Test Oracles

- Discussion
  - Automation of oracle necessary
  - Expected behavior given
  - Necessary parts of an oracle
  - Name spaces

Test Oracle

- A test oracle determines whether a system behaves correctly for test execution

Webster Dictionary - Oracle
- a person giving wise or authoritative decisions or opinions
- an authoritative or wise expression or answer

Purpose of Test Oracle

- Sequential Systems
  - Check functionality
- Reactive (event-driven) Systems
  - Check functionality
  - Timing
  - Safety

Reactive Systems

- Complete specification requires use of multiple computational paradigms
- Oracles must judge all behavioral aspects in comparison with all system specifications and requirements
- Hence oracles may be developed directly from formal specifications
Parts of an Oracle

- **Oracle information**
  - Specifies what constitutes correct behavior
  - Examples: input/output pairs, embedded assertions

- **Oracle procedure**
  - Verifies the test execution results with respect to the oracle information
    - Examples: equality

- **Test monitor**
  - Captures the execution information from the run-time environment
    - Examples
      - Simple systems: directly from output
      - Reactive systems: events, timing information, stimuli, and responses

Approach

- **Test class**
  - Set of test data described by a condition that constrains input data and the initial system state
  - Every test class will have an explicitly represented test oracle
  - Results are monitored and verified against the oracle corresponding to all test classes satisfied for the test data

Phases of the Approach

- **Oracle derivation**
  - From specifications for each test class

- **Monitoring test execution**

- **Mapping and applying the oracle procedure to the execution results**
Automated Test Oracles for GUIs

Foundations of Software Engineering, 2000

GUI Test Cases
- GUI Test Case
  - Sequence of events

A Test Case for WordPad

What Is Correct Behavior

Check State, not only Output!!
**Research Focus**

- **Goal**
  - To check the GUI's state after each event
- **Approaches**
  - Manual
  - Automated
- **Challenges**
  - Generating expected state
  - Extracting actual state
  - Comparing expected & actual states

**Outline**

- Overview of GUI Oracle
- Generating Expected State
  - Modeling the GUI's State
    - Objects
    - Properties
  - Modeling the Events
- Obtaining Actual GUI's State
- Comparing Actual & Expected States

---

**Overview of GUI Oracle**

- **Test Case**
  - **Expected-state Generator**
  - **Verifier**
  - **Execution Monitor**
- **Oracle**
  - **Expected State**
  - **Actual State**
  - **Verifier**
  - **Execution Monitor**

---

**Modeling the GUI**

A GUI consists of Objects

<table>
<thead>
<tr>
<th>Form</th>
<th>Window State</th>
<th>Width</th>
<th>AutoScroll</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wsNormal</td>
<td>1088</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>Align</td>
<td>alNone</td>
</tr>
<tr>
<td>Caption</td>
<td>Files of type:</td>
</tr>
<tr>
<td>Color</td>
<td>clrBtnFace</td>
</tr>
<tr>
<td>Font</td>
<td>(tFont)</td>
</tr>
<tr>
<td></td>
<td>Caption</td>
</tr>
<tr>
<td></td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Visible</td>
</tr>
<tr>
<td></td>
<td>Height</td>
</tr>
</tbody>
</table>

---

**Outline**

- Overview of GUI Oracle
- Generating Expected State
  - Modeling the GUI's State
    - Objects
    - Properties
  - Modeling the Events
- Obtaining Actual GUI's State
- Comparing Actual & Expected States
All Properties of Cancel

Determining Properties
- Manual Examination of GUI
  - GUI being tested
- Specifications (Reduced Set)
  - GUI being tested
- Toolkit/Language (Complete Set)
  - All available properties

Now we know how to represent the GUI's state

Modeling Events
- Events are State Transducers

State: \( S_i \)

SelectText ("This")

Event: \( e \)

Notation: \( S_j = [S_i, e] \)

State: \( S_j \)

Representing Events
- We define an event as:
  \( S_j = [S_i, \text{event}] \)
- For example:
  \( S_j = [S_i, \text{cut}] \)
- Need a compact representation
## Operators

**Operator:** `CUT`

**Preconditions:**
- `isCurrent(Menu2)`

**Effects:**
- `FORALL Obj in Objects`  
  - `Selected(Obj)`  
  - `ADD inClipboard(Obj)`  
  - `DEL onScreen(Obj)`  
  - `DEL Selected(Obj)`
- `ADD isCurrent(Menu1)`
- `DEL isCurrent(Menu2)`

## Deriving Expected State

- **Given** $S_0$, the initial state,
- **A sequence of events** $e_1, e_2, e_3, \ldots, e_n$
- **Obtain** $S_1 = [S_0, e_1]$
- **And** $S_i = [S_{i-1}, e_i]$

## Obtaining Actual GUI’s State

- **Execution Monitor**
  - Screen Scrapping
  - Queries
  - Compatible with Expected State
  - Returns `{Object, Property, Value}`
  - `<Button1, “Caption”, “Cancel”>`

## Automated Execution

- **Test Executor**
- **GUI Under Test**
- **Test Cases**
- **Execution Monitor**
- **Verifier**
  - **Expected State**
  - **Actual State:** `{isCurrent ROOT) (Contains ROOT D)`

---
Comparing Actual and Expected States

- Verifier
- Three Levels of Testing
  - Changed Property Set (Operators)
  - GUI Relevant Property Set (Specifications)
  - Complete Property Set (Toolkit/Language)
- Hybrid Approach
  - Use all 3