
Jing Huang
Background

- **problems**
  - what rules will be used to check bugs?
    - undocumented
    - a ad hoc collection of conventions
    - encoded in code

How to find the rules?

How to use the rules to find bugs?
Background – contd.

- **previous methods**
  - **testing and manual inspection**
    - to depend on human judgment
    - to suffer from exponential number of code paths
  - **type system**
    - To require Invasive, strenuous manual work
    - To require specific languages
  - **Specifications**
    - To suffer from missing features and over-simplification
  - **Dynamic invariant inference**
    - To dynamically monitor program execution
    - To suffer from a limited number of code paths
    - Noise is less of concern, all value can be detect
goal and process

- To extract beliefs directly from code
- To check for violated beliefs
- To suppress noise in checking results
- To find bugs based on checking
mythology –contd.

- General internal consistency
  - MUST belief
    - directly observation
      To change state and observe it
  - pre- and post-conditions
    To be based on the pre- and post-condition of actions in code (non-zero)
mythology –contd.

- General internal consistency
  - The definition of consistency checker
    - 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.
Example for internal consistency (null pointer)

```c
1: if (card == NULL) {
2: printk(KERN_ERR "capidrv-%d: .. 7,%d!\n",
3: card->contrnr, id) ;
4: }
```

- The rule template T.
  "do not dereference null pointer <p>,"

- The valid slot instances for T.
  pointer <card> associated with a belief set {null, notnull, empty}
Example for internal consistency (null pointer)

- The code actions that imply beliefs.
  - Compare (line 1)
    - nothing directly impacts
  - Deference (line 3)
    - to signal error
    - to add {not null} into the belief set

- The rules for how beliefs combine, including the rules for contradictions.

- The rules for belief propagation.
  - Compare (line 1)
    - to propagate belief in true branch and false branch
General statistical analysis

- Analysis object
  MAY belief

- Analysis goal
  to promote MAY belief to MUST belief

- The definition of consistency checker
  - To check all potential slot instance combinations and then assume that they are MUST beliefs.
  - To indicates how often a specific slot instance combination was checked and how often it failed the check (errors).
  - To use the count information above to rank the errors from most to least plausible.
mythology –contd.

- General statistical analysis
  - statistical analysis method
    - To filter out coincidences from MAY beliefs by observing typical behaviors
  - Z-statistics

\[
x(n, e) = \frac{e/n - p_0}{\sqrt{p_0 \times (1 - p_0)/n}}
\]

- \(n\): the number of checks (the population size)
- \(e\): errors (the number of counter examples)
- \(P_0\): the probability of the examples (\(n-e\))
- \(1-p_0\): the probability of the counter-examples
General statistical analysis

- To suppress noise
  - To use z-statistic to rank error from most to least credible

- To use latent specifications to filter result and determine where and what to check
  - a special function call
  - a set of data types
  - specific naming conventions
Example for statistical analysis (lock inference)

```c
1: lock l;       // Lock
2: int a, b;     // Variables potentially
                 // protected by l
3: void foo() {
4:     lock(l);   // Enter critical section
5:     a = a + b; // MAY: a,b protected by l
6:     unlock(l); // Exit critical section
7:     b = b + 1; // MUST: b not protected by l
8: }
9: void bar() {
10:    lock(l);
11:    a = a + 1; // MAY: a protected by l
12:    unlock(l);
13: }
14: void baz() {
15:    a = a + 1; // MAY: a protected by l
16:    unlock(l);
17:    b = b - 1; // MUST: b not protected by l
18:    a = a / 5; // MUST: a not protected by l
19: }
```
Example for statistical analysis (lock inference)

- The rule template T. variable a must be protected by lock 1?
- To use internal consistency and record how often the belief satisfied its rule versus gave an error.
- To use z-statistic to analyze these counts and rank errors from most to least credible.
- To define a threshold, z-value is higher than it, we regard it as MUST belief, otherwise, we give up the template.
case study

- **Internal consistency**

<table>
<thead>
<tr>
<th>Template</th>
<th>Action</th>
<th>Belief</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Is &lt;P&gt; a null pointer?&quot;</td>
<td>*p</td>
<td>Is not null.</td>
</tr>
<tr>
<td>Section 6</td>
<td>*p == null?</td>
<td>null on true, not-null on false.</td>
</tr>
<tr>
<td>&quot;Is &lt;P&gt; a dangerous user pointer?&quot;</td>
<td>p passed to copyout or copyin</td>
<td>Is a dangerous user pointer.</td>
</tr>
<tr>
<td>Section 7</td>
<td>*p</td>
<td>Is a safe system pointer.</td>
</tr>
<tr>
<td>&quot;Must IS.ERR be used to check &lt;F&gt;'s returned result?&quot;</td>
<td>Checked with IS.ERR</td>
<td>Must always use IS.ERR.</td>
</tr>
<tr>
<td>Section 8.3</td>
<td>Not checked with IS.ERR</td>
<td>Must never use IS.ERR.</td>
</tr>
</tbody>
</table>

- **Danger user pointer**

```c
/* net/atm/mpoa_proc.c */
ssize_t proc_mpc_write(struct file *file,
                      const char *buff) {
    page = (char *)_get_free_page(GFP_KERNEL);
    if (page == NULL) return -ENOMEM;
    /* Copy user data from buff into page */
    retval = copy_from_user(page, buff, ...);
    if (retval != 0)
        ...
    /* Should pass page instead of buff! */
    retval = parse_qos(buff, incoming);
}
int parse_qos(const char *buff, int len) {
    /*Unchecked use of buff */
    strncpy(cmd, buff, 3);
```
case study

- Statistical analysis

<table>
<thead>
<tr>
<th>Template (T)</th>
<th>Examples (E)</th>
<th>Population (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Does lock &lt;L&gt; protect &lt;V&gt;?&quot;</td>
<td>Uses of v protected by l</td>
<td>Uses of v</td>
</tr>
<tr>
<td>&quot;Must &lt;A&gt; be paired with &lt;B&gt;?&quot;</td>
<td>paths with a and b paired</td>
<td>paths with a</td>
</tr>
<tr>
<td>&quot;Can routine &lt;F&gt; fail?&quot;</td>
<td>Result of f checked before use</td>
<td>Result of f used</td>
</tr>
<tr>
<td>&quot;Does security check &lt;Y&gt; protect &lt;X&gt;?&quot;</td>
<td>y checked before x</td>
<td>x</td>
</tr>
<tr>
<td>&quot;Does &lt;A&gt; reverse &lt;B&gt;?&quot;</td>
<td>Error paths with a and b paired</td>
<td>Error paths with a</td>
</tr>
<tr>
<td>&quot;Must &lt;A&gt; be called with interrupts disabled?&quot;</td>
<td>a called with interrupts disabled</td>
<td>a called</td>
</tr>
</tbody>
</table>
case study

- Statistical analysis
  - Failure/IS_ERR (function <f> must be checked for failure)

```c
/* ipc/shm.c: map_zero_setup */
if (IS_ERR(shp = seg_alloc(...)))
    return PTR_ERR(shp);

/* 2.4.0-test9: ipc/shm.c: newseg */
NOTE: checking 'seg_alloc' */
if (! (shp = seg_alloc(...)))
    return -ENOMEM;
    id = shm_addid(shp);

int ipc_addid(..., struct kern_ipc_perm* new)
    new->cuid = new->uid = current->euid;
    new->gid = new->cgid = current->egid;
    ids->entries[id].p = new;
```
case study

- Statistical analysis
  - no `<a>` after `<b>` (freed memory cannot be used)
  
  ```c
  cut->data = k m a l l o c ( . . . ) ;    \rightarrow \text{if allocating is failed}
  if (!ent->data)
    kfree (ent) ;
  goto out ;
  }
  out :
  return ent ;
  ```
Conclusion

- To automatically extract programmer beliefs from the source code, and we flag belief contradictions as errors by using statistical analysis and internal consistency.
- To automatically find bugs in a system without having a prior knowledge.
- To drastically decreases the manual labor required to re-target our analyses to a new system.
- To enable us to check rules that we had formerly found impractical.
like and dislike

- Like
  - To use simple techniques to find bugs
  - Based on z-statistic, to rank MAY beliefs
  - To find more types of bugs than before
  - To provide a lot of clear analysis for detailed cases

- Dislike
  - Too many terms and too abstract description in analysis and these terms’ definition is scattered in different parts of the paper
Thank you!

Questions?