# Interpolation-based Function Summaries in BMC

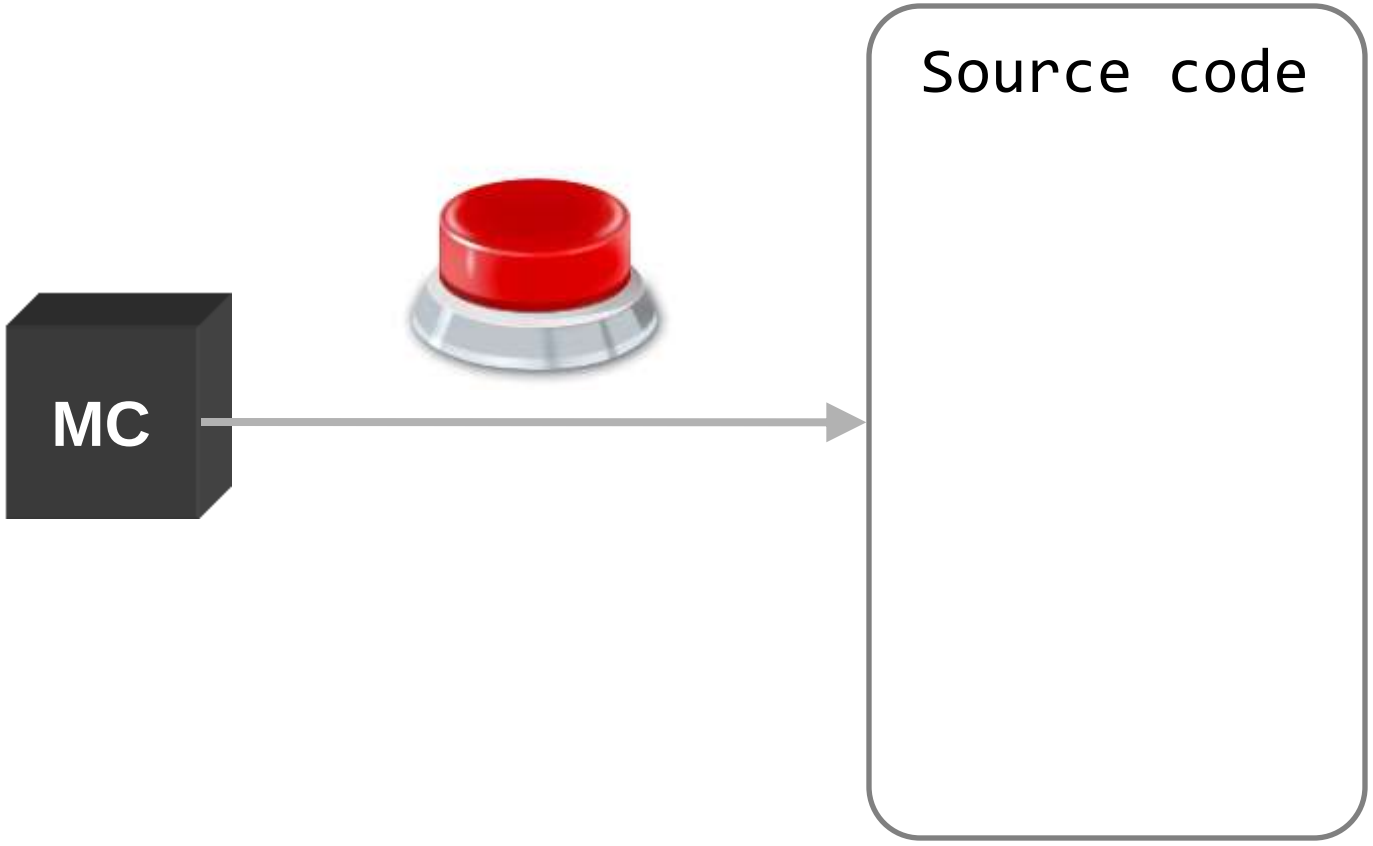
***Ondřej Šerý***, *Grigory Fedyukovich*, *Natasha Sharygina*

**Formal Verication and Security Lab**
http://verify.inf.usi.ch/
University of Lugano

# Ideal application of model checking

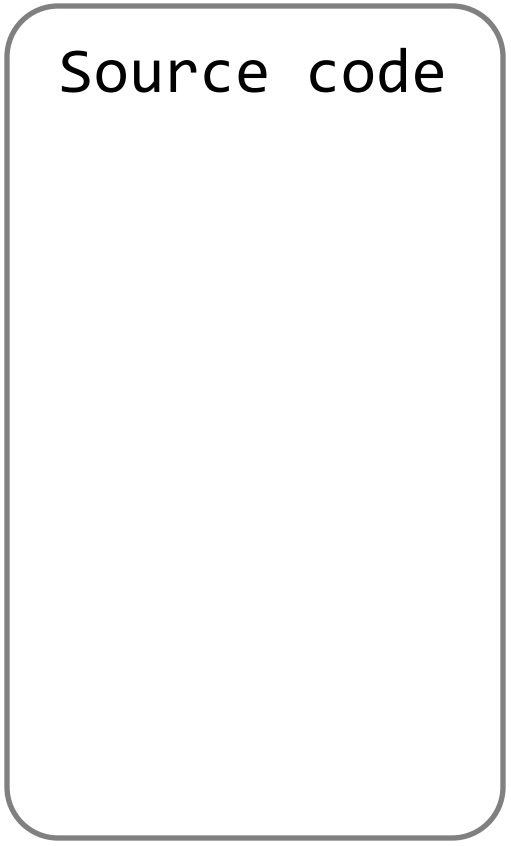
# Ideal application of model checking


# Ideal application of model checking

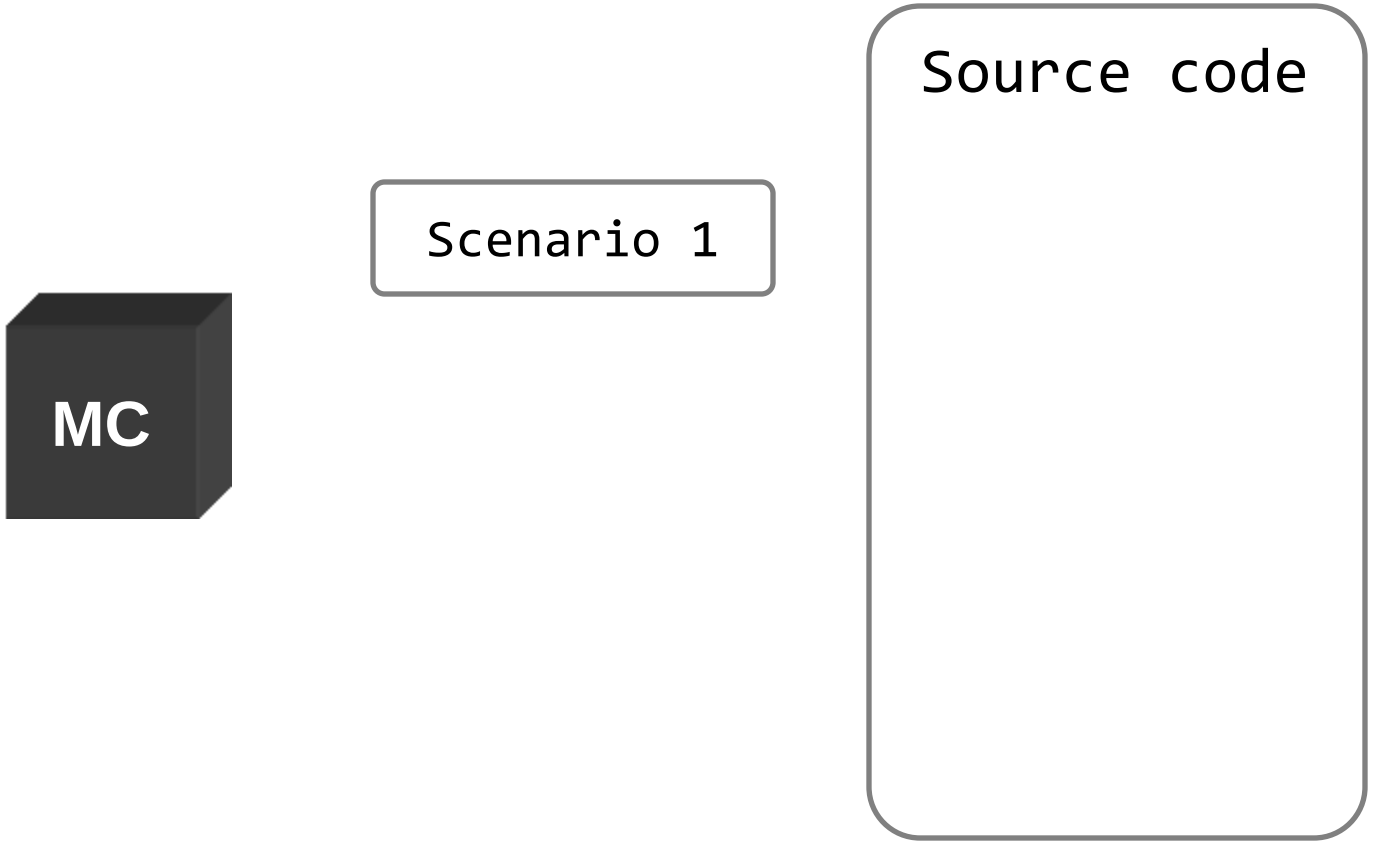


# Realistic application of model checking

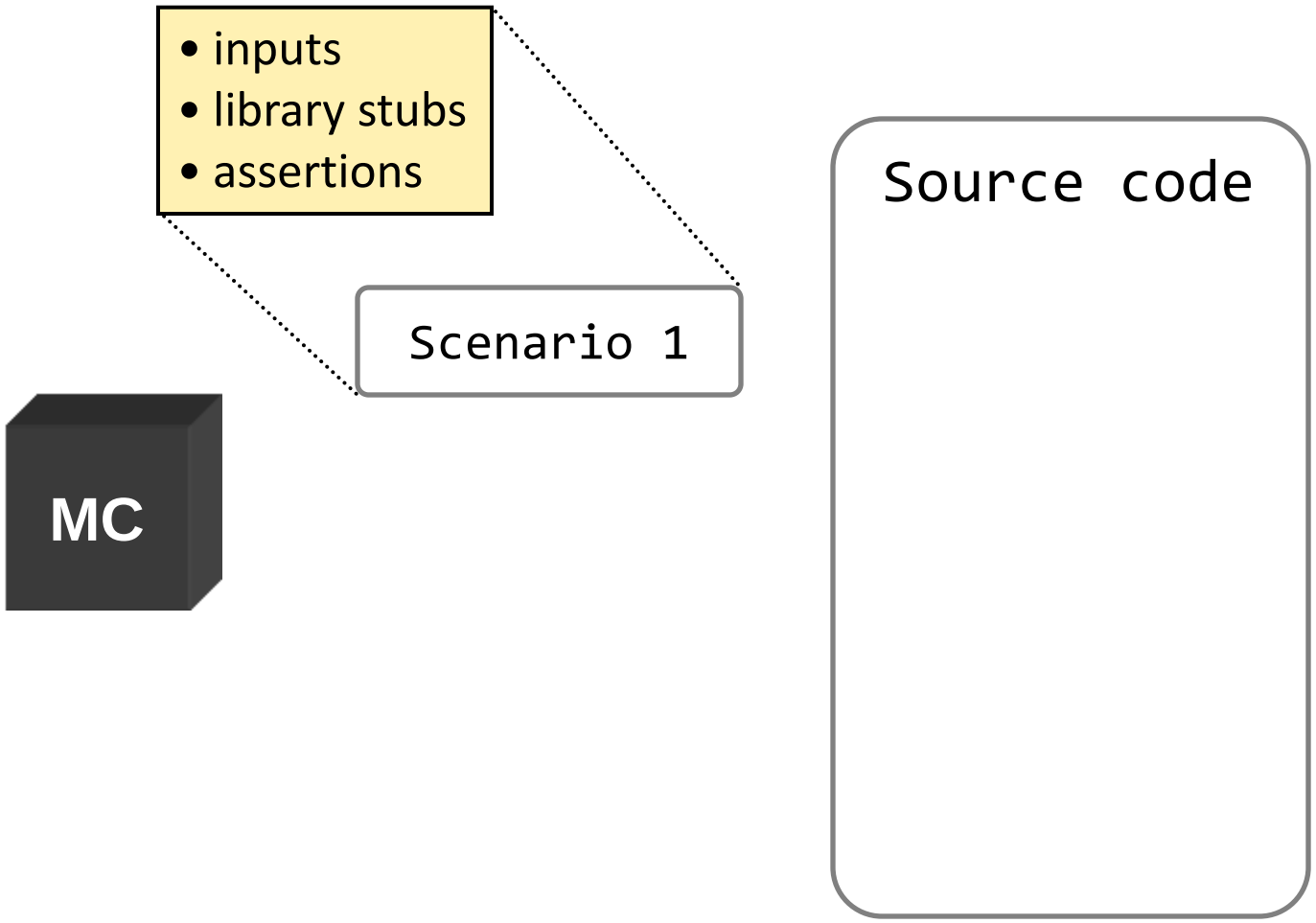
# Realistic application of model checking

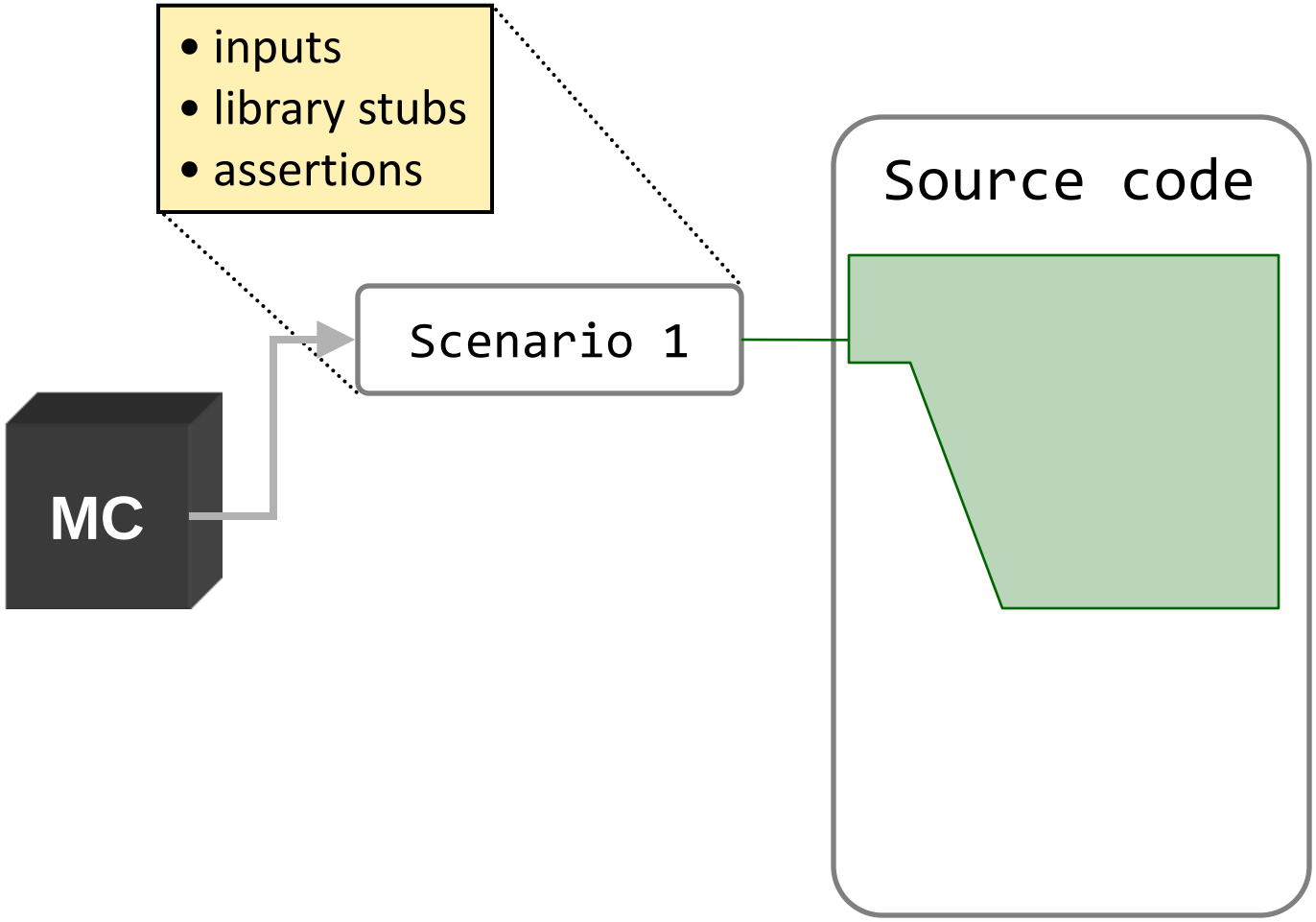
# Realistic application of model checking



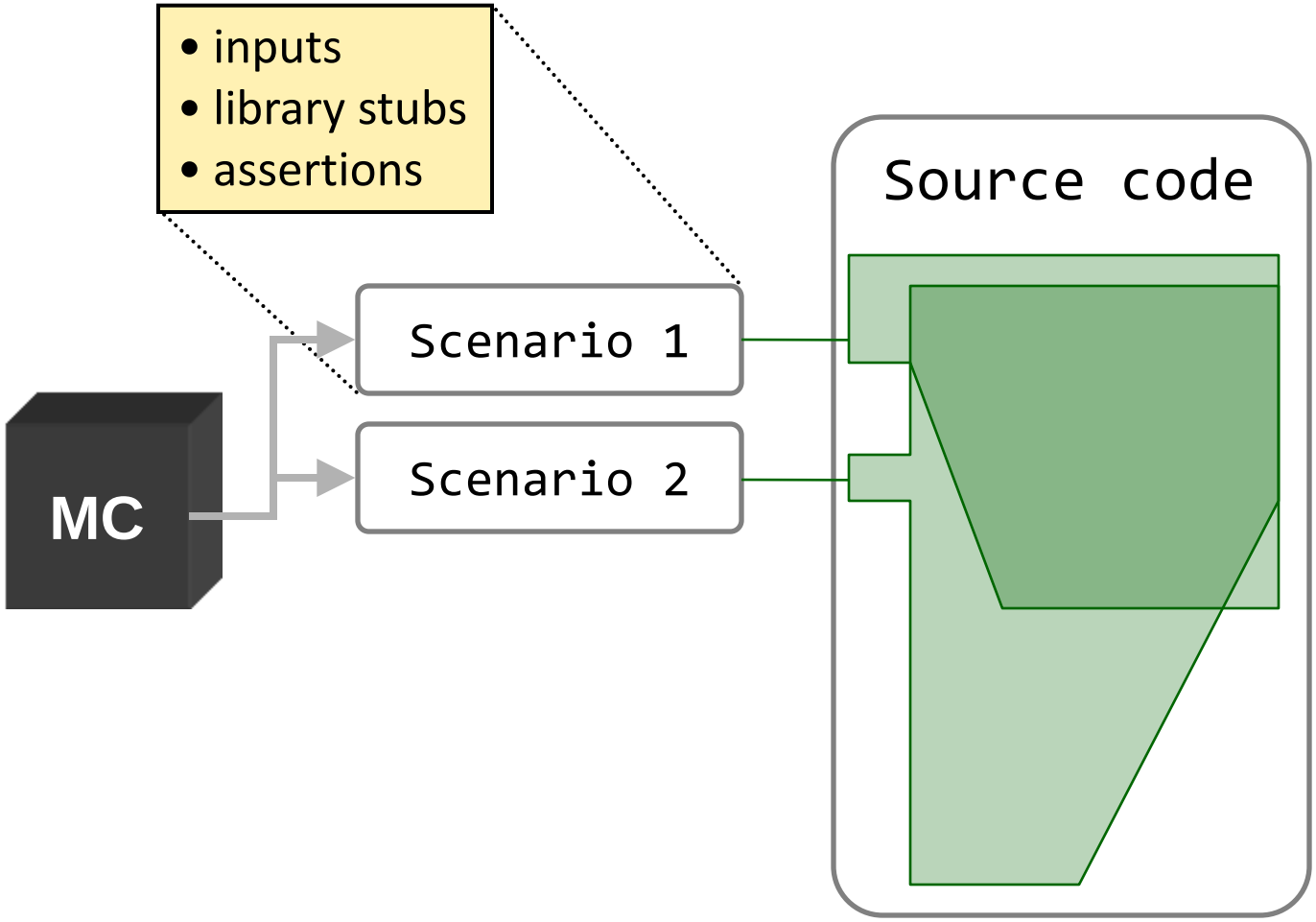
# Realistic application of model checking



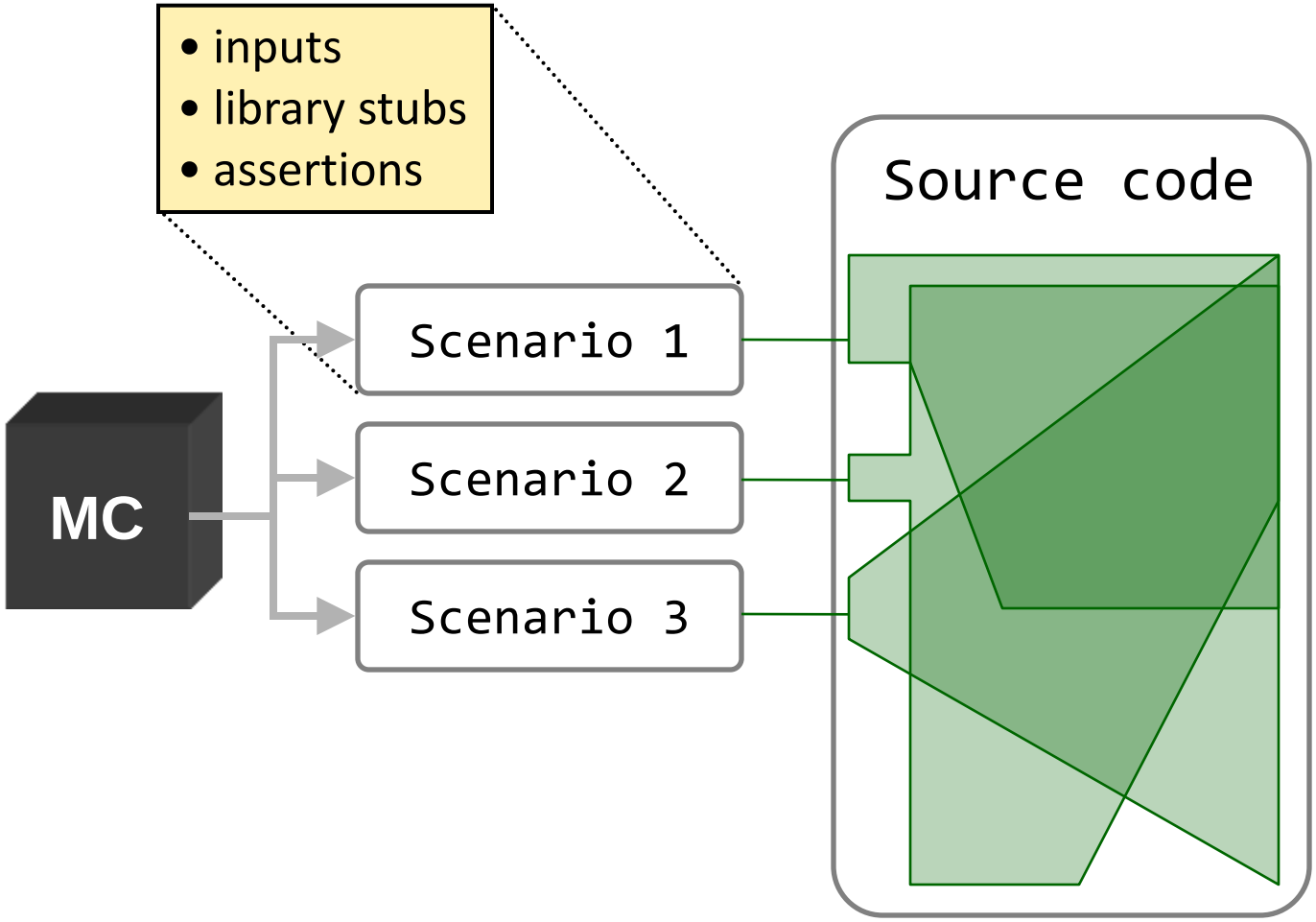
# Realistic application of model checking



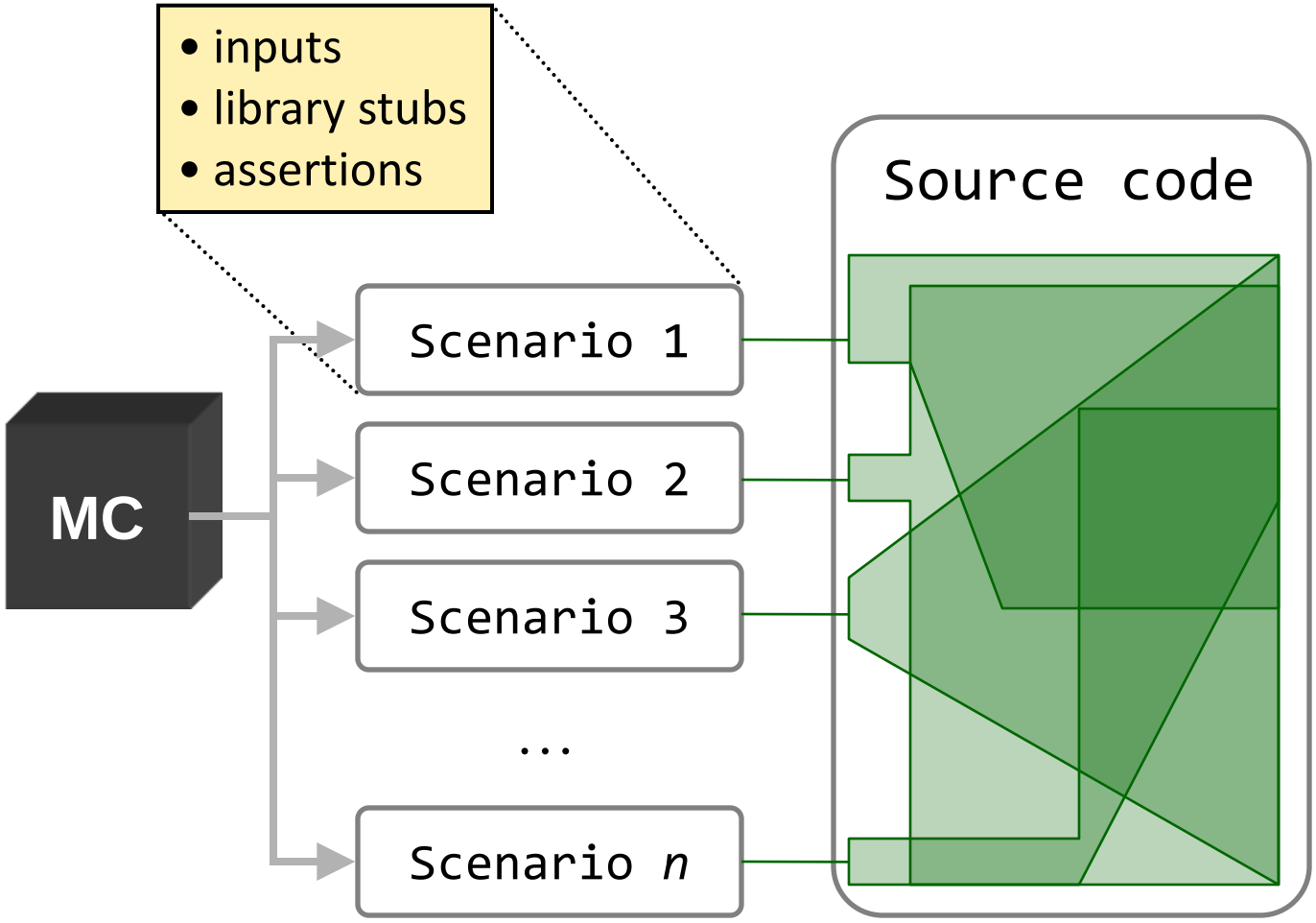
# Realistic application of model checking



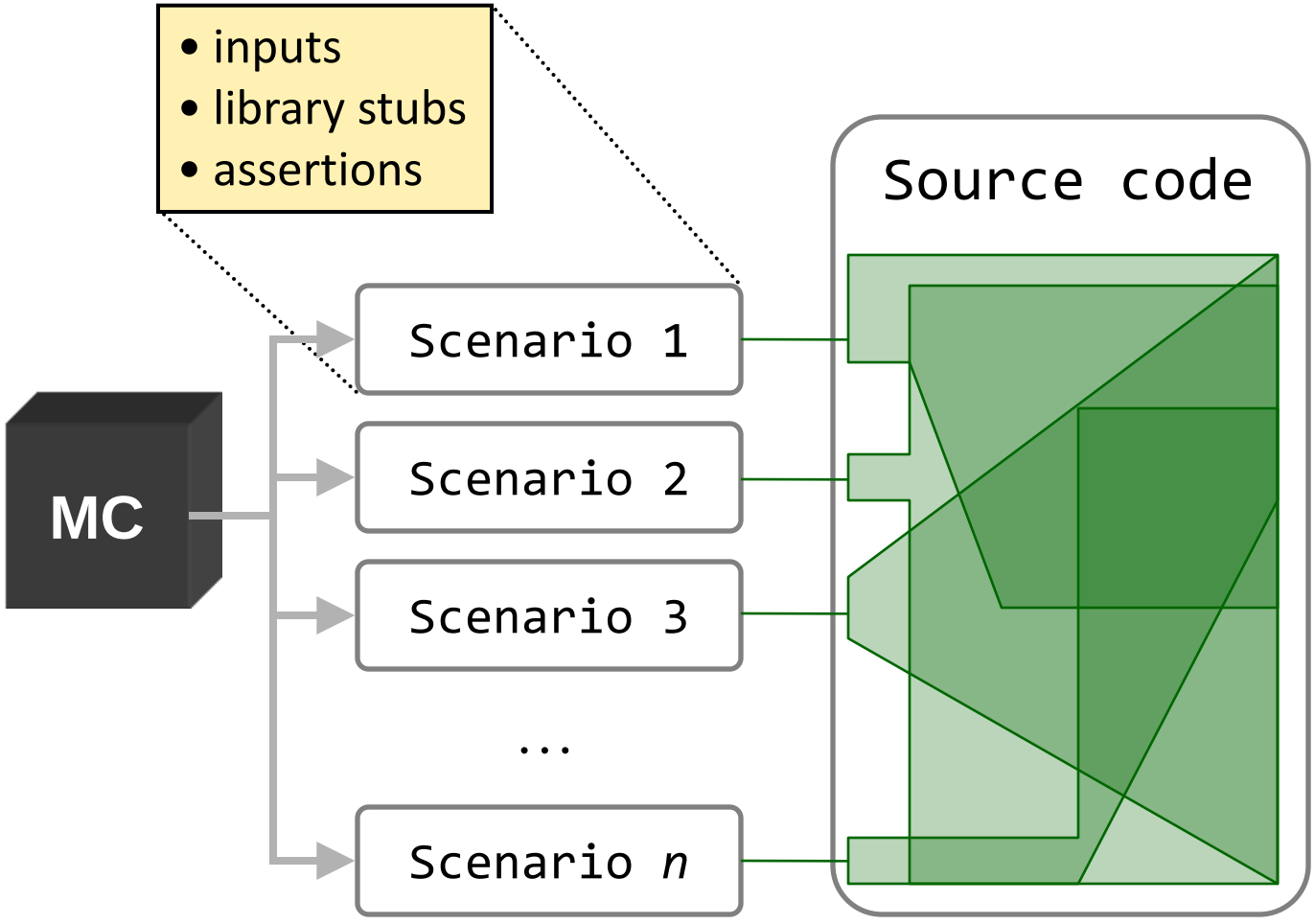
# Realistic application of model checking



# Realistic application of model checking



# Realistic application of model checking




*This is done, for example, in Static Driver Verifier…*

# Motivation

***Problem:***

- Re-verification of already analyzed code

***Our take:***

- Extraction of function summaries
  - Over-approximation using *Craig interpolation*
- Reuse in subsequent verification runs

# Motivation

***Problem:***

- Re-verification of already analyzed code

***Our take:***

- Extraction of function summaries
  - Over-approximation using *Craig interpolation*
- Reuse in subsequent verification runs

# Motivation

***Problem:***

- Re-verification of already analyzed code

***Our take:***

- Extraction of function summaries
  - Over-approximation using *Craig interpolation*
- Reuse in subsequent verification runs
- Refinement
  - If the summaries are too weak

# Bounded model checking

*"For a given bound n, find all errors reachable with all loops unrolled at most n-times"*

- BMC formula
  - SSA form
  - Encoding into formula
  - Bit-blasting
- SAT solver

# Example: SSA form

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a – 10;
}
```

# Example: SSA form

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a – 10;
}
```

1. *Unroll loops n-times*
2. *Versioned variables*
3. *φ-functions on version clashes*

# Example: SSA form

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a – 10;
}
```

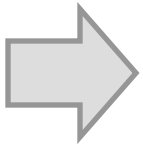
# Example: SSA form

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```



```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

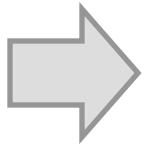
# Example: SSA form

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```



```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 – 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

1. *Expand φ-functions*
2. *Add negated assertion*

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 – 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 – 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

```
// main
y0 = 1;
x0 = nondet();
if (x0 > 0) {
  a0 = x0;
  // f
  if (a0 < 10)
    ret0 = a0;
  else
    ret1 = a0 - 10;
  ret2 = phi(ret0, ret1);
  // end f
  y1 = ret2;
}
y2 = phi(y0, y1);
assert(y2 >= 0);
```

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad\qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad\qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad\qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad\qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

*Bit-blasted and passed to a SAT solver*

# Example: BMC formula

$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad\qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad\qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

*Bit-blasted and passed to a SAT solver*

*UNSAT = safe*

*SAT = unsafe*

# Craig interpolation

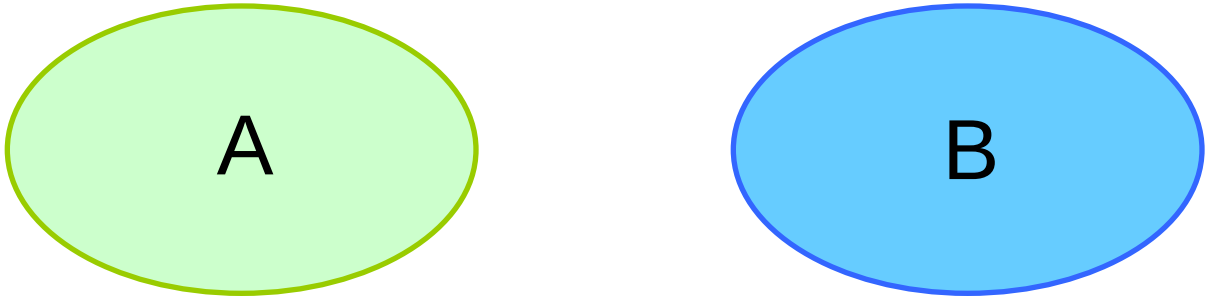
**Definition:**

- Given formulas $A$ and $B$, an *Interpolant* is a formula $I$ such that
  - $A \rightarrow I$
  - $I \wedge B$ is unsatisfiable
  - Variables in $I$ are common to both $A$ and $B$

# Craig interpolation

**Definition:**

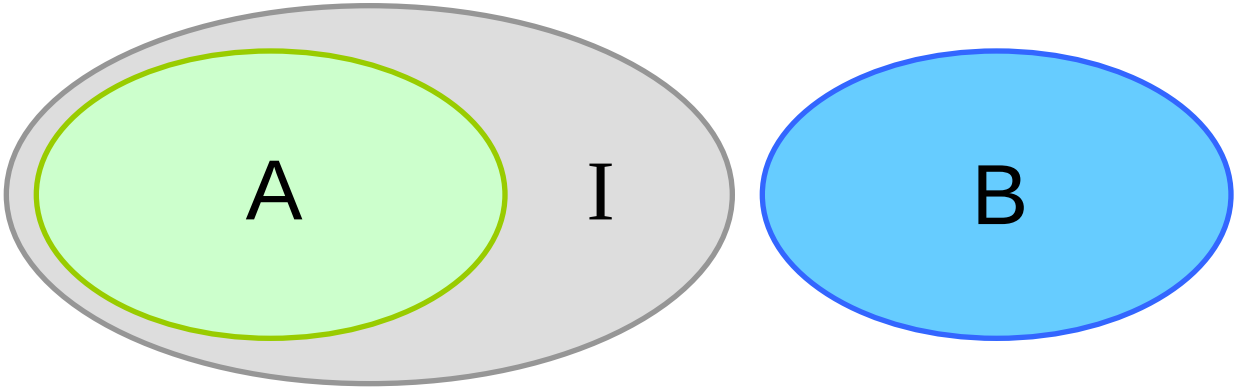
- Given formulas $A$ and $B$, an *Interpolant* is a formula $I$ such that
  - $A \rightarrow I$
  - $I \wedge B$ is unsatisfiable
  - Variables in $I$ are common to both $A$ and $B$



# Craig interpolation

**Definition:**

- Given formulas $A$ and $B$, an *Interpolant* is a formula $I$ such that
  - $A \rightarrow I$
  - $I \wedge B$ is unsatisfiable
  - Variables in $I$ are common to both $A$ and $B$



# Craig interpolation

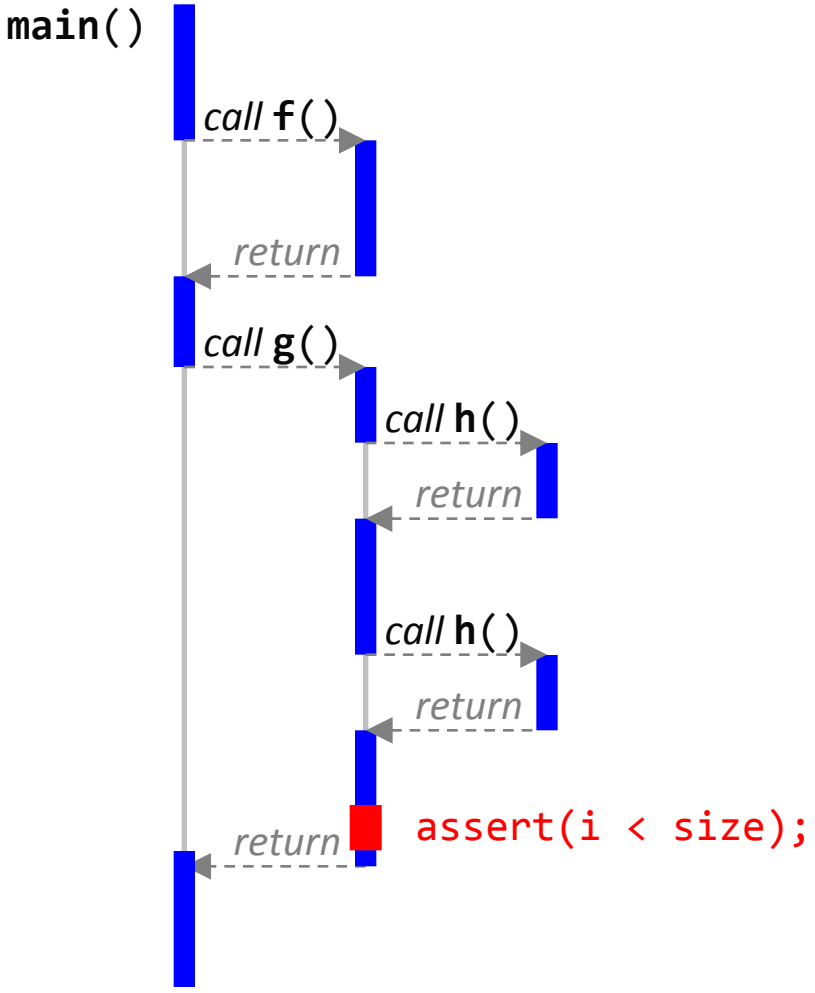
**Definition:**

- Given formulas $A$ and $B$, an *Interpolant* is a formula $I$ such that
  - $A \rightarrow I$
  - $I \wedge B$ is unsatisfiable
  - Variables in $I$ are common to both $A$ and $B$

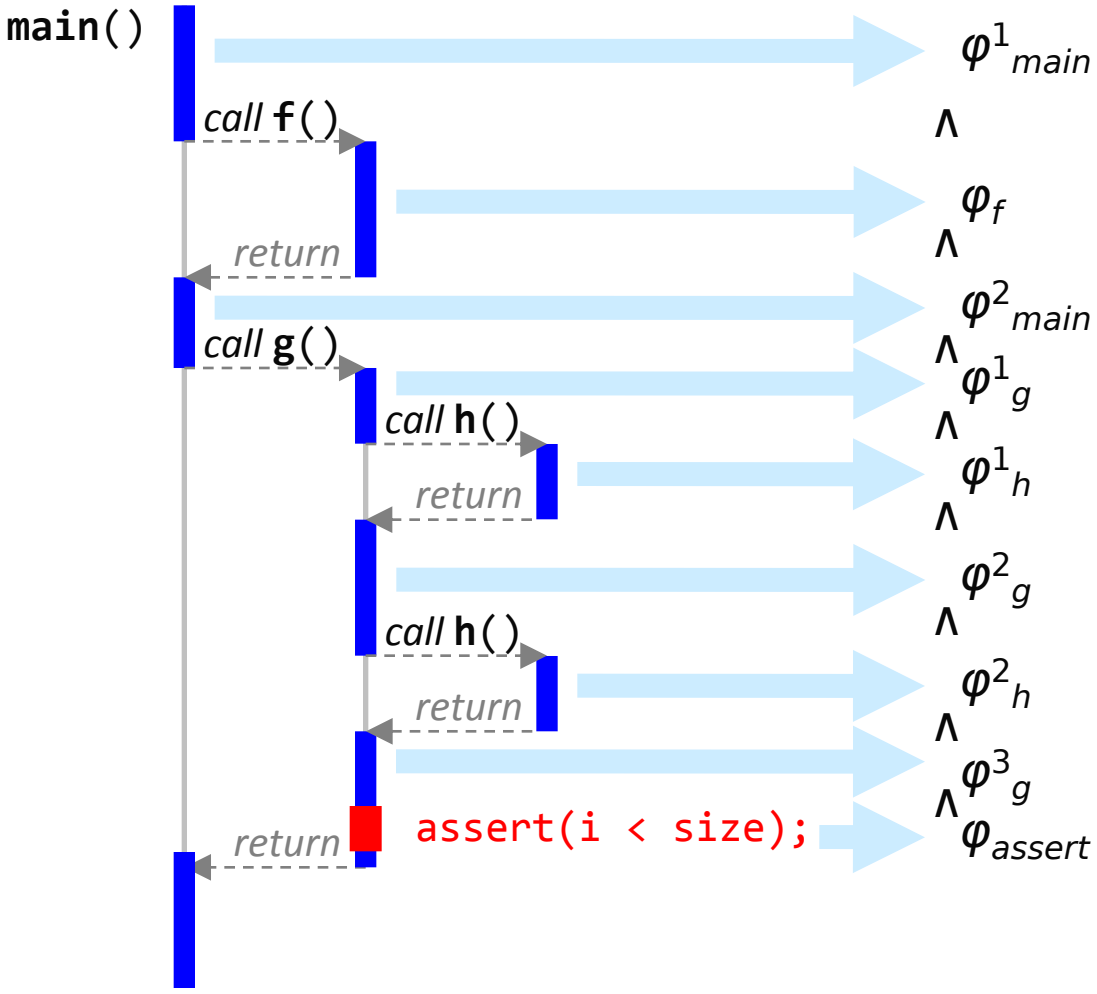
**Fact:**

- For unsatisfiable $A \wedge B$,
  - An interpolant always exists
  - Construction from unsatisfiability proof of $A \wedge B$

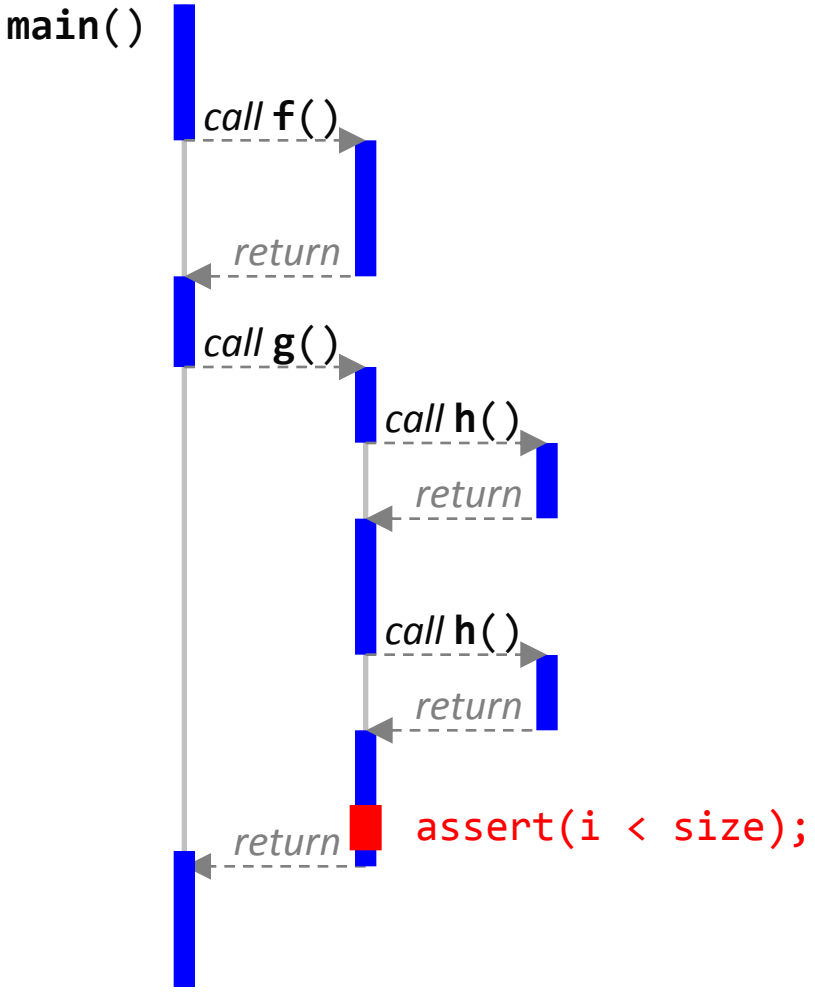
# Function summaries



# Function summaries

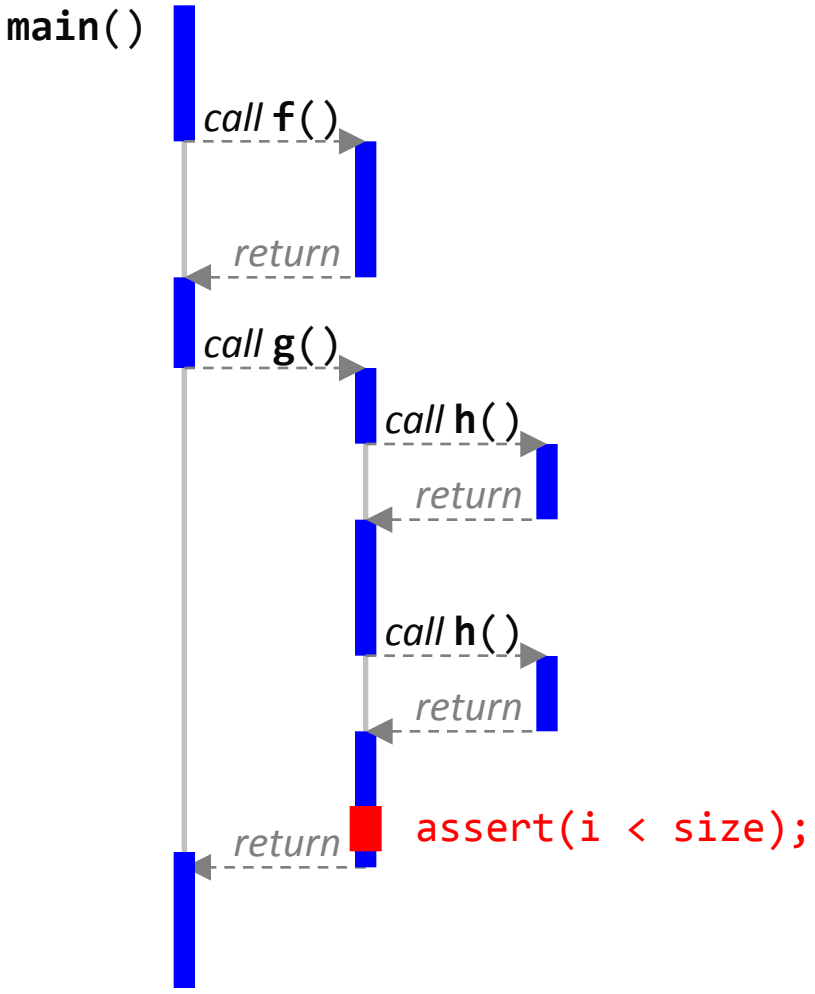


# Function summaries




$$\varphi^1_{main} \wedge \varphi_f \wedge \varphi^2_{main} \wedge \varphi^1_g \wedge \varphi^1_h \wedge \varphi^2_g \wedge \varphi^2_h \wedge \varphi^3_g \wedge \varphi_{assert}$$

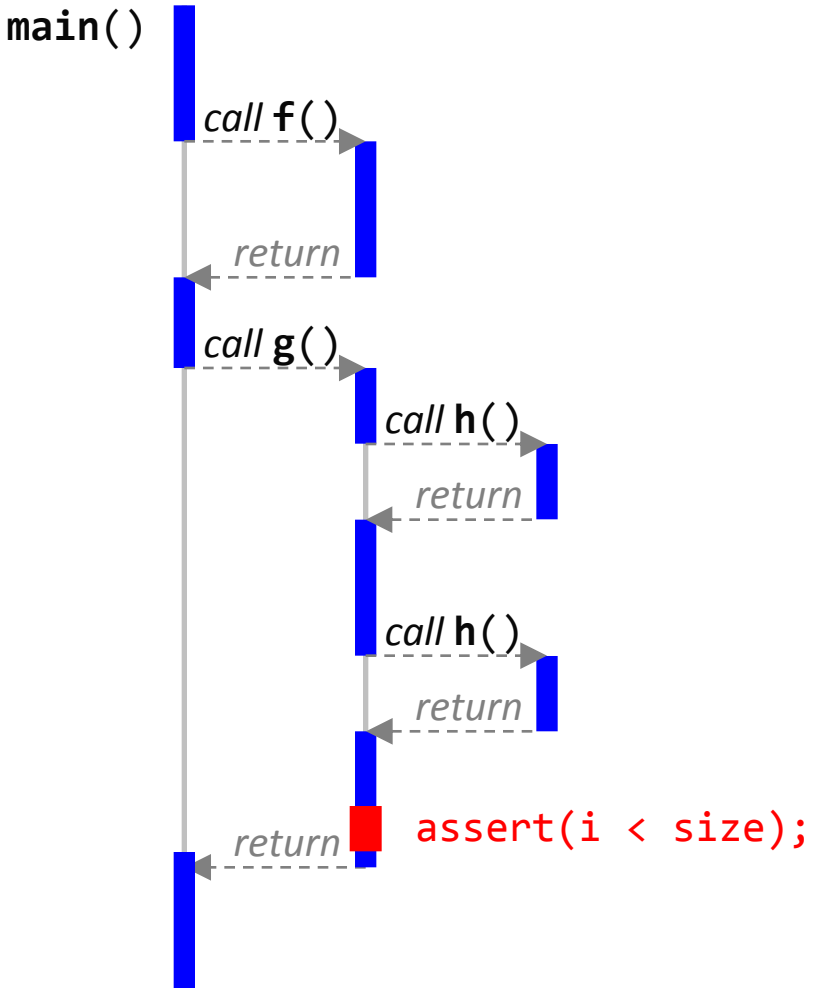
# Function summaries




$$\varphi^1_{main} \wedge \varphi_f \wedge \varphi^2_{main} \wedge \varphi^1_g \wedge \varphi^1_h \wedge \varphi^2_g \wedge \varphi^2_h \wedge \varphi^3_g \wedge \varphi_{assert}$$

*UNSAT* ⇔ *assertion holds*

# Function summaries

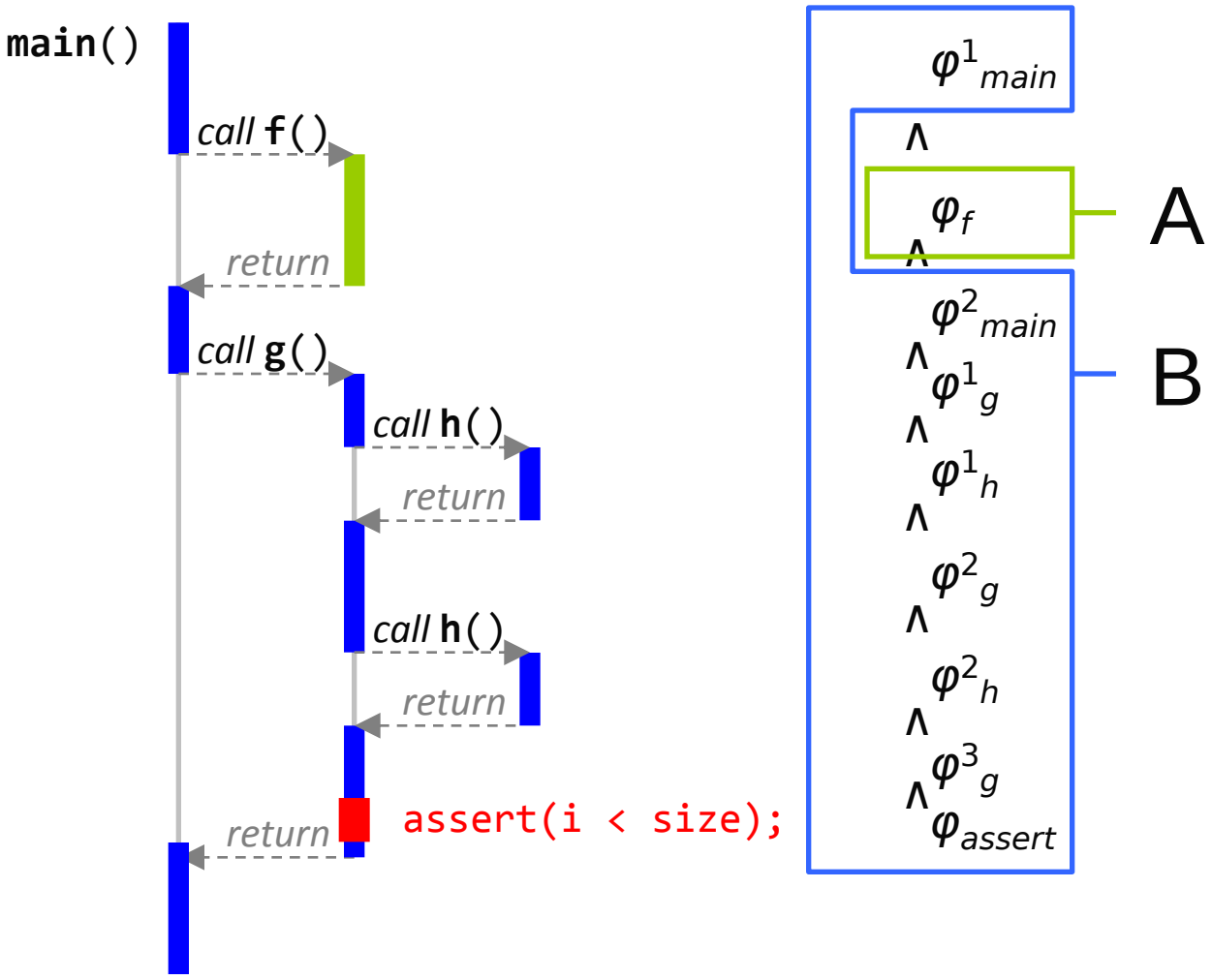



$$\varphi^1_{main} \wedge \varphi_f \wedge \varphi^2_{main} \wedge \varphi^1_g \wedge \varphi^1_h \wedge \varphi^2_g \wedge \varphi^2_h \wedge \varphi^3_g \wedge \varphi_{assert}$$

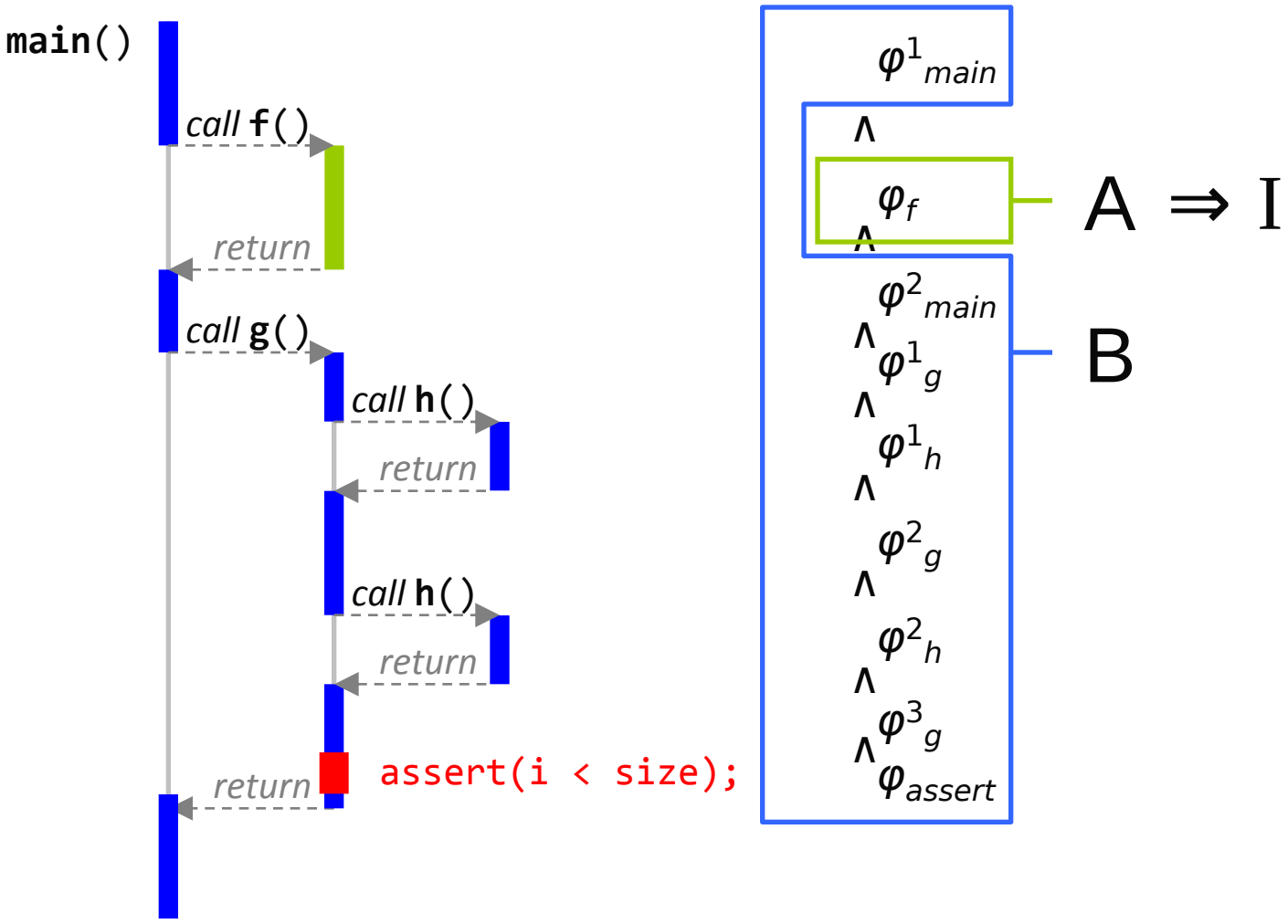
*proof of UNSAT*

*UNSAT* ⇔ *assertion holds*

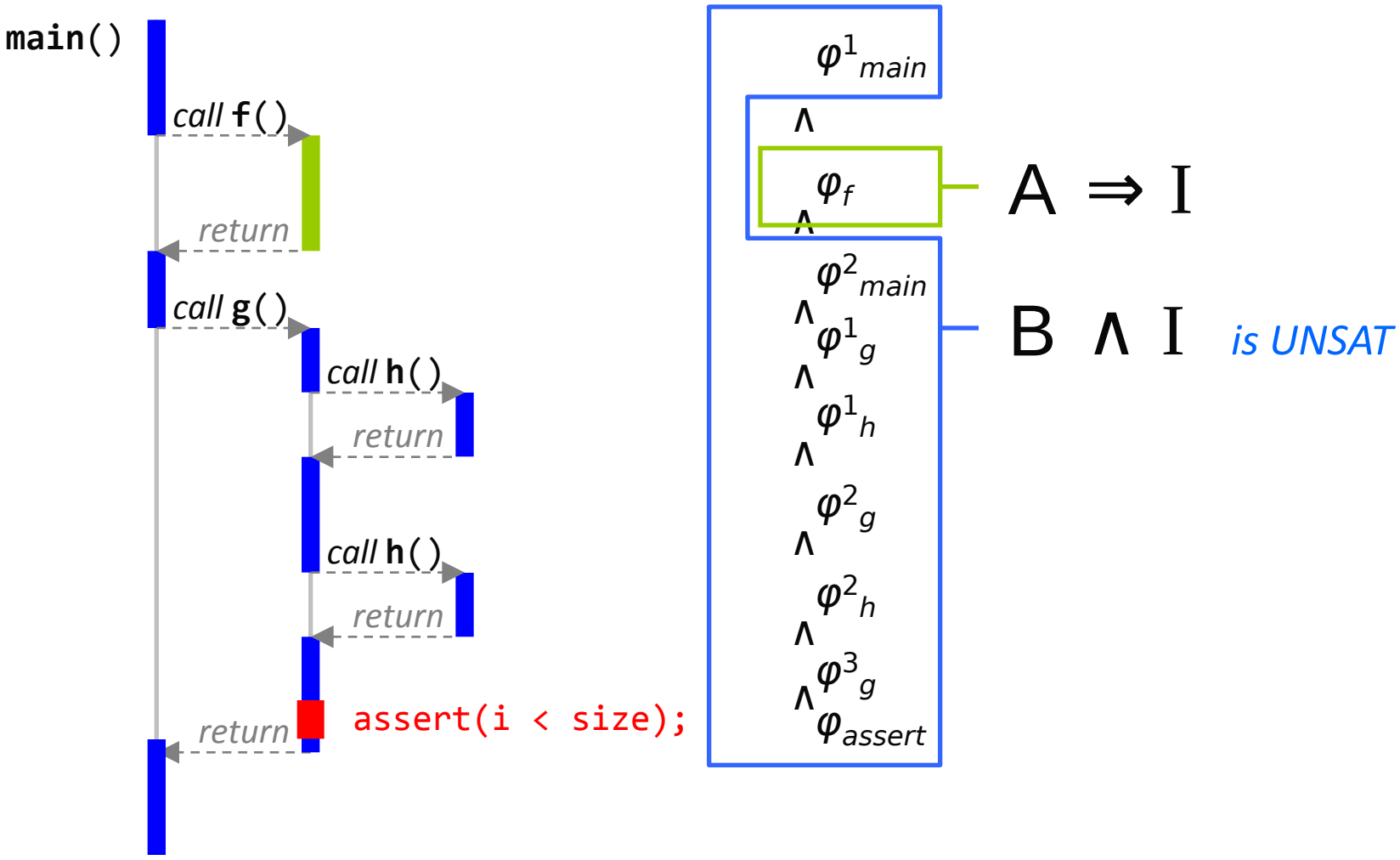
# Function summaries



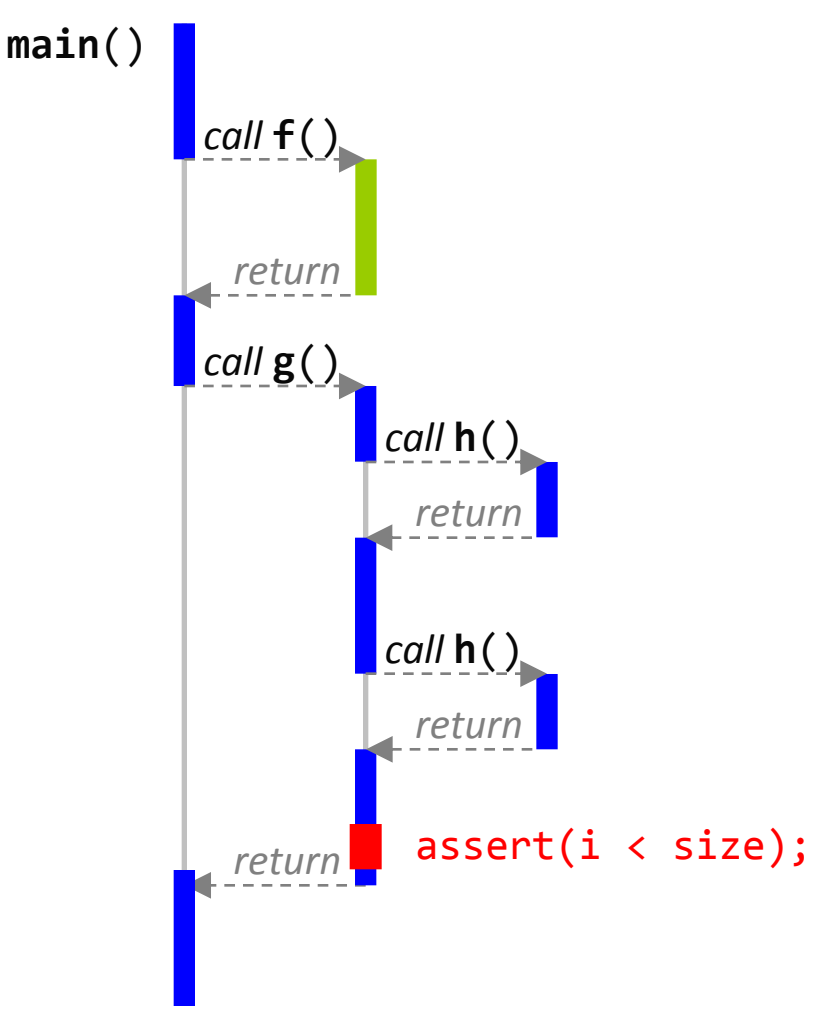
# Function summaries



# Function summaries



# Function summaries




$$\varphi^1_{main} \wedge \varphi_f \wedge \varphi^2_{main} \wedge \varphi^1_g \wedge \varphi^1_h \wedge \varphi^2_g \wedge \varphi^2_h \wedge \varphi^3_g \wedge \varphi_{assert}$$

$A \Rightarrow I$ *function summary*

$B \wedge I$ *is UNSAT*

# Encoding issue

***Issue:***

- BMC formula not in this form
  - We need common variables to be only the interface variables
  - Caller's variables propagate into the callee as a part of the path condition

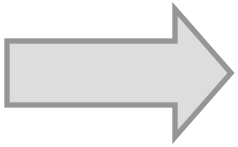
# Example: BMC formula

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```



$$
\begin{aligned}
& y_0 = 1 \wedge \\
& x_0 = nondet_0 \wedge \\
& a_0 = x_0 \wedge \\
& ret_0 = a_0 \wedge \\
& ret_1 = a_0 - 10 \wedge \\
& (x_0 > 0 \wedge a_0 < 10 \Rightarrow \\
& \qquad ret_2 = ret_0) \wedge \\
& (x_0 > 0 \wedge a_0 \geq 10 \Rightarrow \\
& \qquad ret_2 = ret_1) \wedge \\
& y_1 = ret_2 \wedge \\
& (x_0 > 0 \Rightarrow y_2 = y_1) \wedge \\
& (x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge \\
& y_2 < 0
\end{aligned}
$$

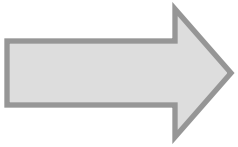
# Example: BMC formula

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```



$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$

main()

$ret_0 = a_0 \wedge$
$ret_1 = a_0 - 10 \wedge$
$(x_0 > 0 \wedge a_0 < 10 \Rightarrow ret_2 = ret_0) \wedge$
$(x_0 > 0 \wedge a_0 \geq 10 \Rightarrow ret_2 = ret_1) \wedge$

f()

$y_1 = ret_2 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$y_2 < 0$

main()

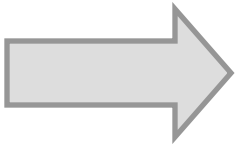
# Example: BMC formula

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```



$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$

main()

$ret_0 = a_0 \wedge$
$ret_1 = a_0 - 10 \wedge$
$(x_0 > 0 \wedge a_0 < 10 \Rightarrow ret_2 = ret_0) \wedge$
$(x_0 > 0 \wedge a_0 \geq 10 \Rightarrow ret_2 = ret_1) \wedge$

f()

$y_1 = ret_2 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$y_2 < 0$

main()

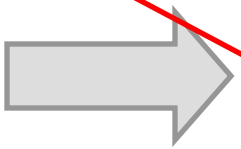
# Example: BMC formula

```
void main() {
  int y = 1;
  int x = nondet();

  if (x > 0)
    y = f(x);

  assert(y >= 0);
}

int f(int a) {
  if (a < 10)
    return a;
  return a - 10;
}
```



$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$

main()

$ret_0 = a_0 \wedge$
$ret_1 = a_0 - 10 \wedge$
$(x_0 > 0 \wedge a_0 < 10 \Rightarrow ret_2 = ret_0) \wedge$
$(x_0 > 0 \wedge a_0 \geq 10 \Rightarrow ret_2 = ret_1) \wedge$

f()

$y_1 = ret_2 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$y_2 < 0$

main()

# Encoding issue

***Issue:***

- BMC formula not in this form
  - We need common variables to be only the interface variables
  - Caller's variables propagate into the callee as a part of the path condition

# Encoding issue

***Issue:***

- BMC formula not in this form
  - We need common variables to be only the interface variables
  - Caller's variables propagate into the callee as a part of the path condition

***Solution:***

- Partitioned bounded model checking formula

  SSA form
  - PBMC using special symbols $callstart_f$ and $callend_f$
  - Bit-blasting

# Example: Partitioned BMC formula

$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$

$ret_0 = a_0 \wedge$
$ret_1 = a_0 - 10 \wedge$
$(x_0 > 0 \wedge a_0 < 10 \Rightarrow$
$\quad ret_2 = ret_0) \wedge$
$(x_0 > 0 \wedge a_0 \geq 10 \Rightarrow$
$\quad ret_2 = ret_1) \wedge$

$y_1 = ret_2 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$y_2 < 0$

# Example: Partitioned BMC formula

$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$

$y_1 = ret_2 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$y_2 < 0$

```
// Process f() later
...
```

# Example: Partitioned BMC formula

$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$
$(callstart_f \Leftrightarrow x_0 > 0) \wedge$

$y_1 = ret_0 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$(callend_f \vee x_0 \leq 0) \wedge y_2 < 0$

*// Process f() later*
*...*

# Example: Partitioned BMC formula

$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$
$(callstart_f \Leftrightarrow x_0 > 0) \wedge$

$y_1 = ret_0 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$(callend_f \vee x_0 \leq 0) \wedge y_2 < 0$

$ret_1 = a_0 \wedge$
$ret_2 = a_0 - 10 \wedge$
$(callstart_f \wedge a_0 < 10 \Rightarrow ret_0 = ret_1) \wedge$
$(callstart_f \wedge a_0 \geq 10 \Rightarrow ret_0 = ret_2) \wedge$
$callend_f \Rightarrow callstart_f$

# Example: Partitioned BMC formula

$\varphi_{main}$

$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$
$(callstart_f \Leftrightarrow x_0 > 0) \wedge$

$y_1 = ret_0 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$(callend_f \vee x_0 \leq 0) \wedge y_2 < 0$

$\varphi_f$

$ret_1 = a_0 \wedge$
$ret_2 = a_0 - 10 \wedge$
$(callstart_f \wedge a_0 < 10 \Rightarrow ret_0 = ret_1) \wedge$
$(callstart_f \wedge a_0 \geq 10 \Rightarrow ret_0 = ret_2) \wedge$
$callend_f \Rightarrow callstart_f$

# Example: Partitioned BMC formula

$\varphi_{main}$

$y_0 = 1 \wedge$
$x_0 = nondet_0 \wedge$
$a_0 = x_0 \wedge$
$(callstart_f \Leftrightarrow x_0 > 0) \wedge$

$y_1 = ret_0 \wedge$
$(x_0 > 0 \Rightarrow y_2 = y_1) \wedge$
$(x_0 \leq 0 \Rightarrow y_2 = y_0) \wedge$
$(callend_f \vee x_0 \leq 0) \wedge y_2 < 0$

$\varphi_f$

$ret_1 = a_0 \wedge$
$ret_2 = a_0 - 10 \wedge$
$(callstart_f \wedge a_0 < 10 \Rightarrow ret_0 = ret_1) \wedge$
$(callstart_f \wedge a_0 \geq 10 \Rightarrow ret_0 = ret_2) \wedge$
$callend_f \Rightarrow callstart_f$

PBMC formula: $\varphi_f \wedge \varphi_{main}$

# Using the summaries

# Using the summaries

- Substitution scenario:
  - Ω: $F \rightarrow \{$ *havoc*, *sum*, *inline* $\}$
- Approximation level for functions
  - *havoc* – treated as nondeterministic
  - *sum* – an existent summary is used
  - *inline* – precise representation

# Using the summaries

- Substitution scenario:
  - Ω: $F \rightarrow \{$ *havoc*, *sum*, *inline* $\}$
- Approximation level for functions
  - *havoc* – treated as nondeterministic
  - *sum* – an existent summary is used
  - *inline* – precise representation
- Initial substitution scenario
  - Eager – *inline* functions w/o summaries
  - Lazy – *havoc* functions w/o summaries

# Refinement

***Issue:***

- Function summary is over-approximation

  → Spurious errors may be observed

# Refinement

***Issue:***

- Function summary is over-approximation
  → Spurious errors may be observed

***Solution:***

- Refinement based on analysis of the error trace

# Refinement strategy – *counter example guided*

- If PBMC formula for $\Omega_i$ is SAT →
  a) try to refine into $\Omega_{i+1}$
  b) real error found

# Refinement strategy – *counter example guided*

- If PBMC formula for $\Omega_i$ is SAT →
  a) try to refine into $\Omega_{i+1}$
  b) real error found

- Refine function calls (i.e., *sum/havoc* → *inline*)
  - SAT assignment of the PBMC formula
    - Defines error trace through *inlined* functions
    - Identifies *summarized* and *havoced* function calls along the trace ($callstart_f = true$)
  - Additional data dependency analysis shows direct influence on the violated assertion

# Refinement strategy – *counter example guided*

- If PBMC formula for $\Omega_i$ is SAT →
  a) try to refine into $\Omega_{i+1}$
  b) real error found

- Refine function calls (i.e., *sum/havoc* → *inline*)
  - SAT assignment of the PBMC formula
    - Defines error trace through *inlined* functions
    - Identifies *summarized* and *havoced* function calls along the trace ($callstart_f = true$)
  - Additional data dependency analysis shows direct influence on the violated assertion

- If no such function call is found → real error

# Implementation

- FunFrog www.verify.inf.usi.ch/funfrog
  - Based on the CProver framework
    - D. Kroening *et al.* www.cprover.org
  - Checking assertion violation in C code
  - Bounded model checking
  - Function summarization
  - Uses the OpenSMT solver
    - For SAT checks & interpolation
    - R. Bruttomesso *et al.* verify.inf.usi.ch/opensmt

# Benchmarks

- We compare FunFrog with state-of-the-art tools
  - CBMC, SATABS, and CPAchecker
  - Note that SATABS and CPAchecker do CEGAR
- Benchmark properties
  - Multiple assertions checked separately
  - All assertions hold, i.e., summarization is possible
- Benchmarks
  - Artificial examples: `art`*N*
  - VeriSec suite: `verisec`*N*
  - Windows device drivers: `kbfiltr`*N*, `diskperf`*N*, `floppy`*N*

# Results

| benchmark | | | FunFrog | | CBMC | SATABS | CPAchecker |
|---|---|---|---|---|---|---|---|
| name | #assert | LoC | total time | itp. time | | | |
| verisec1 | 2 | 63 | 0.020 | 0.002 | 0.003 | 1.004 | 2.851 |
| verisec2 | 2 | 101 | 0.515 | 0.005 | 0.016 | 0.003 | 0.896 |
| verisec3 | 2 | 81 | 0.093 | 0.004 | 0.011 | TO | 1.91 |
| art1 | 2 | 242 | 1.731 | 0.034 | 0.280 | 534.8 | 65.37 |
| art2 | 2 | 63 | 3.327 | 0.030 | 0.408 | 881.2 | WA |
| art3 | 4 | 120 | 1.811 | 0.076 | 2.112 | TO | WA |
| kbfiltr1 | 8 | 12253 | 6.718 | 0.003 | 5.742 | 106.457 | WA |
| kbfiltr2 | 5 | 12253 | 2.665 | 0.008 | 3.702 | 13.002 | WA |
| diskperf1 | 9 | 6321 | 5.284 | 0.037 | 20.309 | 433.74 | 15.045 |
| diskperf2 | 4 | 6321 | 43.486 | 2.005 | 11.620 | 1064.2 | 24.849 |
| floppy1 | 2 | 10259 | 2.196 | 0.001 | 18.028 | 2735.4 | 15.246 |
| floppy2 | 4 | 10259 | 2.283 | 0.003 | 53.801 | 1402.1 | 47.891 |
| floppy3 | 11 | 10259 | 45.073 | 0.006 | 99.512 | 2208.5 | 97.436 |

# Conclusion

- FunFrog better on larger examples
- On small examples FunFrog is outperformed by CBMC
  - Partitioning overhead
  - Small number of functions to summarize
  - Missing optimizations in FunFrog (e.g., constant propagation)
- Performance comparable with state-of-the-art tools
- Efficiency depends on mutual relevance of assertions

# Future work

- Connection with a loop invariant generation engine
  - e.g., LoopFrog
- Enhancing summaries with locking information
  - As done in Zing
- Use summaries for upgrade checking

# *Thank you for your attention!*

***Problem:***

- Re-verification of already analyzed code

***Our take:***

- Extraction of function summaries
  - Over-approximation using *Craig interpolation*
- Reuse in subsequent verification runs
- Refinement
  - If the summaries are too weak
  - Based on analysis of the error trace