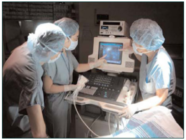
DIG: Dynamic Invariant Generation

ThanhVu (Vu) Nguyen

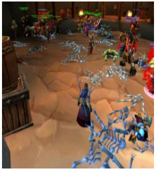
University of Nebraska, Lincoln

CSE Colloquium, Sept 2017

The Problem

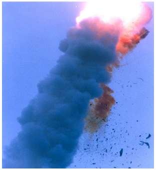




Software Bugs


World of Warcraft bug

Therac-25 machines

X-rays overdose


Ariane-5 rocket self-destructs

North America blackout

The Cost

– Mozilla Developer

"Everyday, almost 300 bugs appear [..] far too many for only the Mozilla programmers to handle."

Software bugs annually cost 0.6% of the U.S GDP and $312 billion to the global economy

Average time to fix a security-critical error: 28 days

Program Analysis

Check Code

Write Code

Automated program analysis techniques and tools can decrease debugging time by an average of 26% and $41 billion annually

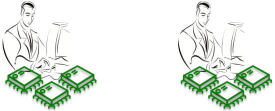
Program Analysis

Check Code

Write Code

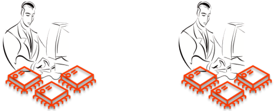
Automated program analysis techniques and tools can decrease debugging time by an average of 26% and $41 billion annually

Program Verification



Check if a program satisfies a given specification

Program Synthesis



Generate a program that meets a given specification

Invariant Generation and Template-based Synthesis

Invariant Generation

```
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;
}
```

- Discover invariant properties at certain program locations
- Answer the question *"what does this program do ?"*

Invariant Generation and Template-based Synthesis

Invariant Generation

```
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;
}
```

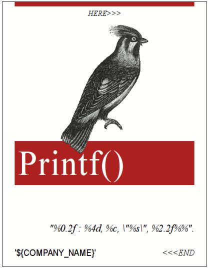
- Discover invariant properties at certain program locations
- Answer the question *"what does this program do ?"*

Template-based Synthesis

```
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 = [???]; b = 2*b;
    }
    r=r-b; q=q+a;
  }

  return [???];
}
```

- Create code under specific templates from partially completed programs
- Can be used for program repair

How We Analyze Programs



File Edit Options Buffers Tools Help

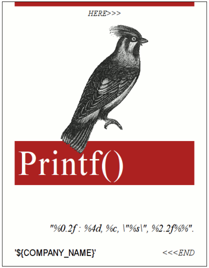
```python
def intdiv(x, y):

    q = 0
    r = x
    while r >= y:
        a = 1
        b = y

        while r >= 2*b:
            a = 2 * a
            b = 2 * b
        r = r - b
        q = q + a

    print "x %d, y %d, q %d, r %d" %(x,y,q,r)
    return q,r
```

-U:--- intdiv.py All (18,0) (Python)--5:

```
x   0, y  1, q   0, r 0
x   1, y  1, q   1, r 0
x   1, y  5, q   0, r 1
x   1, y 10, q   0, r 1
x   3, y  1, q   3, r 0
x   3, y  4, q   0, r 3
x   3, y  7, q   0, r 3
x   8, y  1, q   8, r 0
x   8, y  2, q   4, r 0
x   8, y  9, q   0, r 8
x   8, y 10, q   0, r 8
x  15, y  1, q  15, r 0
x  15, y  5, q   3, r 0
x  15, y  7, q   2, r 1
x  20, y  2, q  10, r 0
x  20, y  7, q   2, r 6
x  20, y 10, q   2, r 0
x 100, y  1, q 100, r 0
x 100, y  5, q  20, r 0
x 100, y 10, q  10, r 0
```

-U:--- intdiv.traces All (21,0)

How We Analyze Programs




```
File  Edit  Options  Buffers  Tools  Help

def intdiv(x, y):

    q = 0
    r = x
    while r >= y:
        a = 1
        b = y

        while r >= 2*b:
            a = 2 * a
            b = 2 * b
        r = r - b
        q = q + a

    print "x %d, y %d, q %d, r %d" %(x,y,q,r)
    return q,r

-U:---  intdiv.py      All (18,0)     (Python)--5:
```

```
x   0, y  1, q   0, r 0
x   1, y  1, q   1, r 0
x   1, y  5, q   0, r 1
x   1, y 10, q   0, r 1
x   3, y  1, q   3, r 0
x   3, y  4, q   0, r 3
x   3, y  7, q   0, r 3
x   8, y  1, q   8, r 0
x   8, y  2, q   4, r 0
x   8, y  9, q   0, r 8
x   8, y 10, q   0, r 8
x  15, y  1, q  15, r 0
x  15, y  5, q   3, r 0
x  15, y  7, q   2, r 1
x  20, y  2, q  10, r 0
x  20, y  7, q   2, r 6
x  20, y 10, q   2, r 0
x 100, y  1, q 100, r 0
x 100, y  5, q  20, r 0
x 100, y 10, q  10, r 0

-U:---  intdiv.traces   All (21,0)
```

```
File  Edit  Options  Buffers  Tools  Python  Help

def intdiv(x, y):
    assert y != 0

    # .. compute result ..

    assert r >= 0
    assert x >= q

    return q,r

--:---  intdiv.py      All (11,0)
```

UDACITY – Software Testing course

"GCC: 9000 assertions,
LLVM: 13,000 assertions [..]
1 assertion per 110 loc"

Program Invariants

"invariants are asserted properties, such as relations among variables, at certain locations in a program"

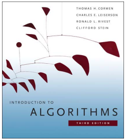

```
assert(x == 2*y);
assert(0 <= idx < |arr|);
```

Program Invariants

"invariants are asserted properties, such as relations among variables, at certain locations in a program"


```
assert(x == 2*y);
assert(0 <= idx < |arr|);
```


```
int getDateOfMonth(int m){
   /*pre: 1 <= m <= 12*/
   ..

   /*post: 0 <= result <= 31*/
}
```



"a loop invariant is a condition that is true on entry into a loop and is guaranteed to remain true on every iteration of the loop [..]"

Approaches to Finding Invariants

Approaches to Finding 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(r ≥ 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  [L]
  return q;
}
```

Static Analysis

- Analyze source code directly
- Pros: results guaranteed on any inputs
- Cons: computationally intensive, produce simple invariants

Approaches to Finding 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(r ≥ 2*b){
      a = 2*a; b = 2*b;
    }
    r=r-b; q=q+a;
  }
  [L]
  return q;
}
```

x	y	q	r
0	1	0	0
1	1	1	0
3	4	0	3
8	1	8	0
15	5	3	0
20	2	10	0
100	1	100	0
	⋮	⋮	

Static Analysis

- Analyze source code directly
- Pros: results guaranteed on any inputs
- Cons: computationally intensive, produce simple invariants

Dynamic Analysis

- Run program and analyze execution traces
- Pros: fast, source code not required
- Cons: results depend on traces, might not hold for all runs

Numerical Invariants

- Relations over numerical variables
  - $x = 3.5$
  - $x = 2y$
  - $x = qy + r$
  - $x^2 \geq y + z^3$
  - $|\texttt{arr}| \geq \texttt{idx} \geq 0, \ \ldots$
- Nonlinear polynomials: required in scientific and engineering applications, implemented in Astrée analyzer for Airbus systems

Numerical Invs: understanding programs

```
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?

Numerical Invs: understanding programs

```
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?

- loop invariants at L1:

$$\begin{array}{ll} x = qy + r & b = ya \\ y \leq b & b \leq r \\ r \leq x & a \leq b \\ 2 \leq a + y & \end{array}$$

- postconditions at L2:

$$\begin{array}{ll} x = qy + r & \\ 1 \leq q + r & r \leq y - 1 \\ 0 \leq r & r \leq x \end{array}$$

Describe the semantic the program (e.g., $x = qy + r$ for integer division) and reveal useful information (e.g., remainder r is non-negative)

Numerical Invariants: analyze program complexities

```
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

Numerical Invariants: analyze program complexities

```
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: $t = O(N + NM + P)$
- Both are nonlinear invariants

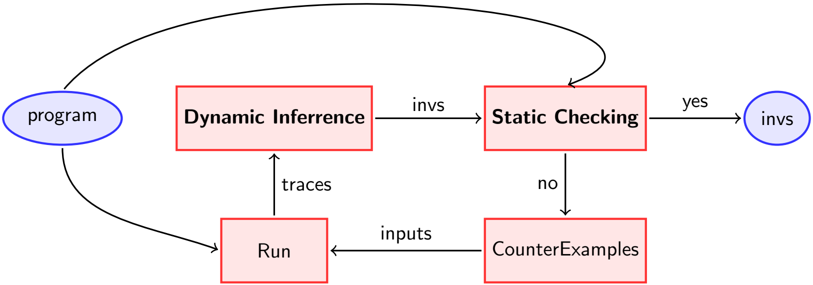
Numerical Invs: verify programs

```
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]
  assert(a + c == b + d);
}
```

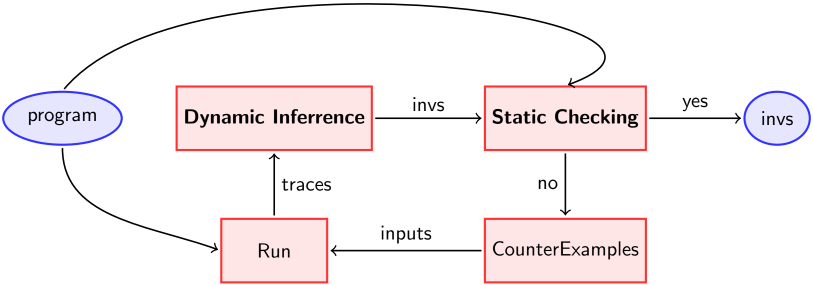
```
void g(int n, int u1) {
  assert(u1 > 0);
  int x = 0;
  int m = 0;

  while (x < n) {
    if (u1) {
      m = x;
    }
    x = x + 1;
  }
  [L]
  if (n > 0){
    assert(0 <= m && m < n);
  }
}
```

Assertions hold if matched or implied by discovered invariants at L

DIG: Dynamic Invariant Generation



DIG: Dynamic Invariant Generation



Goal: developing efficient methods to capture precise and correct program numerical invariants

- Efficient: reformulate and solve using techniques such as equation solving and polyhedral construction
- Precise: employ expressive templates and infer invariants *directly* from traces
- Sound: use static analysis to verify results

Polynomial Relations

DIG discovers polynomial relations of the forms

Equalities $c_0 + c_1x_1 + c_2x_n + c_3x_1x_2 + \cdots + c_mx_1^{d_1} \ldots x_n^{d_n} = 0$

Inequalities $c_0 + c_1x_1 + c_2x_n + c_3x_1x_2 + \cdots + c_mx_1^{d_1} \ldots x_n^{d_n} \geq 0, \quad c_i \in \mathbb{R}$

Examples

cubic $z - 6n = 6,\ \frac{1}{12}z^2 - y - \frac{1}{2}z = -1$

extended gcd $\gcd(a, b) = ia + jb$

sqrt $x + \varepsilon \geq y^2 \geq x - \varepsilon$

Polynomial Relations

DIG discovers polynomial relations of the forms

Equalities $c_0 + c_1x_1 + c_2x_n + c_3x_1x_2 + \cdots + c_mx_1^{d_1}\ldots x_n^{d_n} = 0$

Inequalities $c_0 + c_1x_1 + c_2x_n + c_3x_1x_2 + \cdots + c_mx_1^{d_1}\ldots x_n^{d_n} \geq 0, \quad c_i \in \mathbb{R}$

Examples

cubic $z - 6n = 6,\ \frac{1}{12}z^2 - y - \frac{1}{2}z = -1$

extended gcd $\gcd(a, b) = ia + jb$

sqrt $x + \varepsilon \geq y^2 \geq x - \varepsilon$

Method

- Equalities: solve equations
- Inequalities: construct polyhedra

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

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

- 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

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

- 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

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

Infer Nonlinear Equations using Equation Solver

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

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

- 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

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

- Solve for coefficients c_i

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

Geometric Invariant Inference

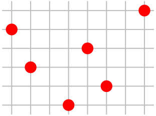
- Treat trace values as points in multi-dimensional space
- Build a convex hull (polyhedron) over the points
- Representation of a polyhedron: a set (conjunction) of inequalities

Geometric Invariant Inference

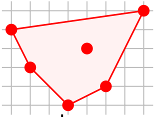
- Treat trace values as points in multi-dimensional space
- Build a convex hull (polyhedron) over the points
- Representation of a polyhedron: a set (conjunction) of inequalities

x	y
-2	1
-1	-1
1	-3
2	0
3	-2
5	2

program traces



trace pts
in 2D



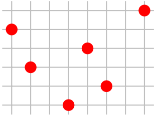
polygon
$c_1 x + c_2 y \geq c$

Geometric Invariant Inference

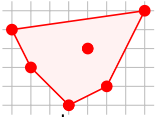
- Treat trace values as points in multi-dimensional space
- Build a convex hull (polyhedron) over the points
- Representation of a polyhedron: a set (conjunction) of inequalities

x	y
-2	1
-1	-1
1	-3
2	0
3	-2
5	2

program traces

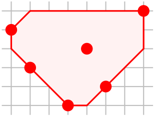


trace pts in 2D

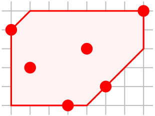


polygon $c_1x + c_2y \geq c$

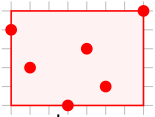
- Support simpler shapes (decreasing precision, increasing efficiency)



octagon $\pm x \pm y \geq c$



zone $x - y \geq c$



box $\pm x, y \geq c$

Spurious Invariants

File Edit Options Buffers Tools Help

```python
def intdiv(x, y):

    q = 0
    r = x
    while r >= y:
        a = 1
        b = y

        while r >= 2*b:
            a = 2 * a
            b = 2 * b
        r = r - b
        q = q + a

    print "x %d, y %d, q %d, r %d" %(x,y,q,r)
    return q,r
```

-U:--- **intdiv.py** All (18,0) (Python)--5:

```
x   0, y   1, q   0, r 0
x   1, y   1, q   1, r 0
x   1, y   5, q   0, r 1
x   1, y  10, q   0, r 1
x   3, y   1, q   3, r 0
x   3, y   4, q   0, r 3
x   3, y   7, q   0, r 3
x   8, y   1, q   8, r 0
x   8, y   2, q   4, r 0
x   8, y   9, q   0, r 8
x   8, y  10, q   0, r 8
x  15, y   1, q  15, r 0
x  15, y   5, q   3, r 0
x  15, y   7, q   2, r 1
x  20, y   2, q  10, r 0
x  20, y   7, q   2, r 6
x  20, y  10, q   2, r 0
x 100, y   1, q 100, r 0
x 100, y   5, q  20, r 0
x 100, y  10, q  10, r 0
```

-U:--- **intdiv.traces** All (21,0)

Valid results

- x, y, q, r are integers
- $r \geq 0$
- $x = q * y + r$

$\vdots$

Spurious Invariants

File Edit Options Buffers Tools Help

```python
def intdiv(x, y):

    q = 0
    r = x
    while r >= y:
        a = 1
        b = y

        while r >= 2*b:
            a = 2 * a
            b = 2 * b
        r = r - b
        q = q + a

    print "x %d, y %d, q %d, r %d" %(x,y,q,r)
    return q,r
```

-U:--- intdiv.py All (18,0) (Python)--5:

```
x   0, y  1, q   0, r 0
x   1, y  1, q   1, r 0
x   1, y  5, q   0, r 1
x   1, y 10, q   0, r 1
x   3, y  1, q   3, r 0
x   3, y  4, q   0, r 3
x   3, y  7, q   0, r 3
x   8, y  1, q   8, r 0
x   8, y  2, q   4, r 0
x   8, y  9, q   0, r 8
x   8, y 10, q   0, r 8
x  15, y  1, q  15, r 0
x  15, y  5, q   3, r 0
x  15, y  7, q   2, r 1
x  20, y  2, q  10, r 0
x  20, y  7, q   2, r 6
x  20, y 10, q   2, r 0
x 100, y  1, q 100, r 0
x 100, y  5, q  20, r 0
x 100, y 10, q  10, r 0
```

-U:--- intdiv.traces All (21,0)

Valid results

- x, y, q, r are integers
- $r \geq 0$
- $x = q * y + r$

$\vdots$

Spurious results

- $100 \geq x \geq 0$
- $10 \geq y \geq 1$
- $100 \geq q - r \geq -8$

$\vdots$

Static Checking

- **Goal**: prove/refute candidate invariants using program code
- **Approach**: reduce invariant checking to reachability

Static Checking

- **Goal**: prove/refute candidate invariants using program code
- **Approach**: reduce invariant checking to reachability
  - Transform program and invariant into another program consist of a special location L'

```
...
[L] //is x=qy+r valid?
...
```

$\Longrightarrow$

```
...
if (!(x==qy+r)){
  [L'] //x=qy+r is invalid
  abort();
}
//x=qy+r is valid
...
```

Static Checking

- **Goal**: prove/refute candidate invariants using program code
- **Approach**: reduce invariant checking to reachability
  - Transform program and invariant into another program consist of a special location L'

```
...
[L] //is x=qy+r valid?
...
```

$\Longrightarrow$

```
...
if (!(x==qy+r)){
  [L'] //x=qy+r is invalid
  abort();
}
//x=qy+r is valid
...
```

  - L' reachable $\Longrightarrow$ inv is spurious (inputs reaching L' represent cex's)
  - L' not reachable (within a time bound) $\Longrightarrow$ DIG *accepts* the invariant

Static Checking

- **Goal**: prove/refute candidate invariants using program code
- **Approach**: reduce invariant checking to reachability
  - Transform program and invariant into another program consist of a special location L'

```
...
[L] //is x=qy+r valid?
...
```

$\Longrightarrow$

```
...
if (!(x==qy+r)){
  [L'] //x=qy+r is invalid
  abort();
}
//x=qy+r is valid
...
```

  - L' reachable $\Longrightarrow$ inv is spurious (inputs reaching L' represent cex's)
  - L' not reachable (within a time bound) $\Longrightarrow$ DIG *accepts* the invariant
- Use static analysis (symbolic execution) to check reachability
  - Incomplete, can timeout, but in practice is *very effective* in refuting bad invariants and finding cex's
  - Can use other verifiers or test-input generation techniques instead

Evaluation

Setup

- DIG is implemented in SAGE/Python (with Z3 backend solver)
- Test machine: 10-core 2.4GHZ CPU, 128GB Ram, Linux OS

Benchmark

- Program Understanding: NLA testsuite, 27 programs with nonlinear invariants
- Complexity Analysis: 19 programs collected from static complexity analysis work
- 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?

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?

- loop invariants at L1:

$$\begin{array}{ll} x = qy + r & b = ya \\ y \leq b & b \leq r \\ r \leq x & a \leq b \\ 2 \leq a + y & \end{array}$$

- postconditions at L2:

$$\begin{array}{ll} x = qy + r & \\ 1 \leq q + r & r \leq y - 1 \\ 0 \leq r & r \leq x \end{array}$$

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?

- loop invariants at L1:

$$
\begin{array}{ll}
x = qy + r & b = ya \\
y \leq b & b \leq r \\
r \leq x & a \leq b \\
2 \leq a + y &
\end{array}
$$

- postconditions at L2:

$$
\begin{array}{ll}
x = qy + r & \\
1 \leq q + r & r \leq y - 1 \\
0 \leq r & r \leq x
\end{array}
$$

Indicate the exact semantic of integer division and reveal other useful correctness information (e.g., remainder is non-negative)

Results: Program Understanding

Prog	Invs	Time (s)	Correct
cohendiv	11	24.5	✓
divbin	12	116.8	✓
manna	5	30.8	✓
hard	13	71.4	✓
sqrt1	5	19.3	✓
dijkstra	14	89.3	✓
freire1	-	-	-
freire2	-	-	-
cohencu	5	22.5	✓
egcd1	9	284.5	✓
egcd2	-	-	-
egcd3	-	-	-
prodbin	7	45.1	✓
prod4br	11	87.3	✓
knuth	9	84.6	✓
fermat1	26	185.3	✓
fermat2	8	101.8	✓
lcm1	22	175.2	✓
lcm2	7	163.8	✓
geo1	7	24.4	✓
geo2	9	24.3	✓
geo3	7	32.3	✓
ps2	3	17.0	✓
ps3	4	17.8	✓
ps4	4	18.5	✓
ps5	4	19.3	✓
ps6	3	21.0	✓

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	Invs	Time (s)	Correct
cohendiv	11	24.5	✓
divbin	12	116.8	✓
manna	5	30.8	✓
hard	13	71.4	✓
sqrt1	5	19.3	✓
dijkstra	14	89.3	✓
freire1	-	-	-
freire2	-	-	-
cohencu	5	22.5	✓
egcd1	9	284.5	✓
egcd2	-	-	-
egcd3	-	-	-
prodbin	7	45.1	✓
prod4br	11	87.3	✓
knuth	9	84.6	✓
fermat1	26	185.3	✓
fermat2	8	101.8	✓
lcm1	22	175.2	✓
lcm2	7	163.8	✓
geo1	7	24.4	✓
geo2	9	24.3	✓
geo3	7	32.3	✓
ps2	3	17.0	✓
ps3	4	17.8	✓
ps4	4	18.5	✓
ps5	4	19.3	✓
ps6	3	21.0	✓

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: DIG found correct invariants in 23/27 progs

- Most results equiv 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?

- Existing result: $t = O(N + NM + P)$

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?

- Existing result: $t = O(N + NM + P)$
- DIG 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?

- Existing result: $t = O(N + NM + P)$
- DIG found a very *unexpected* inv:

$$\begin{aligned} P^2Mt + PM^2t - PMNt - M^2Nt \\ -PMt^2 + MNt^2 + PMt - PNt - 2MNt \\ +Pt^2 + Mt^2 + Nt^2 - t^3 - Nt + t^2 = 0 \end{aligned}$$

- Solve for t yields the most precise, unpublished bound:

$$\begin{aligned} t &= 0 && \text{when} \quad N = 0, \\ t &= P + M + 1 && \text{when} \quad N \leq P, \\ t &= N - M(P - N) && \text{when} \quad N > P \end{aligned}$$

- Nonlinear invariants can represent *disjunctive properties* capturing different complexity bounds

Results: Complexity Analysis

Prog	Invs	Time (s)	
cav09_fig1a	1	14.3	✓
cav09_fig1d	1	14.2	✓
cav09_fig2d	3	36.0	✓
cav09_fig3a	3	14.2	✓
cav09_fig5b	5	46.8	✓
pldi09_ex6	7	54.1	✓
pldi09_fig2 (triple)	6	93.5	✓✓
pldi09_fig4_1	3	44.2	✓
pldi09_fig4_2	5	43.7	✓
pldi09_fig4_3	3	37.5	✓
pldi09_fig4_4	4	56.6	-
pldi09_fig4_5	3	31.6	✓
popl09_fig2_1	2	211.7	✓✓
popl09_fig2_2	2	65.1	✓✓
popl09_fig3_4	4	54.7	✓
popl09_fig4_1	2	42.7	✓
popl09_fig4_2	2	158.3	✓✓
popl09_fig4_3	5	39.2	✓
popl09_fig4_4	3	34.2	✓

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	14.3	✓
cav09_fig1d	1	14.2	✓
cav09_fig2d	3	36.0	✓
cav09_fig3a	3	14.2	✓
cav09_fig5b	5	46.8	✓
pldi09_ex6	7	54.1	✓
pldi09_fig2 (triple)	6	93.5	✓✓
pldi09_fig4_1	3	44.2	✓
pldi09_fig4_2	5	43.7	✓
pldi09_fig4_3	3	37.5	✓
pldi09_fig4_4	4	56.6	-
pldi09_fig4_5	3	31.6	✓
popl09_fig2_1	2	211.7	✓✓
popl09_fig2_2	2	65.1	✓✓
popl09_fig3_4	4	54.7	✓
popl09_fig4_1	2	42.7	✓
popl09_fig4_2	2	158.3	✓✓
popl09_fig4_3	5	39.2	✓
popl09_fig4_4	3	34.2	✓

Experiment

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

Results: Obtain equiv (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]
  assert(a + c == b + d);
}
```

L: $b + 1 = c, a + 1 = d, a + b \leq 2, 2 \leq a$

```
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]
  if (n > 0){
    assert(0 <= m && m < n);
  }
}
```

L: $m^2 = nx - m - x, mn = x^2 - x - m \leq x, x \leq m + 1, n \leq x$

Results: Verification

Experiment

- HOLA benchmark: 49 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: 49 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 (23) or stronger (13) invariants in 36/46 programs
- Time: mean 30s, median 13s
- Nonlinear invariants can prove many nontrivial and *unsupported* properties

Conclusion

DIG: integrate dynamic and static analyses for *numerical* invariant generation

- Dynamic Inference: compute nonlinear invariants from execution traces
- Static Checking: check candidate invariants and obtain counterexamples

Results

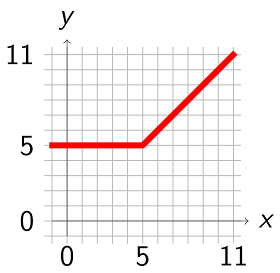
- Discover necessary nonlinear invariants to understand programs
- Find useful invariants capturing nontrivial runtime complexity
- Compete well with existing work
- General polynomial invariants (e.g., nonlinear properties) can *surprisingly* represent and prov nontrivial and complex program properties

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

Analyzing Disjunctive Invariants using Max-Plus Algebra

```
def ex(x):
  y = 5
  if x > y: x = y
  while[L] x ≤ 10:
    if x ≥ 5:
      y = y + 1
    x = x + 1

  assert y ≡ 11
```

x	y
-1	5
⋮	⋮
5	5
6	6
⋮	⋮
11	11



$$L : (x < 5 \wedge 5 = y) \ \vee \ (x \geq 5 \wedge x = y), 11 \geq x$$

Disjunction of 2 cases:

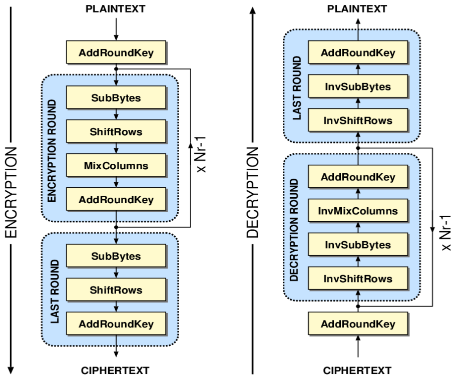
1. if $x < 5$ then $y = 5$
2. if $x \geq 5$ then $x = y$

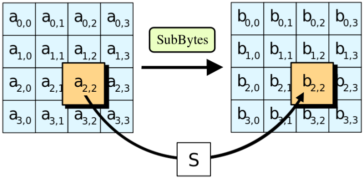
$\longleftrightarrow$ if $0 > x - 5$ then $0 = y - 5$ else $x - 5 = y - 5$

$\max(0, x - 5) = y - 5$

a linear relation .. in max-plus algebra

Array Invariants in AES (Advanced Encryption Standard)






```
def SubBytes(S,a):
  #S is 1D array, a is 2D array
  b =
    [[S[a[0][0]],S[a[0][1]],
      S[a[0][2]],S[a[0][3]]],
     [S[a[1][0]],S[a[1][1]],
      S[a[1][2]],S[a[1][3]]],
     [S[a[2][0]],S[a[2][1]],
      S[a[2][2]],S[a[2][3]]],
     [S[a[3][0]],S[a[3][1]],
      S[a[3][2]],S[a[3][3]]]]

  [L]
  return b
```

[L]: b[i][j] = S[a[i][j]]

Research

Verification

- Finding programs' space and time complexity
- Analyzing programs with pointer data structures
- Understanding programs' configuration spaces

Synthesis

- Automatic program repair
- Generating library axioms for synthesizing object-oriented programs

Acknowledgement

History of DIG's developments & Publications

- DIG: nonlinear equations, inequalities, array relationships (ICSE '12)
  ACM Distinguished Paper award
- geometric invs and complexity analyses for array invariants (TOSEM '13)
- disjunction, max-plus algebra, prove by *k*-induction (ICSE '14)
- DIG2: refute spurious results using symbolic exec, runtime complexity analysis (FSE '17)
- DIG3: use symbolic traces to infer and check invs (ASE '17)

Coauthors (8)

- Deepak Kapur (U. New Mexico), Wes Weimer (UMich), Stephanie Forrest (U. Arizona)
- Timos Antonopoulos (Yale), Andrew Ruef (UMD), Michael Hicks (UMD)
- Matthew Dwyer (UNL), Willem Visser (Stellenbosch U., South Africa)