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

Software
- is composite,

uses
Motivation

Software
- is composite,
- is hard to verify, and
Motivation

Software
- is composite,
- is hard to verify, and
- it’s pieces evolve over time at different speeds
Motivation

Software
- is composite,
- is hard to verify, and
- it’s pieces evolve over time at different speeds

Can we efficiently detect when a client is affected by a library update?
Example

double mpf_get_d_2exp(signed long int *expptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
    double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0)) {
        *expptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *expptr = EXP(src) * GMP_NUMB_BITS - cnt;
    - return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + d = mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + return size >= 0 ? d : -d;}

```c
double mpf_get_d_2exp(signed long int *expptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
    double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0))
    {
        *expptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *expptr = EXP(src) * GMP_NUMB_BITS - cnt;

    return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + d = mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + return size >= 0 ? d : -d;
}
```
double mpf_get_d_2exp(signed long int *expptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
+double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0))
    {
        *expptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *expptr = EXP(src) * GMP_NUMB_BITS - cnt;

    return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));

    +d = mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    +return size >= 0 ? d : -d;}

Library with changes

New version always returns positive number
double mpf_get_d_2exp(signed long int *expptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
    double d;
    size = SIZ(src);
    if (UNLIKELY(size == 0)) {
        *expptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *expptr = EXP(src) * GMP_NUMB_BITS - cnt;
    return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
}

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;
}
Example

```c
double mpf_get_d_2exp(signed long int *exp_ptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
+ double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0)) {
        *exp_ptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *exp_ptr = EXP(src) * GMP_NUMB_BITS - cnt;

    return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + d = mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + return size >= 0 ? d : -d;
}
```

```
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;
}
```
Example

```c
double mpf_get_d_2exp(signed long int *exp.ptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
    + double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0))
    {
        *exp.ptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *exp.ptr = EXP(src) * GMP_NUMB_BITS - cnt;
    + return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + d = mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + return size >= 0 ? d : -d;
}

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;
}
```

Library with changes

New version always returns positive number

Client that is affected by change

Log of negative number undefined
Example

```
double mpf_get_d_2exp(signed long int *exp_ptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
    double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0)) {
        *exp_ptr = 0;
        return 0.0;
    }

    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *exp_ptr = EXP(src) * GMP_NUMB_BITS - cnt;

    return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
}
```

```
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;
}
```

```
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;
    ...}
```
double mpf_get_d_2exp(signed long int *expptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
+
    double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0)) {
        *expptr = 0;
        return 0.0;
    }

    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *expptr = EXP(src) * GMP_NUMB_BITS - cnt;

    return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
}

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;
}

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;
    ...
}
double mpf_get_d_2exp(signed long int *expptr, mpf_srcptr src) {
    mp_size_t size, abs_size;
    mp_srcptr ptr;
    int cnt;
    +double d;

    size = SIZ(src);
    if (UNLIKELY(size == 0)) {
        *expptr = 0;
        return 0.0;
    }
    ptr = PTR(src);
    abs_size = ABS(size);
    count_leading_zeros(cnt, ptr[abs_size - 1]);
    cnt -= GMP_NAIL_BITS;
    *expptr = EXP(src) * GMP_NUMB_BITS - cnt;
    - return mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + d = mpn_get_d(ptr, abs_size, 0, -(abs_size * GMP_NUMB_BITS - cnt));
    + return size >= 0 ? d / d;}

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;
}

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;
    ...
}
Can we efficiently detect when a client is affected by a library update?

First Attempt: Apply Existing Techniques.
Preliminaries

We consider partial functional equivalence

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

We consider partial functional equivalence

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

We consider partial functional equivalence

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

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

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

```
Client
  ↓
Library

Client
  ↓
Library*
```
1. Checking Equivalence of Client-Library Pairs

Different Ways to Apply Existing Solutions

Client

Library

? =

Client

Library*

Right strength, but...
Different Ways to Apply Existing Solutions

1. Checking Equivalence of Client-Library Pairs

Ignores the fact that the client remains unchanged
Different Ways to Apply Existing Solutions

2. Checking Equivalence of Libraries

- Client
- Library
- Library*
- ?
Different Ways to Apply Existing Solutions

2. Checking Equivalence of Libraries

Does not consider how the client uses the library

Client

Library

Client

Library*

= ?
Different Ways to Apply Existing Solutions

2. Checking Equivalence of Libraries

- Library
- Client
- Library*

Does not consider how the client uses the library

Too Strong!
Different Ways to Apply Existing Solutions

3. Checking Equivalence Of Libraries Under a Condition
Different Ways to Apply Existing Solutions

3. Checking Equivalence Of Libraries Under a Condition

But what condition?

Client

Library

Library*

Condition
Different Ways to Apply Existing Solutions

3. 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*]?

Client

Library

Library*

36
Different Ways to Apply Existing Solutions

3. 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 $c \in C$, $c[\text{lib}] = c[\text{lib}^*]$?

- Exploring part of client that calls library, and part of library used by client.
- Library
- Library*
Let $C$ be the set of all the calling contexts of the library in the client. For all $c$ in $C$, $c[lib] = c[lib^*]$?

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

3. Checking Equivalence of Libraries Under a Condition

- $lib(x) := x$
- $lib'(x) := -x$
- $client := lib(1) + lib(-1)$

But too strong again!
Can we efficiently detect when a client is affected by a library update?

Second Attempt: **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
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

Further, let’s target patterns observed “in the wild”
- Identify when a client doesn’t use the library change
- When it does, look for quick counterexample
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

Further, let’s target patterns observed “in the wild”
● Identify when a client doesn’t use the library change
● When it does, look for quick counterexample
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

Further, let’s target patterns observed “in the wild”
● Identify when a client doesn’t use the library change
● When it does, look for quick counterexample
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

Further, let’s target patterns observed “in the wild”
- Identify when a client doesn’t use the library change
- When it does, look for quick counterexample
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

Further, let’s target patterns observed “in the wild”
● Identify when a client doesn’t use the library change
● When it does, look for quick counterexample

Consider exactly what we need, at strength that we need
CLEVER In Detail

**Algorithm**
- Explore Client with the library uninterpreted
  - Collect usescontexts of the library
CLEVER In Detail

**Algorithm**
- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
Algorithm

- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
CLEVER In Detail

**Algorithm**
- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
CLEVER In Detail

Algorithm
- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
- For each client context
  - Explore the libraries restricted to this context
  - If change is inactive, discard
  - Else, check for quick counterexample
    - If counterexample found, return
    - Else store paths
**CLEVER In Detail**

**Algorithm**
- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
- For each client context
  - Explore the libraries restricted to this context
  - If change is inactive, discard
  - Else, check for quick counterexample
    - If counterexample found, return
    - Else store paths
Algorithm

- Explore Client with the library uninterpreted
  - Collect uses CONTEXTS of the library
- For each client context
  - Explore the libraries restricted to this context
  - If change is inactive, discard
  - Else, check for quick counterexample
    - If counterexample found, return
    - Else store paths
CLEVER In Detail

Algorithm
- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
- For each client context
  - Explore the libraries restricted to this context
  - If change is inactive, discard
  - Else, check for quick counterexample
    - If counterexample found, return
    - Else store paths
CLEVER In Detail

**Algorithm**
- Explore Client with the library uninterpreted
  - Collect uses/contexts of the library
- For each client context
  - Explore the libraries 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]
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]
```
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]
```

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

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

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

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
Cactus Plot: Equivalent Cases

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

Benefit from early counterexample detection
Conclusions & Beyond
Summary

- We identified client-specific equivalence checking
- Produced a technique and tool for checking it
  - Insight: existing techniques are too strong, or consider too much.
  - We consider only how the client uses the library and where the library change is active
  - We target patterns observed “in the wild”
- Evaluated our tool against the state-of-the-art
  - It does well!
Future Work

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/updates 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
Frequently Asked Questions
How Often Is a Client Unaffected by a Change/Does this problem exist in real life?
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

<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>
<tr>
<td>All</td>
<td>All</td>
<td>66</td>
<td>19</td>
<td>47</td>
</tr>
</tbody>
</table>
Applicability Study Results

<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>
<tr>
<td>All</td>
<td>All</td>
<td>66</td>
<td>19</td>
<td>47</td>
</tr>
</tbody>
</table>

- ~71% of the clients are unaffected!
How does CLEVER scale with increasing number of paths?

(Since our approach is based on path exploration)
CLEVER Vs. ModDiff by Number of Paths

Experiment: Take ModDiff equivalent benchmarks, keep structure, but increase number of paths

Non-Equivalent Cases even more stark.