# Symbiotic 10

Program Analysis Framework for C/LLVM

**Martin Jonáš** + Symbiotic team

Masaryk University, Brno, Czechia

# What is Symbiotic

**Framework for analysis of C/LLVM programs**

- static code analysis (range analysis, point-to analysis, alias analysis, ...)
- program instrumentation
- program slicing
- bug finding
- verification
- test code generation

# What is Symbiotic

**Framework for analysis of C/LLVM programs**

- static code analysis (range analysis, point-to analysis, alias analysis, . . .)
- program instrumentation
- program slicing
- bug finding
- verification
- test code generation

# What is Symbiotic

**Framework for analysis of C/LLVM programs**

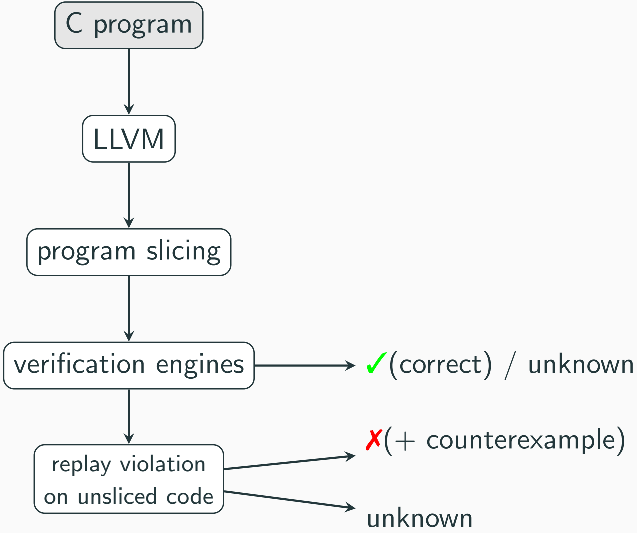
- static code analysis (range analysis, point-to analysis, alias analysis, . . .)
- program instrumentation
- program slicing
- bug finding
- verification
- test code generation

# Verification: Supported Properties

**Bug finding/verification for**

- assertion safety (no violated assertions)
- memory safety (no invalid dereferences)
- memory cleanup (no memory leaks)
- overflow safety (no signed overflows)
- termination (no infinite executions)

# Workflow for assertion safety



# Verification Engines

## Main, used by default

- JetKlee: symbolic execution (our fork of klee)
- slowbeast: backwards symbolic execution + loop folding
- slowbeast: compact symbolic execution

## Integration with many more

- CPAchecker
- Divine
- Nidhugg
- SeaHorn
- smack
- + other experimental

# Verification Engines

## Main, used by default

- JetKlee: symbolic execution (our fork of klee)
- slowbeast: backwards symbolic execution + loop folding
- slowbeast: compact symbolic execution

## Integration with many more

- CPAchecker
- Divine
- Nidhugg
- SeaHorn
- smack
- + other experimental

# JetKlee and SlowBeast

| | KLEE | JETKLEE | SLOWBEAST |
|---|---|---|---|
| symbolic pointers | ✓ | ✓ | ✓ |
| symbolic-sized allocations | ✗ | ✓ | ✓ |
| symbolic addresses | ✗ | ✓ | ✓ |
| lazy memory | ✗ | ✓ | ✓ |
| symbolic floats | ✗ | ✗ | ✓ |
| parallel programs | ✗ | ✗ | ✓ |
| invariant generation | ✗ | ✗ | ✓ |

# Program Slicing

```
n = input();
i = 0;
while (i < n) {
  c = input();
  if (i == 0) {
    min = c;
    max = c;
  }
  if (c < min)
    min = c;
  if (c > max)
    max = c;
  i = i + 2;
}
assert(min <= c);
```

# Program Slicing

```
n = input();
i = 0;
while (i < n) {
  c = input();
  if (i == 0) {
    min = c;
    max = c;
  }
  if (c < min)
    min = c;
  if (c > max)
    max = c;
  i = i + 2;
}
assert(min <= c);
```

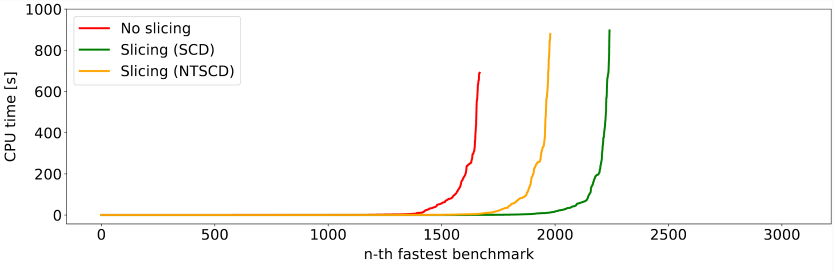
# Program Slicing

```
n = input();
i = 0;
while (i < n) {
  c = input();
  if (i == 0) {
    min = c;
    max = c;
  }
  if (c < min)
    min = c;
  if (c > max)
    max = c;
  i = i + 2;
}
assert(min <= c);
```

```
n = input();
i = 0;
while (i < n) {
  c = input();
  if (i == 0) {
    min = c;
    max = c;
  }
  if (c < min)
    min = c;
  if (c > max)
    max = c;
  i = i + 2;
}
assert(min <= c);
```

# Effect of Slicing

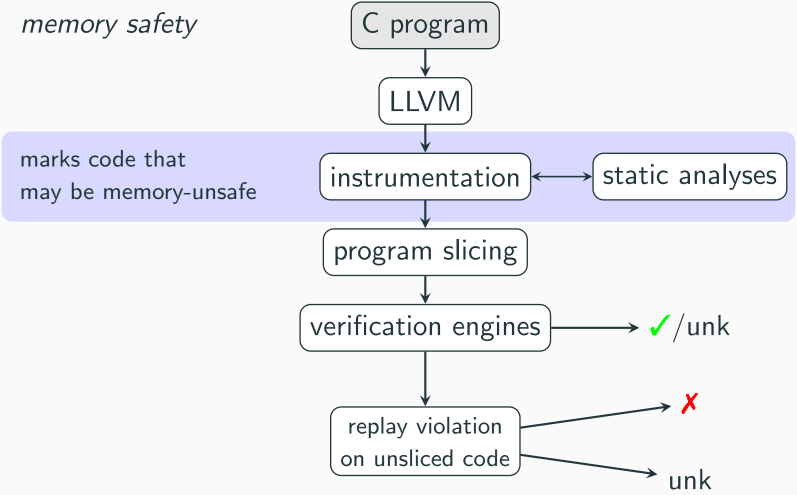
Correct verification results produced by KLEE with slicing on reachability safety tasks of SV-COMP 2019



[Chalupa and Strejček: *Evaluation of Program Slicing in Software Verfication*. iFM 2019.]

# Workflow for other properties

*memory safety*



C program

LLVM

marks code that
may be memory-unsafe

instrumentation

static analyses

program slicing

verification engines

✓/unk

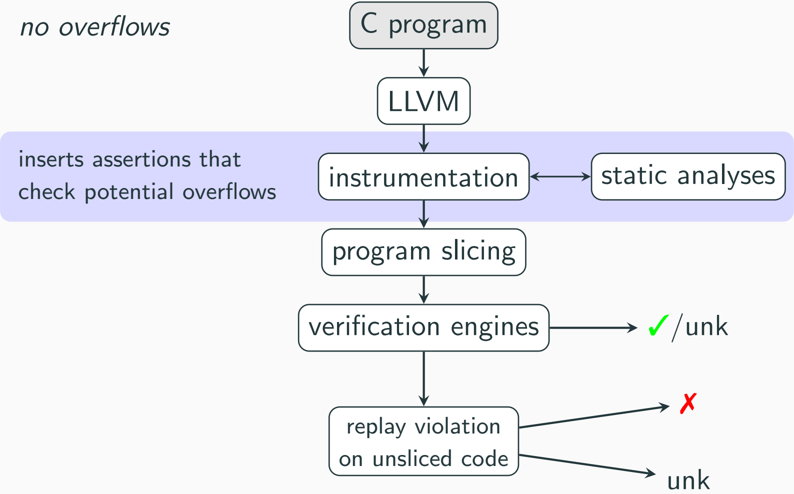
replay violation
on unsliced code

✗

unk

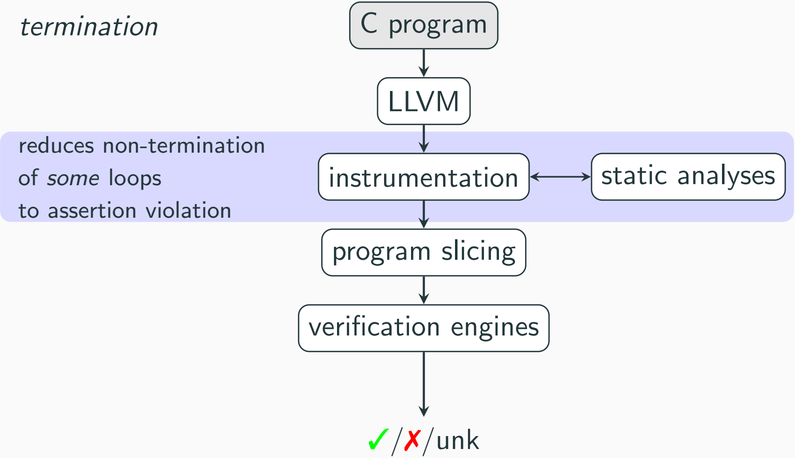
# Workflow for other properties

*no overflows*



# Workflow for other properties

*termination*



reduces non-termination of *some* loops to assertion violation

C program

LLVM

instrumentation

static analyses

program slicing

verification engines

✓/✗/unk

# Program Instrumentation

```
int x = input();
int y = 30;
if (x >= 5 && x <= 10) {
  int z = x * y;
} else {
  int z = x + 1;
}
int v = y * y;
```

# Program Instrumentation

```
int x = input();
int y = 30;
if (x >= 5 && x <= 10) {
  int z = x * y;
} else {
  int z = x + 1;
}
int v = y * y;
```

```
int x = input();
int y = 30;
if (x >= 5 && x <= 10) {
  assert(!mul_overflows(x, y));
  int z = x * y;
} else {
  assert(!add_overflows(x, 1));
  int z = x + 1;
}
assert(!mul_overflows(y, y));
int v = y * y;
```

# Program Instrumentation

```
int x = input(); (x ∈ [INT_MIN,INT_MAX])
int y = 30; (y ∈ [30,30])
if (x >= 5 && x <= 10) {
  int z = x * y; (x ∈ [5,10], y ∈ [30,30])
} else {
  int z = x + 1; (x ∈ [INT_MIN,INT_MAX])
}
int v = y * y; (y ∈ [30,30])
```

```
int x = input();
int y = 30;
if (x >= 5 && x <= 10) {
  assert(!mul_overflows(x, y));
  int z = x * y;
} else {
  assert(!add_overflows(x, 1));
  int z = x + 1;
}
assert(!mul_overflows(y, y));
int v = y * y;
```

# Program Instrumentation

```
int x = input(); (x ∈ [INT_MIN,INT_MAX])
int y = 30; (y ∈ [30,30])
if (x >= 5 && x <= 10) {
  int z = x * y; (x ∈ [5,10], y ∈ [30,30])
} else {
  int z = x + 1; (x ∈ [INT_MIN,INT_MAX])
}
int v = y * y; (y ∈ [30,30])
```

```
int x = input();
int y = 30;
if (x >= 5 && x <= 10) {

  int z = x * y;
} else {
  assert(!add_overflows(x, 1));
  int z = x + 1;
}

int v = y * y;
```

# Program Instrumentation

```
int x = input(); (x ∈ [INT_MIN,INT_MAX])
int y = 30; (y ∈ [30,30])
if (x >= 5 && x <= 10) {
  int z = x * y; (x ∈ [5,10], y ∈ [30,30])
} else {
  int z = x + 1; (x ∈ [INT_MIN,INT_MAX])
}
int v = y * y; (y ∈ [30,30])
```

```
int x = input();
int y = 30;
if (x >= 5 && x <= 10) {

  int z = x * y;
} else {
  assert(!add_overflows(x, 1));
  int z = x + 1;
}

int v = y * y;
```

# SV-COMP Results

## SV-COMP = Competition on Software Verification

- organized by Dirk Beyer since 2012
- 23 805 verification tasks in C (in 2023)
- 52 participating tools (in 2023)

## Symbiotic in SV-COMP

- 5 gold medals in MemSafety (2018, 2019, 2021, 2022, 2023)
- 4 gold medals in SoftwareSystems (2020, 2021, 2022, 2023)
- overall winner of SV-COMP 2022

# Symbiotic Team Throughout the History

**Roughly in chronological order**

- Jan Strejček
- Jiri Slaby
- Marek Trtík
- Marek Chalupa
- Martina Velanová
- Michael Šimáček
- Tomáš Jašek
- Lukáš Tomovič
- Anna Řechtáčková
- Lukáš Zaoral
- Vincent Mihalkovič
- Paulína Ayaziová
- Jakub Novák
- Jindřich Sedláček
- Kristián Kumor

# Conclusions

We are looking for real-world verification tasks!

**Try Symbiotic**

- `https://staticafi.github.io/symbiotic/`

**Contact us**

- `statica@fi.muni.cz`