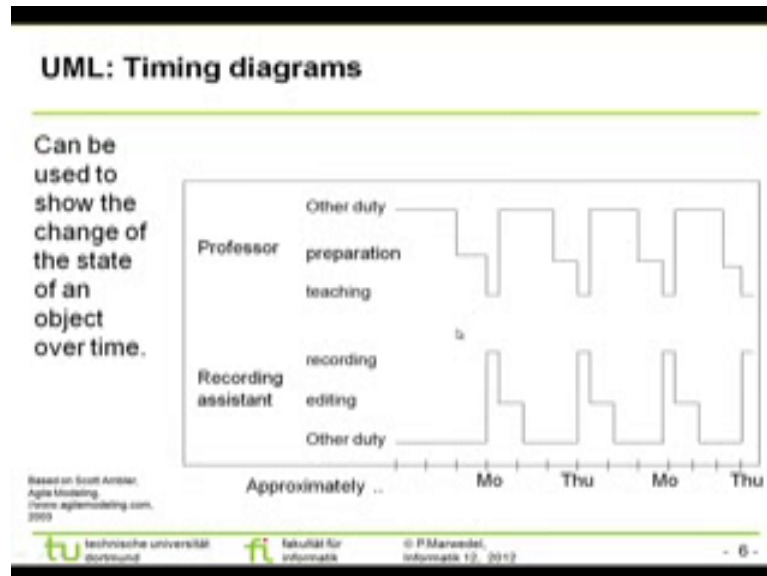


**Embedded Systems Design**  
**Prof. Anupam Basu**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

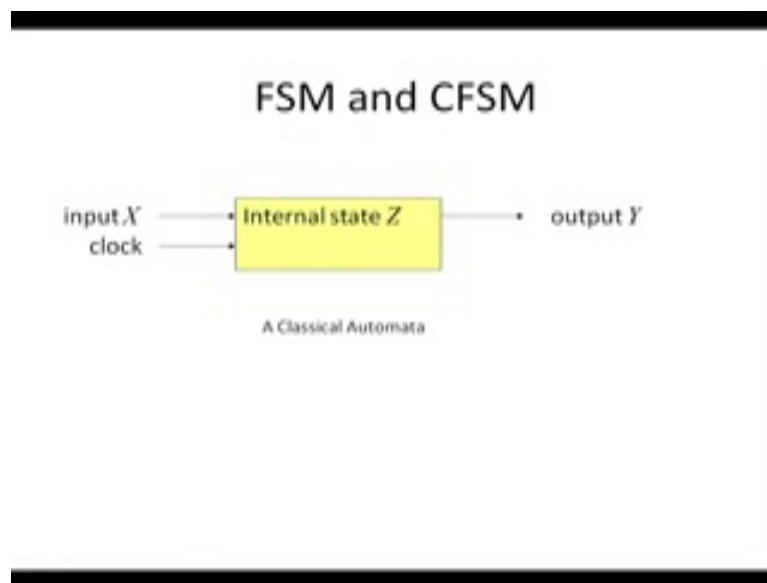
**Lecture – 33**  
**FSM and Statechart**

(Refer Slide Time: 00:34)



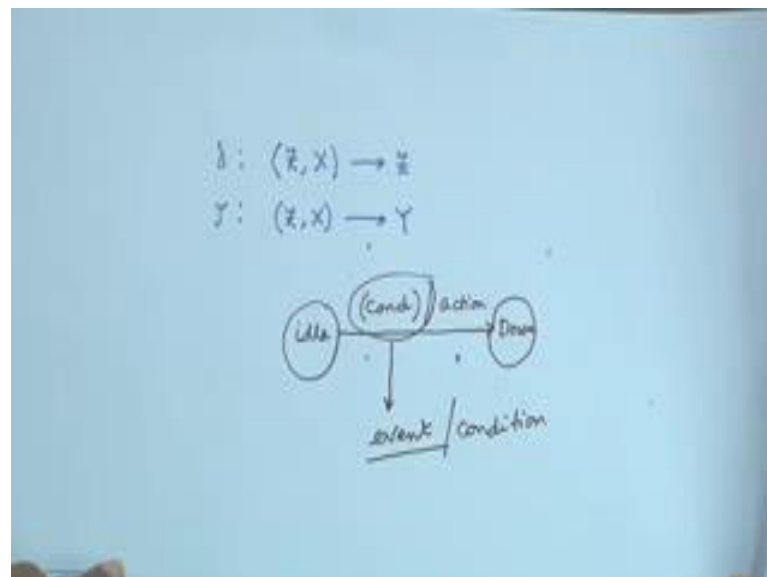
So, in this lecture, today we take up from where we left in the last lecture. In the last lecture, we were talking about this sequence charts, and how we can associate timing along with the sequence chart, which are the part of UML. Now, it has got the number of limitations also because many things are not expressed in this.

(Refer Slide Time: 00:47)



Next, we come to a very important representation which all of you know that is finite state machines. And you know finite state machine we come to the next state, the next we have got an input a clock and output and the system is in a particular state Z say and the next state is determined by the input and the state in which it is in.

(Refer Slide Time: 01:30)




So, there are two functions. There are two function say one is the let me call it the state change function delta, the state say Z is the present state and some input X is the input here we can see X is the input and it is taking it to some other state Z. And there is an

output function let us call it tau the output function is giving us some Y, and that Y can be based on the state and the input can come to Y or purely from the state we can come to Y. So, based on that we have got two different types automata, which are Mealy and Moore automata, these are this you might be knowing that in the mealy automata the output function is defined by output is dependent on the present state as well as the input. Whereas, the Moore automata, it is only dependent on the state, so so far so fine.

So, that is an FSM and we have seen FSMs all through your courses of study in your computer science. Now, CFSM stands for communicating finite state machines, when more than one finite state machine can communicate among each other. And how can they communicate among each other by we have seen there are means of shared memory message passing that sort of things.

(Refer Slide Time: 03:19)

<p>"If the elevator is stationed at a floor and the floor requested is the same as the present floor, then the elevator remains idle.</p> <p>If the elevator is stationed at a floor and the floor requested is less than the current floor, then let the elevator go down to the floor requested.</p> <p>If the elevator is stationed at a floor and the floor requested is greater than the current floor, then let the elevator go up to the requested floor."</p> <p>a) An elevator : English</p>	<pre>loop if (req_floor = curr_floor) then direction := idle; elsif (req_floor &lt; curr_floor) then direction := down; elsif (req_floor &gt; curr_floor) then direction := up; end if; end loop;</pre> <p>b) An elevator: Code</p>
---	---

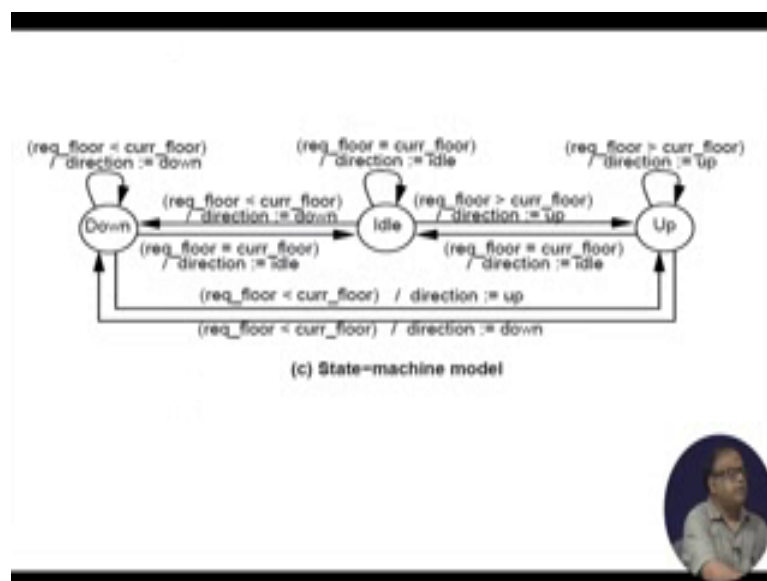


So, with that FSM, let us start to look at some specification with FSM in mind. Look at this read this, left hand side, where we are trying to describe an elevator. And you are going to design an elevator controller that is your objective. So, it is written that if the elevator is stationed at a floor some floor and the floor requested is a same as the present floor then that elevator remains idle, fine. If the elevator is stationed at a floor and the floor requested is less than the current floor then the elevator will go down to the floor requested. If the elevator is stationed at a floor, and the floor requested is greater than the current floor then the elevator should go up to the requested floor, so that is clearly that

is written, but it is not possible for a machine on automatic design system to understand that is one point.

But there are some other issues here; we are not mentioning anything about the timing, the speed that apart. Also if at the same time both of them I mean from some floor you request the higher floor and some floor you request the lower floor, what will happen, all those things are not enumerated here. The same thing can be described in the form of the code right code simple code, it is a loops a continuous loop, if the required floor is a same as the current floor then direction is idle no direction, else if request floor is less than current floor then direction is down. Otherwise, if requested floor is greater than the current floor then direction is up end if end loop. So, there are two commands either go up go down or idle, three situations.

(Refer Slide Time: 05:32)

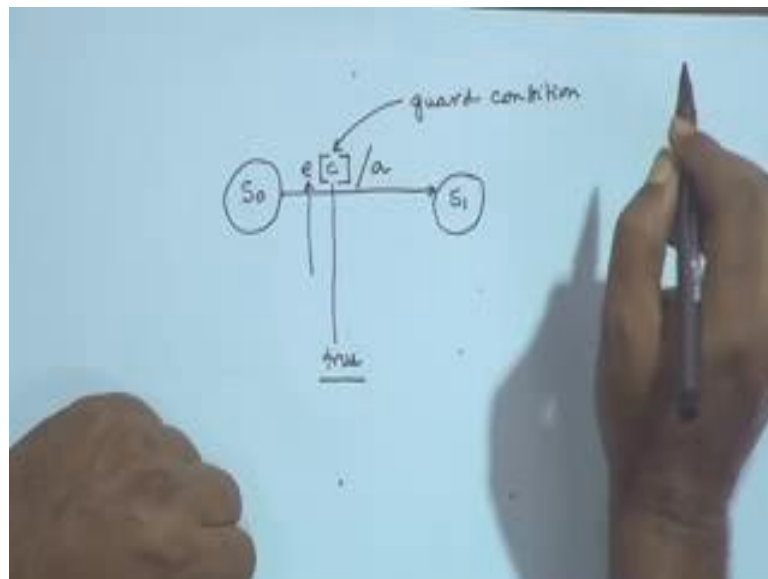


Now this thing you have to understand and the same thing I can put in more information, if I just draw it in the form of finite state machine. Let us try to look at this finite state machine a little bit. There are three states idle, either the controller is in three states either is idle the elevator is not moving up or down, or it is moving up or it is moving down. Now, when will it be in which state? If it is in the idle and the requested floor is if the current requested floor is less than the current floor lower than the current floor then the direction is down. Now, you look at this particular hash here. So, what are the things

that we are showing in this diagram, one is that I am in a particular state for example, I am in idle state, and the transition has got some condition here.

What is the condition? The condition is that the requested floor is less than the current floor. On that condition, this is an action, the direction is down that is an action. So, some control signal goes, now do not confuse with this down written in small letters and this down because this down is a state, the control signal goes that the direction should be down and the state is that is moving down. So, this is a very important part of any FSM transition and this is an event or a condition which must be satisfied.

(Refer Slide Time: 07:37)



We will see later that we can also have state machines, which are have got some state  $s_0$  and I am moving to another state  $s_1$ . And there is some event  $e$ , and some guard conditions like this followed by the action. Here, the meaning of this is that if the system is in state  $s_0$  and then event  $e$  occurs, this transition will take place only if this condition  $c$  is true. If this condition does not hold even if the event  $e$  occurs, this transition will not take place and consequently this action  $a$  will also not be taken. So, this  $c$  is called the guard or the guard condition.

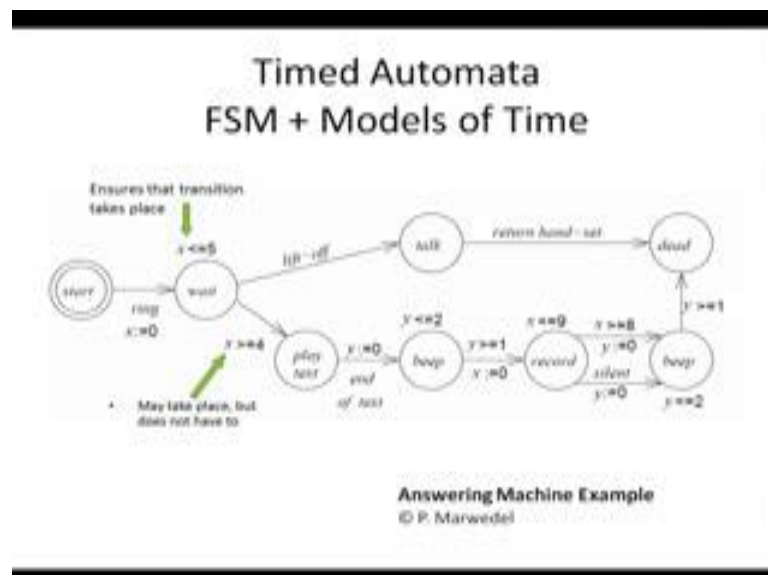
Now, going back to our elevator example, it was in state down, and then what can happen, at assuming that there is a synchronous clock, it is checking that the requested floor is still less than the current floor, so it is being going down. If the request floor is a same as the current floor that means, it has reached the desired floor, then it will come to

idle, direction will be idle sorry the state will be idle. Now, again from idle if the requested floor is higher than the current floor, it will start going up and the same thing continues.

So, here from you can see that from up the system was in the state of moving up and the requested floor is less than the current floor then the direction is moving down. Now, you see this scenario that while it is moving up if a request comes for a lower floor then whether you will put the direction to down or not, now there is no condition here. So, here it is a point that is moving up right, the state of the contrary is up; at that time, I get a request our floor that is lower as this FSM is saying it is going it start moving down. These specifics were not specified all over here, because this was purely sequential.

Now, the merits of FSM are that it represent simple the systems temporal behavior very explicitly no issue in that. It is suitable for control-dominated systems, because it was by the events that take place. Demerits are there is lack of hierarchy and concurrency. As I go on moving for a complex, relatively complex system, the number of states will explore and the number of edges will also explore. So, it becomes very difficult for the designer to really get a feel of what being intended and what is going on.

(Refer Slide Time: 11:31)



Now, I come to another model little bit of extension of finite state machine or automata. We are calling that timed automata. Timed automata is FSM class we are capturing the models of time right. So, here is an example taken from copyright is Peter Marwedel's

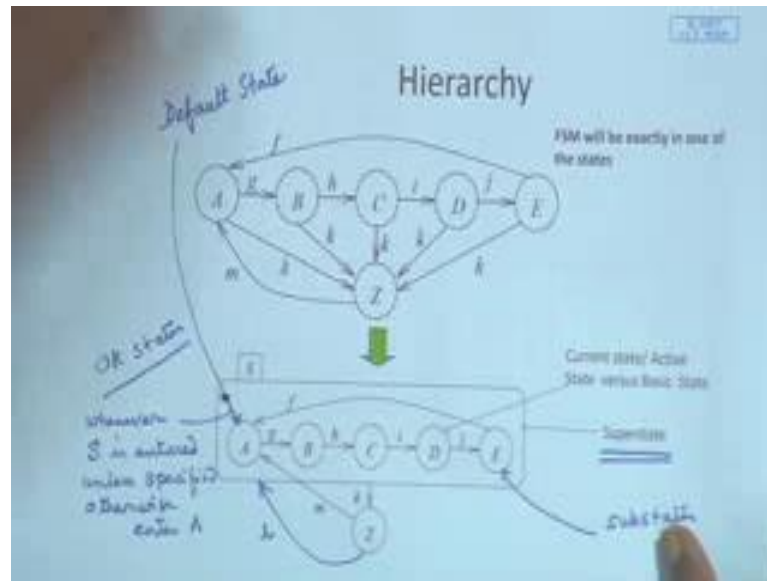
for this. We are starting this is an example of an answering machine, telephone answering machine. We start and the ring starts. Now, when this rings starts  $x$  is a variable that is denoting time and whenever the events starts, this  $x$  is reset. So, here  $x$  is 0 and the clock is ticking. So, setting  $x$  0, so just like an action on the event ring I come to wait state all right.

Now, in this wait states, I am here it will remain in this wait state as long as 5 units have not elapsed. If 5 units are within 5 units events can take place; that means so I am giving the time right the person picks up the phone, lift-off, talks, returns the handset, dead fine. If the person does not lift-off then as four units of time elapses as soon as four units of time elapses, it starts playing a text, the answering machine will play a text mister so and so I am sorry, I am not at home right now, you have reached mister so and so I am not at home right now please leave a message after the beep etcetera, etcetera. So, it plays the text.

And then you see another time another clock is set to 0, as soon as the end of text is given. Then the beep is there and the beep is existing is therefore some duration. And then I start it has already become at that time, I start recording. And when I record I again set the clock  $x$  to 0 and the recording time length can be up to 9 units, whatever that is. Now, after that 9 units, I can go silent with the beep or if the person has not been silent then the time has elapsed, I will give a beep and it will be dead. So, along with this we can put in time as a variable along with FSM that is timed automata that is another way there are different other ways also of representing time and capturing the temporal behavior of this of embedded systems, but here is one example we are showing.

Next, I come to very important concept that is of state chart. Now, state chart, why state chart required? State chart is required because it will provide us with the facilities of abstraction of hierarchy and concurrence both. Now, one major problem that we have already stated in terms of finite state machines is that as the number of states go on increasing the thing becomes incomprehensible.

(Refer Slide Time: 16:17)



On the other hand, if I represent finite state machine like this as this being shown here, this is a finite state machine with 1, 2, 3, 4, 5 states and there are so many edges coming out from A to B on event g; from B to C on event h like that. And from any state if event k occurs, I will come to another particular state z, forget about this part right now. So, and from z we can come back to m. So, consequently there are number of states and if the system becomes more complex, there will be larger number of states and larger number of transitions. One thing to note is that the FSM will be at any point of time will be exactly one of the states. It cannot be simultaneously more than one state.

Now, you see that we can employ a hierarchy over here; we can say that this is one state, this whole thing is a state, and Z is another state. We are now abstracting out and giving less details, we are giving more details in faces. So, I first say that I have got two states here. What are the states S and Z, and what happens if event k occurs then I exit from state s and come to state to state Z? If event m occurs, then I leave the state Z and go to state A. Now, here we are having certain things to be noted that as I said that at any point of time, it will be either of these states if the system remains in either of these states in either if these states then it is in this state in either of these states in A or B, or C or D or E, then it is in state S. So, these states are the OR states. Now, this is known as, this S is known as a super-state. And these ones A, B, C, D, E are known as sub-state.

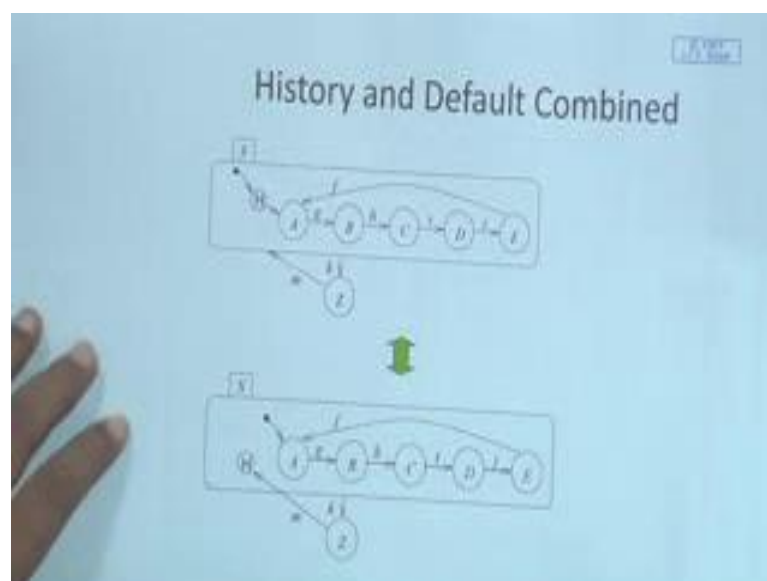


Now, while in the super-state the system can be in any one of the sub-states. Now, what is the current state, a current state is if it is in A, B, C, D, E anywhere then it is in a current state then that OR anyone. So, for example, it is now the machine is in D then that is if the current state of the or the active state, and also this whole thing is a current super-state. So, I have got super-states and sub-states and I have got this OR states.

Now, there is another thing called the default state or entry state. Whenever I am entering S, now S comprises of 5 sub-states. Now, if here I am showing explicitly that if event m occurs, it goes into particular to the state A. It is possible that on some other event say k, k was already here know k is here, so some other event may be l it enters here, I show it here. What does this arrow mean, this arrow means that it goes to this super-state, but I have not specified where in the super-state it goes in. For that we usually denote the default state for example, like this a solid circle with an arrow; that means, whenever this means, what does this mean, this means whenever S is entered unless specified otherwise enter A. So, this is known as then A is the default state or the entry state, all right clear?

Here you please distinguish between these two that I can enter and specify directly that I will go back to this state or this state or this state or I can simply come to the super-state entry point. And say that I am entering somewhere in the super-state. I am not interested in specifying that the responsibility of specifying which is the super-state of S is up to the person who is designing this part. So, thereby we can modularize the tasks also right.

(Refer Slide Time: 23:50)



Now, we come to another point that is a history mechanism. Let us leave aside this, and try to concentrate on this part. The same state chart is being shown here. And here let us look at this, what does this signify. This signifies that it is when S is entered, A is being started then on event I am going on this as my plane is moving I am going on like this. And at some point of time an event k occurs, an event k occurs. As the event k occurs, I come out to Z. Now, do you know that when event k occurred where was I, I could have been what does event k tell you, event k tells you that you are coming out from state S you are coming out of state s, but you do not know which sub-state you are coming out from.

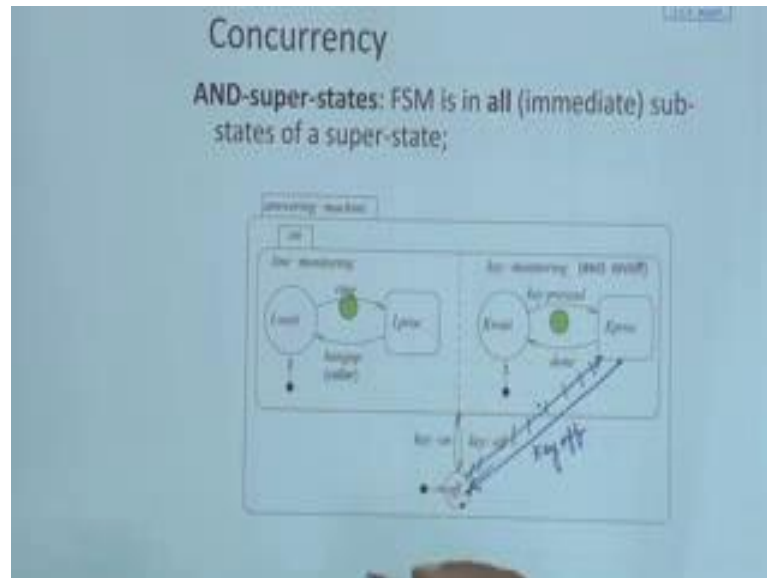
So, if I draw like this that on event m, it goes to a node H, H is nothing but a history node. As if it is keeping a record of the different states through which the machine is traversing I am remembering that. So, if I specify in this way that on m while I am in Z, I will be going to the H node. What does that signify that I will reenter S through the state from where I left suppose I left from here, actually when k happened I was here. Then H will actually take me back to this state C from where I will continue, clear?

I repeat suppose I came out, I am specifying that where ever you are here on k, you are coming out to state z. And on m, you are reentering state S, you are reentering state S. Now, suppose what does it signify. Now when you actually k occurred, suppose you were in state C, so you came out. Now, this H node will remember from where you came out where you were at the last point of execution. So, when this specification tells me that well you go back to s, and follow your history node to find where you were, then this history nodes tell me well I know I was in C. So, when I entered this, I will not again start with A, I will start with C clear? That is the role of the history node.

Now, can we combine this history node and this default entry node? Now, here as this diagram tells shows you that the history and the default can be combined in this way that when I enter m, I will go to S, I will enter S, and I will go the default node. The default node will check the history node if the history node does not tell the f any other node from where I exited if the history is the empty then I will come to A to my default node; otherwise this will direct me to this. You see state chart is the graphical language. So, whatever we are stating here are all stated through graphical diagrams, which are taken by a compiler and converted into some executable form. But when you are drawing this your drawing must have a unique meaning, the semantics must be unambiguous that is

why these meanings must be made clear. Now, we will come to so we have seen the hierarchy and how the hierarchy can be maintained.

(Refer Slide Time: 29:14)



Next, we come to the issue of concurrency. Again we are taking the example of an answering machine. And let us try to understand this answering machine. I assume that all of you have got some idea about what an answering machine is. An answering machine is connected to a telephone as the phone comes in either you pick up or after sometime the control goes to the answering machine, and the answering machine answers you, etcetera.

Now, that thing should be irrespective of my explanation. When I am specifying an answer machine, the behavior of the machine should be clear from this diagram. What is being shown here, again look at the super-states and sub-states. This is the super-super-state, which is the answering machine as a whole. Now, this answering machine can be in on-state or can be in off-state; in on-state or off-state. Now, from off-state, now when the system enters by default in this block - entire block, here is the default entry node it comes to the off-state. When I put the key on, it goes to the on-state. So, off and on are these the AND states or OR states, these are OR states because I am either in the off-states or in the on-states.

When I entered the on-state, because I have put the key on, then this is the behavior. This behavior comprises of two other states, which are separated by this dotted line. This

dotted line means that this state and this state are concurrent, they are not sequential both of them are going on simultaneously. What are these states these state is a line monitoring state where I am checking the line where is the some call is coming or not. And here is the key monitoring state where excluding on and off, because on and off are being done with this switch on main on and off switch besides that, these are telephone keys or something. Now let us see.

So, when I enter key on when I when this transistor key on takes place, I enter this on-state, where do I enter the on-state I enter in both. So, both of them have got their defaults states specified. For the line monitoring state, I am entering in the sub-state L wait. Now, you see line monitoring and key monitoring at the hierarchical at the higher level. If I decompose them further then I get the states like line wait and line process, key wait and key process this things. So, I enter this line monitoring through line wait, I entered the key monitoring through key wait, thus the next level of hierarchy. Next level of hierarchy tells me what is the behavior of the line monitoring and what is the behavior of key monitoring.

Line monitoring, it waits till the event ringing of the phone occurs. Ringing is occurring then you process the line. What you do I am not specifying at this point of time I can specify at a lower level of hierarchy where I can say connect to the phone raise the receiver, talk, etcetera, etcetera that was shown there, but all those details I am hiding that is why the abstraction is so beautifully captured in state charts. And from line process, whatever might be the details of the line process, when I hang up - the caller hangs up then I go back to the line wait or I keep the phone down and also go up go to the line wait.

Now, in the key monitoring, I am waiting for the key to a take place, I mean I am waiting for some key pressing, and as the key is pressed, I go to the key process I mean key processing. And when the key processing is done, I go to the key wait. Now, what are the typical key processing activities we have seen in earlier lecture with an answering machine when we are throwing that UML chart or I mean before that UML we are showing some use cases that the keys can be record message, read message, delete message all those things. So, that is hidden inside this block whatever processing I am doing. So, this is the top-level, now this known as AND-super state.

Why is this known as AND super-state? It is known as AND super-state because when I enter on-state, I am simultaneously in line monitoring and in key monitoring both of them are active. So, the FSM is in all immediate sub-states of the super-state. It may be that in its in l wait and it is in k wait fine. Look at my fingers, I am right now in l wait and k wait, but; that means, I am in line monitoring as well as in key monitoring both of them. If I am in line l wait, and I am doing some key processing no phone has come, but I am just recording my messages or whatever, then also I am inline monitoring as well as in key monitoring. In this big picture, line monitoring and key monitoring, I am within that both of them I am in, so that is active. If I am doing line processing and I am in key wait then also both of these are active. So, it is in all super-state, I mean sub-state of the super-states this is super-state in these that is why it is called AND-super-state.

Now, how can I do some I mean entering and leaving this? Say for example, from key processing I do somehow I was here, I was here, I was in k proc. And then I have pressed the key key-off, I have pressed that then that will immediately bring me to this off-state. And when I press key on, it will take me to the key processing state.

Student: (Refer Time: 38:08).

Very good. So, when I do key on, when I has been pointed out since history is not there, I am not been able to proc, I do not really know, where I should go, therefore I should simply enter key on I should go here. So, I did not I could go here provided there was a history node. If there was a history node, I could have gone here since the history node is not there I cannot go in there I will take this path, I will take this path. So, on key off, I am coming over here. So, entering and leaving the AND super-states, so whenever I put the key off I am coming to off that means, I am exiting what? What am I exiting; I am exiting the on super-state.

As I am coming out as the arrow shows, I am coming out of course, key processing sub-state, but the same time I am exiting the super-state all together. And when I am so then my line wait also goes. So, I can further go in on key on, and I can start from here. So, so much for now, we will have to continue with this state chart there are some more things that we have to discuss that we will do in the next lecture.