

Dynamically Discovering Likely Program Invariants to Support Program Evolution

Written by Michael J. Ernst, Jake Cockrell,
William G. Griswold, and David Notkin

Paper Discussion

The Problem

The Solution

Detecting Invariants

Applying to an Example

Scalability

Test Suite Usage

Conclusion

Discussion

The Problem

How do we ensure software maintains its intended purpose during code development and evolution stages?

Proposed Solution

Identify program properties that must be preserved

- **Discover invariants**
 - **Instrumenter – Daikon Tool**
 - **Inference Engine**
- **Apply solution**

Invariants

Definition - A constant or unchanging quality

Usage

- Program development
- Software Evolution
- Inference

How Does This Work?

Instrumentation (Daikon – invariant detector)

- Where do we insert instrumentation?**
- What variables do we examine?**
- How do we know we are on the right track?**

Example – Gries program

i, s := 0,0;

do i != n ->

i, s := i + 1, s + b[i]

od

Invariant – Infer or Derive?

Inference

- Detects at each instrumentation point
- Negative invariants
- Coincidental ranges

Derivation

- Infer invariants not hard-coded into its list
- Expressions
- Usually not introduced until after pre-existing invariants

Siemens - replace

563 lines of undocumented C code

- Command line arguments – regular expression and replacement string**
- Copies an input stream to an output stream**
- Replaces substring matched by the regular expression with the replacement string**

Invariants and *replace*

Evaluated with invariants and found need for modifications

- Unreported array bounds error
- Program compiled literals by prefixing them with the character `c` and puts the Kleene-* expressions into prefix form

Dynamically Detected Invariants

Explicated data structures

**Confirmed and contradicted
expressions**

Bug revelation

Better Programmers?

Invaluable insight

Programmers can validate their own inferences

Lead to better software development

Early bug detection

Scalability

Dynamic invariant inferences grow

- Program points**
- Variables instrumented**
- Variables checked**
- Test cases run**

Instrumentation Points

Daikon inferred invariants over an average of 71 variables per instrumentation point in the *replace* program

On average 1000 test cases produce 10,120 samples per instrumentation point

Daikon took 220 seconds to infer the invariants for an average instrumentation point

3000 test cases (33801 samples) took 540 seconds

Variables Instrumented

Most important factor

Encompasses number of variables in scope at a program point

– Grows slowly with program size

Difficult to predict the number of derived variables

Invariant detection grows quadratically with the number of variables over which invariants are checked

Test Suite Size

Less pronounced

Runtime is linearly related to test suite size

Improvements

**Global variables, large-scale structures,
module entry and exit points only**

Points of interest

Performance enhancements

- **Supplying fewer test cases**
- **Checking fewer invariants**
- **Only derive more complicated variables**

Text editor

Related Work

Dynamic Inference

– Artificial Intelligence

- Inductive Logic Programming
- Neural Nets

Static Inference

- Operate on program text but conservative
- Limited by uncertainty about properties beyond capabilities

Conclusions

Advantageous to work on programs not created by the authors

Can be applied to larger systems if performance enhanced

Dynamically inferred invariants are useful

- Test case generation**
- Validation of a test suite**

Discussion

How do we develop a balance between invariant detection overkill and performance degradation?

How does this compare to modern programming debugging tools?

How can this be applied to other programming languages?

How can applying this help the design of future software?