Oracle-free Testing with Two-layer Covering Arrays

Rick Kuhn
National Institute of Standards and Technology
Gaithersburg, MD

East Carolina University
NSF Research Experiences for Undergraduates
June 29, 2015
Some current approaches

• Fuzz testing
  – crash system w/ random values, then analyze memory dump
  – Good for major faults that cause crashes

• Metamorphic testing –
  – e.g. $\cos(x) = \cos(x+360)$, so compare outputs for both, with a difference indicating an error
  – Good for numerical software

• Partial test oracle –
  – e.g., insert element $x$ in data structure $S$ check $x$ in $S$ after
  – Usually not fully automatable
New method

- Consider equivalence classes
- Example: shipping cost based on distance $d$ and weight $w$, with packages < 1 pound are in one class, 1..10 pounds in another, 10 in a third class.
- Then for cost function $f(d,w)$, $f(d, 0.2) = f(d, 0.9)$, for equal values of $d$.
- But $f(d, 0.2) \neq f(d, 5.0)$, because two different weight classes are involved.
Basic property of equivalence classes

when $a_1$ and $a_2$ are in the same equivalence class,

$$f(a_1, b, c, d, \ldots) \approx f(a_2, b, c, d, \ldots),$$

where $\approx$ is equivalence with respect to some predicate.

If not,
- then either the code is wrong,
- or equivalence classes are not defined correctly.
Can we use this property for testing?

- Let’s do an example: access control. Access is allowed if (1) subject is employee and time is in working hours and it’s a weekday; or (2) subject is an employee with administrative privileges; or (3) subject is an auditor and it is a weekday.

- Equivalence classes for time of day and day of the week
  - time = minutes past midnight (0..0539), (0540..1020), (1021..1439).
  - Days of the week weekend and weekdays, designated as (1,7) and (2..6) respectively.
int access_chk() {
    if (emp && t >= START && t <= END &&
        d >= MON && d <= FRI) return 1;
    else
        if (emp && p) return 2;
    else
        if (aud && d >= MON && d <= FRI) return 3;
    else
        return 0;
}
Establish equivalence classes

- emp: boolean
- day: (1,7), (2,6)
  - A1 A2
- time:
  - emp (bool): 0,1
  - day (enum): A1,A2
  - time (enum): B1,B2,B3
- priv: boolean
- aud: boolean
  - priv (bool): 0,1
  - aud (bool): 0,1
All of these should be equal

Eq. class B1

\[ f(0, \begin{array}{c} 1 \\ 7 \\ 539 \end{array}, 0, 0) \]

Eq. class A1

\[ f(0, \begin{array}{c} 1 \\ 539 \end{array}, 0, 0) \]
These should also be equal

Eq. class B1  →  Eq. class A2

Now we’re using class A2

\[
f(0, \begin{pmatrix} 2 \\ 6 \\ 539 \end{pmatrix}, 0, 0) = f(0, \begin{pmatrix} 2 \\ 6 \\ 539 \end{pmatrix}, 0, 0)
\]

\[
f(0, \begin{pmatrix} 2 \\ 6 \\ 100 \\ 539 \end{pmatrix}, 0, 0) = f(0, \begin{pmatrix} 2 \\ 6 \\ 100 \\ 539 \end{pmatrix}, 0, 0)
\]

\[
f(0, \begin{pmatrix} 2 \\ 6 \\ 100 \\ 539 \end{pmatrix}, 0, 0) = f(0, \begin{pmatrix} 2 \\ 6 \\ 100 \\ 539 \end{pmatrix}, 0, 0)
\]

\[
f(0, \begin{pmatrix} 2 \\ 6 \\ 100 \\ 539 \end{pmatrix}, 0, 0) = f(0, \begin{pmatrix} 2 \\ 6 \\ 100 \\ 539 \end{pmatrix}, 0, 0)
\]
Run the tests

- Correct code output:
  3333
  0000
  0000
  1111
  0000
  2222

Faulty code:
if (emp && t>=START && t==END && d>=MON && d<=FRI) return 1;

Faulty code output:
  3333
  0000
  0000
  3311
  0000
  2222
What’s happening here?

We simply detect inconsistency between partitions.
Can this really work on practical code?

Experiment: TCAS code (same used in earlier model checking tests)

- Small C module, 12 variables
- Seeded faults in 41 variants

Results:

<table>
<thead>
<tr>
<th>Primary x secondary</th>
<th>#tests</th>
<th>total</th>
<th>faults detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-way x 3-way</td>
<td>285x8</td>
<td>2280</td>
<td>6</td>
</tr>
<tr>
<td>4-way x 3-way</td>
<td>970x8</td>
<td>7760</td>
<td>22</td>
</tr>
</tbody>
</table>

- More than half of faults detected
- Large number of tests -> but fully automated, no human intervention
- We envision this type of checking as part of the build process; can be used in parallel with static analysis, type checking
Prototype tool has been developed.
Next Steps

• Realistic trial use
• Different constructions for secondary array, e.g., random values
• Formal analysis of applicability – range of applicability/effectiveness, limitations, special cases
• Determine how many faults can be detected this way
• Develop tools to incorporate into build process