Software Reliability via Machine Learning

Aditya V. Nori

Programming Languages and Tools group

Microsoft Research

Joint work with Rahul Sharma, Alex Aiken (Stanford University)

Software validation problem

I hope some hacker cannot steal all my money, publish all my email on the web!

I hope it doesn't crash!

Does the software work?

I hope it can handle my peak transaction load!

I hope this version still interoperates with other software!

A problem has been detected and Windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Possible solution: Testing

- The "old-fashioned" and practical method of validating software
- Generate test inputs and see if we can find a test that violates the assertion

What's wrong with testing?

program correctness. Today a usual technique is to make a program and then to test it. But: program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence. The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness. But one should not first make

If we view testing as a "black-box" activity, Dijkstra is right!

After executing many tests, we still don't know if there is another test that can violate the assertion

Program verification

The algorithmic discovery of properties of a program by inspection of the source text

- *Manna and Pnueli, "Algorithmic Verification"*

Also known as: static analysis, static program analysis, formal methods, ….

Program verification

UNDECIDABL

```
1:   x = y = 0;
2:   while (*)
3:     x++;
4:   w
5:
6:   ass
```

Safety
Is the assertion satisfied for all possible inputs?

```
                , int y)
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    }
```

Termination
Does gcd terminate for all inputs x, y?

Current state of the affairs

Safety

- SLA… (device driv…
- ASTREE (… soft…
- Tec…gy: predicate abstraction, abstract interpretation …

Termination

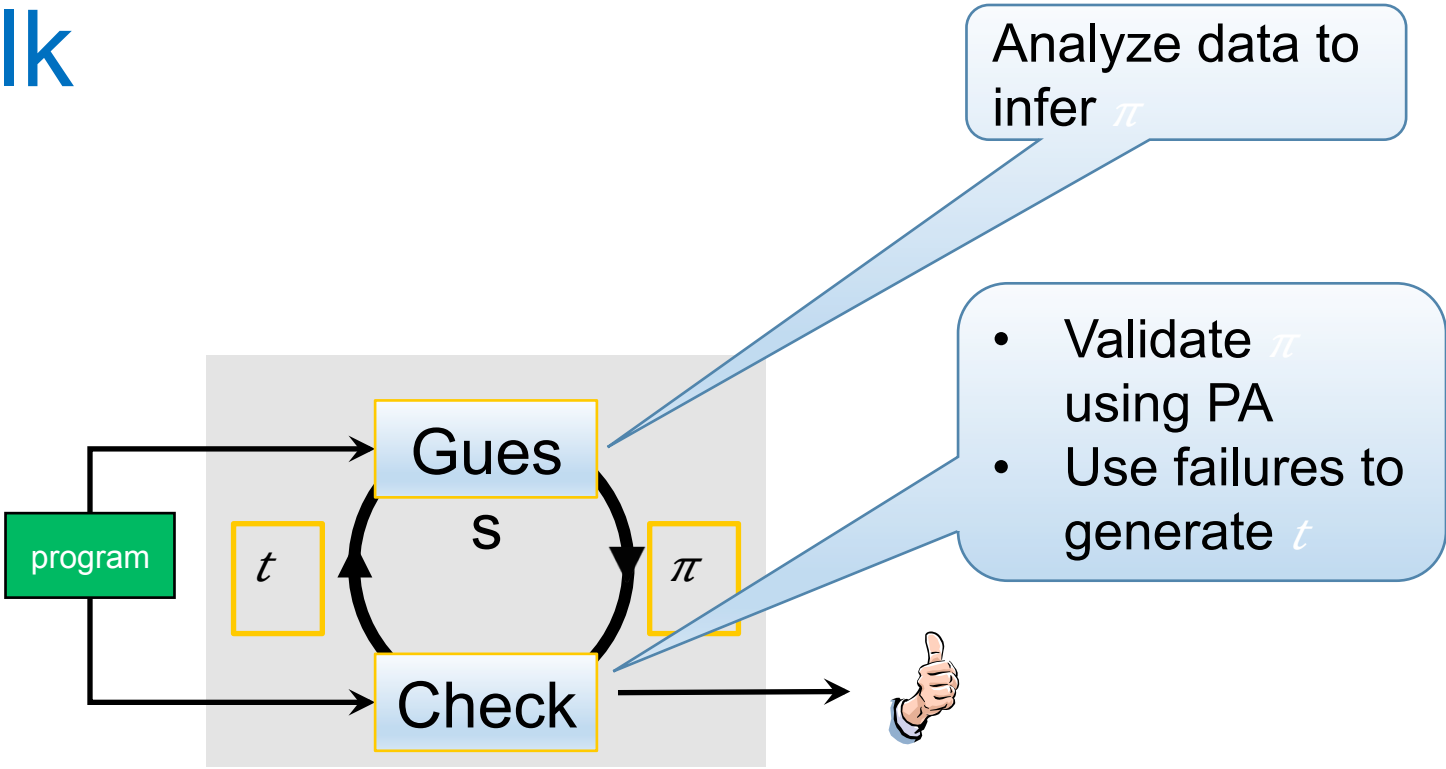
- Termin… vice d…, LF …
- …logy: abstract transi…nts, ranking func…s …

Question

- Most applications are associated with test suites, primarily used for regression or random testing

- Can we use th test suites for proving program correc ss?

This talk



- Proving safety: Guess=Classification, π=loop invariant
- Proving termination: Guess = Regression, π=loop bound
- Bias-Variance tradeoffs in program analysis

Proving correctness

```
1:  x = y = 0;
```
$(pc=2) \Rightarrow 2y \leq 2x+1 \wedge 2y \geq 2x-1$
```
2:  while (*)
```
$(pc=3) \Rightarrow 2y \leq 2x+1 \wedge 2y \geq 2x-1$
```
3:      x++; y++;
```
$(pc=4) \Rightarrow 2y \leq 2x+1 \wedge 2y \geq 2x-1$
```
4:  while (x != 0)
```
$(pc=5) \Rightarrow 2y \leq 2x+1 \wedge 2y \geq 2x-1$
```
5:      x--; y--;
```
$(pc=6) \Rightarrow x=0 \wedge y=0$
```
6:  assert(y == 0);
```

Invariants

Example …

- $A \wedge B = \bot$
- $I(x,y) \equiv 2y \leq 2x+1 \wedge 2y \geq 2x-1$

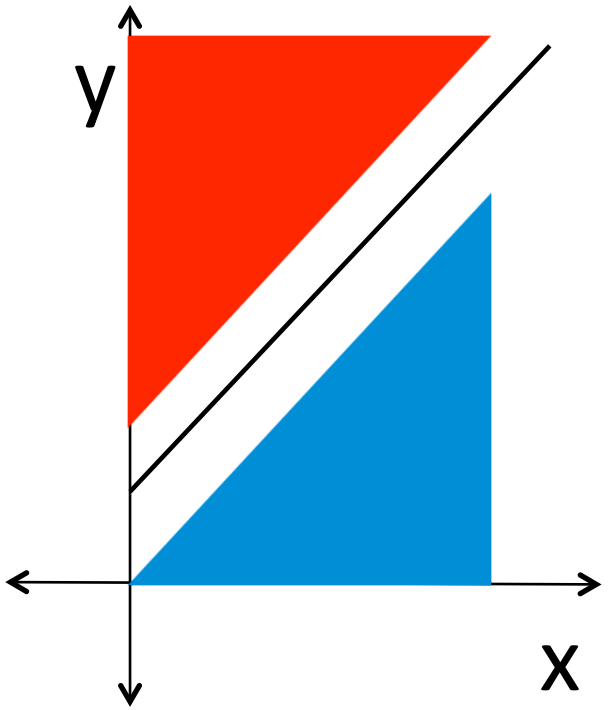
```
1:   x = y = 0;
2:   if (*)
3:     x++; y++;
4:
5:   if (x != 0)
6:     x--; y--;
7:   if (x == 0)
8:     assert (y == 0);
```

A B

- $A \equiv x\downarrow 1 = 0 \wedge y\downarrow 1 = 0 \wedge ite(b, x = x\downarrow 1 + 1 \wedge y = y\downarrow 1 + 1, x = x\downarrow 1 \wedge y = y\downarrow 1)$
- $B \equiv ite(x = 0, x\downarrow 2 = x - 1 \wedge y\downarrow 2 = y - 1, x\downarrow 2 = x \wedge y\downarrow 2 = y) \wedge x\downarrow 2 = 0 \wedge y\downarrow 2 \neq 0$

Interpolants (simple invariants)

- $A = x \geq y$
- $B = y \geq x + 1$
- $I = 2x + 1 \geq 2y$



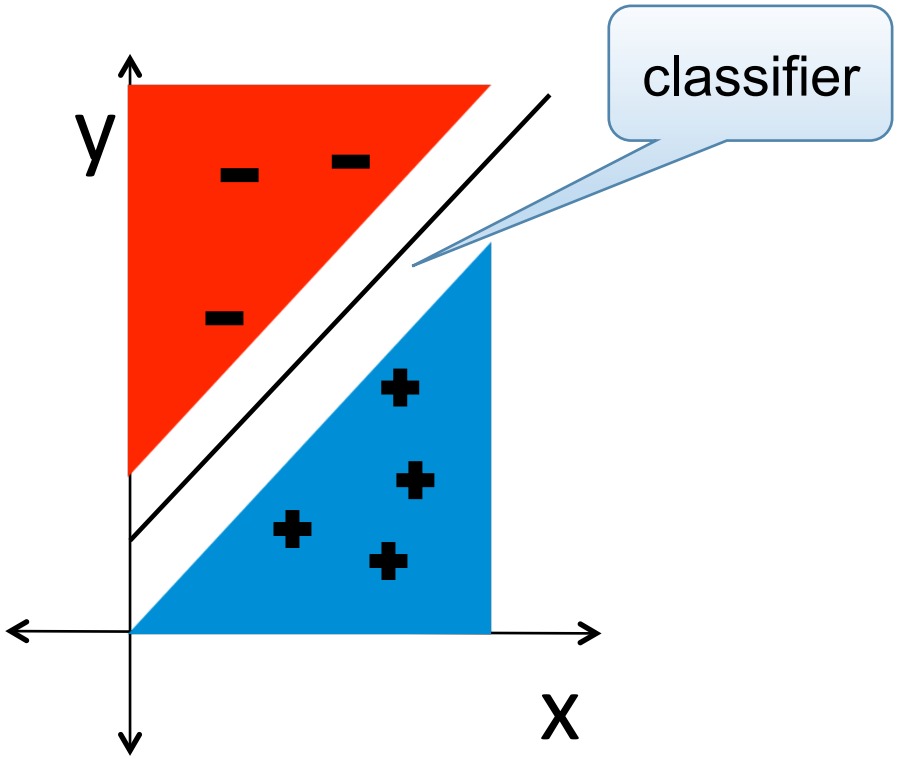
- $A \Rightarrow I$
- $I \wedge B = \bot$
- $vars(I) \subseteq vars(A) \cap vars(B)$

Existing work

- Interpolants used in tools
  - BLAST, IMPACT …
- Based on symbolic techniques
  - Interpolants from proofs (Krajícek[‘97], Pudlák[‘97], McMillan[‘05], …)
  - Interpolants from constraint solving (Rybalchenko et al. [‘07])

Interpolants (simple invariants)

- $A = x \geq y$
- $B = y \geq x + 1$
- $I = 2x + 1 \geq 2y$



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:
  - $C(a) = true, \forall a \in X\,.\, label(a) = +1$, and
  - $C(b) = false, \forall b \in$ X $.\, label(b) = -1$

Binary classification

Training data

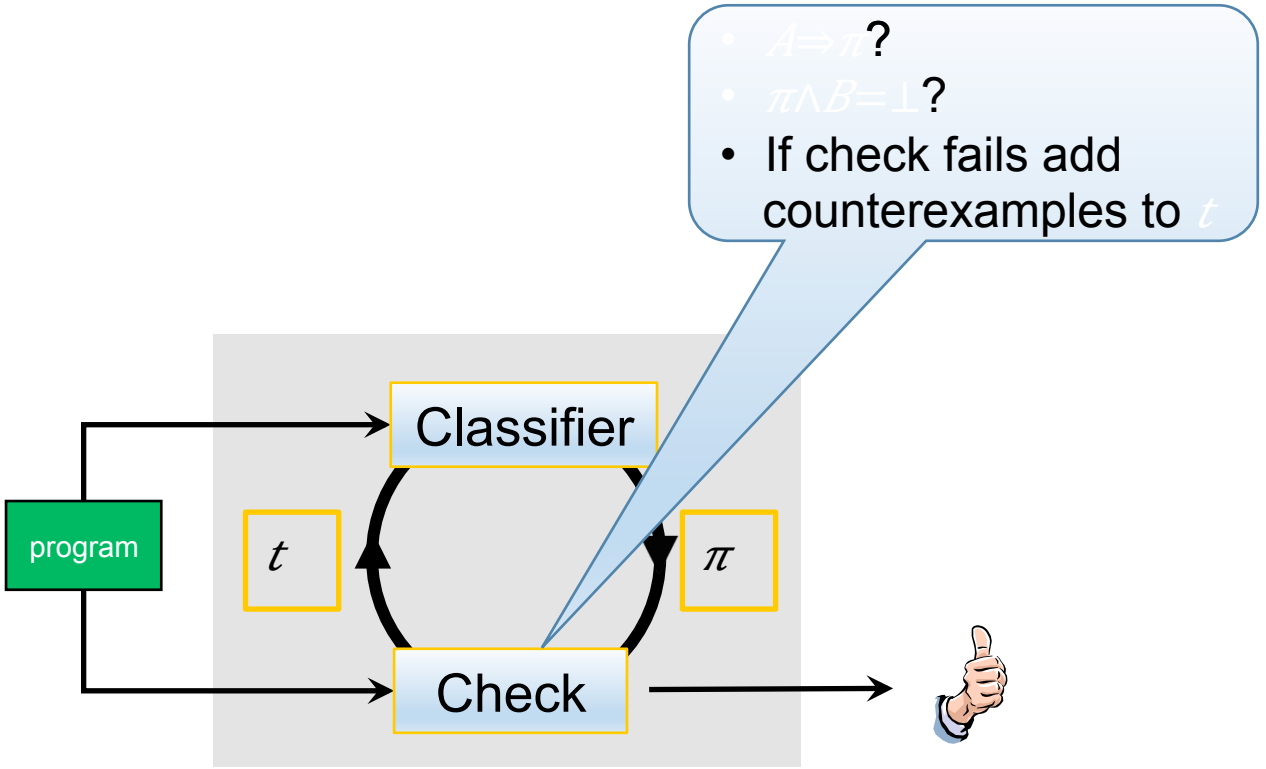
- Input: a set of points X with labels $l \in \{+1, -1\}$

Training

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

Also, c should be predictive

Interpolants as classifiers



- Interpolants as Classifiers. Sharma, Nori, Aiken, *Computer-Aided Verification (CAV 2012)*

Example

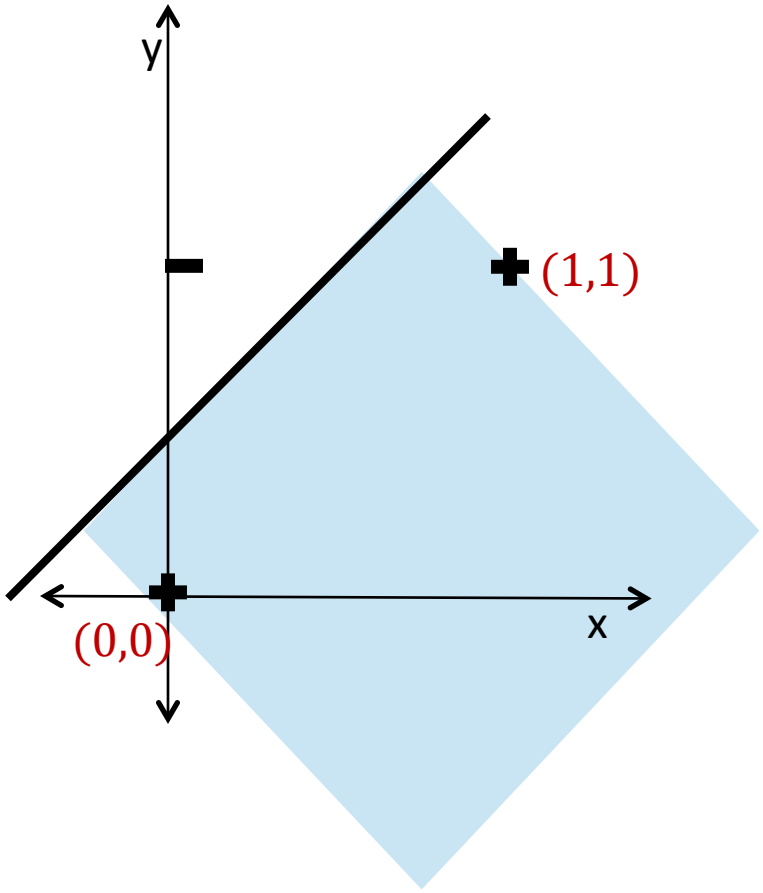
```
1:  x = y = 0;
2:  while (*)
3:    x++; y++;
4:  while (x != 0)
5:    x--; y--;
6:  assert (y == 0);
```

Example

```
1:   x = y = 0;
2:   if (*)
3:     x++; y++;
4:
5:   if (x != 0)
6:     x--; y--;
7:   if (x == 0)
8:     assert (y == 0);
```

- $A \equiv x\downarrow 1 = 0 \wedge y\downarrow 1 = 0 \wedge ite(b, x = x\downarrow 1 + 1 \wedge y = y\downarrow 1 + 1, x = x\downarrow 1 \wedge y = y\downarrow 1)$
- $B \equiv ite(x \neq 0, x\downarrow 2 = x - 1 \wedge y\downarrow 2 = y - 1, x\downarrow 2 = x \wedge y\downarrow 2 = y) \wedge x\downarrow 2 = 0 \wedge y\downarrow 2 \neq 0$
- $A \wedge B = \bot$, $I(x,y) \equiv x = y$

Example



- $A \equiv x\downarrow 1 = 0 \wedge y\downarrow 1 = 0 \wedge ite(b, x = x\downarrow 1 + 1 \wedge y = y\downarrow 1 + 1, x = x\downarrow 1 \wedge y = y\downarrow 1)$
- $B \equiv ite(x = 0, x\downarrow 2 = x - 1 \wedge y\downarrow 2 = y - 1, x\downarrow 2 = x \wedge y\downarrow 2 = y) \wedge x\downarrow 2 = 0 \wedge y\downarrow 2 \neq 0$
- $I(x,y) \equiv 2y \leq 2x + 1$

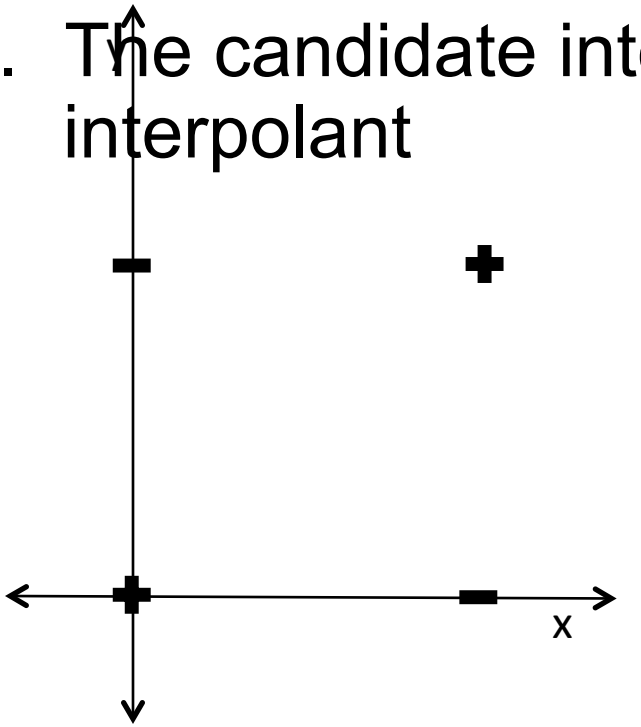
The Basic algorithm

$Basic(A, B)$
$vars$:= Common variables of A and B;
Add $Samples(vars, A)$ to $X\uparrow+$;
Add $Samples(vars, B)$ to $X\uparrow-$;
sep := $BinaryClassifer(X\uparrow+, X\uparrow-)$;
h := $ContainingPred(sep, X\uparrow+)$;
$return\ h$

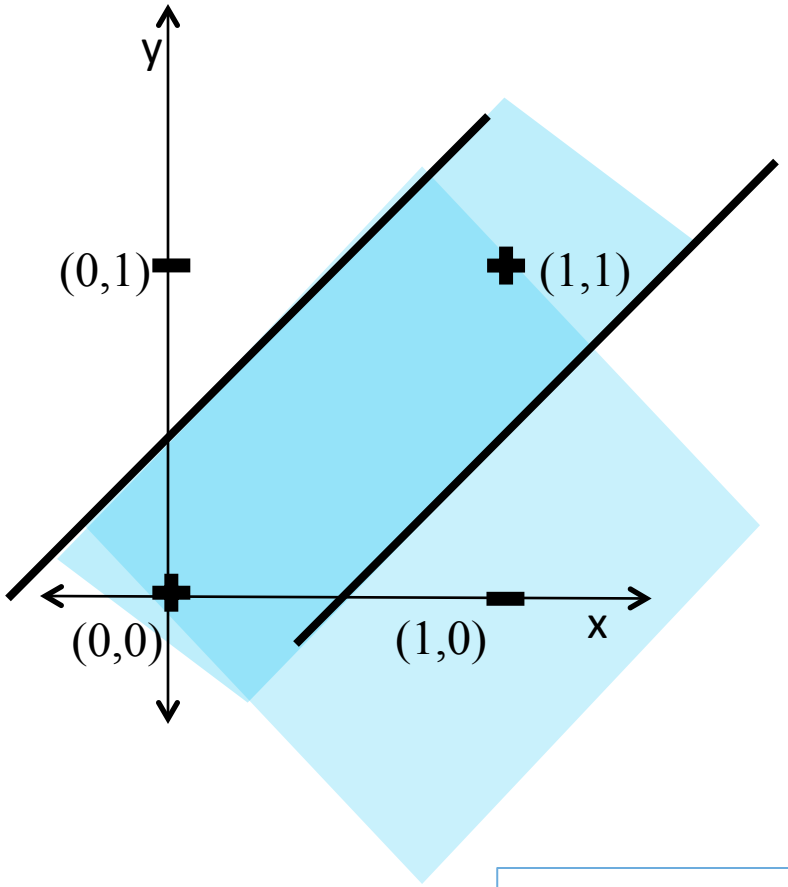
Problems with Basic

1. Data is not linearly separable

2. The candidate interpolant might not an interpolant



No separating inequality?



- For each $x \in X\uparrow -$

$$h\downarrow x = BC(X\uparrow +, \{x\})$$

return $\bigwedge x\uparrow \blacksquare h\downarrow x$

$I \equiv 2y \leq 2x+1 \wedge 2y \geq 2x-1$

Candidate is not an interpolant?

$Interpolant(A,B)$

$(X\uparrow+ , X\uparrow-) = Init(A,B)$
while(true)
{
  $H = BCI(X\uparrow+ , X\uparrow-)$ ⟵ Find candidate interpolant

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

  if ($SAT(B \wedge \neg H)$) ⟵ $I \wedge B = \bot$
    Add s to $X\uparrow-$ and continue;

  break; ⟵ Exit if interpolant found
}
return H;

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

Handling superficial non-linearities

```
void foo()
{
 1:   z = nondet();
 2:   x = 4 * sin(z) * sin(z);
 3:   y = 4 * cos(z) * cos(z);
 4:   assert(x != 2 || y == 2)
}
```

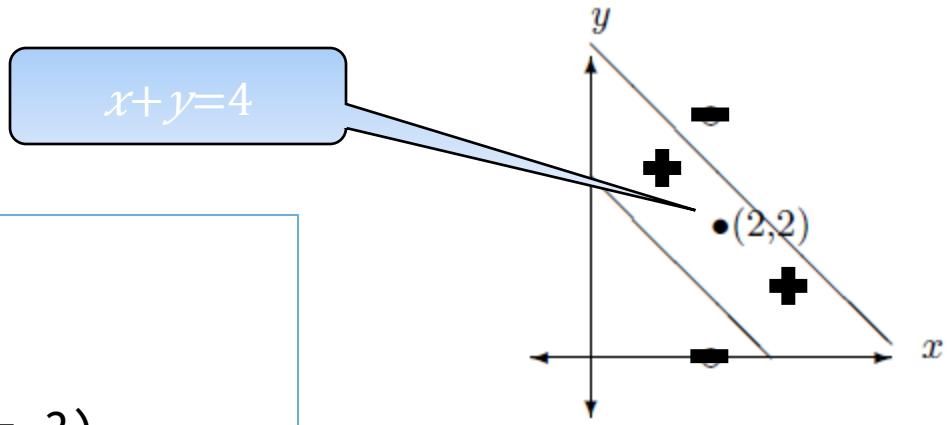
Trace = <1,2,3,4,5>

$A = symExec(<1,2,3>)$

$B = symExec(<4>)$

$A \equiv x \downarrow 1 = \sin \uparrow 2\ z \wedge y = \cos \uparrow 2\ z$

$B \equiv x = 2 \wedge y \neq 2$

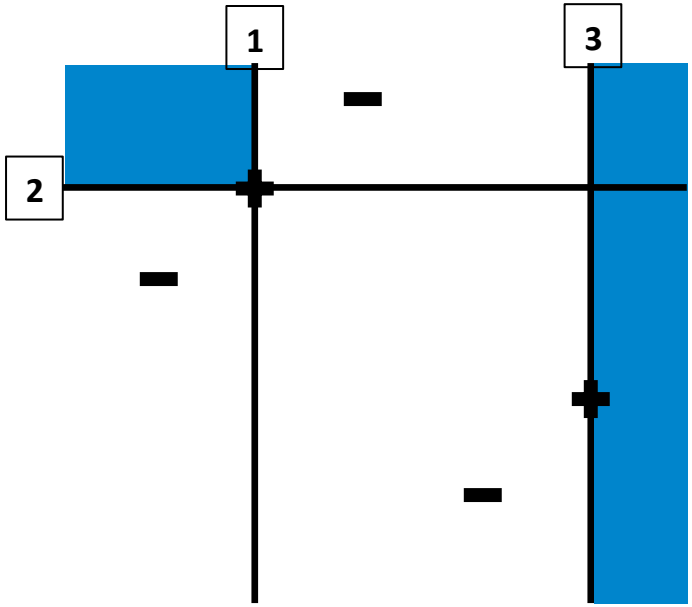


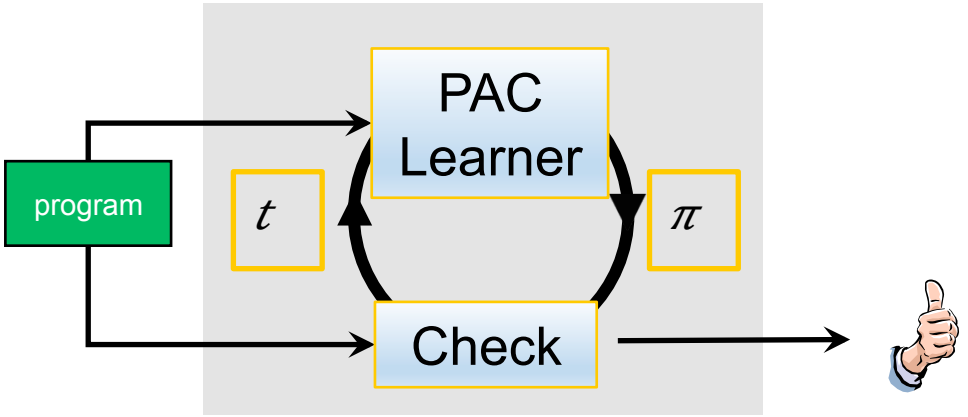

```
void foo()
{
 1: assume(x+y == 4)
 2: assert(x != 2 || y == 2)
}
```

Evaluation

Program	LOC	Interpolant	#Tests	Time (s)
f1a	20	$x=y$	12	0.017
ex1	22	$x+2y\geq 0$	13	0.019
f2	18	$3x\geq y$	13	0.021
nec1	17	$x\leq 8$	19	0.015
nec2	22	$x<y$	12	0.014
nec3	15	$y\leq 9$	11	0.014
nec4	22	$x=y$	20	0.019
nec5	9	$s\geq 0$	11	0.013
pldi08	10	$x<0 \vee y>0$	17	0.02
fse06	8	$y\geq 0 \wedge x\geq 0$	11	0.014

Interpolant not conjunctive?

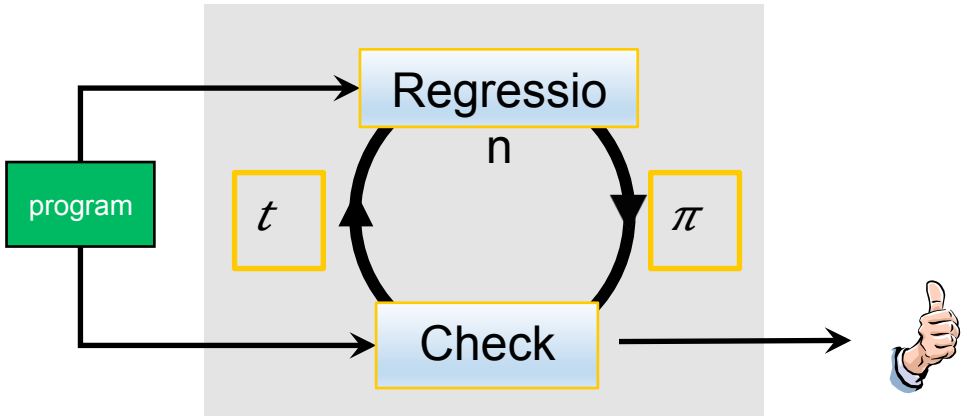




- Program Verification as Learning Geometric Concepts. Sharma, Gupta, Hariharan, Aiken, Nori. *Static Analysis Symposium (SAS 2013)*
- A Data Driven Approach for Algebraic Loop Invariants. Sharma, Gupta, Hariharan, Aiken, Liang, Nori. *European Symposium on Programming (ESOP 2013)*

Proving termination

- *guess* a loop bound π
- *check* the loop bound π with a safety checker



❑Termination proofs from tests. Nori, Sharma. *Foundations of Software Engineering (FSE 2013)*

Example: GCD

```
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

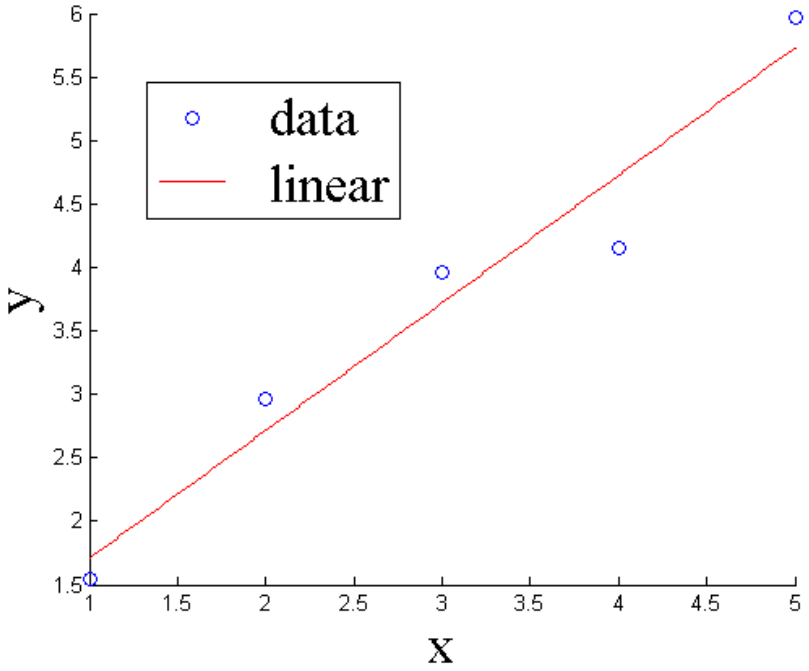
```
1:  gcd(int x, int y)
2:  {
3:    assume(x>0 && y>0);
4:    // instrumented code
5:    a = x; b = y; c = 0;
6:    while (x !=y ) {
7:      // instrumented code
8:      c = c+1;
9:      writeLog(a, b, c, x, y);
10:     if (x > y) x = x-y;
11:     if (y > x) y = y-x;
12:   }
13:   return x;
14:  }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

- Inputs

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

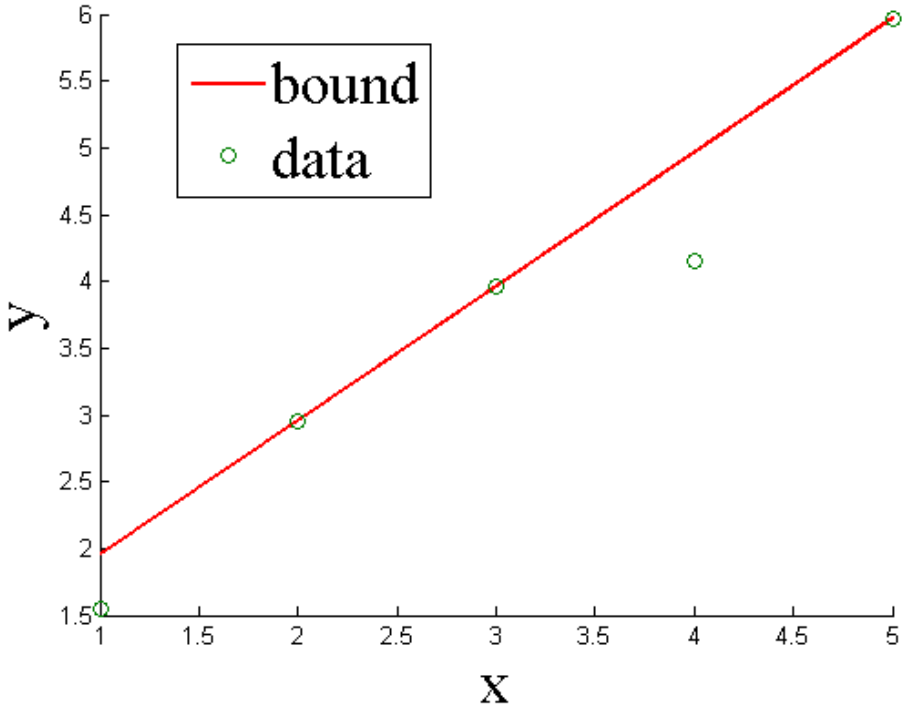
- $A=[\blacksquare a\&b@1\&2@2\&1@1\&3@1\&3@3\&1@3\&1\ ]$, $C=[\blacksquare c@1@1@1@2@1@2\ ]$
- Find $c\approx w\downarrow 1\ a+w\downarrow 2\ b+w\downarrow 3$ (linear regression)

Linear regression



- $\min \sum i \uparrow \blacksquare (w\downarrow 1\ a+w\downarrow 2\ b+w\downarrow 3 - c\downarrow i\ ) \uparrow 2$

Quadratic programming



- $\min \sum i \uparrow \square (w\downarrow 1\ a+w\downarrow 2\ b+ w\downarrow 3 -c\downarrow i\ ) \uparrow 2$

  $s.t. \qquad Aw \geq C$

- Guess is $\tau(a,b)=a+b-2$

Example: Annotated GCD

```
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:      c = c+1;
8:      assert(c <= a+b-2);
9:       if (x > y) x = x-y;
10:     if (y > x) y = y-x;
11:   }
12:   return x;
13:  }
```

- Check with a safety checker

Evaluation – Micro-benchmarks

- Guess implemented in MATLAB
  - $quadprog(A\uparrow T * A, -A\uparrow T * C, -A, -C)$
- Benchmarks
  - Octanal distribution
  - Driver distribution
  - Poly distribution
- TpT works on 15% more benchmarks!

Evaluation – Device drivers

Driver	LOC	#Loops	Guess (s)	Check (s)	TpT (s)
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

Observation

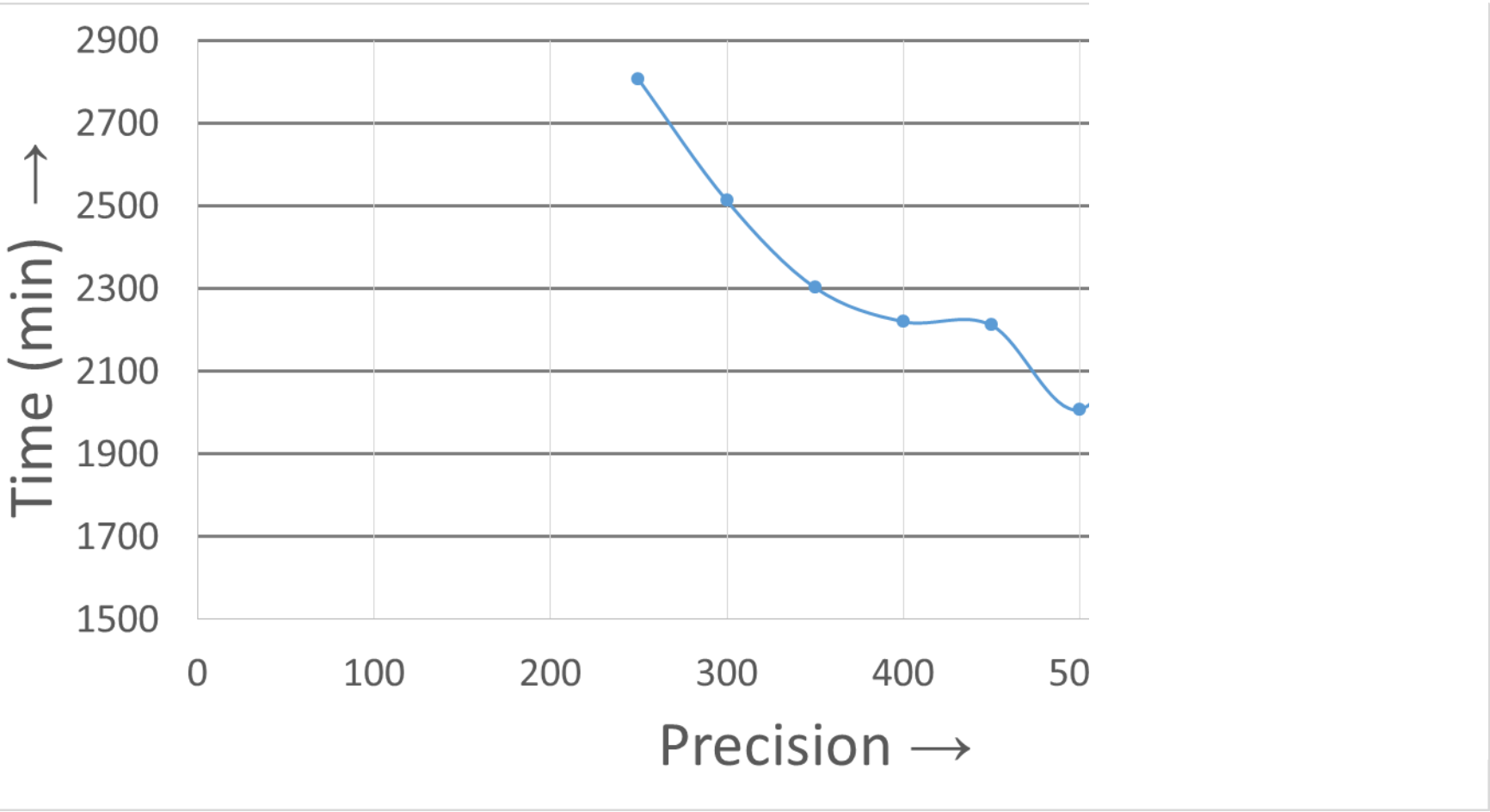
```
int i=1, j=0;
while (i<=5) {
   j = j+i ;
   i = i+1;
}
```

Increasing precision

Invariant inference

- Intervals
  - $I \equiv 1 \leq i \leq 5 \wedge 0 \leq j$
- Octagons
  - $I \wedge 1 - j \leq i \leq j + 1$
- Polyhedra
  - $i \leq 5$

Another Example: Yogi



Nori and Rajamani. An empirical study of optimizations in YOGI.
International Conference on Software Engineering (ICSE 2010)

The Problem

- Increased precision is causing worse results

- Program analysis
  - Analyze all behaviors
  - Run for a finite time

- In finite time, observe only finite behaviors

- Need to generalize

Generalization

- Abstract interpretation: widening
- CEGAR: interpolants
- Parameter tuning of tools
- Lot of folk knowledge, heuristics, …

Machine Learning

- “It’s all about generalization”
- Learn a function from observations
- Hope that the function generalizes
- Existing formal theory for generalization

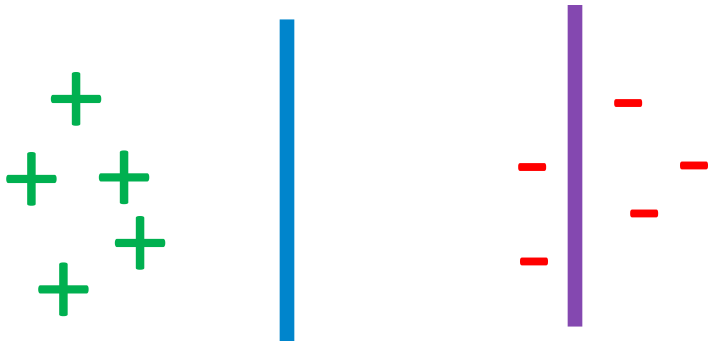
PAC Learning Framework

- Find hypothesis that works on given behaviors
- Hope that it generalizes

- Assume an arbitrary but fixed distribution Δ
- Given m (iid) samples from Δ
- Each sample x_i has a *label* y_i $(+/-)$

PAC Learning Framework

- Find hypothesis that works on given behaviors
- Hope that it generalizes

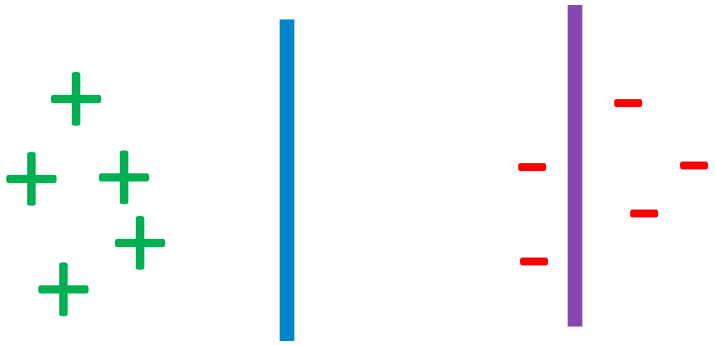


- *Empirical error* of a hypothesis h

$$\epsilon(h) = 1/m \sum i\uparrow\blacksquare h(x\downarrow i) \neq y\downarrow i$$

PAC Learning Framework

- Find hypothesis that works on given behaviors
- Hope that it generalizes



- *Empirical risk minimization* (ERM)
  - Given a set H of possible hypotheses (precision)
  - Select $h \in H$ that minimizes empirical error

PAC Learning Framework

- Find hypothesis that works on given behaviors
- Hope that it generalizes

- Generalization error: for a new sample (x,y)

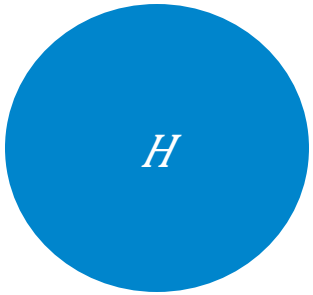
  $\epsilon(h)=\Pr[h(x)\neq y]$

- Relate generalization error to empirical error and precision

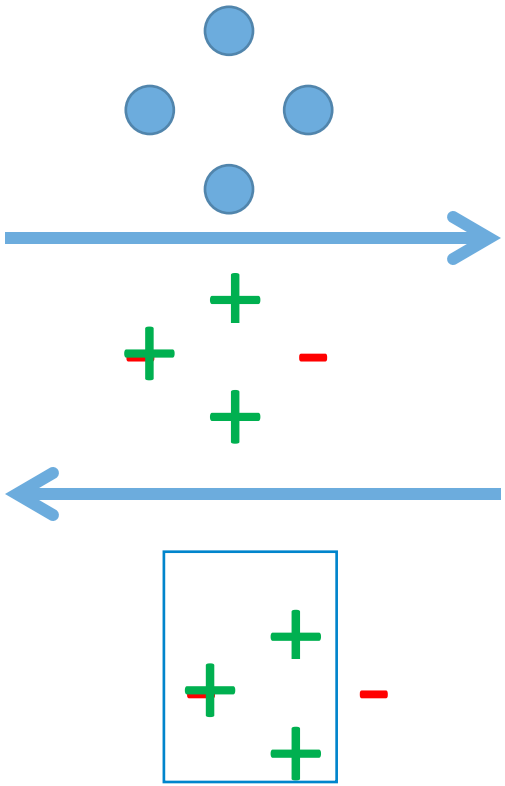
Precision

- Capture precision by *VC dimension* (VC-d)
- Higher precision -> More possible hypotheses

$VC(H)=d$

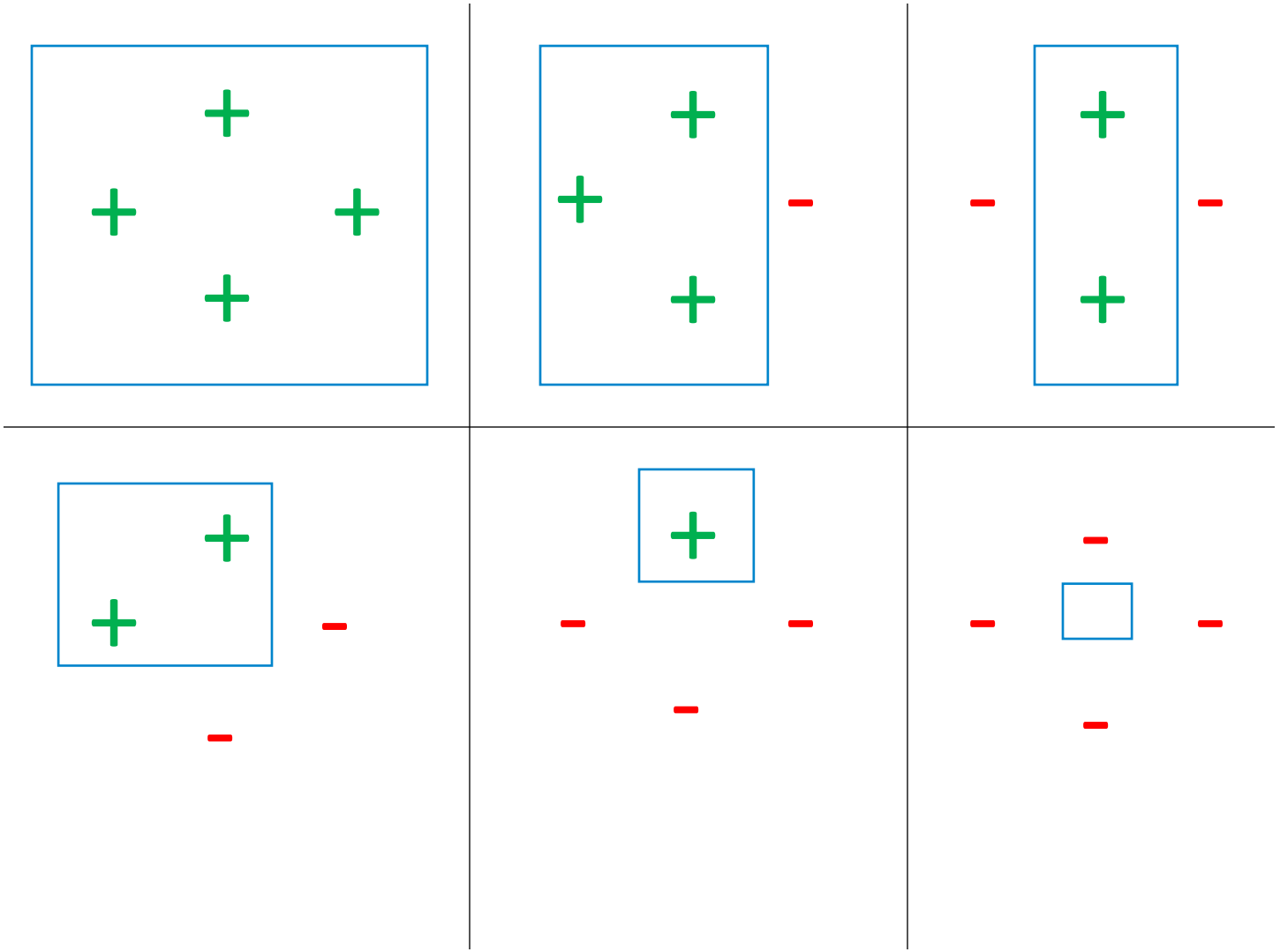


- i≤2,j≥5,…

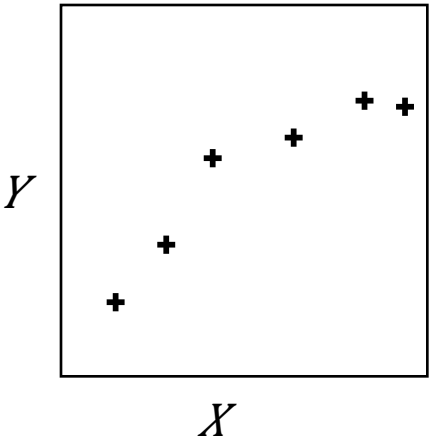


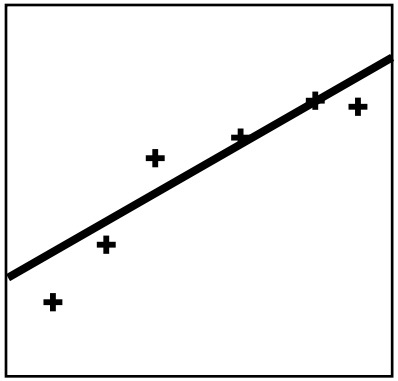
For any arbitrary labeling

VC-d Example



Regression Example



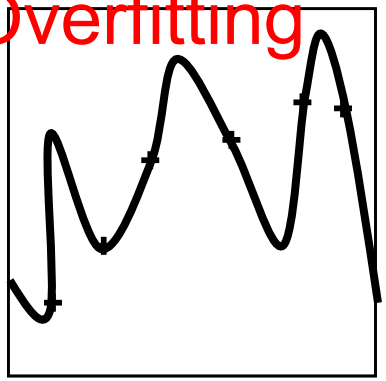


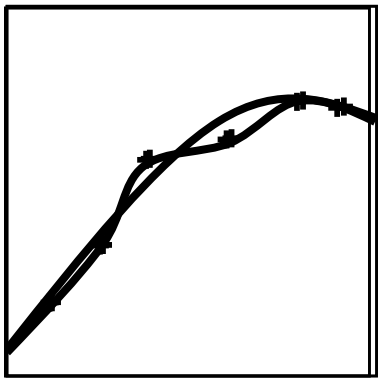
Precision is low

Underfitting

Precision is high

Overfitting

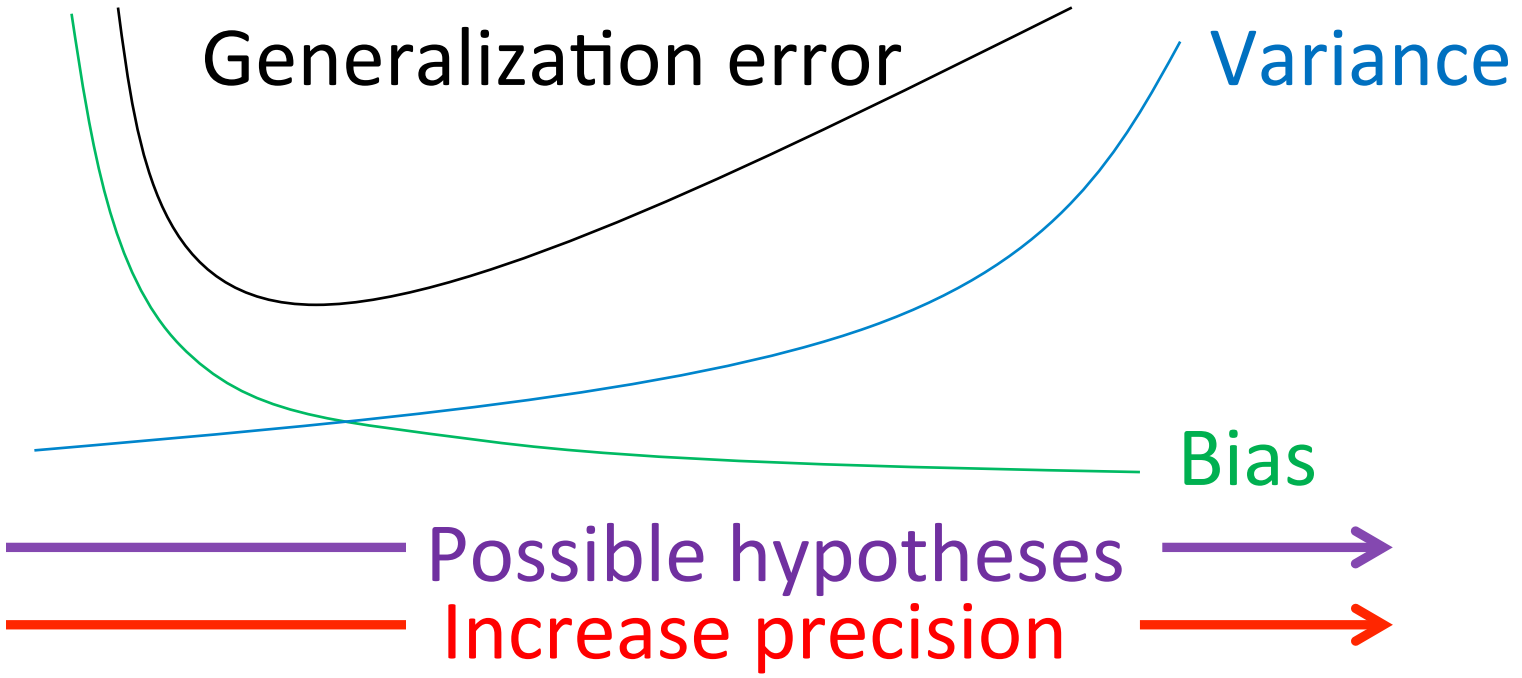




Good fit

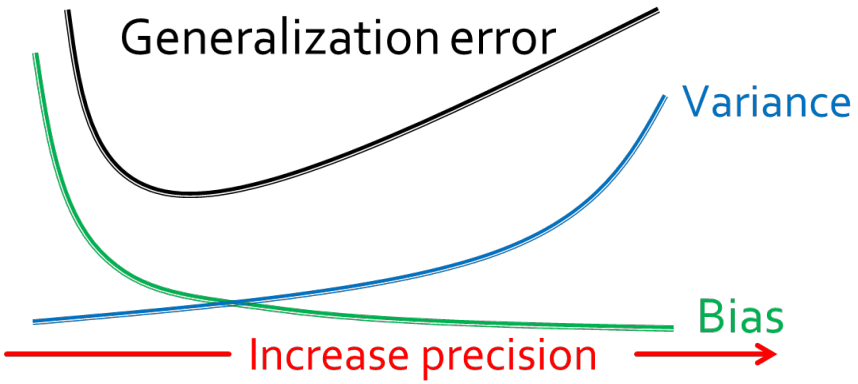
Main Result of PAC Framework

- Generalization error is bounded by sum of
  - *Bias*: Empirical error of best available hypothesis
  - *Variance*: $\mathcal{O}$(VC-d)



Example Revisited

```
int i = 1, j = 0;
while (i<=5) {
   j = j+i ;
   i = i+1;
}
```



Invariant inference

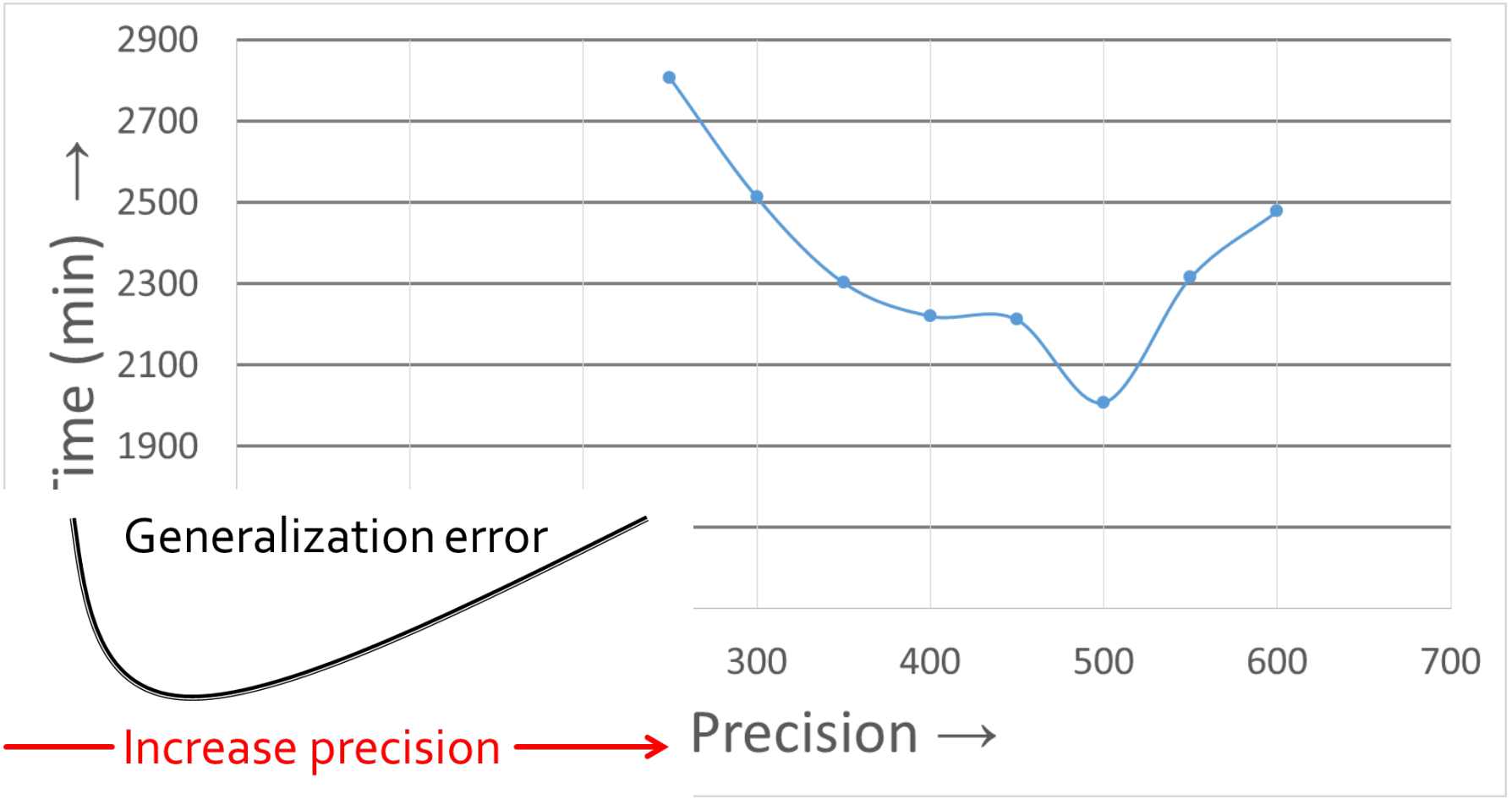
- Intervals
  - $2n$
- Octagons
  - $O(nlogn)$
- Polyhedra
  - ∞

Abstract Interpretation

Bench	PK/OCT			PK/BOX			OCT/BOX		
	$\subsetneq$	$\supsetneq$	unc.	$\subsetneq$	$\supsetneq$	unc.	$\subsetneq$	$\supsetneq$	unc.
a2ps	12.74	0.78	0	21.64	0	2.13	18.94	0	0.93
gawk	21.34	0	0	26.96	0	0	17.97	0	0
chess	5.99	5.78	2.47	12.67	3.68	2.24	14.87	0	0
gnugo	18.75	2.08	2.08	22.50	1.66	1.11	10.86	0	1.12
grep	3.30	0	0	8.26	0	0	8.26	0	0
gzip	21.16	2.18	0	32.84	0.72	1.45	26.27	0	0
lapack	11.84	5.67	0.85	78.96	2.16	2.99	85.03	0	0
make	6.50	4.00	5.50	6.52	4.34	5.97	11.94	0	0
tar	5.17	4.20	0	9.70	3.23	0.97	9.38	0	0

Henry, Monniaux, and Pagai: A path sensitive static analyser. *Electr. Notes Theor. Comput. Sci. 2012*.

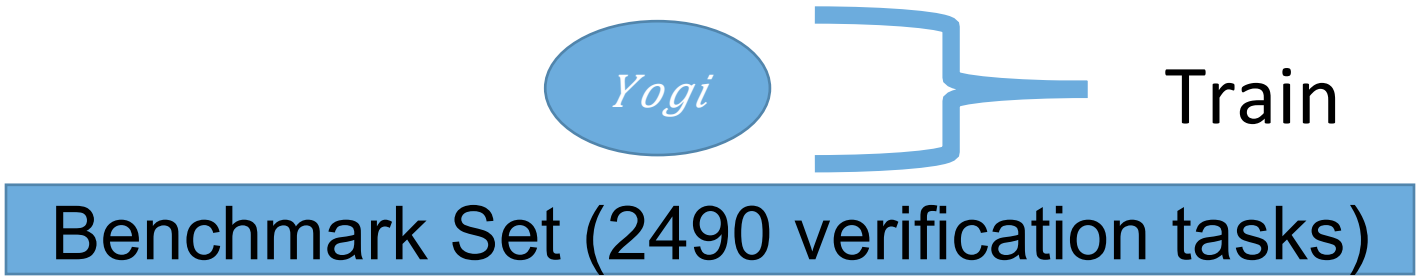
Yogi



Nori and Rajamani. An empirical study of optimizations in YOGI. *International Conference on Software Engineering* (*ICSE 2010*)

Case Study

- Parameter tuning for program analyses

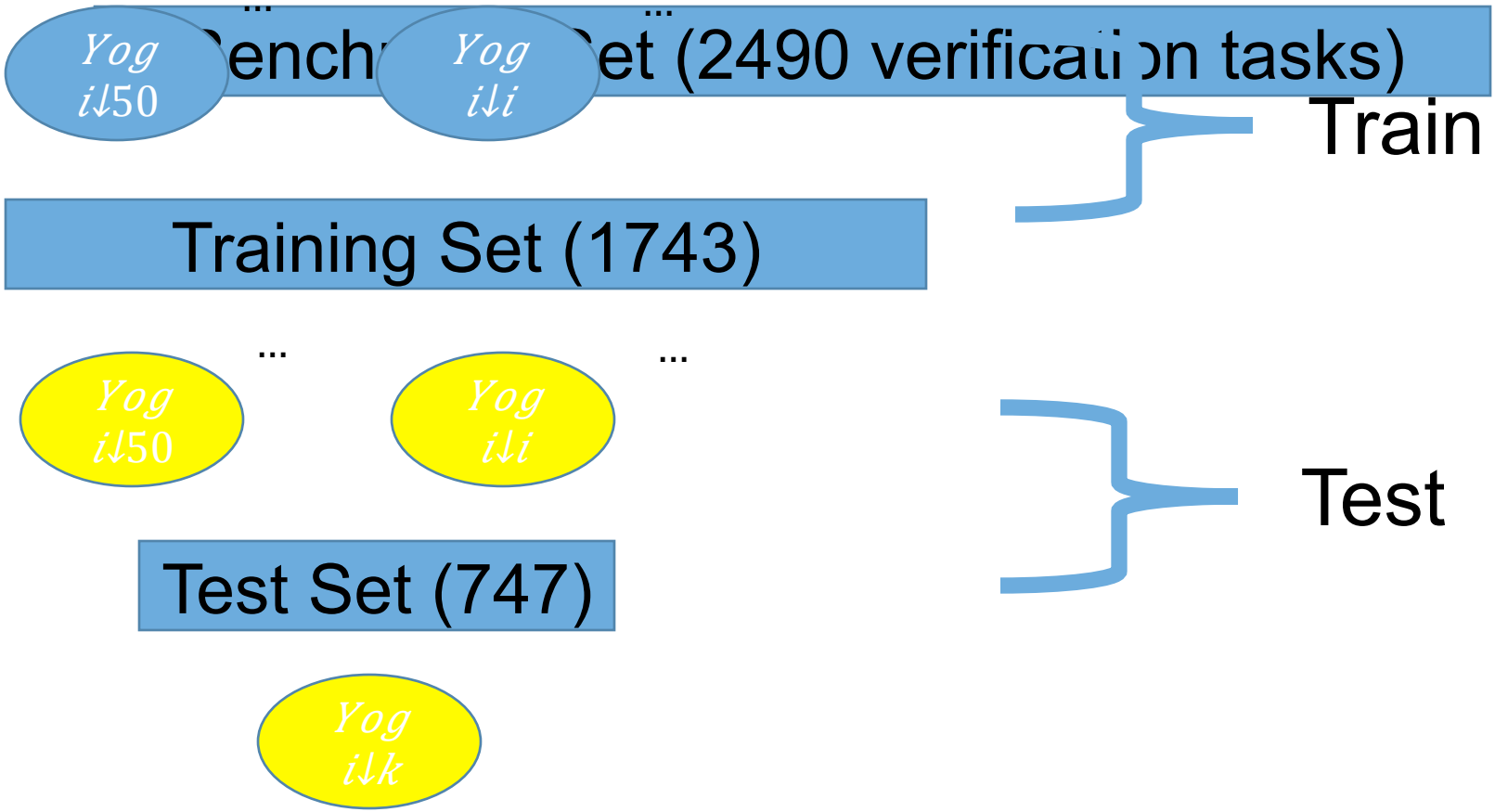


- Overfitting? Generalization on new tasks?

Godefroid, Nori, Rajamani, and Tetali. Compositional may-must program analysis: unleashing the power of alternation. *Principles of Programming Languages* (*POPL 2010)*

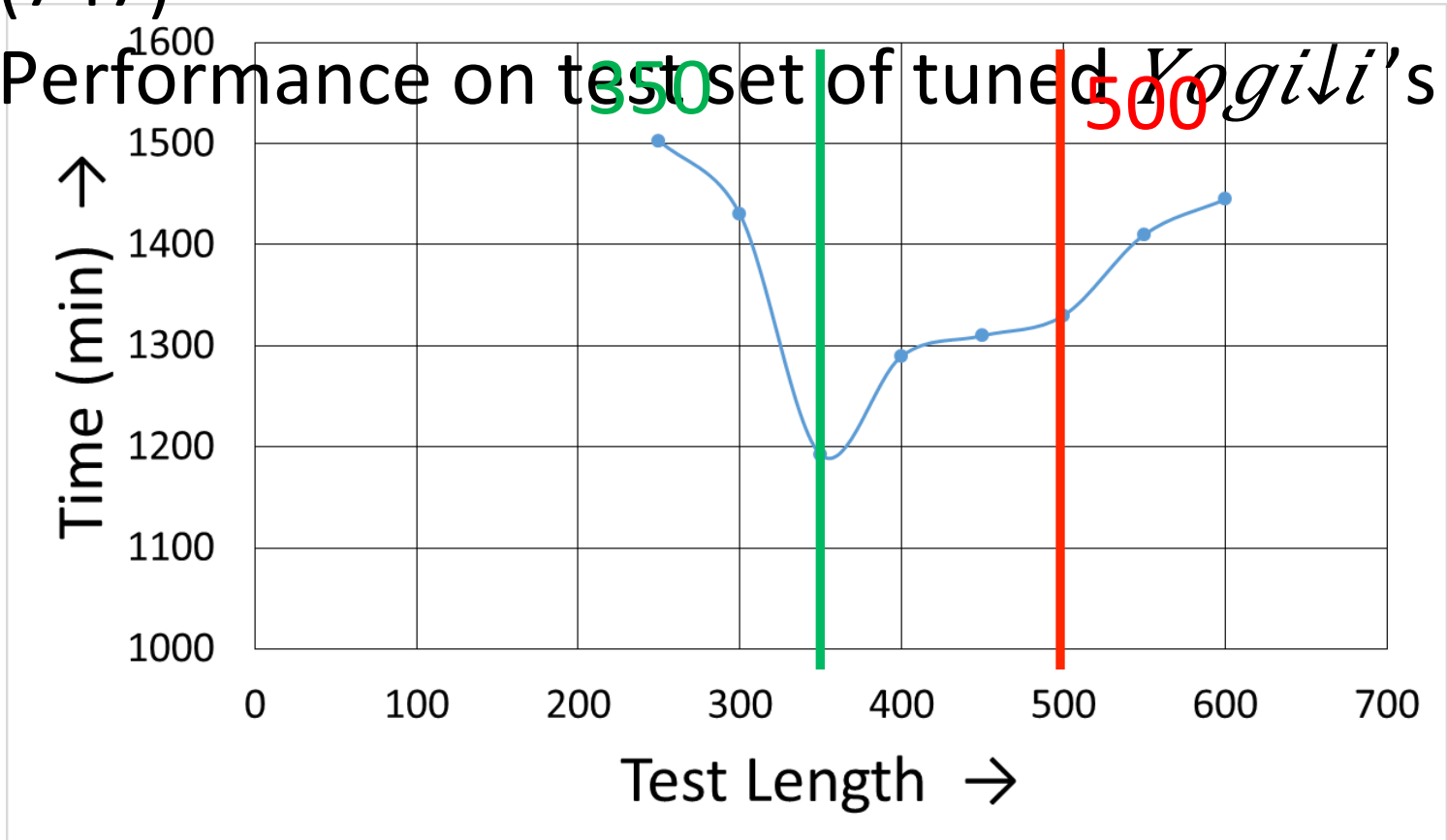
Cross Validation

- How to set the test length in Yogi?



Cross Validation on Yogi

- Total 2490 benchmarks: Train (1743), Test (747)
- Performance on test set of tuned *Yogili*'s



Comparison

- On 2106 new verification tasks

Tool	Time (min)	#time-outs	#defects
$Yogi_{350}$	4704	21	189
$Yogi_{500}$	8279	66	183

- 40% performance improvement!

Recommendations

- Keep separate training and test sets
  - Design of the tools governed by training set
  - Test set as a check

- SVCOMP: all benchmarks are public
  - Test tools on some new benchmarks too

More in:

Sharma, Nori, Aiken. Bias-Variance tradeoffs in program analysis. *Principles of Programming Languages (POPL 2014)*

- VC-d of TCMs: intervals, octagons, etc.
- Templates: $P\downarrow 1 \vee P\downarrow 2$
- Arrays, separation logic
- Expressive abstract domains -> higher VC-d
- VC-d can help choose abstractions

Summary

- Guess-and-Check data-driven approach to program verification
- Simple machine learning algorithms used to solve hard verification problems
- **Data Driven Program Analysis**
  - A model to understand generalization: Bias-Variance tradeoffs
  - Understand these tradeoffs for better tools