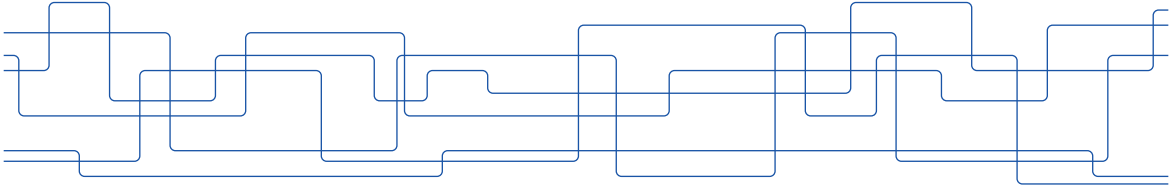
Evaluation of Methods for Effective Control Flow Recovery

Robin Eklind



Disposition

1. What? *Control Flow Recovery*
2. Why? *Applications of Control Flow Recovery*
3. How? *Control Flow Recovery Methods*
4. Technical Contributions
5. Future Work
6. Demo!

What?

Control Flow Recovery

Analysis of control flow graphs (CFGs) to recover high-level control flow primitives (e.g. `if`-statements and `for`-loops) from assembly or low-level intermediate representations.

Control Flow Recovery

Listing (1) LLVM IR assembly.

```
define i32 @f(i32 %n) {
entry:
  br label %loop_cond
loop_cond:
  %sum = phi i32 [ 0, %entry ], [ %sum.2, %loop_post ]
  %i = phi i32 [ 0, %entry ], [ %i.1, %loop_post ]
  %cond1 = icmp slt i32 %i, %n
  br i1 %cond1, label %loop_body, label %exit
loop_body:
  %cond2 = icmp slt i32 %sum, 100
  br i1 %cond2, label %if_body, label %if_follow
if_body:
  %sum.1 = add i32 %sum, %i
  br label %if_follow
if_follow:
  %sum.2 = phi i32 [ %sum.1, %if_body ], [ %sum, %loop_body ]
  br label %loop_post
loop_post:
  %i.1 = add i32 %i, 1
  br label %loop_cond
exit:
  ret i32 %sum
}
```

Listing (2) C source file.

```
int f(int n) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    if (sum < 100) {
      sum += i;
    }
  }
  return sum;
}
```

Figure: *Reverse compilation*, going from low-level (left) to high-level (right).

Why?

Applications of Control Flow Recovery

- Malware analysis
- Security assessments
  - Automated vulnerability scanning
- Control-flow aware compiler passes
- Enable high-level data-flow analysis on IR (Rosen's method)
- Verification of compiler output (*Reflections on Trusting Trust*)
- Reverse compilation
  - Transpilation between programming languages ($n + m$)
  - Migrate proprietary software from legacy architectures
  - Re-optimization of software where source code or tool chain is missing

Control-flow Aware Compiler Passes

Google Git

go / go / **9e21e9c5cb27e5f2b5acba14efb6bb6f126595cc**

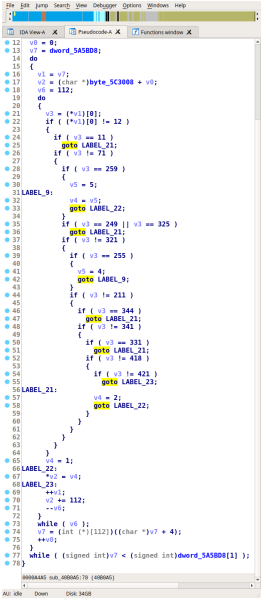
commit	9e21e9c5cb27e5f2b5acba14efb6bb6f126595cc	[log] [tgz]
author	David Chase <drchase@google.com>	Fri Apr 28 16:48:11 2017 -0400
committer	David Chase <drchase@google.com>	Thu Oct 05 18:49:10 2017 +0000
tree	51167527a921bf25fe00abf5c2326bc0aa4b01b6	
parent	acdb44765d86a5fd66cbbe24735f7dde658a295f [diff]	

```
cmd/compile: make loop finder more aware of irreducible loops

The loop finder doesn't return good information if it
encounters an irreducible loop.  Make a start on improving
this, and set a function-level flag to indicate when there
is such a loop (and the returned information might be flaky).

Use that flag to prevent the loop rotater from getting
confused; the existing code seems to depend on artifacts
of the previous loop-finding algorithm. (There is one
irreducible loop in the go library, in "inflate.go").
```

Issues with State-of-the-Art Reverse Compilation Tools



Listing (3) Corresponding Go source code.

```
func f_40B0A5() {
   for i := 0; i < 112; i++ {
      for j := 0; j < 112; j++ {
         switch g_5A5BD8[i][j] {
         case 12, 71, 211, 321, 341, 418:
            g_5C3008[i][j] = 1
         case 3, 11, 33, 249, 344, 421:
            g_5C3008[i][j] = 2
         case 255:
            g_5C3008[i][j] = 4
         case 259:
            g_5C3008[i][j] = 5
         }
      }
   }
}
```

Figure: IDA output (left) and corresponding Go source code (right).

How?

Control Flow Recovery Methods

- Hammock method
- Interval method
- Pattern-independent method

There are benefits and drawbacks with each method.

Evaluation metric

- **False positive**: control flow primitive recovered but *not* present in original source code.
- **False negative**: control flow primitive *not* recovered but present in original source code.

Hammock method

Model high-level control flow primitives as subgraphs and use *subgraph isomorphism search* to locate the corresponding subgraphs in CFGs.

Pros

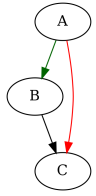
- *no* false positives
- reconstructed control flow semantically equivalent

Cons

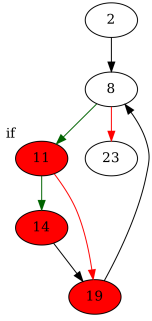
- *many* false negatives
- requires single-entry/single-exit invariant for subgraphs.

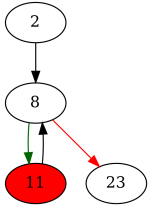
Hammock method

```
if (A) {
  B
}
C
```



(a) Canonical 1-way conditional.





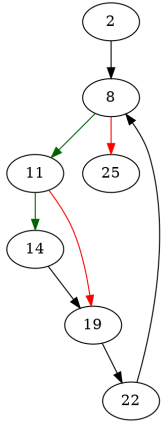
(b) Subgraph isomorphism of canonical 1-way conditional located in control flow graph (left) and its nodes merged (right).

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



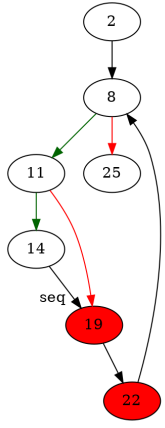
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



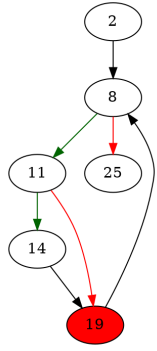
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



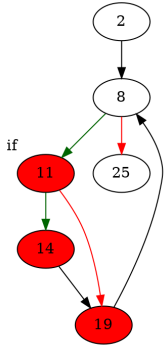
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



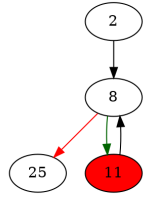
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



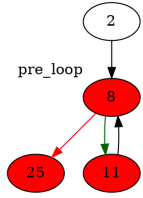
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



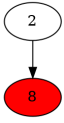
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



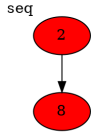
(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.



(b) Control flow graph.

Hammock method - example

Example with nested control flow primitives.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8      }
9      x += 30;
10   }
11   return x;
12 }
```

(a) Original C source code.

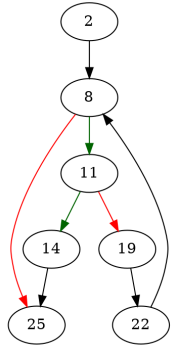
(b) Control flow graph; analysis **complete**.

Hammock method - counter-example 1

Counter-example with *multi-exit loop*.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8        break;
9      }
10     x += 30;
11   }
12   return x;
13 }
```

(a) Original C source code.



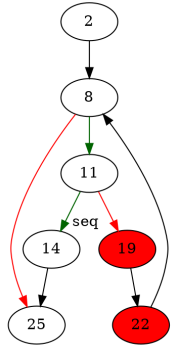
(b) Control flow graph.

Hammock method - counter-example 1

Counter-example with *multi-exit loop*.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8        break;
9      }
10     x += 30;
11   }
12   return x;
13 }
```

(a) Original C source code.



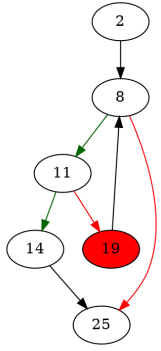
(b) Control flow graph.

Hammock method - counter-example 1

Counter-example with *multi-exit loop*.

```
1  int main(int argc, char **argv) {
2    int i, x;
3
4    x = 0;
5    for (i = 0; i < 10; i++) {
6      if (x >= 100) {
7        x += 3*i;
8        break;
9      }
10     x += 30;
11   }
12   return x;
13 }
```

(a) Original C source code.



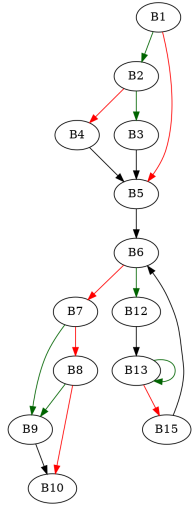
(b) Control flow graph; analysis **incomplete**.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



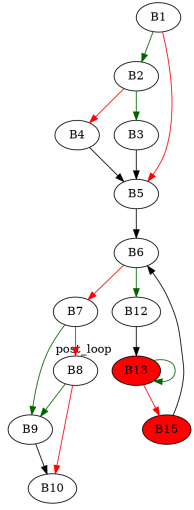
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



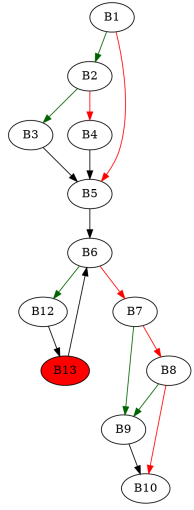
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



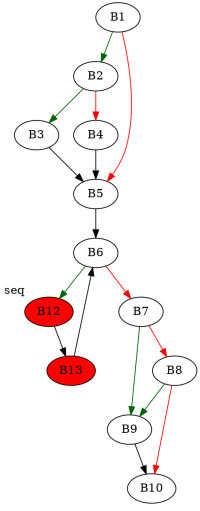
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



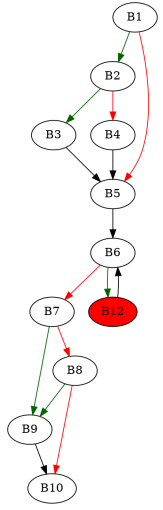
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



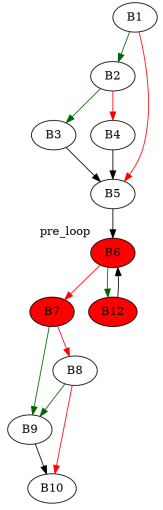
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



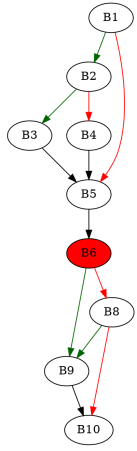
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



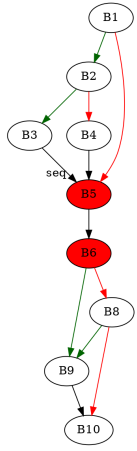
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



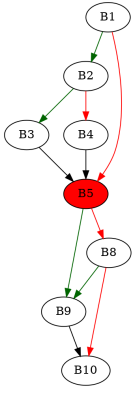
(b) Control flow graph.

Hammock method - counter-example 2

Counter-example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



(b) Control flow graph; analysis **incomplete**.

Interval method

Identify intervals in CFGs to determine the nesting-levels of loops and the follow nodes of high-level control flow primitives.

Pros

- handles multi-level `continue`- and `break`-statements in loops
- handles jump threading optimized control flow graphs
- handles short-circuit evaluation
- reconstructed control flow semantically equivalent

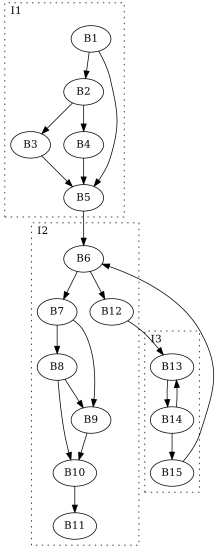
Pros/Cons

- *some* false positives
- *some* false negatives

Interval method

Intervals have interesting properties.

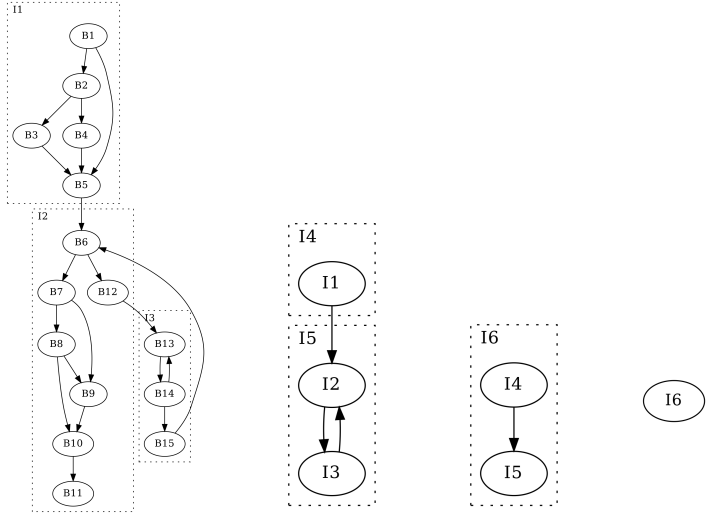
*An **interval** $I(h)$ with header node h is the maximal single-entry subgraph of a CFG in which h is the only entry node and in which all cycles contain h.*



(a) Intervals outlined in control flow graph[1].

[1] Figure adapted from Cristina Cifuentes. "Reverse Compilation Techniques". PhD thesis. Queensland University of Technology, 1994

Interval method



(a) G^1 (b) G^2 (c) G^3 (d) G^4

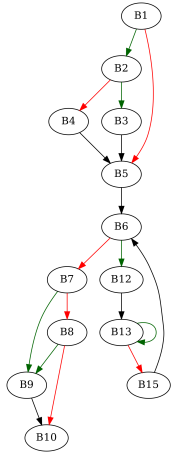
Figure: Derived sequence of graphs, $G^1, ..., G^n$.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



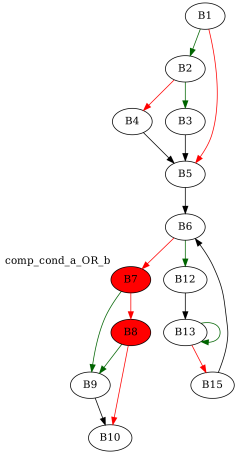
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



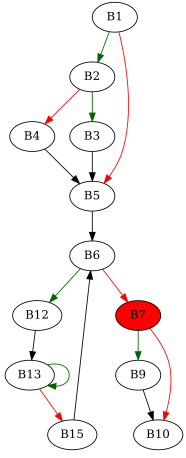
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



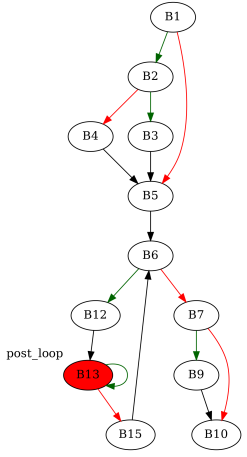
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



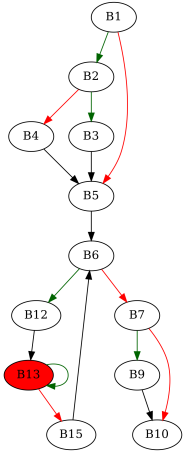
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



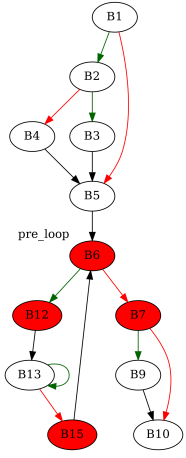
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



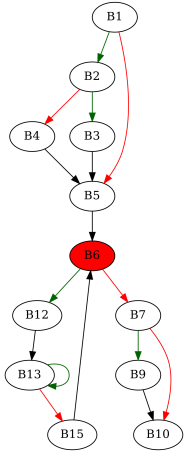
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



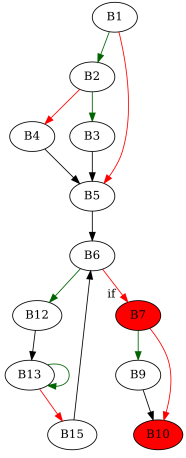
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



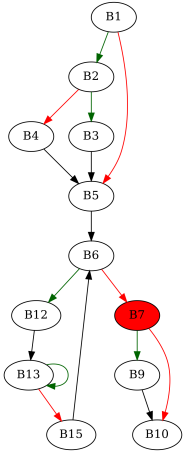
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



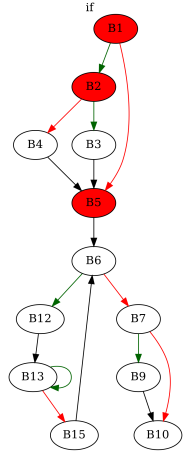
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



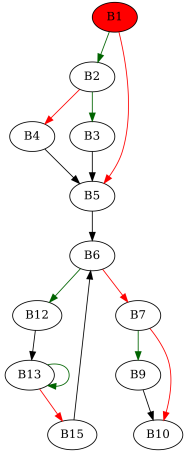
(b) Control flow graph.

Interval method - example

Example with *jump threading* optimization and *short-circuit* evaluation.

```
1  void f(void) {
2    int b1 = B1();
3    if (b1) {
4      int b2 = B2();
5      if (b2) {
6        B3();
7      } else {
8        B4();
9      }
10   }
11   B5();
12   while (B6()) {
13     B12();
14     int b14;
15     do {
16       B13();
17       b14 = B14();
18     } while(b14);
19     B15();
20   }
21   int b7 = B7();
22   if (b7 || B8()) {
23     B9();
24   }
25   B10();
26   B11();
27 }
```

(a) Original C source code.



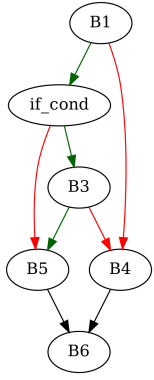
(b) Control flow graph; analysis *almost* **complete**.

Interval method - counter-example

Counter-example with *jump threading* optimization and boolean constraint propagation.

```
1  void f(void) {
2    int b1 = B1();
3    int b2 = B2();
4    if (b1 && b2) {
5      B3();
6    }
7    if (!b1) {
8      B4();
9    }
10   if (b1) {
11     B5();
12   }
13   B6();
14 }
```

(a) Original C source code.



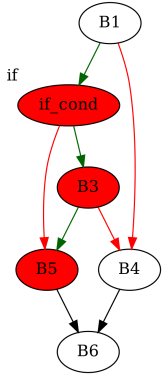
(b) Control flow graph.

Interval method - counter-example

Counter-example with *jump threading* optimization and boolean constraint propagation.

```
1  void f(void) {
2    int b1 = B1();
3    int b2 = B2();
4    if (b1 && b2) {
5      B3();
6    }
7    if (!b1) {
8      B4();
9    }
10   if (b1) {
11     B5();
12   }
13   B6();
14 }
```

(a) Original C source code.



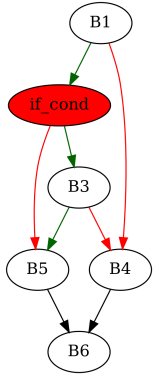
(b) Control flow graph.

Interval method - counter-example

Counter-example with *jump threading* optimization and boolean constraint propagation.

```
1  void f(void) {
2    int b1 = B1();
3    int b2 = B2();
4    if (b1 && b2) {
5      B3();
6    }
7    if (!b1) {
8      B4();
9    }
10   if (b1) {
11     B5();
12   }
13   B6();
14 }
```

(a) Original C source code.



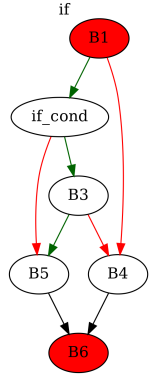
(b) Control flow graph.

Interval method - counter-example

Counter-example with *jump threading* optimization and boolean constraint propagation.

```
1  void f(void) {
2    int b1 = B1();
3    int b2 = B2();
4    if (b1 && b2) {
5      B3();
6    }
7    if (!b1) {
8      B4();
9    }
10   if (b1) {
11     B5();
12   }
13   B6();
14 }
```

(a) Original C source code.



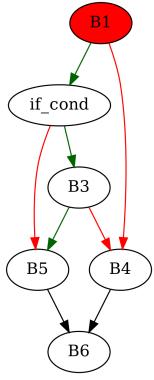
(b) Control flow graph.

Interval method - counter-example

Counter-example with *jump threading* optimization and boolean constraint propagation.

```
1  void f(void) {
2    int b1 = B1();
3    int b2 = B2();
4    if (b1 && b2) {
5      B3();
6    }
7    if (!b1) {
8      B4();
9    }
10   if (b1) {
11     B5();
12   }
13   B6();
14 }
```

(a) Original C source code.



(b) Control flow graph; analysis **incomplete**.

Pattern-independent method

Considers the conditions required to reach a node in the CFG rather than modelling explicit patterns. Relies on semantic-preserving transformations to pre-process CFGs.

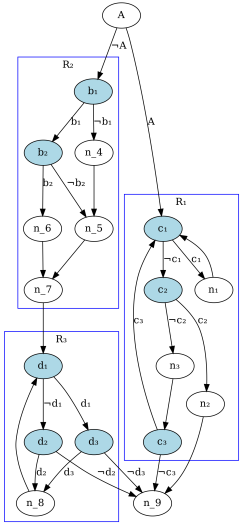
Pros

- *no* false negatives
- `goto`-free output

Cons

- *many* false positives
- introduces auxiliary condition variables (not present in original source)
- reconstructed control flow functionally but *not* semantically equivalent

Pattern-independent method



(a) Single-entry single-successor regions[2].

Listing (4) Recovered control flow primitives.

```
1  if (A) {
2     do {                                  // /Region 1.
3        while (c1) {                       // |
4           n1;                             // |
5        }                                  // |
6        if (c2) {                          // |
7           n2;                             // |
8           break;                          // |
9        }                                  // |
10       n3;                                // |
11    } while (c3);                         // \_
12 } else {                                 // /Region 2.
13    if (!b1) {                            // |
14       n4;                                // |
15    }                                     // |
16    if (b1 && b2) {                       // |
17       n6;                                // |
18    } else {                              // |
19       n5;                                // |
20    }                                     // |
21    n7;                                   // \_
22    while ((d1 && d3) || (!d1 && d2)) {   // /Region 3.
23       n8;                                // |
24    }                                     // \_
25    n9;
26 }
```

[2]Figure and listing adapted from Khaled Yakdan et al. "No More Gotos: Decompilation Using Pattern-Independent Control-Flow Structuring and Semantics-Preserving Transformations". In: 2015

Technical Contributions

- Give insight into how different control flow recovery method operate.
- Highlight the benefits and drawbacks of different control flow recovery methods – so users may select the method best suited for their needs.
- Develop tools to facilitate an understanding of the inner workings of different control flow recovery methods.
- Provide intuition for deficiencies in control flow recovery methods, detailing their failure modes and giving insight to help guide future research.

Future Work

- Explore combining key principles from different control flow recovery methods to leverage their strengths and work around their deficiencies.
- Investigate cross-validation of results produced by different methods to define a notion of *confidence score*.
- Facilitate binary analysis capabilities using control flow analysis results – which enable high-level data flow analysis.
- Integrate control flow recovery information when lifting binary executables to intermediate representations and high-level languages.

Demo

Time for a demo!