Pending Constraints in Symbolic Execution for Better Exploration and Seeding

Timotej Kapus    Frank Busse    Cristian Cadar
Imperial College London
Symbolic Execution

- Program analysis technique
- Active research area
- Used in industry
  - IntelliTest, SAGE
  - KLOVER
  - Angr
Why symbolic execution?

- *No false positives!*
  - Every bug found has a concrete input triggering it
- Can interact with the environment
  - I/O, unmodeled libraries
- Only relevant code executed
  - “symbolically”, the rest is fast “native” execution
Why (not) symbolic execution?

- Scalability, scalability, scalability
  - Constraint solving is hard
  - Path explosion
This talk

Introduce pending constraints which enhance the scalability of symbolic execution via aggressively tackling paths that are known to be feasible.
“known to be feasible”

Caching

- Cache assignments from previous solver queries
- Already widely adopted

Seeding

- External, usually valid concrete inputs
- Used to bootstrap symbolic execution
- From test-suites, examples, production data
Symbolic execution example: get_sign

```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments

∅
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

**Known assignments**

∅
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments

- x = -2
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments

\[ x = -2 \]
\[ x = 7 \]
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

**Known assignments**

- \(x = -2\)
- \(x = 7\)
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments

- \( x = -2 \)
- \( x = 7 \)
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments

- \(x = -2\)
- \(x = 7\)
- \(x = 0\)
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

Known assignments

- \( x = -2 \)
- \( x = 7 \)
- \( x = 0 \)
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments:

- `x = -2`
- `x = 7`
- `x = 0`
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments:
- \( x = -2 \)
- \( x = 7 \)
- \( x = 0 \)
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

**Known assignments**

- \( x = -2 \)
- \( x = 7 \)
- \( x = 0 \)
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments:

- \(x = -2\)
- \(x = 7\)
- \(x = 0\)
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

Known assignments

\[
\begin{align*}
    x &= -2 \\
    x &= 7 \\
    x &= 0
\end{align*}
\]
Symbolic execution with pending constraints

- Explore paths that are known to be feasible
- Solve constraints only when necessary to make progress

```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments

∅
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

**Known assignments**

Φ
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

get_sign(x);

Known assignments
∅
get_sign(x);

Pending constraints

\[
\begin{align*}
\text{if } (x \geq 1) & \quad r = 1; \\
\text{if } (x == 0) & \quad r = 0;
\end{align*}
\]

- Not known to be feasible
- When only pending constraints left
  - Pick one and check
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

Known assignments

x = -2

get_sign(x);

\[
\begin{align*}
\text{if } (x \geq 1) & \quad r = 1; \\
\text{if } (x = 0) & \quad r = 0; \\
\end{align*}
\]

return \( r \);

\[
\begin{align*}
x \geq 1 & \quad x \geq 1 \\
x < 1 & \quad x < 1 \\
? & \quad ?
\end{align*}
\]
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
get_sign(x);

Pending constraints: still not known if feasible

if (x >= 1) r = 1;
if (x == 0) r = 0;
return r;

Known assignments

x = -2

x = 0

x ≠ 0

x < 1

x >= 1

x ≥ 1
get_sign(x);

Pending constraints: known feasible path!

```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments:
- `x = -2`
- `x ≠ 0`

x = -2
x ≥ 1
x < 1
x == 0
x ≠ 0
x = 0

Pending constraints: known feasible path!
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

Known assignments

x = -2

return r;
get_sign(x);

Only pending constraints: check one

if (x >= 1) r = 1;
if (x == 0) r = 0;
return r;

Known assignments

x = -2

return r;
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments:
- \(x = -2\)
- \(x = 0\)
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

Known assignments

\[ x = -2 \]
\[ x = 0 \]
\[ x = 7 \]
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

Known assignments:
- $x = -2$
- $x = 0$
- $x = 7$
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}

Known assignments

x = -2
x = 0
x = 7

get_sign(x);

r = -1;

r = 1;

x <= 1

x < 1

x = 0

x ≠ 0

r = 0;

return r;

return r;

return r;
```c
int get_sign(int x) {
    int r = -1;
    if (x >= 1) r = 1;
    if (x == 0) r = 0;
    return r;
}
```

**Known assignments**

- $x = -2$
- $x = 0$
- $x = 7$
Pending constraints: why would they be useful?

- Reversing md5 hash
  - Very hard for SMT solvers
- 1471037522 = md5("ase2020")[0]
  - Use as seed

```c
char msg[8] = symbolic;
uint32_t *hash = md5(msg, 8);
assert(hash[0] == 1471037522)
```
Suppose this exploration tree for md5
Suppose this path

\[ \text{md5("ase2020")} \]
Solver queries: 0

Pending

Vanilla
Solver queries: 0

Pending

Vanilla
Solver queries: 0

Pending

Vanilla
Solver queries: 0

Pending

Vanilla
Solver queries: 1

Pending

Vanilla
Solver queries: 2

Pending

Vanilla
Solver queries: 3

Pending

Vanilla
Solver queries: 4
Solver queries: 5

Pending

Vanilla
Solver queries: 6

Pending

Vanilla
Solver queries: 7

Pending

Vanilla
Solver queries: 8
Why pending constraints?

- More efficient use of solver solutions
  - Explore more instructions per query
  - Less time solving infeasible queries
- Prefers deeper search tree exploration
- Empowering search heuristics
  - Control over constraint solving
  - ZESTI
Evaluation

- Based on an implementation in KLEE
- 8 real world applications
- Hard targets for symbolic execution
Experiment design

- 2 hour runs
- With and without seeds
- 3 search strategies: random path, DFS, depth biased
- 3 repetitions

- Case study on SQLite3 with 24 hour runs
Vanilla vs Pending without seeds (random path)
Proportion of time spent solving queries that were infeasible
SQLite3: 24 hour run without seeds (random path)
Vanilla vs Pending with seeds (random path)
SQLite3: 24 hour run with seed
ZESTI and seeding

- Extension of KLEE for augmenting test suites
- Explores paths “around” a seed
- Easy to implement with pending constraints
- Found 2 bugs in tar, dwarfdump that were fixed
Conclusion

● Pending constraints
  ○ Tackles scalability of symbolic execution by aggressively following paths that are known to be feasible
● Effective in improving coverage for 8 challenging programs
Without seeds

(a) Random Path

(b) Depth-Biased

(c) DFS
Time spent constraint solving by vanilla KLEE