ISP RAS

# Strong Optimistic Solving for Dynamic Symbolic Execution

Darya Parygina, Alexey Vishnyakov, Andrey Fedotov

September 23, 2022

ISP RAS

arxiv.org/abs/2209.03710

# Work Context

- The security development lifecycle (SDL) is widely used for improving software quality
- Hybrid fuzzing is the state-of-the-art solution for finding bugs and discovering new coverage
- Dynamic symbolic execution (DSE) as a hybrid fuzzer component helps reaching difficult program parts by inverting complicated branches along some execution path
- Our team develops DSE tool **Sydr** and dynamic analysis tool **sydr-fuzz** for supporting SDL
- **OSS-Sydr-Fuzz** contains open source software targets for **sydr-fuzz** that combines fuzzing (libFuzzer, AFL++) with the power of dynamic symbolic execution (Sydr)
- We found and reported **80+** new different errors in 18 projects

# Motivation

- Path predicate contains taken branch constraints such that:
  - it represents the explored path
  - solution to conjunction of these constraints is an input data to reproduce the same execution path
- **Overconstraint:** there are many redundant constraints in the path predicate that may cause its complication or even unsatisfiability
- **Underconstraint:** some variable is not treated as symbolic, though it should be
- Over- and underconstraint prevent new program paths exploration

# Overconstraint Example

Example of the program with overconstraint:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");       //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

Target constraint:

$(\texttt{buf}[3] = \text{'6'} \wedge \texttt{buf}[0] = \text{'5'})$

# Overconstraint Example

Example of the program with overconstraint:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");       //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

**Target constraint:**

$(\text{buf}[3] = \text{'6'} \wedge \text{buf}[0] = \text{'5'})$

**Path predicate:**

$\Pi = (\text{buf}[2] < \text{'0'})$

# Overconstraint Example

Example of the program with overconstraint:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");       //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

**Target constraint:**

$(\texttt{buf}[3] = \texttt{'6'} \wedge \texttt{buf}[0] = \texttt{'5'})$

**Path predicate:**

$\Pi = (\texttt{buf}[2] < \texttt{'0'}) \wedge (\texttt{buf}[0] = \texttt{'3'})$

# Overconstraint Example

Example of the program with overconstraint:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");       //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

**Target constraint:**

$(\text{buf}[3] = \text{'6'} \wedge \text{buf}[0] = \text{'5'})$

**Path predicate:**

$\Pi = (\text{buf}[2] < \text{'0'}) \wedge (\text{buf}[0] = \text{'3'}) \wedge (\text{buf}[1] - \text{buf}[3] = 1)$

# Overconstraint Example

Example of the program with overconstraint:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");       //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

Target constraint:

$(\text{buf}[3] = \text{'6'} \wedge \text{buf}[0] = \text{'5'})$

Path predicate:

$\Pi = (\text{buf}[2] < \text{'0'}) \wedge (\text{buf}[0] = \text{'3'}) \wedge (\text{buf}[1] - \text{buf}[3] = 1) \wedge (\text{buf}[3] = \text{'6'} \wedge \text{buf}[0] = \text{'5'})$

# Overconstraint Example

Example of the program with overconstraint:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");    //target point
4     else
5       printf("Fail\n");        //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

Target constraint:

$(\text{buf}[3] = \text{'6'} \wedge \text{buf}[0] = \text{'5'})$

Path predicate:

$\Pi = (\text{buf}[2] < \text{'0'}) \wedge (\text{buf}[0] = \text{'3'}) \wedge (\text{buf}[1] - \text{buf}[3] = 1) \wedge (\text{buf}[3] = \text{'6'} \wedge (\text{buf}[0] = \text{'5'})$

— UNSAT

# Related Work

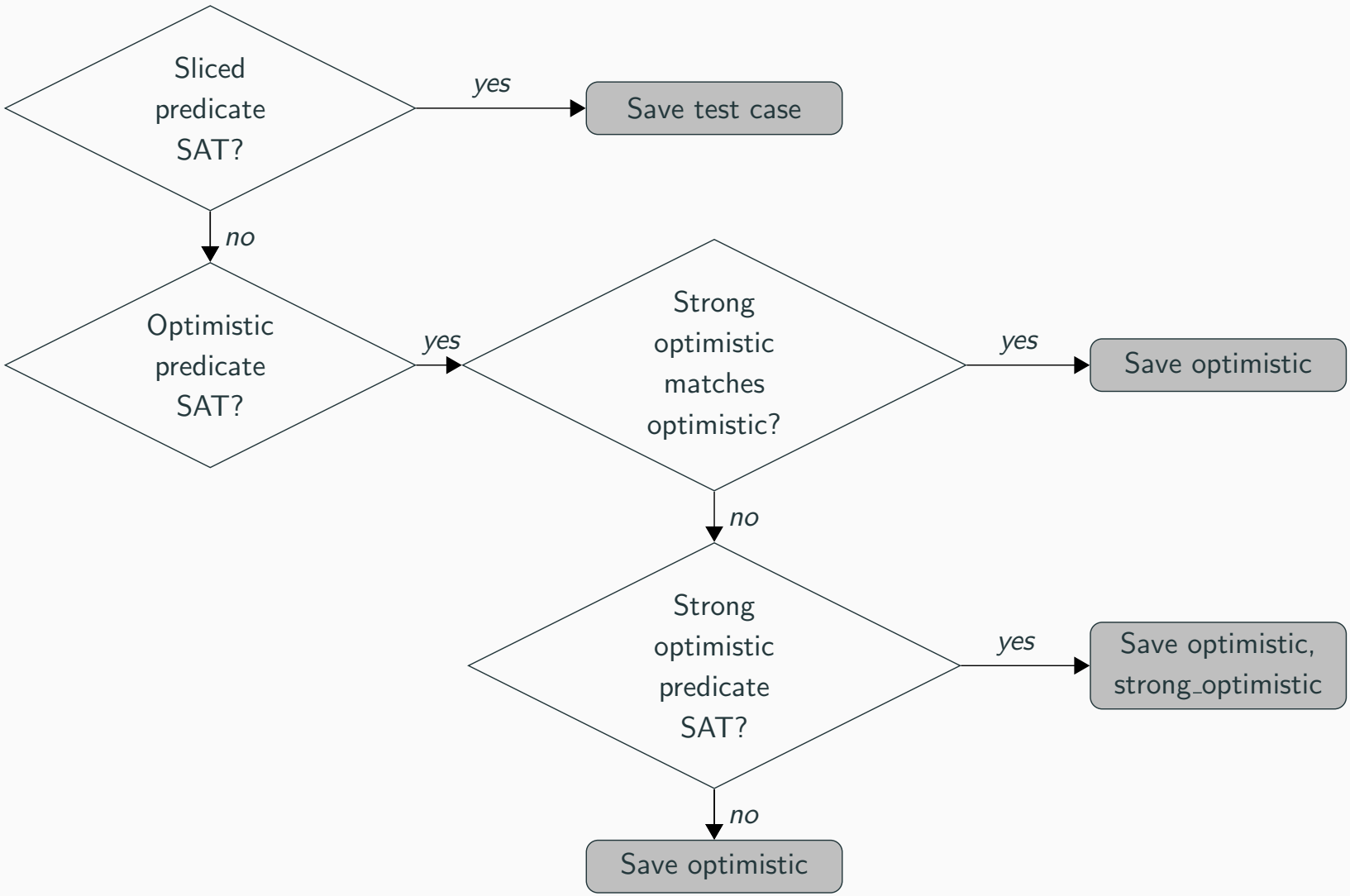
- **Irrelevant constraint elimination:**
  - KLEE: constraint independence optimization
  - Sydr: path predicate slicing
- **Optimistic solving:** solving the last (target branch) constraint (QSYM)
- **Function semantics modeling:**
  - KLEE: replacing libc functions with precompiled LLVM IR versions
  - SymCC: additional constraints modeling the concrete execution trace
  - FUZZOLIC: a single branch representing different function results
  - $S^2E$, Sydr, angr: symbolically computed expression for return value
  - Skipping redundant functions (malloc, free, printf, etc.)
- **Index-based memory model:** handling symbolic addresses according to the memory index value (Mayhem, Sydr)
- **Basic block pruning:** detecting the repetitive basic blocks and pruning them from symbolic execution (QSYM)
- **Taint-assisted partial symbolization:** partial input symbolization separately for each branch (LeanSym)

# Strong Optimistic Solving

- **Path predicate slicing idea:** eliminate path predicate constraints whose symbolic variables do not directly or transitively depend on symbolic variables from the target branch constraint
- **Optimistic solving idea:** eliminate all symbolic constraints except the last one
- **Strong optimistic solving idea:** eliminate symbolic branches, which the target branch is not control and data dependent on

Strong optimistic predicate is **more likely to be SAT** unlike sliced predicate and **more likely to be correct** unlike optimistic predicate

# Strong Optimistic Solving Flowchart



# Strong Optimistic Solving Algorithm

**Idea:** eliminate symbolic branches, which the target branch is not control and data dependent on

1. Choose the target branch, apply path predicate slicing
2. Initialize resulting predicate with the target constraint
3. Iterate over sliced predicate constraints in reversed order:
    - check if the the target branch is control dependent on the analyzed branch
    - if true: add constraint to the predicate

# Strong Optimistic Solving Algorithm

**Input:** *call_stack* – program call stack, *point* – target branch address, $c_{br}$ – target branch constraint, $\Pi$ – sliced path predicate for inverting target branch.

**Output:** $\Pi_{sopt}$ – strong optimistic predicate for inverting target branch.

---

$\Pi_{sopt} \leftarrow c_{br}$ ▷ $\Pi_{sopt}$ – strong optimistic predicate

$cs \leftarrow call\_stack(point)$ ▷ save call stack at the target branch

**for all** $c \in reversed(\Pi)$ **do** ▷ iterate over reversed sliced predicate

  **if** $call\_stack(c)$ is not a prefix of $cs$ **then**

    **continue** ▷ $c \in$ function returned before *point*

  **if** $call\_stack(c)$ is prefix of $cs$ **and** $call\_stack(c) \neq cs$ **then** ▷ $c$ and *point* in different function frames

    $point \leftarrow cs[size(call\_stack(c))]$ ▷ save call site of first distinguishing function

    $cs \leftarrow call\_stack(c)$ ▷ save call stack at new *point*

  **if** $src(c) < point < dst(c)$ **or** `ret` / `far jmp` / `far jcc` inside $(src(c),\ dst(c))$ **then** ▷ *point* is control dependent on $c$

    $\Pi_{sopt} \leftarrow \Pi_{sopt} \wedge c$ ▷ add $c$ to strong optimistic predicate

**return** $\Pi_{sopt}$

# Strong Optimistic Solving Example

Symbolic constraints selected by slicing:

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");    //target point
4     else
5       printf("Fail\n");        //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

# Strong Optimistic Solving Example

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2, point
3       printf("Success!\n");    //target point
4     else
5       printf("Fail\n");        //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

| Iter | point | cs | | c | $\Pi_{sopt}$ |
|---|---|---|---|---|---|
| 0 | line 2 | 1<br>2 | main()<br>func() | — | $c_2$ |

# Strong Optimistic Solving Example

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");      //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);                         //point
17  }
```

| Iter | point | cs | | c | $\Pi_{sopt}$ |
|---|---|---|---|---|---|
| 0 | line 2 | 1<br>2 | main()<br>func() | — | $c_2$ |
| 1 | line 16 | 1 | main() | buf[1] - buf[3] == 1 | $c_2 \wedge c_{15}$ |

# Strong Optimistic Solving Example

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");   //target point
4     else
5       printf("Fail\n");       //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);                         //point
17  }
```

| Iter | point | cs | | c | $\Pi_{sopt}$ |
|---|---|---|---|---|---|
| 0 | line 2 | 1<br>2 | main()<br>func() | — | $c_2$ |
| 1 | line 16 | 1 | main() | buf[1] - buf[3] == 1 | $c_2 \wedge c_{15}$ |
| 2 | line 16 | 1 | main() | buf[0] == '3' | $c_2 \wedge c_{15}$ |

# Strong Optimistic Solving Example

```
1   void func(const char* buf) {
2     if (buf[3] == '6' & buf[0] == '5') //c_2
3       printf("Success!\n");    //target point
4     else
5       printf("Fail\n");        //reached point
6   }
7
8   int main() {
9     char buf[4];
10    read(0, buf, 4);
11    if (buf[2] < '0')                    //c_11
12      printf("Eliminated by slicing\n");
13    if (buf[0] == '3')                   //c_13
14      printf("Independent branch\n");
15    if (buf[1] - buf[3] == 1)            //c_15
16      func(buf);
17  }
```

Strong optimistic predicate:

$$\Pi_{sopt} = (\mathrm{buf}[3] = \text{'6'} \wedge \mathrm{buf}[0] = \text{'5'}) \wedge (\mathrm{buf}[1] - \mathrm{buf}[3] = 1)$$

# The Number of Discovered Correct Branches in One Hour

| Application | Base | StrOpt | Opt | Opt + StrOpt |
|---|---|---|---|---|
| bzip2recover | 2101 | 2148 | **2264** | **2264** |
| decompress | 992 | 1550 | **4063** | 3091 |
| faad | 372 | 384 | **409** | 407 |
| foo2lava | 363 | 770 | **779** | 706 |
| hdp | 4172 | 7435 | 8091 | **8379** |
| jasper | **19558** | 16921 | 17961 | 16352 |
| libxml2 | 1035 | 5708 | 7896 | **7936** |
| minigzip | 3928 | 4110 | **5105** | **5105** |
| muraster | 2203 | 2282 | **2465** | 2461 |
| pk2bm | 190 | 194 | 273 | **275** |
| pnmhistmap (ppm) | 113 | 131 | 558 | **571** |
| re2 | 471 | 745 | 1964 | **2046** |
| readelf | 648 | 1057 | **2661** | 2645 |
| sqlite3 | 8412 | 9500 | **10432** | **10432** |
| yices-smt2 | 4407 | 5170 | 6002 | **6006** |
| yodl | 1130 | 2316 | 3078 | **3257** |

Branch is correct if at least one of its inverting predicates has correct solution (solution that actually inverts the branch)

# Explored Program Coverage

| Application | Base | Opt | Opt / Base | Sopt | Sopt / Opt |
|---|---|---|---|---|---|
| bzip2recover | 281 | 282 | +0.36% | 282 | 0% |
| cjpeg | 2412 | 2427 | +0.62% | 2428 | +0.04% |
| decompress | 4523 | 4535 | +0.27% | 4679 | +3.18% |
| faad | 3772 | 4657 | +23.46% | 4689 | +0.69% |
| foo2lava | 2396 | 2417 | +0.88% | 2418 | +0.04% |
| hdp | 6979 | 7431 | +6.48% | 7472 | +0.55% |
| jasper | 1556 | 1571 | +0.96% | 1574 | +0.19% |
| libxml2 | 8763 | 8914 | +1.71% | 8928 | +0.16% |
| minigzip | 1675 | 1675 | 0% | 1696 | +1.25% |
| muraster | 3368 | 3394 | +0.77% | 3394 | 0% |
| pk2bm | 1572 | 1575 | +0.19% | 1575 | 0% |
| pnmhistmap (ppm) | 1561 | 1705 | +9.22% | 1705 | 0% |
| re2 | 4101 | 4547 | +10.88% | 4878 | +7.28% |
| readelf | 4086 | 4971 | +21.66% | 5076 | +2.11% |
| sqlite3 | 4674 | 4702 | +0.6% | 4702 | 0% |
| yices-smt2 | 6091 | 6145 | +0.89% | 6145 | 0% |
| yodl | 3156 | 3174 | +0.57% | 3174 | 0% |

# Symbolic Execution Accuracy Evaluation

| Application | Base | StrOpt | Opt | Opt + StrOpt |
|---|---|---|---|---|
| bzip2recover | 100.0% | 98.53% | 100.0% | 100.0% |
| decompress | 83.36% | 65.73% | 64.38% | 60.79% |
| faad | 64.03% | 52.6% | 38.66% | 38.61% |
| foo2lava | 99.45% | 77.62% | 54.86% | 54.48% |
| hdp | 68.38% | 65.65% | 66.83% | 70.35% |
| jasper | 98.33% | 74.99% | 61.59% | 61.44% |
| libxml2 | 83.0% | 85.91% | 89.32% | 89.77% |
| minigzip | 51.9% | 49.93% | 56.87% | 56.87% |
| muraster | 99.91% | 92.65% | 75.24% | 74.73% |
| pk2bm | 99.48% | 73.76% | 53.42% | 53.82% |
| pnmhistmap (ppm) | 91.13% | 8.72% | 8.66% | 8.87% |
| re2 | 100.0% | 23.68% | 48.82% | 52.29% |
| readelf | 46.99% | 45.62% | 59.08% | 61.4% |
| sqlite3 | 99.98% | 99.62% | 35.31% | 35.31% |
| yices-smt2 | 79.55% | 76.77% | 74.1% | 74.15% |
| yodl | 98.26% | 73.01% | 70.23% | 71.04% |

# Symbolic Execution Speed (Correct/min) Evaluation

| Application | Base | StrOpt | Opt | Opt + StrOpt |
|---|---|---|---|---|
| bzip2recover | 135.55 | 93.73 | 141.8 | 131.76 |
| decompress | 16.53 | 25.83 | 67.72 | 51.52 |
| faad | 6.2 | 6.4 | 6.82 | 6.78 |
| foo2lava | 6.05 | 12.83 | 12.98 | 11.77 |
| hdp | 69.53 | 123.92 | 134.85 | 139.65 |
| jasper | 325.97 | 282.02 | 299.35 | 272.53 |
| libxml2 | 509.02 | 2329.8 | 3616.49 | 3283.86 |
| minigzip | 434.83 | 441.94 | 577.92 | 539.26 |
| muraster | 36.72 | 38.03 | 41.08 | 41.02 |
| pk2bm | 670.59 | 554.29 | 963.53 | 750.0 |
| pnmhistmap (ppm) | 10.51 | 8.55 | 44.58 | 31.58 |
| re2 | 7.85 | 12.42 | 32.73 | 34.1 |
| readelf | 10.8 | 17.62 | 44.35 | 44.08 |
| sqlite3 | 3554.37 | 2740.38 | 3243.11 | 2952.45 |
| yices-smt2 | 798.85 | 937.16 | 1087.98 | 1085.42 |
| yodl | 4520.0 | 7720.0 | 10260.0 | 9771.0 |

# Questions?

# Handling Call Stack and CTI Example

```
1  // cJSON_ParseWithLengthOpts() - f_1
2  {   ...
3      item = cJSON_New_Item(&global_hooks);
4      if (item == NULL) /* memory fail */
5          goto fail;
6      ...
7      if (!parse_value(item,
   ↪   buffer_skip_whitespace(skip_utf8_bom(&buffer)))) {
8          ...
9      }
10     ...
11 fail: ...
12 }
13 // parse_value() - f_2
14 {   ...
15     if (can_read(input_buffer, 4) &&
16     (strncmp((constchar*)buffer_at_offset(input_buffer),
   ↪   "null", 4) == 0)) {                          //point
17         item->type = cJSON_NULL;
18         input_buffer->offset += 4;
19         return true;
20     }
21      /* false */
22      ...
23  }
```

# Handling Call Stack and CTI Example

```
1   // cJSON_ParseWithLengthOpts() - f_1
2   {   ...
3       item = cJSON_New_Item(&global_hooks);
4       if (item == NULL) /* memory fail */
5           goto fail;
6       ...
7       if (!parse_value(item,
    ↪   buffer_skip_whitespace(skip_utf8_bom(&buffer)))) {
8           ...
9       }
10      ...
11  fail: ...
12  }
13  // parse_value() - f_2
14  {   ...
15      if (can_read(input_buffer, 4) &&
16      (strncmp((constchar*)buffer_at_offset(input_buffer),
    ↪   "null", 4) == 0)) {                          //point
17          item->type = cJSON_NULL;
18          input_buffer->offset += 4;
19          return true;
20      }
21       /* false */
22       ...
23   }
```

| Iter | point | cs | | c | $\Pi_{sopt}$ |
|---|---|---|---|---|---|
| 0 | line 16 | 1<br>2 | $f_1$<br>$f_2$ | — | $c_{16}$ |

# Handling Call Stack and CTI Example

```
1   // cJSON_ParseWithLengthOpts() - f_1
2   {   ...
3       item = cJSON_New_Item(&global_hooks);
4       if (item == NULL) /* memory fail */
5           goto fail;
6       ...
7       if (!parse_value(item,
    ↪   buffer_skip_whitespace(skip_utf8_bom(&buffer)))) {
    ↪   //point
8           ...
9       }
10      ...
11  fail: ...
12  }
13  // parse_value() - f_2
14  {   ...
15      if (can_read(input_buffer, 4) &&
16      (strncmp((constchar*)buffer_at_offset(input_buffer),
    ↪   "null", 4) == 0)) {
17          item->type = cJSON_NULL;
18          input_buffer->offset += 4;
19          return true;
20      }
21       /* false */
22       ...
23   }
```

| Iter | point | cs | | c | $\Pi_{sopt}$ |
|---|---|---|---|---|---|
| 0 | line 16 | 1<br>2 | $f_1$<br>$f_2$ | — | $c_{16}$ |
| 1 | line 7 | 1 | $f_1$ | $c_4$ | $c_{16} \wedge c_4$ |

# Handling Call Stack and CTI Example

```
1   // cJSON_ParseWithLengthOpts() - f_1
2   {   ...
3       item = cJSON_New_Item(&global_hooks);
4       if (item == NULL) /* memory fail */
5           goto fail;
6       ...
7       if (!parse_value(item,
    ↪   buffer_skip_whitespace(skip_utf8_bom(&buffer)))) {
8           ...
9       }
10      ...
11  fail: ...
12  }
13  // parse_value() - f_2
14  {   ...
15      if (can_read(input_buffer, 4) &&
16      (strncmp((constchar*)buffer_at_offset(input_buffer),
    ↪   "null", 4) == 0)) {
17          item->type = cJSON_NULL;
18          input_buffer->offset += 4;
19          return true;
20      }
21       /* false */
22       ...
23   }
```

Strong optimistic predicate:

$\Pi_{sopt} =$
$(strncmp((constchar*)$
$buffer\_at\_offset(input\_buffer),$
$"null", 4) \neq 0) \wedge$
$(item \neq NULL)$

# The Number of Discovered Correct Branches in One Hour (symptr)

| Application | Base | StrOpt | Opt | Opt + StrOpt |
|---|---|---|---|---|
| decompress | 572 | 793 | 999 | **1034** |
| hdp | 1006 | 1311 | **1432** | 1419 |
| jasper | **37227** | 34845 | 35430 | 32972 |
| libxml2 | 1184 | 6152 | 8674 | **8727** |
| minigzip | 577 | 604 | **614** | **614** |
| re2 | 332 | 399 | **1271** | 1270 |
| readelf | 127 | 160 | 179 | **207** |
| sqlite3 | 10338 | 11502 | 12462 | **12463** |
| yices-smt2 | 2178 | 2326 | **2734** | 2561 |
| yodl | 1402 | 2625 | 3387 | **3566** |

# Symbolic Execution Accuracy Evaluation (symptr)

| Application | Base | StrOpt | Opt | Opt + StrOpt |
|---|---|---|---|---|
| decompress | 80.56% | 66.14% | 48.47% | 50.27% |
| hdp | 64.86% | 64.33% | 68.52% | 68.39% |
| jasper | 99.19% | 88.43% | 80.61% | 80.62% |
| libxml2 | 91.22% | 88.16% | 88.13% | 88.67% |
| minigzip | 59.73% | 60.04% | 60.02% | 60.2% |
| re2 | 100.0% | 19.66% | 49.73% | 49.76% |
| readelf | 91.37% | 76.19% | 25.83% | 30.17% |
| sqlite3 | 99.98% | 99.47% | 39.44% | 39.45% |
| yices-smt2 | 82.25% | 74.74% | 74.31% | 74.38% |
| yodl | 98.66% | 75.43% | 72.2% | 72.88% |

# Symbolic Execution Speed (Correct/min) Evaluation (symptr)

| Application | Base | StrOpt | Opt | Opt + StrOpt |
|---|---|---|---|---|
| decompress | 9.53 | 13.22 | 16.65 | 17.23 |
| hdp | 16.77 | 21.85 | 23.87 | 23.65 |
| jasper | 620.45 | 580.75 | 590.5 | 549.53 |
| libxml2 | 215.93 | 932.12 | 1320.91 | 1198.22 |
| minigzip | 9.62 | 10.07 | 10.23 | 10.23 |
| re2 | 5.53 | 6.65 | 21.18 | 21.17 |
| readelf | 2.12 | 2.67 | 2.98 | 3.45 |
| sqlite3 | 1797.91 | 1691.47 | 1917.23 | 1776.2 |
| yices-smt2 | 36.3 | 38.77 | 45.57 | 42.68 |
| yodl | 1682.4 | 2812.5 | 3628.93 | 3626.44 |