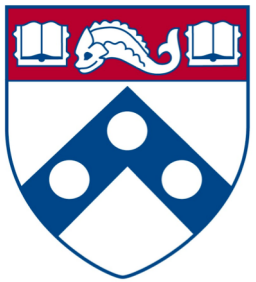
Learning Loop Invariants for Program Verification

Xujie Si*, Hanjun Dai*, Mukund Raghothaman, Mayur Naik, Le Song



University of Pennsylvania
Georgia Institute of Technology

NeurIPS 2018

Code: https://github.com/PL-ML/code2inv

* equal contribution

Program verification

- Prove whether your code is bug-free

Program verification

- Prove whether your code is bug-free

  -- Some of rules can be automated:

  sequence rule, conditional rule, ...

Program verification

- Prove whether your code is bug-free

  -- Some of rules can be automated:

  sequence rule, conditional rule, ……

  -- Except ‘while rule’

  Loop Invariant <> Halting Problem

What is loop invariant?

What is loop invariant?

Program

$$
\begin{array}{l}
x := -50; \\
\textbf{while}\ (x < 0)\ \{ \\
\quad x := x + y; \\
\quad y := y + 1\ \} \\
\textbf{assert}(y > 0)
\end{array}
$$

What is loop invariant?

Program	Loop Invariant
$x := -50;$ **while** $(x<0)$ { $\quad x := x+y;$ $\quad y := y+1$ } **assert**$(y>0)$	$(x<0 \vee y>0)$

What is loop invariant?

Program

$$
\begin{array}{l}
x := -50; \\
\textbf{while}\ (x<0)\ \{ \\
\quad x := x+y; \\
\quad y := y+1\ \} \\
\textbf{assert}(y>0)
\end{array}
$$

Loop Invariant

$$(x<0 \vee y>0)$$

Requirement:

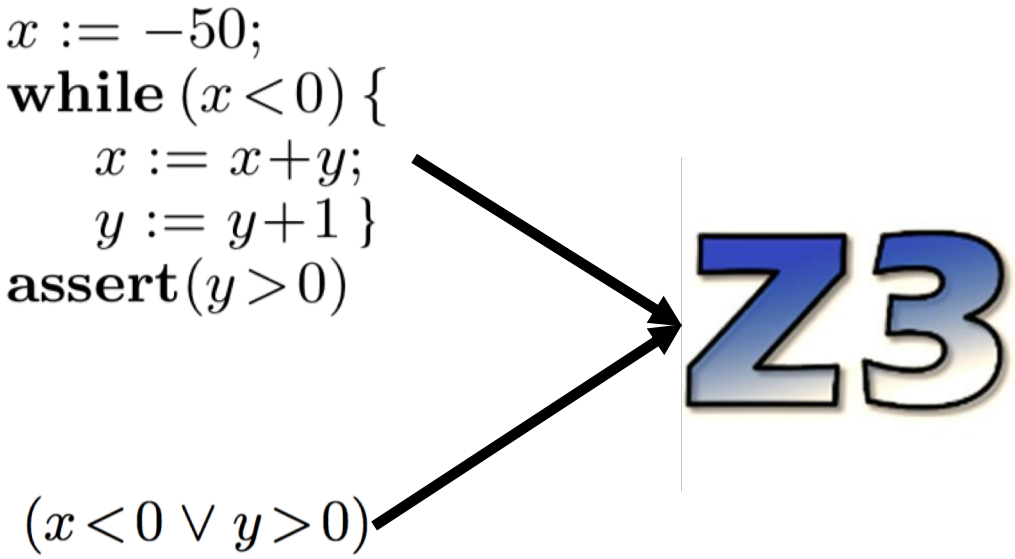
$$\forall x,y: \begin{cases} \texttt{true} \Rightarrow I[-50/x] & (pre) \\ I \wedge x<0 \Rightarrow I[(y+1)/y,(x+y)/x] & (inv) \\ I \wedge x \geq 0 \Rightarrow y>0 & (post) \end{cases}$$

Loop Invariant Checker

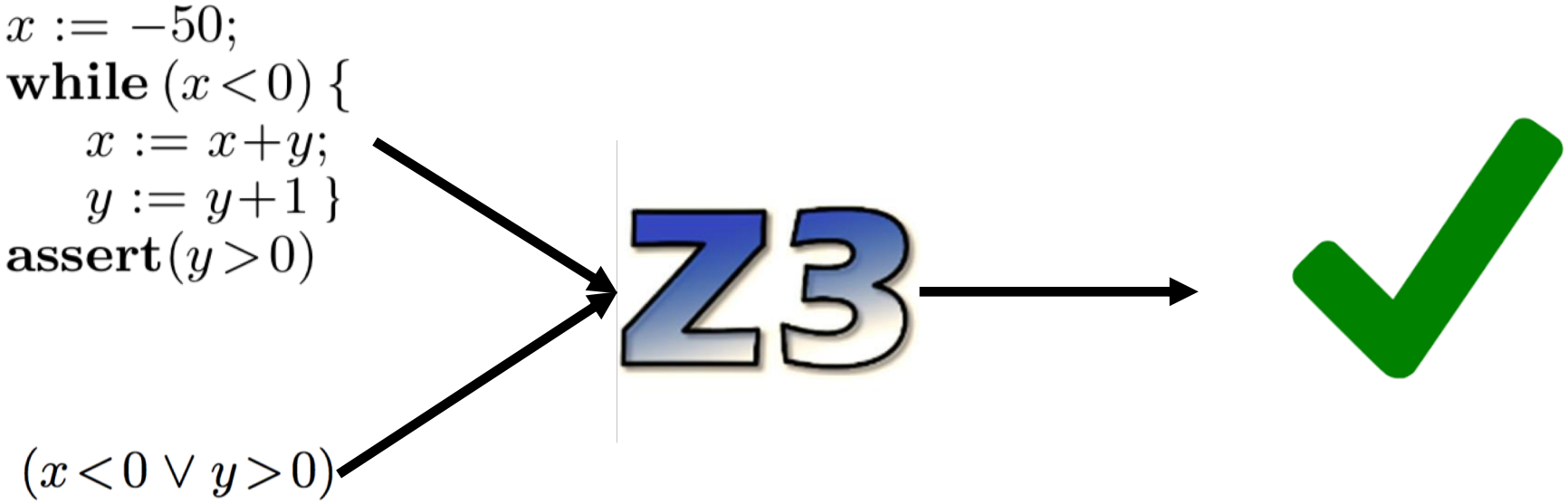
$x := -50;$
while $(x < 0)$ {
    $x := x + y;$
    $y := y + 1$ }
assert $(y > 0)$

$(x < 0 \vee y > 0)$

Loop Invariant Checker



Loop Invariant Checker



Difficulties of learning loop Invariant

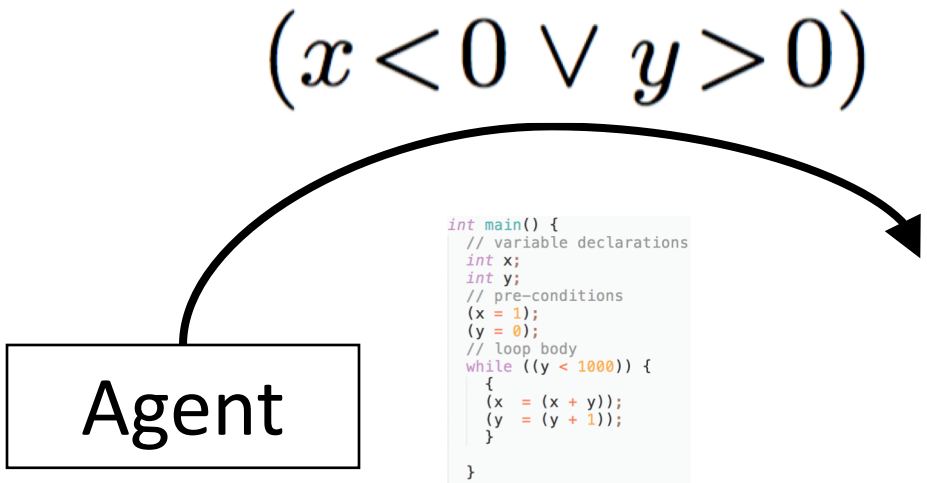
1. Highly sparse and non-smooth reward

```
int main() {
  // variable declarations
  int x;
  int y;
  // pre-conditions
  (x = 1);
  (y = 0);
  // loop body
  while ((y < 1000)) {
    {
    (x  = (x + y));
    (y  = (y + 1));
    }

  }
  // post-condition
assert( (x >= y) );
}
```

code

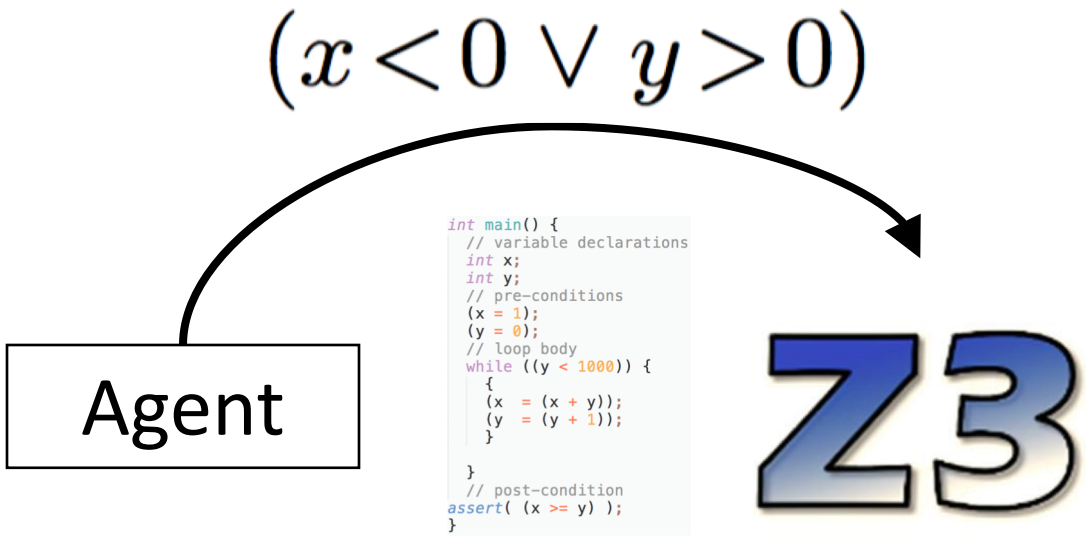
Difficulties of learning loop Invariant

1. Highly sparse and non-smooth reward



Difficulties of learning loop Invariant

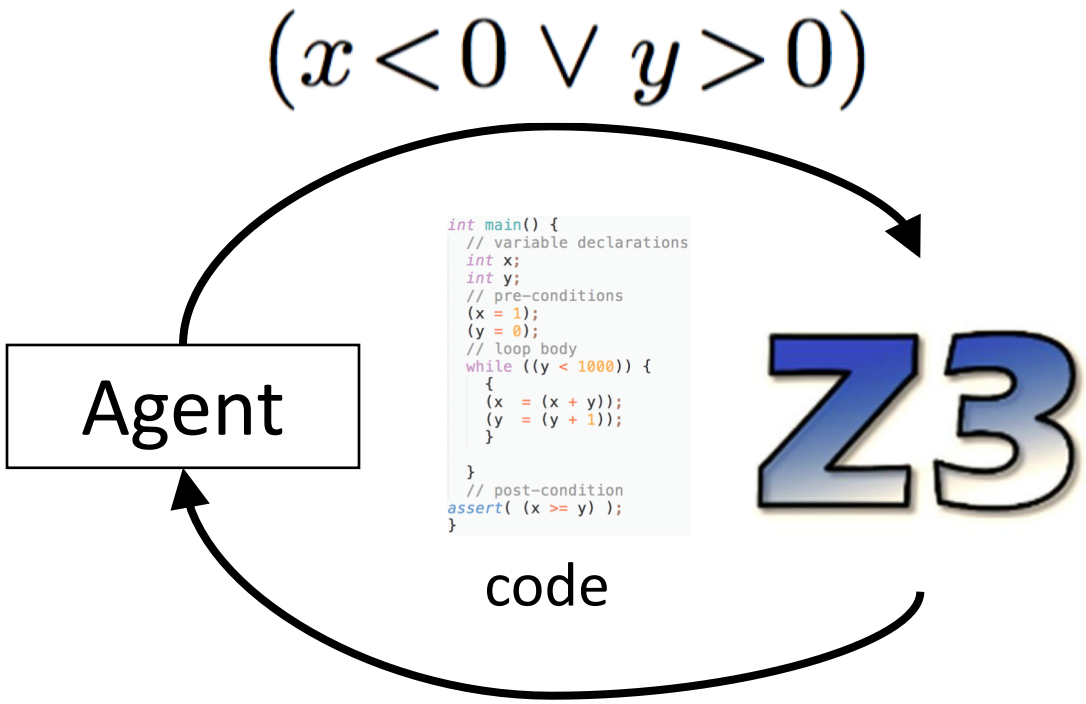
1. Highly sparse and non-smooth reward



code

Difficulties of learning loop Invariant

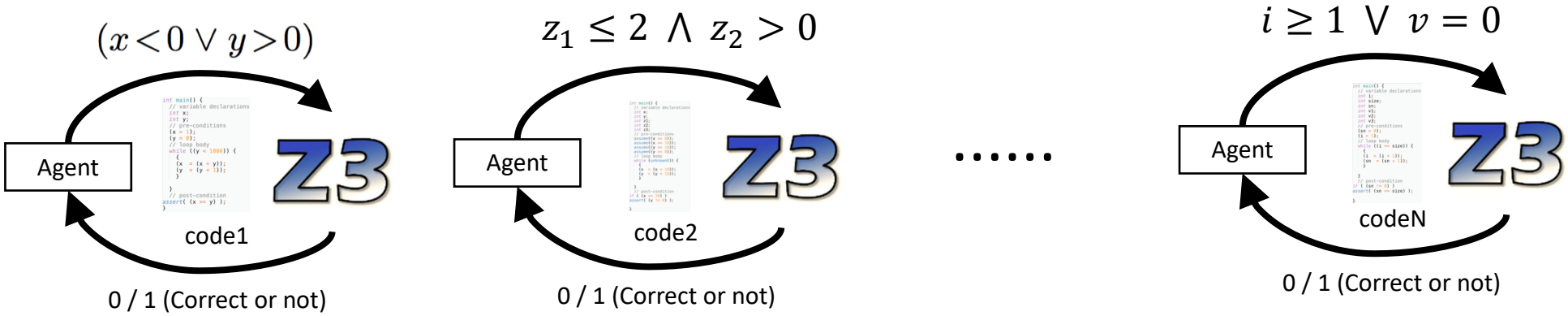
1. Highly sparse and non-smooth reward



0 / 1 (Correct or not)

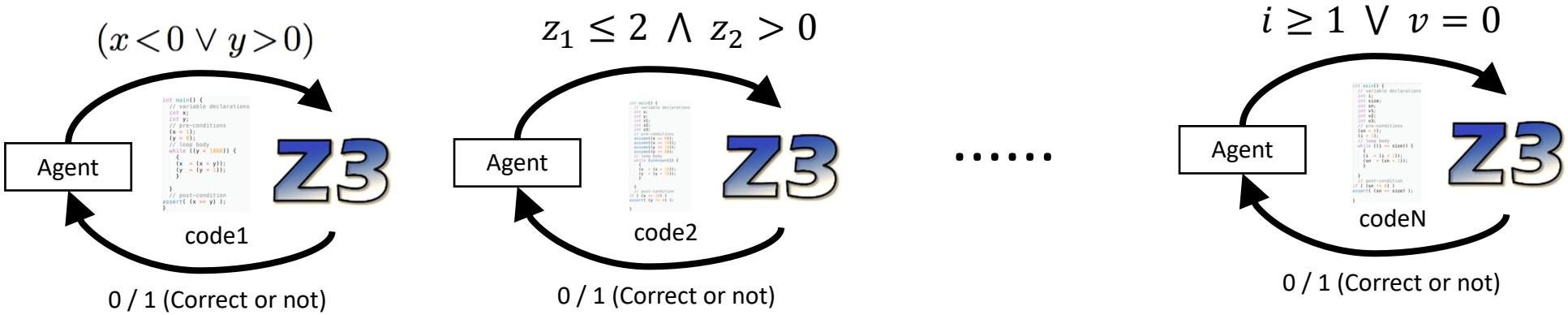
Difficulties of learning loop invariant

2. Generalization ability



Difficulties of learning loop invariant

2. Generalization ability



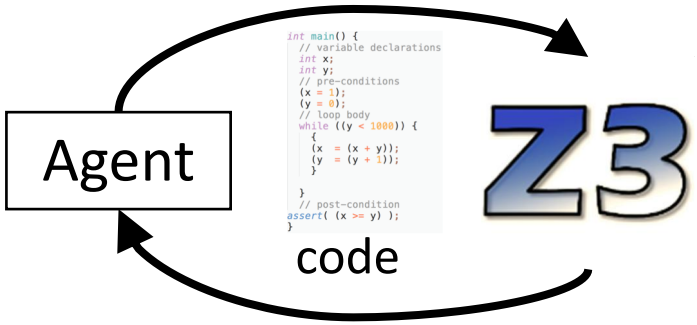
New code

```
int main() {
  // variable declarations
  int i;
  int j;
  int x;
  int y;
  // pre-conditions
  (i = x);
  (j = y);
  // loop body
  while ((x != 0)) {
    {
    (x  = (x - 1));
    (y  = (y - 1));
    }

  }
  // post-condition
if ( (i == j) )
assert( (y == 0) );

}
```

=> Agent

Solution to sparsity and non-smoothness

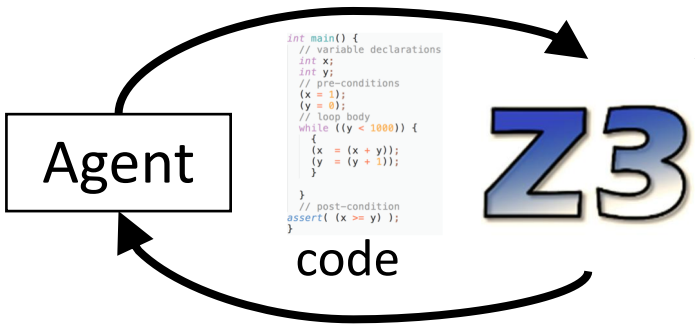
$$(x < 0 \vee y > 0)$$



0 (not correct)

Solution to sparsity and non-smoothness

$(x<0 \vee y>0)$

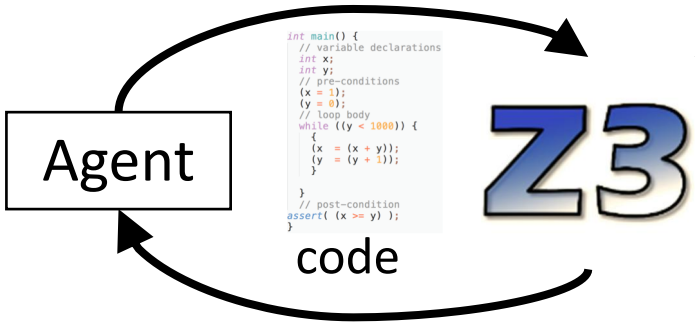


Counter-example: why am I wrong?

$$x = 1, y = -10$$

Solution to sparsity and non-smoothness

$(x<0 \vee y>0)$



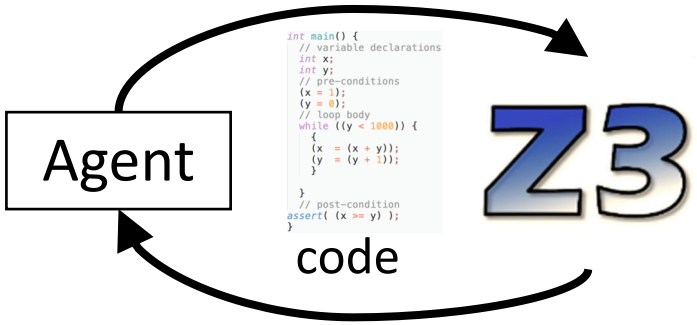
Collection of counter-examples:

Counter-example: why am I wrong?

$x = 1, y = -10$

Solution to sparsity and non-smoothness

$(x<0 \vee y>0)$



0 (not correct)

Collection of counter-examples:

Pre
$x=0, y=-2$
$x=0, y=-1$
$x=1, y=-1$

Inv
$x=3, y=-2$
$x=3, y=-1$
$x=3, y=-1$
$x=2, y=-2$
$x=2, y=-1$
$x=2, y=-1$

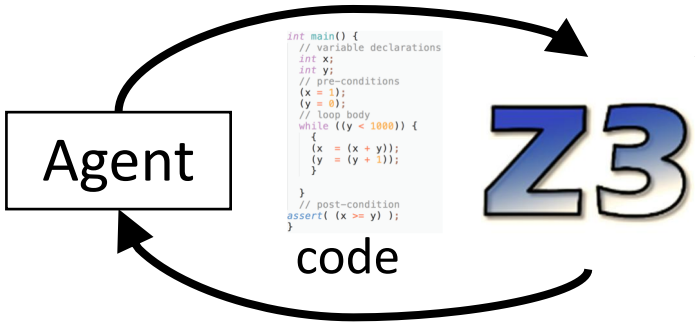
Post
$x=0, y=-4$
$x=0, y=-3$

Counter-example: why am I wrong?

$x=1, y=-10$

Solution to sparsity and non-smoothness

$(x<0 \vee y>0)$



0 (not correct)

Collection of counter-examples:

Pre
$x=0, y=-2$
$x=0, y=-1$
$x=1, y=-1$

Inv
$x=3, y=-2$
$x=3, y=-1$
$x=3, y=-1$
$x=2, y=-2$
$x=2, y=-1$
$x=2, y=-1$

Post
$x=0, y=-4$
$x=0, y=-3$

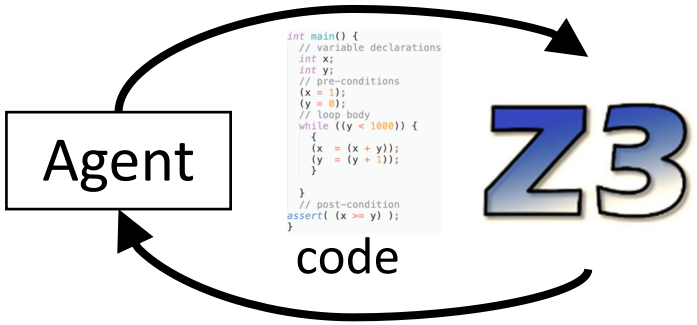
Counter-example: why am I wrong?

$x=1, y=-10$

- Smoothed reward

Solution to sparsity and non-smoothness

$(x<0 \vee y>0)$



Collection of counter-examples:

Pre
$x=0, y=-2$
$x=0, y=-1$
$x=1, y=-1$

Inv
$x=3, y=-2$
$x=3, y=-1$
$x=3, y=-1$
$x=2, y=-2$
$x=2, y=-1$
$x=2, y=-1$

Post
$x=0, y=-4$
$x=0, y=-3$

Counter-example: why am I wrong?

$x=1, y=-10$

- Smoothed reward
- Reduced Z3 calls

Solution to generalization

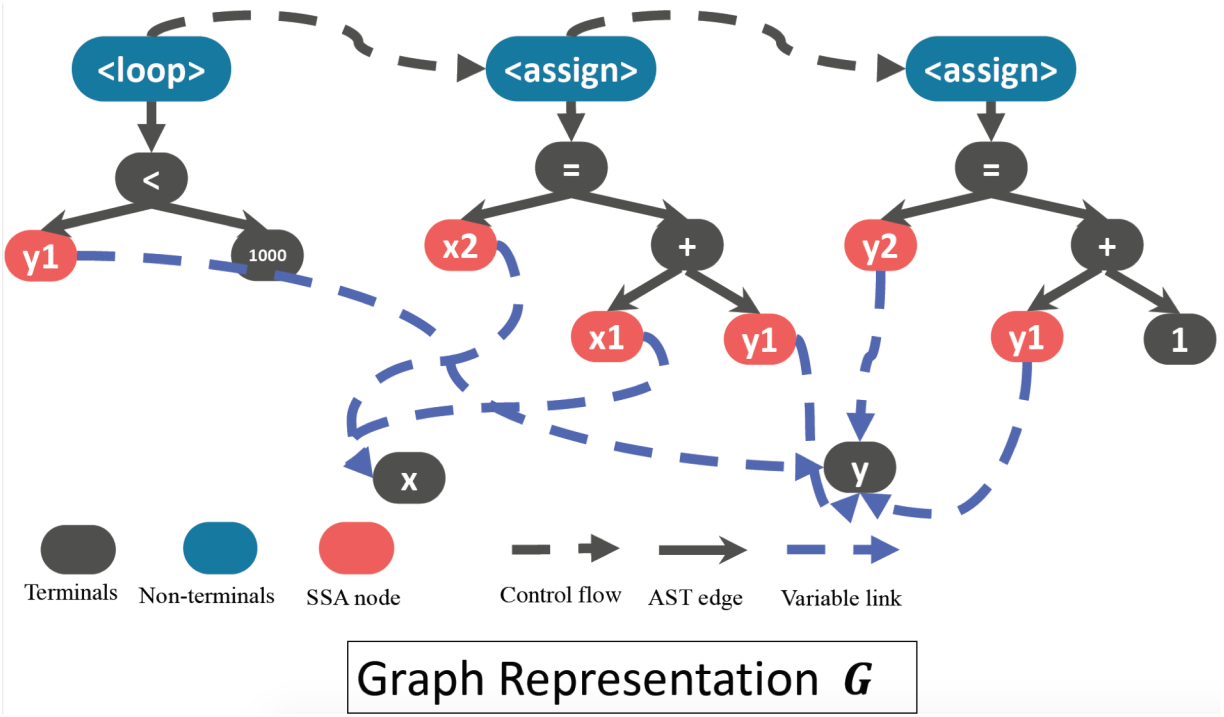
- Transferable graph representation of source code

```
while (y < 1000) {
    x = x + y
    y = y + 1
}
```

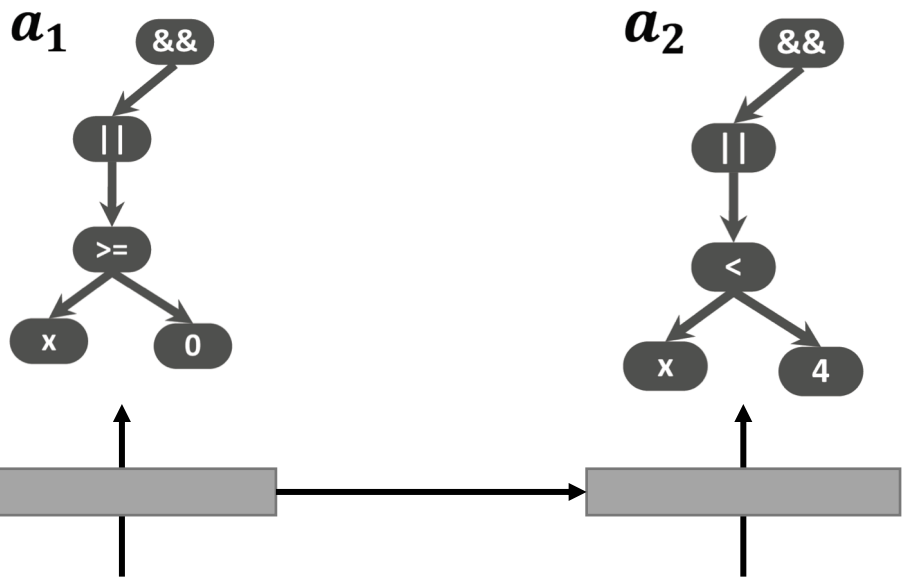
⇓ SSA Transformation

```
x1 = ϕ(x0, x2)
y1 = ϕ(y0, y2)
while (y1 < 1000) {
    x2 = x1 + y1
    y2 = y1 + 1
}
```

⇒



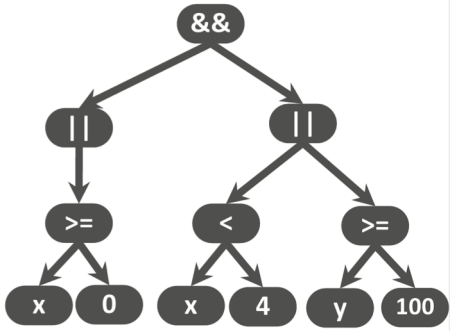
Code2Inv: End-to-end learning framework




```
x1 = ϕ(x0, x2)
y1 = ϕ(y0, y2)
while (y1 < 1000) {
    x2 = x1 + y1
    y2 = y1 + 1
}
```

```
x1 = ϕ(x0, x2)
y1 = ϕ(y0, y2)
while (y1 < 1000) {
    x2 = x1 + y1
    y2 = y1 + 1
}
```

.....



$x \geq 0 \;\&\&\; x < 4 \;||\; y \geq 100$

Experimental evaluation of Code2Inv

- We collect 133 benchmark programs

OOPSLA 2013, Dillig et al

POPL 2016, Garag et al

Experimental evaluation of Code2Inv

- We collect 133 benchmark programs

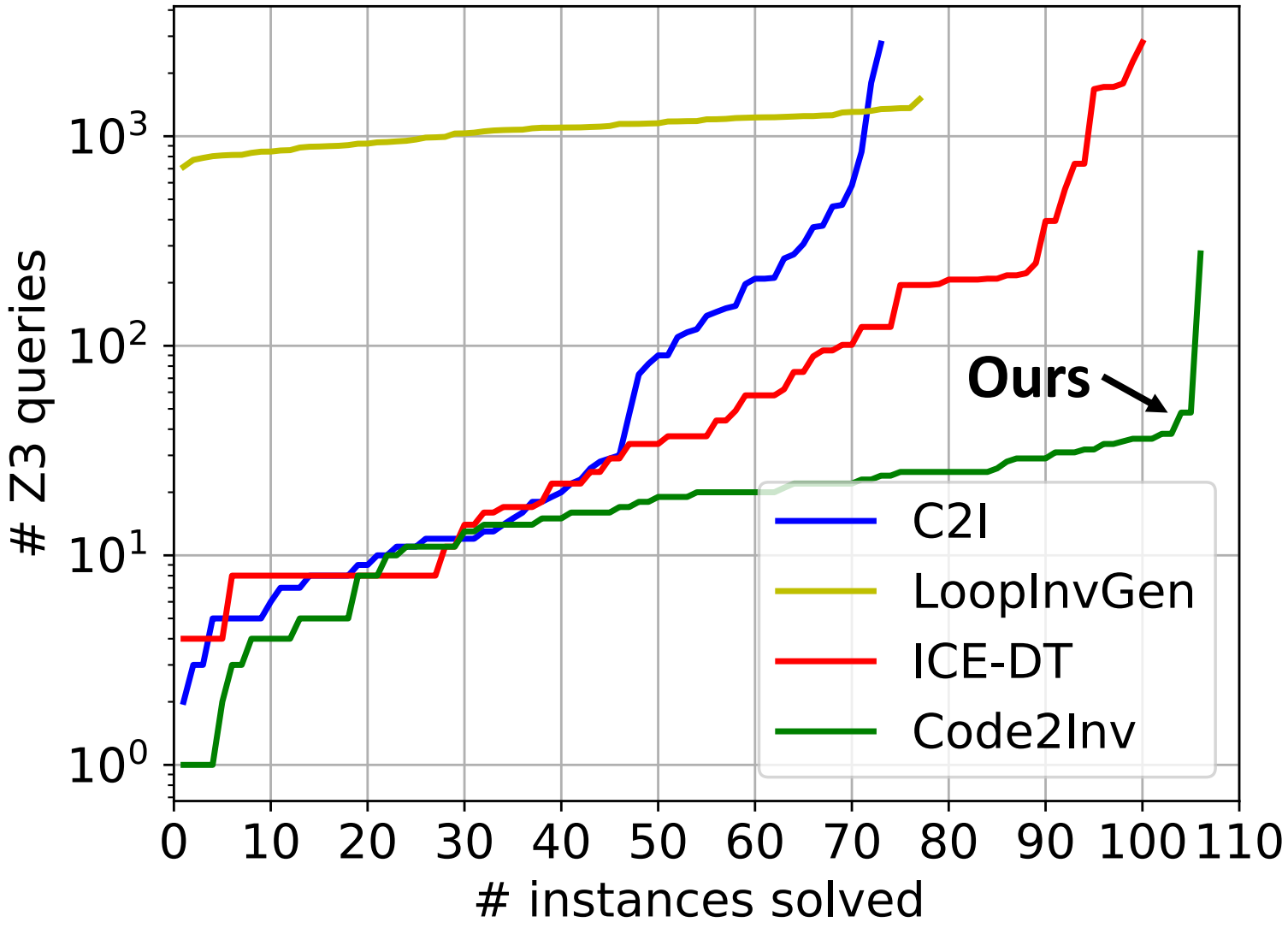
OOPSLA 2013, Dillig et al

POPL 2016, Garag et al

```
int main() {
 int x = 0, y = 0;
 while (*) {
   if (*) {
     x++;
     y = 100;
   } else if (*) {
     if (x >= 4) {
       x++;
       y++;
     }
     if (x < 0) y--;
   }
 }
 assert( x < 4 || y > 2);
}
```

Code2Inv as an out-of-the-box solver

Solved more instances with same # Z3 calls



Generalize to new programs

```
void main (int n) {
    int x = 0

    int m = 0

    while (x < n) {

        if (unknown()) {
            m = x

        }
        x = x + 1

    }
    if (n > 0) {
        assert (m < n)
    }
}
```

Generalize to new programs

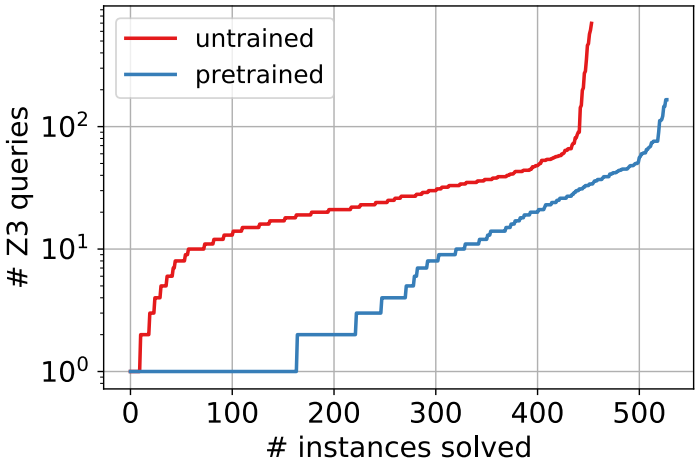
```
void main (int n) {
    int x = 0
                                    int w = 0
    int m = 0
                                    int z = 0
    while (x < n) {

        if (unknown()) {
            m = x

        }
        x = x + 1

    }
    if (n > 0) {
        assert (m < n)
    }
}
```

Generalize to new programs

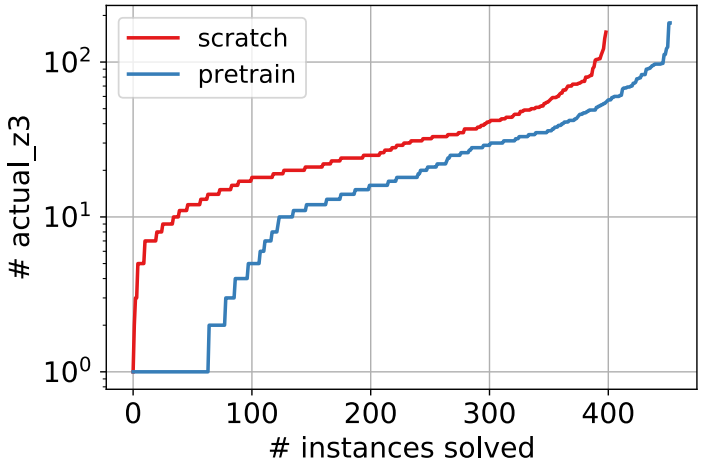
```
void main (int n) {
    int x = 0
                                    int w = 0
    int m = 0
                                    int z = 0
    while (x < n) {
                                    z = z + 1
        if (unknown()) {
            m = x
                                    z = m + 1
        }
        x = x + 1
                                    w = m + x
    }
    if (n > 0) {
        assert (m < n)
    }
}
```

Generalize to new programs

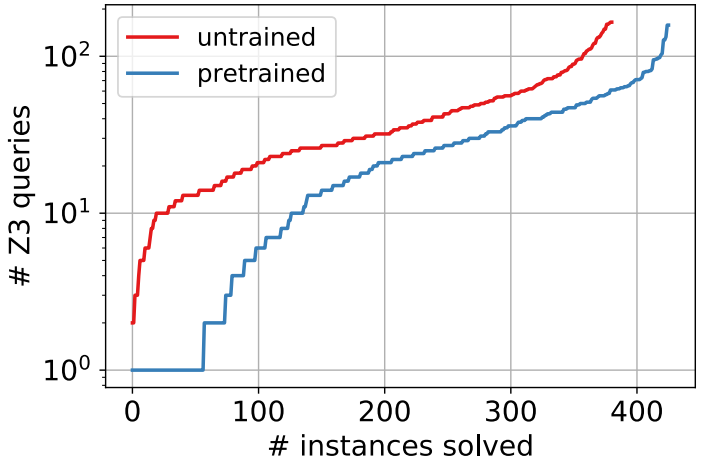

```
void main (int n) {
    int x = 0
    int w = 0
    int m = 0
    int z = 0
    while (x < n) {
        z = z + 1
        if (unknown()) {
            m = x
            z = m + 1
        }
        x = x + 1
        w = m + x
    }
    if (n > 0) {
        assert (m < n)
    }
}
```



1 confounding variable



3 confounding variables



5 confounding variables

Generalization ability of Code2Inv

Poster session:

05:00 -- 07:00 PM

Room 210 & 230 AB #23