State-Driven Testing

Düsseldorf, 1.12.2016
Johannes Döing
Agenda

- **Introduction**
  - Methods
  - Demo 1
  - Eulerian Test Case
  - Coverage
  - Demo 2
  - Conclusion & Discussion
  - References
Introduction

- State-Based Systems?

- Embedded Systems
- Finite State Machines
- Example: Parking Automata
- Software Components
- GUI
Introduction

- Model-based Testing
- Generation and testing based on test artifacts
- Generate test cases from state diagrams
Introduction

- Node = State = Vertex
- Start and End Node
- Arc = Transition = Arrow
- Directed graph (Digraph)
- Multi(di)graph
Methods

- Transition Tree
- n-Switches
Transition Tree

- Simple Method

- Intermediate Step of building a transition tree

1. Start vertex is the tree root

2. Process all transitions of the current examined node
   - For each transition create a new branch / node

3. Repeat Step 2 for all leaves until either
   1. State is already in the path to the root node.
      Exception: One Pass Loops
   2. An end / final node is reached
Transition Tree

Start

init[0]

tear down[0]

DT

coin[1]

abort[1]

print[1]

End

PT

Start

init

tear down

coin

End

PT

coin

abort

print

DT

DT

DT
Transition Tree – Failure Transitions

- State Diagram is based on Finite State Machines (FSM)
- Goal: Find not specified transitions
- Set Q State
  - Q = \{Start, DT, PT, End\}
- Set Σ Inputs
  - Σ = \{start, coin, abort, print, end\}
Transition Tree – Failure Transitions

- Start and end nodes are not considered
- Neither their associated transitions
- Set $Q$ State
  - $Q = \{DT, PT\}$
- Set $\Sigma$ Inputs
  - $\Sigma = \{\text{coin, abort, print}\}$
- Find the not executed combinations
Transition Tree – Failure Transitions

- DT → abort → Failure
- DT → print → Failure
Transition Tree - Rating

- 4 leafs = 4 test cases
- Failure Transitions
- Loops?
- Cycles?

Diagram showing transition tree with nodes labeled as 'Start', 'DT', 'PT', and 'End'. Nodes are connected with arrows indicating transitions.
Methods

- Transition Tree
- **n-Switches**
n-Switches

- Chows Metric
- Testing all possible Transitions
  - Including cycles (history)
- Every node is a “start” node
- Switching States
- Depth (n)
n-Switches

- Limited Depth Search (parameter n + 1)
n-Switches

- Loops
- Cycles
- Start sequences
- No Failure Transitions
Demo One
Eulerian Test Case

- **Further Definitions**
  - Eulerian Test Case
  - Related Algorithms
  - Example – Create an Eulerian Test Case
  - Coverage
Further Definitions

- Degrees
  - In-Degree
  - Out-Degree

- Strongly Connected Components

- Bridges
Eulerian Test Case

- Eulerian Trail
  - Start != End
  - All Transitions are covered

- In-Degree = Out-Degree
  - Exception: start and end vertices

- One start and one end vertex

- Strongly connected components (SCC)

- Bridges points to end
Related Algorithms

- Bellman-Ford Algorithm
  - finds the shortest path between two nodes

- Tarjans Algorithms
  - finds strongly connected components

- Chinese Postman Algorithm
  - adds additional (existing!) arcs

- Hierholzers Algorithm
  - finds the Eulerian Trail
Example – Create an Eulerian Test Case

- Tarjans Algorithm finds set $S_1 = \{DT,PT\}$
- Apply Chinese Postman on each SCC (Bellman-Ford)
Example – Create an Eulerian Test Case

- Adopted Hierholzer Algorithm
- Kicks off at the Start node
- Use Bellman-Ford to find shortest Path to End
Adopted Hierholzer Algorithm
Coverage

- Calculation of n-Switch Coverage ( $n = 1$ )
- n-Switch test cases exist in the Eulerian test case

Diagram:

1. Start
2. DT → PT
3. PT
4. DT
5. PT
6. DT
7. End

Diagram:

1. Start
2. DT
3. PT
4. DT
5. PT
6. PT
7. PT
8. DT
Coverage

- 12 1-Switches test cases
- 6 1-Switch test cases in the Eulerian Test Case
- 1-Switch Coverage = 6 / 12 * 100 = 50%
Demo Two
Conclusion

- Automated generation of test cases based on UML

- 3 Methods
  - Transition Tree
  - n-Switch
  - Eulerian Test Case

- Complexity / Effort

- Coverage
Future Work

- Interface to Specification Tool
- Guards
- Equivalence Class Partitioning / Boundary-value Analysis
- Composite States Entry Points
- Automation of test cases
Quellen

- [B1] Basiswissen Softwaretest
  Spillner und Linz

- [B2] Praxiswissen Softwaretest
  Bath und McKay

- [B3] UML 2.0
  Kecher
Quellen

- [B4] Graphen für Einsteiger
  Nitzsche

- [B5] Formale Sprachen
  Böckenhauer und Hromkovic

- [B6] Algorithmische Graphentheorie
  Turau
Further Challenges

- Composite States
Composite States

- Reduce complexity
- Top down approach
- Introduce hierarchy
Composite State -> Flat Graph

- Algorithms and hierarchies?
  - Does not match!
- Re-organize the graph
- New source / new target

Start

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>tear down</td>
</tr>
<tr>
<td>DT</td>
<td>coin</td>
</tr>
<tr>
<td>DCV</td>
<td>abort</td>
</tr>
<tr>
<td>DPT</td>
<td>print</td>
</tr>
</tbody>
</table>

End 1

End 2
Additional Information

- **Strongly Connected Components Special Case**
SCC - Special Case

- Each vertex is a strongly connected component
  - Except Start and End
  - Simple Example of an Eulerian Trail

State 1  State 2  State 3

Start  →  State 1  →  State 2  →  State 3  →  End
Live

- Graphic Recording
Live - Graphic recording
About Capgemini

With more than 120,000 people in 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2011 global revenues of EUR 9.7 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini