Luxembourg National Research Fund

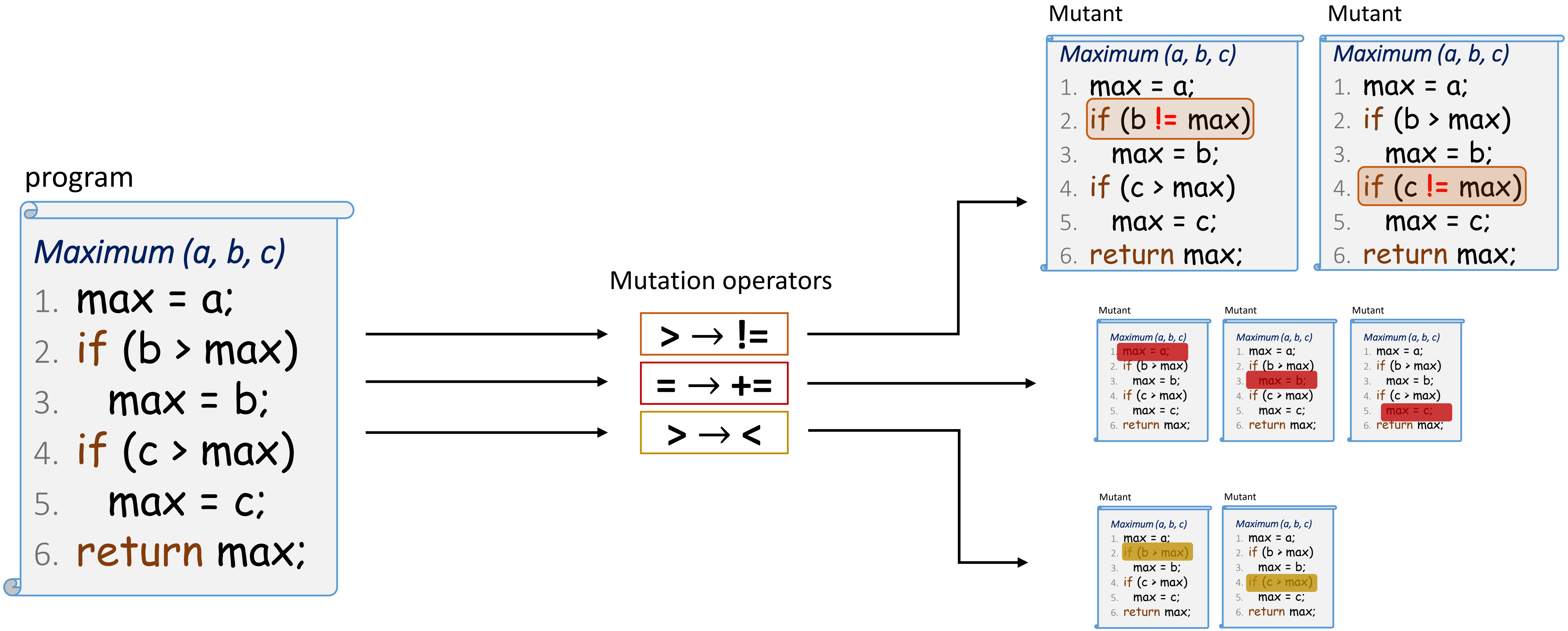
Killing Stubborn Mutants with Symbolic Execution

Thierry Titcheu Chekam Mike Papadakis Maxime Cordy Yves Le Traon

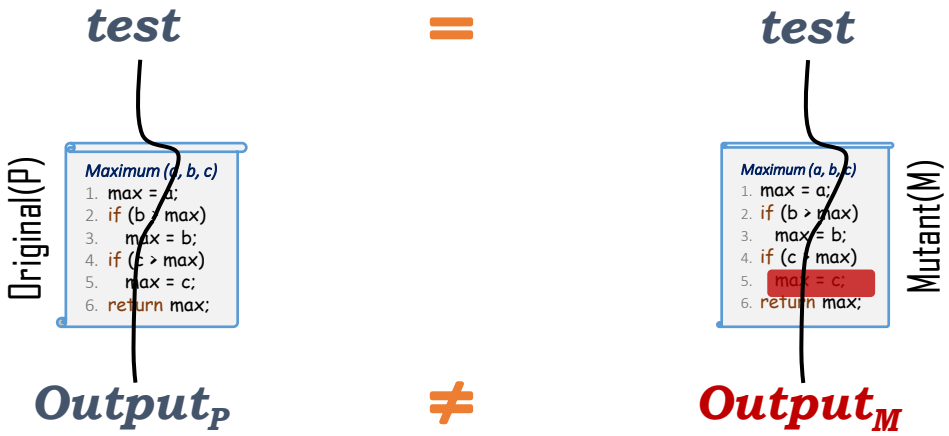
ACM Transactions on Software Engineering and Methodology (TOSEM) Journal

KLEE Workshop 2021

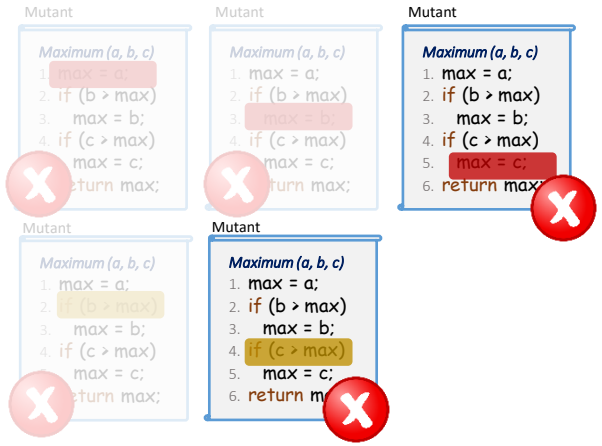
Mutation


program
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutation operators
> → !=
= → +=
> → <
Mutant
Maximum (a, b, c)
1. max = a;
2. if (b != max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutant
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c != max)
5. max = c;
6. return max;
Mutant
Mutant
Mutant
Mutant
Mutant

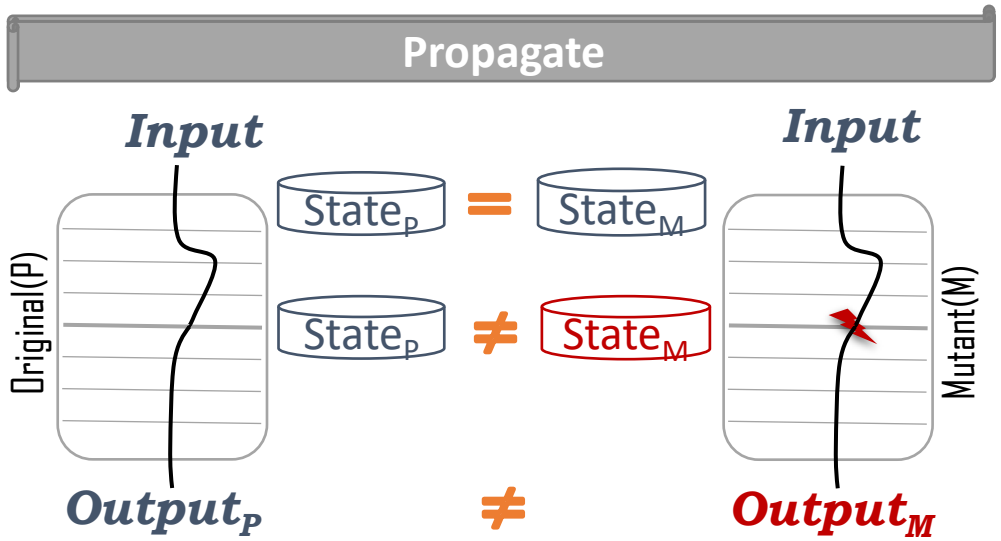
Mutation


test
=
test
Original(P)
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutant(M)
Output_P
≠
Output_M

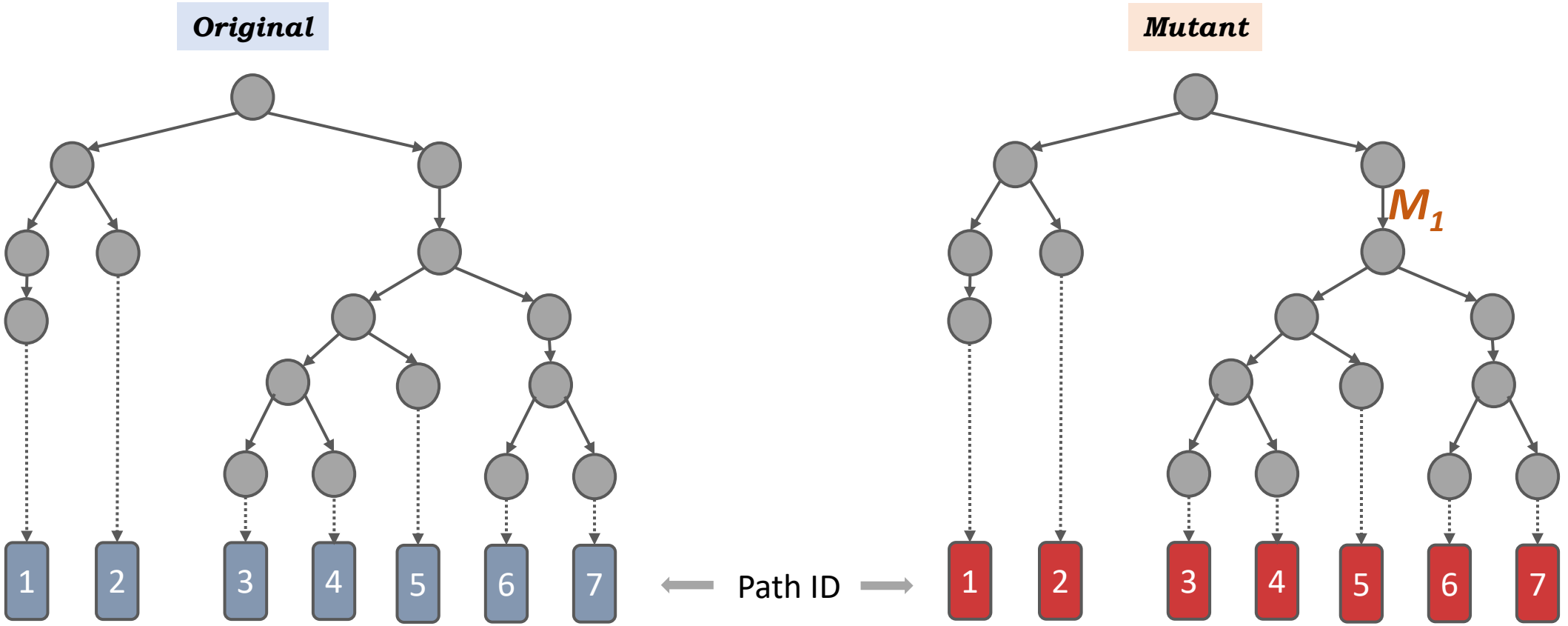

Existing
Test suite
Generated
tests


Mutant
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutant
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutant
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutant
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;
Mutant
Maximum (a, b, c)
1. max = a;
2. if (b > max)
3. max = b;
4. if (c > max)
5. max = c;
6. return max;

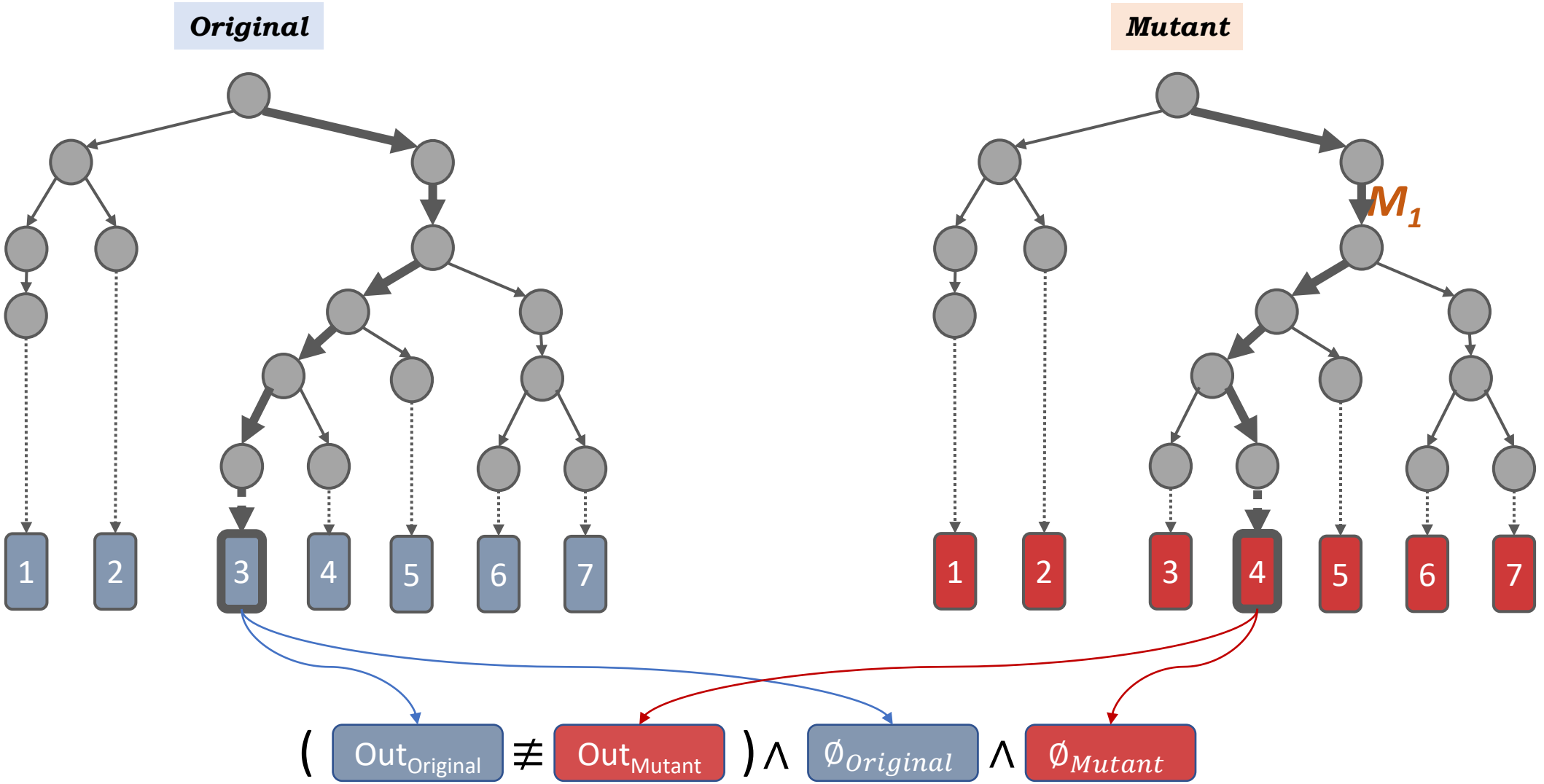
Mutant Killing



Symbolic Mutant Test Generation

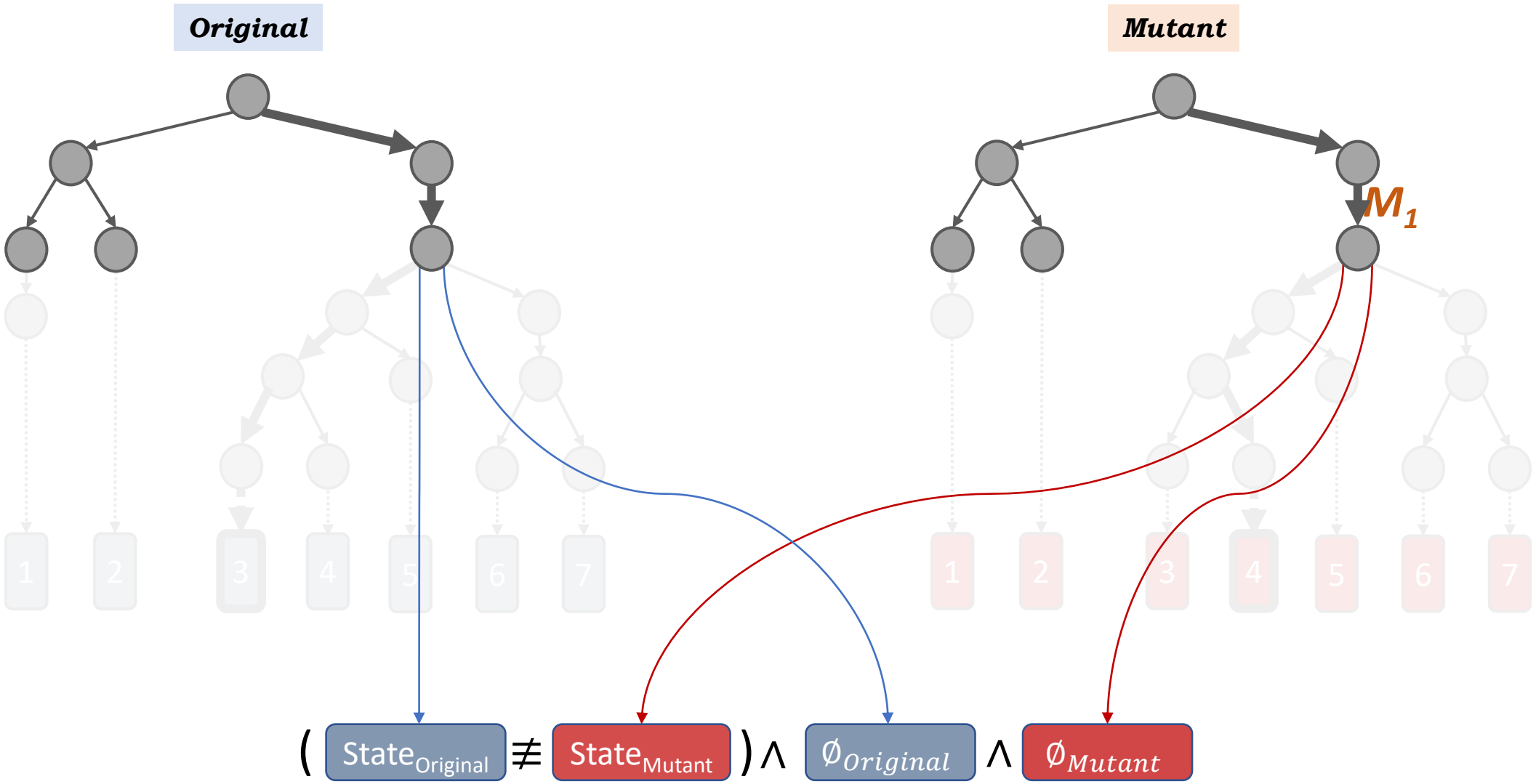


Symbolic Mutant Test Generation



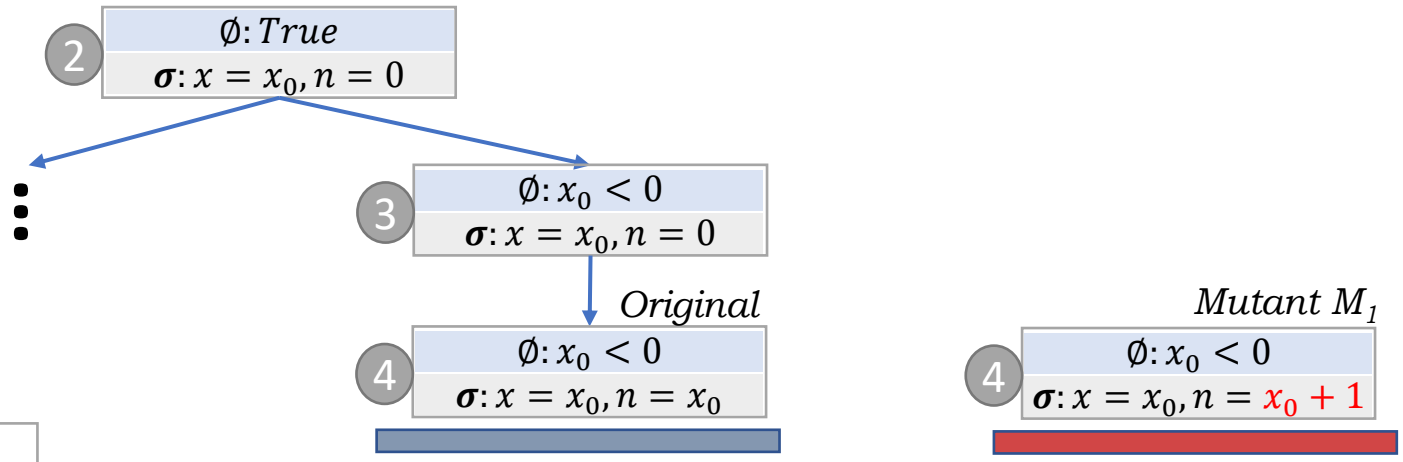
Symbolic Mutant Test Generation

Infect-only



Symbolic Mutant Test Generation

Infect-only



Program

```
int func (int x) {
1.   int n = 0;
2.   if (x < 0) {
3.       n = x; // M1 (n = x + 1;)
4.       if (n)
5.           n++;
     }
7.   return n;
}
```

The mutant M_1 is **not** killed by the test

$$(\ State_{Original} \not\equiv State_{Mutant}\ ) \wedge \emptyset_{Original} \wedge \emptyset_{Mutant}$$

$$(\ x_0 \neq x_0 + 1\ ) \wedge x_0 < 0 \wedge x_0 < 0$$

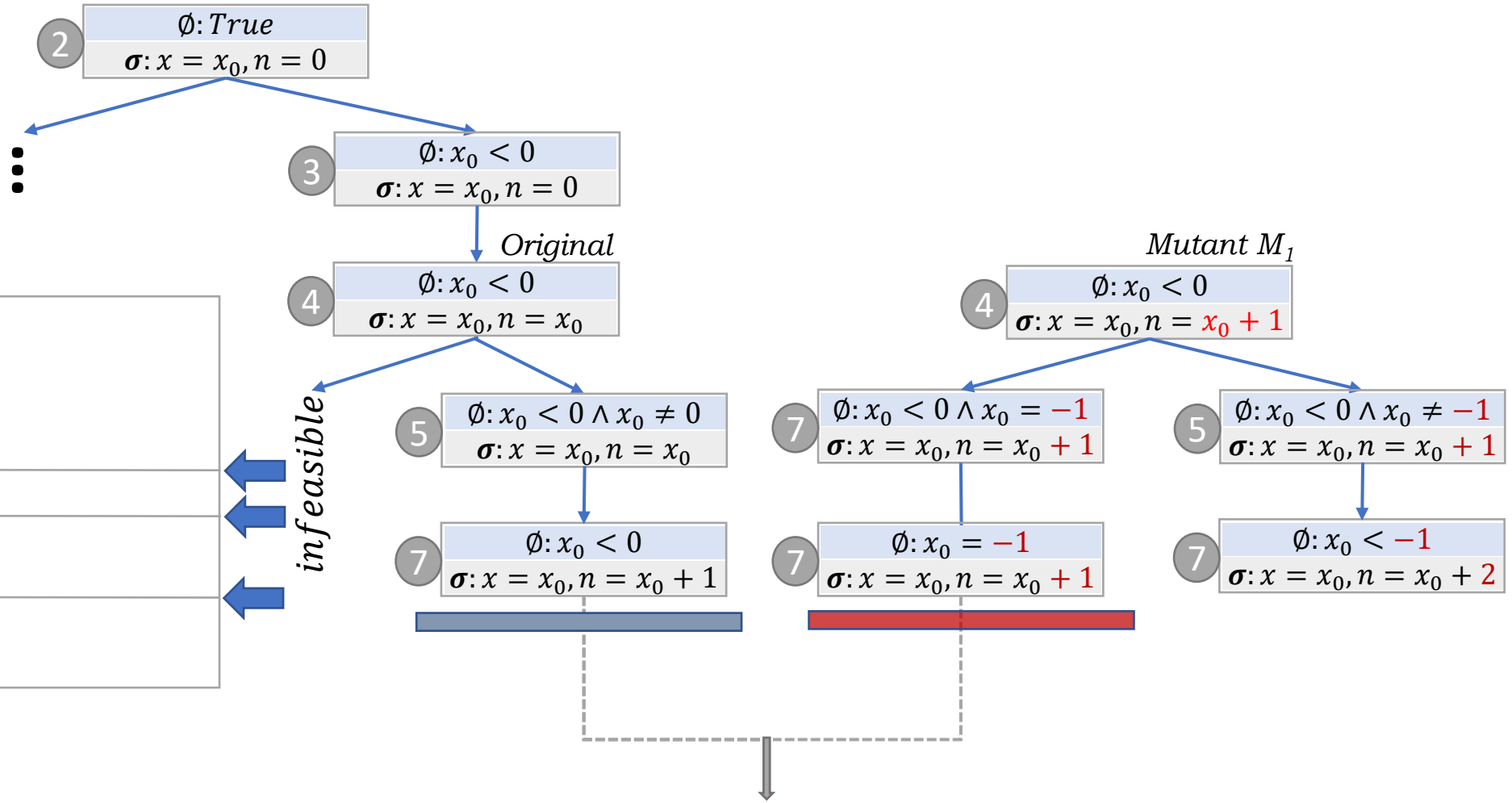
Z3 solver

$$x_0 = -1$$

$$Out_{Original} = 0$$
$$Out_{Mutant} = 0$$

Symbolic Mutant Test Generation

SEMu



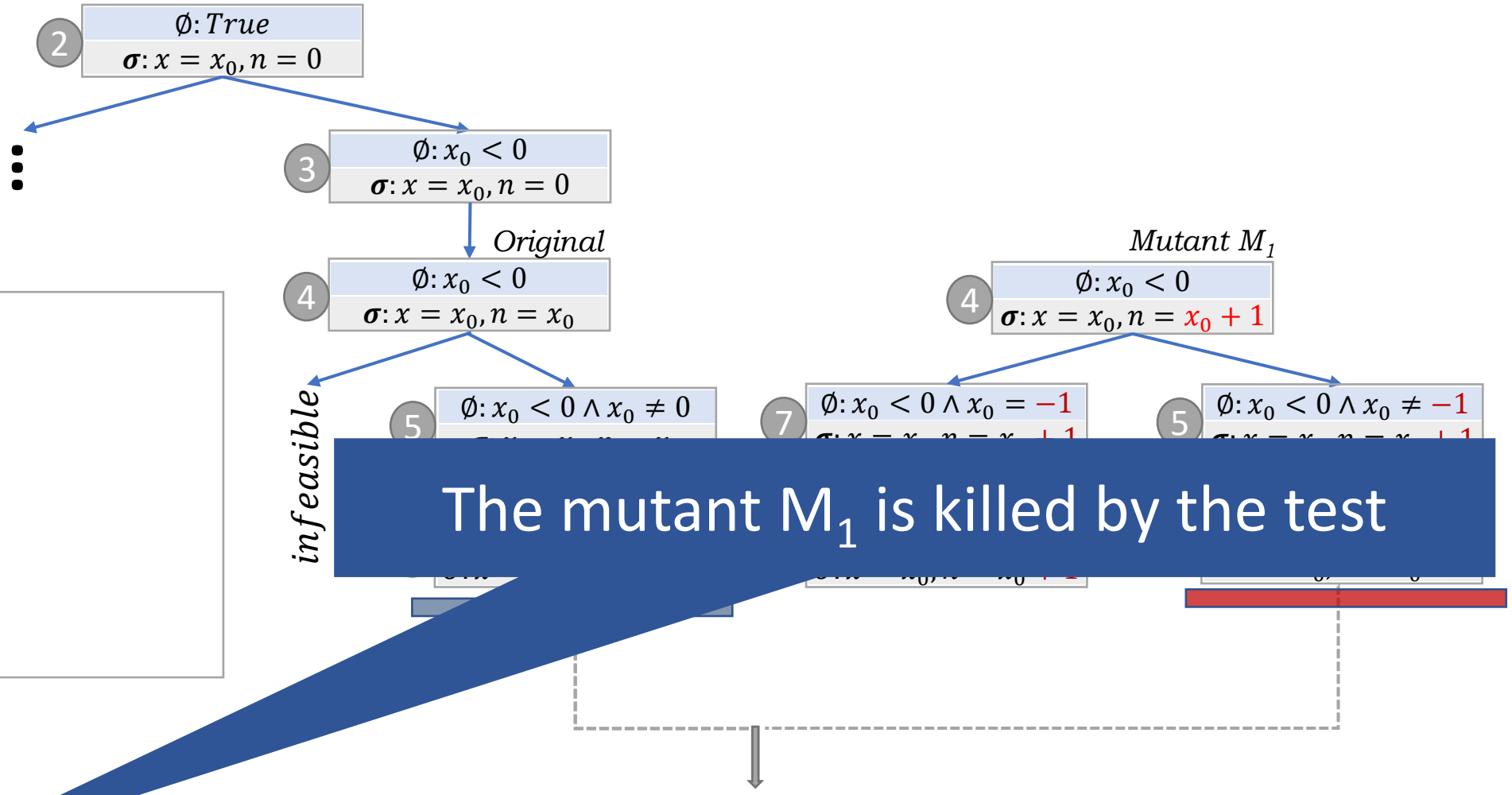
Program

```
int func (int x) {
1.   int n = 0;
2.   if (x < 0) {
3.       n = x; // M1 (n = x + 1;)
4.       if (n)
5.           n++;
     }
7.   return n;
}
```

$Infeasible$ ← *Z3 solver* $(x_0 + 1 \neq x_0 + 1) \wedge x_0 < 0 \wedge x_0 = -1$

Symbolic Mutant Test Generation

SEMu



Program

```
int func (int x) {
1.   int n = 0;
2.   if (x < 0) {
3.       n = x; // M1 (n = x + 1;)
4.       if (n)
5.           n++;
     }
7.   return n;
}
```

The mutant M_1 is killed by the test

$Out_{Original} = -1$
$Out_{Mutant} = 0$

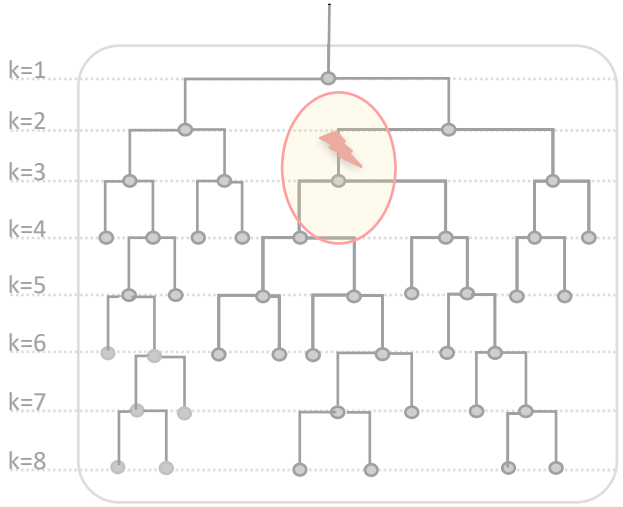
$x_0 = -2$ ← *Z3 solver*

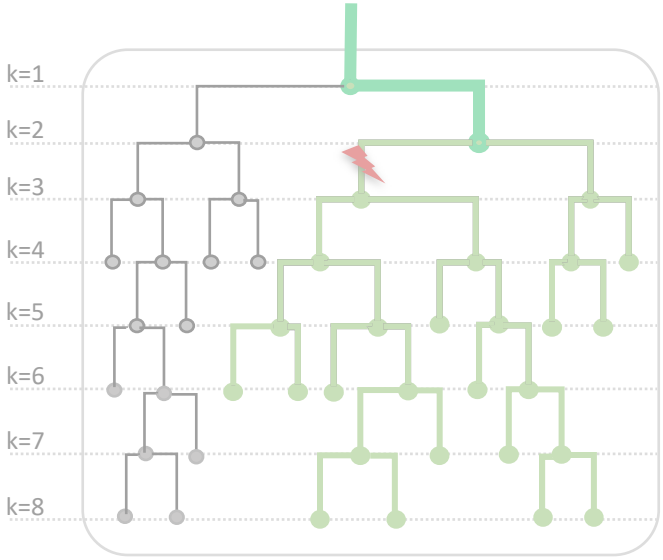
$(\ x_0 + 1 \neq x_0 + 2\ ) \wedge x_0 < 0 \wedge x_0 < -1$

SEMu

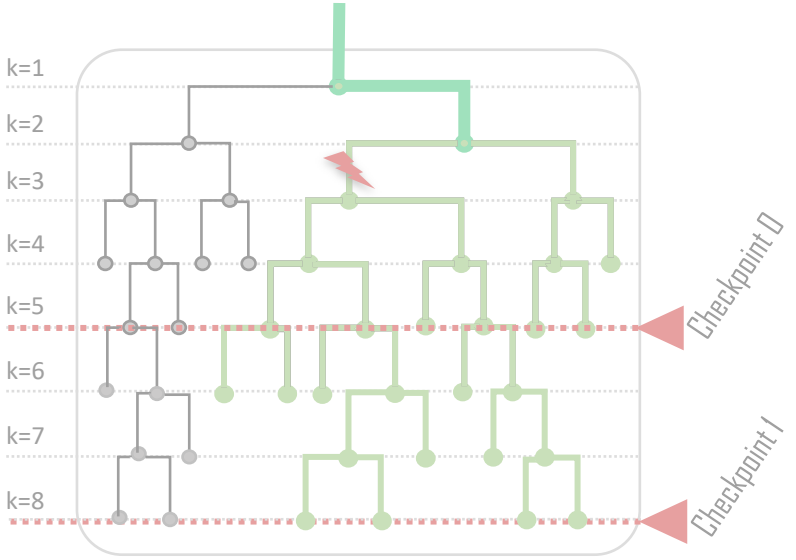
***SEMu* implements heuristics that enable scalable and deeper mutant-error propagation**

SEMu Cost-control Heuristics

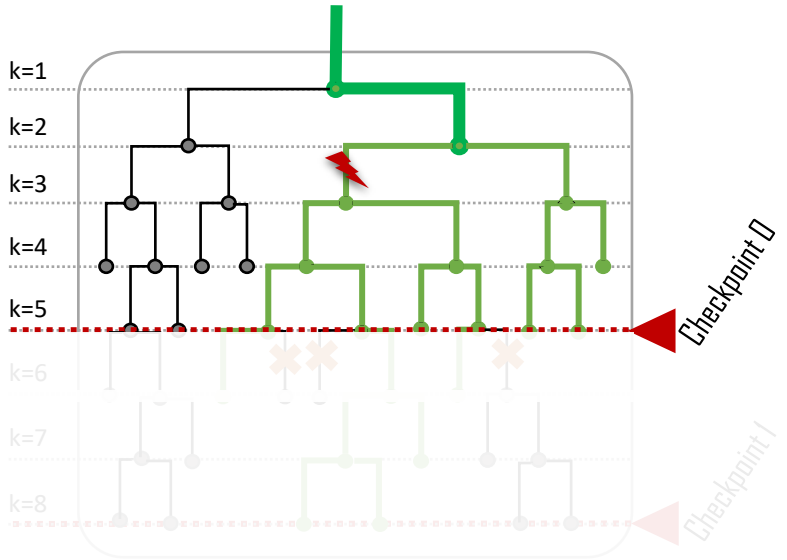




(a)



(b)

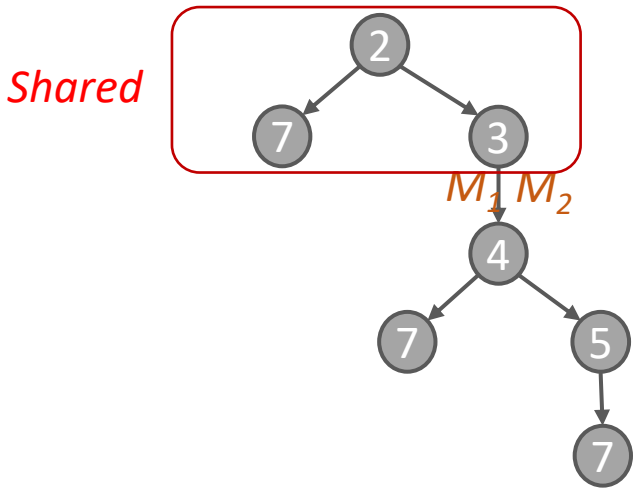


(c)

Completeness Preserving Optimizations

Program

```
int func (int x) {
1.   int n = 0;
2.   if (x < 0) {
3.       n = x; // M1 (n = x + 1;) M2 (n += x;)
4.       if (n)
5.           n++;
     }
7.   return n;
}
```



Meta-mutant Program

```
unsigned MUTANT_ID;
int func (int x) {
1.   int n = 0;
2.   if (x < 0) {
3.       switch(MUTANT_ID){
           case 1: n = x + 1; break;
           case 2: n += x; break;
           default: n = x;
         }
4.       if (n)
5.           n++;
     }
7.   return n;
}
```

Implementation

SEMu **is implemented on KLEE**

Our implementation added/modified more than 8,000 lines of code.

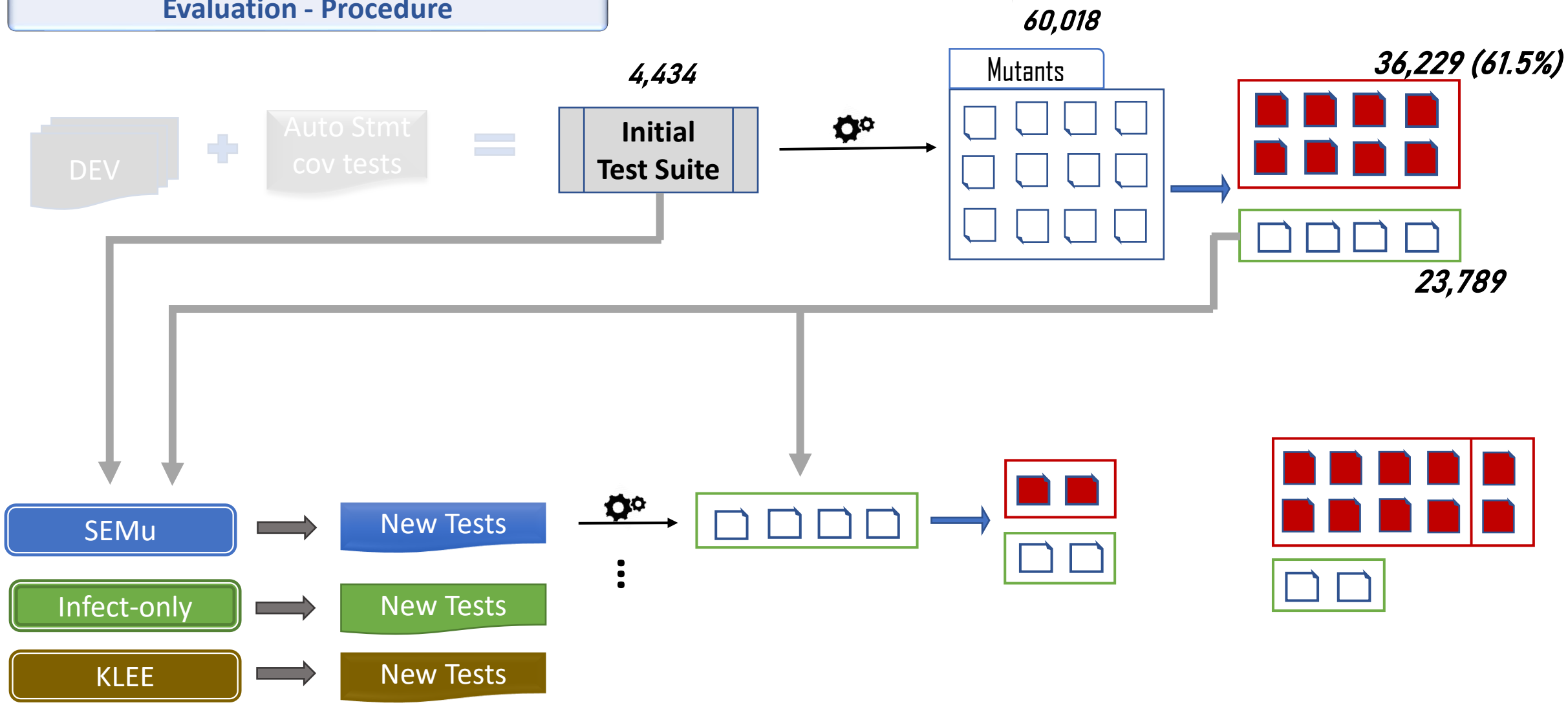
https://github.com/thierry-tct/KLEE-SEMu

Evaluation

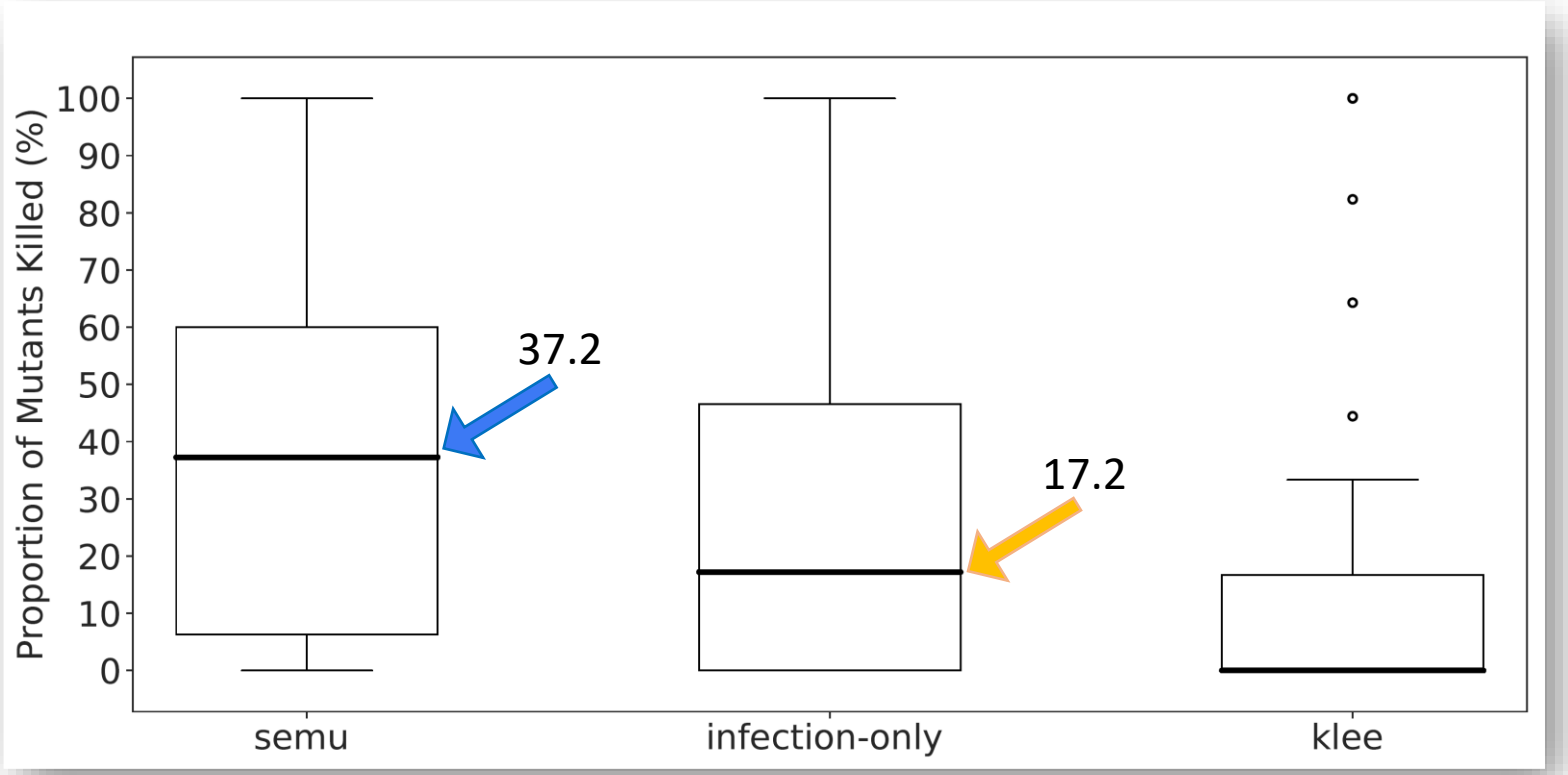
We evaluate *SEMu* and compare with KLEE and the state-of-the-art-approach (*infect-only*)

Subjects: **36** Programs from GNU Coreutils

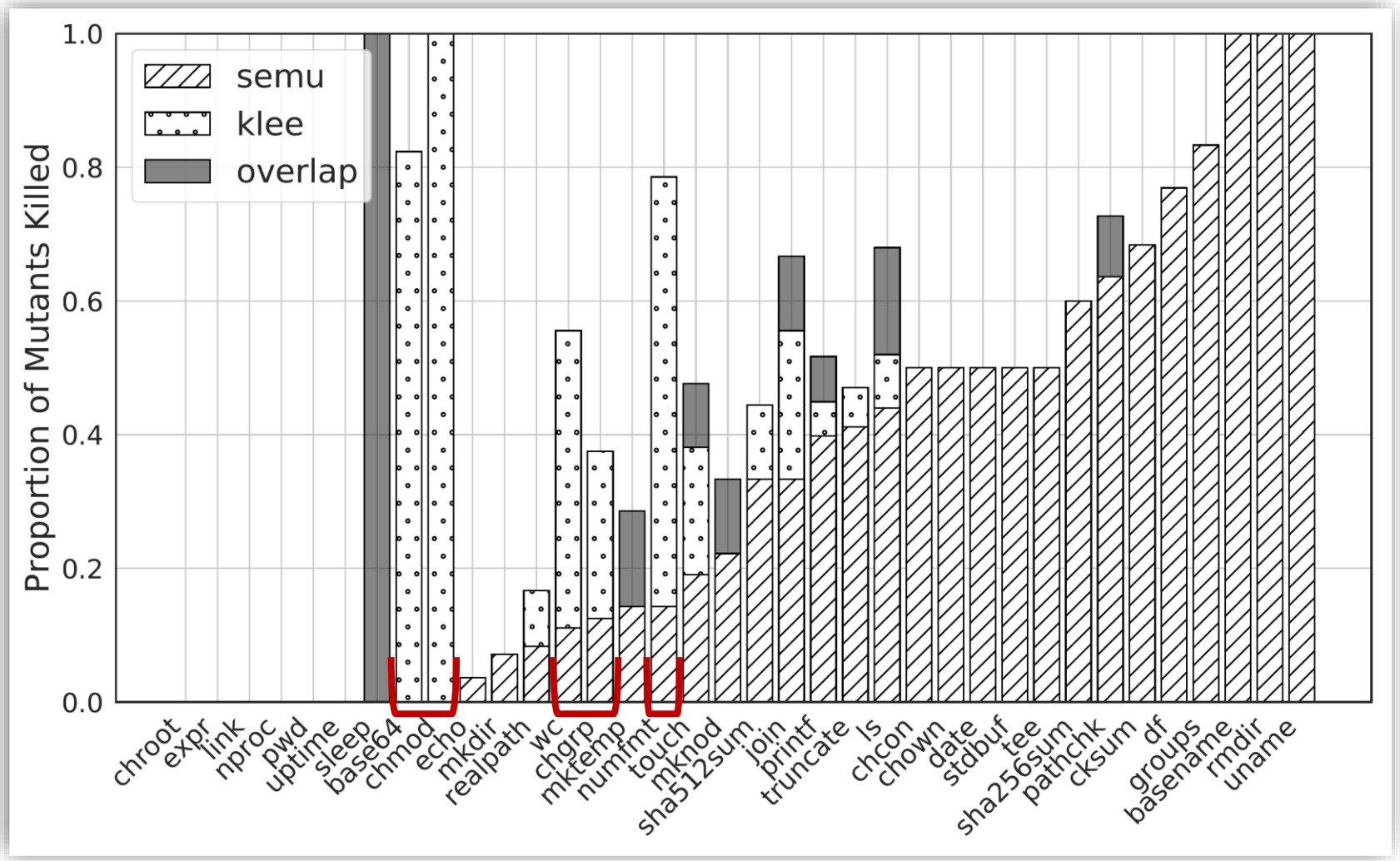
Evaluation - Procedure



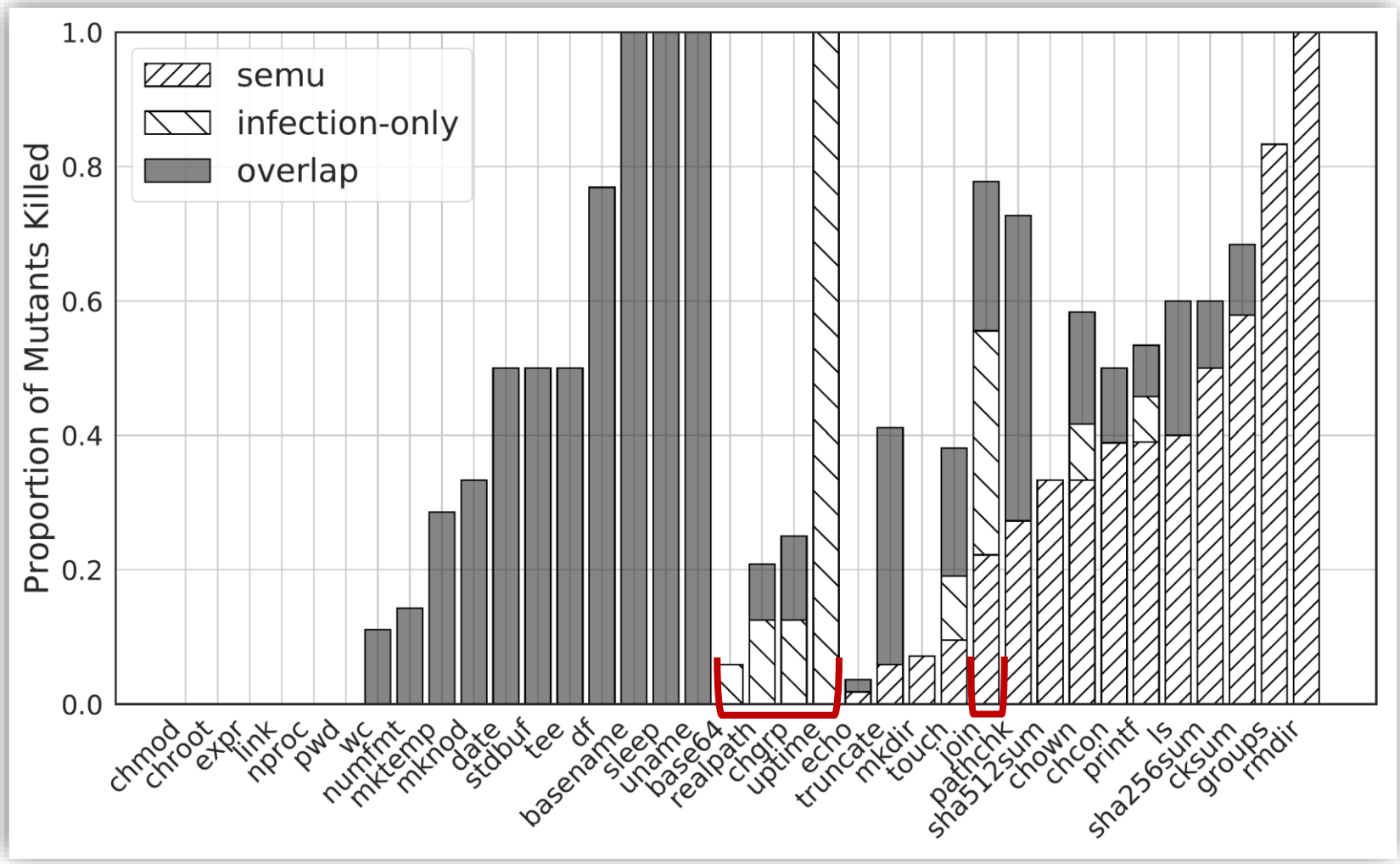
Evaluation - Results



Evaluation - Results



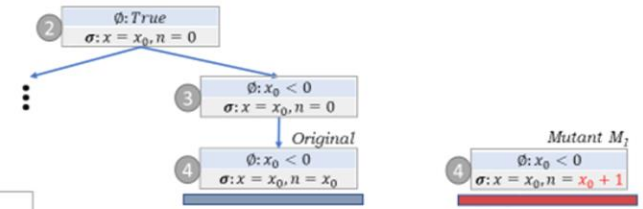
Evaluation - Results



Mutation

Symbolic Mutant Test Generation

Infect-only



Program

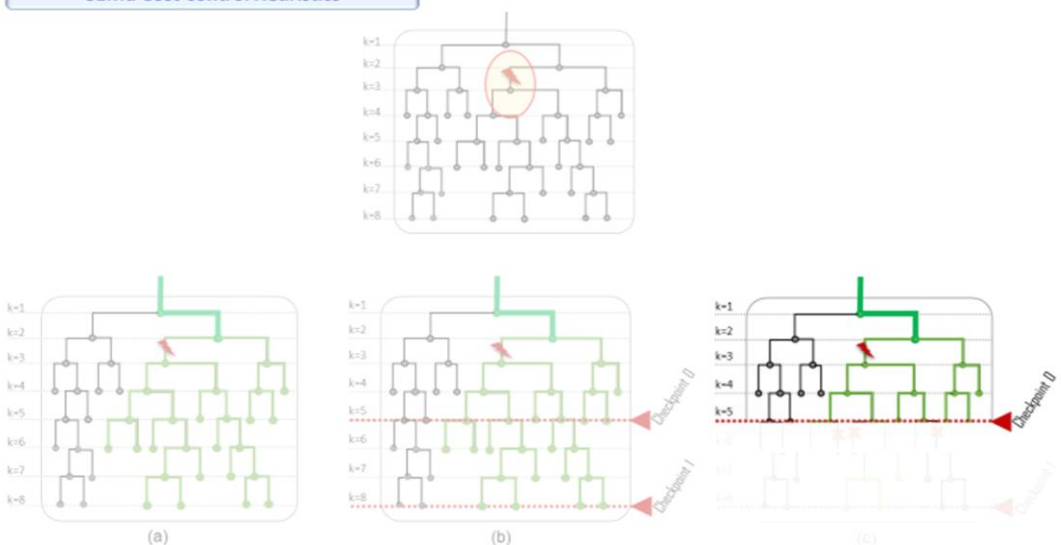
```
int func (int x) {
1.  int n = 0;
2.  if (x < 0) {
3.      n = x; // M1 (n = x + 1;)
4.      if (n)
5.          n++;
    }
7.  return n;
}
```

The mutant M_1 is **not** killed by the test

$$(State_{Original} \not\equiv State_{Mutant}) \wedge \emptyset_{Original} \wedge \emptyset_{Mutant}$$

$$Out_{Original} = 0 \quad Out_{Mutant} = 0 \quad x_0 = -1 \quad \text{Z3 solver} \quad (x_0 \neq x_0 + 1) \wedge x_0 < 0 \wedge x_0 < 0$$

SEMu Cost-control Heuristics



Evaluation - Results