X-machines

• Recall the diagrams we saw before:

• When we trace through the diagram there are a number of key things to consider:
  • Taking a transition from a state to another state may be the result of:
  – An input that only enables one out of several transitions to be valid
eg. A mouse click such as ClickCustomer
  – There are two or more transitions possible but each transition can only be enabled under a specific set of conditions

• For example, suppose that there is a mouse click that is in an area that does not correspond to a button.
  • We could add a new arrow to the diagram:

• With the transition called Ignore
  • This transition is taken under the conditions that the mouse is clicked in any area other than the buttons:
    Customers or Orders
  – or any key is pressed
  • From this state we have one of three mutually exclusive situations:
    – Mouse is clicked over the Customer button;
    – Mouse is clicked over the Order button;
    – Mouse is clicked over the rest of the screen or a key is pressed.

Transition conditions

• This introduces an important property of X-machines.
• There can be two arrows leaving from the same state – usually going to different states (but not always)
• Which transition is taken depends on some condition.
• The conditions could be based on some property of the input
• Or on an internal property

• In some transitions we have to consider the state of some internal memory or database data before identifying which transition should be taken.
• Consider a simple Login system with two classes of users:
  – Admin
  – User
• An X-machine might look like this:
Recall that the square states denote situations where the system consults some internal database to look up necessary information.

- In some cases you may find a state where there could be cases that have not been specified
  - in these cases it is important to decide what the system should do
  - it might ignore an input and remain in the same state, for example,
  - or raise an error and transfer to an error state from which some suitable recovery is defined.

- The 2 functions must cover all possible cases so that the system does not suddenly stop because it doesn’t know what to do – or it just decides to do what it likes!!!

This problem is called non-determinism. We need to deal with it.

- In this system consider what happens in the Check password state.
  - There are three possible transitions from this state:
    - The password is OK and the user has Admin privileges;
    - The password is OK and the user has User privileges;
    - The password fails
      - If it fails 3 times we reach the Error state and a suitable message is displayed
    - There are no other possibilities and so the system will be fully defined at this point.
  - We call the machine complete.

System testing from XXMs

Paths and sequences
- Take any path from the start state and trace out a sequence of transitions
- This will represent a test set
- To trigger this path we will have to interact with the GUI:
  - Press a suitable button
  - Feed in suitable data
- We Observe what happens
  - What state do we get to?
  - What output is generated?
XXM editor in Eclipse

- If you use this tool to build your X-machines it will generate all the paths for you
- These paths are a transition tour – they visit every state and every transition
- There is still the issue of how do you know what state you have reached
- This can be solved using a more complex sequence – probably not needed for our type of systems

What are we testing for?

- 1). there are too few states;
- 2). there are too many states;
- 3) there are transitions going from an incorrect state;
- 4). there are transitions going to the wrong state;
- 5). there are transitions that carry out the wrong function.

Missing state fault

- Test such as: ClickOrders; ClickOrders
- Should detect this fault

Too many states

- A test: Quit finds this fault

Incorrect transition source

Test: Quit finds this fault

Incorrect transition target

- A test: ClickCustomer; EnterCustomer Will detect this
Transition with wrong operation

UploadCustomer operation should insert details from the EnterCustomer operation into the database

To test this we need to output the result of this operation somehow.
This might be done as part of the user interface but, more generally, we will need to run a report on the database to see if the operation was successful.
This leads to a set of tests which vary according to the data being input.
Thus, it might work for some cases but not for others.

System testing involves both transition paths and data entry
- It may be possible to automate some of this
- Make sure that you use realistic data values
- Try out unusual, extreme or invalid values
- Check for data validation errors
- Check for completeness
- Check for unexpected interface behaviour

Design for test
- Sometimes it isn’t possible to test something directly
- We may need to drive the system into a specific state in order to test a specific transition
- It may not be designed to allow this
- We introduce some extra test functions into the machine

Controllability

• Extra software is written to access the function concerned

Observability

• Extra software is written so we can see what is happening
• E.g. writing out some values that are not normally accessible
• This will allow us to apply sequences of inputs that are not normally allowed
• The point is to make sure that nothing unexpected happens
• These tests should fail
Deep test sequences

- ClickCustomer; EnterOrder
  - Cannot normally be applied without amending the machine as above
- ClickCustomer
  {should pass}
- ClickCustomer; EnterOrder
  {should fail}