SymInfer: Inferring Program Invariants using Symbolic States

ThanhVu (Vu) Nguyen*,

Matthew B. Dwyer*, Willem Visser†

*University of Nebraska-Lincoln, †Stellenbosch University

ASE 2017

Introduction

Program invariants are asserted properties, such as relations among variables that always hold at certain locations in a program

- Pre/Post conditions, loop invariants, assertions

Introduction

Program invariants are asserted properties, such as relations among variables that always hold at certain locations in a program

- Pre/Post conditions, loop invariants, assertions

Numerical invariants, e.g., relations among numerical variables

- E.g., $x = 2y + 3, 0 \leq idx \leq |arr| - 1, x \leq y^2, x = qy + r$
- *Nonlinear* polynomial invariants: $x \leq y^2, x = qy + r, \ldots$

Introduction

Program invariants are asserted properties, such as relations among variables that always hold at certain locations in a program

- Pre/Post conditions, loop invariants, assertions

Numerical invariants, e.g., relations among numerical variables

- E.g., $x = 2y + 3, 0 \leq idx \leq |arr| - 1, x \leq y^2, x = qy + r$
- *Nonlinear* polynomial invariants: $x \leq y^2, x = qy + r, \ldots$

Techniques for automatic invariant generation

- *Statically* examine program code, *dynamically* analyze concrete states (traces), or hybridization of dynamic inference and static checking
- SymInfer: hybridization using *symbolic states*
  - Symbolic states: obtained from symbolic execution, intermediate representation of states, consist of path conditions and local variables
  - Infer: use symbolic states to generate sample traces and infer invariants
  - Check: use symbolic states to check candidate invariants

Example: Numerical Invariants

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

What does this program do? What properties hold at L1 and L2?

Example: Numerical Invariants

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

What does this program do? What properties hold at L1 and L2?

SymInfer automatically generates

- loop invariants at L1:
  $x = qy + r, \quad b = ya, \quad y \leq b,$
  $b \leq r, \quad r \leq x, \quad a \leq b, \quad 2 \leq a + y$
- postconditions at L2:
  $x = qy + r, \quad 1 \leq q + r,$
  $r \leq y - 1, \quad 0 \leq r, \quad r \leq x$

Example: Numerical Invariants

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

What does this program do? What properties hold at L1 and L2?

SymInfer automatically generates

- loop invariants at L1:
  $x = qy + r, \quad b = ya, \quad y \leq b,$
  $b \leq r, \quad r \leq x, \quad a \leq b, \quad 2 \leq a + y$
- postconditions at L2:
  $x = qy + r, \quad 1 \leq q + r,$
  $r \leq y - 1, \quad 0 \leq r, \quad r \leq x$
- Invariants describe program's semantic, e.g., $x = qy + r$ for integer division and reveal useful information, e.g., remainder r is non-negative

Symbolic States

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

Run *symbolic execution* to obtain

- Path conditions over input variables
- Relationships among local variables

Symbolic States

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

Run *symbolic execution* to obtain

- Path conditions over input variables
- Relationships among local variables
- At L1:

Pathconds	**Locals**
$x \geq y \wedge y > 0$	$q = 0 \wedge r = x \wedge a = 1 \wedge b = y$
$x \geq 2y \wedge y > 0$	$q = 0 \wedge r = x \wedge a = 2 \wedge b = 2y$
$4y > x \geq 2y + y \wedge y > 0$	$q = 2 \wedge r = x - 2y \wedge a = 1 \wedge b = y$
$\vdots$	$\vdots$

Symbolic States

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

Run *symbolic execution* to obtain

- Path conditions over input variables
- Relationships among local variables
- At L1:

Pathconds	**Locals**
$x \geq y \wedge y > 0$	$q = 0 \wedge r = x \wedge a = 1 \wedge b = y$
$x \geq 2y \wedge y > 0$	$q = 0 \wedge r = x \wedge a = 2 \wedge b = 2y$
$4y > x \geq 2y + y \wedge y > 0$	$q = 2 \wedge r = x - 2y \wedge a = 1 \wedge b = y$
$\vdots$	$\vdots$

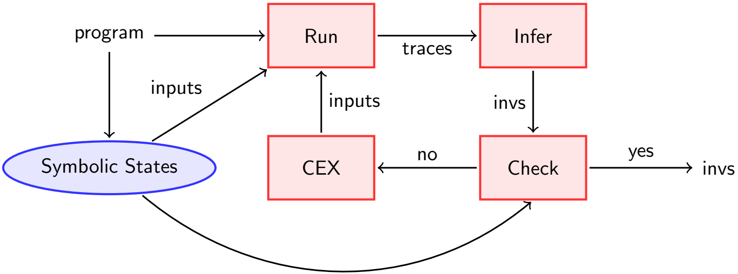
Symbolic states at L1

- Disjunctions of pathconds and locals

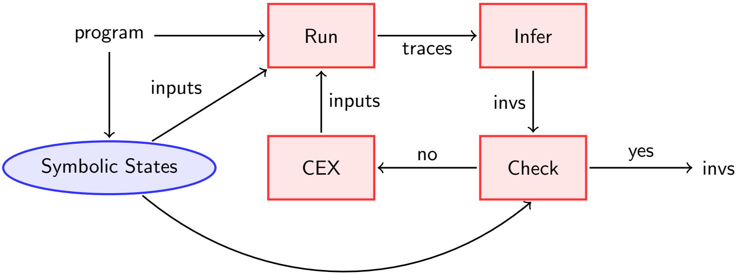
$$(x \geq y \wedge y > 0 \wedge q = 0 \wedge r = x \wedge a = 1 \wedge b = y) \quad \vee$$
$$(x \geq 2y \wedge y > 0 \wedge q = 0 \wedge r = x \wedge a = 2 \wedge b = 2y) \quad \vee \quad \ldots$$

- An intermediate representation of states

SymInfer: Invariants Inference using Symbolic States



SymInfer: Invariants Inference using Symbolic States



- Use symbolic states for both inference and checking
- An iterative approach
  - Inferring: use symbolic states to generate traces, then apply DIG's algorithms to infer numerical invariants from traces
  - Checking: use symbolic states to check candidate invariants and generate counterexample traces

Example: Dynamic Inference using DIG

```
int cohendiv(int x, int y){
  assert(x>0 ; y>0);
  int q=0; int r=x;
  while(r >= y){
    int a=1; int b=y;
    while[L1](r >= 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  return q;
}
```

Example: Dynamic Inference using DIG

```
int cohendiv(int x, int y){
  assert(x>0 ; y>0);
  int q=0; int r=x;
  while(r >= y){
    int a=1; int b=y;
    while[L1](r >= 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  return q;
}
```

Traces:

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
			⋮		
4	1	1	1	0	4
4	1	2	2	0	4
			⋮		

Example: Dynamic Inference using DIG

```
int cohendiv(int x, int y){
  assert(x>0 ; y>0);
  int q=0; int r=x;
  while(r >= y){
    int a=1; int b=y;
    while[L1](r >= 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  return q;
}
```

Traces:

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
			⋮		
4	1	1	1	0	4
4	1	2	2	0	4
			⋮		

Loop invariants at L1:

$$\begin{array}{llll} \textit{equations}: & x = qy + r & b = ya & \\ \textit{inequalities}: & 2 \leq a + y & a \leq b & y \leq b \\ & b \leq r & r \leq x & \end{array}$$

Infer Nonlinear Equations using Equation Solver

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Infer Nonlinear Equations using Equation Solver

- Terms and degrees

$$V = \{r, y, a\};\ \texttt{deg} = 2$$
$$\downarrow$$
$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Infer Nonlinear Equations using Equation Solver

- Terms and degrees

$$V = \{r, y, a\};\ \texttt{deg} = 2$$
$$\downarrow$$
$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

- Nonlinear equation template

$$c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

Infer Nonlinear Equations using Equation Solver

x	y	a	b	q	r
15	**2**	**1**	2	0	**15**
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

- Terms and degrees

$$V = \{r, y, a\};\ \text{deg} = 2$$
$$\downarrow$$
$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

- Nonlinear equation template

$$c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0$$

- System of *linear* equations

$$\begin{array}{rcl} \text{trace } 1 & \rightarrow & \{r = 15, y = 2, a = 1\} \\ \text{eq } 1 & \rightarrow & c_1 + 15c_2 + 2c_3 + c_4 + 30c_5 + 15c_6 + 2c_7 + 225c_8 + 4c_9 + c_{10} = 0 \\ & \vdots & \end{array}$$

Infer Nonlinear Equations using Equation Solver

- Terms and degrees

$$V = \{r, y, a\};\ \texttt{deg} = 2$$
$$\downarrow$$
$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4

- Nonlinear equation template

$$c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0$$

- System of *linear* equations

$$\begin{aligned} \text{trace 1} \quad &\rightarrow \quad \{r = 15, y = 2, a = 1\} \\ \text{eq 1} \quad &\rightarrow \quad c_1 + 15c_2 + 2c_3 + c_4 + 30c_5 + 15c_6 + 2c_7 + 225c_8 + 4c_9 + c_{10} = 0 \\ &\ \ \vdots \end{aligned}$$

- Solve for coefficients c_i

$$V = \{x, y, a, b, q, r\};\ \texttt{deg} = 2 \qquad \longrightarrow \qquad x = qy + r, b = ya$$

Checking Using Symbolic States

General Idea

- **Goal**: prove/refute candidate invariants (I) using symbolic states (S)
- **Approach**: call SMT solver to check for validity of $S \Rightarrow I$
  - *valid*: invariant is valid and accepted
  - *invalid*: invariant is spurious and rejected, solver produces cex's to help inference

Checking Using Symbolic States

General Idea

- **Goal**: prove/refute candidate invariants (I) using symbolic states (S)
- **Approach**: call SMT solver to check for validity of $S \Rightarrow I$
  - *valid*: invariant is valid and accepted
  - *invalid*: invariant is spurious and rejected, solver produces cex's to help inference

Implementation: use JPF/SPF to obtain symbolic states

- Bounded by depth k: invariants only valid over symbolic states S computed with k
- If I is valid with S_k, then check again if I is also valid with S_{k+1} to gaint confidence
- Can be *unsound* (will not attempt all possible depths), but in practice is *very effective* in refuting bad invariants and finding cex's

Evaluation

Setup

- SymInfer works with Java programs, implemented in SAGE/Python (with JPF/SPF and Z3 SMT solver),
- Test machine: 10-core 2.4GHZ CPU, 128GB Ram, Linux OS

Benchmark (3 objectives)

1. Program Understanding: NLA testsuite, 27 programs with nonlinear invariants
2. Complexity Analysis: 19 programs collected from static complexity analysis work
3. Program Verification: HOLA benchmark, 46 programs with assertions, compare against PIE

Example: Program Understanding

```
int cohendiv(int x, int y){
  assert(x>0 && y>0);
  int q=0; int r=x;
  while(r ≥ y){
    int a=1;
    int b=y;
    while[L1](r ≥ 2*b){
      a = 2*a;
      b = 2*b;
    }
    r=r-b;
    q=q+a;
  }
  [L2]
  return q;
}
```

What does this program do? What properties hold at L1 and L2?

SymInfer automatically generates

- loop invariants at L1:
  $x = qy + r, \quad b = ya, \quad y \leq b,$
  $b \leq r, \quad r \leq x, \quad a \leq b, \quad 2 \leq a + y$
- postconditions at L2:
  $x = qy + r, \quad 1 \leq q + r,$
  $r \leq y - 1, \quad 0 \leq r, \quad r \leq x$
- Invariants describe program's semantic, e.g., integer division and reveal useful information, e.g., remainder is non-negative

Results: Program Understanding

Prog	Locs	Invs	Time (s)	Correct
cohendiv	2	10	21.05	✓
divbin	2	11	58.97	✓
manna	1	6	35.33	✓
hard	2	6	29.40	✓
sqrt1	1	5	20.03	✓
dijkstra	2	16	93.01	✓
freire1	1	-	-	-
freire2	1	-	-	-
cohencu	1	4	21.90	✓
egcd1	1	14	122.22	✓
egcd2	2	-	-	-
egcd3	3	-	-	-
prodbin	1	7	56.17	✓
prod4br	1	9	84.37	✓
knuth	1	-	-	-
fermat1	3	17	60.26	✓
fermat2	1	8	36.83	✓
lcm1	3	24	248.17	✓
lcm2	1	7	34.17	✓
geo1	1	8	158.27	✓
geo2	1	9	147.75	✓
geo3	1	-	-	-
ps2	1	3	18.39	✓
ps3	1	3	19.69	✓
ps4	1	3	19.92	✓
ps5	1	3	46.19	✓
ps6	1	3	41.19	✓

Experiment

- *NLA suite*: 27 programs
- Require nonlinear invariants
- Use documented invariants (loop invariants and postconds) as ground truths
- **Goal**: obtain invariants and compare to ground truths

Results: Program Understanding

Prog	Locs	Invs	Time (s)	Correct
cohendiv	2	10	21.05	✓
divbin	2	11	58.97	✓
manna	1	6	35.33	✓
hard	2	6	29.40	✓
sqrt1	1	5	20.03	✓
dijkstra	2	16	93.01	✓
freire1	1	-	-	-
freire2	1	-	-	-
cohencu	1	4	21.90	✓
egcd1	1	14	122.22	✓
egcd2	2	-	-	-
egcd3	3	-	-	-
prodbin	1	7	56.17	✓
prod4br	1	9	84.37	✓
knuth	1	-	-	-
fermat1	3	17	60.26	✓
fermat2	1	8	36.83	✓
lcm1	3	24	248.17	✓
lcm2	1	7	34.17	✓
geo1	1	8	158.27	✓
geo2	1	9	147.75	✓
geo3	1	-	-	-
ps2	1	3	18.39	✓
ps3	1	3	19.69	✓
ps4	1	3	19.92	✓
ps5	1	3	46.19	✓
ps6	1	3	41.19	✓

Experiment

- *NLA suite*: 27 programs
- Require nonlinear invariants
- Use documented invariants (loop invariants and postconds) as ground truths
- **Goal**: obtain invariants and compare to ground truths

Results: SymInfer found correct invariants in 21/27 (✓) programs

- Most results equivalent to or stronger than (imply) ground truths
- Several unexpected and undocumented invariants
- Some invariants reveal "how" program works in details

Example: Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations

Example: Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations
- At first glance: $t = O(MNP)$
- A more precise complexity bound: $t = O(N + NM + P)$

Example: Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations
- At first glance: $t = O(MNP)$
- A more precise complexity bound: $t = O(N + NM + P)$
- SymInfer found a very *unexpected* inv:
  $P^2Mt + PM^2t - PMNt - M^2Nt - PMt^2 + MNt^2 + PMt - PNt - 2MNt + Pt^2 + Mt^2 + Nt^2 - t^3 - Nt + t^2 = 0$

Example: Complexity Analysis

```
void triple(int M, int N, int P){
  assert (0 <= M);
  assert (0 <= N);
  assert (0 <= P);
  int i = 0, j = 0, k = 0;
  int t = 0;
  while(i < N){
    j = 0; t++;
    while(j < M){
      j++; k = i; t++;
      while (k < P){
        k++; t++;
      }
      i = k;
    }
    i++;
  }
  [L]
}
```

Complexity of this program?

- Use t to count loop iterations
- At first glance: $t = O(MNP)$
- A more precise complexity bound: $t = O(N + NM + P)$
- SymInfer found a very *unexpected* inv:
  $$P^2Mt + PM^2t - PMNt - M^2Nt - PMt^2 + MNt^2 + PMt - PNt - 2MNt + Pt^2 + Mt^2 + Nt^2 - t^3 - Nt + t^2 = 0$$
- Solve for t yields the most precise, unpublished bound:
  $$\begin{array}{lll} t = 0 & \text{when} & N = 0, \\ t = P + M + 1 & \text{when} & N \leq P, \\ t = N - M(P - N) & \text{when} & N > P \end{array}$$
- Nonlinear invariants can represent *disjunctive properties* capturing different complexity bounds

Results: Complexity Analysis

Prog	Invs	Time (s)	
cav09_fig1a	1	12.41	✓
cav09_fig1d	1	12.44	✓
cav09_fig2d	3	58.40	✓
cav09_fig3a	3	8.75	✓
cav09_fig5b	6	49.44	✓
pldi09_ex6	6	57.00	✓
pldi09_fig2	6	60.60	✓✓
pldi09_fig4_1	3	56.24	✓
pldi09_fig4_2	5	28.32	✓
pldi09_fig4_3	3	59.19	✓
pldi09_fig4_4	-	-	-
pldi09_fig4_5	3	103.70	✓
popl09_fig2_1	2	50.86	✓✓
popl09_fig2_2	2	53.48	✓✓
popl09_fig3_4	4	58.62	✓
popl09_fig4_1	4	65.19	✓
popl09_fig4_2	2	51.24	✓✓
popl09_fig4_3	5	31.57	✓
popl09_fig4_4	3	36.89	✓

Experiment

- 19 progs from static complexity work
- Obtain postconds representing complexity
- **Goal**: compare against results from prev work

Results: Complexity Analysis

Prog	Invs	Time (s)	
cav09_fig1a	1	12.41	✓
cav09_fig1d	1	12.44	✓
cav09_fig2d	3	58.40	✓
cav09_fig3a	3	8.75	✓
cav09_fig5b	6	49.44	✓
pldi09_ex6	6	57.00	✓
pldi09_fig2	6	60.60	✓✓
pldi09_fig4_1	3	56.24	✓
pldi09_fig4_2	5	28.32	✓
pldi09_fig4_3	3	59.19	✓
pldi09_fig4_4	-	-	-
pldi09_fig4_5	3	103.70	✓
popl09_fig2_1	2	50.86	✓✓
popl09_fig2_2	2	53.48	✓✓
popl09_fig3_4	4	58.62	✓
popl09_fig4_1	4	65.19	✓
popl09_fig4_2	2	51.24	✓✓
popl09_fig4_3	5	31.57	✓
popl09_fig4_4	3	36.89	✓

Experiment

- 19 progs from static complexity work
- Obtain postconds representing complexity
- **Goal**: compare against results from prev work

Results: Obtain equivalent (14 ✓) or more precise bounds (4 ✓✓) in 18/19 progs

Example: Verification

```
void f(int u1, int u2) {
  assert(u1 > 0 && u2 > 0);
  int a = 1, b = 1, c = 2, d = 2;
  int x = 3, y = 3;
  int i1 = 0, i2 = 0;
  while (i1 < u1) {
    i1++;
    x = a + c; y = b + d;
    if ((x + y) % 2 == 0) {
      a++; d++;
    } else { a--;}
    i2 = 0;
    while (i2 < u2 ) {
      i2++; c--; b--;
    }
  }
  [L] //SymInfer found:
  //b + 1 = c, a + 1 = d,
  //a + b <= 2, 2 <= a
  assert(a + c == b + d);
}
```

```
void g(int n, int u1) {
  assert(u1 > 0);
  int x = 0;
  int m = 0;

  while (x < n) {
    if (u1) {
      m = x;
    }
    x = x + 1;
  }
  [L] //SymInfer found:
  //m^2 = nx - m - x, mn = x^2 - x
  //-m  <= x, x <= m + 1, n <= x
  if (n > 0){
    assert(0 <= m && m < n);
  }
}
```

Results: Verification

Experiment

- HOLA benchmark: 46 programs
- Various assertions (mostly postconds)
- **Goal**:
  - Obtain and compare invariants: if match or imply assertions, then assertions hold
  - Also compare with existing tool PIE

Results: Verification

Experiment

- HOLA benchmark: 46 programs
- Various assertions (mostly postconds)
- **Goal**:
  - Obtain and compare invariants: if match or imply assertions, then assertions hold
  - Also compare with existing tool PIE

Results: Found equiv or stronger invariants in 40/46 programs

- Time: median 9.3s, mean 5.4s
- Nonlinear invariants can prove many nontrivial and *unsupported* properties

documentation, code, benchmark programs

https://bitbucket.org/nguyenthanhvuh/symtraces/

Extra Slides

Inferring Octagonal Inequalities

- Basic CEGIR does not work well for inequalities (e.g., $t \leq 1000$)
  - E.g., real inv: $t \leq 1000$
  - Basic CEGIR: iter 1: $t \leq 2$, iter: 2 $t \leq 3$, iter 3: $t \leq 7$, ...
  - Not terminating if t has no bounds

Inferring Octagonal Inequalities

- Basic CEGIR does not work well for inequalities (e.g., $t \leq 1000$)
  - E.g., real inv: $t \leq 1000$
  - Basic CEGIR: iter 1: $t \leq 2$, iter: 2 $t \leq 3$, iter 3: $t \leq 7, \ldots$
  - Not terminating if t has no bounds
- **Approach**: Divide and Conquer
  - Only consider invariants within fixed bounds, e.g., $-k \leq t \leq k$, where $k = 100000$

Inferring Octagonal Inequalities

- Basic CEGIR does not work well for inequalities (e.g., $t \leq 1000$)
  - E.g., real inv: $t \leq 1000$
  - Basic CEGIR: iter 1: $t \leq 2$, iter: 2 $t \leq 3$, iter 3: $t \leq 7, \ldots$
  - Not terminating if t has no bounds
- **Approach**: Divide and Conquer
  - Only consider invariants within fixed bounds, e.g., $-k \leq t \leq k$, where $k = 100000$
  - Finding upperbound (ub): check if $t \leq k$
  - If no (i.e., $t > k$) then will not find ub of t

Inferring Octagonal Inequalities

- Basic CEGIR does not work well for inequalities (e.g., $t \leq 1000$)
  - E.g., real inv: $t \leq 1000$
  - Basic CEGIR: iter 1: $t \leq 2$, iter: 2 $t \leq 3$, iter 3: $t \leq 7, \ldots$
  - Not terminating if t has no bounds
- **Approach**: Divide and Conquer
  - Only consider invariants within fixed bounds, e.g., $-k \leq t \leq k$, where $k = 100000$
  - Finding upperbound (ub): check if $t \leq k$
  - If no (i.e., $t > k$) then will not find ub of t
  - Otherwise use divide and conquer to find ub of t within range $[-k, k]$
    - Compute mid value $mv = (-k + k)/2$, check if $t \leq mv$
    - If yes, find ub of t within the range $[-k, mv]$
    - If no, find ub of t within the range $[mv, k]$

Inferring Octagonal Inequalities

- Basic CEGIR does not work well for inequalities (e.g., $t \leq 1000$)
  - E.g., real inv: $t \leq 1000$
  - Basic CEGIR: iter 1: $t \leq 2$, iter: 2 $t \leq 3$, iter 3: $t \leq 7, \ldots$
  - Not terminating if t has no bounds
- **Approach**: Divide and Conquer
  - Only consider invariants within fixed bounds, e.g., $-k \leq t \leq k$, where $k = 100000$
  - Finding upperbound (ub): check if $t \leq k$
  - If no (i.e., $t > k$) then will not find ub of t
  - Otherwise use divide and conquer to find ub of t within range $[-k, k]$
    - Compute mid value $mv = (-k + k)/2$, check if $t \leq mv$
    - If yes, find ub of t within the range $[-k, mv]$
    - If no, find ub of t within the range $[mv, k]$
- Support octagonal invariants: term t represent $x, y, x - y, x + y, -x - y, \ldots$

Using Symbolic States for Invariant Inference

- Reusability: pre-compute and reuse symbolic states at L, e.g., for checking
- Expressiveness: a symbolic state (e.g., $x \geq 0, y \geq x$) represents many concrete states and also encodes relationahips among variables ($e.g., y \geq x$)
- Diversity: each symbolic state represent a different program "path", produce better traces
- Usability and Optimization: encoded logical formulas, checked with different solvers and optimized (e.g., perform slicing when checking)