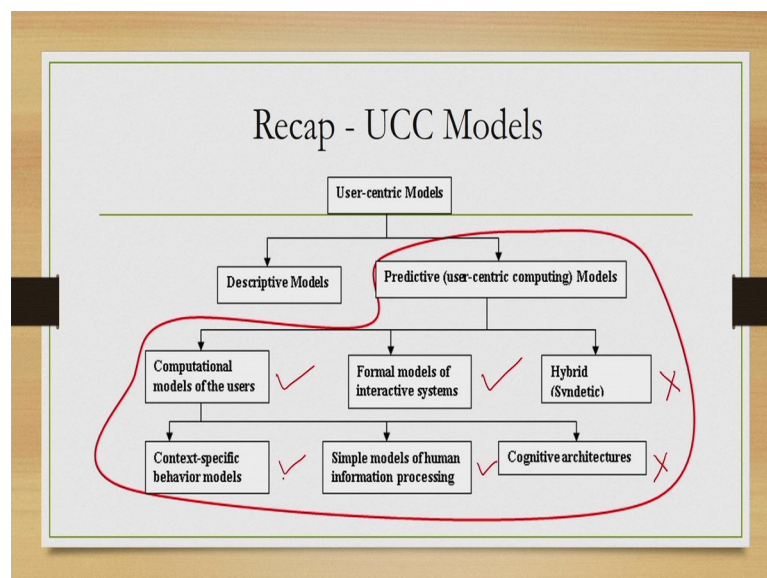


User-Centric Computing for Human-Computer Interaction
Prof. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture - 23
Introduction to formal models in UCD

Hello and welcome to lecture number 23, in the course User Centric Computing for Human Computer Interaction. So, today we are going to talk about another class of computational user models which we are discussing.

(Refer Slide Time: 01:00)



So, before that let us just try to recap the different categories of user centric computing models and what we have learned and what we are going to discuss. So, if you may recollect, we have discussed about this complete taxonomy of user centric computing models, under which our main focus is predictive models.

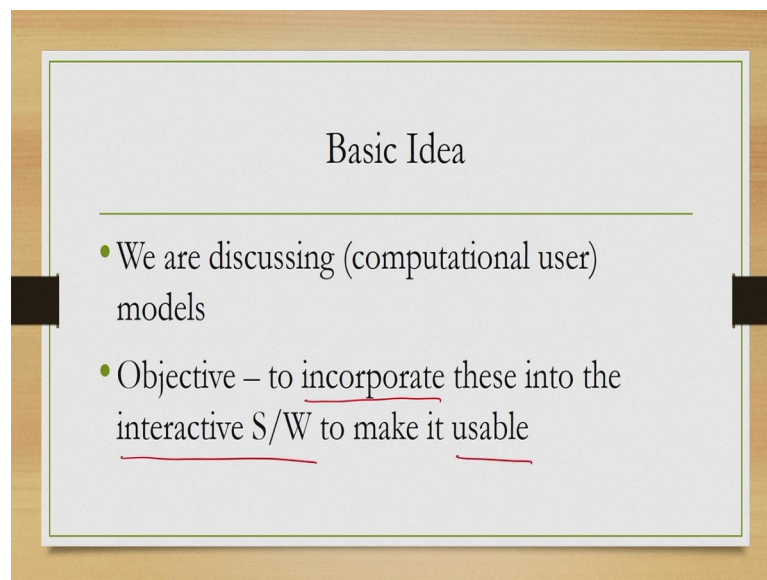
In predictive models there are subcategories. So, we mentioned that we are going to discuss about two subcategories computational models of the users and formal models of interactive systems and we are not going to discuss hybrid or synthetic models.

Under computational user models, we mentioned that we are going to discuss about context specific behavioral models and simple models of human information processing and we are not going to discuss about cognitive architectures. So far, we have discussed

these two categories; context specific behavior models, namely the Fitts law, the Hick Hyman law and then simple models of human information processing namely, the Klm and the Goms family of models.

Along with that we have discussed another set of models inspired by these classical models namely, the 2D and 3D pointing models, the constraint navigation models, the models for mobile typing and models for touch performance which comes under the contemporary models of interactions. All these belong to these overall category of computational models of the users. In today's lecture, we are going to discuss about the other category which is remaining, that is formal models of interactive systems.

(Refer Slide Time: 02:56)

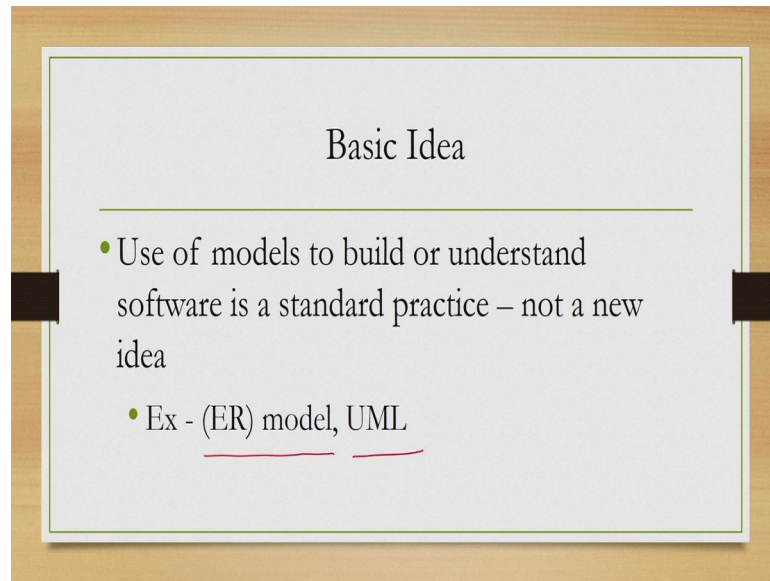


Basic Idea

- We are discussing (computational user) models
- Objective – to incorporate these into the interactive S/W to make it usable

Now, when we are talking of models, what is our objective? Our objective is to have models that we can incorporate into the overall interactive system software so that the software becomes usable. So, that is our overall objective.

(Refer Slide Time: 03:08)



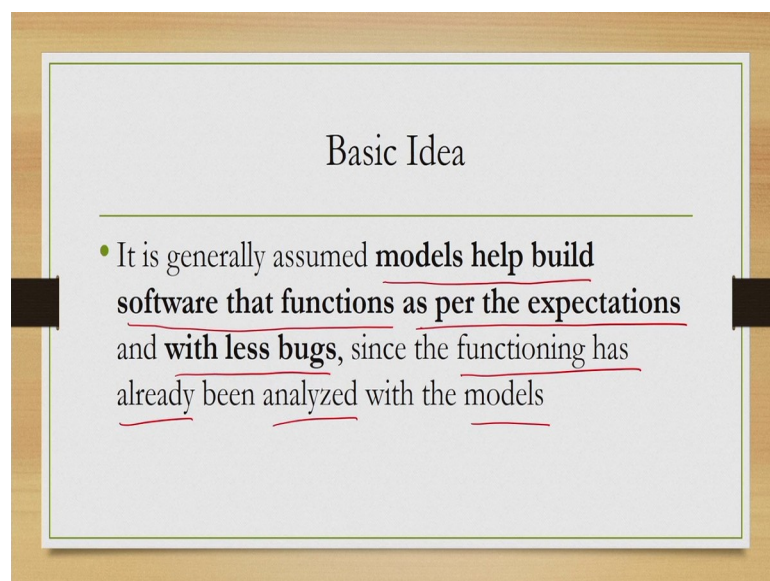
Basic Idea

- Use of models to build or understand software is a standard practice – not a new idea
 - Ex - (ER) model, UML

Now, the idea of models in the building of software's or in the analysis of software's is not new, it is already there. There are several models that people use to build software's or to analyze or understand the behavior of a software.

For example, the entity relationship model or the UML these are some of the widely known; these are some of the widely known models that are used in the building or analysis of software systems.

(Refer Slide Time: 03:44)



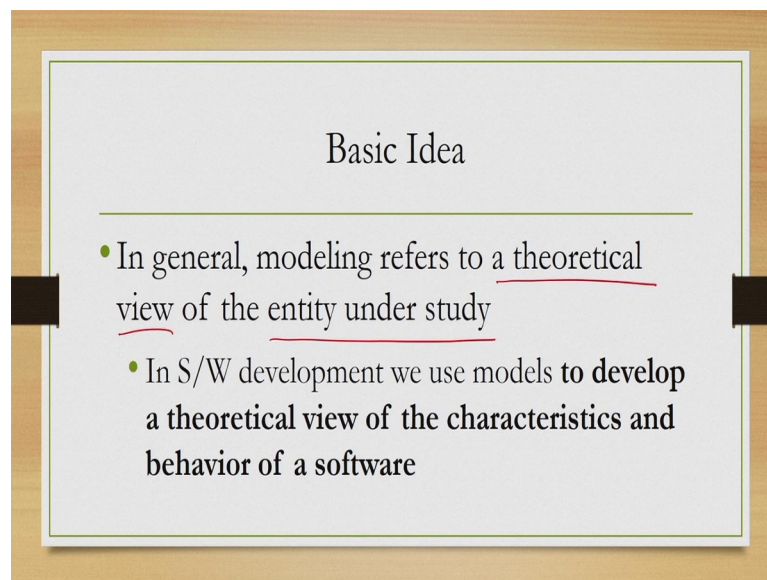
Basic Idea

- It is generally assumed models help build software that functions as per the expectations and with less bugs, since the functioning has already been analyzed with the models

Now, why we go for models? Again, if we recap earlier, we discussed this point, let us restate it. Generally, we assume that models help us build software that functions as per the expectations and with less bugs, because the functioning has already been analyzed with the models.

So, when we say that we are interested in using models in a software system, our intention is to develop a software that has less bugs and that behaves as per our expectation, because we have already analyzed the behavior and we know what are the faults and accordingly, we have rectified the faults with the help of the models. So, that is the main objective of using models in software system development.

(Refer Slide Time: 04:38)

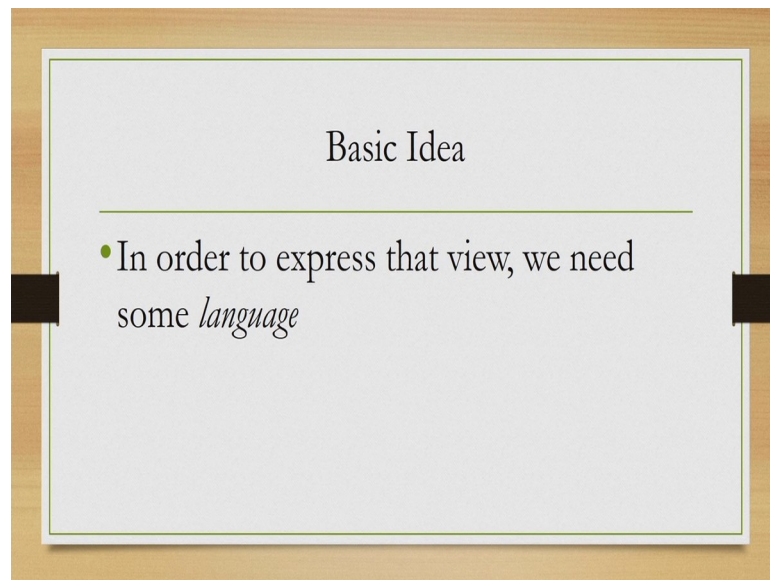


Basic Idea

- In general, modeling refers to a theoretical view of the entity under study
- In S/W development we use models **to develop a theoretical view of the characteristics and behavior of a software**

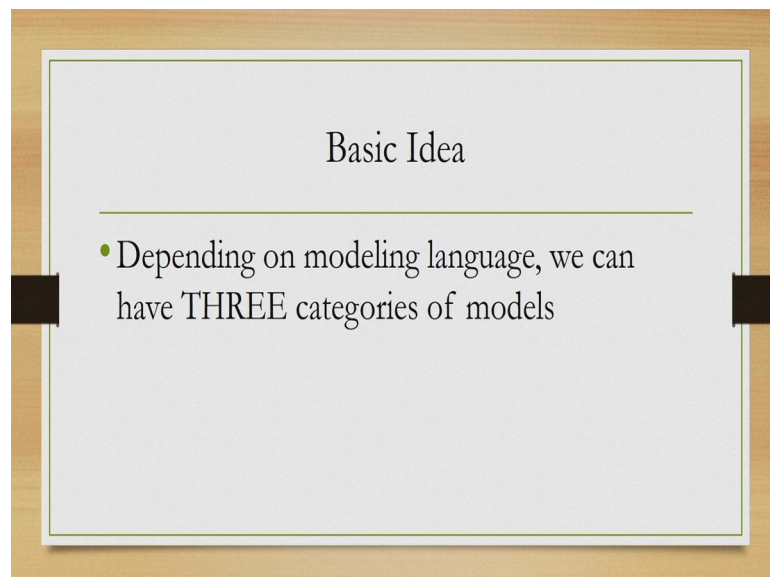
So, then when we talk of models what we refer to? We refer to specifically, a theoretical view of the entity under study. So, when you talk of a model of a software; we refer to a theoretical view of the software that we are interested to develop.

(Refer Slide Time: 04:57)

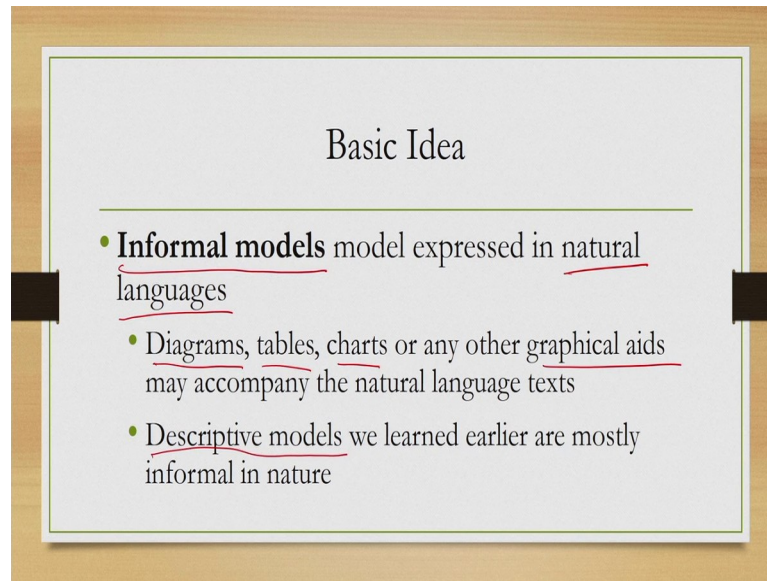


Now, this view needs to be expressed. So, how do we express the view? We express it using some language, depending on the language that we are using to express the theoretical view of a software system, we can have three types of models, three categories of models

(Refer Slide Time: 05:17)



(Refer Slide Time: 05:19)



The slide is titled "Basic Idea" and contains a bulleted list. The text is as follows:

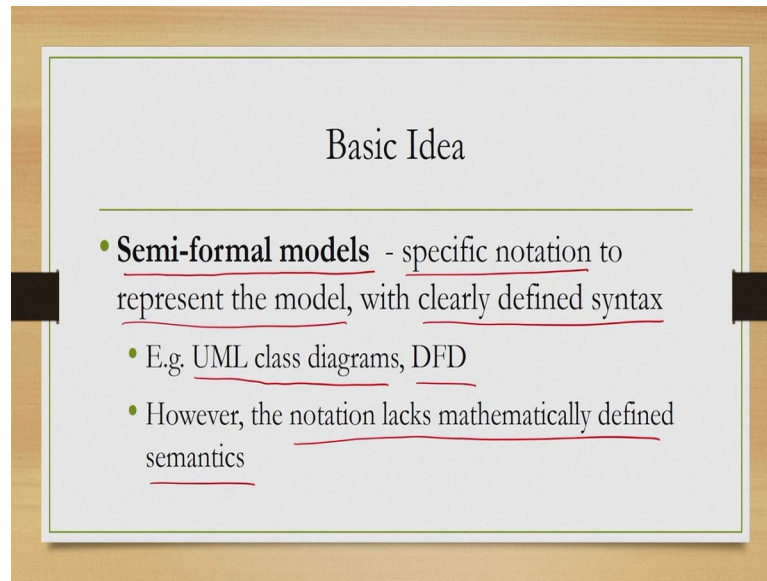
- **Informal models** model expressed in natural languages
 - Diagrams, tables, charts or any other graphical aids may accompany the natural language texts
 - Descriptive models we learned earlier are mostly informal in nature

First one is the informal model. Now, in these models, we express the theoretical view using a natural language. For example, if we are explaining the behavior of a software using some text, some natural language text, which may or may not be accompanied by diagrams, tables, charts or other graphical aids then we call it an informal model.

Earlier, if you may recollect, we discussed descriptive models. Now, in those models we explained the model in terms of natural languages, those belong to informal models. The main problem with these models is that natural language can be ambiguous. So, when we are using a natural language to explain or express a theoretical view of a model, due to the ambiguous nature of the language the interpretation of the model may be different depending on the person, who is interpreting it.

So, the same model expressed in the form of a natural language may mean one thing to one person and it may mean another thing to another person, because of this inherent ambiguity of the natural language.

(Refer Slide Time: 06:51)



The slide is titled "Basic Idea" and contains a list of points. The text is underlined in the original image. The points are:

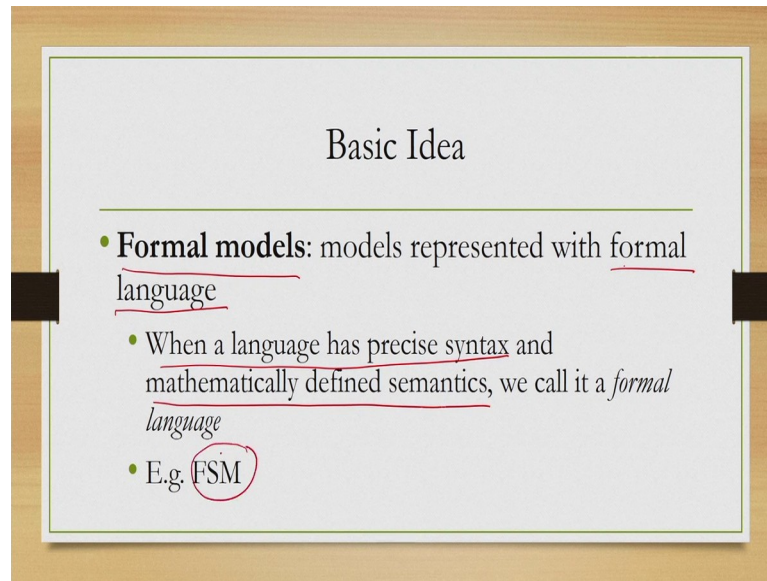
- **Semi-formal models** - specific notation to represent the model, with clearly defined syntax
 - E.g. UML class diagrams, DFD
 - However, the notation lacks mathematically defined semantics

There is another category of models, which is called semiformal models. Here, unlike in the case of natural language based in formal models here, we have a specific notation with clearly defined syntax to represent the model

For example, the UML class diagrams or the data flow diagrams, these are some notations we use to represent a software system and these notations have specific syntax. So, we need to follow that syntax to represent the system using those notations; however, the notations do not have mathematically defined semantics. So, we cannot actually use the notations to reason about the properties of the system.

We can simply represent it, but the notation is not suitable for reasoning in a formal sense, informal reasoning is always possible, but formal reasoning is not possible and finally, we have formal models.

(Refer Slide Time: 07:51)



The slide is titled "Basic Idea" and contains the following text:

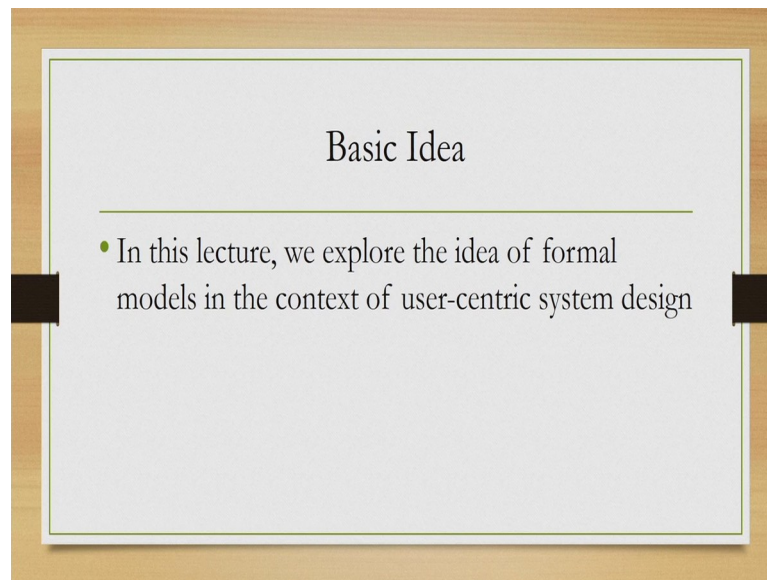
- **Formal models:** models represented with formal language
 - When a language has precise syntax and mathematically defined semantics, we call it a *formal language*
 - E.g. FSM

Now, these are models that are represented with a formal language. What is a formal language? When a language has precise syntax and mathematically defined semantics, we call it a formal language.

For example, if we represent a system behavior or the system in terms of a finite step machine or FSM, which typically uses state diagrams or state transition networks and state definitions, then we can call it a formal language and the corresponding model of the system as a formal model, because the way FSM is represented using the notation, that notation typically has mathematically defined semantics and we can use that notation to actually derive mathematically certain properties of the system without any need for informal analysis.

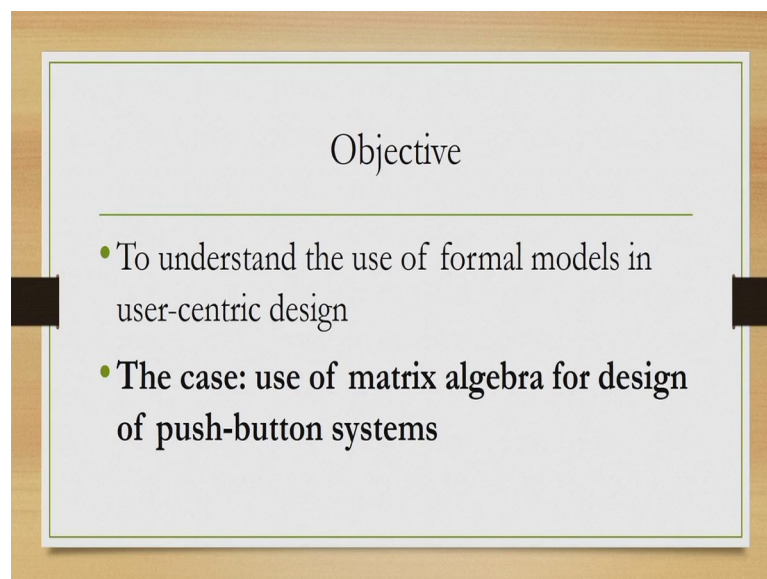
So, that is the idea of formal model. So, whenever we are using a formal language that has a mathematically defined semantics and well defined syntax and we use that language to represent a model, that model is a formal model.

(Refer Slide Time: 09:12)



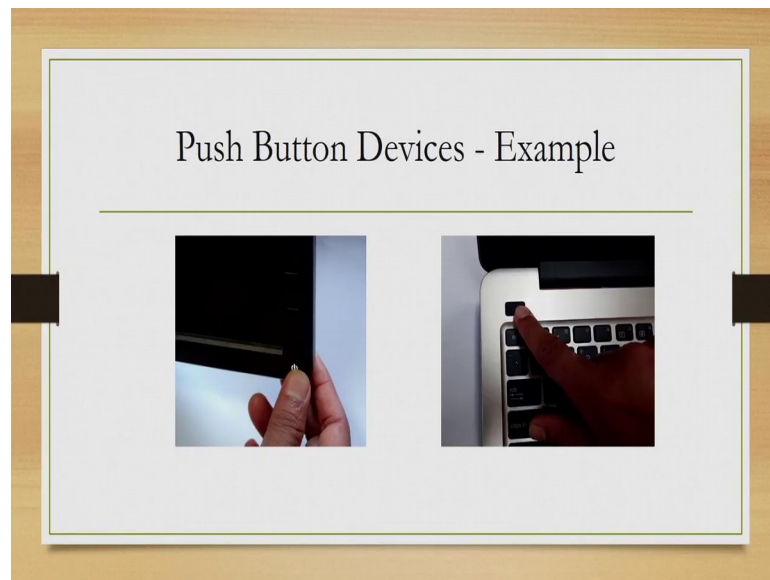
Now, in this lecture we are going to explore, the use of formal languages in modeling interactive systems or in other words, we are going to discuss about formal models of interactive systems.

(Refer Slide Time: 09:31)



So, we will start with a case study to make the things clearer. The case in question is the use of matrix algebra to design push button systems. So, before we explain or before we delve deeper into the case study first, we need to understand what is a push button system.

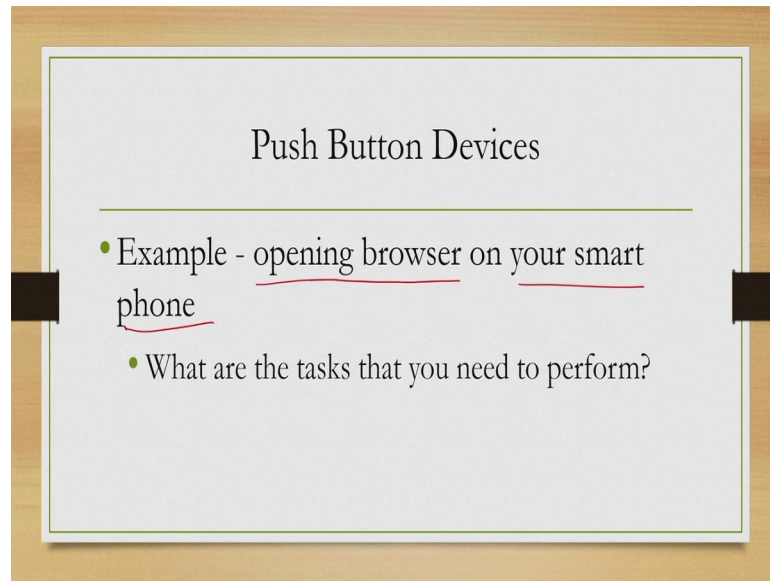
(Refer Slide Time: 09:45)



You probably have seen, such systems in practice many a times. So, it is in fact, all around us for example, look at these figures. So, the left figure shows the switch on a button of a monitor, of a computer monitor and the right figure shows the on off button on a laptop. So, these buttons are push buttons.

So, we push it to switch it on or again push it to switch it off and we see such push button devices all around us and the most interesting thing is that many interactions that we perform with standard GUI or graphical user interfaces can be mapped to the push button interactions. Let us see one example, how that is possible.

(Refer Slide Time: 10:36)

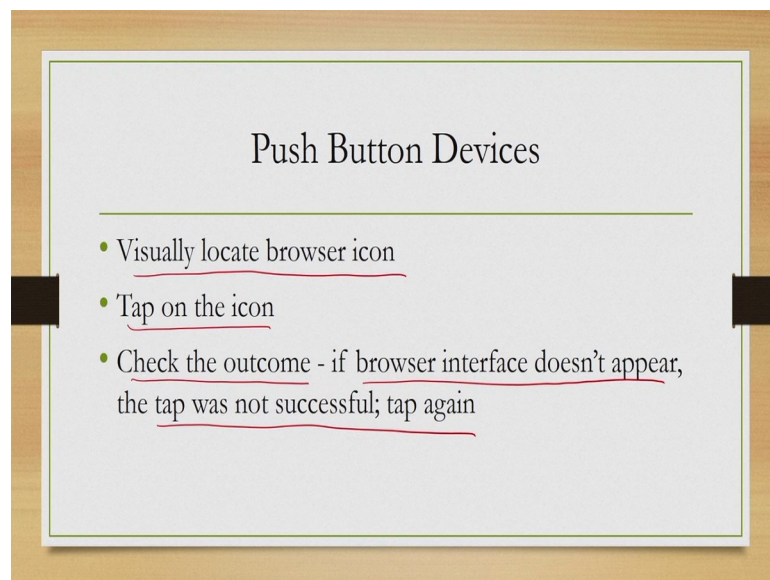


Push Button Devices

- Example - opening browser on your smart phone
 - What are the tasks that you need to perform?

Suppose, you are opening a web browser on your smart phone now, in order to do that what are the tasks that you need to perform?

(Refer Slide Time: 10:46)



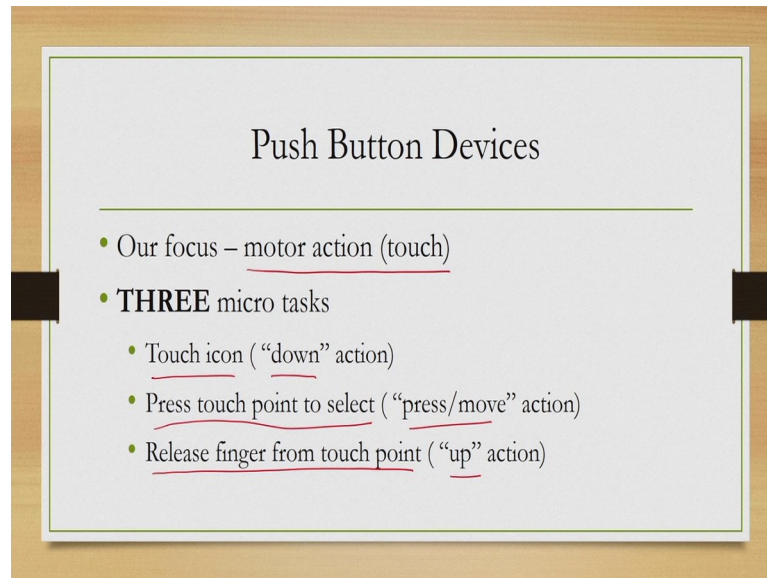
Push Button Devices

- Visually locate browser icon
- Tap on the icon
- Check the outcome - if browser interface doesn't appear, the tap was not successful; tap again

Of course, at a very broad or macro level, what you need to perform is visually locate the browser icon. So, where the icon is, then tap on the icon to select it and finally, check the outcome.

If the browser interface appears, then you have done the task successfully otherwise, you tap again. So, these are the three macro level tasks that are involved in opening a web browser on your phone.

(Refer Slide Time: 11:19)

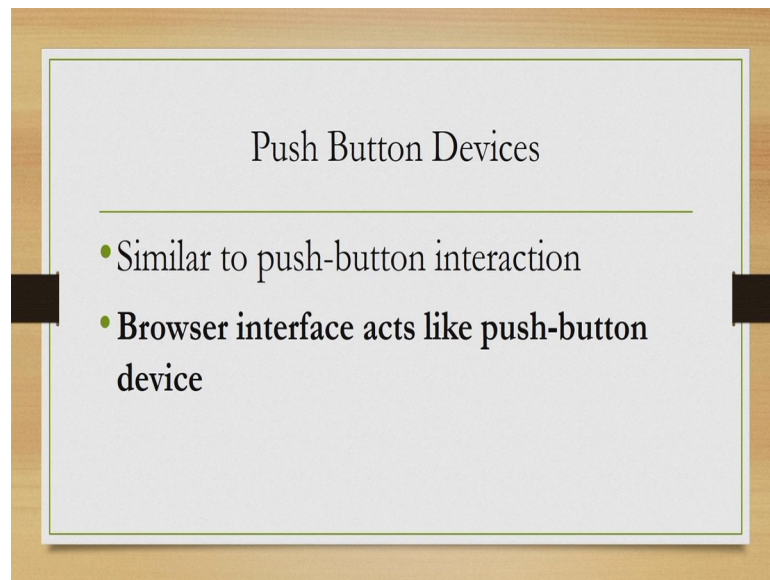


However, in this lecture our focus is on the motor action or touch action, because I have noticed in the previous sequence of activities visually locating the browser icon or perceiving the state after tapping on the icon are essentially cognitive activities and we are not interested about cognitive activities, at this stage of the lecture.

Instead, we will focus on the motor activities and in the previous sequence of action the only motor activity was the action of tap. Now, in order to perform a tap to select an item on a screen, what are the micro level tasks that we do? There are three tasks that typically we perform; one is touch the icon which is often called a down action, then press the touch point to select is often called press or move action and release the finger from the touch point which is called up action.

So, any touch selection involves these three actions down, press or move and up, together they indicate the selection of an item.

(Refer Slide Time: 12:27)



Now, if you notice the or if you compare these series of micro actions for selecting an onscreen item on a touch device and the push button actions that we have seen before namely switching on or off of a monitor or a laptop screen or switching on or off of a laptop that activities are similar. In case of switching on or off of a monitor, we do the same things, we touch the button, we press it and we release the finger. Similarly, in case of opening a browser through touch, we touch the browser icon, we press or move and we release the finger.

So, both are similar. In fact, many such interactions on GUIs or any wimp interfaces can be modeled as a push button activity and the interfaces are actually can be considered as push button devices.

(Refer Slide Time: 13:31)

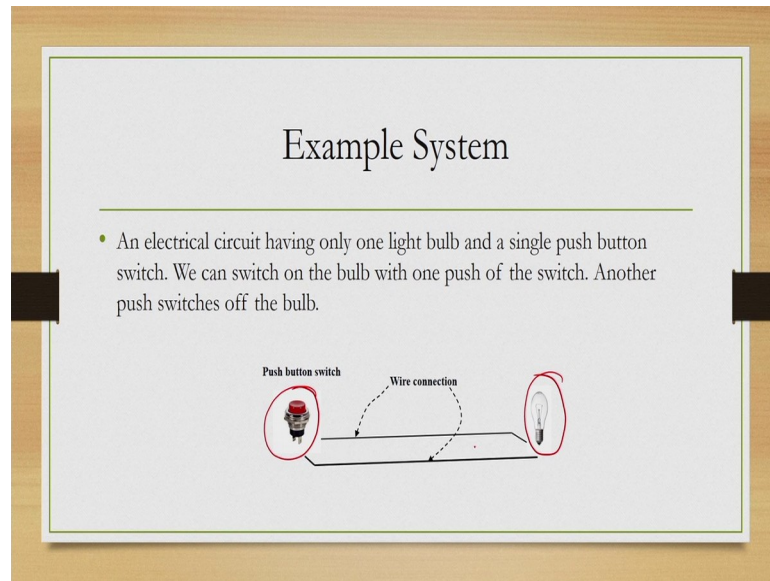
Push Button Devices

- Push button interaction very important for many real-world interactive systems
 - Design and verification of user-centric systems that employ push button interaction is very important
- Let us see how we can do this with matrices and matrix algebra

So, it is very important to be able to model such interactions so that we can actually build better push button interfaces or interactions for our purpose

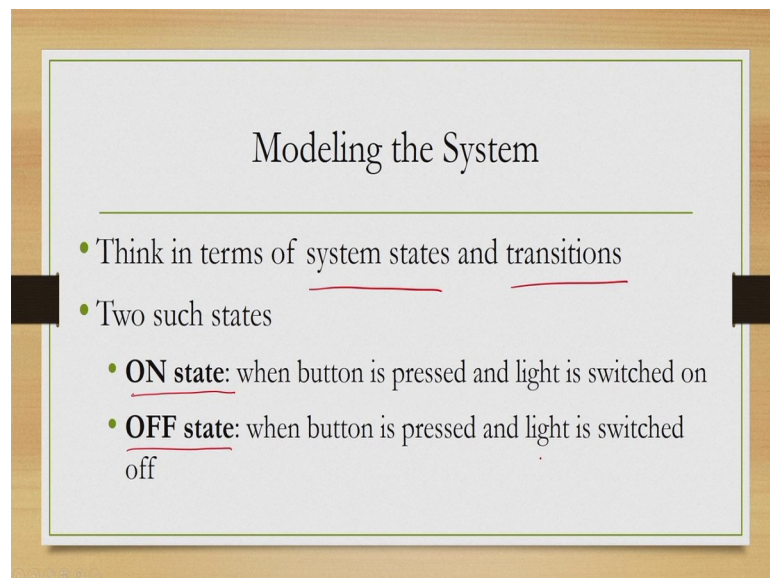
So, as I said, push button interactions are very important for many real world interactive systems and design and verification of user centric systems that employ these push button interactions is therefore, very important. So, in order to design and verify such systems, what we can do? We can employ formalisms, formal models, for the purpose. Let us try to understand, how with the use of matrices and matrix algebra to do the task the task of design and verification of user centric systems that employ push button interactions.

(Refer Slide Time: 14:22)



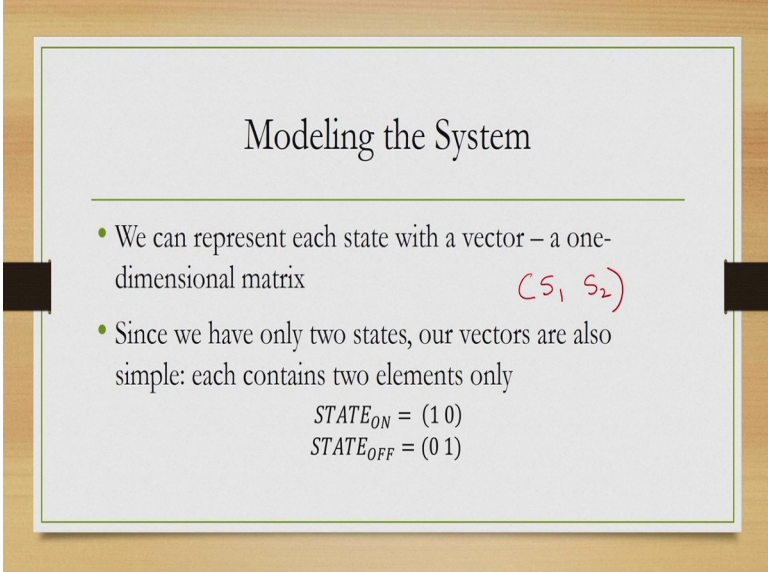
So, we will illustrate the idea with a very simple system. This is the system here, as you can see there is one push button switch and one light bulb. These are connected through wires. Now, if you push the button down, the bulb gets switched on, you can switch on the bulb or switch it off by pushing the button. So, one push may switch it on, the next push may switch it off. So, this is a very simple system with a single push button switch to switch the circuit on or off.

(Refer Slide Time: 15:06)



So, then if I want to model this system, if I want to have a formal model of this system, what we can do? We can think in terms of the system states and the transition between the states. Now, as you have probably noticed there are two states; one is the ON state one is the OFF states. So, when the button is pushed and the light is switched on we call it an ON state. So, when the light is switched on the system is in ON state, when the light is switched off the system is in the OFF state

(Refer Slide Time: 15:42)



Modeling the System

- We can represent each state with a vector – a one-dimensional matrix (S₁ S₂)
- Since we have only two states, our vectors are also simple: each contains two elements only

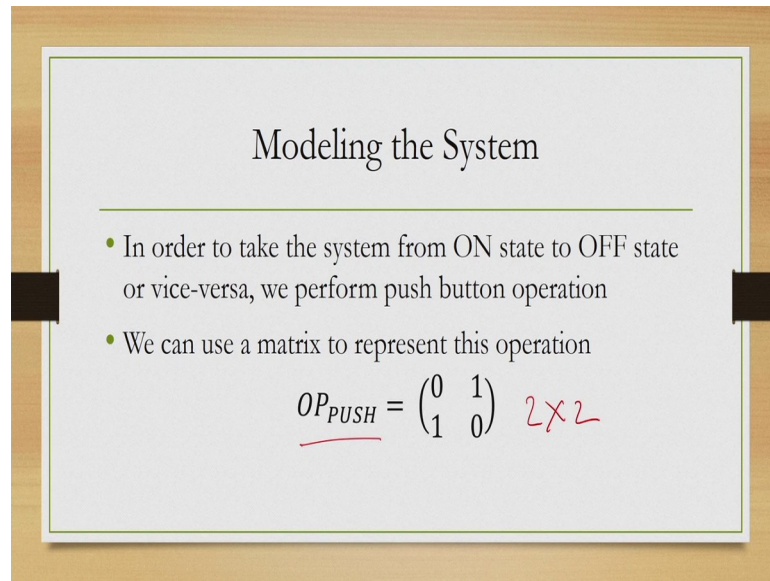
$$STATE_{ON} = (1\ 0)$$
$$STATE_{OFF} = (0\ 1)$$

So, there are two states. Now, each state can be represented with a vector with the one dimensional matrix. There are two states. So, the vector general form of this vector can be something like this, where S 1 is one state, S 2 is the other state. Now, when we have the ON state so, we can set S 1 to be 1 and S 2 to be 0, 1 indicates ON state, 0 indicates OFF state.

So, S 1 indicates the state variable when the state is on. So, in that case we can represent the ON state with these vector 1 0 and OFF state with these vector 0 1 that is a very intuitive way of representing a state. So, each state is represented with two elements in a vector form, one element indicates a particular state of the system and the other element indicates other state of the system.

So, when the state of the system is on, then that particular element which represents that state should be 1 otherwise, it should be 0 when, by this convention we can define the two states in this way on state is 1 0 and off state is 0 1.

(Refer Slide Time: 17:07)



The slide is titled "Modeling the System" and contains the following content:

- In order to take the system from ON state to OFF state or vice-versa, we perform push button operation
- We can use a matrix to represent this operation

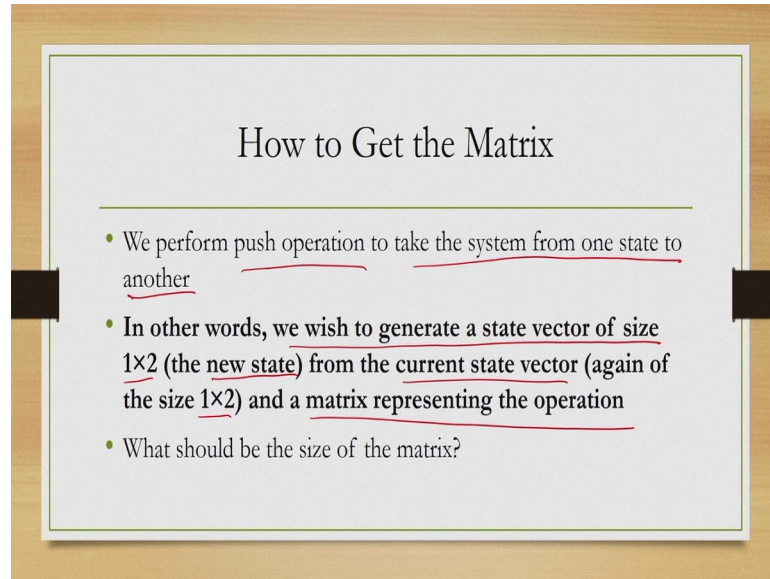
$$\underline{OP_{PUSH}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad 2 \times 2$$

Then the other concern that we have is to represent the transitions that from ON state, how to go to the OFF state and from off state how to come to the on state

So, this transition we can represent in the form of another matrix. See earlier, the states were represented in the form of one dimensional matrices, which are the vectors. The transition also you can represent in the form of a matrix and in our case these matrix will look like this we call it OP PUSH which can be represented as this 2 by 2 matrix where the elements are 0 1 1 0.

So, essentially it represents a push button operations. Now, how we got these matrix that is that require some clarification.

(Refer Slide Time: 17:50)



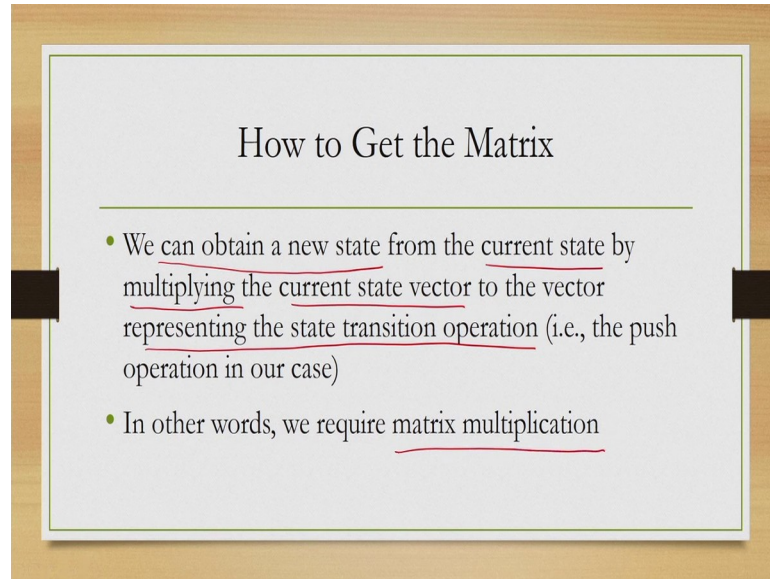
How to Get the Matrix

- We perform push operation to take the system from one state to another
- In other words, we wish to generate a state vector of size 1×2 (the new state) from the current state vector (again of the size 1×2) and a matrix representing the operation
- What should be the size of the matrix?

So, let us go to the basics, what we want we want to perform the push operation to take the system from one state to another. This is same as saying that we wish to generate a state vector of size 1 by 2, the new state, from the current state vector which is again of the size 1 by 2 and the matrix representing the operation.

So, the state vector is of course, as we have seen is size 1 by 2 or a vector with two elements. So, our objective is to get to 1 vector from another vector, using an operation that we want to represent with the matrix. Now, if we formulate this problem in this way, then what should be the size of the matrix that is our point of concern.

(Refer Slide Time: 18:38)



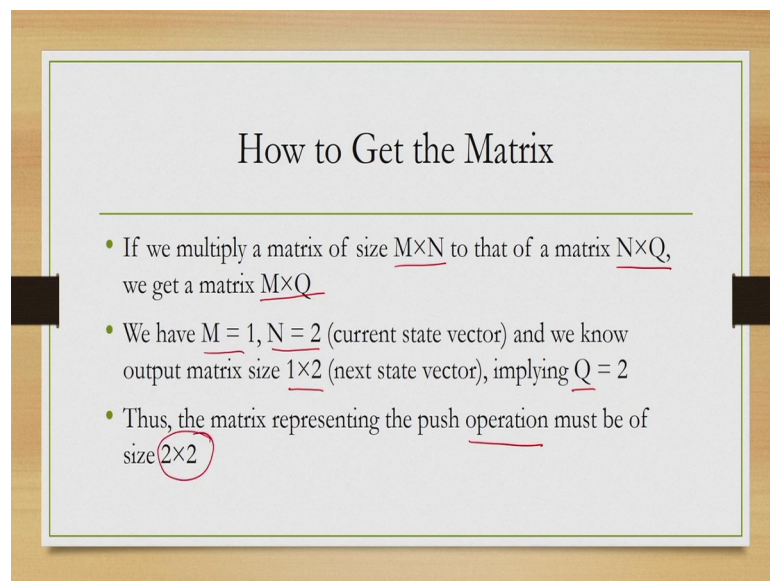
How to Get the Matrix

- We can obtain a new state from the current state by multiplying the current state vector to the vector representing the state transition operation (i.e., the push operation in our case)
- In other words, we require matrix multiplication

You may know if you have the basic knowledge of matrices, that we can obtain a new state from the current state by multiplying the current state vector to the vector representing the state transition operation.

So, for operation we have a matrix and each state is represented with the matrix. So, we multiply these two matrix to get another matrix that represents the next state or the new state. So, this is nothing, but a matrix multiplication operation.

(Refer Slide Time: 19:09)



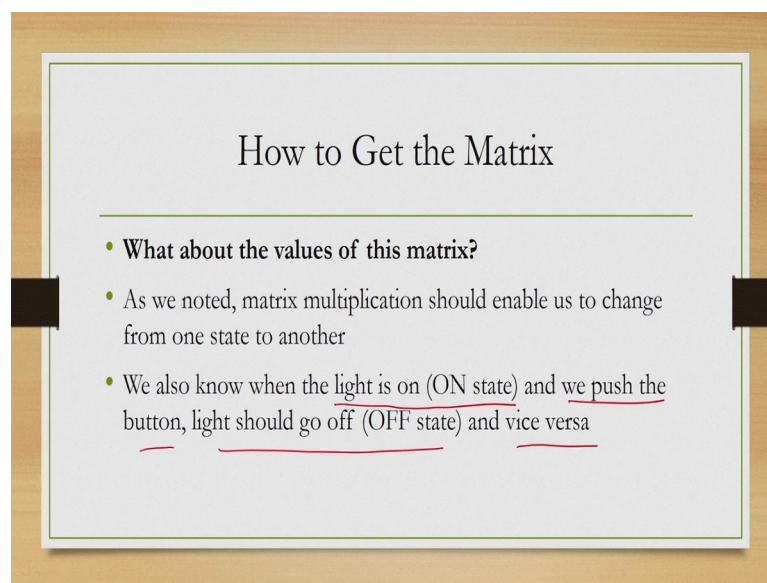
How to Get the Matrix

- If we multiply a matrix of size $M \times N$ to that of a matrix $N \times Q$, we get a matrix $M \times Q$
- We have $M = 1$, $N = 2$ (current state vector) and we know output matrix size 1×2 (next state vector), implying $Q = 2$
- Thus, the matrix representing the push operation must be of size 2×2

Now, we also know that if we multiply, a matrix of size M by N to that of a matrix of N by Q then we get a matrix of M by Q .

Now, in our case we have M equal to 1 and N equal to 2, the current state vector which is having two elements. So, we can have M equal to 1 and N equal to 2 and we also know that the next state vector is having the same size that is 1 by 2. So, in other words Q is also 2. So, then the size of the matrix that represents the operation must be N by Q . Now, we have N equal to 2 and Q equal to 2. So, it must be 2 by 2

(Refer Slide Time: 19:49)



How to Get the Matrix

- **What about the values of this matrix?**
- As we noted, matrix multiplication should enable us to change from one state to another
- We also know when the light is on (ON state) and we push the button, light should go off (OFF state) and vice versa

Now, how to get the values of the elements of this 2 by 2 matrix? We can set up a system of equations to get the values, what we know, when the light is on or in the ON state and we push the button the light should go off and vice versa. So, when we are in the ON state and push the button, then we get the OFF state. Similarly, when we are in the OFF state and push the button we get to the ON state.

(Refer Slide Time: 20:14)

How to Get the Matrix

- We can set up following system of eqn with the knowledge

$$\begin{array}{l} \checkmark \quad \checkmark \quad \checkmark \\ (1 \ 0) \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix} \\ \checkmark \quad \checkmark \quad \checkmark \\ (0 \ 1) \cdot \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \end{pmatrix} \end{array}$$

- Solve to get values of the four variables a, b, c and d

So, then we can say that this multiplication will lead to this state and this multiplication will lead to this state. Now, if we multiply these two matrices in the first case as well as in the second case we will get a and equate it with the terms in the right hand side here, in both the cases, then we will get a series of equations. If we solve those equations, we will get the values of the four elements; abc and d and that is how we got the transition matrix that we have seen before.

(Refer Slide Time: 20:46)

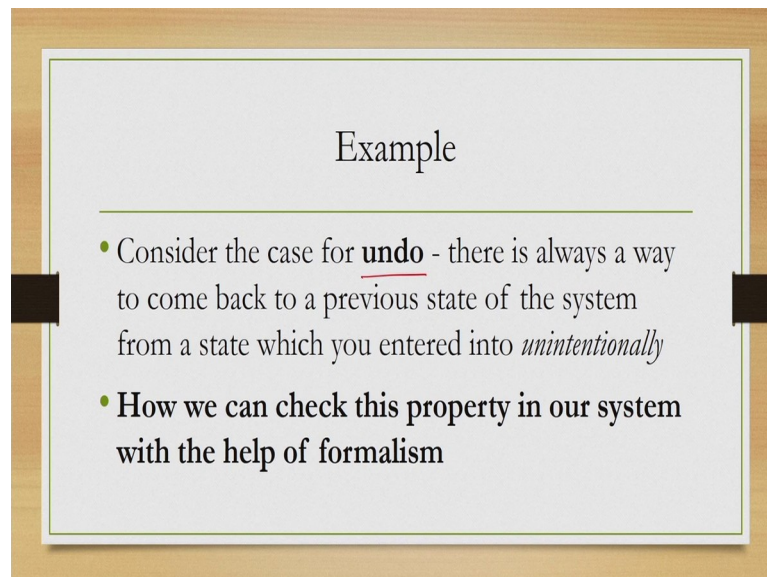
Formalism

- We use matrices to represent system states and operations that change states
- We use the matrix multiplications to compute new states
- Matrices and matrix multiplication represents system as well as its behavior (how state changes)

Now, we can use these matrices to represent the system states and the operations that change states and we use the matrix multiplications to compute the new states. So, these matrices and matrix multiplication represent the system as well as its behavior, how the state changes.

So, that is the formalism that we are using to represent our system and represent its behavior. So, in this case if you may have noticed that we are using a language to represent the formalism, the language is matrices and matrix algebra. Using that language we manage to represent our system and represent its behavior.

(Refer Slide Time: 21:34)



Example

- Consider the case for undo - there is always a way to come back to a previous state of the system from a state which you entered into *unintentionally*
- **How we can check this property in our system with the help of formalism**

Now, with that representation what we can do? How we can use it? In order to understand that we will go through another example, but before that let us just try to see with respect to the current example of our simple system what we can do with this matrix multiplication and matrix representation to understand the behavior of the system, to understand its strengths and its weaknesses.

Let us start with the case for undo, now we all know what this undo operation means. It means that we have entered into a state by mistake and we are we want to come back without any difficulty, whether our system supports that or not we want to verify. And we can check this property whether this property is supported by the system or not by using the formalism that we have used to represent our system.

(Refer Slide Time: 22:22)

Example

- Suppose you pushed the button for the first time
- you push the button again
- What shall be the effect?

Suppose, you push the button for the first time and then you push the button again. So, what shall be the effect? Let us try to set up the equations.

(Refer Slide Time: 22:23)

Example

- A push implies a state transition
- Let the current state be S_C (can be either $STATE_{ON}$ or $STATE_{OFF}$)
- With the first push, we go to a new state (say, S_N which again can either be $STATE_{ON}$ or $STATE_{OFF}$ depending on S_C)

First, a push implies a state transition as we have already mentioned. Let us denote by S_C the current state. Now, S_C can either be the ON state or the OFF state of course. Now, with the first push we go to a new state. Let us denote it by S_N which again, can be either the ON state or the OFF state, depending on the current state S_C .

(Refer Slide Time: 23:04)

Example

$$S_N = S_C \cdot OP_{PUSH}$$

- With second push, we go to another state (S'_N)
$$S'_N = S_N \cdot OP_{PUSH}$$
- Thus,
$$S'_N = S_C \cdot (OP_{PUSH} \cdot OP_{PUSH})$$

So, then what we can say is that S_N is S_C multiplied by the push matrix OP_{PUSH} that we have already derived before.

Now, with the second push we go to another state let us denote it by S_N dash. Now, S_N dash is essentially, S_N into OP_{PUSH} S_N multiplied by OP_{PUSH} . Now, S_N we already know is S_C multiplied by OP_{PUSH} . So, S_N dashed can be simplified as S_C multiplied by OP_{PUSH} multiplied by OP_{PUSH} . So, this OP_{PUSH} is appearing twice. Now, let us see what happens when we multiply these OP_{PUSH} with itself.

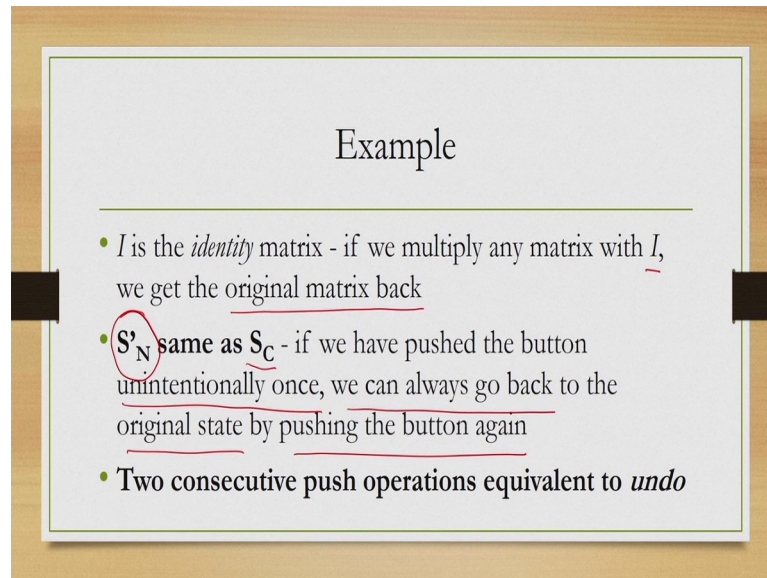
(Refer Slide Time: 23:51)

Example

$$OP_{PUSH} \cdot OP_{PUSH} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
$$= I$$

Earlier, we have seen op push indicates this matrix. Now, we are multiplying it with itself, what we get is I or the identity matrix. What is an identity matrix? It is a matrix that when multiplied with another matrix results in the same matrix.

(Refer Slide Time: 24:11)



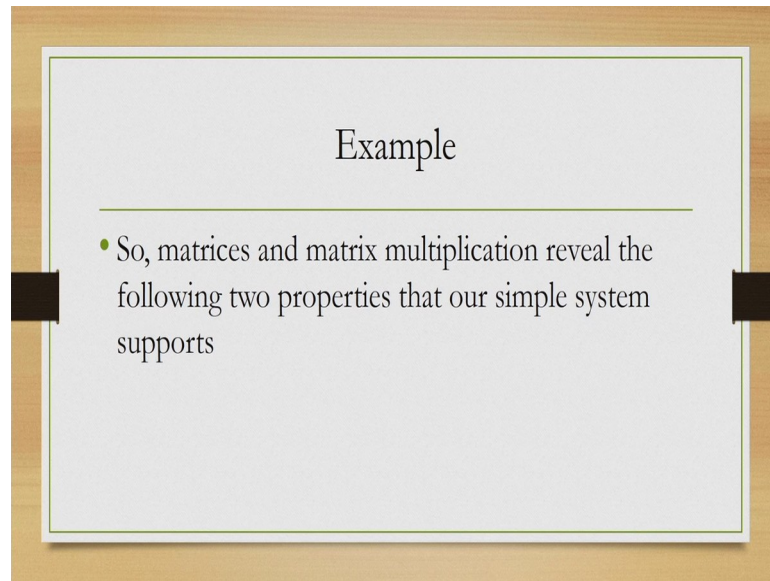
Example

- I is the *identity* matrix - if we multiply any matrix with I , we get the original matrix back
- S'_N same as S_C - if we have pushed the button unintentionally once, we can always go back to the original state by pushing the button again
- **Two consecutive push operations equivalent to *undo***

So, if we multiply any matrix with I , then we get the matrix back, the original matrix back. So, in other words the new state S_N dashed should be same as S_C or the current state when we started, because eventually, it is getting multiplied with I or the entity matrix. So, then what we can say is that if we have pushed the button unintentionally once, then we can always go back to the original state by pushing the button again.

So, suppose we have made an unintentional push and we reached an undesired state. So, we can push it again to go back to the original state.

(Refer Slide Time: 25:03)

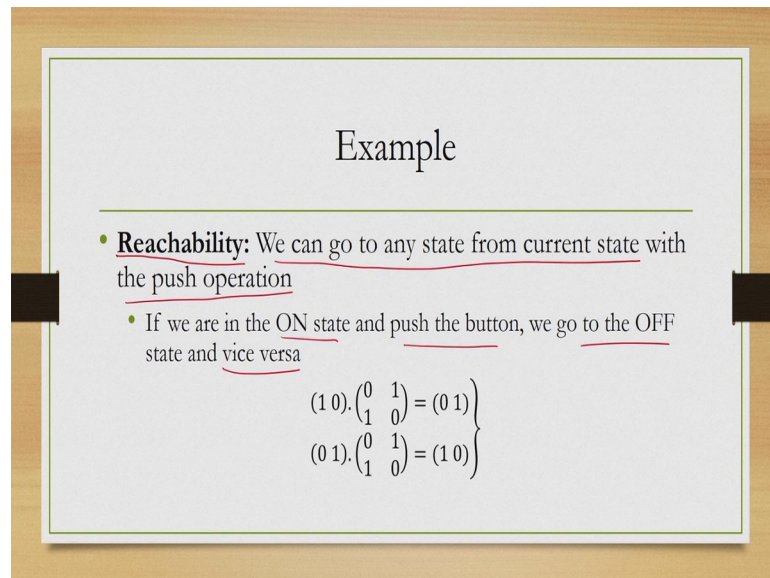


Example

- So, matrices and matrix multiplication reveal the following two properties that our simple system supports

In other words, the system supports undo, which we can establish through this formal analysis, based on the matrix representation, that is the major advantage of using formalism. Here, we can formally establish the existence of certain desirable properties of the system.

(Refer Slide Time: 25:19)



Example

- **Reachability:** We can go to any state from current state with the push operation
 - If we are in the ON state and push the button, we go to the OFF state and vice versa

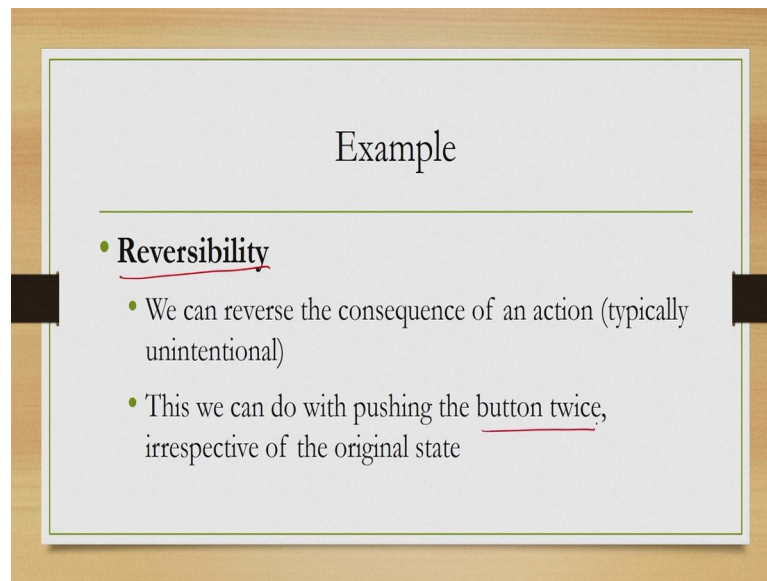
$$(1\ 0) \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = (0\ 1)$$
$$(0\ 1) \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = (1\ 0)$$

So, with this analysis what properties we have verified, we have checked? One is reachability, it states that we can go to any state from the current state with the push operation. For example, if we are in the ON state and push the button, we can go to the

OFF state and vice versa. So, we are in if in the OFF state, if we are in the OFF state and push the button, then we can go to ON state, if we are in the ON state and push the button we can go to OFF state.

So, from any state we can go to the other state, because this is a very simple system with only two states. Nonetheless, it shows that the system supports reachability, reaching to any state from any other state.

(Refer Slide Time: 26:04)

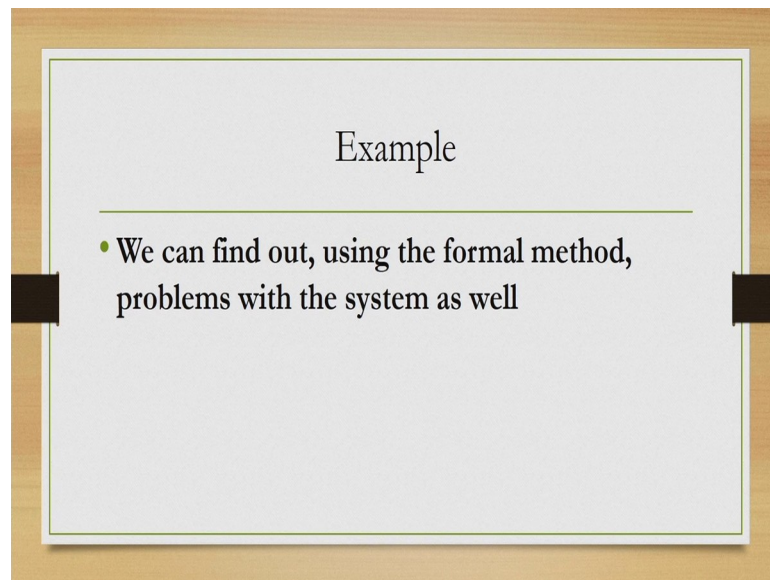
A slide titled "Example" with a light gray background and a thin green border. The title "Example" is centered at the top. Below the title is a horizontal line. Underneath the line is a bullet point with the word "Reversibility" in bold and underlined. Below this are two more bullet points: "We can reverse the consequence of an action (typically unintentional)" and "This we can do with pushing the button twice, irrespective of the original state". The slide is set against a wooden background with two black rectangular markers on the left and right sides.

Example

- Reversibility
 - We can reverse the consequence of an action (typically unintentional)
 - This we can do with pushing the button twice, irrespective of the original state

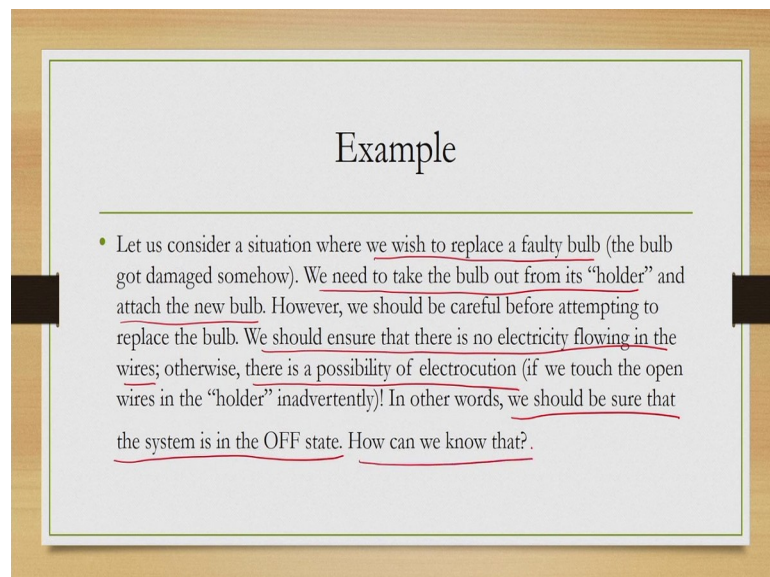
It also supports reversibility or undo that we have already seen. We can reverse the action by pushing the button twice. So, in order to undo some undesired action, we need to push the button twice, if we do that then we have undone the earlier, action which is called reversibility property of the system.

(Refer Slide Time: 26:24)



So, that is one side that with the help of the formal representation, we can actually check for the existence of desirable property in the design that we have come up with. The other side is that with the help of this formal system, with the help of the same formal representation we can actually test for problems that maybe there in the system.

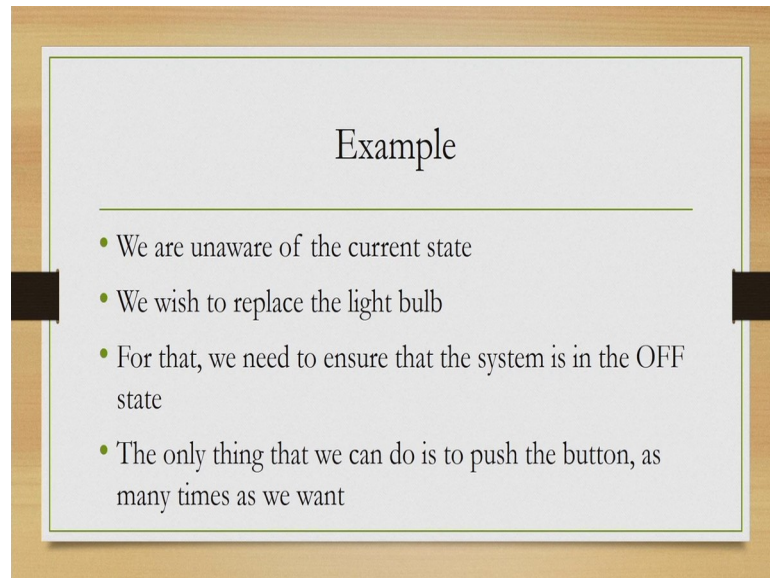
(Refer Slide Time: 26:55)



Let us see in terms of an example, how that is done, let us consider a situation. In this situation we wish to replace a faulty bulb. Now, we need to take the bulb out from its holder and attach the new bulb. So, before doing that we should ensure that there is no

electricity flowing in the wires otherwise, there is a possibility of getting electrocuted. This is same as saying that we want to be ensure that the system is in the OFF state before we can replace the bulb.

(Refer Slide Time: 27:38)



Example

- We are unaware of the current state
- We wish to replace the light bulb
- For that, we need to ensure that the system is in the OFF state
- The only thing that we can do is to push the button, as many times as we want

Can we be knowing that with certainty? Is there a possibility that we can go to a OFF state with certainty? Now, suppose we assume further that that we do not know of the current state, before we attempted to replace the bulb. So, in that case what we can do? We can at most push as many times as we wish. Our problem is; can we ensure by pushing that we can reach to a state which is safe or we can reach to the OFF state.

(Refer Slide Time: 28:17)

Example

- Let us assume that the button is pushed n times
- Does there exist any number n such that the Eq. is satisfied

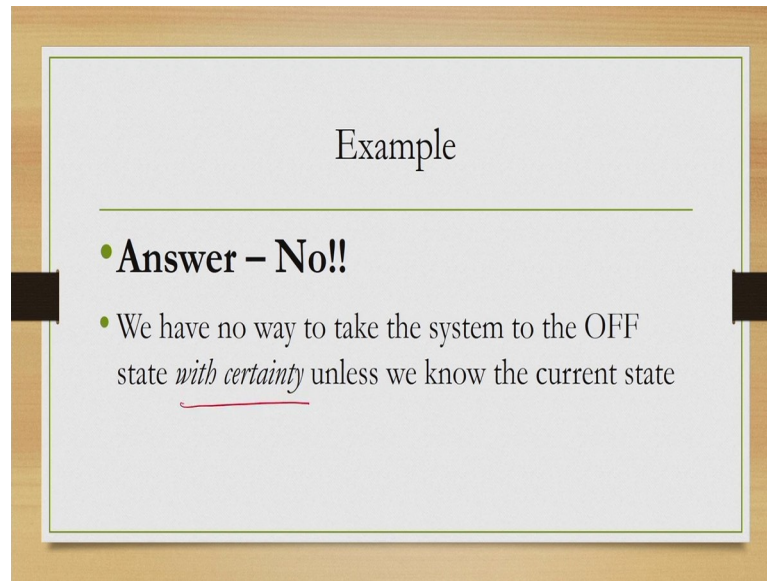
$$S_C \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^n = \underline{\underline{(0 \quad 1)}}$$

Mathematically, this problem we can formulate in the following way that we have a current state which we do not know and we are pushing the button n times. In other words, this matrix gets multiplied n times with the intention that we reach to the OFF state.

So, what is the suitable value of n such that we can always ensure that irrespective of the current state we can reach to the OFF state. So, if n equal to 1 and we do not know current state, can we reach to the OFF state, if n equal to 2 and we do not know current state, can we reach to the OFF state and so on you can try with different values of n and see whether that is possible. So, the current state can be either ON state or OFF state

So, assuming current state is ON state and n equal to 1 that is one thing and assuming that the current state we do not know and n equal to 1 that is another thing. So, you can try to figure out whether you can find a value of n so that you can say with certainty that irrespective of the current state whether we are in the on state or off state we can always reach to the off state with so many number of pushes.

(Refer Slide Time: 29:25)

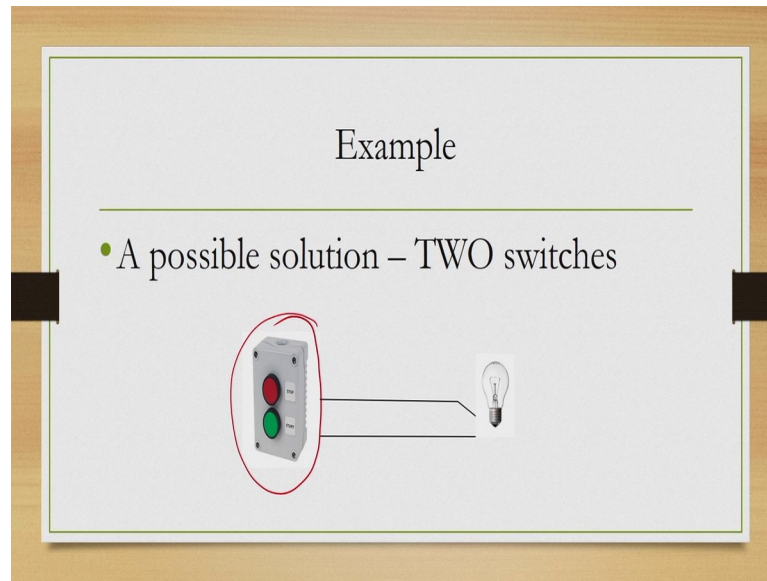


Unfortunately, we cannot say that. The answer is no, we cannot have a value of n which ensures that irrespective of the current state, we can always reach to a safe state or OFF state in this case. So, then unless we know the current state we cannot know which state it will take us to with certainty, that is a problem. Unless, we know that we are in the OFF state, we cannot change the bulb.

Now, if the bulb gets damaged so, we do not know whether, we are actually in the ON state or OFF state, because the bulb is not getting switched on. So, we may be pushing the button properly, but since the bulb is not getting switched on, because it is damaged so, the system is never going to the on state and unless we replaced the bulb we cannot bring the system back into its regular behavior and so, the entire system will collapse. Unfortunately there is no way for us to know when the system is in the off state irrespective of the number of push that we may perform this simple system with single push button device actually cannot resolve this problem.

So, this is an inherent structural problem of this system with single push button device unless, we know the current state we cannot know the next state with certainty irrespective of the number of push that we perform.

(Refer Slide Time: 30:45)

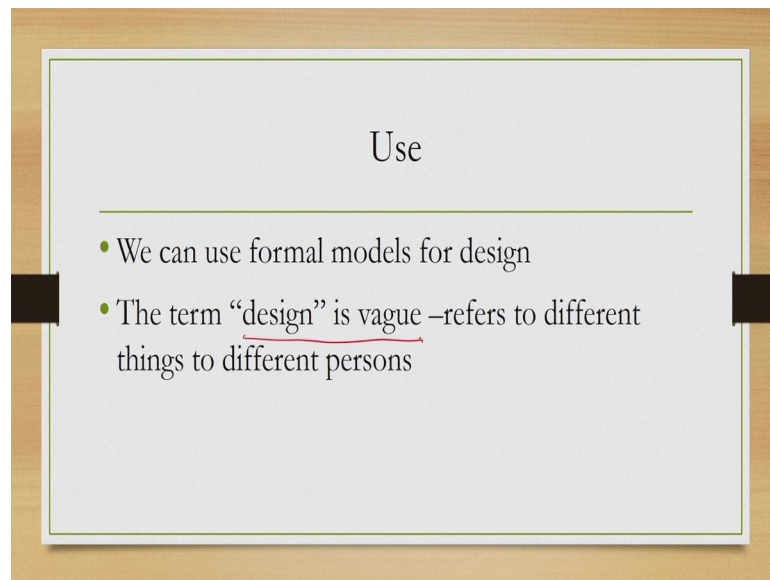


However, we can resolve this issue by modifying the system. Instead of having a single push button device, if we replace it with a system with two push button devices, then we can always be sure of the system state, because now, we have two separate buttons for two separate states.

So, each button represents one state. So, we can always press that button to take the system to that particular state and we do not need to rely on the output of the light bulb. So, we remove the uncertainty and we can resolve this issue. So, the idea that we wanted to illustrate here is that with the representation of the system in terms of matrix and matrix multiplication, we can actually figure out this problem and we can come up with a solution once, the problem is identified.

Although, the solution is not dependent on the formal representation or the formal model, but the model helps us in identifying the problem and that is actually good for us, because once we are able to identify the problem, we can think of a solution and come up with the solution in most of the cases. So, then we can actually try to understand in a better way what are the benefits and challenges in the in the case of formal representation of interactive systems.

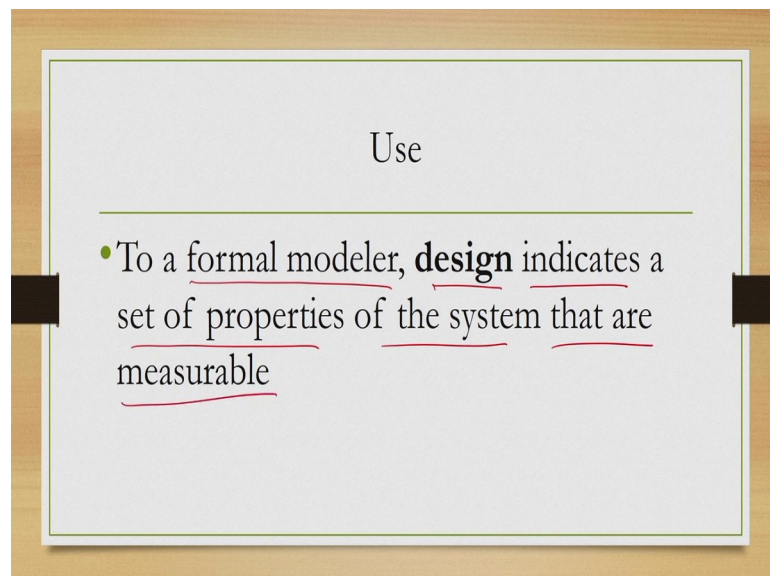
(Refer Slide Time: 32:08)



So, earlier we have tried to explain it in terms of a very simple system which is having only two states and equally small number of transitions. Now, we are going to discuss it in more detail, the benefits of using such a formal representation and the challenges involved. So, the overall thing that we want to achieve with a formal representation or a formal model of a system is to design the system.

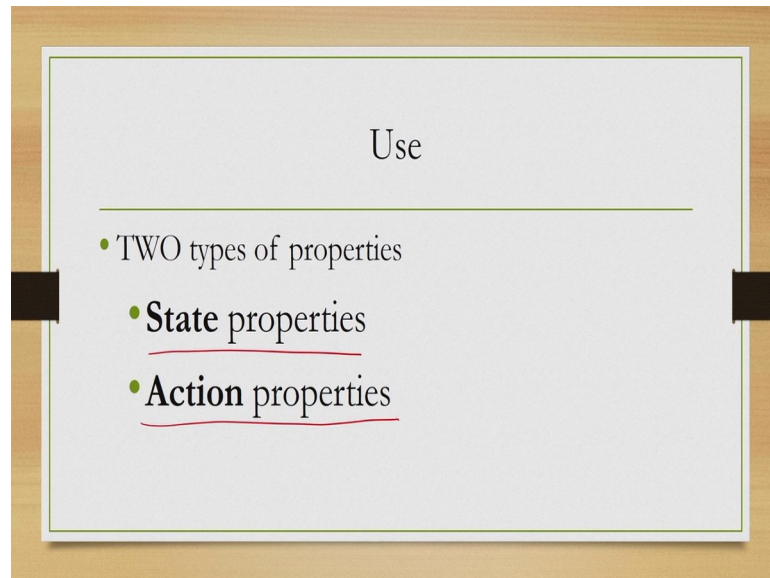
Now, design is a very vague term and earlier, also we mentioned in one of the previous lectures that it refers to different things, different concepts to different persons.

(Refer Slide Time: 32:47)



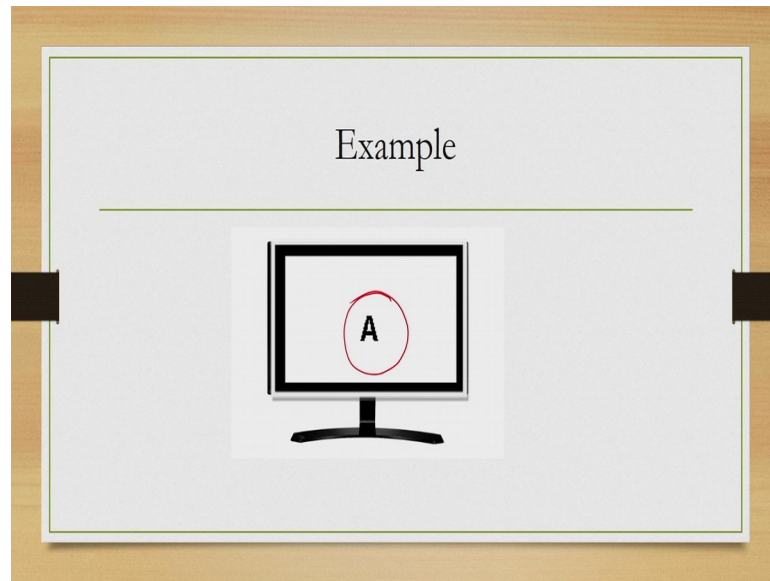
Now, in the context of formal system design to a formal system modeler, the term design actually indicates a set of properties of the system that are measurable and these properties are also indicative of the good or bad nature of the design.

(Refer Slide Time: 33:07)



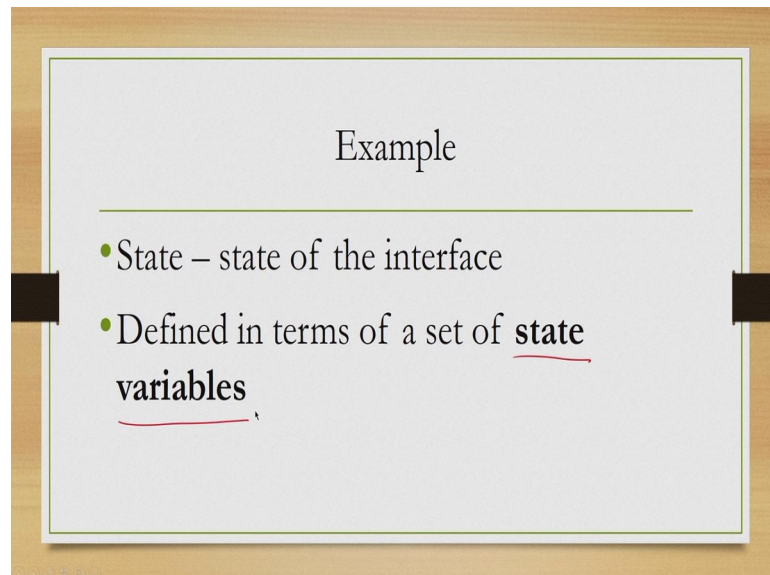
Now, these properties we can broadly classify into two categories; a set of property belongs to the state properties, properties of the states of the system and another set of properties can belong to the action properties. Let us try to understand, these two concepts with another more detailed example with more number of states than the one we have seen earlier.

(Refer Slide Time: 33:31)



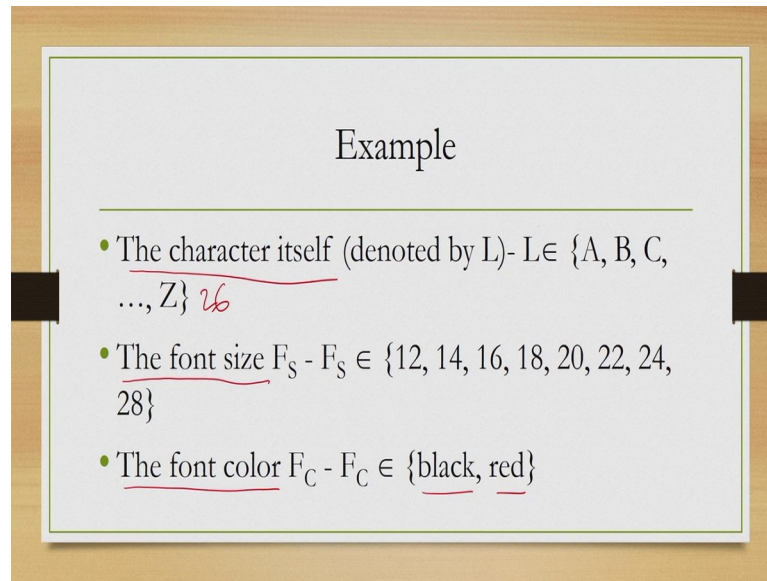
Consider again, a very simple system although this term simple. Here, maybe a bit deceiving as we will soon see the system is not that simple at all. Now, in this system we have only a simple interface which shows only one letter of the English alphabet in capital form.

(Refer Slide Time: 33:55)



Now, we can define the state of the system in terms of a set of state variables, that is typically what we do to define the state of a system.

(Refer Slide Time: 34:07)



Example

- The character itself (denoted by L)- $L \in \{A, B, C, \dots, Z\}$ *26*
- The font size F_S - $F_S \in \{12, 14, 16, 18, 20, 22, 24, 28\}$
- The font color F_C - $F_C \in \{\text{black}, \text{red}\}$

Now, in this case there are three variables; one is the variable representing the character, one is the variable representing the size of the font and the third variable represents the color of the font.

Now, for simplicity we are assuming that the character can take value from the English alphabet of capital letters. So, there are 26 letters in the alphabet and the character can take any of these values. The size of the font again is a variable which can get a value from this set of predefined values. So, we are defining 8 font sizes and this font size variable can take any one of these 8 values font color. There are only two values black and red and this font color variable can take either of these values.

So, then total how many states are there in the system? For character, there are 26 possible values, for font size there are 8 possible values and for font color there are 2 possible values.

(Refer Slide Time: 35:33)

Example

- A state can be defined as $S(L, F_s, F_c)$
 - i.e., a state is an instantiation of the state variables with permissible values
- E.g., $S(A, 20, \text{black})$

So, total number of states can be 26 into 8 into 2. So, then what each state indicates? It is a three tuple. In this form L F S and F C L is the character FS is the font size and F C is the font color, when we instantiate, these tuples with specific value we get a particular state for example, S A 20 black can be a state of the system. So, here L is A, font size is 20 and font color is black.

So, once we instantiate, these variables with values we get a particular state and total number of possible states is 416, because for each value we can get one state.

(Refer Slide Time: 36:20)

Example

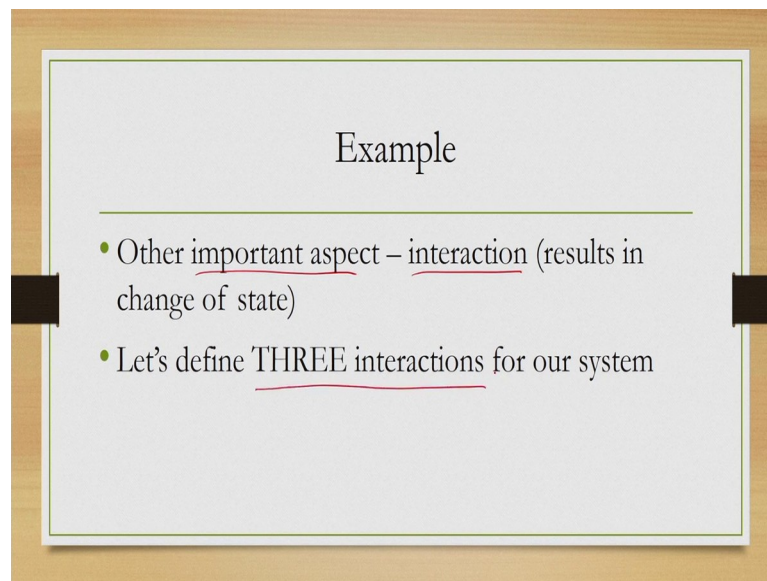
- We can have $26 \times 8 \times 2 = 416$ states for our simple system
- States are based on our visual perception of the interface rather than anything that goes on inside the computer (e.g., the values in the registers and the main memory)

So, there are total 416 combination of values for the three variables. Another thing you can note here is that this states that we have defined are defined in terms of our perception of the interface, rather than how things are happening inside the system.

So, whatever we get to see we are defining that in terms of state variables and values. The three variables that we have defined for our state namely; the character, the font color, and font size are all perceivable all these things we can perceive. So, we are not bothered about the memory content or the register content, for displaying a particular character with a particular font or size to define a state.

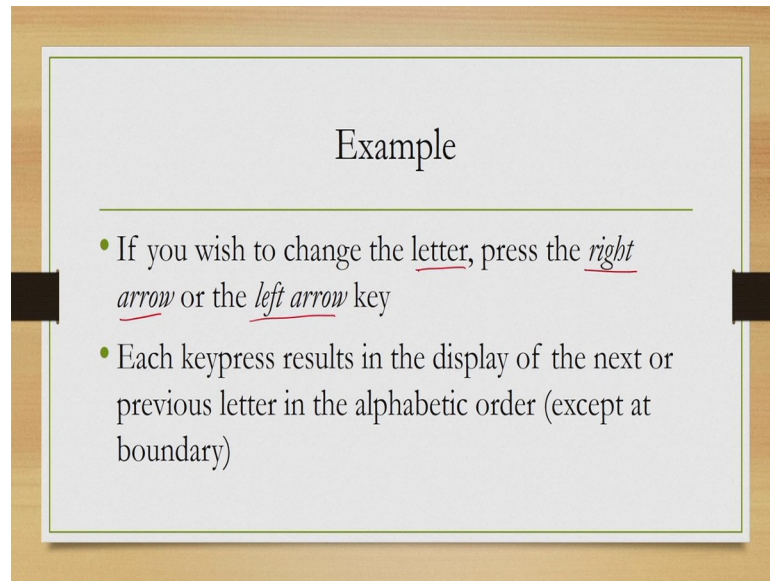
We are defining the state purely in terms of perceptible qualities of the system and that is what we typically do in the context of user centric system design.

(Refer Slide Time: 37:30)



Now, apart from state there is another important aspect that is the interaction between states or in other words change of states. We can actually define our interactions, but for the simplicity, let us define three basic interactions for our system.

(Refer Slide Time: 37:45)



The slide is titled "Example" and is set against a light gray background with a thin green border. It is presented on a wooden-textured surface with two black rectangular markers on the left and right sides. The text is as follows:

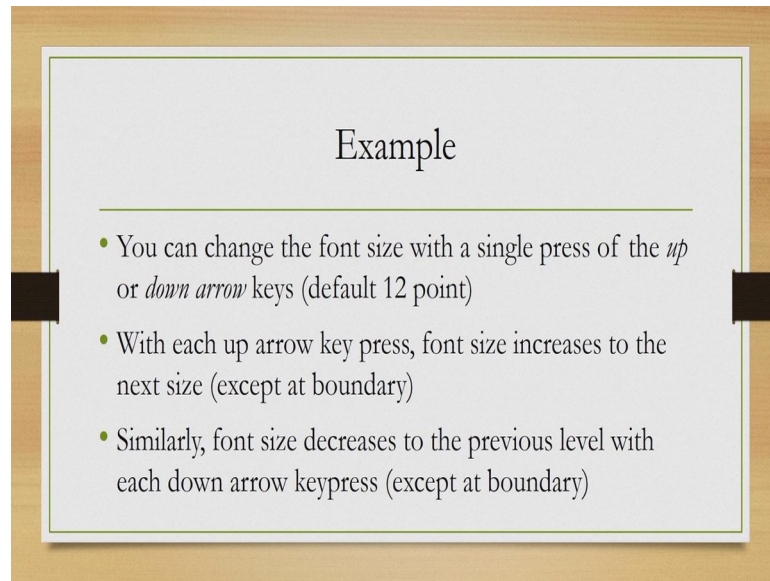
Example

- If you wish to change the letter, press the right arrow or the left arrow key
- Each keypress results in the display of the next or previous letter in the alphabetic order (except at boundary)

First thing is we want to change the letter and we define an interaction for that our proposed interaction is press the right arrow or the left arrow. So, when you press the right arrow, the display changes to the next letter in the alphabet. Suppose, if currently a is displayed and you press the right arrow b will be displayed suppose, currently b is displayed and if you press the left arrow, then a will be displayed. So, by pressing of the right arrow next character in the alphabet gets displayed and pressing of the left arrow displays the previous character.

Now, of course, if you are at the boundary say, you are you are currently seeing z and you press the right arrow then either nothing may happen or you may define the interactions such that the next character is the first character of the letter, the circular list you can implement. Similarly, if you are at a and you press the left arrow either you can display z or you display nothing. So, no change takes place, that is for changing the letter.

(Refer Slide Time: 38:46)



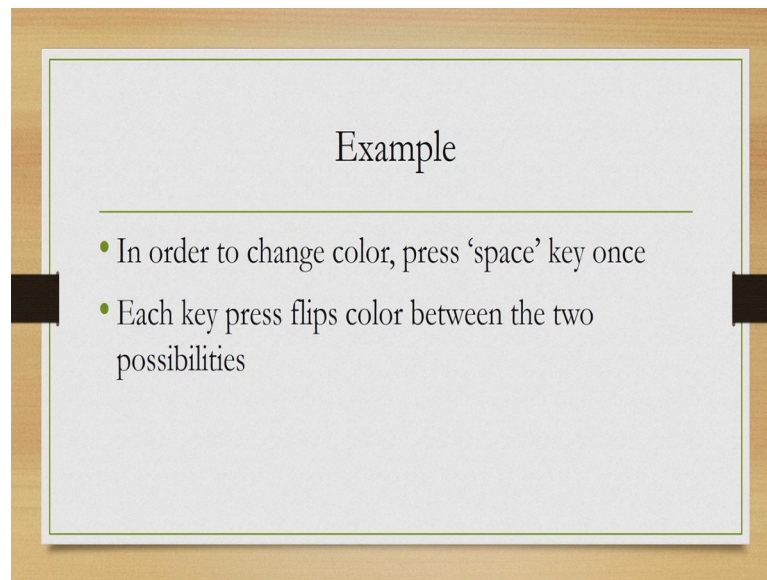
Example

- You can change the font size with a single press of the *up* or *down arrow* keys (default 12 point)
- With each up arrow key press, font size increases to the next size (except at boundary)
- Similarly, font size decreases to the previous level with each down arrow keypress (except at boundary)

To change the font size, we can use another set of keys that is up key and down key. So, if I press the up key then the font size changes to the next size. So, for example, current size is 12 and I press the up key, then it becomes 14 and so on.

Similarly, if I press the down key then it gets changed to the previous size that is if current size is 16 and I press the down key, then it becomes 14. So, it reduce with press of down key and increases with press of up key, except at the boundaries. Similar things we can do at the boundaries as we just mentioned as we just mentioned in the case of changing of letters.

(Refer Slide Time: 39:37)

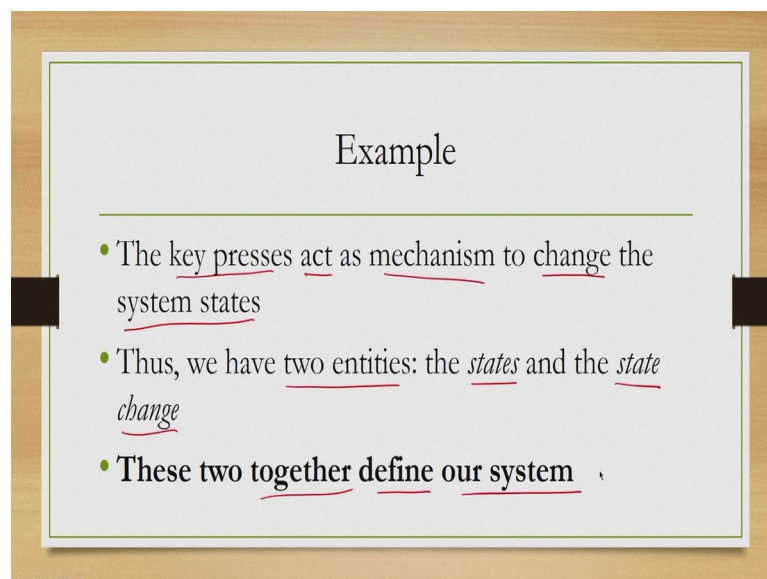


The slide is titled "Example" and is set against a light gray background with a thin green border. It is presented on a wooden surface with two black binder rings on the left and right sides. A horizontal line is drawn below the title. The content consists of two bullet points:

- In order to change color, press 'space' key once
- Each key press flips color between the two possibilities

Thirdly, if we if we want to change the color, we can use the space key. So, by one press of the space key, we can flip the colors. So, if currently black color is displayed and we press the space key, then it becomes red and vice versa.

(Refer Slide Time: 39:56)

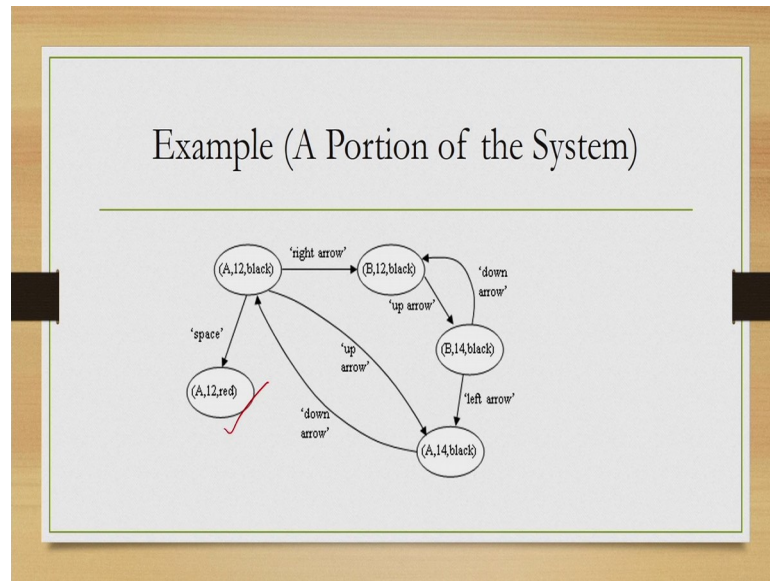


The slide is titled "Example" and is set against a light gray background with a thin green border. It is presented on a wooden surface with two black binder rings on the left and right sides. A horizontal line is drawn below the title. The content consists of three bullet points, with several words underlined in red:

- The key presses act as mechanism to change the system states
- Thus, we have two entities: the states and the state change
- These two together define our system

So, the key presses act as mechanisms to change the states. So, essentially there are two entities; one is the states and the other is the state change. Together these entities define our system. So, in our system we have states and state changes, together they represent our system.

(Refer Slide Time: 40:16)



Here, we are showing a portion of the overall system. Remember that there are 416 states and equally large number of transitions. So, entire thing is not possible to be shown on a limited display area, but just to give you some idea, we are showing you a portion of it and this portion only five states are listed and seven state transitions are shown.

So, as you can see the arrow keys are state transitions and the ovals are the states. So, each state is labeled with the values of the state variables and the transitions are labeled with the particular action that took place.

(Refer Slide Time: 40:56)

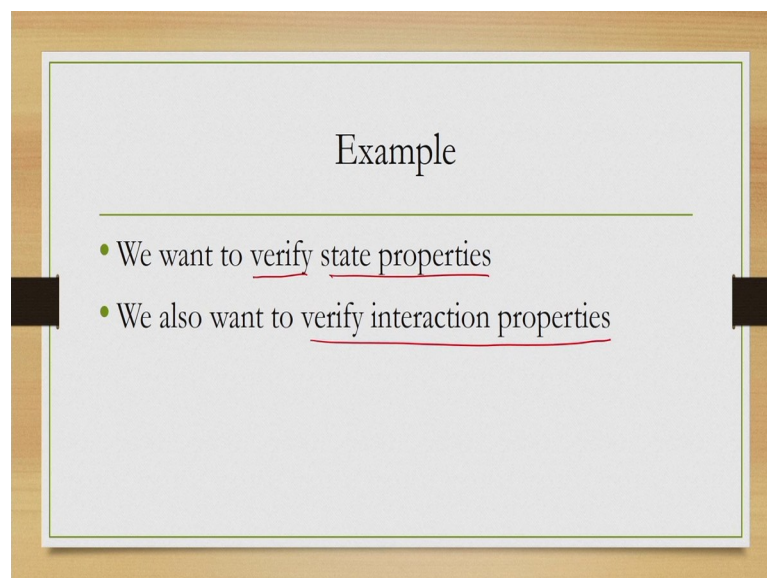
Example

- How can we check if it's going to behave the way we want?
 - Note the large number of states and transitions
- Formal verification helps

So, what we can do with this representation? With this representation of the system, we can check if the system is going to behave the way we want, but again the issue is that we have a large number of states now, more than 400 and large number of interactions.

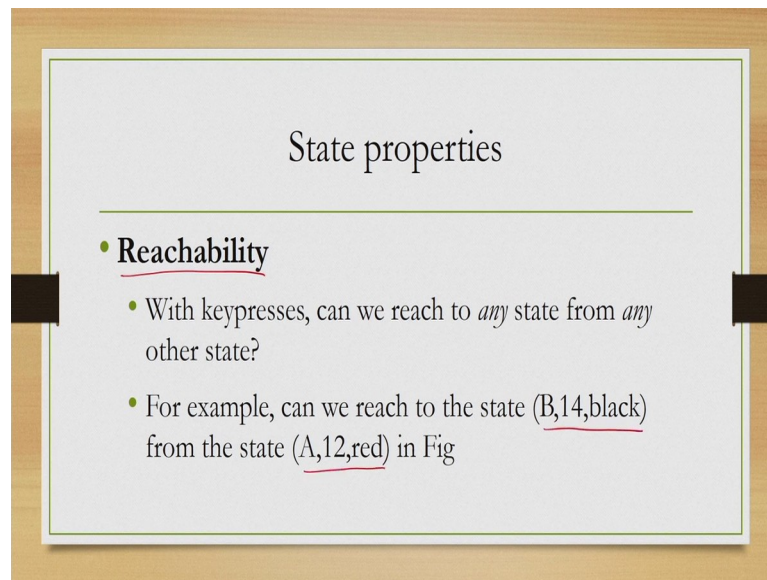
So, how can we check manual checking is difficult, because if we try to draw this entire state transition diagram on paper or a screen then it will become so clumsy and complex that we are likely to miss many properties, because of this complexity of display instead, we can go for automatic verification of this Texan state transitions, through the means of formal verification.

(Refer Slide Time: 41:46)



So, two things we want to verify; verify the state properties and verify the interaction or action properties. So, remember our system is characterized by the states and transitions or interactions and each of these has setup properties, we want to verify both.

(Refer Slide Time: 42:06)



The slide is titled "State properties" and is presented on a light gray background with a thin green border. Below the title is a horizontal line. A single bullet point is listed under the heading "Reachability", which is underlined. The bullet point contains two sub-points, each starting with a small green dot. The first sub-point asks a general question about reaching any state from any other state using keypresses. The second sub-point provides a specific example, asking if the state (B,14,black) can be reached from the state (A,12,red) in a figure.

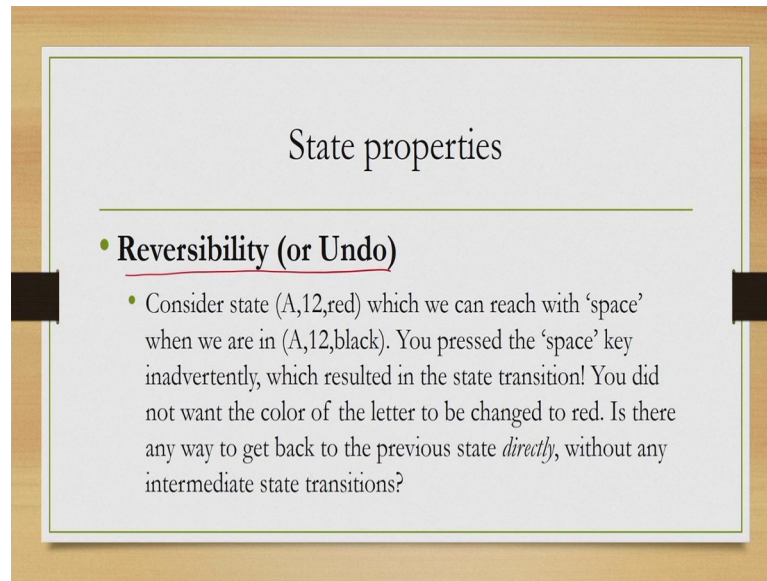
State properties

- Reachability
 - With keypresses, can we reach to *any* state from *any* other state?
 - For example, can we reach to the state (B,14,black) from the state (A,12,red) in Fig

So, what are there in state properties? Already we have discussed couple of those before; one is reachability, we want to reach to any state from any other state. For example in the portion we have shown suppose that is our entire system. So, there we want to check, if we can reach to the state B 14 black from the state A 12 red, can we do that?

So, in this simple system, if this entire portion is the actual system so, we want to reach to the state suppose here and we are here, can we reach there, if we follow the transition, we can see that it is not possible, but if the graph is much more complex than this one then manual inspection is difficult and there we can automatically check for it through formal verification.

(Refer Slide Time: 42:55)



State properties

- **Reversibility (or Undo)**
 - Consider state (A,12,red) which we can reach with 'space' when we are in (A,12,black). You pressed the 'space' key inadvertently, which resulted in the state transition! You did not want the color of the letter to be changed to red. Is there any way to get back to the previous state *directly*, without any intermediate state transitions?

The property that we can try to verify is reversibility or undo again, we have discussed it. So, by mistake if we enter a state and do we want to come back to the original state, does our design support that reversibility or undo operations? We can verify that also with this formal notation.

In this case by mistake, if we enter this state A 12 red from the state A 12 black, can we actually go back to this A 12 black, because the space key I pressed by mistake and I want to rectify my error.

So, can I do that directly without any intermediate transitions from A 12 black to sorry, from A 12 red to A 12 black, can we go back; that means, here I was here at this state A 12 black.

Now, by mistake I press the space key and I entered this state A 12 red. I want to check if from here, if I can go back to this original state in this portion of course, it is not possible, but this is only a portion of the whole system and in the whole system, I may see whether I have designed a path from this state to from this state to the state, if not then there is a problem with reversibility property and if yes, then it supports the reversibility property.

(Refer Slide Time: 44:21)

State properties

- **Dangerous (unsafe) states**
 - Sometimes, there can be states that are “unsafe” (sometimes referred to as “dangerous”). We do not want to be in those states, at least easily or frequently. The system might behave unpredictably once we reach to those states (due to faulty design). Sometimes, the states may not support other properties (e.g., reachability or reversibility). Hence, we may be stuck in that state forever! For example, if Fig. represents a complete system, the state (A,12,red) may be categorized as an unsafe state. We cannot do anything once we are in this state (neither go to any other state nor return to the previous state). Thus, it is important to know the presence of such unsafe states and the possible paths to reach those.

The third state property that we want to check for is the presence of dangerous or unsafe states. Sometimes there can be states that we do not want to reach or at least we do not want to reach easily or without effort, because those states may actually refer to a situation where it is difficult to get back to any other state or the system behavior is not the one that we expect, the states may not support properties which are desirable like reachability or reversibility and the system may behave unpredictably once, we reach those states.

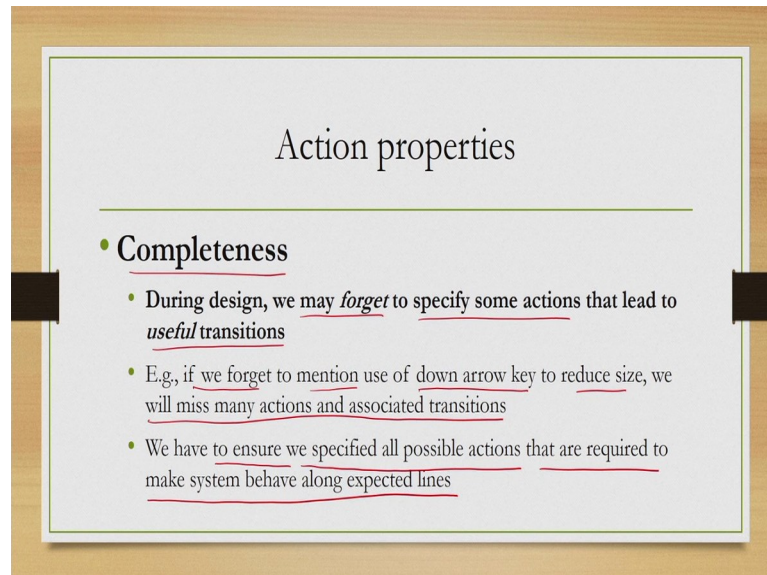
Now, sometimes it may happen by design, the designer wants the presence of such states for various reasons. So, in that case it is not an issue, but if that is not the case and the such states appear due to faulty design then it is necessary to check for those states and identify the presence of those states. So, that we can rectify the design, let us see for example, the state A 12 red.

Now, this is the state. So, once we reach here we cannot do anything else, if we assume that this entire portion actually represents the whole system. Of course, this is only a portion and there may be paths from this state, which we are not showing in this portion, but if we assume that this is the complete system then clearly, this is a dangerous state A 12 red. From here we cannot do anything else.

Once, we reach here we cannot go back. So, the entire system stops now, that may happen by design or that may happen by mistake and our objective is to check if it

happened by mistake. So, that we can rectify our design and determination of this answer for dangerous states is another activity that can be added with the use of formal models.

(Refer Slide Time: 46:32)



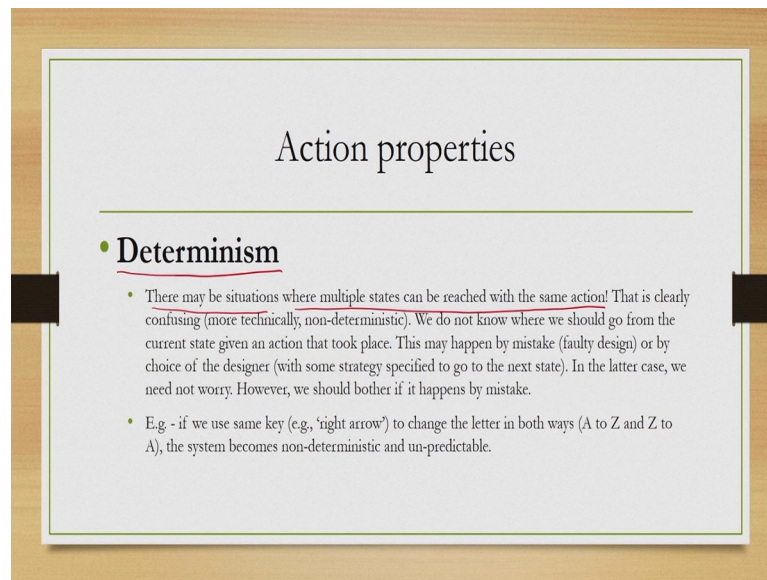
The slide is titled "Action properties" and is set against a light gray background with a thin green border. Below the title is a horizontal line. The main content is a bulleted list under the heading "Completeness".

- **Completeness**
 - During design, we may forget to specify some actions that lead to useful transitions
 - E.g., if we forget to mention use of down arrow key to reduce size, we will miss many actions and associated transitions
 - We have to ensure we specified all possible actions that are required to make system behave along expected lines

Now, those are three major state properties that are typically checked for by formal notations. There are couple of major action properties also that we can check with formal models; one is completeness, what it indicates? During design we may forget to specify some actions that lead to useful transitions for example, if we forget to mention that we need to use the down arrow key to reduce size.

So, then we will miss many actions and associated transitions in the example of course, there are very few interactions, but in large systems with many interactions, it is quite possible to actually forget to include many such interactions which may lead to useful transitions and in such cases, it is necessary to find out whether we made that mistake, whether we have made that mistake whether we have forgotten to add some useful actions. Our objective is that to ensure is specified all possible actions that are required to make the system behave along the expected lines, that is the property of completeness which is a property of the interaction.

(Refer Slide Time: 47:52)



Action properties

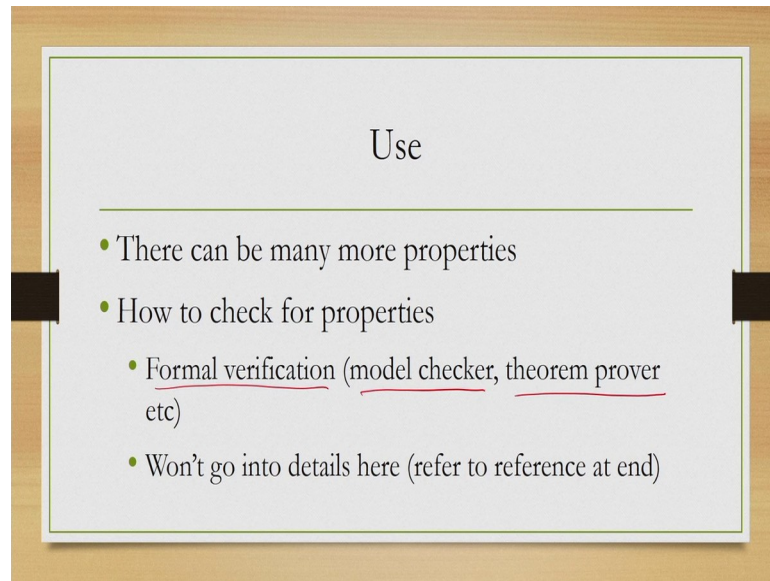
- **Determinism**
 - There may be situations where multiple states can be reached with the same action! That is clearly confusing (more technically, non-deterministic). We do not know where we should go from the current state given an action that took place. This may happen by mistake (faulty design) or by choice of the designer (with some strategy specified to go to the next state). In the latter case, we need not worry. However, we should bother if it happens by mistake.
 - E.g. - if we use same key (e.g., 'right arrow') to change the letter in both ways (A to Z and Z to A), the system becomes non-deterministic and un-predictable.

The other property is determinism. Now, sometimes there may be situations where multiple states can be reached with the same action. Now, that is confusing. So, you perform a an interaction and you do not know what the next state will be, because multiple states can be reached based on that action again this may be by design, the designer, because of various reasons may have designed the system in a way such that this happens technically this is called non determinism.

We cannot determine with certainty what will be the next state even the current state and action, because multiple states can result from the current state based on the given action, if it is by design of course, we do not have any issue, but if it is not by design, if it is by mistake, then there is a problem we need to check for such mistakes that is a property of the interaction, whether the interaction or whether the actions that we perform are deterministic and if they are found to be non deterministic, we have to ensure that those are by design if not then we have to take some corrective action.

So, in summary what we can do is basically check for these properties namely state properties. So, we have mentioned three major properties of the state namely reachability, reversibility and dangerous states and action or interaction properties two major properties we have mentioned; one is determinism, other one is completeness.

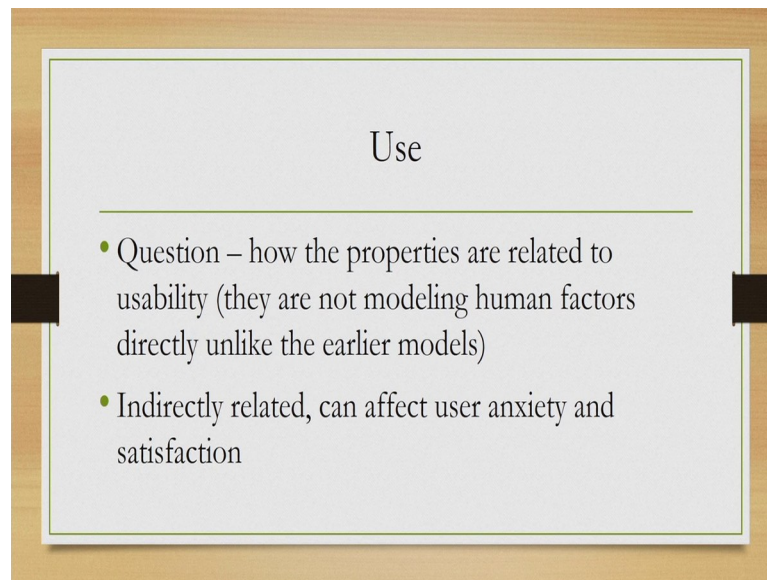
(Refer Slide Time: 49:36)



These are the most well known properties there can be many other problems specific properties of the states and actions that we may like to identify verify that is possible. So, formal notation actually, allows us to check for the presence of these properties and how we can check it will not going to the details of that property checking mechanisms.

So, there are actually techniques that comes under formal verification of such properties namely, model checking techniques, theorem proving techniques or equivalence checking techniques. These are somewhat complicated and we did not go in to the details of such mechanisms. However, if you are interested so, you may refer to the reference that will be mentioned at the end of this lecture for more details.

(Refer Slide Time: 50:35)



So, once we manage to identify these properties or verify these properties. We can rectify our design, but there is a fundamental question here. These properties are not directly related to our cognition. Unlike, in the previous cases, the previous models that we have seen there we actually tried to predict something which is directly related to human cognition and as a result, they are related to human factors.

Here, the properties that we are talking of namely reachability, reversibility, determinism or completeness or dangerous states, these are not directly related to our cognitive activities or mental activities.

So, then if we verify for these properties how can I be sure that the resulting system will be usable or it will have more usability compared to systems that do not support one or more of these properties that is a very valid question and that is a fundamental question. Now, if you can recollect from our earlier discussions at the beginning of these discussions on usability, we mentioned that sometimes we specify usability in terms of direct measures sometimes indirect measures.

Now, the properties that we are discussing are essentially indirect indicators of usability. So, if for example, if we do not support reversibility in our system the soon, the user will get frustrated, because to be human people are likely to make mistakes and if they are unable to come back from those erroneous situations, then they will become frustrated

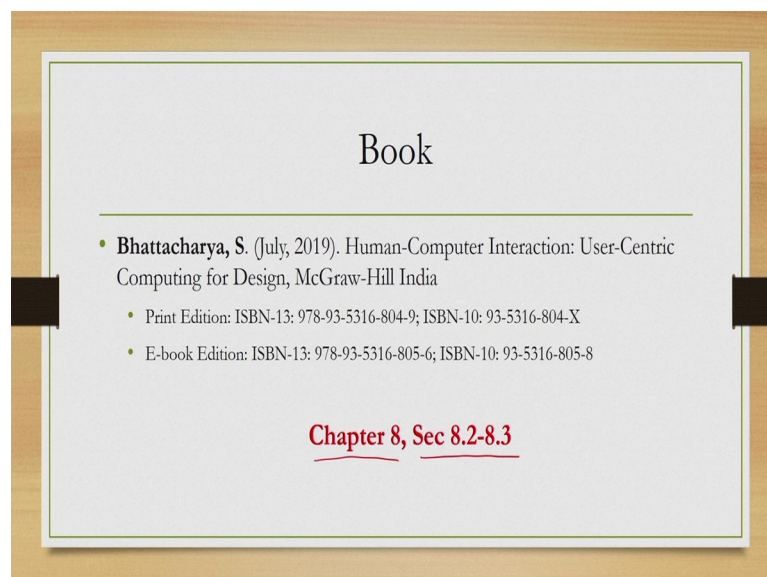
annoyed with the system, which will affect the overall satisfaction level of the user which in turn is going to affect usability.

Same argument holds for other properties, most of the other properties that we have highlighted. So, essentially these properties are related to the human factors in an indirect way and since, they are in directly related to human factors, we can always argue that if these properties are satisfied in a system, then the overall system is likely to have more usability than a system which does not support these properties. So, that is the overall idea.

Now, to recap what we have learnt today is essentially an introduction to the idea of formal models, the notion of formal representation in terms of few case studies and the use of such models in the context of user centric system design, which essentially is related to verification of properties of the system, state properties and action properties and we are assuming that these properties are indirect indicators of usability and if we can verify this properties if we can ensure that these properties are present and satisfied in the proposed system or the system that you are designing, then there is a likelihood that the resulting system is going to be usable.

In the next lecture, we will talk about some specific formalisms that are used to model interactive systems. Today, we mostly discussed examples. In the next lecture, we are going to discuss some specific formalisms applied to the design of interactive systems.

(Refer Slide Time: 54:19)



The material that I have covered today are taken from this book. You are advised to refer to chapter 8, section 8.2 to 8.3, for more details on the topic that I have covered today, including some discussion on the approaches used for formal verification of this properties.

Thank you and goodbye.