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

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
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

Program Verification

Generates a program that meets a given specification

Checks if a program satisfies a given specification
Invariant Generation

```python
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

```python
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

```python
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: “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

GCC: 9,000 assertions,
LLVM: 13,000 assertions

1 assertion per 110 loc
How We Analyze Programs

```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
```

GCC: 9000 assertions, LLVM: 13,000 assertions [..] 1 assertion per 110 loc
How We Analyze Programs

“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”

```c
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”

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

```c
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

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
    return q, r

[...

]
Approaches to Finding Invariants

```python
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, 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

```python
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, r
```

[\[L\]\]

### Static Analysis

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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

1. Polynomials
   Relations over numerical variables
   \[ x = 3.5, x = 2y, x = qy + r, x^2 \geq y + z, |arr| \geq idx \geq 0, \ldots \]

2. Nonlinear polynomials: required in scientific and engineering applications, implemented in Astrée analyzer for Airbus systems

3. Disjunctions
   Represent branching behaviors in programs, e.g., search, sort
   \[ a \lor b, (i = \text{even}) \Rightarrow (A[i] = B[i]), \text{if } (a = b) \text{ then } (c = 5) \text{ else } (c = d + 7) \]

4. Arrays
   Relations among (multi-dimensional) array variables
   \[ A[i] = B[i], A[i][j] = B[i] + 3C[i] - 1, A[i] = B[C[i]][D[2i]], r = f(g(x), h(y, z)) \]

Many loops in OpenSSH require disjunctive invariants

Popular data structure to implement strings, vectors, matrices, memory, ..
Three Challenging Classes of Invariants

1. Polynomials
   - Relations over numerical variables
     \( x = 3.5, \ x = 2y, \ x = qy + r, \ x^2 \geq y + z^3, \ |\text{arr}| \geq \text{idx} \geq 0, \ldots \)
   - 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
     \[ x = 3.5, \ x = 2y, \ x = qy + r, \ x^2 \geq y + z^3, |arr| \geq idx \geq 0, \ldots \]
   - 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
     \[ a \lor b, \ (i = \text{even}) \Rightarrow (A[i] = B[i]), \ \text{if} \ (a = b) \ \text{then} \ (c = 5) \ \text{else} \ (c = d + 7) \]
   - Many loops in OpenSSH require disjunctive invariants
Three Challenging Classes of Invariants

1. **Polynomials**
   - Relations over numerical variables
     \[ x = 3.5, \ x = 2y, \ x = qy + r, \ x^2 \geq y + z^3, |\text{arr}| \geq \text{idx} \geq 0, \ldots \]
   - **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
     \[ a \lor b, \ (i = \text{even}) \Rightarrow (A[i] = B[i]), \ \text{if} \ (a = b) \ \text{then} \ (c = 5) \ \text{else} \ (c = d + 7) \]
   - 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
     \[ x = 3.5, \ x = 2y, \ x = qy + r, \ x^2 \geq y + z^3, |\text{arr}| \geq \text{idx} \geq 0, \ldots \]
   - 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
     \[ a \lor b, \ (i = \text{even}) \Rightarrow (A[i] = B[i]), \ \text{if} \ (a = b) \ \text{then} \ (c = 5) \ \text{else} \ (c = d + 7) \]
   - 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)

**INV GENERATOR**
- polynomial relations
- disjunctive relations
- flat array relations
- nested array relations

**Candidate Invariants**

**PROVER**
- $k$-induction
- SMT solving
- lemma learning
- weak invs pruning
- multi processing

**Program Invariants**

**Program Code**

**Program Traces**
**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
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

```python
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
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Invariants at **L**: \( b = ya, \ x = qy + r, \ r \geq 2ya \)
Polynomial Relations (ICSE ’12)

DIG discovers polynomial relations of the forms

**Equalities** \[ c_0 + c_1x + c_2y + c_3xy + \cdots + c_nx^d y^d = 0 \]

**Inequalities** \[ c_0 + c_1x + c_2y + c_3xy + \cdots + c_nx^d y^d \geq 0, \quad c_i \in \mathbb{R} \]

**Examples**

- **Cubic** \[ z - 6n = 6, \quad \frac{1}{12}z^2 - y - \frac{1}{2}z = -1 \]
- **Extended gcd** \[ \text{gcd}(a, b) = ia + jb \]
- **Sqrt** \[ x + \varepsilon \geq y^2 \geq x - \varepsilon \]
Polynomial Relations (ICSE ’12)

DIG discovers polynomial relations of the forms

Equalities  \[ c_0 + c_1 x + c_2 y + c_3 xy + \cdots + c_n x^d y^d = 0 \]

Inequalities  \[ c_0 + c_1 x + c_2 y + c_3 xy + \cdots + c_n x^d y^d \geq 0, \quad c_i \in \mathbb{R} \]

Examples

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

```python
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
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Invariants at L: $b = ya$, $x = qy + r$, $r \geq 2ya$
### Finding Nonlinear Equations using Equation Solver

Terms and degrees

\[ V = \{ r, y, a \} \]

\[ \deg = 2 \]

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

Equation template

\[
\begin{align*}
    c_1 + c_2 r + c_3 y + c_4 a + c_5 ry + c_6 ra + c_7 ya + c_8 r^2 + c_9 y^2 + c_{10} a^2 &= 0
\end{align*}
\]

System of linear equations

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( a )</th>
<th>( b )</th>
<th>( q )</th>
<th>( r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Finding Nonlinear Equations using Equation Solver

- Terms and degrees

\[ V = \{r, y, a\}; \quad \text{deg} = 2 \]

\[ \downarrow \]

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Finding Nonlinear Equations using Equation Solver

Terms and degrees

\[ V = \{r, y, a\}; \text{deg} = 2 \]

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

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

\[
\begin{array}{c||cccc}
 x & y & a & b & q & r \\
 15 & 2 & 1 & 2 & 0 & 15 \\
 15 & 2 & 2 & 4 & 0 & 15 \\
 15 & 2 & 1 & 2 & 4 & 7 \\
 4 & 1 & 1 & 1 & 0 & 4 \\
 4 & 1 & 2 & 2 & 0 & 4 \\
\end{array}
\]
Finding Nonlinear Equations using Equation Solver

- **Terms and degrees**

  \[ V = \{r, y, a\}; \ \text{deg} = 2 \]

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

- **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**

\[ V = \{r, y, a\}; \ \text{deg} = 2 \]

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

**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**

trace 1 : \[ \{r = 15, y = 2, a = 1\} \]

eq 1 : \[ c_1 + 15c_2 + 2c_3 + c_4 + 30c_5 + 15c_6 + 2c_7 + 225c_8 + 4c_9 + c_{10} = 0 \]

::
Finding Nonlinear Equations using Equation Solver

- **Terms and degrees**

\[ V = \{ r, y, a \}; \text{deg} = 2 \]

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

- **Equation template**

\[
c_1 + c_2 r + c_3 y + c_4 a + c_5 r y + c_6 r a + c_7 y a + c_8 r^2 + c_9 y^2 + c_{10} a^2 = 0
\]

- **System of linear equations**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>a</td>
<td>b</td>
<td>q</td>
<td>r</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

- **Solve for coefficients \( c_i \)**

\[ V = \{ x, y, a, b, q, r \}; \text{deg} = 2 \quad \rightarrow \quad b = ya, x = qy + r \]
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

\[
\begin{align*}
 & x - y \\ & -2 \\ & -1 \\ & 1 \\ & 2 \\ & 3 \\ & 5 \\
& y \\ & 1 \\ & -1 \\ & -3 \\ & 0 \\ & -2 \\ & 2
\end{align*}
\]
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

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure: ***

program traces

trace pts in 2D

Figure: *

polygon

$c_1 x + c_2 y \geq c$
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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure: *

trace pts
in 2D

Figure: *

polygon
\[ c_1x + c_2y \geq c \]

Figure: *

program traces

- 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

\[
\begin{array}{cc}
  x & y \\
  -1 & 5 \\
  \vdots & \vdots \\
  5 & 5 \\
  6 & 6 \\
  \vdots & \vdots \\
  11 & 11 \\
\end{array}
\]

Disjunction of 2 cases:
1. if \( x < 5 \) then \( y = 5 \)
2. if \( x ≥ 5 \) then \( x = y \)

\[
\max(0, x - 5) = y - 5
\]
a linear relation in max-plus algebra
Example: Disjunctive Invariants

```python
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
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

\[ L : (x < 5 \land 5 = y) \lor (x ≥ 5 \land x = y), 11 ≥ x \]
Example: Disjunctive Invariants

```python
def ex(x):
    y = 5
    if x > y: x = y
    while x ≤ 10:
        if x ≥ 5:
            y = y + 1
        x = x + 1
    assert y ≡ 11
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

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

Disjunction of 2 cases:

1. if $x < 5$ then $y = 5$
2. if $x \geq 5$ then $x = y$
Example: Disjunctive Invariants

```python
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
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

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

Disjunction of 2 cases:

1. if $x < 5$ then $y = 5$
2. if $x ≥ 5$ then $x = y$

$\iff$ 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

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

Disjunction of 2 cases:

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

$\iff$ 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

Addition \( \oplus \) max

Multiplication \( \times \) +

Zero element \( 0 \)

Unit element \( 1 \)

Relation form

\[
\begin{align*}
0 & \leq t_1 + \cdots + t_n \\
& \leq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)
\end{align*}
\]

Convex hull

Not convex in classical sense!
## Max-plus Algebra

<table>
<thead>
<tr>
<th></th>
<th>Linear</th>
<th>Max-plus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>$\mathbb{R}$</td>
<td>$\mathbb{R} \cup {-\infty}$</td>
</tr>
<tr>
<td><strong>Addition</strong></td>
<td>$+$</td>
<td>$\text{max}$</td>
</tr>
<tr>
<td><strong>Multiplication</strong></td>
<td>$\times$</td>
<td>$+$</td>
</tr>
<tr>
<td><strong>Zero elem</strong></td>
<td>$0$</td>
<td>$-\infty$</td>
</tr>
<tr>
<td><strong>Unit elem</strong></td>
<td>$1$</td>
<td>$0$</td>
</tr>
<tr>
<td><strong>Relation form</strong></td>
<td>$c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0$</td>
<td>$\max(c_0, c_1 + t_1, \ldots, c_n + t_n) \geq \max(d_0, d_1 + t_1, \ldots, d_n + t_n)$</td>
</tr>
</tbody>
</table>
# Max-plus Algebra

<table>
<thead>
<tr>
<th>Linear</th>
<th>Max-plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>$\mathbb{R}$</td>
</tr>
<tr>
<td>Addition</td>
<td>$+$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$\times$</td>
</tr>
<tr>
<td>Zero elem</td>
<td>$0$</td>
</tr>
<tr>
<td>Unit elem</td>
<td>$1$</td>
</tr>
<tr>
<td>Relation form</td>
<td>$c_0 + c_1 t_1 \cdots + c_n t_n \geq 0$</td>
</tr>
</tbody>
</table>

![Line shapes](image)
Max-plus Algebra

<table>
<thead>
<tr>
<th></th>
<th>Linear</th>
<th>Max-plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>( \mathbb{R} )</td>
<td>( \mathbb{R} \cup {-\infty} )</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
<td>max</td>
</tr>
<tr>
<td>Multiplication</td>
<td>( \times )</td>
<td>+</td>
</tr>
<tr>
<td>Zero elem</td>
<td>0</td>
<td>(-\infty)</td>
</tr>
<tr>
<td>Unit elem</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Relation form</td>
<td>( c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0 )</td>
<td>( \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) )</td>
</tr>
</tbody>
</table>

Line shapes

Convex hull
# Max-plus Algebra

<table>
<thead>
<tr>
<th>Linear</th>
<th>Max-plus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>$\mathbb{R}$</td>
</tr>
<tr>
<td>Addition</td>
<td>$+$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$\times$</td>
</tr>
<tr>
<td>Zero elem</td>
<td>$0$</td>
</tr>
<tr>
<td>Unit elem</td>
<td>$1$</td>
</tr>
<tr>
<td>Relation form</td>
<td>$c_0 + c_1 t_1 + \cdots + c_n t_n \geq 0$</td>
</tr>
</tbody>
</table>

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

\[ z = \max(x, y) \equiv (x < y \land z = x) \lor (x \geq y \land z = y) \]
\[ \text{strncpy}(s, d, n) \equiv (n \geq |s| \land |d| = |s|) \lor (n < |s| \land |d| \geq n) \]
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

\[
z = \max(x, y) \equiv (x < y \land z = x) \lor (x \geq y \land z = y)
\]

\[
\text{strncpy}(s, d, n) \equiv (n \geq |s| \land |d| = |s|) \lor (n < |s| \land |d| \geq n)
\]

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

```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
```

Valid results

- $x, y, q, r$ are integers
- $r \geq 0$
- $x = q \times y + r$
Spurious Invariants

Valid results

1. \( x, y, q, r \) are integers
2. \( r \geq 0 \)
3. \( x = q \times y + r \)

Spurious results

1. \( 100 \geq x \geq 0 \)
2. \( 10 \geq y \geq 1 \)
3. \( 100 \geq q - r \geq -8 \)
KIP: k-Induction Prover (ICSE ’14)

KIP, an automatic theorem prover, for verifying invariants

- Implements $k$-induction
- Employs powerful constraint solving
- Learns new lemmas
- Uses multi-processing
- 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 $\equiv$ 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

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.
Current dynamic analysis cannot find any of these 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

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)

```python
def SubBytes(S, a):
    # S is 1D array, a is 2D array
    b =
    [[S[a[i][j]], S[a[i][j+1]], S[a[i][j+2]], S[a[i][j+3]]],
     [S[a[i+1][j]], S[a[i+1][j+1]], S[a[i+1][j+2]], S[a[i+1][j+3]]],
     [S[a[i+2][j]], S[a[i+2][j+1]], S[a[i+2][j+2]], S[a[i+2][j+3]]],
     [S[a[i+3][j]], S[a[i+3][j+1]], S[a[i+3][j+2]], S[a[i+3][j+3]]]]
    return b
```

A Case Study: Advanced Encryption Standard (AES)

def SubBytes(S, a):
    # S is 1D array, a is 2D array
    b =
    
    for i in range(4):
        for j in range(4):
            b[i][j] = S[a[i][j]]
    
    return b

L =

return b

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

$$a[i_1] \ldots [i_n] = b_1[\ldots [b_l[c_0 + c_1i_1 + \cdots + c_ni_n]] \ldots] ?$$

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

$$a[i_1] \ldots [i_n] = b_1[\ldots [b_l[c_0 + c_1i_1 + \cdots + c_ni_n]] \ldots ]$$

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., $a[i][j] = 2b_1[b_2[2i + 3]][b_3[i][b_2[j]]]]$
Experimental Results

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

- **xor2Word** \( R[i] = \text{xor}(A[i], B[i]), A[i] = \text{xor}(R[i], B[i]), B[i] = \text{xor}(R[i], A[i]) \)
- **addRoundKey** \( R[i][j] = \text{xor}(T[i][j], H[i][j]) \)
- **multWord** \( R[i] = T[\text{mod}(L[A[i]] + L[B[i]], 255)] \)
Experimental Results

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

- **xor2Word**
  \[ R[i] = \text{xor}(A[i], B[i]), \ A[i] = \text{xor}(R[i], B[i]), \ B[i] = \text{xor}(R[i], A[i]) \]

- **addRoundKey**
  \[ R[i][j] = \text{xor}(T[i][j], H[i][j]) \]

- **multWord**
  \[ R[i] = T[\text{mod}(L[A[i]] + L[B[i]], 255)] \]

**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**
  \[ R[i][j] = \text{cipherKey}[4i + j] \text{ for } i = 0, \ldots, 7; \ j = 0, \ldots, 3 \]
Experimental Results

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

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

**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** $R[i][j] = \text{cipherKey}[4i + j]$ 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

**Program Synthesis**
Creates a program that meets a given specification

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

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

```python
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

```python
def Q(i, u, d):
    if i:
        b = c[0] + c[1] * u + c[2] * d  # linear exp template
    else:
        b = u

    if (b > d):
        r = 1
    else:
        r = 0

    return r
```

Test suite

\[
Q(1, 0, 100) = 0 \\
Q(1, 11, 110) = 1 \\
Q(0, 100, 50) = 1 \\
Q(1, -20, 60) = 1 \\
Q(0, 0, -10) = 1
\]
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

```python
def P(x, y):
    if 2 * x == y:
        if x > y + 10:
            return 0
        print('L')  # reachable, e.g., x = -20, y = -40
    return 0
```

Template-based Synthesis

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

```python
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
```

- 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 techniques and tools to synthesis
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

```python
def Q(i, u, d):
    if i:
        b = c_0 + c_1 * u + c_2 * d
    else:
        b = u
    if b > d:
        r = 1
    else:
        r = 0
    return r
```

Test suite

- `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`

```python
def main(c_0, c_1, c_2):
    e = Q(1, 0, 100, c_0, c_1, c_2) == 0 and Q(1, 11, 110, c_0, c_1, c_2) == 1 and Q(0, 100, 50, c_0, c_1, c_2) == 1 and Q(1, -20, 60, c_0, c_1, c_2) == 1 and Q(0, 0, 10, c_0, c_1, c_2) == 0 and Q(0, 0, -10, c_0, c_1, c_2) == 1
    if e:
        return 0
```

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

```python
def Q(i, u, d):
    if i:
        b = c_0 \times c_1 \times u + c_2 \times d  #syn template
    else: b = u
    if (b > d): r = 1
    else: r = 0
    return r
```

Test suite

<table>
<thead>
<tr>
<th>Q(i, u, d)</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q(1, 0, 100)</td>
<td>0</td>
</tr>
<tr>
<td>Q(1, 11, 110)</td>
<td>1</td>
</tr>
<tr>
<td>Q(0, 100, 50)</td>
<td>1</td>
</tr>
<tr>
<td>Q(1, -20, 60)</td>
<td>1</td>
</tr>
<tr>
<td>Q(0, 0, 10)</td>
<td>0</td>
</tr>
<tr>
<td>Q(0, 0, -10)</td>
<td>1</td>
</tr>
</tbody>
</table>

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

```python
def Q(i, u, d):
    b = c_0 \times c_1 \times u + c_2 \times d  #syn template
    if i:
        else:
            if (b > d):
                r = 1
            else:
                r = 0
    return r
```

Test suite

```
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
```

Figure: *

Input: a synthesis instance

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

```python
def pQ(i, u, d, c_0, c_1, c_2):
    if i:
        b = c_0 + c_1 \times u + c_2 \times d
    else:
        b = u
    if b > d:
        r = 1
    else:
        r = 0
    return r

def p_main(c_0, c_1, c_2):
    e = pQ(1, 0, 100, c_0, c_1, c_2) == 0 and
    pQ(1, 11, 110, c_0, c_1, c_2) == 1 and
    pQ(0, 100, 50, c_0, c_1, c_2) == 1 and
    pQ(1, -20, 60, c_0, c_1, c_2) == 1 and
    pQ(0, 0, 10, c_0, c_1, c_2) == 0 and
    pQ(0, 0, -10, c_0, c_1, c_2) == 1
    if e:
        [L]  #pass the given test suite
    return 0
```

Figure: *
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 reachability 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 reachability 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
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

Reachability \(\equiv\) 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 reachability problem, solvable using off-the-shelf test-input generators

- **Source code, benchmarks, etc:**
  - [http://www.cs.unm.edu/~tnguyen](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: \( \text{sorted}(A) \), \( \text{member}(a, A) \), \( \text{reverse}(A, B) \), \( A = B \)

- 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: \( x + 2y - 3z + 4 = 0, \quad x = y^2 \)
  - Arrays: \( \text{sorted}(A), \text{member}(a, A), \text{reverse}(A, B), \ A = B \)
- User-defined: \( x = y^2 + 10, \quad x = y^3 \)
- Filters out templates from traces
- Does not find general linear or nonlinear relations
  - e.g., \( b = ya, \quad x = qy + r, \quad r \geq 2ya \)
- Has limited support for relations among arrays and disjunctive invariants
Sample Examples of Polynomials and Disjunctive Invariants

Polynomial Invariants

cohen cb : \( z^2 = 12y + 6z - 12, yz - 18x - 12y + 2z = 6, 6n + 6 = z \)

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

cohendv : \( b = ya, x = qy + r, r \geq 2ya \)

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

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

Max/Min-plus Invariants

ex1 : \( (x < 5 \land y = 5) \lor (5 \leq x \leq 11 \land x = y) \)

strncpy : \( (n \geq |s| \land |d| = |s|) \lor (n < |s| \land |d| \geq n) \)

oddeven5 : \( o_1 = \min(i_1, i_2, i_3, i_4, i_5) \)
\( o_5 = \max(i_1, i_2, i_3, i_4, i_5) \)
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
Precision Matters

- $x, y, q, r$ are integers

```
def intdiv(x, y):
    q = 0
    r = x
    while r >= y:
        a = 1
        b = y
        while r >= 2*b:
            a = a - 2*x
            b = b - 2*x
            r = r - b
        q = q + a
    print "x %d, y %d, q %d, r %d" % (x, y, q, r)
    return q, r
```
Precision Matters

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

- \( x, y, q, r \) are integers
- \( r \geq 0 \)
- \( x \geq q \)
- \( x \geq qy \)
- \( x = qy + r \)
Finding Nonlinear Inequalities using Polyhedra

```python
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
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

```
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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>a</th>
<th>b</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Finding Nonlinear Inequalities using Polyhedra

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>a</td>
<td>b</td>
<td>q</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

```
```
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

- Trace points in 2D
- Polygon: \( c_1 x + c_2 y \geq c \)
- Octagon: \( \pm x \pm y \geq c \)
- Zone: \( x - y \geq c \)
- 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

- trace pts in 2D
- polygon $c_1x + c_2y \geq c$
- octagon $\pm x \pm y \geq c$
- box $\pm x, y \geq c$
- zone $x - y \geq c$
Example: Finding Nested Array Relations

A

7  1  -3
0  1  2

B

1  -3  5  1  0  7  1
0  1  2  3  4  5  6

C

8  5  6  6  2  1  4
0  1  2  3  4  5  6

\[ A[0] = B[C[?]] \]
\[ A[1] = B[C[?]] \]
Example: Finding Nested Array Relations

A

\[ A[0] = B[C[?]] \]
Example: Finding Nested Array Relations

\[
A[0] = B[C[1]]
\]
Example: Finding Nested Array Relations

\[
\]
Example: Finding Nested Array Relations

\[ A \]

\begin{array}{cccccc}
7 & 1 & -3 \\
0 & 1 & 2 \\
\end{array}

\[ B \]

\begin{array}{cccccccc}
1 & -3 & 5 & 1 & 0 & 7 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}

\[ C \]

\begin{array}{cccccccc}
8 & 5 & 6 & 6 & 2 & 1 & 4 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}

- \( A[0] = B[C[1]] \)
Equation Solving for $A[i] = B[C[j]]$

- Reachability Analysis

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

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

- Reachability Analysis

\[
A[0] = B[C[1]]
\]
\[
\]

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

\[
A[0] = B[C[1]], \ A[1] = B[C[2]] \implies \{1 = 0p + q, 2 = 1p + q\}
\]
Equation Solving for $A[i] = B[C[j]]$

Reachability Analysis

\[ A[0] = B[C[1]] \]

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

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

Solve for $p, q$

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

- Reachability Analysis
  
  \[
  A[0] = B[C[1]] \\
  \]

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

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

- Verify (obtained candidate invs guarantee to hold for $i = 0, 1$)
  
  $A[i] = B[C[1i + 1]] \Rightarrow$ 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

\[
A[0] = B[C[1]] \\
\]

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

\[
\begin{align*}
A[0] = B[C[1]], A[1] = B[C[2]] & \Rightarrow \{1 = 0p + q, 2 = 1p + q\} \\
A[0] = B[C[1]], A[1] = B[C[3]] & \Rightarrow \{1 = 0p + q, 3 = 1p + q\}
\end{align*}
\]

- Solve for $p, q$

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

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

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

- Reachability Analysis

\[
A[0] = B[C[1]] \\
\]

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

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

- Solve for $p, q$

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

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

\[
A[i] = B[C[1i + 1]] \implies \text{invalid, does not hold when } i = 2, \text{ i.e., } A[2] \neq B[C[3]]
\]
Equation Solving for $A[i] = B[C[j]]$

- **Reachability Analysis**

  \[
  A[0] = B[C[1]] \\
  \]

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

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

- Solve for $p, q$

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

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

  \[
  A[i] = B[C[1i + 1]] \ \Rightarrow \ \text{invalid, does not hold when } i = 2, \text{ i.e., } A[2] \neq B[C[3]]
  \]
  \[
  A[i] = B[C[2i + 1]] \ \Rightarrow \ \text{valid, holds when } i = 2, \text{ i.e., } A[2] = B[C[5]]
  \]
## Results for Nonlinear Polynomial Invariants

<table>
<thead>
<tr>
<th>Program</th>
<th>Locs</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>intdiv</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>divbin</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>mannadv</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>hard</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>sqrt1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>dijkstra</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>freire1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>freire2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>cohencb</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>egcd1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>egcd2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>egcd3</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>lcm1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>lcm2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>prodbin</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>prod4br</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>fermat1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>fermat2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>knuth</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>geo1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>geo2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>geo3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>ps2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps6</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

**total** 27

<table>
<thead>
<tr>
<th>Program</th>
<th>Locs</th>
<th>Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>egcd3</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>lcm1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>lcm2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>prod4br</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>fermat2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>knuth</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>geo2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>geo3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>ps2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ps6</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

**total** 27
## Results for Nonlinear Polynomial Invariants

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

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>multWord</td>
<td>mult</td>
</tr>
<tr>
<td>xor2Word</td>
<td>xor</td>
</tr>
<tr>
<td>xor3Word</td>
<td>xor</td>
</tr>
<tr>
<td>subWord</td>
<td>subs</td>
</tr>
<tr>
<td>rotWord</td>
<td>shift</td>
</tr>
<tr>
<td>block2State</td>
<td>convert</td>
</tr>
<tr>
<td>state2Block</td>
<td>convert</td>
</tr>
<tr>
<td>subBytes</td>
<td>subs</td>
</tr>
<tr>
<td>invSubByte</td>
<td>subs</td>
</tr>
<tr>
<td>shiftRows</td>
<td>shift</td>
</tr>
<tr>
<td>invShiftRow</td>
<td>shift</td>
</tr>
<tr>
<td>addKey</td>
<td>add</td>
</tr>
<tr>
<td>mixCol</td>
<td>mult</td>
</tr>
<tr>
<td>invMixCol</td>
<td>mult</td>
</tr>
<tr>
<td>keySetEnc4</td>
<td>driver</td>
</tr>
<tr>
<td>keySetEnc6</td>
<td>driver</td>
</tr>
<tr>
<td>keySetEnc8</td>
<td>driver</td>
</tr>
<tr>
<td>keySetEnc</td>
<td>driver</td>
</tr>
<tr>
<td>keySetDec</td>
<td>driver</td>
</tr>
<tr>
<td>keySched1</td>
<td>driver</td>
</tr>
<tr>
<td>keySched2</td>
<td>driver</td>
</tr>
<tr>
<td>aesKeyEnc</td>
<td>driver</td>
</tr>
<tr>
<td>aesKeyDec</td>
<td>driver</td>
</tr>
<tr>
<td>aesEncrypt</td>
<td>driver</td>
</tr>
<tr>
<td>aesDecrypt</td>
<td>driver</td>
</tr>
</tbody>
</table>

25 functions
## Results for Array Invariants

<table>
<thead>
<tr>
<th>Function</th>
<th>Desc</th>
<th>Gen</th>
<th>V, D</th>
<th>$T_{Gen}$ (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>multWord</td>
<td>mult</td>
<td>1 N₄</td>
<td>7, 2</td>
<td>11.0</td>
</tr>
<tr>
<td>xor2Word</td>
<td>xor</td>
<td>3 N₁</td>
<td>4, 2</td>
<td>0.8</td>
</tr>
<tr>
<td>xor3Word</td>
<td>xor</td>
<td>4 N₁</td>
<td>5, 3</td>
<td>2.0</td>
</tr>
<tr>
<td>subWord</td>
<td>subs</td>
<td>2 N₁</td>
<td>3, 1</td>
<td>1.3</td>
</tr>
<tr>
<td>rotWord</td>
<td>shift</td>
<td>1 F</td>
<td>2, 1</td>
<td>0.5</td>
</tr>
<tr>
<td>block2State</td>
<td>convert</td>
<td>1 F</td>
<td>2, 2</td>
<td>4.1</td>
</tr>
<tr>
<td>state2Block</td>
<td>convert</td>
<td>1 F</td>
<td>2, 2</td>
<td>4.2</td>
</tr>
<tr>
<td>subBytes</td>
<td>subs</td>
<td>2 N₁</td>
<td>3, 2</td>
<td>3.2</td>
</tr>
<tr>
<td>invSubByte</td>
<td>subs</td>
<td>2 N₁</td>
<td>3, 2</td>
<td>3.3</td>
</tr>
<tr>
<td>shiftRows</td>
<td>shift</td>
<td>1 F</td>
<td>2, 2</td>
<td>3.7</td>
</tr>
<tr>
<td>invShiftRow</td>
<td>shift</td>
<td>1 F</td>
<td>2, 2</td>
<td>3.6</td>
</tr>
<tr>
<td>addKey</td>
<td>add</td>
<td>2 N₁</td>
<td>4, 2</td>
<td>3.5</td>
</tr>
<tr>
<td>mixCol</td>
<td>mult</td>
<td>0</td>
<td>-</td>
<td>1.0</td>
</tr>
<tr>
<td>invMixCol</td>
<td>mult</td>
<td>0</td>
<td>-</td>
<td>1.0</td>
</tr>
<tr>
<td>keySetEnc4</td>
<td>driver</td>
<td>1 F</td>
<td>2, 2</td>
<td>76.4</td>
</tr>
<tr>
<td>keySetEnc6</td>
<td>driver</td>
<td>1 F</td>
<td>2, 2</td>
<td>78.8</td>
</tr>
<tr>
<td>keySetEnc8</td>
<td>driver</td>
<td>1 F</td>
<td>2, 2</td>
<td>79.3</td>
</tr>
<tr>
<td>keySetEnc</td>
<td>driver</td>
<td>1 F</td>
<td>2, 1</td>
<td>76.3</td>
</tr>
<tr>
<td>keySetDec</td>
<td>driver</td>
<td>0</td>
<td>-</td>
<td>73.0</td>
</tr>
<tr>
<td>keySched1</td>
<td>driver</td>
<td>0</td>
<td>-</td>
<td>77.9</td>
</tr>
<tr>
<td>keySched2</td>
<td>driver</td>
<td>1 F</td>
<td>2, 2</td>
<td>79.5</td>
</tr>
<tr>
<td>aesKeyEnc</td>
<td>driver</td>
<td>1 F, 1 eq</td>
<td>2, 1</td>
<td>76.2</td>
</tr>
<tr>
<td>aesKeyDec</td>
<td>driver</td>
<td>1 eq</td>
<td>2, 1</td>
<td>73.6</td>
</tr>
<tr>
<td>aesEncrypt</td>
<td>driver</td>
<td>1 F</td>
<td>2, 2</td>
<td>70.5</td>
</tr>
<tr>
<td>aesDecrypt</td>
<td>driver</td>
<td>1 F</td>
<td>2, 2</td>
<td>73.8</td>
</tr>
</tbody>
</table>

25 functions | 17/30 invs | 878.5s
CETI: Correcting Errors using Test Inputs (FSE ’14, in submission)

1. Obtain suspicious statements using an existing fault localization tool.
2. Apply synthesis templates to create template-based synthesis instances.
3. Convert to reachability programs using reduction theorem.
4. Employ an off-the-shelf test-input generator to solve reachability, i.e., creating repairs.
5. Outperform other automatic program repair techniques.
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 reachability 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 reachability 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**
  
  \[ A = \begin{bmatrix} 7, 1, -3 \end{bmatrix} \]
  
  \{ \begin{bmatrix} 1, -3, 5, 1, 0, 7, 1 \end{bmatrix}, \begin{bmatrix} 8, 5, 6, 6, 2, 1, 4 \end{bmatrix} \} 

- **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

Reachability ≡ Synthesis
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

Reachability $\equiv$ 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**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo(1, 0, 100)</td>
<td>0</td>
</tr>
<tr>
<td>foo(1, 11, 110)</td>
<td>1</td>
</tr>
<tr>
<td>foo(0, 100, 50)</td>
<td>1</td>
</tr>
<tr>
<td>foo(1, -20, 60)</td>
<td>1</td>
</tr>
<tr>
<td>foo(0, 0, 10)</td>
<td>0</td>
</tr>
</tbody>
</table>
CETI: Correcting Errors using Test Inputs

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

```python
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
```

```python
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
```

```python
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

```python
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
```

```python
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
```

```python
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
```