Compiler Optimization with Retrofitting Transformations: Is there a Semantic Mismatch?

Santosh Nagarakatte

Joint work with my PhD student Jay Lim and collaborator Vinod Ganapathy
Department of Computer Science,
Rutgers University

RUTGERS

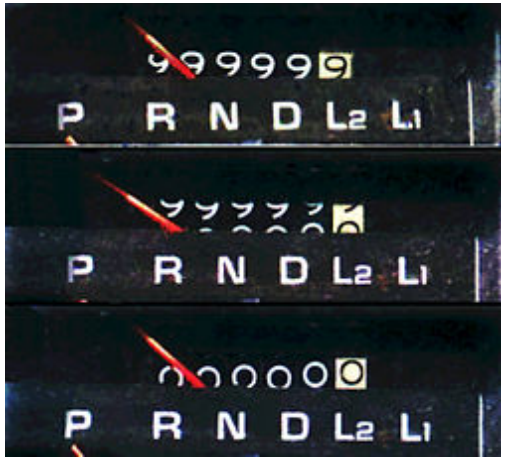
Retrofitting Transformation

Source

Property of interest

Retrofitting Transformation

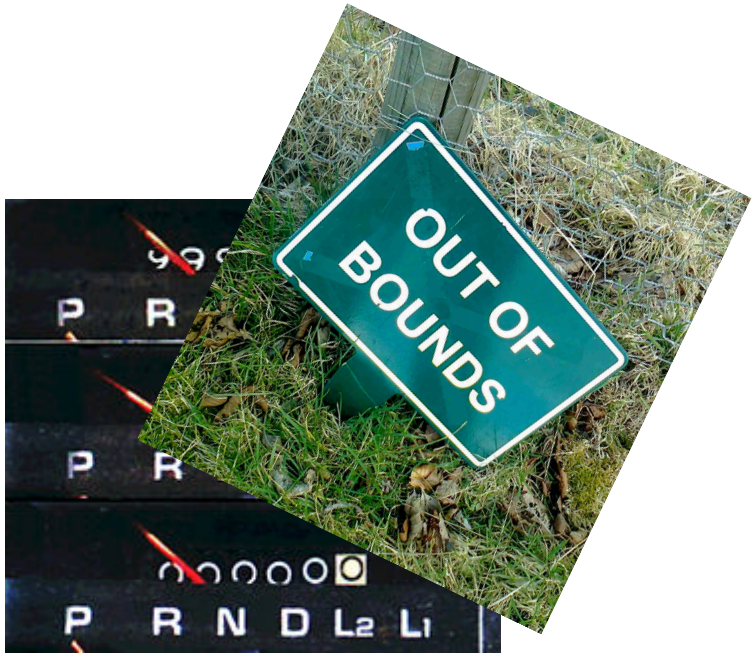
Source



Undefined Behavior

Retrofitting Transformation

Source



Bounds Safety

Retrofitting Transformation

Source

Control Flow Integrity

Retrofitting Transformation

Source

LLVM

```
  .globl  "_*add_forty_two<Int32>:Int32"
  .align  4, 0x90
"_*add_forty_two<Int32>:Int32":
  .cfi_startproc
  pushq %rbp
Ltmp1992:
  .cfi_def_cfa_offset 16
Ltmp1993:
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
Ltmp1994:
  .cfi_def_cfa_register %rbp
  addl  $42, %edi
  movl  %edi, %eax
  popq  %rbp
  retq
  .cfi_endproc
```

Assembly

Instrumentation

Retrofitting Transformation

LLVM : Optimize ⟶

Instrument ⟶ Optimize

Motivating Example

```
int foo(int a, int b) {
  int c = a + b;
  return c;
}
```

LLVM : Optimize → Instrument → Optimize

Motivating Example

```
int foo(int a, int b) {
  int c = a + b;
  return c;
}
```

signed integer overflow
= undefined behavior

LLVM : Optimize → Instrument → Optimize

Motivating Example

Integer overflow checker

Instrumentation

```
int foo(int a, int b) {
  int c = a + b;
  return c;
}
```

signed integer overflow
= undefined behavior

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

LLVM : Optimize → Instrument → Optimize

Motivating Example

```
int foo(int a, int b) {
  int c = a + b;
  return c;
}
```

Integer overflow checker

Optimize (Linux LLVM)

signed integer overflow
= undefined behavior

Instrumentation removed

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

LLVM : Optimize → Instrument → Optimize

Motivating Example

```
int foo(int a, int b) {
  int c = a + b;
  return c;
}
```

Integer overflow checker

Optimize (Linux LLVM)

signed integer overflow
= undefined behavior

Instrumentation removed

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

Optimize
(MacOSX LLVM)

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Instrumentation
modified

LLVM : Optimize → Instrument → Optimize

Problem Statement

Can we detect erroneously removed/modified instrumentation due to compiler optimizations?

Challenges:

1. Checks may be completely removed.
2. Checks may be partially removed.
3. Checks may be moved.
4. Some checks are indeed redundant.

Solution:

Can we frame this as a reachability problem?

RUTGERS

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

RUTGERS

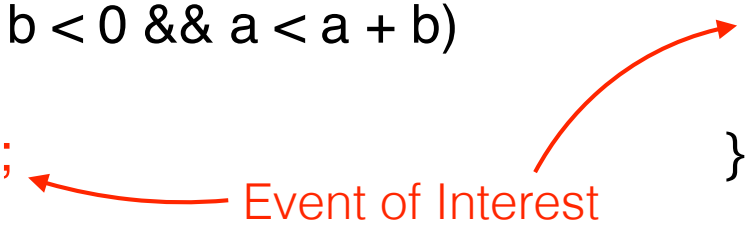
Reachability

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```



Reachability

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Event of Interest

Direct unsafe execution to exit path.

Reachability

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Event of Interest

Semantics same as P_{retro}

Direct unsafe execution to exit path.

Reachability

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Event of Interest

Semantics same as P_{retro}

Direct unsafe execution to exit path.

Given same inputs,

Reachability

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Event of Interest

Semantics same as P_{retro}

Direct unsafe execution to exit path.

Given same inputs,

1. P_{retro} and P_{opt} reaches the Event of Interest

Reachability

P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

P_{opt}: optimized P_{retro}

```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Event of Interest

Semantics same as P_{retro}

Direct unsafe execution to exit path.

Given same inputs,

1. P_{retro} and P_{opt} reaches the Event of Interest
2. P_{retro} and P_{opt} reaches exit(0)

Reachability

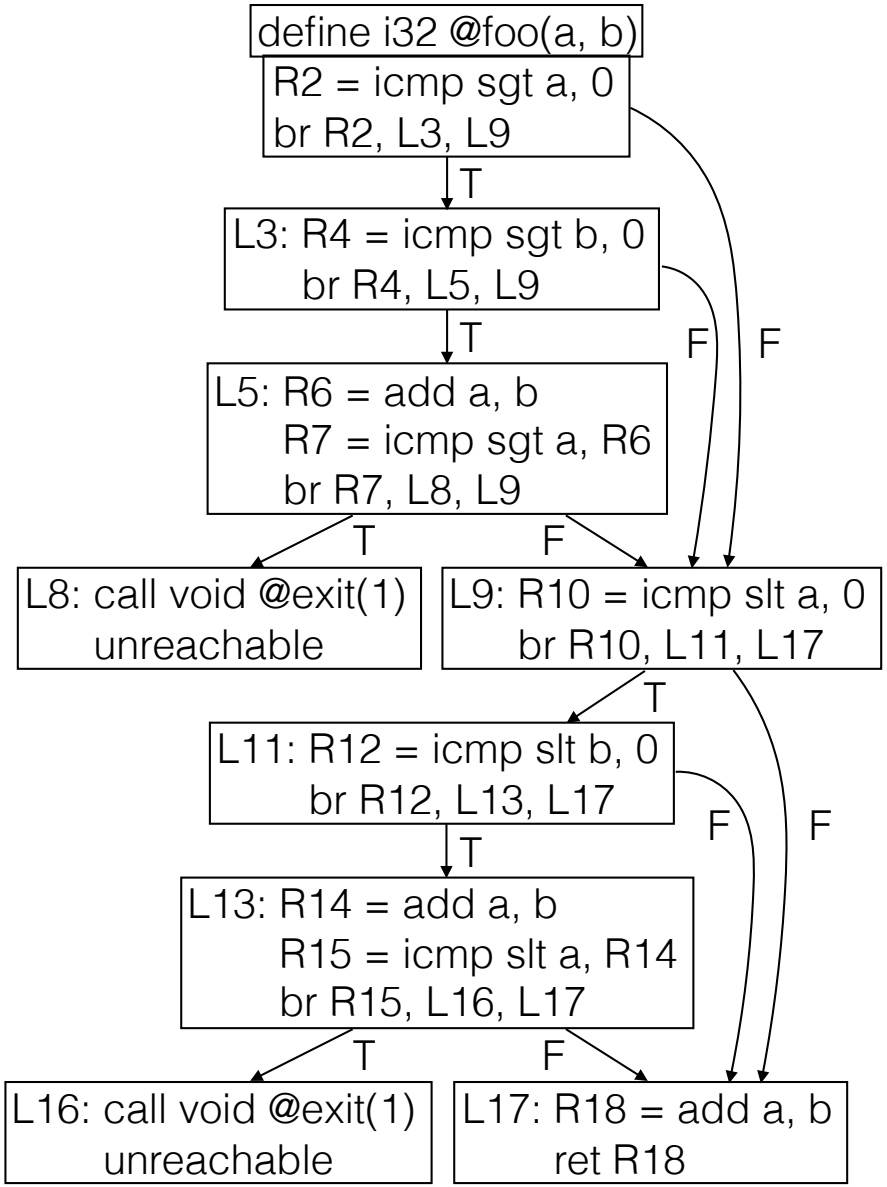
P_{retro}: retrofitted program

```
int foo(int a, int b) {
  if (a > 0 && b > 0 && a > a + b)
    exit(1);
  if (a < 0 && b < 0 && a < a + b)
    exit(1);
  int c = a + b;
  return c;
}
```

RUTGERS

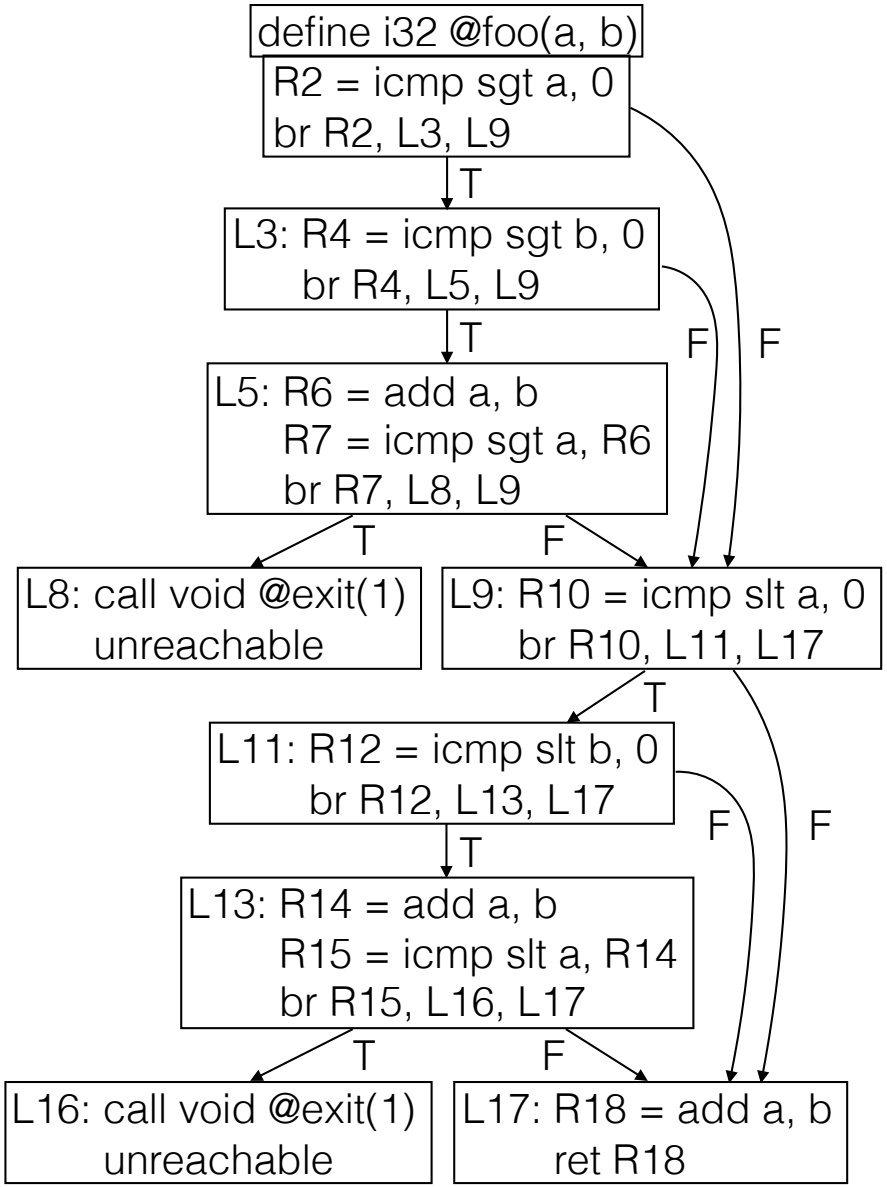
Reachability

P_{retro}: retrofitted program



Reachability

P_{retro}: retrofitted program

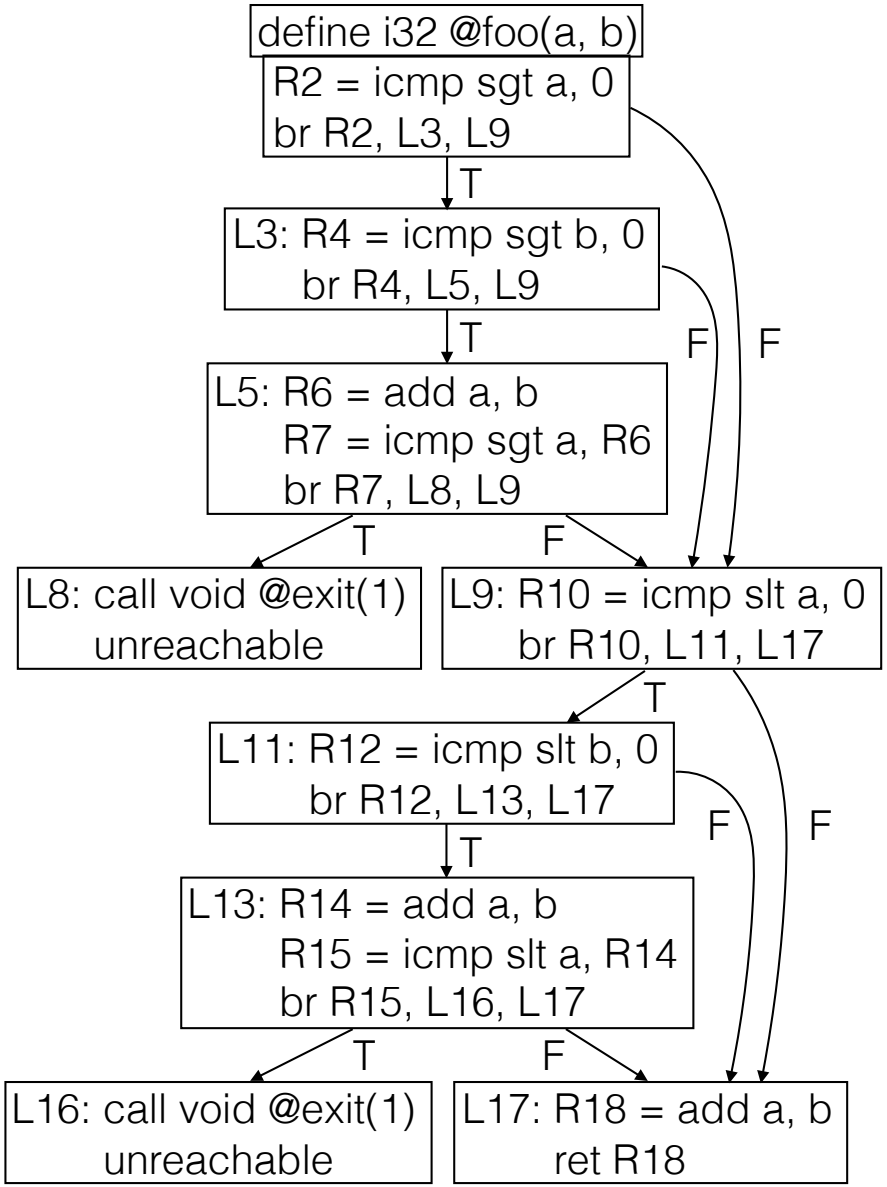


P_{opt}: optimized P_{retro}

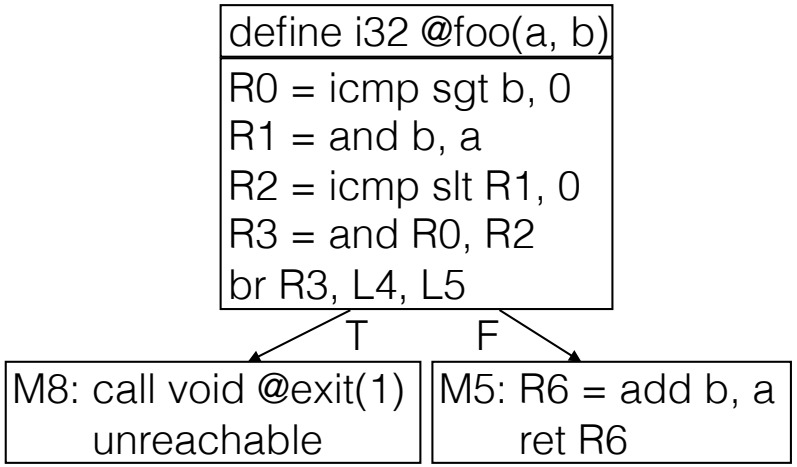
```
int foo(int a, int b) {
  if (b > 0 && (a & b) > 0)
    exit(0)
  int c = a + b;
  return c;
}
```

Reachability

P_{retro}: retrofitted program

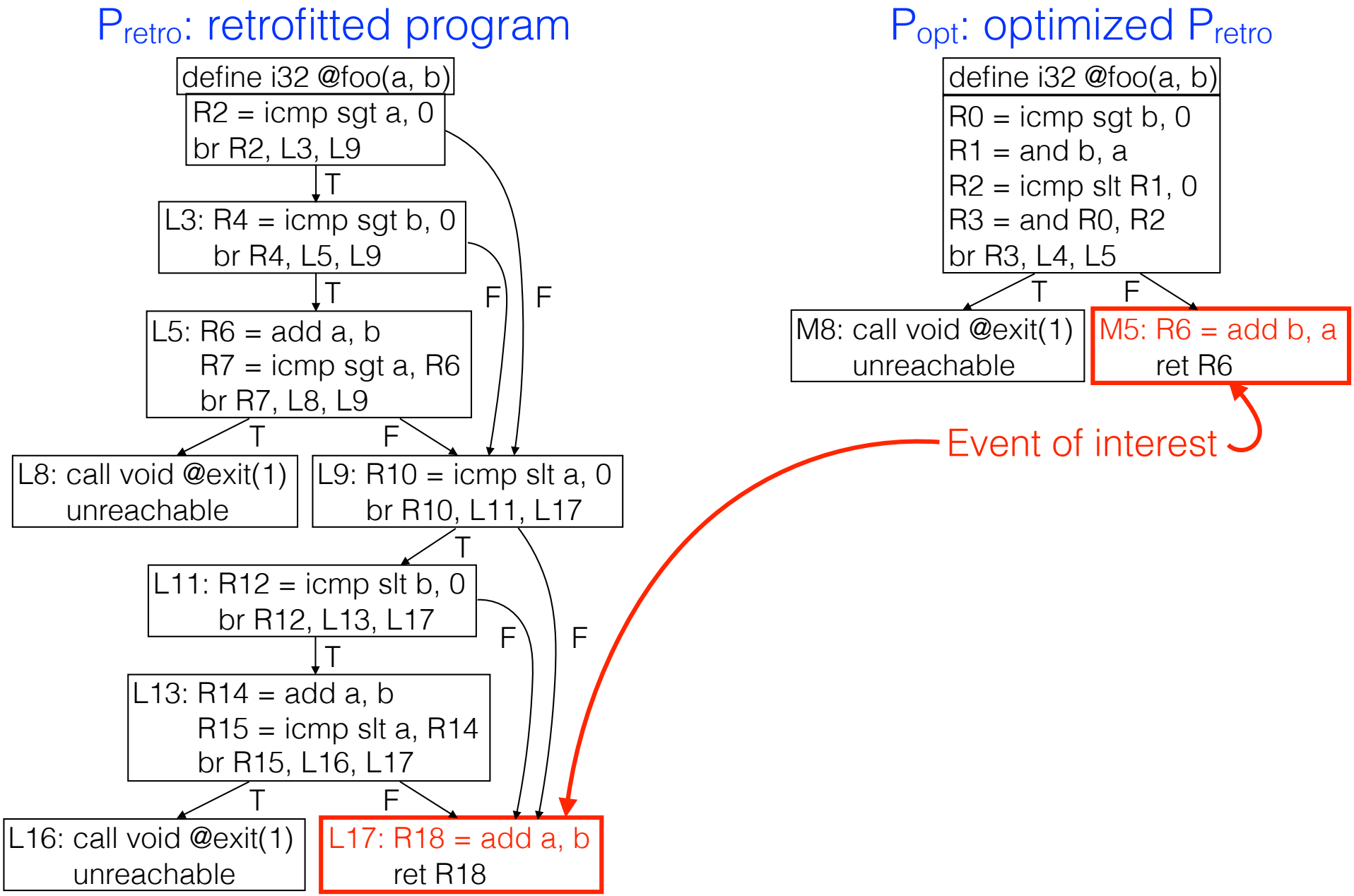


P_{opt}: optimized P_{retro}

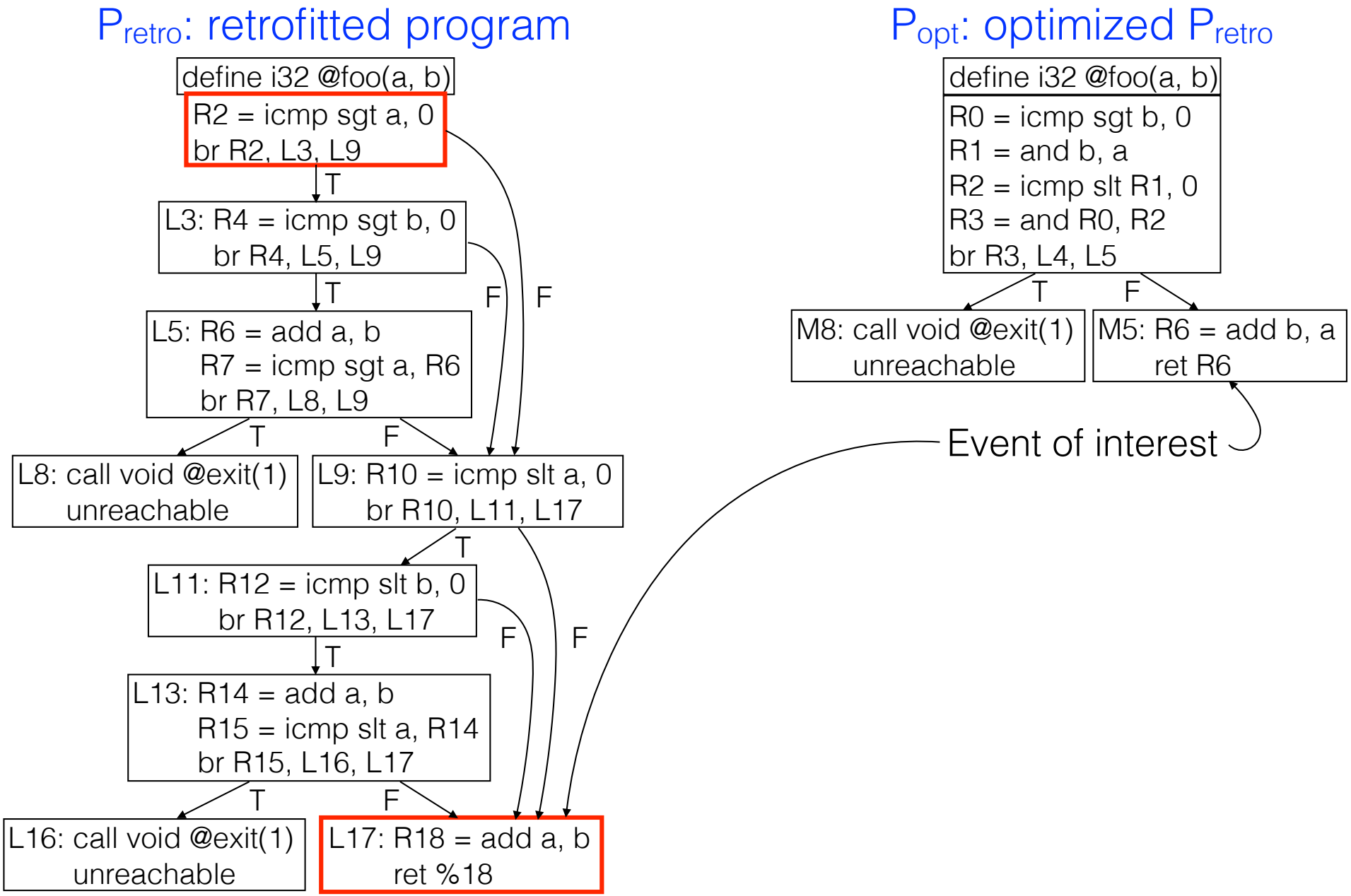


RUTGERS

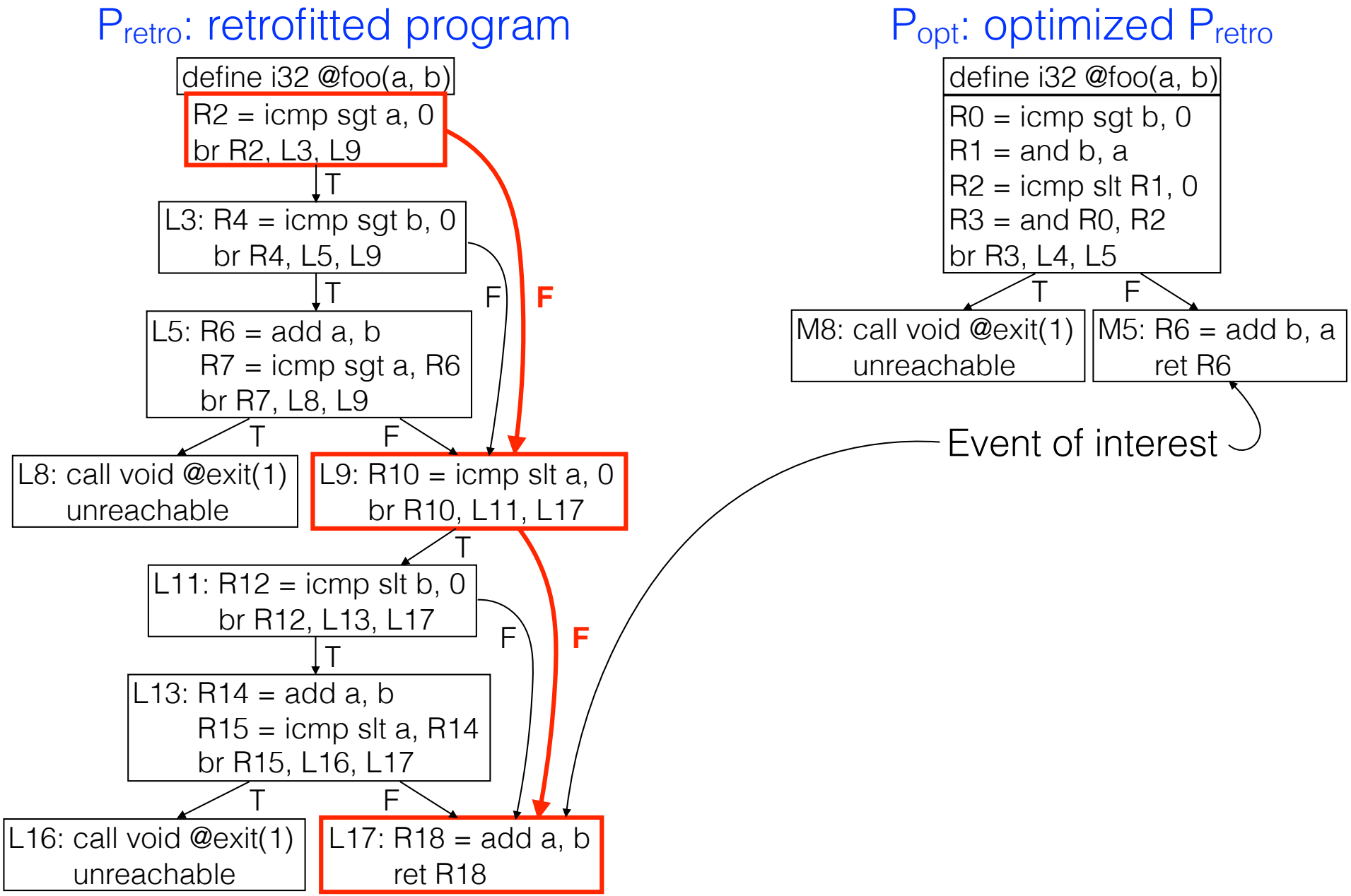
Reachability



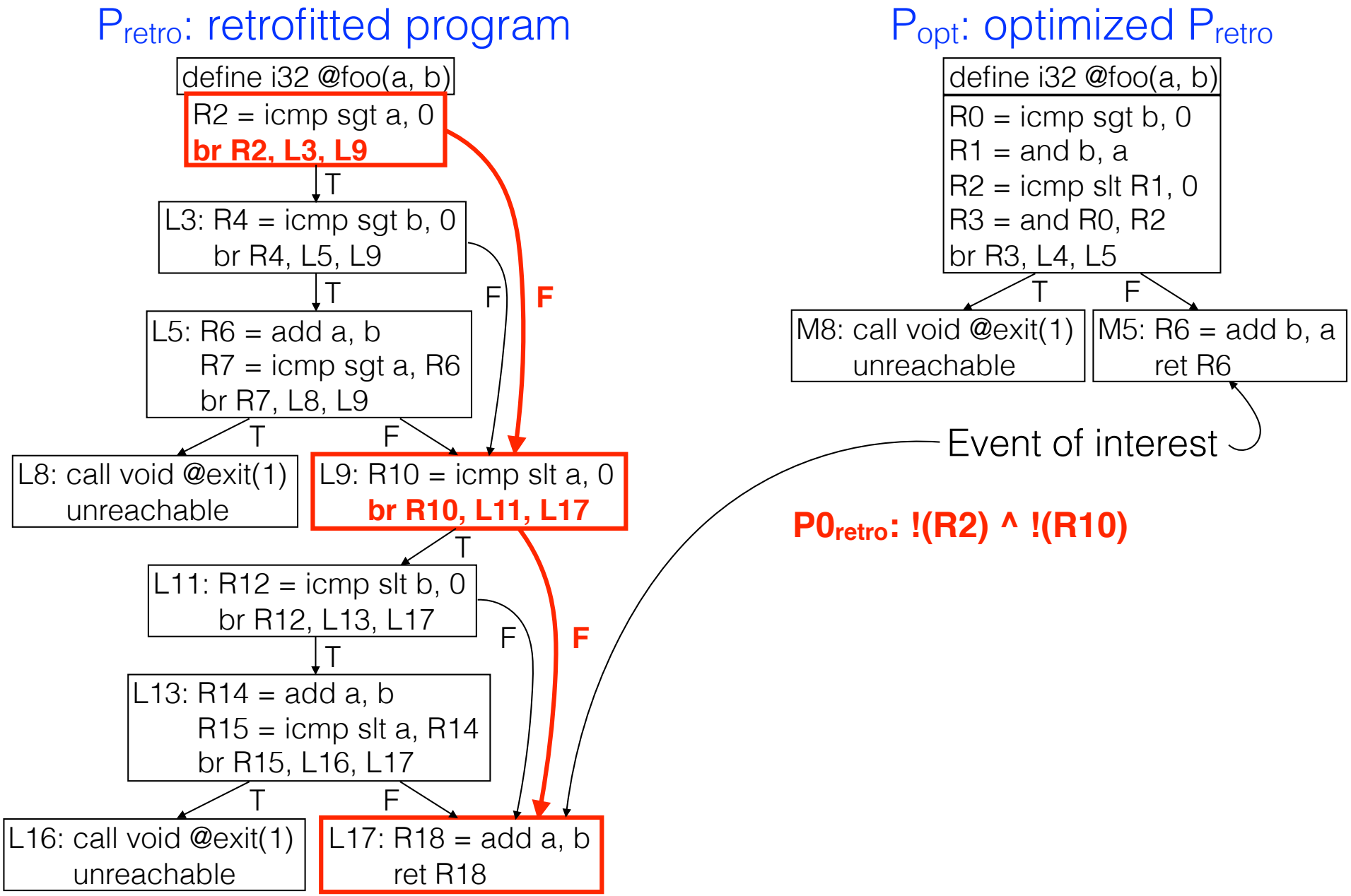
Reachability



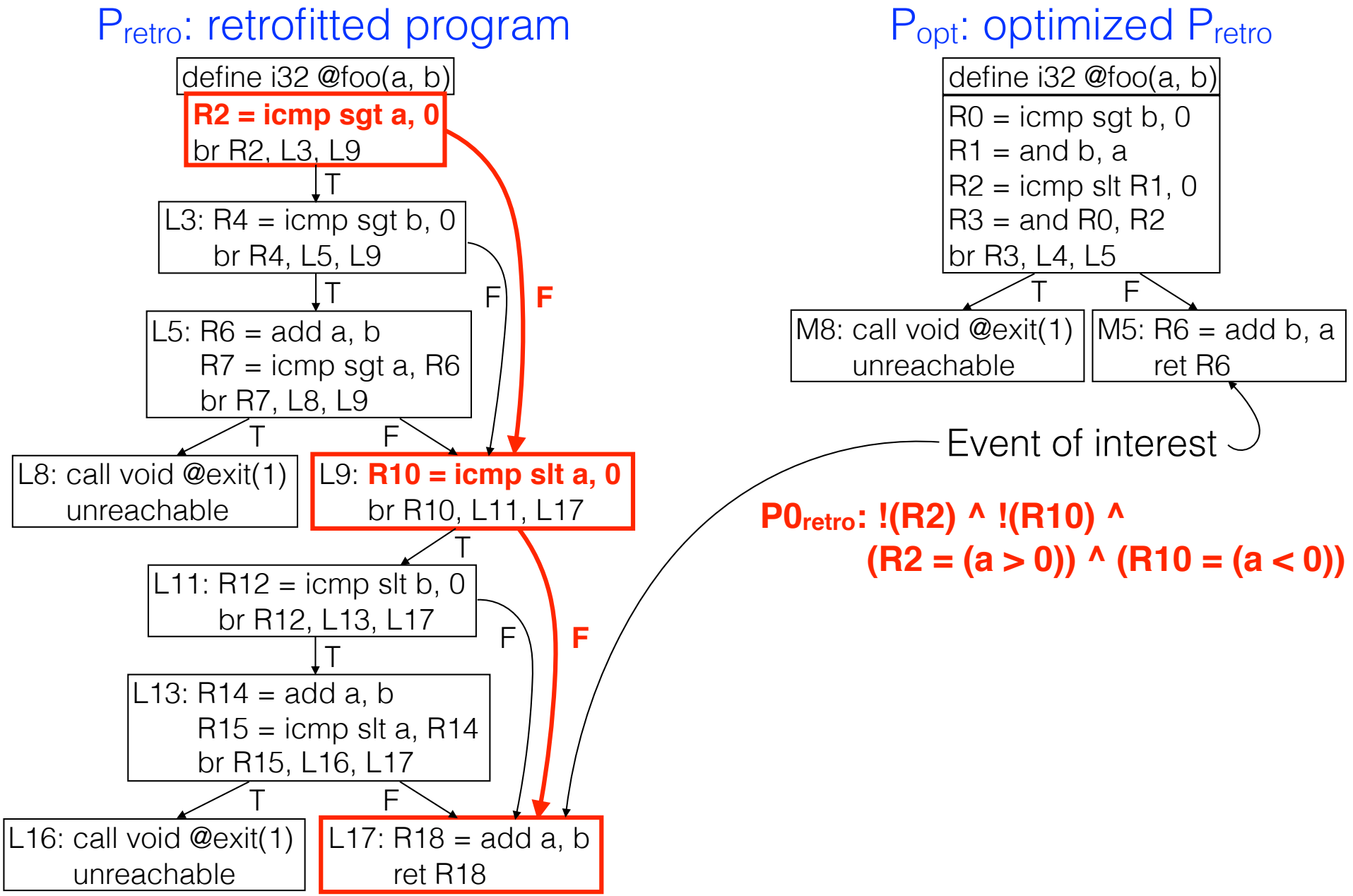
Reachability



Reachability

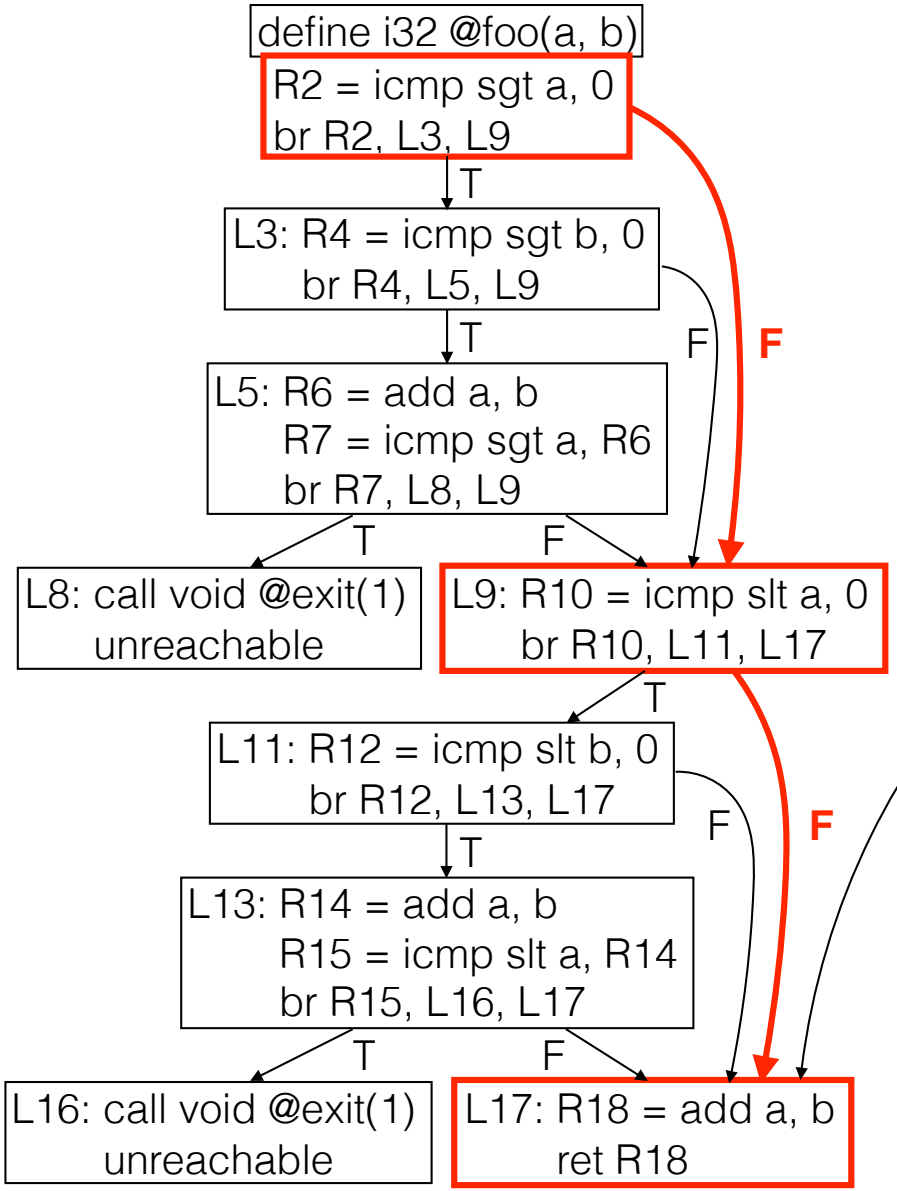


Reachability

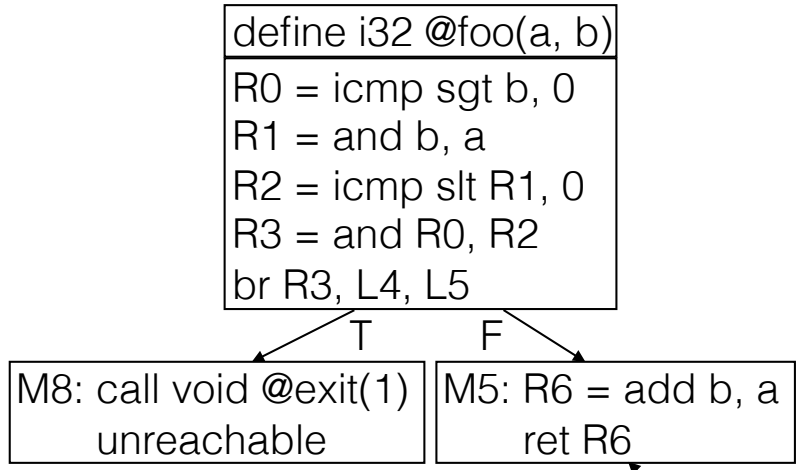


Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



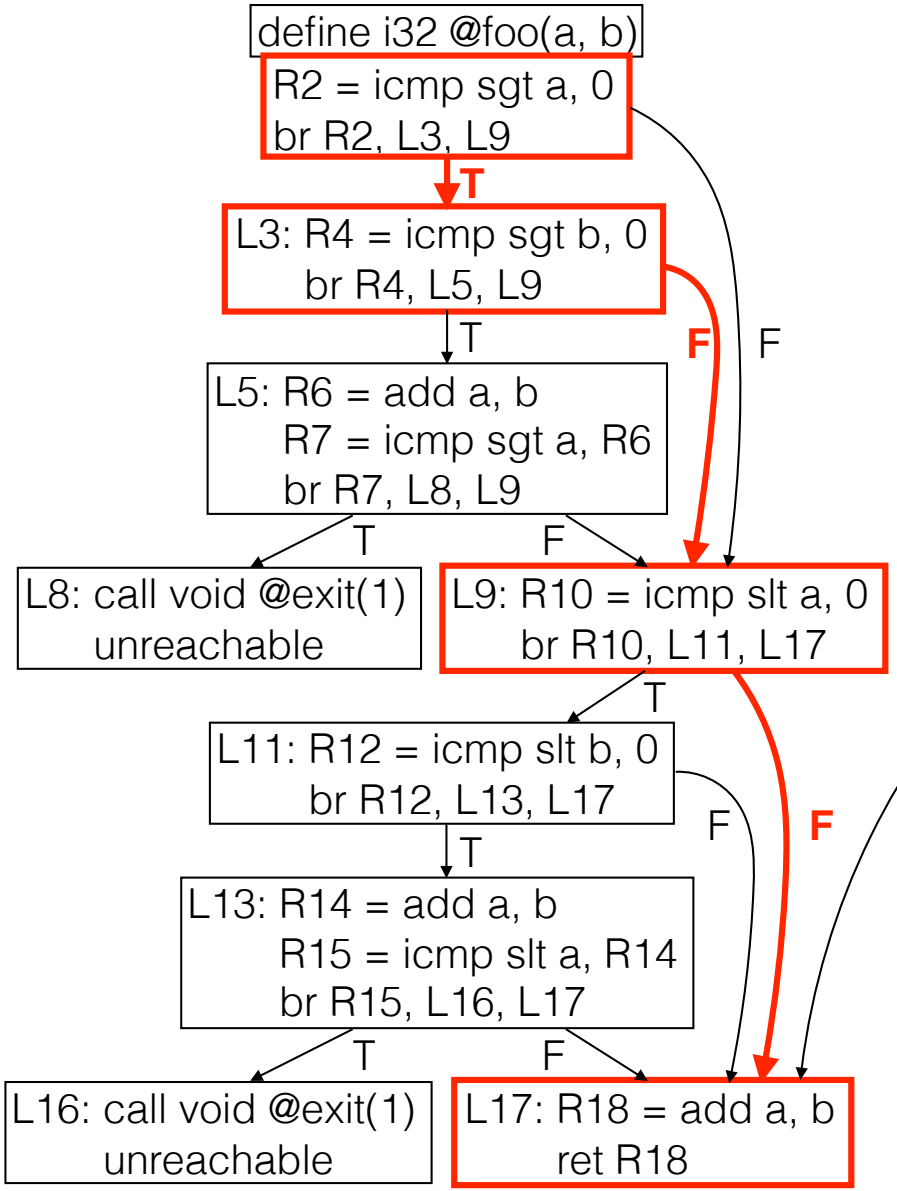
Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

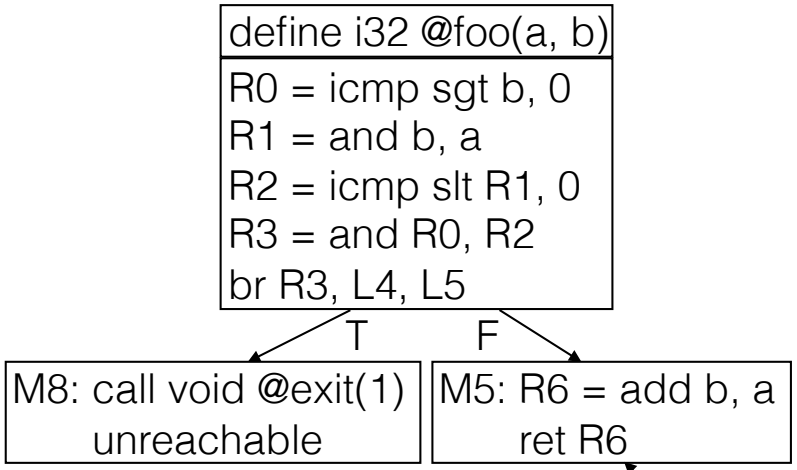
$E_{retro} = P0_{retro}$

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



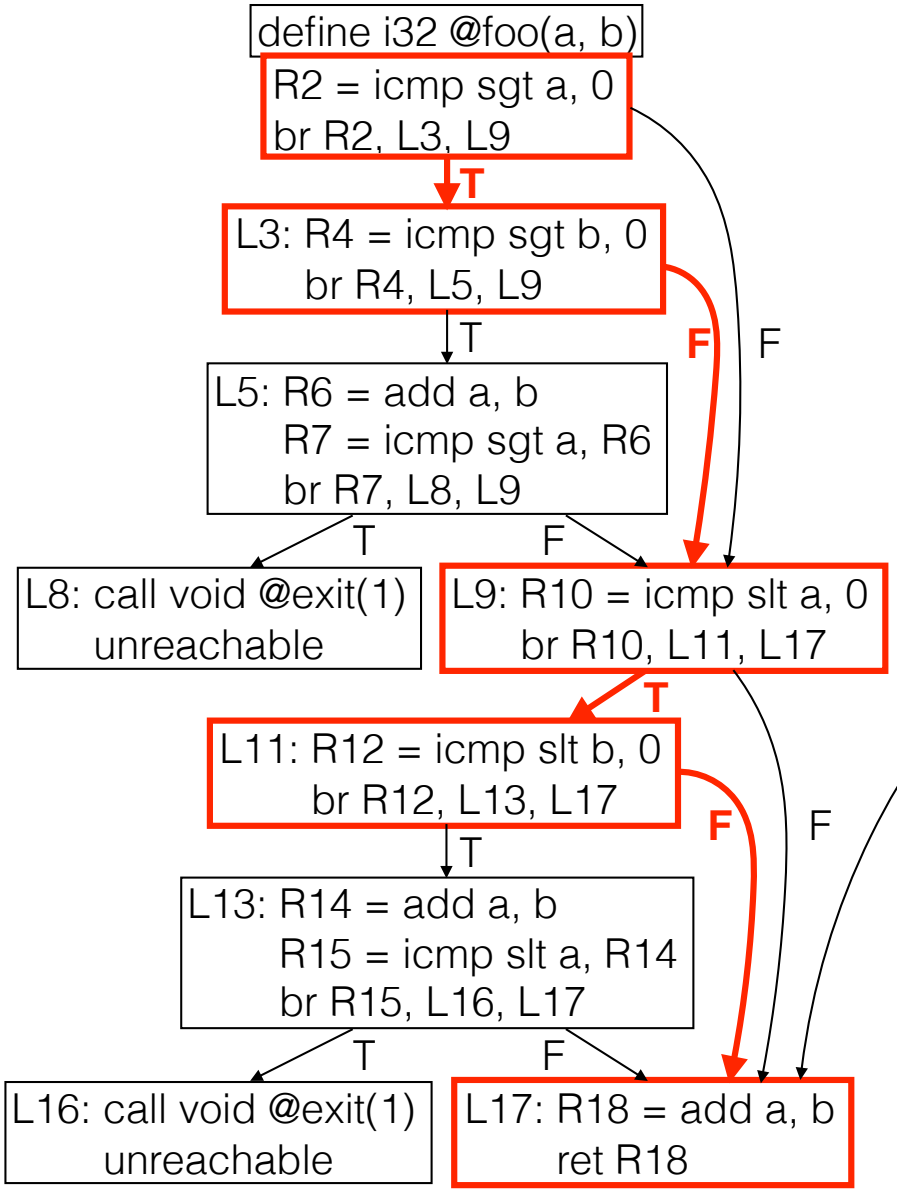
Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

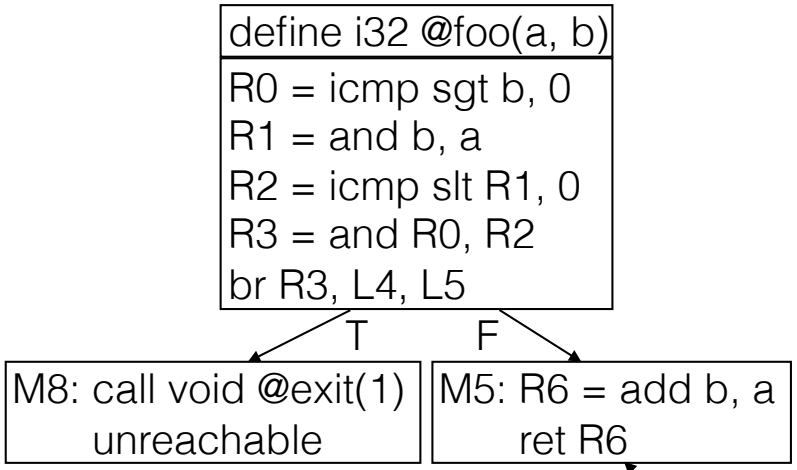
$E_{retro} = P0_{retro}$ v … v Pi_{retro}

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



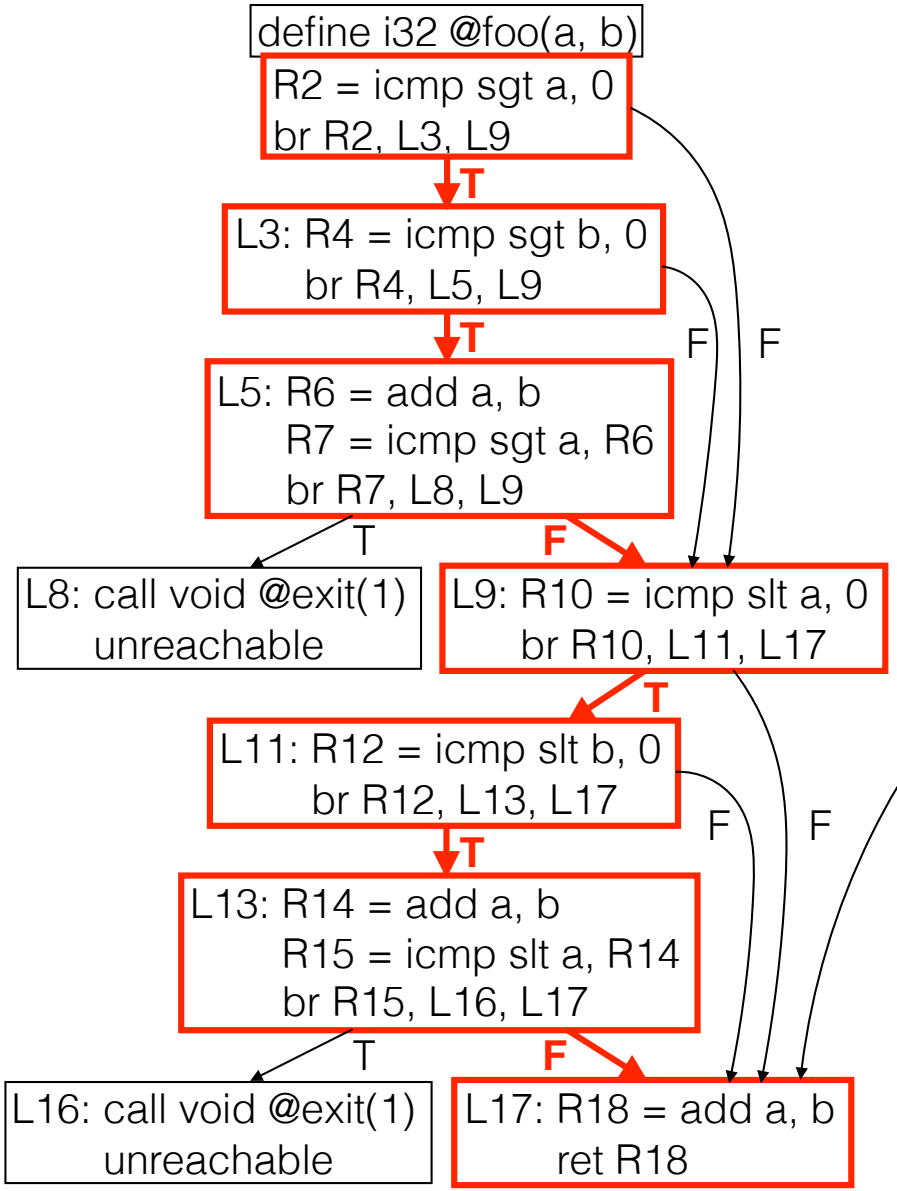
Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

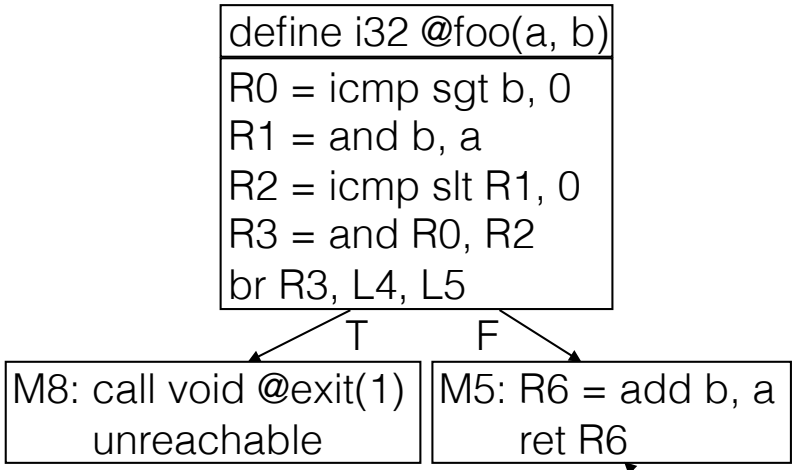
$E_{retro} = P0_{retro}$ v … v Pi_{retro}

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



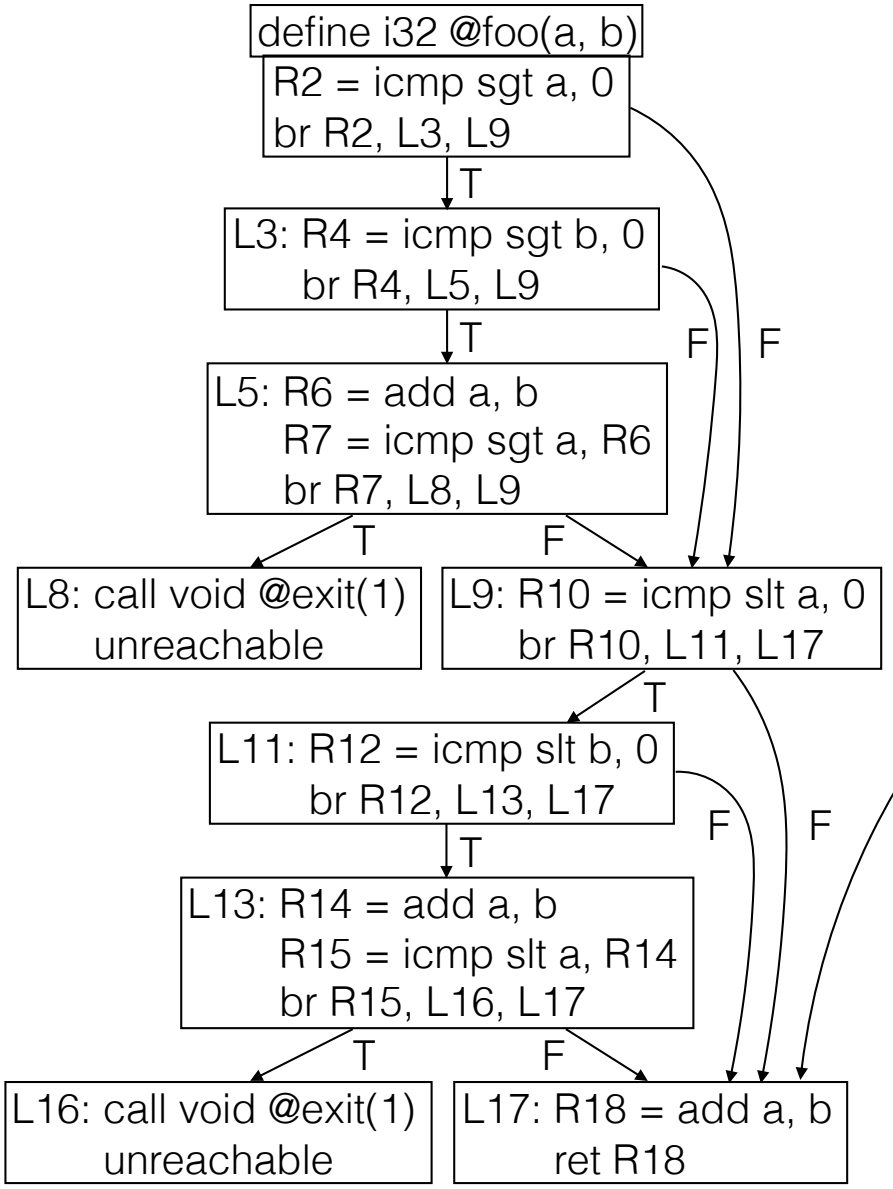
Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

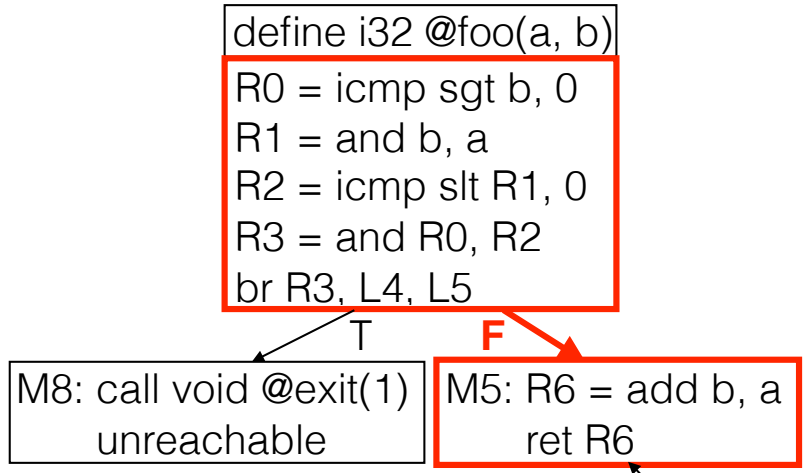
$E_{retro} = P0_{retro}$ v … v Pi_{retro}

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

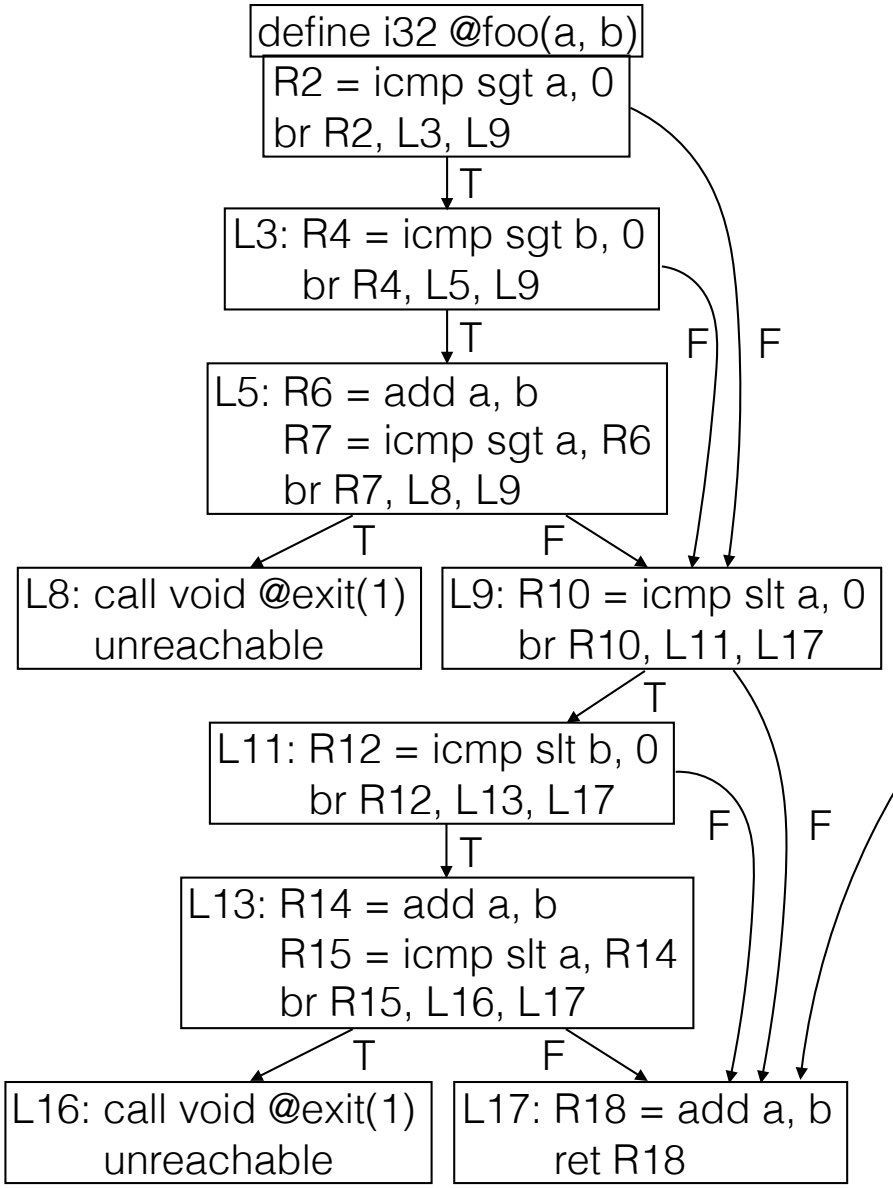
$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

$E_{retro} = P0_{retro}$ v … v Pi_{retro}

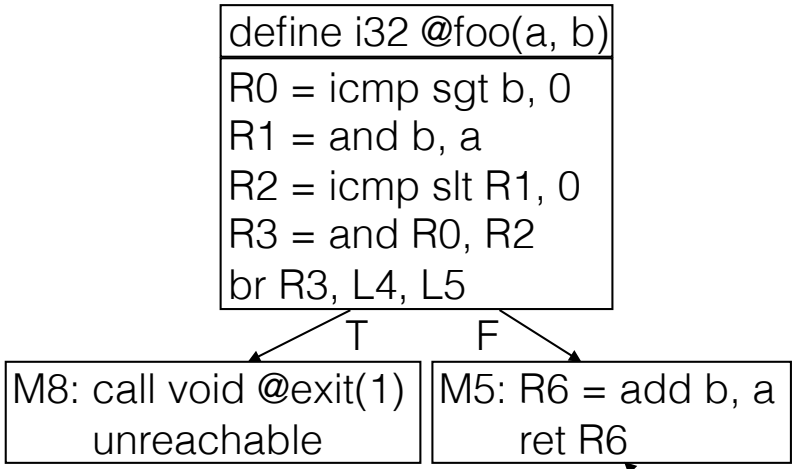
$E_{opt} = P0_{opt}$ v … v Pj_{opt}

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

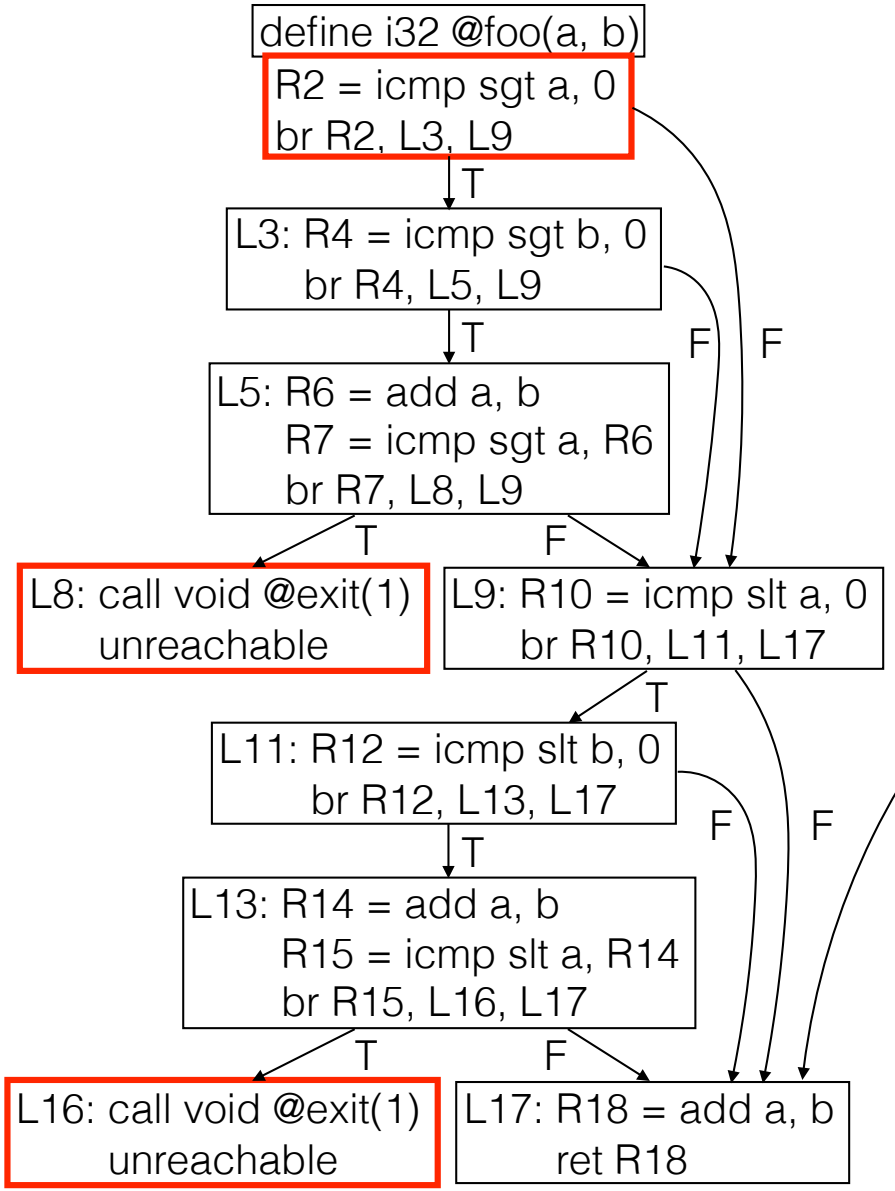
$E_{retro} = P0_{retro} \vee \ldots \vee Pi_{retro}$

$E_{opt} = P0_{opt} \vee \ldots \vee Pj_{opt}$

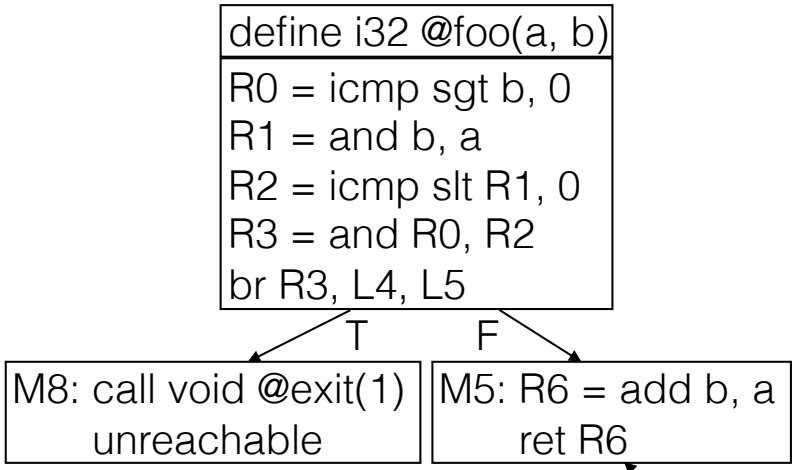
If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} \Rightarrow E_{opt}$)

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

$E_{retro} = P0_{retro} \vee \ldots \vee Pi_{retro}$

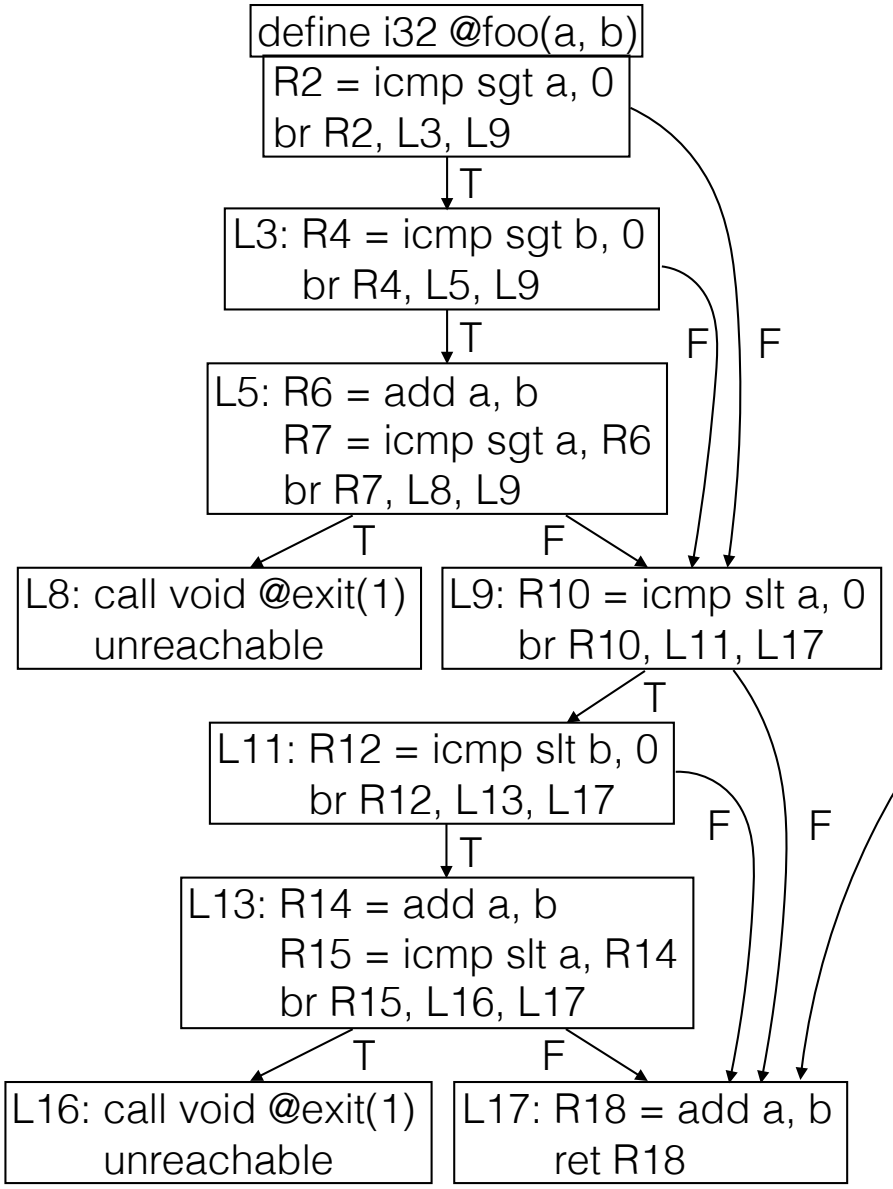
$E_{opt} = P0_{opt} \vee \ldots \vee Pj_{opt}$

If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} => E_{opt}$)

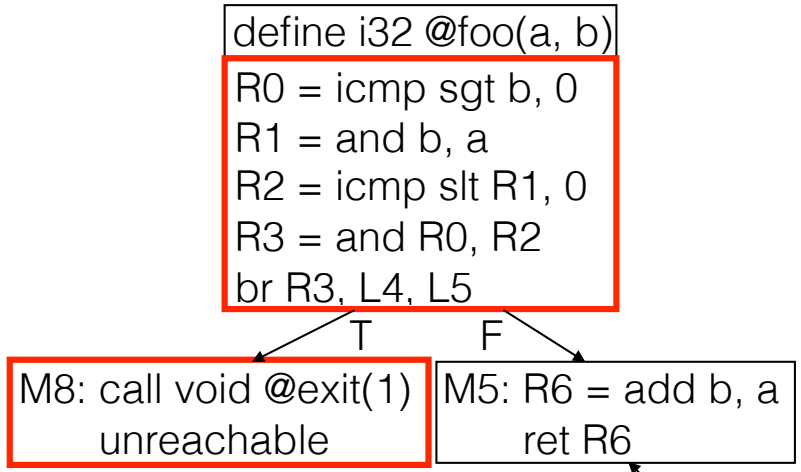
$\sim E_{retro}$

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

$E_{retro} = P0_{retro} \vee \ldots \vee Pi_{retro}$

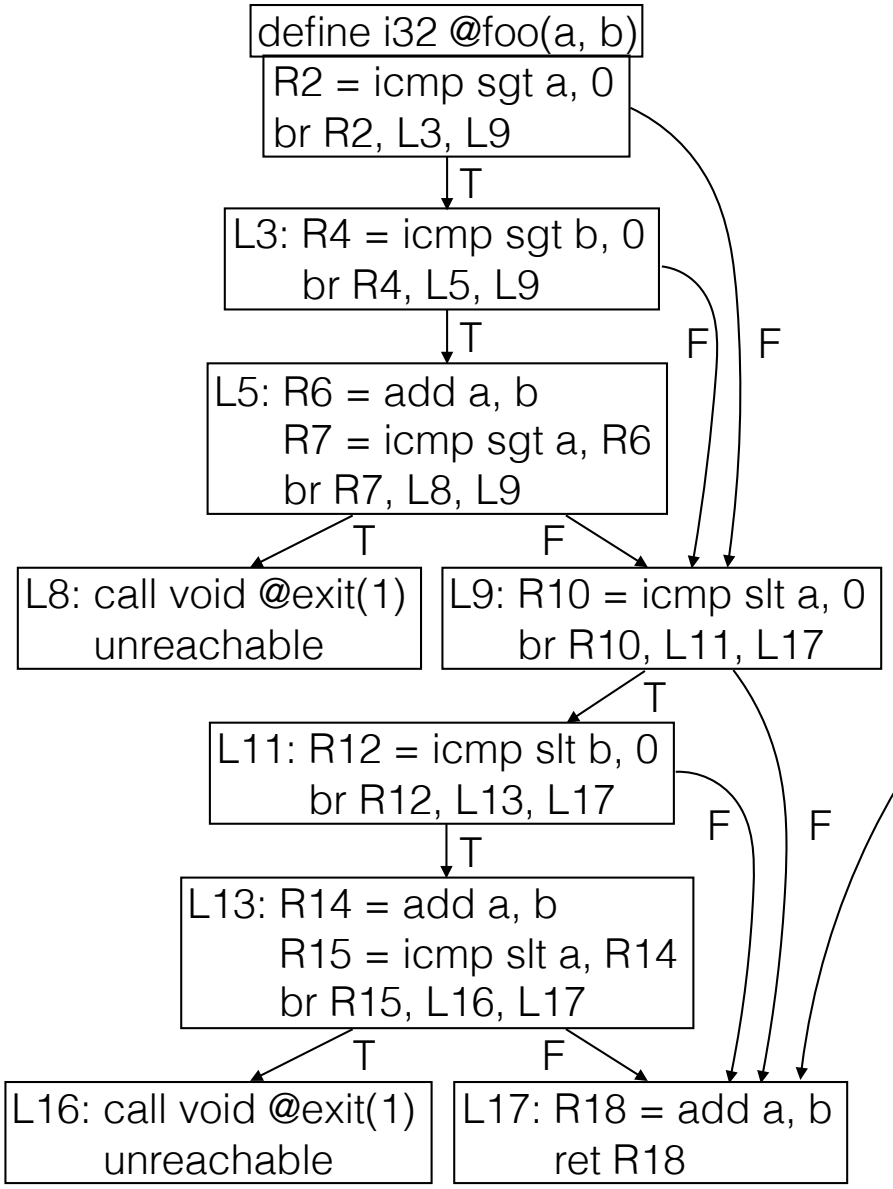
$E_{opt} = P0_{opt} \vee \ldots \vee Pj_{opt}$

If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} => E_{opt}$)

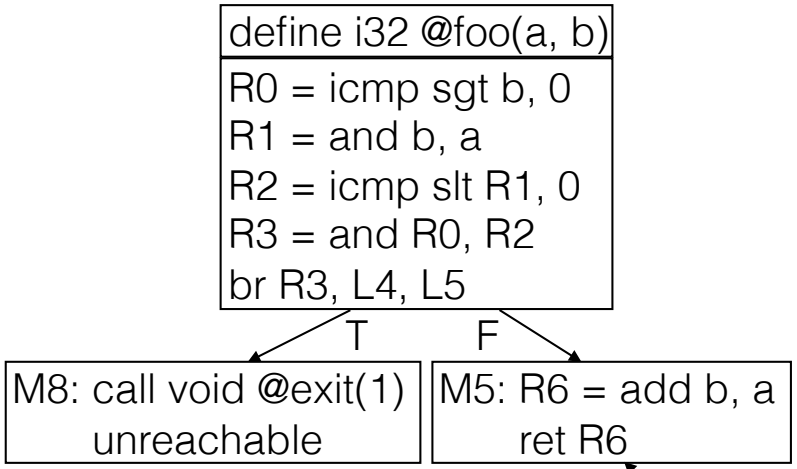
$\sim E_{retro}$, **$\sim E_{opt}$**

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

$E_{retro} = P0_{retro} \vee \ldots \vee Pi_{retro}$

$E_{opt} = P0_{opt} \vee \ldots \vee Pj_{opt}$

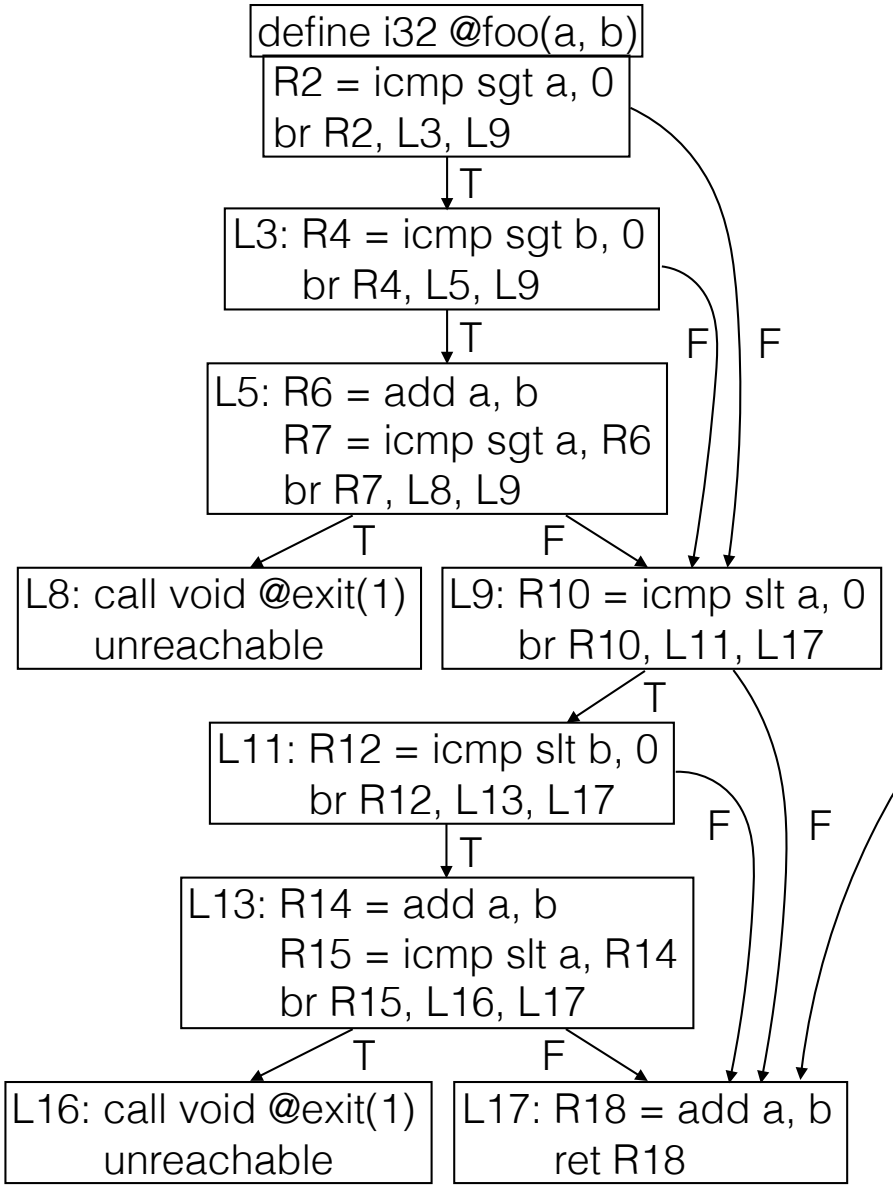
If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} => E_{opt}$)

$\sim E_{retro}$, $\sim E_{opt}$

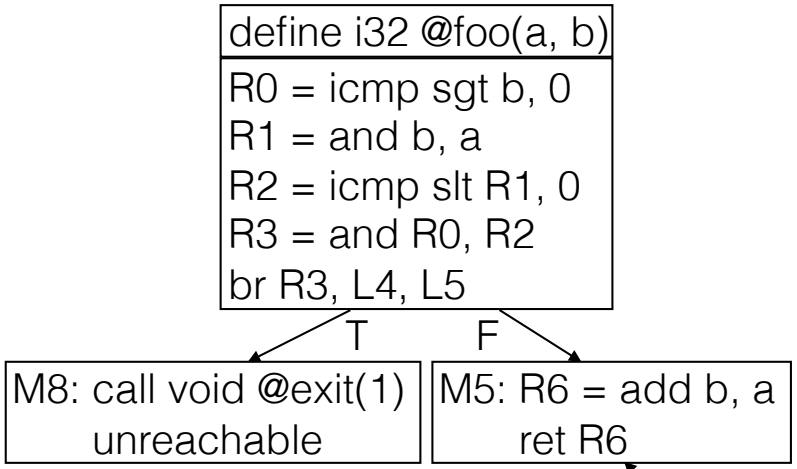
If P_{retro} does not reach event of interest, then P_{opt} does not reach event of interest ($\sim E_{retro} => \sim E_{opt}$)

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

$E_{retro} = P0_{retro} \vee \ldots \vee Pi_{retro}$

$E_{opt} = P0_{opt} \vee \ldots \vee Pj_{opt}$

If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} \Rightarrow E_{opt}$)

$\sim E_{retro}$, $\sim E_{opt}$

If P_{opt} reaches event of interest, then P_{retro} reaches event of interest ($E_{opt} \Rightarrow E_{retro}$)

Initial Prototype

- Built prototype for LLVM IR programs.
- Modified retrofitting transformation to mark event of interests.
- Z3 for query.
- Naive integer overflow checker, Address Sanitizer, SoftboundCETS
- Discovered a bug in SoftBoundCETS

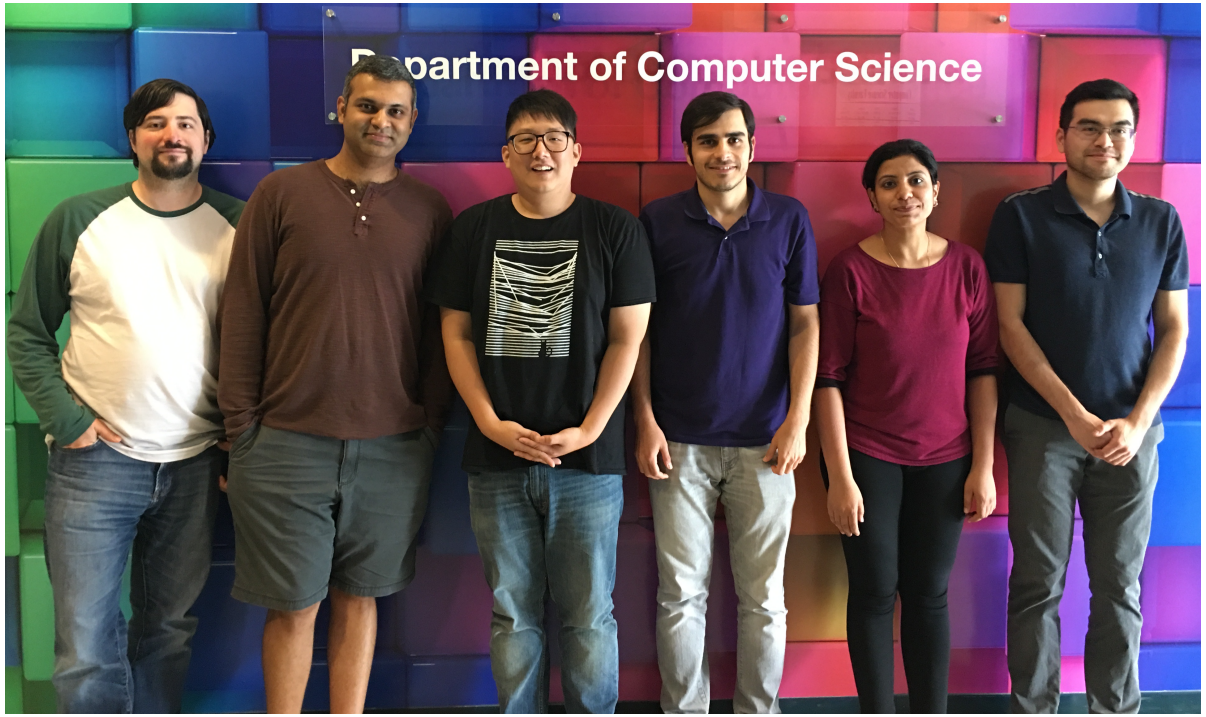
RUTGERS

Summary

- Optimization do erroneously remove checks.
- Formulating as Reachability detects errors.
- Leverage translation validation approaches to identify cut-points
  - Memory equivalence axioms?
  - More precise encoding for loops?

RUTGERS

Push formal methods into the tool chain of mainstream developers



Alive-NJ:
https://github.com/rutgers-apl/alive-nj/

TaskProf:
https://github.com/rutgers-apl/TaskProf

PTRacer:
https://github.com/rutgers-apl/PTRacer

Other software prototypes from the Rutgers Programming Languages Group:
https://github.com/rutgers-apl/

RUTGERS