Synthesis, Verification, and Inductive Learning

Sanjit A. Seshia

EECS Department

UC Berkeley

Joint work with Susmit Jha (UTC)

Dagstuhl Seminar → Verified SW Working Group
August 2014 → July 15, 2015

Messages of this Talk

[Seshia DAC'12; Jha & Seshia, SYNT'14, ArXiV'15]

1. **Synthesis Everywhere**
   - **Many (verification) tasks involve synthesis**
2. **Effective Approach to Synthesis: Induction + Deduction + Structure**
   - ***Induction*: Learning from examples**
   - ***Deduction*: Logical inference and constraint solving**
   - ***Structure*: Hypothesis on syntactic form of artifact to be synthesized**
   - **"Syntax-Guided Synthesis"** [Alur et al., FMCAD'13]
     - Counterexample-guided inductive synthesis (CEGIS) [Solar-Lezama et al., ASPLOS'06]
3. **Analysis of Counterexample-Guided Synthesis**
   - **Counterexample-driven learning**
   - **Sample Complexity**

Artifacts Synthesized in Verification

- Inductive invariants
- Auxiliary specifications (e.g., pre/post-conditions, function summaries)
- Environment assumptions / Env model / interface specifications
- Abstraction functions / abstract models
- Interpolants
- Ranking functions
- Intermediate lemmas for compositional proofs
- Theory lemma instances in SMT solving
- Patterns for Quantifier Instantiation
- …

Formal Verification as Synthesis

- **Inductive Invariants**

- **Abstraction Functions**

One Reduction from Verification to Synthesis

NOTATION

Transition system $M = (I, \delta)$

Safety property $\Psi = G(\psi)$

VERIFICATION PROBLEM

Does M satisfy Ψ?

SYNTHESIS PROBLEM

Synthesize ϕ s.t.

$$I \Rightarrow \phi \wedge \psi$$

$$\phi \wedge \psi \wedge \delta \Rightarrow \phi' \wedge \psi'$$

Two Reductions from Verification to Synthesis

NOTATION

Transition system $M = (I, \delta)$, S = set of states

Safety property $\Psi = G(\psi)$

VERIFICATION PROBLEM

Does M satisfy Ψ?

SYNTHESIS PROBLEM #1

Synthesize ϕ s.t.

$$I \Rightarrow \phi \wedge \psi$$

$$\phi \wedge \psi \wedge \delta \Rightarrow \phi' \wedge \psi'$$

SYNTHESIS PROBLEM #2

Synthesize $\alpha : S \rightarrow \hat{S}$ where

$$\alpha(M) = (\hat{I}, \hat{\delta})$$

s.t.

$\alpha(M)$ satisfies Ψ

iff

M satisfies Ψ

Common Approach for both: “Inductive” Synthesis

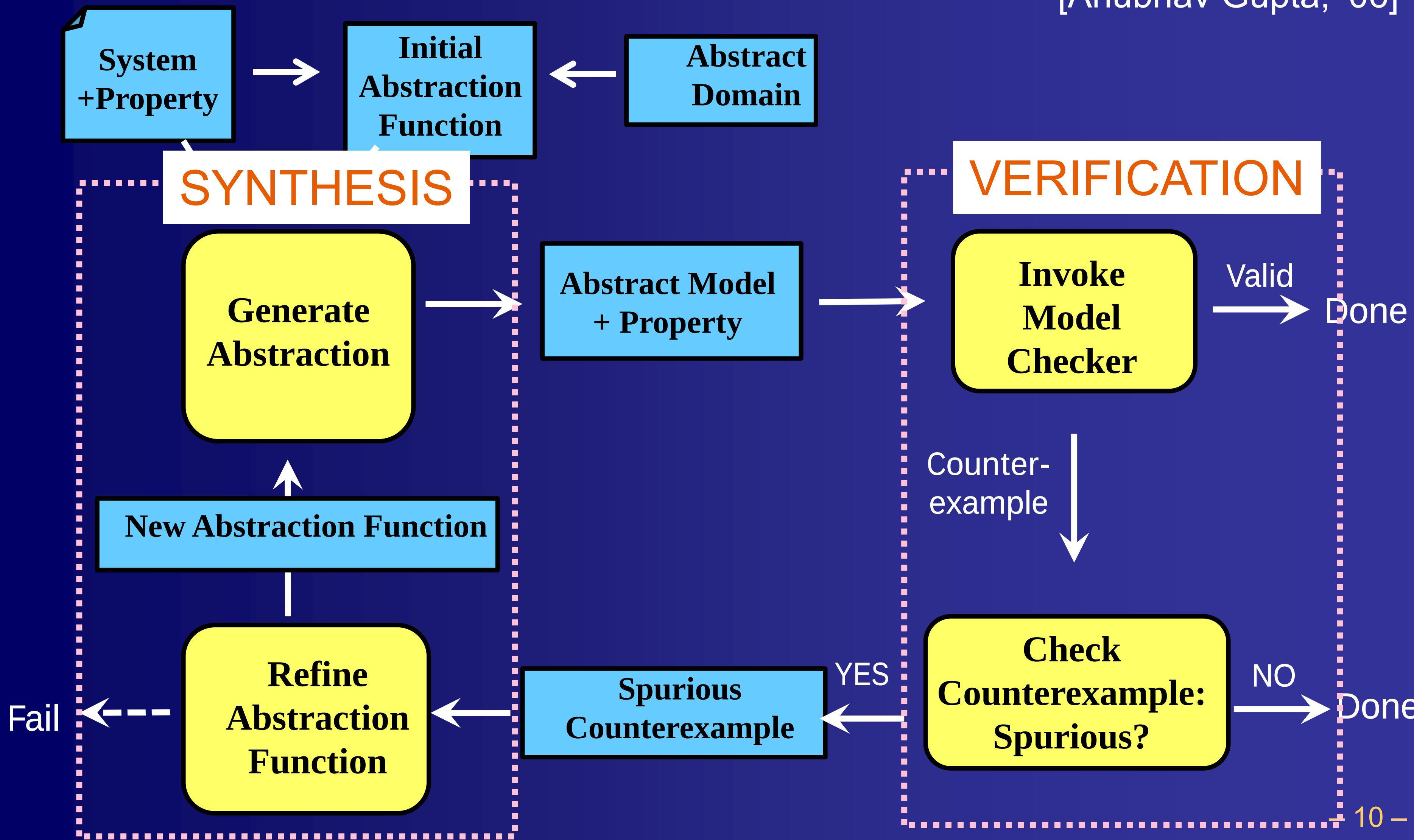
Synthesis of:-

- **Inductive Invariants**
  – Choose templates for invariants
  – Infer likely invariants from tests (examples)
  – Check if any are true inductive invariants, possibly iterate

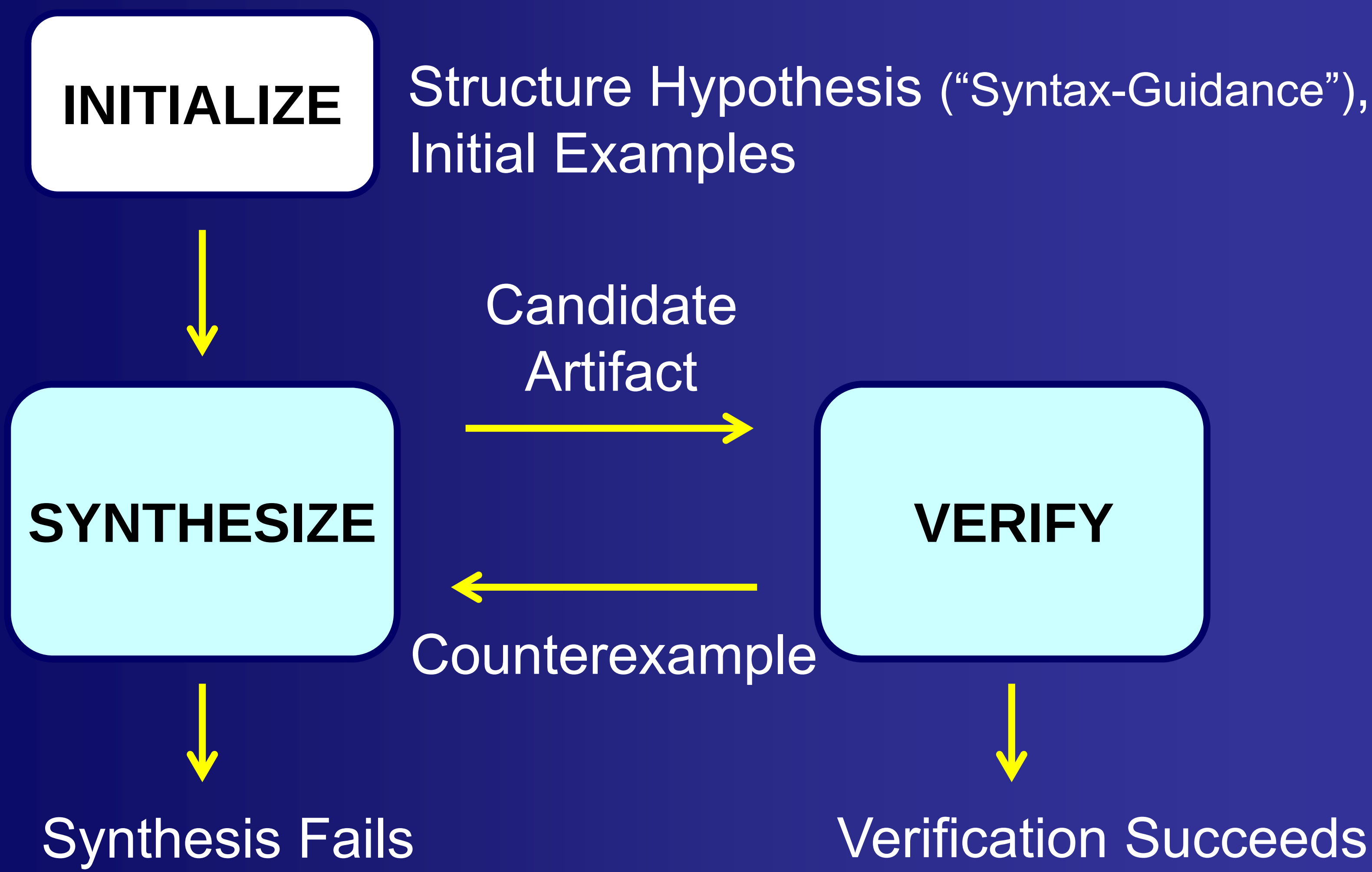
- **Abstraction Functions**
  – Choose an abstract domain
  – Use Counter-Example Guided Abstraction Refinement (CEGAR)

Counterexample-Guided Abstraction Refinement is Inductive Synthesis

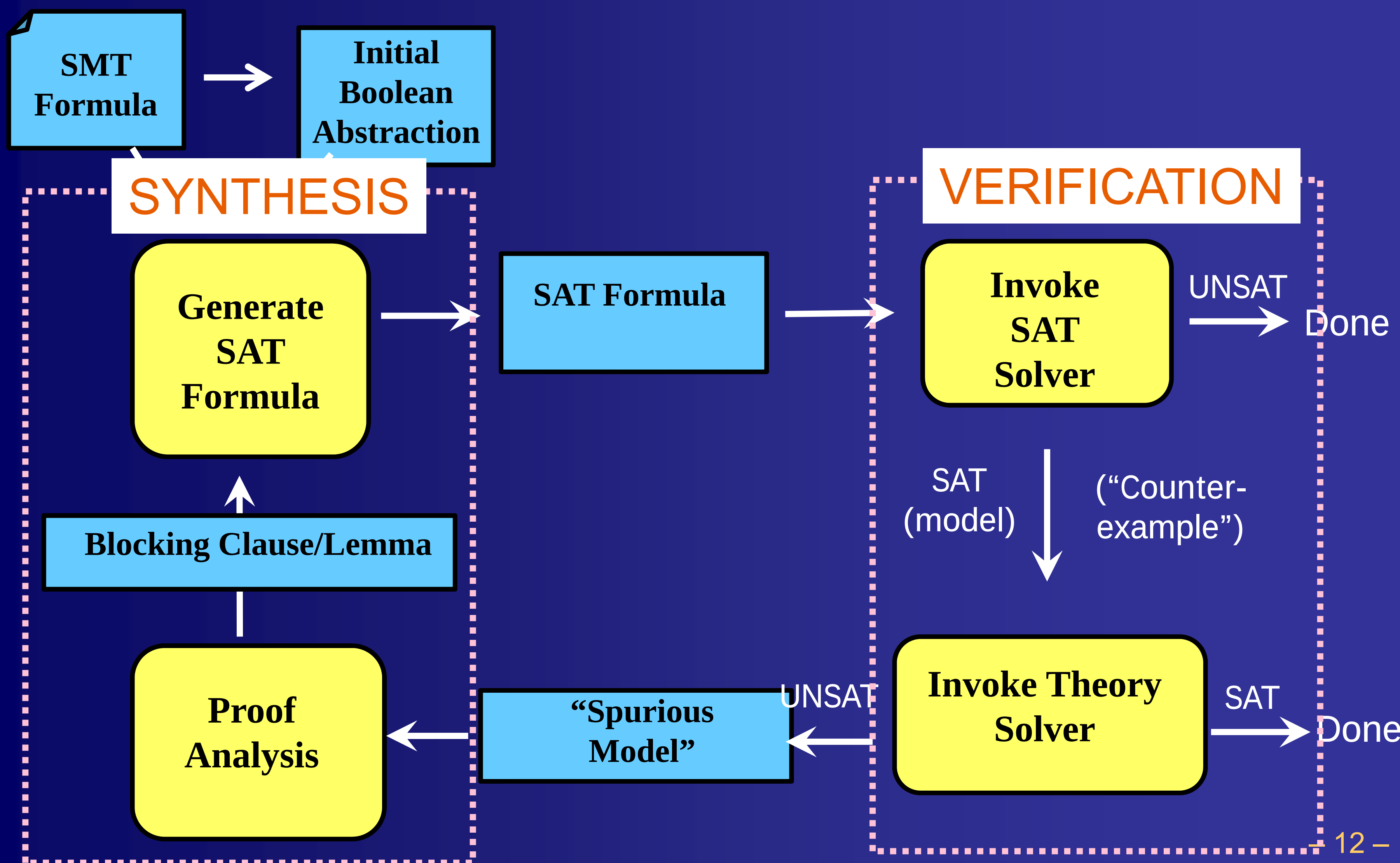
[Anubhav Gupta, '06]



CEGAR = Counterexample-Guided Inductive Synthesis (of Abstractions)

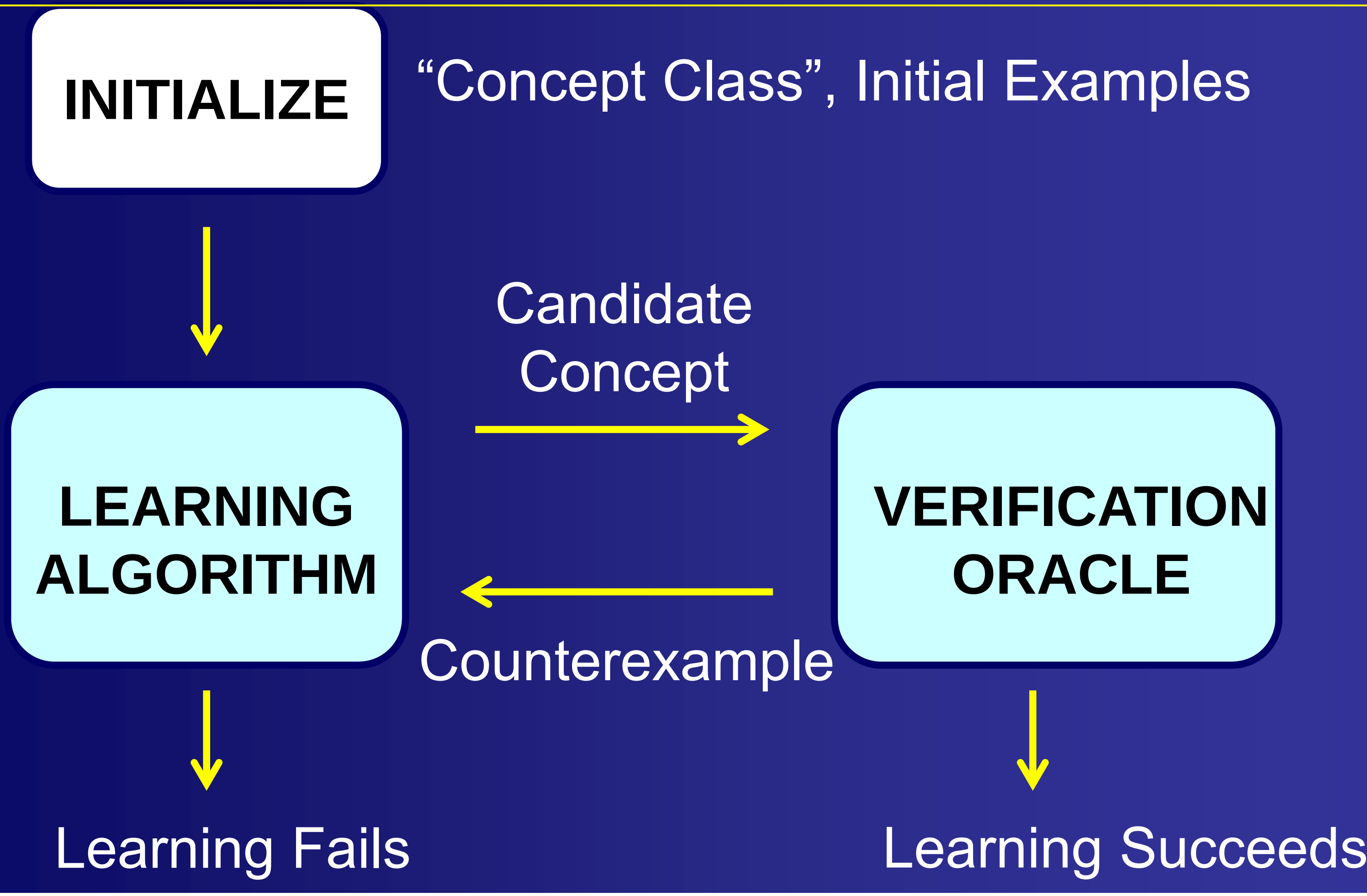


Lazy SMT Solving performs Inductive Synthesis (of Lemmas)



CEGAR = CEGIS = Learning from (Counter)Examples

What's different from std learning theory: Learning Algorithm and Verification Oracle are typically general Solvers

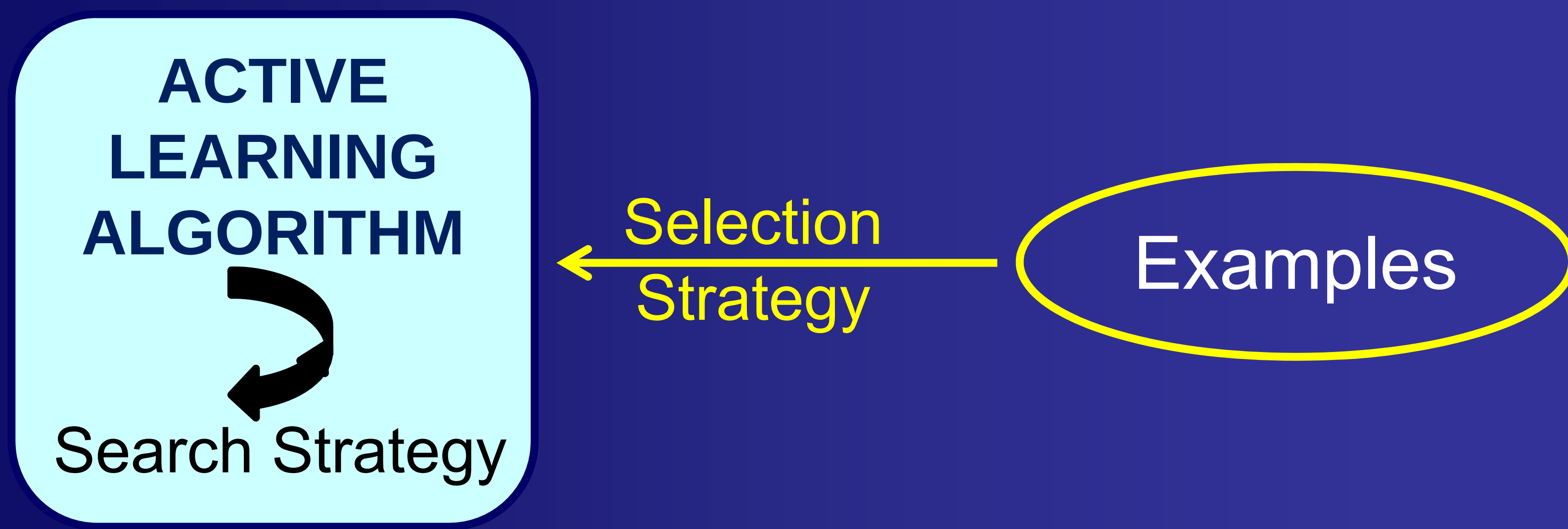


Comparison*

Feature	Formal Inductive Synthesis	Machine Learning
Concept/Program Classes	Programmable, Complex	Fixed, Simple
Learning Algorithms	**General-Purpose Solvers**	**Specialized**
Learning Criteria	Exact, w/ Formal Spec	Approximate, w/ Cost Function
Oracle-Guidance	Common (can control Oracle)	Rare (black-box oracles)

* Between typical inductive synthesizer and machine learning algo

Active Learning: Key Elements



1. Search Strategy: How to search the space of candidate concepts?
2. Example Selection: Which examples to learn from?

Counterexample-Guidance: A Successful Paradigm for Synthesis and Learning

- ***Active Learning*** **from Queries and Counterexamples** [Angluin '87a,'87b]
- **Counterexample-Guided Abstraction-Refinement (CEGAR)** [Clarke et al., '00]
- **Counterexample-Guided Inductive Synthesis (CEGIS)** [Solar-Lezama et al., '06]

…

- **All rely heavily on** ***Verification Oracle***
- **Choice of Verification Oracle determines Sample Complexity of Learning**
  - **# of examples (counterexamples) needed to converge (learn a concept)**

Questions

- Fix a concept class
  - abstract domain, template, etc.

1. Suppose Countexample-Guided Learning is guaranteed to terminate. What are lower/upper bounds on sample complexity?
2. Suppose termination is not guaranteed. Is it possible for the procedure to terminate on some problems with one verifier but not another?
   - Learner (synthesizer) just needs to be consistent wth examples; e.g. SMT solver
   - Sensitivity to type of counterexample

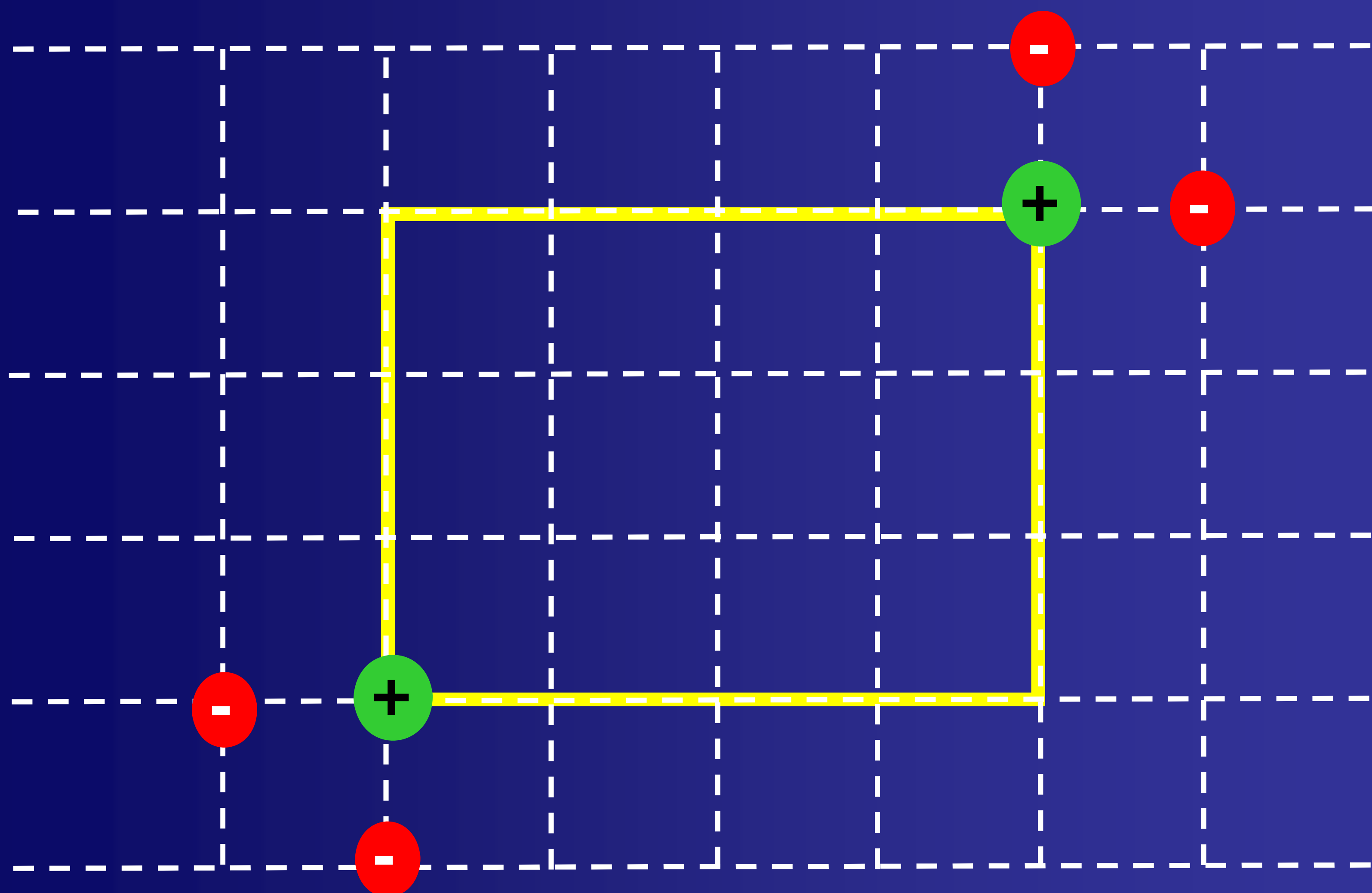
Problem 1: Bounds on Sample Complexity

Teaching Dimension

[Goldman & Kearns, '90, '95]

- **The *minimum* number of (labeled) examples a teacher must reveal to *uniquely* identify any concept from a concept class**

Teaching a 2-dimensional Box



What about N dimensions?

Teaching Dimension

- **The *minimum* number of (labeled) examples a teacher must reveal to *uniquely* identify any concept from a concept class**

$$TD(C) = \max_{c \in C} \ \min_{\sigma \in \Sigma(c)} |\sigma|$$

where

C **is a concept class**

c **is a concept**

σ **is a teaching sequence (uniquely identifies concept** c**)**

Σ **is the set of all teaching sequences**

Theorem: *TD*(*C*) is lower bound on Sample Complexity

- **Counterexample-Guided Learning: TD gives a lower bound on #counterexamples needed to learn any concept**
- **Finite TD is necessary for termination**
  - **If C is finite, $TD(C) \leq |C|-1$**
- **Finding Optimal Teaching Sequence is NP-hard (in size of concept class)**
  - **But heuristic approach works well ("learning from distinguishing inputs")**
- **Finite TD may not be sufficient for termination!**
  - **Termination may depend on verification oracle**

[some results appear in Jha et al., ICSE 2010]

Problem 2: Termination of Counterexample-guided loop

Query Types for CEGIS

LEARNER

ORACLE

Positive Witness

$x \in \phi$, if one exists, else $\perp$

Equivalence: Is $f = \phi$?

Yes / No + $x \in \phi \oplus f$

Subsumption: Is $f \subseteq \phi$?

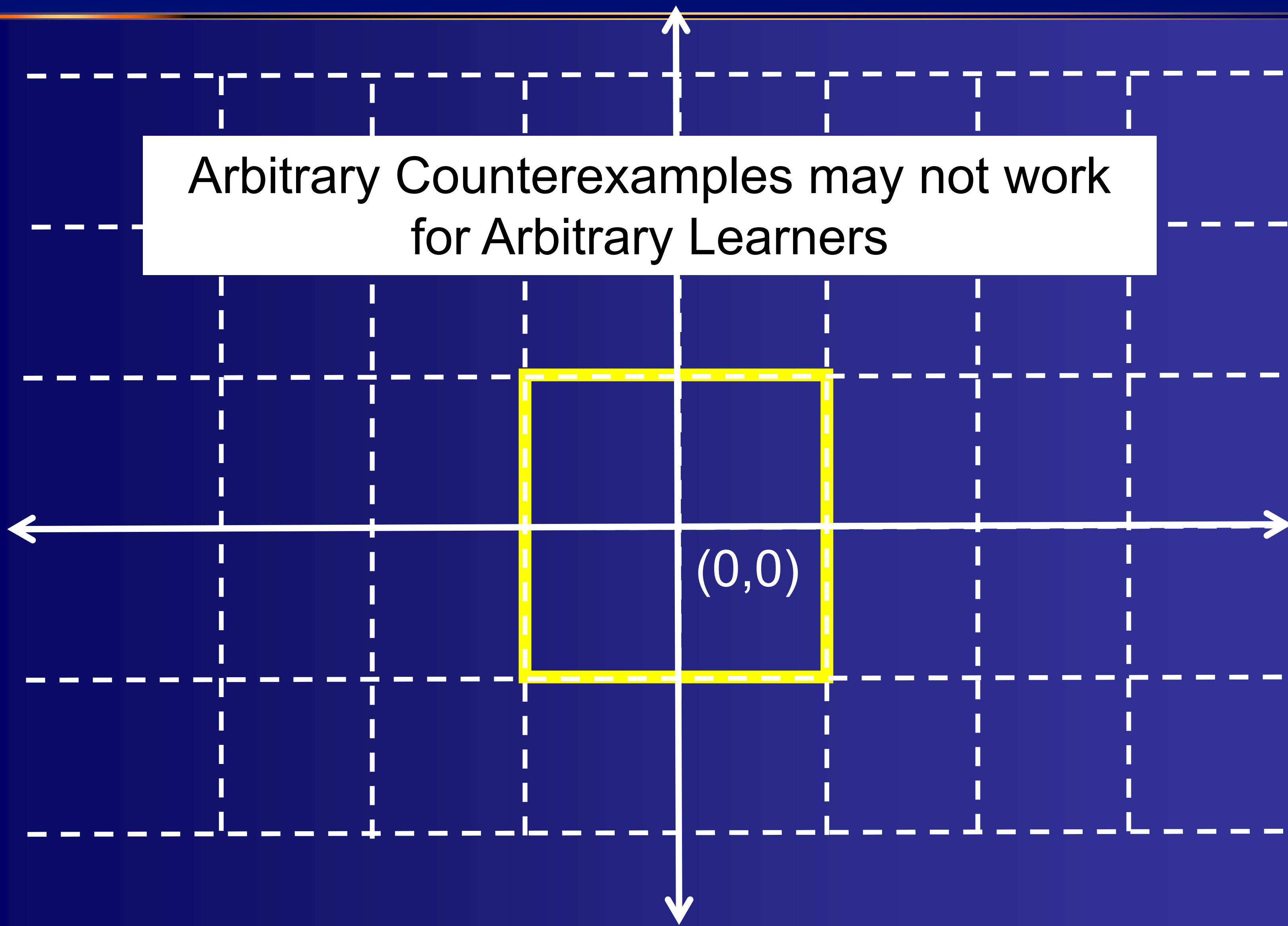
Yes / No + $x \in f \setminus \phi$

- Finite memory vs Infinite memory

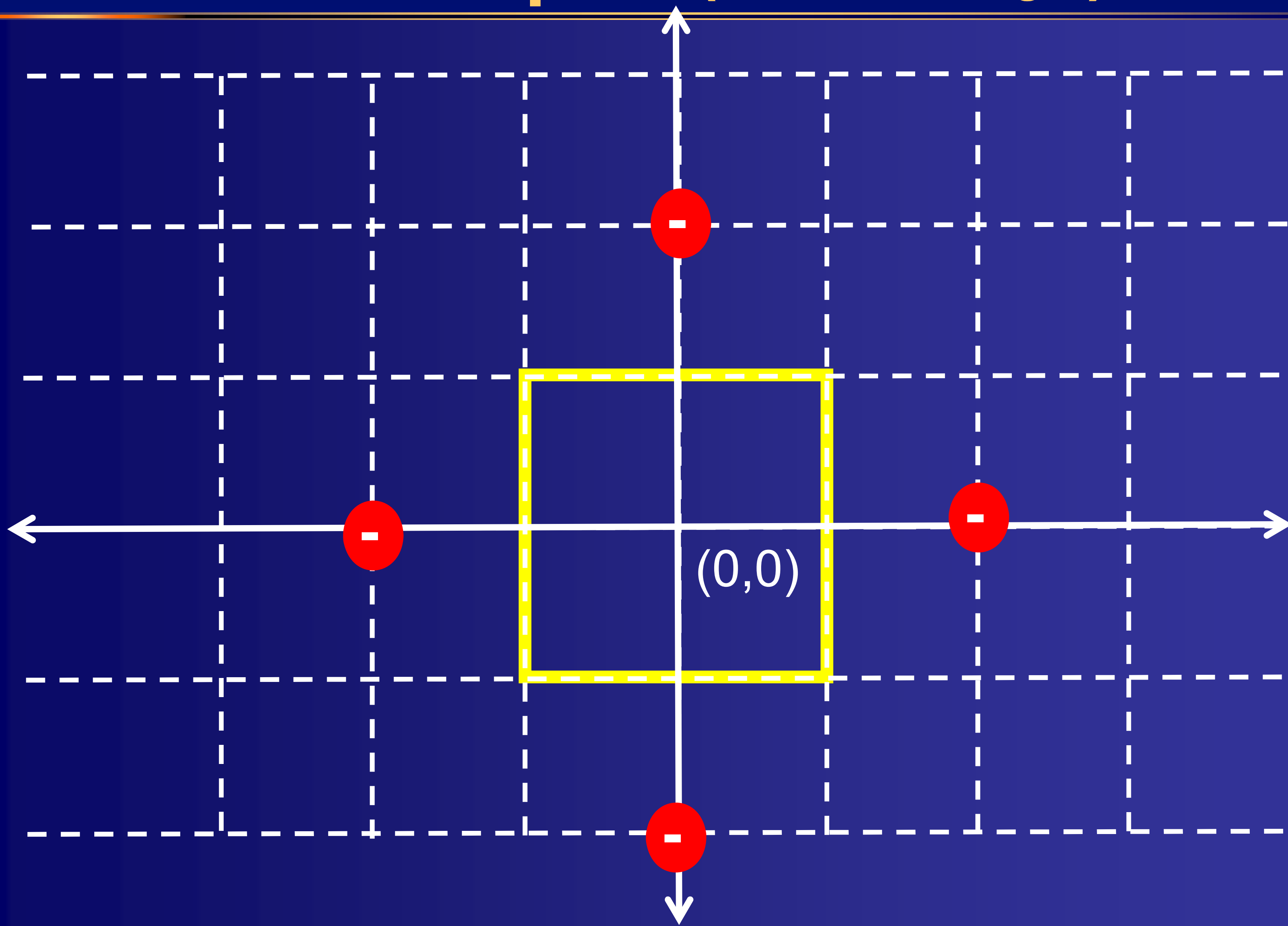
- Type of counter-example given

Concept class: Any set of recursive languages

Learning $-1 \leq x \leq 1$ /\ $-1 \leq y \leq 1$
(*C* = Boxes around origin)



Learning $-1 \leq x,y \leq 1$ from Minimum Counterexamples (dist from origin)



Types of Counterexamples

Assume there is a function **size: D → N**

- Maps each example x to a natural number
- Imposes total order amongst examples

- **CEGIS**: Arbitrary counterexamples
  - Any element of $f \oplus \phi$
- **MinCEGIS**: Minimal counterexamples
  - A least element of $f \oplus \phi$ according to **size**
  - Motivated by debugging methods that seek to find small counterexamples to explain errors & repair

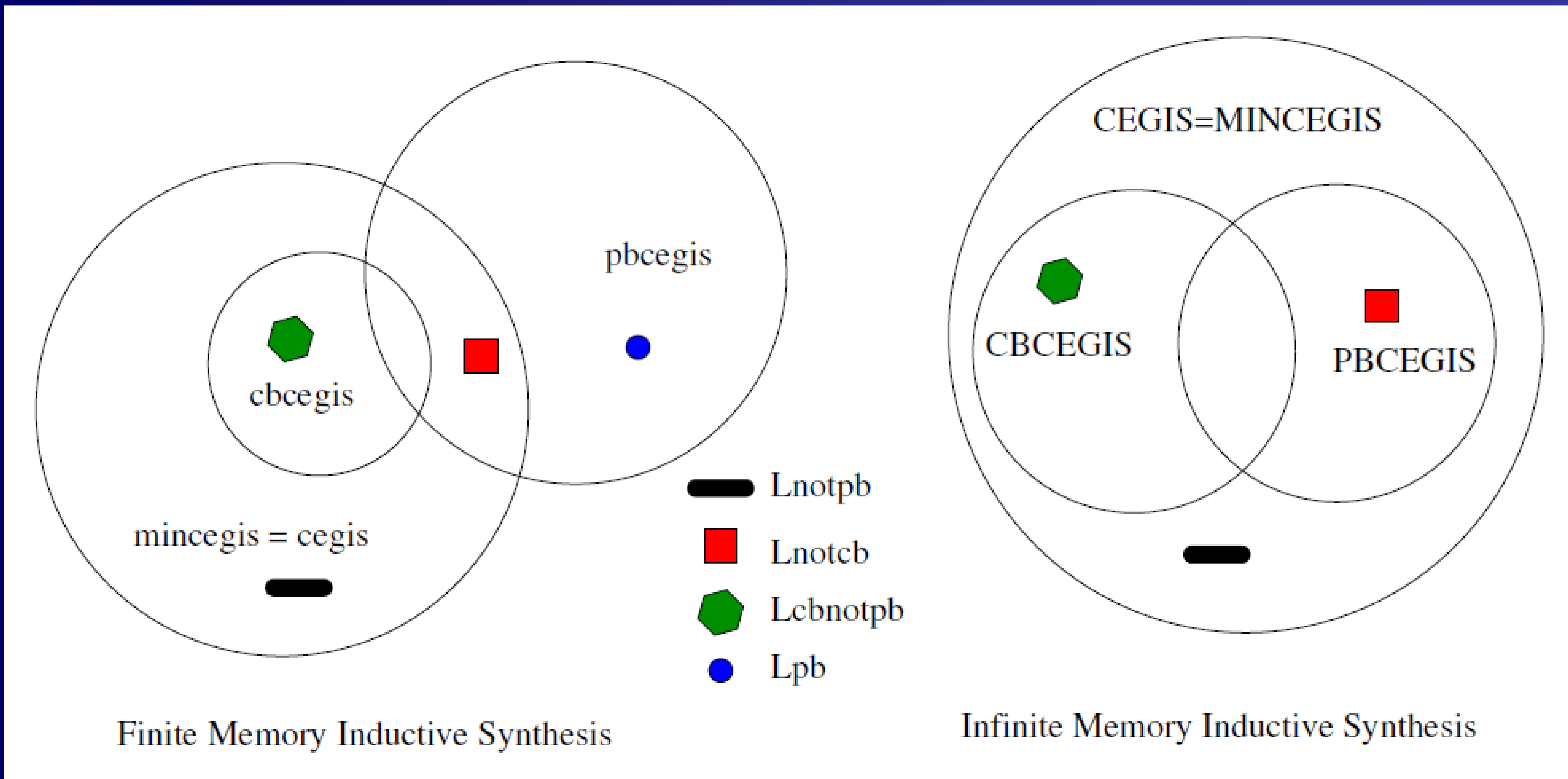
Types of Counterexamples

Assume there is a function size: $D \rightarrow N$

- **CBCEGIS**: Constant-bounded counterexamples (bound B)
  - An element x of $f \oplus \phi$ s.t. $size(x) < B$
  - Motivation: Bounded Model Checking, Input Bounding, Context bounded testing, etc.
- **PBCEGIS**: Positive-bounded counterexamples
  - An element x of $f \oplus \phi$ s.t. size(x) is no larger than that of any positive example seen so far
  - Motivation: bug-finding methods that mutate a correct execution in order to find buggy behaviors

Summary of Results

[Jha & Seshia, SYNT'14; TR'15]



Summary

- Verification by reduction to Synthesis
- Counterexample-guided Synthesis is Inductive Learning
- Teaching Dimension relevant for analyzing counterexample-guided learning
- Termination analysis for CEGIS can be non-trivial for infinite domains (concept classes)

- Lots of scope for future work in understanding efficiency / termination behavior of inductive learners based on deductive/verification oracles