Compiler Design

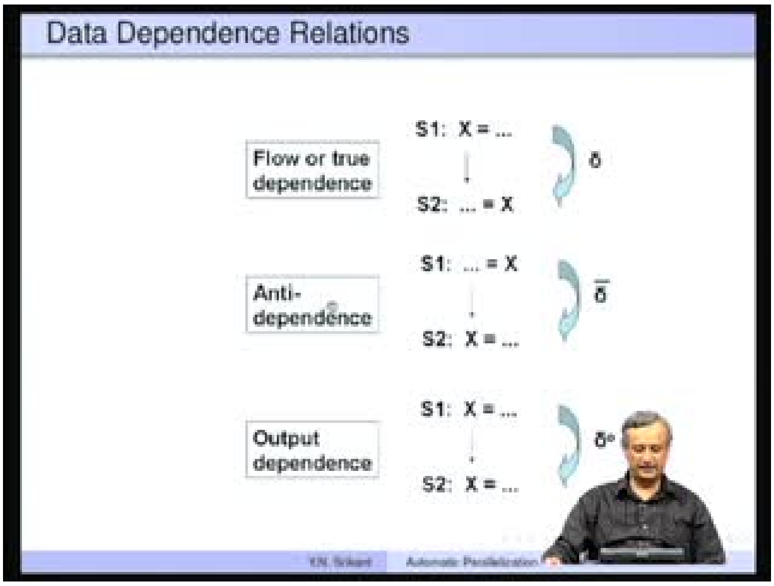
Prof. Y. N. Srikant
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Module No. # 14

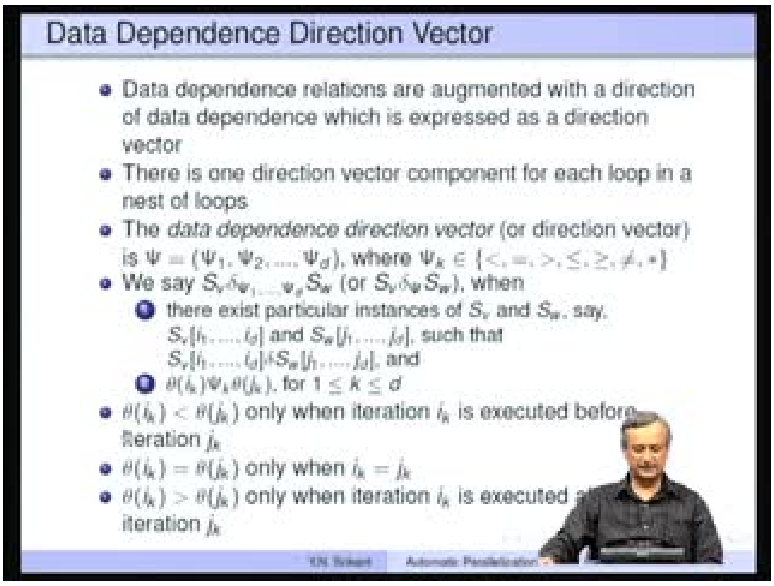
Lecture No. # 26

Automatic Parallelization–Part 3

(Refer Slide Time: 00:19)

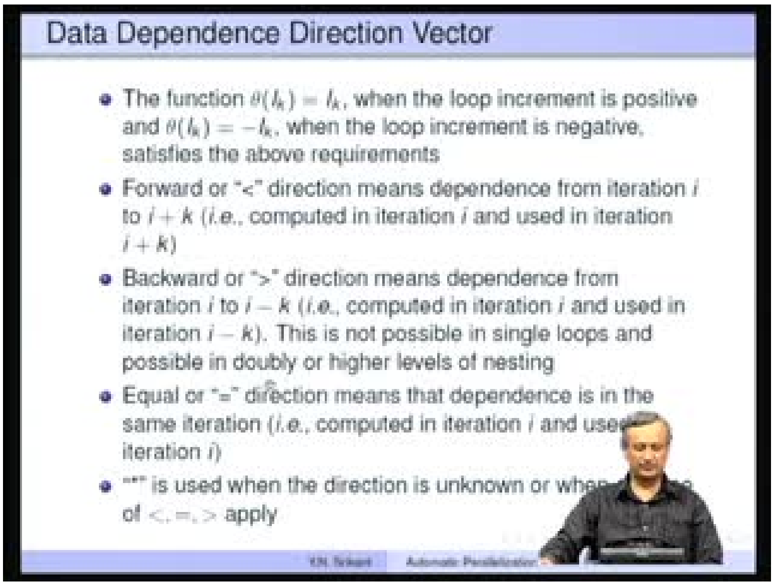


(Refer Slide Time: 00:31)



Welcome to part 3 of the lecture on automatic parallelization. So far, we have learnt about Data Dependence Relations - there are three types of relations: Flow, Anti and Output dependence. We learnt about the Data Dependence Direction Vector which is one of the most important concepts. So, if there are nested loops - let us say two statements nested within the loop - we look at instances of these statement S v and S w for some iteration values i 1 to i d and j 1 to j d, we look at the statements which are executing with those iteration values at that instant and those are called the instances.

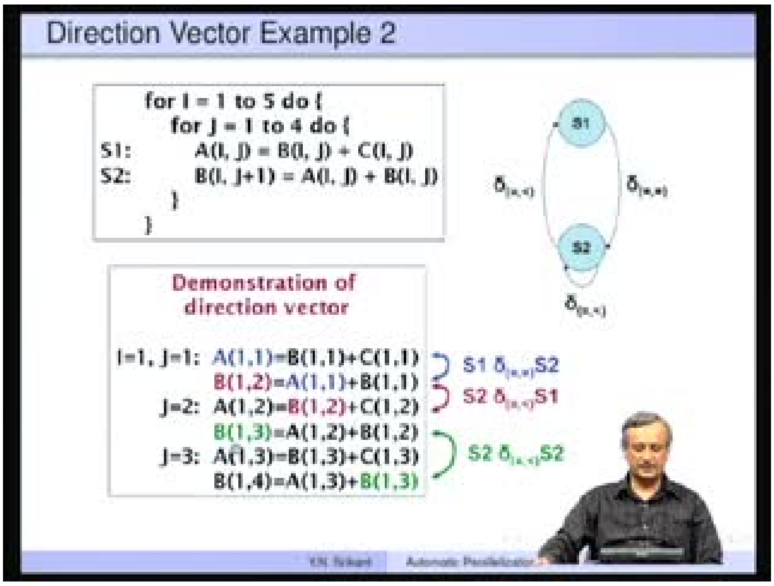
(Refer Slide Time: 01:38)



So, if there is a dependence between these two statements, the value which is completed in S v is actually used in S w and the relationship between these iteration counter values i 1, j 1, i 2, j 2 etcetera, is captured by the theta function; so, if the increments are positive, then theta could be a very simple function - theta I k equal to I k and if the loop increment is negative, then we just say theta I k equal to minus I k, and that is how the relationship would be.

There are three types of Direction Vectors possible: less than, greater than and equal to. Less than means that the dependence is from the iteration i to i plus k; greater than means that the dependence is from iteration i to i minus k - this is not possible in single loops; equal to means that the dependence is in the same iteration - computed in iteration i and used in iteration i itself.

(Refer Slide Time: 02:15)

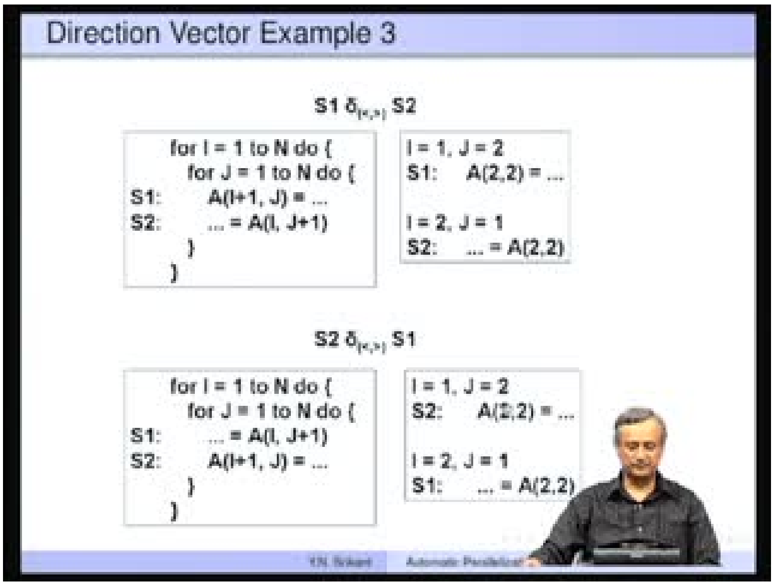


So, here is a very simple example to help you recapitulate what we have done - in this example, there are two loops - I and J; there are two statements - S1 and S2; look at the dependence between these two statements; observe A(I,J) on the left hand side of S1 and the right on the right hand side of S2 - these two have identical subscripts, and when we unroll the loop you know that A(1,1) is computed in I equal to 1 and J equal to 1 and it is also used in the same iteration I equal to 1 and J equal to 1. So, between these two, there is a dependence S1 delta equal to, equal to S2, because we have same I and J values for both these instances of statements S1 and S2.

Similarly, between these two - B(1,2) and B(1,2), we have a dependence from S2 to S1 with direction vector - equal to and less than; that is because it is S2 computes B(1,2) with I equal to 1 and J equal to 1 and S1 consumes the same values B(1,2) with I equal to 1 and J equal to 2. So the relationship between I is equal to, and between J it is 1 less than 2.

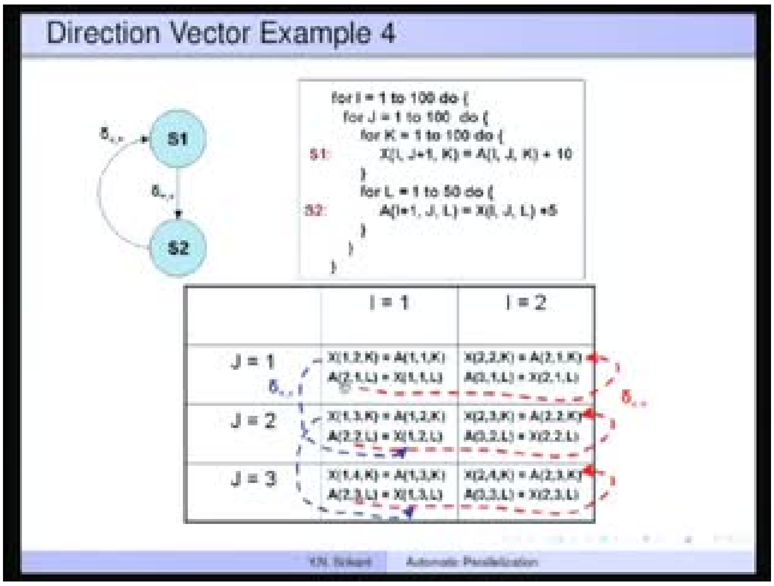
Thirdly, S2 delta equal to, less than S2 is a loop here, in the dependence diagram. That arises because of this B(1,3), which is computed in S2 and used in S2 again, but with a different J value; I value is the same.

(Refer Slide Time: 03:56)



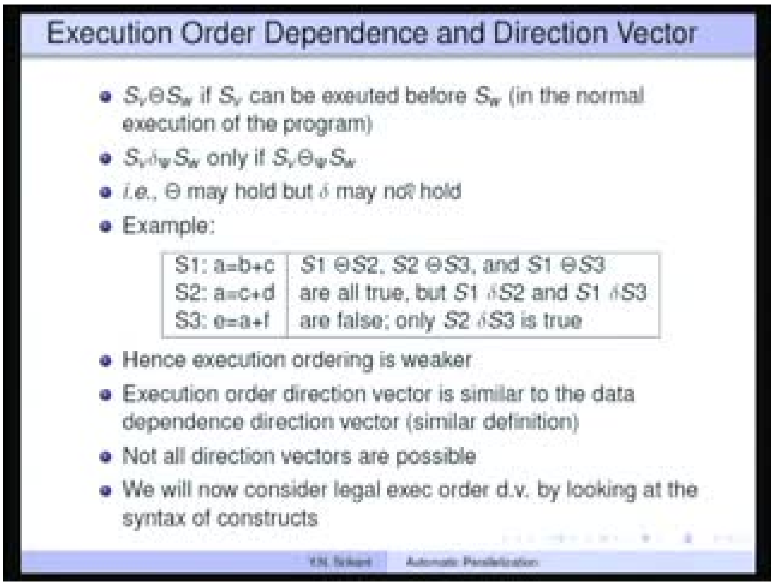
And here is an example of S1 delta less than, greater than S2. This is possible because I value increases from 1 to 2, but J value decreases from 2 to 1 - this is legal because we are really looking at a different set of iterations of J for a different value of I; so, there is no illegality here; similar example in this part as well.

(Refer Slide Time: 04:25)



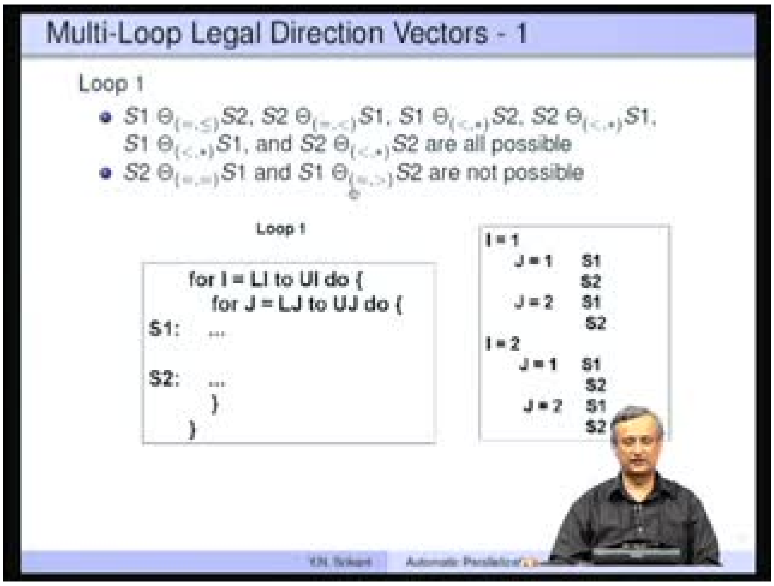
And finally, here is the example which shows that there is a dependence from S1 to S2 with delta equal to and less than - that is shown here; equal to and less than - computed here and used here - X(1,2,K) X(1,2,L). So, S1 and S2 - these two actually show that dependence; similarly, this is from S2 to S1 - A(2,1,l) and A(2,1,l); so, this is again delta less than, equal to; so, what is computed here, is used here. So, these are the examples that we have already looked at.

(Refer Slide Time: 05:06)



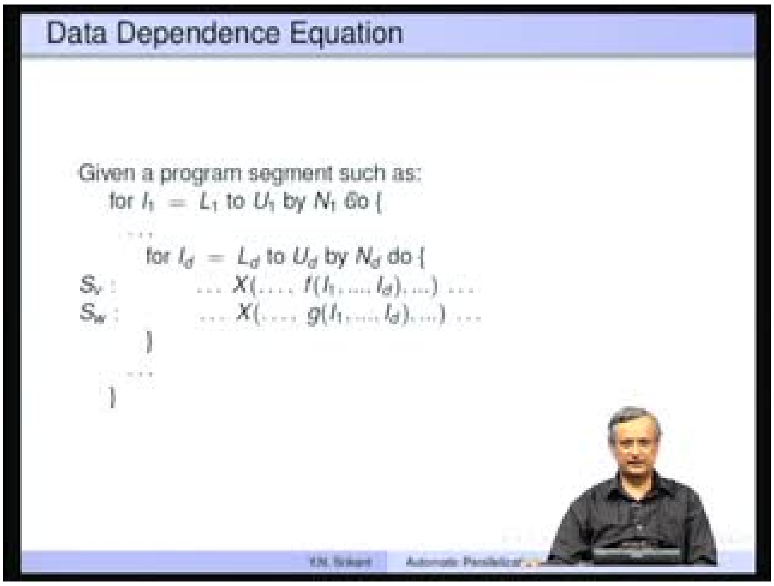
Then we saw the execution order dependence and direction vector. Basically, we are looking at the flow of the program and we say that if S v can be executed before S w, then we have S v theta S w and this execution order dependence is weaker than the data dependence itself. So, that example is here; we will not repeat the example again and there is a direction vector attached to this execution order dependence in exactly the same way we attached on to the data dependence itself.

(Refer Slide Time: 05:56)



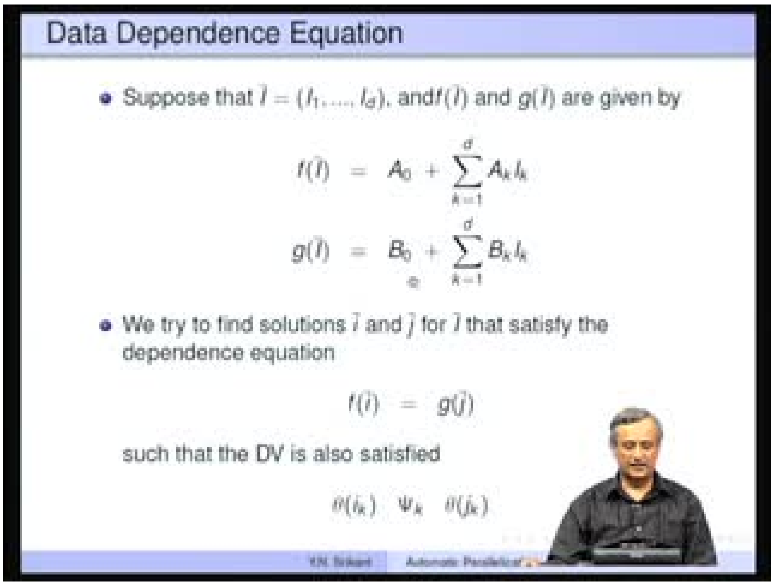
Now, not all direction vectors for execution order dependences are possible. So, we can consider actually the legal execution order direction vectors by looking at the syntax itself. So, here is a very simple example - for this particular loop nest, S1 theta equal to, less than or equal to S2 etcetera are all possible, but S2 theta equal to equal to S1 is not possible for the simple reason that S2 we cannot reorder the statements at all; in this S1 and S2 remain the same; so, we really cannot reorder statement; so, in the same iteration of I and J, we cannot execute S2 first and then S1. S1 theta equal to, greater than S2 is obviously not possible, because greater than is a relationship which is valid only when the first is less than.

(Refer Slide Time: 06:36)



Then, we looked at the data dependence equation; there are two statements S v and S w, which have the array references X of f, I 1 to I d; and X of g, I 1 to I d; so, there is the loop nest of 1 to d.

(Refer Slide Time: 06:53)

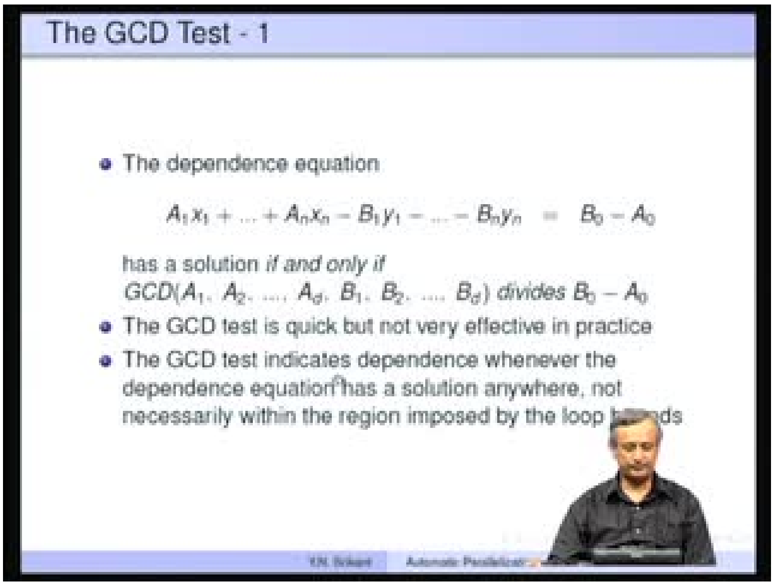


Here, we have f I bar is a naught plus k equal to 1 to d sigma A k I k. So, this is just to show that the constraint on the form of the subscript should be a linear function of the loop indices; similarly, the second one is also a linear function of the loop indices.

Now, we try to find solutions for i bar and j bar for the equation f I bar equal to g I bar; then the dependence equation is satisfied as f i bar equal to f j bar; so, those values of i bar little i bar and little j bar are the ones we try to find, but when we satisfy that equation, the direction vector relationship must also be satisfied; so, there will i 1 less than i 2 etcetera. These are the relationship which will be satisfied.

So, we also try to normalize the index; so, that is dealt with here so that we have an increment of loop lower bound as 1 etcetera. The relationship between the loop index variables with the normalized and non-normalized version is here. This is the fairly straight forward relation - L k plus I k n into N k is equal to I k - that is the relationship. With that, we modify the equation here, but from now on, we are just going to assume that, always, loop are normalized with loop lower bound as 1 and the increment as 1.

(Refer Slide Time: 08:30)



The GCD test: we briefly looked at; we have a dependence equation A 1 x 1 plus A 2 x 2 etcetera A n x n; and then the right hand side - other statement would have B 1 y 1 B 2 y 3 etcetera B n y n; and constant values are B naught and A naught.

So, if we solve this equation when this is equal to B naught minus A naught, then we have found a solution to the dependence equation. This has been shown to have solution if and only if GCD of A 1 to A d and B 1 to B d divides B naught minus A naught.

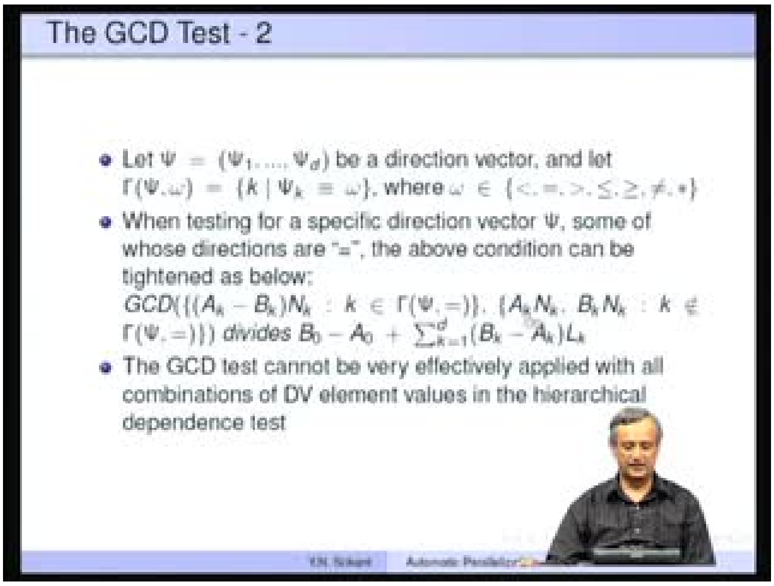
We will see example of this now; the GCD test is very quick, but not very effective in practice - we will see that very soon and it indicate dependence whenever the dependence equation has a solution anywhere, but not necessary with in the region imposed by the loop bounds; this is the major problem.

When we have a direction vector also available and we want to test for a particular direction that is possible; only equal to direction can be tested separately.

So, let us say the function gamma psi coma omega is k, such that psi k is equal to omega, where omega is one of the direction vectors; in other words, we are trying to pick the component of the direction vectors, which is equal to omega; that is what this gamma function does. When we are testing for specific direction vector psi, some of whose directions are equal to, and then the above condition that we saw - GCD condition, can be straight slightly tightened.

So, what we really do is pick those components rather those coefficients, which correspond to the equal to value of the direction vector; and make that as A k minus B k into N k; the other remain as they are; only thing is instead of A k, B k etcetera, now, we use A k, N k, B k, N k etcetera for the others and the condition would be modified slightly as divides B naught minus A naught plus sigma k equal to 1 to d, B k minus A k, L k.

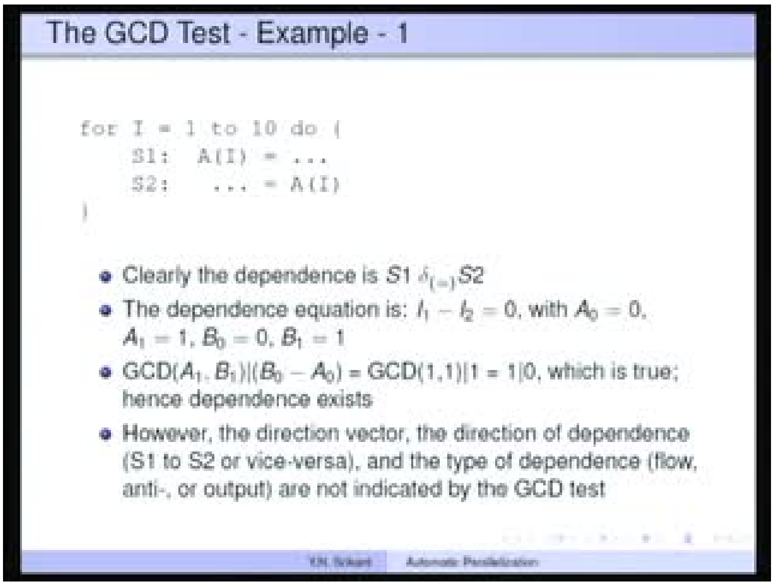
(Refer Slide Time: 10:52)



Whereas here, we just had GCD of A 1 to A d, B 1 to B d divides B naught minus A naught; here, we have A k minus B k into N k; similarly, A k, N k, B k, N k; for the rest of them; and on the right hand side, we have B naught minus A naught plus the sum of B k minus A k into L k.

Even this test is not very effective, and problem is that it requires l k it requires the direction vector and so on, but we cannot use it so with other combination of direction vectors, for example - less than etcetera. It can be used only with equal to; so, this is not very useful to us later on.

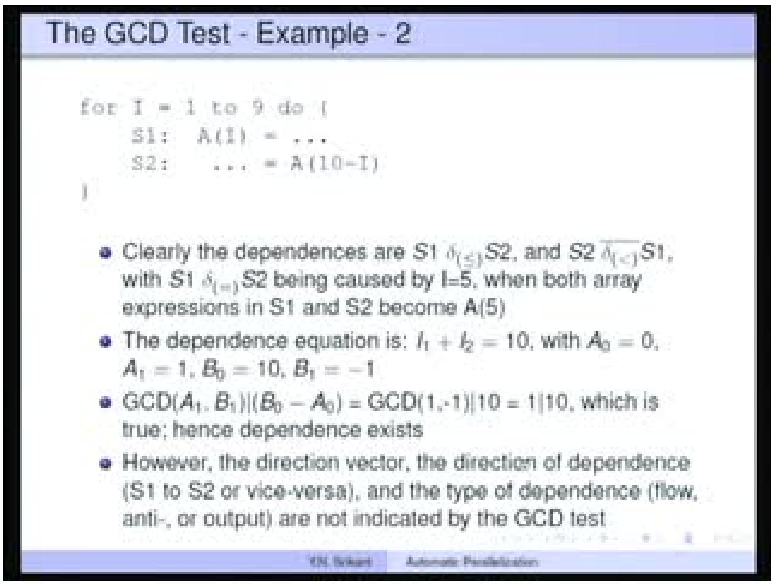
(Refer Slide Time: 11:29)



Let us look at some examples; here is a loop I equal to 1 to 10, do S1 is A I equal to something; and S2 is … equal to A I. Now, if you unroll the loop, we see that A 1 equal to A 1; A 2 equal to A 2 etcetera. So, the dependence is within the iteration; it does not go beyond the iteration; so, we have S1 delta equal to S2; that is the dependence in this particular loop.

Let us see what the dependence equation is? On the left hand side, we have I; that becomes I 1; on the right hand side expression we have I again; so, that becomes I 2; so, instead of writing this equal to this, we simply wrote I 1 minus I 2 equal to 0. So, that is your dependence equation; here A naught is 0, A 1 is 1; this is A 1, B naught is 0 and B 1 is 1 so, that is this. Now, the GCD test becomes GCD of A 1, B 1 divides B naught minus A naught; so, GCD of 1 coma 1 divides B naught minus A naught is 0; so, 1 divides 0, which is true; this should have been 0. Hence, the dependence exists - that is what the GCD test really tells us. However, the direction vector the depend direction of dependence S1 to S2, vice versa and the type of dependence - flow anti or output, these are not indicated by the GCD test.

(Refer Slide Time: 13:07)



So, example number 2 - we have I equal 1 to 9; A I on in S1, equal to something; and in S2, we have something equal to A of 10 minus I; this is special. So, if you unroll the loop, you see that you will have A 1 equal to A 9, A 2 equal to A 8, A 3 equal to A 7, A 4 equal to A 6 and A 5 equal to A 5; then the loop runs in the other direction - A 6 equal to A 4, A 7 equal to A 3 etcetera.

So, in the first half of the loop - from say 1 to 4, statement S1 would be assigning something to A I; so, A 1, A 2, A 3, A 4 etcetera are being a signed. Whereas, here, on this side we would have A 9, A 8, A 7, A 6 etcetera being read.

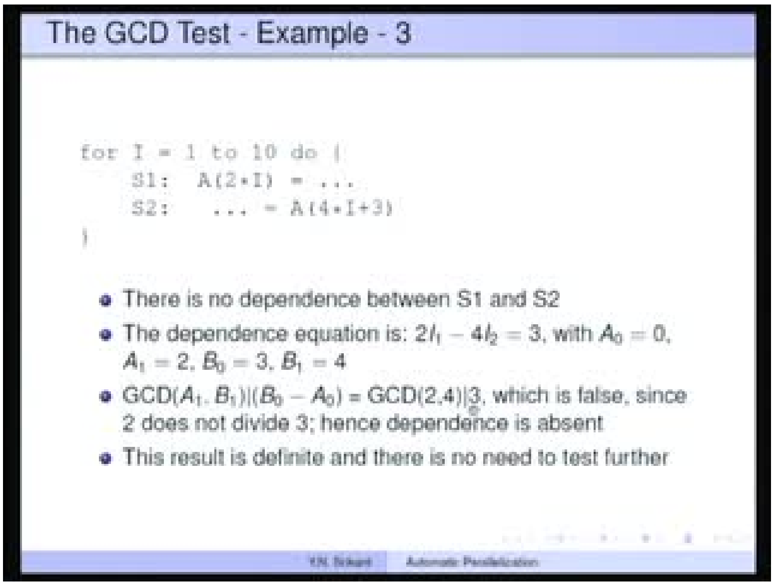
So, in the first half of the loop, we will have S1 delta, whatever is in the middle of the loop, not in the first half of the loop - in the middle of the loop we will have A 5 equal to A 5 - that dependence will give us S1 delta equal to S2 so, that is where one of these equal to dependences comes.

Then we have assigned then read; so these two - reverse when the loop run from 6 to 9 so, we get S1 delta less than S2 and S2 delta less than S1. So, together in the 2 hales of the loop, we are going to assign and read the same locations; so, that is how S1 delta less than S2 and S2 delta bar less than S1 come in to picture; so, these two are actually going to cause these dependencies.

So, let us see what happens in the dependence equation: I 1 plus I 2 equal 10 - is the dependence equation; why? this would be I 1 this would be I 2 and this is 10 so, I 1 equal to 10 minus I; so, that would be nothing but I 1 plus I 2 equal to 10; that is the standard form of equation; A naught equal to 0, A 1 equal to 1, B naught equal to 10 and B 1 equal to minus 1; so, this is the B 1, this is A 1; so, this is B naught so, A 1 equal to 1, B 1 equal to minus 1.

Now, the GCD becomes GCD of 1 coma minus 1; this and this, divides 10; so, that means 1 divides 10, which is correct; so, the dependence exists.

(Refer Slide Time: 16:15)



However, again, we cannot determine the direction vector, direction of dependence, type of dependence etcetera by this GCD test. So, we are going to look at the same examples with Banerjee's test, which is slightly more powerful than this a little later.

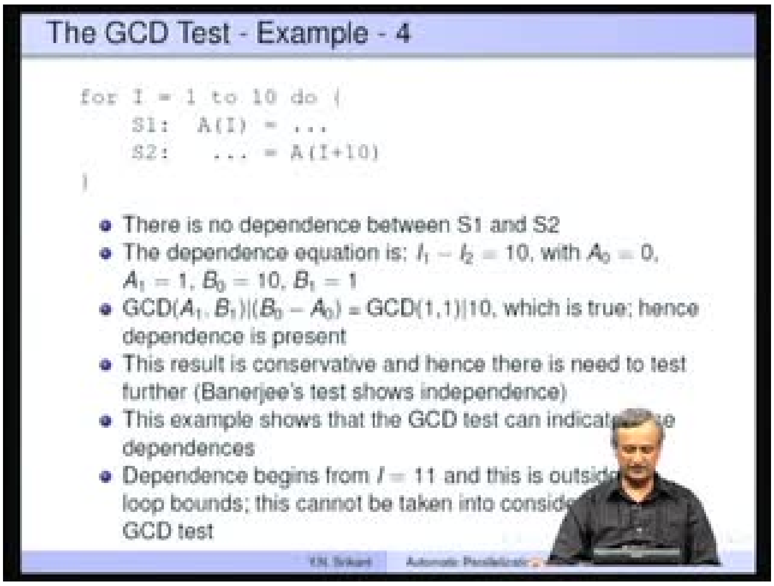
So, far, the 2 examples we saw told us that there is dependence. Now, let us look at an example where there is no dependence. This is A of 2 star I and this is A of 4 star I plus 3. If you expand the loop, you could see that this will be A 2, A 4, A 6 etcetera and here, we would have 4 plus 3, - I equal to 1 - this become 7; When I equal 2, it becomes 8 plus

3 - 11 so 11 plus 41 - 5; this and this (Refer Slide Time: 17:00), this is even and this is odd; they never intersect; so, that means there will be no dependence between S1 and S2.

What does our dependence test tell us? So, the equation is here this side will be 2I 1 and this side it will be 4I to plus 3 so, we have 2I 1 minus 4I 2 equal to 3 so, this is A 1 this is B 1, this is B naught minus a naught.

So, again we have GCD of 2 coma 4, 2 coma 4 divide 3 which is false, that means there is no dependence so, in the previous 2 examples; this (Refer Slide Time: 17:34) and this (Refer Slide Time: 17:35) when the test said there is a dependence; it is a conservative estimate in other words there could be dependence possibly there is no dependence as well. If the GCD test that there is no dependence there is no need to test further so, that is how the GCD test is.

(Refer Slide Time: 17:56)

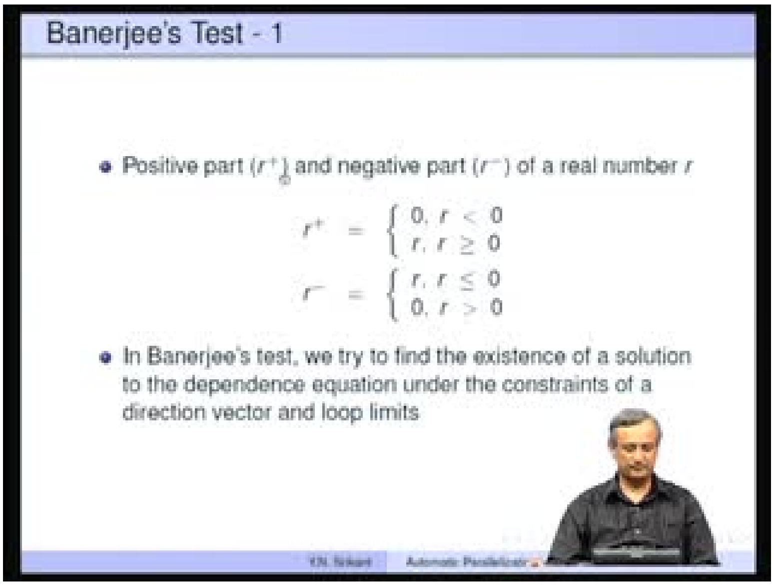


So, now the next example, we have A I in S1 and in S2 we have A I plus 10. Actually speaking there is no dependence between the 2 because the loop is running only from 1 to 10; this will be A 1, this will be A 11, this will be A 2 and A 12 etcetera and then the loop stop at A 10 so this become A 20. If the loop had run little further, 15 something like that this would have become A 11, A 12 etcetera. We would have possibly, had some dependence so, this would have this is in the I equal to 1 this would become A 11 so, we would have computed in A 11 in nitration number 11, but since, the loop run till only 10 there is not dependence between S1 and S2. The dependence equation is I 1 equal to I 2 plus 10 so, that is I 1 minus I 2 equal to 10, this is A 1, this is B 1 and this is B naught minus A naught.

So, that means we get GCD of A 1, B 1 divides B naught minus A naught that would daily become GCD of 1 coma 1 divides 10 which is true. The GCD test really tells us that there is dependence, but there is no really dependence so, this is this is the conservative nature of the GCD test why get this happened?

The problem is from I equal to 11 on words the dependence really begins. I equal to 11 is outside the bounds of this particular loop, but GCD test does not vary about the loop bound therefore, it is unable to decide that there is no dependence between S1 and S2.

(Refer Slide Time: 19:49)

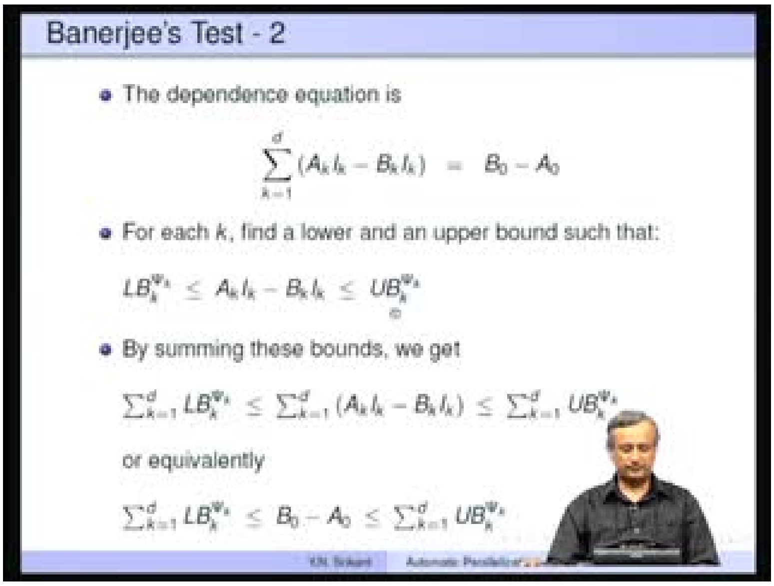


The next form of test which is more powerful called the Banerjee's test; will take this in to consideration. What is exactly Banerjee's test? Let us understand this carefully to begin with you must look at some definitions before we understand the notation and the Banerjee's test.

We define what is known as positive part and negative part of a real number. r plus is 0 if r is less than 0 so this is a positive and if it is a positive number r greater than equal to 0 then it is just the number itself so this is the positive part. In the negative part r minus it is r if r is negative and it is 0 if r is positive so this is negative part of the number.

So, what we really do is we try to find the existence of a solution to the dependence equation under the constraints of a direction vector and also the loop limits.

(Refer Slide Time: 20:45)



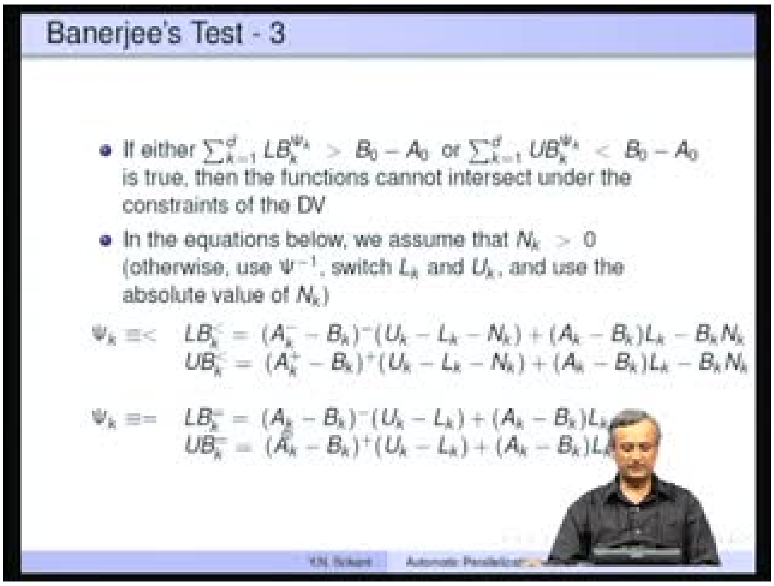
Let us look at the dependence equation again. So k equal to 1 to d A k I k minus B k I k is equal to B naught minus A naught so this is our dependence equation.

Now we want to find a lower and an upper bound for k. So k equal to 1 to d the values of k correspond to the loop indices it is a d nested loop that is what we are looking at.

So for each k we want to find a lower bound and upper bound such that LB k psi k. So again psi k is the direction vector component of psi is less than equal to A k I k minus B k I k less than equal to UB k psi k. Here, is a lower bound and here is a upper bound. let us some these constrains one by one. We would have LB 1 LB 2 etcetera here UB 1 UB 2 etcetera here A 1, I 1, B 1, I 1 etcetera here. You some of all this you get k equal to 1 to d of LB k k equal to 1 to d of UB k and in between we have k equal to 1 to d, A k I k minus B k I k.

So, this middle thing is nothing but b naught minus a naught. Therefore, we get k equal to 1 to d of LB k psi i k lees than equal to B naught minus A naught lees than equal to k equal to 1 to d of UB k psi i k.

(Refer Slide Time: 22:25)

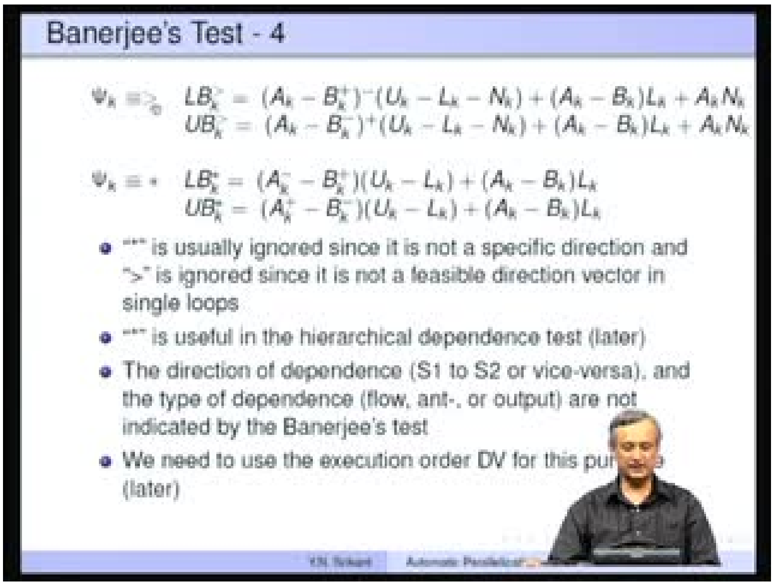


So, what Banerjee has really done is to provide the equations to compute LB k psi k for each type of direction vector in terms of the coefficient of the dependence equation. So, those are here.

So, now what is the advantage (Refer Slide Time: 22:38) of this particular setup constrains? That is here. If the constrains are violated in other words see it should be (Refer Slide Time: 22:46) this LB k psi k sum should be less than equal to B naught minus A naught; suppose this is not so or on this side B naught minus A naught must be less than equal to the sum of UB k psi k; suppose this is not so either one of them is violated. Then, if either this first side is violated are the second side is violated then the functions cannot intersect under the constraints of the direction vector. That means independents in that particular direction vector part.

We are assuming that the loop increment is greater than 0, but otherwise we can use the inverse of the direction vector which we will tell you later and then switch L k and U k and use the absolute value of N k. So no generality is lost if you use that the increment is greater than 0.

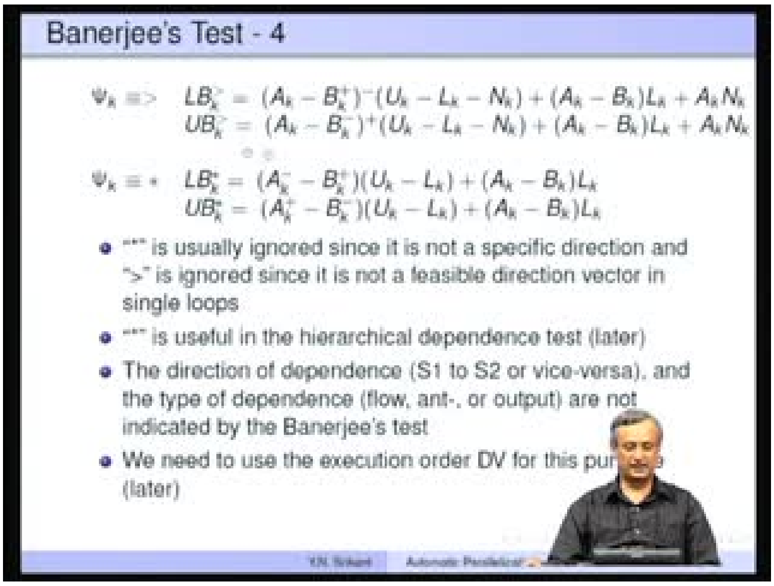
(Refer Slide Time: 23:58)



If psi k is less than is one of the direction vector components; psi k equal to equal to is the second direction vector component; greater than is a third one and star is the last so, for each of these combination Banerjee has provided equation to compute LB k this is LB k less than, UB k less than, LB k equal to UB k equal to etcetera. How do you read this right hand side? So, this A k minus is nothing but the negative part of the number A k and A k minus minus B k the whole thing again minus means that if this number is positive this entire this minus really says it will be a 0; if this number is negative then we use the number as it is.

Then we have U k minus L k minus N k so, U k and L k are the upper and lower bounds of the loop at level k. Then we have the coefficient A k minus B k into L k that again the k th level coefficient minus B k N k so that how read this.

(Refer Slide Time: 25:20)



This side is A k plus so that means; that is a positive part of the real number; A k plus minus B k whole thing plus – so, if this number is positive, we take it as it is, but if the number is negative, we make it 0; the other stuff remains the same. Similarly, for psi k equal to, we have two equations and then psi k equal to greater than, we have two more equations and psi equal to star which means it is really any direction has two more equations.

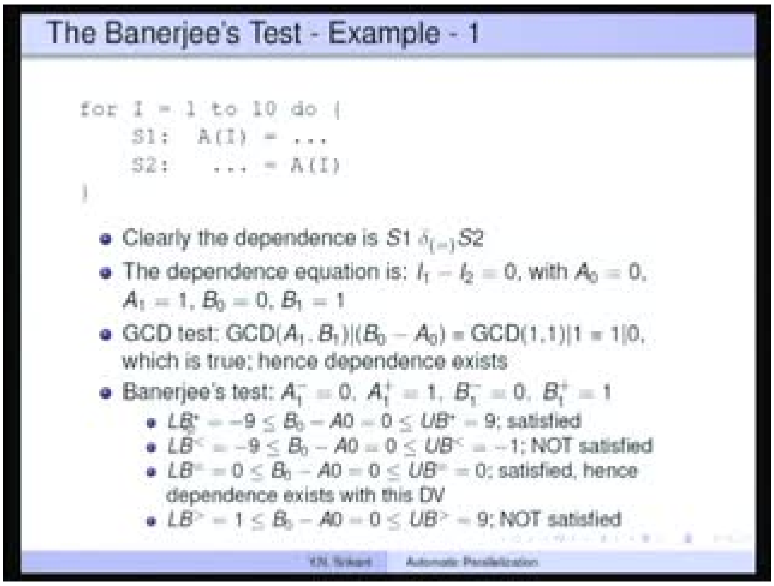
So, this star in if it is a singly nested loop or even w nested loop without the use of hierarchical dependence test this star is useless. We are going to see the hierarchical test a little later.

Star is usually ignored, if it is not a specific direction and greater than is ignored since it not a feasible direction vector in single loops. We are really going to have only the less than and equal to direction vector components.

Star is very useful in the hierarchical dependence test. We are going to see that a little later. So even then the direction of dependence, whether it is S1 to S2 or vice versa and the type of dependence: flow anti or outputs are not indicated by the Banerjee’s test.

We need to use the execution order direction vector for this purpose and we will see how to do this little later.

(Refer Slide Time: 26:39)



Let us look at some examples of using Banerjee's test. We are going to the run through the same examples that we did as before. So, we have A I here and A I here the we know that dependence is S1 delta equal to S2 is the same example that we saw during the GCD test. dependence equation is I 1 minus I 2 equal to 0 GCD test returns true and says a there is the dependence, but what does Banerjee's test say? Obviously it can either say yes or it can say no.

So, just for this example let us see what A i minus etcetera are. We know that A naught is 0 here and B naught is also 0 and A 1 is 1 and A 2 is 1 so A 1 minus is 0 because A 1 is plus 1; A 1 plus is 1 because that is the positive part, B 1 minus is 0 because B 1 is also plus positive quantity and B 1 plus is 1. If we substitute these values in the equation for LB star, LB less than, LB equal to and LB greater than we get these values: LB star becomes minus 9 LB less than also is minus 9, LB equal to is 0 and LB greater than is 1. B naught minus A naught is 0 that we know here B naught this side is 0 and if we substitute the values and calculate UB star UB less than equal to and greater than, we get UB star as 9 UB less than is minus 1 UB equal to is 0 and UB greater than is 9.

That means now you are ready to really check the constraint. So, LB star equal to minus 9 this is less than or equal to 0 no problem it is defiantly satisfied and 0 less than are equal to 9 there, is also satisfied. What I would like to point out here is when you check for star dependence if there is dependence in any of the other with any of the other direction vectors this star dependence constraint will always hold.

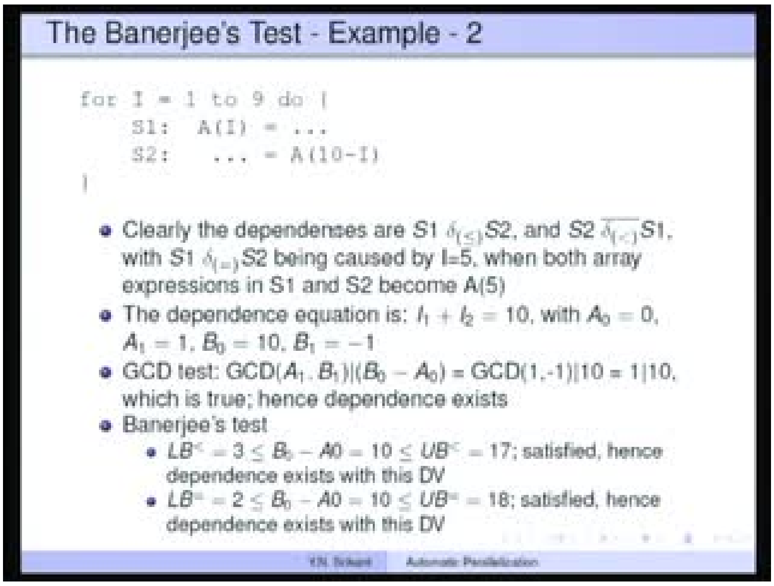
If that is showing a non-dependence then there can be no other dependence which is possible because this star really stand for any type of dependence. Then LB less than is minus 9 this is less than are equal to 0 so it is this part minus 9 less than are equal to 0 is indeed satisfied, but 0 less than are equal to minus 1 is definitely not satisfied. So, the constraint is violated that means, there can be no direction vector component with less than so this is not satisfied.

With equal to we have LB equal to 0 less than equal to 0 less than equal to another 0 so, this is satisfied so, that means dependence exists with that this direction vector i did not wright that comment with star because star really stand for any direction vector.

So, equal to direction there is a direction vector, there is a dependence and that we know is true. The last one greater than is shown only for completion with single loop there can be no direction vector with greater than direction. This is 1 less than equal to 0, 0 greater than equal to 1 is obviously not satisfied and 0 less than are equal to 9 is satisfied, but one side is enough to kill the constraint so this is not satisfied.

So, the only one which is satisfied and is meaningful is the LB equal to. So, we know that there is the dependence with direction vector equal to, but we do not know whether the dependence is from S1to S2 are from S2 to S1, we do not know whether it is a flow dependence, anti-dependence or output dependence.

(Refer Slide Time: 30:49)

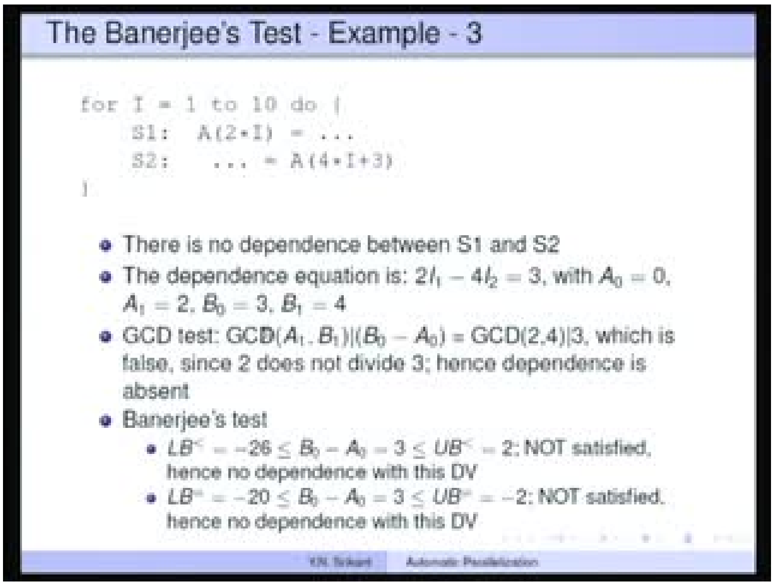


That is look at the next equation this was very interesting because it had many direction vectors: equal to, less than and another less than and there is the anti-dependence as well.

So, first one is from S1 to S2 second one is from S2 to S1. The dependence equation is again I 1 plus I 2 equal to 10, no problem with this and then the GCD test said there is dependence so let us see what the Banerjee’s test says. I did not right down LB and UB for star and greater than because we know it is not useful, star will always be true if there is the dependence and greater than always be the false for single loops. So, LB less than is 3, B naught minus A naught is 10 and UB less than is 17 so, 3 less than equal to 10, less than equal to 17 is satisfied. So, there is defiantly a dependence with the direction vector less than so that is true S1 to S2 also there is less than direction vector, S2 to S1 also there is less than direction vector so, this is correct.

We do not know which one of which one is which are both are true, but we do not know whether both are true are one of them is true. For equal to it is 2 less than equal to 10, less than equal to 18, again this is satisfied so the equal to direction vector also exists that is also correct because S1 with delta equal to S2 holds so, the dependence actually the pointed out by the GCD test is indeed correct here.

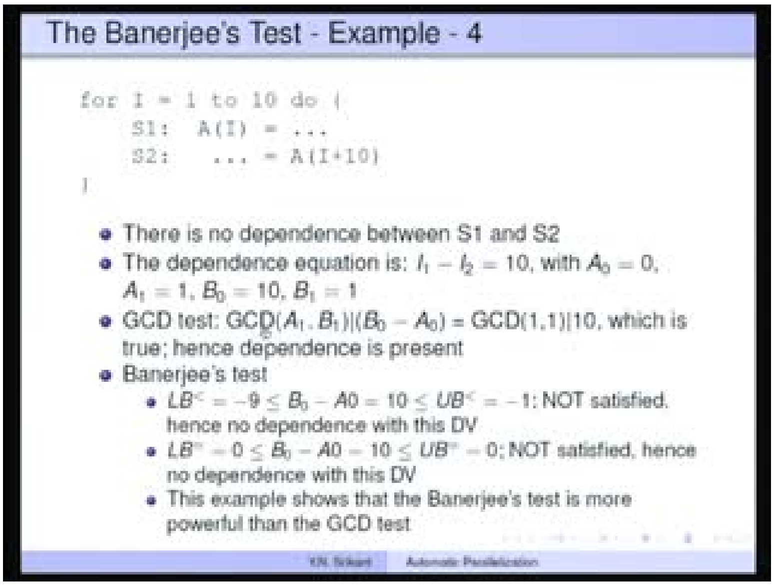
(Refer Slide Time: 32:32)



Third one GCD test there is no dependence so, Banerjee's test obviously should say the there is no the dependence so, again the dependence equation is 2I 1 minus 4I 2 equal 3. The GCD test there is no dependence. And the Banerjee's test also says LB less than is minus 26, B naught minus A naught is 3 and UB less than is 2.

So, 2 greater than equal to 3 is obviously not possible so, this is not satisfied. There can be no dependence with direction vector less than with equal to this is minus 20 less than equal to 3, less than equal to minus 2 is obviously not satisfied, again there can be no dependence with equal to as well as, star and greater than are not useful as i told you earlier.

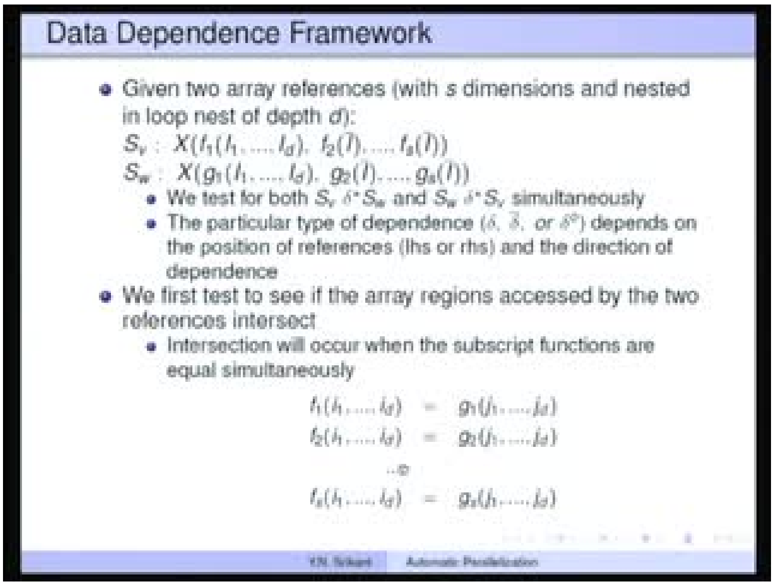
(Refer Slide Time: 33:25)



Now, this really shows the power of Banerjee's test. So, A I equal to A I plus 10 so there is really no dependence between S1 and S2. The dependence equation is I 1 minus I 2 equal to 10, the GCD test said GCD of 1 coma 1 divides 10 so there is dependence.

So, let us see whether Banerjee's test also says there is dependence or it says there is no dependence. So, we compute LB less than that gives as minus 9, B naught minus A naught is 10 so, minus 9 less than or equal to 10, less than or equal to minus 1 obviously not satisfied. So, there can be no dependence with the less than direction vector with equal to it is 0, less than or equal to 10, less than or equal to 0 again so, obviously not satisfied. Hence, there can be no dependence with equal to less than either, star and greater than or not useful as before. So, that means Banerjee's test says there is no dependence possible. The GCD test said there is dependence. Obviously, GCD test is less power full of Banerjee's test and the dependence, which really does not exist because of the loop bond being 1 to 10 is indicated here. So, if the loop bond increases to say 100 or something like that this equation will indeed show that there is dependence so, that is from Banerjee's test as well.

(Refer Slide Time: 35:04)



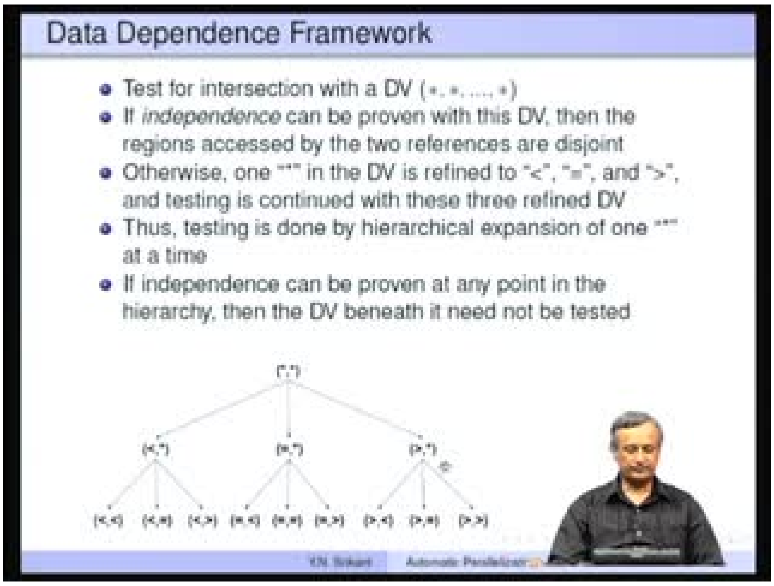
So far we saw how to apply the GCD test and the Banerjee's test independently. We really did not see how to apply it when there is a nest of loops more than one loop nest and then there may be many combinations and then you know there are also subscripts possible for arrays, many their array we have seen so far had just 1 subscript 1 dimensional array. But there could be multi-dimensional arrays; there could be multiple loops so the number of possibilities really becomes very large so how do we do that. We really do not know.

The hierarchical data dependence test gives you a way of determining the dependence, when there are multiple subscripts in an array expression. So for example: S v and S w are nested in a loop nest of depth d, the two arrays are obviously X, we did not write the complete statement we just wrote the references. First subscript is f 1, f 1 of I 1 to I d; second subscript is f 2 of I bar I bar stands for I 1 to I d; third subscript is f 3 similarly, the S th subscript is f s so, there are s th subscripts s dimensional array. Similarly, S w has g 1, g 2, g 3 etcetera g s.

Now, how do we go about testing the whether there is dependence between these two statements or not. Obviously, if we test the subscripts separately that is f 1 and g 1, f 2 and g 2, f 3 and g 3 f s and g s and check whether there is intersection or not, we get some answers. How do we combine these answers and check whether there is [depensense/dependence] at the complete array level that is what we want to know. The method is simple we test for both S v delta star S w and S w delta star S v simultaneously, we that is the first statement how to do that we will see.

So, the particular type of dependence delta, delta bar, delta naught depends on the position of the references and the direction of dependence. How do we use this we will see a little later. We first test to see if the array regions accessed by the two references intersect. So obviously, the subscript functions are equal simultaneously so that means f 1 of I 1 to I d must be equal to g 1 of i 1 to i d, at the same time f 2 of i 1 to i d must be equal to g 1 of j 1 to j d for the same values. All of them must be equal f 1 equal g 1, f 2 g 2, f s equal to g s for the same values of g 1 and g d. If we knew the values of i 1 to i d and j 1 to j d all this was very easy to find out, but we do not. We only have test which check whether there is a dependence or non-dependence between two reference equations.

(Refer Slide Time: 38:30)

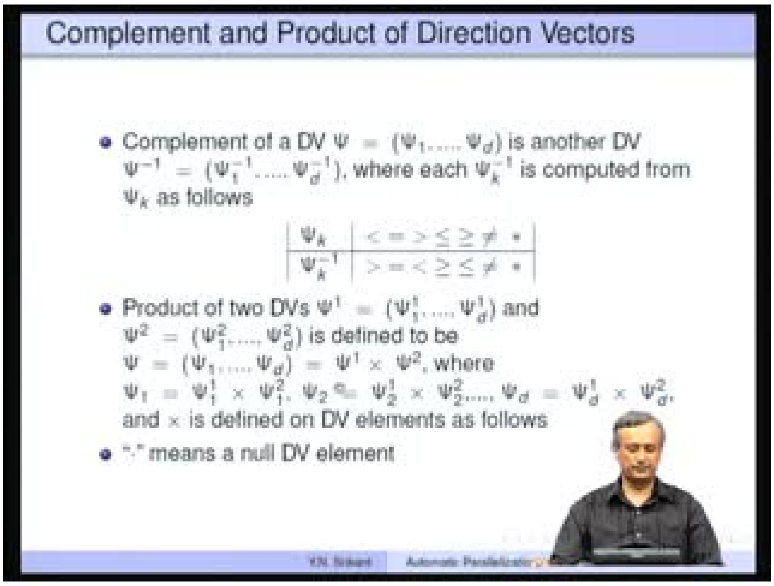


So, we start off with a direction vector star comma star comma star and test for the dependence using either the GCD test or the Banerjee's test. If independence can be proven at this level, then the regions accessed by the two references are obviously disjoint so the GCD test says independence with this direction vector so, that is the one we already studied. What it really tests is if you say GCD of a 1 comma b 1 comma etcetera divides p naught minus a naught that test is for star comma star. If it is for Banerjee's test we know how to compute LB star UB star and so on for each one of the direction for the loop variables. So, then the regions accessed by the two references are disjoint. If this direction vector test gives you independence the problem is solved.

Suppose the test says there is dependence. Then one star in the direction vector is expanded to less than equal to and greater than and the testing is continued with these three refined direction vectors.

So, this is the hierarchical expansion that is done one at a time. Start with star comma star test at this level using say either the GCD test or the Banerjee’s test if we find that there is dependence then we expand the left star less than comma star is tested so, for this we cannot apply GCD test we will have to apply only Banerjee’s test. So, equal to comma star and greater than comma star. If we find that there is no dependence at any one of these stages for example, this is not a feasible direction vector first component cannot be greater than in a direction vector only the second onwards it can be greater than provided the first component is a less than. So, there is no need to really expand this tree and test for more dependence here, we can simply say this is not possible proven this tree at this point. But out of these two if one of them says there is no dependence the test says no dependence that tree can need not be processed further, but suppose both of them say dependency is present then we need to actually expand further. This becomes less than, less than, less than equal to less than, greater than, this becomes first component is equal second is less than equal to or greater than. This is again not possible we know that, but this is possible all these - five are possible. So, we need to check for dependences with these direction vectors one at a time using the Banerjee’s test and then see which one returns true and which one returns false. Now so whatever is returned as true remains and whatever is returned as false just goes away so we collect only those dependences which are written as true. So we did this for each subscript.

(Refer Slide Time: 41:55)

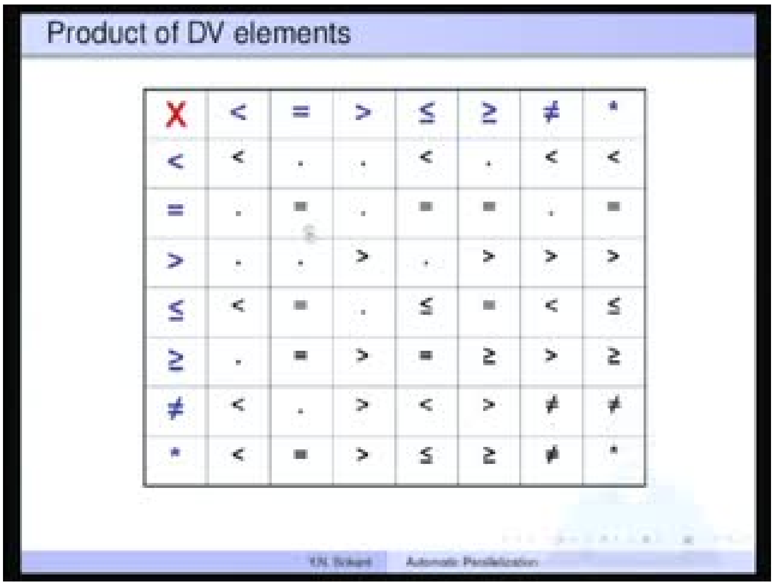


Now, we also need to understand how to combine the direction vectors of various subscripts and get one direction vector product so, that requires the definition of what is known as a product of direction vectors. If we know that there is no dependence from S1 to S2 for some reasons somehow we know that. Then we need to check whether there is dependence from S2 to S1, to do that we need to computed suppose we have already computed the direction vector from S1 to S2 and we want to compute now the direction vector from S2 to S1 that requires complement operation on the direction vector. So, the we do not have to compute the new direction vector all the way starting with the Banerjee's test. So, what is the complement of a direction vector?

Psi is psi 1 to psi d and psi inverse is psi 1 inverse etcetera psi d inverse. How do we compute it? If psi k is less than, psi k inverse is greater than so, if there is a dependence from S1 to S2 with a less than then in the complement it will become greater than; if it is equal to it remains as equal to greater than becomes less than, less than or equal to becomes greater than or equal to, greater than or equal to becomes less than or equal to, not equal to remains not equal to and star remains star so, this is how we compute the inverse. So this will become clear a little later.

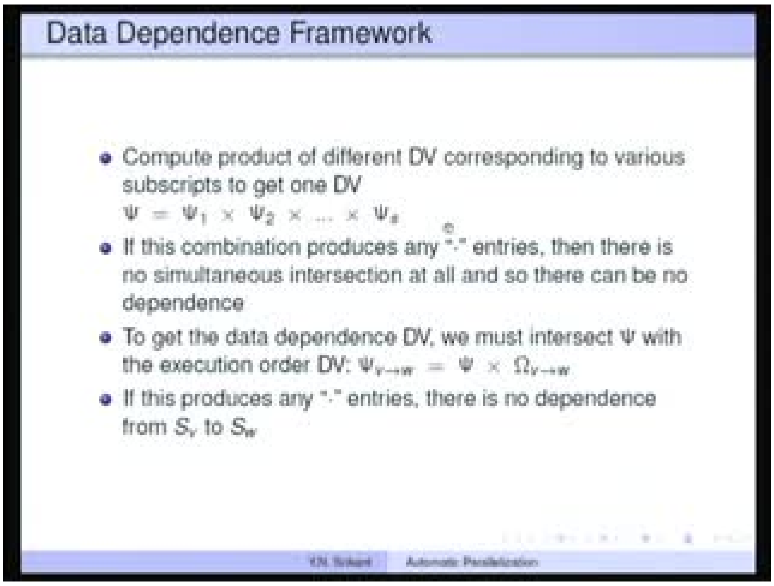
What about the product of two direction vectors? So, if we have a two dimensional array and we compute the direction vectors for each of the subscripts separately, how do we compute the direction vector of the entire reference we need to take a product of these two DV s. So, let us say psi 1 is psi 1 1 to psi 1 d; psi 2 is psi 2 1 to psi 2 d; so, psi is psi 1 to psi d which is defined as psi 1 cross psi 2. So, you just take the cross product of the individuals, how to do we will it is in the next slide. So, psi 1 1 cross psi 2 1, psi 2 1 to psi 2 2 etcetera psi d equal to psi 1 d cross psi 2 d and the cross operator is defined later.

(Refer Slide Time: 44:47)



So, in the next in this slide whenever we have a dot that means it is a null direction element it is not possible that is why it is null. And whenever we reach a null that means there is no dependence. So, one of the subscripts has a less than direction vector the other subscript returns a less than direction vector, then the product is also less than so, imagine if you have a two dimensional array one of the dependence in one of the dimensions is less than, the other is also less than, then it makes sense to have the entire dependence as the direction as less than. But suppose one of the direction the direction vectors says it is less than that is you compute in iteration i 1 and use in iteration i 1 plus k. The other subscript says equal to that means i compute in i 1 and use in i 1 so obviously, there is a difference of opinion between these two direction vectors and when we take a product it would yield a null that means no dependences possible. Similarly, less than and greater than would give a null, less than and less than or equal to will give a less than, less than and greater than or equal to will give a null etcetera. With not equal to and star you just get less than. So, that is how the direction vectors are really computed with equal to we have just one with another subscript also being equal to, we get equal to otherwise, less than or equal to we get equal to greater than or equal to, we get equal to not equal to will give a dot and star gives a an equal to. So, this is how we compute the product of direction vector.

(Refer Slide Time: 46:47)



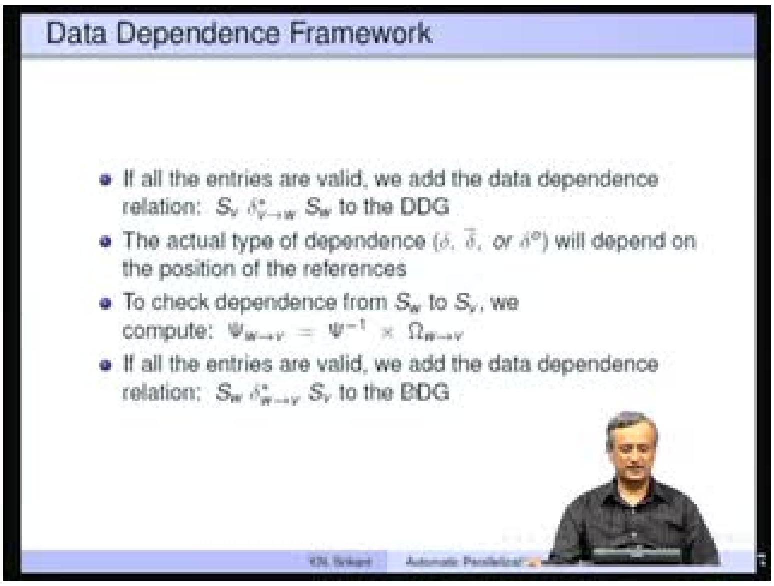
What is the data dependence direction framework? The product of direction vectors corresponding to the various subscripts is obtained and then we get one direction vector so that is the next step we have the direction vectors for the various subscripts so, now we have a single direction vector. So, we define the direction vector for product direction vector for a psi 1 cross psi 2, but it is easy to see that we can do this and then another cross and third cross etcetera and get psi.

Suppose, this combination produce any dot entries, then that means in this direction vector there we have less than comma greater than comma equal to comma dot. Then there is no simultaneous intersection between these direction vectors; one of them is a dot so it is not possible direction vector and there can be no dependence.

But suppose, all of them yield correct direction vectors then to get the exact data dependence itself direction vector. We must intersect psi with the execution order DV so, we are not yet done. The execution order DV will tell us whether there can be dependence from S1 to S2 or from S2 to S1. So, we already saw how to compute the execution order direction vector using the syntax of the constructs so, that we do psi of v to w will be psi cross omega v to w so, if this product produces any dot entries then there is no dependence from s v to s w. So, this the execution order direction vector from one

S1 to S2 let us say s v to s w. That yielded us some dot entries so, there can be no intersection.

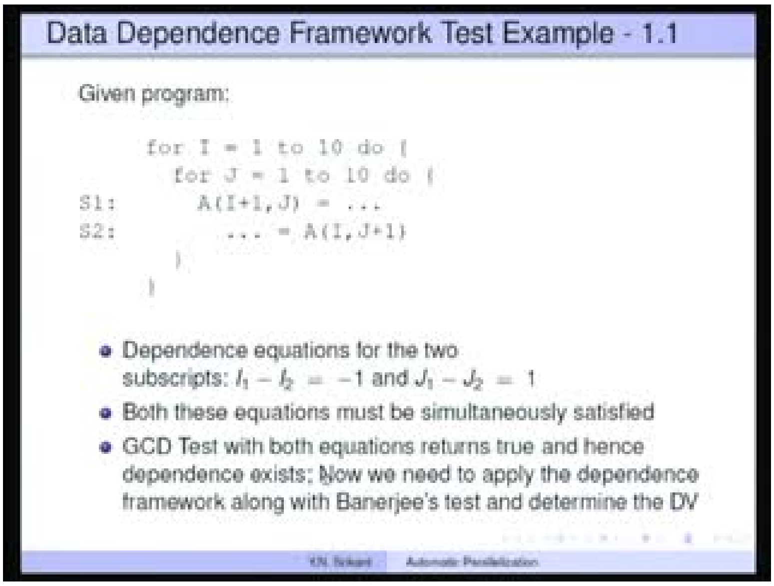
(Refer Slide Time: 49:05)



But, if all the entries are valid we add the dependence S v delta star v to w. S w to the data dependence graph. Now even now we are not yet done we still have a star here, the actual type of dependence whether it is delta or delta o or delta bar will depend on the position of the references. We know that there is at least one component S v to S w, S v first and then S w is decided, but the type of delta we decide based on the position. So, if S v is on the left side and S w is on the right side of the assignment obviously, it is a flow. If S v is on the right side and S w is on the left side then it is an anti if both are on the left side it is output dependence.

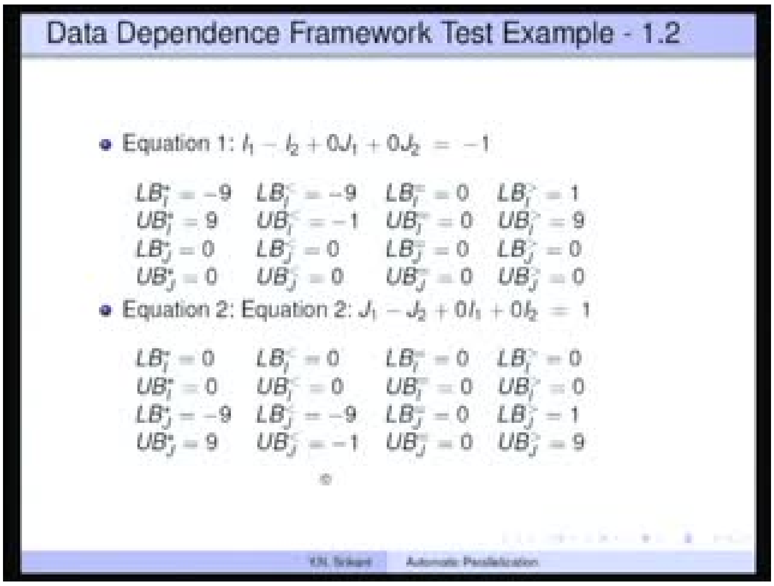
Now, (Refer Slide Time: 50:00) if we got this dot and there was no dependence from S v to S w (Refer Slide Time: 50:05) then we need to check whether there is any dependence from S w to S v. So, how to do that? Now we need to compute the direction vector of the dependence from w to v so, this is where the inverse comes into picture. So, we take the inverse of the original and then multiply it with omega of w to v. So, this is the direction vector from w to v, this is the inverse of the original directional vector so, now we are computing the dependences from S w to S v. So, if this product gives valid entries there is no dots. Then there is dependence from S w to S v with a delta star and again we use the position in order to determine whether it is a delta o or delta or, delta bar. So, this is how the hierarchical dependence framework is used.

(Refer Slide Time: 51:02)



So, let us look at an example; here is a nested program I equal to 1 to 10 do, J equal to 1 to 10 do. So, there are two subscripts for this array A I plus 1 comma J and this side is A I comma J plus 1 very simple reference. The first dimension uses only I, second dimension uses only J. The next example we see we will combine both of them. So, the dependence equations for the two subscripts: I 1 plus 1 equal to I 2 is I 1 minus I 2 equal to minus 1, J equal to J plus 1 becomes J 1 minus J 2 equal to 1. Both these must be simultaneously satisfied in order for the dependence to exist. So, GCD test with both equations returns true so, these things we have seen before simple equations and hence dependence exist. So, now we need to apply the dependence framework along with Banerjee’s test and determine the direction vector.

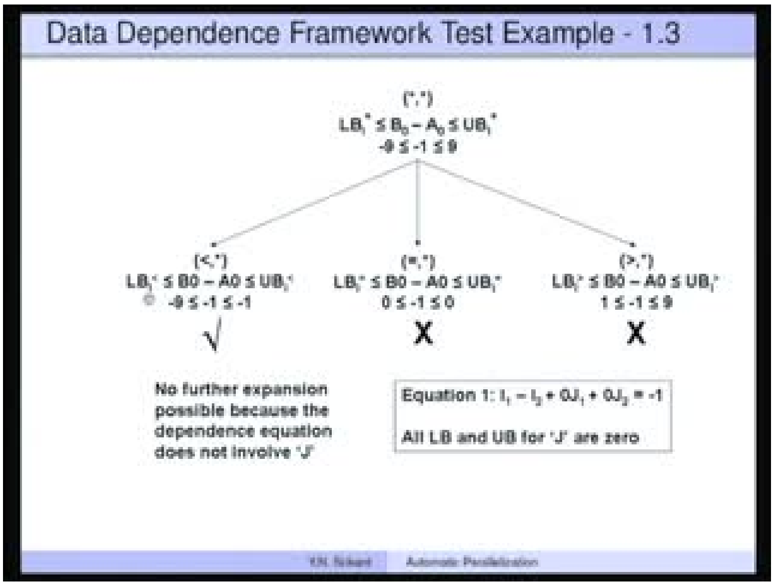
(Refer Slide Time: 52:05)



So, we have equation 1: I 1 minus I 2 plus 0J 1 plus 0J 2 equal to minus I 1. We just wrote 0 here to show that this equation we have taken care of the J as well, but obviously, the direction vector components the rather the limits LB j etcetera turn out to be 0. So, we compute the values of LB I, UB I, LB with star less than equal to and greater than. Similarly, LB j, UB j etcetera for star, less than, equal to and greater than. For j all of them become 0, for i they give the right values they are all computed using the coefficients of this particular equation.

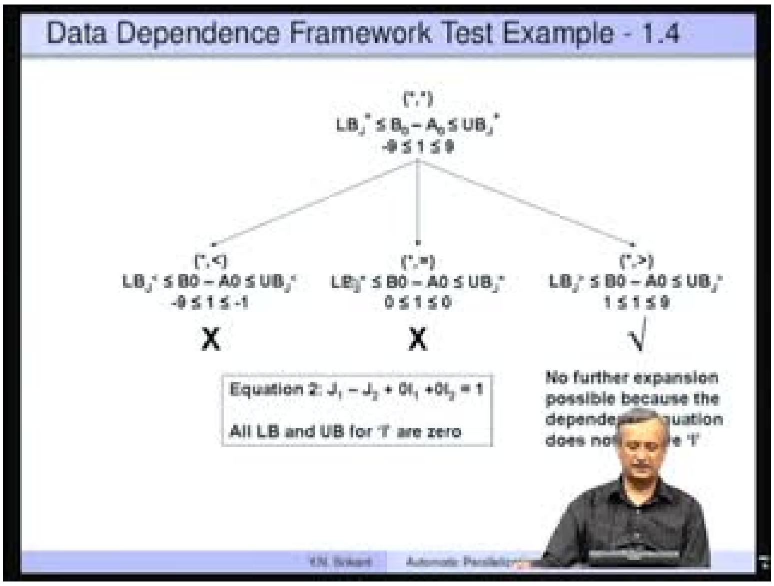
The same is true for equation 2: J 1 minus J 2 equal to plus 0I 1 plus 0I 2 equal to 1. So here the i components are 0 because we do not have any I index expressions in the second dimension of the array subscript. So, here the i values are all 0, but the J values will be ones computed from the equation.

(Refer Slide Time: 53:21)



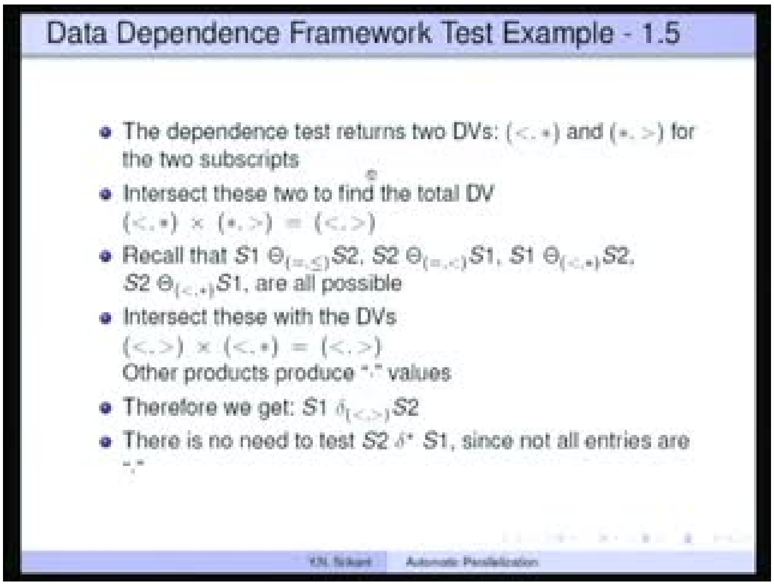
Now you if take LB i star plus LB j star so obviously, LB j star is 0 so, you get only LB i star. Check whether less than equal to B naught A naught, less than equal to UB i star so the inequality is written here, minus 9 less than or equal to minus 1 less than equal to 9. So, with any dependence if there is some dependence then this will be true so this is true. We try to expand the first one only because we are now looking at equation 1 which has only I, the further expansion with we do not have to do with J because it is all 0 coefficient in the dependence equation. All LB and UB for j are 0 we are not going to expand the second component for j. In this case again we check LB i and with less than, less than or equal to B naught minus A naught UB i with less than. So, the inequality will be minus 9 less than or equal to minus 1 this is correct holds. Whereas, for equal to and star we have LB i equal to less than equal to B naught minus A naught, less than equal to UB i equal to. So, LB i equal to plus LB i star gives us only LB i star because the second component all values are 0 so, this inequality is not satisfied 0 less than equal to minus 1, less than equal to 0. The third inequality with respect to greater than which will obviously, not be satisfied because the first component is greater than. 1 less than or equal to minus 1, less than or equal to 9 this is not satisfied. So, this is so far as the first equation goes.

(Refer Slide Time: 55:09)

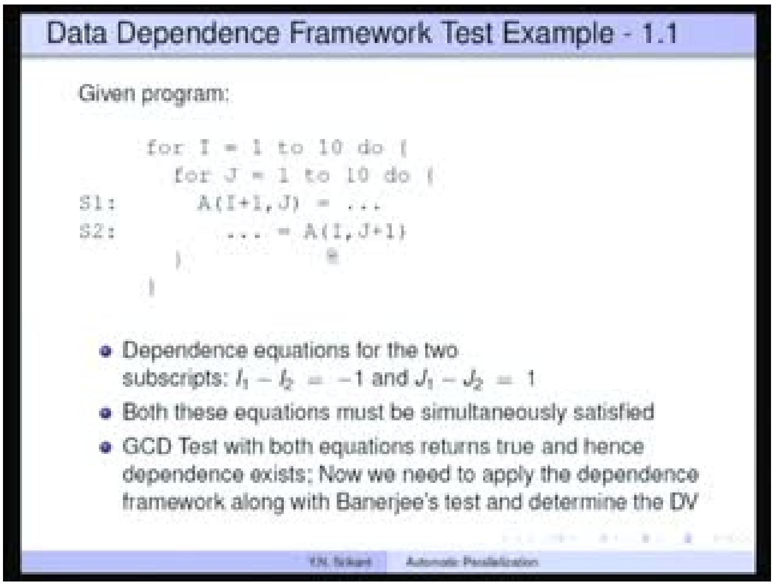


Now, apply for the second equation. Similarly, this is not satisfied, in this (Refer Slide Time: 55:16) only one direction vector less than comma star is correct. In this only star comma greater than is correct the first star because it involves I all LB and UB for i are 0 so no expansion on the first component is possible. So, this holds star comma greater than holds. So, now we have two direction vectors first one is less than comma star, second one is star comma greater than.

(Refer Slide Time: 55:46)

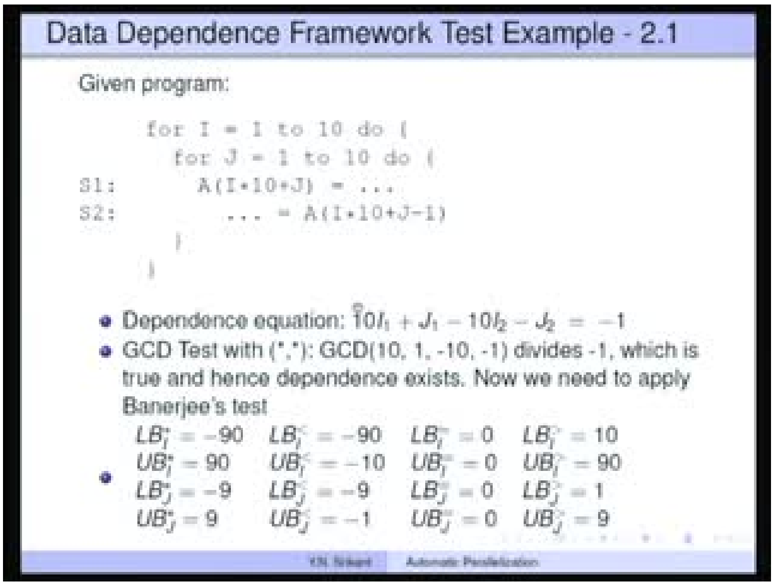


(Refer Slide Time: 56:52)



Now, the dependence test returns less than star and star greater than intersect these two so, you get less than star cross star greater than equal to less than greater than no null entries. So recall the direction vector for the execution order dependence. So, you have S1 theta equal to less than or equal to S2 etcetera. Intersect reach of these with less than greater than so, you get less than greater than cross less than star is less than greater than, all other products give you dot values so they are not possible. Therefore we get S first one this is from S1 to S2 so, we get S1 delta less than equal to S2. This did not give if this was also not possible, let us say even this entry was dot then we would have tested for S2 delta star S1, but now there is no need to test S2 to S1 because S1 to S2 the delta holds. Why is it delta that is because the references if you look at it S1 is on the left side and S2 is on the right side so, there is a flow dependence. That is why this particular dependence (Refer Slide Time: 56:58) is from S1 to S2 with a delta and the direction vector is given as less than and greater than so, this example shows that there are two subscripts, we need to reduce it to a single direction vector check whether there is dependence and finally, check whether the dependence is from S1 to S2 or S2 to S1.

(Refer Slide Time: 57:22)



So, this is the end of this lecture and in the next lecture we are going to look at one more example, a more complicated one and so good bye for today.