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

```
x = y = 0;
while (*)
  x++; y++;
while (x != 0)
  x--; y--;
assert (y == 0);
```

Question

Is the assertion satisfied for all possible inputs?

```
gcd(int x, int y)
{
  assume(x>0 && y>0);
  while (x !=y ) {
    if (x > y) x = x-y;
    if (y > x) y = y-x;
  }
  return x;
}
```

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

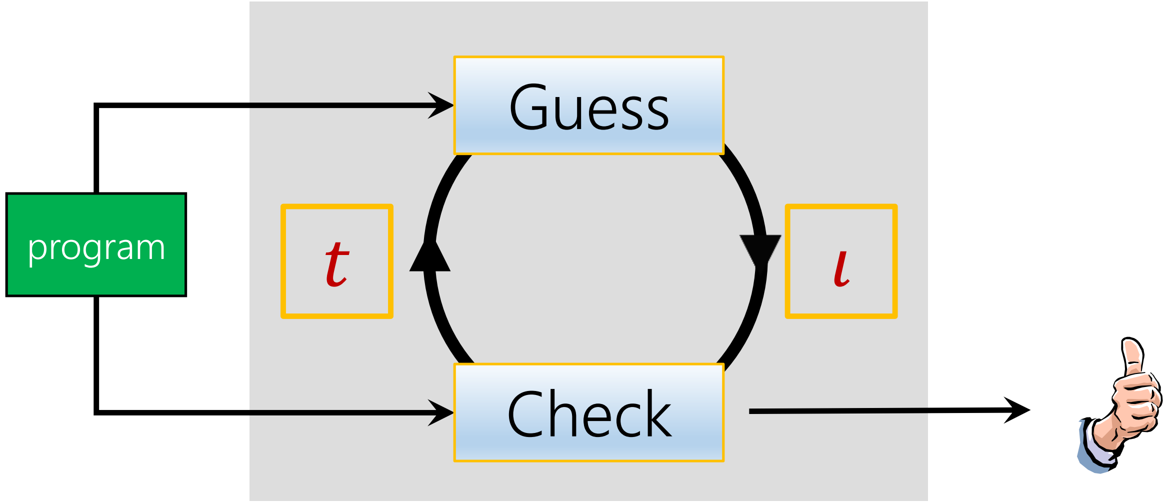
Question …

- 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

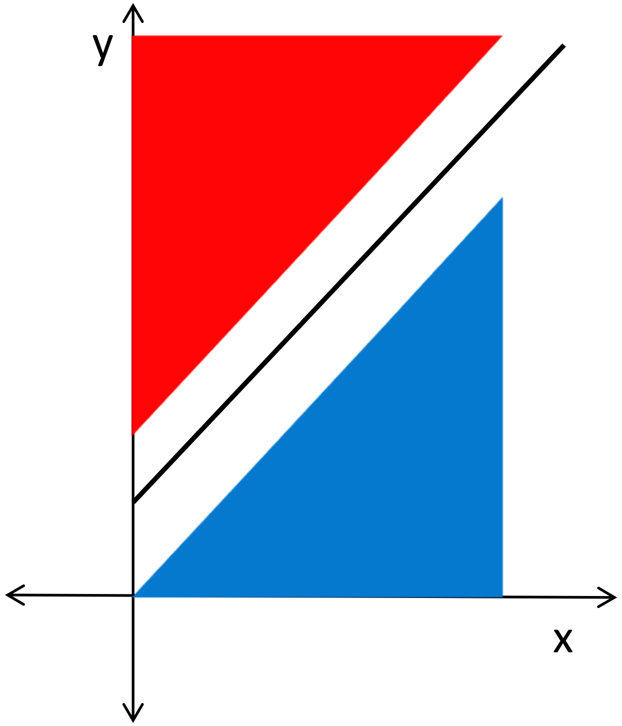
Interpolants

- An interpolant for a pair of formulas (A, B) s.t. $(A \wedge B = \perp)$ is a formula I satisfying:
  - $A \Rightarrow I$
  - $I \wedge B = \perp$
  - $vars(I) \subseteq vars(A) \cap vars(B)$

- 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 $C: \mathrm{X} \rightarrow \{true, false\}$ such that:
  - $C(a) = true, \forall a \in X \,.\, label(a) = +1$, and
  - $C(b) = false, \forall b \in \mathrm{X} \,.\, label(b) = -1$

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 ($\mathbf{SVMs}$) for computing invariants

- $\mathbf{SVMs}$ are predictive $\rightarrow$ 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

```
x = y = 0;
while (*)
  x++; y++;
while (x != 0)
  x--; y--;
assert (y == 0);
```

Example …

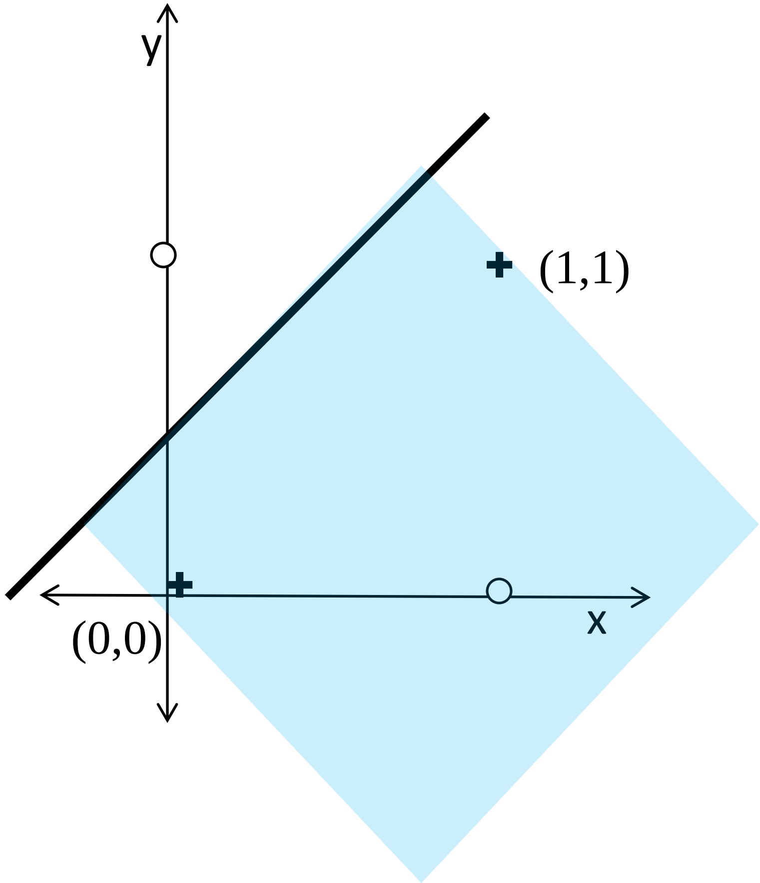
A

B

```
x = y = 0;
while (*)
  x++; y++;
while (x != 0)
  x--; y--;
assert (y == 0);
```

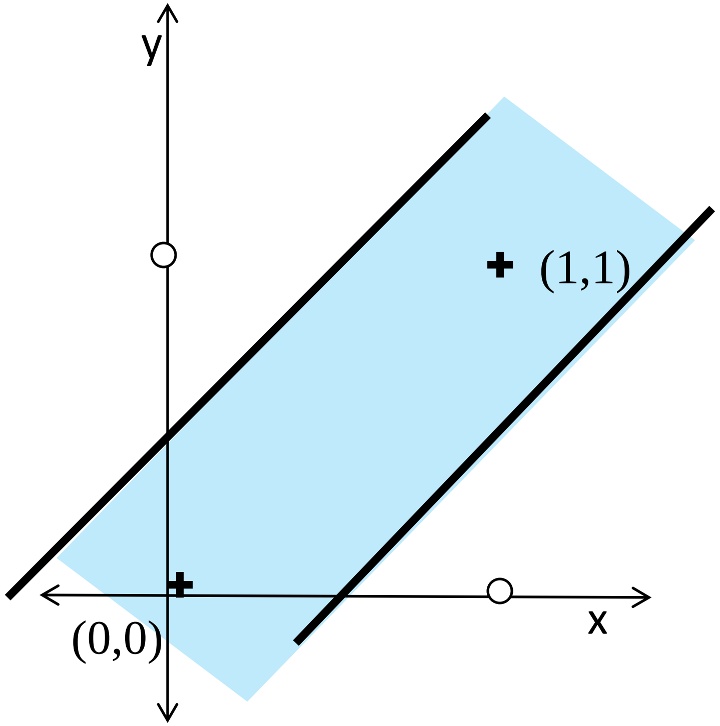
- $A \equiv x_1 = 0 \wedge y_1 = 0 \ \wedge ite(b, x = x_1 + 1 \wedge y = y_1 + 1, x = x_1 \wedge y = y_1)$
- $B \equiv ite(x = 0, x_2 = x - 1 \wedge y_2 = y - 1, x_2 = x \wedge y_2 = y) \ \wedge \ x_2 = 0 \wedge y_2 \neq 0$
- $A \wedge B = \perp$
- $I(x, y) \equiv x = y$

Example



- $A \equiv x_1 = 0 \wedge y_1 = 0 \ \wedge ite(b, x = x_1 + 1 \wedge y = y_1 + 1, x = x_1 \wedge y = y_1)$
- $B \equiv ite(x = 0, x_2 = x - 1 \wedge y_2 = y - 1, x_2 = x \wedge y_2 = y) \wedge \ x_2 = 0 \wedge y_2 \neq 0$
- $I_1 \equiv 2y \leq 2x + 1$

Example



- $A \equiv x_1 = 0 \wedge y_1 = 0 \;\wedge ite(b, x = x_1 + 1 \wedge y = y_1 + 1, x = x_1 \wedge y = y_1)$
- $B \equiv ite(x = 0,\, x_2 = x - 1 \wedge y_2 = y - 1, x_2 = x \wedge y_2 = y) \wedge\; x_2 = 0 \wedge y_2 \neq 0$
- $I_2 \equiv 2y \leq 2x + 1 \wedge 2y \geq 2x - 1$

Interpolant!

The algorithm

Theorem: $Interpolant(A, B)$ terminates only if output H is an interpolant between A and B

$Interpolant(A, B)$

$(X^+, X^-) = Init(A, B)$
while(true)
{
    $H = SVMI(X^+, X^-)$ — Find candidate interpolant

    if $(SAT(A \wedge \neg H))$
        Add s to X^+ and continue; — $A \Rightarrow I$

    if $(SAT(B \wedge \neg H))$
        Add s to X^- and continue; — $I \wedge B = \perp$

    break; — Exit if interpolant found
}
return H;

Evaluation

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

File	LOC	Interpolant	Total Ex	Time (s)	Interpolant	Iterations	Time (s)
f1a	20	x == y	12	0.017	x==y & y >= 0	4	0.017
ex1	22	xa + 2*ya >= 0	13	0.019	xa + 2*ya >= 0	4	0.02
f2	18	3*x >= y	13	0.021	3*x >= y	12	0.022
nec1	17	x <= 8	19	0.015	x <= 8	9	0.02
nec2	22	x < y	12	0.014	x < y	2	0.019
nec3	15	y <= 9	11	0.014	y <= 9	1	0.012
nec4	22	x == y	20	0.019	x == y	4	0.017
nec5	9	s >= 0	11	0.013	s >= 0	1	0.016
pldi08	10	x < 0 | y > 0	17	0.02	6*x < y	1	0.013
fse06	8	y >= 0 & x >= 0	11	0.014	y >= 0 & x >= 0	2	0.015

Proving termination

- For every loop, *guess* a bound on the number of iterations
- *Check* the bound with a safety checker

Example: GCD

```
gcd(int x, int y)
{
  assume(x>0 && y>0);
  while (x !=y ) {
    if (x > y) x = x-y;
    if (y > x) y = y-x;
  }
  return x;
}
```

Example: Instrumented GCD

```
gcd(int x, int y)
{
  assume(x>0 && y>0);
  // instrumented code
  a = x; b = y; c = 0;
  while (x !=y ) {
    // instrumented code
    c = c+1;
    writeLog(a, b, c, x, y);
    if (x > y) x = x-y;
     if (y > x) y = y-x;
  }
  return x;
 }
```

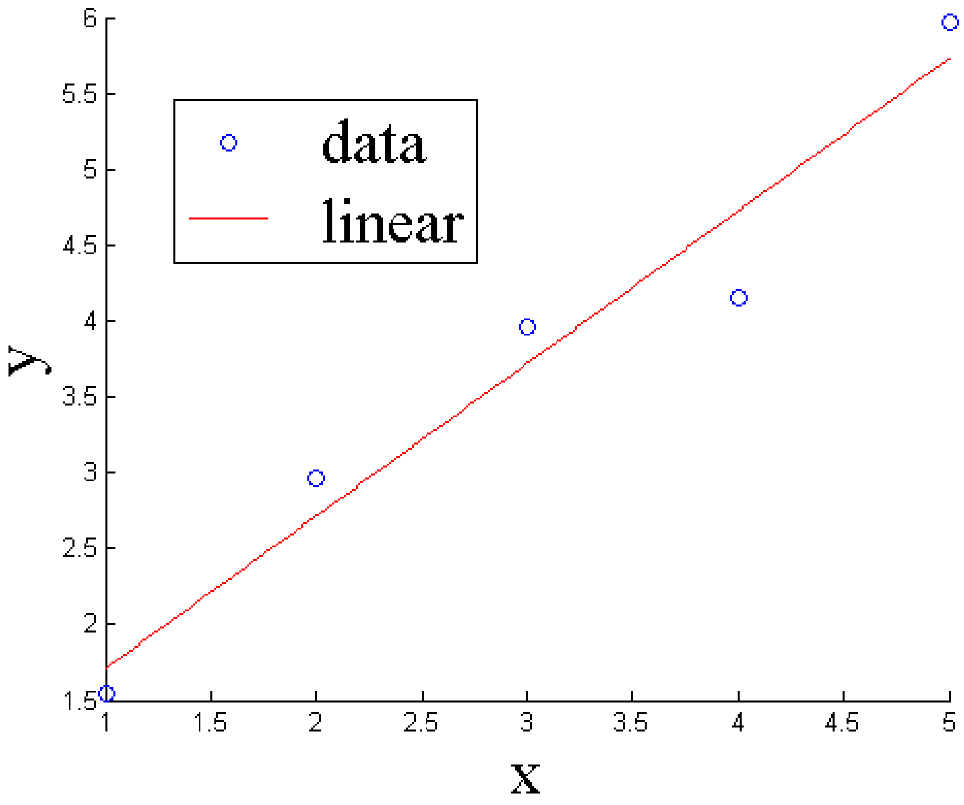
- Inputs

  $(x, y) = \{(1,2), (2,1), (1,3), (3,1)\}$

- $A = \begin{array}{c} \begin{matrix} 1 & a & b \end{matrix} \\ \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 1 & 3 \\ 1 & 3 & 1 \\ 1 & 3 & 1 \end{bmatrix} \end{array}, C = \begin{array}{c} c \\ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 2 \end{bmatrix} \end{array}$

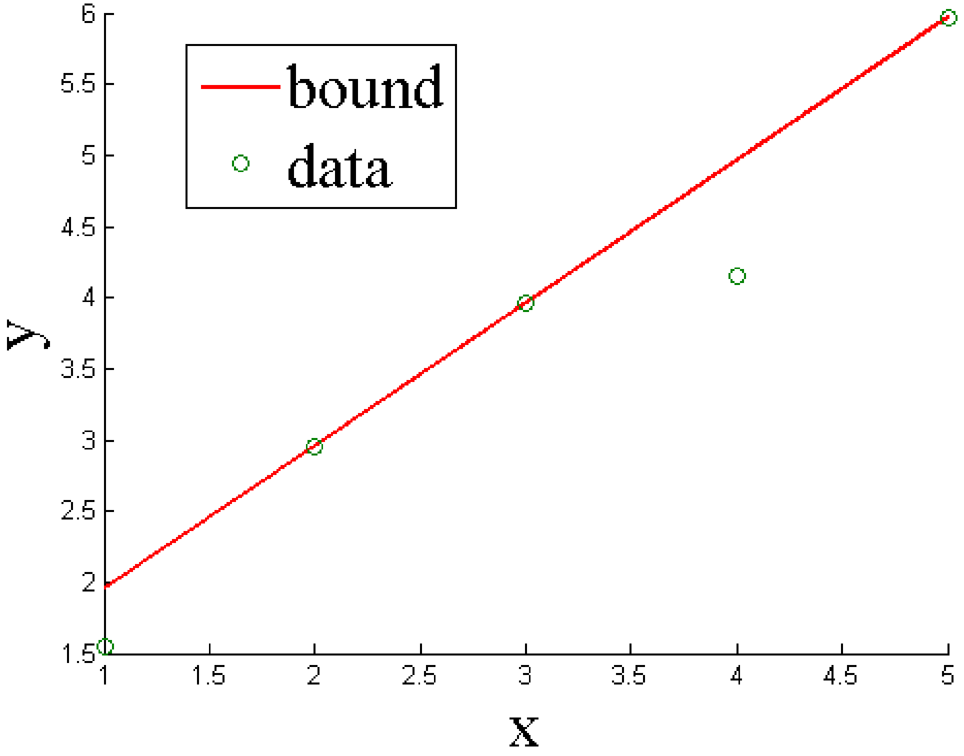
- Find $c \approx w_1 a + w_2 b + w_3$ (linear regression)

Linear regression



- $\min \sum_i (w_1 a + w_2 b + w_3 - c_i)^2$

Quadratic programming



- $\min \sum_i (w_1 a + w_2 b + w_3 - c_i)^2$

  $s.t. \quad Aw \geq C$
- Guess is $\tau(a,b) = a + b - 2$

Example: Annotated GCD

```
gcd(int x, int y)
{
  assume(x>0 && y>0);
  a = x; b = y; c = 0;
  while (x !=y ) {
    // annotation
    free_invariant(c <= a+b-x-y);
    // annotation
    assert(c <= a+b-2);
    if (x > y) x = x-y;
    if (y > x) y = y-x;
  }
  return x;
}
```

- 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

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

Group	Total	TPT	O [7]	P [7]	PR [7]	T [7]	LR [11]	LTA [28]	LF [28]	CTA [28]
Oct	6	6	6	6	2	4	6	6	5	4
Driver	10	10	10	10	1	9	10	10	5	8
Poly	11	6	0	2	11	3	2	2	0	0
All	27	22	16	18	14	16	18	18	10	12

Name	LOC	#Loops	Infer time (sec)	Validate time (sec)	TPT time (sec)
kbfiltr	0.9K	2	0.001	8.8	8.8
diskperf	2.3K	4	0.001	41.8	41.8
fakemodem	3.1K	3	0.001	2841.7	2841.7
serenum	5.3K	17	0.04	2081.3	2081.3
flpydisk	6K	24	0.04	305.4	305.4
kbdclass	6.5K	16	0.05	1822.3	1822.4

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