Program verification via Machine learning

Aditya V. Nori
Programming Languages & Tools group
Microsoft Research India
Joint work with Rahul Sharma, Alex Aiken (Stanford University)
Program verification

1: x = y = 0;
2: while (*)
3:   x++; y++; 
4: while (x != 0)
5:   x--; y--; 
6: assert (y == 0);

1: gcd(int x, int y)
2: {
3:   assume(x>0 && y>0);
4:   while (x !=y ) {
5:     if (x > y) x = x-y;
6:     if (y > x) y = y-x;
7:   }
8:   return x;
9: }

Question
Is the assertion satisfied for all possible inputs?

Question
Does gcd terminate for all inputs x, y?
Current state of affairs

• Precision

• Scalability

• Testing is still the dominant technique for establishing software quality
Question ...

• Most applications are associated with test suites, primarily used for regression or fuzz testing

• Can we use these test suites profitably for proving program correctness?
Here’s the plan ...

• **Guess**: analyse data from tests in order to infer a candidate invariant (use ML techniques)

• **Check**: validate candidate invariant using sound program analysis techniques
  • If check succeeds, then we have a proof!
  • If check fails, use failure to generate more data and repeat guess+check

• Why is this nice?
  • Program analysis not so good at guessing invariants
  • Program analysis is good at checking invariants
  • Able to make use of data generated from programs and existing ML algorithms for analysis
Instantiations of Guess

• Classification
  - Interpolants as Classifiers. Sharma, N, Aiken, Computer-Aided Verification (CAV 2012)
  - Program Verification as Learning Geometric Concepts. Sharma, Gupta, Hariharan, Aiken, N. Submitted

• Linear algebra
  - A Data Driven Approach for Algebraic Loop Invariants. Sharma, Gupta, Hariharan, Aiken, N. European Symposium on Programming (ESOP 2012)

• Regression
  - Termination proofs from tests. N, Sharma. submitted
Interpolants

• An interpolant for a pair of formulas \((A, B)\) s.t. \((A \land B = ⊥)\) is a formula \(I\) satisfying:
  • \(A \Rightarrow I\)
  • \(I \land B = ⊥\)
  • \(\text{vars}(I) \subseteq \text{vars}(A) \cap \text{vars}(B)\)

• An interpolant is a “simple” proof
Example

• $A = x \geq y$

• $B = y \geq x + 1$

• $I = 2x + 1 \geq 2y$
Binary classification

• **Input**: a set of points $X$ with labels $l \in \{+1, -1\}$

• **Goal**: find a classifier $C : X \rightarrow \{true, false\}$ such that:
  - $C(a) = true, \forall a \in X.\text{label}(a) = +1$, and
  - $C(b) = false, \forall b \in X.\text{label}(b) = -1$
Verification & Machine-learning

• **Interpolant**: separates formula $A$ from formula $B$

• **Classifier**: separates positive examples from negative examples

Is there a connection?
Yes!

- **Main result**: view interpolants as classifiers which distinguish “+” examples from “−” examples

- Use state-of-the-art classification algorithms (SVMs) for computing invariants

- SVMs are predictive → generalized predicates for verification
Verification & Machine-learning

Unroll the loops
- Find interpolants
- Get general proofs (loop invariants)

Get positive and negative examples
- Find a classifier
- This is a predicate which generalizes to test data
Example

1:   x = y = 0;
2:   while (*)
3:     x++; y++;
4:   while (x != 0)
5:     x--; y--;
6:   assert (y == 0);
Example ...

A

1: \( x = y = 0; \)
2: \( \text{while } (*) \)
3: \( x++; y++; \)
4: \( \text{while } (x != 0) \)
5: \( x--; y--; \)
6: \( \text{assert } (y == 0); \)

B

\[ A \equiv x_1 = 0 \land y_1 = 0 \land \text{ite}(b, x = x_1 + 1 \land y = y_1 + 1, x = x_1 \land y = y_1) \]

\[ B \equiv \text{ite}(x = 0, x_2 = x - 1 \land y_2 = y - 1, x_2 = x \land y_2 = y) \land x_2 = 0 \land y_2 \neq 0 \]

\[ A \land B = \bot \]

\[ I(x, y) \equiv x = y \]
Example

- \( A \equiv x_1 = 0 \land y_1 = 0 \land \text{ite}(b, x = x_1 + 1 \land y = y_1 + 1, x = x_1 \land y = y_1) \)

- \( B \equiv \text{ite}(x = 0, x_2 = x - 1 \land y_2 = y - 1, x_2 = x \land y_2 = y) \land x_2 = 0 \land y_2 \neq 0 \)

- \( I_1 \equiv 2y \leq 2x + 1 \)
Example

- $A \equiv x_1 = 0 \land y_1 = 0 \land \text{ite}(b, x = x_1 + 1 \land y = y_1 + 1, x = x_1 \land y = y_1)$

- $B \equiv \text{ite}(x = 0, x_2 = x - 1 \land y_2 = y - 1, x_2 = x \land y_2 = y) \land x_2 = 0 \land y_2 \neq 0$

- $I_2 \equiv 2y \leq 2x + 1 \land 2y \geq 2x - 1$
The algorithm

\( \text{Interpolant}(A,B) \)

\[ (X^+,X^-) = \text{Init}(A,B) \]

while(true)
{
    \[ H = \text{SVMI}(X^+,X^-) \]
    
    if \( \text{SAT}(A \land \neg H) \)
        Add s to \( X^+ \) and continue;
    
    if \( \text{SAT}(B \land \neg H) \)
        Add s to \( X^- \) and continue;
    
    break;
}

return \( H \);

Theorem: \( \text{Interpolant}(A,B) \) terminates only if output \( H \) is an interpolant between \( A \) and \( B \)

Find candidate interpolant

\[ A \Rightarrow I \]

\[ I \land B = \bot \]

Exit if interpolant found
Evaluation

- 1000 lines of C++
- LIBSVM for SVM queries
- Z3 theorem prover
Proving termination

• For every loop, *guess* a bound on the number of iterations
• *Check* the bound with a safety checker
Example: GCD

```c
1:  gcd(int x, int y)  
2:   {  
3:     assume(x>0 && y>0);  
4:     while (x != y) {  
5:       if (x > y) x = x-y;  
6:       if (y > x) y = y-x;  
7:     }  
8:     return x;  
9:   }
```
Example: Instrumented GCD

1: gcd(int x, int y)
2: {
3:   assume(x>0 && y>0);
4:   // instrumented code
5:   a = x; b = y; c = 0;
6:   while (x != y ) {
7:     // instrumented code
8:       c = c+1;
9:     writeLog(a, b, c, x, y);
10:    if (x > y) x = x-y;
11:     if (y > x) y = y-x;
12:   }
13:   return x;
14: }

• Inputs
\[(x, y) = \{(1,2), (2,1), (1,3), (3,1)\}\]

\[
\begin{bmatrix}
1 & a & b \\
1 & 1 & 2 \\
1 & 2 & 1 \\
1 & 3 & 1 \\
\end{bmatrix} ,
\begin{bmatrix}
c \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]

• \(A = \begin{bmatrix}
1 & 1 & 3 \\
1 & 1 & 3 \\
1 & 3 & 1 \\
1 & 3 & 1 \\
\end{bmatrix}, C = \begin{bmatrix}
c \\
1 \\
1 \\
2 \\
\end{bmatrix}\)

• Find \(c \approx w_1 a + w_2 b + w_3\) (linear regression)
Linear regression

\[
\min \sum (w_1 a + w_2 b + w_3 - c_i)^2
\]
Quadratic programming

• \( \min \sum_i (w_1 a + w_2 b + w_3 - c_i)^2 \)
  
  \[ s.t. \quad A w \geq C \]

• Guess is \( \tau(a, b) = a + b - 2 \)
Example: Annotated GCD

```c
1:  gcd(int x, int y)  
2:  {  
3:    assume(x>0 && y>0);  
4:    a = x; b = y; c = 0;  
5:    while (x != y) {  
6:      // annotation  
7:      free_invariant(c <= a+b-x-y);  
8:      // annotation  
9:      assert(c <= a+b-2);  
10:     if (x > y) x = x-y;  
11:     if (y > x) y = y-x;  
12:     }  
13:    return x;  
14: }  
```

- Check with a safety checker
- Free invariant to aid checker
  \[ c \leq a + b - x - y \land x > 0 \land y > 0 \]
- Corrective measures
  - Sound rounding for polynomials with integer coefficients
  - Partitioning of tests for discovering disjunctive loop bounds
## Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Infer time (sec)</th>
<th>Validate time (sec)</th>
<th>Total time (sec)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct1</td>
<td>0.0044</td>
<td>0.98</td>
<td>0.98</td>
<td>✓</td>
</tr>
<tr>
<td>Oct2</td>
<td>0.0069</td>
<td>1.00</td>
<td>1.01</td>
<td>✓</td>
</tr>
<tr>
<td>Oct3</td>
<td>0.081</td>
<td>0.98</td>
<td>1.06</td>
<td>✓</td>
</tr>
<tr>
<td>Oct4</td>
<td>0.0035</td>
<td>0.95</td>
<td>0.95</td>
<td>✓</td>
</tr>
<tr>
<td>Oct5</td>
<td>0.0015</td>
<td>0.92</td>
<td>0.92</td>
<td>✓</td>
</tr>
<tr>
<td>Oct6</td>
<td>0.0025</td>
<td>0.95</td>
<td>0.95</td>
<td>✓</td>
</tr>
<tr>
<td>Driver1</td>
<td>0.0066</td>
<td>1.91</td>
<td>1.92</td>
<td>✓</td>
</tr>
<tr>
<td>Driver2</td>
<td>0.0031</td>
<td>2.67</td>
<td>2.67</td>
<td>×</td>
</tr>
<tr>
<td>Driver3</td>
<td>0.0010</td>
<td>2.26</td>
<td>2.26</td>
<td>×</td>
</tr>
<tr>
<td>Driver4</td>
<td>0.0010</td>
<td>2.20</td>
<td>2.20</td>
<td>✓</td>
</tr>
<tr>
<td>Driver5</td>
<td>0.0003</td>
<td>0.94</td>
<td>0.94</td>
<td>✓</td>
</tr>
<tr>
<td>Driver6</td>
<td>0.0042</td>
<td>1.02</td>
<td>1.02</td>
<td>✓</td>
</tr>
<tr>
<td>Driver7</td>
<td>0.0065</td>
<td>1.01</td>
<td>1.02</td>
<td>✓</td>
</tr>
<tr>
<td>Driver8</td>
<td>0.0065</td>
<td>1.00</td>
<td>1.01</td>
<td>✓</td>
</tr>
<tr>
<td>Driver9</td>
<td>0.0012</td>
<td>0.94</td>
<td>0.94</td>
<td>×</td>
</tr>
<tr>
<td>Driver10</td>
<td>0.18</td>
<td>14.69</td>
<td>14.87</td>
<td>✓</td>
</tr>
<tr>
<td>Poly1</td>
<td>0.24</td>
<td>10.11</td>
<td>10.35</td>
<td>✓</td>
</tr>
<tr>
<td>Poly2</td>
<td>0.017</td>
<td>0.95</td>
<td>0.97</td>
<td>✓</td>
</tr>
<tr>
<td>Poly3</td>
<td>0.035</td>
<td>1.33</td>
<td>1.37</td>
<td>✓</td>
</tr>
<tr>
<td>Poly4</td>
<td>FAIL</td>
<td>NA</td>
<td>0.096</td>
<td>FAIL</td>
</tr>
<tr>
<td>Poly6</td>
<td>0.011</td>
<td>3.57</td>
<td>3.58</td>
<td>✓</td>
</tr>
<tr>
<td>Poly7</td>
<td>FAIL</td>
<td>NA</td>
<td>0.011</td>
<td>FAIL</td>
</tr>
<tr>
<td>Poly8</td>
<td>FAIL</td>
<td>NA</td>
<td>0.011</td>
<td>FAIL</td>
</tr>
<tr>
<td>Poly9</td>
<td>0.019</td>
<td>3.36</td>
<td>3.38</td>
<td>✓</td>
</tr>
<tr>
<td>Poly10</td>
<td>FAIL</td>
<td>NA</td>
<td>0.00</td>
<td>FAIL</td>
</tr>
<tr>
<td>Poly11</td>
<td>0.28</td>
<td>1.47</td>
<td>1.75</td>
<td>✓</td>
</tr>
<tr>
<td>Poly12</td>
<td>FAIL</td>
<td>NA</td>
<td>0.016</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

### Group Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Driver</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Poly</td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>11</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>27</td>
<td>22</td>
<td>16</td>
<td>18</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

### Kernel Results

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>#Loops</th>
<th>Infer time (sec)</th>
<th>Validate time (sec)</th>
<th>TPT time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>kbfilt</td>
<td>0.9K</td>
<td>2</td>
<td>0.001</td>
<td>8.8</td>
<td>8.8</td>
</tr>
<tr>
<td>diskperf</td>
<td>2.3K</td>
<td>4</td>
<td>0.001</td>
<td>41.8</td>
<td>41.8</td>
</tr>
<tr>
<td>fakemodem</td>
<td>3.1K</td>
<td>3</td>
<td>0.001</td>
<td>2841.7</td>
<td>2841.7</td>
</tr>
<tr>
<td>serenum</td>
<td>5.3K</td>
<td>17</td>
<td>0.04</td>
<td>2081.3</td>
<td>2081.3</td>
</tr>
<tr>
<td>fpydisk</td>
<td>6K</td>
<td>24</td>
<td>0.04</td>
<td>305.4</td>
<td>305.4</td>
</tr>
<tr>
<td>kbdclass</td>
<td>6.5K</td>
<td>16</td>
<td>0.05</td>
<td>1822.3</td>
<td>1822.3</td>
</tr>
</tbody>
</table>
Summary

• Classification based algorithms can be used for computing proofs in program verification

• Follow-up work on using techniques from linear algebra and PAC learning for scalable proofs

• Proving program termination via linear regression

• Data Driven Program Analysis