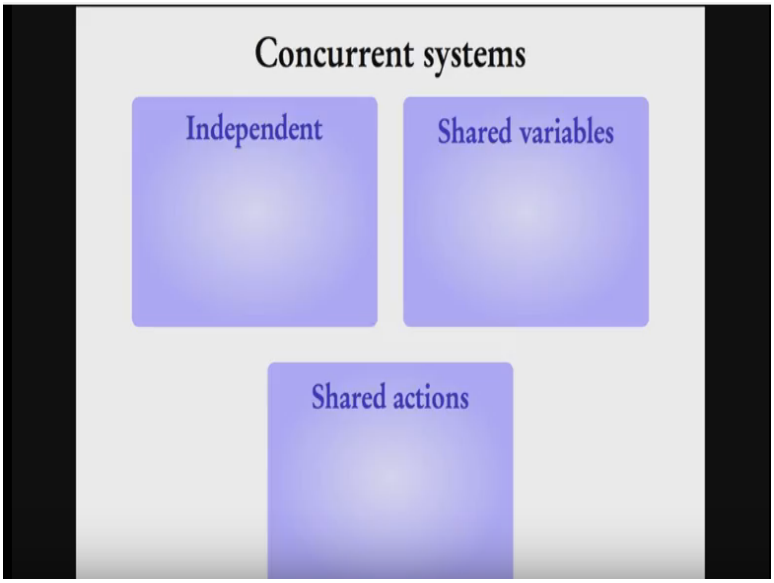
Model Checking
Prof. B. Srivathsan
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture - 05
Modeling Concurrent Systems

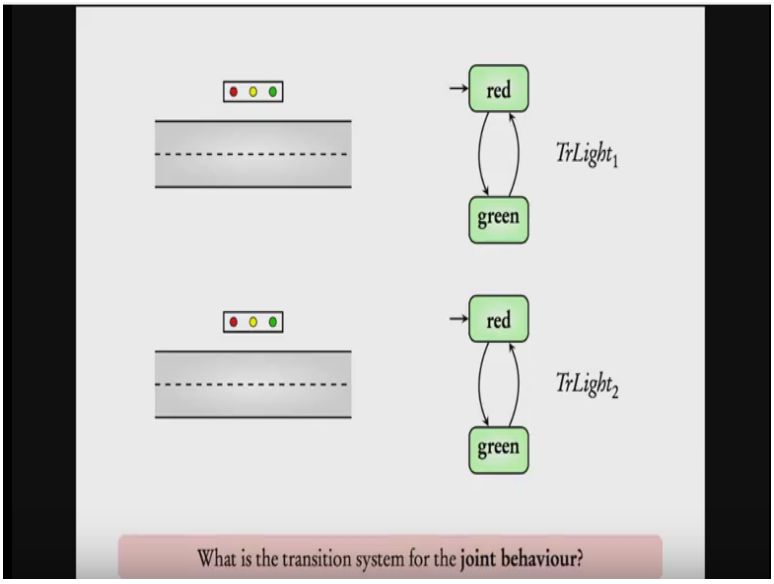
In the previous modules we saw how to model sequential programs and sequential hardware circuits as transitions systems. In this module we will be looking at how to model parallel programs. These are the situations where manual testing becomes quite difficult and model checking could turn out to be useful.

(Refer Slide Time: 00:37)



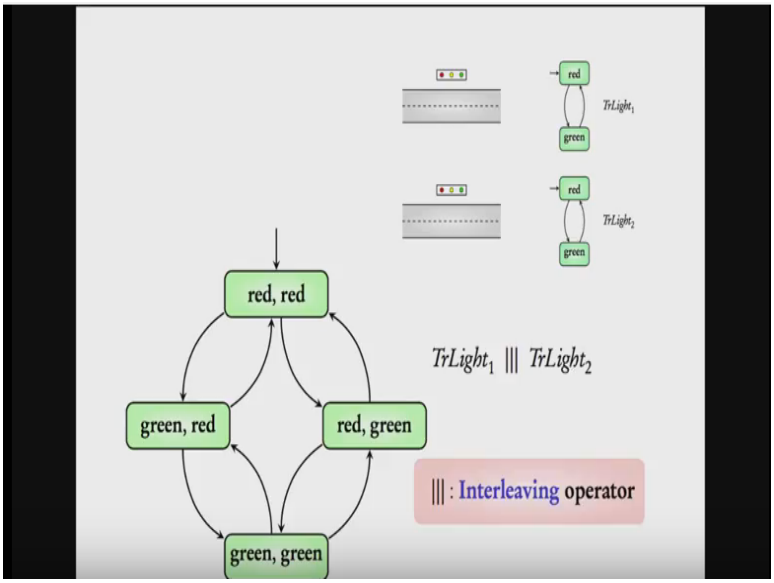
We will be looking at three kinds of concurrent systems independent system that have shared variables systems that have shared actions. Let us start with independent concurrent systems. I will be using the words parallel and concurrent inter changeably.

(Refer Slide Time: 01:03)



Consider two parallel roads with traffic signals. The traffic lights of this road is represent by the simple transition system. From red it can go to green and from green it can go back to red. This is the simple model it does not consider the intermediate yellow light. Similarly for the other parallel road there is a traffic light which is model by this same transition system. How do we model the joint behavior of this system consisting of two traffic lights?

(Refer Slide Time: 01:50)



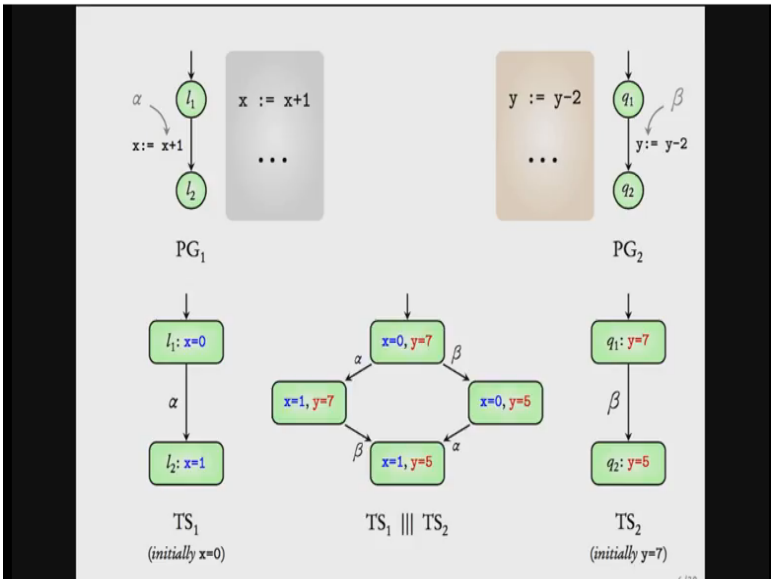
Initially let us say that traffic light one and traffic light two are red either traffic light one can go to green or traffic light two can go to green. Depending on the choice if traffic light one goes to green we go to the state where the first component is green and the second component is red. If

traffic light two goes to green then we go to a state where the first component is red and the second component is green this is a non deterministic choice.

From this state either traffic light one can go to red which is this or traffic light two can go to green which will take us to the state where both are green. Now in this state where both are green if traffic light one changes we go the state if traffic light two changes we go to the state. Finally in this state if traffic light one changes you can come to this state, if traffic light two changes you go to this state. This is a transition system consisting of states and transitions representing the joint behavior of these independent traffic lights. We denote this transition system as traffic light one interleaved with traffic light two.

And this symbol with three lines is called interleaving operator. As the name suggest this transition system represents the behavior where the actions of the participating components or interleaved with each other.

(Refer Slide Time: 04:26)



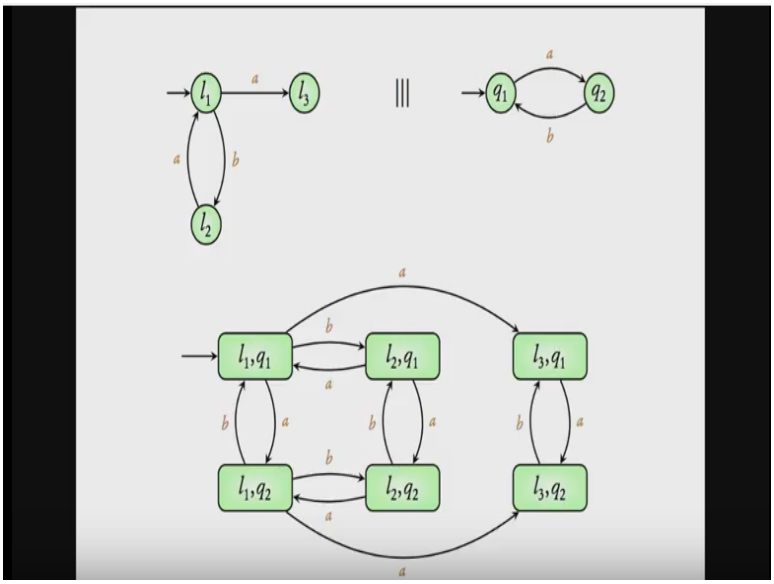
Let us see an other example consider two simple programs, program one assigns x to x+1, program two assigns an other variable y to y-2. This is the program graph corresponding to this program. This is the program graph corresponding to this program. Assume that these two programs are running parallely. How do we represent the joint behavior of this program?

Let us first look at the transition system corresponding to this programs with some values of variable that we have taken. We have taken the initial value of x to be 0 and the initial value of the y to be 7. Starting from l1 x equal to 0 the transition system goes to l2 with x equal to 1. This is the transition system corresponding to the program graph PG1 and the value x equal to 0. Similarly, this is the transition system corresponding to the program graph PG2 and with the initial value of y taken to be 7.

Let us consider the interleaved transition system. For clarity I have not written l1 and q1 however this state is going to be l1 x equal to 0 q1 y equal to 7. There are two possibilities either program one can move forward or program two can move forward if program one moves forward the value of y remains unchanged the value of x increases by 1. If program two moves forward the value of x remains unchanged but the value of y is reduced by 2.

Let us look at this state program one cannot move forward only program can move and the next state would be x equal to 1 and y equal to 5. Similarly, in this state program two cannot move forward program one moves forward and comes to the state where x equal to 1 and y equal to 5. No matter which ever order we take we reach the state where x is 1 and y is 5. This transition system represents the interleaving of TS1 and TS2.

(Refer Slide Time: 07:34)



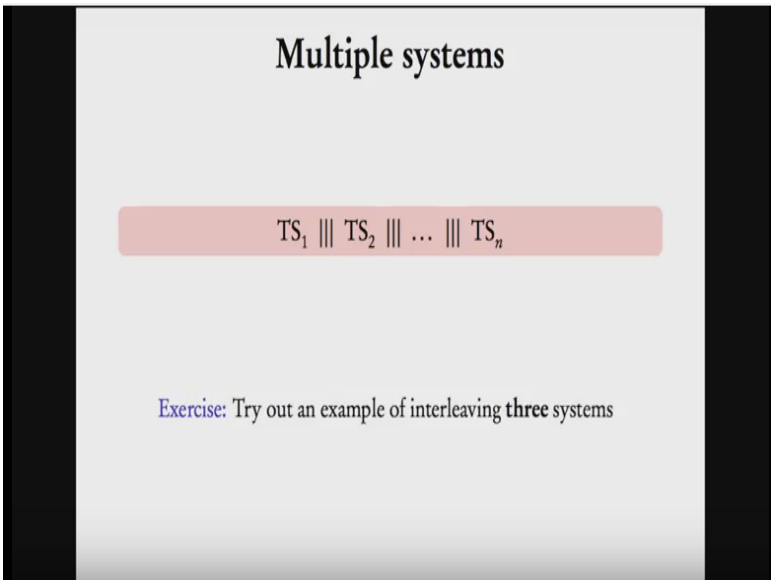
Let me give another example consider these two transition systems. Let us now try to build the transition system which is interleaving of this one and this one. First what are the states of the interleaved transition systems.

They will be l1 q1 or l2 q2, l2 q1, l2 q2, l3 q1, l3 q2. This state represents the fact that transition system one is in state l1, transition system 2 is in state q1. This state represents the fact that transition system one is in state l3 and transition system 2 is in state q 2. What are the transitions of the interleaved transition system? On an a either transition system can move forward or transition system 2 can move forward.

If 1 moves forward the next state would be l3 q1, if 2 moves forward the next state would be l1 q2. What about b on a b transition system 2 cannot move forward the only option is for l1 to go to l2. On a b the next state would be l2 q1 this component unchanged the first component goes from l1 to l2. At l2 q1 on an a either l2 goes to l1 and q1 remains wherever it is or transition system remains as such and q1 goes to q2. This is depicted by these two transitions.

At l3 q1 on an a l3 cannot move only q1 can go to q2 that's why on a l3 q1 goes to l3 q2. With similar arguments you can complete this picture. This picture gives the interleaved transition system.

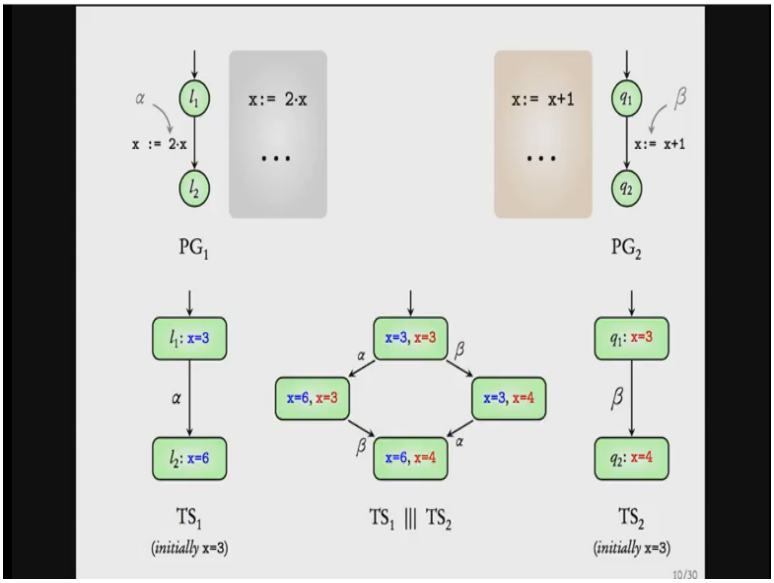
(Refer Slide Time: 10:32)



The interleaving operator works for multiple transition systems as well if you have TS1 TS2 till TSn you can interleave each of them. The states would be a Cartesian product of each of these transition systems and the transition are given in the same way as discussed in this example. As an exercise try out an example of interleaving three transition systems. We have come with the end of the first part of this module we were looking at independent programs or independent systems.

To model the joint behavior of these independent systems we defined the interleaving operator. The next kind of systems that we are going to look at will not be independent they would have something in common they would have shared variables.

(Refer Slide Time: 11:50)



Consider two parallel programs working on a shared variable x. Program one assigns x to 2x , program two assigns x to x +1. These are the corresponding program graphs. Let us look at the transition systems starting with x equal to 3. Here l1 x equal to 3 this is the initial state when action alpha which x is going to x is applied. The transition system moves to l2 x equal to 6. Here the initial state is q1 x equal to 3 then action beta is apply this state become q 2 x equal to 4.

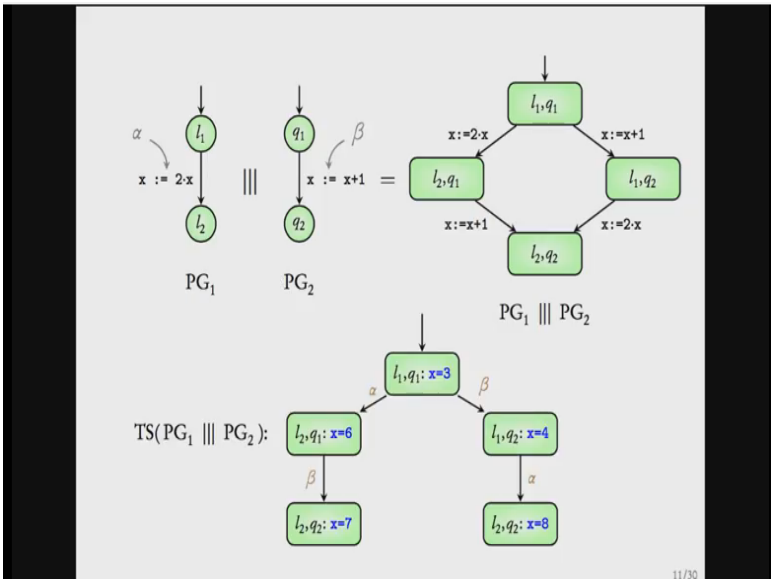
If we use the interleaving operator define before for independent systems this is what we get. The initial state is l1 x equal to 3 q1 x equal to 3. There are two possibilities either the first

components moves or the second component moves. If the first component moves the state becomes l2 x equal to 6 the second component remains the same.

If the second component moves the state becomes l1 x equal to 3 and q2 x equal to 4. In this state TS1 cannot move only TS2 can move when it moves the value in of x in the second component becomes four. Now similarly in this state only TS1 can move and it goes to the same state where the value of x is 6 in the first component and the value of x is 6 in the second component. Clearly such states are unrealistic.

This state says that the value of x is 6 and the value of x is 4 which is not possible. Hence the interleaving operator that we defined for independent systems does not work when there are shared variables.

(Refer Slide Time: 14:36)



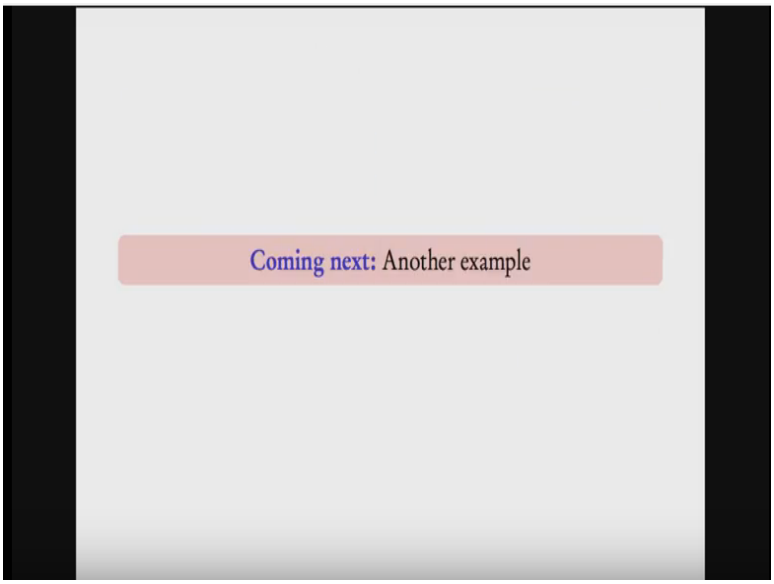
Let us see what to do when there are shared variables. Consider the program graphs not the transitions systems but the programs graph and interleave the program graphs directly. The states are l1 q1, l1 q2, l2 q1 and l2 q2. From l1 q1 there are two possibilities either l1 goes to l2 or q1 goes to q2. If l1 goes to l2 the effect is x going to 2x. If q1 goes to q2 the effect is x goes to x+1. From l2 q1 only PG2 can move and the effect of this transition to make x2 x+1. Similarly from l1 q2 only l1 can move and the effect is x goes to 2x. This defines the interleaving of the two program graphs.

Let us try to see a transition system of this program graph starting from x equal to 3. So the states are the location and the value of the variable. Initially the location is l1 q1 with x equal to 3. If this transition is taken the location is l2 q1 with the value of x changing to x equal to 6. If this is the transition that this is the transition then the transition system moves l1 q2 with x being 4. Similarly from l2 q1 with x equal to 6 if you take this transition the value of x is becomes x+1 so you go to l2 q2 x equal to 7.

On the other hand from l1 q2 if you take this transition you go to l2 q2 however the value of x is not 7 but it two times 4 which is 8. The transition system of this program graph clearly distinguishes the order in which the two actions are taken. When we have shared variables we should not interleave the transition systems instead we should interleave the program graphs and look at the transition system corresponding to this interleaved program graph.

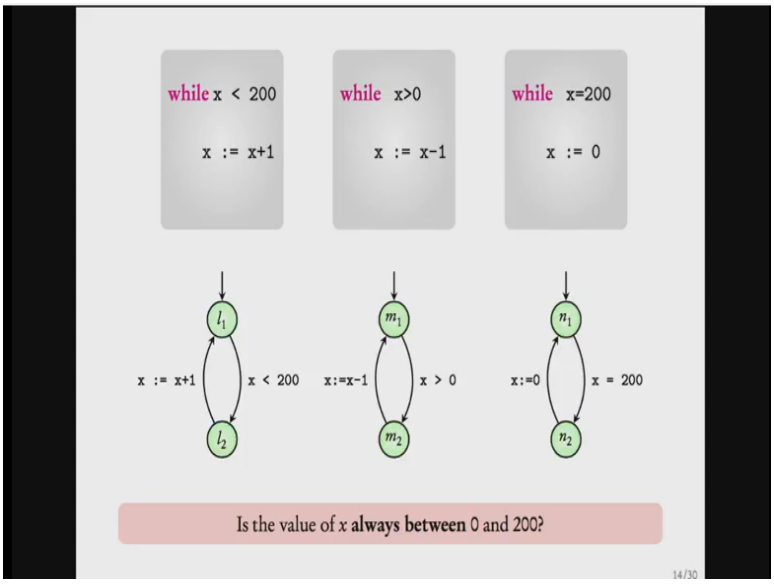
This is the summary for shared variables we should not look at TS1 interleave with TS2 interleave with TS3 so on. In still we should look at PG1 interleave with PG2 interleave with till PGn and then we should look at the transition systems corresponding to this interleaved program graph.

(Refer Slide Time: 17:57)



Let us look at another example.

(Refer Slide Time: 18:03)



Consider these three simple programs the first one if check x is less than 200 and increments it by 1 as long as x is less than 200. The second one decrements x is as long as x is bigger than 0. The last one resets x to 0 when x is 200. Assume that these three programs are run parallely for example, assume that the initial value of x is 5. Program one takes control checks if x is less than 200 increments x by 1. So the value of x would now be 6.

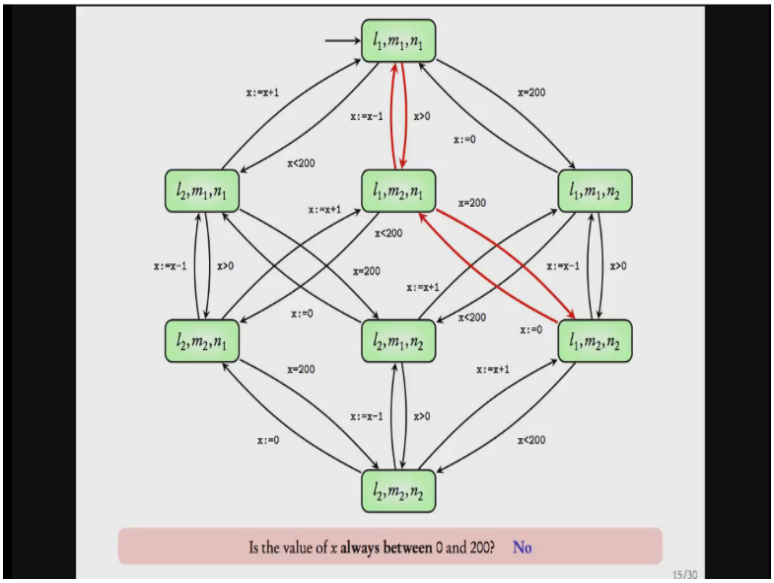
Then program two could take control checks x is bigger than 0 is decrements x and now the value of x would be 5. Then program one might take control increase it yet again program one might take control increase it then program two might take control decrease it and so on. Given these three programs suppose we ask the question is the value of x always between 0 and 200? At first glance this looks to be true. However, this might not be true always. Suppose the initial value of the x is 200.

Let us say that program three takes control it checks if x is 200 yes and it enters. Then program two takes control it checks if x is bigger than 0 yes is enters. Now again program three takes control its inside the while loop it will set the value of x to be 0 now the value of x is 0. Now the control goes back to the program two which is already here the value of x is 0 now and after this

execution the value of x would become minus 1. Are assumption that once the program enters this while loop it will finish executing it before the control goes to another program is not true.

These are situations which are hard to catch manually. Let us try to model these programs as programs graphs. For this program we have two locations location l 1 represents this place where the condition to entire the while loop is checked. If x is less than 200 the inside of the while loop is reached the inside of the while loop is model by the location l2. Now the control goes back to l1 after setting x to x +1. Similarly the program graph for this program is this one and for this it is this.

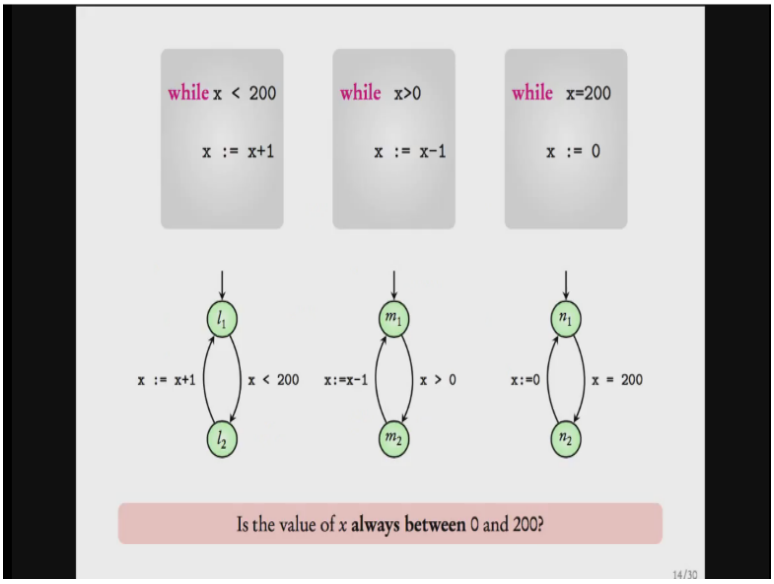
(Refer Slide Time: 21:33)



This graph represents the interleaving of these three program graphs. The states represents the current location of each of these three programs. The transitions are given as we discussed before. If we look at this program graph and look at the transition system of this program graph corresponding to the initial value x equal to 200. We would be able to see that the path which checks for x bigger than 0 and then checks if x equal to 200. Resets x, decrements x will give us a state where the value of x would be minus 1.

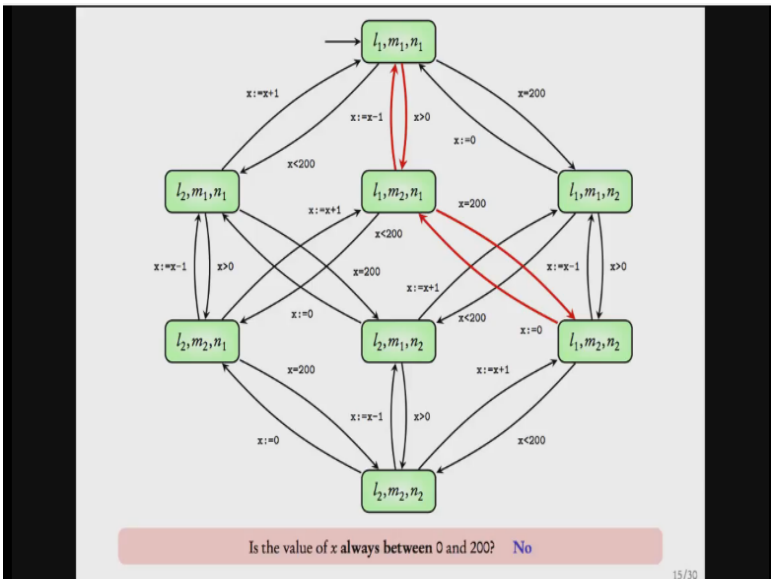
So the answer to this question is the value of x is always between 0 and 200 is no. It could be nice to have tools which can answer this question given

(Refer Slide Time: 22:42)

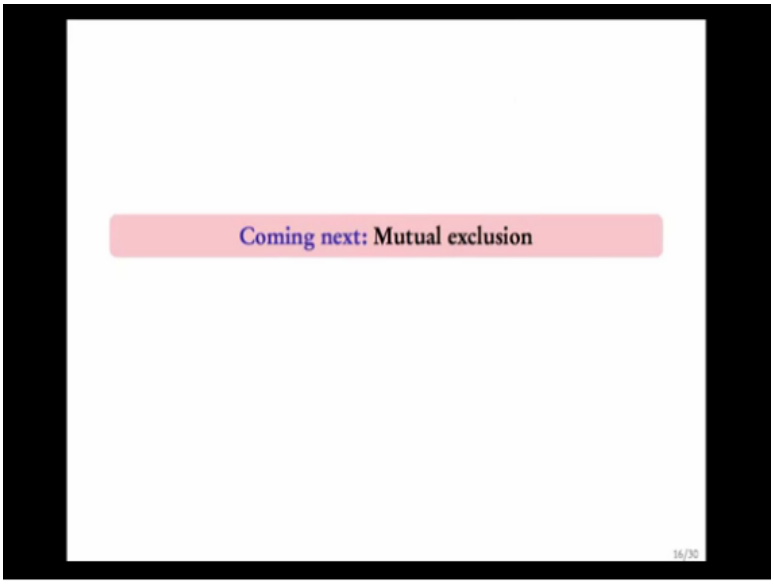


just these three program graphs. We will be looking at such tools later during the course.

(Refer Slide Time: 22:50)

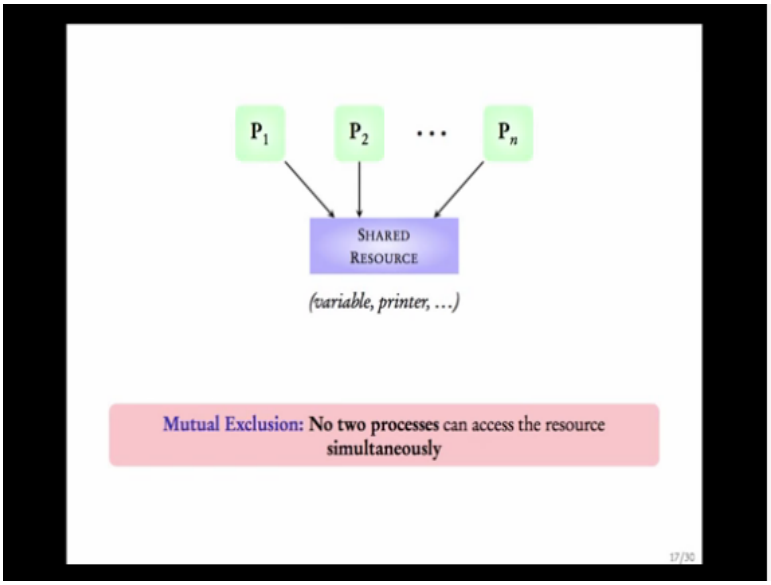


(Refer Slide Time: 22:52)



As a follow up to the example let us look at the concept of mutual exclusion.

(Refer Slide Time: 23:11)

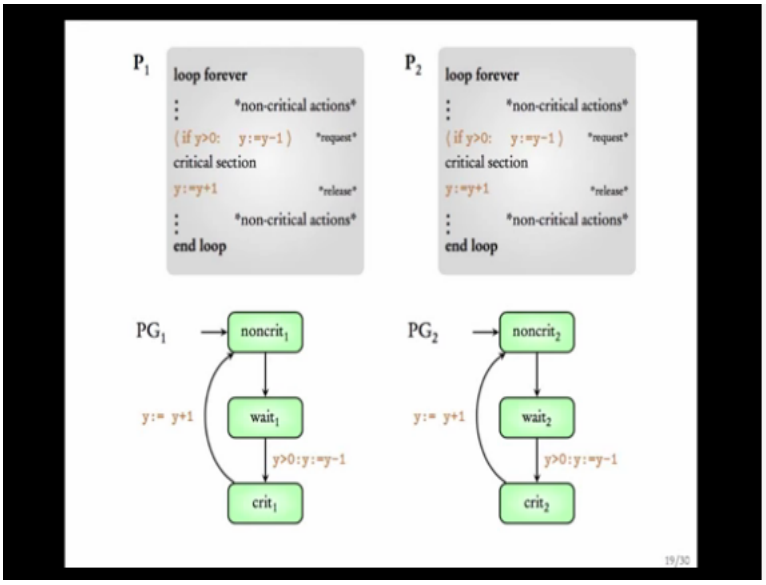


Suppose there are n parallelly executing programs assume they have a shared resource. This shared resource could be either a variable or some hardware etc. Mutual exclusion is a rule which says that no two of these programs can access this shared resource simultaneously. For example if the resource is a variable mutual exclusion demands that no two programs are simultaneously in the part of the their code which writes to this shared variable.

(Refer Slide Time: 23:49)

To ensure mutual exclusion the programs have to be modify in a suitable way. The goal is to model these modifications in other words the goal is to model the protocols that are used for mutual exclusion.

(Refer Slide Time: 24:11)



Let us consider just two programs we divide the program into critical and non critical sections. In this part of the program the shared resource is not used for instance. Here the program gives a request that it wants to access the resource and enters its critical section. Once its access is over it releases the critical section. It follows up with non critical actions and starts the loop again. This is the simplistic view of the program here are the corresponding program graphs.

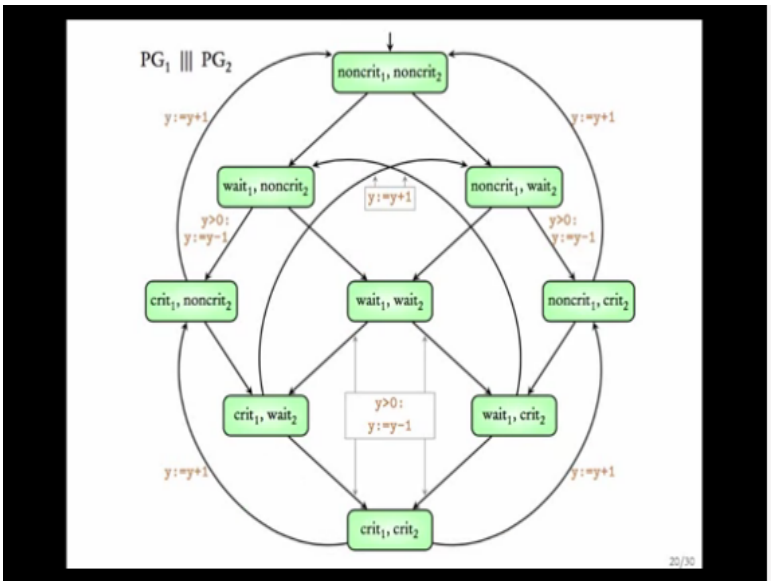
Initially the program is in a non critical location it enters a state where it is waiting for access to the critical section, it enters the critical sections once it is over it goes back to the non critical sections and so on. This is the program graph for program P1, this is the program graph for program p2, So far we have not ensured mutual exclusion of these two programs they could very well be in their critical sections simultaneously. Let us look at a simple protocol for ensuring mutual exclusion.

Add a new Boolean variable y to the two programs before entering the critical sections p1 checks if y is bigger than 0 if so it decrements the value of y by 1. Once the critical section is over it increments the value of y by 1. P2 is identical before entering the critical section p1 will check if y is bigger than 0 since y is Boolean it could either be 0 or 1. If y is bigger than 0 it can only be 1 it will immediately make the value of y to 0 and then it will enter the critical sections.

Why it is inside the critical section? the value of y is still 0. Process p2 cannot enter the critical section now because before entering it needs to checks if y is bigger than 0. But y is still 0 it has to wait till process p1 hence its critical sections increases the value of y back to 1.

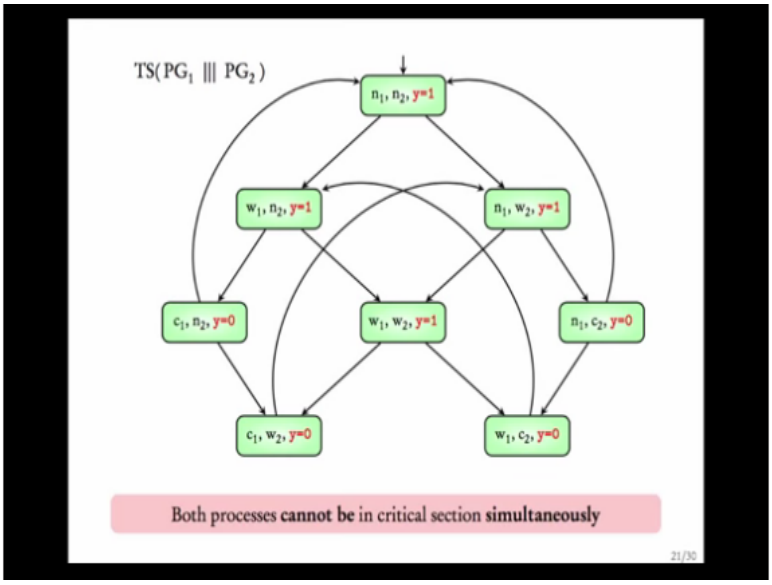
Note that for this protocol to work correctly these two statements should be executed consecutively. For instance if p1 enters by checking if y is bigger than 0 and before it can decrement y if p2 executes if y greater than 0 and enters this part of the code it is very well possible that p1 and p2 enter the critical sections simultaneously.

To ensure that these two statements happen consecutively they need to be declared atomic. Atomicity of simple statements like this can be ensured at the hardware level. If this block of code is declared atomic and if p1 enters this block of code then p2 cannot be executed till p1 finishes executing this block of code. This atomic block of code can be represented in a single transition of the program graph which checks if y is greater than 0 and decrements the value of y.
(Refer Slide Time: 28:30)



Now that we have the program graphs corresponding to these two process we can construct the interleaving of these two program graphs. The initial value of y is 1 given this interleaving and given that the initial value of y is 1 we can construct the transition system of this interleaved program graphs.

(Refer Slide Time: 28:45)



n1 represents that process p1 is in its non critical section, n2 represents that process p2 is in its non critical sections. Similarly w1 says that process 1 is waiting to enter its critical section, c1 says that the process p1 has enter its critical section. In this transition system we can see that the location where both the process are in its critical section cannot be reached. Let us look at the sample executions both of them are in the non critical location. The value of y is 1 process p1

enters its waiting state. It checks that the value of y is 1 it then decrements the value of y to 0 and enters its critical section.

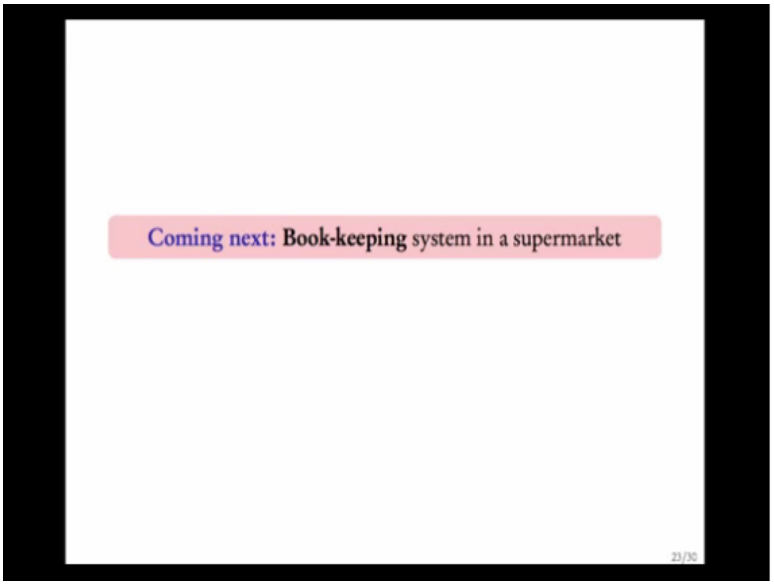
Now process two enters its waiting state it wants to enter the critical section however the value of y is 0. So it cannot go further till process 1 completes its tasks and increments the value of y back to 1. Hence we can conclude that the protocol that we have used for mutual exclusions is correct. Both process cannot be in critical section simultaneously.

Let me summarize what we have seen so for. We have seen two kinds of parallel programs. The first one consisted of independent parallel programs in independent parallel programs the joint behavior can be describe using the interleaving of the corresponding transition systems of each of the programs. When the parallel programs have shared variables we saw that doing just the interleaving of the transition systems will fail.

Instead we should interleave their program graphs and look at the transition systems corresponding to this interleaved program graphs. When we discuss parallel programs consisting of shared variables the notion of mutual exclusion is important. We saw how to model protocols used for mutual exclusions. We modeled a simple mutual exclusion protocol and check that if this protocol is follow both the process cannot be in the critical sections simultaneously.

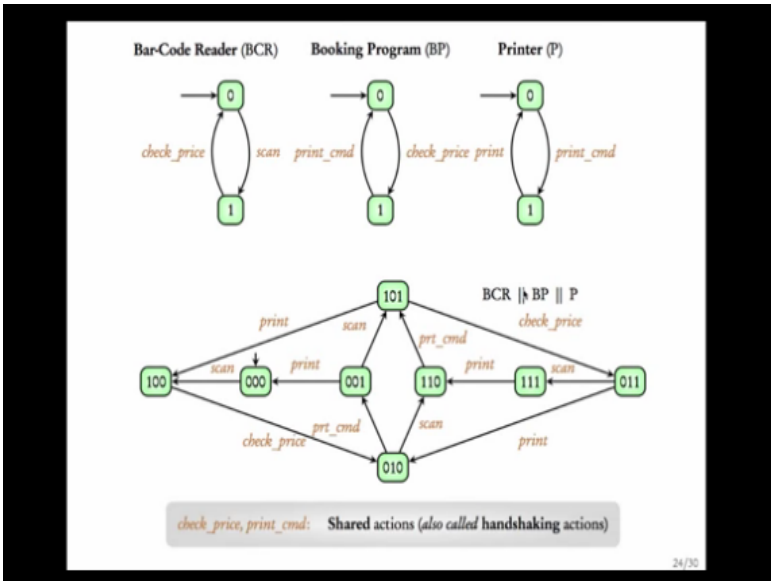
We will now look at another kind of concurrent systems which have common actions these actions are called shared actions. As usual we will be studying this concept by means of examples.

(Refer Slide Time: 31:54)



We will first look at the book keeping system in a supermarket let me explain what this is. Suppose you are at the billing desk in the supermarket. The cashier at the billing desk has a bar code reader which scans the barcode of your product sends this barcode to a program which searches its data base and finds the price of this product. This program in turn sends the price to the printer which finally prints it in your bill.

(Refer Slide Time: 32:16)



This is the system that I just described. It consists of three systems which are interacting with each other a bar code reader, a booking program and a printer. The bar code reader scans the bar code and sends a message to say the price of this product needs to be found. The booking program receive this message checks the price of the product and sends a print command. The

printer on receipt of this print command finally prints it on your bill. These are the transition systems of the individual components.

Let us now try to find the transition system that describes to joint behavior of these three components. Initially all the components are in state 0. On a scan the bar code reader goes to state 1 so the joint state would now be 1 0 0. At state 1 of the bar code reader there is an action on check price the same action occurs from state 0 of the booking program. So now on check price both the bar code reader and the booking program change their state.

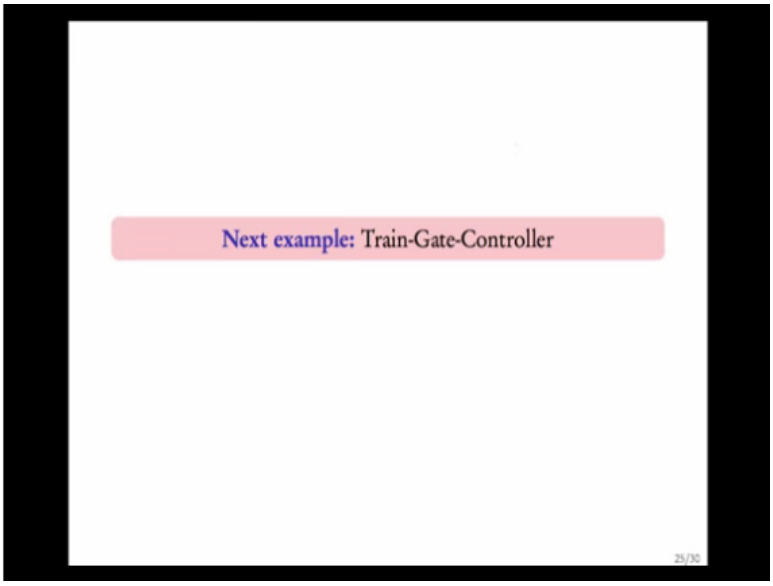
The first component moves from 1 to 0 and the second component which is the booking program goes from 0 to 1. This is an action which is common to both bar code reader and the booking program and hence when it is available in a state both the components would have to move such an action is a shared action also known as a handshake action. At state 0 1 0 the booking program can move from 1 to 0 on a print command the printer which is in state 0 on a print command can go to 1.

This gives us the following transition from 0 1 0 you can go to 0 0 1. The bar code reader does not change its state the booking program goes from 1 to 0 and the printer goes from 0 to 1 the print command is yet another shared action. We can thus finish the transition system whenever there is a shared action all the components that are sharing the action would move. For other actions which are not shared only the component that plays the action can move.

For example scan is not a shared action so from 0 0 1 on a transition with the scan only the first component moves the others stay where they are. On the other hand check price is a shared action which is common between the booking program and the bar code reader on a transition with a check price both booking program as well as the bar code reader should have to move.

This transition system is represented this way we would be using two lines to denote the operation that we just described this will be called the hand shake operator. Note that this is different from the interleaving operator where there were no shared actions.

(Refer Slide Time: 36:52)

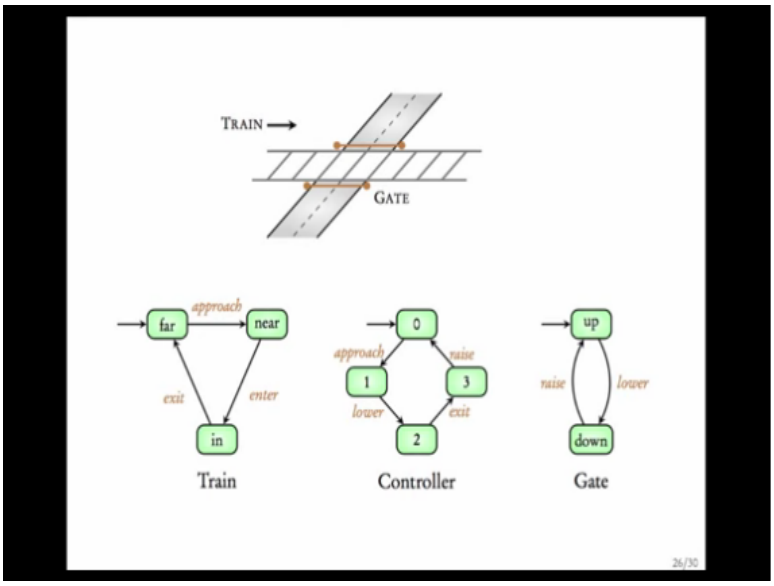


Let us look at another example of a train gate controlling system. Consider a railway crossing with an automatic gate controller when the train is approaching the gate it should send a signal to the controller the controller receives the signal and lowers the gate. Once the train has crossed it should send a signal to the controller saying that it has crossed. The controller should then send the signal to the gate and raise it. This is depicted using the following transition systems for the train, the controller and the gate.

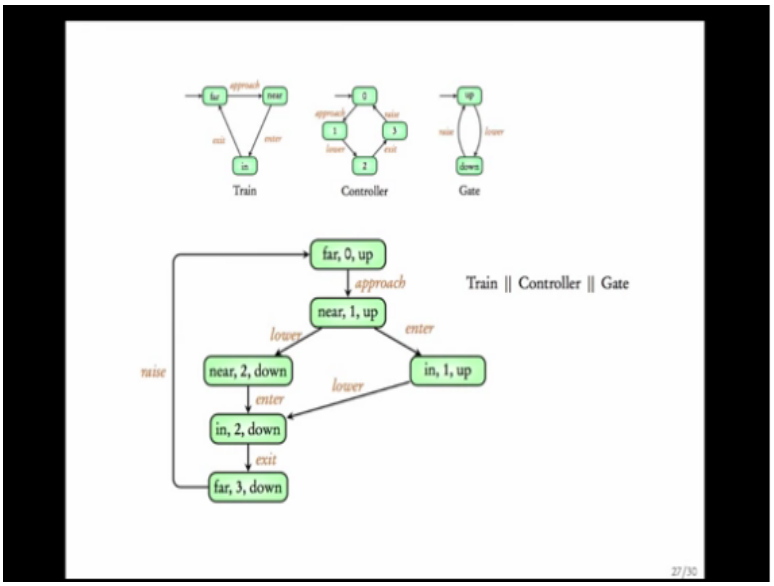
Let us start with the train, the train is initially in the far state it sends a signal approach and enters a near state. From the near state it sends the signal saying that it is entering the crossing and goes into the state where it is inside the crossing. From the instate it goes back to the far state through an action exit. Let us now look at the controller, the controller receives the approach signal and then lowers the gate. In this state when it receives exit it should raise the gate.

This is the actual gate initially it is up when it receives a lower action it should come to the state and when it receives a raise it should go to the state.

(Refer Slide Time: 38:52)



(Refer Slide Time: 38:55)



This is the transition system representing joint behavior of the train controller and gate. Approach, exit, lower, raise are handshake actions. Approach a shared between train and controller, raise a shared between controller and gate and so on. Initially the joint state is given by far 0 up approach is possible from the train and controller. So on an approach both the train and the controller change state.

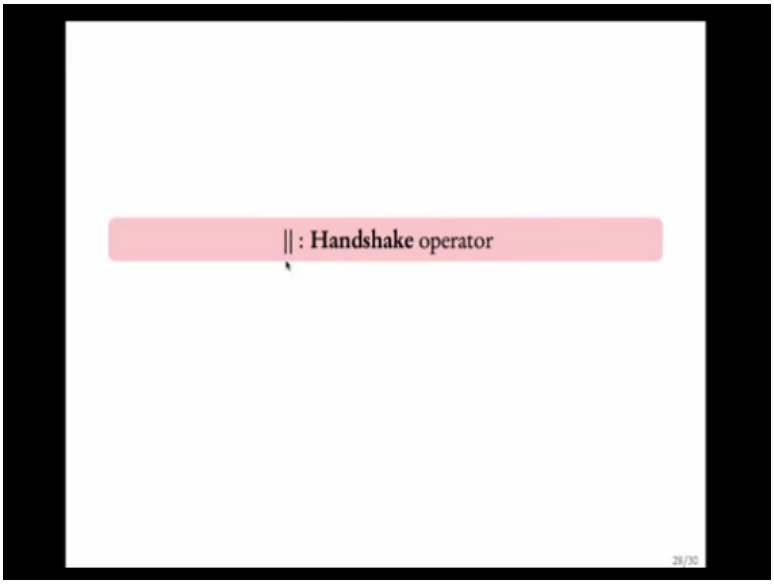
The train goes from far too near the controller goes from 0 to 1 this is the handshake action. We say that the train and the controller handshake on the action approach. In the state near one up

when the train enters it goes into the state in one ups enter is not a handshake action so it is possible that only the train moves.

In this state the controller can play the action lower. Since lower is a handshake action between controller and gate both the gate and the controller change state and the new state is given by near 2 and down. Similarly we can finish this transition system. Note that the state in one up describes the fact that the train is in and the gate is still up this is not a safe state.

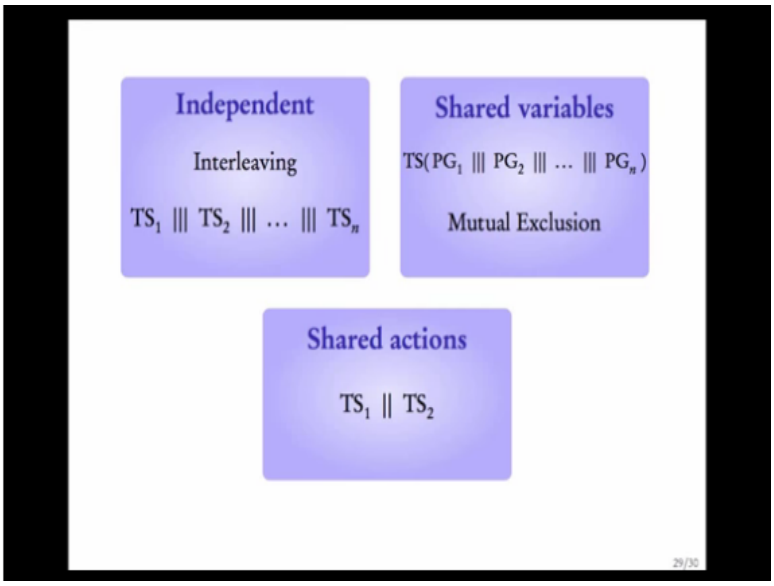
Therefore the simple design has a major flow we will see later during the course that we would have to incorporate timing details into the transition system. For now let us understand that this transition system represents the handshake composition of train, controller and the gate.

(Refer Slide Time: 41:22)



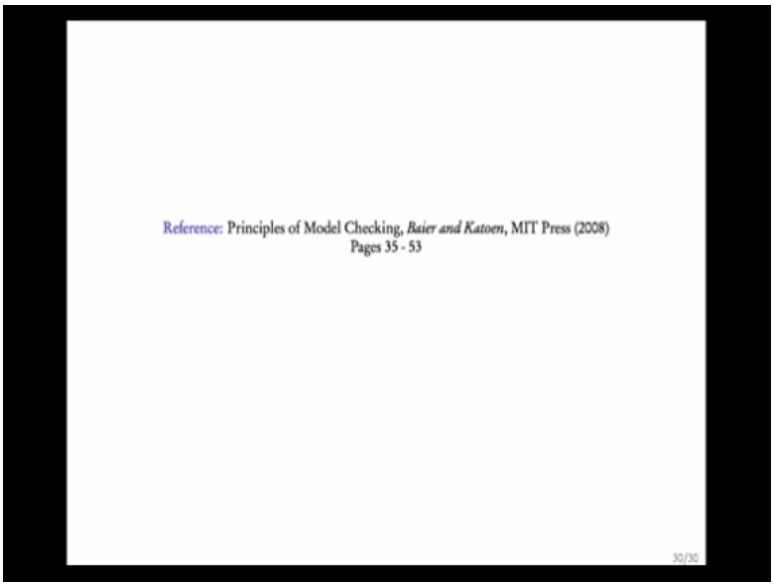
This operator which is denoted using two lines is called the Handshake operator.

(Refer Slide Time: 41:33)



This brings us to the end of this module. We have seen three kinds of concurrent systems independent concurrent systems with shared variables and concurrent systems which shared actions. For independent systems we define the interleaving operator for those that shared variables the interleaving operator was used on the program graphs and we looked at the transition system corresponding to this program graph. For concurrent systems with shared actions we have defined the handshake operator.

(Refer Slide Time: 42:15)



For a more detailed description of the ideas discussed you could refer the book.