

Dynamically Discovering Likely Program Invariants to Support Program Evolution

by
Michael D. Ernst
Jake Cockrell
William G. Griswold

10/15/2003

1

Outline

- Role of Invariants
- Daikon Description
- Simple Example
- Choosing What to Infer
- How Daikon Works
- Daikon in Use
- Scalability of Daikon
- Use of Test Suites
- Future Work
- Summary, Critique, and Discussion

10/15/2003

2

Role of Invariants

- Program development
 - Refining specification into a correct program
 - Static verification of properties (type declarations)
 - Runtime checking (assert statements)
- Software evolution
 - Prevents changes from violating assumptions for correct behavior
- Documentation of program's operation and data structures

10/15/2003

3

Inferring Invariants

- Inferred statically or dynamically
- Dynamically inferred invariants are inferred from captured variable traces
 - Depend on the test suite
- Uses
 - Double-check existing documentation
 - Indicate a bug
 - Assist in test-case generation or validate a test suite

10/15/2003

4

Daikon

- Dynamic invariant detector
 - Runs program with a test suite
 - Infers invariants from program runs
 - Can detect context-dependent properties
- Static analyses are limited
 - Typically sound, but conservative
 - Limited by uncertainty

10/15/2003

5

Simple Example

Array Sum Program

```
i, s := 0, 0;  
do i != n ->  
    i, s := i + 1, s + b[i]  
od
```

Precondition: $n \geq 0$

Postcondition: $s = (\sum j : 0 \leq j < n : b[j])$

Loop invariant: $0 \leq i \leq n$ and $s = (\sum j : 0 \leq j < i : b[j])$

10/15/2003

6

Array Sum Program

```
i, s := 0, 0;
do i != n ->
  i, s := i + 1, s + b[i]
od
```

- Preconditions, postconditions, and loop invariants were specified by author
- Ran Daikon on 100 randomly-generated arrays of length 7 to 12
- Found the fundamental invariants
 - $S = \text{sum}(B)$ (postcondition)
- Also found some not specified by author
 - $N = \text{size}(B)$ (obvious?)
- Shows potential, but not overly impressive

10/15/2003

7

Choosing what to Infer

- Invariants involving a single variable
 - $x \in \{a, b, c\}$ (limited values)
 - $x \leq a$
- Invariants involving multiple variables
 - $x = ay + bz + c$
 - $x \geq y$
- Invariants over sequence variables
 - min/max, range, nondecreasing
 - x is a subsequence of y

10/15/2003

8

Performing the Inference

- Infers at specific program points
 - Procedure entries and exits
 - Loop heads
- Each potential invariant instantiated and tested for each variable or variable tuple
 - Only checked once if shown false
 - Cost roughly proportional to number of invariants discovered
- Does not include all potential invariant types
 - Users can add their own invariants

10/15/2003

9

Performing the Inference (2)

- Computes invariant confidences
 - Probability that property is random
 - Only reports invariants above specified confidence level
 - Avoids long list of spurious invariants
 - Larger, more complete test suite helps
 - Shows counterexamples
 - Adds to confidence if it agrees

10/15/2003

10

Performing the Inference (3) Derived Variables

- Derives variables, treated like all others
- Derived from a sequence
 - $\text{Size}(s)$
 - $S[\text{size}(s)]$
 - $\text{Sum}(s)$
- Derived from a sequence and variable i
 - $s[i]$
 - Subsequences: $s[a..b]$
- Derived in iterations
 - Stops computing derivations of derivations after a fixed number of iterations

10/15/2003

11

Program Instrumentation

- Daikon instruments the program to output information
- All values of variables in scope written to data trace file at each program point
- Daikon operates only with scalar numbers and arrays of numbers
- Use Abstract Syntax Trees to determine variables in scope
- Insert code at program point to dump variable values into output file

10/15/2003

12

Program Instrumentation (2)

- Computation based programs slowed down the most by I/O of instrumentation
- Must maintain runtime status information on each variable
 - Initialized or not?
 - Pointer references array or scalar?

10/15/2003

13

Daikon in Use

- Used program `replace`
 - Takes a regular expression and replacement string as arguments
 - Replaces instances of the regular expression with the replacement
 - Has no comments or documentation
- Wanted to add Kleene-+ closure
 - Kleene-* closure included
 - `a+` = one or more copies of `a`
- 2 programmer team to do this

10/15/2003

14

Daikon in Use (2)

- Used 100 test case suite for Daikon
 - 5542 were provided, these randomly chosen
 - Invariants produced at entry and exit of procedures
 - Output provided to the programmers
- Began by analyzing the call structure and high-level definitions

10/15/2003

15

Daikon in Use (3)

- Changes made based on previous code
 - Modified version of Kleene-* closure
- Analyzed the data structures used
- Looked at invariants to confirm assumptions they gathered
- Queried trace database based on invariants
- Found an array bound error
 - Had been previously undetected, despite 5,542 test cases in test suite

10/15/2003

16

Daikon in Use (4)

- The trace database also helped understanding of how pattern was stored
- Added test cases for Kleene-+
- Recomputed invariants
 - Mostly same, appropriately changed
- Found untested code
 - Result of the small test suite

10/15/2003

17

Results

- Clearer understanding of data structures
- Confirmed/contradicted expectations
 - Refuting expectation helped their understanding
 - Prevents future bug introduction based on flawed understanding
- Revealed a bug
- Showed inadequacy of test suite
- Validated program changes

10/15/2003

18

Analysis

- Invariants provide insight that would be difficult to manually extract
- Leads to thinking in terms of invariants
 - Fewer errors
- Reveal facts that would not have been noticed
- 2 Daikon tools were useful
 - Trace database queries
 - Invariant set comparator (for changes)

10/15/2003

19

Scalability

- Factors leading to increased costs
 - Number of program points
 - Proportional to program size
 - Number of variables
 - Potentially cubic (tuples involve up to 3)
 - Most important factor
 - Number of invariants checked
 - Number of test cases
- Analysis of portion of large program same as an entire small program

10/15/2003

20

Test Suites

- Play a large role in Daikon
- Too few test cases
 - Small number of invariants (not justified statistically)
 - Spurious invariants
- Too many test cases
 - Linearly increases run time
- Range where benefit per test-case increase is greatest
- Program correctness improves invariants

10/15/2003

21

Test Suite Generation

- Randomly-generated
 - Ineffective
 - Miss many portions of code
 - Many inputs are invalid, wasting time
- Grammar-generated
 - Random, but generated from grammar describing valid inputs
 - More effective, but more effort
 - Can be simplified by allowing some errors

10/15/2003

22

Future Work

- Increase the relevance of discovered invariants
- Improve performance
 - Very little optimization used
 - Much room for improvement
- Better invariant view tools
 - Easier to find relevant invariants
 - Could categorize or predict usefulness
- Richer Invariants
 - Discover over pointer-based recursive data structures

10/15/2003

23

Summary

- Discovering invariants on execution traces can be feasible and effective
- Found all invariants in set of formally-specified programs (sort of)
- Useful in software evolution task
- Using method can qualitatively affect programmers
- Useful invariants at acceptable cost

10/15/2003

24

Critique

- Paper makes positive aspects sound better than they are
 - “automatically detected all the stated invariants in a set of formally-specified programs”
 - “invariants erroneously omitted from the formal specification but detected by Daikon”
- Would be very slow for large programs
- Does show some positive qualities though

10/15/2003

25

Discussion

- What types of programs would Daikon be good/bad for?
- Is it too slow for legitimate use? (563 line C program w/ 21 procedures, 3000 test cases: 540 seconds)
- Is the reliance on the test suite a major problem?
- Other Potential Uses?
- Too cumbersome with useless invariants output?

10/15/2003

26