Specification-Driven Generation of Summaries for Symbolic Execution

Rafael Gonçalves, Frederico Ramos, Pedro Adão, José Fragoso Santos

Carnegie Mellon University

inesc id lisboa

TÉCNICO LISBOA

fct Fundação para a Ciência e a Tecnologia

Specification-Driven Generation of Summaries for Symbolic Execution

Rafael Gonçalves, Frederico Ramos, Pedro Adão, José Fragoso Santos

Carnegie Mellon University

inesc id lisboa

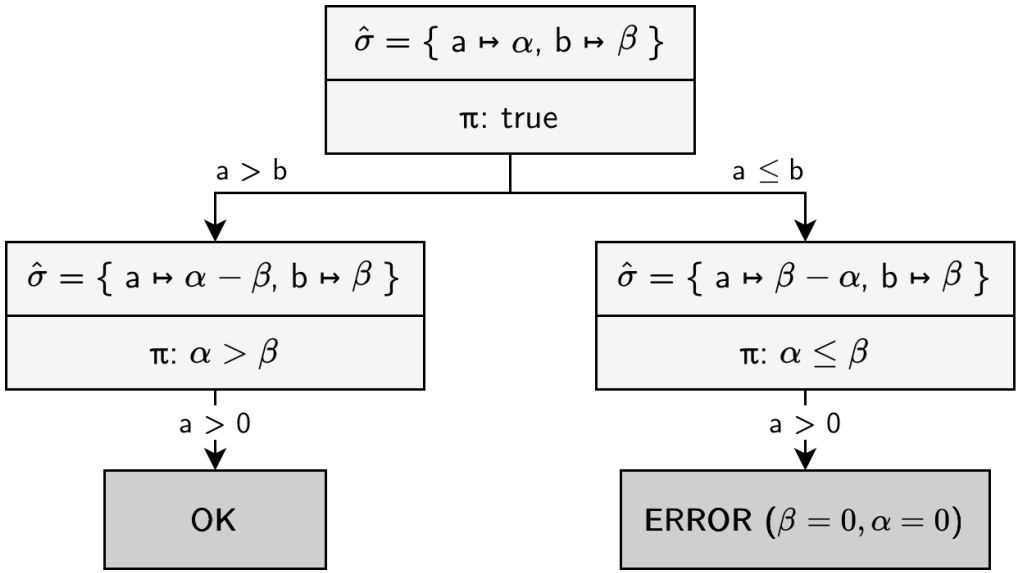
TÉCNICO LISBOA

fct Fundação para a Ciência e a Tecnologia

Symbolic Execution (SE)

- Execute programs with **symbolic values** instead of concrete values

```
1  void foo(int a, int b) {
2      if (a > b) {
3          a = a - b;
4      } else {
5          a = b - a;
6      }
7      assert(a > 0);
8  }
```



Many success stories!

angr

KLEE

...

Challenges

- Interactions with the runtime environment

```
printf("%s\n", foo);
```

```
scanf("%s", &bar);
```

- Path explosion

```
if (a > b) {
    if (a > c) { ... }
}
```

```
while (i < j) {
    ...
}
```

Challenges

- Interactions with the runtime environment

```
printf("%s\n", foo);
```

```
scanf("%s", &bar);
```

- Path explosion

```
if (a > b) {
    if (a > c) { ... }
}
```

```
while (i < j) {
    ...
}
```

Challenges

- Interactions with the runtime environment

```
printf("%s\n", foo);
```

```
scanf("%s", &bar);
```

- Path explosion

```
if (a > b) {
    if (a > c) { ... }
}
```

```
while (i < j) {
    ...
}
```

Specification-Driven Generation of Summaries for Symbolic Execution

Rafael Gonçalves, Frederico Ramos, Pedro Adão, José Fragoso Santos

Carnegie Mellon University

inesc id lisboa

TÉCNICO LISBOA

fct Fundação para a Ciência e a Tecnologia

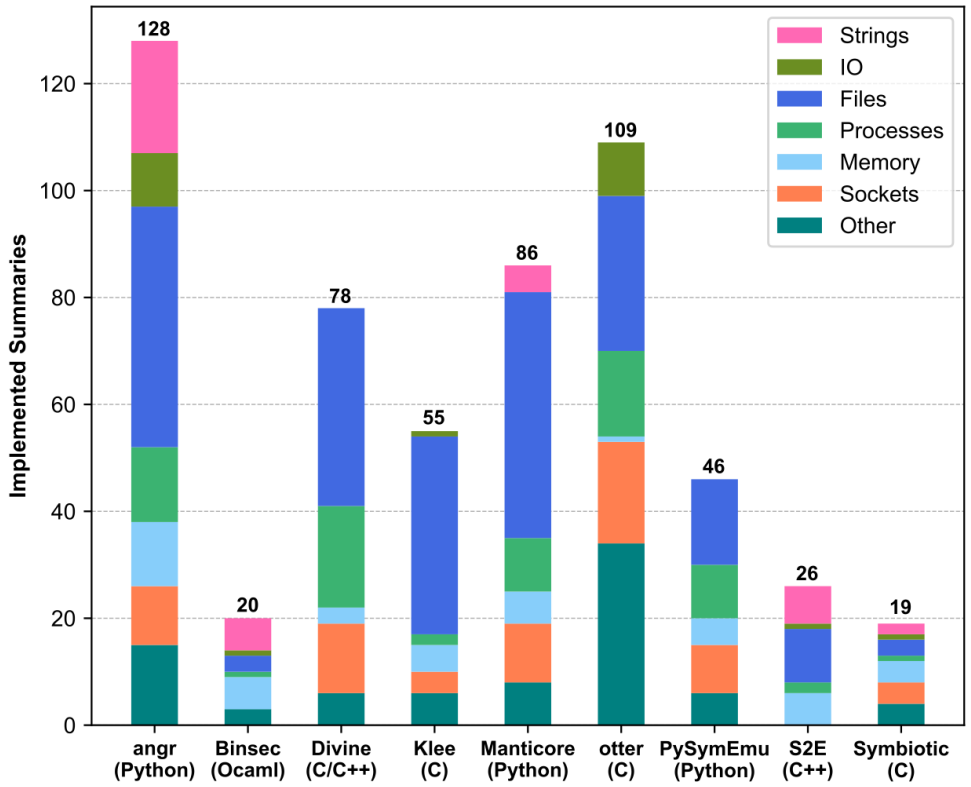
Why summaries?

- Model SE of concrete functions by directly interacting with the symbolic state

Avoid path explosion!

Why summaries?

- Model SE of concrete functions by directly interacting with the symbolic state
- Summaries are widely adopted…



But…

Summaries are often buggy!

And in general, hard to get right

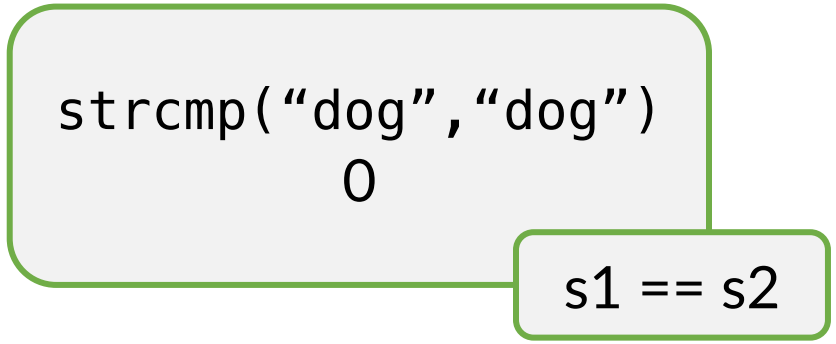
```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```

Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```



Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```

strcmp(“dog”,“dog”)
0

strcmp(“dog”,“cat”)
1

s1 > s2

Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```

$\hat{s}_1 \mapsto$ | $\hat{c}_1$ | $\hat{c}_2$ | $\hat{s}_2 \mapsto$ | $\hat{c}_3$ | $\hat{c}_4$ | $\hat{c}_1 \ldots \hat{c}_4$ symbolic

$$\texttt{strcmp}(\hat{s}_1, \hat{s}_2)$$

$$ITE\left(\hat{c}_1 \neq \hat{c}_3, \hat{c}_1 - \hat{c}_3, ITE\left(\hat{c}_2 \neq \hat{c}_4, \hat{c}_2 - \hat{c}_4, 0\right)\right)$$

Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```

$\hat{s}_1 \mapsto \boxed{\hat{c}_1 \mid \hat{c}_2}$ $\hat{s}_2 \mapsto \boxed{\hat{c}_3 \mid \hat{c}_4}$ $\hat{c}_1 \ldots \hat{c}_4$ symbolic

$$\texttt{strcmp}(\hat{s}_1, \hat{s}_2)$$

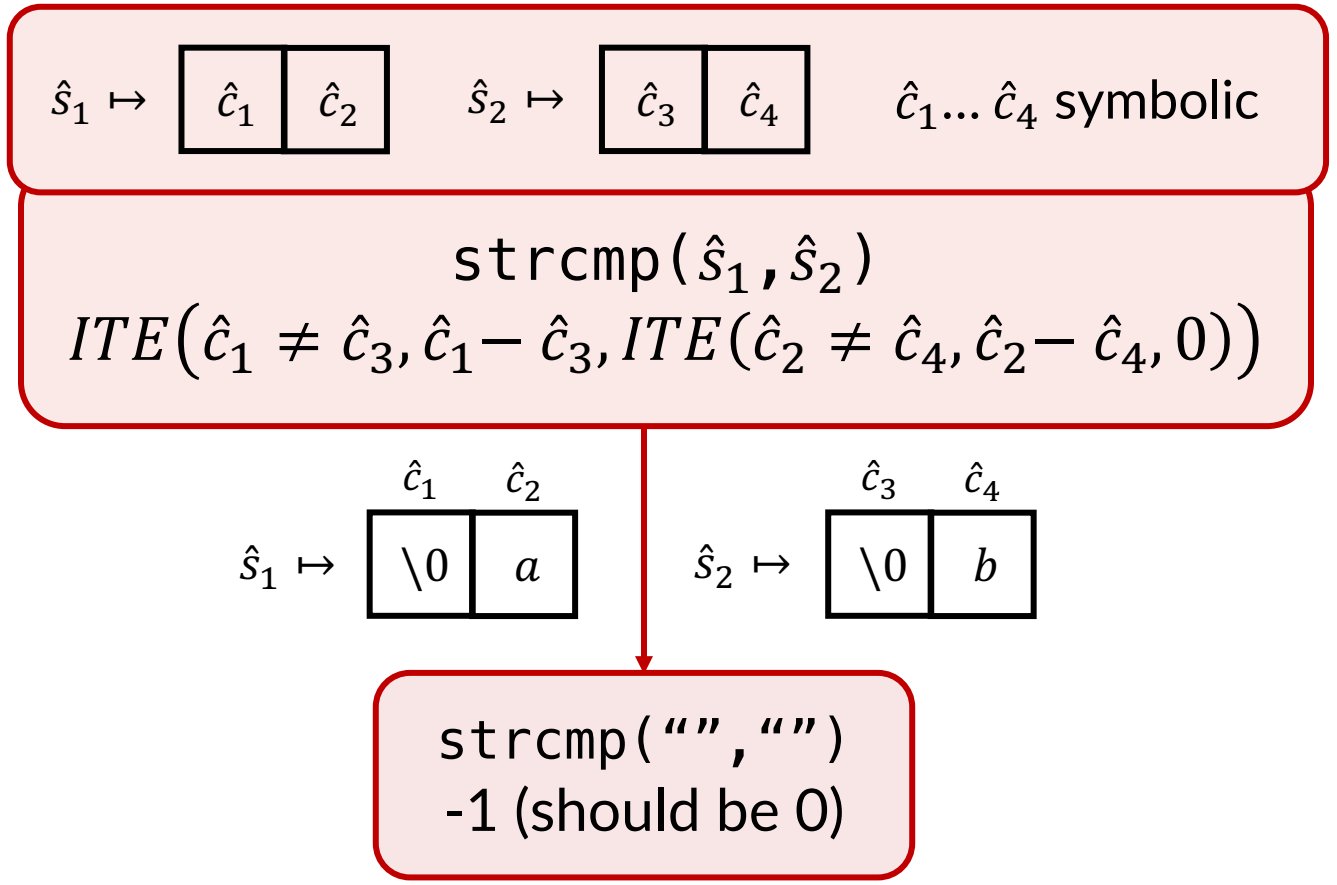
$$ITE\left(\hat{c}_1 \neq \hat{c}_3, \hat{c}_1 - \hat{c}_3, ITE\left(\hat{c}_2 \neq \hat{c}_4, \hat{c}_2 - \hat{c}_4, 0\right)\right)$$

Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```



Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 != c2:
17       ret = c1 - c2
18
19   return ret
```

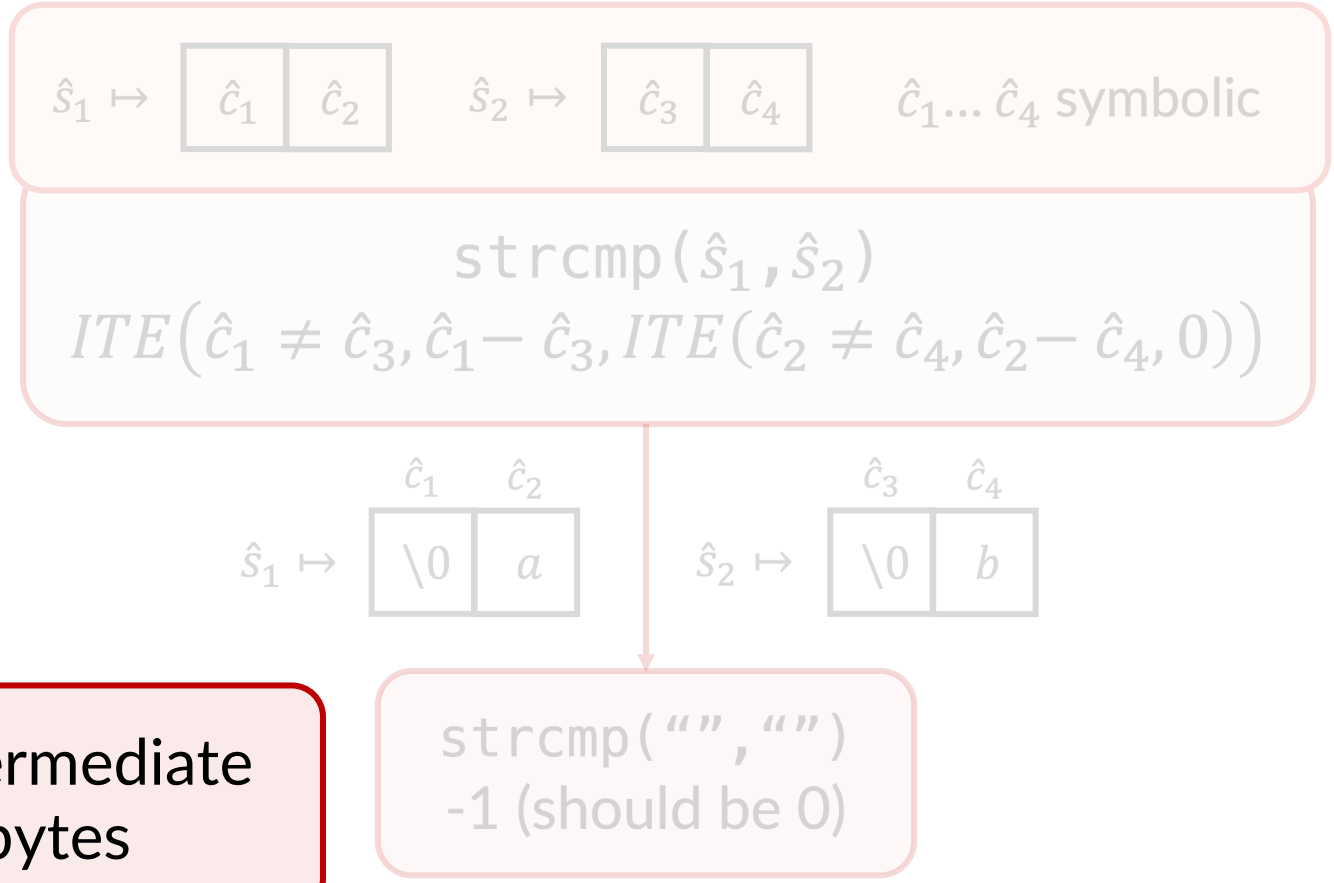

Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 !=
17       ret = c1
18
19   return ret
```

Not stopping at intermediate (symbolic) null bytes



Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Summaries are often buggy (in a subtle way)!

And in general, hard to get right

```
1  def strcmp(s1, s2):
2    zero_idx1 = api.find_zero(s1)
3    zero_idx2 = api.find_zero(s2)
4    min_zero_idx = min(zero_idx1, zero_idx2)
5
6    ret = s1[min_zero_idx] - s2[min_zero_idx]
7
8    for i in range(min_zero_idx - 1, -1, -1):
9      c1 = s1[i]
10     c2 = s2[i]
11
12     if api.is_symb(c1) or api.is_symb(c2):
13       cnstr = c1 != c2
14       ret = api.mk_ite(cnstr, c1 - c2, ret)
15
16     elif c1 !=
17       ret = c1
18
19   return ret
```

$\hat{s}_1 \mapsto \boxed{\hat{c}_1 \mid \hat{c}_2}$ $\hat{s}_2 \mapsto \boxed{\hat{c}_3 \mid \hat{c}_4}$ $\hat{c}_1 \ldots \hat{c}_4$ symbolic

$$\texttt{strcmp}(\hat{s}_1, \hat{s}_2)$$
$$ITE(\hat{c}_1 \neq \hat{c}_3, \hat{c}_1 - \hat{c}_3, ITE(\hat{c}_2 \neq \hat{c}_4, \hat{c}_2 - \hat{c}_4, 0))$$

$\hat{s}_1 \mapsto \boxed{\hat{c}_1 = \backslash 0 \mid \hat{c}_2 = a}$ $\hat{s}_2 \mapsto \boxed{\hat{c}_3 = \backslash 0 \mid \hat{c}_4 = b}$

strcmp(“”,“”)
-1 (should be 0)

Not stopping at intermediate (symbolic) null bytes

Manticore. `strcmp` summary. https://github.com/trailofbits/manticore/blob/master/manticore/native/models.py\#L99

Specification-Driven Generation of Summaries for Symbolic Execution

Rafael Gonçalves, Frederico Ramos, Pedro Adão, José Fragoso Santos

Carnegie Mellon University

inesc id lisboa

TÉCNICO LISBOA

fct Fundação para a Ciência e a Tecnologia

Pipeline at a glance

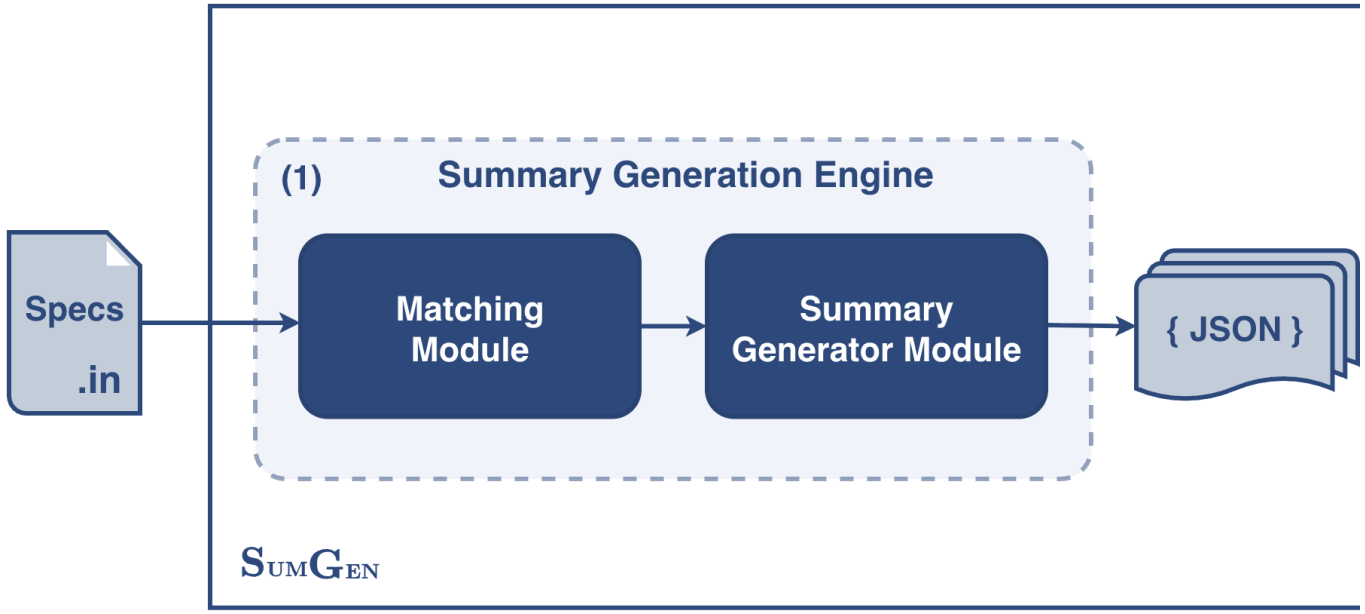
Goal: Automatically generate **tool-independent**, **correct-by-construction** summaries

Pipeline at a glance

Goal: Automatically generate **tool-independent**, **correct-by-construction** summaries

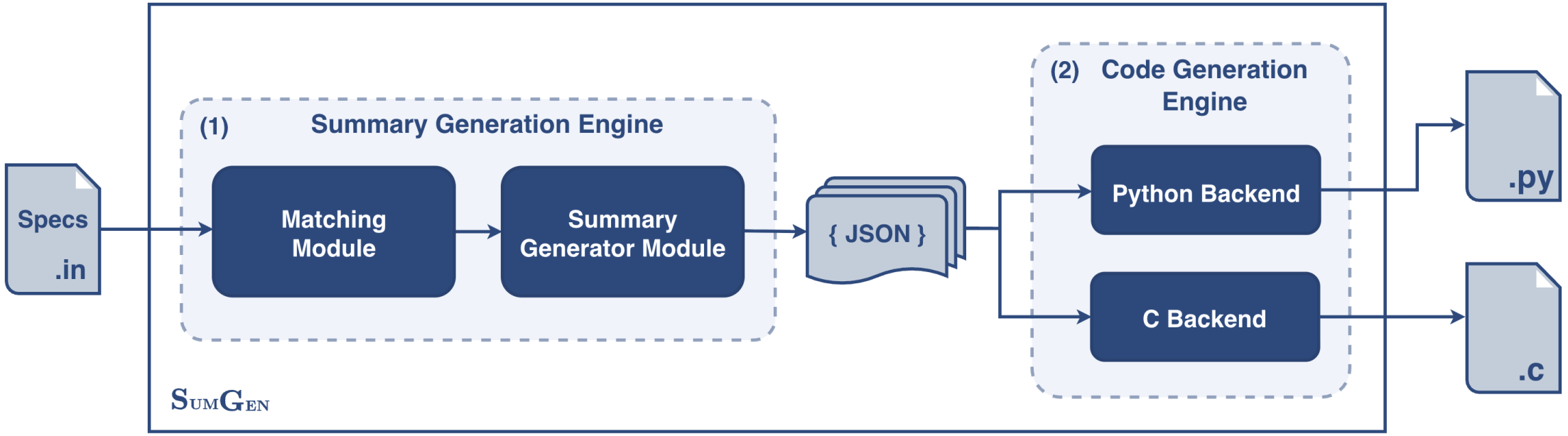
Pipeline at a glance

Goal: Automatically generate **tool-independent**, **correct-by-construction** summaries



Pipeline at a glance

Goal: Automatically generate **tool-independent**, **correct-by-construction** summaries



A running example: `strcmp`

Step 0: 

A running example: `strcmp`

Step 0: 

Prelim: What is the "difference" (δ) between two strings $s1$ and $s2$?

$$
\begin{aligned}
&\mathrm{strd}(s1, s2, \delta) \triangleq \\
&\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 = \text{`\textbackslash 0'} \vee c2 = \text{`\textbackslash 0'} \vee c1 \neq c2) \circledast \delta = c1 - c2 \\
&\quad \vee \\
&\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 \neq \text{`\textbackslash 0'} \wedge c2 \neq \text{`\textbackslash 0'} \wedge c1 = c2) \circledast \mathrm{strd}(s1 + 1, s2 + 1, \delta)
\end{aligned}
$$

A running example: `strcmp`

Step 0: 

Prelim: What is the "difference" (δ) between two strings $s1$ and $s2$?

$$\begin{array}{l}\mathrm{strd}(s1, s2, \delta) \triangleq \\ \quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \uplus \delta = c1 - c2 \\ \quad \vee \\ \quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \uplus \mathrm{strd}(s1 + 1, s2 + 1, \delta)\end{array}$$

(Base)

A running example: `strcmp`

Step 0: 

Prelim: What is the "difference" (δ) between two strings $s1$ and $s2$?

$$\begin{aligned}&\mathrm{strd}(s1, s2, \delta) \triangleq \\ &\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \circledast \delta = c1 - c2 \\ &\quad \vee \\ &\quad \boxed{s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2)} \circledast \boxed{\mathrm{strd}(s1 + 1, s2 + 1, \delta)}\end{aligned}$$

(Recursive)

A running example: `strcmp`

Step 0: 

Prelim: What is the "difference" (δ) between two strings $s1$ and $s2$?

$$\begin{aligned}
&\mathrm{strd}(s1, s2, \delta) \triangleq \\
&\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \circledast \delta = c1 - c2 \\
&\quad \vee \\
&\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \circledast \mathrm{strd}(s1 + 1, s2 + 1, \delta)
\end{aligned}$$

Overlapping conjuction

A running example: `strcmp`

Step 0: 

Prelim: What is the "difference" (δ) between two strings $s1$ and $s2$?

$$\begin{array}{l}\mathrm{strd}(s1, s2, \delta) \triangleq \\ \quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \circledast \delta = c1 - c2 \\ \quad \vee \\ \quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \circledast \mathrm{strd}(s1 + 1, s2 + 1, \delta)\end{array}$$

Then: If $s1 - s2 = \delta$, `strcmp(s1,s2)` returns δ

$$\{\mathrm{strd}(s1, s2, \delta)\}\ \mathsf{int}\ \mathsf{strcmp}(\mathsf{char} * \mathsf{s1},\ \mathsf{char} * \mathsf{s2})\ \{\mathsf{ret} = \delta\}$$

Generating a summary for `strcmp`

$$\{\mathrm{strd}(s1, s2, \delta)\}\ \mathsf{int}\ \mathsf{strcmp}(\mathsf{char} * \mathsf{s1},\ \mathsf{char} * \mathsf{s2})\ \{\mathsf{ret} = \delta\}$$

Generating a summary for `strcmp`

$$\{\mathrm{strd}(s1, s2, \delta)\}\ \text{int strcmp(char} * \text{s1, char} * \text{s2)}\ \{\text{ret} = \delta\}$$

Goal: Compute the return value (δ)

Generating a summary for `strcmp`

$\{\mathrm{strd}(s1, s2, \delta)\}$ int strcmp(char $*$ s1, char $*$ s2) $\{\mathrm{ret} = \delta\}$

Goal: Compute the return value (δ)

```
fn strcmp(s1, s2) {
    δ ← ?
    return δ
}
```

Generating a summary for `strcmp`

$\{\boxed{\mathrm{strd}(s1, s2, \delta)}\}$ int strcmp(char $*$ s1, char $*$ s2) $\{\mathsf{ret} = \delta\}$

Goal: Compute the return value (δ)

fn strcmp($s1, s2$) {
    $\delta \leftarrow fold_{\mathrm{strd}}(s1, s2)$;
    return δ
}

Generating a summary for `strcmp`

$\{\mathrm{strd}(s1, s2, \delta)\}$ int strcmp(char $*$ s1, char $*$ s2) $\{\mathrm{ret} = \delta\}$

Goal: Compute the return value (δ)

fn strcmp($s1, s2$) {
    $\delta \leftarrow fold_{\mathrm{strd}}(s1, s2)$;
    return δ
}

How to generate $fold_{\mathrm{strd}}$?

Matching Trees

Step 1: Matching Module

$$\begin{aligned}&\mathrm{strd}(s1, s2, \delta) \triangleq \\ &\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \circledast \delta = c1 - c2 \\ &\quad \vee \\ &\quad s1 \mapsto c1 \circledast s2 \mapsto c2 \circledast (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \circledast \mathrm{strd}(s1 + 1, s2 + 1, \delta)\end{aligned}$$

How to “bring” predicate close to executable code? **Matching Trees**

Matching Trees

Step 1: Matching Module

$$\begin{aligned}&\mathrm{strd}(s1, s2, \delta) \triangleq \\ &\quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \uplus \delta = c1 - c2 \\ &\quad \vee \\ &\quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \uplus \mathrm{strd}(s1 + 1, s2 + 1, \delta)\end{aligned}$$

Key idea: Learn new bindings from what we already know

a) Learn $c1$, $c2$ from $s1$, $s2$

b) Branch on $c1 = \text{'}\backslash 0\text{'} \vee c2 = \text{'}\backslash 0\text{'} \vee c1 \neq c2$

c) Learn δ in each of the branches

Matching Trees

Step 1: Matching Module

$$\mathrm{strd}(s1, s2, \delta) \triangleq$$
$$\boxed{s1 \mapsto c1 \circledast s2 \mapsto c2} \circledast (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \circledast \delta = c1 - c2$$
$$\vee$$
$$\boxed{s1 \mapsto c1 \circledast s2 \mapsto c2} \circledast (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \circledast \mathrm{strd}(s1 + 1, s2 + 1, \delta)$$

Key idea: Learn new bindings from what we already know

a) Learn $c1$, $c2$ from $s1$, $s2$

b) Branch on $c1 = \text{'}\backslash 0\text{'} \vee c2 = \text{'}\backslash 0\text{'} \vee c1 \neq c2$

c) Learn δ in each of the branches

$s1 \mapsto c1$
|
$s2 \mapsto c2$
|
?

Matching Trees

Step 1: Matching Module

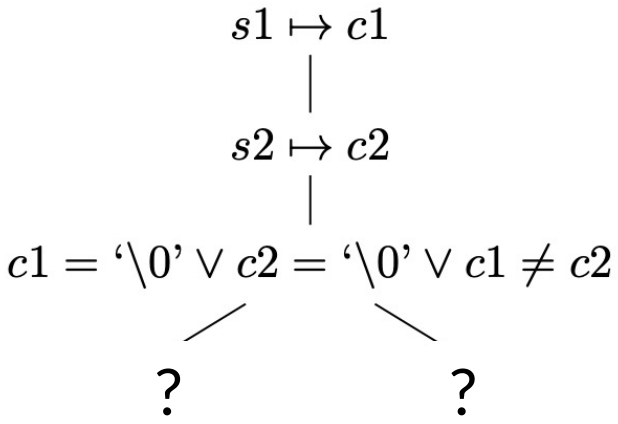
$$\begin{aligned}&\mathrm{strd}(s1, s2, \delta) \triangleq \\ &\quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \uplus \delta = c1 - c2 \\ &\quad \vee \\ &\quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \uplus \mathrm{strd}(s1 + 1, s2 + 1, \delta)\end{aligned}$$

Key idea: Learn new bindings from what we already know

a) Learn $c1$, $c2$ from $s1$, $s2$

b) Branch on $c1 = \text{'}\backslash 0\text{'} \vee c2 = \text{'}\backslash 0\text{'} \vee c1 \neq c2$

c) Learn δ in each of the branches



Matching Trees

Step 1: Matching Module

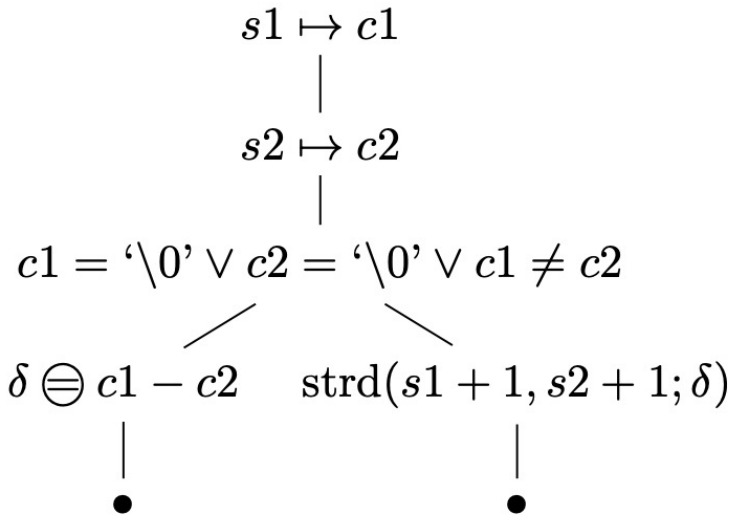
$$\begin{aligned}&\mathrm{strd}(s1, s2, \delta) \triangleq \\ &\quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 = \text{`}\backslash 0\text{'} \vee c2 = \text{`}\backslash 0\text{'} \vee c1 \neq c2) \uplus \boxed{\delta = c1 - c2} \\ &\quad \vee \\ &\quad s1 \mapsto c1 \uplus s2 \mapsto c2 \uplus (c1 \neq \text{`}\backslash 0\text{'} \wedge c2 \neq \text{`}\backslash 0\text{'} \wedge c1 = c2) \uplus \boxed{\mathrm{strd}(s1 + 1, s2 + 1, \delta)}\end{aligned}$$

Key idea: Learn new bindings from what we already know

a) Learn $c1$, $c2$ from $s1$, $s2$

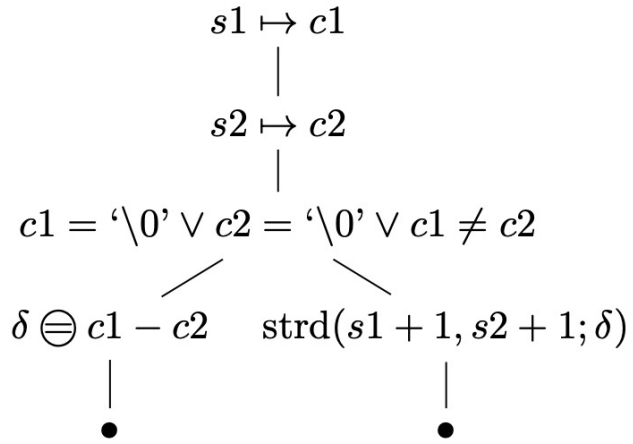
b) Branch on $c1 = \text{'}\backslash 0\text{'} \vee c2 = \text{'}\backslash 0\text{'} \vee c1 \neq c2$

c) Learn δ in each of the branches



Generating the fold function

Step 2: **Summary Generator**



$\text{fn } fold^{\text{EX}}_{\text{strd}}(s1, s2)\ \{$

?

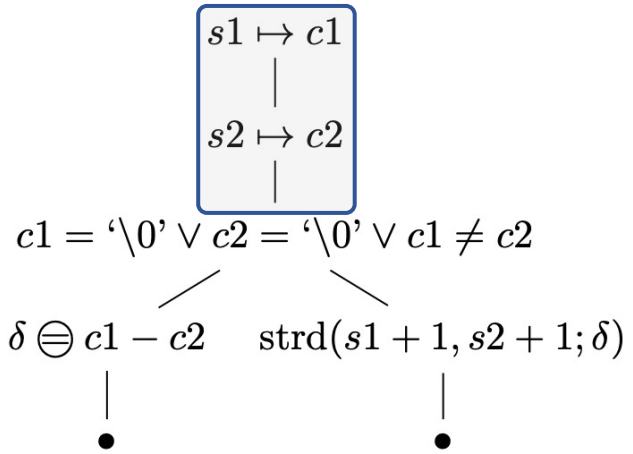
$\}$

Generating the fold function

Step 2:

Summary Generator

- $c1$, $c2$ reads translated directly



$$\text{fn } fold^{\text{EX}}_{\text{strd}}(s1, s2)\ \{$$
$$c1 \leftarrow *s1;$$
$$c2 \leftarrow *s2;$$

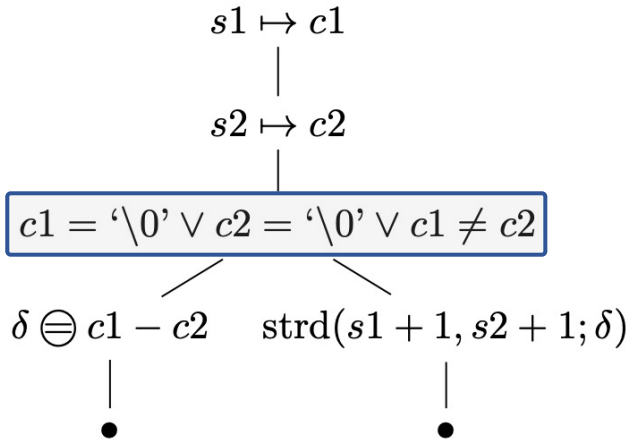
?

$$\}$$

Generating the fold function

Step 2: Summary Generator

- $c1, c2$ reads translated directly
- On branch…



$$\text{fn } fold^{\text{EX}}_{\text{strd}}(s1, s2)\ \{$$
$$c1 \leftarrow *s1;$$
$$c2 \leftarrow *s2;$$
$$\pi \leftarrow c1 = \text{`\0'} \vee c2 = \text{`\0'} \vee c1 \neq c2;$$

?

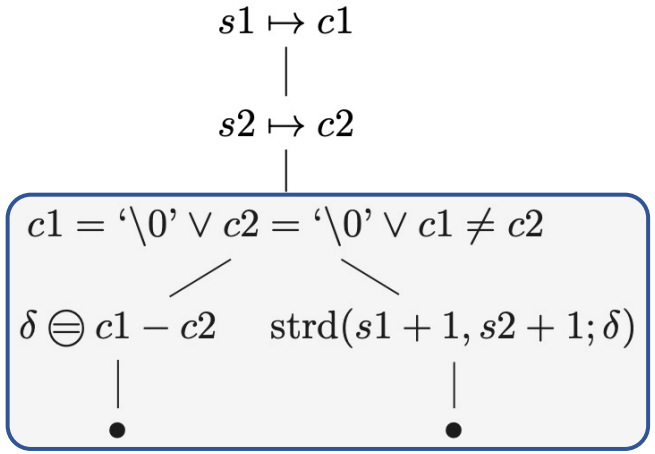
$$\}$$

Generating the fold function

Step 2:

Summary Generator

- $c1, c2$ reads translated directly
- On branch, translate each case using $\text{isCertain}(e)$[1]

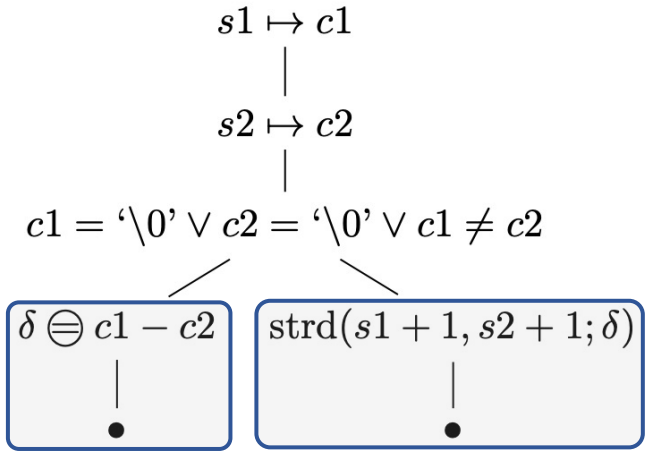


$$
\begin{aligned}
&\text{fn } fold^{\text{EX}}_{\text{strd}}(s1, s2) \{ \\
&\quad c1 \leftarrow *s1; \\
&\quad c2 \leftarrow *s2; \\
&\quad \pi \leftarrow c1 = \text{'\0'} \vee c2 = \text{'\0'} \vee c1 \neq c2; \\
&\quad \text{if } (\text{isCertain}(\pi)) \{ \delta \leftarrow c1 - c2 \} \\
&\quad \text{elif } (\text{isCertain}(\neg\pi)) \{ \\
&\qquad \delta \leftarrow fold^{\text{EX}}_{\text{strd}}(s1 + 1, s2 + 1) \\
&\quad \} \text{ else } \{ \\
&\qquad ? \\
&\quad \}; \\
&\quad \text{return } \delta \\
&\}
\end{aligned}
$$

[1] $\text{isCertain}(e) \triangleq \neg\, \text{isSAT}(\neg e)$

Generating the fold function

Step 2: Summary Generator



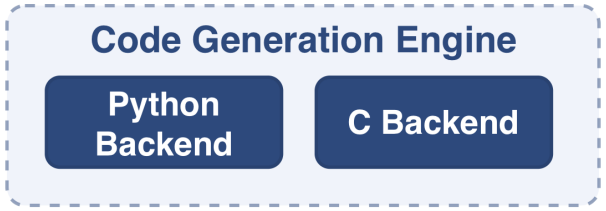
- $c1$, $c2$ reads translated directly
- On branch, translate each case using $\text{isCertain}(e)$[1]
- If either possible, default to branch modeling both cases through scoped calls
  - Run function under extended path condition

[1] $\text{isCertain}(e) \triangleq \neg\,\text{isSAT}(\neg e)$

$$
\begin{aligned}
&\text{fn } fold^{\text{EX}}_{\text{strd}}(s1, s2)\,\{\\
&\quad c1 \leftarrow *s1;\\
&\quad c2 \leftarrow *s2;\\
&\quad \pi \leftarrow c1 = \text{`\0'} \vee c2 = \text{`\0'} \vee c1 \neq c2;\\
&\quad \text{if } (\text{isCertain}(\pi))\,\{\,\delta \leftarrow c1 - c2\,\}\\
&\quad \text{elif } (\text{isCertain}(\neg\pi))\,\{\\
&\qquad \delta \leftarrow fold^{\text{EX}}_{\text{strd}}(s1 + 1, s2 + 1)\\
&\quad \}\ \text{else}\ \{\\
&\qquad \delta_1 \leftarrow \underline{fold^{\text{EX}}_{\text{strd}}(s1, s2)\ \text{with}\ \pi;}\\
&\qquad \delta_2 \leftarrow \underline{fold^{\text{EX}}_{\text{strd}}(s1, s2)\ \text{with}\ \neg\pi;}\\
&\qquad \delta \leftarrow \text{ITE}(\pi, \delta_1, \delta_2)\\
&\quad \};\\
&\quad \text{return } \delta\\
&\}
\end{aligned}
$$

Code generation for SE tools

Step 3:



- **Pipeline so far:** generate summaries in an intermediate representation
  - Can transpile to whichever SE backend/language the developer wishes to support!
- Two backends supported out of the box
  - *angr*[1] summaries transpiled to Python
  - Ramos et al.'s[2] tool-independent summaries transpiled to C

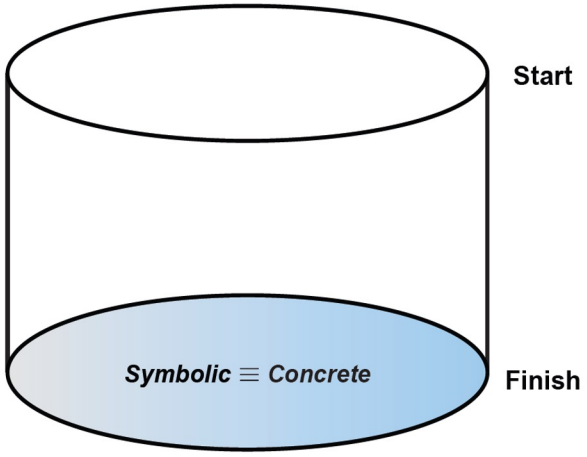
---

1 angr. *angr*. GitHub. https://github.com/angr/angr

2 Ramos, F., Sabino, N., Adão, P., Naumann, D. A., & Fragoso Santos, J.: Toward Tool-Independent Summaries for Symbolic Execution. ECOOP '23 (2023)

Summaries: Correctness Properties

- Summaries (and specs) can be:

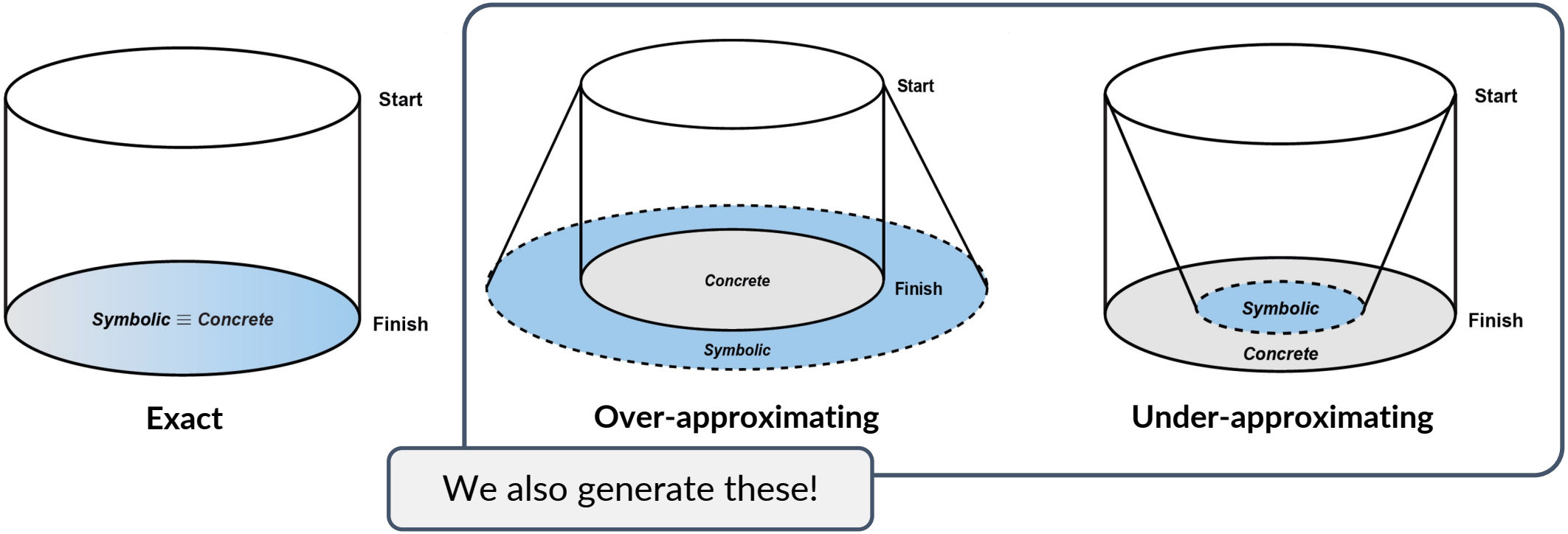


Exact

The **`strcmp`** spec/ summary we just saw!

Summaries: Correctness Properties

- Summaries (and specs) can be:



Summaries: Correctness

- Generated summaries are **correct-by-construction**

Summaries: Correctness

- Generated summaries are **correct-by-construction**

*If I write an **exact spec** and I apply the **summary generation procedure for exact summaries**, then I will get back an **exact summary***

Theorem 1 (Correctness of Generated Summaries). *Let* $\beta \in \{\mathrm{EX}, \mathrm{OX}, \mathrm{UX}\}$ *and* $\mathcal{S}^\beta$ *be the procedure that generates a summary. Suppose that:*

$$\mathcal{S}^\beta(\{P\}\ \mathsf{fn}\ f(\bar{x})\ \{Q; y; \pi\}) = \mathsf{fn}\ f(\bar{x})\ \{s\}$$

and $\{P\}\ \mathsf{fn}\ f(\bar{x})\ \{Q; y; \pi\}$ *is a* β*-specification. Then,* $\mathsf{fn}\ f(\bar{x})\ \{s\}$ *is a* β*-summary.*

Also for under-approximating, over-approximating

Evaluation

- Generated 131 summaries for 47 LIBC functions:
  - 42 exact, 47 over-approximating, 42 under-approximating
- Three main questions:
  - Are the generated summaries correct? **(EQ1)**
  - Are specs easier to write than summaries? **(EQ2)**
  - How well do the generated summaries perform? **(EQ3)**
- Experimental setup:
  - Ubuntu 18.04.5 LTS with Intel Xeon E5–2620 CPU and 32 GB of RAM
  - **Baseline:** *angr*, Ramos et al.'s tool-independent summaries

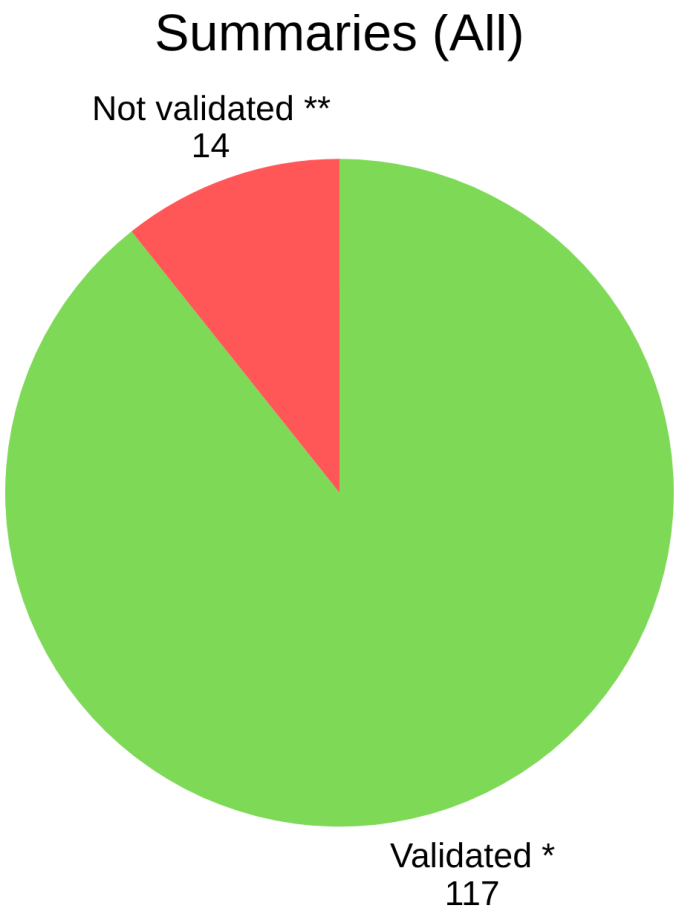
Evaluation

- Generated 131 summaries for 47 LIBC functions:
  - 42 exact, 47 over-approximating, 42 under-approximating
- Three main questions:
  - Are the generated summaries correct? **(EQ1)**
  - Are specs easier to write than summaries? **(EQ2)**
  - How well do the generated summaries perform? **(EQ3)**
- Experimental setup:
  - Ubuntu 18.04.5 LTS with Intel Xeon E5–2620 CPU and 32 GB of RAM
  - **Baseline:** *angr*, Ramos et al.'s tool-independent summaries

Evaluation

- Generated 131 summaries for 47 LIBC functions:
  - 42 exact, 47 over-approximating, 42 under-approximating
- Three main questions:
  - Are the generated summaries correct? **(EQ1)**
  - Are specs easier to write than summaries? **(EQ2)**
  - How well do the generated summaries perform? **(EQ3)**
- Experimental setup:
  - Ubuntu 18.04.5 LTS with Intel Xeon E5–2620 CPU and 32 GB of RAM
  - **Baseline:** *angr*, Ramos et al.'s tool-independent summaries

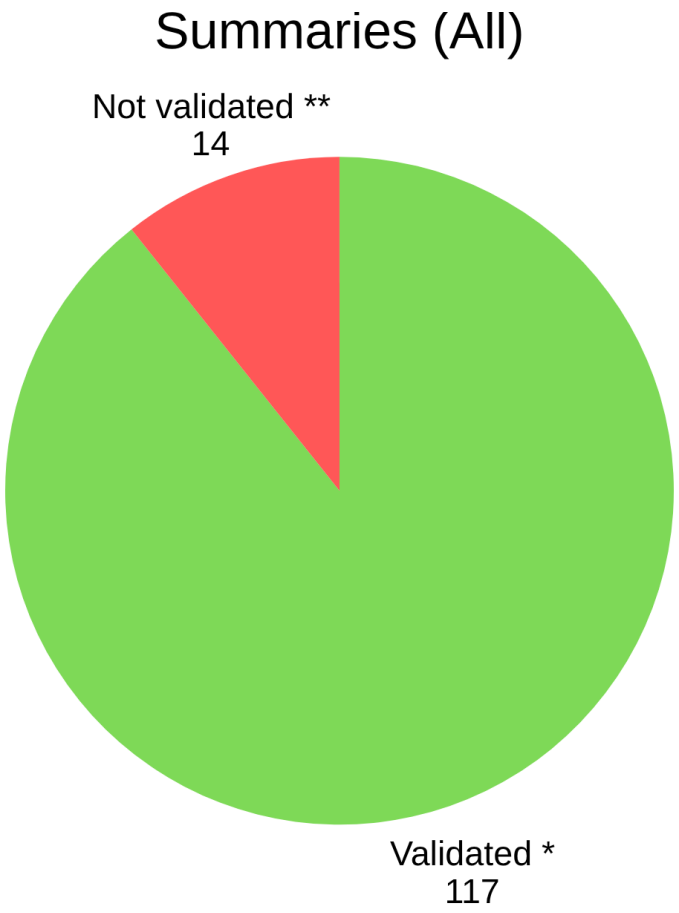
EQ1: Summary Correctness



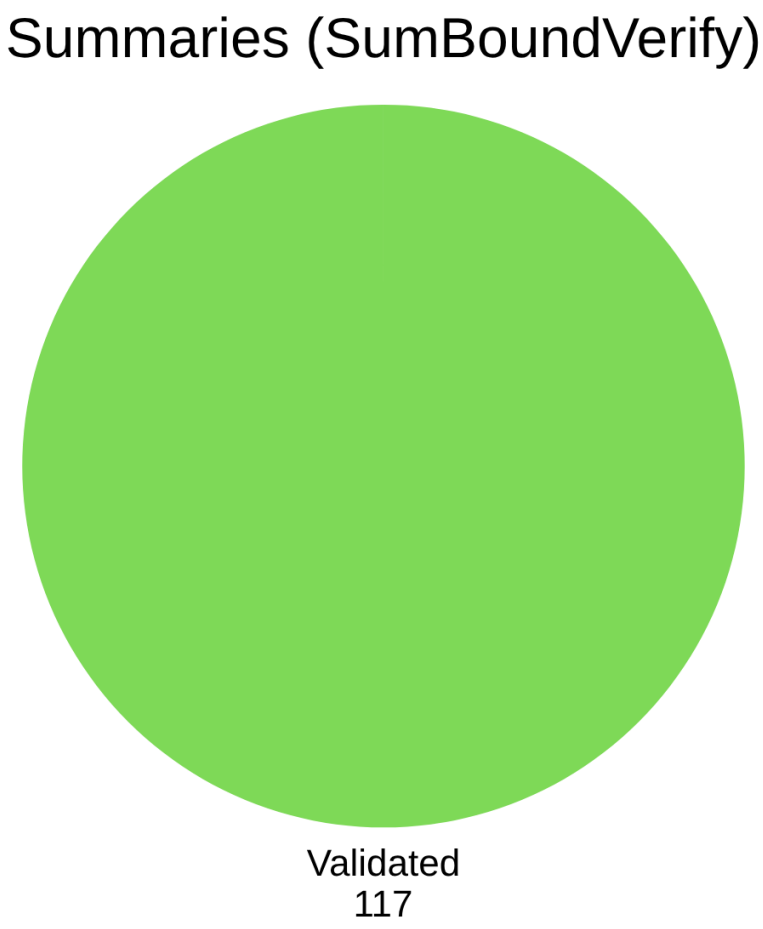
* With *SumBoundVerify*[1]
** Not suported by *SumBoundVerify*[1]

1 Ramos, F., Sabino, N., Adão, P., Naumann, D. A., & Fragoso Santos, J.: Toward Tool-Independent Summaries for Symbolic Execution. ECOOP '23 (2023)

EQ1: Summary Correctness

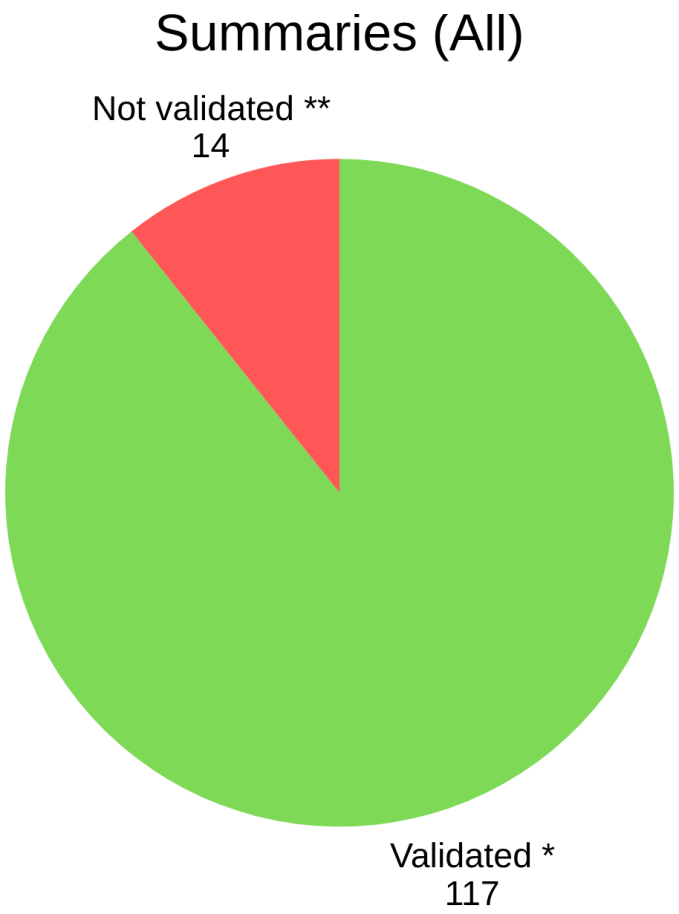


* With *SumBoundVerify*[1]
** Not suported by *SumBoundVerify*[1]

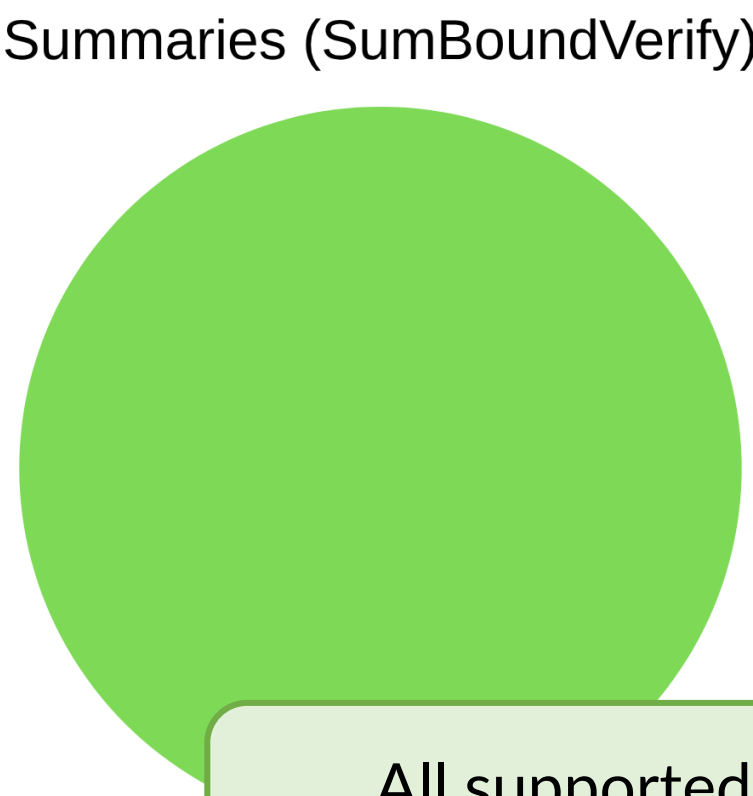


[1] Ramos, F., Sabino, N., Adão, P., Naumann, D. A., & Fragoso Santos, J.: Toward Tool-Independent Summaries for Symbolic Execution. ECOOP '23 (2023)

EQ1: Summary Correctness



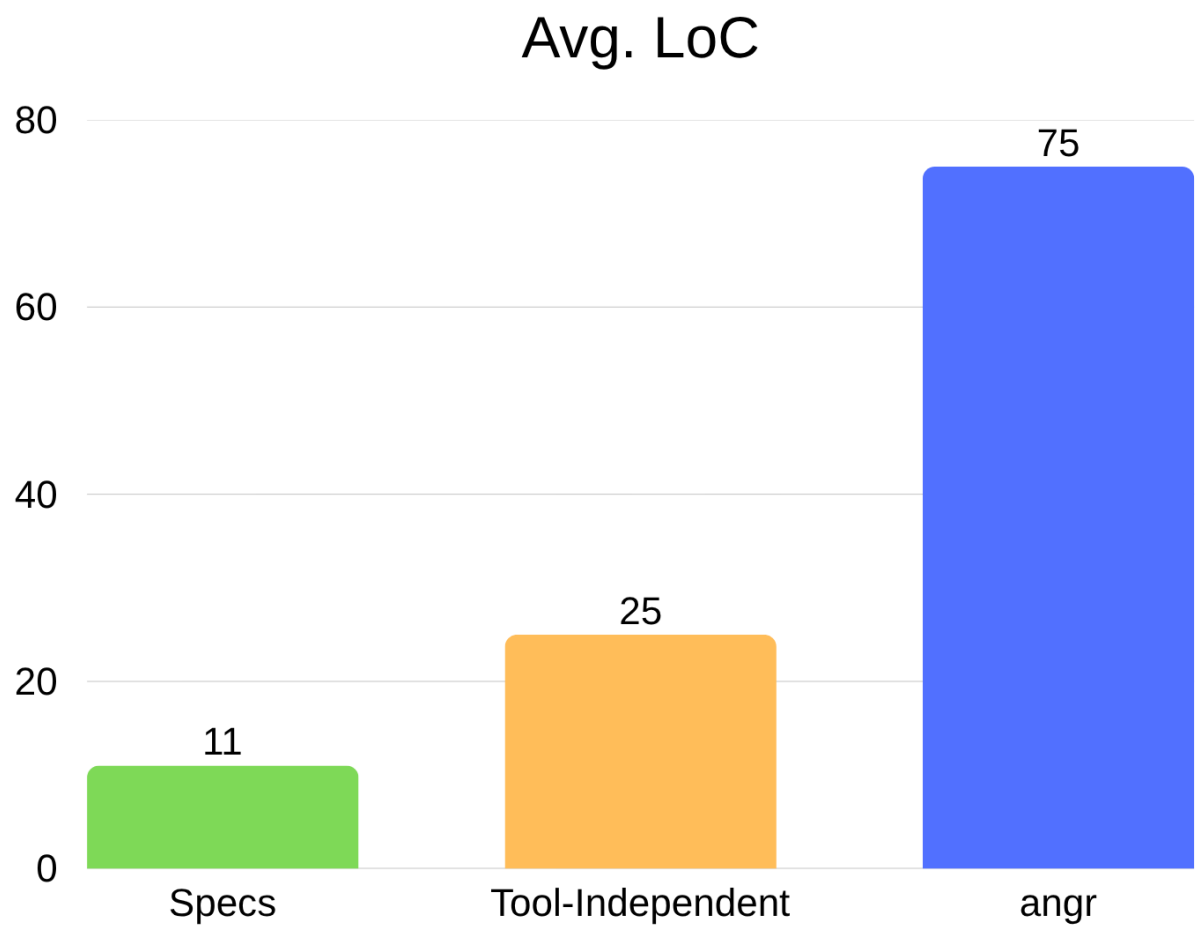
* With *SumBoundVerify*[1]
** Not suported by *SumBoundVerify*[1]



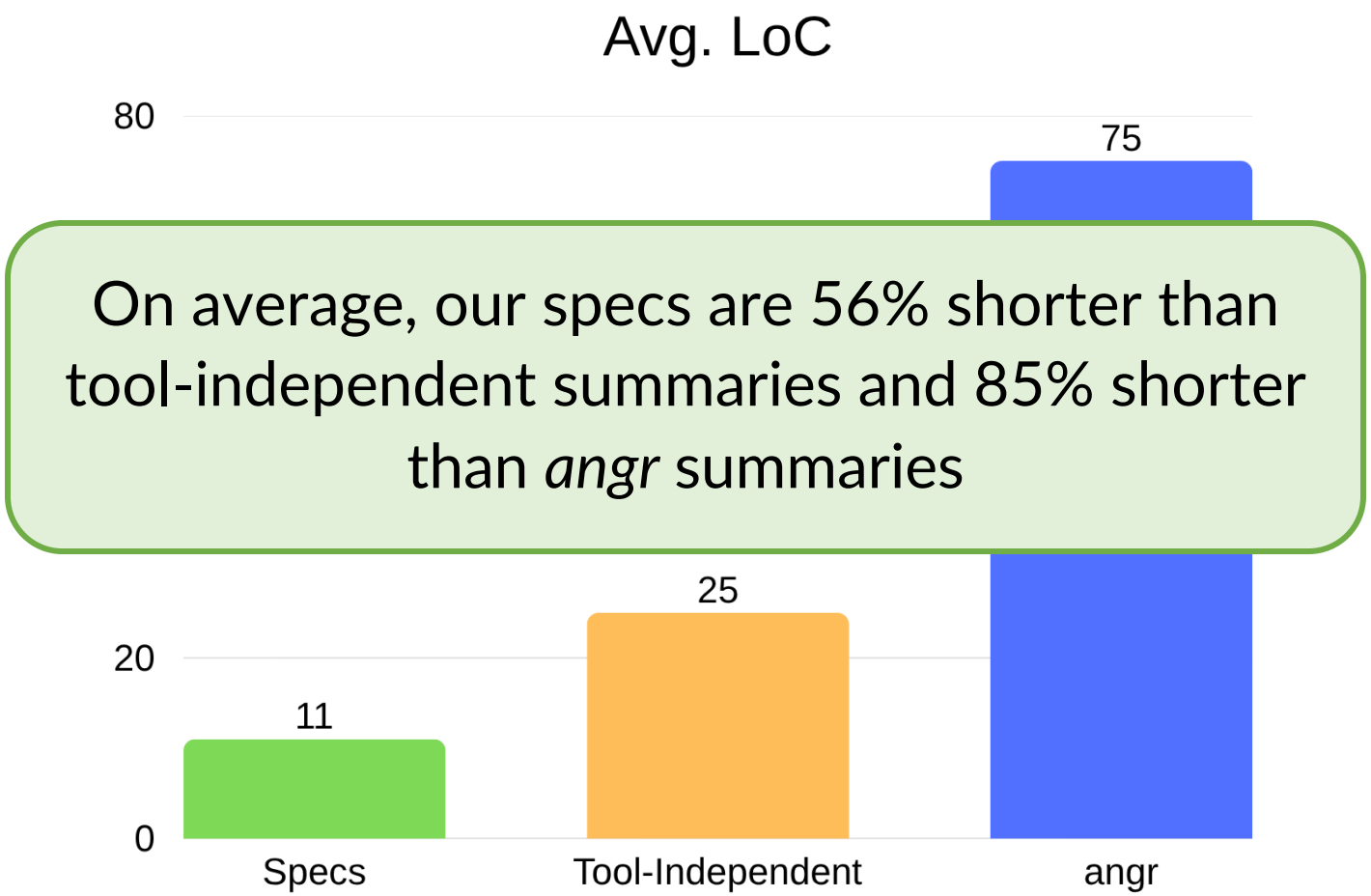
All supported summaries validated!

[1] Ramos, F., Sabino, N., Adão, P., Naumann, D. A., & Fragoso Santos, J.: Toward Tool-Independent Summaries for Symbolic Execution. ECOOP '23 (2023)

EQ2: Summary Complexity

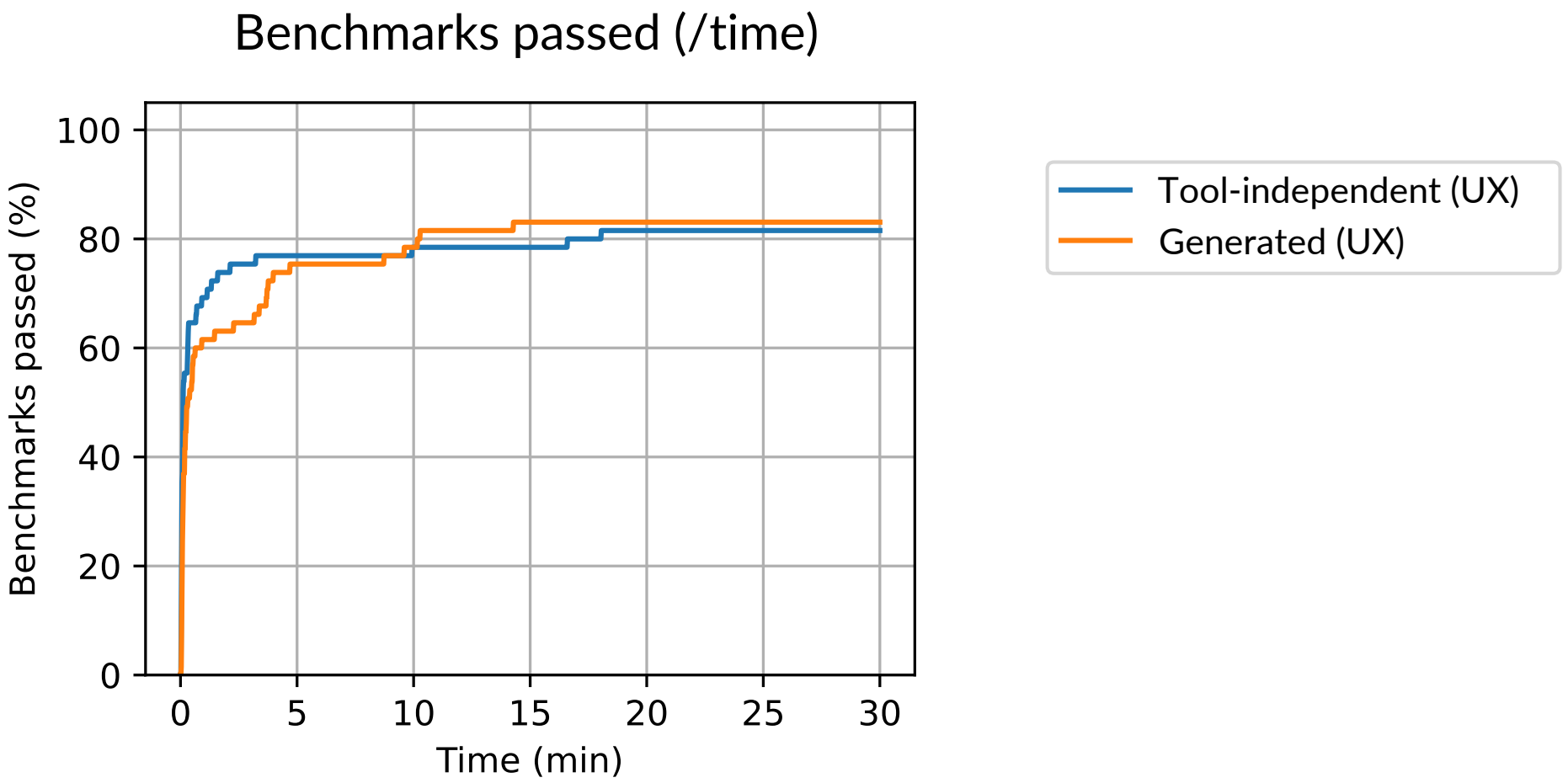


EQ2: Summary Complexity

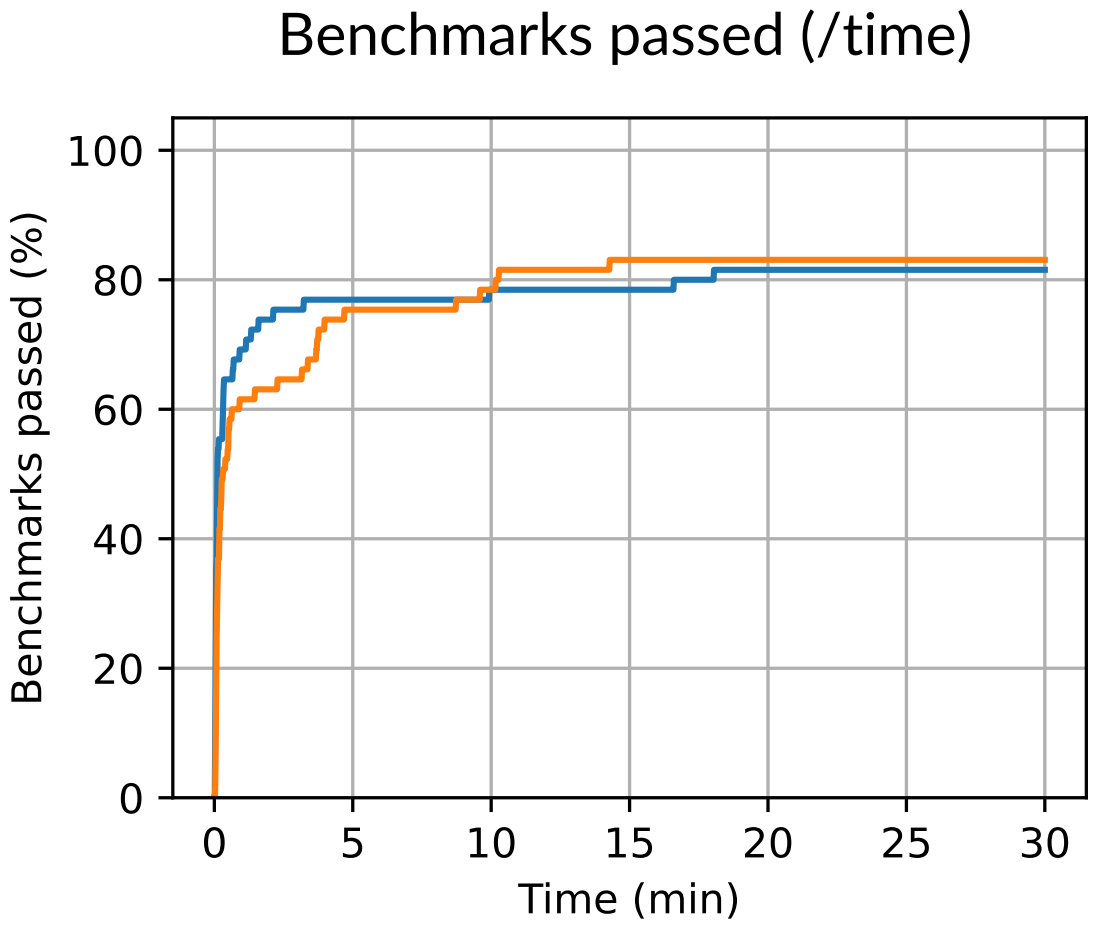


On average, our specs are 56% shorter than tool-independent summaries and 85% shorter than *angr* summaries

EQ3: Summary Performance

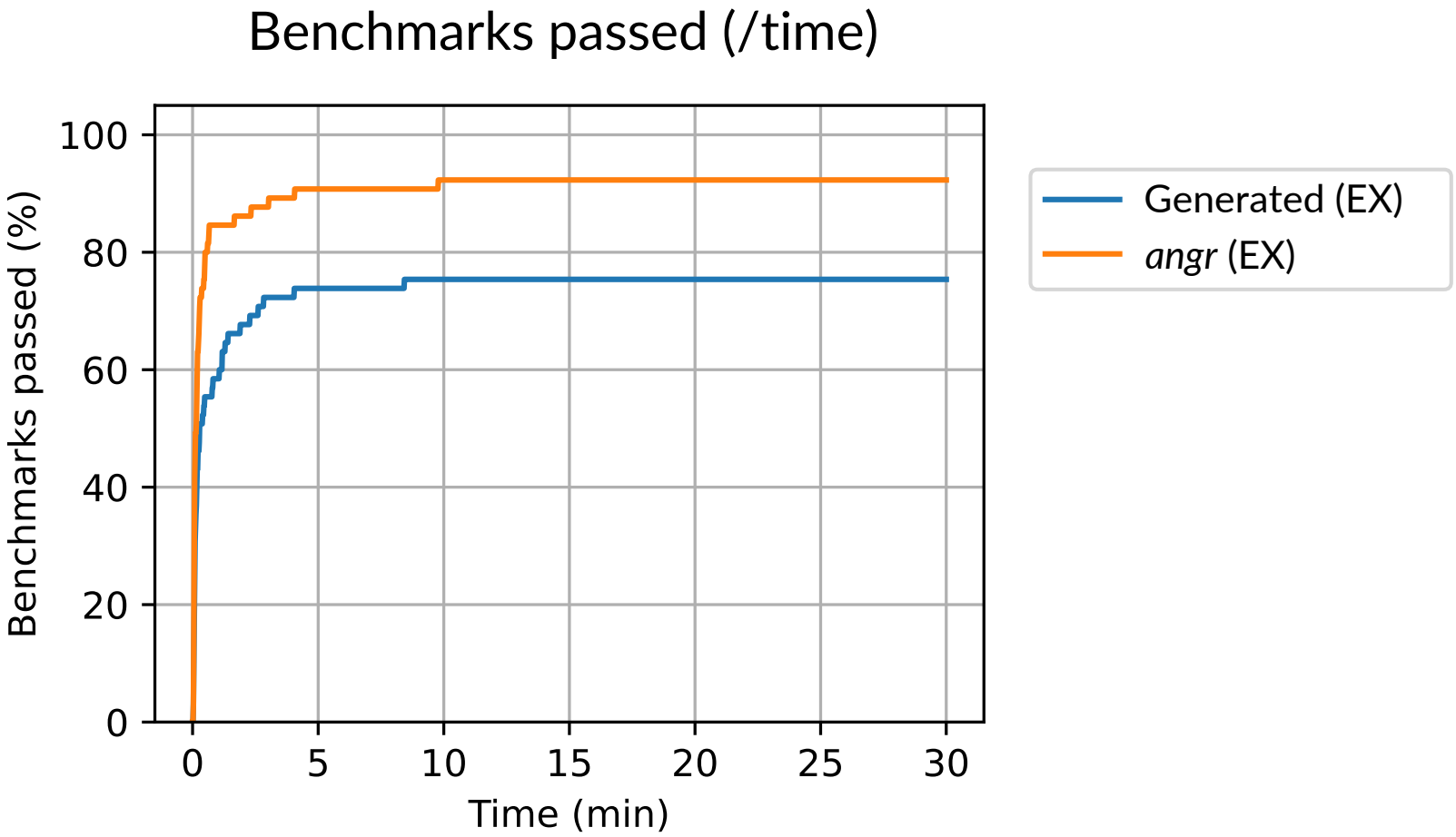


EQ3: Summary Performance

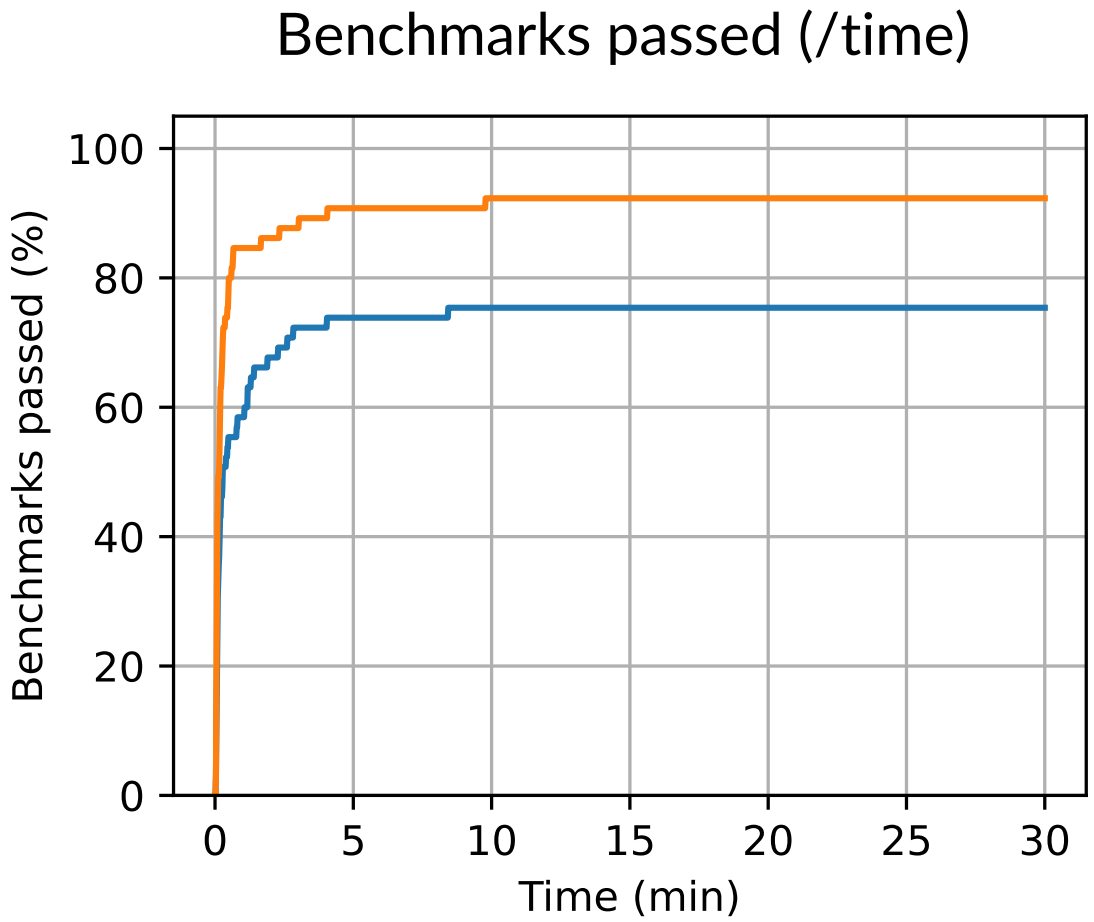


Generated summaries (slightly) **overperform** the tool-independent baseline

EQ3: Summary Performance

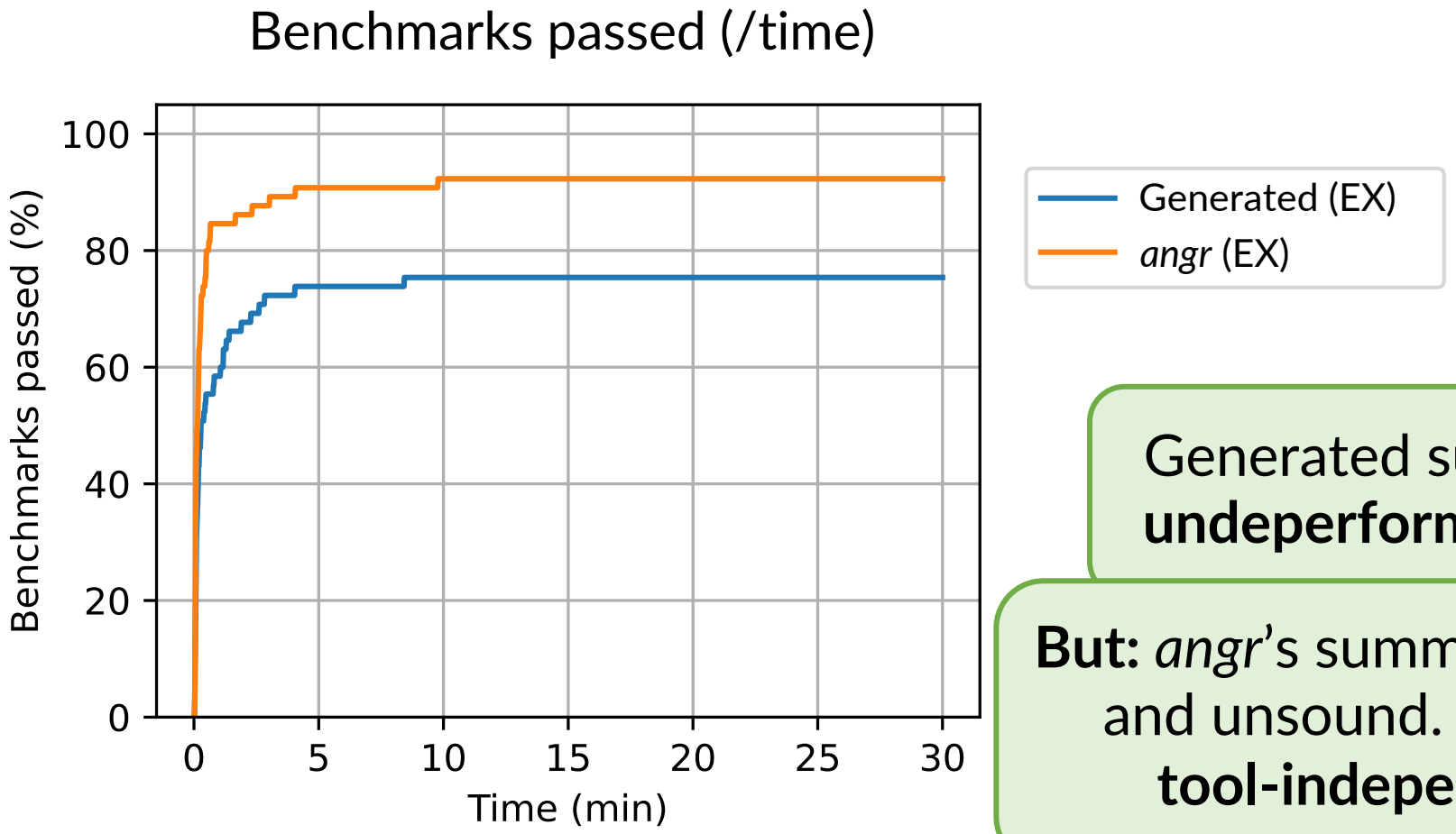


EQ3: Summary Performance



Generated summaries (slightly) **undeperform** the *angr* baseline

EQ3: Summary Performance



Generated summaries (slightly) **undeperform** the *angr* baseline

But: *angr*'s summaries are tool-specific and unsound. Our summaries are **tool-independent** and **sound**

More in the paper

- **SumGen:** our tool for summary generation
- **Under- and over-approximating** summary generation procedures
- Handling **heap mutation**
- **Soundness** guarantees
- More…

Specification-Driven Generation of Summaries for Symbolic Execution

Rafael Gonçalves[1,2,3] (✉), Frederico Ramos[2,3], Pedro Adão[2,4], and José Fragoso Santos[2,3]

[1]Carnegie Mellon University, Pittsburgh, PA, USA
rgoncalv@andrew.cmu.edu
[2]Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
{frederico.ramos,pedro.adao,jose.fragoso}@tecnico.ulisboa.pt
[3]INESC-ID, Lisbon, Portugal
[4]Instituto de Telecomunicações, Aveiro, Portugal

Abstract. Symbolic execution is a popular program analysis technique that has been successfully used for bug-finding and bounded verification in various modern programming languages. Despite its popularity, however, symbolic execution suffers from two main limitations when applied to real-world code: interactions with the runtime environment and path explosion. Symbolic summaries are the standard solution to tackle these challenges. Yet, the development of summaries remains to this day a manual task that is known to be highly error-prone. To address this, we propose SumGen, a new tool for automatically generating correct-by-construction summaries from function specifications. With SumGen, we were able to generate a total of 131 summaries for 47 libc functions, demonstrating the effectiveness of our methodology in producing correct summaries for real-world, highly complex code.

Keywords: Symbolic Execution, Symbolic Summaries, Summary Generation, Separation Logic.

Specification-Driven Generation of Summaries for Symbolic Execution

- **Summaries** mitigate the impact of **interactions with the runtime environment** and **path explosion**
  - But are difficult to (manually) produce and **prone to bugs**
- Proposed **SumGen**, a new methodology/tool for automatically generating correct-by-construction summaries from function specs
- **SumGen** is open-source and available online

Link to paper

Link to SumGen