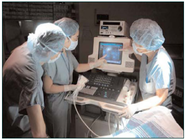
Automating Program Verification and Repair Using Invariant Analysis and Test-input Generation

ThanhVu (Vu) Nguyen

University of New Mexico

June, 2014
Ph.D. Dissertation Defense

The Problem



SOFTWARE BUGS

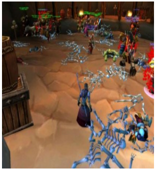


Figure: *

World of Warcraft bug

Figure: *

Therac-25 machines

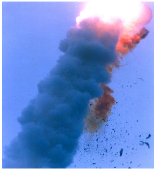


Figure: *

Ariane-5 rocket self-destructs

Figure: *

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

Write Code


Check Code

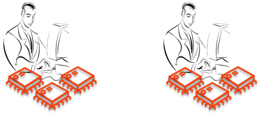

Program Analysis

Write Code

Check Code

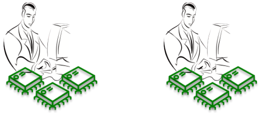
Automated program analysis techniques and tools can decrease debugging time by an average of 26% and $41 billion annually

Program Synthesis



Generates a program that meets a given specification

Program Verification



Checks if a program satisfies a given specification

Invariant Generation and Template-based Synthesis

Invariant Generation

```
def intdiv(x, y):
  q = 0
  r = x
  while r ≥ y:
    a = 1
    b = y
    while [??] r ≥ 2b:
      a = 2a
      b = 2b
    r = r - b
    q = q + a
    [??]
    return q
```

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

Invariant Generation and Template-based Synthesis

Invariant Generation

```
def intdiv(x, y):
  q = 0
  r = x
  while r ≥ y:
    a = 1
    b = y
    while [??] r ≥ 2b:
      a = 2a
      b = 2b
    r = r - b
    q = q + a
    [??]
    return q
```

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

Template-based Synthesis

```
def intdiv(x, y):
  q = 0
  r = x
  while r [**] y:
    a = 1
    b = [**]
    while r ≥ 2b:
      a = [**]
      b = 2b
    r = r - b
    q = q + a

  return [**]
```

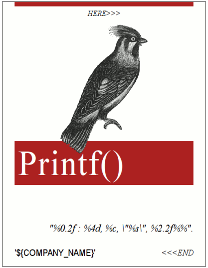
- Creates code under specific templates from partially completed programs
- Can be used for automatic program repair

Thesis and Outline

***Thesis**: "build efficient techniques to automatically generate invariants and programs by encoding these tasks as solutions to existing problem instances in the constraint and verification domains"*

How We Analyze Programs

How We Analyze Programs



File Edit Options Buffers Tools Help

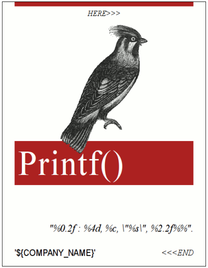
```python
def intdiv(x, y):

    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”

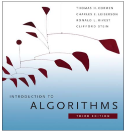

```
assert(x == 2*y);
assert(0 <= idx < |arr|);
```

Program Invariants

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

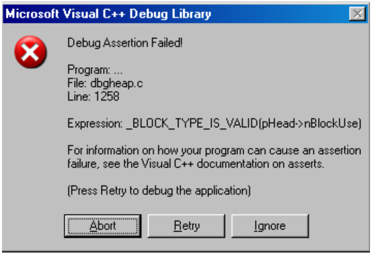

```
assert(x == 2*y);
assert(0 <= idx < |arr|);
```


```
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 [..]"

Uses of Invariants



- Understand and verify programs
- Formal proofs
- Debug (locate errors)
- Documentations

Approaches to Finding Invariants

Approaches to Finding Invariants

```
def intdiv(x,y):
    q = 0
    r = x
    while r ≥ y:
        a = 1
        b = y
        while r ≥ 2b:
            a = 2a
            b = 2b
        r = r - b
        q = q + a
    [L]
    return q,r
```

Static Analysis

- Analyzes source code directly
- Pros: results guaranteed on any input, proofs of correctness or errors
- Cons: computationally intensive, deduce simple invariants

Approaches to Finding Invariants

```
def intdiv(x,y):
    q = 0
    r = x
    while r ≥ y:
        a = 1
        b = y
        while r ≥ 2b:
            a = 2a
            b = 2b
        r = r - b
        q = q + a
    [L]
    return q,r
```

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

- Analyzes source code directly
- Pros: results guaranteed on any input, proofs of correctness or errors
- Cons: computationally intensive, deduce simple invariants

Dynamic Analysis

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

Three Challenging Classes of Invariants

Three Challenging Classes of Invariants

1. Polynomials
   - 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

Three Challenging Classes of Invariants

1. Polynomials
   - 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

2. Disjunctions
   - Represent branching behaviors in programs, e.g., search, sort
     $a \vee b,\ (i = \texttt{even}) \Rightarrow (A[i] = B[i]),\ \texttt{if}\ (a = b)\ \texttt{then}\ (c = 5)\ \texttt{else}\ (c = d + 7)$
   - Many loops in OpenSSH require disjunctive invariants

Three Challenging Classes of Invariants

1. Polynomials
   - 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

2. Disjunctions
   - Represent branching behaviors in programs, e.g., search, sort
     $a \vee b,\ (i = \texttt{even}) \Rightarrow (A[i] = B[i]),\ \texttt{if}\ (a = b)\ \texttt{then}\ (c = 5)\ \texttt{else}\ (c = d + 7)$
   - Many loops in OpenSSH require disjunctive invariants

3. Arrays
   - Relations among (multi-dimensional) array variables
     $A[i] = B[i],\ A[i][j] = B[i] + 3C[i] - 1.2,\ A[i] = B[C[i]][D[2i]],\ r = f(g(x), h(y,z))$
   - Popular data structure to implement strings, vectors, matrices, memory, ..

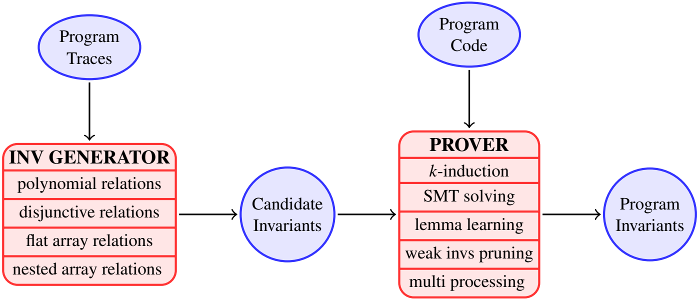
Three Challenging Classes of Invariants

1. Polynomials
   - 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

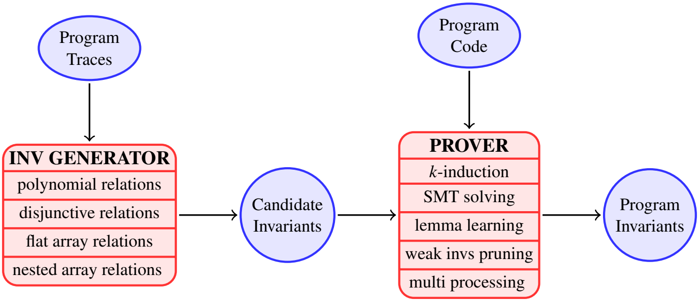
2. Disjunctions
   - Represent branching behaviors in programs, e.g., search, sort
     $a \vee b,\ (i = \texttt{even}) \Rightarrow (A[i] = B[i]),\ \texttt{if}\ (a = b)\ \texttt{then}\ (c = 5)\ \texttt{else}\ (c = d + 7)$
   - Many loops in OpenSSH require disjunctive invariants

3. Arrays
   - Relations among (multi-dimensional) array variables
     $A[i] = B[i],\ A[i][j] = B[i] + 3C[i] - 1.2,\ A[i] = B[C[i]][D[2i]],\ r = f(g(x), h(y,z))$
   - Popular data structure to implement strings, vectors, matrices, memory, ..

DIG: Dynamic Invariant Generation (TOSEM '14, ICSE '14, ICSE '12)



DIG: Dynamic Invariant Generation (TOSEM ’14, ICSE ’14, ICSE ’12)



Goal: developing efficient methods to capture precise and provably correct program 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: integrate theorem proving to formally verify results

Outline

Example: Cohen Integer Division

```
def intdiv(x, y):
    q = 0
    r = x
    while r ≥ y:
        a = 1
        b = y
        while r ≥ 2b:
            [L]
            a = 2a
            b = 2b
        r = r - b
        q = q + a
    return q,r
```

Example: Cohen Integer Division

```
def intdiv(x, y):
    q = 0
    r = x
    while r ≥ y:
        a = 1
        b = y
        while r ≥ 2b:
            [L]
            a = 2a
            b = 2b
        r = r - b
        q = q + a
    return q,r
```

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

Invariants at L: $b = ya,\ x = qy + r,\ r \geq 2ya$

Polynomial Relations (ICSE '12)

DIG discovers polynomial relations of the forms

Equalities $c_0 + c_1x + c_2y + c_3xy + \cdots + c_nx^dy^d = 0$

Inequalities $c_0 + c_1x + c_2y + c_3xy + \cdots + c_nx^dy^d \geq 0, \quad c_i \in \mathbb{R}$

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 (ICSE '12)

DIG discovers polynomial relations of the forms

Equalities $c_0 + c_1x + c_2y + c_3xy + \cdots + c_nx^dy^d = 0$

Inequalities $c_0 + c_1x + c_2y + c_3xy + \cdots + c_nx^dy^d \geq 0, \quad c_i \in \mathbb{R}$

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: solving equations
- Inequalities: constructing polyhedra

Example: Cohen Integer Division

```
def intdiv(x, y):
    q = 0
    r = x
    while r ≥ y:
        a = 1
        b = y
        while r ≥ 2b:
            [L]
            a = 2a
            b = 2b
        r = r - b
        q = q + a
    return q,r
```

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

Invariants at L: $b = ya,\ x = qy + r,\ r \geq 2ya$

Finding 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

Finding 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

Finding Nonlinear Equations using Equation Solver

- Terms and degrees

$$V = \{r, y, a\};\ \texttt{deg} = 2$$

$$\downarrow$$

$$T = \{1, r, y, a, ry, ra, ya, r^2, y^2, a^2\}$$

$$T = \{\ldots, \log(r), a^y, \sin(y), \ldots\}$$

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

Finding 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

- Equation template

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

Finding 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

- 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

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

Finding 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

- 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

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

- Solve for coefficients c_i

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

Geometric Invariant Inference (TOSEM '14)

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

Geometric Invariant Inference (TOSEM '14)

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

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

Figure: *

program traces

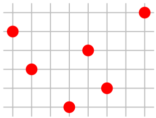


Figure: *

trace pts in 2D

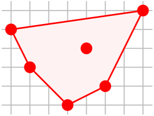


Figure: *

polygon $c_1x + c_2y \geq c$

Geometric Invariant Inference (TOSEM '14)

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

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

Figure: *

program traces

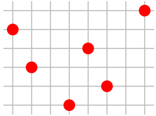


Figure: *

trace pts in 2D

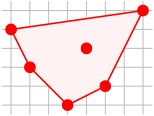


Figure: *

polygon $c_1x + c_2y \geq c$

- Supports simpler shapes (decreasing precision, increasing efficiency)

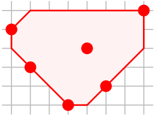


Figure: *

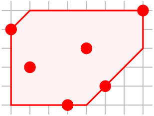


Figure: *

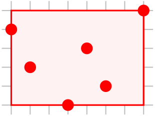


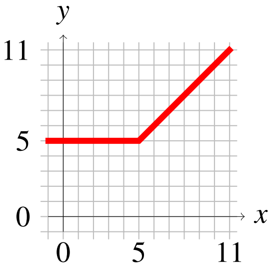
Figure: *

Outline

Example: Disjunctive Invariants

```
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
```

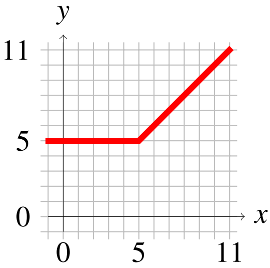
x	y
-1	5
$\vdots$	$\vdots$
5	5
6	6
$\vdots$	$\vdots$
11	11



Example: Disjunctive Invariants

```
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
$\vdots$	$\vdots$
5	5
6	6
$\vdots$	$\vdots$
11	11

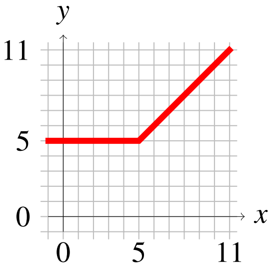


$$L : (x < 5 \wedge 5 = y) \ \vee \ (x \geq 5 \wedge x = y), 11 \geq x$$

Example: Disjunctive Invariants

```
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
$\vdots$	$\vdots$
5	5
6	6
$\vdots$	$\vdots$
11	11



$$L : (x < 5 \land 5 = y) \lor (x \geq 5 \land x = y), 11 \geq x$$

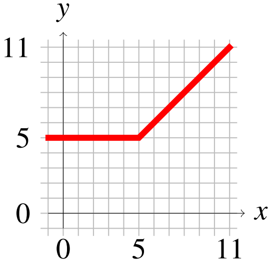
Disjunction of 2 cases:

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

Example: Disjunctive Invariants

```
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
$\vdots$	$\vdots$
5	5
6	6
$\vdots$	$\vdots$
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 $x = y$

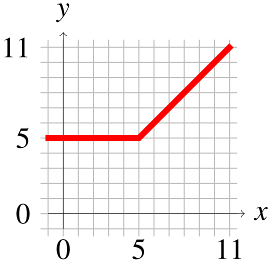
$\longleftrightarrow$ if $0 > x - 5$ then $0 = y - 5$ else $x - 5 = y - 5$

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

Example: Disjunctive Invariants

```
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 $x - 5 = y - 5$

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

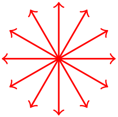
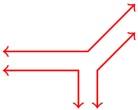
a linear relation .. in max-plus algebra

Max-plus Algebra

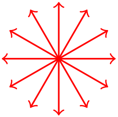
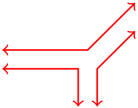
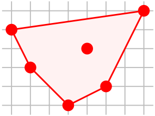
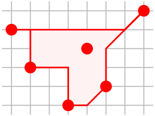
Max-plus Algebra

	Linear	**Max-plus**
Domain	$\mathbb{R}$	$\mathbb{R} \cup \{-\infty\}$
Addition	$+$	max
Multiplication	$\times$	$+$
Zero elem	0	$-\infty$
Unit elem	1	0
Relation form	$c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0$	$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq$ $\max(d_0, d_1 + t_1, \ldots, d_n + t_n)$

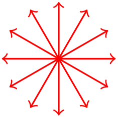
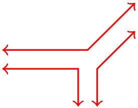
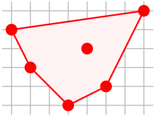
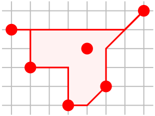
Max-plus Algebra

	Linear	**Max-plus**
Domain	$\mathbb{R}$	$\mathbb{R} \cup \{-\infty\}$
Addition	$+$	max
Multiplication	$\times$	$+$
Zero elem	0	$-\infty$
Unit elem	1	0
Relation form	$c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0$	$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)$
Line shapes		

Max-plus Algebra

	Linear	Max-plus
Domain	$\mathbb{R}$	$\mathbb{R} \cup \{-\infty\}$
Addition	$+$	max
Multiplication	$\times$	$+$
Zero elem	0	$-\infty$
Unit elem	1	0
Relation form	$c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0$	$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)$
Line shapes		
Convex hull		

Max-plus Algebra

	Linear	**Max-plus**
Domain	$\mathbb{R}$	$\mathbb{R} \cup \{-\infty\}$
Addition	$+$	max
Multiplication	$\times$	$+$
Zero elem	0	$-\infty$
Unit elem	1	0
Relation form	$c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0$	$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)$
Line shapes		
Convex hull		

Not convex in classical sense!

Disjunctive Invariants (ICSE '14)

DIG discovers disjunctive relations of the max-plus form

$$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)$$

Examples

$$\begin{array}{rcccc} z = \max(x, y) & \equiv & (x < y \wedge z = x) & \vee & (x \geq y \wedge z = y) \\ \text{strncpy}(s, d, n) & \equiv & (n \geq |s| \wedge |d| = |s|) & \vee & (n < |s| \wedge |d| \geq n) \end{array}$$

Disjunctive Invariants (ICSE '14)

DIG discovers disjunctive relations of the max-plus form

$$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)$$

Examples

$$\begin{array}{rclcl} z = \max(x, y) & \equiv & (x < y \land z = x) & \lor & (x \geq y \land z = y) \\ \mathrm{strncpy}(s, d, n) & \equiv & (n \geq |s| \land |d| = |s|) & \lor & (n < |s| \land |d| \geq n) \end{array}$$

Method

- Uses terms to express variables
- Builds a max-plus convex polyhedron and extract facets
- Introduces simpler max-plus shapes for lower computational complexity

Outline

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

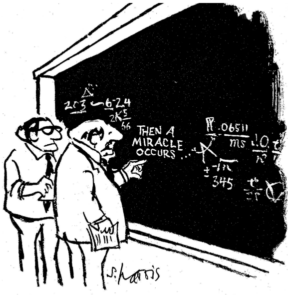
$\vdots$

Spurious results

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

$\vdots$

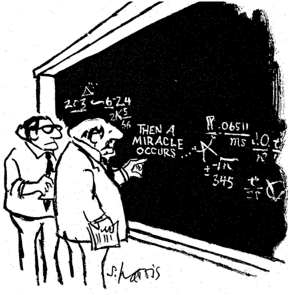
KIP: k-Induction Prover (ICSE ’14)



KIP, an automatic theorem prover, for verifying invariants

1. Implements *k*-induction
2. Employs powerful constraint solving
3. Learns new lemmas
4. Uses multi-processing
5. Identifies strongest invariants

KIP: k-Induction Prover (ICSE ’14)



KIP, an automatic theorem prover, for verifying invariants

1. Implements k-induction
2. Employs powerful constraint solving
3. Learns new lemmas
4. Uses multi-processing
5. Identifies strongest invariants

DIG + KIP ≡ hybridization of dynamic and static analysis

An efficient and sound technique to generate complex program invariants

Experimental Results

Benchmarks

- Nonlinear test suite: 27 programs require nonlinear invariants
- Disjunctive testsuite: 14 programs require disjunctive invariants

Setup

- Implemented in SAGE/Python (with Z3 backend solver)
- Test machine: 64-core 2.6GHZ CPU, 128GB RAM, Linux OS
- Invariants obtained at loop entrances and program exits

Experimental Results

Benchmarks

- Nonlinear test suite: 27 programs require nonlinear invariants
- Disjunctive testsuite: 14 programs require disjunctive invariants

Setup

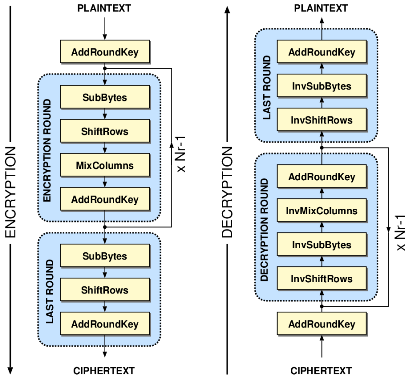
- Implemented in SAGE/Python (with Z3 backend solver)
- Test machine: 64-core 2.6GHZ CPU, 128GB RAM, Linux OS
- Invariants obtained at loop entrances and program exits

Results

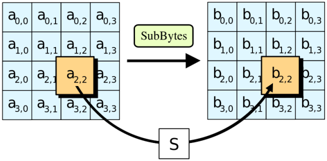
- All generated equalities are valid and most inequalities are spurious (and removed by KIP)
- Invariants generated are sufficiently strong to explain program behavior
  - Proved correctness of 36/41 programs, 2 mins per program
  - No spurious results
- Current dynamic analysis cannot find any of these invariants

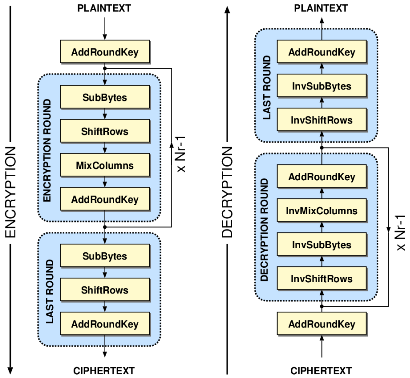
Outline

A Case Study: Advanced Encryption Standard (AES)



A Case Study: Advanced Encryption Standard (AES)






```
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]]

Nested Array Problem (TOSEM '14, ICSE '12)

The Array Nesting (AN) problem

Given an n-dimensional array a and set B of single dimensional arrays, does there exist a *nesting* $(b_1, \ldots, b_l)$ from B such that

$$a[i_1] \ldots [i_n] = b_1[\ldots[b_l[c_0 + c_1 i_1 + \cdots + c_n i_n]] \ldots] \,?$$

Example: $a[i] = b_1[i + 3j + 5],\ a[i][j][k] = b_1[b_2[i + 2j + 3k]]$

Nested Array Problem (TOSEM '14, ICSE '12)

The Array Nesting (AN) problem

Given an n-dimensional array a and set B of single dimensional arrays, does there exist a *nesting* $(b_1, \ldots, b_l)$ from B such that

$$a[i_1]\ldots[i_n] = b_1[\ldots[b_l[c_0 + c_1 i_1 + \cdots + c_n i_n]]\ldots]\ ?$$

Example: $a[i] = b_1[i + 3j + 5],\ a[i][j][k] = b_1[b_2[i + 2j + 3k]]$

Complexity (**m**: number of array variables, **n**: size of the largest array variable)

- AN is strongly NP-Complete in **m** (reduction from *Exact Covering*)
- AN can be solved in polynomial time in **n** using reachability analysis
- Same complexity for the generalized version with multi-dimensional and repeating arrays, e.g., $\mathsf{a}[i][j] = 2\mathsf{b}_1[\mathsf{b}_2[2i+3]][[\mathsf{b}_3[i][b_2[j]]]]$

Experimental Results

Nested Array Relations (functions are treated as a special type of arrays)

xor2Word	$R[i] = \mathsf{xor}(A[i], B[i]),\ A[i] = \mathsf{xor}(R[i], B[i]), B[i] = \mathsf{xor}(R[i], A[i])$
addRoundKey	$R[i][j] = \mathsf{xor}(T[i][j], H[i][j])$
multWord	$R[i] = T[\mathsf{mod}(L[A[i]] + L[B[i]], 255)]$

Experimental Results

Nested Array Relations (functions are treated as a special type of arrays)

xor2Word	$R[i] = \mathsf{xor}(A[i], B[i]),\ A[i] = \mathsf{xor}(R[i], B[i]), B[i] = \mathsf{xor}(R[i], A[i])$
addRoundKey	$R[i][j] = \mathsf{xor}(T[i][j], H[i][j])$
multWord	$R[i] = T[\mathsf{mod}(L[A[i]] + L[B[i]], 255)]$

Flat Array Relations (flattening array elements and solving equations)

block2State	$R[i][j] = T[4i + j]$
RotWord	$R = [W[1], W[2], W[3], W[0]]$
keySetupEnc8	$R[i][j] = \mathsf{cipherKey}[4i + j]$ for $i = 0, \ldots, 7;\ j = 0, \ldots, 3$

Experimental Results

Nested Array Relations (functions are treated as a special type of arrays)

xor2Word	$R[i] = \mathsf{xor}(A[i], B[i]),\ A[i] = \mathsf{xor}(R[i], B[i]), B[i] = \mathsf{xor}(R[i], A[i])$
addRoundKey	$R[i][j] = \mathsf{xor}(T[i][j], H[i][j])$
multWord	$R[i] = T[\mathsf{mod}(L[A[i]] + L[B[i]], 255)]$

Flat Array Relations (flattening array elements and solving equations)

block2State	$R[i][j] = T[4i + j]$
RotWord	$R = [W[1], W[2], W[3], W[0]]$
keySetupEnc8	$R[i][j] = \mathsf{cipherKey}[4i + j]$ for $i = 0, \ldots, 7;\ j = 0, \ldots, 3$

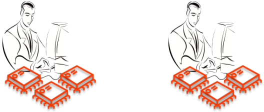
DIG found 60% of the documented array relations in AES under 15 minutes

Outline

From Verification to Synthesis

Program Verification

Checks if a program satisfies a given specification



Significant research development, e.g., formal methods, software testing

From Verification to Synthesis

Program Verification

Checks if a program satisfies a given specification

Significant research development, e.g., formal methods, software testing

Program Synthesis

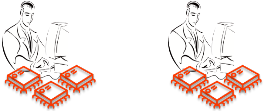
Creates a program that meets a given specification

Less work, *“among the last tasks that computers will do well”*

From Verification to Synthesis

Program Verification

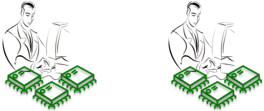
Checks if a program satisfies a given specification



Significant research development, e.g., formal methods, software testing

Program Synthesis

Creates a program that meets a given specification



Less work, *"among the last tasks that computers will do well"*

Goal: connecting verification to synthesis to leverage existing verification techniques and tools to synthesize programs

Program Reachability and Template-based Synthesis

Verification as a Reachability problem

- Shows a program state violating a given specification is not reachable
- Test-input generation: finds inputs that reach a program location

```
def P(x, y):
 if 2 × x == y:
  if x > y + 10:
     [L] #reachable, e.g., x = -20, y = -40

  return 0
```

Program Reachability and Template-based Synthesis

Verification as a Reachability problem

- Shows a program state violating a given specification is not reachable
- Test-input generation: finds inputs that reach a program location

```
def P(x, y):
 if 2 × x == y:
  if x > y + 10:
     [L] #reachable, e.g., x = -20, y = -40

  return 0
```

Template-based Synthesis

- A practical form of synthesis that creates code under specific templates

```
def Q(i, u, d):
  if i:
    b = c0 + c1 × u + c2 × d # linear exp template
  else:
    b = u
  if (b > d): r = 1
  else: r = 0
  return r
```

Test suite

$$\begin{aligned}
Q(1, 0, 100) &= 0 \\
Q(1, 11, 110) &= 1 \\
Q(0, 100, 50) &= 1 \\
Q(1, -20, 60) &= 1 \\
Q(0, 0, 10) &= 0 \\
Q(0, 0, -10) &= 1
\end{aligned}$$

- Is applicable to automatic program repair: identify suspicious program statements and synthesize repairs for those statements

Goal: *connecting* verification to synthesis to leverage *existing* verification

From Reachability to Synthesis

Theorem: Template-based Synthesis is reducible to Reachability

Given a general instance of synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution

From Reachability to Synthesis

Theorem: Template-based Synthesis is reducible to Reachability

Given a general instance of synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution

```
def Q(i, u, d):
  if i:
    b = c0 × c1 × u + c2 × d #syn template
  else: b = u
  if (b > d): r = 1
  else: r = 0
  return r
```

Test suite

$$
\begin{aligned}
Q(1,0,100) &= 0\\
Q(1,11,110) &= 1\\
Q(0,100,50) &= 1\\
Q(1,-20,60) &= 1\\
Q(0,0,10) &= 0\\
Q(0,0,-10) &= 1
\end{aligned}
$$

Figure: *

Input: a synthesis instance

From Reachability to Synthesis

Theorem: Template-based Synthesis is reducible to Reachability

Given a general instance of synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution

```
def Q(i, u, d):
  if i:
    b = c0 × c1 × u + c2 × d #syn template
  else: b = u
  if (b > d): r = 1
  else: r = 0
  return r
```

Test suite

$$
\begin{aligned}
Q(1,0,100) &= 0\\
Q(1,11,110) &= 1\\
Q(0,100,50) &= 1\\
Q(1,-20,60) &= 1\\
Q(0,0,10) &= 0\\
Q(0,0,-10) &= 1
\end{aligned}
$$

Figure: *

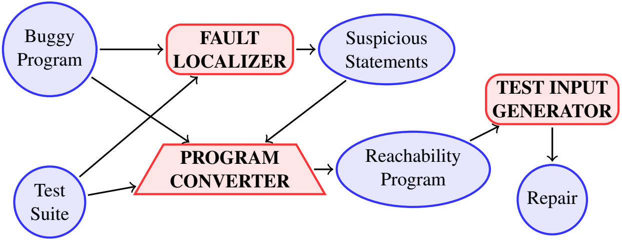
Input: a synthesis instance

```
def pQ(i, u, d, c0, c1, c2):
  if i:
    b = c0 + c1 × u + c2 × d
  else: b = u
  if b > d: r = 1
  else: r = 0
  return r

def pmain(c0, c1, c2):
   e = pQ(1, 0,100, c0,c1,c2) == 0 and
       pQ(1,11,110, c0,c1,c2) == 1 and
       pQ(0,100,50, c0,c1,c2) == 1 and
       pQ(1,-20,60, c0,c1,c2) == 1 and
       pQ(0, 0, 10, c0,c1,c2) == 0 and
       pQ(0, 0,-10, c0,c1,c2) == 1

   if e:
      [L]  #pass the given test suite
   return 0
```

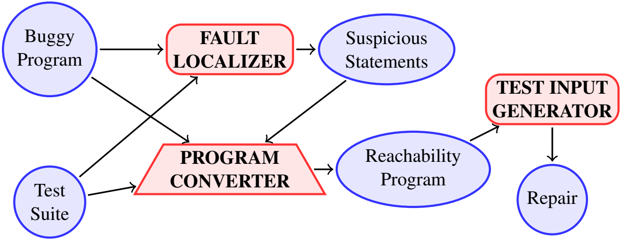
Figure: *

Output: a reachability instance, solvable using a test-input

CETI: Correcting Errors using Test Inputs (FSE ’14, in submission)

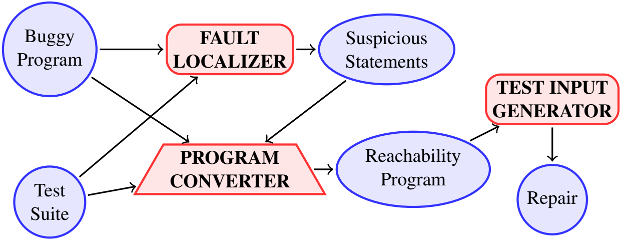


CETI: Correcting Errors using Test Inputs (FSE '14, in submission)



- **CETI**: automatic program repair using test-input generation
  1. Obtain suspicious statements using an existing fault localization tool
  2. Apply synthesis templates to create template-based synthesis instances
  3. Convert to reachabilty programs using reduction theorem
  4. Employ an off-the-shelf test-input generator to solve reachability, i.e., creating repairs

CETI: Correcting Errors using Test Inputs (FSE '14, in submission)



- **CETI**: automatic program repair using test-input generation
  1. Obtain suspicious statements using an existing fault localization tool
  2. Apply synthesis templates to create template-based synthesis instances
  3. Convert to reachabilty programs using reduction theorem
  4. Employ an off-the-shelf test-input generator to solve reachability, i.e., creating repairs
- Outperform other automatic program repair techniques

Equivalence Theorem (FSE '14, in submission)

Synthesis is reducible to Reachability

- *Theorem*: given a general instance of template-based synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution
- *Application*: apply reachability techniques, e.g., test-input generation, to repair programs automatically

Equivalence Theorem (FSE '14, in submission)

Synthesis is reducible to Reachability

- *Theorem*: given a general instance of template-based synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution
- *Application*: apply reachability techniques, e.g., test-input generation, to repair programs automatically

Reachability is reducible to Synthesis

- *Theorem*: given a general instance of reachability, create a specific instance of template-based synthesis, where a successful synthesis indicates the reachability of the target location
- *Application*: apply synthesis techniques, e.g., automated program repair algorithms, to find test-inputs that reach non-trivial program locations

Equivalence Theorem (FSE '14, in submission)

Synthesis is reducible to Reachability

- *Theorem*: given a general instance of template-based synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution
- *Application*: apply reachability techniques, e.g., test-input generation, to repair programs automatically

Reachability is reducible to Synthesis

- *Theorem*: given a general instance of reachability, create a specific instance of template-based synthesis, where a successful synthesis indicates the reachability of the target location
- *Application*: apply synthesis techniques, e.g., automated program repair algorithms, to find test-inputs that reach non-trivial program locations

$$\boxed{\text{Reachability} \equiv \text{Synthesis}}$$

Outline

What's Next ?

Invariant Generation

- Implement algorithms based on polynomial time proof for finding other array properties (objective: 90% of the array invs in AES).
- Extend techniques for generating array relations to analyze other data structures such as trees and lists

Program Synthesis/Repair

- Apply techniques in program synthesis to reachability, e.g., using repair tools to generate high quality test inputs
- Combine CETI and other repair techniques, e.g., random search, to handle a wider range of errors

Conclusion

***Thesis**: "build efficient techniques to automatically generate invariants and programs by encoding these tasks as solutions to existing problem instances in the constraint solving and verification domains"*

Conclusion

Thesis: *"build efficient techniques to automatically generate invariants and programs by encoding these tasks as solutions to existing problem instances in the constraint solving and verification domains"*

- Invariant Generation
  - **DIG**: treat invariants as set of equations and constraints and solve them to identify nonlinear polynomial relations, disjunctive invariants, and array properties
  - **KIP**: an automatic theorem prover for verifying candidate invariants
- Program Synthesis/Repair
  - **Equivalence theorem**: a direct link between program reachability and template-based synthesis
  - **CETI**: apply equivalence theorem to reduce repair task to rechability problem, solvable using off-the-shelf test-input generators
- Source code, benchmarks, etc:

  `http://www.cs.unm.edu/~tnguyen`

BACK UP slides

Daikon (Dynamic Conjecture)

- Ships with a large set of pre-defined templates

Polynomials	$x + 2y - 3z + 4 = 0,\ x = y^2$
Arrays	$\texttt{sorted}(A),\ \texttt{member}(a, A),\ \texttt{reverse}(A, B),\ A = B$

- User-defined: $x = y^2 + 10,\ x = y^3$
- Filters out templates from traces

Daikon (Dynamic Conjecture)

- Ships with a large set of pre-defined templates

Polynomials	$x + 2y - 3z + 4 = 0,\ x = y^2$
Arrays	$\texttt{sorted}(A),\ \texttt{member}(a, A),\ \texttt{reverse}(A, B),\ A = B$

- User-defined: $x = y^2 + 10,\ x = y^3$
- Filters out templates from traces
- Does not find *general* linear or nonlinear relations

  e.g., $b = ya,\ x = qy + r,\ r \geq 2ya$
- Has limited support for relations among arrays and disjunctive invariants

Sample Examples of Polynomials and Disjunctive Invariants

Polynomial Invariants

$$\text{cohencb} : z^2 = 12y + 6z - 12, yz - 18x - 12y + 2z = 6, 6n + 6 = z$$

$$\text{ps6} : y^6 + 3y^5 + \frac{5}{2}y^4 - \frac{1}{2}y^2 = 6x$$

$$\text{cohendv} : b = ya, x = qy + r, r \geq 2ya$$

$$\text{sqrt1} : t = 2a + 1, 4s = t^2 + 2t + 1, s = (a+1)^2, s \geq t$$

$$\text{dijkstra} : h^2p - 4hnq + 4hqr + 4npq - pq^2 = 4pqr \text{ (z3 froze)}$$

Max/Min-plus Invariants

$$\text{ex1} : (x < 5 \wedge y = 5) \ \vee \ (5 \leq x \leq 11 \wedge x = y)$$

$$\text{strncpy} : (n \geq |s| \wedge |d| = |s|) \ \vee \ (n < |s| \wedge |d| \geq n)$$

$$\text{oddeven5} : o_1 = \min(i_1, i_2, i_3, i_4, i_5)$$

$$o_5 = \max(i_1, i_2, i_3, i_4, i_5)$$

Precision Matters

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
```

```
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.py All (18,0) (Python)--5: -U:--- intdiv.traces All (21,0)

Precision Matters

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)

- x, y, q, r are integers

Precision Matters

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:

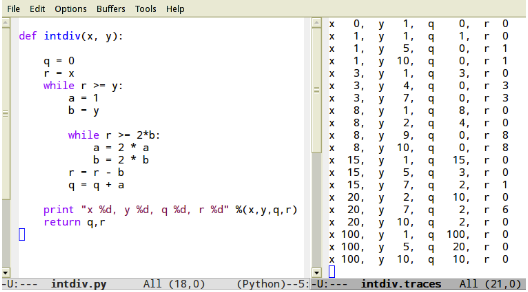
```
x   0, y  1, q   0, r 0
x   1, y  1, q   1, r 0
x   1, y  5, q   0, r 1
x   1, y 10, q   0, r 1
x   3, y  1, q   3, r 0
x   3, y  4, q   0, r 3
x   3, y  7, q   0, r 3
x   8, y  1, q   8, r 0
x   8, y  2, q   4, r 0
x   8, y  9, q   0, r 8
x   8, y 10, q   0, r 8
x  15, y  1, q  15, r 0
x  15, y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

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

$\vdots$

Precision Matters



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
```

```
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.py All (18,0) (Python)--5:-U:--- intdiv.traces All (21,0)

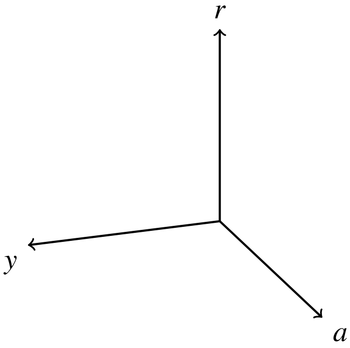
- x, y, q, r are integers
- $r \geq 0$
- $x \geq q$
- $x \geq qy$
- $x = qy + r$

$\vdots$

Finding Nonlinear Inequalities using Polyhedra

```
def intdiv(x, y):
   q = 0
   r = x
   while r ≥ y:
      a = 1
      b = y
      while r ≥ 2b:
         [L:   r ≥ 2ay]
         a = 2a
         b = 2b
      r = r - b
      q = q + a
   return q,r
```

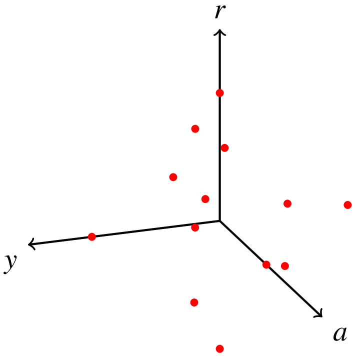
x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4



Finding Nonlinear Inequalities using Polyhedra

```
def intdiv(x, y):
    q = 0
    r = x
    while r ≥ y:
        a = 1
        b = y
        while r ≥ 2b:
            [L:  r ≥ 2ay]
            a = 2a
            b = 2b
        r = r - b
        q = q + a
    return q,r
```

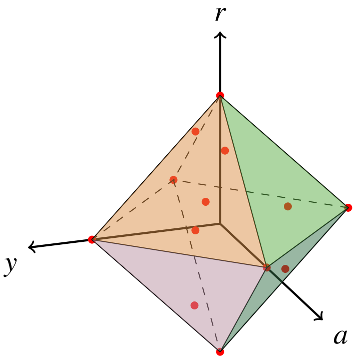
x	y	a	b	q	r
15	2	1	2	0	15
15	2	2	4	0	15
15	2	1	2	4	7
4	1	1	1	0	4
4	1	2	2	0	4



Finding Nonlinear Inequalities using Polyhedra

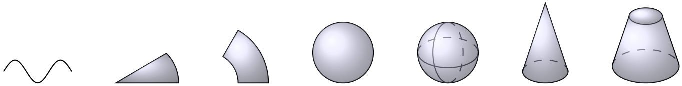
```
def intdiv(x, y):
  q = 0
  r = x
  while r ≥ y:
    a = 1
    b = y
    while r ≥ 2b:
      [L:   r ≥ 2ay]
      a = 2a
      b = 2b
    r = r - b
    q = q + a
  return q,r
```

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



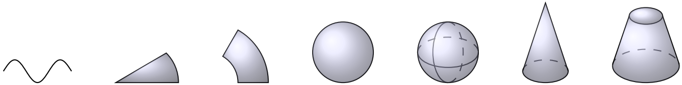
Geometric Invariant Inference (TOSEM '14)

Terms can represent complex shapes: $t_1 = \sin(x), t_2 = 3.1415, \ldots$



Geometric Invariant Inference (TOSEM '14)

Terms can represent complex shapes: $t_1 = \sin(x), t_2 = 3.1415, \ldots$



Simpler geometric shapes, better computational complexity

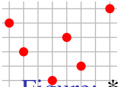


Figure: *

trace pts
in 2D

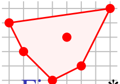


Figure: *

polygon
$c_1x + c_2y \geq c$

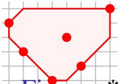


Figure: *

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

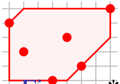


Figure: *

zone
$x - y \geq c$

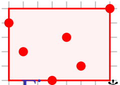
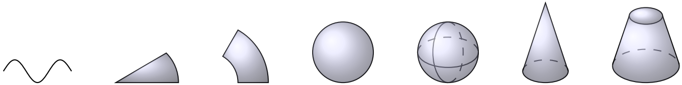


Figure: *

box
$\pm x, y \geq c$

Geometric Invariant Inference (TOSEM '14)

Terms can represent complex shapes: $t_1 = \sin(x), t_2 = 3.1415, \ldots$



Simpler geometric shapes, better computational complexity

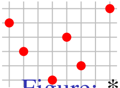


Figure: *

trace pts
in 2D

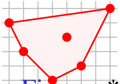


Figure: *

polygon
$c_1x + c_2y \geq c$

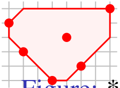


Figure: *

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

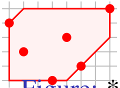


Figure: *

zone
$x - y \geq c$

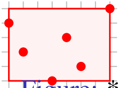
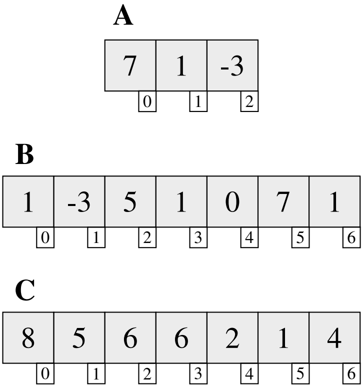


Figure: *

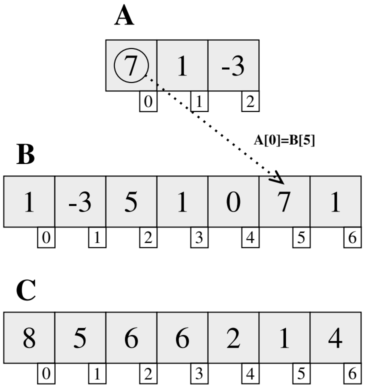
box
$\pm x, y \geq c$

Example: Finding Nested Array Relations



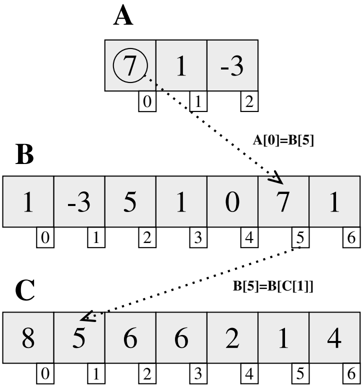
- $A[0] = B[C[?]]$
- $A[1] = B[C[?]]$

Example: Finding Nested Array Relations



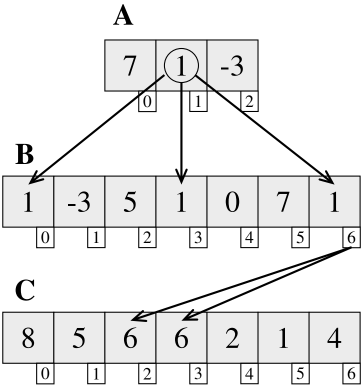
- $A[0] = B[C[?]]$

Example: Finding Nested Array Relations



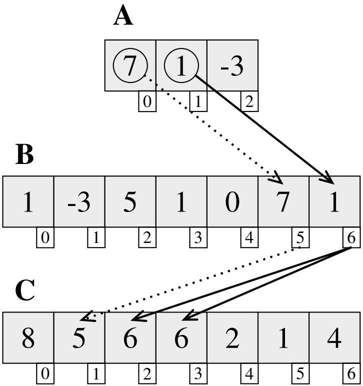
- $A[0] = B[C[1]]$

Example: Finding Nested Array Relations



- $A[1] = B[C[2]] \vee B[C[3]]$

Example: Finding Nested Array Relations

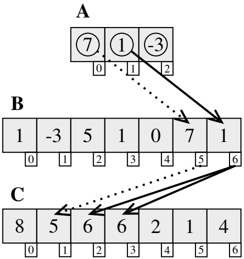


- $A[0] = B[C[1]]$
- $A[1] = B[C[2]] \vee B[C[3]]$

Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

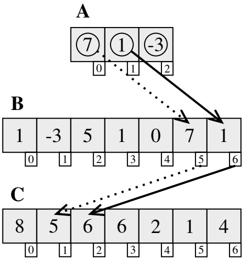
$$
\begin{aligned}
A[0] &= B[C[1]] \\
A[1] &= B[C[2]] \vee B[C[3]]
\end{aligned}
$$



Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

$$A[0] = B[C[1]]$$
$$A[1] = B[C[2]] \vee B[C[3]]$$



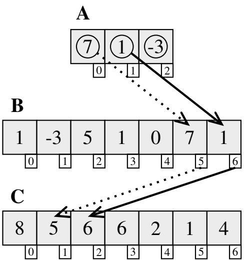
- Express relation between $A[i]$ and $B[C[j]]$ as $j = ip + q$

$$A[0] = B[C[1]],\ A[1] = B[C[2]] \quad \Rightarrow \quad \{1 = 0p + q, 2 = 1p + q\}$$

Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

$$
\begin{aligned}
A[0] &= B[C[1]] \\
A[1] &= B[C[2]] \vee B[C[3]]
\end{aligned}
$$



- Express relation between $A[i]$ and $B[C[j]]$ as $j = ip + q$

$$A[0] = B[C[1]],\ A[1] = B[C[2]] \quad \Rightarrow \quad \{1 = 0p + q, 2 = 1p + q\}$$

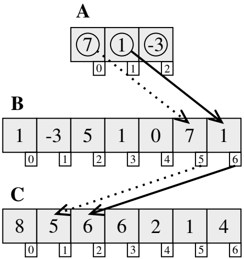
- Solve for p, q

$$\{1 = 0p + q, 2 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 1 \quad \Rightarrow \quad A[i] = B[C[1i + 1]]$$

Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

$$\begin{aligned} A[0] &= B[C[1]] \\ A[1] &= B[C[2]] \vee B[C[3]] \end{aligned}$$



- Express relation between $A[i]$ and $B[C[j]]$ as $j = ip + q$

$$A[0] = B[C[1]],\ A[1] = B[C[2]] \quad \Rightarrow \quad \{1 = 0p + q, 2 = 1p + q\}$$

- Solve for p, q

$$\{1 = 0p + q, 2 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 1 \quad \Rightarrow \quad A[i] = B[C[1i + 1]]$$

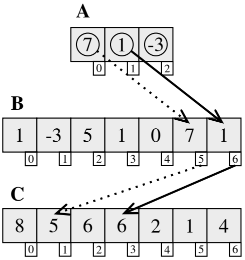
- Verify (obtained candidate invs guarantee to hold for $i = 0, 1$)

$$A[i] = B[C[1i + 1]] \quad \Rightarrow \quad \text{invalid, does not hold when } i = 2 \text{, i.e., } A[2] \neq B[C[3]]$$

Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

$$A[0] = B[C[1]]$$
$$A[1] = B[C[2]] \vee B[C[3]]$$



- Express relation between $A[i]$ and $B[C[j]]$ as $j = ip + q$

$$A[0] = B[C[1]],\ A[1] = B[C[2]] \quad \Rightarrow \quad \{1 = 0p + q, 2 = 1p + q\}$$
$$A[0] = B[C[1]],\ A[1] = B[C[3]] \quad \Rightarrow \quad \{1 = 0p + q, 3 = 1p + q\}$$

- Solve for p, q

$$\{1 = 0p + q, 2 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 1 \quad \Rightarrow \quad A[i] = B[C[1i + 1]]$$

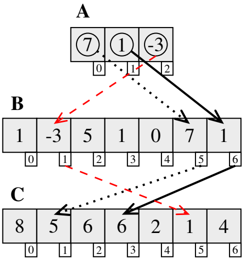
- Verify (obtained candidate invs guarantee to hold for $i = 0, 1$)

$$A[i] = B[C[1i + 1]] \quad \Rightarrow \quad \text{invalid, does not hold when } i = 2\text{, i.e., } A[2] \neq B[C[3]]$$

Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

$$\begin{aligned} A[0] &= B[C[1]] \\ A[1] &= B[C[2]] \vee B[C[3]] \end{aligned}$$



- Express relation between $A[i]$ and $B[C[j]]$ as $j = ip + q$

$$A[0] = B[C[1]],\ A[1] = B[C[2]] \quad \Rightarrow \quad \{1 = 0p + q, 2 = 1p + q\}$$

$$A[0] = B[C[1]],\ A[1] = B[C[3]] \quad \Rightarrow \quad \{1 = 0p + q, 3 = 1p + q\}$$

- Solve for p, q

$$\{1 = 0p + q, 2 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 1 \quad \Rightarrow \quad A[i] = B[C[1i + 1]]$$

$$\{1 = 0p + q, 3 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 2 \quad \Rightarrow \quad A[i] = B[C[2i + 1]]$$

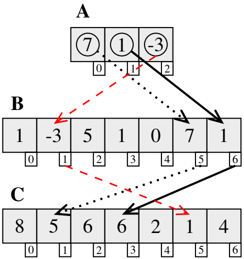
- Verify (obtained candidate invs guarantee to hold for $i = 0, 1$)

$A[i] = B[C[1i + 1]] \quad \Rightarrow \quad$ invalid, does not hold when $i = 2$, i.e., $A[2] \neq B[C[3]]$

Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

$$A[0] = B[C[1]]$$
$$A[1] = B[C[2]] \vee B[C[3]]$$



- Express relation between $A[i]$ and $B[C[j]]$ as $j = ip + q$

$$A[0] = B[C[1]],\ A[1] = B[C[2]] \quad \Rightarrow \quad \{1 = 0p + q, 2 = 1p + q\}$$
$$A[0] = B[C[1]],\ A[1] = B[C[3]] \quad \Rightarrow \quad \{1 = 0p + q, 3 = 1p + q\}$$

- Solve for p, q

$$\{1 = 0p + q, 2 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 1 \quad \Rightarrow \quad A[i] = B[C[1i + 1]]$$
$$\{1 = 0p + q, 3 = 1p + q, 5 = 2p + q\} \quad \Rightarrow \quad q = 1, p = 2 \quad \Rightarrow \quad A[i] = B[C[2i + 1]]$$

- Verify (obtained candidate invs guarantee to hold for $i = 0, 1$)

$A[i] = B[C[1i + 1]] \quad \Rightarrow \quad$ invalid, does not hold when $i = 2$, i.e., $A[2] \neq B[C[3]]$

$A[i] = B[C[2i + 1]] \quad \Rightarrow \quad$ valid, holds when $i = 2$, i.e., $A[2] = B[C[5]]$

Results for Nonlinear Polynomial Invariants

Program	**Locs**	**Var**
intdiv	2	6
divbin	2	5
mannadv	1	5
hard	2	6
sqrt1	1	4
dijkstra	2	5
freire1	1	3
freire2	1	4
cohencb	1	5
egcd1	1	8
egcd2	2	10
egcd3	3	12
lcm1	3	6
lcm2	1	6
prodbin	1	5
prod4br	1	6
fermat1	3	5
fermat2	1	5
knuth	1	8
geo1	1	4
geo2	1	4
geo3	1	5
ps2	1	3
ps3	1	3
ps4	1	3
ps5	1	3
ps6	1	3
total 27		

Results for Nonlinear Polynomial Invariants

Program	**Locs**	**Var**	**Gen**	$\mathbf{T_{Gen}}(s)$	**deg**
intdiv	2	6	152	26.2	2
divbin	2	5	96	37.7	2
mannadv	1	5	49	19.2	2
hard	2	6	107	14.2	2
sqrt1	1	4	27	25.3	2
dijkstra	2	5	61	30.7	3
freire1	1	3	25	22.5	2
freire2	1	4	35	26.0	2
cohencb	1	5	31	23.6	3
egcd1	1	8	108	43.1	2
egcd2	2	10	209	60.8	2
egcd3	3	12	475	67.0	2
lcm1	3	6	203	38.9	2
lcm2	1	6	52	14.9	2
prodbin	1	5	61	28.3	2
prod4br	1	6	42	9.6	3
fermat1	3	5	217	75.7	2
fermat2	1	5	70	25.8	2
knuth	1	8	113	57.1	3
geo1	1	4	25	16.7	2
geo2	1	4	45	24.1	2
geo3	1	5	65	22.1	3
ps2	1	3	25	21.1	2
ps3	1	3	25	21.9	3
ps4	1	3	25	23.5	4
ps5	1	3	24	24.9	5
ps6	1	3	25	25.0	6
total 27			2392	825.9s	

Results for Nonlinear Polynomial Invariants

Program	**Locs**	**Var**	**Gen**	$\mathbf{T_{Gen}}(s)$	**deg**	**Val**	$\mathbf{T_{Val}}(s)$
intdiv	2	6	152	26.2	2	7	8.2
divbin	2	5	96	37.7	2	8	8.7
mannadv	1	5	49	19.2	2	3	5.6
hard	2	6	107	14.2	2	11	9.2
sqrt1	1	4	27	25.3	2	3	4.3
dijkstra	2	5	61	30.7	3	8	10.9
freire1	1	3	25	22.5	2	2	2.2
freire2	1	4	35	26.0	2	3	5.1
cohencb	1	5	31	23.6	3	4	4.2
egcd1	1	8	108	43.1	2	1	12.8
egcd2	2	10	209	60.8	2	8	14.6
egcd3	3	12	475	67.0	2	14	23.4
lcm1	3	6	203	38.9	2	12	14.2
lcm2	1	6	52	14.9	2	1	0.9
prodbin	1	5	61	28.3	2	3	1.1
prod4br	1	6	42	9.6	3	4	8.6
fermat1	3	5	217	75.7	2	6	6.2
fermat2	1	5	70	25.8	2	2	5.2
knuth	1	8	113	57.1	3	4	24.6
geo1	1	4	25	16.7	2	2	1.5
geo2	1	4	45	24.1	2	1	2.1
geo3	1	5	65	22.1	3	1	2.7
ps2	1	3	25	21.1	2	2	4.0
ps3	1	3	25	21.9	3	2	4.2
ps4	1	3	25	23.5	4	2	4.9
ps5	1	3	24	24.9	5	2	7.4
ps6	1	3	25	25.0	6	2	69.5
total 27			2392	825.9s		118	266.3s

Results for Nonlinear Polynomial Invariants

Program	**Locs**	**Var**	**Gen**	**$T_{Gen}(s)$**	**deg**	**Val**	**$T_{Val}(s)$**	**Strength**
intdiv	2	6	152	26.2	2	7	8.2	✓
divbin	2	5	96	37.7	2	8	8.7	–
mannadv	1	5	49	19.2	2	3	5.6	✓
hard	2	6	107	14.2	2	11	9.2	–
sqrt1	1	4	27	25.3	2	3	4.3	✓
dijkstra	2	5	61	30.7	3	8	10.9	–
freire1	1	3	25	22.5	2	2	2.2	✓
freire2	1	4	35	26.0	2	3	5.1	✓
cohencb	1	5	31	23.6	3	4	4.2	✓
egcd1	1	8	108	43.1	2	1	12.8	–
egcd2	2	10	209	60.8	2	8	14.6	✓
egcd3	3	12	475	67.0	2	14	23.4	✓
lcm1	3	6	203	38.9	2	12	14.2	✓
lcm2	1	6	52	14.9	2	1	0.9	✓
prodbin	1	5	61	28.3	2	3	1.1	–
prod4br	1	6	42	9.6	3	4	8.6	✓
fermat1	3	5	217	75.7	2	6	6.2	✓
fermat2	1	5	70	25.8	2	2	5.2	✓
knuth	1	8	113	57.1	3	4	24.6	✓
geo1	1	4	25	16.7	2	2	1.5	✓
geo2	1	4	45	24.1	2	1	2.1	✓
geo3	1	5	65	22.1	3	1	2.7	✓
ps2	1	3	25	21.1	2	2	4.0	✓
ps3	1	3	25	21.9	3	2	4.2	✓
ps4	1	3	25	23.5	4	2	4.9	✓
ps5	1	3	24	24.9	5	2	7.4	✓
ps6	1	3	25	25.0	6	2	69.5	✓
total 27			2392	825.9s		118	266.3s	**22/27**

Results for Disjunctive Invariants

Program	**Locs**	**Var**	**Gen**	$\mathbf{T_{Gen}}(s)$	**Val**	$\mathbf{T_{Val}}(s)$	**Strength**
ex1	1	2	15	0.2	4	1.5	✓
strncpy	1	3	69	1.1	4	7.7	✓
oddeven3	1	6	286	3.7	8	16.0	✓
oddeven4	1	8	867	12.7	22	46.0	✓
oddeven5	1	10	2334	56.8	52	1319.4	✓
bubble3	1	6	249	4.1	8	4.9	✓
bubble4	1	8	832	11.7	22	47.6	✓
bubble5	1	10	2198	53.9	52	938.2	✓
partd3	4	5	479	10.5	10	50.8	✓
partd4	5	6	1217	23.3	15	181.1	✓
partd5	6	7	2943	53.3	21	418.1	✓
parti3	4	5	464	10.3	10	45.5	✓
parti4	5	6	1148	22.4	15	165.1	✓
parti5	6	7	2954	53.6	21	405.6	✓
total 14			16055	317.6s	264	3647.5s	**14/14**

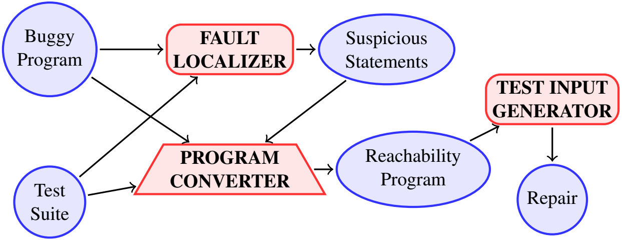
Results for Array Invariants

Function	Desc
multWord	mult
xor2Word	xor
xor3Word	xor
subWord	subs
rotWord	shift
block2State	convert
state2Block	convert
subBytes	subs
invSubByte	subs
shiftRows	shift
invShiftRow	shift
addKey	add
mixCol	mult
invMixCol	mult
keySetEnc4	driver
keySetEnc6	driver
keySetEnc8	driver
keySetEnc	driver
keySetDec	driver
keySched1	driver
keySched2	driver
aesKeyEnc	driver
aesKeyDec	driver
aesEncrypt	driver
aesDecrypt	driver
25 functions	

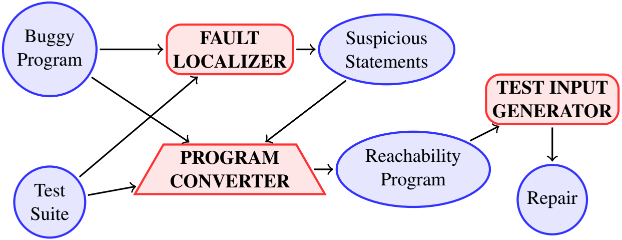
Results for Array Invariants

Function	**Desc**	**Gen**	**V, D**	$\mathbf{T_{Gen}}(s)$
multWord	mult	1 N_4	7, 2	11.0
xor2Word	xor	3 N_1	4, 2	0.8
xor3Word	xor	4 N_1	5, 3	2.0
subWord	subs	2 N_1	3, 1	1.3
rotWord	shift	1 F	2, 1	0.5
block2State	convert	1 F	2, 2	4.1
state2Block	convert	1 F	2, 2	4.2
subBytes	subs	2 N_1	3, 2	3.2
invSubByte	subs	2 N_1	3, 2	3.3
shiftRows	shift	1 F	2, 2	3.7
invShiftRow	shift	1 F	2, 2	3.6
addKey	add	2 N_1	4, 2	3.5
mixCol	mult	0	-	1.0
invMixCol	mult	0	-	1.0
keySetEnc4	driver	1 F	2, 2	76.4
keySetEnc6	driver	1 F	2, 2	78.8
keySetEnc8	driver	1 F	2, 2	79.3
keySetEnc	driver	1 F	2, 1	76.3
keySetDec	driver	0	-	73.0
keySched1	driver	0	-	77.9
keySched2	driver	1 F	2, 2	79.5
aesKeyEnc	driver	1 F, 1 eq	2, 1	76.2
aesKeyDec	driver	1 eq	2, 1	73.6
aesEncrypt	driver	1 F	2, 2	70.5
aesDecrypt	driver	1 F	2, 2	73.8
25 functions		**17/30 invs**		**878.5s**

CETI: Correcting Errors using Test Inputs (FSE ’14, in submission)

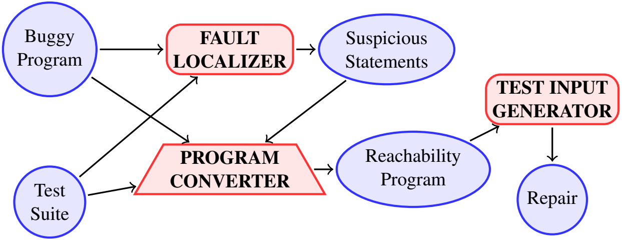


CETI: Correcting Errors using Test Inputs (FSE ’14, in submission)



- **CETI**: automatic program repair using test-input generation
  1. Obtain suspicious statements using an existing fault localization tool
  2. Apply synthesis templates to create template-based synthesis instances
  3. Convert to reachabilty programs using reduction theorem
  4. Employ an off-the-shelf test-input generator to solve reachability, i.e., creating repairs

CETI: Correcting Errors using Test Inputs (FSE '14, in submission)



- **CETI**: automatic program repair using test-input generation
  1. Obtain suspicious statements using an existing fault localization tool
  2. Apply synthesis templates to create template-based synthesis instances
  3. Convert to reachabilty programs using reduction theorem
  4. Employ an off-the-shelf test-input generator to solve reachability, i.e., creating repairs
- Outperform other automatic program repair techniques

Example: Finding Nested Array Relations

- Given

$$A = [7, 1, -3]$$
$$\{B = [1, -3, 5, 1, 0, 7, 1], C = [8, 5, 6, 6, 2, 1, 4]\}$$

Example: Finding Nested Array Relations

- Given

$$A = [7, 1, -3]$$
$$\{B = [1, -3, 5, 1, 0, 7, 1], C = [8, 5, 6, 6, 2, 1, 4]\}$$

- Generate nestings

$$C, B, (C, B), (B, C)$$

Example: Finding Nested Array Relations

- Given

$$A = [7, 1, -3]$$
$$\{B = [1, -3, 5, 1, 0, 7, 1], C = [8, 5, 6, 6, 2, 1, 4]\}$$

- Generate nestings

$$C, B, (C, B), (B, C)$$

- Apply reachability analysis

For nesting (B, C), finds existence of relations
$$A[i] = B[C[ip + q]]$$

Example: Finding Nested Array Relations

- Given

$$A = [7, 1, -3]$$
$$\{B = [1, -3, 5, 1, 0, 7, 1], C = [8, 5, 6, 6, 2, 1, 4]\}$$

- Generate nestings

$$C, B, (C, B), (B, C)$$

- Apply reachability analysis

For nesting (B, C), finds existence of relations
$$A[i] = B[C[ip + q]]$$

- For efficiency: analyze $d + 1$ random indices from the d-dimensional array A

E.g., chooses indices $i = 0, 1$ from the 1-dim array A
Finds existence of relations $A[i] = B[C[ip + q]]$ when $i = 0, 1$

Equivalence Theorem (FSE '14, in submission)

Synthesis is reducible to Reachability

- *Theorem*: given a general instance of template-based synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution
- *Application*: apply reachability techniques, e.g., test-input generation, to synthesize programs automatically

Equivalence Theorem (FSE '14, in submission)

Synthesis is reducible to Reachability

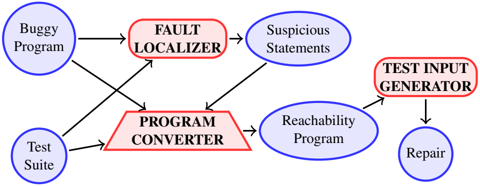
- *Theorem*: given a general instance of template-based synthesis, create a specific instance of reachability consisting of a special location reachable iff synthesis has a solution
- *Application*: apply reachability techniques, e.g., test-input generation, to synthesize programs automatically

Reachability is reducible to Synthesis

- *Theorem*: given a general instance of reachability, create a specific instance of template-based synthesis, where a successful synthesis indicates the reachability of the target location
- *Application*: apply synthesis techniques, e.g., automated program repair algorithms, to find test-inputs that reach non-trivial program locations

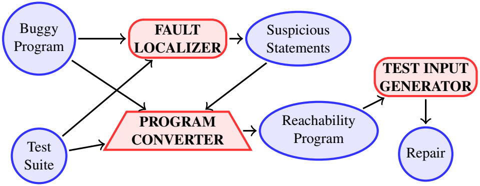
$$\boxed{\text{Reachability} \equiv \text{Synthesis}}$$

CETI: Correcting Errors using Test Inputs



- Repair programs using techniques for test input generation

CETI: Correcting Errors using Test Inputs



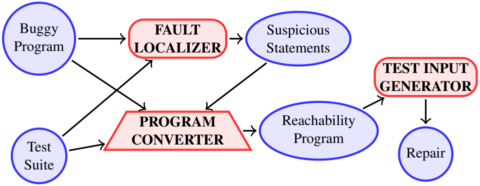
- Repair programs using techniques for test input generation

```
def foo(i, u, d):
  if i:
    b = d #bug b=u+100
  else: b = u
  if b > d: r = 1
  else: r = 0
  return r
```

Test suite

$$
\begin{aligned}
\texttt{foo}(1,0,100) &= 0\\
\texttt{foo}(1,11,110) &= 1\\
\texttt{foo}(0,100,50) &= 1\\
\texttt{foo}(1,-20,60) &= 1\\
\texttt{foo}(0,0,10) &= 0
\end{aligned}
$$

CETI: Correcting Errors using Test Inputs



- Repair programs using techniques for test input generation
- Leverage existing, off-the-shelf test input generation tools

```
def foo(i, u, d):
  if i:
    b = d #bug b=u+100
  else: b = u
  if b > d: r = 1
  else: r = 0
  return r
```

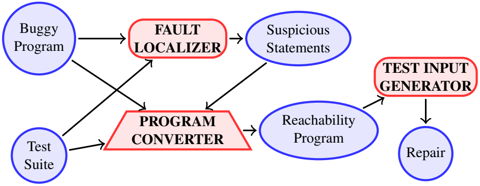
Test suite

foo$(1, 0, 100)$	$=$	0
foo$(1, 11, 110)$	$=$	1
foo$(0, 100, 50)$	$=$	1
foo$(1, -20, 60)$	$=$	1
foo$(0, 0, 10)$	$=$	0

$\rightarrow$

```
def foo2(i, u, d, c1, c2, c3):
  if i:
    b = c1+c2*u+c3*d #synthesize stmt
  else: b = u
  if b > d: r = 1
  else: r = 0
  return r

def foo1():
  if foo2(1, 0,100,c1,c2,c3)==0 and
     foo2(1,11,110,c1,c2,c3)==1 and
     foo2(0,100,50,c1,c2,c3)==1 and
     foo2(1,-20,60,c1,c2,c3)==1 and
     foo2(0, 0, 10,c1,c2,c3)==0 :
     [L] #ci represent the fixes
         #f1=100,f2=1,f3=0 => b=u+100
```

CETI: Correcting Errors using Test Inputs



- Repair programs using techniques for test input generation
- Leverage existing, off-the-shelf test input generation tools
- Outperform other automatic program repair techniques

```
def foo(i, u, d):
  if i:
    b = d #bug b=u+100
  else: b = u
  if b > d: r = 1
  else: r = 0
  return r
```

Test suite

$$
\begin{aligned}
\texttt{foo}(1,0,100) &= 0\\
\texttt{foo}(1,11,110) &= 1\\
\texttt{foo}(0,100,50) &= 1\\
\texttt{foo}(1,-20,60) &= 1\\
\texttt{foo}(0,0,10) &= 0
\end{aligned}
$$

→

```
def foo2(i, u, d, c1, c2, c3):
  if i:
    b = c1+c2*u+c3*d #synthesize stmt
  else: b = u
  if b > d: r = 1
  else: r = 0
  return r

def foo1():
  if foo2(1, 0,100,c1,c2,c3)==0 and
     foo2(1,11,110,c1,c2,c3)==1 and
     foo2(0,100,50,c1,c2,c3)==1 and
     foo2(1,-20,60,c1,c2,c3)==1 and
     foo2(0, 0, 10,c1,c2,c3)==0 :
     [L] #ci represent the fixes
         #f1=100,f2=1,f3=0 => b=u+100
```