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, ...

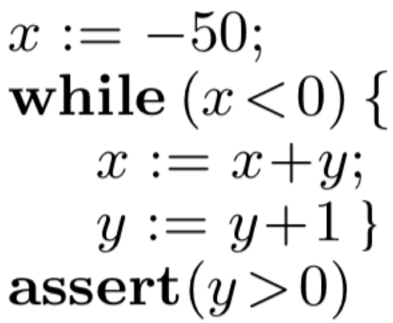
-- Except ‘while rule’

Loop Invariant <> Halting Problem

What is loop invariant?

What is loop invariant?

Program



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

What is loop invariant?

Program

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

Loop Invariant

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

What is loop invariant?

Program

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

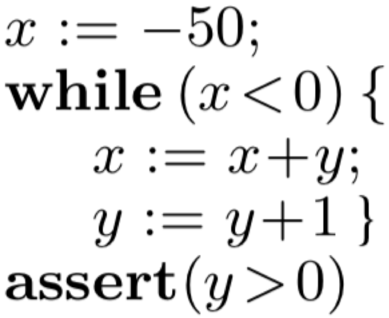
Loop Invariant

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

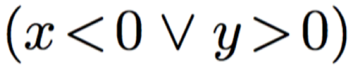
Requirement:

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

Loop Invariant Checker

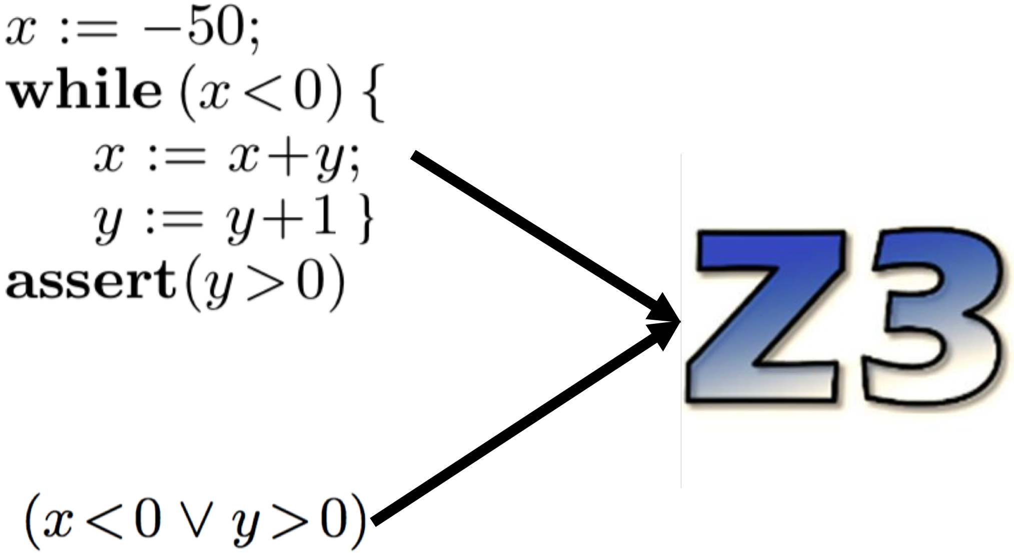


```
$x := -50;$
while $(x < 0)$ {
    $x := x + y;$
    $y := y + 1$ }
assert$(y > 0)$
```



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

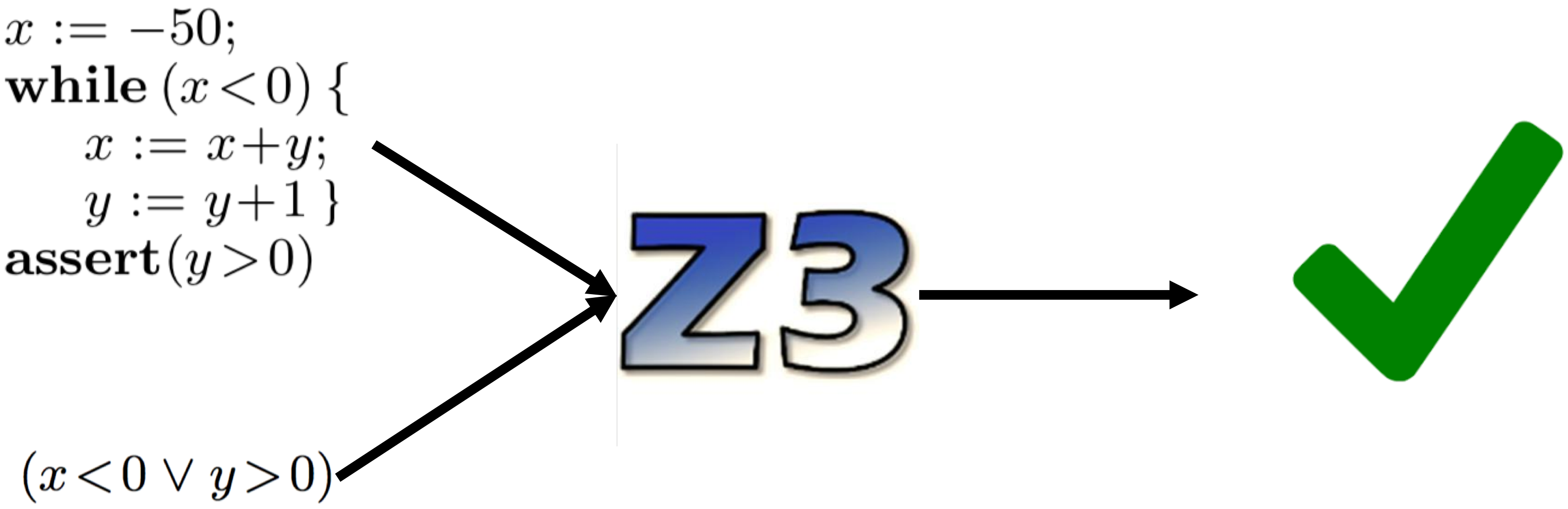
Loop Invariant Checker



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

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

Loop Invariant Checker

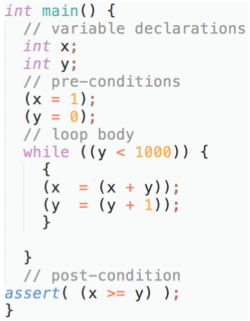


```
$x := -50;$
while $(x < 0)$ {
    $x := x + y;$
    $y := y + 1$ }
assert$(y > 0)$
```

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

Difficulties of learning loop Invariant

1. Highly sparse and non-smooth reward

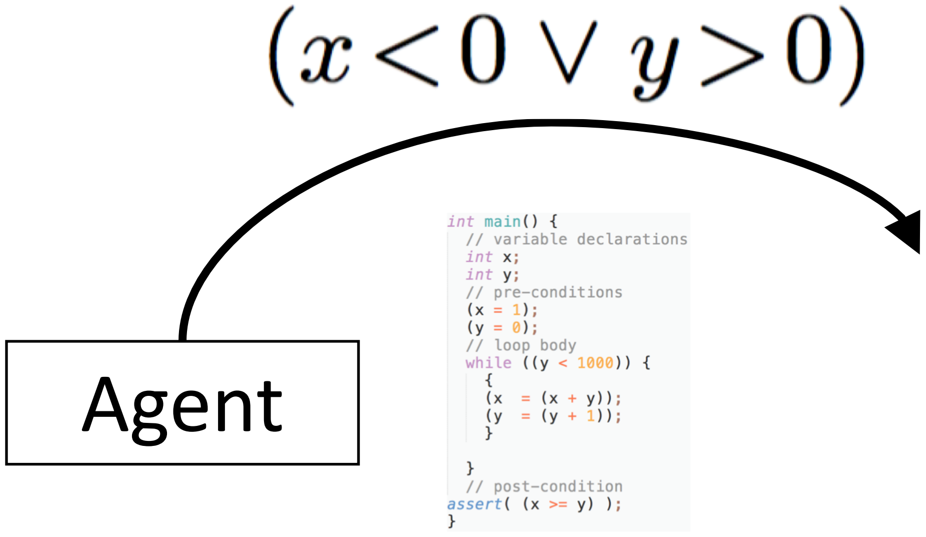


```
int main() {
  // variable declarations
  int x;
  int y;
  // pre-conditions
  (x = 1);
  (y = 0);
  // loop body
  while ((y < 1000)) {
    {
    (x  = (x + y));
    (y  = (y + 1));
    }

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

code

Difficulties of learning loop Invariant

1. Highly sparse and non-smooth reward



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

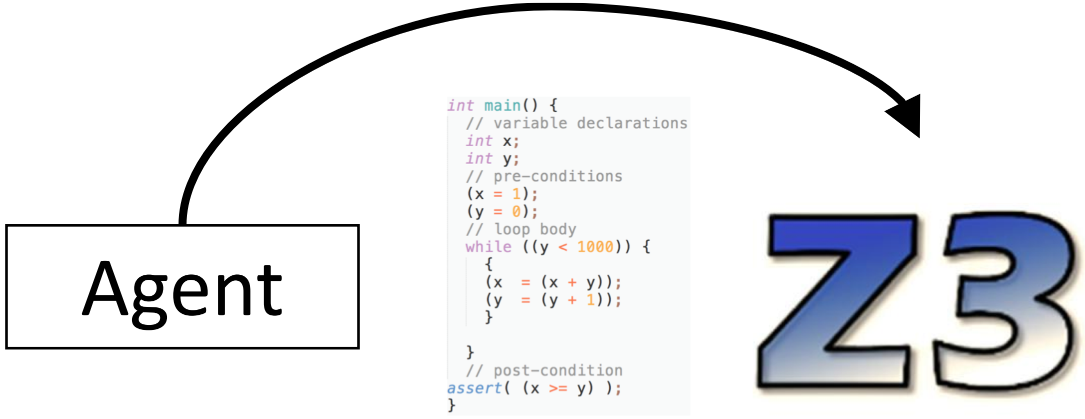
```
int main() {
  // variable declarations
  int x;
  int y;
  // pre-conditions
  (x = 1);
  (y = 0);
  // loop body
  while ((y < 1000)) {
    {
    (x  = (x + y));
    (y  = (y + 1));
    }

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

code

Difficulties of learning loop Invariant

1. Highly sparse and non-smooth reward

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

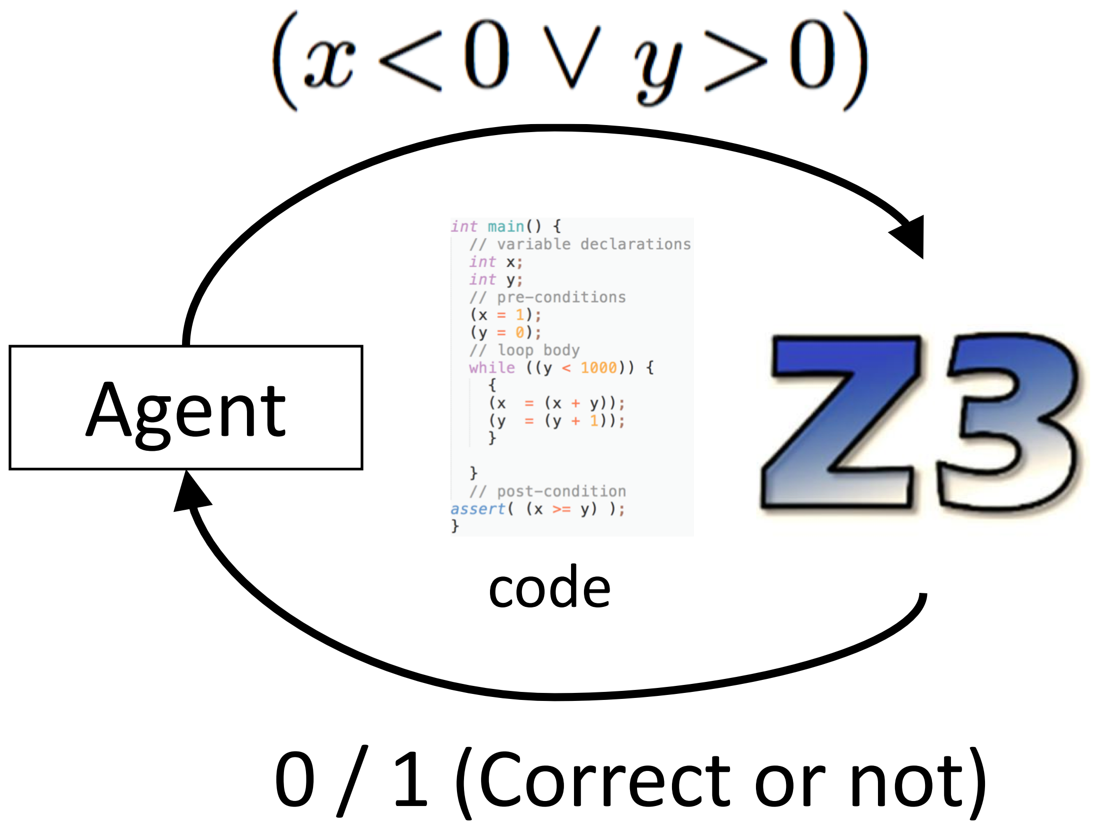


```
int main() {
  // variable declarations
  int x;
  int y;
  // pre-conditions
  (x = 1);
  (y = 0);
  // loop body
  while ((y < 1000)) {
    {
    (x  = (x + y));
    (y  = (y + 1));
    }

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

code

Difficulties of learning loop Invariant

1. Highly sparse and non-smooth reward



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

```
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) );
}
```

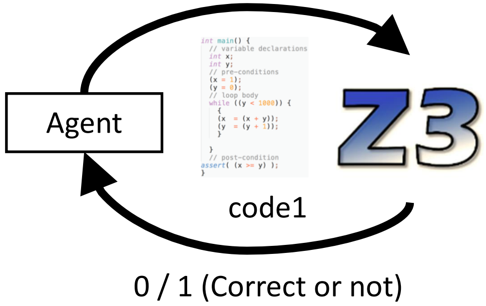
code

0 / 1 (Correct or not)

Difficulties of learning loop invariant

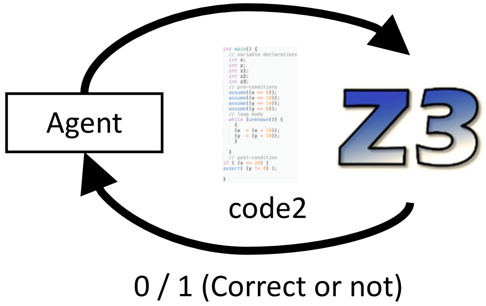
2. Generalization ability

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



```
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) );
}
```

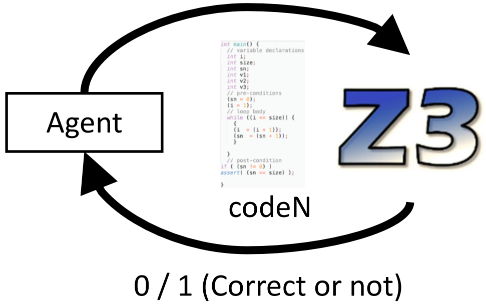
$z_1 \leq 2 \wedge z_2 > 0$



```
int main() {
  // variable declarations
  int x;
  int y;
  int z1;
  int z2;
  int z3;
  // pre-conditions
  assume((x >= 0));
  assume((x <= 10));
  assume((y <= 10));
  assume((y >= 0));
  // loop body
  while (unknown()) {
    {
    (x  = (x + 10));
    (y  = (y + 10));
    }

  }
  // post-condition
if ( (x == 20) )
assert( (y != 0) );

}
```

......

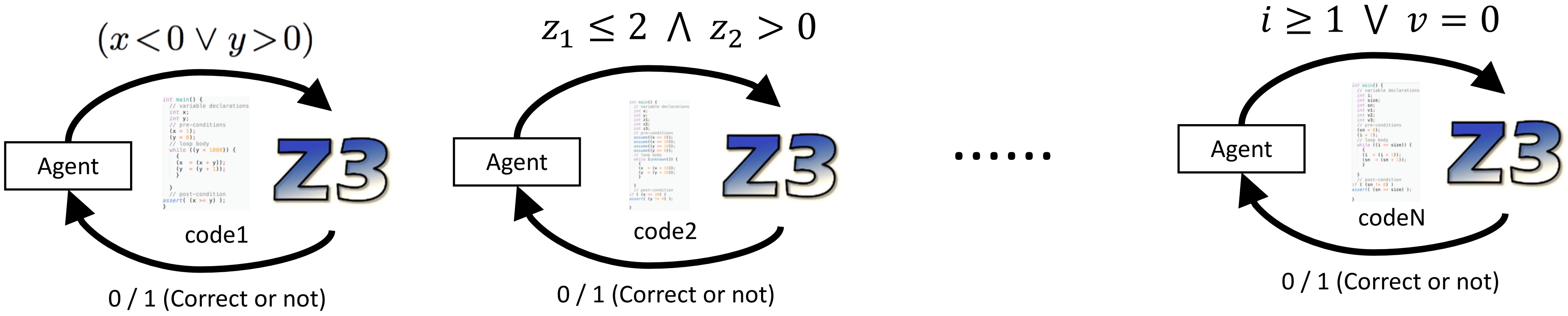
$i \geq 1 \vee v = 0$



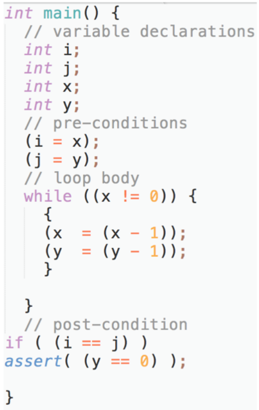
```
int main() {
  // variable declarations
  int i;
  int size;
  int sn;
  int v1;
  int v2;
  int v3;
  // pre-conditions
  (sn = 0);
  (i = 1);
  // loop body
  while ((i <= size)) {
    {
    (i  = (i + 1));
    (sn  = (sn + 1));
    }

  }
  // post-condition
if ( (sn != 0) )
assert( (sn == size) );

}
```

Difficulties of learning loop invariant

2. Generalization ability



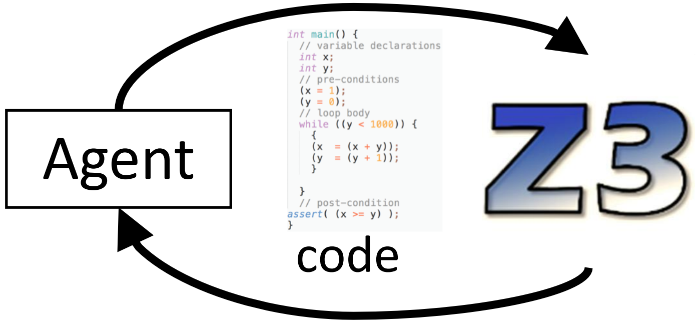
New code



```
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



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

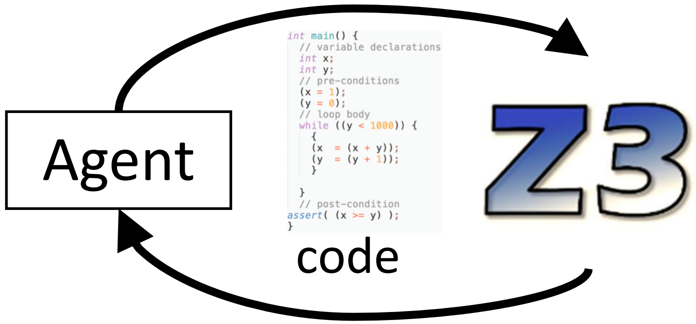
```
int main() {
  // variable declarations
  int x;
  int y;
  // pre-conditions
  (x = 1);
  (y = 0);
  // loop body
  while ((y < 1000)) {
    {
    (x  = (x + y));
    (y  = (y + 1));
    }

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

code

0 (not correct)

Solution to sparsity and non-smoothness

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



```
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) );
}
```

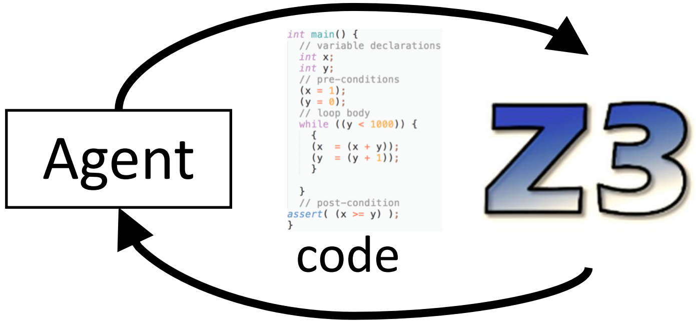
0 (not correct)

Counter-example: why am I wrong?

$x = 1, y = -10$

Solution to sparsity and non-smoothness

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



Agent

```
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

0 (not correct)

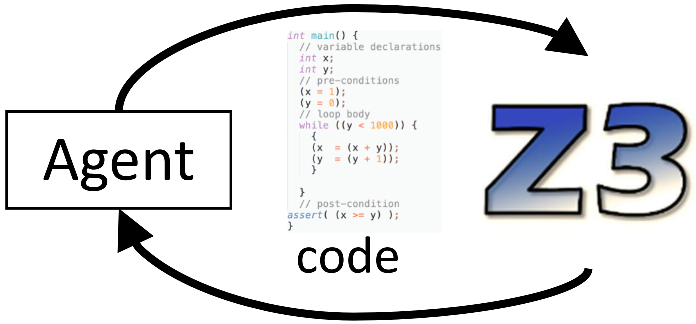
Counter-example: why am I wrong?

$x = 1, y = -10$

Collection of counter-examples:

Solution to sparsity and non-smoothness

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



```
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) );
}
```

0 (not correct)

Counter-example: why am I wrong?

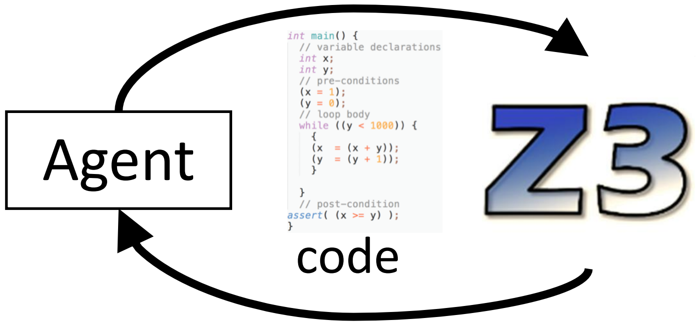
$x = 1, y = -10$

Collection of counter-examples:

Pre	Inv	Post
	$x = 3, y = -2$	
	$x = 3, y = -1$	
	$x = 3, y = -1$	
$x = 0, y = -2$	$x = 2, y = -2$	
$x = 0, y = -1$	$x = 2, y = -1$	$x = 0, y = -4$
$x = 1, y = -1$	$x = 2, y = -1$	$x = 0, y = -3$

Solution to sparsity and non-smoothness

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



Agent

```
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

0 (not correct)

Counter-example: why am I wrong?

$x = 1, y = -10$

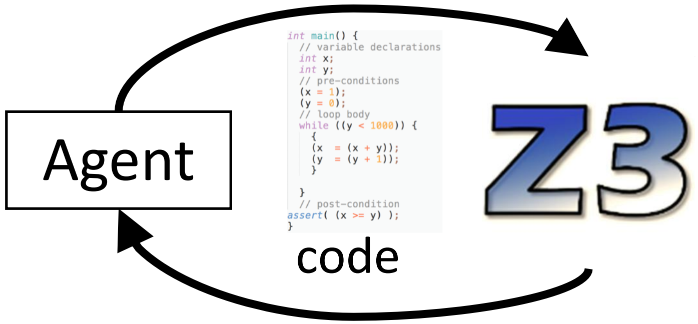
Collection of counter-examples:

Pre	Inv	Post
	$x = 3, y = -2$	
	$x = 3, y = -1$	
	$x = 3, y = -1$	
$x = 0, y = -2$	$x = 2, y = -2$	
$x = 0, y = -1$	$x = 2, y = -1$	$x = 0, y = -4$
$x = 1, y = -1$	$x = 2, y = -1$	$x = 0, y = -3$

- Smoothed reward

Solution to sparsity and non-smoothness

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



```
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) );
}
```

0 (not correct)

Counter-example: why am I wrong?

$x = 1, y = -10$

Collection of counter-examples:

Pre	Inv	Post
	$x = 3, y = -2$	
	$x = 3, y = -1$	
	$x = 3, y = -1$	
$x = 0, y = -2$	$x = 2, y = -2$	
$x = 0, y = -1$	$x = 2, y = -1$	$x = 0, y = -4$
$x = 1, y = -1$	$x = 2, y = -1$	$x = 0, y = -3$

- Smoothed reward
- Reduced Z3 calls

Solution to generalization

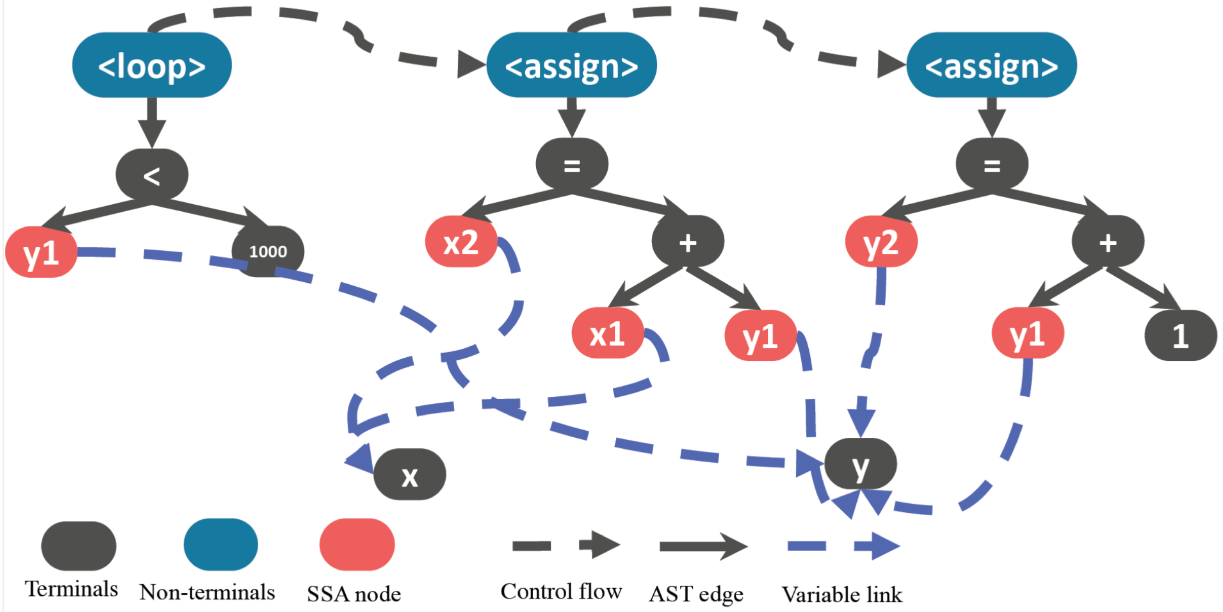
- Transferable graph representation of source code

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

⇓ SSA Transformation

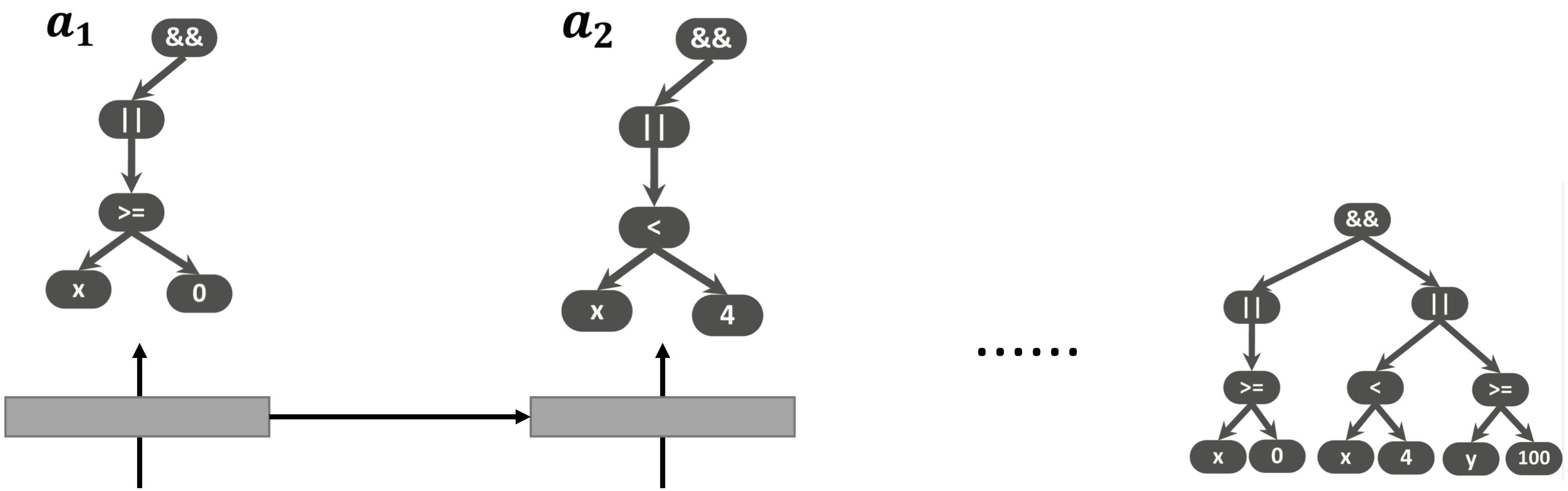
$x_1 = \phi(x_0, x_2)$
$y_1 = \phi(y_0, y_2)$
$\text{while } (y_1 < 1000)\ \{$
$\quad x_2 = x_1 + y_1$
$\quad y_2 = y_1 + 1$
$\}$

⇒



Graph Representation G

Code2Inv: End-to-end learning framework



```
$x_1 = \phi(x_0, x_2)$
$y_1 = \phi(y_0, y_2)$
while ($y_1 < 1000$) {
    $x_2 = x_1 + y_1$
    $y_2 = y_1 + 1$
}
```

```
$x_1 = \phi(x_0, x_2)$
$y_1 = \phi(y_0, y_2)$
while ($y_1 < 1000$) {
    $x_2 = x_1 + y_1$
    $y_2 = y_1 + 1$
}
```

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

Experimental evaluation of Code2Inv

- We collect 133 benchmark programs

SyGuS
Syntax-Guided Synthesis

OOPSLA 2013, Dillig et al

POPL 2016, Garag et al

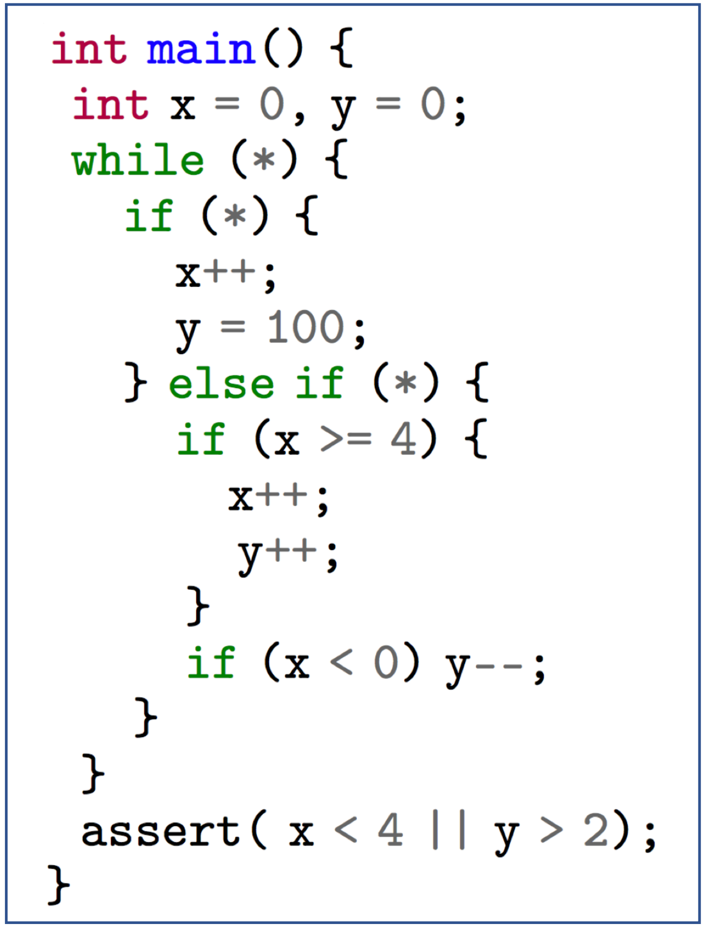
Experimental evaluation of Code2Inv

- We collect 133 benchmark programs

SyGuS
Syntax-Guided Synthesis

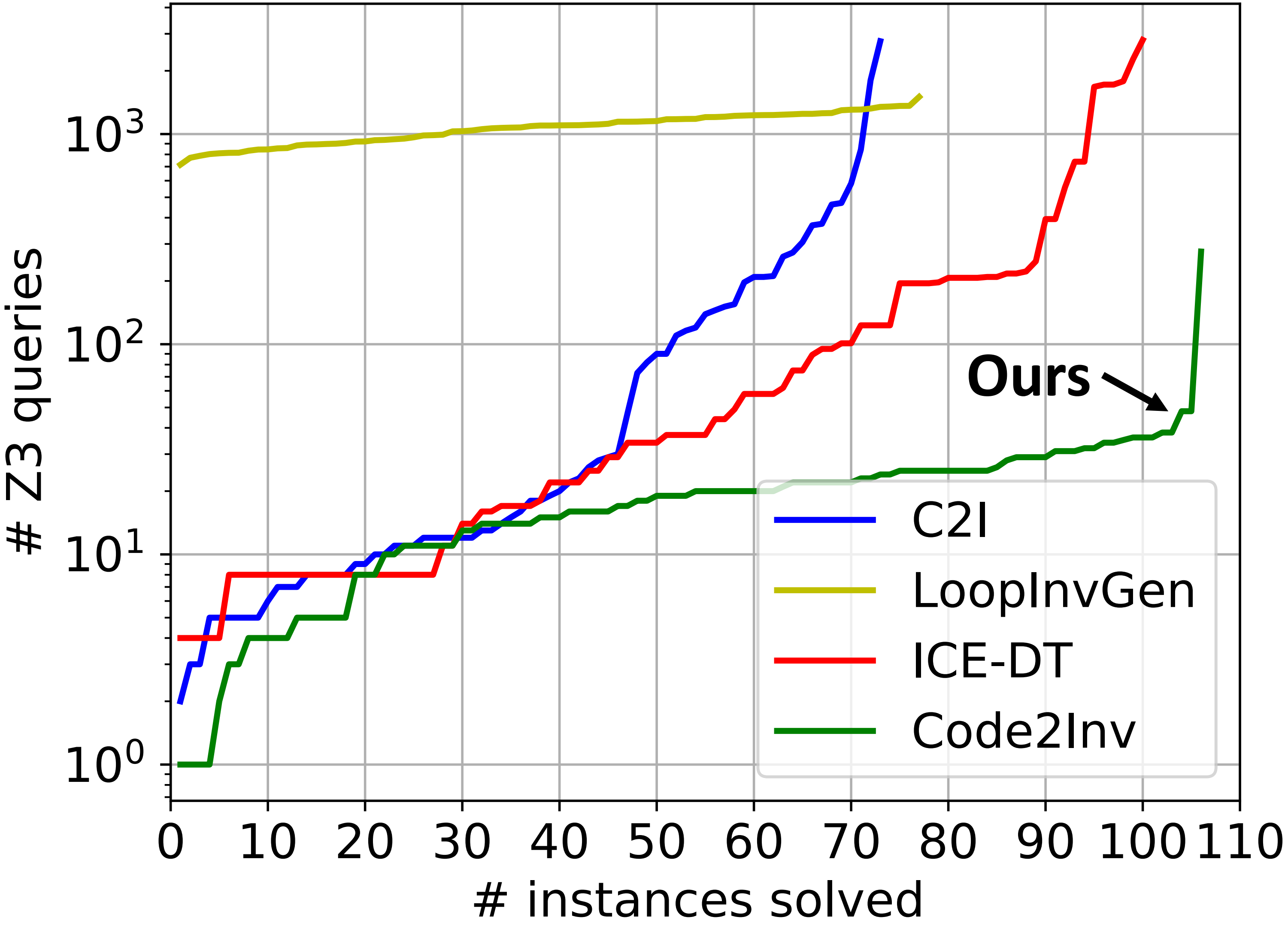
OOPSLA 2013, Dillig et al

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

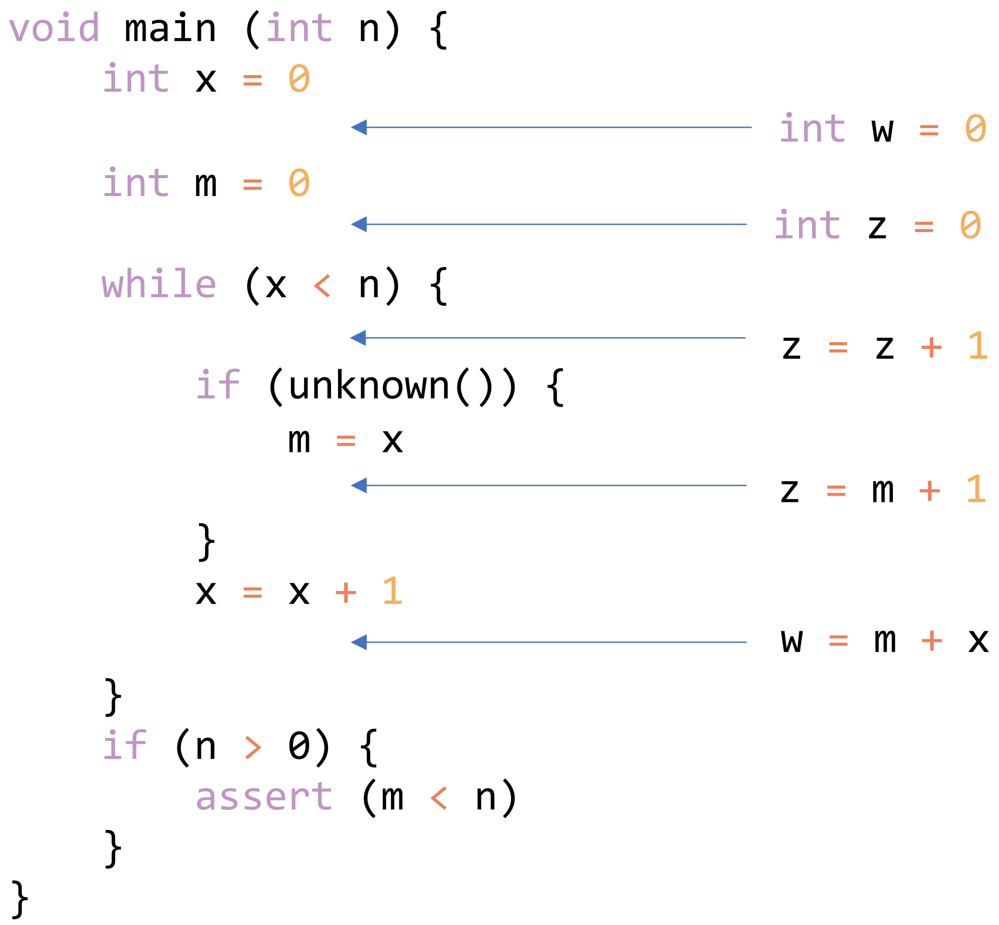
Generalize to new programs

```
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

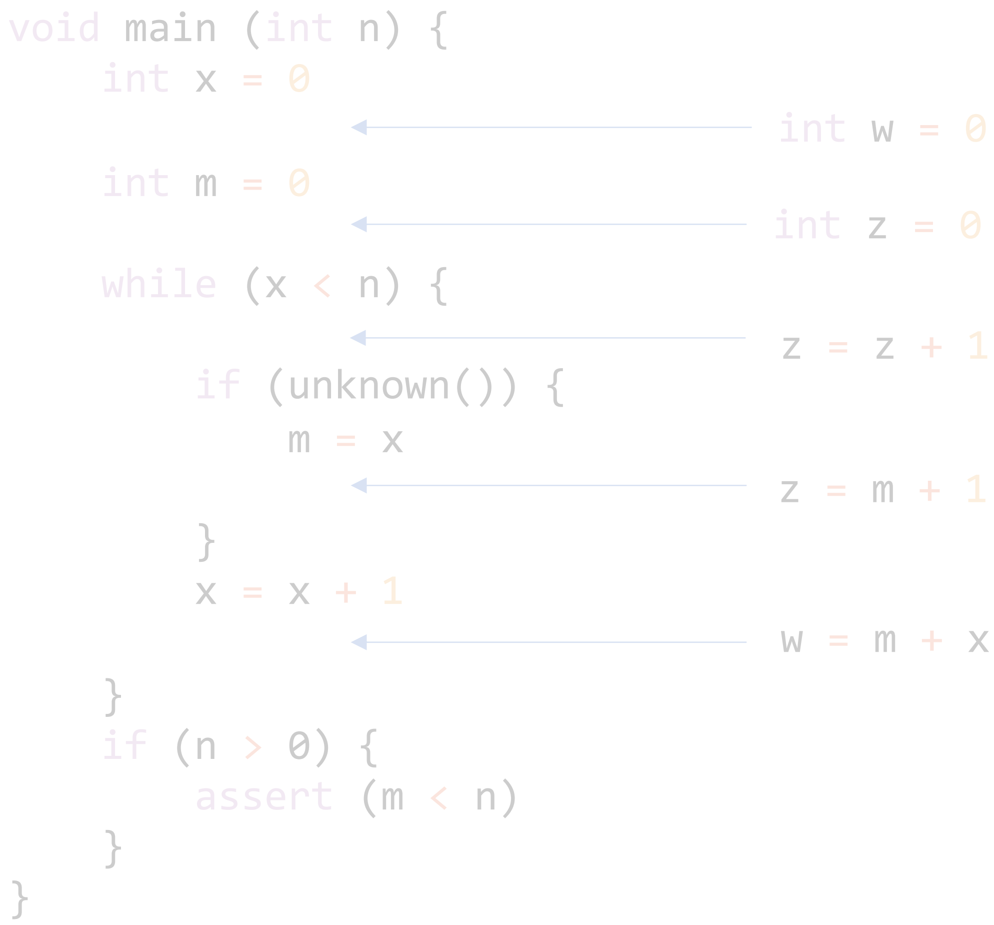
```
void main (int n) {
    int x = 0
                  ←————————————————  int w = 0
    int m = 0
                  ←————————————————  int z = 0
    while (x < n) {

        if (unknown()) {
            m = x

        }
        x = x + 1

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

Generalize to new programs



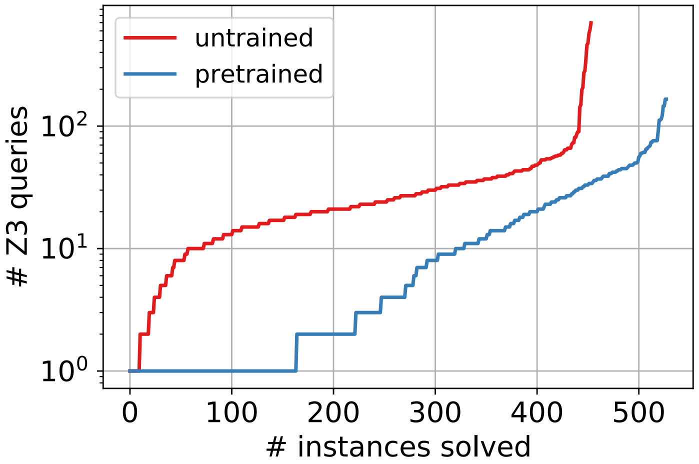
```
void main (int n) {
    int x = 0
                                       int w = 0
    int m = 0
                                       int z = 0
    while (x < n) {
                                       z = z + 1
        if (unknown()) {
            m = x
                                      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

Generalize to new programs

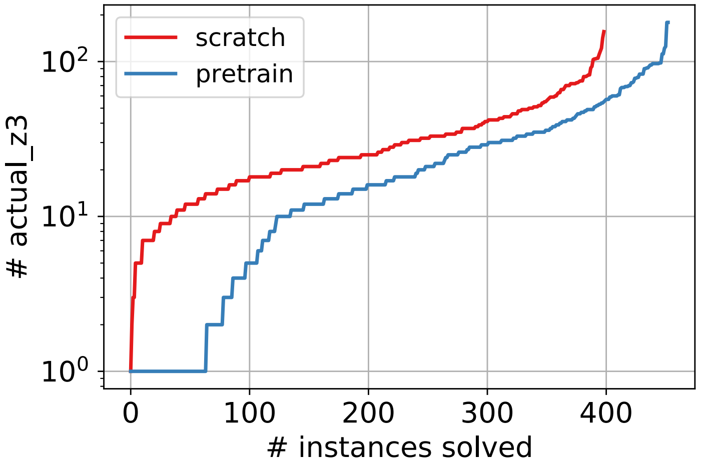


```
void main (int n) {
    int x = 0
                                  int w = 0
    int m = 0
                                  int z = 0
    while (x < n) {
                                  z = z + 1
        if (unknown()) {
            m = x
                                  z = m + 1
        }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

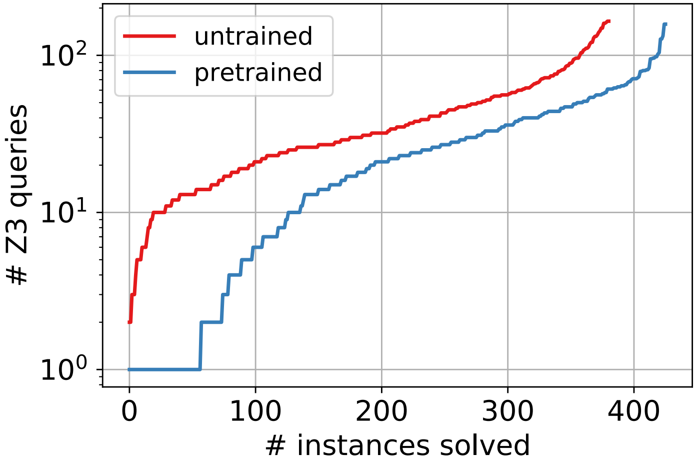
```
void main (int n) {
    int x = 0
    int w = 0
    int m = 0
    int z = 0
    while (x < n) {
        z = z + 1
        if (unknown()) {
            m = x
            z = m + 1
        }
        x = x + 1
        w = m + x
    }
    if (n > 0) {
        assert (m < n)
    }
}
```



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