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

CSE231 Presentation
Lakshmi Krishnaswamy

Background and motivation

- Different methods used to find errors in a system
    - Testing and manual inspection
    - Type systems
    - Formal verification
    - High-level compilation
    - Dynamic analysis

image reference

Motivation

- Often difficult to derive the exact correctness rules for a system
- How can we still design a checker without prior knowledge about the system ?

image reference

Design Principles

- Requires no knowledge about system correctness rules
- Infer programmer’s beliefs from source code
    - *“if two beliefs contradict, we know that one is an error without knowing what the correct belief is.”*
- If there is a contradiction, then there is ***at least one*** statement which is wrong.

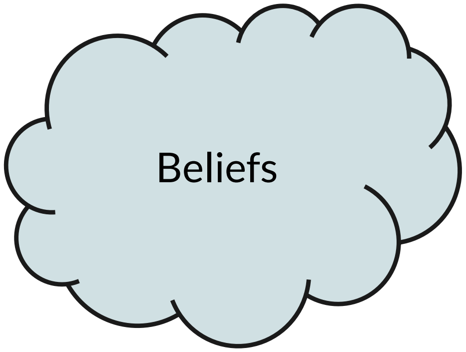
System Design

Inferring Beliefs

Templates

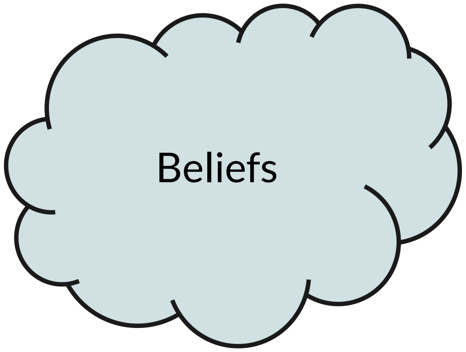
Checkers

System Design



- MUST beliefs, directly implied by the code
- Any contradiction means there is an error in the code

System Design



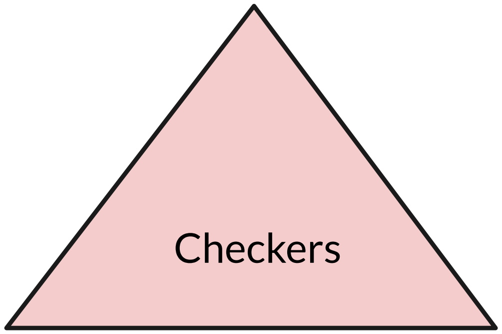
- MAY beliefs, suggested beliefs, could be a coincidence
- Not all contradictions are errors
- Need to separate out noise from errors

System Design

Templates

- Outline for a rule
- Example, <a> must be paired with <b>
  - <a> and <b> positions are slots
  - Filled with elements from source code
    - Slot instances, example “lock” and “unlock” function calls.

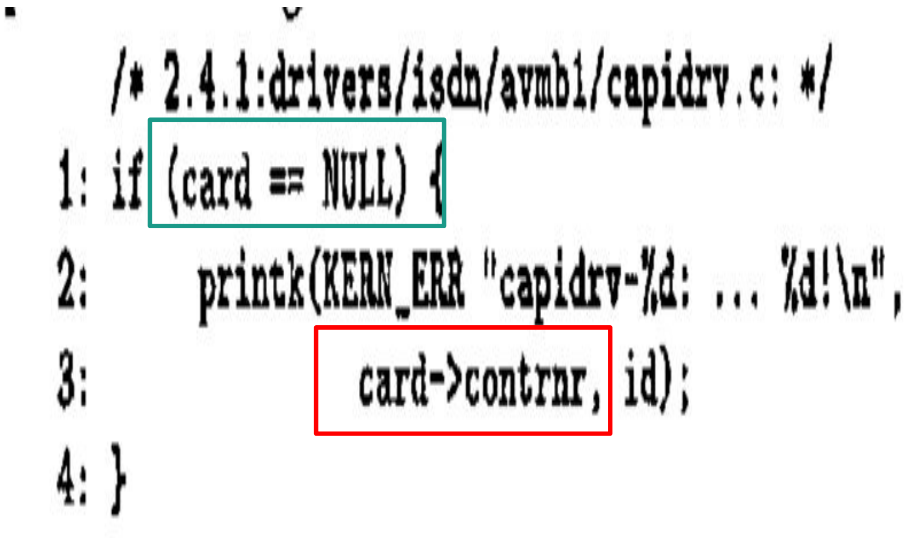
System Design



- General method for finding bugs
- Internal consistency checkers used with MUST beliefs
- Statistical analysis checkers used with MAY beliefs

Framework for internal consistency checkers

- The rule template T
- The valid slot instances for T
- The code actions that imply beliefs
- The rules for how beliefs combine, including the rules for contradictions
- The rules for belief propagation



```
/* 2.4.1:drivers/isdn/avmb1/capidrv.c: */
if (card == NULL) {
        printk(KERN_ERR "capidrv-%d: ... %d!\n",
                card->contrnr, id);
}
```

Reference : from paper

Internal consistency checkers

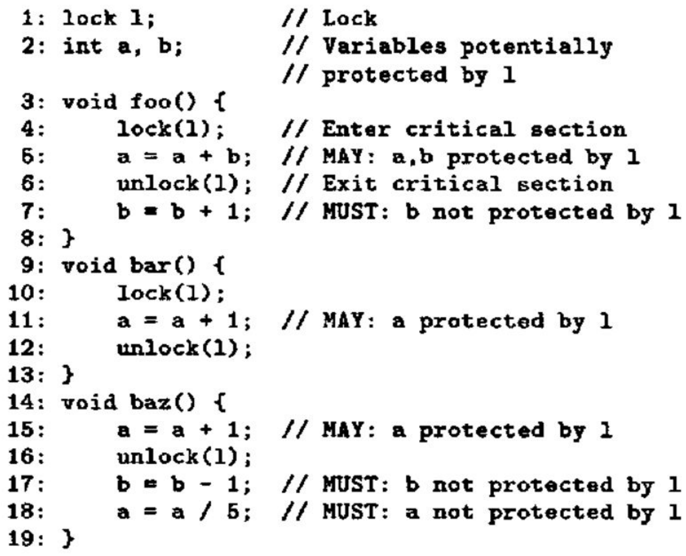
- MUST beliefs inference
    - Direct observation
    - Implied pre and post conditions

- More beliefs found, more applicable the checker

- Ranking results not necessary because a single contradiction results in an error

Framework for statistical analysis checkers

- It applies the check to all potential slot instance combinations, it assumes that all combinations are MUST beliefs.
- It indicates how often a specific slot instance combination was checked and how often it failed the check (errors).
- It is augmented with a function, *rank*, that uses the count information above to rank the errors from all slot combinations from most to least plausible.



```
lock l;          // Lock
int a, b;        // Variables potentially
                 // protected by l
void foo() {
     lock(l);    // Enter critical section
     a = a + b;  // MAY: a,b protected by l
     unlock(l);  // Exit critical section
     b = b + 1;  // MUST: b not protected by l
}
void bar() {
     lock(l);
     a = a + 1;  // MAY: a protected by l
     unlock(l);
}
void baz() {
     a = a + 1;  // MAY: a protected by l
     unlock(l);
     b = b - 1;  // MUST: b not protected by l
     a = a / 5;  // MUST: a not protected by l
}
```

Figure 1: A contrived, useful-only-for-illustration example of locks and variables

Statistical analysis checkers

- z statistics for proportions used for sorting between noise and errors

$$z(n, e) = (e/n - p_0)/\sqrt{(p_0 * (1 - p_0)/n}$$

n = number of check
e = number of successful checks
p_0 = probability of the examples

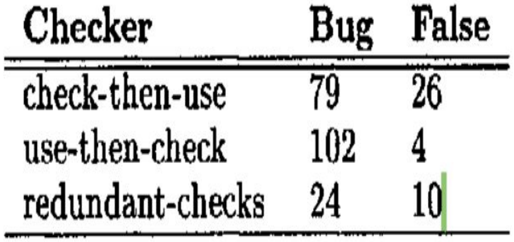
- Latent specifications to prune the search space

Performance Evaluation

- Analyses written using ***Metal*** - high- level state machine (SM) language for writing system- specific compiler extensions
- Tested on Linux and OpenBSD
    - Linux 2.4.1 and 2.4.7
    - OpenBSD 2.8
- Checkers implemented and tested
    - Internal Null Consistency
    - Security Checker
    - Failure Checker
    - Temporal rules derivation

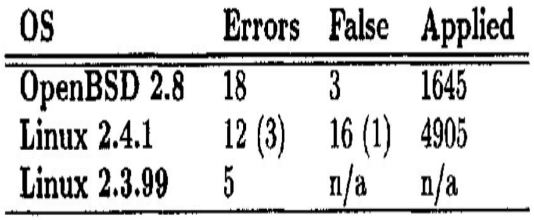
Performance Evaluation

Internal Null Consistency : Finds pointer errors, flags three types of contradictory or redundant beliefs



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

Security : checks for kernel safe pointers, and “tainted” pointers, raise error if a pointer is both



OS	Errors	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

What’s the overhead associated with
so many applications of the checker ?!

Performance Evaluation

Failure checker : Find routines that aren’t checked for failures

- Found some unexpected, error - not detected before !! IS_ERR consistency checking

Violation of temporal rules : checking to make sure sequence of actions is followed. One case is making sure, freed memory is not used.

- Made use of latent specifications to prune for applicable function pairs
- Hierarchical ranking for reducing the number of false positives.

Takeaways

- Hundreds of errors discovered in real systems, resulting in kernel patches !!
- Some unexpected, serious bugs discovered too!!
- Fairly higher number of false positives reported

Conclusion

- Automatic inference of bugs without system knowledge
- Presents two checker frameworks that implement this
- Easily re-targetable to new systems and fixed overheads*
- Future works on complete automation using machine learning approaches

A very interesting work, with promising performance and future directions, which could address the issues of existing need for added manual inspection and analysis overheads.

Questions and Discussion

1) In a system that is designed as a checker, as in this paper, how would we model and account for “completeness”, given they can find bugs but can’t guarantee the absence of bugs.
2) What if a “belief” doesn’t fit a template? How common would these cases be and how scalable/ adaptable is this method in such cases - how expensive is it to come up with new templates, or would we have to then come up with other tools in order to analyze such beliefs ?
3) They mention about augmenting static analysis with dynamic monitoring, which seems very promising. What could be some advantages ? Is this something used today ?