

Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code

Dawson Engler, David Yu Chen, Andy Chou, and Benjamin Chelf

1

Outline

- Introduction
- Background
- Method
- Implementation
- Related Work
- Conclusion
- Future Work

2

Introduction



3

Introduction

Software Bugs:

- Increase development cost
- Increase time-to-market
- Damage reputation
- Cause unusual or unpredictable problems

4

Introduction

Major Software Problems:

- 1979: Five nuclear reactors shut down
- 1983: Soviet early-warning system causes false-alarm
- 1987: Chicago cat owners billed for unlicensed Dachshunds
- 1988: Missile system shoots down commercial flight (USS Vincennes, Airbus 320)

5

Introduction

Searching For Bugs:

- *Proactive* versus *Reactive*
- Eliminate unexpected surprises
- Bugs discovered earlier are typically less expensive to fix

6

Background

Bug Detection Software:

- Augments traditional testing methods
- Attempts to find as many serious bugs as possible

Problem:

- It is difficult to determine the appropriate rules for each system

7

Background

Solution:

- Create an automated method for determining correctness rules

8

Background

How to find a lie?

- Cross-check statements from numerous witnesses
- For each contradiction, at least one statement is false

9

Background

How to determine correct behavior?

- Observe numerous examples
- Assume the majority is correct

10

Method

- Beliefs
- Internal Consistency
- Statistical Analysis

11

Method

Beliefs:

- Beliefs are facts about the system implied by the code
- MUST beliefs – directly implied by the code, there is no doubt that the programmer has that belief
- MAY beliefs – suggested by the code, but may instead be a coincidence

12

Method

Uses for Beliefs:

- For each MUST belief – search for contradictions
- For each MAY belief – attempt to separate valid beliefs from coincidences

13

Method

Separating Valid MAY Beliefs From Coincidences:

- Temporarily assume MAY beliefs are MUST beliefs and search for contradictions
- Use statistical analysis to determine the likeliness that the beliefs are valid

14

Method

Internal Consistency:

- Infer MUST beliefs at one code location
- Propagate MUST beliefs to related locations
- Conflicts found are errors

15

Method

Inferring MUST Beliefs:

- Direct observation or implied pre- and post-conditions
- Direct observation uses standard compiler analysis to track actions that reveal code state
- After changing state, the programmer must believe the state change took effect

16

Method

- Actions that have specific pre- and post-conditions infer beliefs

Examples:

- Deallocation of a pointer implies it was dynamically allocated
- Division by a number implies that number is non-zero

17

Method

Example - Null Pointer Consistency

- The rule – “do not dereference null pointer <p>”
- Possible belief sets for <p> - {}, {null}, {not null}, {null, not null}
- For pointer <p> propagate it's set of belief values through the code

18

Method

If an action dereferences a pointer –

1. Signal an error if p's belief set contains the belief "null"
2. Add the belief "not null" to p's belief set

19

Method

Propagating Beliefs:

- If an action implies a belief, that belief must be propagated
- This can mean propagating forward, backward, from caller to callee, or to any other related code

20

Method

Example: If-Then-Else statement with p==NULL

- The true branch is propagated with the "null" belief
- The false branch is propagated with the "not null" belief
- After joining, both are included

21

Method

Statistical Analysis:

- MAY beliefs are acted on as MUST beliefs
- The more checks a belief passes, the more credible the violations of it are
- The highest ranked errors will be those with the most examples and the fewest counter examples

22

Method

Example – Statistical Lock Analysis

Code Segment:

```
lock(l); // Enter critical section
a = a + b; // MAY: a,b protected by
          l
unlock(l); // Exit critical section
```

23

Method

- If lock `l` protects variable `a` in 95 times out of 100, it is likely that this is the intended belief
- If lock `l` protects variable `b` less frequently, that belief will receive a lower ranking

24

Implementation

Checkers:

- Null Pointer
- Unsafe User-Pointer
- Failure Point
- Temporal Rules

25

Implementation

Implementing Checkers:

- Analyses written in metal, a high-level state machine language for writing systems-specific compiler extensions
- These extensions are dynamically linked using xgcc, an extended version of gcc
- The main features of each extension are relatively simple (50-200 lines of code)

26

Implementation

Null Pointer Checker:

- Linux 2.4.7

Checker:	Bug	False
check-then-use	79	26
use-then-check	102	4
redundant-checks	24	10

27

Implementation

Unsafe User-Pointer:

- OpenBSD 2.8, Linux 2.4.1, Linux 2.3.99
- (x) - cross-checking functions pointers

OS	Bug	False	Applied
OpenBSD 2.8	18	3	1645
Linux 2.4.1	12 (3)	16 (1)	4905
Linux 2.3.99	5	n/a	n/a

28

Implementation

Failure Point Checker:

- Linux 2.4.1, OpenBSD 2.8
- Bugs – derived + unknown functions

Version:	Bug	False
Linux 2.4.1	52+102	16
OpenBSD 2.8	27+14	21
Total	195	37

29

Implementation

Temporal Rules Checker:

- Linux 2.4.1

Rule	Bug	False
No <a> after 	23	11
 must follow <a>	23	11

30

Related Work

- Daikon – uses derived program rules, does not find a large number of bugs
- Eraser – flags inconsistent behavior, able to find a relatively large number of bugs
- Both of these use dynamic checking

31

Conclusion

- The approach is a significant improvement over using manually specified rules
- This greatly reduces the manual labor involved in re-targeting to new systems
- The approach is shown to work on real systems code (6 template checkers)

32

Future Work

- Using machine learning techniques to automatically generate temporal rules and rule templates from source code
- Forming a general model of actions and control flow using probabilistic automata with probabilities initialized from static branch prediction

33

Resources

Painter, Robert. "Bugs As Deviant Behavior."
<http://www.cs.wm.edu/~coppit/csci780-fall2003/presentations/09-deviant-behavior.pdf>

Dallas Fort Worth Pest Control. "Cool Bug Pictures."
<http://www.dfwpest.com/beetles.htm>

34