

Integration of Formal Assertions with Bounded Exhaustive Testing

Ashwin Mundra and Richard Dutton
Software Engineering 780

1

Outline

- Introduction
- Background & Related Work
- Approach
- Experiment
- Results
- Evaluation
- Future Work
- Conclusion

2

Introduction

- Testing – a partial proof of correctness
 - Therac-25 Accident
- Exhaustive testing is infeasible
- Formal Specification based assertions
 - Higher Confidence
 - Runtime Checks
 - Computation of output no longer necessary
- Bounded exhaustive testing
 - All possible input up to a given bound

3

Background & Related Work

- Assertions
 - Powerful tool for detection of errors but little widespread use – Rosenblum [1995]
 - Reasons for not being used
 - Programmers have little idea of what information should be specified in assertions
 - Tools for adding assertions are inadequate
 - Annotation Preprocessor (APP)
 - More study needed

4

Assertions Continued...

- Meyers's design-by-contract [1997]
 - Runtime Checks
 - Method preconditions and postconditions
- Our Formal Assertions
 - Validation
 - Verification
 - Act as test oracles

5

Testing

- TestEra
 - Bounded test data set on being given the formal specification
 - Framework of TestEra
 - Generating input by using Alloy Analyzer (AA)
 - Testing done on each instance
 - Resultant output again converted back to Alloy
- Pseudo-Exhaustive Testing
 - Testing done on the hardware level

6

Approach

- Formal Specification Based Assertions
- Bounded Exhaustive Testing

7

Example

- Implementation of a *Sort()* algorithm

```
list<> r = Sort(list<> r, r.length);
list<> Sort(list<> inList, int sizeOfArray) {
    for (int i = 0; i < sizeOfArray; i++) {
        int temp = inList[i];
        for (int j = i; j > 0 && (inList[j-1] > temp); j--)
            inList[j] = inList[j-1];
        inList[j] = temp;
    }
}
```

8

Formal Specification based Assertions

- Specification of Sort

Sort

$Sort : seq \mathbb{Z} \rightarrow seq \mathbb{Z}$

$\forall in, out : seq \mathbb{Z} \bullet Sort(in) = out \Leftrightarrow$

$items(in) = items(out) \wedge$

$(\forall i, j : 1..#out \mid i < j \bullet out(i) \leq out(j))$

9

Derivation of Assertions

```
list<> Sort(list<> inList, int sizeOfArray) {
    int *oldList = copyArray(inList, sizeOfArray);

    for (int i = 0; i < sizeOfArray; i++) {
        int temp = inList[i];
        for (int j = i; j > 0 && (inList[j-1] > save); j--)
            inList[j] = inList[j-1];
        inList[j] = temp;
    }

    assert(IsPermutation(oldList, inList, sizeOfArray);
    for (int i = 0; i < size - 1; i++)
        assert(inList[i] <= inList[i+1]);
}
```

- Correctness ensured – Acts as proof of correctness

10

Bounded Exhaustive Testing

- Scope
- *Sort()* example continued...
- Enumeration could involve two areas of input
 - Possible lengths of the *list*, where the length of the list is x

$$0 \leq x < \infty$$

- Characteristics of elements

11

Characteristics of Elements

- Different combination of numbers
 - Number of distinct elements in the list can range from 1 to the length of the list
- Types of Value
 - Positives, negatives, zero or any combination

12

Tabulating Distinct Values for a list r

Length of r	Distinct values in r
0	0
1	1
2	1,2
3	1,2,3

13

Experiment

- Case Study: Nova Solver
 - Computes reliability of a system
 - Takes a fault tree as input
- Add formal specification based assertions
- Conduct bounded exhaustive testing
- Generate results

14

Adding Formal Assertions

- Understanding of Nova Solver
- Conversion of formal specification to assertions
- Lock subsections of code

15

Bounded Exhaustive Testing

- Cover substantial portion of the functional domain
 - Test up to a given bound
- Use TestEra
 - Just for generation
- Run the test cases on Nova Solver version containing formal specifications
- No longer need to verify correctness of the output

16

Experimental Results

- Each phase of the project revealed bugs and inconsistencies
- Initial code review findings
 - Certain operator overloading defined in the specification was not implemented
 - **Probability** constructor parameter
 - **Fault_Tree** copy constructor

17

Formal Assertions

- 2300 lines of C++ code added by 3 programmers over a 2 month period
- Two implementation bugs found through unit test suite
 - **Time** and **Threshold** objects
- Unit test suite problems
 - Creating illegal states and objects
 - Resolution: *finalized()* method

18

Bounded Exhaustive Testing

- Scope of dynamic fault trees
 - Natural
 - Injective Sequence
 - Event
 - Functional Dependency
- Currently generated and tested through scope 3

19

Bounded Exhaustive Testing(2)

- Table II

Main Scope	Test Cases	Failures	Type
natural2	56525	0	Unconstrained
natural3	2870095	18552	Unconstrained
natural3	48790	8260	Constrained

- Failures correspond to 2 distinct bugs
 - **Probability** greater than 1 – due to numerical imprecision
 - *First_Occurrence_Time()* – copy-and-paste error

20

Bounded Exhaustive Testing(3)

- Timings – Test cases run on Intel Pentium 4 3.00 GHz processor
- Table III

Main Scope	Sub-Scopes	Test Cases	Time	Type
natural2	13	56525	33.5 hrs	Unconstrained
natural3	19	2870095	346 hrs	Unconstrained
natural3	12	48790	11.7 hrs	Constrained

- Note: much of the time is I/O cost

21

Evaluation

- Advantages
 - Undetected bugs in the system were found
 - Computation of the output is unnecessary
 - Our approach revealed inadequacies of the original traditional test suite

22

Evaluation (2)

- Limitations
 - Formal assertions are not trivial to add
 - The mapping from specification to code may lead to translation problems
 - Bounded exhaustive testing is expensive

23

Evaluation(3)

- Problems with the experiment
 - Tested only up through scope 3
 - Bounded testing done without optimization flags
 - I/O costs

24

Future Work

- Using it on a real system as it is being developed
- Identifying right bound
 - Determining the distribution of errors
 - When to stop testing?
- Optimize assertions to minimize performance degradation
- Tweaking of assertions to avoid numerical imprecision

25

Conclusion

- Integration of formal assertions and bounded exhaustive testing is a novel approach
- Favorable initial results
- Most suitable for critical systems
- We highlighted the pros and cons of using our method
- More research is needed for complete evaluation of the method

26

References

- [1] E. W. Dijkstra, "The humble programmer," *Communications of the ACM*, vol. 15, no. 10, pp. 859–866, Oct. 1972.
- [2] W. R. Adrion, M. A. Branstad, and J. C. Cherniavsky, "Validation, verification, and testing of computer software," *ACM*, vol. 14, no. 2, June 1982.
- [3] D. S. Rosenblum, "A practical approach to programming with assertions," *IEEE Transactions on Software Engineering*, vol. 21, no. 1, pp. 19–31, Jan. 1995.
- [4] P. E. Ammann, S. S. Brilliant, and J. C. Knight, "The effect of imperfect error detection on reliability assessment via life testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 2, pp. 142–8, Feb. 1994.
- [5] D. C. Luckham and F. W. von Henke, "Overview of Anna, a specification language for Ada," *IEEE Software*, vol. 2, no. 2, pp. 9–224, Mar. 1985.
- [6] J. M. Voas and K. W. Miller, "Putting assertions in their place," in *Proceedings of the International Symposium on Software Reliability Engineering*, Monterey, CA: IEEE, 6–9 Nov. 1994.
- [7] B. Meyer, *Object-Oriented Software Construction*, 2nd ed. Prentice Hall, 1997.

27

References

- [8] D. Marinov and S. Khurshid, "TestEra: A novel framework for automated testing of Java programs," in *Proceedings of the 16th IEEE Conference on Automated Software Engineering (ASE 2001)*, San Diego, CA: IEEE, 26–29 Nov. 2001.
- [9] R. W. Butler and G. B. Finelli, "The infeasibility of quantifying the reliability of life-critical real-time software," *IEEE Transactions on Software Engineering*, vol. 19, no. 1, pp. 3–12, Jan. 1993.
- [10] W. B. Jone, J. C. Rau, and S. C. Chang, "A tree-structured lfsr synthesis scheme for pseudo-exhaustive testing of vlsi circuits," *IEEE*, Oct. 1998.
- [11] J. M. Spivey, *The Z Notation: A Reference Manual*, 2nd ed. Prentice Hall International Series in Computer Science, 1992.
- [12] D. Coppit, "Engineering modeling and analysis: Sound methods and effective tools," Ph.D. dissertation, The University of Virginia, Charlottesville, Virginia, Jan. 2003, uRL: <http://www.cs.wm.edu/coppit/papers/dissertation.pdf>.
- [13] K. J. Sullivan, J. B. Dugan, J. Knight, *et al.*, "Galileo: An advanced fault tree analysis tool," 1997, uRL: <http://www.cs.virginia.edu/ftree/index.html>. [Online]. Available: <http://www.cs.virginia.edu/ftree/index.html>

28

Thank You

29

Derivation of Assertions

```
list<> Sort(list<> inList, int sizeOfArray)
{
    int *oldList =
    copyArray(inList, sizeOfArray);
    //Implementation details

    assert(IsPermutation(oldList, inList,
    sizeOfArray);
    for (int i = 0 ; i < size - 1 ; i++)
        assert( inList[i] <= inList[i+1] );
}
```

- Correctness ensured – Acts as proof of correctness

30