

TestEra: A Novel Framework for Testing Java Programs

Sarfraz Khurshid and Darko Marinov
 MIT Lab for Computer Sciences
 Invited submission to the Automated Software Engineering Journal.
 Proceedings of the 16th IEEE Conference on Automated Software Engineering (ASE 2001)

1 10/1/04

Outline

- Introduction
- Alloy
- TestEra Analysis
- Case Studies
- Implementation & Performance
- Conclusions

2 10/1/04

Introduction

- Problem: Manual software testing and test data generation are labor-intensive processes
- Solution: Automated testing that will significantly reduce the cost of software development and maintenance
- TestEra: automatically tests Java programs using specification-based testing

3 10/1/04

TestEra

- Uses Alloy to express invariants and pre/post-conditions
- Problem: manually modeling using Alloy is complicated
- Solution: use Alloy Analyzer to automatically generate test cases

4 10/1/04

TestEra

- Can skip test cases that lead to bugs
- Generates data structures from structural invariants
 - Test code at concrete data type level
 - Do not need to construct using method calls

5 10/1/04

Alloy

- First-order declarative language
 - Atoms → types
 - Relations → variables
 - Signatures → user-defined type
 - Functions → methods
 - Specifications → classes
- Alloy Analyzer: fully automatic tool that finds instances of Alloy specifications

6 10/1/04

Linked List Test Case – Alloy Specification

```

module list
open java/primitive/integer
sig List {
  header: option Node
}
sig Node {
  elem: Integer,
  next: option Node
}
fun List::RepOk() {
  //acyclic
  all n: this.header.*next | n !in n.^next
}

```

7

10/1/04

Linked List Test Case – Merge Sort

```

//sorted
all n: header.*next | some n.next =>
  n.elem <= n.next.elem

//output is permutation of the input
all i: Integer |
  #{n: header.*next | n.elem = i} =
  #{n: header'.*next' | n.elem' = i}

```

8

10/1/04

TestEra Analysis

- At command line TestEra is given:
 - Method declaration
 - Method return type
 - Method name
 - Parameter types
 - Name of Java source file
 - Method body
 - Class invariant(s)
 - Pre/Post-condition(s)
 - Bound on input size

9

10/1/04

Two Phases of TestEra

- Phase I: generate input using Alloy Analyzer
 - Concretization translation (a2j)
- Phase II: Java test driver - tests each instance of the input and check for correctness
 - Abstraction translation (j2a)
 - If check fails, report a counterexample
 - If check succeeds, test next input

10

10/1/04

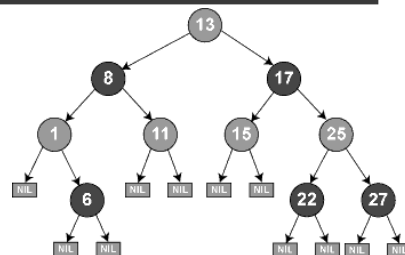
Case Study – Linked List

- Class invariant & pre-condition: acyclic
- Post-condition: acyclic, sorted in ascending order, permutation of original list
- No counterexamples initially for mergesort method
- Introduce an error
 - Reverse comparison from <= to >=
- Error detected
 - Output failed 1 of 3 post-conditions
 - Counterexamples found

11

10/1/04

Case Study – Red-Black Trees



12

10/1/04

Case Study – Red-Black Trees

- Class invariants: parent field must be contained in either a left or right subtree, tree must be a BST and must satisfy the four RBT properties
- TE does not produce any counterexamples on `remove()` method
- Explicitly create an error by swapping RED and BLACK in `restructure` command
- TE detects error and generates counterexamples

13

10/1/04

Case Study – Intentional Naming System (INS)

- Naming system intended for naming and discovering resources in networks of devices and computers
- Design flaws found in `Lookup-Name` function
 - Wildcards
 - Adding services to database

14

10/1/04

Case Study – Alloy-alpha Analyzer

- Tested two conjectures
 - Instances have distinct basic signatures
 - Atoms in signature are unique
- Generate simplified specifications then re-run them through Analyzer to see if same specifications generated
- Two counterexamples found that violated uniqueness properties

15

10/1/04

Implementation & Performance

Case Study	Property/Method Tested	Max size of a sig	Phase 1		Phase 2	
			#Tests	t[sec]	#Passed	t[sec]
Singly Linked Lists	<code>mergeSort</code>	4	42	10	42 (100%)	7
	<code>mergeSort (erroneous)</code>	4	42	0	0 (0%)	7
Red Black Trees	<code>deleteEntry</code>	5	54	31	54 (100%)	13
	<code>deleteEntry (erroneous)</code>	5	54	0	30 (56%)	13
INS	published wildcard claim	3	12	9	10 (83%)	6
	monotonicity of addition	4	160	14	150 (93%)	9
	<code>Lookup-Name (original)</code>	3	16	8	10 (62%)	6
AA	<code>Lookup-Name (corrected)</code>	3	16	0	16 (100%)	6
	disjoint sigs, unique atoms	2	12	5	6 (50%)	25

Table 1. Summary of TestEra's analyses

16

10/1/04

Test Cases

- When TestEra detects a violation of the property being tested it generates concrete counterexamples
- If no violation detected, can increase confidence in implementation by generating test inputs in a larger space
 - Rule out inputs generated by smaller bounds
- Can perform test generation on an input-by-input basis

17

10/1/04

Conclusions

- Several programs were quickly and effectively analyzed for the given small input bounds
- Violations of correctness property generated concrete counterexamples
- Approach promises scalability and wide application
 - Computation is not modeled, can be arbitrarily complex
- Plan to extend analysis to report on structural code coverage
 - Help users decide when program has been sufficiently tested

18

10/1/04

Resources

- First-order definition: http://en.wikipedia.org/wiki/First-order_predicate_calculus
- RBT graphic: http://en.wikipedia.org/wiki/Red-black_tree
- Alloy website: <http://geyer.lcs.mit.edu/alloy/reference-manual.pdf>
- INS website: <http://nms.lcs.mit.edu/projects/ins/>

19

10/1/04

Case Study – Alloy-alpha Analyzer

- Meta Specification (MS) of simplified Alloy specification
- Generated all instances (I) of MS, each representing a specification M
- AA takes each I and builds corresponding specification M'
- Invoke AA again to generate all I' of M'
- Check whether I' satisfies uniqueness properties

20

10/1/04