Automatic Extraction of Abstract-Object-State Machines Based on Branch Coverage

Hai YUAN  Tao XIE

Department of Computer Science
North Carolina State University
hyuan3@ncsu.edu  xie@csc.ncsu.edu
Agenda

- Motivation
- Related Work
- Example
- Object State Machine (OSM)
- Framework
- Conclusion and Future work
Motivation

- Software specifications are useful
  - but they often do not exist
- Object State Machine (OSM) can be inferred from program executions
  - but inferred concrete OSM are too complex to understand
- We propose Brastra to abstract concrete OSMs
  - based on branch coverage
  - Inferred OSMs are often succinct and useful
Related Work

- Use return values of observers to abstract concrete states [Xie and Notkin ICFEM 04]
- Use individual field values to abstract concrete states [Xie and Notkin SAVCBS 04]
- Extract statically object state models from source code [Kung et al. COMPSAC 94]
- Extract state models based on only call sequences, without using object-field values or structural coverage [Whaley et al. ISSTA 02].
Example - UBStack

Unique Bounded Stack

- Stack capacity is bounded (e.g., set as 3).
- No duplicated elements in the stack.

push(x):

<table>
<thead>
<tr>
<th>push(3)</th>
<th>push(3)</th>
<th>push(3)</th>
<th>push(4) error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Pop():

<table>
<thead>
<tr>
<th>pop()</th>
<th>pop()</th>
<th>pop() error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Specify Object Behavior with Object State Machine (OSM)

OSM: A sextuple (I, O, S, δ, λ, INIT)
- I: set of method calls in the class interface.
- O: set of return values of the method calls.
- S: set of object’s states.
- INIT ∈ S: initial state of the state machine.
- δ: state transition function. S × I → P(S)
- λ: output function. S × I → P(O)
- P(S) and P(O) are power set of S and O, respectively.

Stack() 
push(1) [1;0;0] 
push(2) [2;0;0] 
INIT [0;0;0]
Build Concrete OSMs

- **Generate tests for UBStack**
  - Manually configure push’s argument to be 1,2,3,4
  - Default stack elements are 0.
  - Automatically generate 263 test cases with Rostra [Xie et al. ASE 04]

- **Collect test execution information with Daikon** [Ernst et al. TSE 01].

- **Build concrete OSMs from Daikon traces.**
  - State: values of object fields.
  - Transition: method calls (with arguments).
  - 41 states and 142 transitions.
Concrete OSM of UBStack - Overview
Concrete OSM of UBStack - Details
**Brastra Framework**

- **Basic idea:**
  - Partition concrete states based on the branch coverage of the methods invoked on these states.

  ![Diagram](image)

- **Procedure:**
  - Build concrete OSMs from Daikon traces.
  - Collect branch coverage using modified jusc tool [Xie&Notkin JASE 06].
  - Merge concrete states based on branch coverage
Define Branch Coverage with Conditional Set

public int pop()
    {
        int ret = -1;
        if (numberOfElements > 0) {
            ...
        } else { ...
        return ret;
    }

A. UBStack.

private void syncMenu()
    {
        ...
    }

6: if (bugInstance != null) {
        ...
        selectSeverity(severity);
        ...
    }

private void selectSeverity(int severity) {
        ...
    }

5: for (int i=0;i<severityItemList.length;i++)
        {
            ...
        }

B. findbugs.classify.SeverityClassificationPulldownAction
Collect Branch Coverage

concrete states  branch coverage

- UBStack: 0 0 0 0 2 1  ➔ pop(): numberOfElements > 0 = false

- UBStack: 0 2 1  ➔ pop(): numberOfElements > 0 = true
Group States by Branch Coverage

<table>
<thead>
<tr>
<th>concrete states</th>
<th>branch coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBStack:</td>
<td>pop(): numberOfElements &gt; 0 = true</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

| UBStack:        | pop(): numberOfElements > 0 = true |
| 0               |                 |
| 3               |                 |
| 2               |                 |
Illustrating Example

public void push(int k) {
    int index; boolean alreadyMember = false;
    for(index=0; index<numberOfElements; index++) {
        if (k==elems[index]) {
            ...
        }
    }
    if (alreadyMember) {
        for (int j=index; j<numberOfElements-1; j++)
            ...
    } else {
        if (numberOfElements < max) {
            ...
        } else {
            System.out.println("Stack full");
            return;
        }
    }
}
**UBStack – Brastra Result**

States:
41 → 5

Transitions:
142 → 11
Abstract State Details

pop(): all calls of pop() on the state will take the same branch.

push(x):
- x not in the stack and stack not full
- x in the stack and not at the top of the stack.
- x in the stack and at the top of the stack.
Conclusion

- Software specifications are useful
  - but often do not exist
- Concrete OSMs can be inferred from program exec
  - but too complex to be useful.
- We proposed Brastra to abstract concrete OSM
  - group concrete states based on method call branch coverage
- Initial results of applying Brastra on UBStack show Brastra’s utility.
Future Work

- Enhance Brastra with existing FSM-based testing techniques
  - Test generation
  - Test reduction
- Extend Brastra to multiple classes instead of one
  - Subsystem behavior
- Slice on fields of interests for further reduction
- Recover non-functional requirements.
Questions?
Thank You!