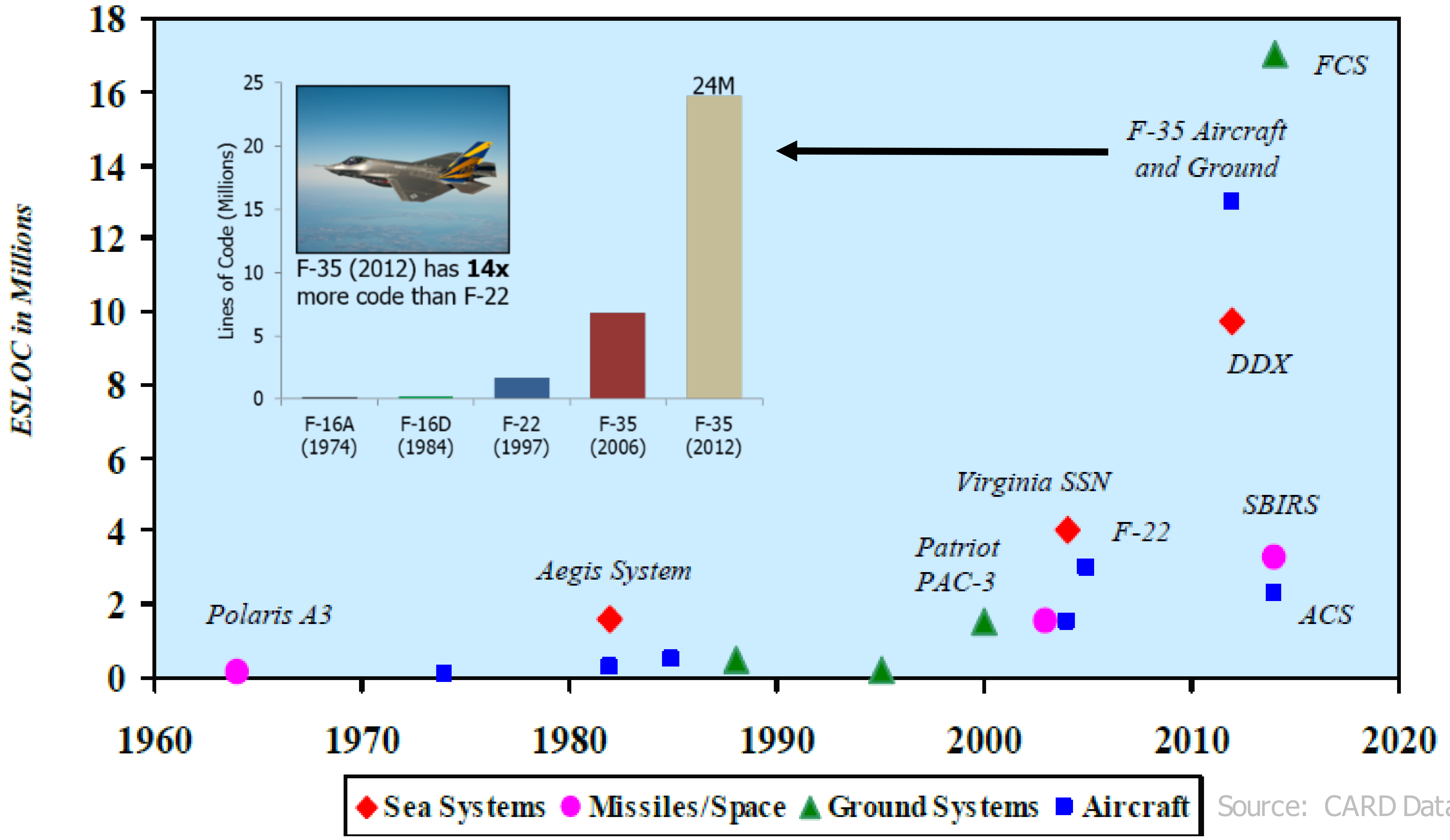
Learning for Automated Synthesis and Verification

Suresh Jagannathan

PURDUE
UNIVERSITY

The Context

Software Content of Sample Major DoD Weapon Systems 1960 - 2020



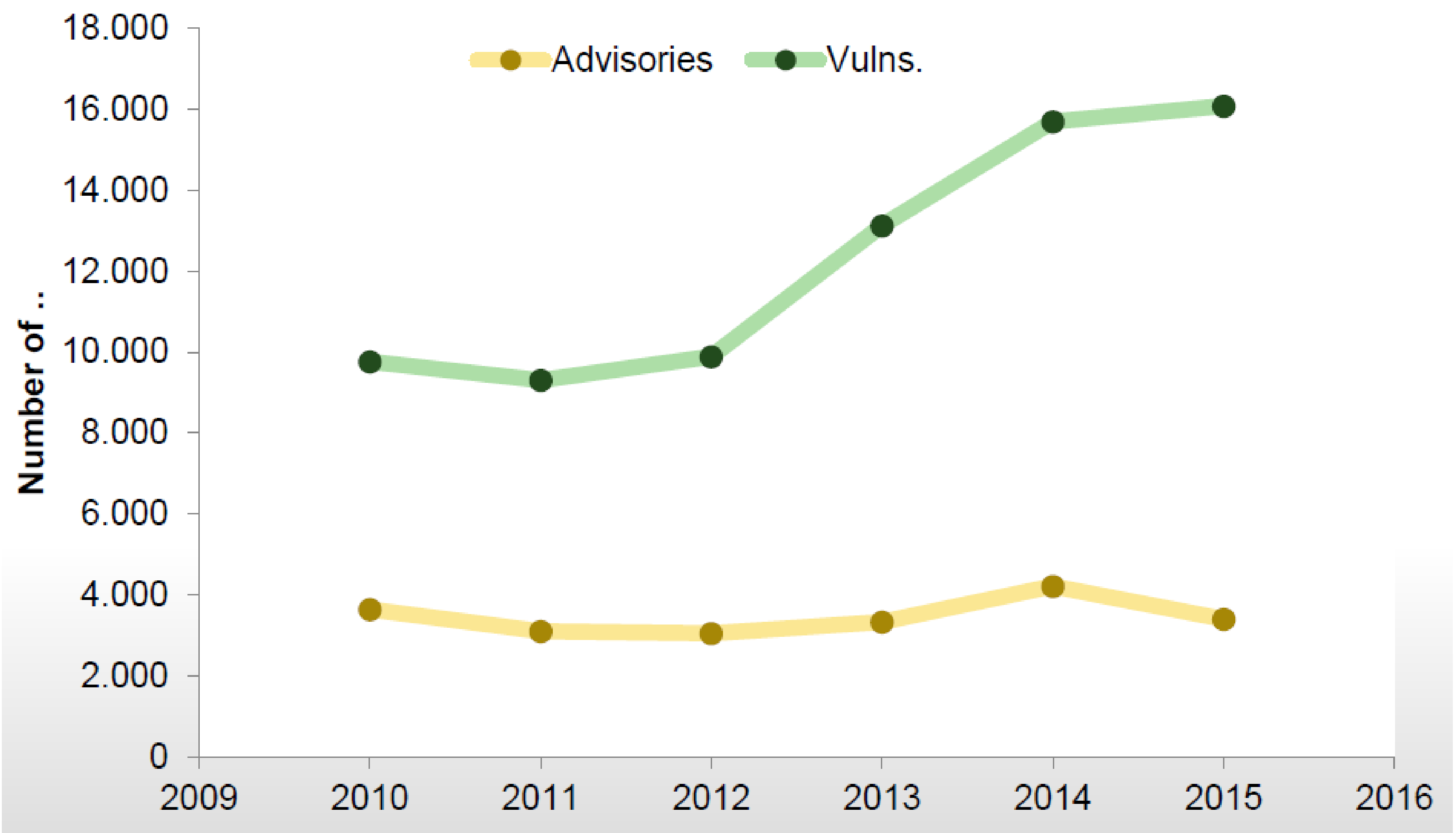
Source: CARD Data, SEI, CSIS Analysis

"Our big issue is software …"

William Roper, Air Force Undersecretary for Acquisitions

"… the services have largely failed to take advantage of an emerging "software-defined world."

Bill Chappel, Director MTO, DARPA



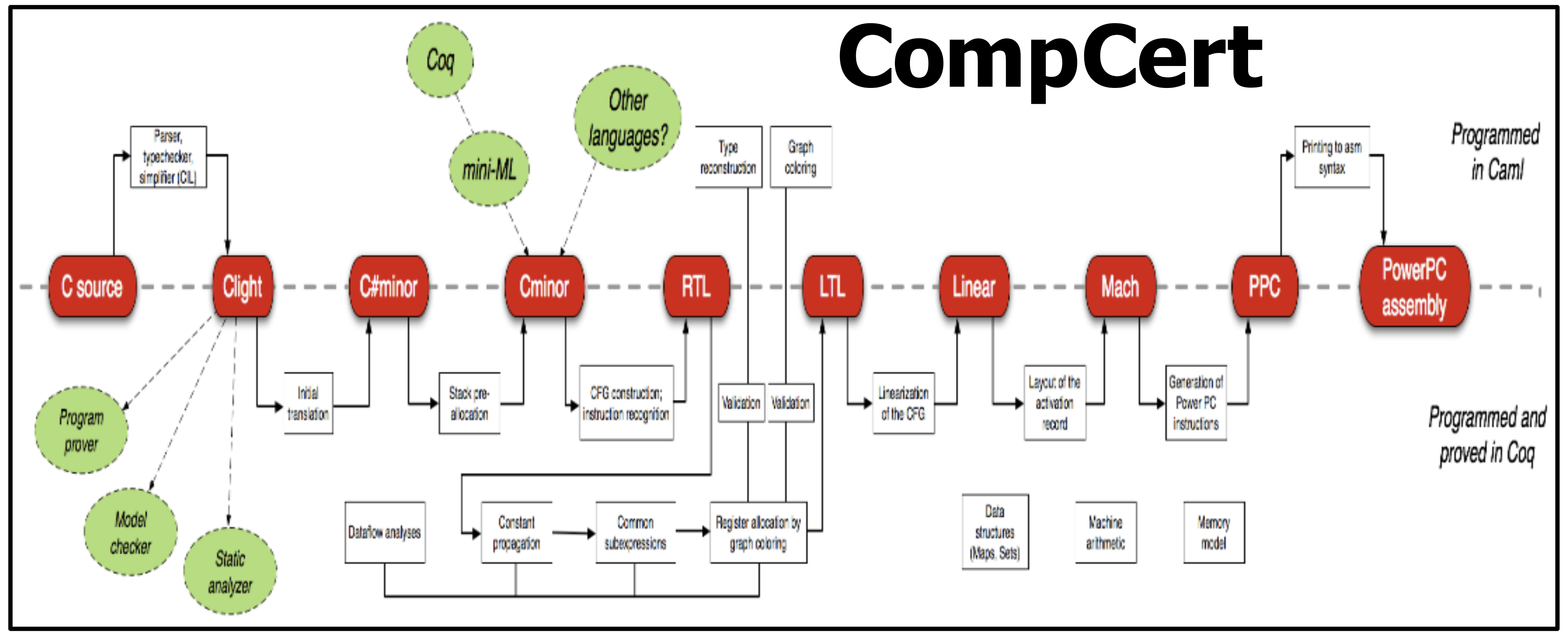
Vulnerability History

- 2,484 applications from 263 vendors
- 40% Microsoft applications and 60% non-Microsoft applications in Windows operating systems

Nearly every U.S. weapons program tested in fiscal 2014 showed "significant vulnerabilities" to cyber attacks, including misconfigured, unpatched and outdated software.

Pentagon Chief Weapon's Tester Report

Trustworthiness through First Principles



8KLOC, 50K LoP

8.5KLOC, 200K LoP

HACMS quadcopter

100K LoC; 30K LoP

(Multi-million dollar effort over 5 years)

… demonstrates feasibility of defining provably correct mission-critical kernels, but what about systems at scale?

Strategies

HACMS

MUSE

Trends

>2.1M repositories

260K projects

250 projects

108K repositories

>10M LoC
(open source)

9.5K projects

28.5K projects

Commits on Mar 30, 2016

Merge changes Iaa88ac02,Ic6f8d1da,Idd07db5d
miaowang14 committed with Gerrit Code Review 14 hours ago

[RenderScript] Update documentation for Allocation.setAutoPadding().
miaowang14 committed 16 hours ago

[RenderScript] Fix IntrinsicConvolve documentation.
miaowang14 committed 17 hours ago

[RenderScript] Update the documentation of Script.LaunchOptions
miaowang14 committed 17 hours ago

30K projects

250K projects

The Central Repository

Navy's newest warship (USS Zumwalt) runs on Linux

35% of all DoD systems use open source software

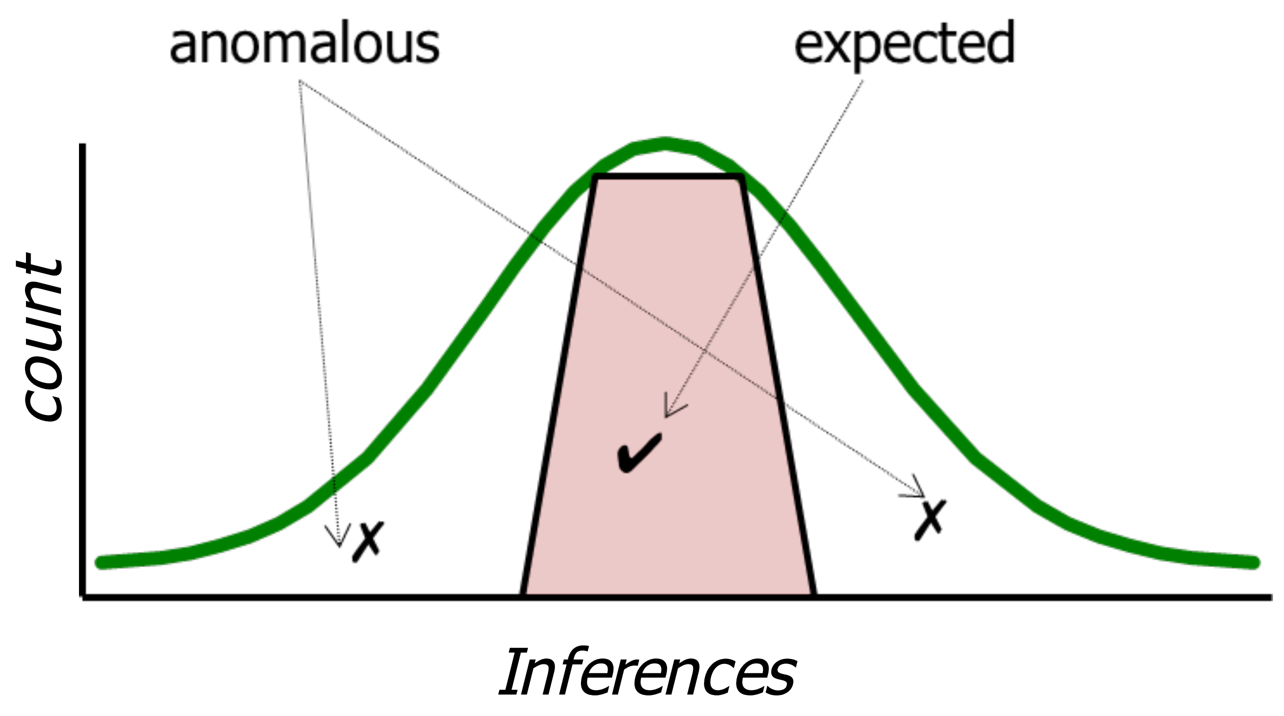
The Pentagon is set to make a big push toward open source software next year

The Pentagon is a software-intensive workplace

The Idea

Data Stream
and Signals
Programs

Observations

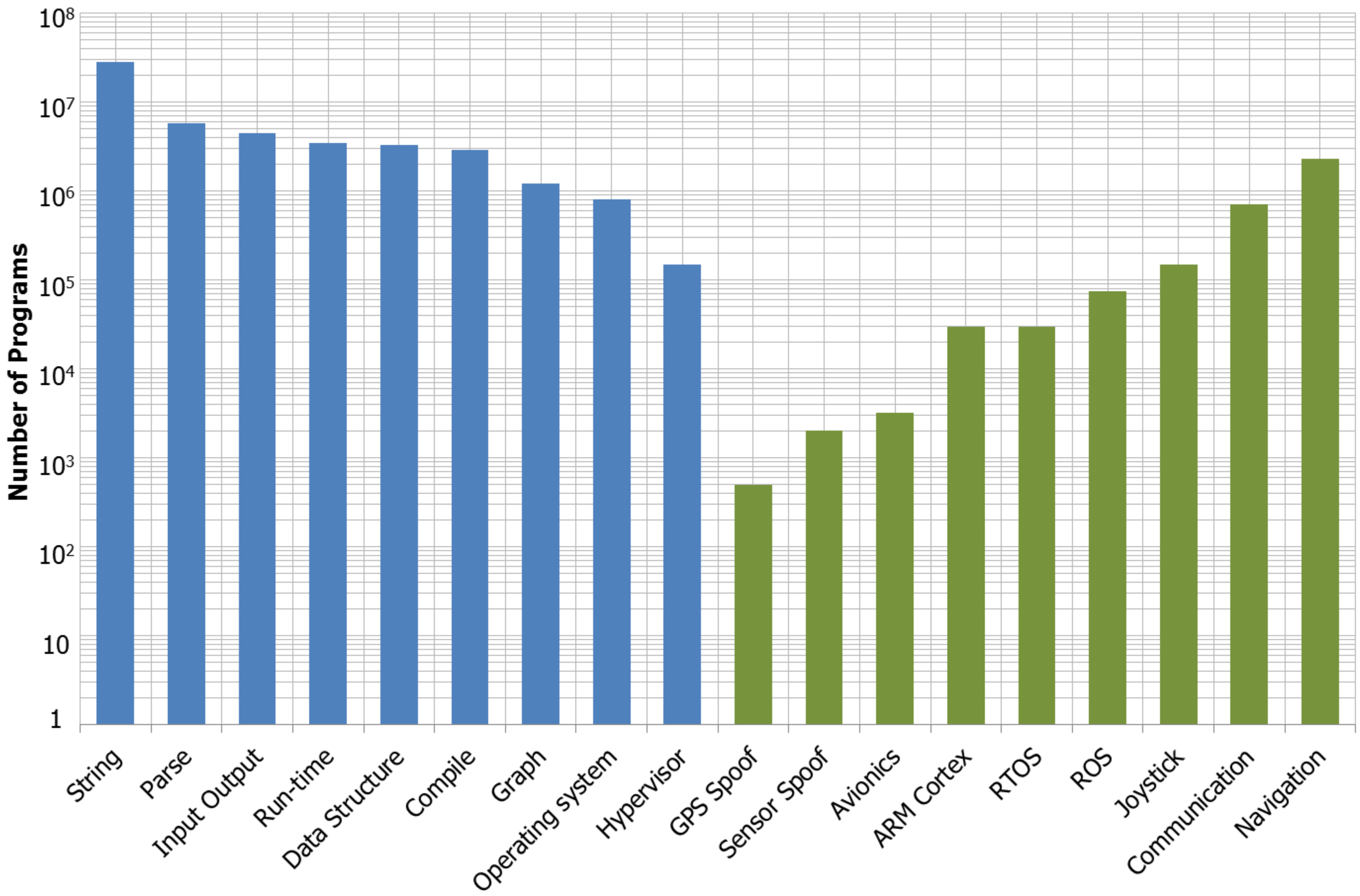


Program Properties,
Behaviors, and Vulnerabilities

- Treat programs (more precisely, semantic objects extracted from programs) as data
- Observations and inferences applied to program properties

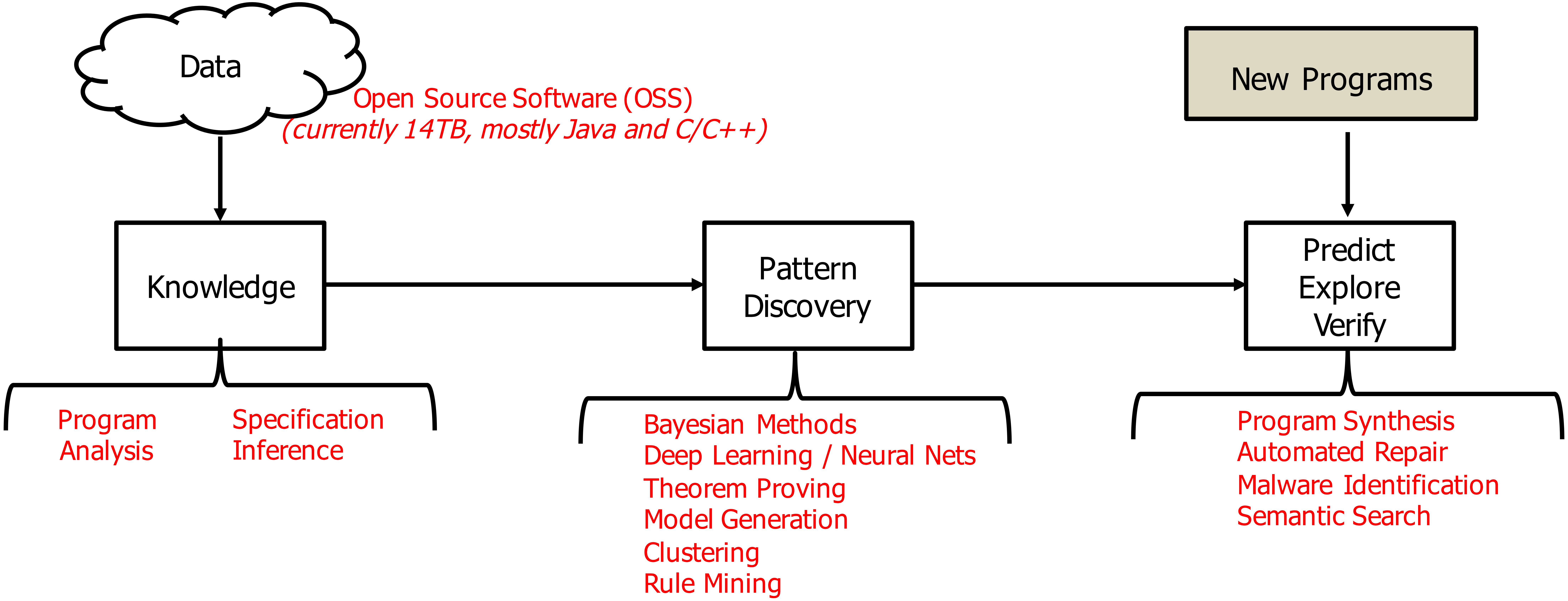
Supervised setting: program repair
Semi- or Un-supervised setting: program synthesis

Rationale

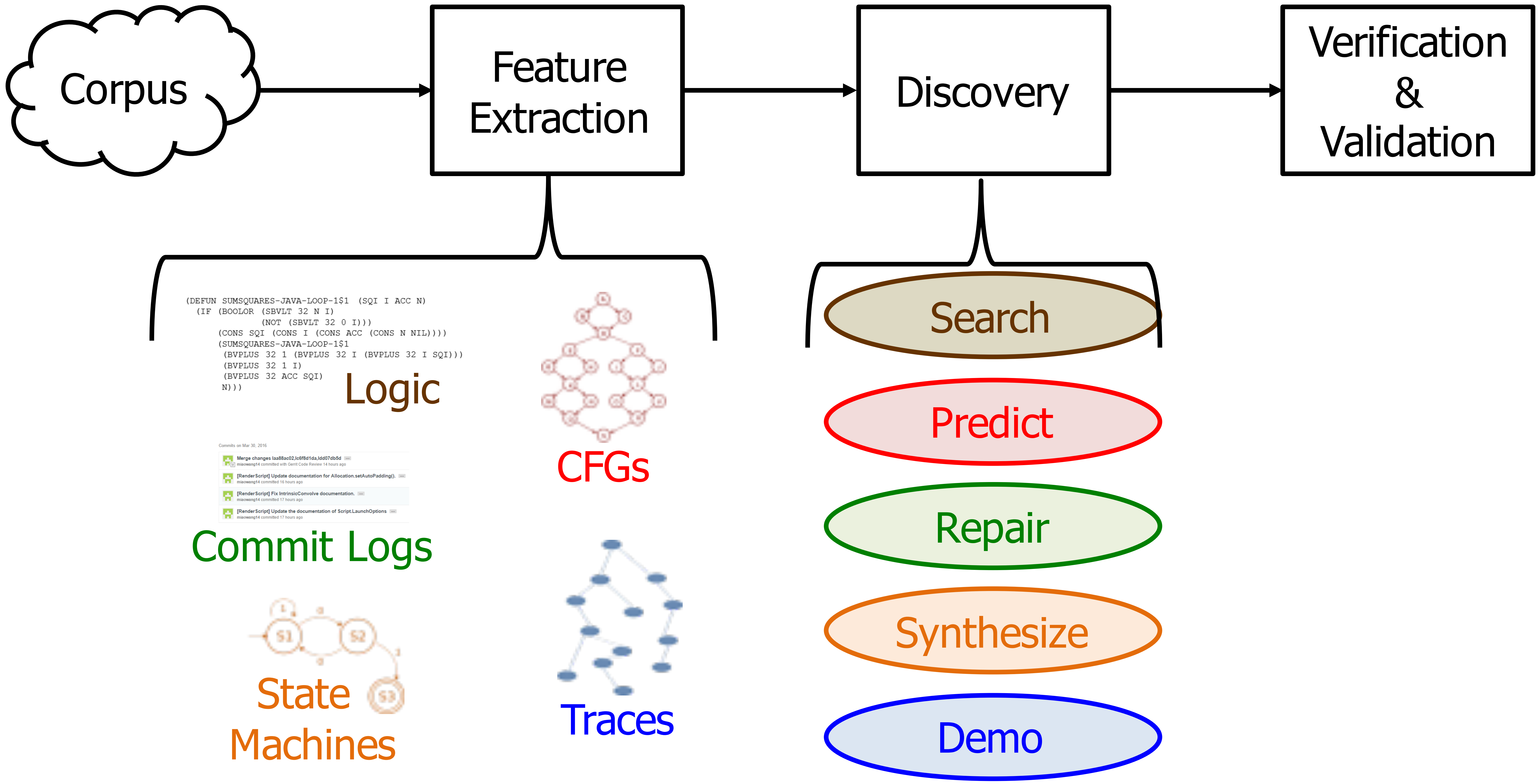


Number of Programs
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10
1
String
Parse
Input Output
Run-time
Data Structure
Compile
Graph
Operating system
Hypervisor
GPS Spoof
Sensor Spoof
Avionics
ARM Cortex
RTOS
ROS
Joystick
Communication
Navigation

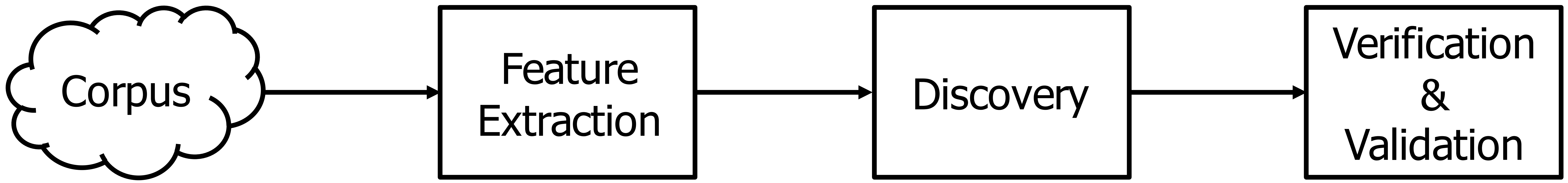
MUSE - Mining and Understanding Software Enclaves

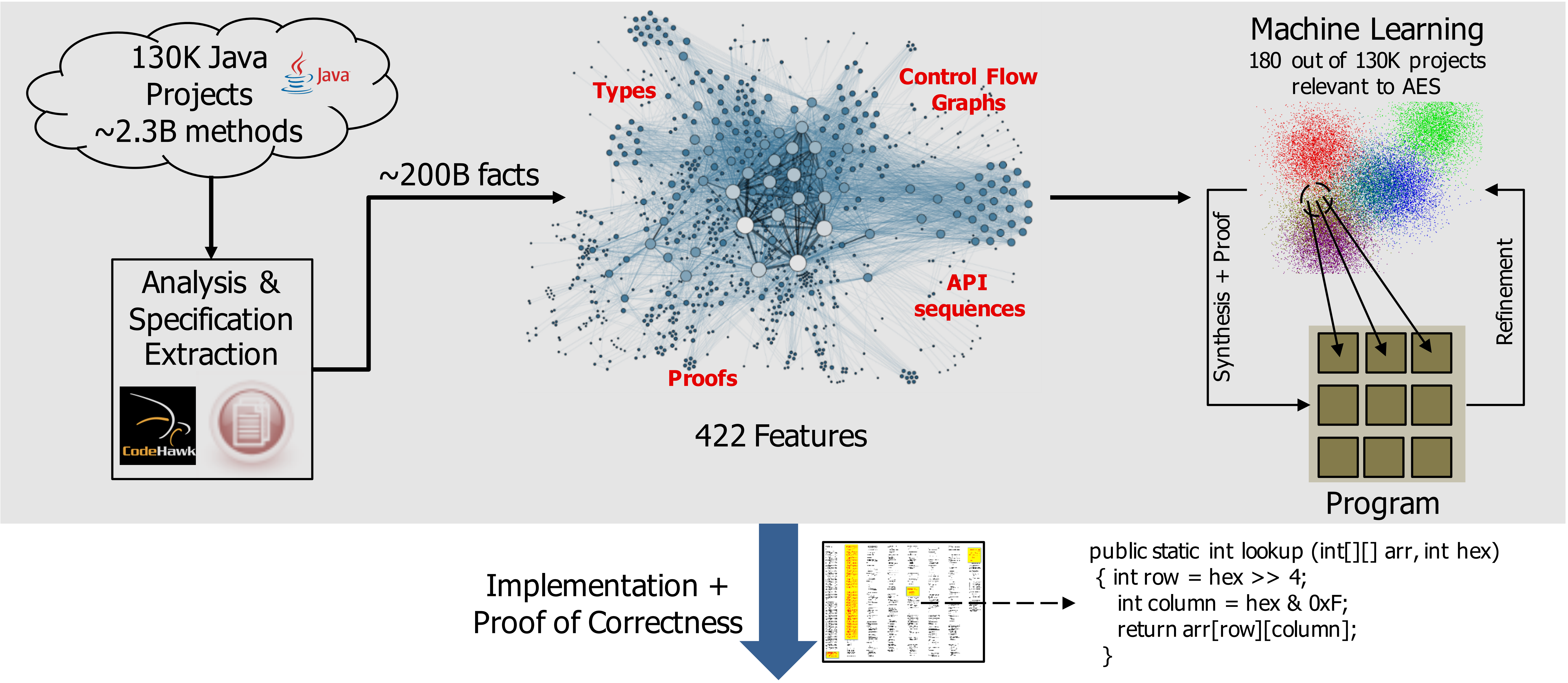


Structure


Corpus
Feature Extraction
Discovery
Verification & Validation
Logic
CFGs
Commit Logs
State Machines
Traces
Search
Predict
Repair
Synthesize
Demo

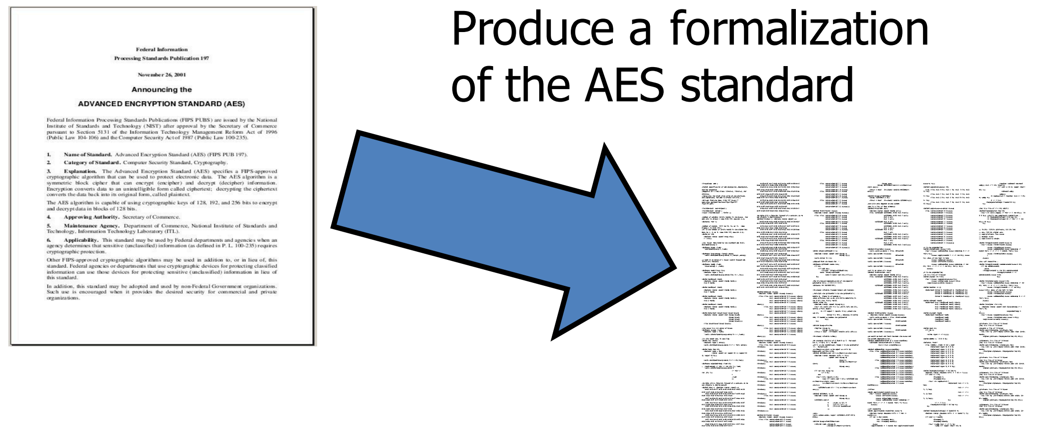
Specification-Driven Synthesis Using Big Code





Analysis

Specification



Produce a formalization of the AES standard

```
...
(defun aes-128-encrypt (plaintext key)
  (let ((nk 4)) ; four 32-bit words
    (cipher plaintext (keyexpansion key nk) nk)))
...
```

Refinement

SmallK - dimensionality reduction via non-negative matrix factorization

bytecode

Static Analysis Via Abstract Interpretation

few (small) methods; expensive analysis

invariants:

i<arraylength(a)
j >= minX + 1
....

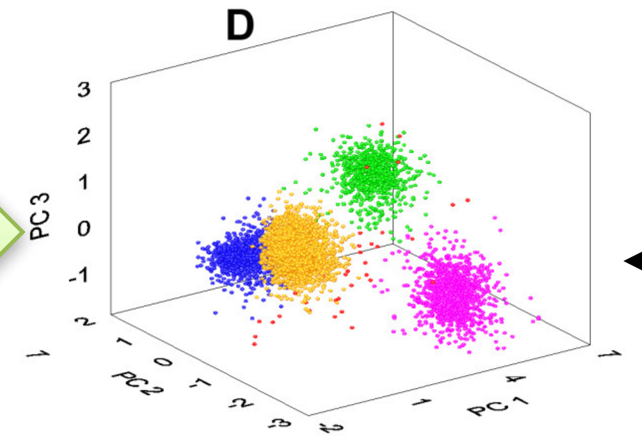
LIFTER

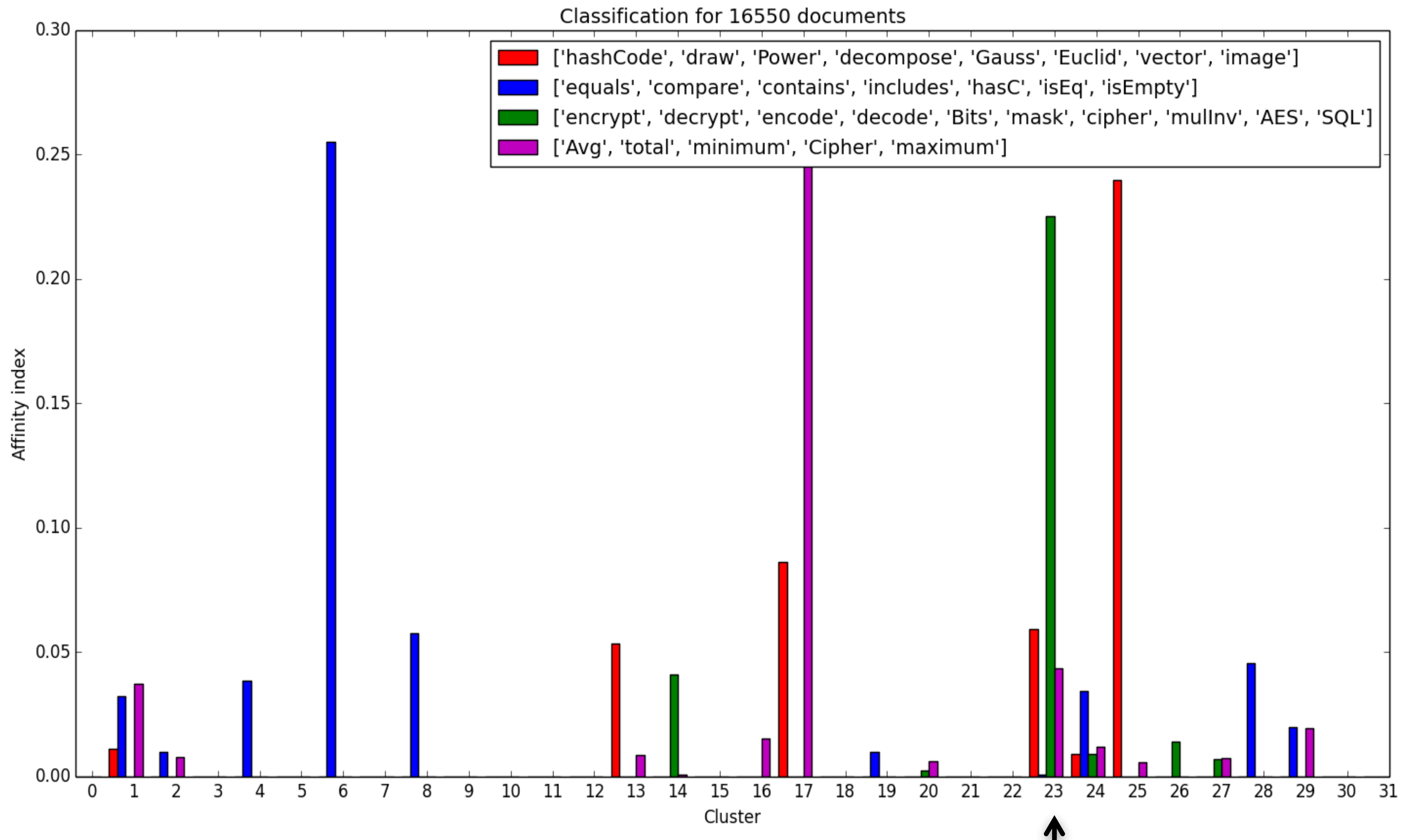
100 million methods cheap analysis

> 400 feature classes

features:

bc frequencies
k-subgraphs
library calls
....





Cluster 23 seems to contains a lot of crypto code.

Verification

Proved correct for 128/192/256-bit encryption/decryption.

"Correct" means the ciphertext bits exactly match the formal spec, for all 2^{256} possible inputs (key + plaintext).

Turn spec and implementation into mathematical terms

- Symbolically execute code (2,880 JVM instructions, 2.2M simplifications, 13 seconds)
- Unroll recursion in spec
- Unrolled code is ~100 billion nodes (spec is ~10^26)
- Must share common subterms!

Apply semantic equivalence checker to prove correspondence

Synthesis

Synthesize novel implementations using pieces from the corpus

Example: replace the Galois Field operation in one AES implementation (2abc6c6a-f84a-4e54-9da9-6a13ef25b9d8) with an optimized version from another project (dcee2208-f0ec-4796-adb5-9bde1d25c07f)

- Mix and match to get best performance
- Prove the resulting hybrid implementation correct

SynthHorn: A Data-Driven ~~CHC~~ ~~Solver~~

Invariant Inference

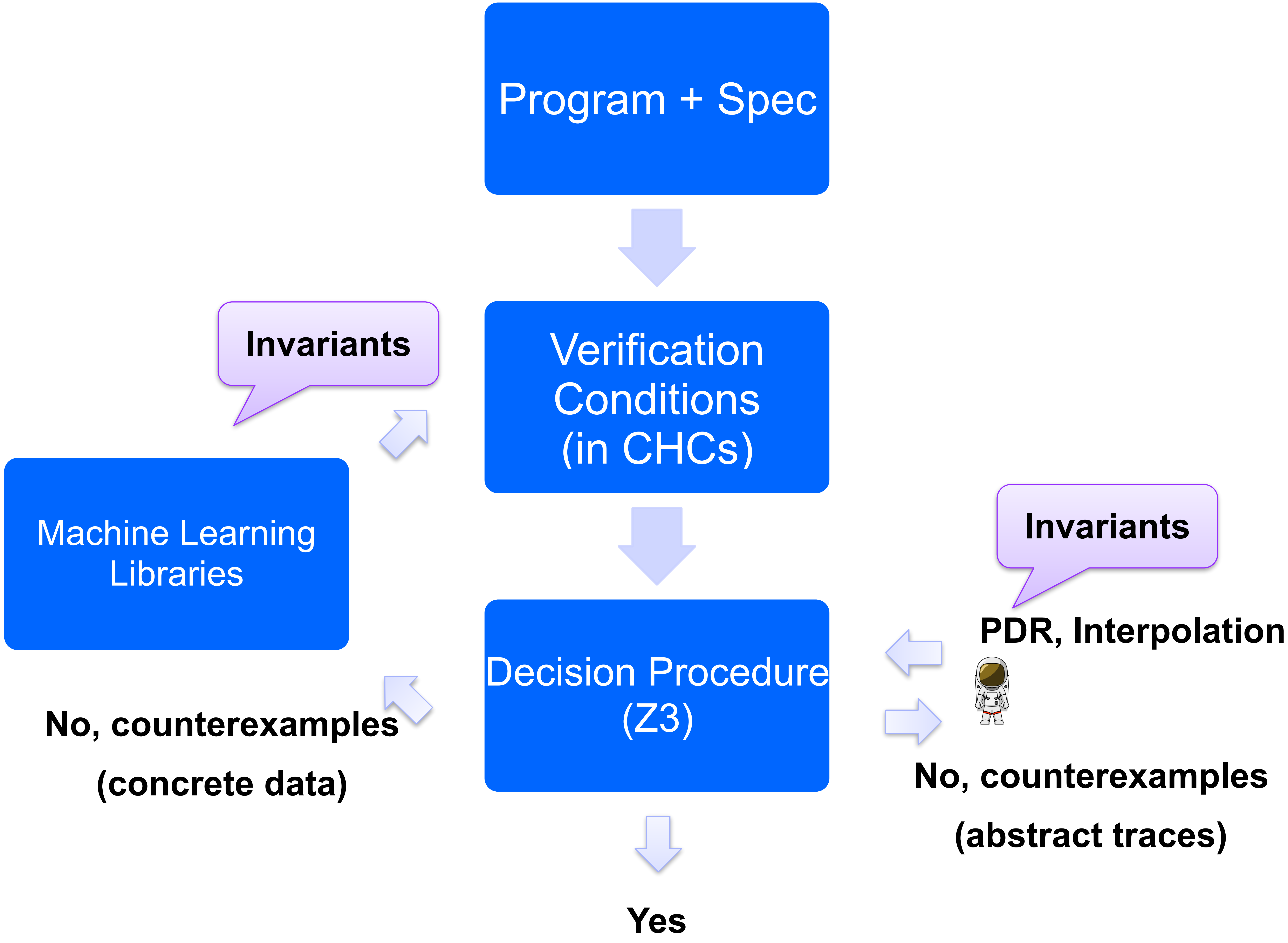
Verification condition (VC)

Joint work with He Zhu and Stephen Magill

galois

PURDUE UNIVERSITY

Automatic Program Verification



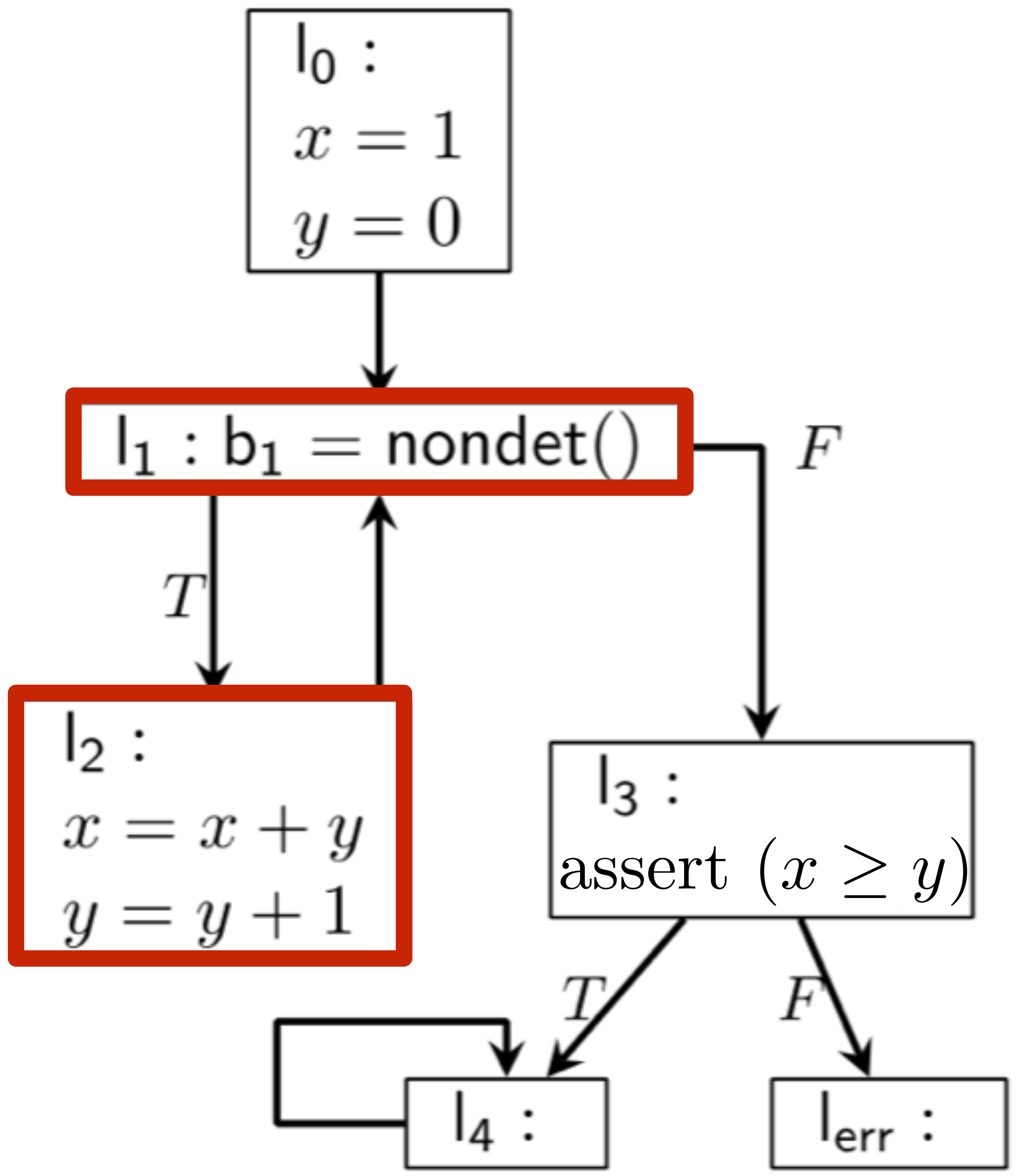
The Context

Program

```
main() {
 int x = 1;
 int y = 0;
 while (*) {
  x = x + y;
  y = y + 1;
 }
 assert (x >= y)
}
```

$p(x, y)$

CFG



VCs

$x = 1 \wedge y = 0 \rightarrow p(x, y)$ Induction

$\boxed{p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow p(x', y')}$

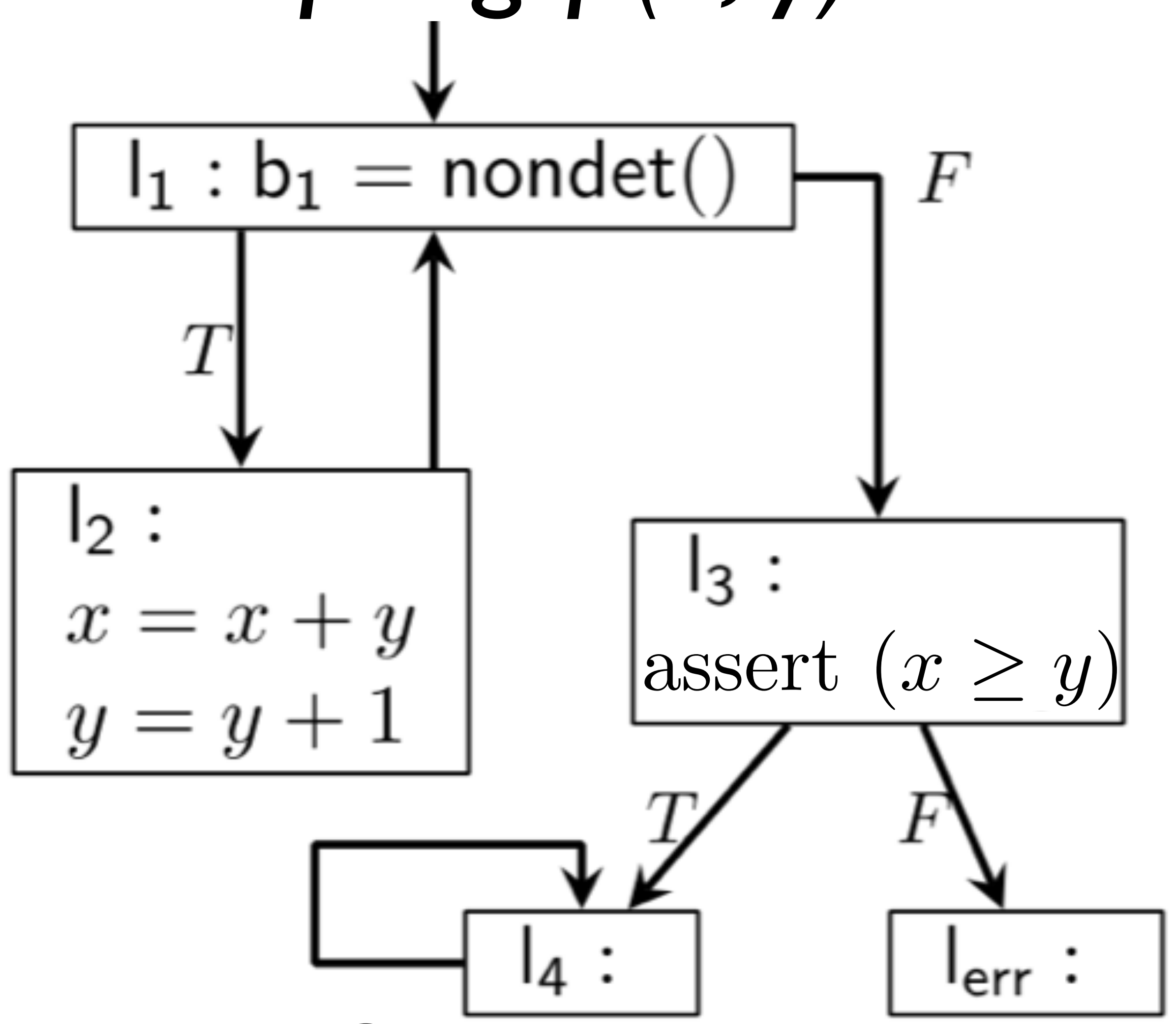
$p(x, y) \wedge x' = x + y \wedge y' = y + 1 \rightarrow x' >= y'$

$x = 1 \wedge y = 0 \rightarrow x >= y$

Spacer fails in this particular case

Data-Driven Invariant Inference

Sampling p(x, y)



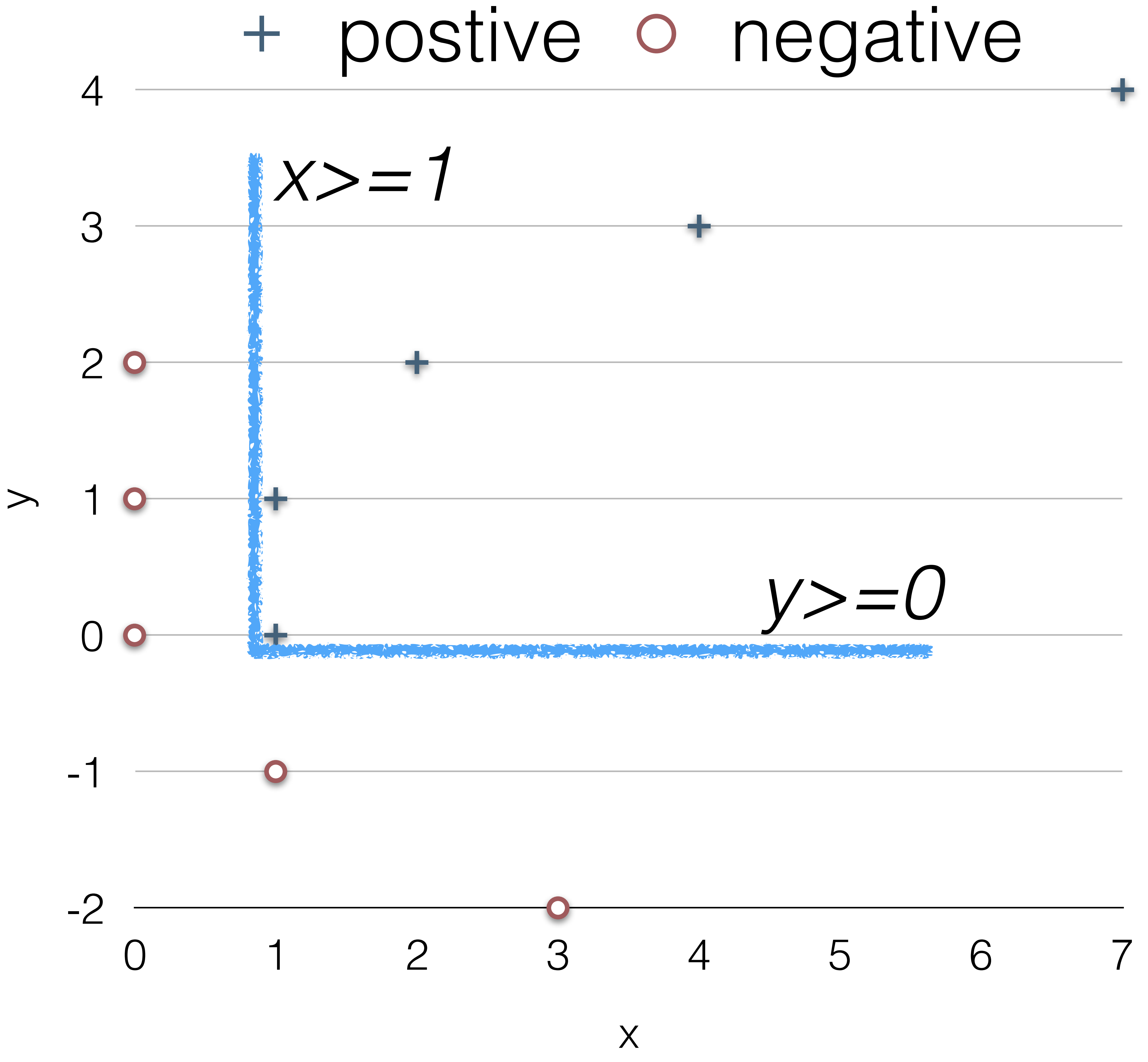
p(1,0), p(1,1), …

p(0,1) p(0,2), …

Ask Z3 *positive* *negative*

classification

$$p(x, y) \equiv \{x >= 1 \wedge y >= 0\}$$

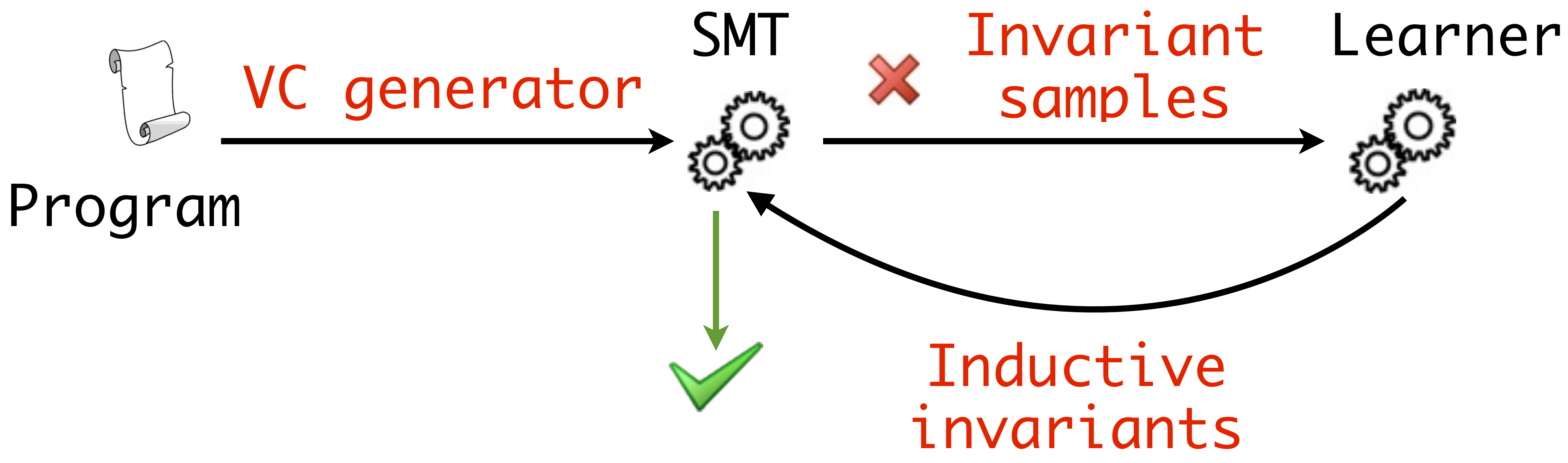


Data-Driven Invariant Inference

<u>Vision</u>:
An inductive invariant can be discovered from data

Goal: Design a learner to learn *inductive* invariants from data

SynthHorn work flow:



Hypothesis Domain

A *Machine Learning* Technique for invariants of *arbitrary Boolean* combination of *arbitrary linear arithmetic* predicates.

$$\bigvee_i \bigwedge_j \mathbf{w}_{ij}^T \cdot \mathbf{x}_{ij} + b_{ij}$$

can potentially support other domains (e.g. heap)

Learning Invariants from Data ...

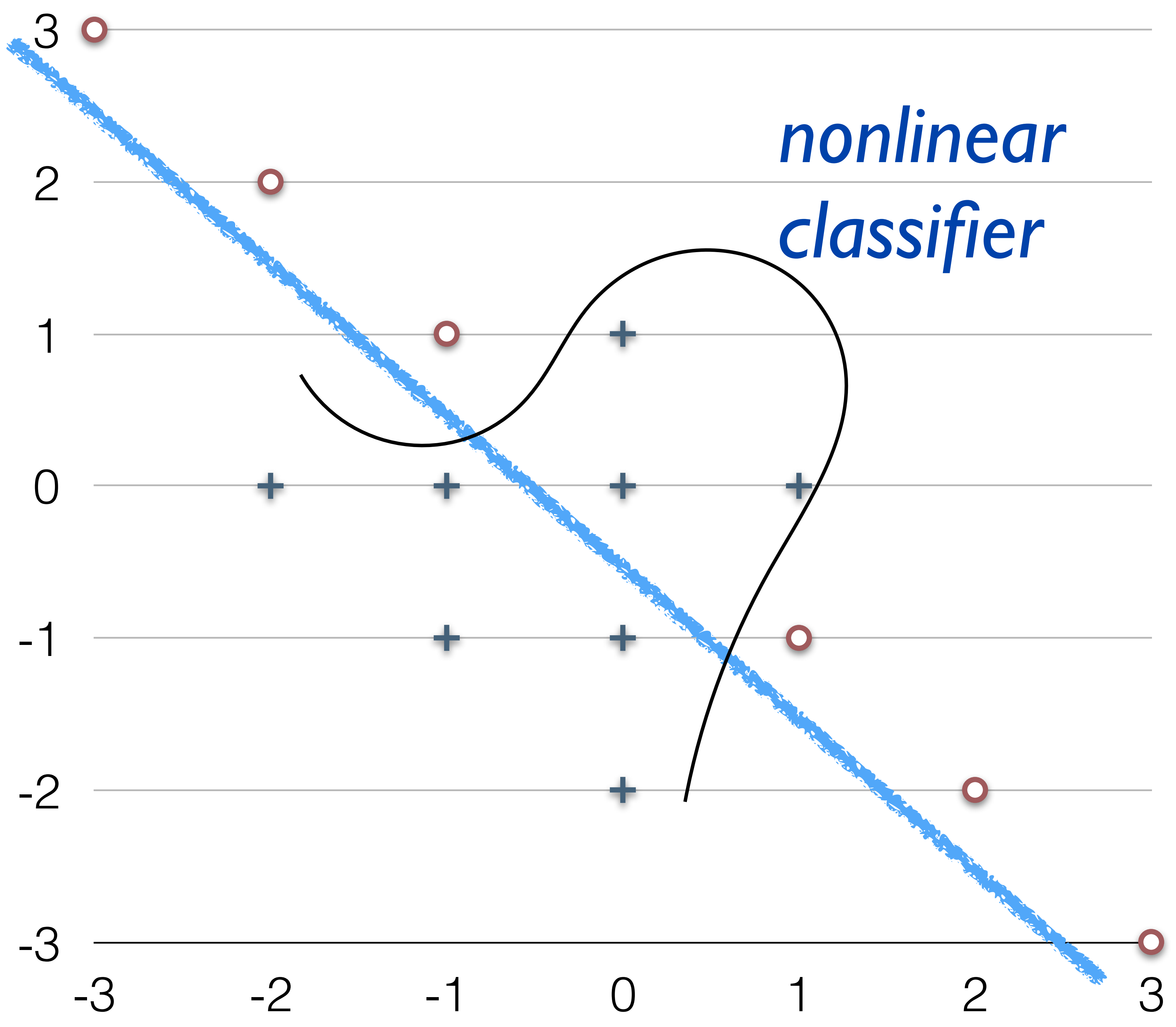
```
main() {
 int x,  y;
 x = 0;  y = *;
 while (y != 0) { // p(x,y)
  if (y < 0)  {x--;  y++;}
  else  {x++;  y—;}
  assert  (x != 0);
 }
}
```

Sampling p(x, y)

p(1,0) p(1,1) p(2, 2) p(4,3) p(7,4)

⟶

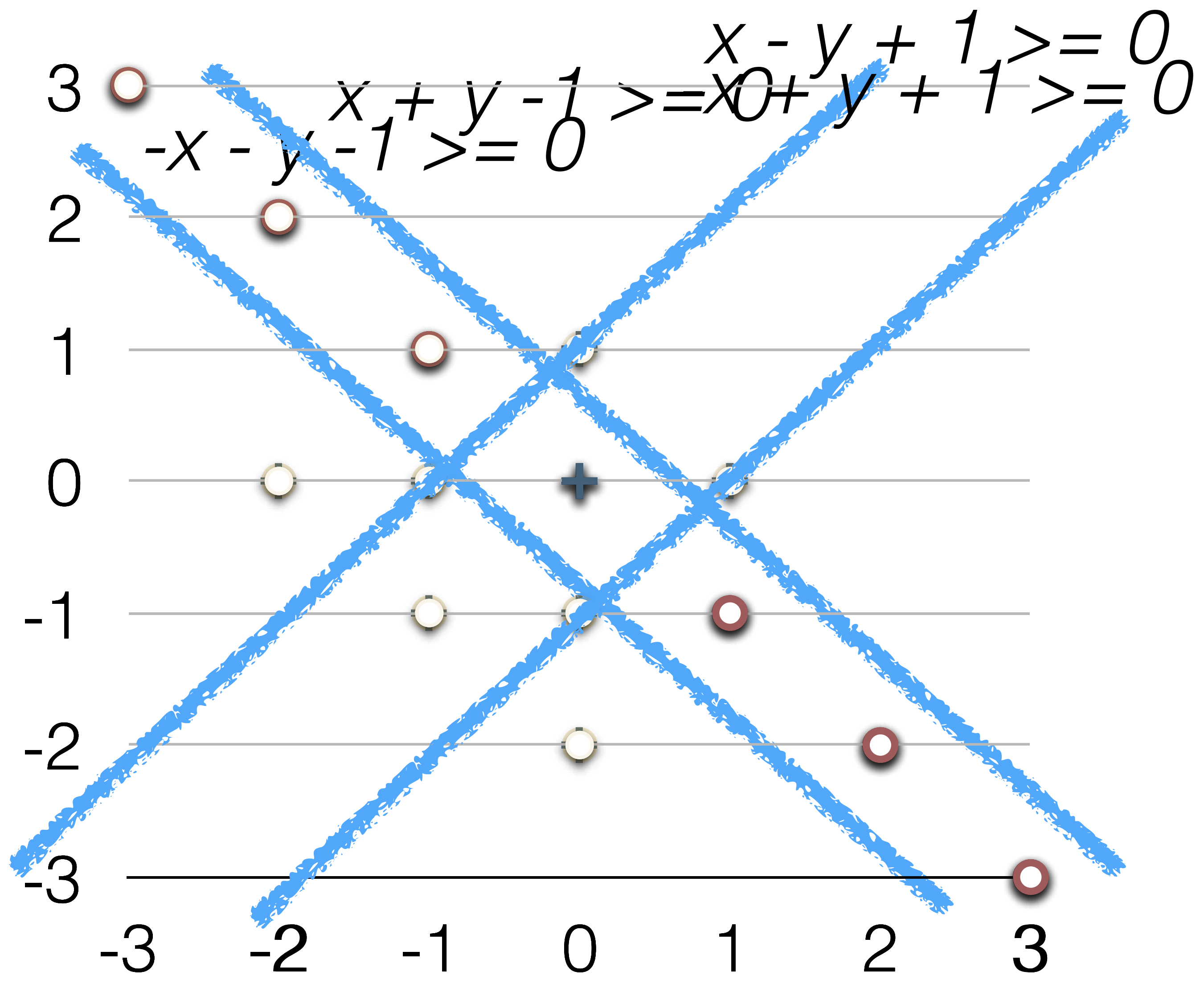
p(3,-2) p(1,-1) p(0,0) p(0,1) p(0,2)



- First take: use linear classification (SVM, Perceptron, Logistic Regression).
- But, there is a tension between Machine Learning and Verification: Generality vs. Safety.

Learning Arbitrarily Shaped Invariants ...

Given the data,



$-x - y - 1 \geq 0 \vee x + y - 1 \geq 0$

$-x - y - 1 \geq 0 \vee x - y + 1 \geq 0 \wedge -x + y + 1 \geq 0 \wedge x + y - 1 \geq 0$

- Generality: Call linear classification by leveraging its ability to infer high quality classifiers even from data that are not linearly separable.

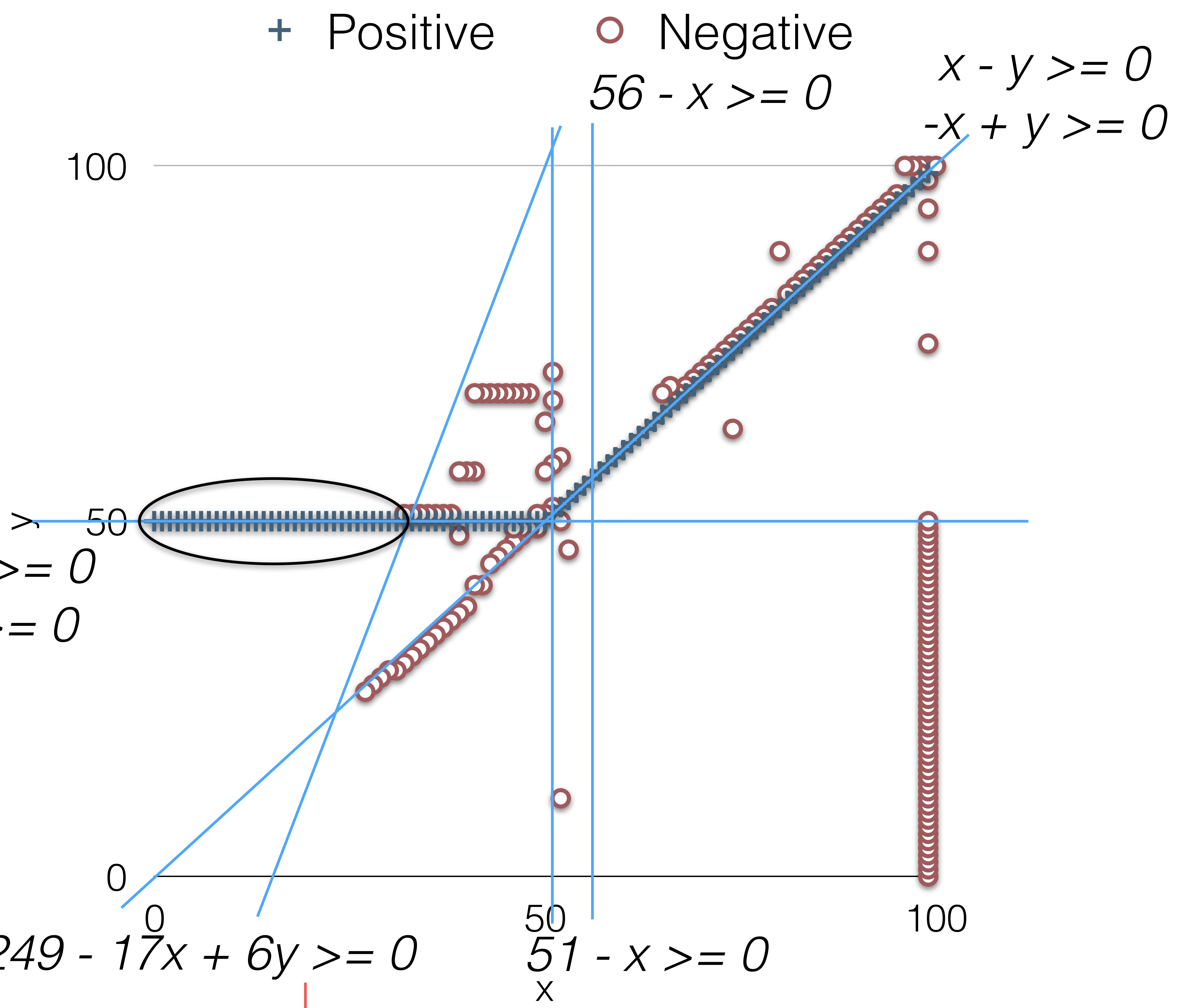
- Safety: Call linear classification recursively until all samples are correctly separated.

- SynthHorn: Combine Generality and Safety together!

Combating Over- and Under-fitting

```
main() {
 int x,  y;
 x = 0;  y = 50;
 while (x < 100) { // p(x,y)
  x = x + 1;
  if (x > 50)  {y = y + 1;}
 }
 assert  (y == 100);
}
```

Sampling p(x, y)



$$\boxed{56 - x \geq 0} \wedge (\boxed{249 - 17x + 6y \geq 0} \vee -50 + y \geq 0 \wedge 50 - y \geq 0 \wedge$$
$$\boxed{51 - x \geq 0} \vee x - y \geq 0 \wedge -x + y \geq 0) \vee x - y \geq 0 \wedge -x + y \geq 0$$

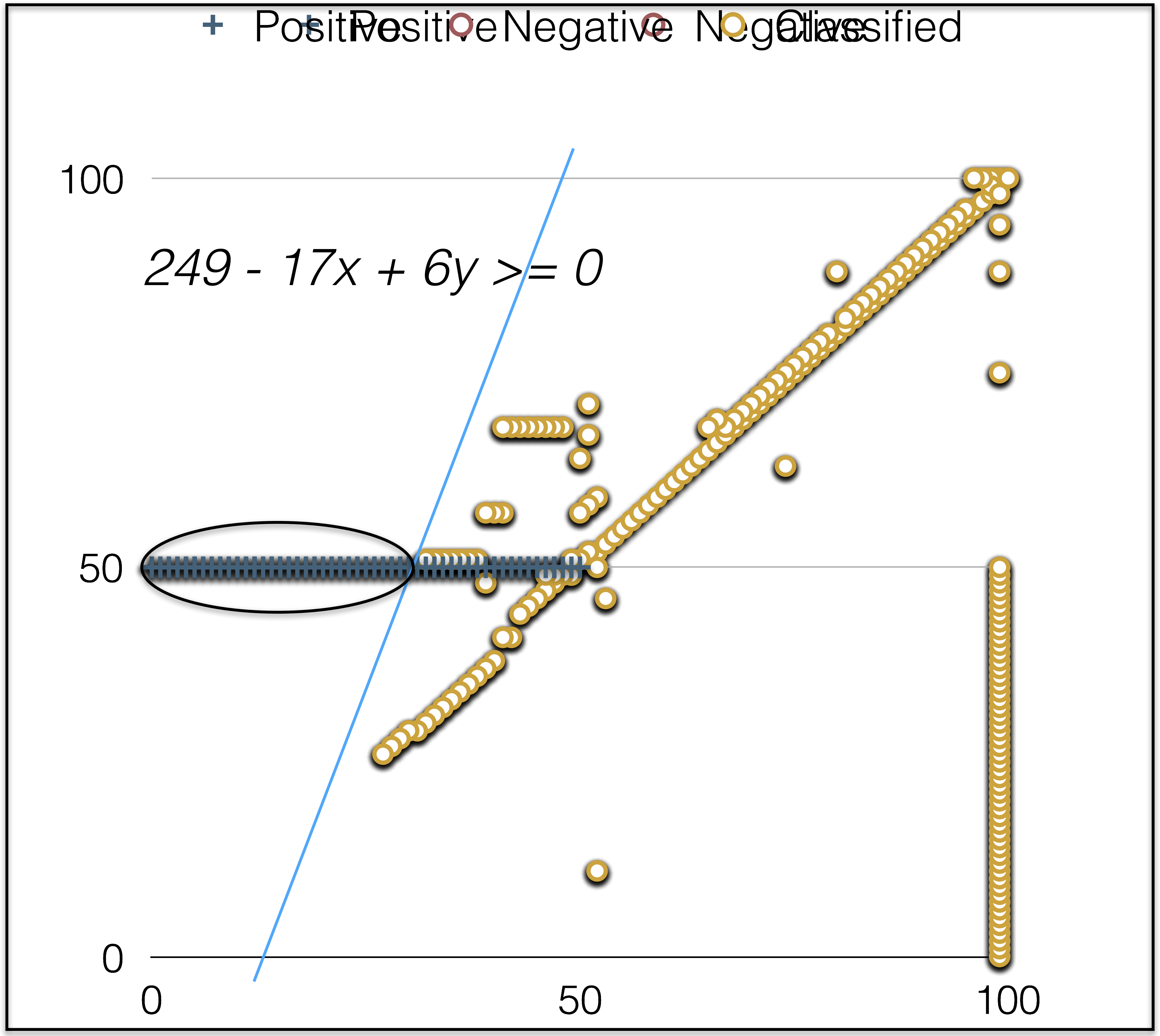
Z3

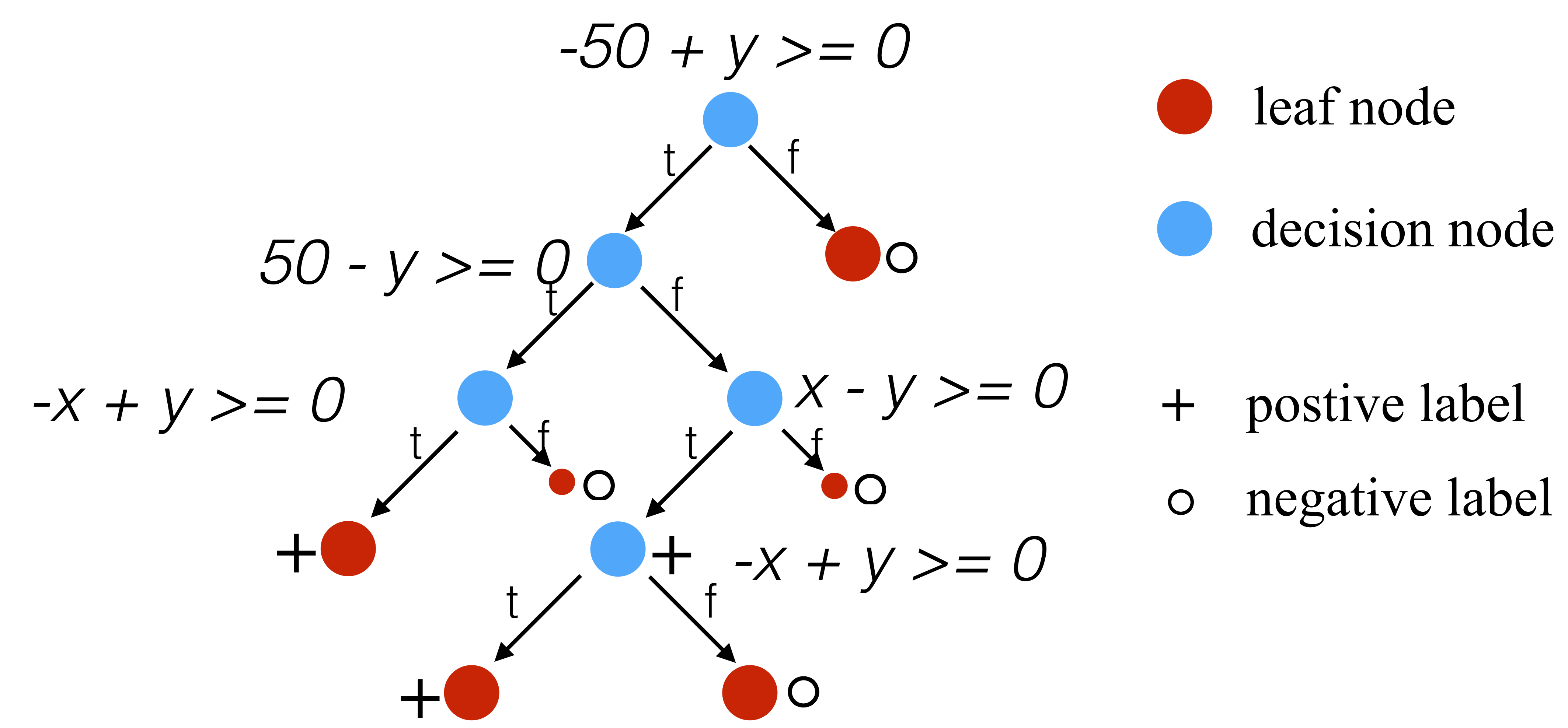
Combating Over- and Under-fitting

- Can we generalize the learned invariant solely using the data from which the linear classifiers are produced?

Vision:
A simple invariant is more likely to generalize.

Goal: Design a learner to learn *simple* invariants





Decision Tree Learning

Data

-50 + y >= 0
50 - y >= 0
-x + y >= 0
x - y >= 0
~~56 - x >= 0~~
~~51 - x >= 0~~
~~249 - 17x + 6y >= 0~~

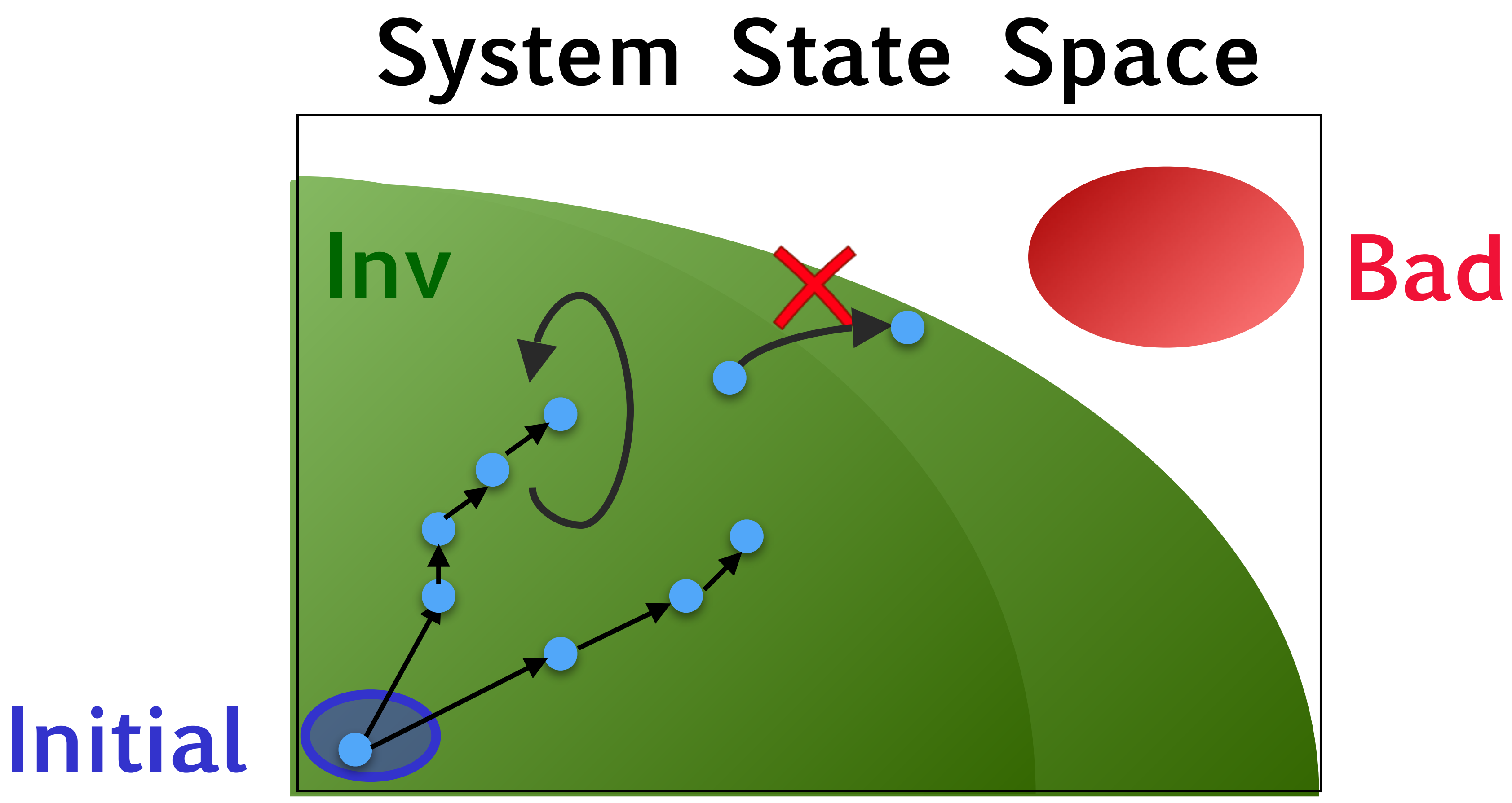
Learned classifiers from linear classification

$$p(x,y) \equiv -50 + y \geq 0 \wedge 50 - y \geq 0 \wedge -x + y \geq 0 \vee$$
$$-50 + y \geq 0 \wedge \neg(50 - y \geq 0) \wedge x - y \geq 0 \wedge -x + y \geq 0$$

Z3

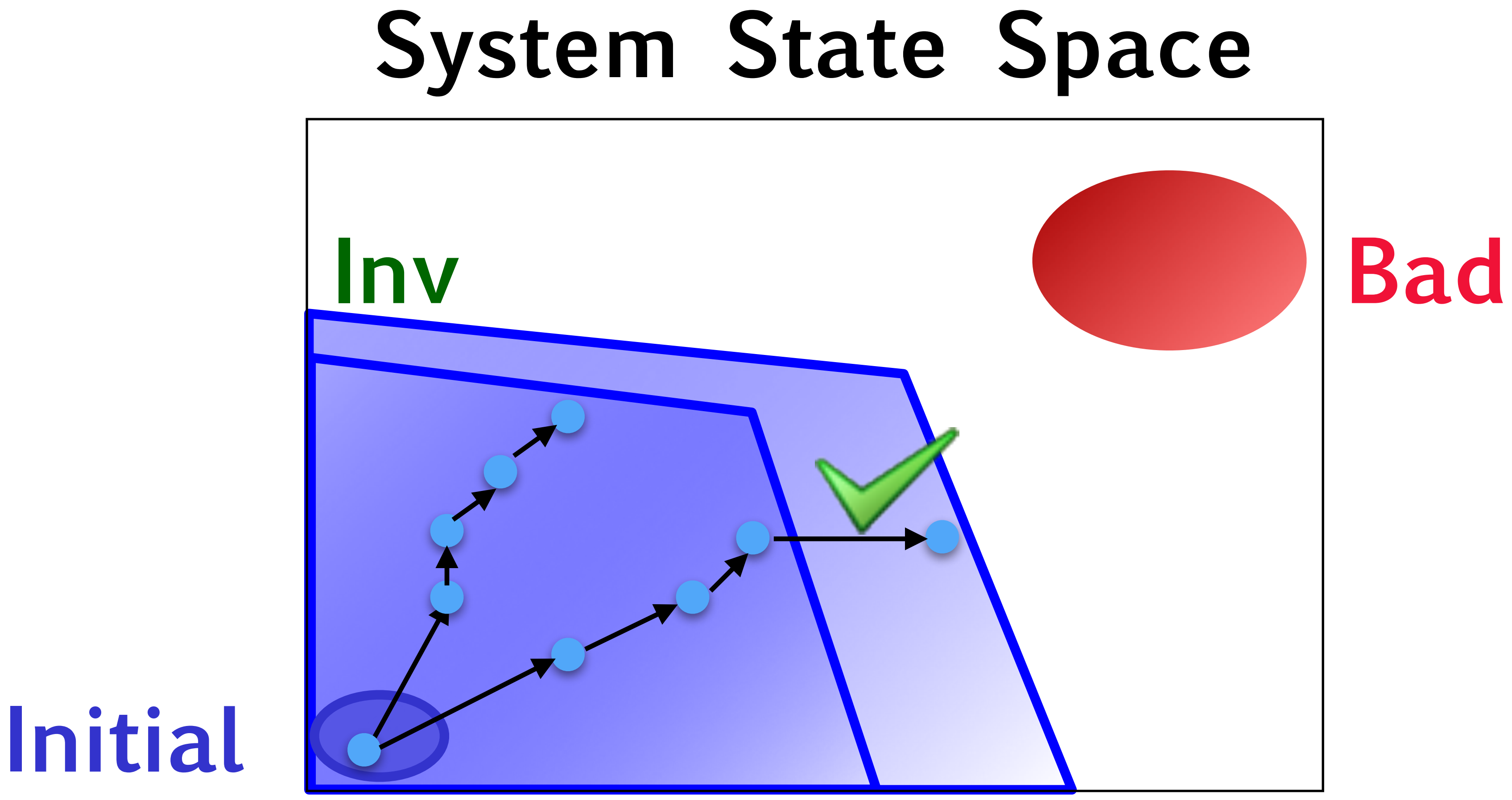
Counterexample guided sampling by Z3

$$Tr(X, X') \land Inv[X] \rightarrow Inv[X']$$

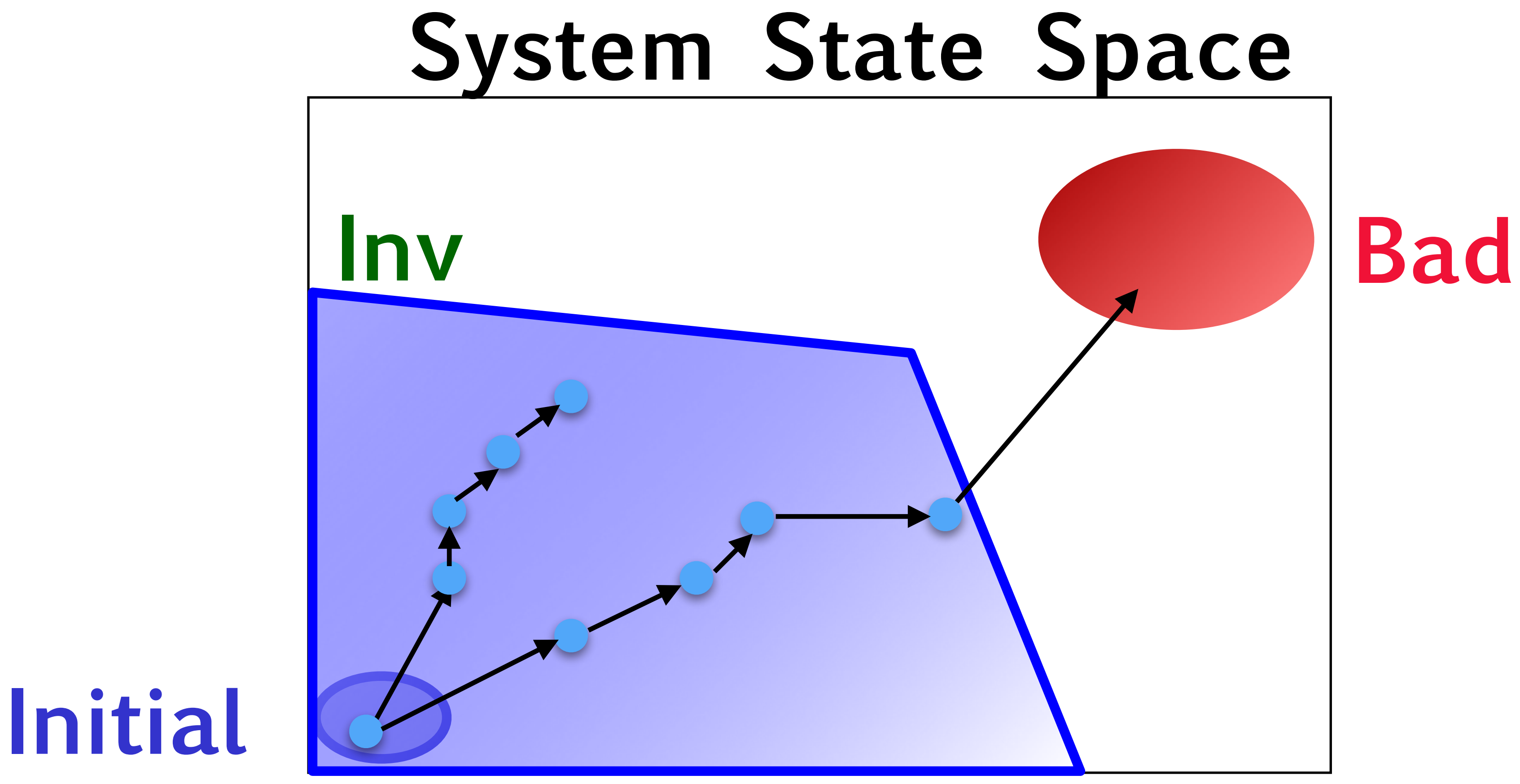


Strengthen Invariant

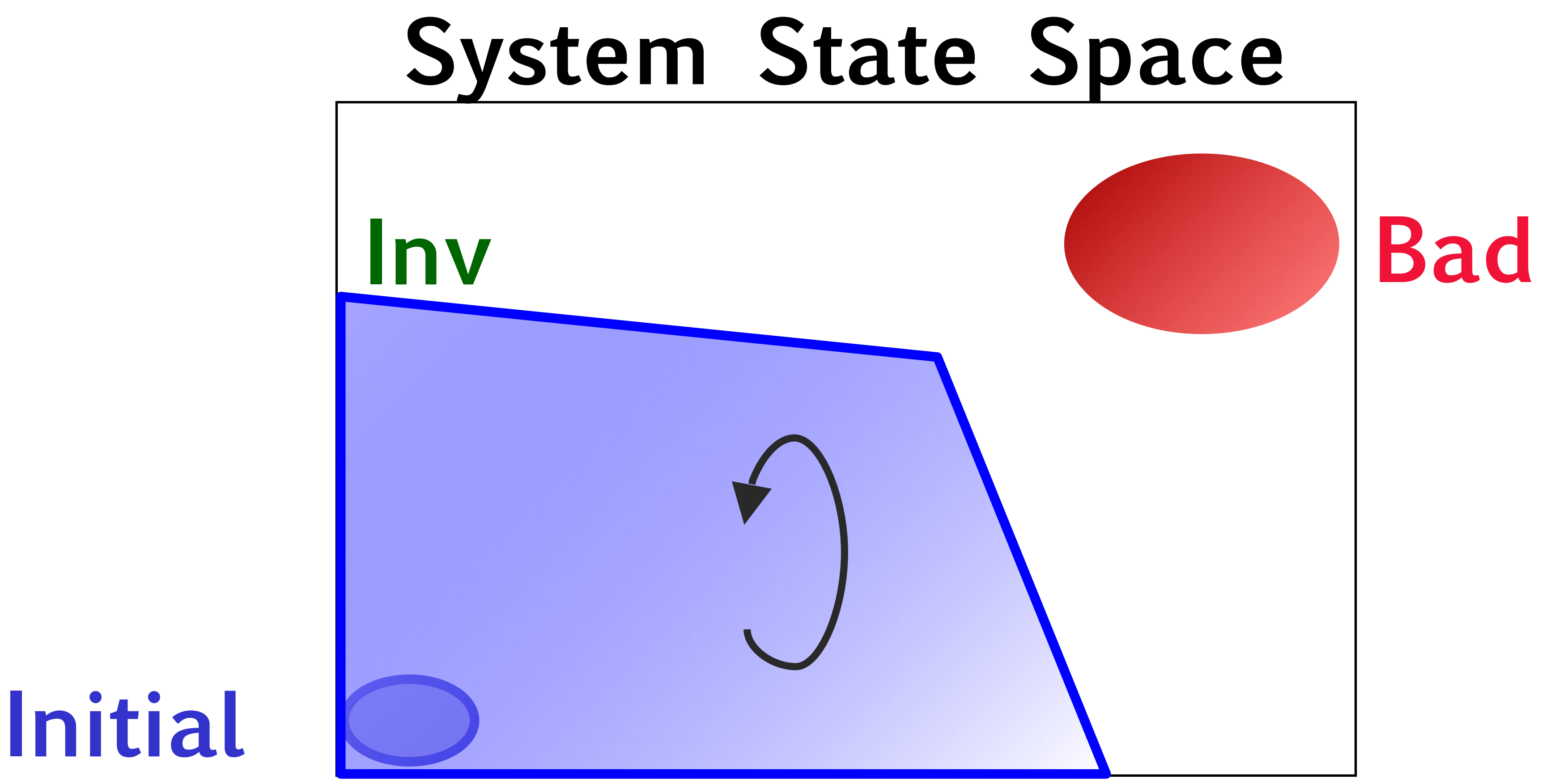
Weaken Invariant



Find a true counterexample



Find an inductive invariant



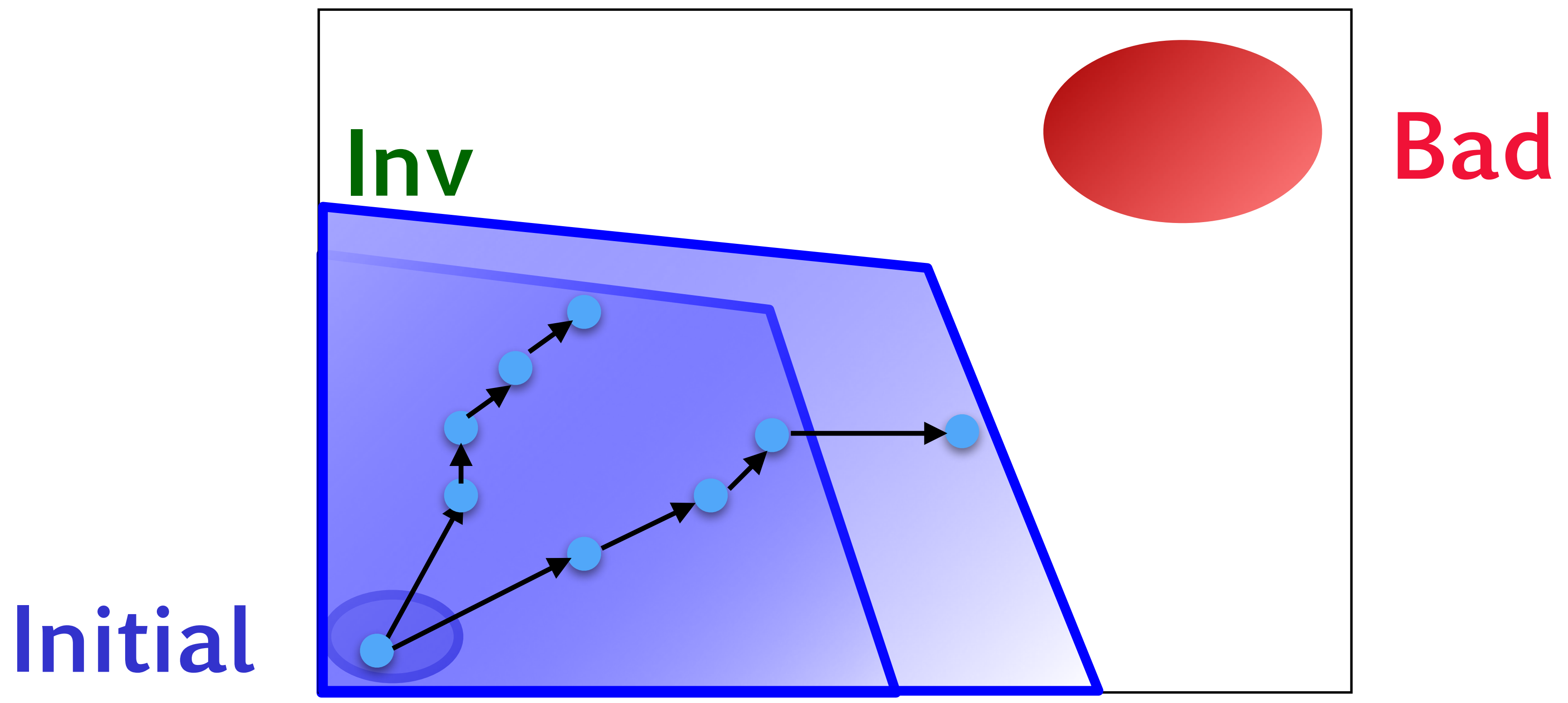
Counterexample guided sampling by Z3

SynthHorn

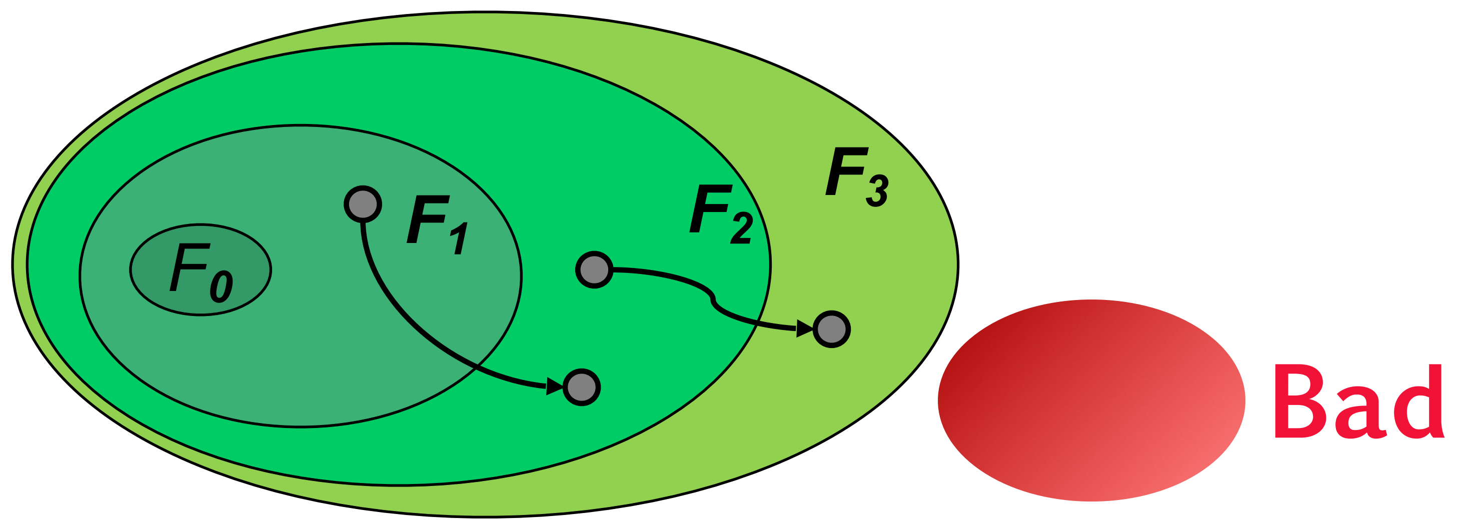
vs

Spacer, GPDR, Duality

System State Space



System State Space



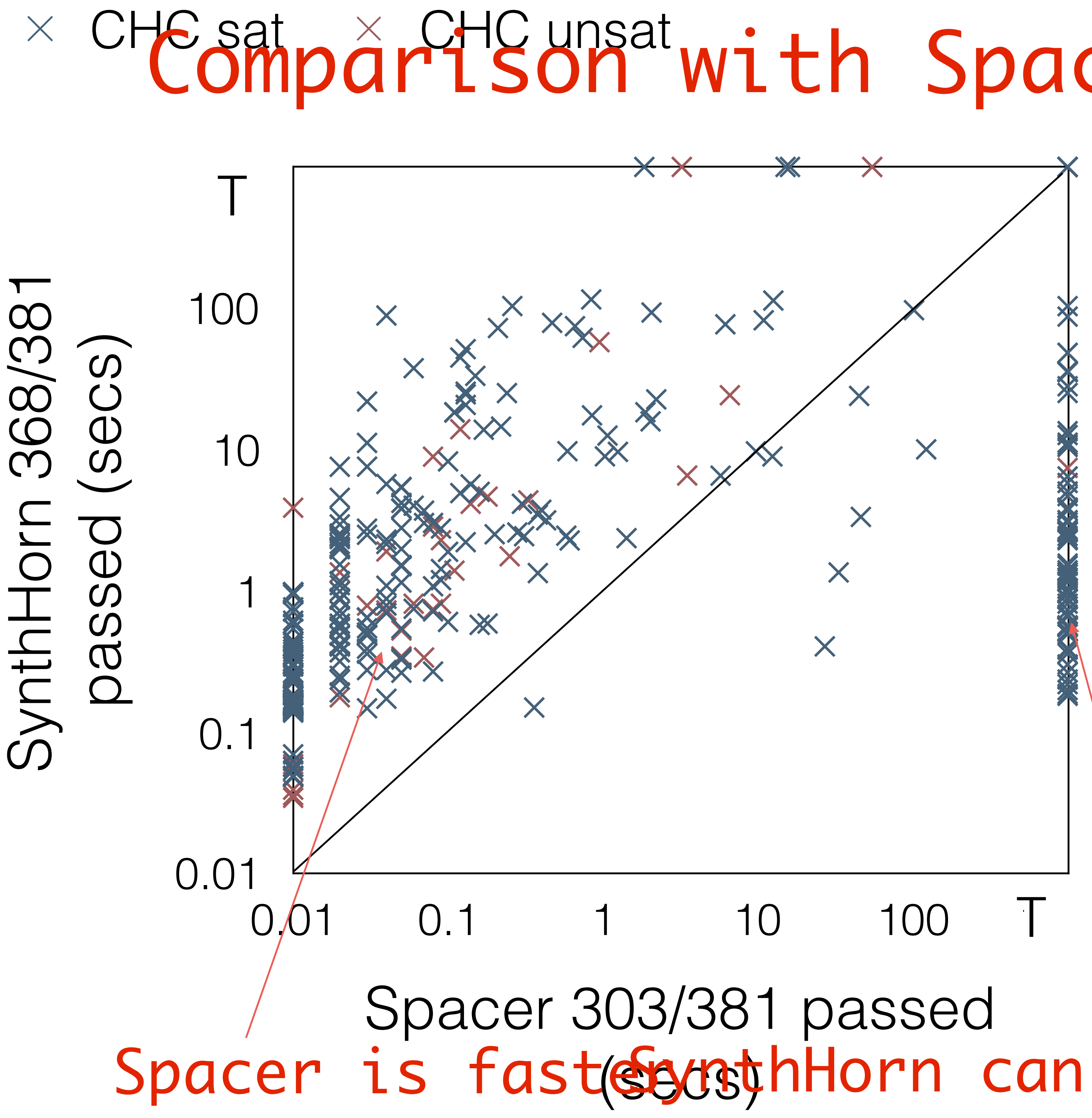
Generalizing from
bounded positive samples
using Machine Learning

Generalizing from
bounded unrolling of CHCs
using Interpolation

Experimental Results

- Collected 381 loop and recursive programs with

Comparison with Spacer



Spacer is faster

SynthHorn can verify more programs

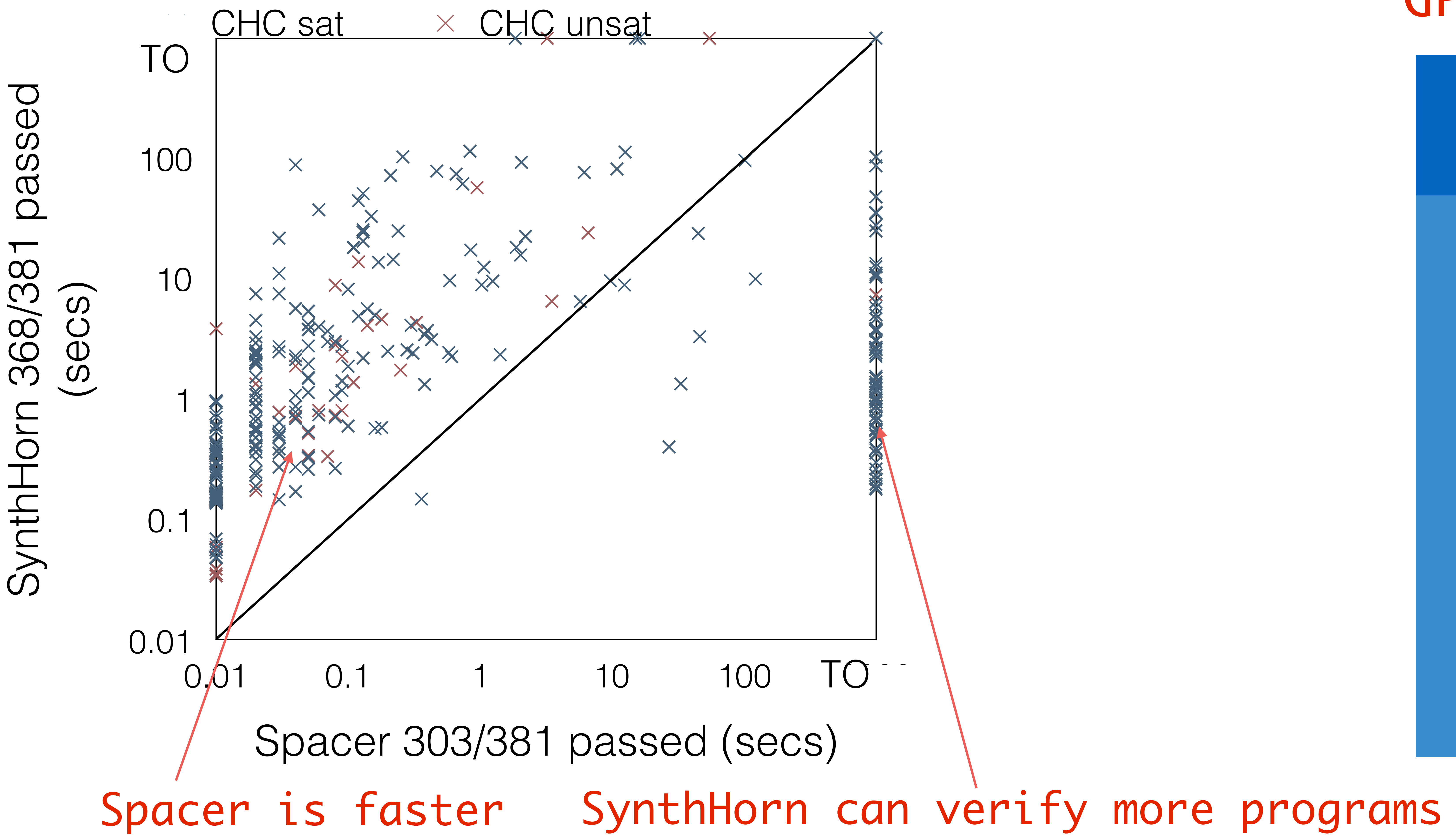
Comparison with GPDR, Spacer, Duality

Total	381
Z3-GPDR	300
Z3-Spacer	303
Z3-Duality	309
SynthHorn	368

Experimental Results

- Collected 381 loop and recursive programs with intricate invariants

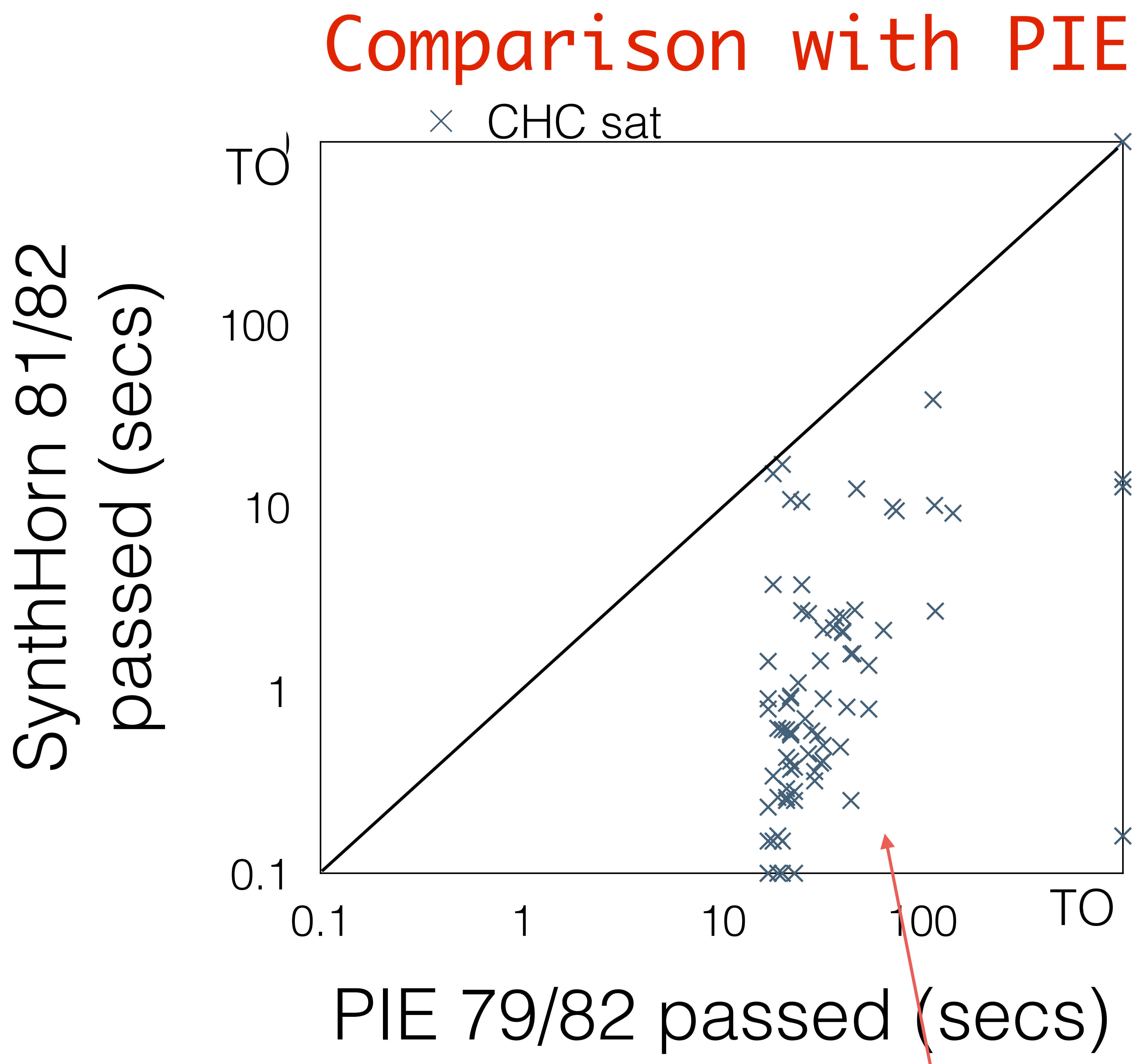
Comparison with Spacer



Comparison with GPDR, Spacer, Duality

Total	381
Z3-GPDR	300
Z3-Spacer	303
Z3-Duality	309
SynthHorn	368

Experimental Results



A data-driven invariant inference tool using enumeration-based search (PLDI'16)

Machine learning leads to order-of-magnitude faster performance than enumeration

Conclusions

Learning in the Large (MUSE)

- Trustworthiness through statistical guarantees drawn from a large corpus
- Embrace generality (learning) to discover properties (synthesis)
- Learning to drive semantic search and model generation
- Feature discovery to guide abstraction and refinement

Learning in the Small (SynthHorn)

- (Small) sample generation from theorem provers
- Tame generality (learning) to realize safety (verification)
- Learning to discover classifiers
- Feature discovery to simplify invariants