

Korat: Automated Testing Based on Java Predicates

Chandrasekhar Boyapati, Sarfraz
Khurshid, Darko Marinov

CSci 780 – Advanced Software
Engineering

11/14/2003

1

Outline

- MulSaw Project
- Problem
- Example
- Algorithm
- Testing Methods
- Results
- TestEra, Alloy Analyzer, and others
- Conclusions
- Discussion

11/14/2003

2

MulSaw Project

- MIT Students
- Goal - design and implement tools for checking *code conformance*
- Korat, TestEra, Alloy Annotation Language(AAL)
- MulSaw Web page

11/14/2003

3

Problem

- Traditional Testing
 - Manual software testing
 - Test data generation
- Labor Intensive
- Software Engineering Utopia – cheaper and better testing methods

11/14/2003

4

Proposed Solution

- Automated Testing!
 - Reduce the cost of development and maintenance

11/14/2003

5

General Idea behind Korat

- Specification-based testing
 - Current prototype uses Java Modeling Language(JML)
- Method Precondition
 - Generate all nonisomorphic test cases
- Execution
- Method Postcondition

11/14/2003

6

Introductory Illustration

- Method for removing the minimum element from a balanced binary tree
- Precondition – input is a binary tree and it is balanced
- Generate all balanced binary trees
- Execute method on all cases
- Postcondition

11/14/2003

7

Examples



11/14/2003

8

Examples

- Binary Tree
 - Linked data structure
- Heap Array
 - Array-based data structure

11/14/2003

9

Binary Tree example

```
class BinaryTree {
    private Node root; // root node
    private int size; //number of nodes in the tree
    static class Node {
        private Node left; // left child
        private Node right; // right child
    }
    public boolean repOk() {
        //class invariant
    }
}
```

11/14/2003

10

Binary Tree example(2)

- Generate test cases – Finitization

```
public static Finitization finBinaryTree(int NUM_Node) {
    Finitization f = new Finitization(BinaryTree.class);
    ObjSet nodes = f.createObjectSet("Node", NUM_Node);
    // #Node = NUM_Node
    nodes.add(null);
    f.set("root", nodes); // root in null + Node
    f.set("size", NUM_Node); // size = NUM_Node
    f.set("Node.left", nodes); // Node.left in null + Node
    f.set("Node.right", nodes); // Node.right in null+ Node
    return f;
}
```

- Mostly generated automatically

11/14/2003

11

Binary Tree example(3)

- Finitization function generated all nonisomorphic trees



- finBinaryTree(3) – 5 trees
- finBinaryTree(7) – 429 trees in < 1 sec

11/14/2003

12

Binary Tree example(4)

- Checking correctness of a method
remove(Node n)

```
// @ public invariant repOk();           // class invariant
//                                     // for BinaryTree
/*@ public normal_behavior               // specification for remove
 @ requires has(n);                     // precondition
 @ ensures !has(n);                      // postcondition
 */
public void remove(Node n) {
    // ... method body
}
```

- Korat translates JML into assertions

11/14/2003

13

Binary Tree example(5)

- Invoke the method with each test case generated
- Report concrete counterexample if runtime assertion fails
- If nothing fails, then every input possible(nonisomorphic) satisfies the correctness criteria before and after the method

11/14/2003

14

Algorithm



11/14/2003

15

Algorithm

- Key to test case generation
 - From a Java predicate(repOk or CheckRep) generate all nonisomorphic inputs
- Finitization
- State Space
- Search
- Nonisomorphism
- Instrumentation

11/14/2003

16

Finitization

- There must be a set of bounds for limiting size of inputs
- Inputs can be from several classes
- Must know number of objects from each class
- Class domain
- Field domain

11/14/2003

17

Finitization

- Korat allows finitization to be written in Java (familiarity)
- Korat automatically generates skeleton function from Java code for your class

11/14/2003

18

Finitization *skeleton*

```
public static Finitization finBinaryTree(int NUM_Node,
                                         int MIN_size, int MAX_size) {
    Finitization f = new Finitization(BinaryTree.class);
    ObjSet nodes = f.createObjectSet("Node", NUM_Node);
    nodes.add(null);
    f.set("root", nodes);
    f.set("size", new IntSet(MIN_size, MAX_size));
    f.set("Node.left", nodes);
    f.set("Node.right", nodes);
    return f;
}
```

- Programmer can specialize the finitization to meet certain criteria(in Java)

11/14/2003

19

State Space

- Inspect state space for BinaryTree(3) example
- Specified objects from finitization:
 - One BinaryTree, three Nodes
- Creates *candidate vector*
- candidate vector* for this example has 8 fields [root, size, 3*(left, right)]

11/14/2003

20

State Space(2)

- State space of inputs is all possible assignments to the fields



$4 * 1 * 4 * 4 * 4 * 4 * 4 * 4$

$= 4 * 1 * (4 * 4)^3 = 2^2 * (2^2)^3 = 2^{14}$

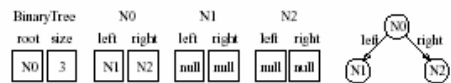
- Possible values of *candidate vector*

11/14/2003

21

State Space(3)

- Candidates can be valid...

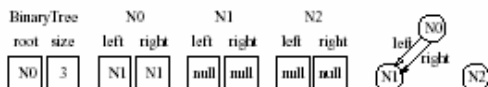


11/14/2003

22

State Space(4)

- ...or not valid



11/14/2003

23

Search

- Korat seeks to systematically explore the state space
- class and field domains are ordered in the *candidate vector*, such that each entry is an index to the field domain
- Initially all values are set to 0

11/14/2003

24

Search(2)

- Korat sets the objects according to the vector values
- Executes `repOk` method on candidate
- Records the fields accessed during invocation of `repOk()`
- Generates next vector through backtracking (increments field that was accessed last)

11/14/2003

25

Search(3)

- Change only fields that were accessed
- Prunes the search space dramatically
- Does NOT rule out any valid data structure
- Quality of `repOk` method determines extent of pruning

11/14/2003

26

Nonisomorphism

- Further optimization of generation process
- Definition of isomorphism:

Two candidates, C and C' , are *isomorphic* iff there is a permutation π on O , mapping objects from O_i to objects from O_i for all $1 \leq i \leq n$, such that:

$\forall o, o' \in O_C. \forall f \in \text{fields}(o). \forall p \in P.$
 $o.f == o'$ in C iff $\pi(o).f == \pi(o')$ in C' and
 $o.f == p$ in C iff $\pi(o).f == p$ in C' .

11/14/2003

27

Nonisomorphism(2)

- What this means: isomorphism is defined by the root object
- Candidates partitioned by the isomorphism criteria
- Only the "smallest" from a partition is generated
- **finBinaryTree(3) example** – 1 tree generated per every 3! distinct trees

11/14/2003

28

Instrumentation

- Korat adds a special constructor, identifier field, and get and set methods for any classes in the `finitization` and `repOk`

11/14/2003

29

Testing Methods

- Generation of finitization for methods
- Small amount of work left for user
 - When Korat cannot determine a field automatically, inserts `/***/`. User must replace this with the appropriate field domain

11/14/2003

30

Testing Methods

- Dependent and Independent Parameters
- Korat(independent) – generate separately and take cross product
- Other methods(dependent and independent) – use the cross product by manually constructing all possibilities

11/14/2003

31

Checking Correctness

- Generate all valid inputs
- Invoke the method with each input
- Check the output with the test oracle (postcondition or class invariant)
- Results of false also provide concrete counterexample in the form of the input

11/14/2003

32

Checking Correctness

- Korat combines and adds to the ideas of testing from JUnit and JML+JUnit
 - Adds automation of test cases

testing activity	Testing framework		
	JUnit	JML+JUnit	Korat
generating test cases			✓
generating test oracle		✓	✓
running tests	✓	✓	✓

11/14/2003

33

Results

- Benchmarks
 - BinaryTree
 - HeapArray
 - LinkedList
 - TreeMap
 - HashSet
 - AVTree

11/14/2003

34

Results

- Due to backtracking, Korat can cover the whole state space

benchmark	size	time (sec)	structures generated	candidates considered	state space
BinaryTree	8	1.53	1430	54418	2^{283}
	9	3.97	4862	210444	2^{613}
	10	14.41	16796	815100	2^{72}
	11	56.21	58786	3162018	2^{82}
	12	233.59	208012	12284830	2^{92}
HeapArray	6	1.21	13139	64533	2^{20}
	7	5.21	117562	519968	2^{16}
	8	42.61	1005075	5231385	2^{19}
LinkedList	8	1.32	4140	5455	2^{91}
	9	3.58	21147	26635	2^{108}
	10	16.73	115975	142646	2^{136}
	11	101.75	678570	821255	2^{155}
	12	690.00	4213597	5034894	2^{166}

11/

35

Results

- The specifications of correctness for methods were simple containment properties plus the class invariants

benchmark	method	max. size	test cases generated	gen. time	test time
BinaryTree	remove	3	15	0.64	0.73
HeapArray	extractMax	6	13139	0.87	1.39
LinkedList	reverse	2	8	0.67	0.76

11/14/2003

36

TestEra, Alloy, and others

- Alloy – a declarative specification language based on first order logic
- Alloy Analyzer(AA) – a tool for analyzing models written in Alloy, designed to do similar things as Korat
 - Because of pruning and nonisomorphism, Korat generates test cases much faster than AA
- TestEra – generates test cases from specifications written in Alloy

11/14/2003

37

The Others

- Static Analysis
 - Use a theorem prover to verify partial correctness
 - Promising technique, but still limited
- Model Checking
 - More focused on checking sequences rather than data structures

11/14/2003

38

Conclusion

- Korat can use preconditions to generate test cases, execute a method on all test cases, then use the postconditions as a test oracle
- Korat utilizes JML, which is very similar to Java itself
- Korat generates and tests much faster than its buddy Alloy Analyzer

11/14/2003

39

Discussion

- Is automated generation and testing the wave of the future?
- Are there downsides or limitations to this method of testing?
- Is Korat generating not enough or too many test cases?
- For the linked data structures, Korat produced, at most, structures of size 12. Is this domain large enough to test for correctness?

11/14/2003

40

Future Work

- Future research possibilities on the MulSaw page:
 - <http://mulsaw.lcs.mit.edu/projects.html>

11/14/2003

41