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

Jay Lim,
Vinod Ganapathy,
Santosh Nagarakatte

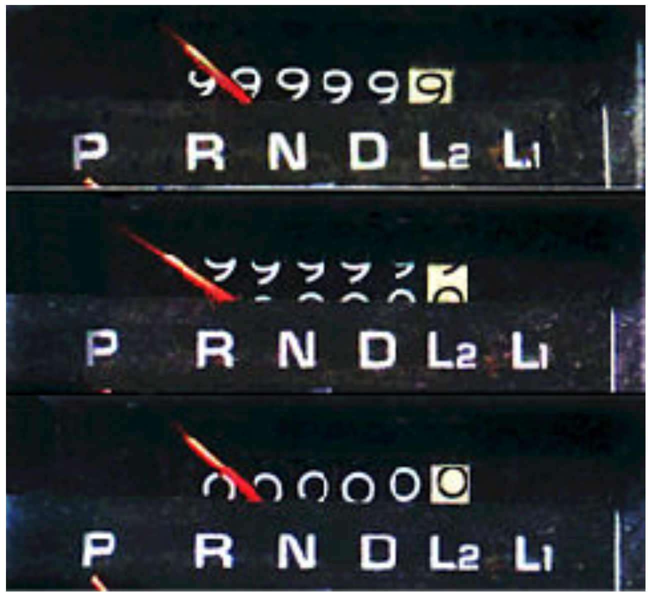
Department of Computer Science,
Rutgers University

Retrofitting Transformation

Source

Retrofitting Transformation

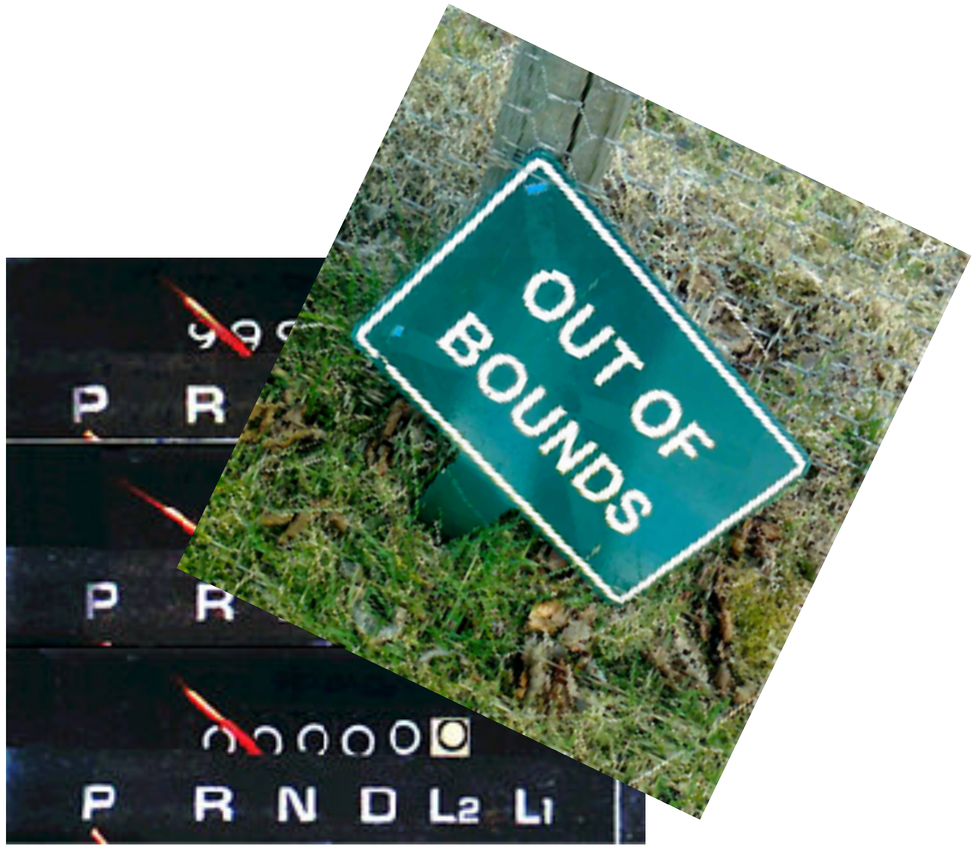
Source



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
Ltmp3552:
  .cfi_def_cfa_offset 16
Ltmp3553:
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
Ltmp3554:
  .cfi_def_cfa_register %rbp
  addl  $42, %edi
  movl  %edi, %eax
  popq  %rbp
  retq
  .cfi_endproc
```

Assembly

Instrumentation

Retrofitting Transformation

LLVM : Optimize $\longrightarrow$ Instrument $\longrightarrow$ 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

RUTGERS

Motivating Example

Integer overflow checker

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

signed integer overflow
= undefined behavior

Instrumentation

```
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;
}
```

signed integer overflow
= undefined behavior

Integer overflow checker

Optimize (Linux LLVM)

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;
}
```

signed integer overflow
= undefined behavior

Integer overflow checker

Optimize (Linux LLVM)

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

Motivating Example

n removed

a > a + b)

a < a + b)

int foo(int

int c = a

return c;

}

signed integ

= undefined

ptimize

OSX LLVM)

0)

LLVM : Optim

return c;

}

Instrumentation
modified

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?

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;
}
```

Event of Interest

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;
}
```

Event of Interest

Direct unsafe execution to exit path.

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;
}
```

Event of Interest

Direct unsafe execution to exit path.

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;
}
```

Semantics same as P_{retro}

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;
}
```

Event of Interest

Direct unsafe execution to exit path.

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;
}
```

Semantics same as P_{retro}

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;
}
```

Event of Interest

Direct unsafe execution to exit path.

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;
}
```

Semantics same as P_{retro}

Given same inputs,

1. P_{retro} and P_{opt} reaches 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;
}
```

Event of Interest

Direct unsafe execution to exit path.

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;
}
```

Semantics same as P_{retro}

Given same inputs,

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

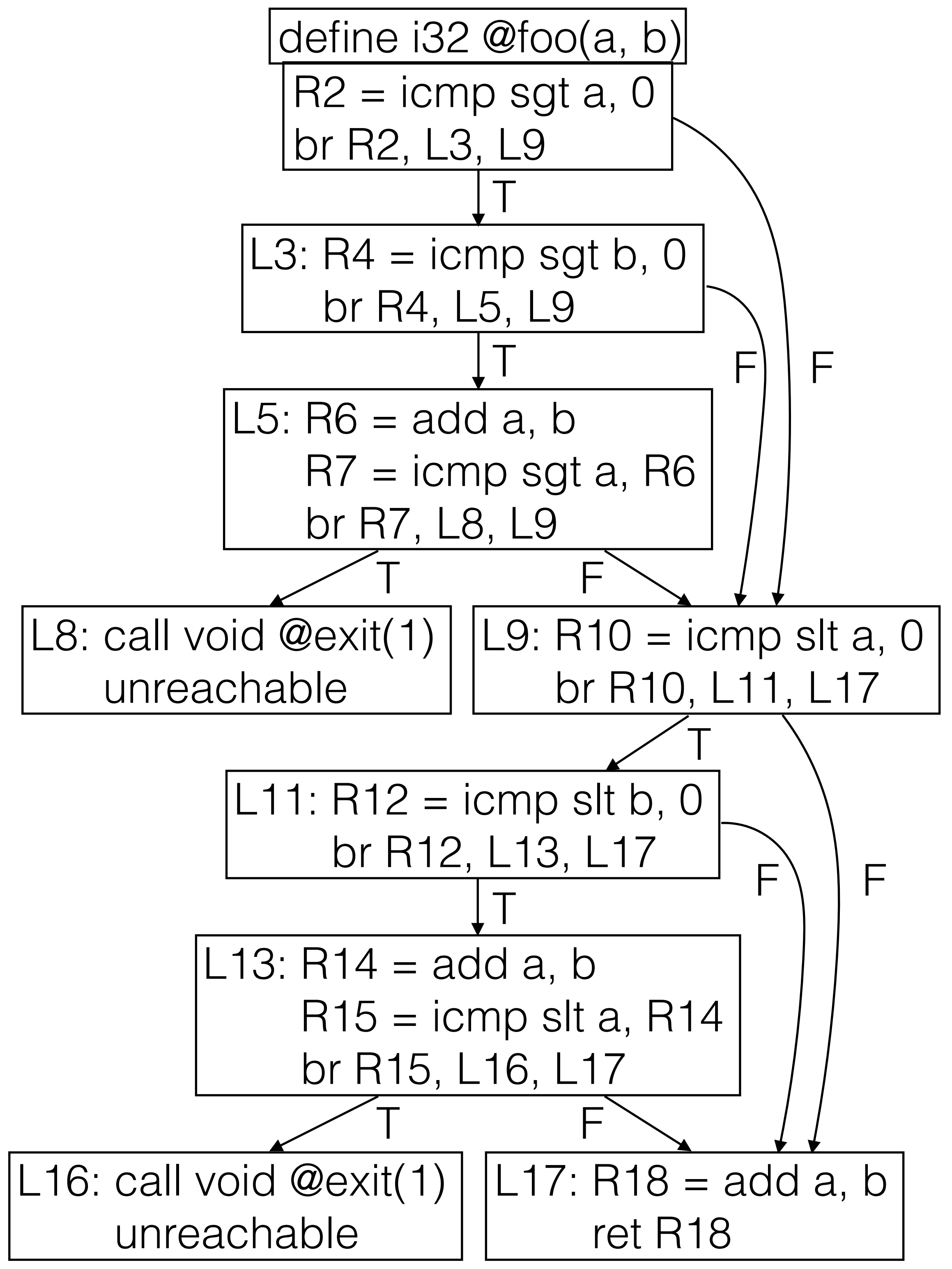
Reachability

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;
}
```

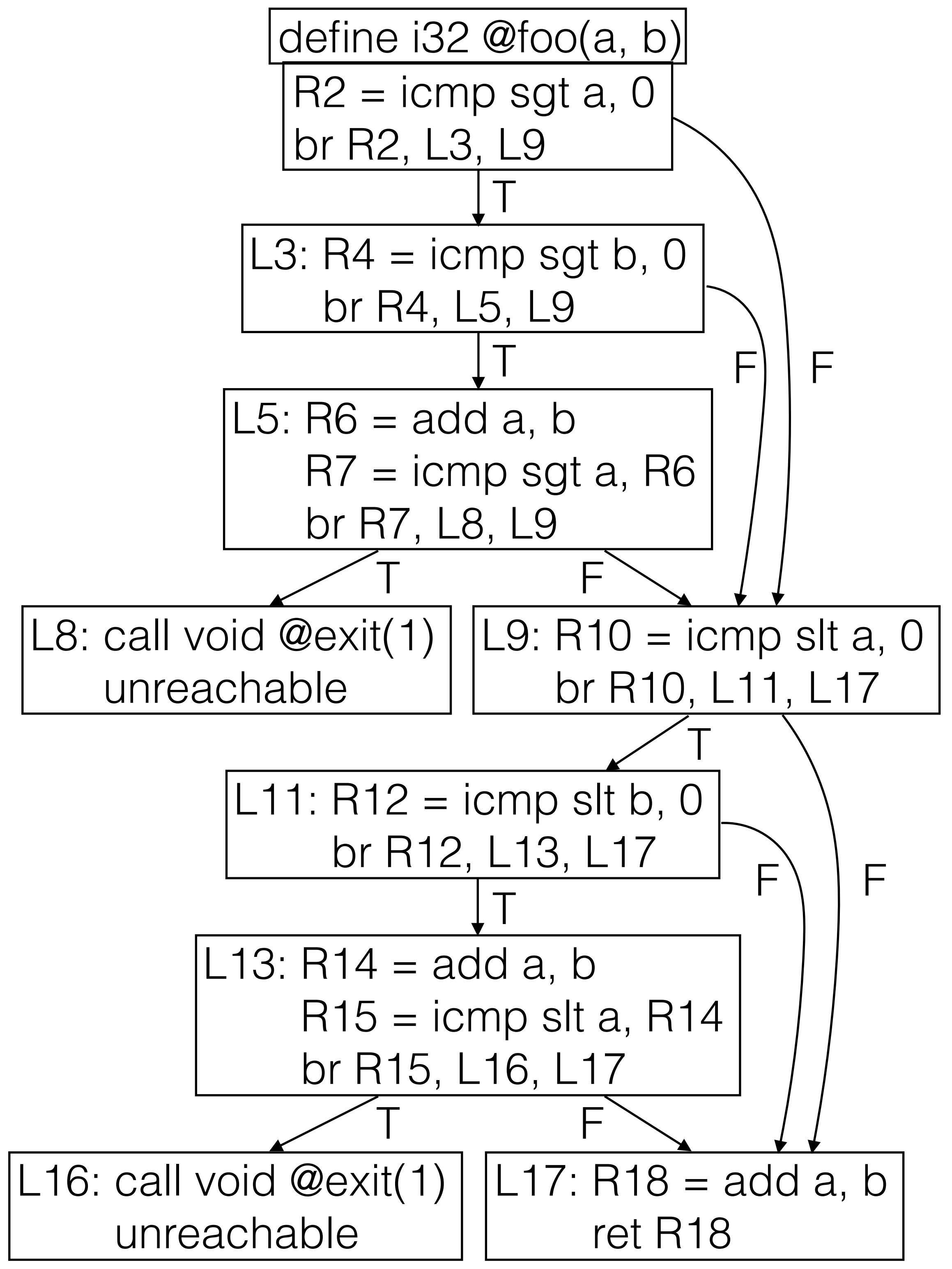
Reachability

P$_{retro}$: retrofitted program



Reachability

P_{retro}: retrofitted program



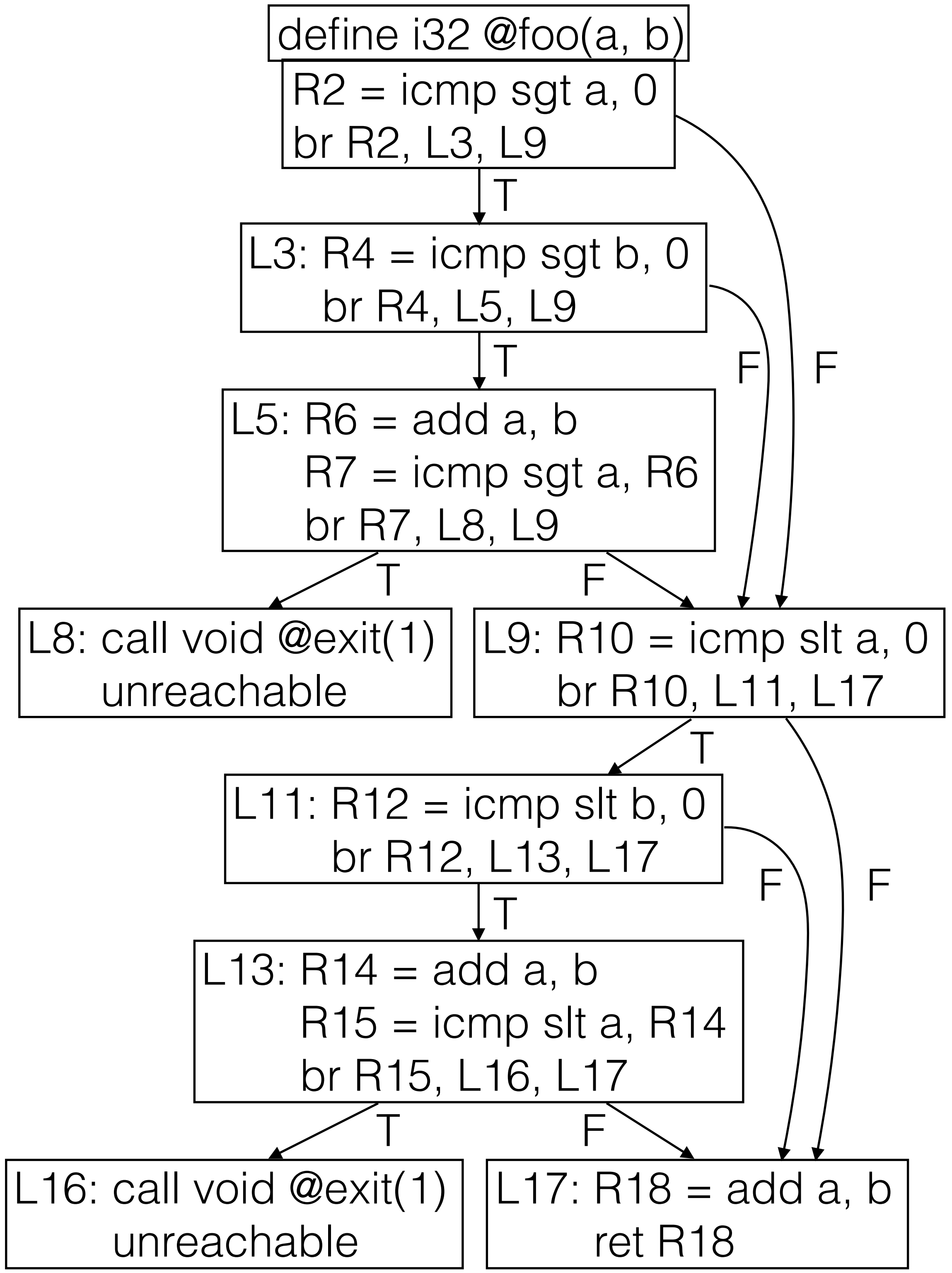
```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9

L3: R4 = icmp sgt b, 0
    br R4, L5, L9

L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9

L8: call void @exit(1)
    unreachable

L9: R10 = icmp slt a, 0
    br R10, L11, L17

L11: R12 = icmp slt b, 0
     br R12, L13, L17

L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17

L16: call void @exit(1)
     unreachable

L17: R18 = add a, b
     ret R18
```

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;
}
```

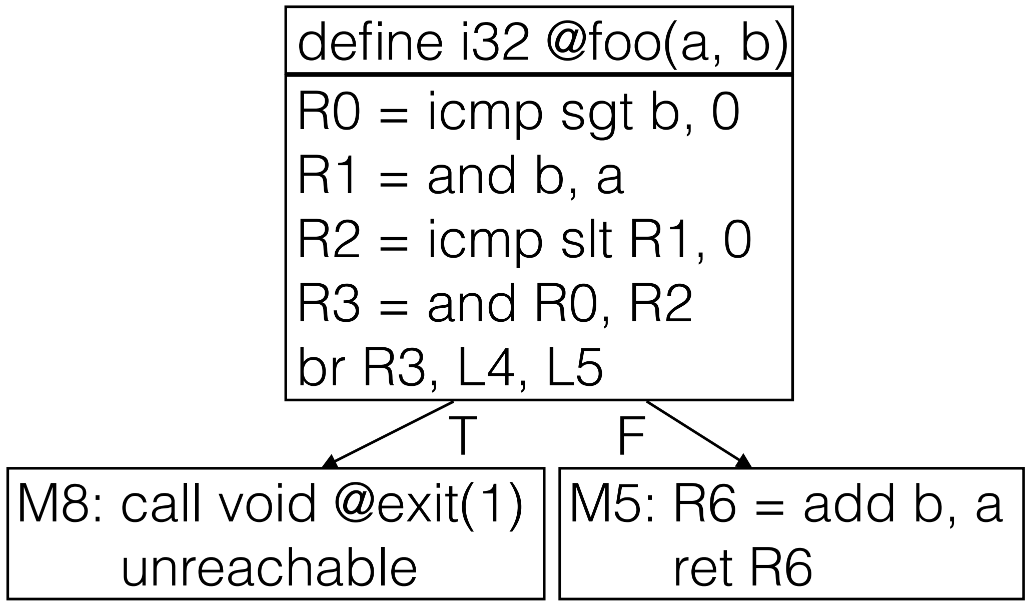
Reachability

P$_{retro}$: retrofitted program



```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9

L3: R4 = icmp sgt b, 0
    br R4, L5, L9

L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9

L8: call void @exit(1)
    unreachable

L9: R10 = icmp slt a, 0
    br R10, L11, L17

L11: R12 = icmp slt b, 0
     br R12, L13, L17

L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17

L16: call void @exit(1)
     unreachable

L17: R18 = add a, b
     ret R18
```

P$_{opt}$: optimized P$_{retro}$

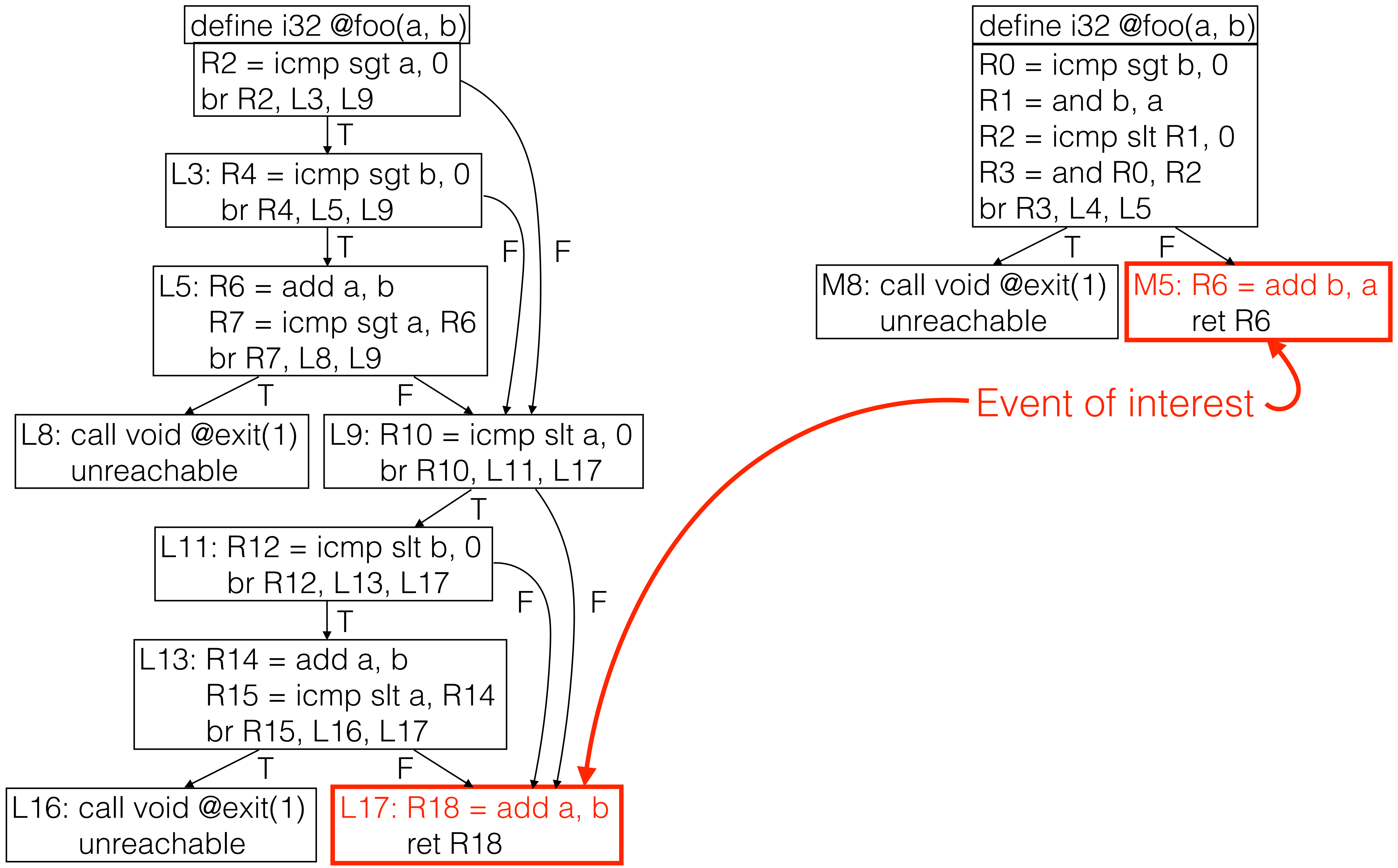


```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5

M8: call void @exit(1)
    unreachable

M5: R6 = add b, a
    ret R6
```

Reachability

P$_{retro}$: retrofitted program

P$_{opt}$: optimized P$_{retro}$

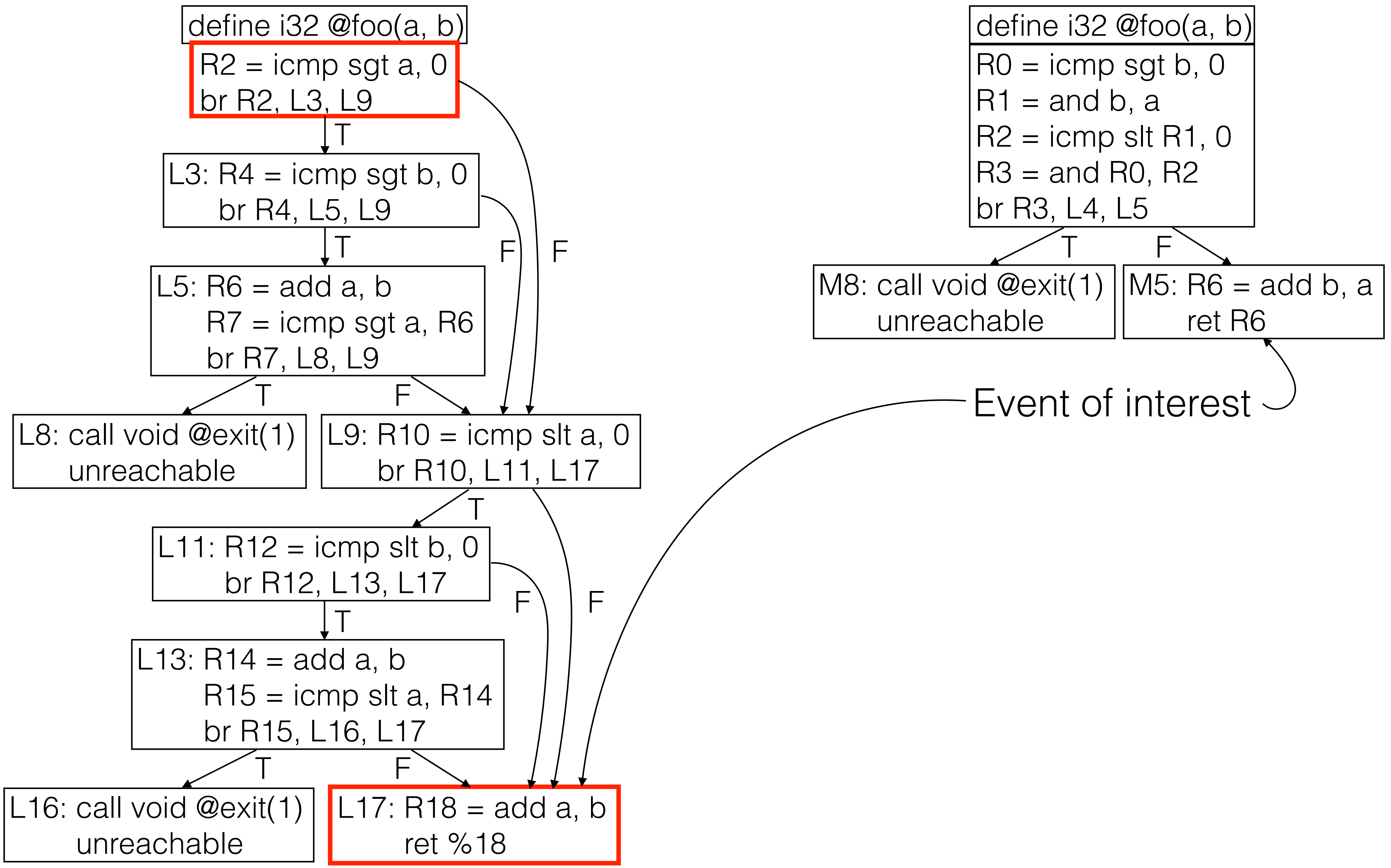


```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9

L3: R4 = icmp sgt b, 0
    br R4, L5, L9

L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9

L8: call void @exit(1)
    unreachable

L9: R10 = icmp slt a, 0
    br R10, L11, L17

L11: R12 = icmp slt b, 0
     br R12, L13, L17

L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17

L16: call void @exit(1)
     unreachable

L17: R18 = add a, b
     ret R18
```

```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5

M8: call void @exit(1)
    unreachable

M5: R6 = add b, a
    ret R6
```

Reachability

P_{retro}: retrofitted program $\quad$ P_{opt}: optimized P_{retro}



```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9
```

```
L3: R4 = icmp sgt b, 0
    br R4, L5, L9
```

```
L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9
```

```
L8: call void @exit(1)
    unreachable
```

```
L9: R10 = icmp slt a, 0
    br R10, L11, L17
```

```
L11: R12 = icmp slt b, 0
     br R12, L13, L17
```

```
L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17
```

```
L16: call void @exit(1)
     unreachable
```

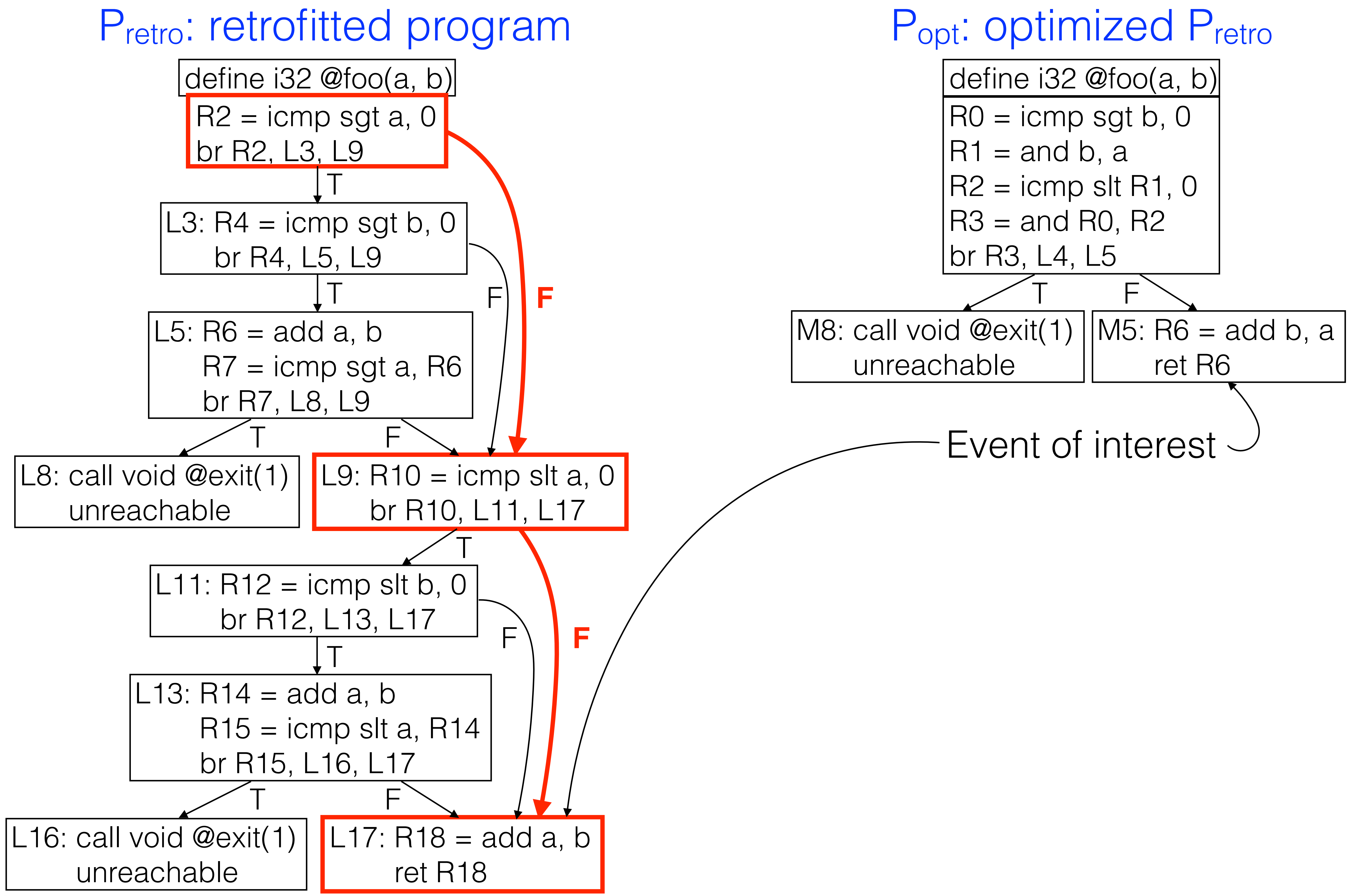
```
L17: R18 = add a, b
     ret %18
```

```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5
```

```
M8: call void @exit(1)
    unreachable
```

```
M5: R6 = add b, a
    ret R6
```

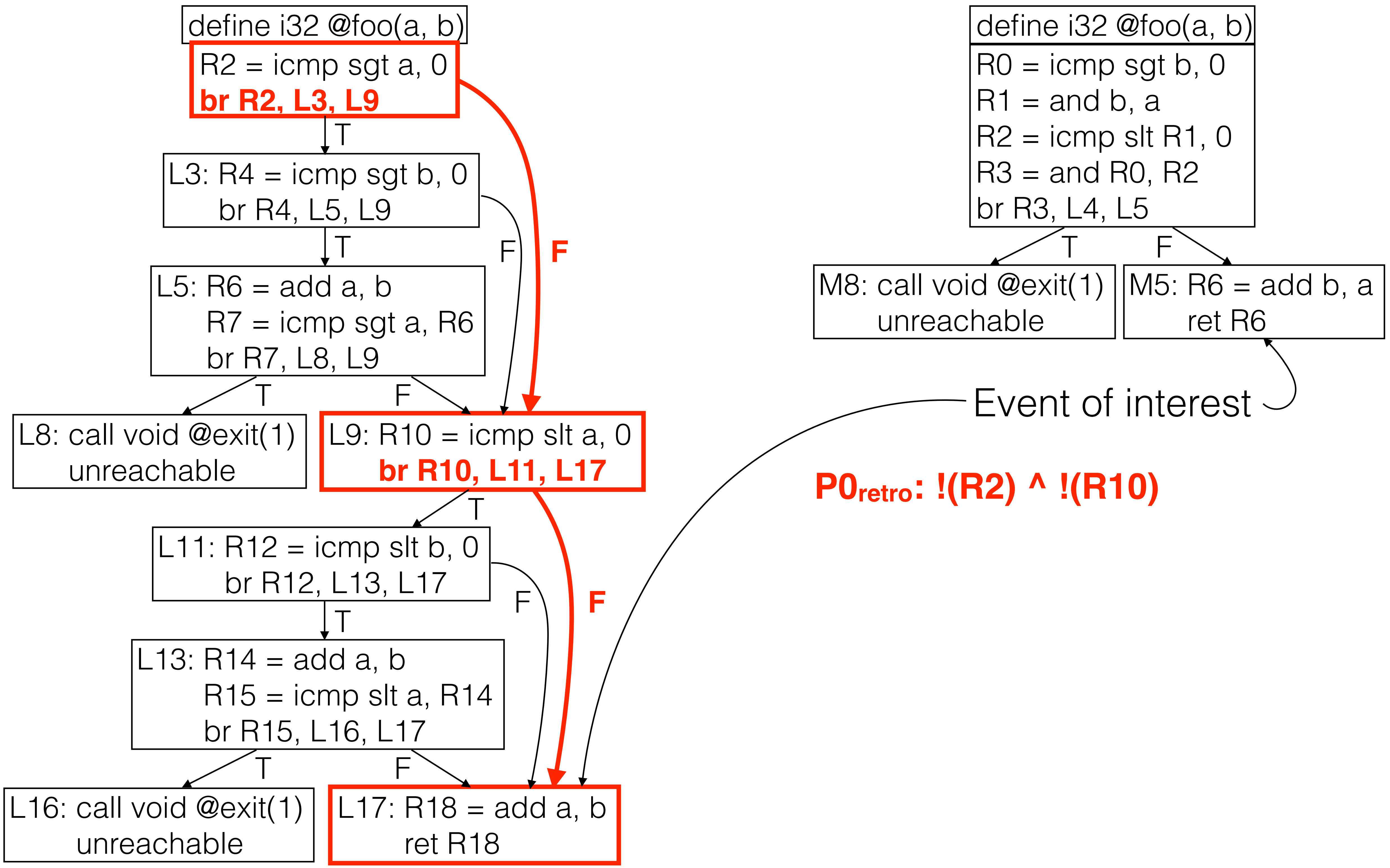
Reachability



Reachability

P_{retro}: retrofitted program

P_{opt}: optimized P_{retro}

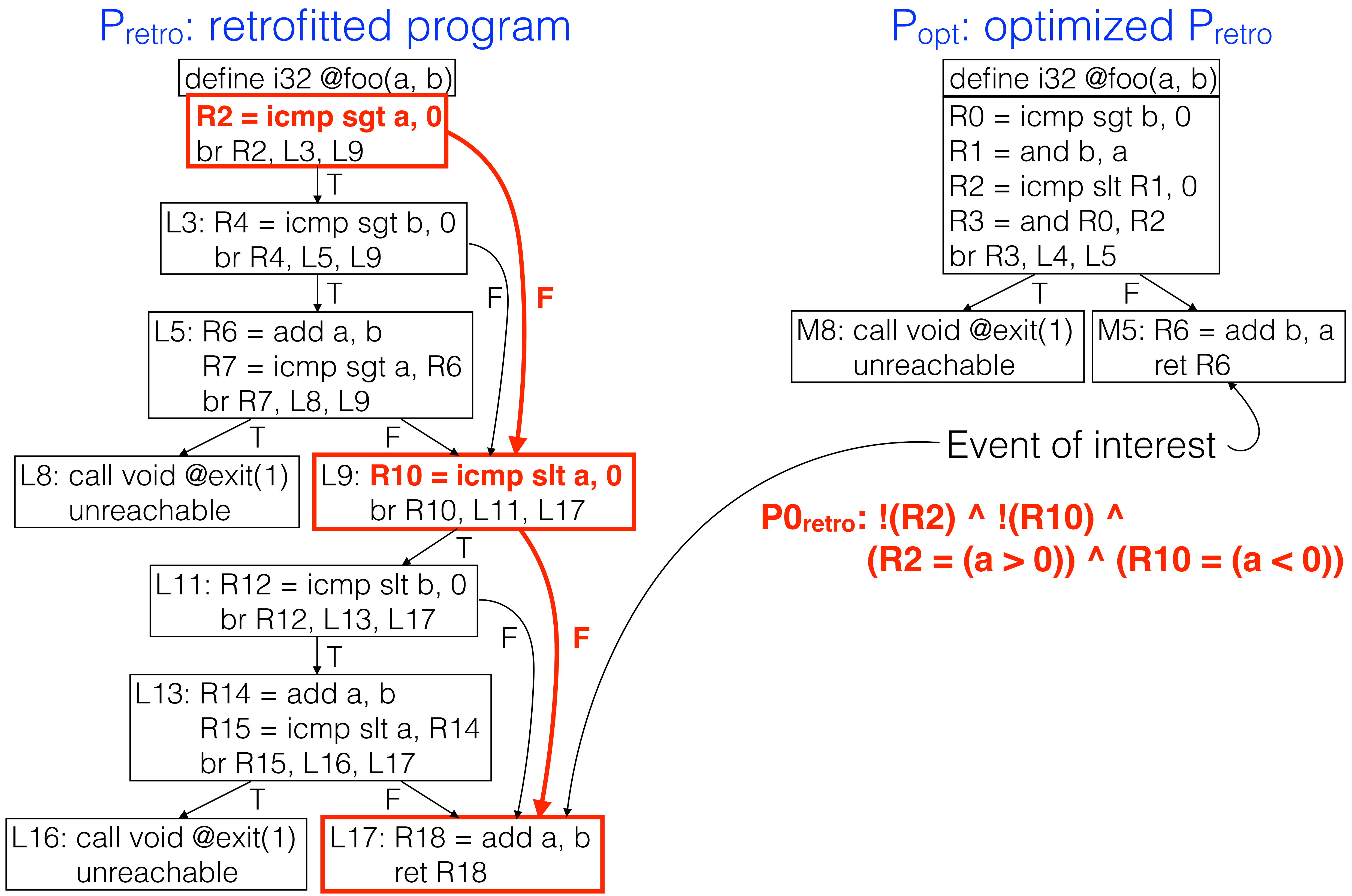


```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
**br R2, L3, L9**

L3: R4 = icmp sgt b, 0
    br R4, L5, L9

L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9

L8: call void @exit(1)
    unreachable

L9: R10 = icmp slt a, 0
    br R10, L11, L17

L11: R12 = icmp slt b, 0
     br R12, L13, L17

L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17

L16: call void @exit(1)
     unreachable

L17: R18 = add a, b
     ret R18
```

```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5

M8: call void @exit(1)
    unreachable

M5: R6 = add b, a
    ret R6
```

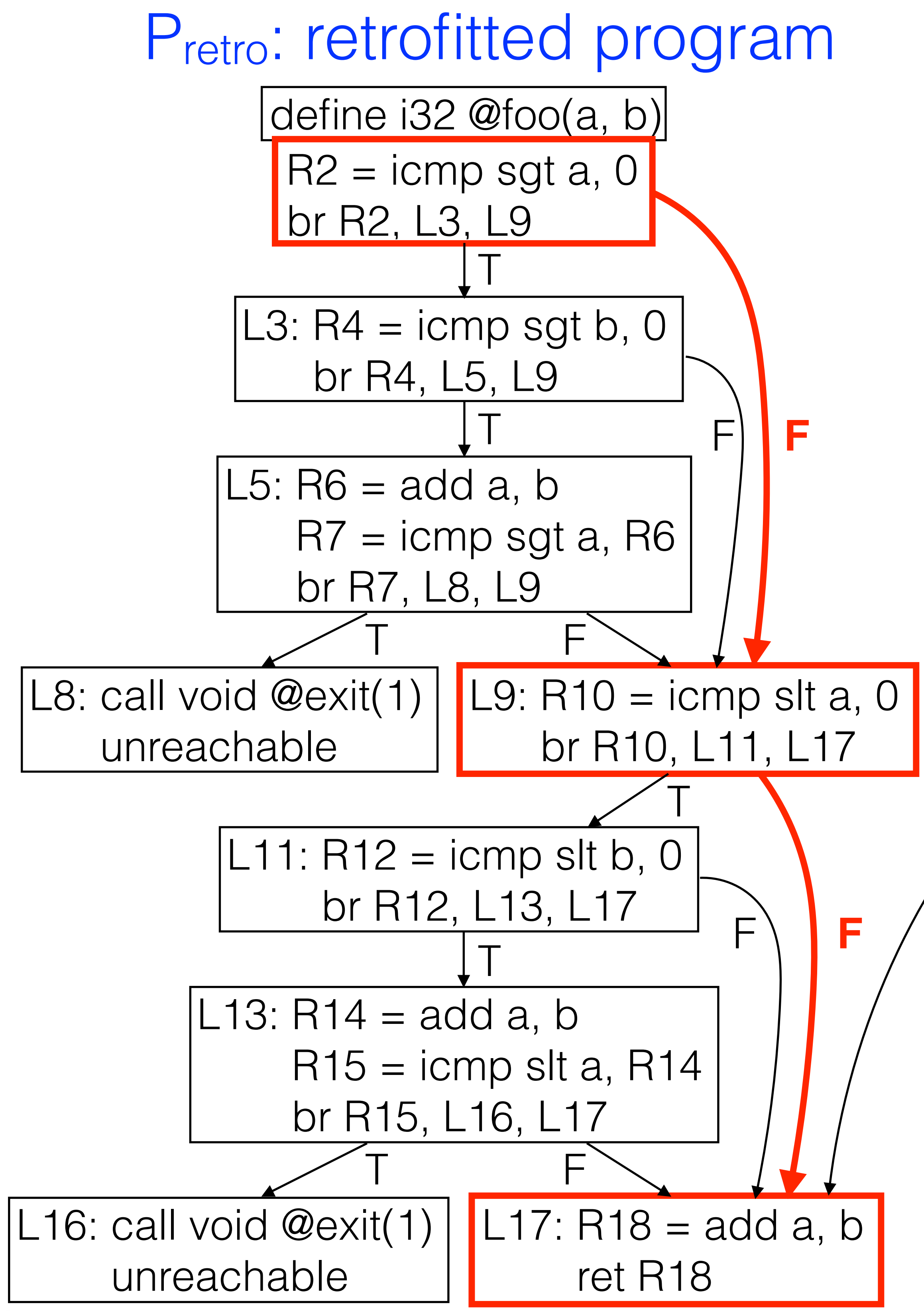
$P0_{retro}$: !(R2) ^ !(R10)

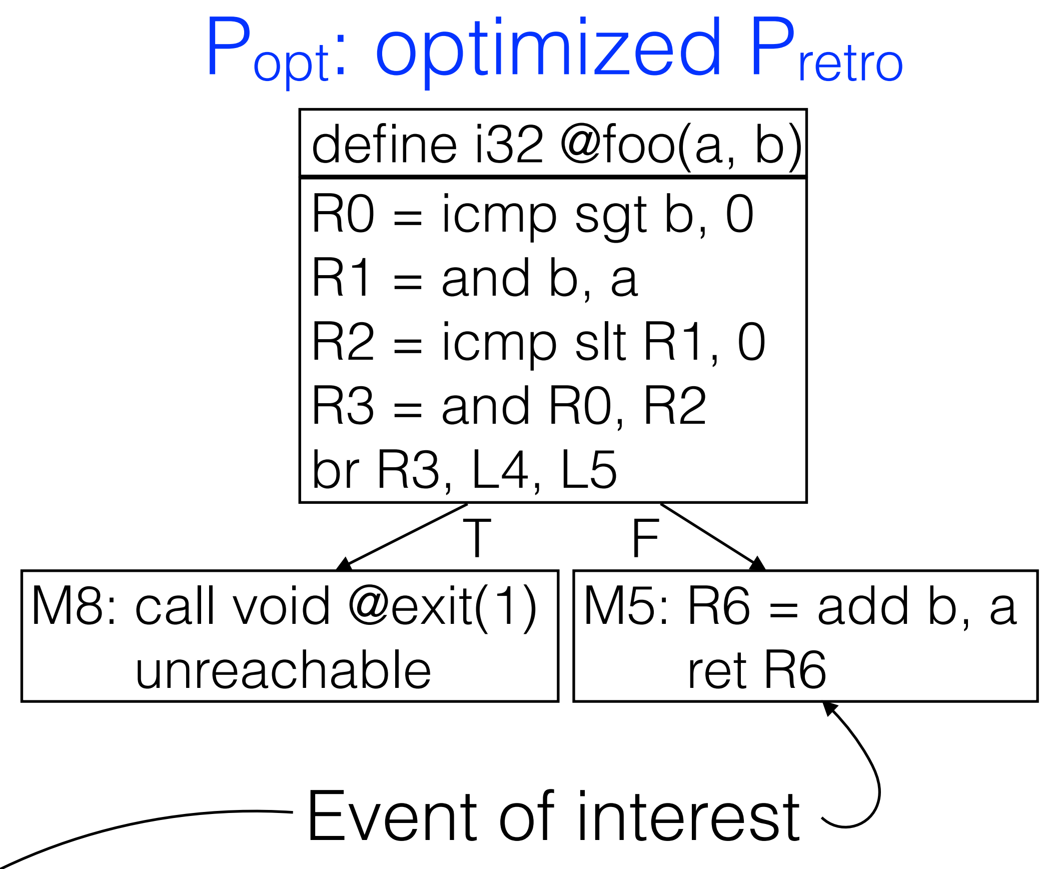
Reachability



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

Reachability

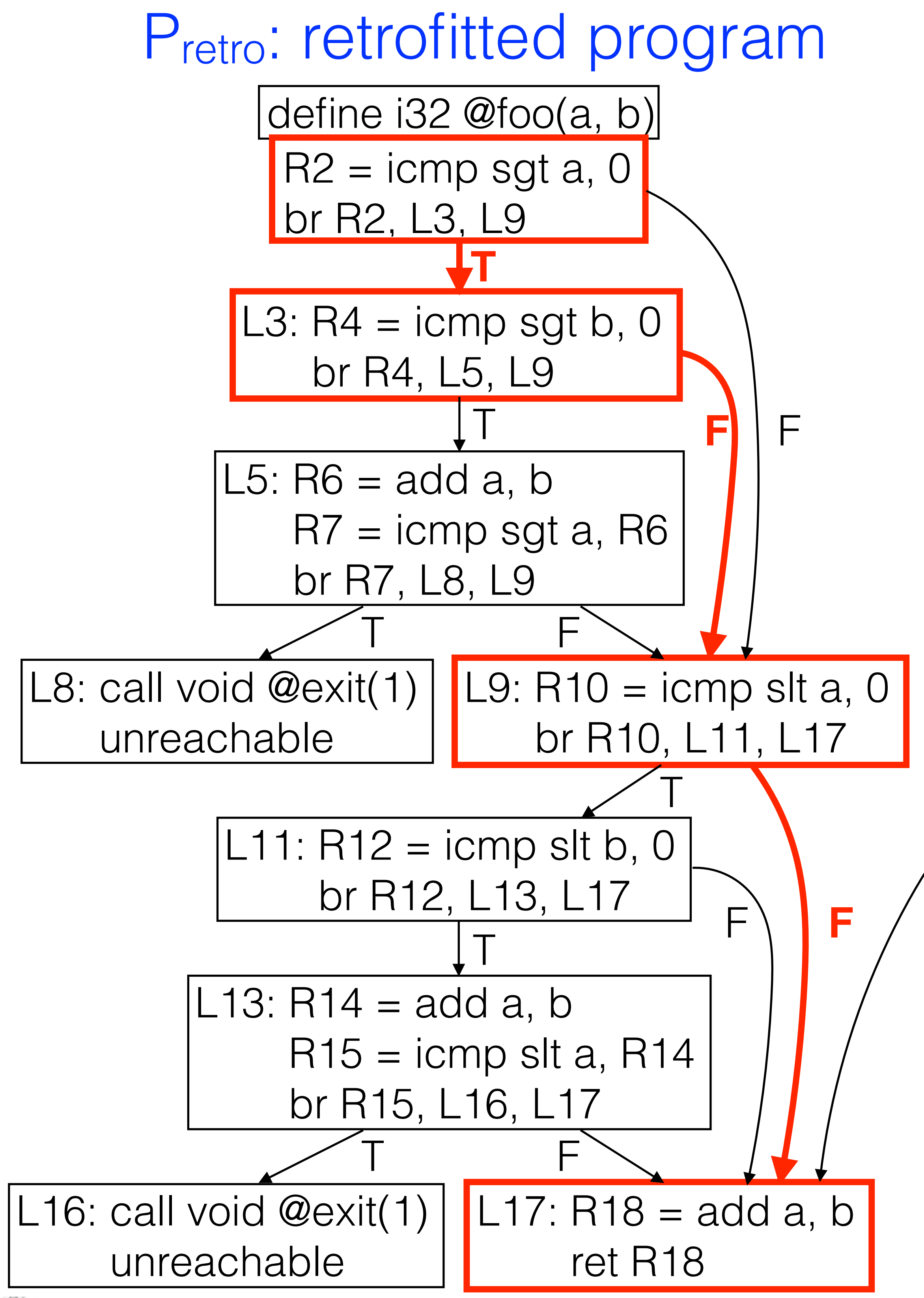


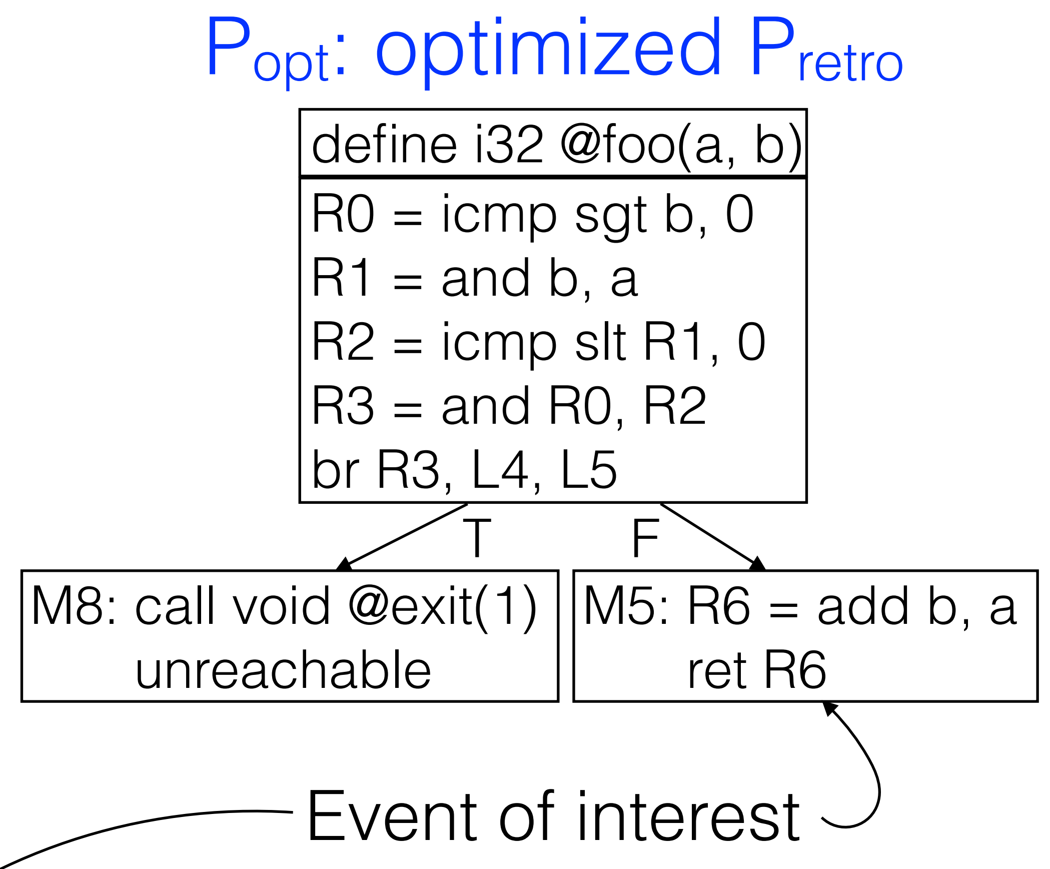


$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

E_{retro} = $P0_{retro}$

Reachability

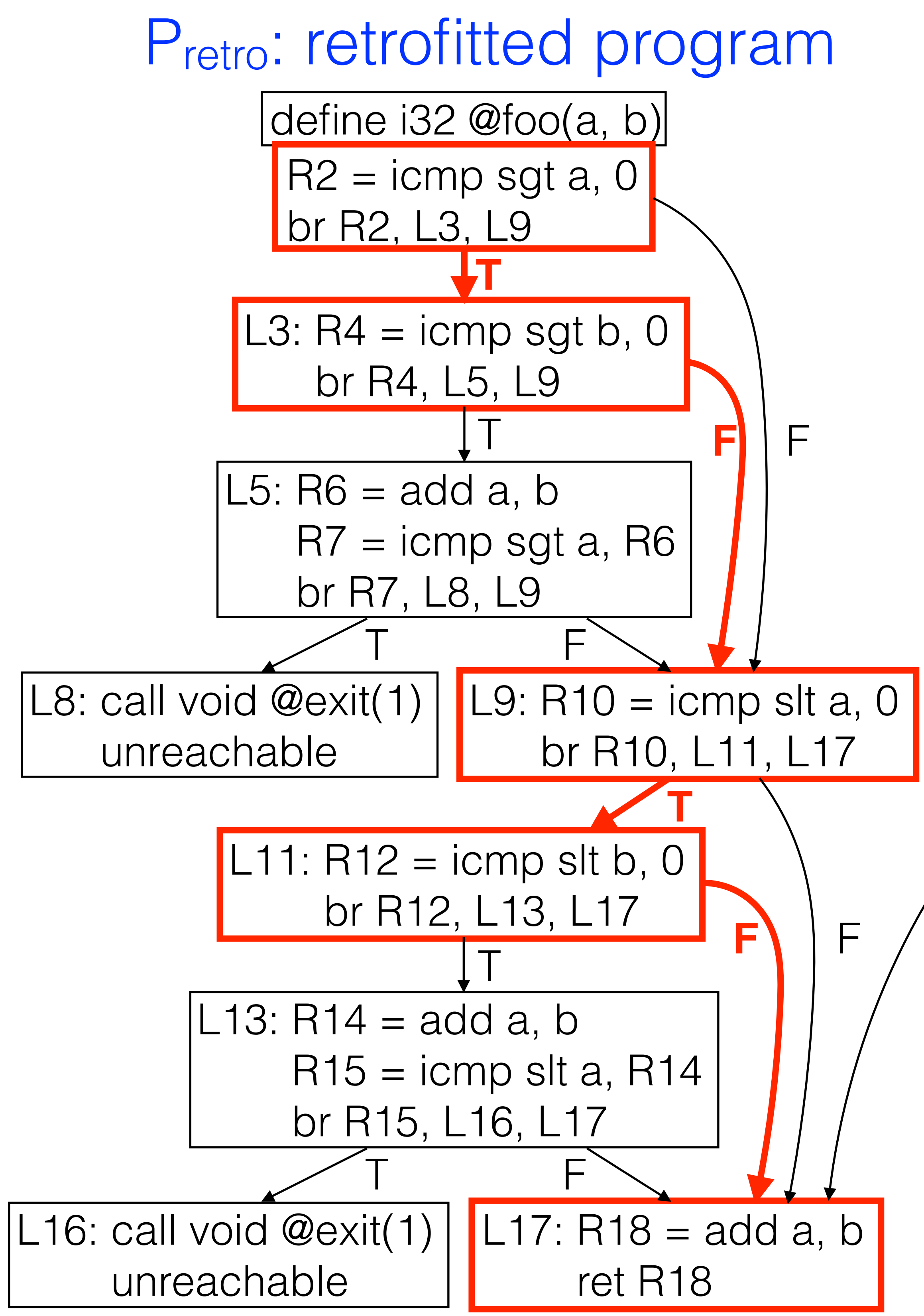


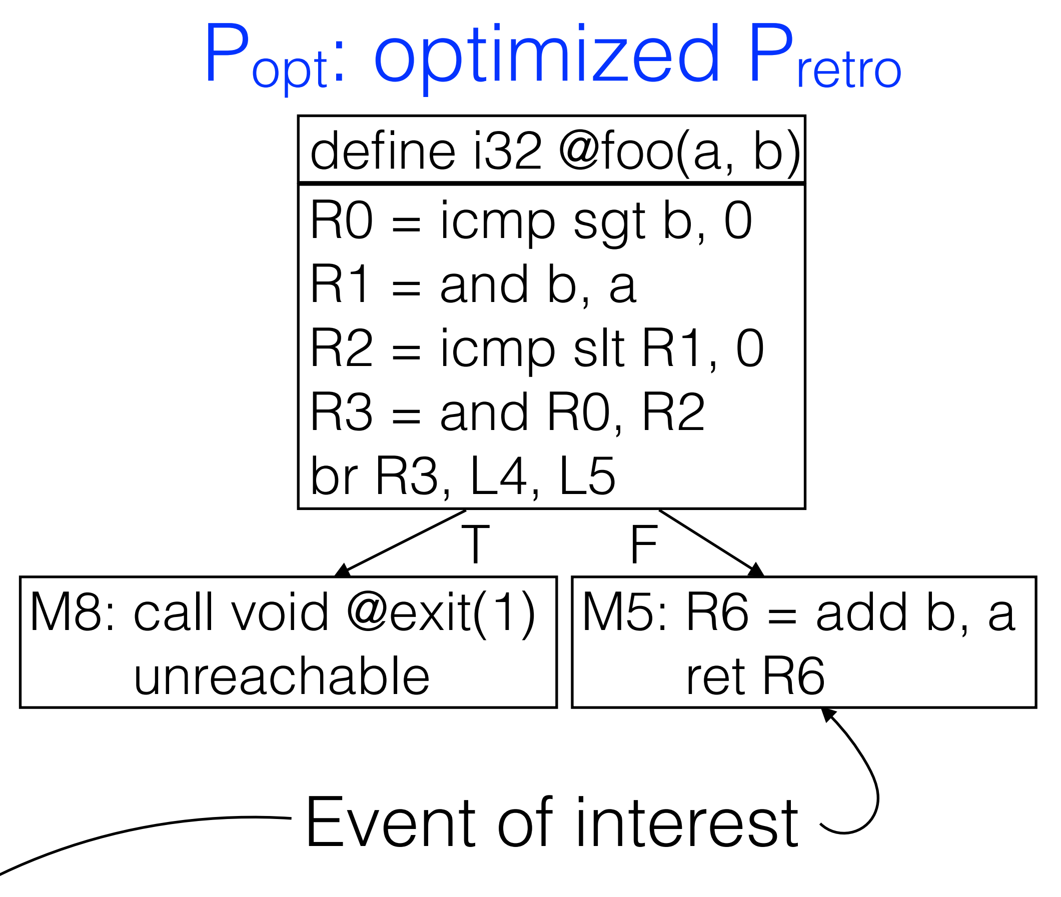


$P0_{retro}$: $!(R2) \wedge !(R10) \wedge$
$(R2 = (a > 0)) \wedge (R10 = (a < 0))$

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

Reachability

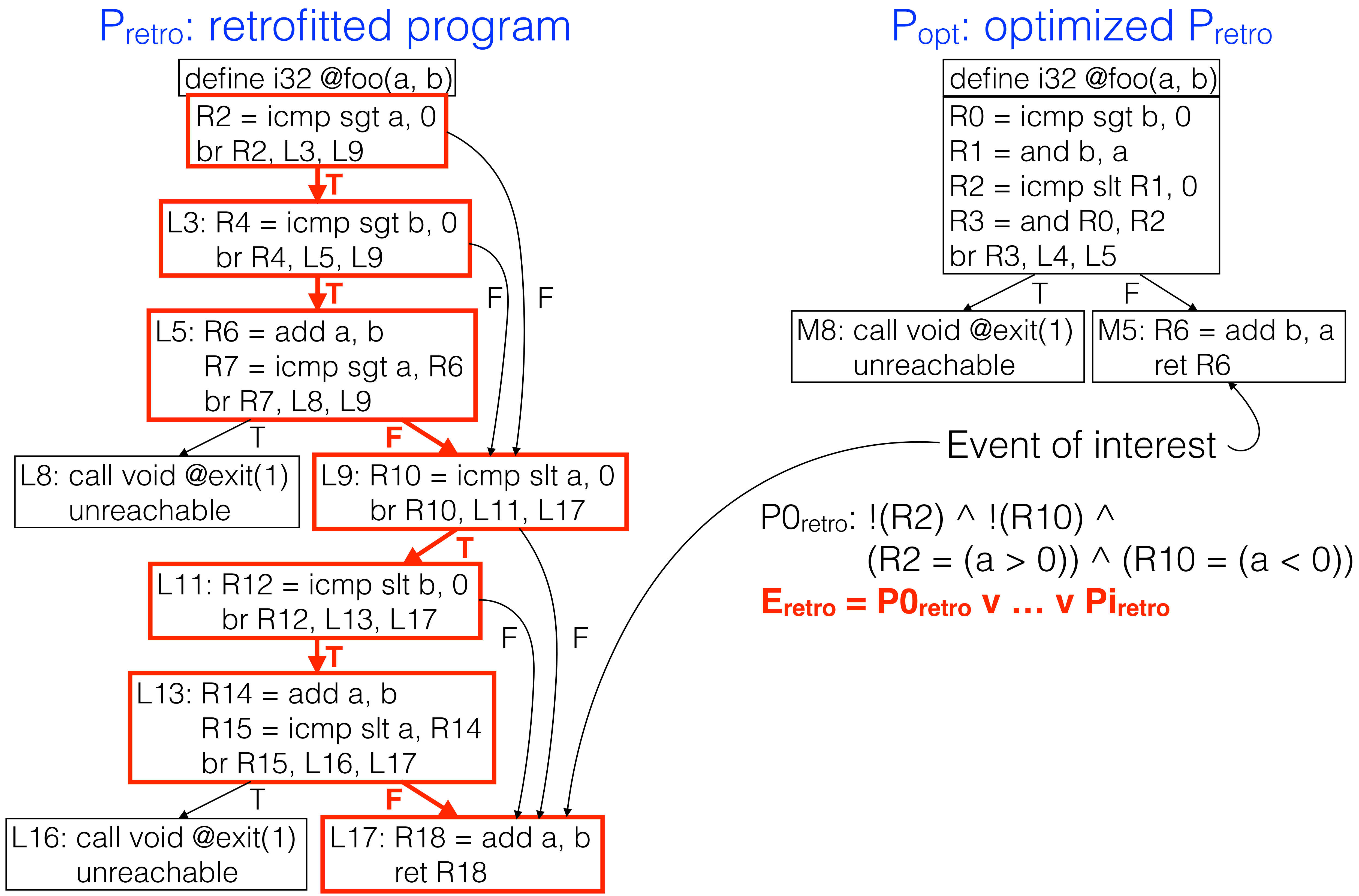




$P0_{retro}$: $!(R2) \wedge !(R10) \wedge$
$(R2 = (a > 0)) \wedge (R10 = (a < 0))$

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

Reachability

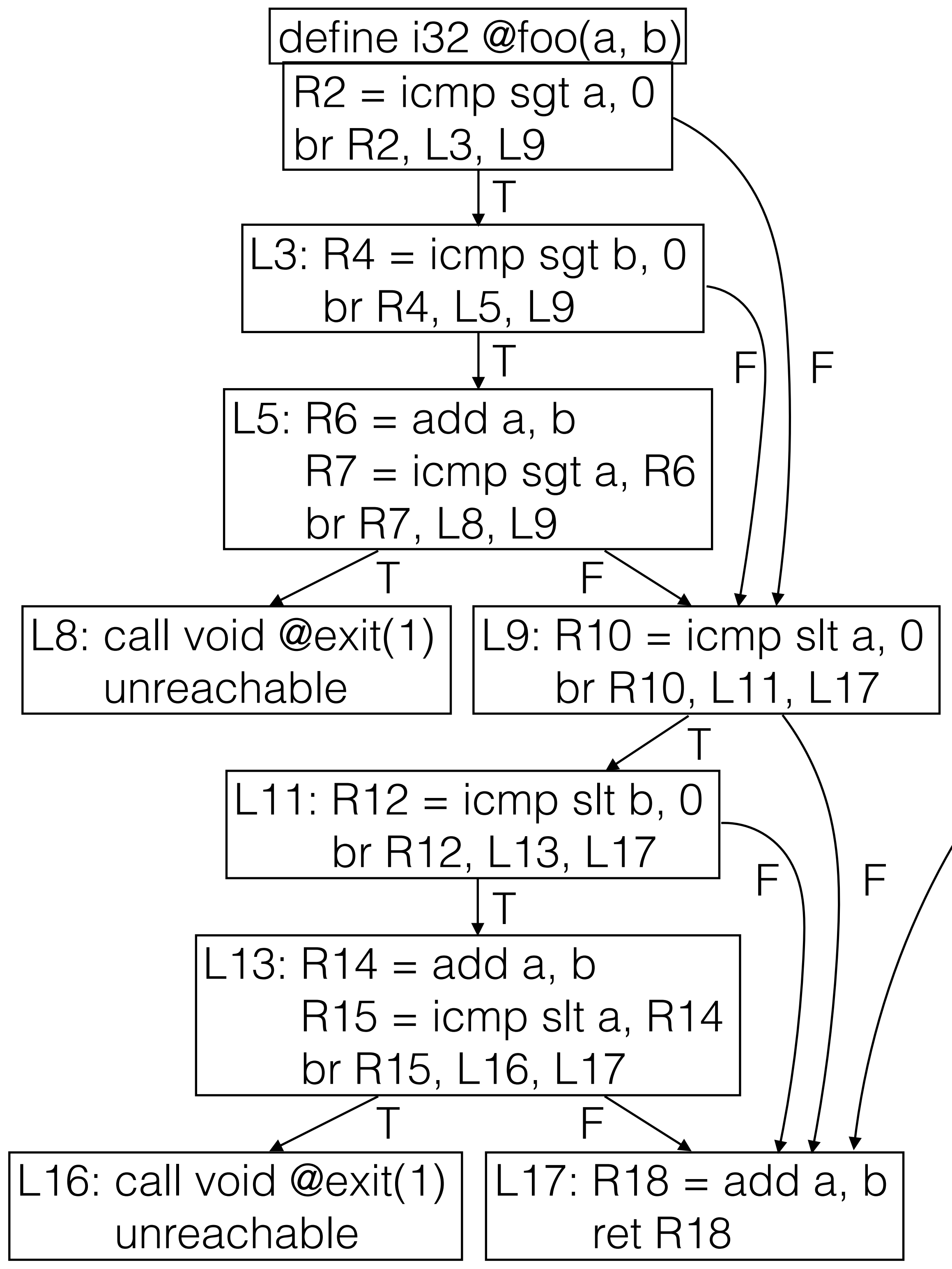


$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

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

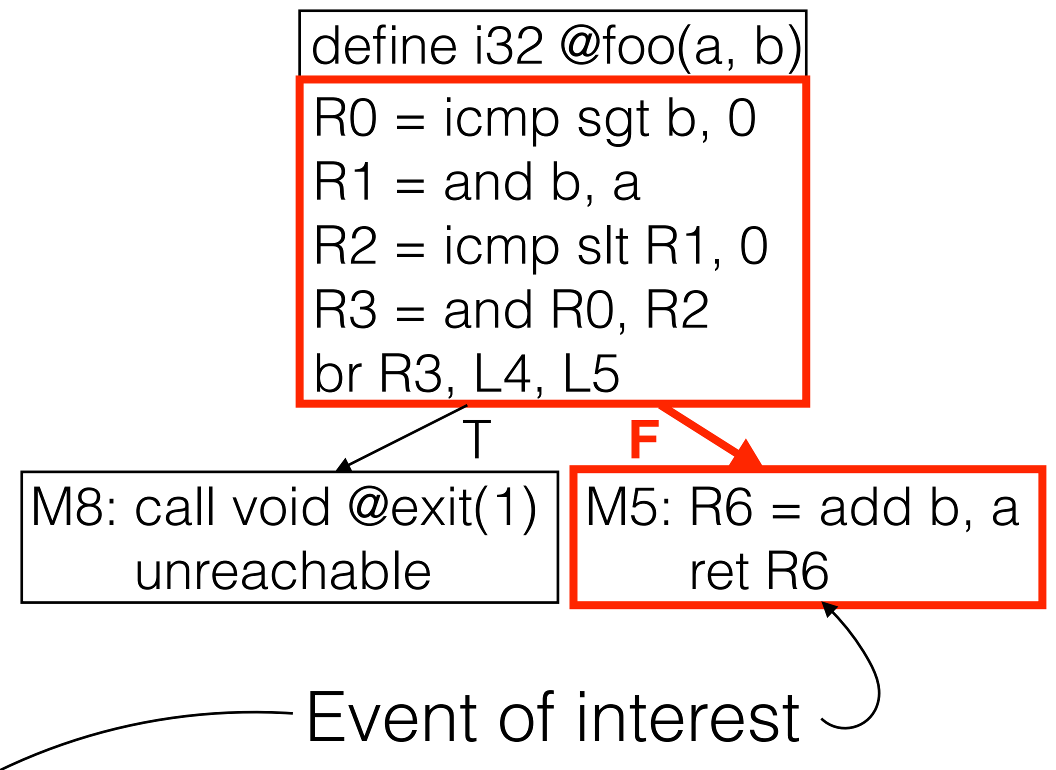
Reachability

P_{retro}: retrofitted program



```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9

L3: R4 = icmp sgt b, 0
    br R4, L5, L9

L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9

L8: call void @exit(1)
    unreachable

L9: R10 = icmp slt a, 0
    br R10, L11, L17

L11: R12 = icmp slt b, 0
     br R12, L13, L17

L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17

L16: call void @exit(1)
     unreachable

L17: R18 = add a, b
     ret R18
```

P_{opt}: optimized P_{retro}



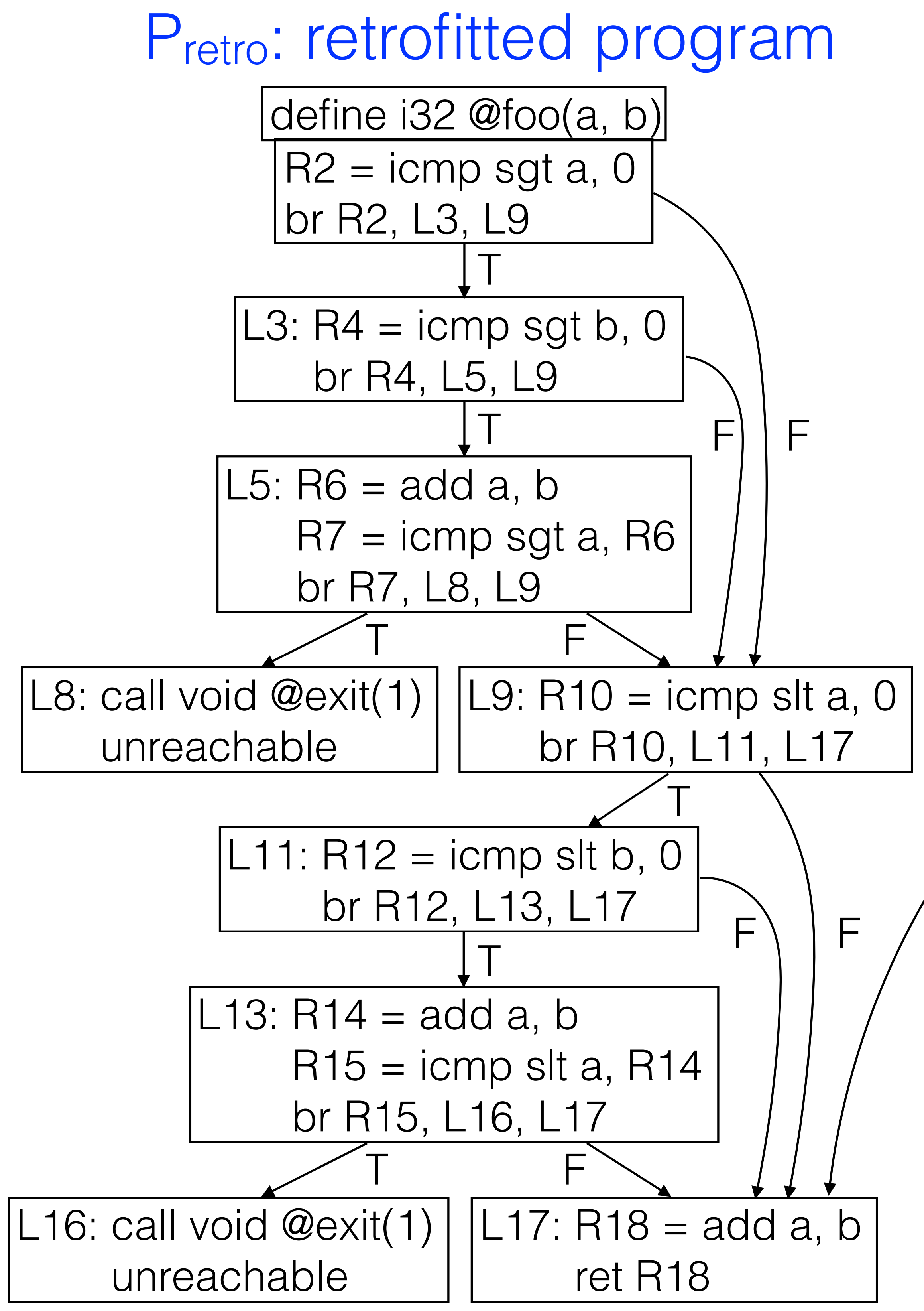
```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5

M8: call void @exit(1)
    unreachable

M5: R6 = add b, a
    ret R6
```

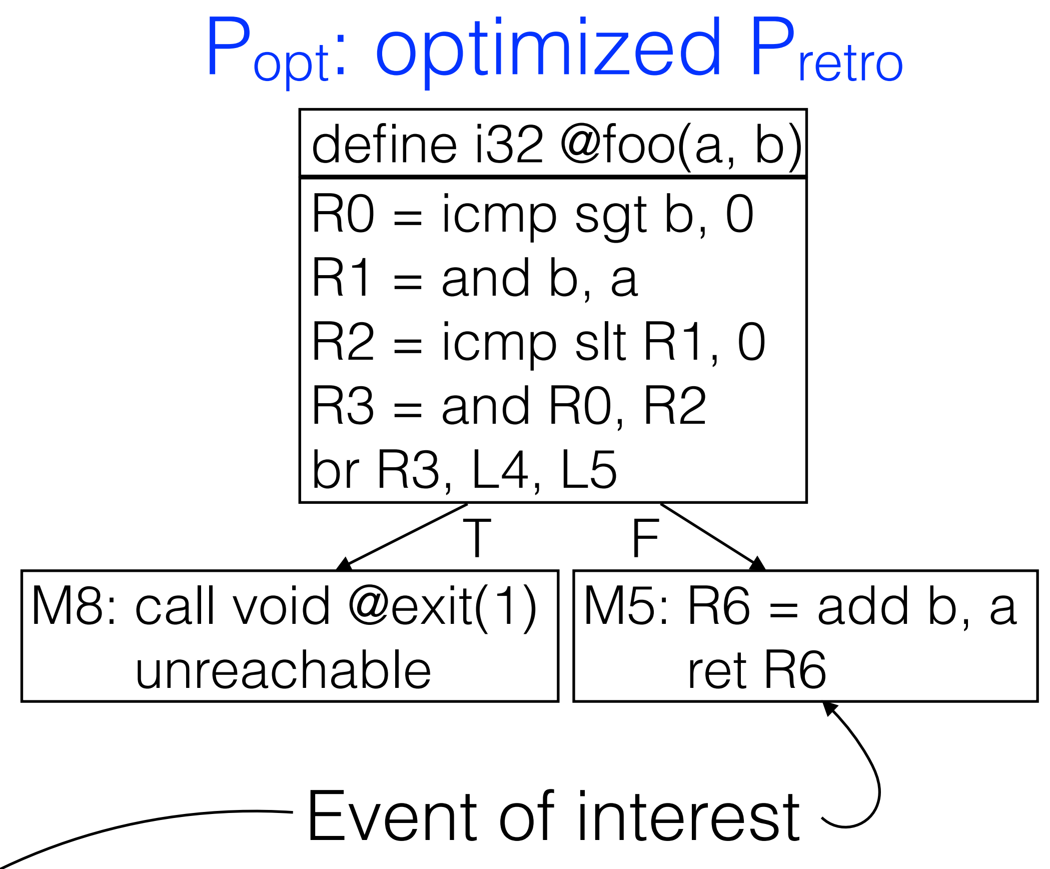
$P0_{retro}$: $!(R2) \wedge !(R10) \wedge$
$(R2 = (a > 0)) \wedge (R10 = (a < 0))$

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

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

Reachability





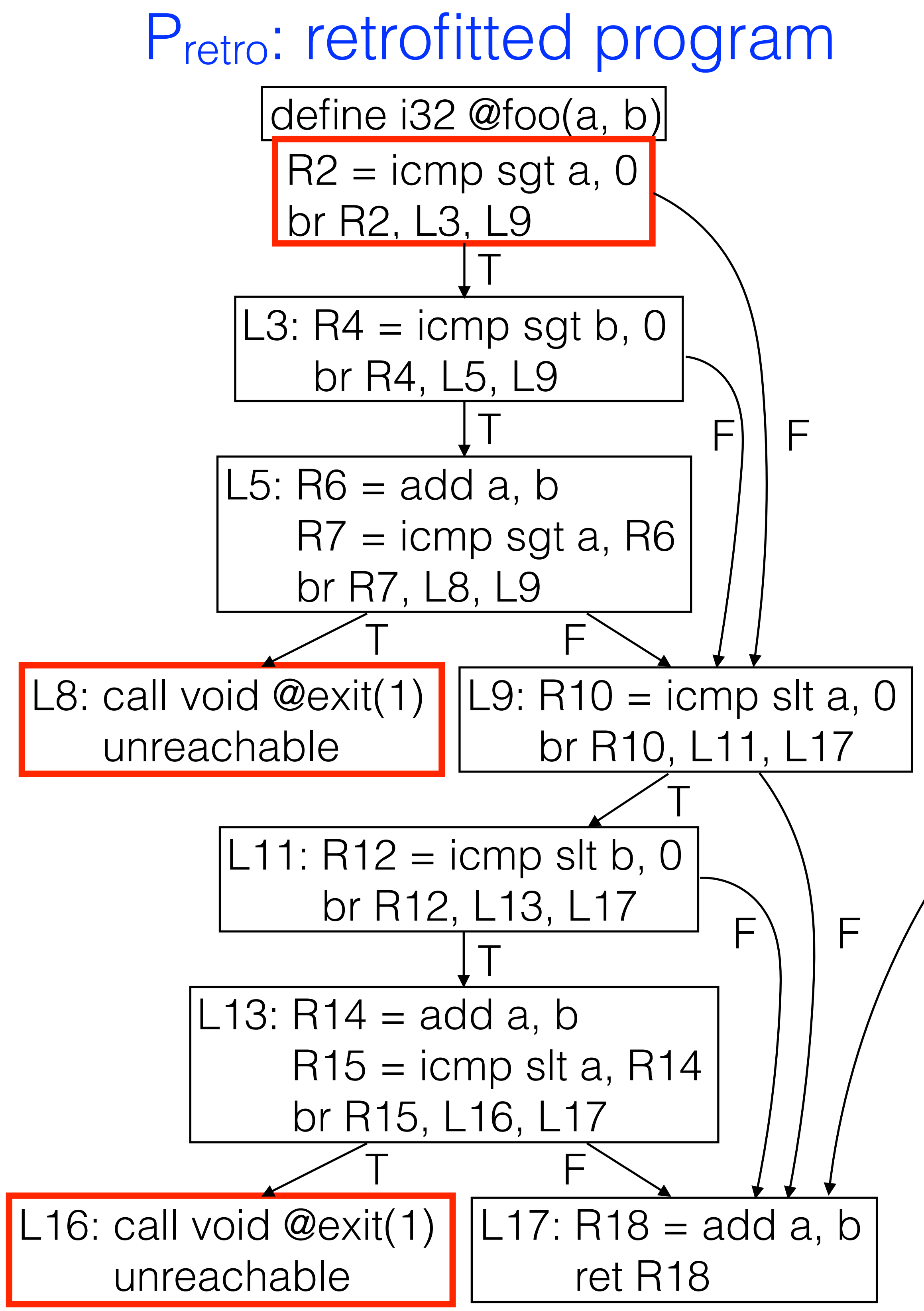
$P0_{retro}$: !(R2) ^ !(R10) ^
(R2 = (a > 0)) ^ (R10 = (a < 0))

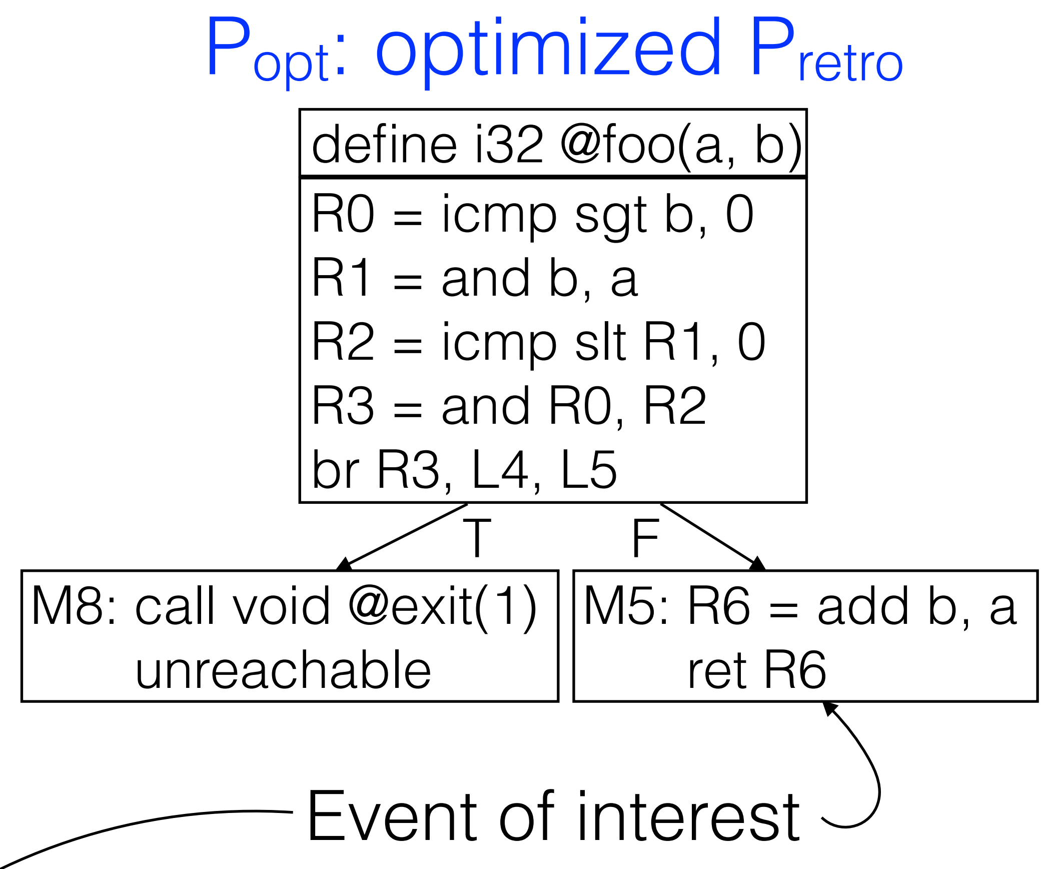
$E_{retro} = P0_{retro} \vee \ldots \vee Pi_{retro}$

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

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

Reachability





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

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

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

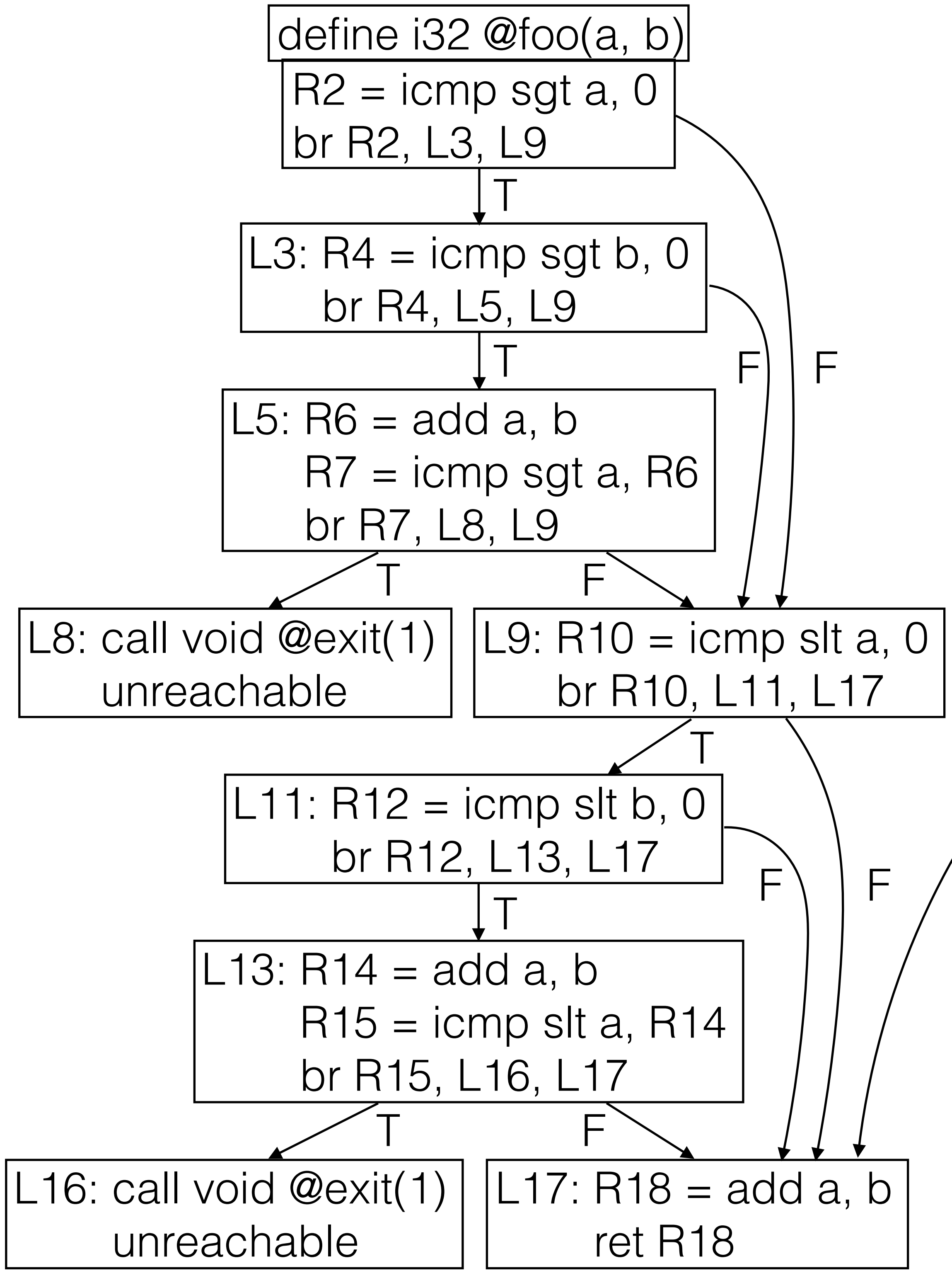
If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} \Rightarrow E_{opt}$)

$\sim E_{retro}$

RUTGERS

Reachability

P_{retro}: retrofitted program



```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9
```

```
L3: R4 = icmp sgt b, 0
    br R4, L5, L9
```

```
L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9
```

```
L8: call void @exit(1)
    unreachable
```

```
L9: R10 = icmp slt a, 0
    br R10, L11, L17
```

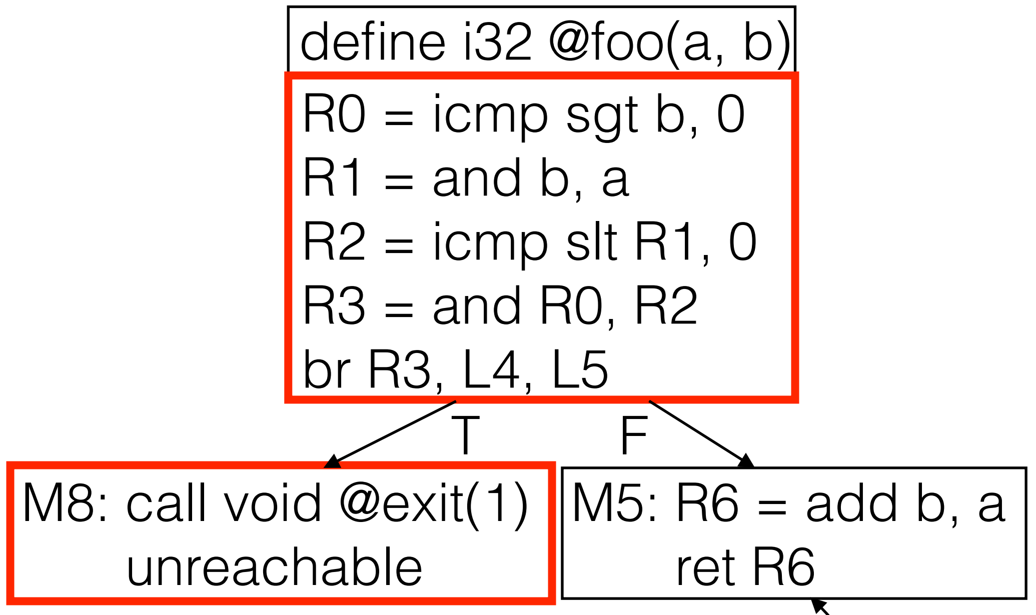
```
L11: R12 = icmp slt b, 0
     br R12, L13, L17
```

```
L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17
```

```
L16: call void @exit(1)
     unreachable
```

```
L17: R18 = add a, b
     ret R18
```

P_{opt}: optimized P_{retro}



```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5
```

```
M8: call void @exit(1)
    unreachable
```

```
M5: R6 = add b, a
    ret R6
```

Event of interest

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

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

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

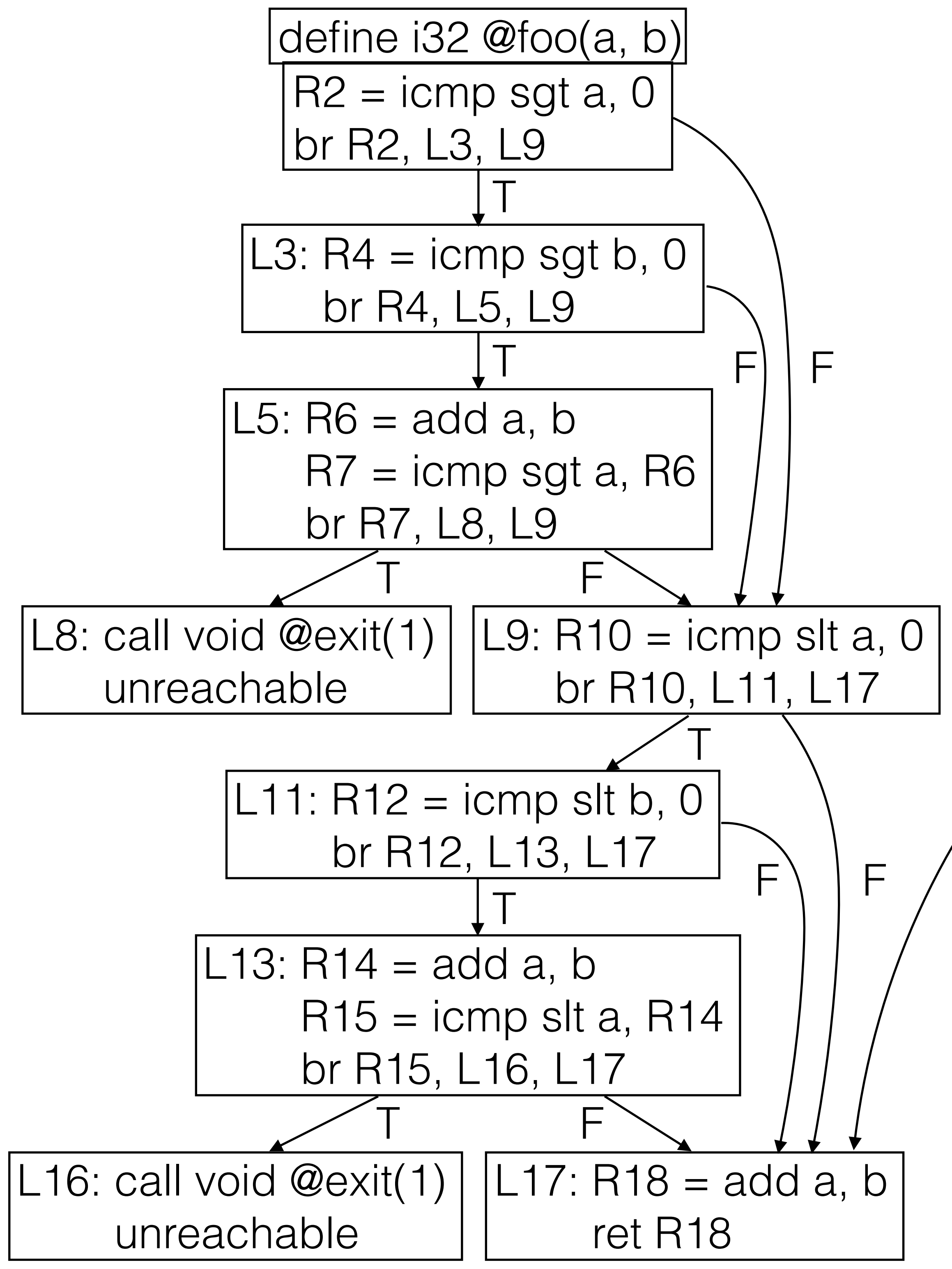
If P_{retro} reaches event of interest, then P_{opt} reaches event of interest ($E_{retro} \Rightarrow E_{opt}$)

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

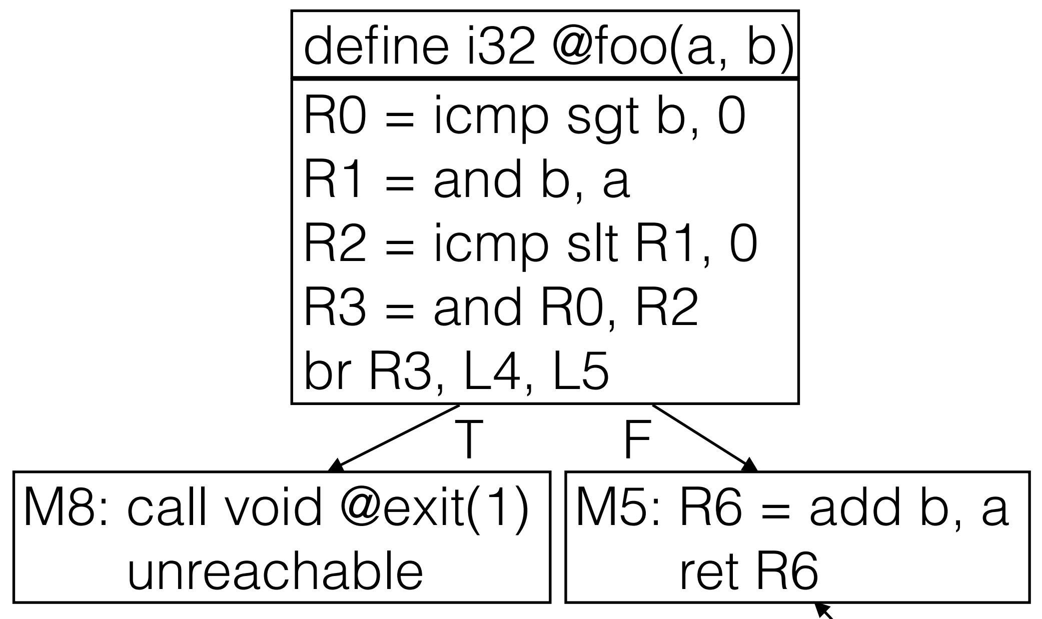
RUTGERS

Reachability

P_{retro}: retrofitted program



P_{opt}: optimized P_{retro}



Event of interest

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

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

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

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

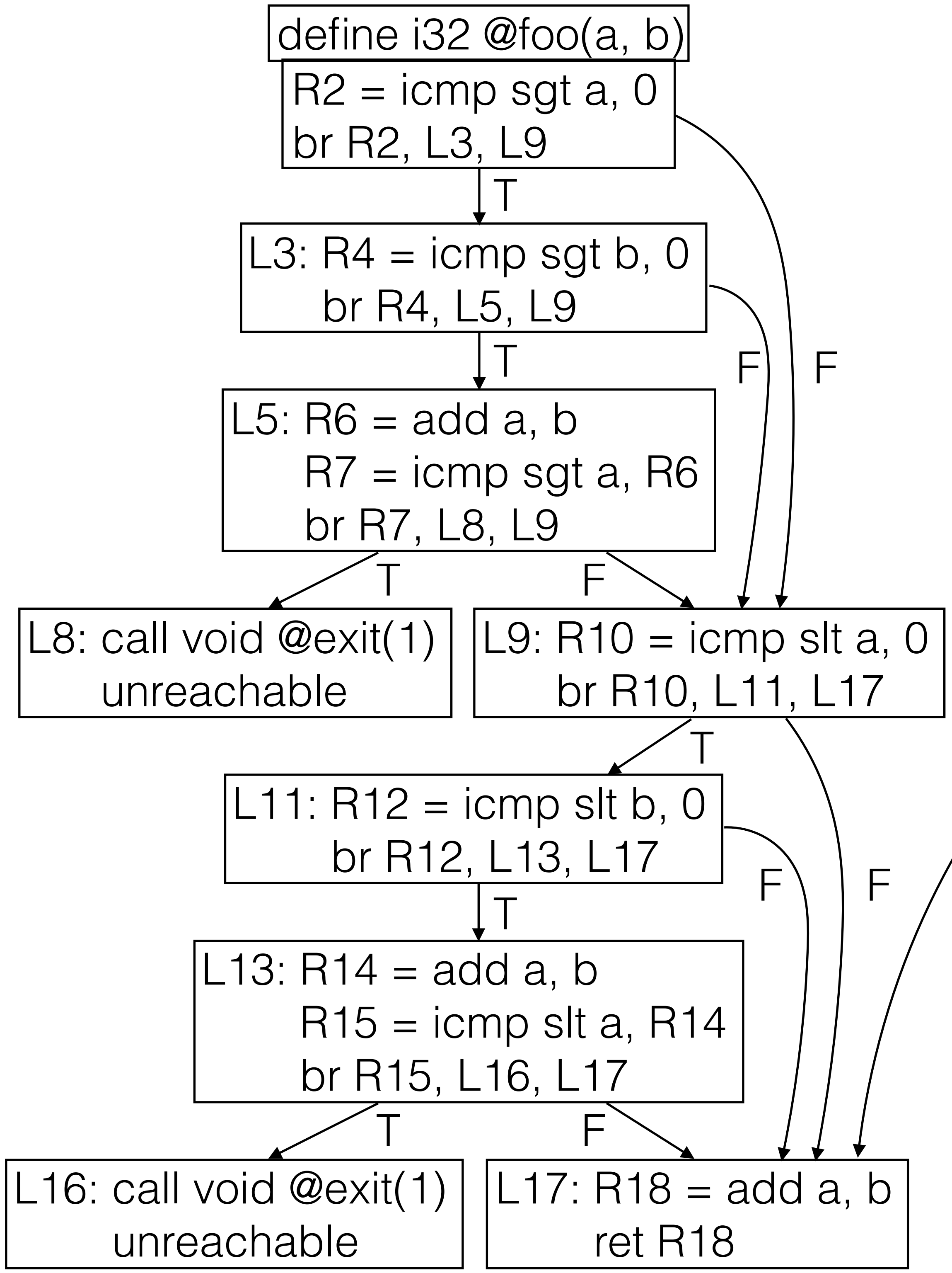
$\sim E_{retro}$, $\sim E_{opt}$

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

RUTGERS

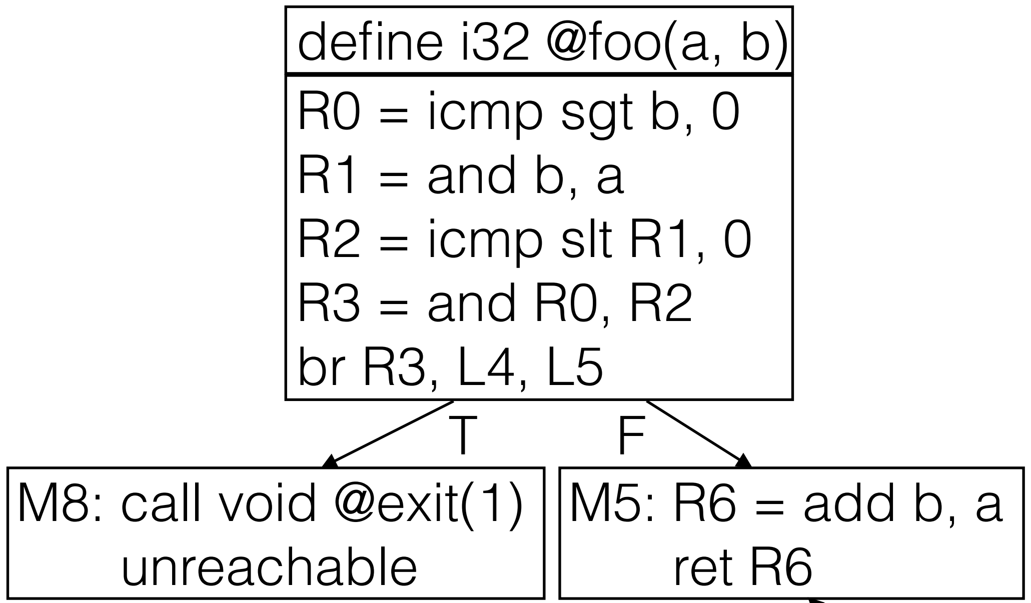
Reachability

P_{retro}: retrofitted program



```
define i32 @foo(a, b)
R2 = icmp sgt a, 0
br R2, L3, L9

L3: R4 = icmp sgt b, 0
    br R4, L5, L9

L5: R6 = add a, b
    R7 = icmp sgt a, R6
    br R7, L8, L9

L8: call void @exit(1)
    unreachable

L9: R10 = icmp slt a, 0
    br R10, L11, L17

L11: R12 = icmp slt b, 0
     br R12, L13, L17

L13: R14 = add a, b
     R15 = icmp slt a, R14
     br R15, L16, L17

L16: call void @exit(1)
     unreachable

L17: R18 = add a, b
     ret R18
```

P_{opt}: optimized P_{retro}



```
define i32 @foo(a, b)
R0 = icmp sgt b, 0
R1 = and b, a
R2 = icmp slt R1, 0
R3 = and R0, R2
br R3, L4, L5

M8: call void @exit(1)
    unreachable

M5: R6 = add b, a
    ret R6
```

Event of interest

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

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

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

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 ($E_{opt} \Rightarrow E_{retro}$)

Evaluation

- Built prototype for LLVM IR programs.
- Modified retrofitting transformation to mark event of interests.
- Z3 for query.
- Naive integer overflow checker, Address Sanitizer, SoftboundCETS
- Scalability optimizations

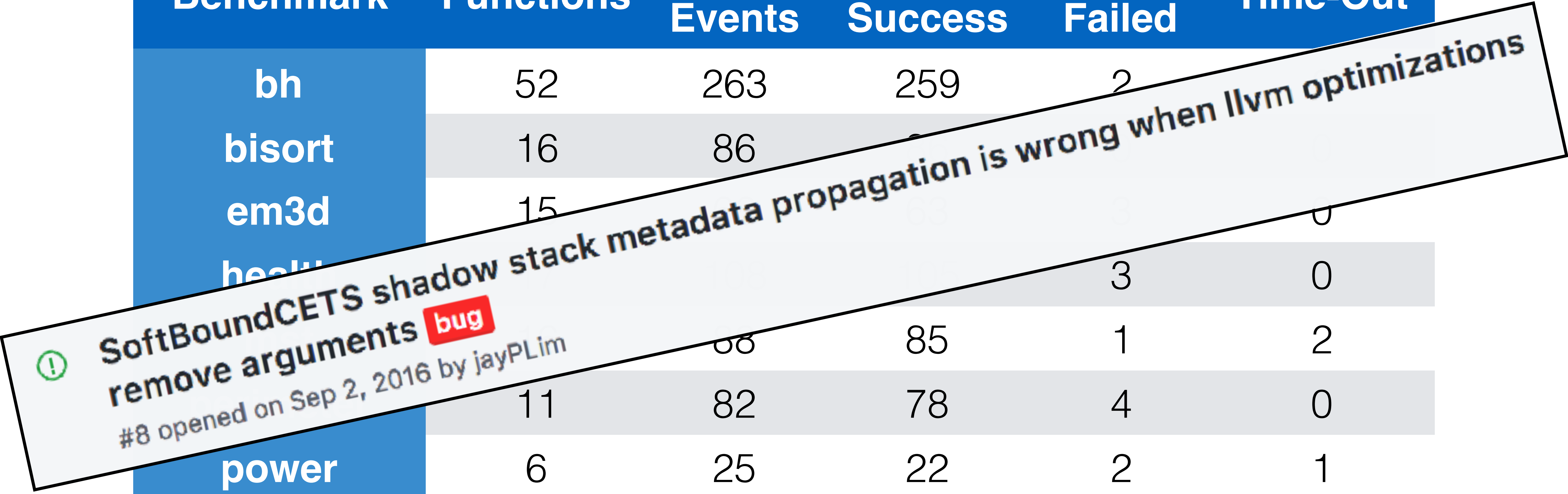
Address Sanitizer

Benchmark	Functions	Total Events	Check Success	Check Failed	Time-Out
bh	38	135	121	9	5
bisort	7	20	18	1	1
em3d	9	27	21	4	2
health	13	41	37	3	1
mst	10	14	11	2	1
perimeter	6	28	27	1	0
power	13	56	48	2	6
treeadd	6	8	6	2	0
tsp	9	19	16	2	1

RUTGERS

SoftBoundCETS

Benchmark	Functions	Total Events	Check Success	Check Failed	Time-Out
bh	52	263	259	2	[illegible]
bisort	16	86	[illegible]	[illegible]	[illegible]
em3d	15	[illegible]	63	3	0
health	[illegible]	105	[illegible]	3	0
mst	[illegible]	88	85	1	2
perimeter	11	82	78	4	0
power	6	25	22	2	1
treeadd	8	43	43	0	0
tsp	13	84	84	0	0


SoftBoundCETS shadow stack metadata propagation is wrong when llvm optimizations remove arguments bug

#8 opened on Sep 2, 2016 by jayPLim

RUTGERS

Conclusion

- Optimization do erroneously remove checks.
- Formulating as Reachability detects errors.
- Must address false positives.
  - better memory axioms.

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp ll cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp ll cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp ll cmp1
  br or.cond.i abort_label next_label
```

RUTGERS

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* pwidth, i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
  R8 = call __softboundcets_get_global_lock()
   …
  cmp = (ptype < R0)
  cmp1 = (ptype[1] > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
   …
  cmp = (ptype < R0)
  cmp1 = (ptype > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
   …
  cmp = (ptype < R0)
  cmp1 = (ptype > R1)
  or.cond.i = cmp ll cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
   …
  cmp = (ptype < R0)
  cmp1 = (ptype > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
   …
  cmp = (ptype < R0)
  cmp1 = (ptype > R1)
  or.cond.i = cmp || cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
   …
  cmp = (ptype < R0)
  cmp1 = (ptype > R1)
  or.cond.i = cmp ll cmp1
  br or.cond.i abort_label next_label
```

SoftboundCETS bug

```
define void quantum_qec_get_status(i32* ptype) {
entry:
  R0 = call __softboundcets_load_base_shadow_stack(2)
  R1 = call __softboundcets_load_bound_shadow_stack(2)
  R2 = call __softboundcets_load_key_shadow_stack(2)
  R3 = call __softboundcets_load_lock_shadow_stack(2)
  R4 = call __softboundcets_load_base_shadow_stack(1)
  R5 = call __softboundcets_load_bound_shadow_stack(1)
  R6 = call __softboundcets_load_key_shadow_stack(1)
  R7 = call __softboundcets_load_lock_shadow_stack(1)
   …
  cmp = (ptype < R0)
  cmp1 = (ptype > R1)
  or.cond.i = cmp ll cmp1
  br or.cond.i abort_label next_label
```

Related Work

- Translation Validation
  - Checks for equivalence of semantics in code before and after optimization.
  - Heuristics may cause false positives.
- SymDiff
  - Checks for equivalence of two C programs using HAVOC and Boogie.
  - We are interested in LLVM IR programs.
- STACK
  - Detects instrumentations removed due to undefined behavior (unstable code).
  - Does not detect all removal of instrumentations.