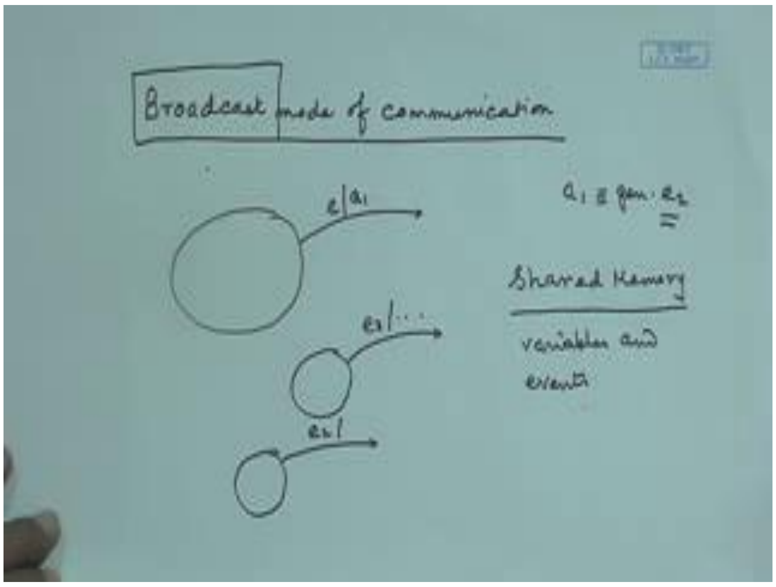
Embedded Systems Design
Prof. Anupam Basu
Department of Computer Science and Engineering.
Indian Institute of Technology, Kharaghpur

Lecture – 35
Statecharts (contd.)

So, we have seen state charts, the history mechanism and how state mate evaluates a state chart. Now, all through our discussion, we have assumed that events are being generated that event is being evaluated in some way with conditions, but how are these events passed. There are so essentially since it is a communicating finite state machine, there are multiple finite state machine, they are communicating among themselves. So, we know that two modes of communication are popular; one is message passing, another is shared memory.

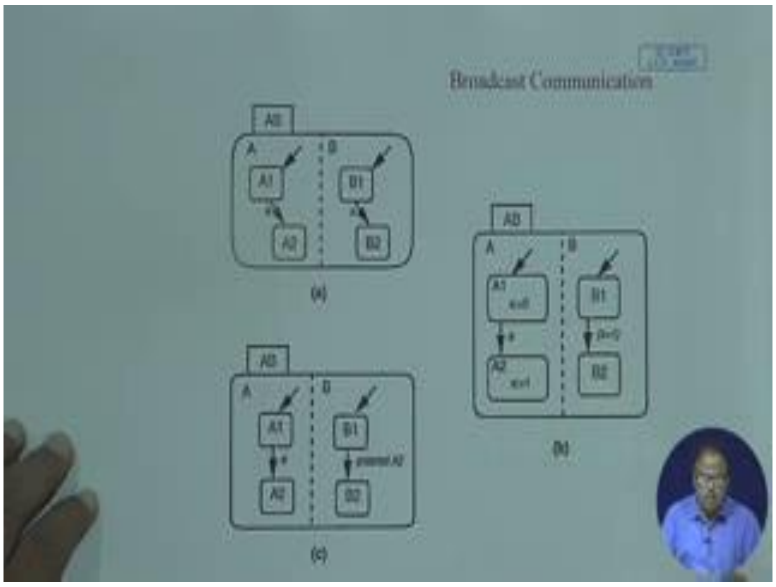
(Refer Slide Time: 01:13)



Now, in state chart, we use broadcast mode of communication in terms of implementation. What does it mean? It means that if from any state I am in some state and some event I generate suppose this was an event, and I take some action, and this action a 1 is actually generate an event generate event e 2. Then this e 2 event e 2 is visible to all other states, which might be waiting for some other event e 3 taking some action, some other state which is might be looking for e 2 some other state which is not looking for the e 2 it will be visible to all.

That means, what it is a shared memory mode of communication just like a global memory right not only for the events, but also for the variables for the variables as well as the events whenever there is a change or in generation of any event that visible to all. So, when the state mate is executing then it can see all those variables is looking like a global variable set right; events and variables are being treated in the same way, so that is very important aspect of the state chart that is it is a broadcast mode of communication. We will later on see that there are other modes of communication as well which are not broadcast mode.

(Refer Slide Time: 03:30)



So, here is an example of a broadcast mode of communication say here let us look at this there is a state A B. So, A and B are two end states that means, both of them are active. Now an event e takes place here and that is visible to this one also, therefore that event is also visible here and both of them make a transition. So, if at the particular status, remember the status, step, status, step then suppose in a status is A 1, B 1 that means, here in A I am in A 1; in B, I am in B 1 and event e occurs then both of them are visible. So, in that step, what will happen, we will compute the event e and we will see what will be the next step. In the third step of execution, our status will be A 2 B 2.

Here is another example say I am in state A 1 and an event e takes place. So, I have entered A 2; this entering A 2 is again an event, because this is being treated as an event here, you can read this that is entered A 2, therefore, B 1 suppose my initial status is A 1

B 1, then e has taken place. Then in that step, what will happen? In that step it will be e to my state will change to A 2 and A 2 to B 1 will be the status. And as I entered A 2, my status changes next cycle it will become A 2 B 2 clear. Here is a third example, here is no longer in event, but data you can see that here I am again my state is A 1 B 1 and x is 0 on an event e I come here.

So, the next status is A 2 B 1. Here I carry out some assignment still that is not reflected in the next cycle, it will be see in that here, it x is equal to 1 and my state will be A 2, B 2 because this event on data x 1, so that is what is meant by broadcast communication. And there is also in way handling with the synchronization which will see soon.

Student: (Refer Time: 06:41).

Pardon.

Student: (Refer Time: 06:45).

See, but oh, B and C, these two are equivalent.

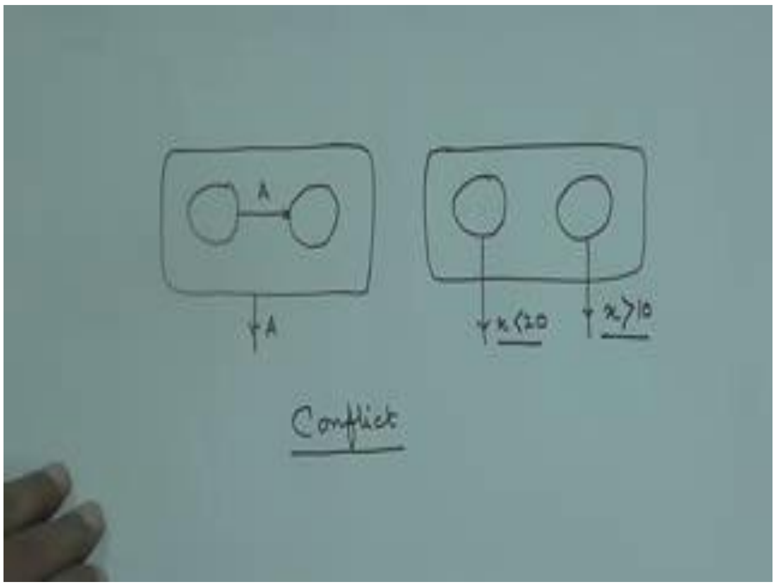
Student: (Refer Time: 06:52).

No.

Student: (Refer Time: 06:54).

No, no just a minute, just a minute. Here what is being shown is that the event of entering A 2 will enable this transition. Here this x is being set to one may be internal within after some actions have been done in A 2, then this is being set and as this is being set then it will be done in the next cycle, this one will be enabled. So, they are not exactly equivalent; one is on entry, another is on assignment.

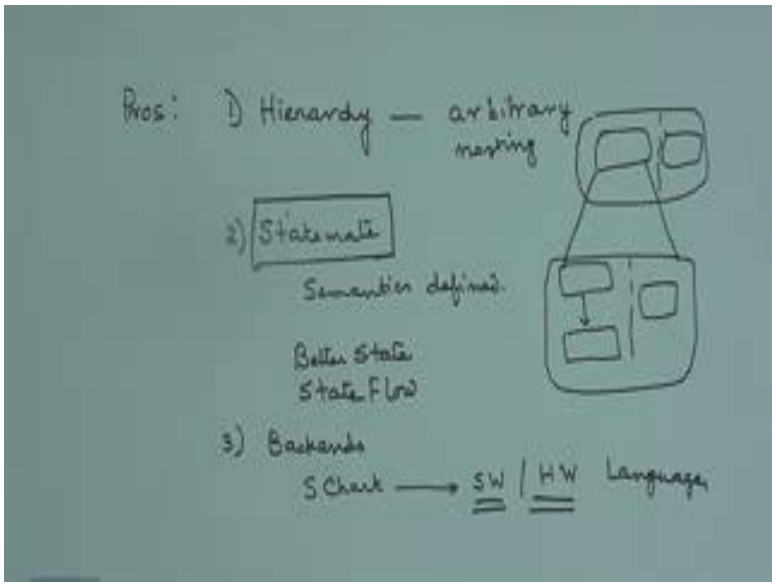
(Refer Slide Time: 07:43)



The other example another issue that comes up with state chart is a conflicting scenario like this. Say I am in a particular state, where on A is shown that I am making a transition and it is also shown this. What will happen in these scenario? If that is a specification then this specification is ambiguous, but state mate does not stop you from making such specifications. If this specification is done then we are cannot guarantee that the implementation will give you determinant result. Because some implementation can take a precedence of an internal transition, another can take a give the precedence to n external higher-level transition.

Similarly, another example could be that I have got to states I am saying x is less than 20, you make this transition; and here I say x is greater than 10 you make this transition. Now, suppose these are concurrent transitions or whatever. Now, obviously, we can see looking at these values x is greater than 10 will certainly do this, but now when x is 15 then this can also take place. So, these are some conflict scenarios, which do not allow us to ensure to guarantee that state chart will always give me deterministic or determinate results.

(Refer Slide Time: 09:54)


Pros: 1) Hierarchy — arbitrary nesting

2) Statemate

Semantics defined.

Better State

State Flow

3) Backends

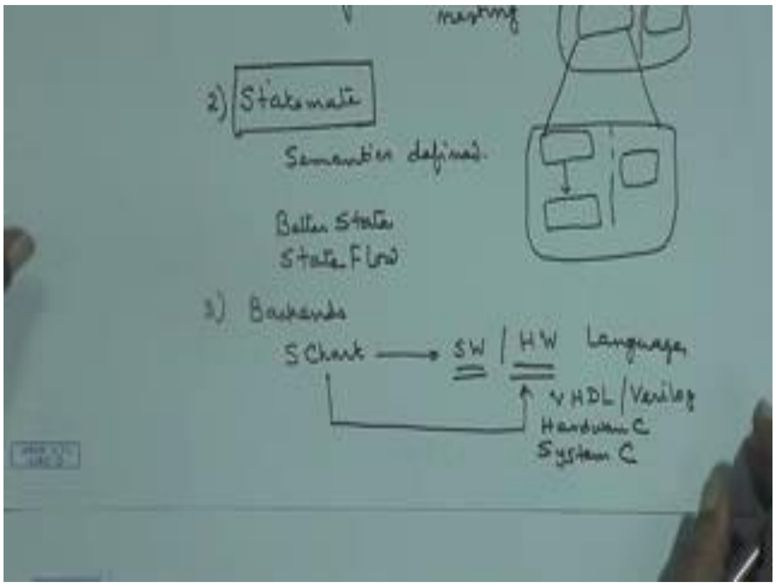
S Chart → SW / HW Language

Now, what are the pros and cons of state charts? The pros are number 1 is hierarchy; hierarchy allows us arbitrary nesting there is no limit on that I can go on nesting one state, and there can be two states under this. And this state can be further decomposed into two sequential state and another state, all these things are possible any arbitrary nesting is possible. And thereby we can reduce a complex scenario into comprehendible specifications. We can look at different phases and can understand it much more nicely and proceed accordingly.

The second thing is state mate, a tool is available; and that state mate has got very clear semantics defined, very clear semantics is defined, therefore, I can have an executable specification. There are others like state mate, state flow, better state, there are others also say better state, state flow all these are commercial tools, but state mate semantics is very nice and very clearly researched.

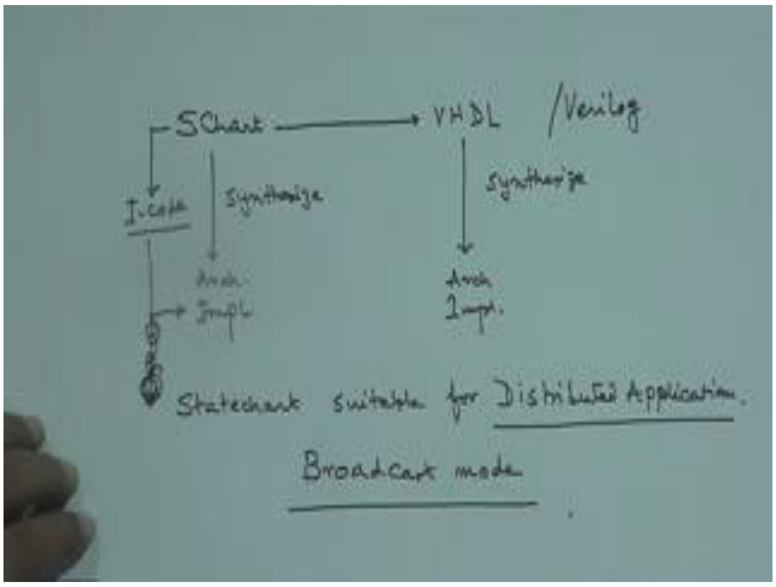
And we have got backends that is very important we must have backends that will take our state chart and translate it into software or hardware languages. Why did I say software or hardware languages, what is the meaning of this software and hardware languages by that I mean state by state chart is not the case that I can use it for specifying embedded system alone; I can specify any complex software also using state chart. So, from that, I can generate the software by some automatic software generation tool. The specification of the language state chart can enable me to specify complex software as well also hardware embedded hardware embedded hardware as we have shown a seen the answering machine example we can have many such.

(Refer Slide Time: 13:12)



And state chart can therefore be translated into some hardware description languages as I said that like VHDL - verilog, there is a hardware C, there is another very powerful popular language system which are variants of C with some special constructs. And from here let me spend a little time on that.

(Refer Slide Time: 13:52)

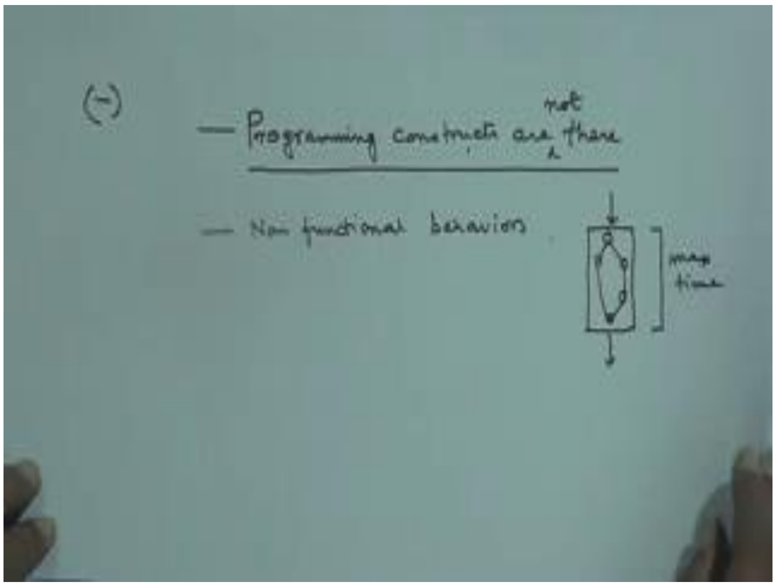


So, what can happen is that I have got a state chart specification. I can directly synthesized from state chart to some architectural implementation that is may our objective or I can compile this state chart into VHDL set or say system C. And I have got some cat tools which can take VHDL and I can have the implementation through synthesis. So, I can do that is an advantage. And another important advantage is state chart is very suitable for distributed applications. For example, you are working in a automobile industry, and want to develop a some embedded solution, and there might be distributed processes at different corners of your car which will be communicating and some event occurring here should trigger some other event occurring somewhere else that can be easily captured in state chart. Why? Because state chart uses the broadcast mode of communication that is the very salient feature of state chart. So, therefore, using this, we can very nicely capture distributed applications.

If you have got some cat tool now from this state chart what will happen, it can come to some intermediate code you need not you have got some tool say tool 1 say k dense tool, which take VHDL and produces architectural implementation or verilog and from there does it. Verilog can be used in k dense actually, I think verilog is used. But here suppose from state chart from this graphical thing, you can generate some intermediate code yourself and you can design a synthesizer which can take this and not this commercial tool k dense or synopsis. But you can have separate tool which will take this intermediate code generated from state chart, and we will generate the synthesis that is possible. Because whatever you are stating here is a language through a language you are specified the requirement and you can show that it into the form of a graph and from using that graph you can do the that that is possible.

Now, what about the cons of state chart? It is I just now said that it is useful for distributed applications, but some people are contesting that often that is not very much applicable for distributed applications although broadcast mechanism is there.

(Refer Slide Time: 17:59)


(–)

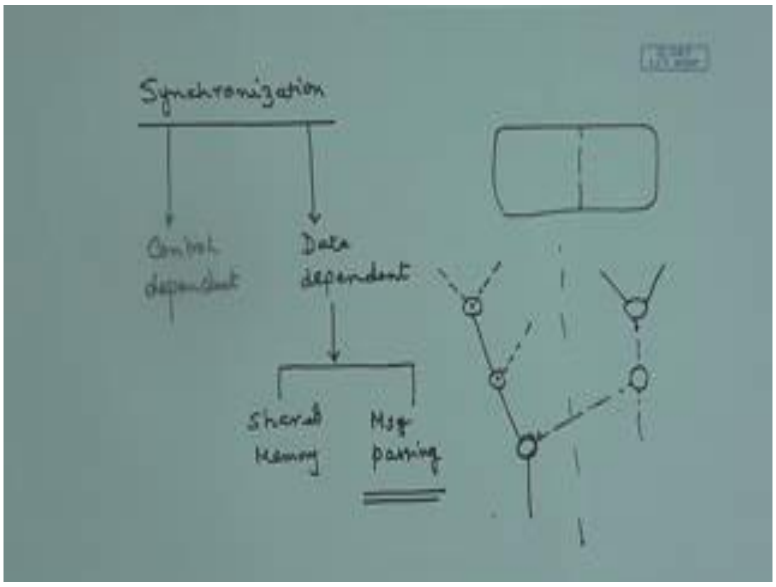
— Programming constructs are not there

— Non functional behavior

Therefore, we are not hinging much on that, but one of the major cons of negatives of state chart is no programming constructs are there, programming; programming constructs are not there; it is all graphically we go on bringing down to the state machine level and everything I have to describe in state machine level. But say for example, I write a sorting program, now it is much easier, suppose I am doing some sort or I am doing some finding the max or whatever that particular thing can be better expressed using programming constructs than using state machine, by state machine it becomes really very cumbersome and meaningless. So, that is the limitation big limitation of state chart.

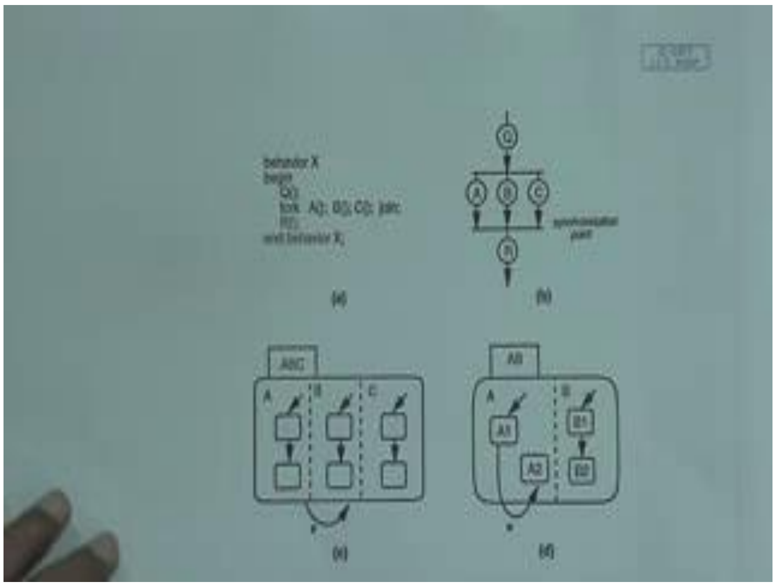
And non functional behaviors cannot be stated. For example, I say that this entire part of the task must be completed max within this time. We have got timer constructs saying that is the delay or how long it can be here. But it is a non-functional not allow elicited the behavior that this entire task, this entire task graph maybe, we will have to be completed within say 10 millisecond that is that specification is not embedded in the state chart it can kept in a separate file. And the generated programs when we generate that may be in sufficient those thing are less important, but one of the major thing that we will try to circumvent let in a later phase may be in today’s lecture if time permits that the programming constructs are not there.

(Refer Slide Time: 21:01)



Now, we have seen many desirable features of our specification languages. We have already taken care of through state chart hierarchy and concurrence. But there are some other issues which need to bother about one is synchronization. Now, because the concurrent processes, we are showing fine that these are processes are or the states or whatever actions these are concurrent, but concurrent processes are often not independent. They run concurrently up to a level, but at some points they need some hand shaking or they need some I mean synchronization. Now, there can be two types synchronization one is controlled dependent synchronization, another is data dependent synchronization. So, I will show you an example of different types of synchronizations.

(Refer Slide Time: 22:18)



So, here is a controlled dependent synchronization example here is a behavior X some behavior this behavior. Now, one is by fork and join we know fork join construct you know from your operating system course beq fork A, B, C join; in this language structure we can do that like here. So, this is a graph, where using fork and join, we can create processes and then we can join here. Let us look at synchronizations, how synchronizations taken care of in state joint how we can achieve that. This behavior A, B, C you entered this; that means, you enter this you enter A, enter B, and enter C.

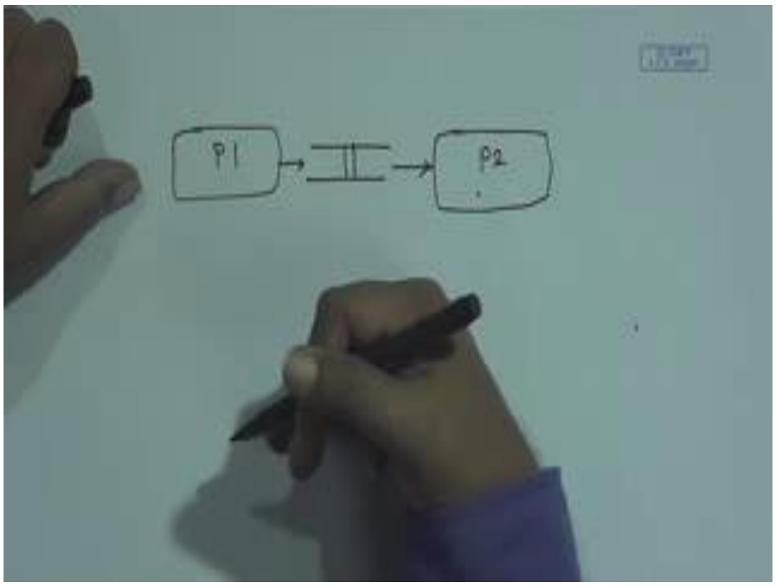
Now, when an event e occurs, now these two I have got no control over the relative speed of this process. So, at a particular point of and also I do not know about these events. So, at a particular point of time, it can be ABC can be in state A 2, B 1 C 2 or anything, but I want to synchronize them, and want to bring them to A, B, C. So, if the event e occurs that is I am exiting from the super state and reentering super state; that means, again as I reenter I am going back to this A 1 B 1 and c 1, so that is one mode of synchronization.

Another mode of synchronization is that suppose I am in A 1, and I want to come to this one is in B 2, say I want to A 2 B 2 to be the state, and then I may or B 1 A 2 to be the state. So, if I make an exit from here, exit from here and enter A 2, then forcibly I am entering A 2 no history, I am just forcing it here and it is not default this one is a default state. But here I am not keeping it as default it is forcing it here, wherever it is B 1 wherever it might be as soon as I exited this also got exited, this one also got exited, because I exited the super state. So, it will become B 1 A because as it reenters it is default and this forced. Therefore, in that way also we can do some synchronization.

Data dependent synchronization is usually done through shared memory. Now, talking of data dependent that means, what is data dependent synchronization. Something is going on, it is proceeding nicely is the data flow graph, some data is available, and some these are operation, operations that are taking place. Now, here I want that I want to wait till here in another part of the thread some data are being produced, and I want that this will proceed only after some data is produced here. So, I can create a data dependent synchronization between these two threads. So, typically that sort of data dependent synchronization that can be because of common data, this one is using a common data that can be done through shared memory.

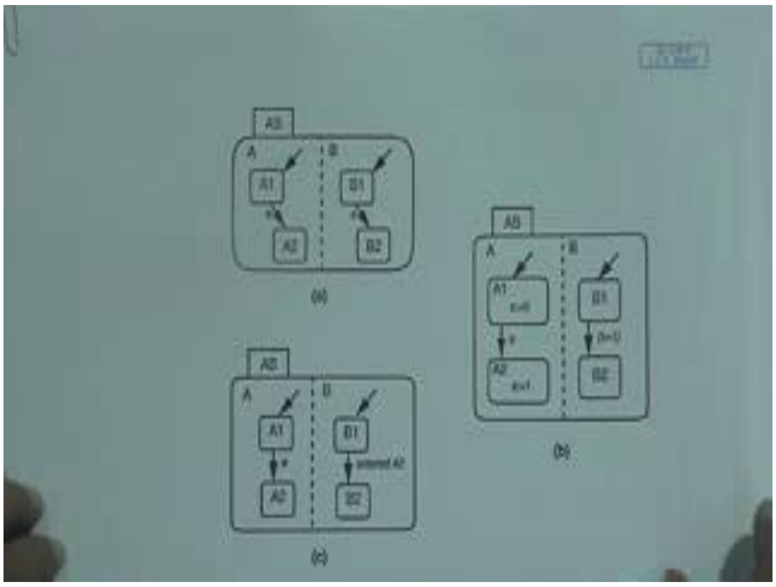
So, what is being meant by that; that means, this operation starts is suspended till the other part produces some data that means, comes up to a particular level, so that can be a common event that this data is ready I can visualize in that form or it can be a shared memory. That here I am waiting for some data that will come in the shared space, so that is possible that is data dependency can be done using shared memory. Or I can also do it using message passing will show both. Message passing can again be of two types one is blocking and non-blocking. Blocking means whenever I pass on a message that message unless until that message which is received by the receiver, the sender will not progress - proceed that has got the advantage that no data will be lost.

(Refer Slide Time: 27:36)



Because suppose there are two processes communicating through the same buffer, some queue or something, say there is one process P 1 another process P 2, and there are this is a queue and this one is sending in the queue and this one is receiving from the queue. Now, suppose this process is slower and you have send it in the queue and that is not being consumed and this is very fast; it may be the queue will be full and sooner, you will start losing data from here right. So, by blocking, what we do is once the data is consumed, I send one, send the next one, what is the disadvantage of this. The disadvantage of this is the entire P 1 is being held back, so the performance will deteriorate. Non-blocking means I will not wait for the signal that my data has been received. So, such message passing can also be used for data dependency synchronization.

(Refer Slide Time: 28:35)



Here is an example of data dependent synchronization. You can see again this actually this one we had shown earlier, when we were explaining the broadcast mechanism. So, here you see again the data dependent synchronization. One is the same event same event both of them are synchronized here some state. This one is data dependent again data, depend data driven synchronization. So, as this data comes here, I make this transition. So, this was made to wait till a process A reaches A 2 and next x to be one that is also data dependent synchronization. So, synchronization is another event.

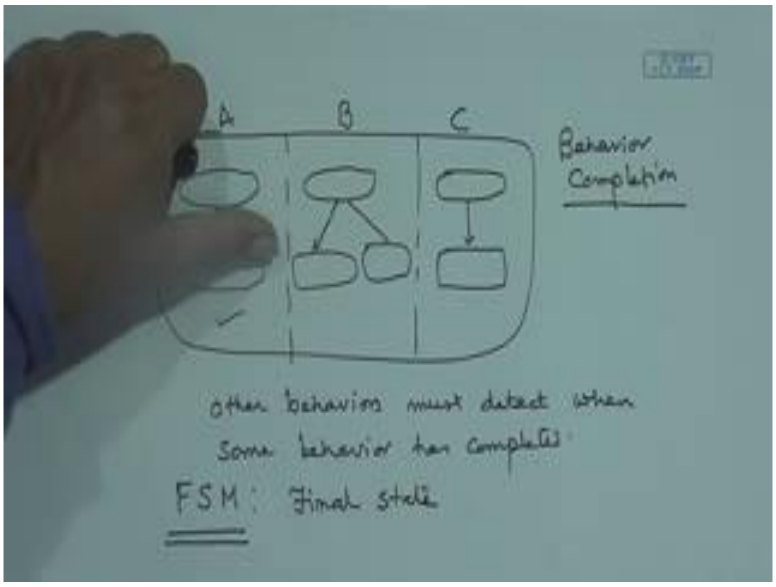
Student: (Refer Time: 29:31).

Pardon.

Student: last one (Refer Time: 29:34).

Last one is event driven, last one is control dependent this, this one is it has entered over here and based on that some event has taken place some event has taken place.

(Refer Slide Time: 29:55)



Behavior Completion

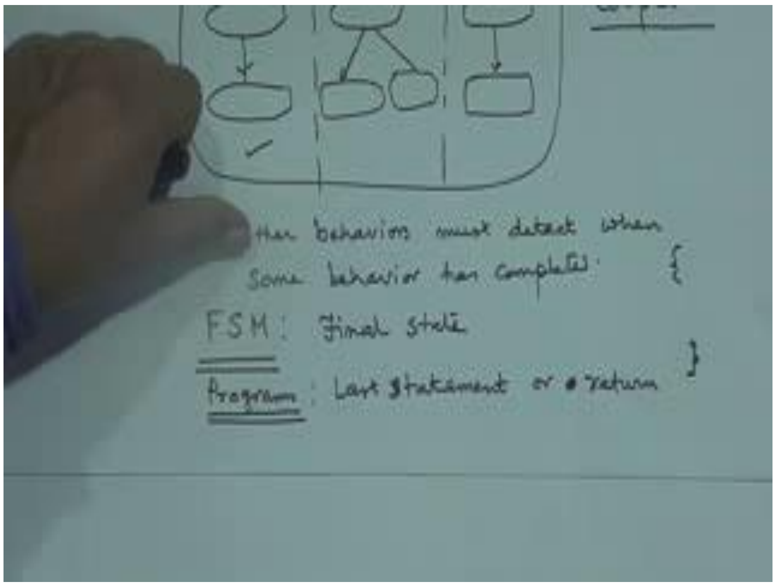
other behaviors must detect when some behavior has completed.

FSM: Final state

Another important that we need to look at is behavior completion. How do we know that? Say whenever I am actually describing a behavior in my state chart, I am actually describing a behavior that can be three different parallel processes. And this is doing something, this is doing something may be. When does thing ends when does it end that is not very clear from here? So, how do you depict when a particular behavior has completed. Each of them are behaviors, I have described the behavior as a complex behavior has been decomposed into three smaller behaviors, but when does it complete, because when one completes, it must be other behavior should be able to detect it. So, other behavior say ABC must detect when some behavior has completed.

Why is this important? This is important, because in hierarchical specification level I can view the sub systems as independent modules, each of them is an independent module, and they do not know directly from each other, and there are being developed independently. So, unless I know that this as completed, I have got know unless it is explicitly told I have got no other way to really understand that this behavior whether one has completed or not. Now, how can this be represented? One thing is in FSM, all of you know, how the completion of a behavior is shown the final state.

(Refer Slide Time: 32:22)



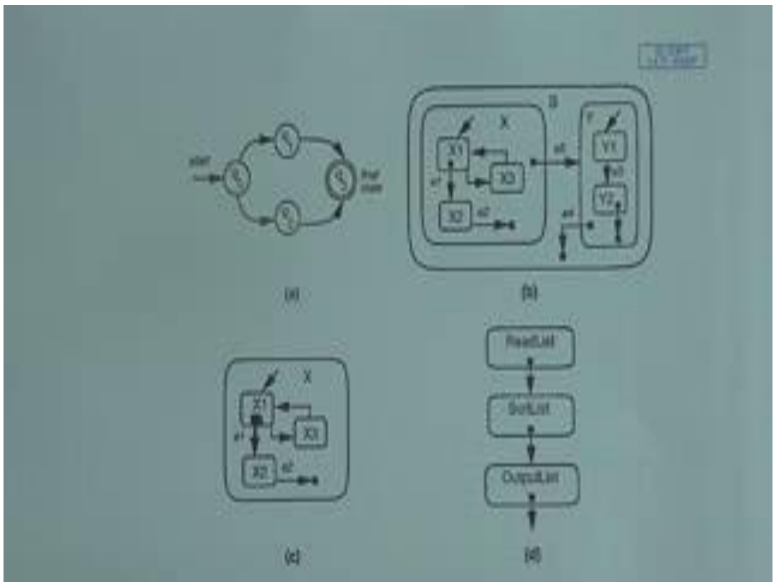
the behavior must detect when
Some behavior has completed.

FSM: Final state

Program: Last statement or return

In programs, how is it known, the last statement or return in a procedure. So or even say in C, when you give this last main reaches the end bracket, then we end, so somehow so C program and another process are going on side by side. Now whenever this reaches an end then or return then return value is sent, so at the global point, if I am continuously looking if am looking at that and I know that this part has completed, clear? And in the state chart form, we have got no scope of doing that there is another way that is I am diagrammatically showing you, we will come to this may be in the next lecture.

(Refer Slide Time: 33:41)



That the program state machine where we use there are some other features you can extend that to your state chart also that another graphical notation of this square blocks here. What is being shown in this diagram is a final state for an FSM. So, let us look at this first, this diagram looks like a state chart the hierarchy is there this one and then I break it down. So, it enters x 1. You can see small block here this one I am making it a little bigger, so that is visible that whenever this reaches this black box that means, it is a completion it is a completion of x 1 and this transition means now it is transition is level d 1. Now; that means even before completion if we one comes it can come over here this is another transition or if these two are connected then this means a different thing. This means that if event e 1 comes and the behavior of x 2 has reached the compellation then only come to x 2.

Here this e 2 meaning something different; e two is meaning that whenever e 2 is occurring you come to come out of it and come out where come out at the completion. This completion is whose completion this completion is the completion of the whole thing. Let us look at here again. Now, this is much more nicer, I have got the entire behavior v B which has got two behaviors x and y; x has got x 1, x 2, x 3. Now, x 1 these are normal transitions this a transition on completion when e 1 occurs and if x 1 has completed I will come to x 2, this just this same behavior.

Here what is happening from y 1 on event e 3, I come to y 2 what is this, this means if there are no events are created as y 2 completes, you complete y; and as you complete y, e 4 and V 4, you have completed y, but e 4 has not taken place. So, you have exited the whole thing has not completed. So, this means the semantics is that y 2 has completed. Here y 2 has completed obviously, otherwise I would not be here and e 4 has taken place I come here. Look at this edge, what does this edge signify? I have come here, x 2 has completed; that means, x has completed, but I am still I have not gone out. I am going out on this completion and e 5 I am entering y. Now, this is what is used in programs state machines, which has got some more flexibilities over state chart which will do in the next class.

So, this one bring us to the behavioral completion in that way we can show the behavioral completion; obviously, that will be useful for communication among the finite state machines.