HCI: Dialog Design – Use of Formalism

Learning Objective

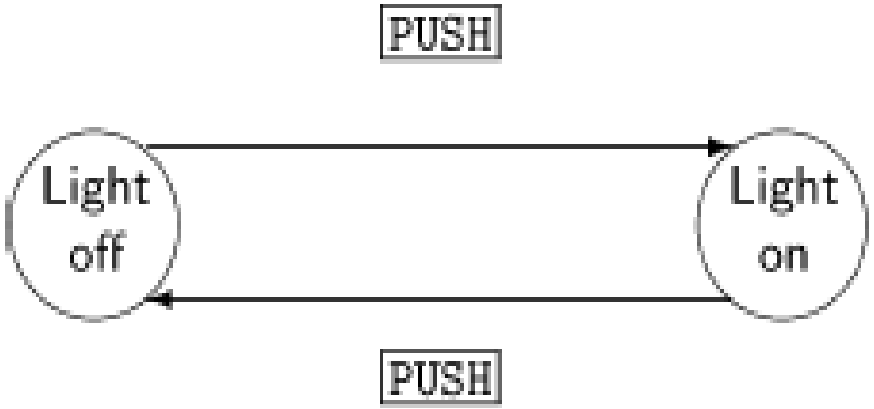
- In the previous lectures, we have learned about different formalisms to represent dialogs
- Some are simple (STN) but not powerful enough to capture typical interactive behavior (complexity and concurrency), others are more suitable (State-Charts, PNs)

Learning Objective

- It’s clear from the discussions that representing interactive systems formally is no easy task and requires expertise
- This brings us to the question, why should we spend time and effort mastering formal representation techniques?
- In this lecture, we shall discuss how to answer this question

Example Use of Formalism

- Let us try to understand the use of formalism in dialog design with a simple example. Consider a simple two state system – a light bulb controlled by a pushbutton switch, which alternately turns the light on or off.
- The corresponding STN is



a. Light with push-on/push-off action.

Example Use of Formalism

- The example system belongs to the general class of push button devices
- They belong to an important class of interactive system (Desktop GUI, touch-screen devices, WWW)
  - Ubiquitous (mobile phones, vending machines, aircraft flight deck, medical unit, cars, nuclear power stations)
- Thus, modelling such devices can actually help model a large number of interactive systems
- Interaction with such devices can be modelled as the Matrix Algebra (MA)

Push Button Device: FSM to MA

- We shall use the following notations
  - N = number of states
  - States are numbered from 0 to N
  - A transition is represented by a matrix (N×N)
  - A state is represented by an unit vector of size N, with N-1 0s and one 1 at the position corresponding to the state number. e.g., states ON = (1 0) and OFF = (0 1)

Push Button Device: FSM to MA

- We shall use the following notations
  - New state = old state (vector) × transition(s) [a matrix multiplication]

    e.g., one pushbutton action push

$$\boxed{\texttt{PUSH}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- We can check several properties of our simple push button device through matrix multiplication

Property Checking: Example

- When light is off, pushing the button puts the light on

$$\begin{aligned} \mathbf{off}\ \boxed{\texttt{PUSH}} &= (0\ 1)\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ &= (1\ 0) \\ &= \mathbf{on} \end{aligned}$$

- Similarly, we can show pushing the button when the light is on puts it off

Push Button Device: FSM to MA

- The case for *undo*

$$\boxed{\text{PUSH}}\,\boxed{\text{PUSH}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$= I$$

- Pressing the button twice return the system to the original state (s $\boxed{\text{PUSH}}\,\boxed{\text{PUSH}}$ = s)
  - What about pushing the button thrice, four times,…? Do the calculations and check for yourself

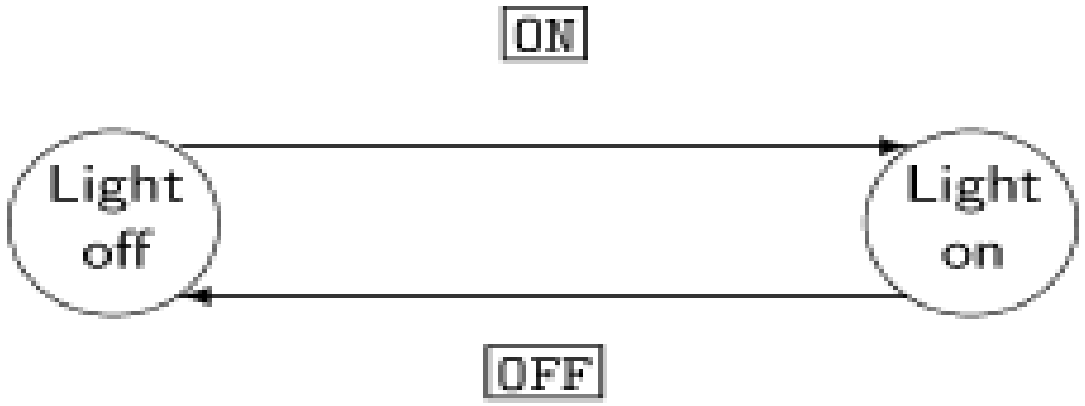
Push Button Device: FSM to MA

- A problem: the lamp has failed and we want to replace it
  - Need to know if it is safe (i.e. the system is in the off state)
  - How many times the user has to press the button so that he is sure that he is in a safe state (remember, we are not sure of the current state)

$$\boxed{\text{PUSH}}'' = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

Push Button Device: FSM to MA

- Mathematically, we can show that NO value of n exists for this system that satisfies the equation
  - We need a two-position switch



b. Light with separate on/off actions.

Push Button Device: FSM to MA

- A two-position switch gives two options: ON and OFF

$$[ON] = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

$$[OFF] = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

- We can check, through matrix algebra, that OFF switch works as intended

$$\text{On}[OFF] = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix} = [OFF]$$

i.e., pressing **Off** after **On** keeps the system in [OFF] state

$$\text{Off}[OFF] = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix} = [OFF]$$

i.e., pressing **Off** after **Off** keeps the system in [OFF] state

Message

- From the previous example, we can see that the formal representation (in this case, the matrices) allow us to check for the system properties through formal analysis (in this case, matrix algebra)
- From this analysis, we can decide if there are any usability problems with the system (e.g., single button is not good in case of failure, as it is dangerous to repair)
- That is the main motivation for having formalism in dialog representation (it allows us to check for properties that should be satisfied for having a usable system)

System Properties

- The properties that are checked for a usable system are of two types
  - Action properties
  - State properties

Action Properties

- There are mainly three action properties that a usable system should satisfy
  - Completeness: if all the transition leads to acceptable/final states or there are some *missed arcs*, i.e., some transition sequence don't lead to final states
  - Determinism: if there are several arcs (transitions) for the same action
  - Consistency: whether the same action results in the same effect (state transition) always

How Action Properties Help

- Completeness ensures that a user never gets struck at some state of the dialog, which s/he doesn't want, and can't come out from there (leads to frustration and less satisfaction)
- Lack of determinism introduces confusion and affects learnability and memorability
- An inconsistent interface reduces memorability and learnability, thus reducing the overall usability

State Properties

- There are mainly three state properties related to system usability
  - Reachability: can we get to any state from any other state?
  - Reversibility: can we return to the previous state from the current state?
  - Dangerous states: Are there any undesirable states that leads to deadlock (i.e. no further transitions are possible)?

How State Properties Help

- Reachability ensures that all system's features can be used
- Reversibility ensures that the user can recover from mistakes, thus increasing confidence and satisfaction
- Detection of dangerous states ensures that the user never goes to one, thus avoiding potential usability problems