Discrete Mathematical Structures

Dr. Kamala Krithivasan
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture # 9

Proving Programs Correct

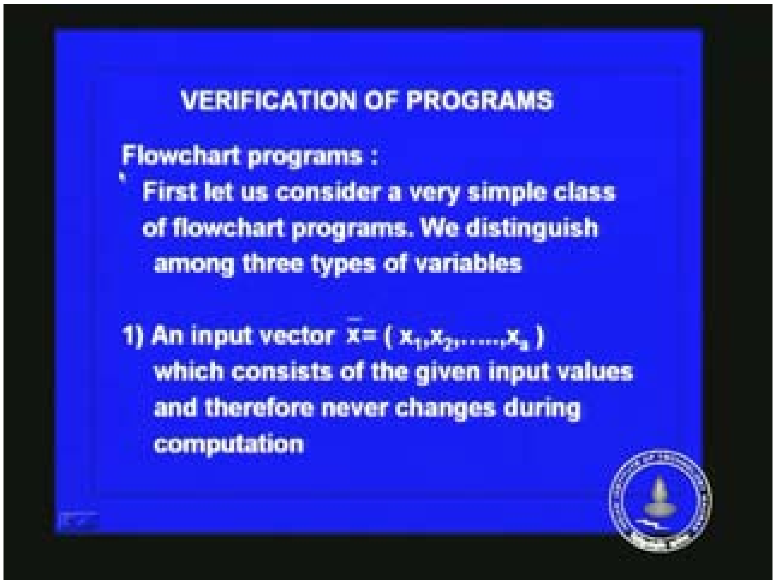
We have studied about propositional logic and predicate logic. We have also studied about resolution principle and we have seen the use of resolution principle in prolog and how logic is used in prolog. Today we shall see how predicate calculus or predicate logic is made use of in verification of programs.

Now, when you write a program you do not know whether it is correct or not, you will test which some test data. But for the given test data if the program works correctly or it gives you the correct output you cannot say that it is always correct because some other input may give a wrong value.

So testing only points out, if there are errors and even if it runs correctly it does not mean that there are no errors. You could have errors and for some other input it may not work properly. It does not tell you the absence of errors. If there are errors present it will point out but it does not point to the absence of errors. So when you write a program it is better to prove that it works correctly. And how you are going to do it with predicate logic is what we are going to see today.

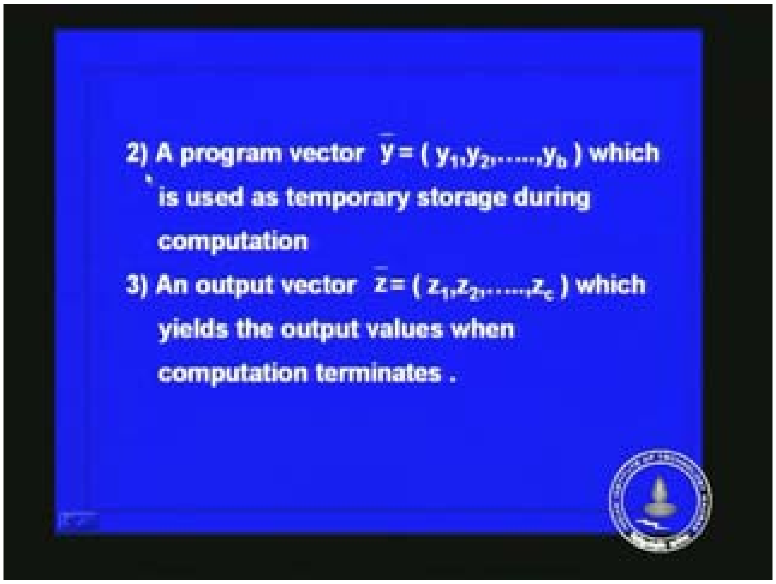
But again I want to tell you that we are going to explain this principle with just a simple example. For very large programs this sort of a method may not work because it is too much involved. For small programs you can use this method but for very large programs testing is the only possibility. And if you test with proper data you have to assume that the program works correctly. Now for simplicity sake I shall take flowchart programs so it is a very old concept, I will take flowcharts. So what we are going to see is verification of programs today and we are going to consider flowchart programs.

(Refer Slide Time: 3:30)



Now let us consider a very simple class of flowchart programs. We distinguish among three types of variables. The program has to read something and they are the input variables and during the execution of the program it will use some other variables they are called program variables. And it will output something and that is called output variable. So you have some input variables say (x_1, x_2, x_a) an input vector which consists of a given input values and therefore it never changes during the computation. These are the inputs. A program vector y bar which consists of variables (y_1, y_2, y_b) are called program variables which is used as temporary storage during the computation.

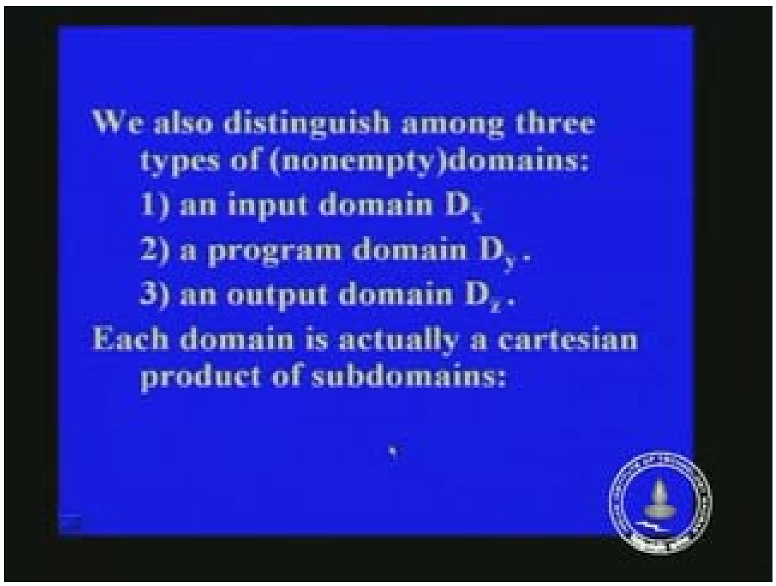
(Refer Slide Time: 4:16)



And then you have some outputs (z_1, z_2, z_c) and output vector z bar consisting of (z_1, z_2, z_c) which yields the output values when computation terminates.

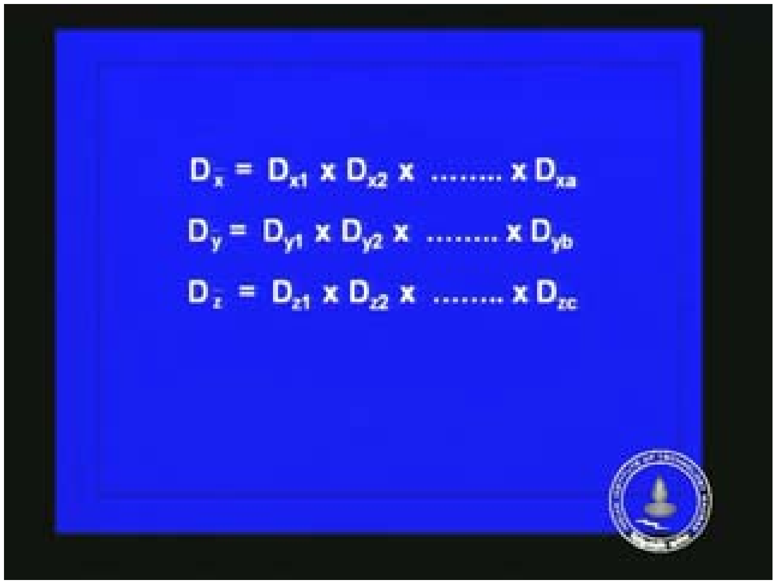
Now the inputs are defined over input domains. We have to specify the input domains. Similarly the program variables each one will have a domain on which it is defined and then there is an output domain on which the output variables are domained. We also distinguish among three types of nonempty domains; an input domain D_x bar, a program domain D_y bar and output domain D_z bar. Each domain is actually a Cartesian product of subdomains.

(Refer Slide Time: 4:48)



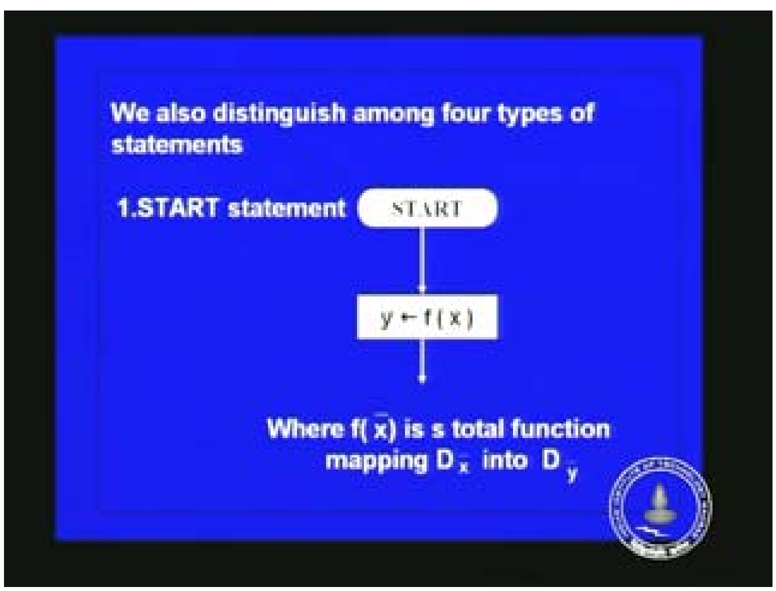
You have input variables (x_1, x_2, x_a) so each one will have a domain D_{x1}, D_{x2} like that D_{xa}. The Cartesian product of that is defined as D_x bar the input domain. Similarly, the program variables are (y_1, y_2, y_b) each one will have a domain D_{y1} for y_1, D_{y2} for y_2 and D_{yb} for y_b. The Cartesian product of that is defined as D_y bar the program domain. And (z_1, z_2, z_c) are the output variables so each one is defined over a domain. The Cartesian product of that is defined as D_z bar the output domain. So you have three domains; input domain, program domain and output domain.

(Refer Slide Time: 5:40)



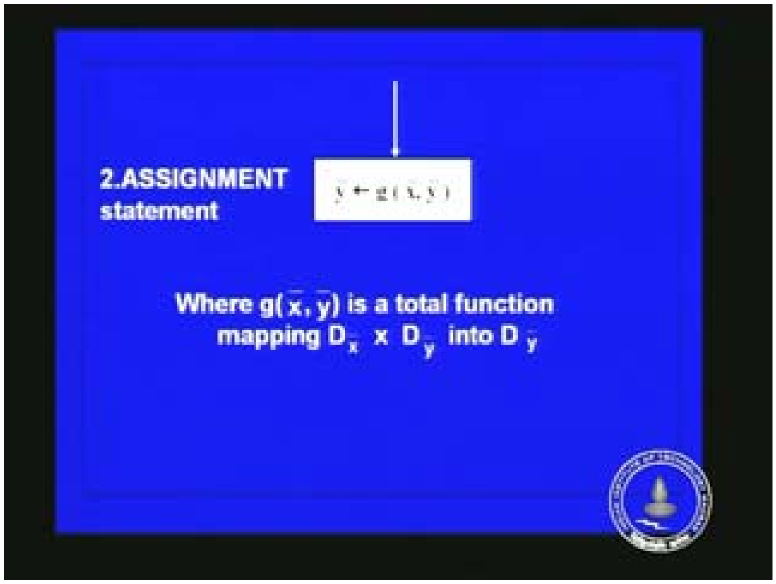
Now, we shall consider simple program statements. We distinguish between four types of statements; the start statement which is start and then the y bar is assigned f(x) bar that is the program variables are assigned some values from the input variables where f(x) bar is a total function mapping D_x bar into D_y bar this is the technical way. So initially you start and you have a small assignment statement where y bar is assigned some values depending upon the x bar or the input variables.

(Refer Slide Time: 6:37)



Then you have assignment statement; it is y bar is g(x) bar y bar. That is g(x) bar y bar is a total function mapping D_x bar into D_y bar into D_y bar.

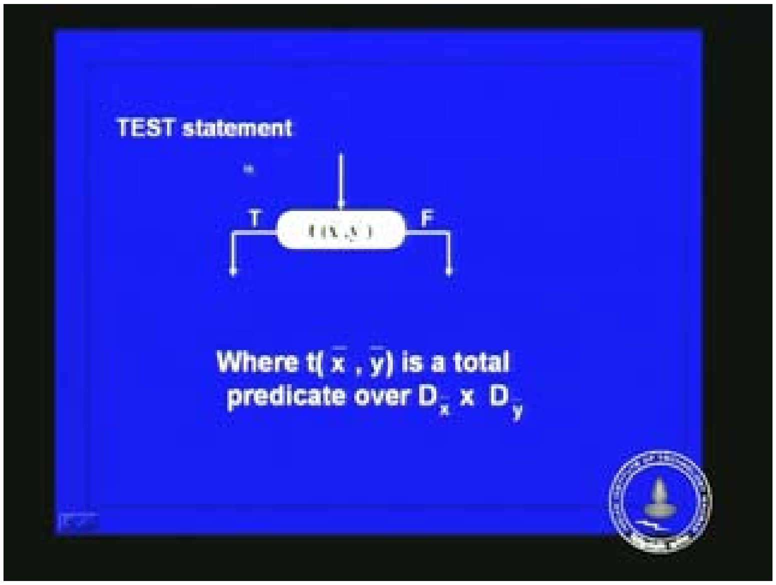
(Refer Slide Time: 7:21)



So depending when the program control reaches this point and depending upon the values of x bar which has never changed and the current value of y bar that is the values of the program variable (y_1, y_2, y_b) new values for the program variables (y_1, y_2, y_3 up to y_b) are assigned and this is a total function. And you have test statements you have t(x bar, y bar) where t(x bar, y bar) this is the same as this t it is a total predicate over D_x bar into D_y bar.

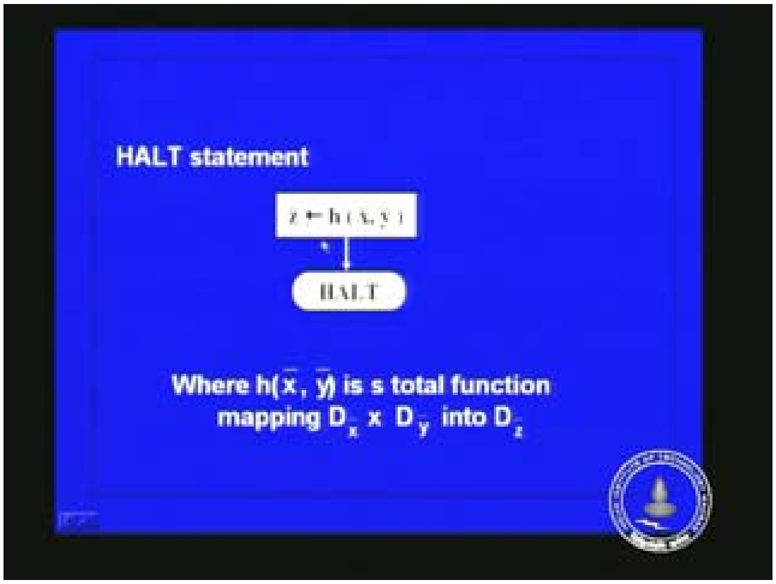
There are two outlets. Depending upon when the control reaches at this point depending upon the value you give for x bar and y bar or the value of x bar is unchanged and when the control reaches at this point y bar has a particular value depending upon that this predicate is evaluated and it will take the value true or false. There are only two exits there is no exit here. This true exit exists like this, if the predicate is true you take this exit and if the predicate is false you take this exit.

(Refer Slide Time: 8:11)



Then you have the HALT statement finally after the computation is finished you arrive at this point. Then the output variables are assigned some particular values from the input and the program variables. It is h(x bar, y bar) where h(x bar, y bar) is a total function mapping from D_x bar into D_y bar into D_z bar and then you HALT.

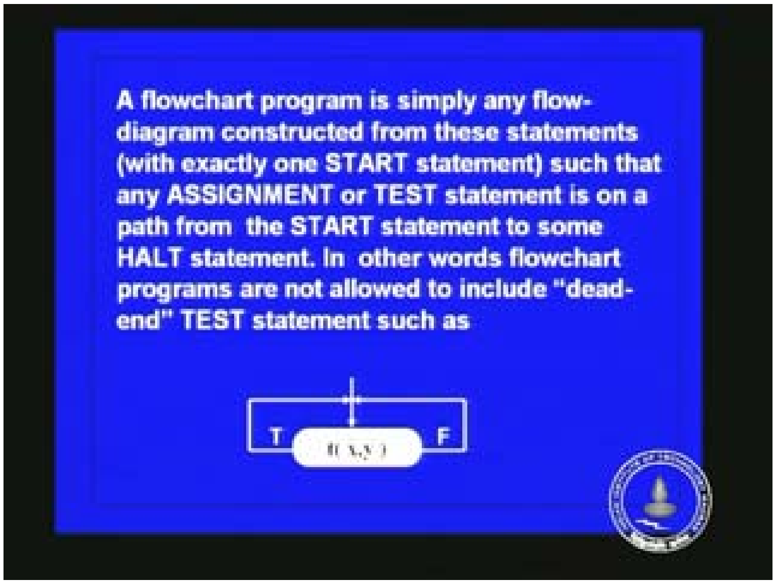
(Refer Slide Time: 9:12)



A flowchart program is simply any flowchart diagram constructed from these statements which exactly one START statement such that ASSIGNMENT or TEST statement is on a path from the START statement to some HALT statement.

You may have more than one HALT statement but usually there is only one START statement. In other words flowchart programs are not allowed to include “dead-end” TEST statements such as this. That is, after performing this test again you go here and keep on performing. You are not supposed to gain into a loop like this.

(Refer Slide Time: 9:36)

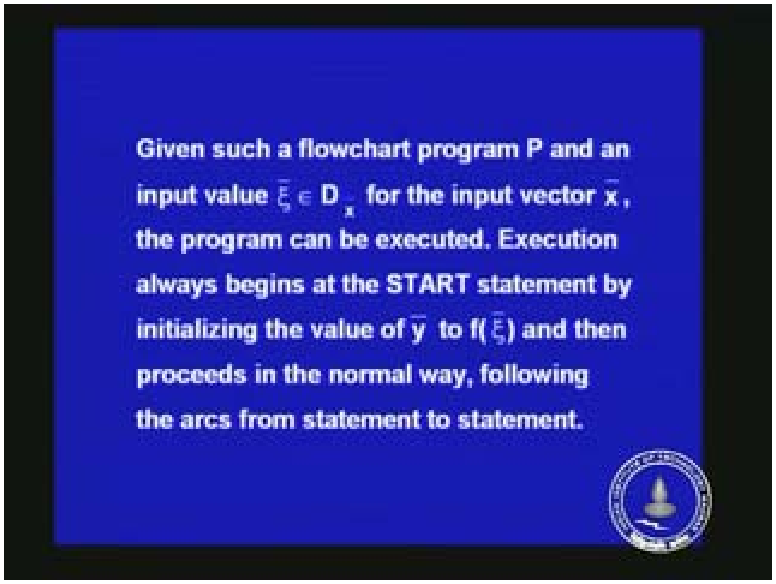


So a flowchart program consists of START statements, ASSIGNMENT statements, TEST statements and HALT statements. And every ASSIGNMENT or TEST statement occurs in a path from a START statement to a HALT statement and you do not have dead-end loops like this. You know what it is; you must have been very familiar with what is a mean by a flowchart and so on. But still I am repeating all this things which you are very familiar with.

Now how do we make use of predicate logic to prove that a program which is given in the form of a flowchart is correct it does what you indent it to do? Now, given such a flowchart program P and an input value si bar belonging to D_x bar. So you are taking a particular input si bar for the input vector x bar the program can be executed after taking the input si bar the program is executed.

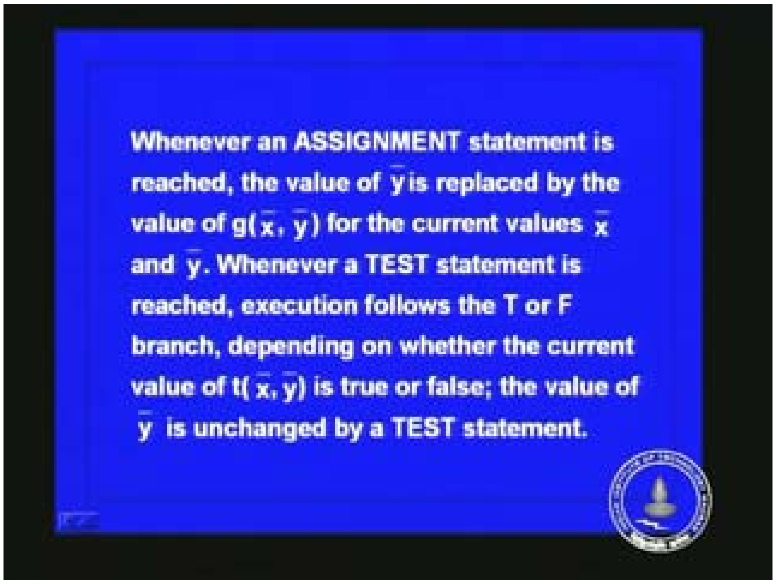
Execution always begins at the START statement by initializing the value of y bar to f(si) bar because of the first statement if you look back is y bar is assigned f(x) bar. By initializing the value of y bar to f(si) bar it proceeds in the normal way following the arcs from statement to statement.

(Refer Slide Time: 11:12)



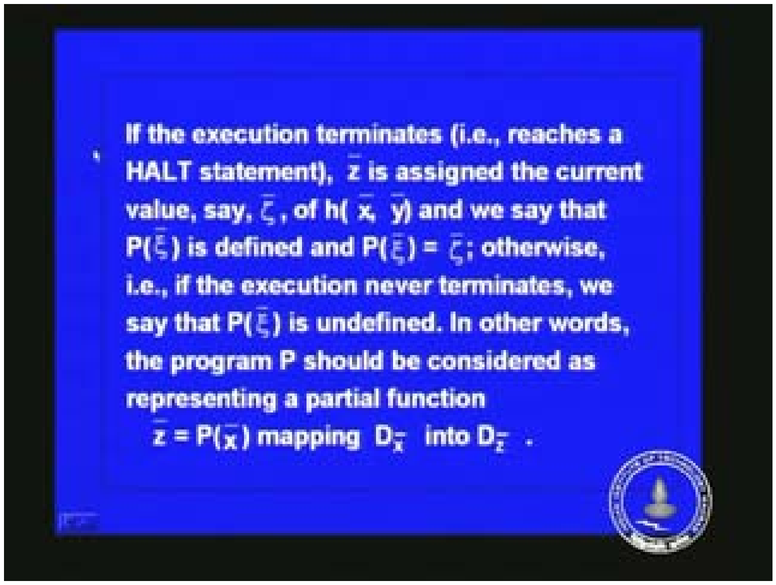
Whenever an ASSIGNMENT statement is reached the value of y bar is replaced by the value of g(x bar, y bar) for the current values of x bar and y bar this is what I mentioned to you. Whenever a TEST statement is reached execution follows the true path or the false path, true branch or the false branch depending on whether the current value of t(x bar, y bar) is true or false. The value of y bar is unchanged by a TEST statement.

(Refer Slide Time: 12:12)



If the execution terminates that is if it reaches a HALT statement z bar is assigned the current value say zeta bar zeta bar of h(x bar, y bar). When it reaches a particular point x bar is never changed y bar will have a particular value and the control reaches that point then h(x bar, y bar) will give you some particular value that is assigned to zeta bar. We say that P(si) bar is defined and P(si) bar is equal to zeta bar. Otherwise this is the output andthis is the input. We can even say like this; if the execution never terminates we say that P(si) bar is undefined. In other words, the program P should be considered as representing a partial function z bar equals P_x bar mapping D_x bar into D_y bar.

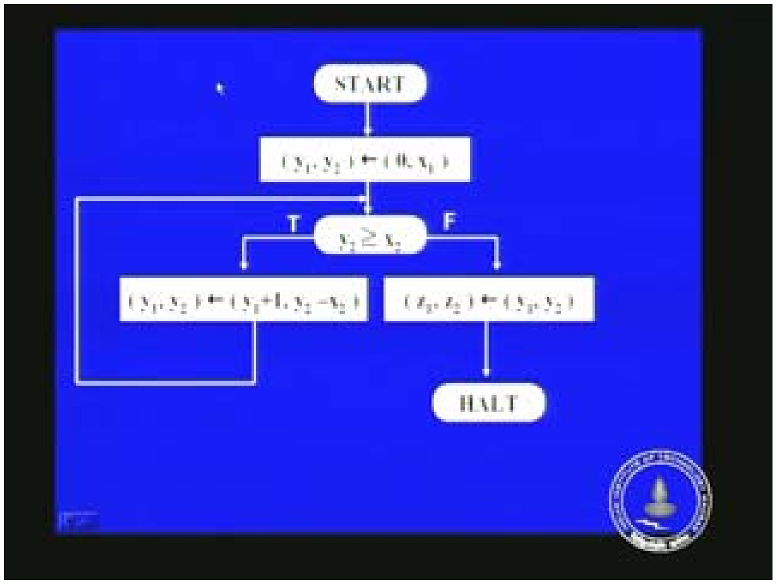
(Refer Slide Time: 12:45)



That is given a particular input you get a particular output. So the input should belong to the particular input domain and the output should belong to the particular domain and they are related by this P(x) bar is equal to z bar. So if you take a particular input si bar you get a particular output zeta bar and they are related like this. si bar belongs to x bar the input domain and then zeta bar is the output and it belongs to the output domain D_x bar into D_z bar this is D_z bar.

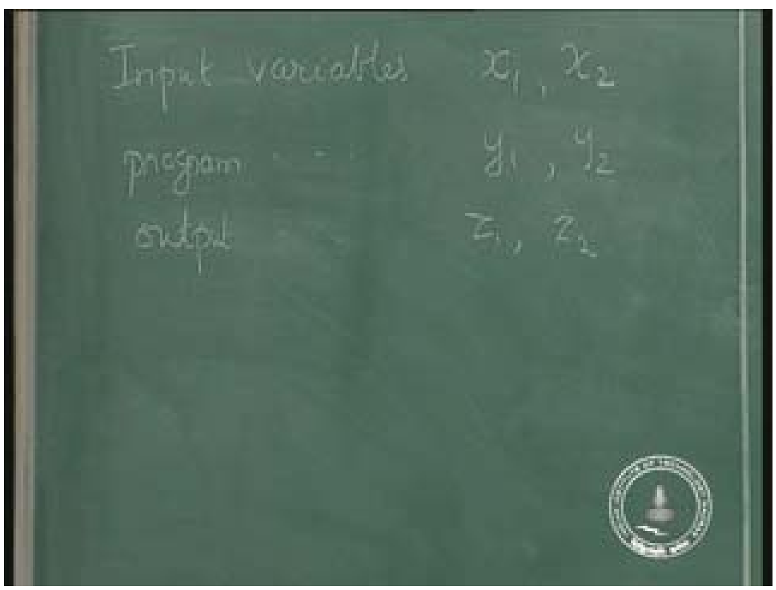
Let us take a particular example and see what happens. I will come back to this slide later. Look at this, this is a small flowchart program and what does this do.

(Refer Slide Time: 14:55)



START and you are having two input variables x_1 and x_2 and two program variables y_1 and y_2 and two output variables z_1 and z_2. So here there are two input variables x_1 x_2, program variables are y_1 y_2 and output variables are z_1 and z_2.

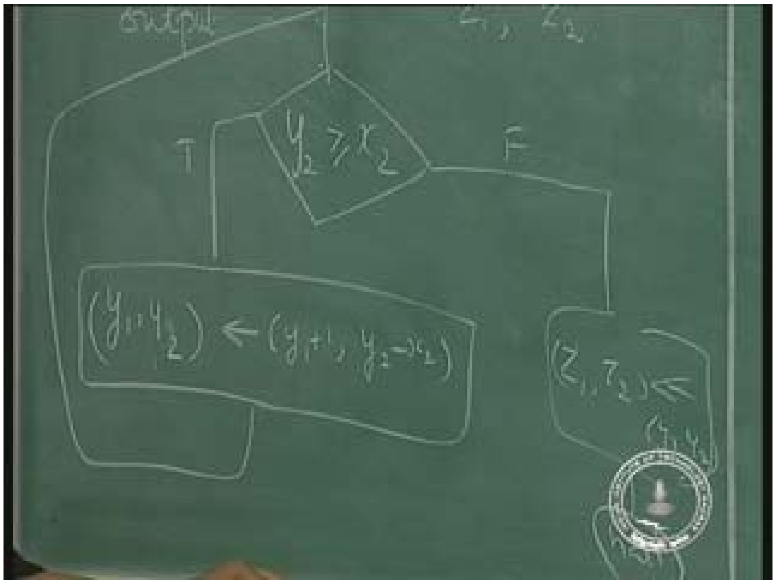
(Refer Slide Time: 15:50)



So initially when you start y_1 is assigned the value 0 and y_2 is assigned the value x_1. And next you go to the control statement y_2 greater than OR is equal to x_2 and if it is true you take this path and y_1 will be assigned y_1 plus 1 and y_2 will be assigned y_2 minus x_2 and you go here. If this is false you take this path, in that case outputs z_1 and z_2 are assigned the values of y_1 and y_2 and you get the HALT statement.

So the test statement is like this here; y_2 greater than OR is equal to x_2 and if it is true the assignment statement y_1 y_2 is assigned y_1 plus 1 then y_2 minus x_2 and then you go back here.

(Refer Slide Time: 17:22)



If it is false z_1 z_2 will be assigned the value y_1 y_2 and then to HALT. Now let us execute this program for a small number then we will know what it is. Take the value of x_1 is 16 and x_2 is 5. So what is the value of y_1 and y_2? The first time you reach y_1 is assigned 0 and y_2 is assigned the value of x_1 so you get this. Then you reach test statement which is y_2 greater than OR is equal to x_2. So you take the true path so you add 1 to y_1 and subtract x_2 from y_2 so you get this. Then again the control reaches the test statement which is y_2 greater than OR is equal to x_2. So again the ASSIGNMENT statement is reached and y_1 is increased 2, you add 1 to the value of y_1 and subtract x_2 from y_2 that is 6. Again the control reaches the test point and the same thing holds. So you add 1 to y_1 and subtract x_2 from y_2 so now this is the value.

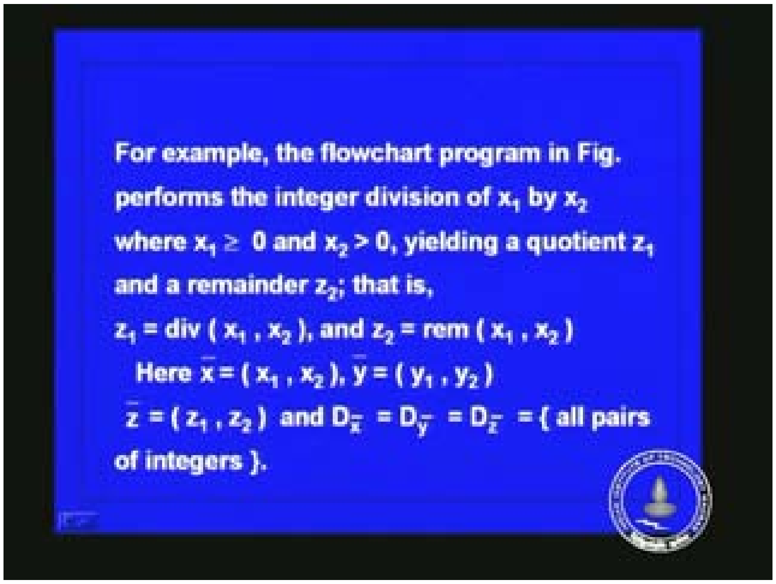
(Refer Slide Time: 18:48)

Now when the control reaches the test point you know that y_2 greater than OR is equal to x_2 is not satisfied so you have to take the false exit. And in that case z_1 will be assigned the value 3 and z_2 is assigned the value 1. Now what does this program do? It divides x_1 by x_2 and the quotient is given in z_1 and the remainder is given in z_2. So z_1 gives the quotient you are dividing x_1 by x_2 and z_1 gives the quotient s_2 and z_2 gives the remainder. This is what the program does. Now, we know that this is correct. It is a very simple program we can easily see that this works correctly.

But how do we technically prove that it is correct and how do we make use of predicate logic for that. The flowchart program in the figure which we have just seen performs the integer division of x_1 by x_2 where x_1 is greater than OR is equal to 0 and x_2 is greater than 0. You cannot divide by 0 so x_2 has to be greater than 0.

Yielding a quotient z_1 and a remainder z_2 that is z_1 is the quotient when x_1 is divided by x_2 and z_2 gives the remainder when x_1 is divided by x_2. Here the input variables are x_1 x_2 and they are pairs of integers non negative integers. The y bar is again y_1 y_2 they are again pairs of integers z bar is again z_1 is an integer z_2 is an integer. So in this case the input domain, program domain, output domain is all pairs of integers. They are all same in this particular example.

(Refer Slide Time: 20:20)

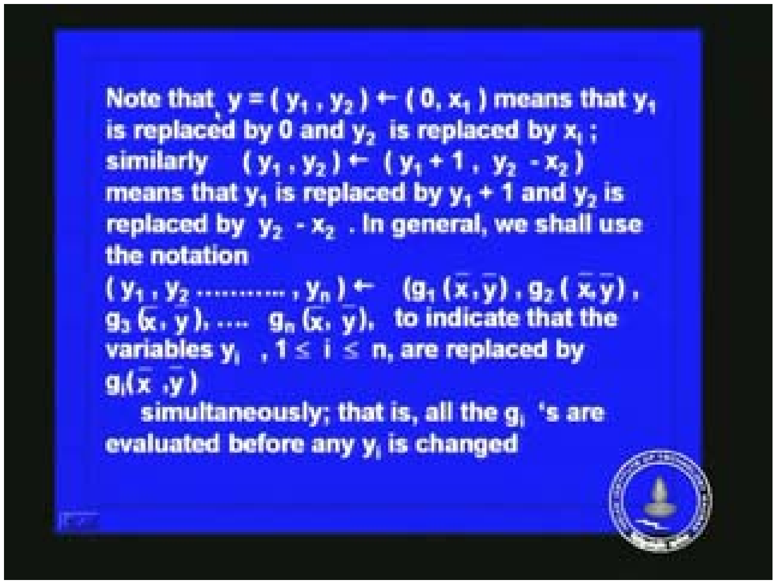


Now there are two aspects to proving a program works correctly. One is partial correctness other is termination. Now, what is this? Generally when you have a program you have what is known as an input predicate which satisfies some conditions. The input should satisfy some conditions in the beginning. Then when the program is executed there is an output predicate which tells you the relationship between the inputs and the outputs. And so when you start the program selecting an input which satisfies the input predicate finally when you reach the HALT statement output predicate should be satisfied, this is what we want.

Now in the partial correctness portion of it you are not bothered about termination what you say is given a input predicate if the program is executed and you reach the HALT statement the output predicate is specified. You are not going to worry about whether it is going to halt or not whereas in the second portion termination you have to worry about the termination of the program. So, given an input predicate you have to show that the program will ultimately terminate for that particular input value. So, in order to prove the program is totally correct you have to prove both parts partial correctness and also termination. Let us see how you do this.

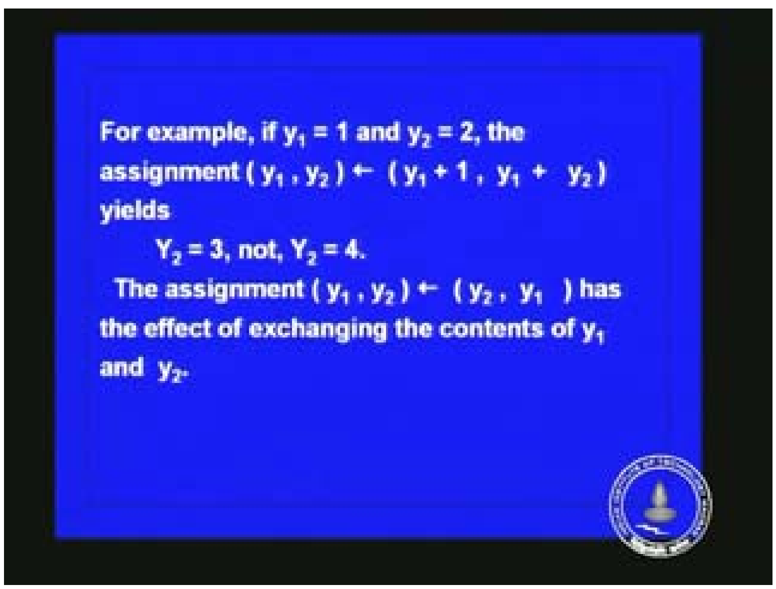
Just for explanation see what an assignment statement is and how the variables are replaced. Now, y_1 y_2 is 0 means y_1 is replaced by 0 and y_2 is replaced by x_1. Similarly, y_1 y_2 is replaced by y_1 plus 1 y_2 minus x_2 means y_1 is replaced by y_1 plus 1 and y_2 is replaced by y_2 minus x_2. In general, we use the notation y_1 y_2 y_n is replaced by g_1(x bar, y bar), g_2(x bar, y bar), g_3(x bar, y bar) and so on to indicate that the variables are replaced by the corresponding values. Simultaneously all the gi's are evaluated before any yi is changed.

(Refer Slide Time: 23:36)



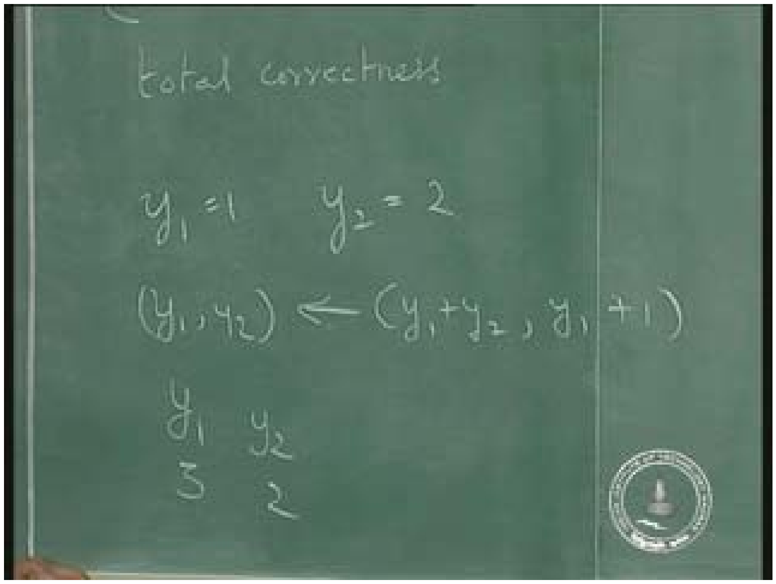
For example if y_1 is 1 and y_2 is 2 the assignment y_1 y_2 is y_1 plus 1, y_2 plus y_2 will yield.

(Refer Slide Time: 24:27)



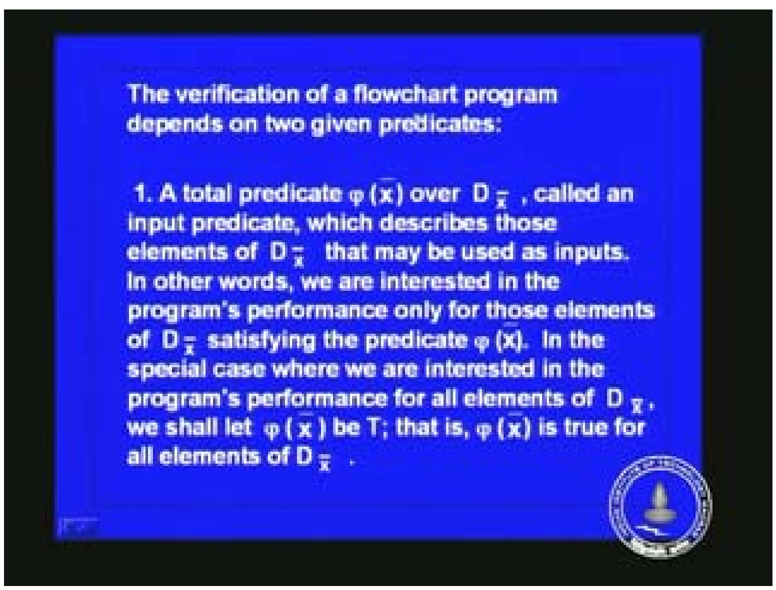
First one y_1 is increased by one value so the new value of y_1 will be 2, let me explain with an example. Suppose you have y_1 is 1, y_2 is 2 and I have y_1, y_2 replaced by y_1 plus y_2 y_1 plus 1 suppose it is like this y_1 will be replaced by y_1 plus y_2. So the new value of y_1 will be 1 plus 2 is equal to 3 and y_2 is replaced by y_1 plus 1 that is 2 like that, 1 plus 1 is equal to 2 like that.

(Refer Slide Time: 25:26)



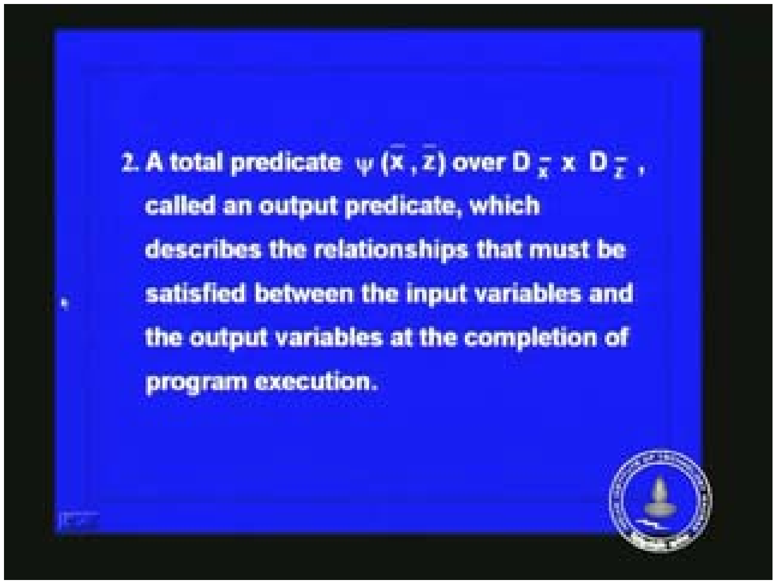
The verification of a flowchart program depends on two given predicates. A total predicate phi x bar over D_x bar is called an input predicate that describes those elements of D_x bar that may be used as inputs. In other words, we are interested in the programs performance only for those elements of D_x bar satisfying the predicate phi x bar. In the special case where we are interested in the programs performance for all the elements of D_x bar we shall let phi x bar to be just true. That is phi x bar is true for all the elements of D_x bar.

(Refer Slide Time: 25:30)



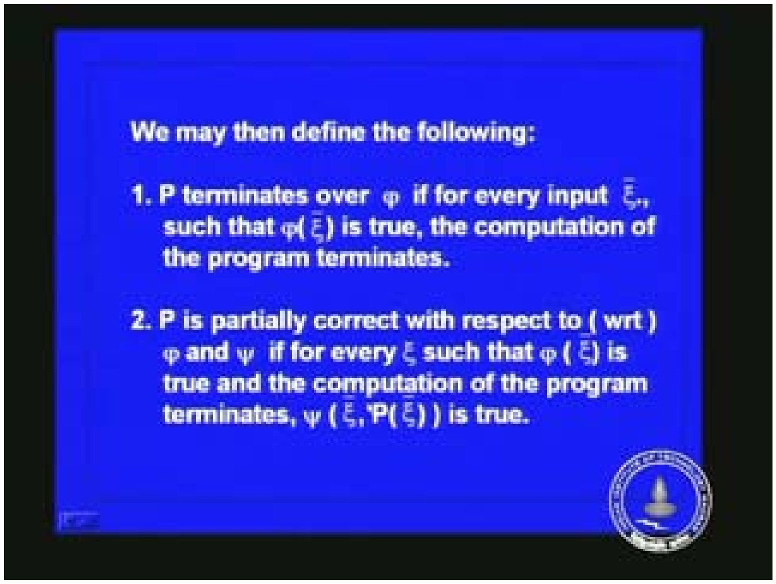
Then you have an output predicate which relates x bar and z bar. A total predicate si x bar z bar over D_x bar into D_z bar called the output predicate which describes the relationship that must be satisfied between the input variables and the output variables at the completion of the program execution.

(Refer Slide Time: 26:15)



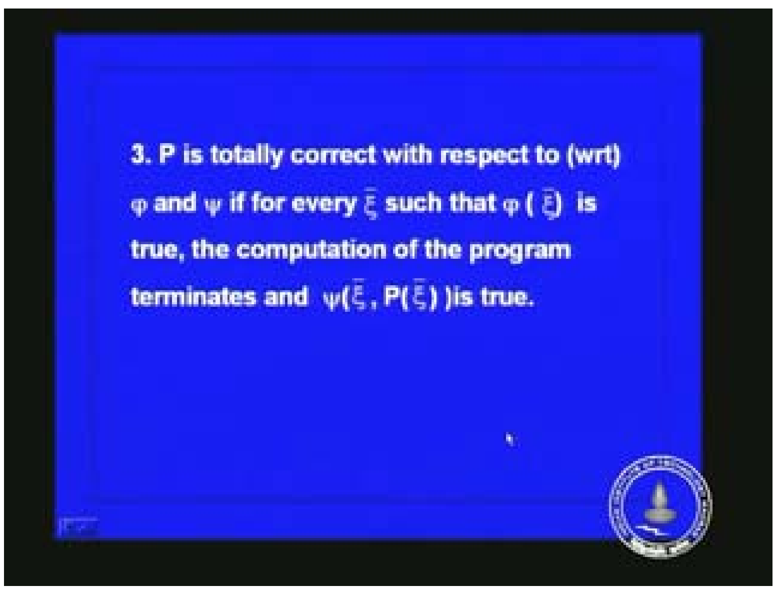
Again as I mentioned to you what is partial correction and what is termination is what you have to see. You say that P terminates over phi if for every input si bar such that phi si bar is true the computation of the program terminates. P is partially correct with respect to phi and si. If for every si such that phi si bar is true and the computation of the program terminates si si bar P(si) bar is true.

(Refer Slide Time: 26:38)



P is totally correct with respect to phi and si if for si bar such that phi si bar is true the computation of the program terminates and si bar P(si) bar is true.

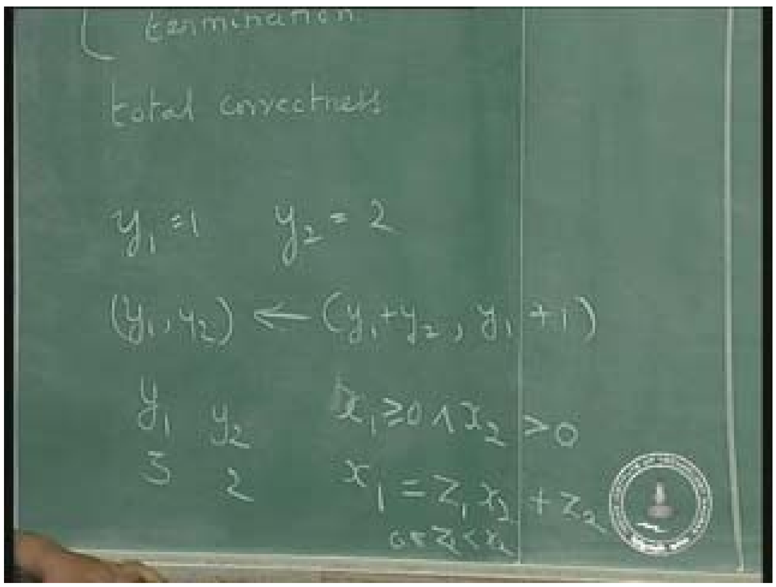
(Refer Slide Time: 27:15)



That is the termination portion if the input predicate satisfies some condition the program terminates. The partial correctness if the input predicate is satisfied and the program terminates you are not bothered about that you assume that it terminates then the output predicate will be satisfied. Now it is totally correct if both these conditions are satisfied given a value si bar which satisfies the input predicate the program will terminate and output predicate will be satisfied. So in this particular example let us see what are the input predicates and output predicates.

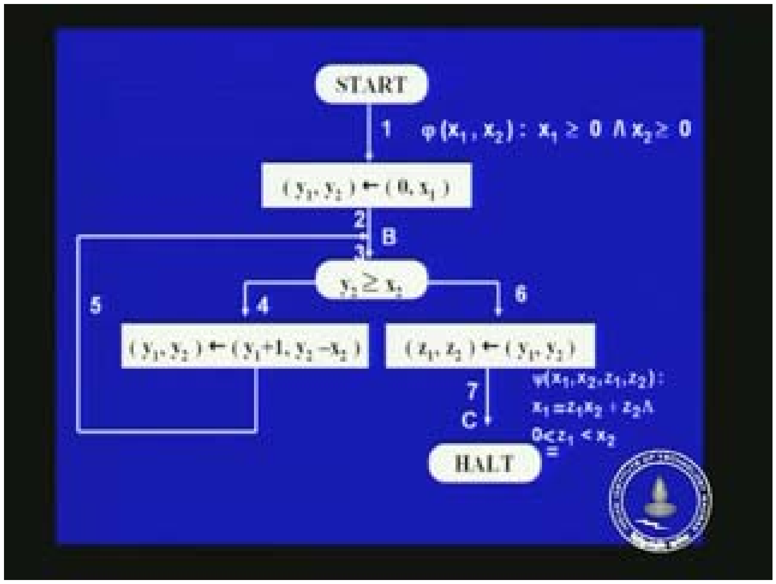
Actually in this case x_2 cannot be 0 because you cannot divide by x_2. So the input predicate will be x_1 greater than OR is equal to 0 AND x_2 greater than 0 this has to be the input predicate. And the output predicate should relate the output variable and the input variables. So what is that? X_1 is the number you are going to divide and you are going to divide by x_2 and the quotient is z_1 so z_1 x_2 the remainder is z_2. So this relation should be satisfied by the outputs. Not only that the remainder should be less than the divisor. So z_2 should be less than x_2 of course it cannot be negative it has to be greater than OR is equal to 0. So these two conditions should be satisfied when the program terminates.

(Refer Slide Time: 29:21)



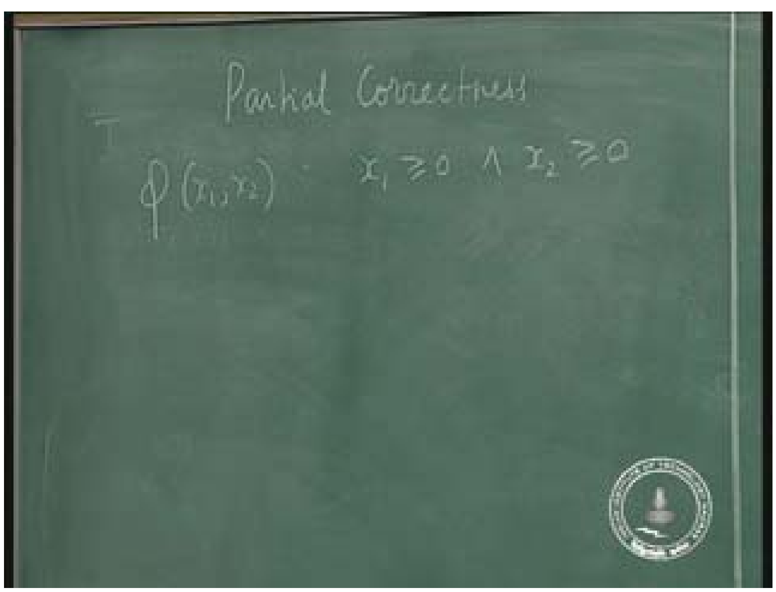
Now, first we are considering partial correctness, when you consider partial correctness even if you take x_2 greater than OR is equal to 0 it is okay. Only for proving termination you require it should be greater than 0.

(Refer Slide Time: 29:41)



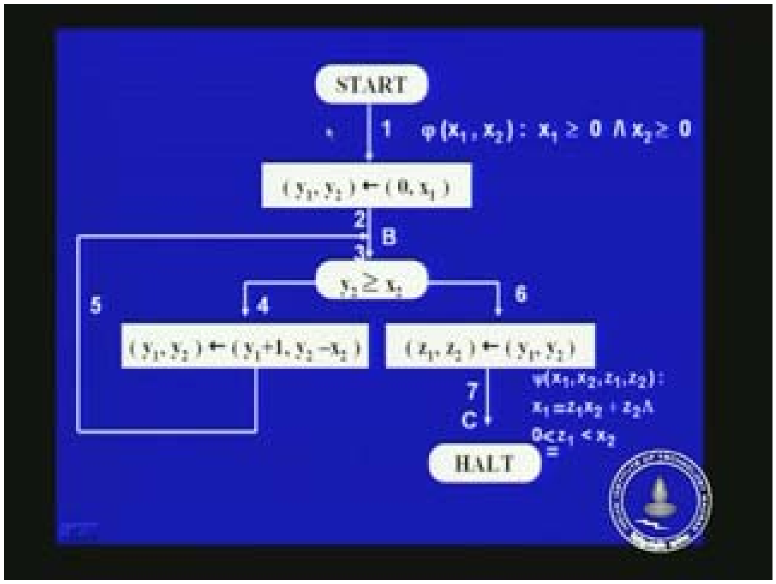
So let us first prove partial correctness then we shall prove termination and then because you prove both the parts we will be proving total correctness. So first we shall consider partial correctness. Here we take the input predicate phi x_1 x_2 to be x_1 greater than OR is equal to 0 AND x_2 greater than OR is equal to 0. Actually for total program x_2 has to be greater than but for proving partial correctness even if you take as x_2 greater than OR is equal to 0 it is okay.

(Refer Slide Time: 30:47)



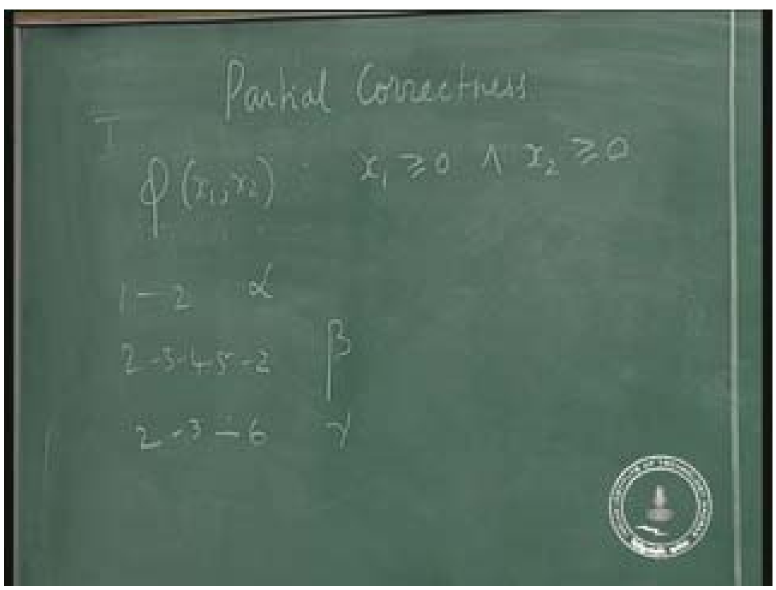
Now, look at the way the program works. You have to divide the program into paths.

(Refer Slide Time: 30:58)



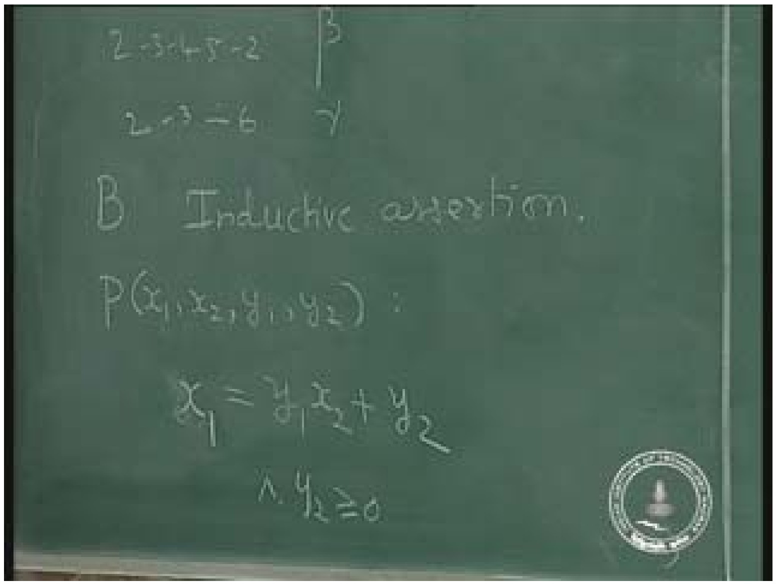
Now the portion 1 to 2 is taken as one path, 1 to 2 is taken as a path alpha. And the portion 2, 4, 5, 2 again is taken as another path and the portion 2, 3, 6 is taken as another path gamma. So the loop is taken as 2, 3, 4, 5, 2 is taken as a path beta. And 2, 3, 6 is taken as a path gamma.

(Refer Slide Time: 31:40)



Now at the point of the loop that is at the point B you must attach a predicate which is called an inductive assertion. And that should bring out the relationship between the input variables and the program variables when the control reaches at that point. So now at B you define an inductive assertion $P(x_1, x_2, y_1, y_2)$. And what is that? It should bring out the relationship between the program variables and the input variables. What is y_2? Initially it is x_1 and you keep on decrementing it. So y_2 is equal to y_1 times x_2 plus y_2, x_1 is equal to y_1 times x_2 plus y_2. And that is you have subtracted x_2 from x_1 finite number of times and the remainder portion is y_2 and y_2 should be greater than OR is equal to 0. This is the condition which should be satisfied by the program variables and the input variables when the control reaches the point P this is the inductive assertion.

(Refer Slide Time: 33:36)



Now taking this we have to form verification conditions for each path and we have to prove that it is correct. Now, look at the path alpha you initial before you have that y_1, y_2 is replaced by 0, x_1. What is the input predicate? Input predicate is x_1 greater than OR is equal to 0, AND x_2 greater than OR is equal to 0. If this is true before this statement is executed what should be true at this point that is what you have to write.

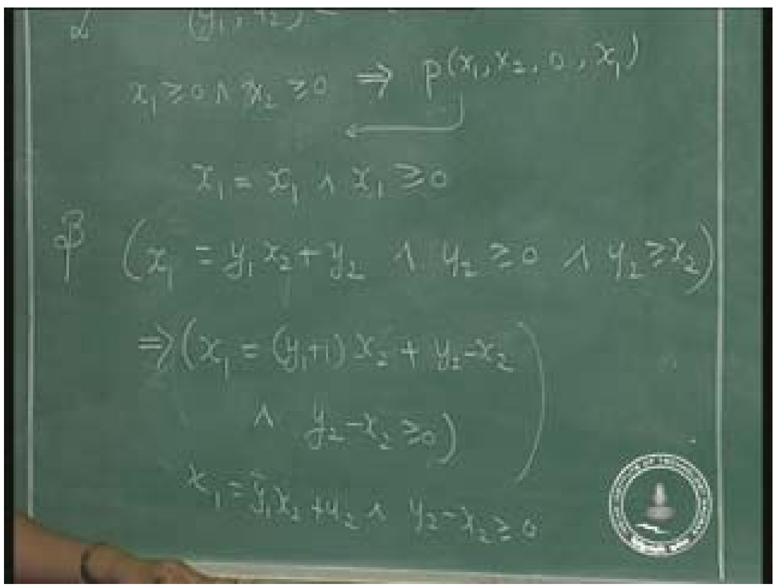
At this point y_1 takes the value 0 and y_2 takes the value x_1 and the inductive assertion at this point is $P(x_1, x_2, y_1, y_2)$. And replacing y_1 by 0 and y_2 by x_1 what is that? So this should imply $P(x_1, x_2, y_1, y_2)$ but (y_1, y_2) at that point is 0 and x_1 because of this. So this is the first verification condition.

And if you replace it what is this? This is, if I expand this then what does that become? It is x_1 is equal to y_1 is 0 so it is y_2 is x_1 so it becomes x_1 is equal to x_1 AND what is y_2? y_2 is x_1 greater than OR is equal to 0. So this is the first verification condition x_1 greater than OR is equal to 0 AND x_2 greater than OR is equal to 0 should imply x_1 is equal to x_1 AND x_1 greater than OR is equal to 0 which is true. So this is the first verification condition and that is proved. Now at the point consider the path beta initially when you start what is the value of $P(x_1, y_1, y_2)$? The inductive assertion is x_1 is equal to y_1 x_2 plus y_2 AND y2 is greater than OR is equal to 0.

Now if you execute the path y_2 greater than OR is equal to x_2 evaluates to true so you also have the condition AND y_2 greater than OR is equal to x_2. So this should imply the value after the assignment is made. The second time the control reaches B what will be the inductive assertion? The values of y_1 y_2 will be changed by the new values y_1 plus 1 AND y_2 minus x_2. So the new value should be x_1 is equal to, now y_1 is changed to y_1 plus 1 AND x_2 plus y_2 minus x_2 which is the new value of y_2 AND what is the new value of y_2? y_2 minus x_2 greater than OR is equal to 0. Let us check whether it is true or not.

Now if you simplify this what will you get? You will again get this because this x_2 will cancel with this x_2 and so you will again get y_1 x_2 plus y_2. So this portion reduces to x_1 is equal to y_1 x2 plus y_2 AND y_2 minus x_2 greater than OR is equal to 0.

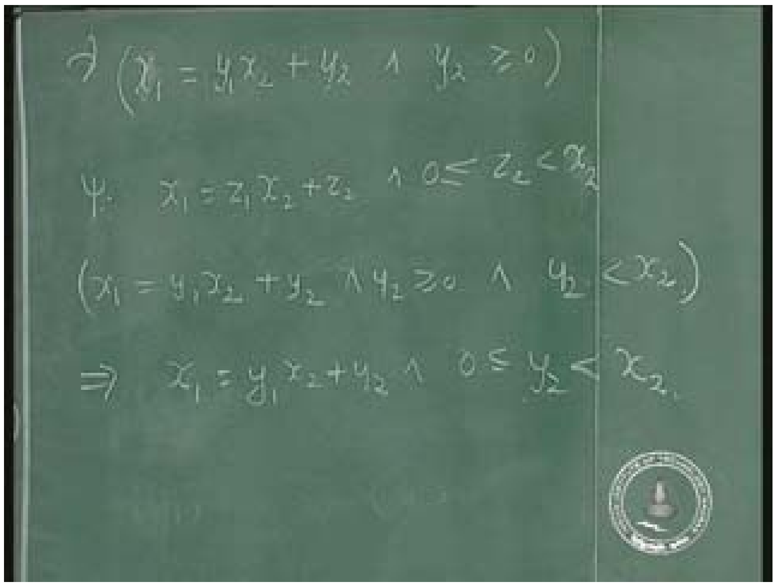
(Refer Slide Time: 38:05)



Now you can see that this is the same as this and because y_2 is greater than OR is equal to 0 AND y_2 is greater than OR is equal to x_2 obviously y_2 minus x_2 will be greater than OR is equal to 0 because y_2 is greater than OR is equal to x_2 you know that y_2 minus x_2 is greater than OR is equal to 0. So this implies this, so the second verification condition for the path beta taking you from B to B along 2, 3, 4, 5 and again back to 2 is also verified and found to be true. Now, consider the path gamma which is from B to C. And here when you start at the point what is the condition that is satisfied? x_1 is equal to y_1 x_2 plus y_2 AND y_2 greater than OR is equal to 0 this is the condition which is satisfied. And when you reach the point C what is the relationship? The relationship between z_1 and z_2 should be satisfied the si value should be satisfied. This x_1 is equal to z_1 x_2 plus z_2 AND z_2 is greater than OR is equal to 0 and less than y_2. This is the condition that should be satisfied.

Now, before coming to the part you make the assignment z_1 as y_2 the new values are you are giving to z_1 AND z_2. So instead of z_1 AND z_2 I can write y_1 and y_2. So the third verification condition is x_1 is equal to y_1 x_2 plus y_2 AND y_2 greater than OR is equal to 0 and you are taking the false path that is y_2 less than x_2. This is the condition which should be satisfied. And all these together should imply the si but now you know that the last step you replace z_1 by y_1 AND z_2 by y_2. So instead of z_1 and z_2 the latest values of y_1 AND y_2 I can use. So this should satisfy x_1 is equal to y_1 x_2 plus y_2 AND 0 less than OR is equal to y_2 less than x_2 the divisor. Now you can very easily see that this is the same as this and this one you can split, actually the combination of these two gives you this.

(Refer Slide Time: 41:50)

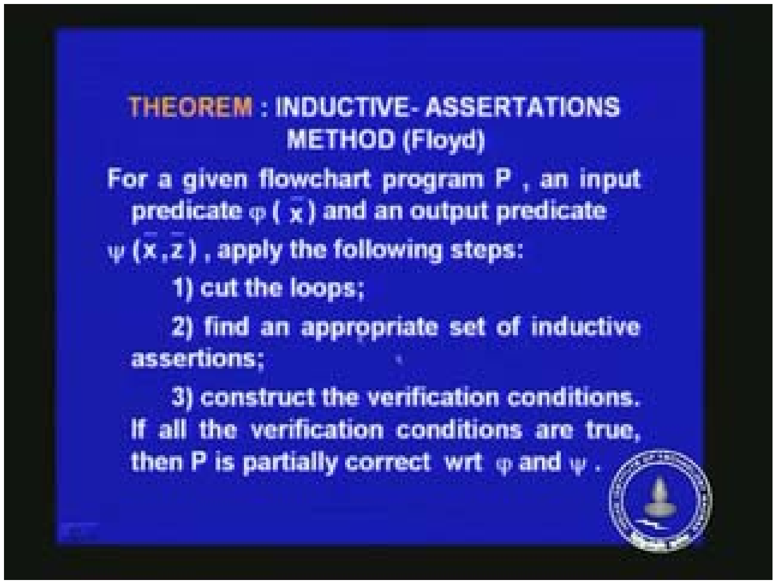


So this is the verification condition for the path gamma and that is also found to be true. So you know that the program is partially correct that is whenever it terminates the output predicate and whenever it starts with the variables satisfying the input predicate finally when it goes to the halt statement the output predicate is satisfied. It performs the integer division which you want. We have seen what input predicate is and the output predicate and so on. So with respect to this, this is the partial correctness, this is the input predicate, this is the output predicate.

Now the inductive assertions method, how do you go about proving the partial correctness?

For a given program flowchart P an input predicate pi x bar and an output predicate si x bar z bar apply the following steps:

(Refer Slide Time: 42:23)

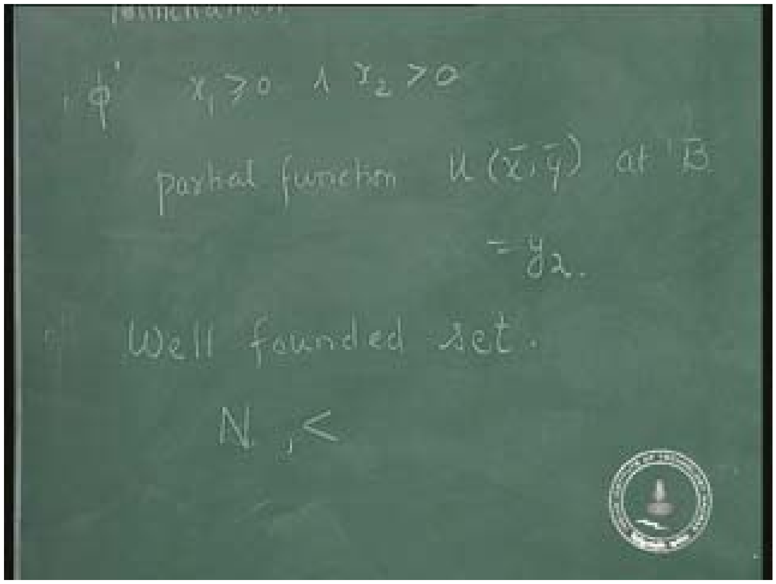


You have to cut the loop you have to cut it from B to B again. Find an appropriate set of inductive assertions in this case at the point B we attach one inductive assertion. In general, the program is more complicated, you may have to attach at every cut point one inductive assertion and so several inductive assertions you may have to write. Those inductive assertions should bring out the relationship between the program variables and the input variables when the control reaches at that point. So construct the verification conditions, so here we have construct the verification conditions for the path alpha for the path beta and for the path gamma.

If all the verification conditions are true then P is partially correct with respect to phi and si. This is for termination. Now, the second portion we have to consider. Let us take the same flowchart and consider termination. For proving termination in this particular example you have to start with the phi dash which is x_1 greater than OR is equal to 0 AND x_2 greater than 0 because you know that when x_2 is equal to 0 the program will not terminate you cannot divide by 0. With respect to this you have to show that ultimately the program will terminate. What you do here is you attach a partial function u(x bar, y bar) at the point B. And in this case the function is just y_2 here. And y_2 should take values from a well founded set, we have not yet studied what is a reflexive and so on.

A well founded set it is a set with an order relation for example here it is less than and here it is the set of non negative integers with the less than relation. And it satisfies three conditions it is irreflexive, antisymmetric and transitive. And every sequence has a least point. In fact here if you take N and less than if you take any sub set of N that will have a least element. Such a set is called as well founded set.

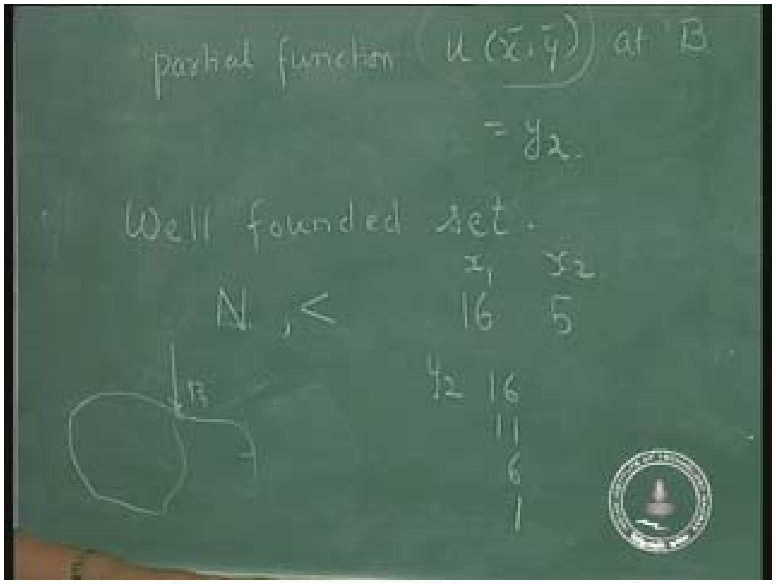
(Refer Slide Time: 45:51)



Now y_2 should take values from a well founded set and you should show that the control reaches the point B and the loop is executed and you go to the point B again. Now the point is when you go from B to B the value of this is reduced. So the first time you go here and second time you go here the value should decrease and because you cannot have a infinite decreasing sequence there is a least element so at some point you will get out of the loop.

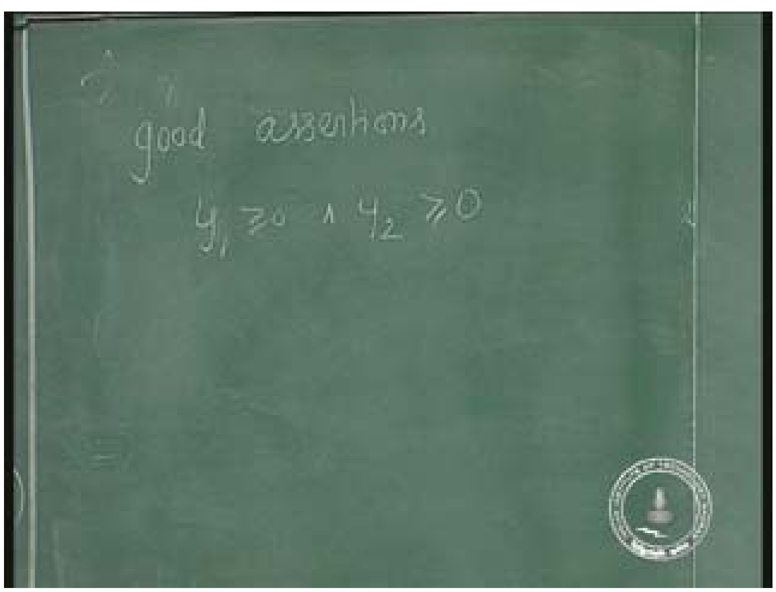
The main point to prove here is that if a loop is executed ultimately you will get out of the loop. So in this case what you have to prove is at this point you must have a partial function like this in this case it is y_2. Now let us take the values of y_2 by taking the example; x_1 is equal to 16 and x_2 is equal to 5 now y_2 was initially 16 next it became 11 then it became 6 then it became 1. It kept on decreasing, the first time the control reached B it was 16 next time 11 next time 6 and so on. This keeps on decreasing and it has to take positive values only and so at some point you will get out of the loop.

(Refer Slide Time: 47:09)



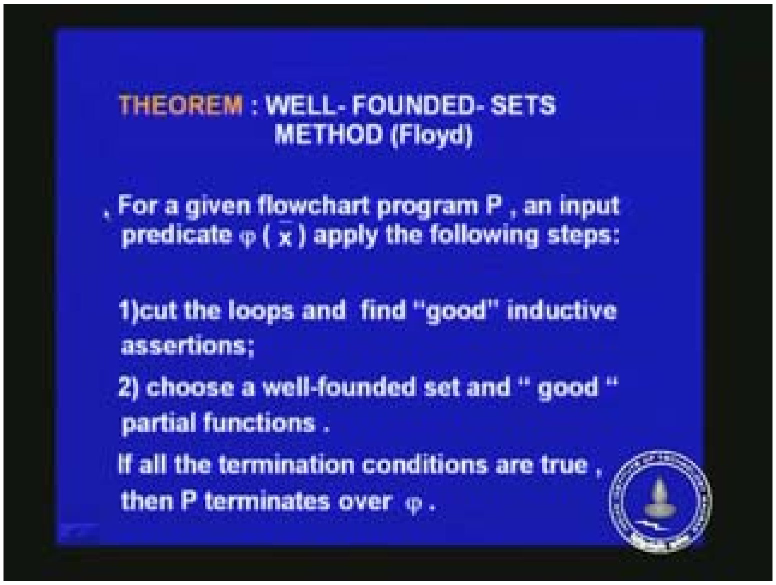
This is what meant by termination so you prove that the program terminates. Not only that usually termination for proving termination you attach "good" assertion at that point like verification condition here also you have "good" assertions. And in this case it can be say y_1 greater than OR is equal to 0 AND y_2 greater than OR is equal to 0 you can say. So these are "good" inductive assertions which should be satisfied at the point where you cut the loops. They need not bring out all the relationship between the input and the output variables but they should satisfy some conditions.

(Refer Slide Time: 48:00)



So you can have something slightly less effective like you know you can just have y_1 greater than OR is equal to 0, y_2 greater than OR is equal to 0 something like that at that point. But the point is you must attach a function u x bar y bar which takes continuously lesser and lesser values from a well founded set and so ultimately you will exit the loop. That is what we want the program will terminate and that is what we want to prove. Well founded sets method: This is the method for proving termination.

(Refer Slide Time: 49:00)

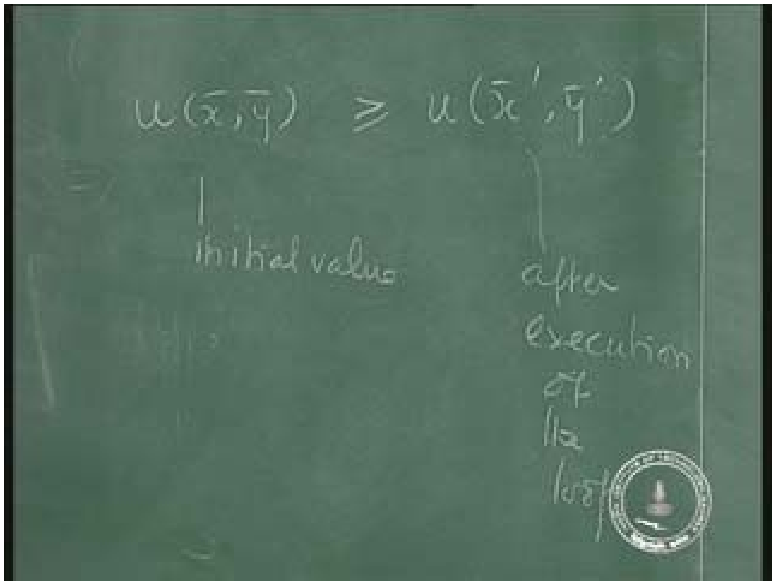


For a given flowchart program P and an input predicate phi si bar apply the following steps:
Cut the loops, here again we have to cut the loops and find "good" inductive assertions. So here we say "good" inductive assertion. There must be some assertions but these assertions need not bring out all the relationship between the input and the output variables. And now choose a well founded set and "good" partial functions, the function which we defined here is a "good" partial function.

What is a "good" partial function? When you start from a point and after executing the loop you reach a same point it should keep on decreasing. Choose well founded set and "good" partial functions. If all the termination conditions are true then the P terminates over phi.

What are the termination conditions here? The termination condition is here, is the first time you have u(x bar, y bar) and after execution of the loop the new values if I say this should be greater than OR is equal to the new value x bar dash and y bar dash. This is the initial value and this is the value after execution of the loop.

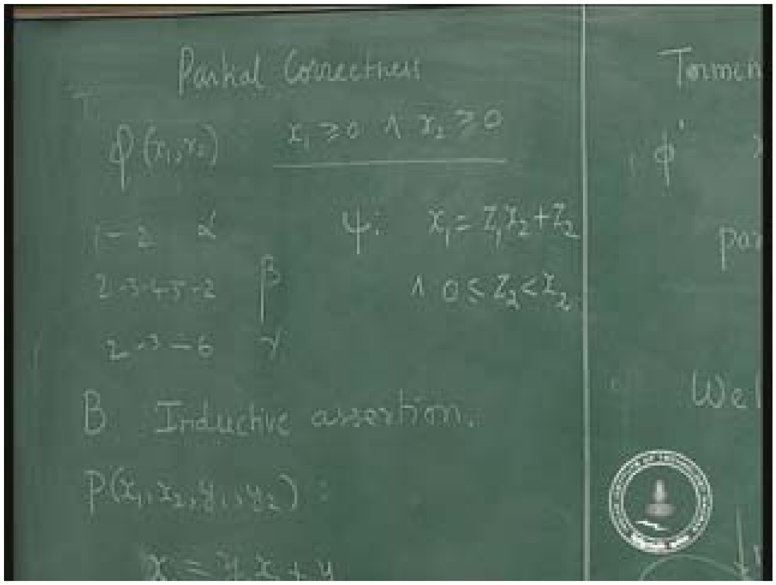
(Refer Slide Time: 50:49)



So it should keep on decreasing like that. So, if you want to prove total correctness of a program you must prove both partial correctness and termination. Thus, only when you prove both the program is totally correct with respect to the input predicate you have specified. Here in this example when you want to prove total correctness you must not take this you must take this because the earlier one it will not terminate. So with this input predicate the program is partially correct it also terminates so it will be totally correct.

And what is the output predicate? The output predicate brings out the relation between the input and output values. In this case it is x_1 is equal to z_1 x_2 plus the quotient and the remainder AND the remainder is less than x_2 but it is a positive quantity.

(Refer Slide Time: 52:05)



So this is the one. But you see there is no automatic way of writing the inductive assertions. You have to think what the program does at that point, what are the conditions satisfied by the input and the program variables at that point and write. So, for a small program you can do that and you can write all the verification conditions and termination conditions etc.

For, if the program is very big you know that this is going to take a lot of time and so you resort to testing rather than proving programs correct. And testing how do you do? You give proper input values thinking that these are the possible values and then run the program and if it gives the proper output values you assume that what you have done is correct, the program you have written is correct. But the correct way of going about any program is you have to prove that the program is correct.

But for practical purposes when the program is very large this may not be possible. But still this way when we have proved it tells you the use of logic, the use of propositional logic, the use of predicate logic in proving that the program is correct. So logic is the basis of computer science and this tells us how it forms the basis in writing programs and proving programs.