Teaching vs. Learning, and Course Wrap-Up

Sanjit A. Seshia

EECS 219C

EECS Department
UC Berkeley

Teaching vs. Learning

- Learning: Examples → Concept
- Teaching: Concept → Examples
  - Given a concept, give a "good" set of examples such that a learner can uniquely identify that concept
    - "good" typically means smallest
- Teaching dimension (TD) of a concept class C: the minimum number of examples a teacher must reveal to uniquely identify any concept in C
- Observation: [Goldman & Kearns]
  #(membership queries to identify a concept in C) $\geq$ TD(C)

Relevance to Verification and Synthesis

- As discussed earlier, Verification "=" Synthesis
- Learning is Synthesis from Examples
- Teachability of a concept (as measured by TD) can give us guidance on designing a learning algorithm
- Optimal teaching sequence:
  Given a concept, what's the smallest sequence of examples to provide so as to uniquely identify the concept?
  - Example: Rectangles on a 2D grid; Hyperboxes in n dimensions

Some Examples from Our Work

- "Oracle-Guided Component-Based Program Synthesis", S. Jha et al., ICSE 2010
- "Synthesizing Switching Logic for Safety and Dwell-Time Requirements", S. Jha et al, ICCPS 2010.

Security: The Growth of Malware

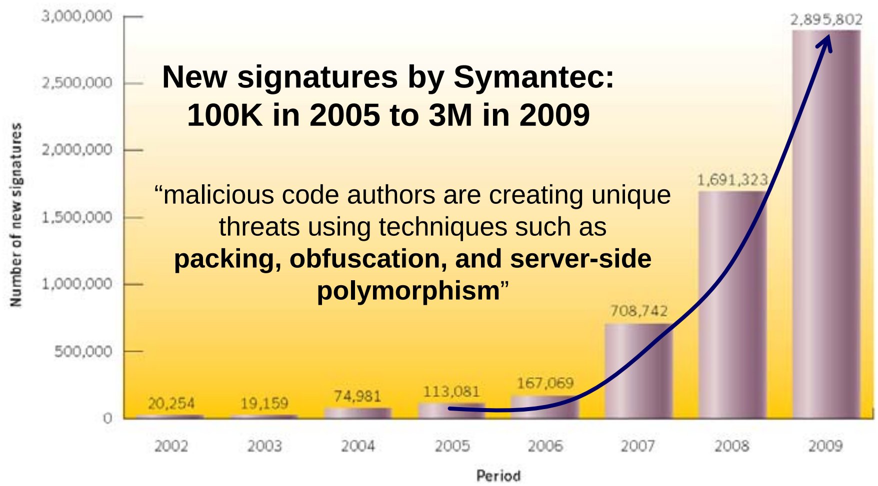


Figure 10. New malicious code signatures

Source: Symantec.

Motivating Problem: Deobfuscating Malware

Obfuscated code:

Input: y   Output: modified value of y

```
{ a=1; b=0; z=1; c=0;
  while(1) {
    if (a == 0) {
    if (b == 0) { y=z+y; a =~a;
    b=~b;  c=~c; if (~c) break; }
   else {
     z=z+y; a=~a; b=~b; c=~c;
     if (~c) break; } }
    else if(b == 0) {z=y << 2; a=~a;}
     else { z=y << 3; a=~a; b=~b;}
 } }
```

What it does:
Multiplies y by 45

We solve this using program synthesis.

We get:

```
{  z = y << 2; y = z + y;
   z = y << 3; y = z + y;
}
```

FROM CONFICKER WORM

Paper: S. Jha et al., “Oracle-Guided Component-Based Program Synthesis”, ICSE 2010.

Sciduction for Program Synthesis

Structure Hypothesis:
Programs are Loop-Free Compositions of Known Components

+

Inductive Inference:
Learning from Distinguishing Examples

+

Deductive Engine:
SMT solving to generate distinguishing inputs

Class of Programs

- **Programs implementing functions: $\mathbf{I} \rightarrow \mathbf{O}$**

P($\mathbf{I}$):
$O_1 = f_1 (V_1)$
$O_2 = f_2 (V_2)$
…
$O_n = f_n (V_n)$

where

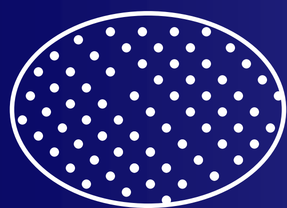
$f_1, f_2, \ldots, f_n$ are functions from a given component library

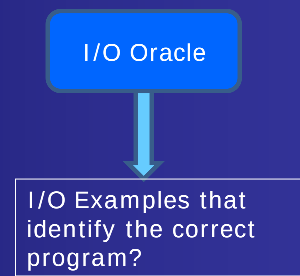
Functions could be if-then-else definitions and hence, the above represents any loop-free code.

Problem

Specification Oracle

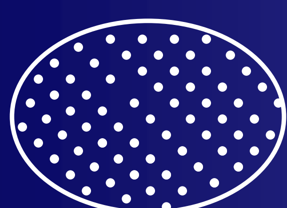
Program Space





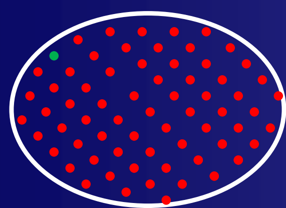
Space of all possible programs obeying our structure hypothesis
Each dot represents semantically unique program

Program Learning as Set Cover



Space of all possible programs
Each dot represents semantically unique program

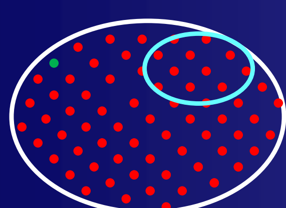
Program Learning as Set Cover



Space of all possible programs
Each dot represents
semantically unique program

Program Learning as Set Cover

(i_1, o_1)



Space of all possible programs
Each dot represents
semantically unique program

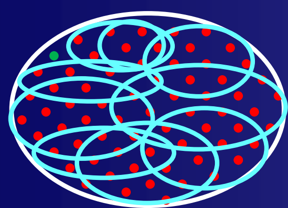
Example ↔ Set of programs ruled out by that example

Program Learning as Set Cover

(i_1, o_1) - E_1
(i_2, o_2) - E_2

………

(i_n, o_n) - E_n



Space of all possible programs
Each dot represents
semantically unique program

Smallest set of I/O examples to learn correct design

IS

Minimum size subset of $\{E_1, E_2, \ldots\ldots, E_n\}$ that cover all the incorrect programs

Optimal teaching seq problem = Min set cover problem

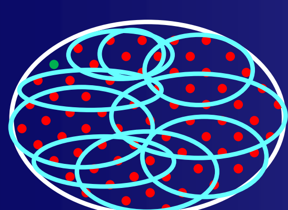
Program Learning as Set Cover

(i_1, o_1) - E_1
(i_2, o_2) - E_2

Bad news: can't enumerate all inputs and find set E_i for each

………

(i_n, o_n) - E_n



Space of all possible programs
Each dot represents
semantically unique program

Smallest set of I/O examples to learn correct design

IS

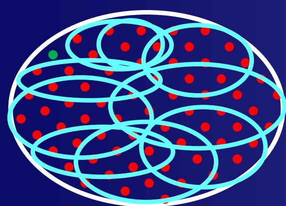
Minimum size subset of $\{E_1, E_2, \ldots\ldots, E_n\}$ that cover all the incorrect programs

Program Learning as Set Cover

(i_1, o_1) - E_1
(i_2, o_2) - E_2

………

(i_n, o_n) - E_n



Space of all possible programs
Each dot represents semantically unique program

ONLINE set-cover:

In each step,
- choose some (i_j, o_j) pair
- eliminated incorrect programs E_j disclosed

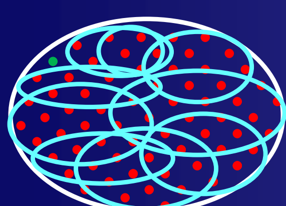
Program Learning as Set Cover

(i_1, o_1) - E_1
(i_2, o_2) - E_2

$|E_j| \geq 1$: atleast one incorrect program identified

………

(i_n, o_n) - E_n

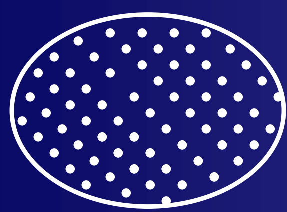


Space of all possible programs
Each dot represents semantically unique program

ONLINE set-cover:

In each step,
- choose some (i_j, o_j) pair
- eliminated incorrect programs E_j disclosed

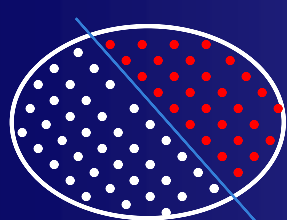
Our Approach



Space of all possible programs
Each dot represents
semantically unique program

Our Approach

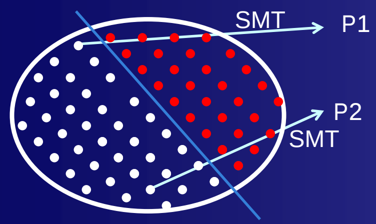
Example I/O set E := $\{(i_1,o_1)\}$



Space of all possible programs

Our Approach

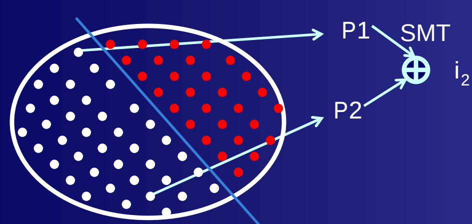
Example I/O set E := $\{(i_1,o_1)\}$



Space of all possible programs

Our Approach

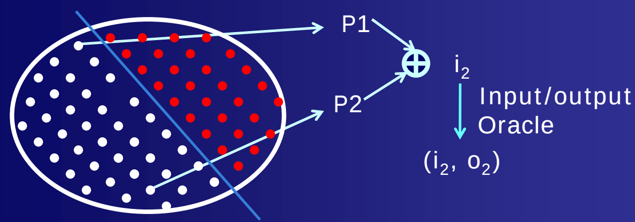
Example I/O set E := $\{(i_1,o_1)\}$



Space of all possible programs

Our Approach

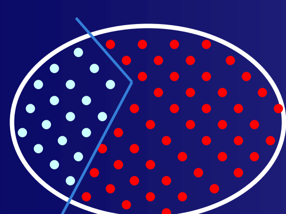
Example I/O set E := $\{(i_1,o_1)\}$



Space of all possible programs

Our Approach

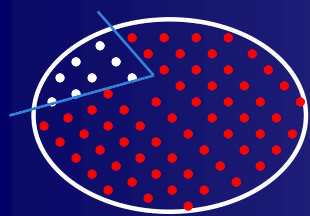
Example I/O set E := E $\cup$ $\{(i_2,o_2)\}$



Space of all possible programs

Our Approach

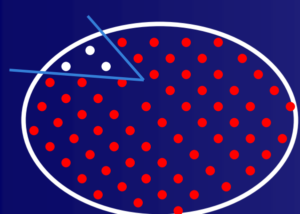
Example I/O set $E := E \cup \{(i_j,o_j)\}$



Space of all possible programs

Our Approach

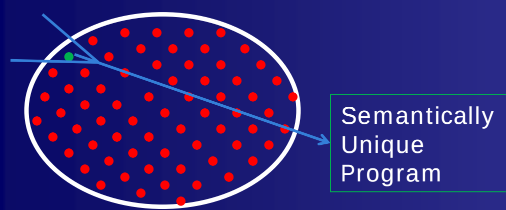
Example I/O set $E := E \cup \{(i_k,o_k)\}$



Space of all possible programs

Our Approach

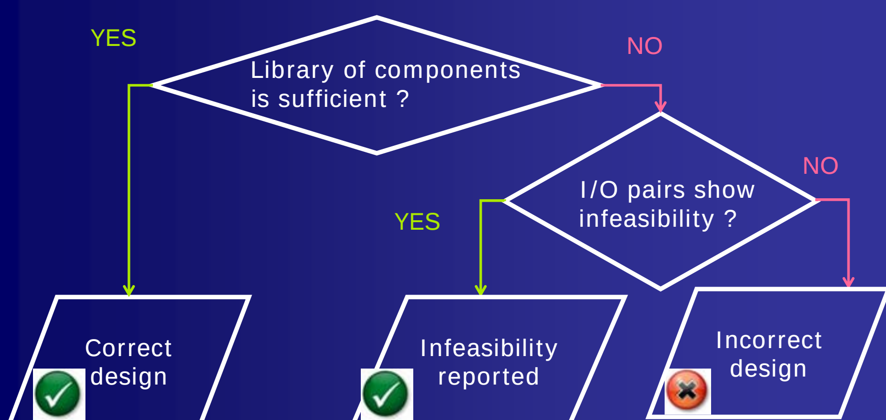
Example I/O set E := E ∪ {(i_n,o_n)}



Correct Program?

Space of all possible programs

Soundness



Other Important Details

[see ICSE ’10, PLDI ’11 papers]

- Representing the space of possible programs using SMT formula
- Obtaining a feasible program for given set of input/output pairs using SMT solving
- Obtained second feasible program and a distinguishing input using SMT solving

Result Highlights

- Malware Deobfuscation
  - – Conficker worm
  - – MyDoom and
  - – survey paper on obfuscations by Collberg et al*
- Synthesized over 35 bit-manipulation programs from Hacker’s delight (the “Bible of bit-manipulation”).
- Program length: 3-15
- Number of input/output examples: 2 to 13.
- Total runtime: < 1 second to 5 minutes.

*C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, Dept. Comp. Sci., The Univ. of Auckland, July 1997.

Discussion

- Notion of “teaching” can be useful in guiding the design of a learning algorithm, or proving bounds on the sample complexity

Course Topics Review

- SAT Solving
  – Complexity, random SAT instances, …
  – CDCL (DPLL) SAT solvers
- BDDs
- SMT Solving
  – Commonly used theories, Nelson-Oppen combination
  – Lazy SMT solving -- DPLL(T), etc.
  – Eager SMT solving – Small-domain encoding, UCLID, …

Course Topics Review

- Model Checking
  - Modeling: things to keep in mind
  - Temporal logic
  - Explicit-state model checking
    - Basic automata-theoretic approach
    - DFS, Nested DFS, …
    - Partial-order reduction, state compression, …
  - Symbolic model checking
    - QBF, fixpoint theory
    - Abstraction: cone-of-influence, CEGAR, proof-based abstraction, interpolation
    - Symmetry reduction
    - K-induction, IC3
  - Simulation/bisimulation, compositional reasoning

Course Topics Review

- Inductive Learning + Deduction
  - Verification “=” Synthesis
  - Compositional reasoning, L* algorithm
  - Survey of learning algorithms: Basics, Batch learning, PAC learning model, online learning model
  - Teaching vs learning
- Synthesis from LTL

Things we did not cover

- Verification of Infinite-State Systems
  - Software, timed/hybrid systems, etc.
- Quantitative Verification / Synthesis
- Error localization and debugging
- Interactive theorem proving
- …

See list of project topics introduced in first lecture for directions for future research