equal to b plus c, that is the value of a computed by this particular definition to be available after definition a equal to k minus m, this kind of over takes the previous definition; so in such a case we say that this definition kills this particular definition.

Now, what is the definition of reachability, what is the reaching definition a definition d reaches a point p, if there is a path from the point immediately following d to p such that, d is not killed along that particular path. So, if there are there may be many paths from d to p, but we are only considering existence of just one path, so if there is more than one path it does not add to reachability, but if there is no path, then the definition is not reachable. So, from d you must be able to go to point p via particular path, that is all that really means, so if there is path then the definition reaches the point p.

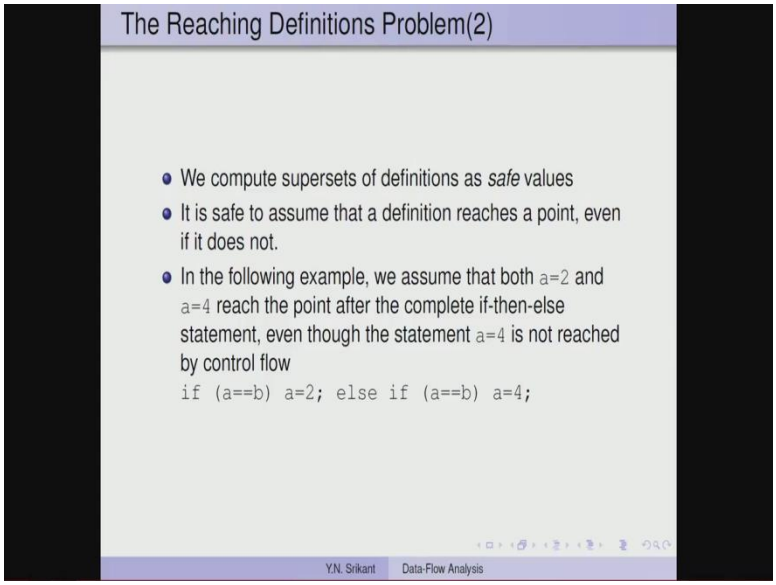
So, remember again more than one path for the same definition does not add to reachability, but if there is no path from the definition to point p, then the definition is not reachable. We must also distinguish between unambiguous and ambiguous definitions of a variable, a equal to b plus c let all the three a b and c, three entities be names. So, in such a case this is called as an unambiguous definition, because we are explicitly defining the variable a.

Later in the point in the program at some point we have a definition such as star p equal to d, this is called ambiguous definition of a, the reason being p may point to many variables, it may point to p, it may point to b, it may to point to x. Therefore, we cannot concretely say that p is defining a particular variable of course, if you already know that p points only to x, then we can say p this statement defines a is a definition for x.

But, in general during static analysis, we can only determine that p points to a couple of variables, we can hardly say p points to exactly a particular variable, so if this is so this is called ambiguous definition. And because it does not definitely define a particular variable, we cannot say that an ambiguous definition kills the previous definition. So, this does not KILL anything, ambiguous definitions do not KILL any definitions, but this is unambiguous definition again a equal to k minus m. So, this kills this unambiguous definition, again this ambiguous definition may not KILL this definition ambiguous

definition, because p may never point to a, so in that case this is not a definition of a at all, so we do not want to KILL such definitions either.

(Refer Slide Time 09:58)



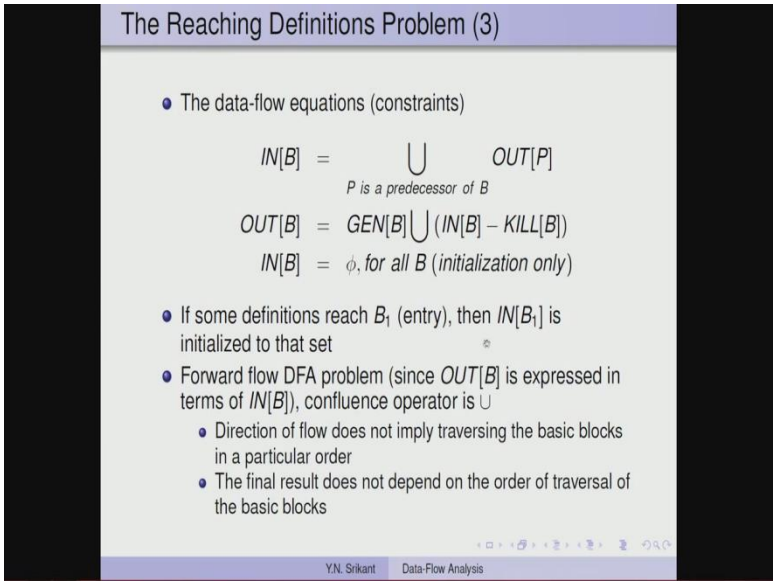
Then in the reaching definitions problem, we compute supersets of definitions as conservatives or safe values, so as I said we want to compute safe estimates, so in this case a safe estimate would be a superset. What does it mean to compute a superset, so it says it is safe to assume that a definition reaches a point, even if it does not, so this is what happens if we include more definitions into the set of reaching definitions than definitions that actually reach.

So, there may be some definitions which do not, but it is ok to include them, this is our definition of safety here, so for example, the let us say we have if a equal to b, then a equal to 2 else, if a equal to b, then a equal to 4, so let us analyze what happens during execution. During execution we compare if equal to b, then if it was true then we would have taken a equal to 2 and exit it, suppose a equal to b was false we would have taken the else part.

And we are comparing a equal to b again that means, if it was false here it has to be false here as well, because we have not assigned anything into either a or b, so this statement will never be executed. So, at this point during actual execution a equal to 4 will never, this definition of a will never reach this point during actual execution, but because we know that by this execution path, we will never take this particular branch.

But, static analysis does not know that, so in static analysis we are going to assume that both a equal to 2 and a equal to 4 will reach the points after this semicolon, so this is the conservative estimate. So, we know that during actual execution a equal to 4 will never reach this point, but because of our static analysis we have no idea that a and b, we do not take the values of a equal to b here and then use it here as well we do not do that in static analysis. So, we will have to assume that both a equal to 2 and a equal to 4 will reach this particular point after the semicolon, so this is an example of a super set that is being computed.

(Refer Slide Time 12:51)



So, here I present the data flow equations are constraints as they are called, and am going to explain these equations in great detail in the coming slides. The first constraint or the equation says, in of basic block that is the set of definitions reaching the entry point of a basic block, that is what IN B is will be a union of the outsets of all it is predecessors blocks, so P is a predecessor. So, this is what this says, I will explain this very soon, OUT B the set of definitions reaching the exit point of a basic block, that is what OUT B is, will be GEN B, the set of definitions generated by the basic block union IN B minus KILL B.

So, IN B is what is obtained from the top, KILL be is the set of definitions killed by the basic block, so remove from IN B the set KILL B and that will also be coming out of the basic block. And IN B is initialized to phi for all basic blocks, so these are the

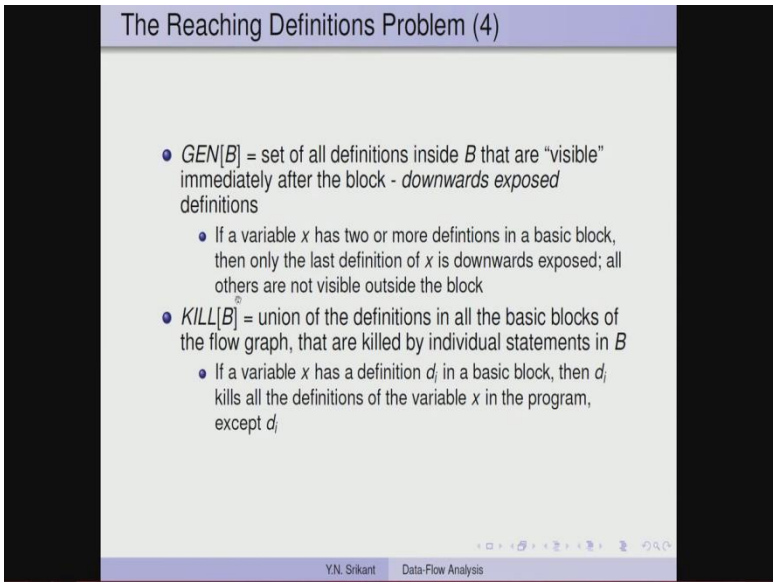
constraints, these are the equations, so in these equations GEN and KILL are constants, so they will be computed only once. And then they will not be modified again and let me go ahead with the definition of GEN and KILL before I explain these equations.

So, if some definitions reach B 1, then in of B 1 is initialized to that particular set instead of phi, this is forward flow problem, the nature of the flow is obtained using the equation for out. So, if the equation for out is in terms of in, then this is called as a forward flow problem, if the equation were turned OUT to be an equation for IN, in terms of OUT then it would become a backward flow problem.

The direction of flow of the data flow values does not imply that we traverse the blocks, in particular order to compute the sets no any order is fine with us and the final result does not depend on the order of traversal of the basic blocks either. I have still not told you how to compute this, how to compute the values for IN and OUT, typically once we compute GEN and KILL, we compute IN and OUT values in a loop, we compute them again and again making sure that we reach what is known as point.

So, at the fixed point after we cross the fixed point the values of IN and OUT do not change again for any basic block, so for the reaching definition problem the fixed point is guaranteed to be reached in a couple of iterations. So, otherwise formally it is necessary for a new data flow analysis problem will have to formally prove that, this sort of thing fixed point can indeed be reached, but this is not a part of our course. So, we are not going to discuss the theory of data flow analysis in this course, so again just repeating that, so we compute IN and OUT sets again and again until a fixed point is reached.

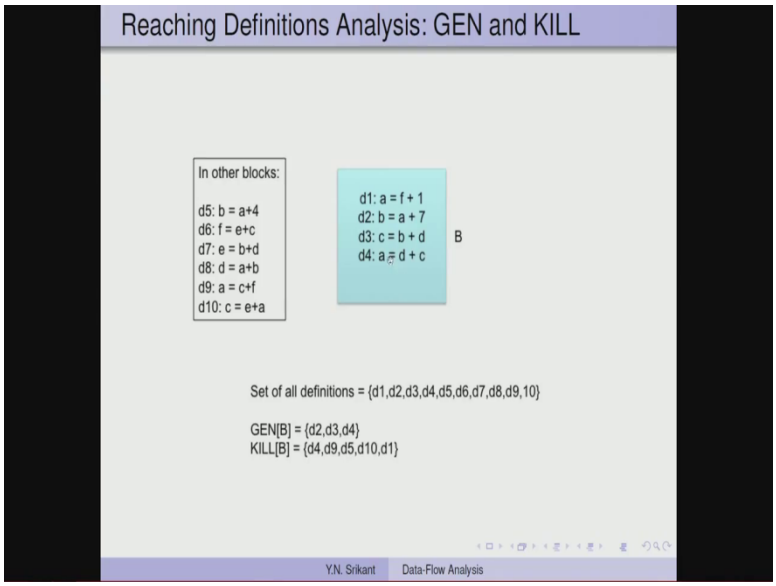
(Refer Slide Time 16:38)



So, let me define GEN and KILL before I go on to the explanation of the two data flow equations, GEN B is the set of definitions inside the basic block, that are visible immediately after the block, they are called as the downwards, exposed definitions. So, let me explain that, suppose the variable x has a only one definition in the block, then obviously the definition will be downwards exposed used and exposed and it will be visible immediately after the block.

But, suppose the variable x has two or more definitions in the block, then only the last definition of x is downwards exposed and others are not visible outside the block, so this is what it is important, we actually take the last definition of the variable. So, we can even compute GEN B by doing a traversal from the end of the basic block to the beginning of the basic block, no problem with that. So, KILL B is the union of the definitions in all the basic blocks of the flow graph that are killed by the individual statements IN B. So, in other words, if a variable x has a definition d i in a basic block, then d i kills all the definitions of the variable x in the whole program except of course, d i, my picture in the next slide will explain this in a better fashion.

(Refer Slide Time 18:08)



Suppose, we consider a basic block B, we want to compute the GEN and KILL for this particular basic block, as I said GEN and KILL will be computed just once and they will be used again and again, the basic block has 4 definitions d 1, d 2, d 3 and d 4. So, d 1 defines a, d 2 defines b, d 3 defines c and d 4 defines a again and in other basic blocks of the program, there are many other definitions d 5, d 6, d 7, d 8 d 9 and d 10, so the set of all definitions in the program would be d 1 to d 10.

So, 4 here and the next 6 in the other blocks, so this is our universal set of all the definitions, to compute GEN, so let us look at the various definitions, d 2 and d 3 compute b and c, they are the only definitions of b and c in the basic block. So, d 2 and d 3 will be definitely included in the GEN set, they are generated by the basic block, now d 1 is a definition of a, d 4 is also a definition of a, but as this is very obvious this definition the value here will not be visible after the redefinition of a, this is just the ordinary programs sense.

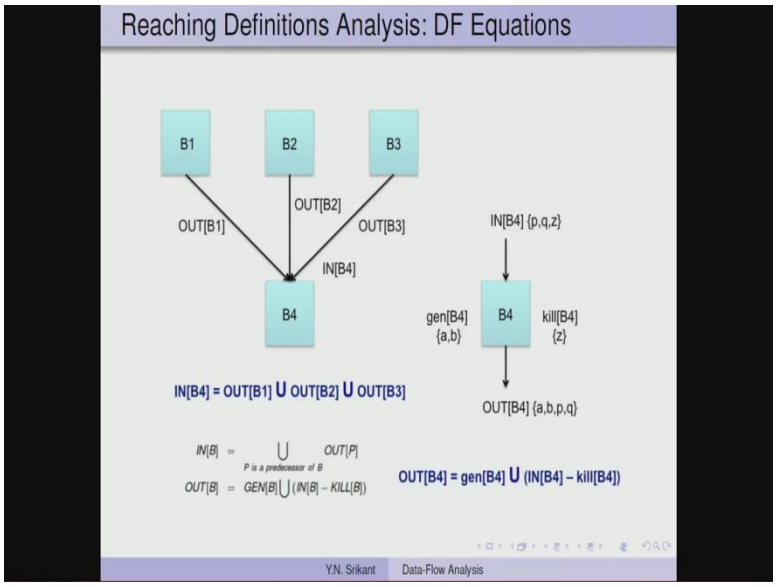
So, it is d 4 which will be visible outside the basic block at this point and not d 1, so we include only d 4 in the set of generated definitions of the basic block b, what about the KILL set, computing the KILL set is a little tricky. So, what we need to do is consider each definition in the basic block, then consider all the definitions of the variable a in the entire program. So, we have a definition which d 4 here, which is the definition of a, we also have the definition d 9 which is also a definition of a.

So, our definition of KILL says take d 1, then it kills both the 4 and 9, so we include d 4 and d 9 in the KILL set, then d 2 defines b, so it kills d 5 in the other basic blocks, so d 5 gets included in the KILL set, then we have d 3 which defines c, so it kills d 10 which is also included in the KILL set. Now, we come to d 4 which is a definition of a, so it kills not d 1 and d 9, d 9 has already been included in the set, so only d 1 gets included in the KILL set.

Why are we computing the KILL set in this fashion, even though we do not consider, whether there is a control flow from this basic block to any other basic block, which defined that variable or not. The answer is simple, we are computing GEN and KILL before the reaching definitions problem is solved to compute the exact KILL set, in fact to check whether there is a flow from one basic block to another and so on, and so forth.

We may end up solving the reaching definitions problem or a variant of it before we compute the exact KILL set, that is the first reason why we have such a big set of definitions for the KILL. The second reason is even if we include many definitions which are not relevant to the basic block b in the KILL set, it is not going to actually affect our computation of IN and OUT. So, because of this it is immaterial whether the KILL set is a super set of the actual values are definitions KILL or is it the exact KILL set that really does not matter to us, so I hope this clarifies the computation of GEN and KILL.

(Refer Slide Time 22:53)



Now, let us move on to the data flow equations, so we have two equations here, one is the inset of the basic block B, is the union of the outsets of the basic block predecessors of B. So, this is the meaning we want to compute the inset of the basic block B 4, there are three predecessors to this basic block B 4, B 1, B 2 and B 3, so there are three sets here OUT B 1, OUT B 2 and OUT B 3. So, IN of B 4 will be the union of OUT B 1, OUT B 2 and OUT B 3, this is actually quite intuitive.

The reason is we now, suppose we consider the set of definitions reaching this point that is OUT of B 1 from here to here there is nothing to stop the definitions from reaching, so all the definitions which reach this point will take this edge, and reach this point as well. So, all the definitions of OUT B 1 should be included in the IN B 4 set, the same is true for OUT B 2 and OUT B 3 as well, so this is the intuitive reason why we take a union here.

The other reason, the actually what happens is the reaching definitions problem we are not really worried about the definition reaching along all paths, we are interested in knowing whether the definition reaches along at least one path, so it may reach along this path or this path or this path. So, reaching along any one of the paths is fine for us, in fact if it reaches along this path and does not reach along these two paths we are still happy, the that is the reason why we want to take the union of all the definitions, which reach this particular basic block.

Then what about the computation of OUT, OUT of B is computed as GEN B union IN B minus KILL B, let me explain why this is, we have the same basic block B 4, there is a set IN B 4 which is already computed, let us say using this equation. Now, the set GEN and KILL has already been computed we want to compute the outset, so here are some values given for IN and GEN, KILL. So, let us say the set IN B 4 is p, q, z, the GEN set is a comma b and the KILL set is z, we need to take the GEN values and put them in the outset that is very clear.

Because, whatever is computed within the basic block B 4 obviously, by definition reaches the end of the basic block and that will be in the outset, so that is the explanation for including GEN of B 4 in the outset. The second is the IN part, there are lots of definitions reaching this point the IN part and few of them get destroyed, because of the

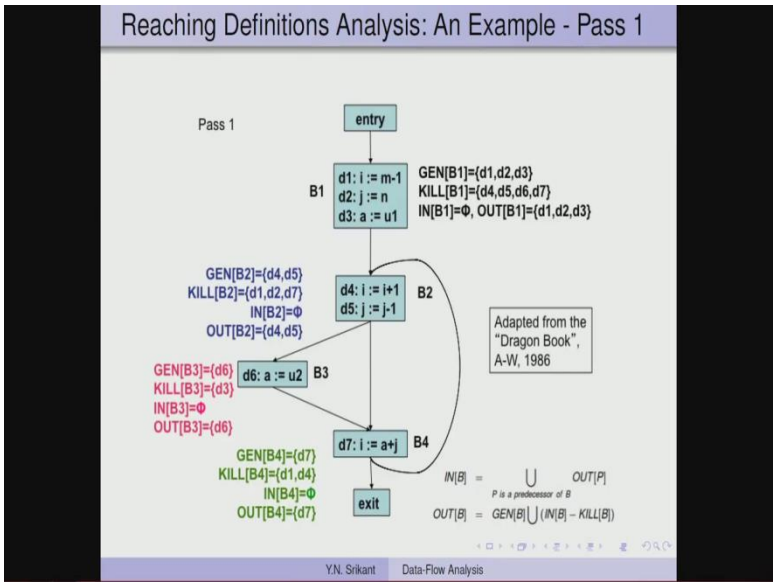
KILL set why there are definitions which are coming in, then some of these very well bells will be refined here.

So, the definitions corresponding to those variables in the inset will have to be removed, so that is why recall the definition of KILL if z is defined here, then that definition kills all the definitions corresponding to z. So, I have not named the definitions here let us assume that these are the variables just for the sake of example, really speaking I should have given the definitions numbers corresponding to these variables, but it helps if we actually keep the variables for the sake of under here.

So, let us assume that there is one definition corresponding to z here, another definition corresponding to this z, another definition corresponding to a, one more corresponding to b etcetera, etcetera. So, the outset now will contain the definitions corresponding to a and b of course, it will also contain the definitions corresponding to p and q, because they are not killed and the definitions corresponding to z will all be removed from the inset, and that is the reason why this has only a, b, p and q.

So, this is how we compute the outset in terms of GEN KILL and IN, so remember if the KILL set contain certain definitions, then they will be removed from the inset, that is what this really means. So, we have removed the definitions of z from here, we have included the definitions of p and q only, the definition of a and b only, so the outset will contain a, b, p and q, so that is what this equation says; include the definitions of GEN take away the KILL set from IN and include it in the outset of B 4.

(Refer Slide Time 28:32)



Let us take a big example and this has been adapted from the book by ((Refer Time: 28:42)), so the first thing to do is to compute the GEN and KILL sets and then make initializations appropriately. So, let us do that this basic block has three definitions d 1, d 2, d 3, so computing GEN here we would include d 1, d 2, d 3 in the GEN set, that is very obvious, because they are all visible here. And KILL set of B 1, if you consider the definition d 1 it kills all the definitions with i, so it kills d 4 and then it kills d 7 as well, so d 4 and d 7 are included in the KILL set.

The second definition d 2 corresponds to the variable j, so it kills the definition d 5, so that is also included in the KILL set the third definition corresponds to variable a, so it KILLs the definition d 6 involving a, so that is also included in the KILL set. So, this is the computation of GEN and KILL, initialization IN of B 1 is made phi and in fact, IN a practical implementation we can say that OUT B will be initialized to GEN, because of this equation. So, ignore this part we will still include a minimum of GEN b in any out computation, so we can initialize OUT B to the GEN set d 1, d 2, d 3 in this case.

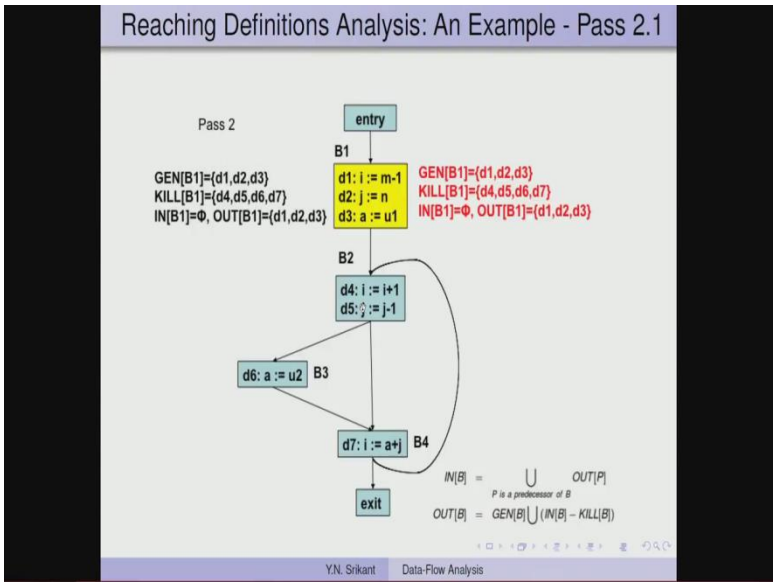
Coming to B 2 this is a bit tricky GEN is easy, so d 4 and d 5 are being defined here, KILL again look at the variable i and the definitions of i elsewhere, so there is d 1 here, then we have d 7. So, and then corresponding to j we have d 2, so all the three are included in the KILL set and then the inset is a phi and the outset is initialized to d 4, d 5. So, you must observe here that the value of i which is used on the right hand side of this

assignment, actually arrives from the value i here, the old value i here or the value here, this is a new value which is computed.

And the same is true for j, this j arrives from this particular value and then of course, after it is computed it will also arrive by other loop, the third basic block GEN is just d 6 and KILL corresponds to the definition of a, which is d 3, IN will be phi and OUT will be d 6. GEN of B 4 is d 7 and KILL of B 4, so we look at the definition of i, so that would be d 4 and then we also have d 1, so these are the two definitions which are killed. So, it is very probably very clear that d 1 will never reach this particular point here, it never reaches here the reason is it is redefined already at this point.

So, we must compulsorily take this part, so only d 4 will perhaps reach this point it will d 1 will never reach this point, so this is not a relevant definition as far as the KILL set is concerned. But, as I said since we cannot compute exact KILL sets without solving the reaching definitions or similar problem we take a super set and it is not going to matter to us.

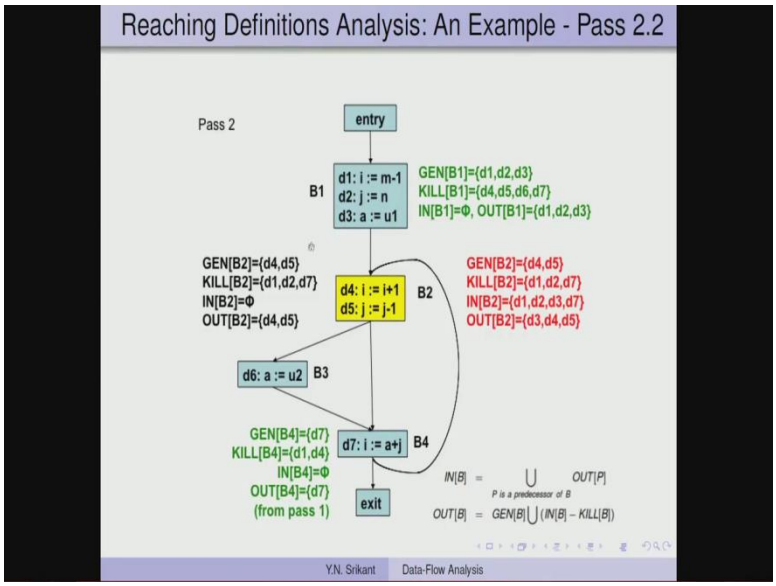
(Refer Slide Time 32:42)



So, now pass 2, so this was the old value of IN and OUT, GEN and KILL down change, old values of IN and OUT are here, these are the new values what does the new value require the equations are here, so the inset is a union of the outsets of the predecessors. So, in this case there is nothing coming from here, so IN B will always remain phi always, what about OUT B you have to include a part of one part is corresponding to

GEN, which is already included, that is d 1, d 2, d 3. And the inset corresponding to this is phi, so there is nothing to take away and this is the old value of IN B it is phi and therefore, we still have d 1, d 2, d 3 as the outset here, so far from here to here there is no change.

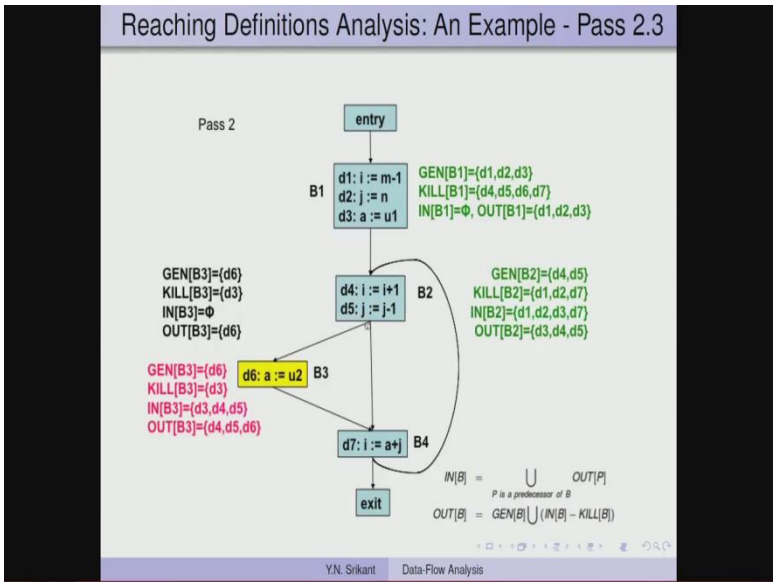
(Refer Slide Time 33:45)



But, once we consider B 2 we see changes, these are the old values of IN and OUT, so when we compute the new value of IN for this particular basic block B 2, we take the union of the two outsets coming from B 1 and another coming from B 4. So, the outset of B 1 here, so that would be d 1, d 2, d 3 and the outset of B 4 is d 7 and therefore, the inset of B 2 would be d 1, d 2, d 3, d 7 it is the union of the two.

So, the remember the old value was phi and the new value is different, even though there was no change of anything here there has been a change here, so what about the OUT value of B 2 obviously, as usual the GEN value d 4, d 5 gets included in the outset, then we take the IN value and remove the KILL value. So, if we do that IN B 2 is d 1, d 2, d 3, d 7, KILL B 2 is d 1, d 2, d 7, so only d 3 remains and that gets included into B 2. So, d 3 is still relevant it goes through, whereas d 1 and d 2 are not relevant, because they are being redefined here that is quite intuitive for us.

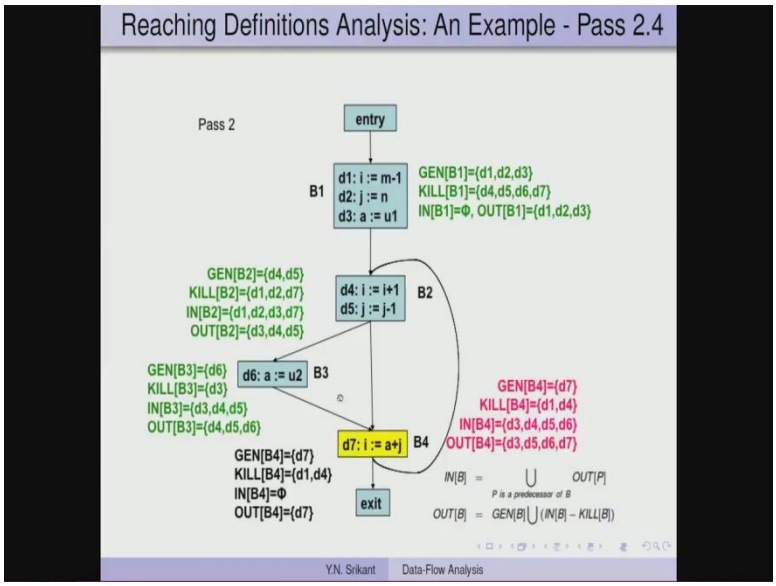
(Refer Slide Time 35:22)



Then the basic block B 3, again the IN value is phi, OUT value is d 6 whereas, here the IN value will be the new OUT value of B 2, the new OUT value of B 2 is d 3, d 4, d 5, so the new in value of B 3 will become d 3, d 4, d 5 it is different from IN. The OUT value of course, is the GEN value first that is included that is d phi and then so that is d 6, that is d 6 which is included and then we take the IN value d 3, d 4, d 5 and remove the KILL that is d 3, so d 4 and d 5 will also be included in OUT.

So, d 4, d 5 it implies that d 3 cannot flow through B 3, which is very obvious, because it is being redefined here, a is being redefined here, but d 4, d 5 will obviously flow through B 3, so that is what OUT B 3 signifies. So, what comes OUT here is this definition and these two definitions which are coming OUT.

(Refer Slide Time 36:44)



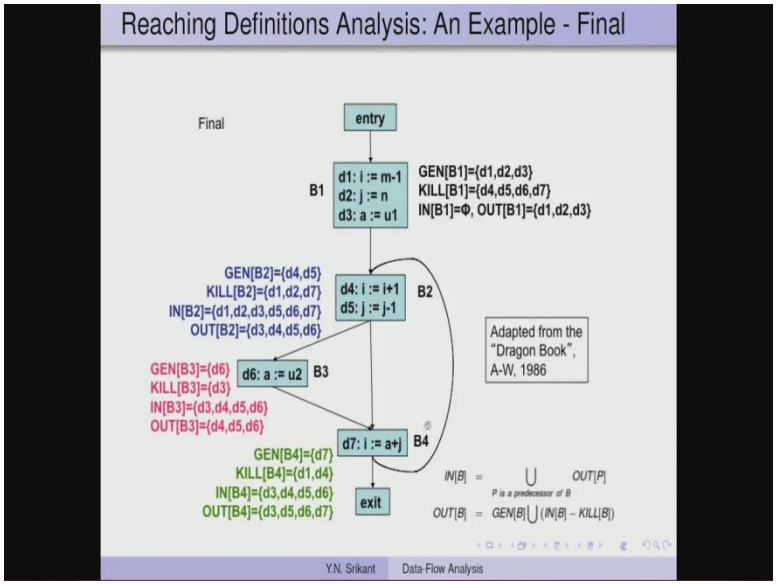
The last 1 is B 4, so of you consider B 4 IN of course, was phi as usual, now the IN value of B 4 is the union of the two outsets, one coming along this direction and the other coming along this direction, OUT of B 3 is d 4, d 5, d 6 and the OUT of B 2 is d 3, d 4, d 5, so we get IN of IN will be d 3, d 4, d 5, d 6 the union of the two. What about the outset it includes d 7, because it is GEN and then we take in and remove the KILL part, so if you remove the KILL part d 4 goes OUT, d 1 is extra which is immaterial, so we include d 3, d 5, d 6 and d 7 as the outset.

So, these are the four new values that have been computed for B 1, B 2, B 3, B 4 and in doing so we have actually looked at the basic blocks in a particular order, B 1, B 2, then B 3 and B 4, actually we could have looked at the basic blocks in the reverse order also B 4, B 3, B 2 and B 1. Since we are going to iterate through the basic blocks in the control flow graph, a number of times and make sure that none of the values of the IN and OUT sets of any basic block remain the same, the order in which we visit the basic blocks is immaterial.

But, it so happens that the even though the values do not change, values do not depend on the order of traversal of the control flow graph, the number of iteration required will definitely change, depending on the order in which we visit them. The heuristic, one of the heuristics which is used is to use a depth first search order, we will discuss this a little

later, so assuming that we use a depth first search order the iterations converge to the fix point value very quickly.

(Refer Slide Time 38:55)



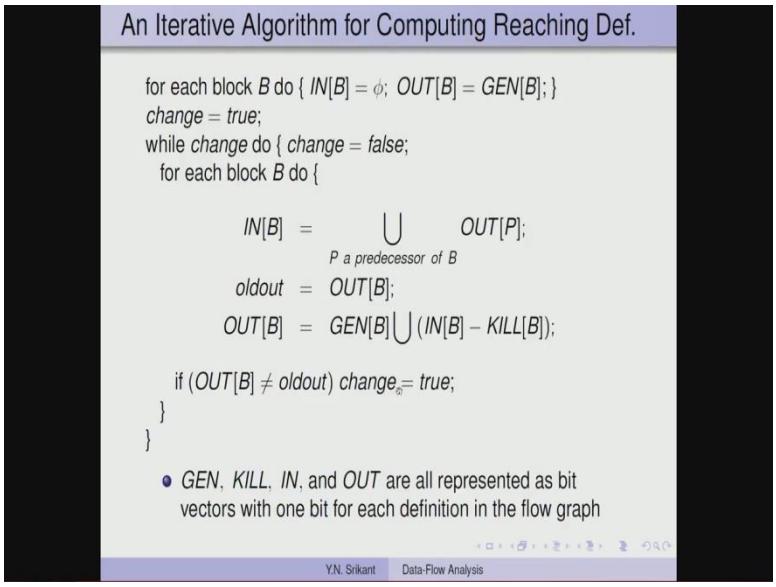
So, this is the final result I would encourage you to verify it, look at the this block this set of results and then apply the same data flow equations once more and then get these values. And after this values are reached any number of iterations further will not give you any changes, will not provide any changes, so these are the fixed point values and these happen to be the reaching definitions for the various basic blocks.

So, let us just take a look at them OUT B 1 is d 1, d 2, d 3 which is very clear, so all these definitions reach this point OUT B 2 is d 3, d 4, d 5, d 6, d 4, d 5 are very clear, then d 3 is also very clear it comes out of this. And then d 6 is some value which is defined here, it goes through the loop and then comes back again, so there are two definitions of a one here and one here, neither overtakes or overwrites the other one. In the first iteration this is relevant and then onwards in some of the iterations if this part is taken, then this a becomes relevant, so both the definitions of a reach this point.

Then OUT B 3 here for the basic block B 3 is d 4, d 5, d 6, so d 6 is very clear d 4, d 5 also come out then finally, OUT B 4 is d 3, d 5, d 6, d 7, so d 7 is very clear, d 6 is very clear, d 4, d 5 d 4 does not reach, because i is redefined here, but d 5 definitely reaches the end of the basic block here. So, this is how the definitions reach the end of the basic block, so remember even though we have no idea of the execution path, we take the

definition which along all paths take union and then say that these are the definitions which reach this particular point. So, this is a conservative estimate, because at execution time exactly one path will be taken, whereas during static analysis time we are going to consider all the paths.

(Refer Slide Time 41:23)



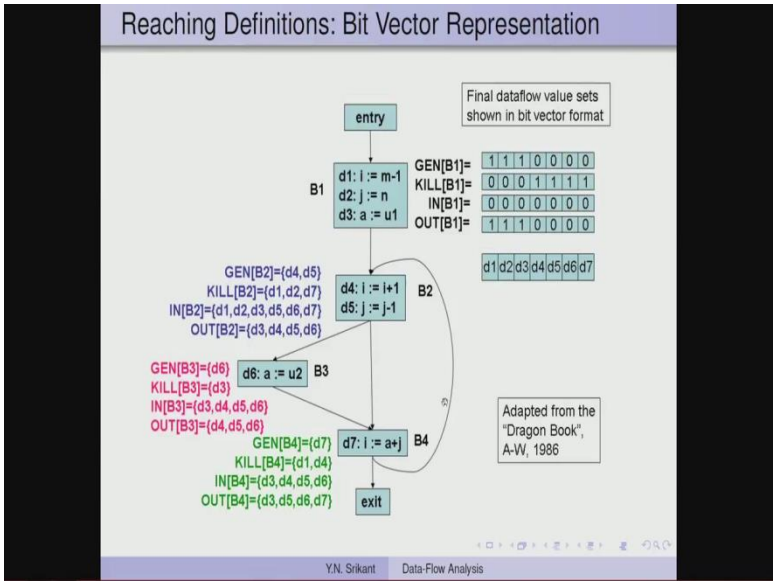
So, now it is time to look at the algorithm in a program like fashion, the algorithm is quite simple this is as I said an iterative algorithm, first of all initialization for each block B do IN B equal to phi and OUT B equal to GEN B, so that completes the initialization. Then to detect whether there has been a change in the computation of values, we use a flag change equal to true initially and then we loop while change do, so set change to false.

And then for each of the basic blocks apply the data flow equations IN B equal to something compute it, then store the old value of out and recomputed the new value of OUT. So, if OUT has changed, then changes if OUT B not equal to old out, that means out has changed, then we reset change to true again and then loop once more. So, we keep doing this until none of the OUT values change, the reason why we are checking only the OUT and not the IN is OUT is dependent on IN, so if there is a change IN, it will automatically be reflected in the change in OUT.

If we had reversed these two equations OUT first and then IN then we would have checked IN here, instead of and store old in instead of old OUT, so of course there is

nothing wrong in storing both old IN and old OUT and checking them, but that is not really necessary. Now, we still have to look at the data structures which are necessary to store the various data flow values, so this iteration will go on until there is no change and once there is no change we quit. So, we actually use bit vectors for the, with one bit for each definition in the program, let me show you an example.

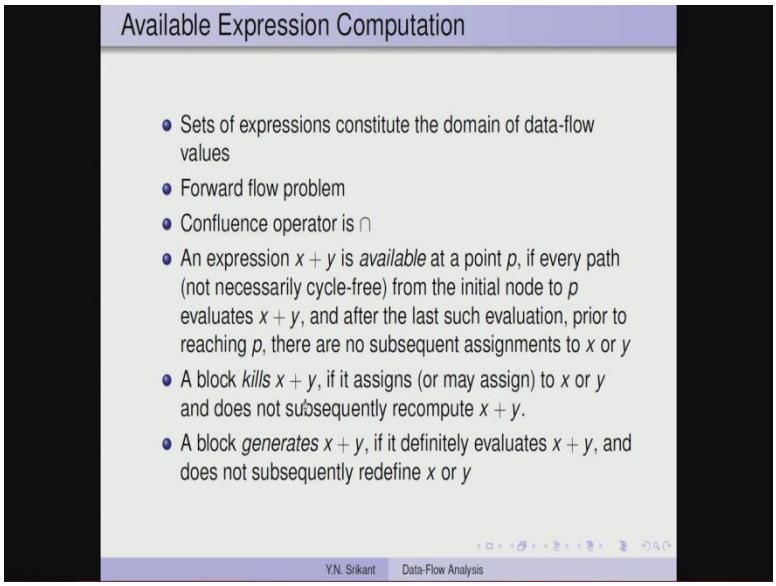
(Refer Slide Time 43:27)



So, here there are actually only seven definitions in the whole program in the control flow graph, so a bit vector of say seven with one bit for each is sufficient to store all the definitions. So, the first bit always stores corresponds to d 1, second bit corresponds to d 2 etcetera, etcetera so the position of the bit itself gives you the definition number, so all our sets GEN, KILL, IN and OUT will also be seven bit vectors bit vectors. So, for example, the set GEN B 1 actually has d 1, d 2, d 3, so it has once in the three places d 1, d 2, d 3 and the rest of them as 0.

The KILL set has a d 1, d 2, d 3 as 0 and d 4, d 5, d 6, d 7 as 1, then IN B was initialized to phi, so this is and remains phi's as I said, so these are all 0's and the OUT B 1 value d 1, d 2, d 3 is represented here. So, this is how the data flow values are stored as bit vectors, the same is true for this basic block, this basic block and also this last basic block.

(Refer Slide Time 45:03)



The data flow analysis problem that we are going to deal with is called as the available expressions computation, so here this problem becomes a very important problem to perform the global common sub-expression elimination for optimization. The sets of expressions constitute the domain of data flow values here, in the reaching definitions problem it was the set of definitions, here we are going to look at the sets of expressions and they constitute the domain of data flow values.

This is a forward flow problem and the confluence operator is intersection, by the way the confluence operator in the reaching definitions problem was the union operator, so let me show you this is here ((Refer Time: 46:08)). So, we have an equation for OUT in terms of IN, so as I said this determines the direction of the data flow and we have the confluence operator which is union here. Whenever we combine the values at a join point in the control flow graph, you must use the confluence operator.

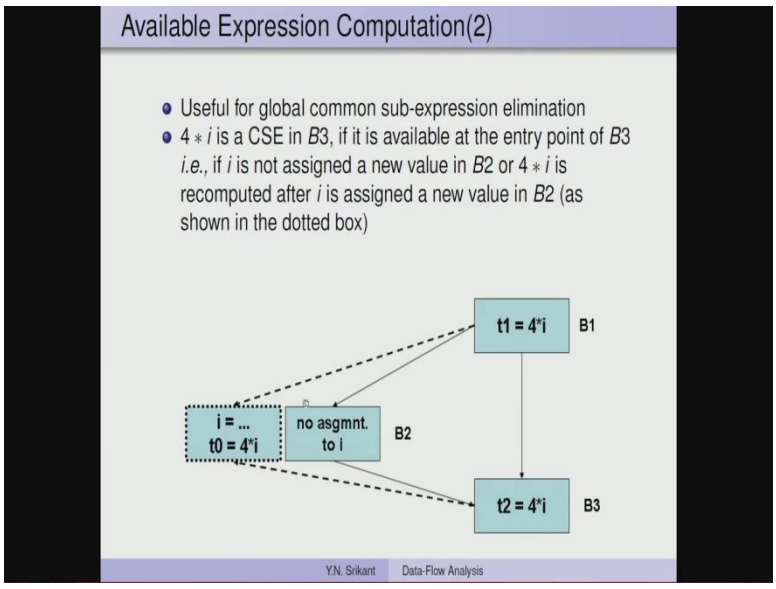
So, in the available expression computation problem, it is a forward flow with a confluence operator as intersection in the previous case it was union, here it is intersection this will become clear very soon. Now, the schema for the data flow analysis, requires the equations we will provide the equations very soon, and it also requires GEN and KILL sets. But, before that let us define what exactly is availability of and expression, an expression x plus y is said to be available at a point p, if every path not necessarily cycle free, so we could go in cycles.

But, what is important is the path must begin from the initial node of the control flow graph, so from the initial node to p, so you must consider every path here in the reaching definitions case we considered any path, here we must consider every path from the initial node to p and it must evaluate the expression x plus y. So, and after the last such evaluation prior to reaching p, there are no subsequent assignments to either x or y.

The important point here is there must be a computation of x plus y from the initial node to p along every path that is number 1 and the other important point is after the last such computation modifications to expend x or y should not happen, if you modify either x or y or both, then the value of x plus y changes. So, therefore, changes to x or y or both should not happen after the last computation of x plus y, otherwise the value of x plus y would have really changed.

So, this is availability most important it must be along all paths and after the last computation of x plus y no changes to either x or y, what do we mean by killing an expression here. So, a block kills x plus y, if it assigns or may assign to either x or y or both and it does not subsequently recompute x plus y. So, the point is we have a computation of x plus y, then we have redefined x let us say, if you recomputed x plus y then the block does not KILL x plus y, but if it simply modifies x and when does not recompute x plus y, then it kills x plus y, so let me show you an example here.

(Refer Slide Time 49:35)

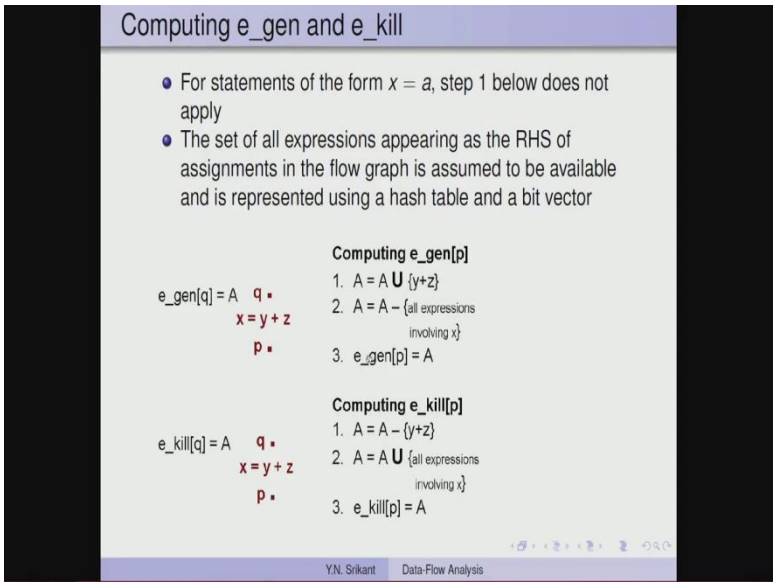


Here is 4 star i a computation, here is another 4 star i another computation and here there are two possibilities, one possibility is to go along, it is not that both the blocks are present at the same time, I have just written the second block as an alternative in dotted lines. So, either this block is present or this block is present, so if there is no assignment to i obviously, 4 star i will not get changed along this path at all, so we can say it is available along this path and also this path, so it is available along all paths at this point.

And suppose we redefine i, so we have defined 4 star i, computed 4 star i here, now we have redefined i here, but then we also computed 4 star i again, so in this case also 4 star i is available along this path, because we have recomputed 4 star i. 4 star i is available along this path, 4 star i is available along this path also, remember we are not considering looking at the exact value of 4 star i here. So, what we are saying is whether we use 4 star i along this path or this path, we do not have to recompute it here, that is all we are trying to say during the common sub-expression elimination.

We want to avoid computation of 4 star i here other by reusing this value or this value, it really does not matter, but suppose there was no recomputation of 4 star i here, it was just a modification of i. Then star this expression 4 star i is not available along this path, so we say 4 star i is not available at this point, what about the generation of x plus y a block generates x plus y, if it definitely evaluates x plus y and does not subsequently redefine either x or y or both. So, this is very simple we just compute x plus y and then do not modify either x or y.

(Refer Slide Time 52:01)



Now, how do we compute this GEN and KILL let us call them as e GEN and e KILL, so let us assume that the statements are of the form x equal to y plus z, we are trying to compute it for each one of the statements in the basic block. So, this must be done one statement at a time and then the effect of the entire basic block will be felt at the end, for the point q let us assume that the e GEN is given. Obviously, at the beginning of the basic block this will be empty, so let the e GEN for this point q be a, now we have this statement x equal to y plus z.

So, what do we do we now y plus z gets computed, so it is actually generated by this statement, so we do a equal to a union y plus z, then x is being redefined here, so all the you expressions involving x will have to be removed from a, so we that is the killing part. So, a equal to a minus all expressions involving x at this point t, therefore e GEN p is the new value of a, so this is how we compute e GEN p.

And now if there is another statement after this, the effect of that statement will be computed exactly like this, because e GEN p is now available. How do you compute e KILL q again your given e KILL at this point, so let us say that is a, now x equal to y plus z is the statement for which we have to compute the effect. So, now y plus z is being computed here, so it is not killed it must be removed from KILL even if it was killed before, now it is being recomputed, so we must remove it from KILL.

Now, x is being redefined here, so x kills all the expressions involving x, so they will all be added to a, so this is the compliment of what we did here, so a equal to a union all expressions involving x. And at this point e KILL of p will be new value of a, so this is how we actually compute the GEN and KILL for each of the statements in the basic block, they are all combined together and used.

So, the set of all expressions appearing on as a right hand side assignments in the flow graph is assumed to be available, so this is our universal set of all the expressions. But, then what do you do with these, we have used a bit vector representation for the reaching definitions and using bit vectors, the union and intersection operations are all trivial. In the case of expressions we cannot directly use a bit vector, what we do is to store all the expressions using a hash table, so we hash them and put them into a table, and the index of that hash table is used as the bit position. So, we make a bit vector OUT of these index positions of the various expressions and that will be used as our bit vector for various union and intersection operations. So, we let us stop here and then continue with the discussion of available expressions problem in the next part of the lecture.

Thank you.