AutoAbstract: Problem Statement and Hypothetical Solutions

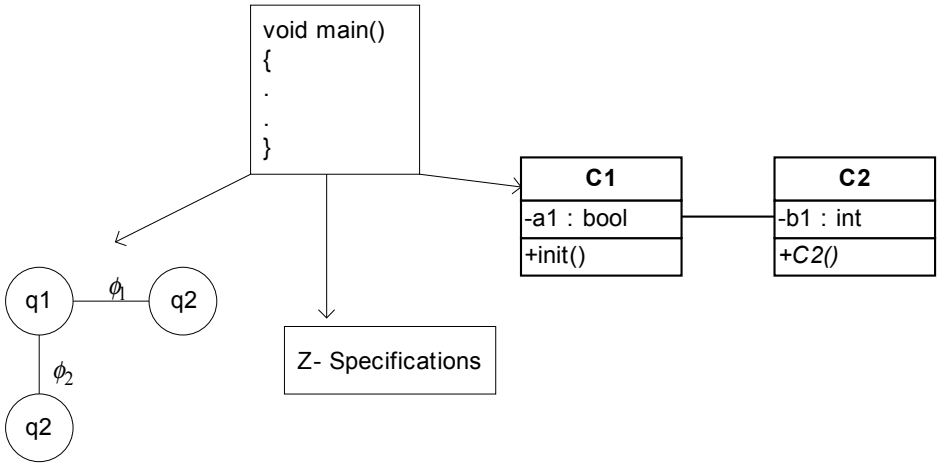
Shaukat Ali
Verification and Testing Group (VT)
Department of Computer Science
University of Sheffield

Layout of the Presentation

- Introduction
- Background
- Proposed Ideas
- Conclusion

Introduction (1/4)

- Testing
- Specifications are normally out-of-date or incomplete
- Rigorous testing methods are available for software modelled using FSMs and X-Machines
- Reverse Engineering



Introduction (2/4)

- Which of the reverse engineered diagram is better??
- A developer knows what is a non-trivial control
- “Dialogue” between reverse engineering tool and a tester is required
- How specifications are reverse engineered
  - Static vs Dynamic

Introduction (3/4)

- Incremental change
- Automated abstraction of code into state-based specification and test generation (AutoAbstract)
  - Extract up-to-date specifications from the code and hints from a developer

Introduction (4/4)

- Hints: Instructions to the reverse engineering tool
  - What is a state, what is a function, etc.
  - Done declaratively
- Extracted specifications will be used for testing

Background (1/3)

- X-Machines
  - Extended FSM
  - Memory and Processing Functions
  - Why X-Machines
    - X-Machine testing methods are formal
    - Applied to different industrial case studies
    - Many testing techniques for testing from software modelled using X-Machines exists

Background (2/3)

- DAIKON
  - Dynamically generates invariants from the code
  - Source code is executed by running different tests
  - Inferred invariants can be used for software evolution and program understanding

Background (3/3)

- Example

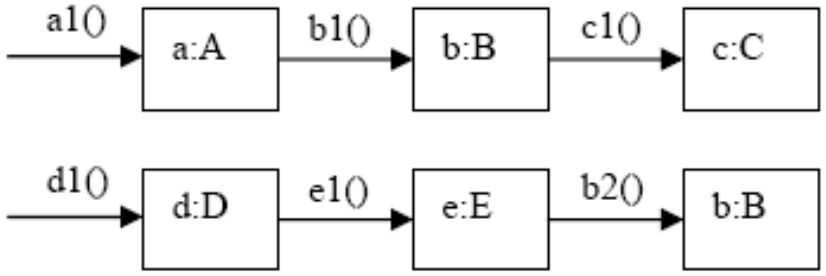
```
==========================
Absolute.abs(int):::EXIT
return >= 0
(orig(arg0) == 0) ==> (return == 0)
(return == 0) ==> (orig(arg0) == 0)
return >= orig(arg0)
==========================
```

```
public class Absolute
{
  public int abs(int no)
  {
    int y=0;
    if (no <0 )
      y=-no;
    else
      y=no;
    return y;
  }
}
```

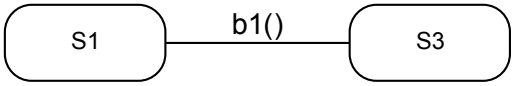
Proposed Ideas

- Reverse Engineering of X-Machines from code
  - Dynamic approach
  - Running different collaborations in the DAIKON
  - Retrieval of states (values of instance variables) at start and end of each called method using DAIKON

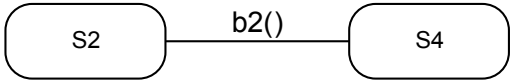
Example



From 1st Collaboration

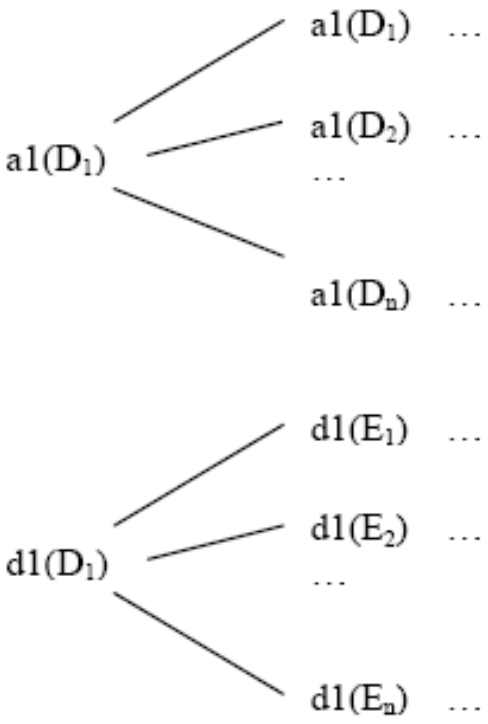


From 2nd Collaboration



Reverse Engineering

- Chaining of collaboration diagrams

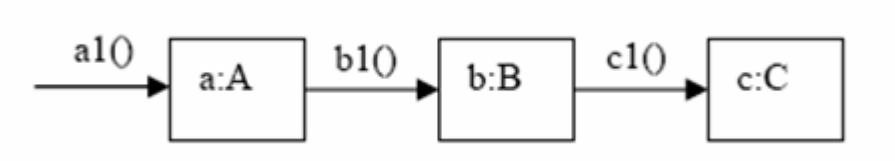


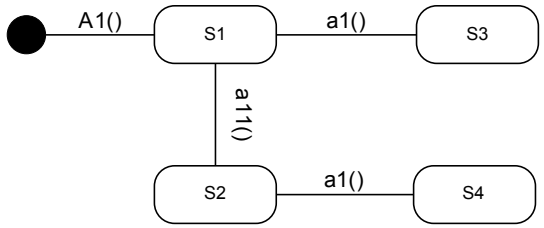
Chaining of Collaboration Diagrams

- Infinite growing tree
- Need to define some stopping criteria
  - Exception thrown, Number of iterations
- Abstraction function

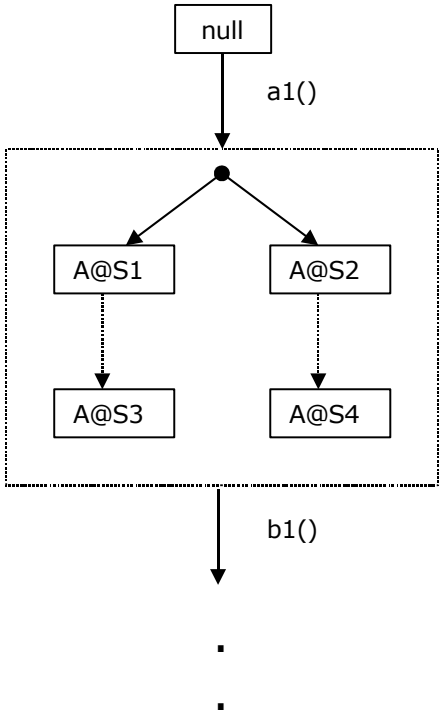
Generation of Test Sequences

- State-COllaboration TEst Model or SCOTEM
  - State transition structure of X-Machines along with collaborations will be used for testing





Example



Conclusion

- AutoAbstract
  - Problems
- Proposed Solutions
  - Reverse Engineering of X-Machines
  - Test case generation
    - SCOTEM

Questions