Object-Oriented System Development Using UML, Java and Patterns

Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

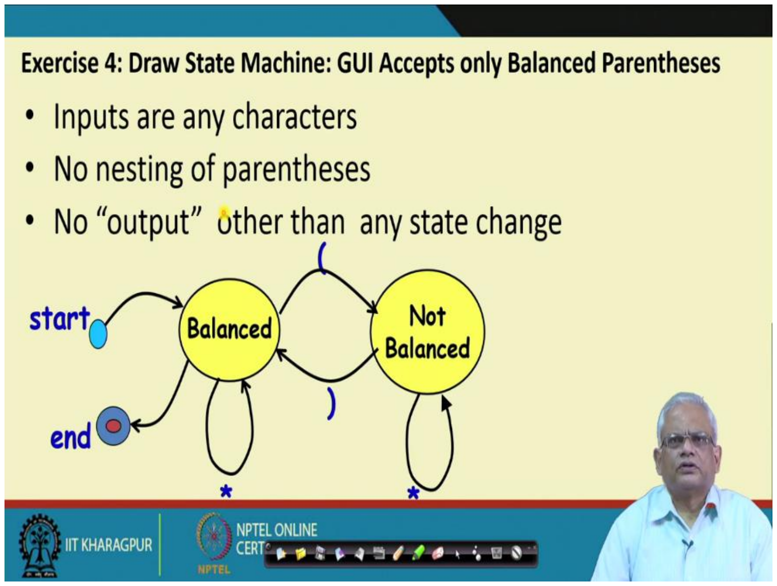
Lecture 20

State Charts Overview

Welcome to this lecture.

In the last lecture, we had some preliminary discussions on state machine diagram. The state machine diagram can model the behavior of stateful objects and we had looked at some examples. All the example we had were a simple state machine, and then we had an extended machine where we had a state variable. Now, let's do one more problem.

(Refer Slide Time: 00:48)



Exercise 4: Draw State Machine: GUI Accepts only Balanced Parentheses

- Inputs are any characters
- No nesting of parentheses
- No "output" other than any state change

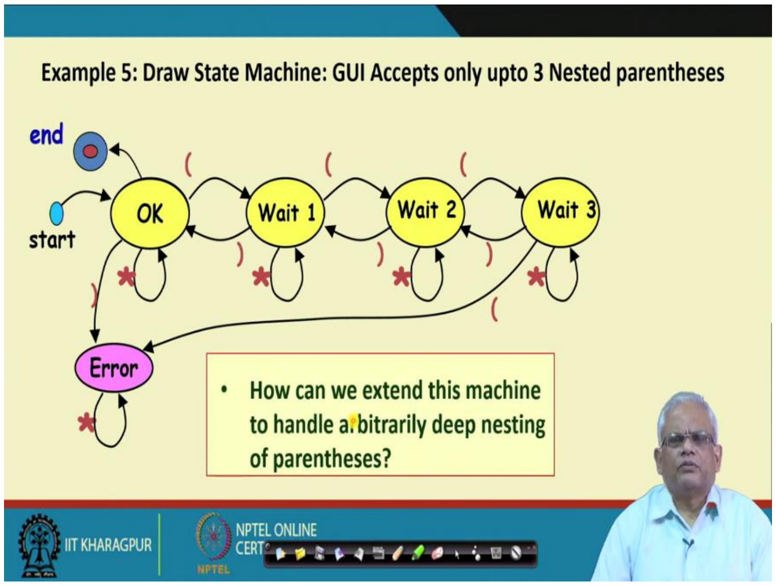
This is about a GUI that accepts only Balanced Parentheses (in the above slide). Since all of you had encountered state models, finite state machines, and so on. In this example, it accepts characters and as long as the parentheses is balanced, it accepts otherwise, it keeps on waiting for other characters.

How do we model this?

The inputs are any characters. Let's assume that there is no nesting of parentheses just one open bracket and close bracket and there is no other output than the state change. So, the model is simple.

From the start the pseudo transition lead to Balanced state and when a left bracket is entered, it goes to not balanced and it keeps on accepting characters in the not balanced state and then, as soon as it accepts a right parentheses it goes to the balanced state and from balanced state, it can accept any number of characters and it can successfully terminate at end. So, this is the simple model of this GUI accepting only balanced parentheses. Now let's do a few more exercises.

(Refer Slide Time: 03:21)



Example 5: Draw State Machine: GUI Accepts only upto 3 Nested parentheses

- How can we extend this machine to handle arbitrarily deep nesting of parentheses?

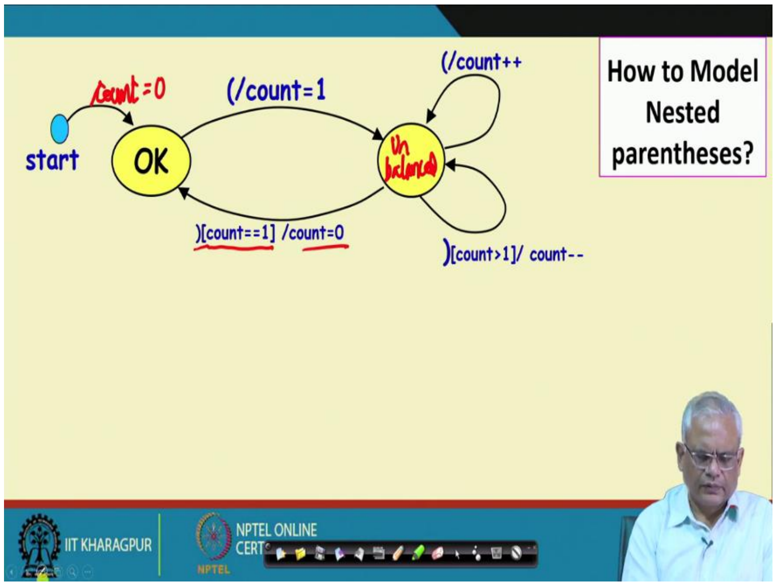
Let say, that a GUI can accept only up to three nested parentheses. The previous example, there are no nesting of parentheses was allowed as long as it is a left bracket transits to unbalanced and right bracket transits to balance. But there can be three nesting of parentheses. How do we model this using a state machine diagram?

In above slide you will find the solution. Let say, we have this OK state and then left bracket, it goes to state Wait 1. In state Wait 1 getting a left parenthesis it goes to State Wait 2 and from that it goes to Wait 3 for a left parenthesis. One right parenthesis goes to Wait 2 and for two parentheses from there it again goes to OK state. If it is character other than parentheses it continues to be in the present state and, more than three left parentheses are not allowed. More than three left parentheses, it goes to the error mode and once it is in error mode it continues to be there does not come out of that. But if it is in OK state it can terminate successfully.

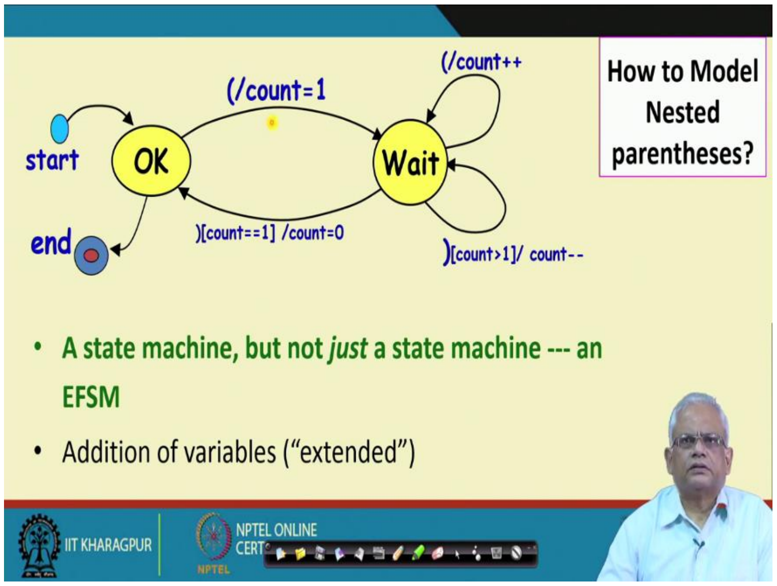
It is a small extension of the no nesting case. But what if we want to model the behavior of the GUI that can accept up to any nesting?

Three nesting is straightforward. We just had to introduce few more states. But then more or less similar to what we had done last time, that is about a non-nested parenthesis. But how do we model, when any number of nesting is allowed? Based on what we have discussed, can we model that?

(Refer Slide Time: 06:29)



How to Model Nested parentheses?



How to Model Nested parentheses?

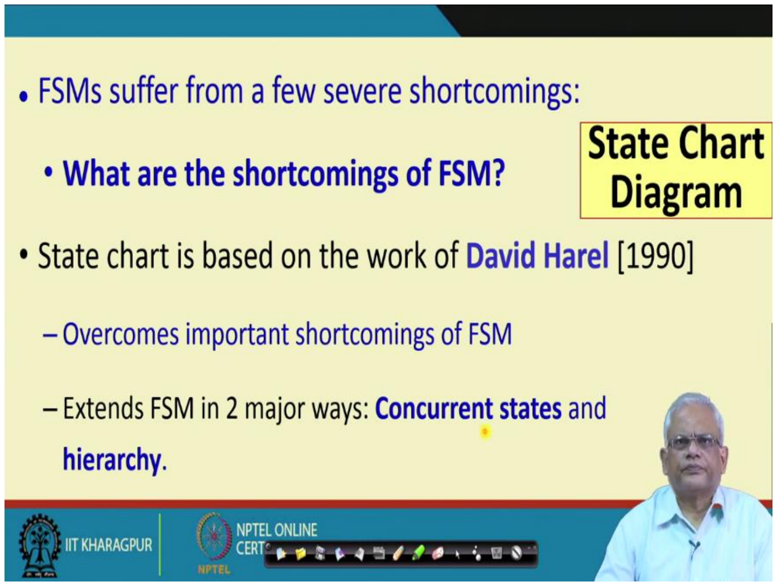
- A state machine, but not *just* a state machine --- an EFSM
- Addition of variables ("extended")

Yes, we can model that by using an extended machine (in the above slide). Here, we keep a count on a left parentheses event. We have the count = 0 initially. And if a left parenthesis occurs, it goes to the unbalanced state and set count =1.

And it remains in the unbalanced state as long as the count > 0. If there is a further left parenthesis the count increments it keeps on staying in the unbalanced. If there is a right parenthesis, count reduces and if the count is greater than 1 it continues in Wait state. When count = 0, it goes to the balanced state after the right parentheses.

We could model this nested parentheses case using an extended finite state machine or EFSM. Here it is called extended because we have added some state variables here.

(Refer Slide Time: 09:35)


- FSMs suffer from a few severe shortcomings:
  - What are the shortcomings of FSM?

State Chart Diagram

- State chart is based on the work of David Harel [1990]
  - Overcomes important shortcomings of FSM
  - Extends FSM in 2 major ways: Concurrent states and hierarchy.

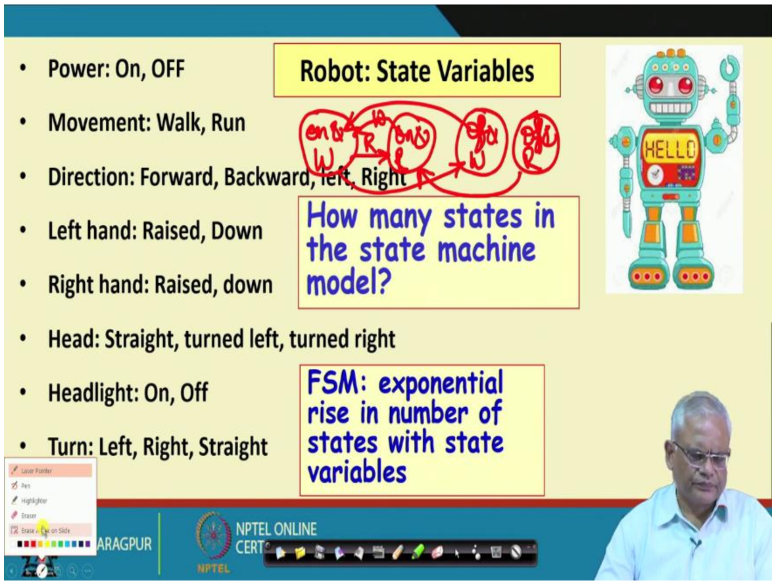
So, far what we discussed are the typical state machines or the finite state automata. But how is a state chart diagram different from the FSM?

The finite state machine actually suffers from some very serious shortcomings and the state chart diagram overcomes those shortcomings. Now, can you identify what are the shortcomings of the state machine diagram?

One of the very serious shortcomings of the state machine diagram is explosion of states. The number of states increases very rapidly. For complex system, it can have thousands of states. But the state chart which was proposed by David Harel in 1990, overcomes this shortcoming of explosion of states. Here the number of states does not increase exponentially, so it is becoming a more manageable. the second problem with the traditional finite state machines is that we do not have concepts of a concurrent state. We will see what it means as we proceed in this lecture. So, these are two important shortcomings of the finite state machine, that is explosion of state and lack of support for concurrent states.

The state chart diagram proposed by David Harel, it extends the FSM with two main mechanisms. There are other things also as we will see, but the two main extensions are concurrent states and hierarchy. The hierarchy handles the explosion of states. We will see how it handles as we proceed; and the concurrent states, they handle concurrency.

(Refer Slide Time: 12:23)



Robot: State Variables

- Power: On, OFF
- Movement: Walk, Run
- Direction: Forward, Backward, left, Right
- Left hand: Raised, Down
- Right hand: Raised, down
- Head: Straight, turned left, turned right
- Headlight: On, Off
- Turn: Left, Right, Straight

How many states in the state machine model?

FSM: exponential rise in number of states with state variables

Now, let see what we mean by the state explosion. We will see that as the number of state variables increases, the number of states in the model increases exponentially. We will just take this example of a robo which accepts commands to move (in the above slide). Now let say we are trying to find how many states in the state machine diagram of this robo.

Let just describe it little more. Suppose the robo has a power switch in the remote and we can either switch on or off the robo and definitely the behavior of the robo is different when the power is on or off. In the power Off state, it does not accept any commands, whereas in power On state, it accepts commands. So definitely there are two states with this simple description, the On and Off state. Now, let say on the remote, we have two switches here. One is a command to walk and another command is run. How many states now? It can be On state and walking, On state and running, Off state and walking (it does not do much there, but it is Off and walking), Off and running (not doing anything here but state will be there). So, if we model here, we need four states here: power On and walking, On and running. Off and walking, Off and running. Now, if there is a power On in 'Off and running state', it will transit to On and running. If it is power On in 'Off and walking', it will go to On and walking. If it is a walk switch pressed from the On switch it will go to On and walking. If it is the Off switch pressed in the On and walking, it will go to Off and walking and so on.

So, still, we could manage with four states in the power On, Off, and movement walk and run. Now let's bring in other state variables. See each of these will actually correspond to a state of the robo and internally this will be kept track by a variable. The power will be a state variable, which value can be On-Off, movement can be a walk and run. Movement also a state variable and its values can be walk and run.

Now, let assume that we have one more state variable. Let say direction is another variable. The direction of the robo can be forward, backward, left, and right. If the direction is set forward, it will walk forward, run forward. But if the direction is backward, it will walk backward, run backward, and so on. If its direction is left, it will start running, walking left and so on. How do we model this?

It can be ‘On and forward and walking’, ‘On and backward and walking’, ‘On and left and walking’, ‘On and right and walking,’ and so on. For each of this state, we need four state models. So, how many state models we need to model these three behaviors?

It will be 2 (On and OFF) * 2 (Run and Walk) * 4 (Forward, Backward, left, right) = 16.

Now let us say, we have one more state variable that the robo left hand can be raised down up. Now it will be On, walking forward, left hand raised or left hand down, and so on. Now the number of states required will be 16 * 2 (left hand raised and down) = 32. Now let say, similarly, the right hand can be raised down and up, then we need 32 * 2 = 64.

Now let us say, the head can be straight, turned left or turned right and then we need 64 * 3 = 192 states. Now let say we have one more, which is the headlight can be turned on and off and then we need 192 * 2 = 386. Now let say, we can set it to take left turn, right turn or straight. So, that will be 386 * 3 = 1158.

If it has n state variables and each state variable has two values, then how many states will be required to model the behavior of the robo?

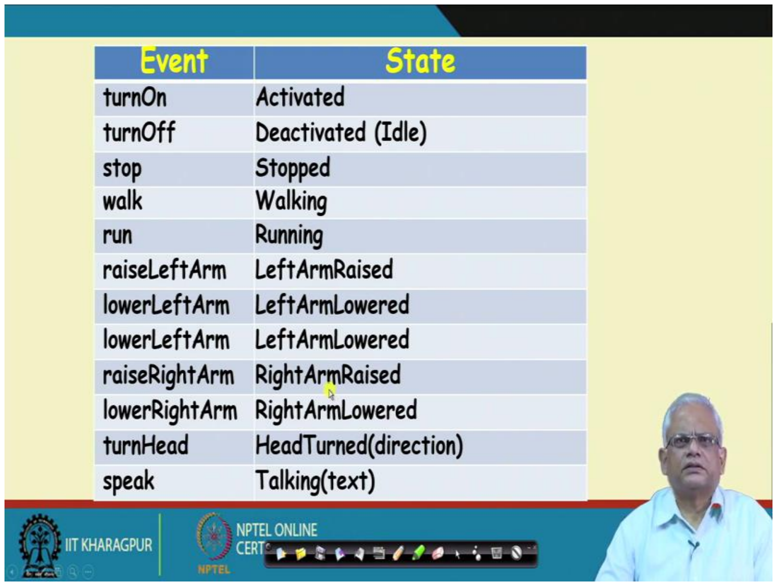
It will be 2^n.

If each state variable can take three states and there are n state variables, how many states will be needed to model the behavior of the robo?

It will be 3^n.

So, we see that there is an exponential increase in the number of states required to model the behavior of the robo. This is a major problem with the finite state automata-based modeling that the number of states required quickly increases as the behavior becomes slightly complicated.

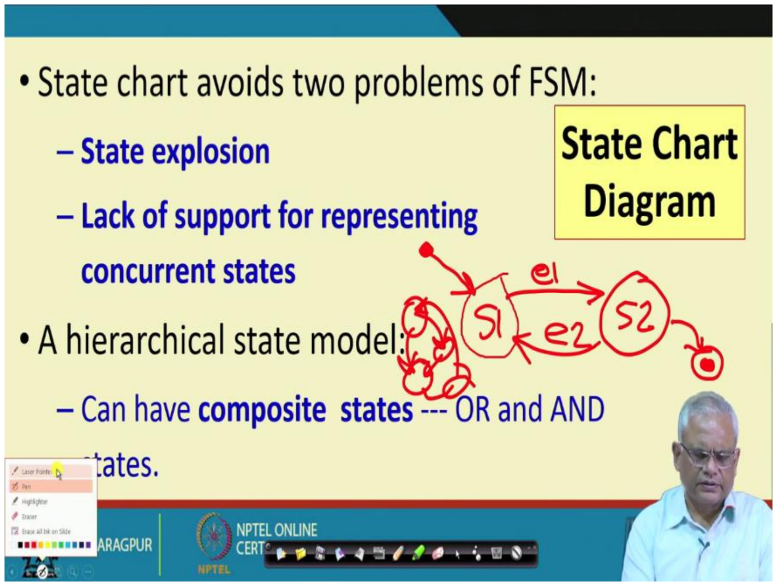
(Refer Slide Time: 20:28)



Event	State
turnOn	Activated
turnOff	Deactivated (Idle)
stop	Stopped
walk	Walking
run	Running
raiseLeftArm	LeftArmRaised
lowerLeftArm	LeftArmLowered
lowerLeftArm	LeftArmLowered
raiseRightArm	RightArmRaised
lowerRightArm	RightArmLowered
turnHead	HeadTurned(direction)
speak	Talking(text)

This is an elaboration of the different events (in the above slide) to which the robo responds. TurnOn, turnOff, stop, walk etc. are the state and we can see that the number of states can be extremely large.

(Refer Slide Time: 20:45)



- State chart avoids two problems of FSM:
  - **State explosion**
  - **Lack of support for representing concurrent states**
- A hierarchical state model:
  - Can have **composite states** --- OR and AND states.

State Chart Diagram

Let's now see how the states chart diagram overcomes the problem of the finite state machine. As we were saying the two major problems of the finite state machine are the state explosion problem and the lack of support for representing concurrent states. We will see how to handle the state explosion problem in the state chart. We will see in state chart diagram the number of states becomes manageable and also the concurrent states representation in the state chart diagram.

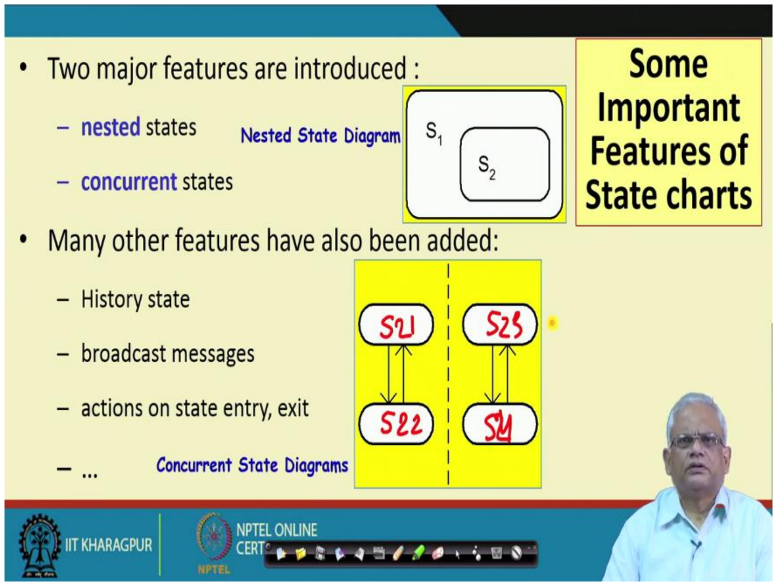
The first one is about the state explosion problem. Here to handle this problem in the state chart diagram, a hierarchical state model is used. In a hierarchical state model, the states are composite states. That means, a state in a state model can have further states inside it.

Let say, in a finite state automaton has two states, state 1 (S1) and state 2 (S2) and the event 1 (e1), in S1 makes it transit to S2 and event e2 in state S2 makes it transit to state S1.

Let say, S1 and S2 are composite state. Now in the composite state, each state will have a further state machine. If we double click on S1, we will see that S1 is not a single state, it is a composite state. If we double click on S2, we will see that it contains another state machine. If we double click on the second level state machine, we might see another state machine and so on. This is a hierarchical representation.

At the top level, we have only two states (in our previous example); it appears simple. But then there is a second-level state here for each of these 2 states and then in the second level if we try to see inside a state it will have another state machine inside it, and so on and that is what we call it a hierarchical state model. we will also have concurrency in the form of AND states. We will just see that.

(Refer Slide Time: 24:14)



Some Important Features of State charts

- Two major features are introduced :
  - nested states
  - concurrent states
- Many other features have also been added:
  - History state
  - broadcast messages
  - actions on state entry, exit
  - ...

Let's look at the nested states and the concurrent states. These are the two major features of the state chart model. The nested state we had just seen that one state in the top level will have further states inside it, and the concurrent state, will have two state machines here and there is a dotted line here (in the above slide diagram) that means that this is a AND state. Meaning od AND means it can be in two sides concurrently. See in the above slide diagram, S21, S22, S23, S24 are the states and and all are represented in AND, that means in this state, it will be in S21 and S23 or S21 and S22 or it will be S22 and 24 and so on (all possible combinations).

There are also other features, the history states, broadcast message, actions and state entry-exit. So, the state chart in summary, I can say that the state chart is a much more powerful modeling mechanism compared to the simple finite state automaton. Here we have many more features, nested states, concurrent states, history state, broadcast messages, action on entry and exit. We will look at these features, these are parts of the UML state machine diagram.

The state chart which was proposed by David Harrell, about 30 years back have been adopted in UML because many classes have non-trivial state model and using the simple finite state machine, we cannot really make a satisfactory model of those.

We will look all these features, the nested state, the concurrent state, etc. in more detail and we will try to model state behavior of objects and also, we will see how we can mechanically generate code either in C, C++, or Java code based on the state model.

We are at the end of this session. We will stop here and continue in the next session.

Thank you.