Client-Specific Equivalence Checking

Federico Mora (University of Toronto)
Yi Li (Nanyang Technological University)
Julia Rubin (University of British Columbia)
Marsha Chechik (University of Toronto)
Motivation

Application Code

Library

Library

Library

Hard to verify

Evolving at different speeds
Motivation

Client-Specific Equivalence: What happens to the client when a library gets upgraded?

Updates can invalidate verification
Example

```c
double mpf_get_d_2exp(signed long int *exp, src) {  
  mp_size_t size, abs_size;  
  mp_srcptr ptr;  
  int cnt;  
  double d;  
  size = SIZ(src);  
  if (UNLIKELY(size == 0))  
    {  
      *exp = 0;  
      return 0.0;  
    }  
  ptr = PTR(src);  
  abs_size = ABS(size);  
  count_leading_zeros(cnt, ptr[abs_size - 1]);  
  *exp = EXP(src) * GMP_NUMB_BITS - cnt;  
  return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));}
```

Library with changes

- New version always returns positive number

```c
REAL log_real(REAL x) {  
  double d;  
  double ln_app;  
  signed long int exp;  
  d = mpf_get_d_2exp(&exp, x.get_mpf_t());  
  ln_app = (double)exp * log(2.0) + log(d);  
  return ln_app;}
```

Client that is affected by change

Log of negative number undefined

Client that is NOT affected by change

Takes absolute value of output

Library with changes

- New version always returns positive number

```c
double F_mpz_poly_eval_horner_d_2exp(  
  long *exp, F_mpz_poly_t poly, double val)  
{  
  ... res = mpf_get_d_2exp(exp, output);  
  // work around bug in earlier versions of GMP/MPIR  
  if ((mpf_sgn(output) < 0) && (res >= 0.0))  
    res = -res;  
  ...  
  }
```
How Often Is a Client Unaffected by a Change?
Applicability Study

Inspected 66 client-library function pairs

- Popular libraries on GitHub (>1,000 stars)
- Written in C and Python
- Went through 100 most recent commits which do not alter signatures
  - mostly bug fixes and
  - new behaviour introductions
- Searched for unique clients on GitHub
# Applicability Study Results

~71% of the clients are unaffected

<table>
<thead>
<tr>
<th>Projects</th>
<th>Library Functions</th>
<th># Client</th>
<th>#Affected</th>
<th>#Unaffected</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>RN_is_prime_fasttest_ex</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>RSA_check_key</td>
<td>32</td>
<td>5</td>
<td>27</td>
</tr>
<tr>
<td>Linux</td>
<td>gcd</td>
<td>11</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>GMP</td>
<td>mpf_get_d_2exp</td>
<td>7</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Delorean</td>
<td>Delorean</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Delorean</td>
<td>Delorean2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
What’s Wrong With Existing Solutions?
Preliminaries

We consider partial functional equivalence

- Loops and recursion unrolled to configurable depth, $d$
- Two unrolled programs $P$, $P'$ are equal iff for all $x$, $P(x) = P'(x)$

For this presentation we represent

- Programs as triangles
  - Single entry point at top
- Libraries calls are triangles inside a larger triangle
- Program paths are lines inside the triangles
- Updates are purple squares inside libraries
int main(int x) {
    if (x>=18 && x<22) {
        return foo(x, 20);
    }
    return 0;
}

int foo(int a, int b) {
    int c=0;
    for (int i=1; i<=b; ++i) {
        c+=a;
    }
    for (int i=1; i<=a; ++i) {
        c+=b;
    }
    return c;
}

[Trostanetski et al, 17]
Different Ways to Apply Existing Solutions
Different Ways to Apply Existing Solutions

1. Checking Equivalence of Libraries

Exploring all of the library, and none of the client

Too Strong!

Library = Library*

Classic equivalence checking problem [Person, 08]
Different Ways to Apply Existing Solutions

2. Checking Equivalence Of Libraries Under a Condition

But what condition?

Conditional equivalence checking problem
[Kawaguchi et al, 10]
[Lahiri et al, 13]
Different Ways to Apply Existing Solutions

2. Checking Equivalence Of Libraries Under a Condition

Let $C$ be the set of all the calling contexts of the library in the client.

For all $c$ in $C$, $c[lib] = c[lib^*]$?

But too strong again!
- $lib(x) = -lib'(x)$
- client := $lib(1) + lib(-1)$

Exploring part of client that calls library, and part of library used by client.

Client

Library

Library*
Different Ways to Apply Existing Solutions

3. Checking Equivalence of Client-Library Pairs

Right strength, but ignores the fact that the client remains unchanged

A special case of regression verification
[Godlin & Strichman, 08]
[Felsing et al, 14]
[Trostanetski et al, 17]

Classic equivalence checking problem
[Person, 08]
Different Ways to Apply Existing Solutions

3. Checking Equivalence of Client-Library Pairs

Ignores the fact that the client remains unchanged

A special case of regression verification

Can we do better?
Our Technique In A Nutshell
Client-Specific EquiValence CheckER

**Insight:** existing techniques are too strong, or consider too much. To get the most precise and efficient analysis let’s consider only

- how the client uses the library and
- where the library change is active.
Algorithm

- Explore Client with library uninterpreted
  - Collect uses/contexts of the library
- For each client context
  - Explore the library restricted to this context
  - If change is inactive, discard
  - Else, check for quick counterexample
    - If counterexample found, return
    - Else store paths
- Create equivalence assertion from stored paths
- Dispatch to existing verifier, or SMT solver
Example Savings

```c
int main(int x) {
    if (x >= 18 && x < 22)
        return foo(x, 20);
    return 0;
}

int foo(int a, int b) {
    int c = 0;
    for (int i = 1; i <= b; ++i)
        c += a;
    for (int i = 1; i <= a; ++i)
        c += b;
    return c;
}
```

[Trostanetski et al, 17]

How the client uses the library

Saves us from computing a non-linear loop invariant: $c == a \times b == a' \times b'$

How the client uses the library

Saves us from computing a non-linear loop invariant: $c == a \times b == a' \times b'$
Evaluation
Implementation & Evaluation

Available at: https://github.com/Client-Specific-Equivalence-Checker/CLEVER

Explores client contexts using symbolic execution
  ● PyExSMT (https://github.com/FedericoAureliano/PyExSMT)

We compare with SymDiff, RVT, and ModDiff (treating client-lib pair as a whole).

Subjects:
  ● 39 client-library pairs with library updates (23 equivalent / 16 inequivalent)
  ● 23 come from the ModDiff suite (small programs)
  ● 16 come from our pre-study
Cactus Plot: Equivalent Cases

Benefit from pruning to client and change relevant paths
Cactus Plot (Log Scale): Non-Equivalent Cases

Benefit from early counterexample detection
Conclusions & Beyond
Summary

We consider a special case of equivalence where usage patterns can be exploited

- We show that this special case is relevant
- Devise an extension/enhancement to classic regression verification
  - Optimized for early discovery of counterexamples
- It does well when compared against the state-of-the-art

Lots of details are not considered, yet

- Go beyond functional equivalence
  - Total path equivalence: maintaining all intermediate executions of the client etc.
- Improvements on usability
  - Explain reasons for equivalence
  - Suggest changes/uploads to clients

Benchmark size is still quite limited

- Call backs, side effects, heap, etc.
- Increase support for primitive types
  - E.g. floating-point numbers, strings, and algebraic datatypes
Thank You!

CLEVER available at https://github.com/Client-Specific-Equivalence-Checker/CLEVER

Benchmarks and more available at https://client-specific-equivalence-checker.github.io/

PyExSMT available at https://github.com/FedericoAureliano/PyExSMT