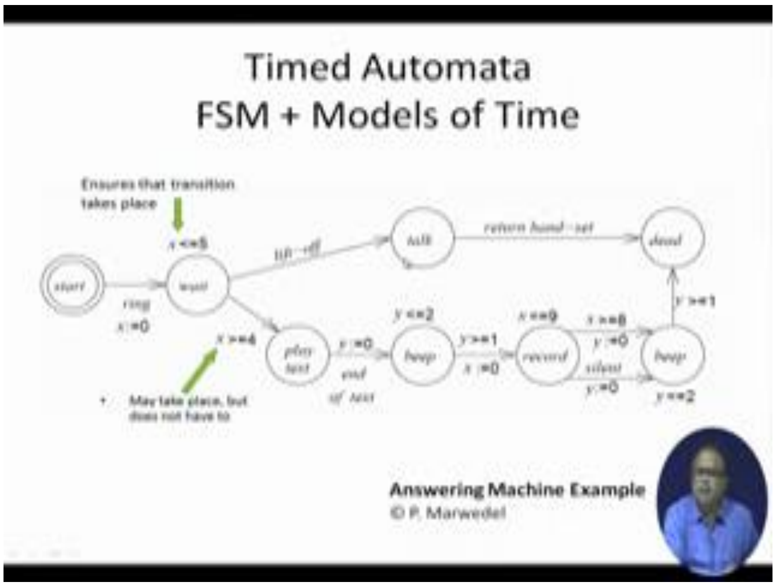
Embedded Systems Design

Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 34
Statechart and Statemate Semantics

(Refer Slide Time: 00:30)



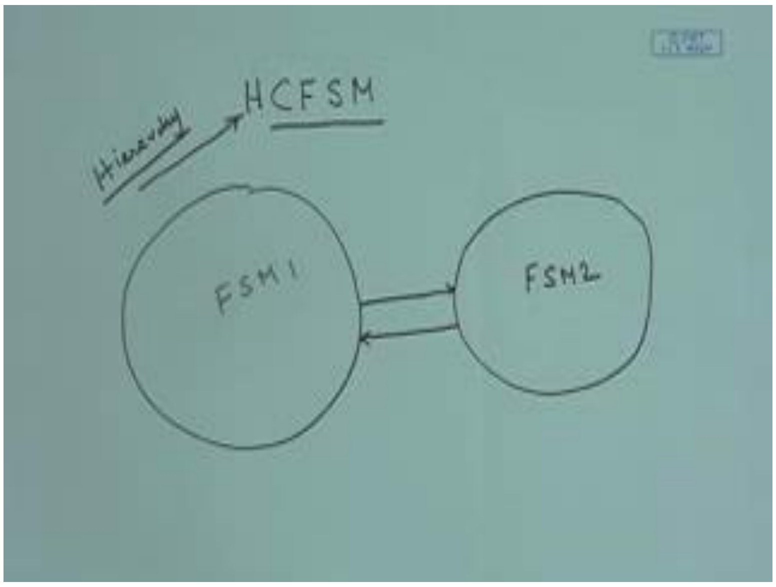
Timed Automata FSM + Models of Time

Answering Machine Example
© P. Marwedel

So, we were discussing about different specification and modeling schemes. And, we saw our very own popular finite state machines. And this, I am just trying to recapitulate some of the things that we did. And, we also found that we can try to capture time through timed automata, which is the finite state machine plus models of time. Here, we can see that we add extra variables or some variables representing the times. And, so at different points we can say that it will be in the wait state, at least for 5 units of time, then from here to here. For example, say this is the model of a answering machine. So, after the recording, it will be recording and up to the 8 units of time. After that it will stop recording. If in between it becomes silent, then obviously this will be reset. So, we can associate timing parameters also without automata.

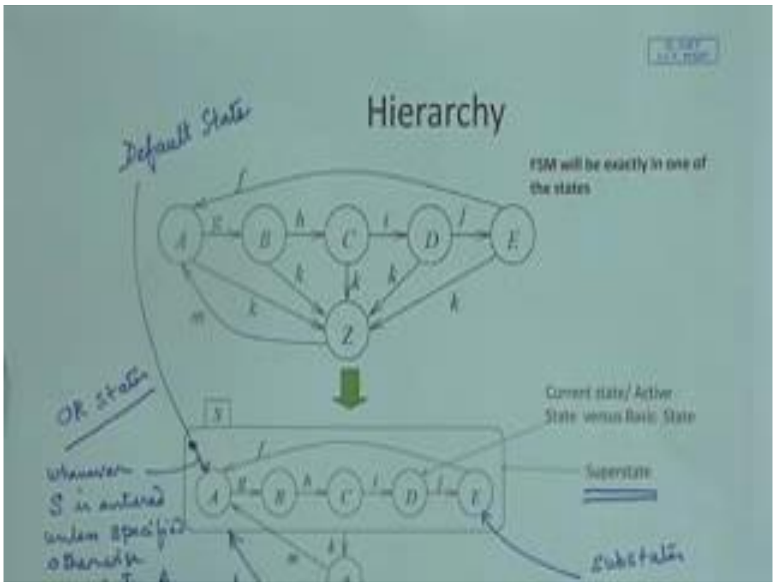
Next, we found that the problem with the automata, normal finite state machine automata is that the number of states can explode, can become very large. So, then we came to the concept of HC FSM. Before that C FSM is communicating finite state machine.

(Refer Slide Time: 02:04)



So, CFSM is communicating finite state machine. So, where I can have one FSM here, another FSM here and they can communicate among themselves. They can communicate among themselves. When I add an H to this and call it as HC FSM, then I bring in this H stands for hierarchy. Now such FSM, when I just take two FSM s and make them communicating FSM, then; obviously the state explosion will not be contained because both of them will explode in state. And, as I compose them the number of states and the ages will also increase. Now, in order to avoid that we thought that it will be advantageous if we can have hierarchical representation and, accordingly we introduce the notion of state charts - State charts we have discussed.

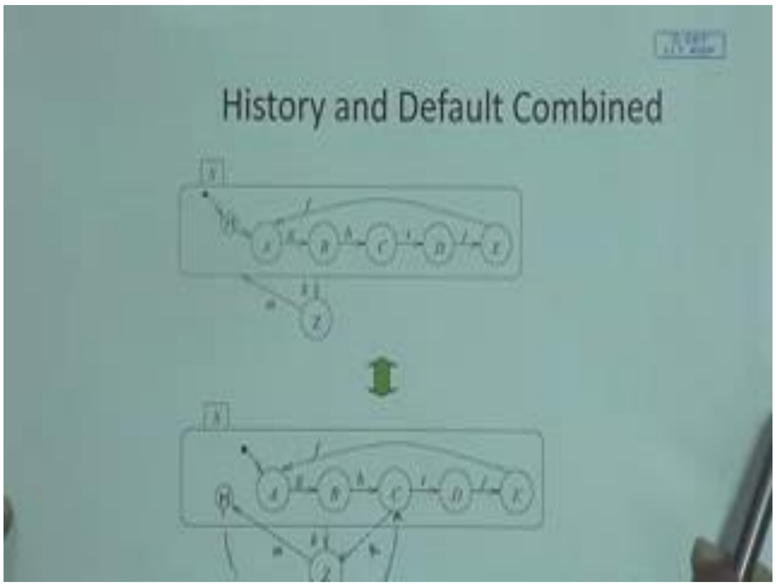
(Refer Slide Time: 03:25)


Hierarchy
Default State
FSM will be exactly in one of the states
OR state
Whenever S is entered unless specified otherwise
Current state/ Active State versus Basic State
Superstate
Substates

So, a typical finite state machine; as this shown here can be contained into another one particular finite state machine, which is the super state, we can have super states and sub states, right, in one machine. And, these are odd states. Each of this is the state. Earlier, I had one, two, three, four, five, six states. But, here I am talking of one state. And, inside that state at the top level of hierarchy I have got one state, but as I decompose this, then I will get five states. I am sorry; at the top level here I am getting two states. One is this state S; another state Z. As I decompose, I will decompose the state S into five sub states.

So, these are the sub states and these are the odd states. It will be either in this state or in this state. In any one of these this is the odd states. Now, so we have seen the hierarchy mechanism. After that we looked at the history mechanism.

(Refer Slide Time: 04:59)


History and Default Combined

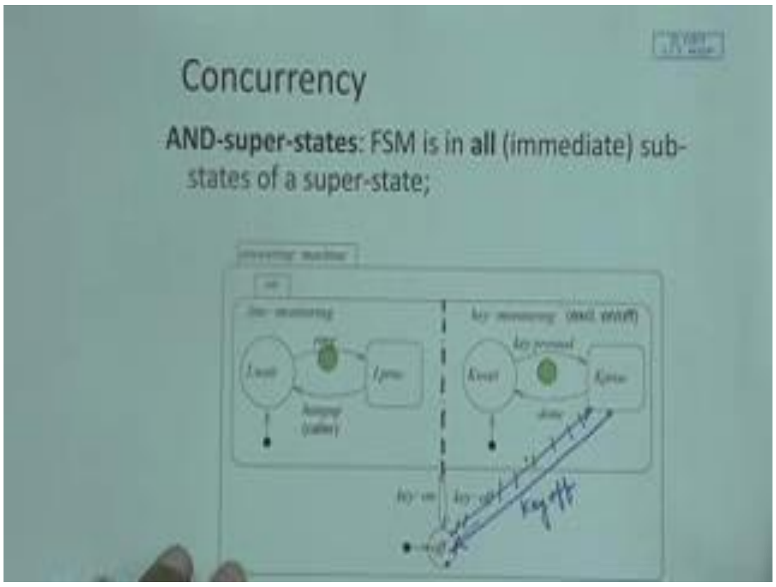
That along with our incorporation of the hierarchical representation, we can also have some history nodes put in, where the history node will remember the state from which we are exiting. So, we will exit from a particular state. We can exit from as we have shown in the earlier slide, earlier diagram that I can exit from any one of these to the state Z, which in a hierarchical representation I am showing here. But, if I just make an exit through this state, it does not really imply, tell me from which state I actually came out and if I have to return back where I should return back.

Therefore, we can incorporate the history state, which will remember in which state I was; fine. Also, as the diagram shows we had a entry state or start state; that means, this bold dot shows that in which state we are first getting in. For example, here when you say that we are getting in here, when I enter S, I am actually entering A. So, this is the default state, unless something is said to the contrary. Now, the history node tells me where I should return back or where I was. So, is also possible to combine the history and the default state like this. Initially, say for example, this one; that this one says that as I enter S, I entered A. A is the starting state. And if I exit, say for example, from C the k event occurred. So, I came out form here. As I go back, I will go back to C. right. History node tells me that.

Now, when I combine these two, then also this can be denoted in this way. See, state chart is a graphical language. So, whatever we are representing we are representing in the form of diagrams. And, the semantics must be very clearly understood because from here we will create an intermediate form, which will be executed. And, we want to see what are the things that will result from my given specification. So, here is what you can do by combining the history node and this and the default state.

A major important thing that comes with; one very important thing that is inherent in state chart is hierarchy, of course. I can had take a complex problem and decompose it at different levels of complexity in a hierarchical manner. The second big advantage of state chart or state chart type of representation is representation of concurrency.

(Refer Slide Time: 08:24)



Concurrency

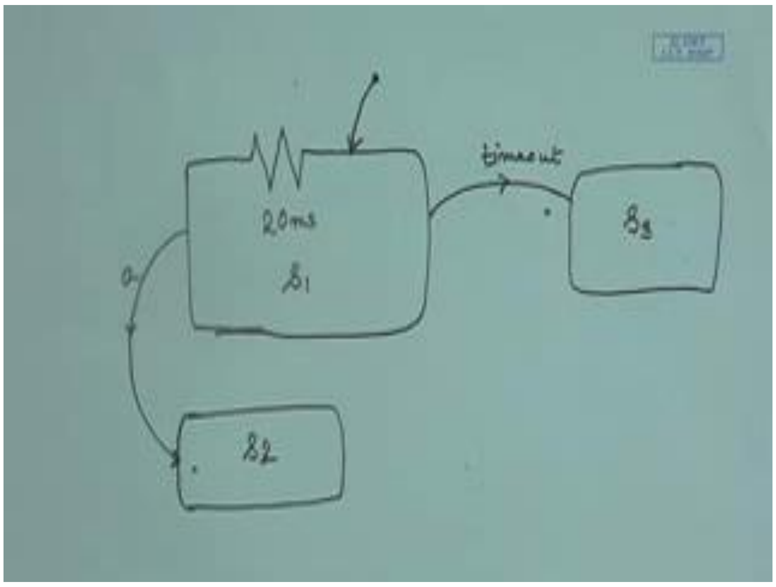
AND-super-states: FSM is in **all** (immediate) sub-states of a super-state;

As we can see again here, this was discussed in your earlier lecture that this is an answering machine representation, where this is a super state answering machine. Now, this super state consists of sub states line monitoring and key monitoring. Now, this line monitoring further consists of two states. One is line wait. Wait on the line or process the line. And that has got their own transitions. As you can see here, this is not an absolutely leaf level of state. L proc can be further decomposed to give more details.

Now, what is more important in this diagram is the presence of this dotted line here. The presence of this dotted line here, which shows that this state and this state are concurrent, they can run simultaneously. So, this whole state is the on state. And, I have got the off state. So, either I am in the on state or I am in the off state. But, when I am in the on state, these two states are end. Line monitoring is running as well as key monitoring is running. Both of them are active. So, they are running concurrently. And inside this line monitoring, either it is in line wait or it is processing line. Inside key monitoring, either it is waiting for a key press or it is processing. So, concurrency was another important thing that we saw.

And, we also saw that entering and the leaving the super states will be can be seen also from here that when for example, here off key is pressed, then we actually come out. Suppose, the key was being processed or something was being done, will come out directly, to the off state when the key off is pressed. Now, I can exit. When I exit from, any exit from this state, this internal state, then I am actually exiting this entire state. When I am being, the key off is being pressed, then I am coming to the off state. And this, I may exist in this on state. If I say that I am entering say because of key on, I am entering here at this border; that means, where I am entering. I am entering on both. What does entering on the both imply? That means, I am entering on this default state as well as this default state. All these things were discussed.

(Refer Slide Time: 11:58)

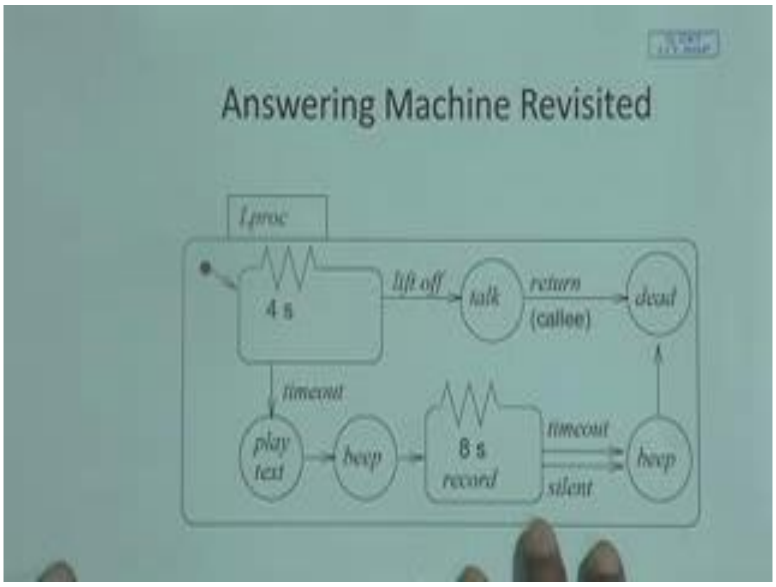


Next - we also mentioned in an earlier context about timers. So, we have got a timer structure in state chart. So, for example, if I draw a state like this and put in this sort of notation, graphical notation. And, so this is one state and we put in say 20 milliseconds here; that means, this is the timer. Now, on an event a I can come out of this state. Let us call this state anything say s 1. I can come out of this a and can go to some other state.

Now, when I am not drawing circles; that means, these are not the terminal states or these are not the atomic level of states. If this is a composite state high level, I can further decompose it. So, on this a it will come out to state s 2 may be. But, if 20 millisecond elapses, if 20 millisecond elapses and the event a has not taken place in between, then the timer will go out and the time out signal will be generated. And, this time out signal will bring me to another state, may be s 3.

So, in state chart along with any particular state at any level, I can associate a timer which will allow me to express the time out requirement. The implementation, when this will be implemented as an embedded system designer, when you will be implementing it; that means whenever you enter this state, it may enter this state from some other say suppose this was the default state. So, I enter here. As soon as you enter here, you set a timer to 20 milliseconds and then start its activity here. That is the semantics. Now, we have already seen the answering machine thing earlier. Now, let us relook at the answering machine with the use of such timers.

(Refer Slide Time: 14:27)



Answering Machine Revisited

Let us revisit the answering machine. We have also seen the answering machine in the, through the timed automata. So, compare this with the timed automata that we have seen. We saw (Refer Time: 14:30) here. Now this being hierarchical, looks much neater. You can see that in the earlier one when we had, I mean there were numbers of states over here, and this talk and all those things, this was only one part of the answering machine example. But, as we go in a hierarchical approach, we had shown that there are different components of it.

We had shown that there are two states; line monitoring, key monitoring. Then, under that the key wait on the key and process the key, etcetera. Now, we are here just elaborating the process, the line part, only one part. So, as I said that this was not the terminal state in this diagram. So, this can be further decomposed. And, here is the further decomposition.

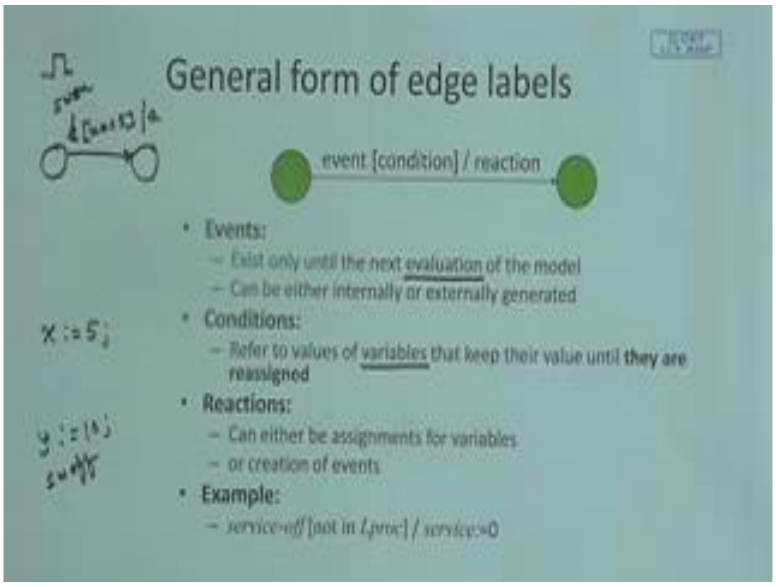
Here is the further decomposition of processing the line. It is not coming clearly. Little hazy it is. Coming clear there? So, I can explain. So, line process consists of a state where I come in. Let us see whether we can explain, understand the behavior from this. As this is being shown I am coming in here. I am there for 4 seconds. If within 4 second; now, when did I come to this line process? When there was some ringing. Some signal came into the line because the earlier diagram had shown that I was coming to this L proc on ring. When there was a ring, I entered L proc, when I entered L proc, if in between 4 second, so this is the wait state. Ring is going on. If somebody lifts off. There is a talking and after that he returns the phone becomes dead.

Otherwise, 4 second has elapsed. The person did not pick up. So, then there is a time out. And, it will play a text Mr. so and so is not here, etcetera, etcetera. There will be a beep. And then, it will again, it will enter the record state after the beep. Please record your message after the beep. So, it enters this. And, the recording is going on. The delay maximum of length is not the delay. It is. Here it was the delay, but here it is the duration. Maximum duration it can record is 8 second. Here is recording. Two thing can happen. He has finished, it goes there. Or, there is a time out. In either case, it will give a beep. That means, what happened? Here he hanged; he talked and then hanged the receiver. And then, it gave a beep and went dead. Otherwise, it will be a timeout. Whatever has been recorded will be given here. So, that is the application of the timer in state chart representation.

Next, we have to, now we are proceeding to understand the semantics of the whole thing. Now, we are actually working on a specification language. State chart is the model. And, we have to have some language, some tool through which we can graphically represent our model. And that should be compiled and some executable form should be generated.

In order that we can generate an unambiguous code, the semantics must be very clear that is true for any programming language. Now, in order to understand that we will have to introduce, we will introduce you to a couple of notions first.

(Refer Slide Time: 19:34)



General form of edge labels

event [condition] / reaction
x := 5;

- Events:
  - Exist only until the next evaluation of the model
  - Can be either internally or externally generated
- Conditions:
  - Refer to values of variables that keep their value until **they are reassigned**
- Reactions:
  - Can either be assignments for variables
  - or creation of events
- Example:
  - *service-off* [not in *Lproc*] / *service*:=0

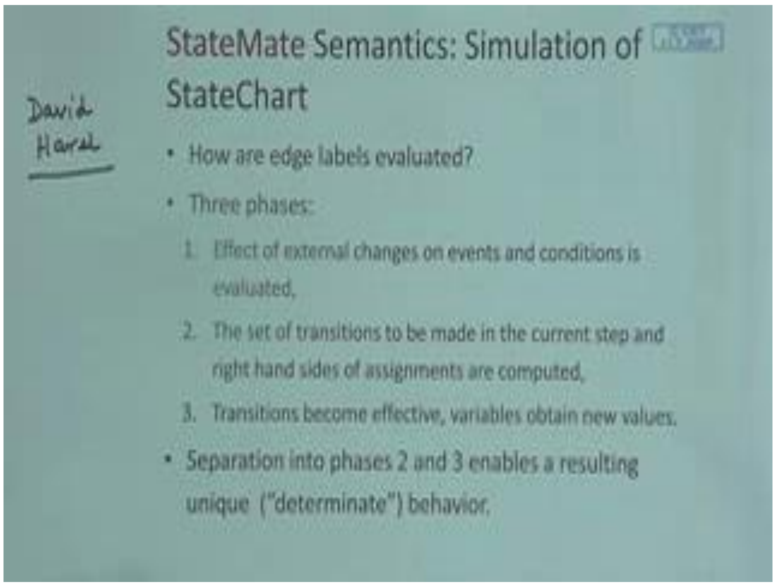
You know if we look at state, any state machine, there are two states and there is a transition between them. The transition is associated with some event and with some output or the reaction. When this event occurs, if you are in this state and some event occurs, you do this action. Along with that here we can also along with the event we can put in some conditions, some data conditions or something, that this is known as the guard condition. We had mentioned that. This is the guard condition that even if the event comes, event takes place, if the condition is not satisfied this transition will not be enabled. So, there is a guard condition.

Events; now, what are these events? Some clock pulse come somewhere. Events exist only until the next evaluation of the model. What is the next evaluation of the model? We will come to that. So, there is an event occurring. And that event will be assumed to be present and alarm has been set or a switch has been put on, whatever. That will be assumed to be true, assumed to be present till the next evaluation of the model. We will explain what this evaluation of the model is. Now, this events can be either internally or externally generated.

The conditions as I said refer to values of variables. Now, these variables will keep their value until they are reassigned; that means, this is more or less temporary, permanent because if I have got some variable assigned the value 5, as long as I do not reassign it, it will remain to be 5. So, if I have got something like some state here, and there is an edge which is taking you to another state and I say that there is some event e and some condition x is 5, then some action or reaction. Action; now, this event may be switched on. Event is; so, I just make it sw on. Some switch has been put on. That event will be there till the next. So, that is an event, an event like this.

The duration of this event, the value, this event is present till the next evaluation. But, this condition which is a variable till it is reassigned, it will remain like that. Actions can be anything. Action can be assignment of another value; some variable may be say y assigned 10 or may be sw off. That is another event. Anything; it can happen. Say, here is an example, service off. Some service off event has occurred. It is not in the line process. I mean, it is not processing any call right now. Then, service will become assign zero. So, that is what these items are. What this standard items are, what are their meanings. Now, how is that evaluated?

(Refer Slide Time: 23:41)



StateMate Semantics: Simulation of StateChart

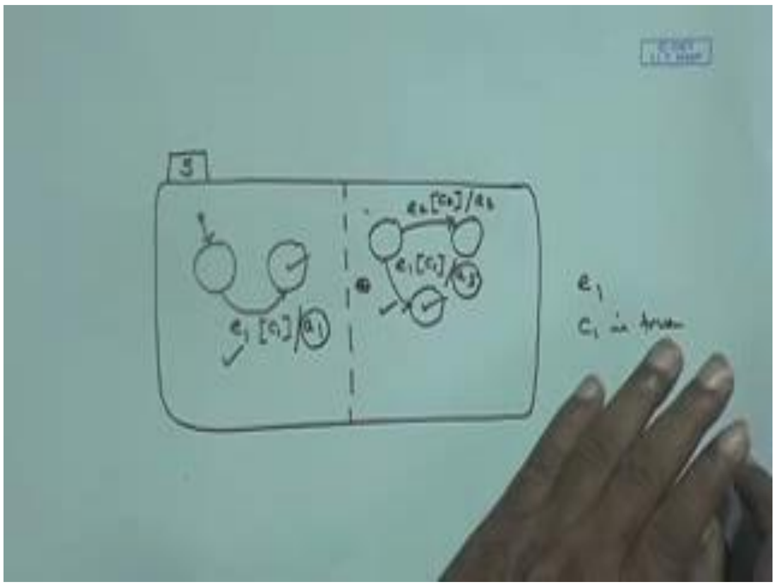
David Harel

- How are edge labels evaluated?
- Three phases:
  1. Effect of external changes on events and conditions is evaluated,
  2. The set of transitions to be made in the current step and right hand sides of assignments are computed,
  3. Transitions become effective, variables obtain new values.
- Separation into phases 2 and 3 enables a resulting unique ("determinate") behavior.

The tool that is used for, that has being generated on state chart is known as state mate. And, state mate. Now, by the way for those of you who are interested, this state chart was proposed by David Harel. And, the idea was taken up and the commercial version came out through a company, who named the product as statemate. I think the name of the company was I-Logix or something; I-Logix, most probably. If I am, if my memory does not betray me it was I-Logix. And then, the name was statemate. So, what we want to know that what is the semantics of the statemate execution.

First thing we will have to; in order to understand the semantics, we have to answer the questions. How are the edge labels evaluated? There are three phases. Please, note down if required and try to understand this. First, the effect of external changes on events and conditions are evaluated. We first check. First phase is whether what is this. Has there have been any change? I checked that; the set of transitions to be made in the current step. Remember that in this diagram I am showing only one state. There are multiple state transitions, events are occurring. So, I have to collect all the events, conditions and see which all are enabled. That is the first phase. The second phase is the set of transitions to be made in the current step and the right hand side of assignments are computed. Which of these I will be computing? Then, the third phase is transitions become effective; that means, my present state now becomes the next state. Why is this important? Why is this important?

(Refer Slide Time: 26:36)



For example, let me take an example; a hypothetical case. Say, I am in a and state. These are and states. I am in this state. So, and inside this there are two states. I came here and, this transition is on event e 1, condition c 1, some action a 1. And, here this is again say event e 2, condition c 2, action a 2 may be and, here event e 1; readable? Or, let me write. Rewrite e 1, condition c 1, may be action a 3. It is possible. Now, what happened in between is e 1 has happened and c 1 is true, then out of all these stages, right now I am in step one. I have to evaluate and find out that this is active, this is active. That is why that is the first phase.

Then, the second phase will be selecting which one. So, I will be doing both of that. But, before that, so before the transition I have to compute what is this a 1, what is this a 2. And, the third step is my present state will be this one and this one. Both of this will be simultaneously transitive. This is the three phase thing I am talking about.

Student: (Refer Time: 28:50).

Professor: Then that will be in the a 1. The question is that what if a 1 is enabling e 2. That will take place in the second step.

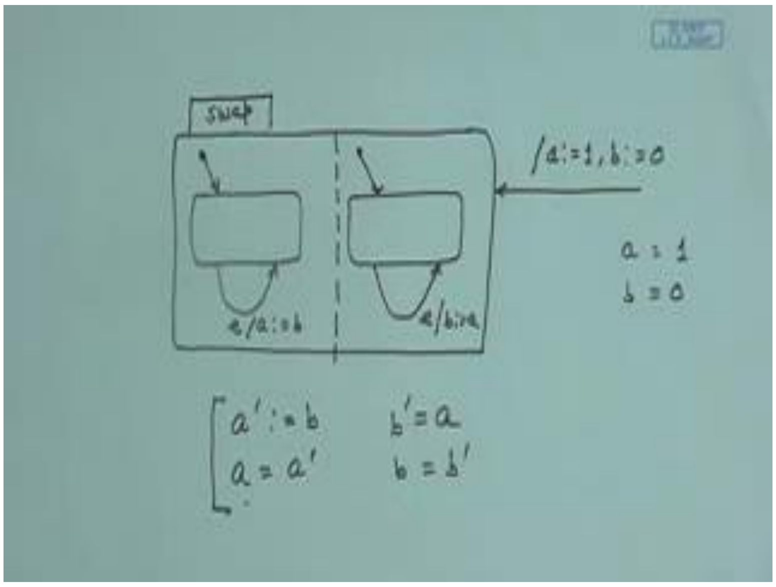
Student: (Refer Time: 29:03).

Second step, I am not taking the action. The action is being computed in the third step. Let us see here. I will show you some examples. Say, the set of transitions to be made in the current step are computed; which are the one that I have to, which are the transitions that I have to do. Then, the transitions become effective. Transitions become effective and variables obtain new values. That is coming in the third phase. And then, I am again cycling back to this one. Separation into phase two and three enables resulting unique determinate behavior. That is why I am not allowing them. Exactly the questions that you are asking that in that case I am not too sure as to what will, which one will take place first. That can generate into a race condition. That will generate the race condition. So, I am not allowing that.

Student: (Refer Time: 30:10).

Professor: Yes, we are assigning it. Now, so with that let us take an example.

(Refer Slide Time: 30:31)



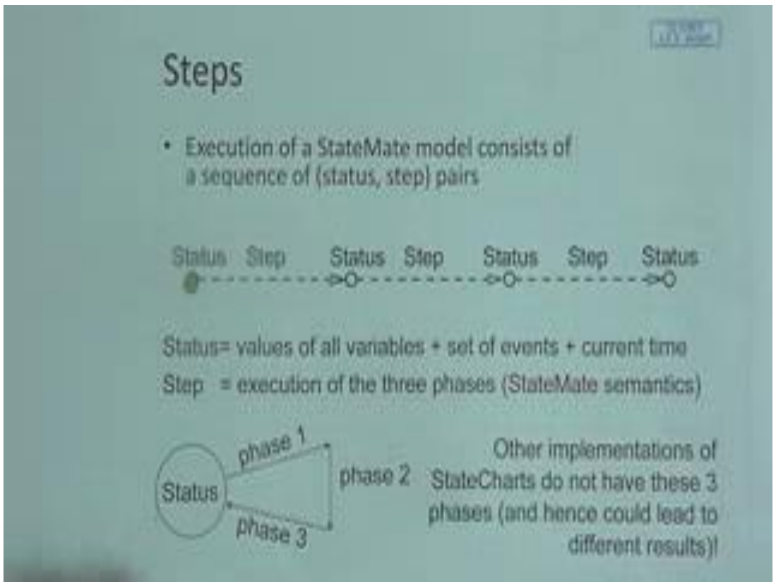
Let us keep it in front of me, so that. Now, let us say those of you do C programing, a favorite thing that you often find is the swap problem. Swapping variables, suppose I am doing some swap and we have to understand what this is meaning. I am drawing according to the notations that I have already described. Now, here some event e leads to the action a assigned b. And, the same event e leads to the action b assigned a.

Now, what will happen? Initially, I entered here with some action; say some action, no event nothing. a was 1 and b was 0. So, a was 1 and b was 0. Now, I am here. Now, event e has occurred. Let us look at our. So extend, the effect of the external change is event e. There are no conditions for the sake of simplicity. So, two things will happen here. The set of transitions to be made in the current step are evaluated. Both this transitions will have to be evaluated. And, the right hand side of the assignments are computed; that means, here what I am having is a prime is getting b. And, here b prime is getting a, internally.

In phase three transitions become effective, variables obtain new values. Therefore, a will get in phase three, a will get a prime and b will get b prime. Therefore, as a result actually they are swapped. If I do this, then actually they are swapped. But, normally that we usually do with pointers and all those things. Otherwise, if I do just a assigned b and b assigned a, then the swapping is not taking place. So, here that is the reason why these two are separated. Otherwise, what could have happened? The result would be indeterminate, depending on which one takes place first. And, might be these are concurrent. So, they are being executed by two separate processors, which are of different speed. So, anything can happen. By separating these two out, I can have a functional approach. That same result will take place what is intended.

So, with that we have understood the statemate semantics. In a single phase environment if I had done this, what would have happened? Let us see the same thing. If I just had done this by in a single phase what would have happened? What would have happened? The same thing here, not this, this one. What would have happened if I had done it in a single phase? So, a was 1 and b was 0. So, in the single phase a would be 0 and b would be 0. Or a would be, b would be 1 and a would be 1, depending on the order in which they are going on.

(Refer Slide Time: 35:46)


Status
Step
Status
Step
Status
Step
Status
Status
phase 1
phase 2
phase 3

Steps

- Execution of a StateMate model consists of a sequence of (status, step) pairs

Status= values of all variables + set of events + current time
Step = execution of the three phases (StateMate semantics)

Other implementations of StateCharts do not have these 3 phases (and hence could lead to different results)!

So, before concluding I will explain therefore the semantics here. The steps are. This diagram will explain you this step; how the execution is taking place. We are starting with some status here. The state; status means the state of the whole thing. So, you are starting with this, we are going through one state step. So, what happens? First, the step means execution of a, say status and step, status and step, like that we go three times. The status is the value of all the variables plus set of events plus the current time. Now, step is execution of the three phases. All the three phases that constitute the step one, two, three; then, the new state comes.

So, when I am in state, status, this diagram tells me phase 1 I do first, then phase 2, then phase 3. And then, I come to the new state. Other implementations do not have these three phases, but statemate implementation actually does it; the simulative works in these three phases. And, there by insures determinant result. The events now, what I was meaning? When I say it that the events leave till the next evaluation of the model. What do I mean by the next evaluation of the model? The event that occurred, say the event that switch on that took place will remain till I come to the next status. So, for all these three steps this event, however transient, will be assumed to be there. So, that is the meaning of till the next evaluation of the model.