# Formal Inductive Synthesis -- Theory and Applications

**Sanjit A. Seshia**

**EECS Department**

**UC Berkeley**

EECS 219C
April 22, 2019

# Formal Synthesis

- **Given:**
  - **Class of Artifacts C**
  - **Formal (mathematical) Specification $\phi$**

- **Find $f \in C$ that satisfies $\phi$**

- **Example 1:**
  - **C: all affine functions f of $x \in R$**
  - **$\phi$: $\forall x.\ f(x) \geq x + 42$**

- **Example 2: SyGuS**

# Induction vs. Deduction

- **Induction**: Inferring general rules (functions) from specific examples (observations)
  - Generalization
- **Deduction**: Applying general rules to derive conclusions about specific instances
  - (generally) Specialization
- **Learning/Synthesis** can be Inductive or Deductive or a combination of the two

# Inductive Synthesis

- Given
  - Class of Artifacts C
  - Set of (labeled) Examples E (or source of E)
  - A stopping criterion $\Psi$
    - May or may not be formally described
- Find, using only E, an f $\in$ C that meets $\Psi$
- Example:
  - C: all affine functions f of x $\in$ R
  - E = {(0,42), (1, 43), (2, 44)}
  - $\Psi$ -- find consistent f

# Inductive Synthesis

- Given
  - Class of Artifacts C
  - Set of Examples E (or source of E)
  - A stopping criterion $\Psi$
- Find using only E an $f \in C$ that meets $\Psi$

- Example:
  - C: all affine functions f of $x \in R$
  - E = {(0,42), (1, 43), (2, 45)}
  - $\Psi$ -- find consistent f

# Inductive Synthesis

- Example:
  - C: all predicates of the form $ax + by \geq c$
  - $E = \{(0,42), (1, 43), (2, 45)\}$
  - $\Psi$ -- find consistent f

- One such: $-x + y \geq 42$
- Another: $-x + y \geq 0$
- Which one to pick: need to augment $\Psi$?

# Machine Learning

- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

  \- Tom Mitchell [1998]

# Machine Learning: Typical Setup

**Given:**

- **Domain of Examples D**
- **Concept class C**
  - **Concept is a subset of D**
  - **C is set of all concepts**
- **Criterion $\Psi$ ("performance measure")**

**Find using only examples from D, $f \in C$ meeting $\Psi$**

# Inductive Bias in Machine Learning

"Inductive bias is the set of assumptions required to *deductively* infer a concept from the inputs to the learning algorithm."

Example:

C: all predicates of the form $ax + by \geq c$

$E = \{(0,42), (1, 43), (2, 45)\}$

$\Psi$ -- find consistent f

Which one to pick: $-x + y \geq 42$ or $-x + y \geq 0$

Inductive Bias resolves this choice

- E.g., pick the "simplest one" (Occam's razor)

# Formal Inductive Synthesis
## (Initial Defn)

- Given:
  - Class of Artifacts C
  - Formal specification $\phi$
  - Domain of examples D

- Find $f \in C$ that satisfies $\phi$ using only elements of D
  - i.e. no direct access to $\phi$, only to elements of D representing $\phi$

- Example:
  - C: all affine functions f of $x \in R$
  - $D = R^2$
  - $\phi$: $\forall x.\ f(x) \geq x + 42$

# Importance

Formal Inductive Synthesis is Everywhere!

– Many problems can be solved effectively when viewed as synthesis

Particularly effective in various tasks in Formal Methods

For the rest of this lecture series, for brevity we will often use “Inductive Synthesis” to mean “Formal Inductive Synthesis”

# Inductive Synthesis for Formal Methods

- **Modeling / Specification**
  – Generating environment/component models
  – Inferring (likely) specifications/requirements
- **Verification**
  – Synthesizing verification/proof artifacts such as inductive invariants, abstractions, interpolants, environment assumptions, etc.
- **Synthesis** (of course)

# Questions of Interest

- How can inductive synthesis be used to solve other (non-synthesis) problems?
- Is there a theory of formal inductive synthesis distinct from (traditional) machine learning?
- Is there a complexity/computability theory for formal inductive synthesis?

# Questions of Interest

- How can inductive synthesis be used to solve other (non-synthesis) problems?
  - Reducing a Problem to Synthesis
- Is there a theory of formal inductive synthesis distinct from (traditional) machine learning?
  - Oracle-Guided Inductive Synthesis (OGIS)
- Is there a complexity/computability theory for formal inductive synthesis?
  - Yes! Can compare different OGIS techniques

# Outline for this Lecture

- Examples of Reduction to Synthesis
  - Specification
  - Verification
- Differences between Inductive Synthesis and Machine Learning
- Oracle-Guided Inductive Synthesis
  - Examples, CEGIS
- Theoretical Analysis of CEGIS
  - Properties of Learner
  - Properties of Verifier

# Further Reading

- S. A. Seshia, **“Combining Induction, Deduction, and Structure for Verification and Synthesis**.”, Proc. IEEE 2015, DAC 2012

  http://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-seshia-dac12.html

  http://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-seshia-pieee15.html

- S. Jha and S. A. Seshia, “**A Theory of Formal Synthesis via Inductive Learning”**

  http://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-jha-arxiv15.html

  https://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-jha-acta17.html

# Reductions to Synthesis

# Artifacts Synthesized in Verification

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

# Example Verification Problem

- Transition System
  - Init: $\mathbf{I}$

$$x = 1 \wedge y = 1$$

  - Transition Relation: $\delta$

$$x' = x+y \;\wedge\; y' = y+x$$

- Property: $\Psi$ = **G** $(y \geq 1)$
- Attempted Proof by Induction:

$$y \geq 1 \wedge\; x' = x+y \;\wedge\; y' = y+x \;\;\Rightarrow\;\; y' \geq 1$$

- Fails. Need to Strengthen Invariant: Find $\phi$ s.t.

$$x = 1 \wedge y = 1 \Rightarrow \;\phi$$

$$\phi \wedge y \geq 1 \wedge\; x' = x+y \;\wedge\; y' = y+x \;\;\Rightarrow\;\; \phi' \wedge y' \geq 1$$

# Example Verification Problem

- Transition System
  - Init: $\mathbf{I}$

    $$x = 1 \wedge y = 1$$

  - Transition Relation: $\delta$

    $$x' = x+y \;\wedge\; y' = y+x$$

- Property: $\Psi$ = **G** $(y \geq 1)$
- Attempted Proof by Induction:

  $$y \geq 1 \wedge\; x' = x+y \;\wedge\; y' = y+x \;\Rightarrow\; y' \geq 1$$

- Fails. Need to Strengthen Invariant: Find $\phi$ s.t.

  $$x \geq 1 \wedge y \geq 1 \wedge\; x' = x+y \;\wedge\; y' = y+x \;\Rightarrow\; x' \geq 1 \wedge y' \geq 1$$

- **Safety Verification → Invariant Synthesis**

# One Reduction from Verification to Synthesis

NOTATION

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

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

VERIFICATION PROBLEM

Does $M$ satisfy $\Psi$?

$\Downarrow$

SYNTHESIS PROBLEM

Synthesize $\phi$ 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 $\Psi$?

SYNTHESIS PROBLEM #1

Synthesize $\phi$ 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 $\Psi$

iff

M satisfies $\Psi$

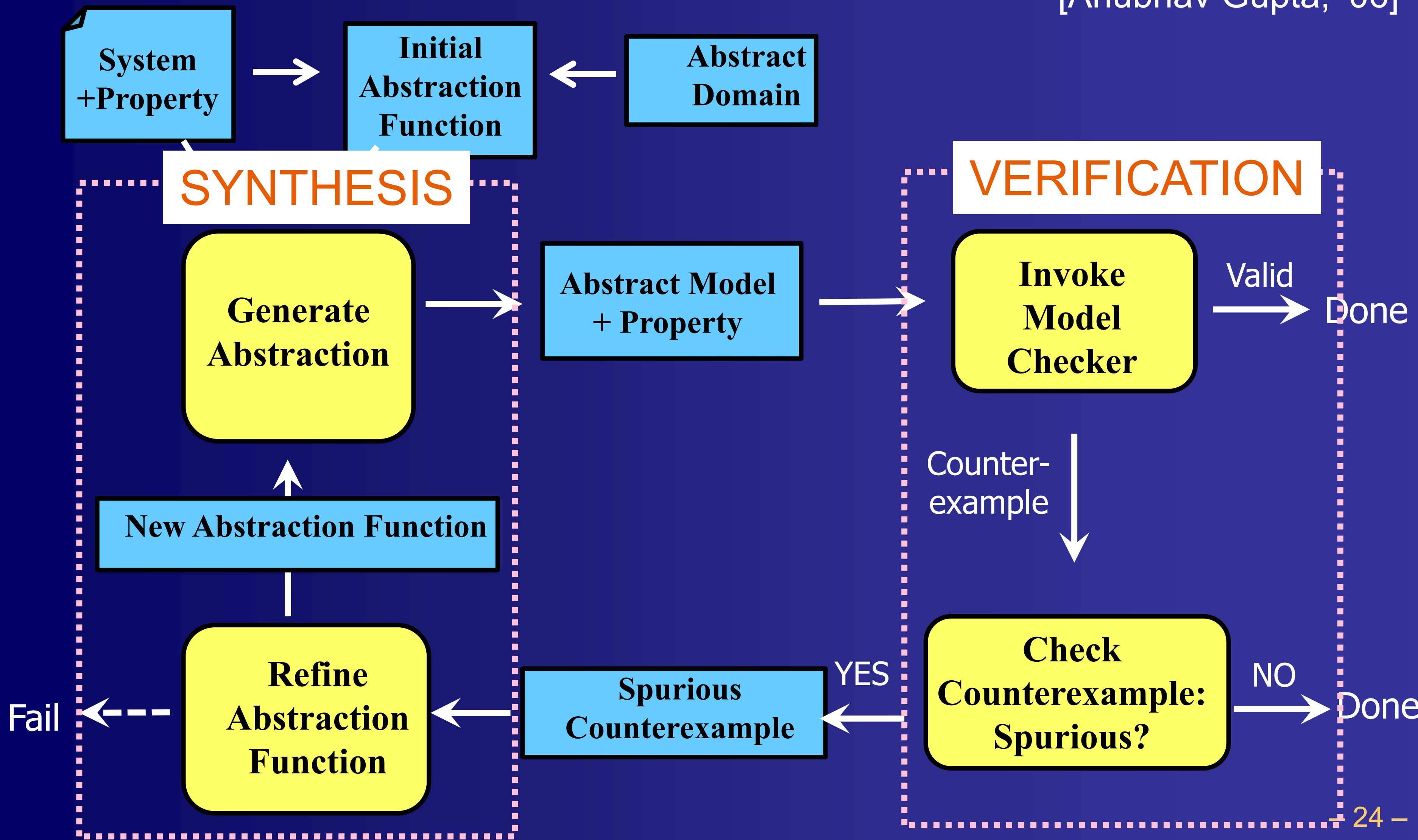
# Common Approach for both: Inductive Synthesis

Synthesis of:-

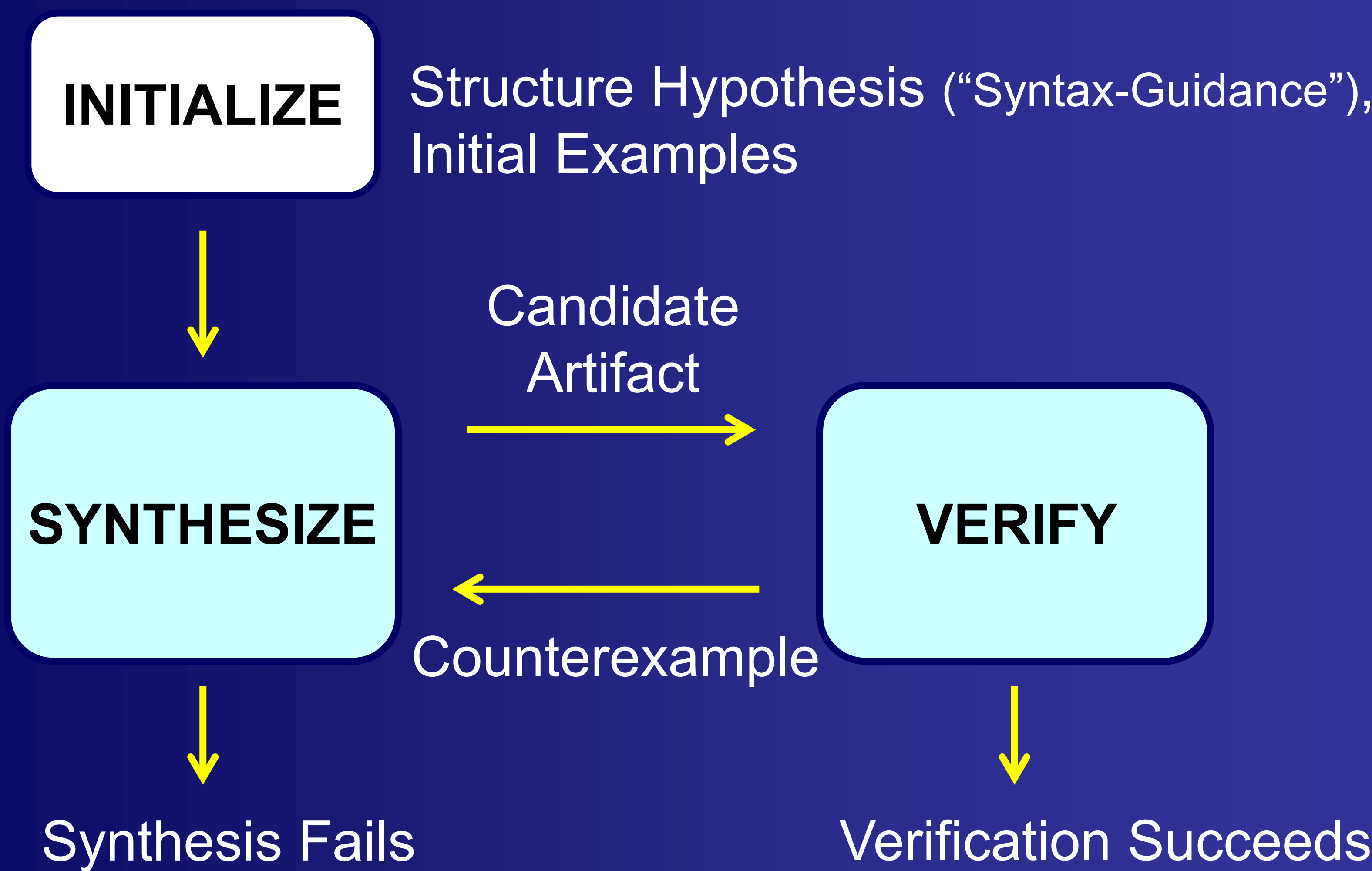
- **Inductive Invariants**
  - Choose templates for invariants
  - Infer likely invariants from tests (examples)
  - Check if any are true inductive invariants, possibly iterate
- **Abstraction Functions**
  - Choose an abstract domain
  - Use Counter-Example Guided Abstraction Refinement (CEGAR)

# Counterexample-Guided Abstraction Refinement is Inductive Synthesis

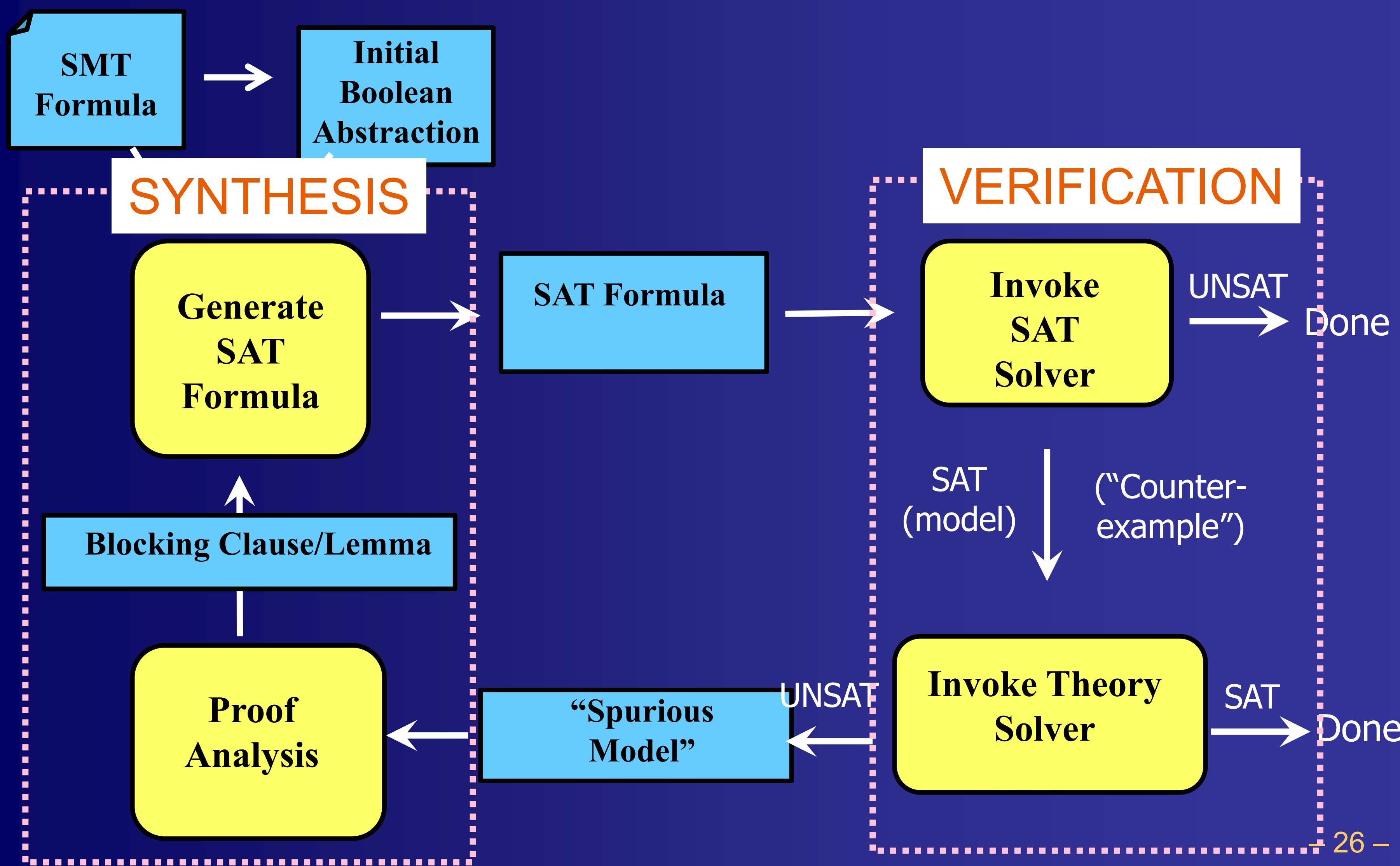
[Anubhav Gupta, '06]



# CEGAR = Counterexample-Guided Inductive Synthesis (of Abstractions)



# Lazy SMT Solving performs Inductive Synthesis (of Lemmas)



# Other Examples

- **Invariant Generation via ICE Learning** [P. Garg & M. Parthasarathy]
- **Invariant Generation, Interpolation via Machine Learning + SMT Solving** [R. Sharma, A. Aiken, et al.]

**and many more…**

# Reducing Specification to Synthesis

- Formal Specifications difficult for non-experts
- Tricky for even experts to get right!
- Yet we need them!

"*A design without specification cannot be right or wrong, it can only be surprising!*"

– paraphrased from [Young et al., 1985]

- Specifications are crucial for effective testing, verification, synthesis, …

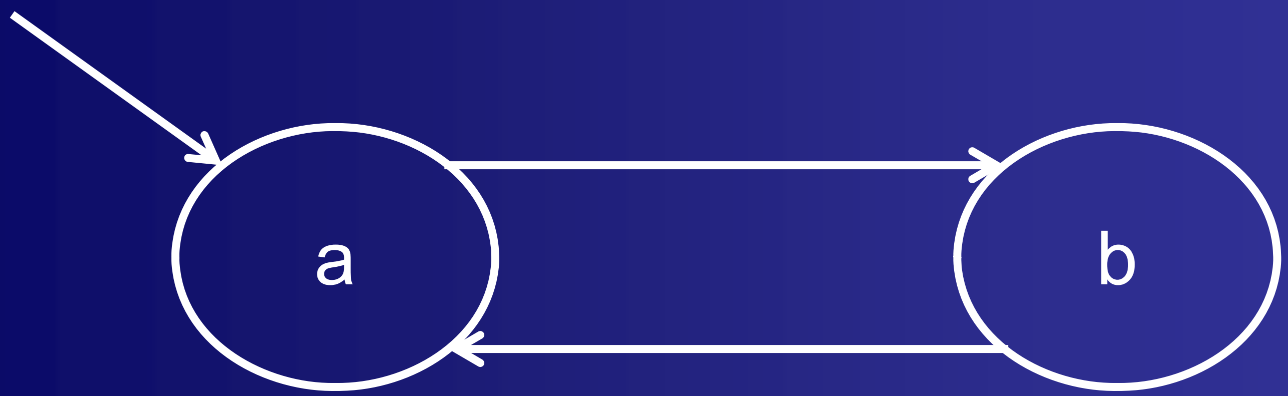
# Reduction of Specification to Synthesis

- **VERIFICATION: Given (closed) system M, and specification $\phi$, does M satisfy $\phi$?**

- **Suppose we don't have (a good enough) $\phi$.**

- **SYNTHESIS PROBLEM: Given (closed) system M, find specification $\phi$ such that M satisfies $\phi$.**
  - **Is this enough?**

# Example




Let a and b be atomic propositions.

What linear temporal logic formulas does the above system satisfy?

# Reduction of Specification to Synthesis

- **VERIFICATION: Given (closed) system M, and specification $\phi$, does M satisfy $\phi$?**

- **SYNTHESIS PROBLEM: Given (closed) system M and class of specifications C, find specification $\phi$ in C such that M satisfies $\phi$.**
  - **C can be defined syntactically (e.g. with a template)**
  - **E.g. $G(\_ \Rightarrow X\ \_)$**

# Reduction of Specification to Synthesis

- **VERIFICATION: Given (closed) system M, and specification $\phi$, does M satisfy $\phi$?**

- **SYNTHESIS PROBLEM: Given (closed) system M and class of specifications C, find "tightest" specification $\phi$ in C such that M satisfies $\phi$.**
  - Industrial Tech. Transfer Story: Requirement Synthesis for Automotive Control Systems [Jin, Donze, Deshmukh, Seshia, HSCC 2013, TCAD 2015]

    http://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-jin-tcad15.html
  - Implemented in Breach toolbox by A. Donze

# Specification Mining

- Inductive Synthesis of Specifications
- Term coined by Ammons et al., POPL 2002 (?)
- See recent Ph.D. dissertation by Wenchao Li: "Specification Mining: New Formalisms, Algorithms and Applications"

http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-20.html

# Two Applications of Inductive Synthesis of Specifications

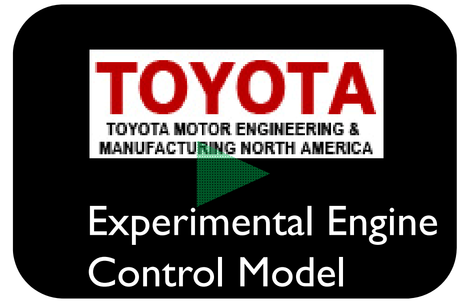
1. Requirements Mining for Closed-Loop Control Systems

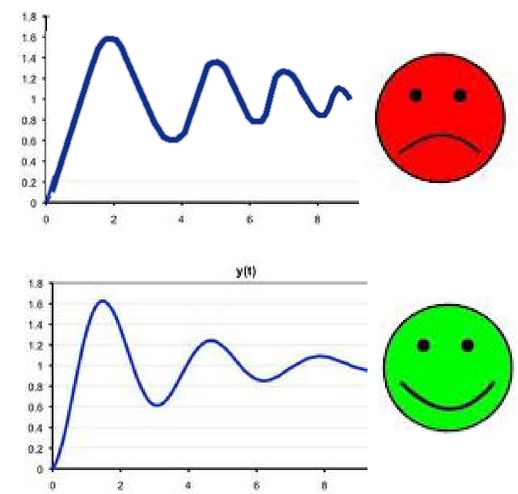
2. Environment Assumptions for Reactive Synthesis [see Wenchao Li thesis]

- Relevance to Robotics/Cyber-Physical Systems

# Challenges for Verification of Automotive Control Systems

- Closed-loop setting very complex
  - software + physical artifacts
  - nonlinear dynamics
  - large look-up tables
  - large amounts of switching
- Requirements Incomplete/Informal
  - Specifications often created concurrently with the design!
  - Designers often only have informal intuition about what is “good behavior”
    - “shape recognition”



# Solution: Requirements Mining

Requirements Expressed in Signal Temporal Logic (STL) [Maler & Nickovic, '04]

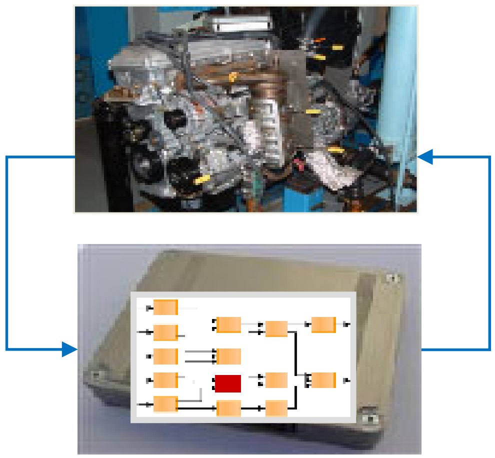
Value added by mining:

- Mined Requirements become useful documentation
- Use for code maintenance and revision
- Use during tuning and testing

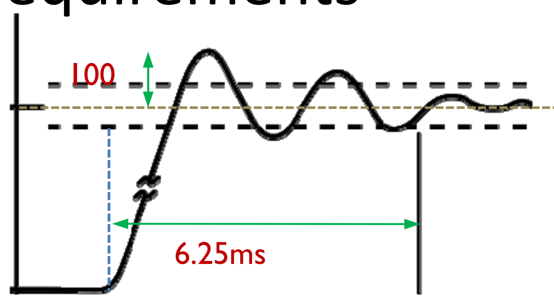
It's working, but I don't understand why!

# Control Designer's Viewpoint of the Method

- Tool extracts properties of closed-loop design



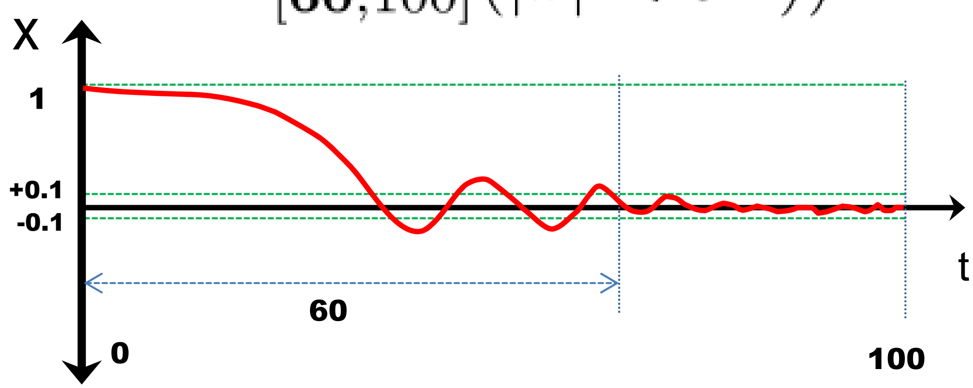
- Designer reviews mined requirements
  - "Settling time is 6.25 ms"
  - "Overshoot is 100 units"
  - Expressed in Signal Temporal Logic [Maler & Nickovic, '04]




# Signal Temporal Logic (STL)

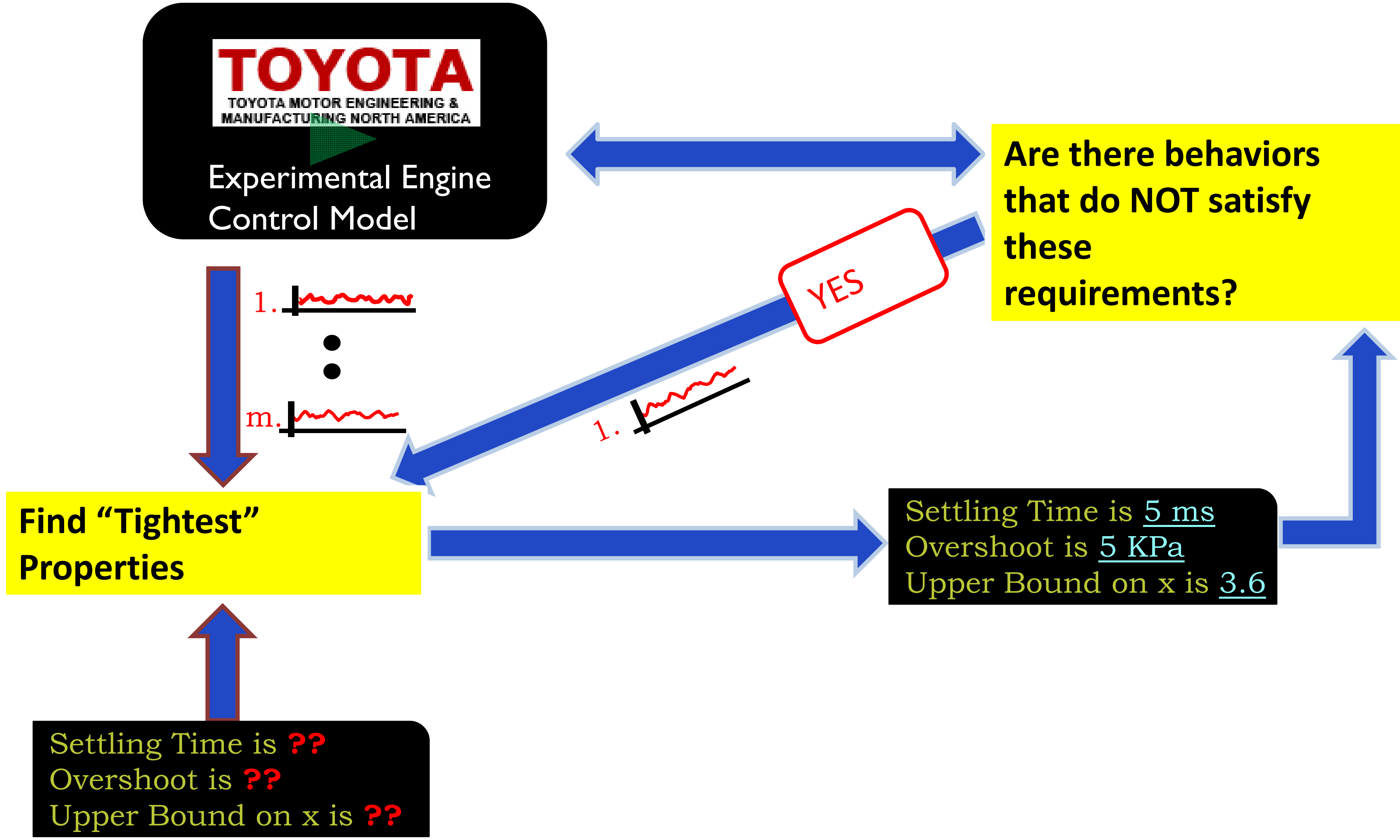
- Extension of Linear Temporal Logic (LTL) and Variant of Metric Temporal Logic (MTL)
  - Quantitative semantics: satisfaction of a property over a trace given real-valued interpretation
  - Greater value → more easily satisfied
  - Non-negative satisfaction value ≡ Boolean satisfaction
- Example: *"For all time points between 60 and 100, the absolute value of $x$ is below 0.1"*

$$\Box_{[\mathbf{60},100]}(|x| < 0.1))$$

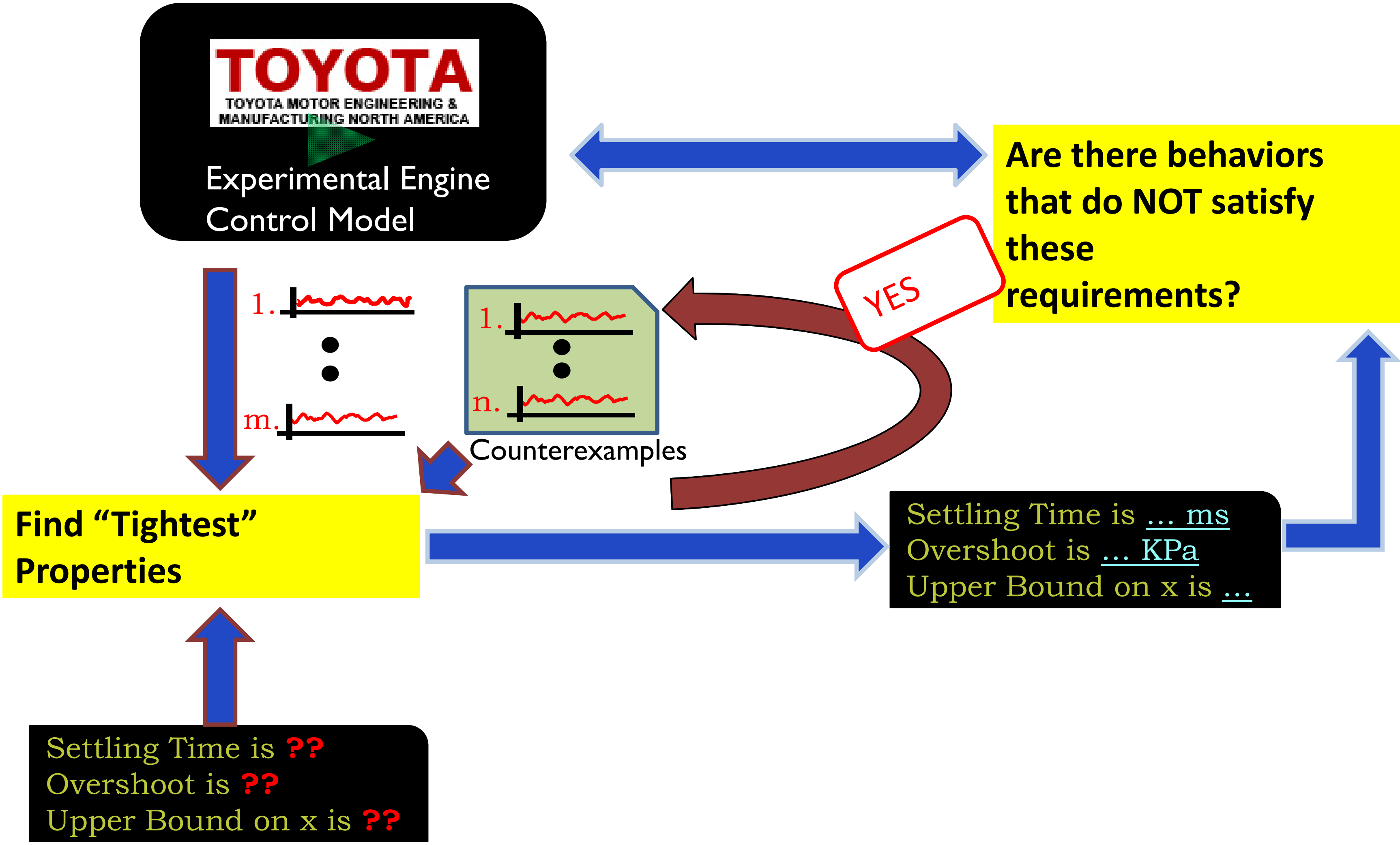


# CounterExample Guided Inductive Synthesis

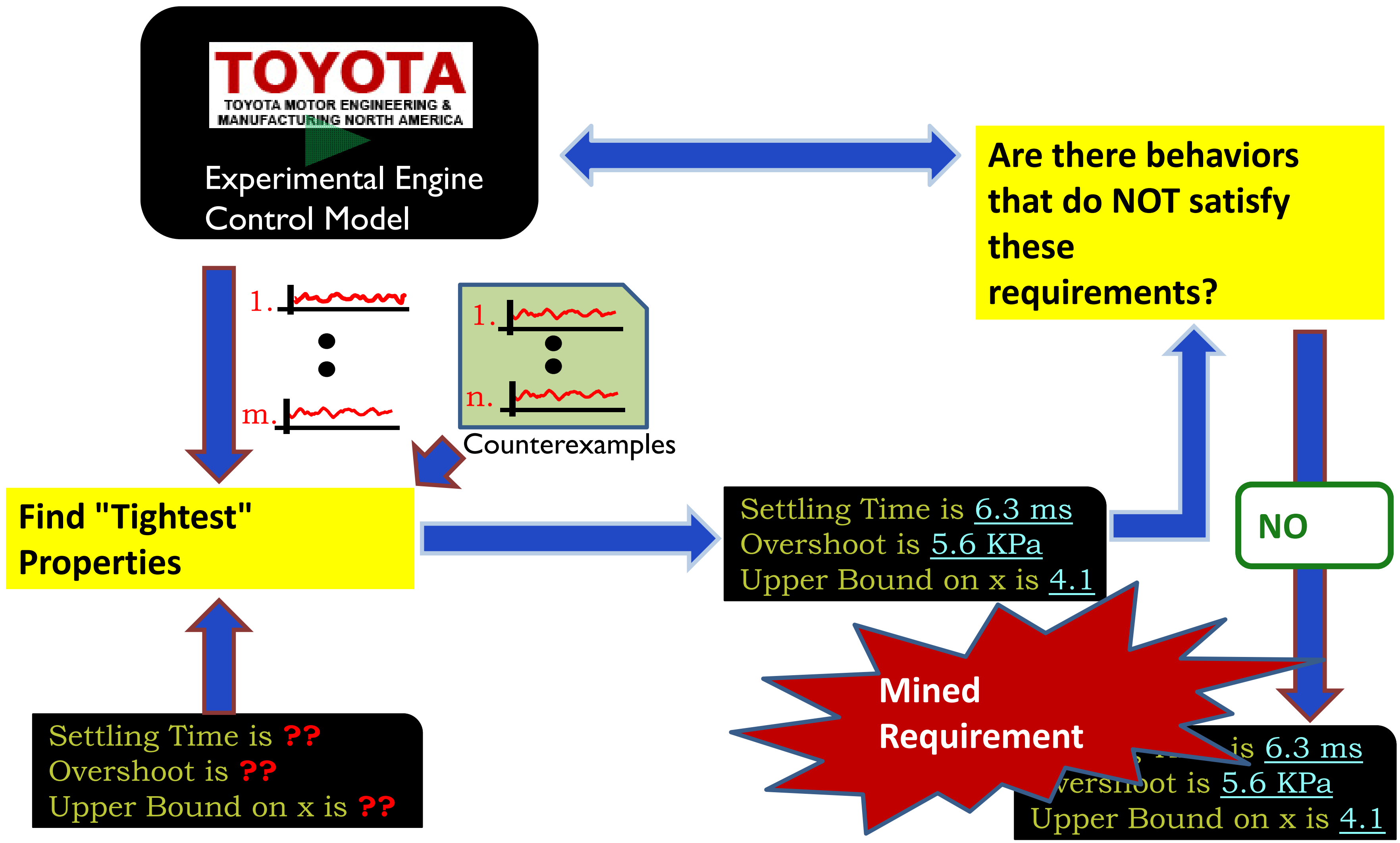
[Jin, Donze, Deshmukh, Seshia, HSCC 2013]



# CounterExample Guided Inductive Synthesis

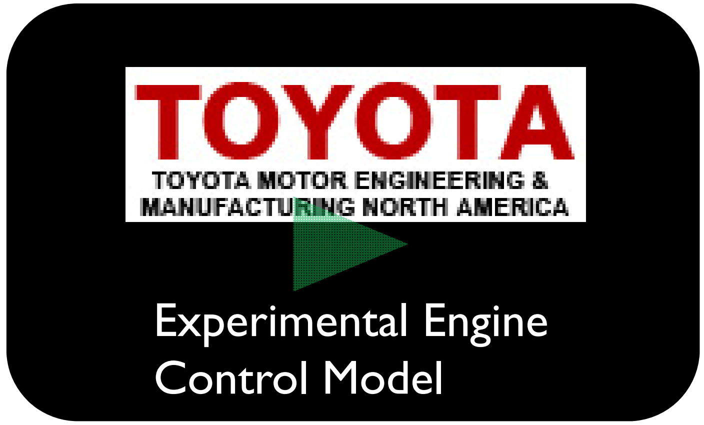


# CounterExample Guided Inductive Synthesis

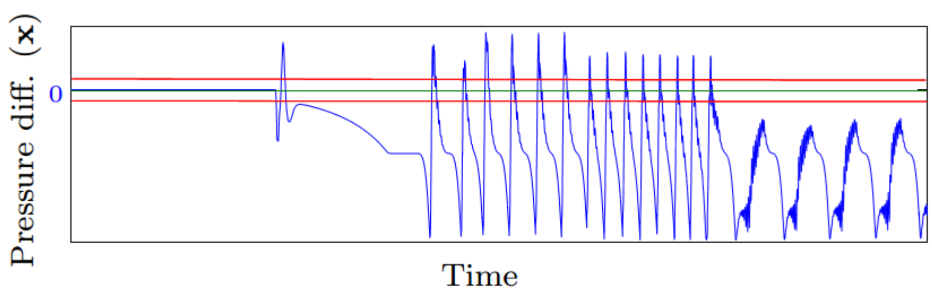


# Experimental Results on Industrial Airpath Controller

[Jin, Donze, Deshmukh, Seshia, HSCC 2013]




- Found max overshoot with 7000+ simulations in 13 hours
- Attempt to mine maximum observed settling time:
  - stops after 4 iterations
  - gives answer $t_{settle}$ = simulation time horizon (shown in trace below)



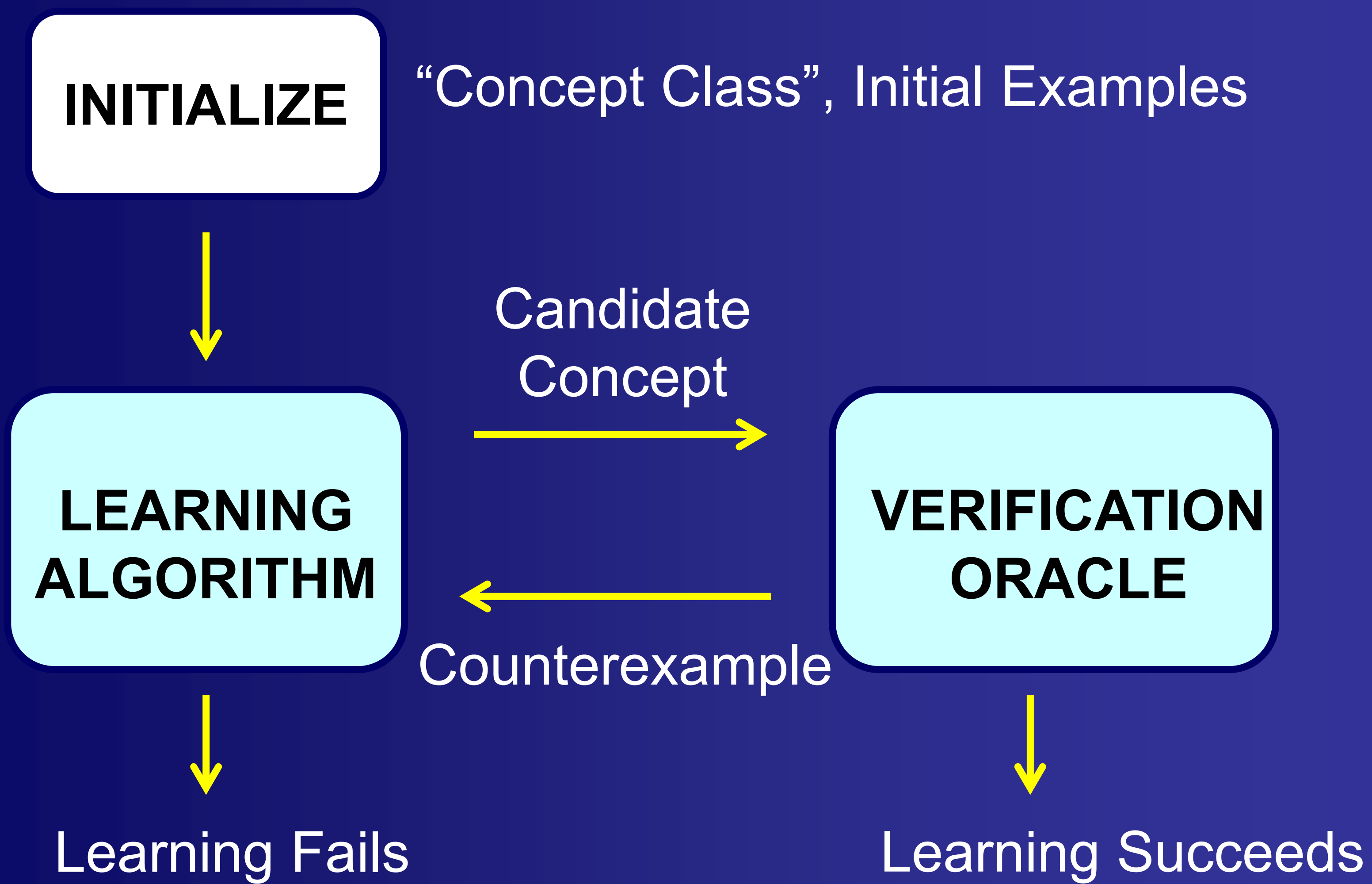
# Mining can expose deep bugs

TOYOTA
TOYOTA MOTOR ENGINEERING & MANUFACTURING NORTH AMERICA
Experimental Engine Control Model


- Uncovered a tricky bug
  - Discussion with control designer revealed it to be a real bug
  - Root cause identified as wrong value in a look-up table, bug was fixed
- Duality between spec mining and bug-finding:
  - Synthesizing “tightest” spec  could uncover corner-case bugs
  - Looking for bugs ≈ Mine for negation of bug

# Theoretical Aspects of Formal Inductive Synthesis

# CEGIS = Learning from Examples & Counterexamples



# How is Formal Inductive Synthesis different from (traditional) Machine Learning?

# Comparison*

[see also, Jha & Seshia, 2015]

| 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 select Oracle)* | *Rare (black-box oracles)* |

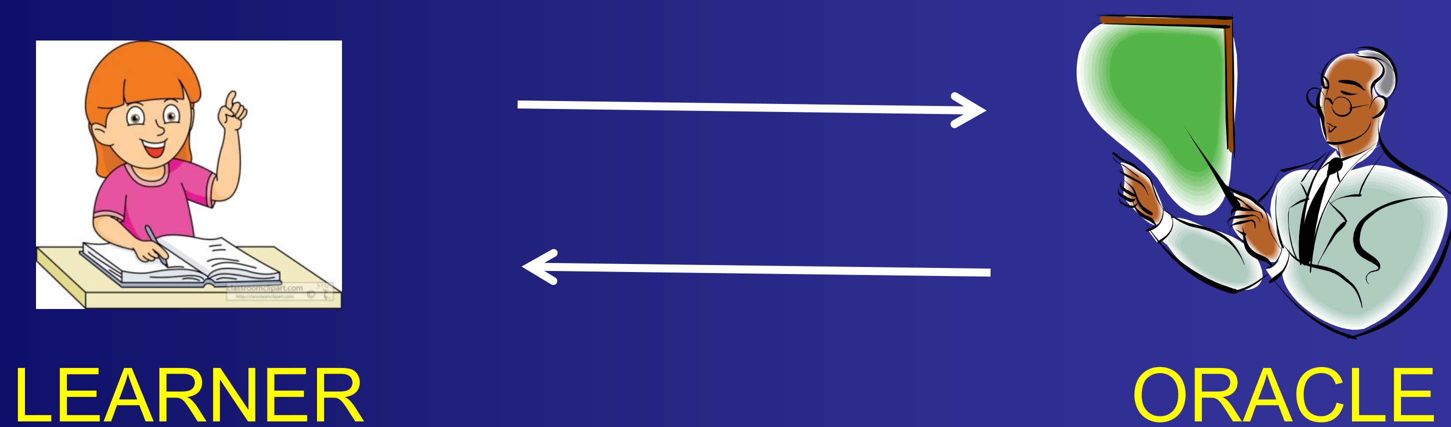
* Between typical inductive synthesizer and machine learning algo

# Formal Inductive Synthesis

- Given:
  - Class of Artifacts C -- Formal specification $\phi$
  - Domain of examples D
  - Oracle Interface O
    - Set of (query, response) types
- Find using only O an $f \in C$ that satisfies $\phi$
  - i.e. no direct access to D or $\phi$
- Example:
  - C: all affine functions f of $x \in R = D$
  - O = {(pos-witness, x satisfying $\phi$)}
  - $\phi$: $\forall x.\ f(x) \geq x + 42$

# Oracle Interface

- **Generalizes the simple model of sampling positive/negative examples from a corpus of data**




- **Specifies WHAT the learner and oracle do**
- **Does *not* specify HOW the oracle/learner is implemented**

# Common Oracle Query Types
## (for trace property $\phi$)

Positive Witness

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

Negative Witness

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

Membership: Is $x \in \phi$?

Yes / No

Equivalence: Is $f = \phi$?

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

Subsumption/Subset: Is $f \subseteq \phi$?

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

Distinguishing Input: $f$, $X \subseteq f$

$f'$ s.t. $f' \neq f \wedge X \subseteq f'$, if it exists; o.w. $\perp$

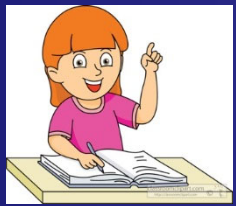
LEARNER

ORACLE

# Formal Inductive Synthesis

- Given:
  - Class of Artifacts C  -- Formal specification $\phi$
  - Domain of examples D
  - Oracle Interface O
    - Set of (query, response) types
- Find using only O an $f \in C$ that satisfies $\phi$
  - i.e. no direct access to D or $\phi$

- How do we solve this?

Design/Select:  

# Oracle-Guided Inductive Synthesis (OGIS)

- A **dialogue** is a sequence of (query, response) confirming to an oracle interface O
- An **OGIS engine** is a pair **<L, T>** where
  - L is a learner, a non-deterministic algorithm mapping a dialogue to a concept c and query q
  - T is an oracle/teacher, a non-deterministic algorithm mapping a dialogue and query to a response r
- An OGIS engine <L,T> solves an FIS problem if **there exists a dialogue between L and T that converges** in a concept $f \in C$ that satisfies $\phi$

# Language Learning in the Limit

INFORMATION AND CONTROL 10, 447–474 (1967)

**Language Identification in the Limit**

E Mark Gold*

*The RAND Corporation*

Language learnability has been investigated. This refers to the following situation: A class of possible languages is specified, together with a method of presenting information to the learner about an unknown language, which is to be chosen from the class. The question is now asked, "Is the information sufficient to determine which of the possible languages is the unknown language?" Many definitions of learnability are possible, but only the following is considered here: Time is quantized and has a finite starting time. At each time the learner receives a unit of information and is to make a guess as to the identity of the unknown language on the basis of the information received so far. This process continues forever. The class of languages will be considered *learnable* with respect to the specified method of information presentation if there is an algorithm that the learner can use to make his guesses, the algorithm having the following property: Given any language of the class, there is some finite time after which the guesses will all be the same and they will be correct.

In this preliminary investigation, a *language* is taken to be a set of strings on some finite alphabet. The alphabet is the same for all languages of the class. Several variations of each of the following two basic methods of information presentation are investigated: A *text* for a language generates the strings of the language in any order such that every string of the language occurs at least once. An *informant* for a language tells whether a string is in the language, and chooses the strings in some order such that every string occurs at least once.

It was found that the class of context-sensitive languages is learnable from an informant, but that not even the class of regular languages is learnable from a text.

1. MOTIVATION: TO SPEAK A LANGUAGE

The study of language identification described here derives its motivation from artificial intelligence. The results and the methods used also

**[E.M. Gold, 1967]**

- **Concept = Formal Language**
- **Class of languages identifiable in the limit if there is a learning procedure that, for each language in that class, given an infinite stream of strings, will eventually generate a representation of the language.**
- **Results:**
  - **Cannot learn regular languages, CFLs, CSLs using just positive witness queries**
  - **Can learn using both positive & negative witness queries (assuming all examples eventually enumerated)**

# Query-Based Learning

Dana Angluin

[Queries and Concept Learning, 1988]

[Queries Revisited, 2004]

- First work on learning based on querying an oracle
  - Supports witness, equivalence, membership, subsumption/subset queries
  - Oracle is BLACK BOX
  - Oracle determines correctness: No separate correctness condition or formal specification
  - Focus on proving complexity results for specific concept classes
- Sample results
  - Can learn DFAs in poly time from membership and equivalence queries
  - Cannot learn DFAs or DNF formulas in poly time with just equivalence queries

# Examples of OGIS

- **L* algorithm to learn DFAs: counterexample-guided**
  - **Membership + Equivalence queries**
- **CEGIS used in SyGuS solvers**
  - **(positive) Witness + Counterexample/Verification queries**
- **CEGIS for Hybrid Systems**
  - **Requirement Mining [HSCC 2013]**
  - **Reactive Model Predictive Control [HSCC 2015]**
- **Two different examples:**
  - **Learning Programs from Distinguishing Inputs** [Jha et al., ICSE 2010]
  - **Learning LTL Properties for Synthesis from Counterstrategies** [Li et al., MEMOCODE 2011]

# Revisiting the Comparison

| Feature | Formal Inductive Synthesis | Machine Learning |
|---|---|---|
| Concept/Program | | imple |
| | | alized |
| Lea | | mate, w/ unction |
| Ora | | lack-box les) |

What can we prove about convergence/complexity of *formal* inductive synthesis for:

- General concept classes (e.g., recursive languages)
- Different properties of "general-purpose" learners
- Different properties of (non black-box) oracles

# Query Types for CEGIS

LEARNER

Positive Witness

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

ORACLE

Equivalence: Is $f = \phi$?

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

Subset: 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

# Questions

- **Convergence:** How do properties of the learner and oracle impact convergence of CEGIS? (learning in the limit for infinite-sized concept classes)

- **Sample Complexity:** For finite-sized concept classes, what upper/lower bounds can we derive on the number of oracle queries, for various CEGIS variants?

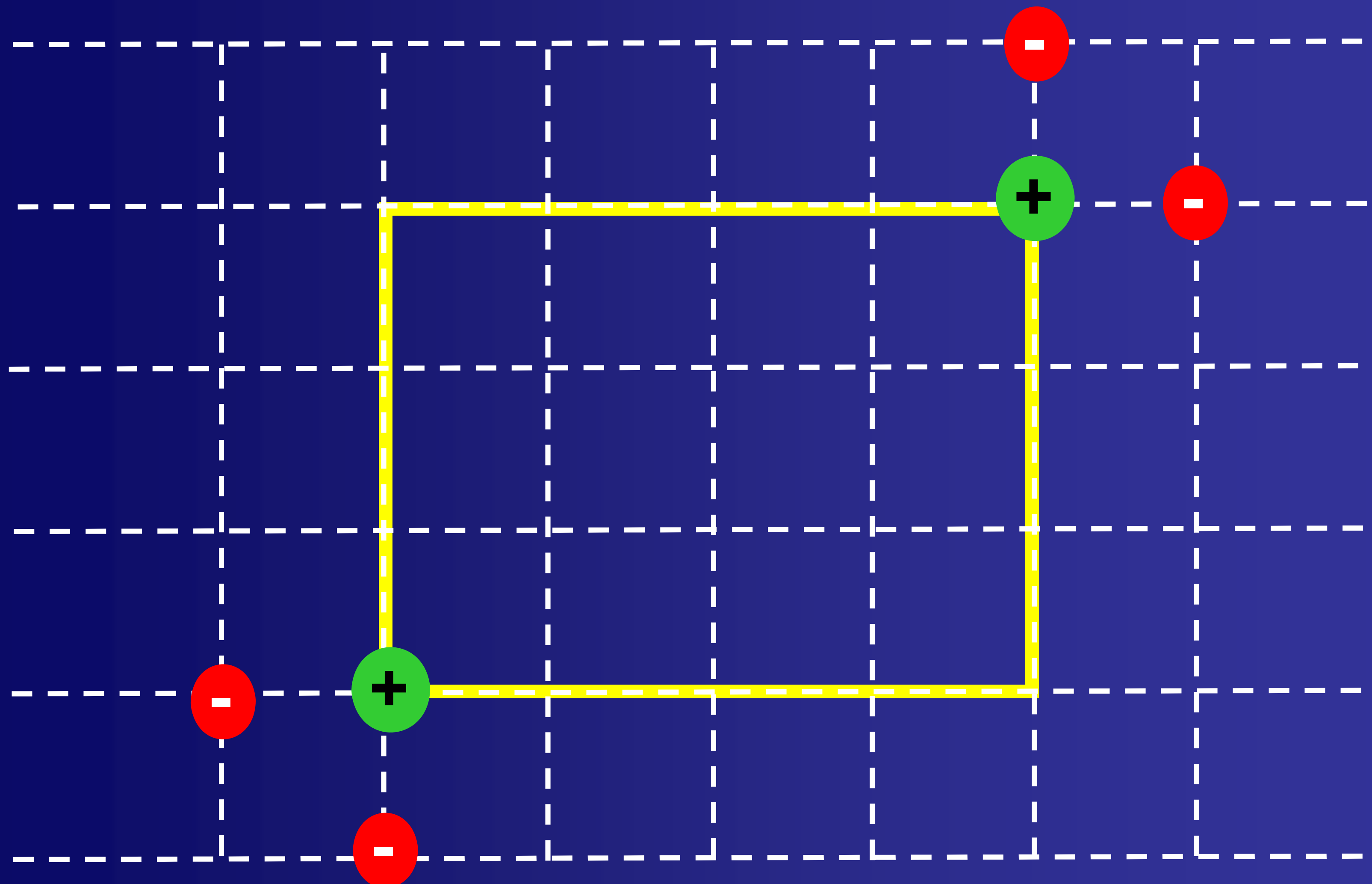
# Problem 1: Bounds on Sample Complexity

# 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**

$\sigma$ **is a teaching sequence (uniquely identifies concept $c$)**

$\Sigma$ **is the set of all teaching sequences**

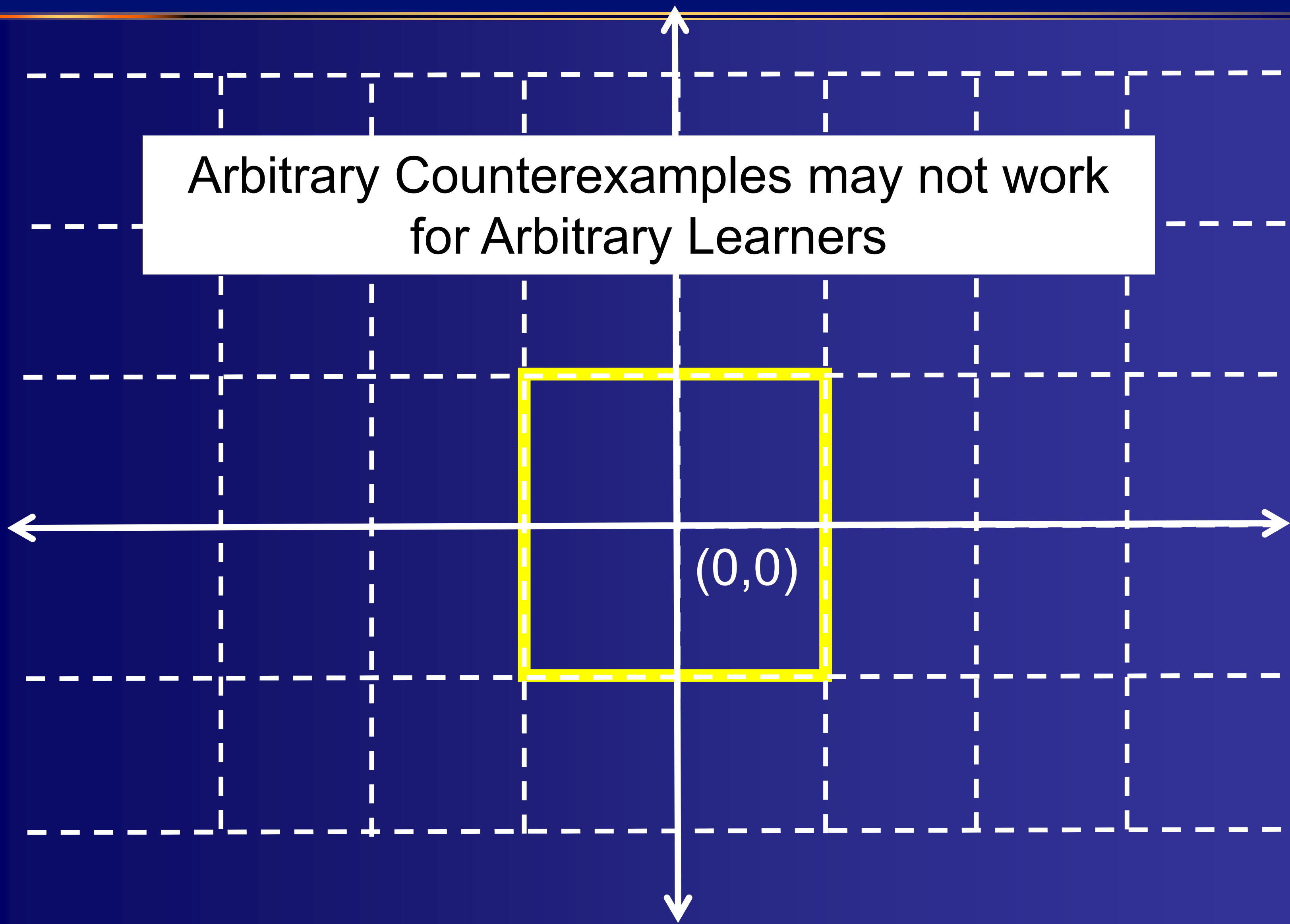
# Theorem: *TD*(*C*) is lower bound on Sample Complexity

- OGIS: TD gives a lower bound on number of counterexample queries to solve FIS problem
- 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)
  - Hence also finding optimal query sequence for OGIS
  - But heuristic approach works well ("learning from distinguishing inputs")
- Open Problems: Compute TD for common classes of SyGuS problems

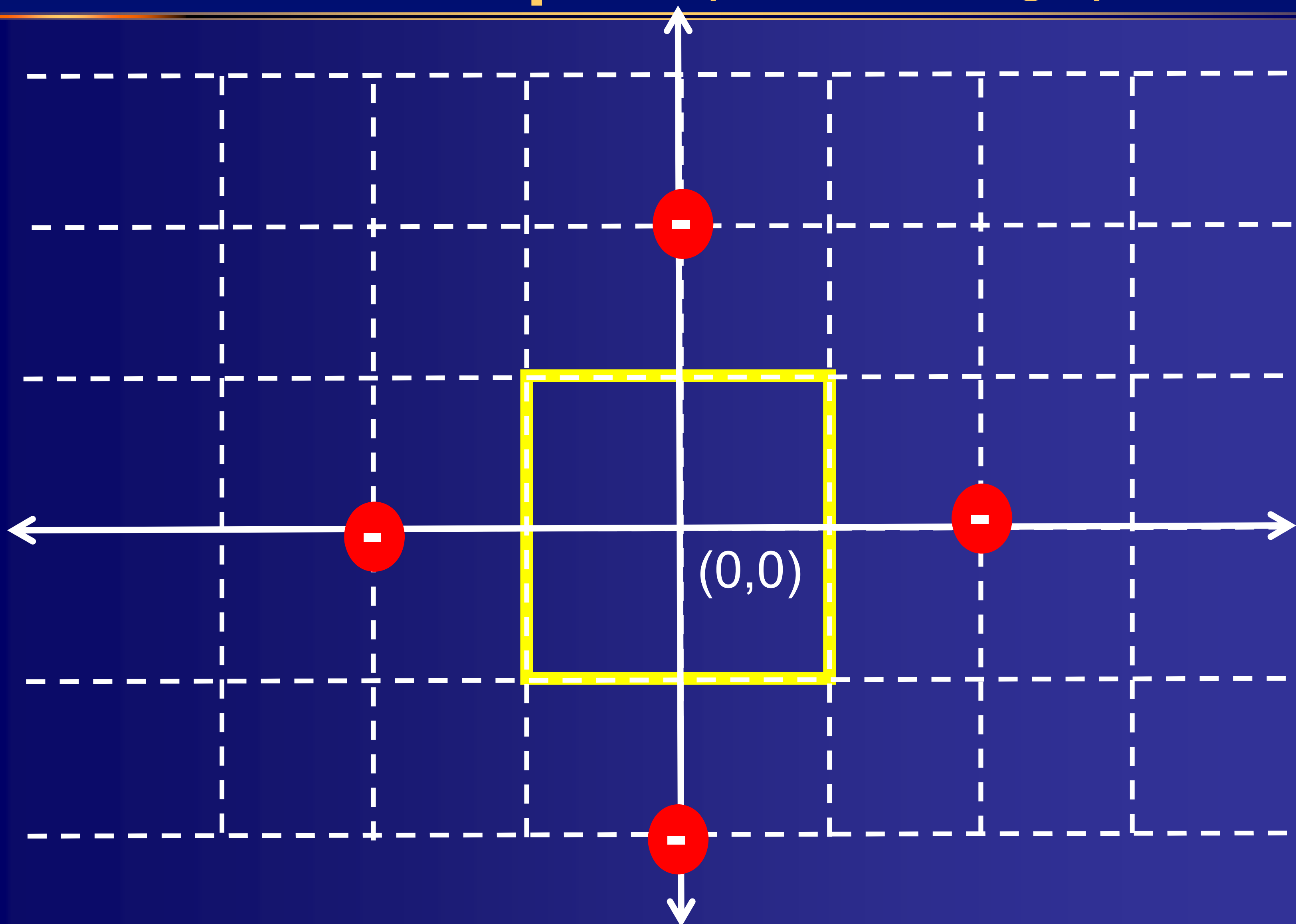
[see Jha & Seshia, 2015]

# Problem 2: Convergence of Counterexample-guided loop with positive witness and counterexample/verification queries

# Learning $-1 \le x \le 1 \wedge -1 \le y \le 1$ (*C* = Boxes around origin)



# Learning -1 ≤ $x,y$ ≤ 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

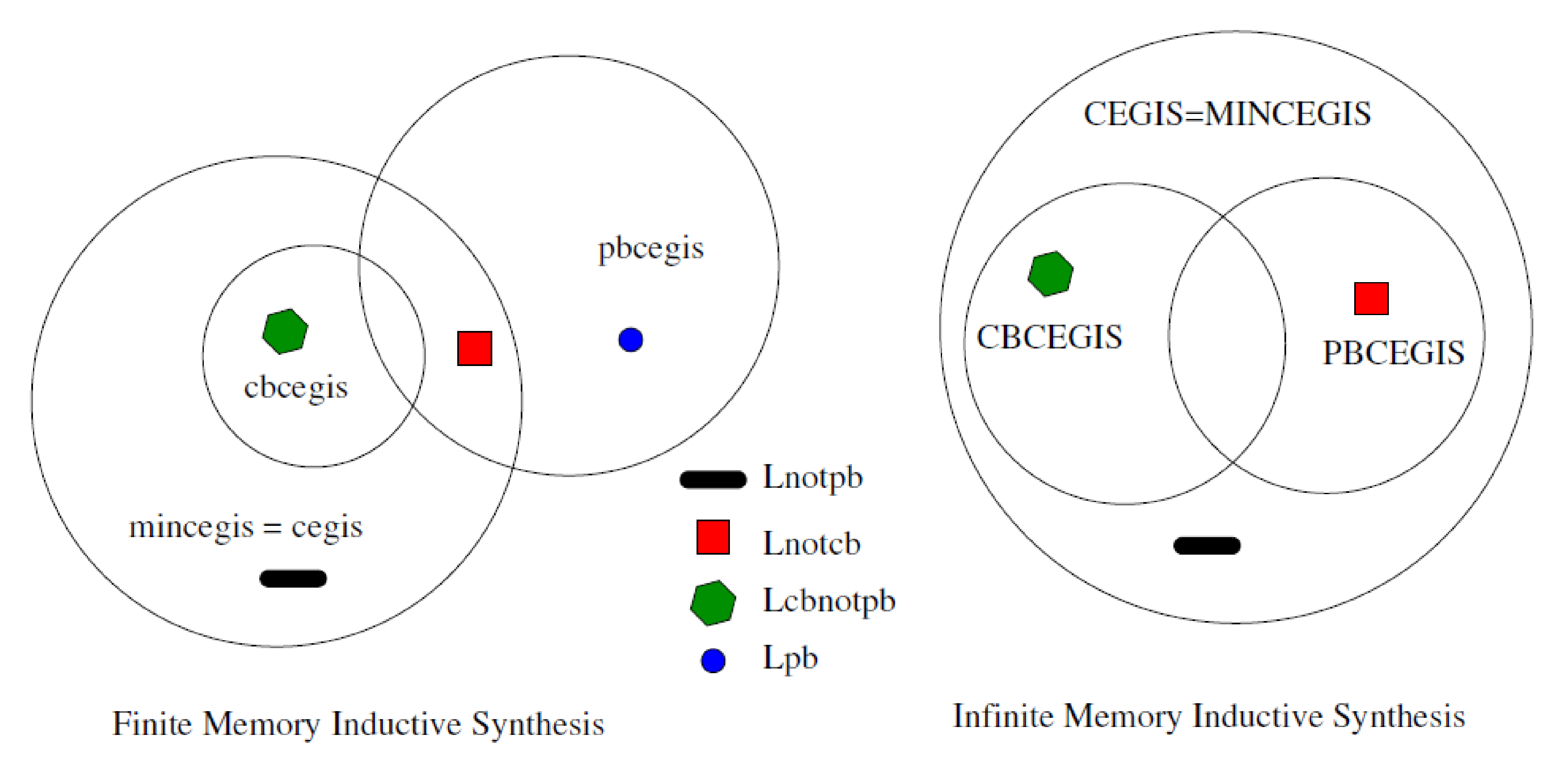
# Types of Counterexamples

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

- **CBCEGIS**: Constant-bounded counterexamples (bound B)
  - An element x of $f \oplus \phi$ s.t. $\mathrm{size}(x) < B$
  - 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]



# Open Problems

- **For Finite Domains: What is the impact of type of counterexample and buffer size to store counterexamples on the speed of termination of CEGIS?**

- **For Specific Infinite Domains (e.g., Boolean combinations of linear real arithmetic): Can we prove termination of CEGIS loop?**

# Summary

- Formal Synthesis
- Verification by Reduction to Synthesis
- Formal Inductive Synthesis
  - Counterexample-guided inductive synthesis (CEGIS)
  - General framework for solution methods: Oracle-Guided Inductive Synthesis (OGIS)
  - Theoretical analysis
- Lots of potential for future work!