SED Lecture 20
Conformance Testing

Finite State Machines
- FSMs model the dynamic behaviour of a system
- STATES describe a condition of the system
  - e.g. light-on, daytime, running, waiting
  - denoted by a circle, may be labelled with the state name. Initial state has a double circle.
- TRANSITIONS describe a change of state
  - may be triggered by an event, some state condition or the passing of time
  - a state may have 0, 1 or more outgoing transitions and may have self-loop transitions

Types of FSM
- FSMs have proved useful in many branches of computer science, and there are many different types of FSM e.g.
- **UML statecharts**: transitions labelled with a string denoting an event, condition or time (see B&D p62)
- **Mealy Machines**: transitions labelled with a pair of events i/o naming an input i and expected output o
- Other types of FSM are labelled with expressions and assignments to represent programs

Mealy FSM example
- We will use Mealy FSMs for conformance testing
- transitions are labelled with input/output pairs
- Example for a digital watch (see B&D ch 2)
  - inputs: button 1 pressed (b1), button 2 pressed (b2)
  - outputs: incr hours (h), incr minutes (m), display full time (d), beep(s) for unexpected input

Identifying a FSM by its outputs
- To test that an implementation conforms to a FSM specification, check that for each input sequence, the implementation delivers the same output sequence as the specification
- Q: for input sequence b1,b2,b2,b1,b1 what is the output sequence of the watch FSM?
- Q: can you find a test input sequence (or more than one sequence) which tests every transition?
- Q: can you find a test input sequence which uniquely identifies state 2? (that is, whose output sequence is not matched by any other state)
The Conformance Test Problem

- Suppose we have
  - I a (black box) implementation &
  - S a formal FSM specification for I
- The problem is to construct
  - a finite set of test experiments T
- Such that
  - if I≈S then I passes every test in T &
  - if not I≈S then I fails some test in T

Assumptions Made

- Each FSM state must have an output for every possible input: FSM is completely specified
- For each input from a given state there is a unique output value and state: FSM is deterministic
- There are no redundant states: FSM is minimal
- We shall assume that the implementation FSM has the same number of states as the specification FSM
- See reference list papers for some things to do when these assumptions are not reasonable

Action Errors

- green/go amber/slow red/stop specification
- green/go amber/go red/stop implementation

Transfer Errors

- green/go amber/slow red/stop specification
- green/go amber/slow red/stop implementation

Worked Example:
Specification of a comment printer
Input consists of the characters * o ( )
o stands for any character except * or ( or )
Print only comments, where a comment is an input subsequence enclosed by (*) on the left and *) on the right.
A comment may contain other (*s but not *)s

Note: there is an error in this example in Chow’s paper

Comment Printer outputs

<table>
<thead>
<tr>
<th>OUTPUT</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>i(gnore)</td>
<td>null action</td>
</tr>
<tr>
<td>e(mpty-bf)</td>
<td>set buffer to empty</td>
</tr>
<tr>
<td>a(cc-bf)</td>
<td>add current input to the buffer</td>
</tr>
<tr>
<td>p(rint-bf)</td>
<td>remove last char added to the buffer and print out contents</td>
</tr>
</tbody>
</table>

Assume that the buffer stores only characters and that its length is unbounded
A Comment Printer FSM

Ex: Give the output sequences generated by this FSM for the input sequences o)((oo**)) and (**(*)

Ex at home: Try also the input: int i (* this is an int.*)

How to Generate Test Cases (1)

- The set of all test sequences required to test an FSM is the concatenation of two sets of sequences P and Z.
- All test sequences must start from the INITIAL state.
- P is any set of input sequences such that:
  - for every transition from state Ai to state Aj & input x
  - P contains a sequence of inputs p.x where:
    - p forces the machine into state Ai from its initial state
  - P also includes the empty input sequence
- Construct P by starting at the initial state and at each stage adding edges which lead from the current frontier states until all edges have been covered.

Example: P for the CP

<table>
<thead>
<tr>
<th>Edge</th>
<th>P</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/i</td>
<td>*</td>
<td>3/a</td>
</tr>
<tr>
<td>1/o</td>
<td>o</td>
<td>3/a</td>
</tr>
<tr>
<td>1/2</td>
<td>()</td>
<td>3/a</td>
</tr>
<tr>
<td>2/o</td>
<td>*</td>
<td>4/a</td>
</tr>
<tr>
<td>2/1</td>
<td>()</td>
<td>4/p</td>
</tr>
<tr>
<td>2/2</td>
<td>()</td>
<td>4/a</td>
</tr>
</tbody>
</table>

A characterization set for the CP

Z = { ) , * }

CHECK that the 2 single element sequences, ) and *, are sufficient to distinguish between any two states in the specification FSM.

Sometimes sequences of more than one element are needed.

How to Generate Test Cases (2)

- Z is used to check that the implementation state reached is the same as the specification state.
- If for an input the implementation gives an unexpected output then we know that I is incorrect.
- However, what if the implementation gives the right output but then goes on to produce incorrect outputs in the future?
- A set Z which can distinguish between every state of an FSM by that state’s response to certain input sequences is called a characterization set for the design.

How to Generate Test Cases (3)

- Now that we have constructed the sets P and Z, the set of all test cases for a given FSM is given by concatenating (joining together) each sequence in P with each sequence in Z.
- The test suite for the CP example is built from:
  - P = { , *, o, ), (, (, (, (**, (*, (*, (*, (**, ( }
  - Z = { ), * }

How to Generate Test Cases (4)

- $P.Z$ is $\{\ast, o\}$ and $\ast$ from (empty sequence) $Z$ with
- $\ast, o, (\ast), (\ast)^2, (\ast^3), (\ast^4), (\ast^5), (\ast^6), (\ast^7), (\ast^8), (\ast^9), (\ast^{10}), (\ast^{11}), (\ast^{12}), (\ast^{13}), (\ast^{14}), (\ast^{15}), (\ast^{16})$

- but $\ast, o, (\ast), (\ast)^2, (\ast^3), (\ast^4), (\ast^5)$ are included in longer sequences and so finally
- $P.Z = \{\ast, o, (\ast), (\ast)^2, (\ast^3), (\ast^4), (\ast^5), (\ast^6), (\ast^7), (\ast^8), (\ast^9), (\ast^{10}), (\ast^{11}), (\ast^{12}), (\ast^{13}), (\ast^{14}), (\ast^{15}), (\ast^{16})\}$

- 29 test cases are needed from the possible $32+2$ tests for this FSM with 16 edges, 4 inputs, 4 outputs and 4 states. In general, up to $N.M$ tests are needed for an FSM with $N$ edges whose states can be distinguished by a set of size $M$.

Conformance Test Summary

- Chow's WP Method shows how to generate test suites for implementations which can be specified by Mealy FSMs
- FSM specs must be completely specified and minimal
- We have also assumed that the implementation has no more states than the specification (this assumption is not essential but it makes life easier)
- $T = P.Z$ is a finite set of test cases with the property that
  - If $I \approx S$ then $I$ passes every test in $T$
  - If not $I \approx S$ then $I$ fails some test in $T$

- The test method is, thus, both reliable and valid

Type of Implementation Faults which can be Detected

- **Action Errors** (see slide 9)
  - There is an edge from state $s$ which is in both $S$ and $I$, so that given input $i$, $S$ outputs $o_1$ but $I$ outputs a different value $o_2$

- **Transfer Errors** (see slide 10)
  - $S$ and $I$ give output the same value given input $i$ in state $s$, but afterwards $S$ has reached state $s_1$ but $I$ has reached a different state $s_2$

- **Doesn't detect Extra States** in the implementation

Further Reading