Solvers, Synthesis, and Learning

Sanjit A. Seshia

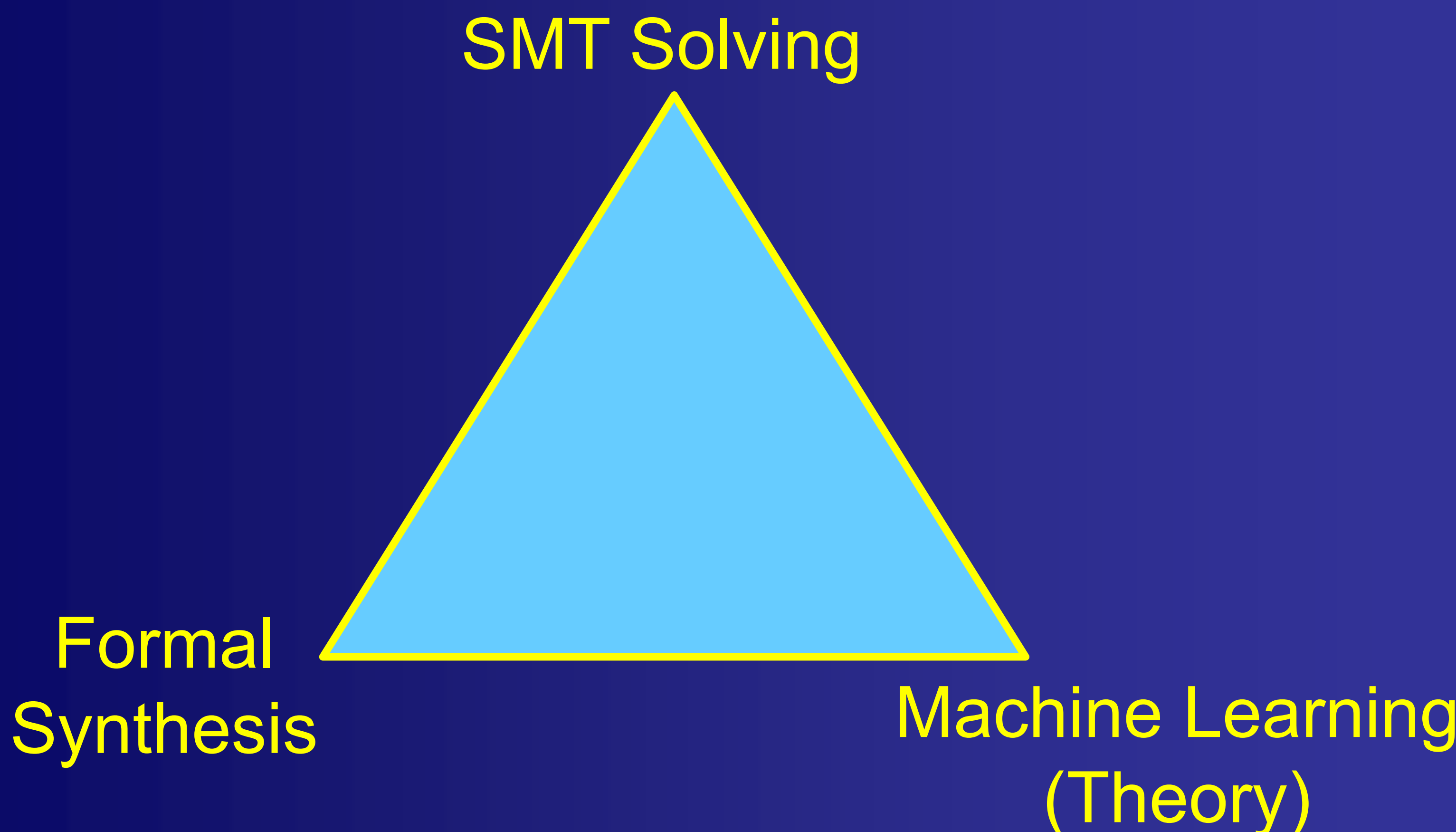
EECS Department

UC Berkeley

Acknowledgments to several Ph.D. students, postdoctoral researchers, and collaborators, and to the students of EECS 219C, Spring 2015, UC Berkeley

SAT/SMT Summer School
July 17, 2015

Connections in this Lecture



Outline

- **Formal Synthesis & Applications**
- **Syntax-Guided Synthesis (SyGuS)**
- **Inductive Synthesis**
  - **Counterexample-Guided Inductive Synthesis (CEGIS)**
- **Conclusion**

Formal Methods ≈ Computational Proof Methods

- Formal Methods is about Provable Guarantees
  - Specification/Modeling ≈ Statement of Conjecture/Theorem
  - Verification ≈ Proving/Disproving the Conjecture
  - Synthesis ≈ Generating (parts of) Conjecture/Proof

Formal Synthesis

- **Given:**
  - **Class of Artifacts C**
  - **Formal (mathematical) Specification ϕ**

- **Find f $\in$ C that satisfies ϕ**

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

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: I**

$$x = 1 \land y = 1$$

  - **Transition Relation: δ**

$$x' = x+y \ \land \ y' = y+x$$

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

$$y \geq 1 \land \ x' = x+y \ \land \ y' = y+x \ \Rightarrow \ y' \geq 1$$

- **Fails. Need to Strengthen Invariant: Find ϕ s.t.**

$$x = 1 \land y = 1 \Rightarrow \ \phi \land y \geq 1$$

$$\phi \land y \geq 1 \land \ x' = x+y \ \land \ y' = y+x \ \Rightarrow \ \phi' \land y' \geq 1$$

Example Verification Problem

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

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

  - **Transition Relation: δ**

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

- **Property: Ψ = 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 ϕ 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 Ψ?

$\Downarrow$

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

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 ϕ, does M satisfy ϕ?**

- **SYNTHESIS PROBLEM: Given (closed) system M and class of specifications C, find "tightest" specification ϕ in C such that M satisfies ϕ.**
  - 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

Recent Efforts in Program Synthesis

Common theme to many recent efforts:

- **Sketch (Solar-Lezama et al)**
- **Implicit Programming (Kuncak et al)**
- **Oracle-guided program synthesis (Jha et al)**
- **FlashFill (Gulwani et al)**
- **Super-optimization (Schkufza et al)**
- **Invariant generation (Many recent efforts…)**
- **TRANSIT for protocol synthesis (Udupa et al)**
- **Auto-grader (Singh et al)**
- **…**

[Slide content from A. Solar-Lezama]

Further Reading for this Tutorial

- R. Alur et al., "**Syntax-Guided Synthesis**", FMCAD 2013.

  http://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-alur-fmcad13.html

- S. A. Seshia, **"Sciduction: Combining Induction, Deduction, and Structure for Verification and Synthesis**.", DAC 2012

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

- **Lecture notes of EECS 219C: "Computer-Aided Verification" class at UC Berkeley, available at:**

  http://www.eecs.berkeley.edu/~sseshia/219c/

Two Central Questions

- **Is there a core computational problem for formal synthesis?**
  - **Shared by many different synthesis problems**

SYNTAX-GUIDED SYNTHESIS

- **Is there a common theory of formal synthesis techniques?**

ORACLE-GUIDED INDUCTIVE SYNTHESIS
(Counterexample-Guided Inductive Synthesis – CEGIS)

Syntax-Guided Synthesis

Formal Synthesis (recap)

- **Given:**
  - **Formal Specification ϕ**
  - **Class of Artifacts C**

- **Find f $\in$ C that satisfies ϕ**

Syntax-Guided Synthesis (SyGuS)

- **Given:**
  - **An SMT formula ϕ in UF + T (where T is some combination of theories)**
  - **Typed uninterpreted function symbols $f_1,..,f_k$ in ϕ**
  - **Grammars G, one for each function symbol f_i**
- **Generate expressions $e_1,..,e_k$ from G s.t.**

$$\phi\,[f_1,..,f_k \leftarrow e_1,..,e_k] \text{ is valid in T}$$

SyGuS $\neq$ $\exists\forall$ SMT

- **Exists-Forall SMT**

$$\exists f \ \forall x \ \phi(f,x)$$

- **SyGuS (abusing notation slightly)**

$$\exists f \in G \ \forall x \ \phi(f,x)$$

- **Sometimes SyGuS is solved by reduction to EF-SMT**

SyGuS Example 1

- Theory **QF-LIA**
  - Types: Integers and Booleans
  - Logical connectives, Conditionals, and Linear arithmetic
  - Quantifier-free formulas

- Function to be synthesized $f(int\ x, int\ y) : int$

- Specification:
  $$x \le f(x,y) \land y \le f(x,y) \land (f(x,y) = x \lor f(x,y) = y)$$

- Grammar
  $$\text{LinExp} := x \mid y \mid \text{Const} \mid \text{LinExp} + \text{LinExp} \mid \text{LinExp} - \text{LinExp}$$

Is there a solution?

SyGuS Example 2

- Theory **QF-LIA**
  - Types: Integers and Booleans
  - Logical connectives, Conditionals, and Linear arithmetic
  - Quantifier-free formulas

- Function to be synthesized $f(int\ x, int\ y): int$

- Specification:
  $x \leq f(x,y)\ \wedge\ y \leq f(x,y)\ \wedge\ (\ f(x,y) = x\ \vee\ f(x,y) = y\ )$

- Grammar

  Term := x | y | Const | If-Then-Else (Cond, Term, Term)

  Cond := Term <= Term | Cond & Cond | ~Cond | (Cond)

Is there a solution?

From SMT-LIB to SYNTH-LIB

```
(set-logic LIA)
(synth-fun max2 ((x Int) (y Int)) Int
    ((Start Int (x y 0 1 (+ Start Start)(- Start Start)
                        (ite StartBool Start Start)))
     (StartBool Bool ((and StartBool StartBool)
                      (or StartBool StartBool)
                      (not StartBool)
                      (<= Start Start)))))
 (declare-var x Int)
 (declare-var y Int)
 (constraint (>= (max2 x y) x))
 (constraint (>= (max2 x y) y))
 (constraint (or (= x (max2 x y)) (= y (max2 x y))))
 (check-synth)
```

Basic demo

Invariant Synthesis via SyGuS

- **Find ϕ s.t.**

$$x = 1 \land y = 1 \Rightarrow \phi \land y \geq 1$$

$$\phi \land y \geq 1 \land x' = x+y \land y' = y+x \Rightarrow \phi' \land y' \geq 1$$

- **Syntax-Guidance: Grammar expressing simple linear predicates of the form $S \geq 0$ where S is an expression defined as:**

$$S ::= 0 \mid 1 \mid x \mid y \mid S + S \mid S - S$$

- **Demo**

More Demos (time permitting)

- Impact of Grammar definition
  - Expression size
  - Symmetries

- Visit http://www.sygus.org for publications, benchmarks and sample solvers

Other Considerations

- Let-Expressions (for common sub-expressions)
  - Example:

    S ::= let [t := T] in t * t

    T ::= x | y | 0 | 1 | T + T | T - T

- Cost constraints/functions (for “optimality” of synthesized function)

Inductive Synthesis

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**

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 Ψ ("performance measure")**

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

Formal Inductive Synthesis

- **Given:**
  - **Class of Artifacts C**
  - **Formal specification ϕ**
  - **Set of (labeled) examples E (or source of E)**

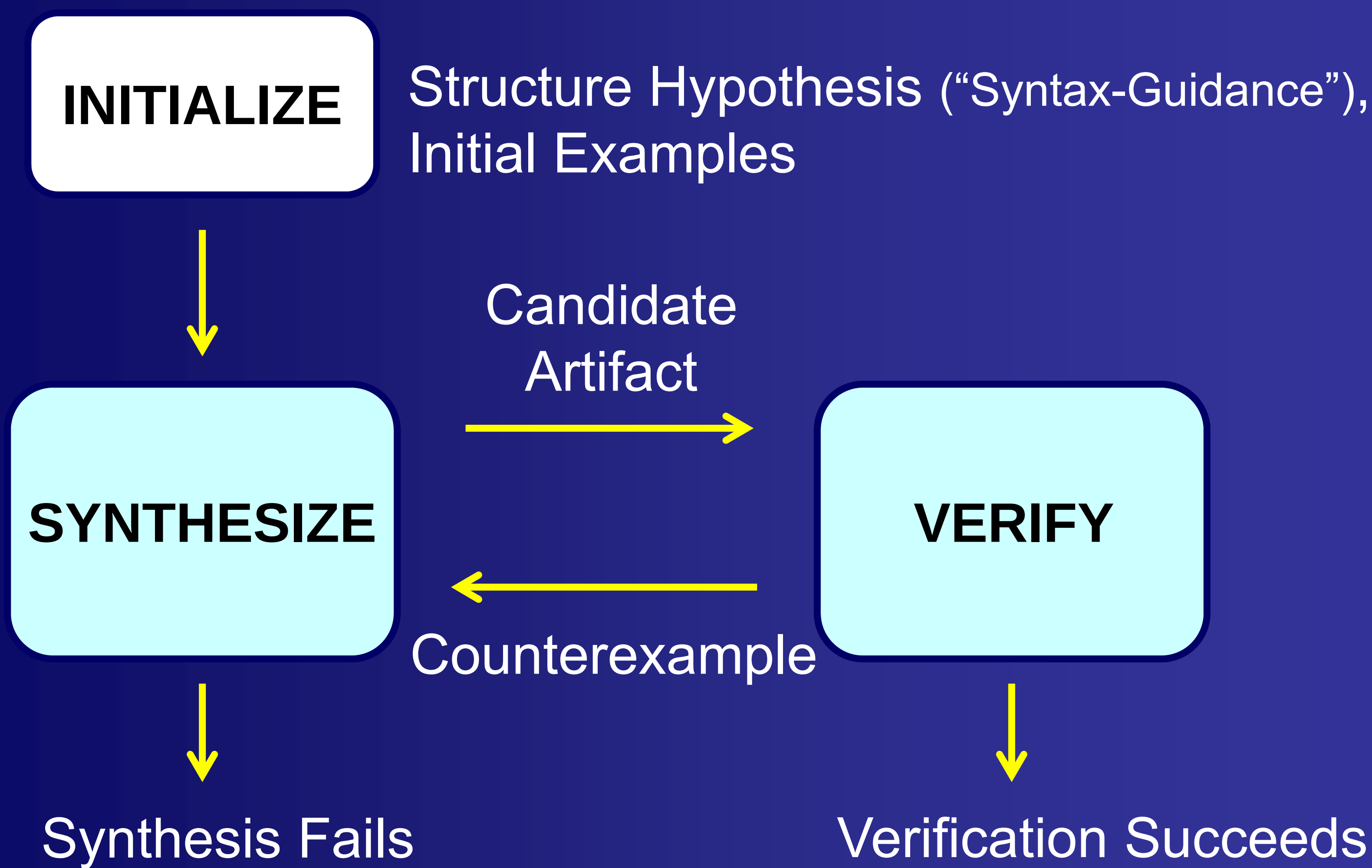
- **Find using only E an $f \in C$ that satisfies ϕ**

- **Example:**
  - **C: all affine functions f of $x \in R$**
  - **E = {(0,42), (1, 43), (2, 44)}**
  - **ϕ: $\forall x.\ f(x) \geq x + 42$**

*For brevity we will often use "Inductive Synthesis" to mean "Formal Inductive Synthesis"

Counterexample-Guided Inductive Synthesis (CEGIS)

[Solar-Lezama, Tancau, Bodik, Seshia, Saraswat, ASPLOS'06]

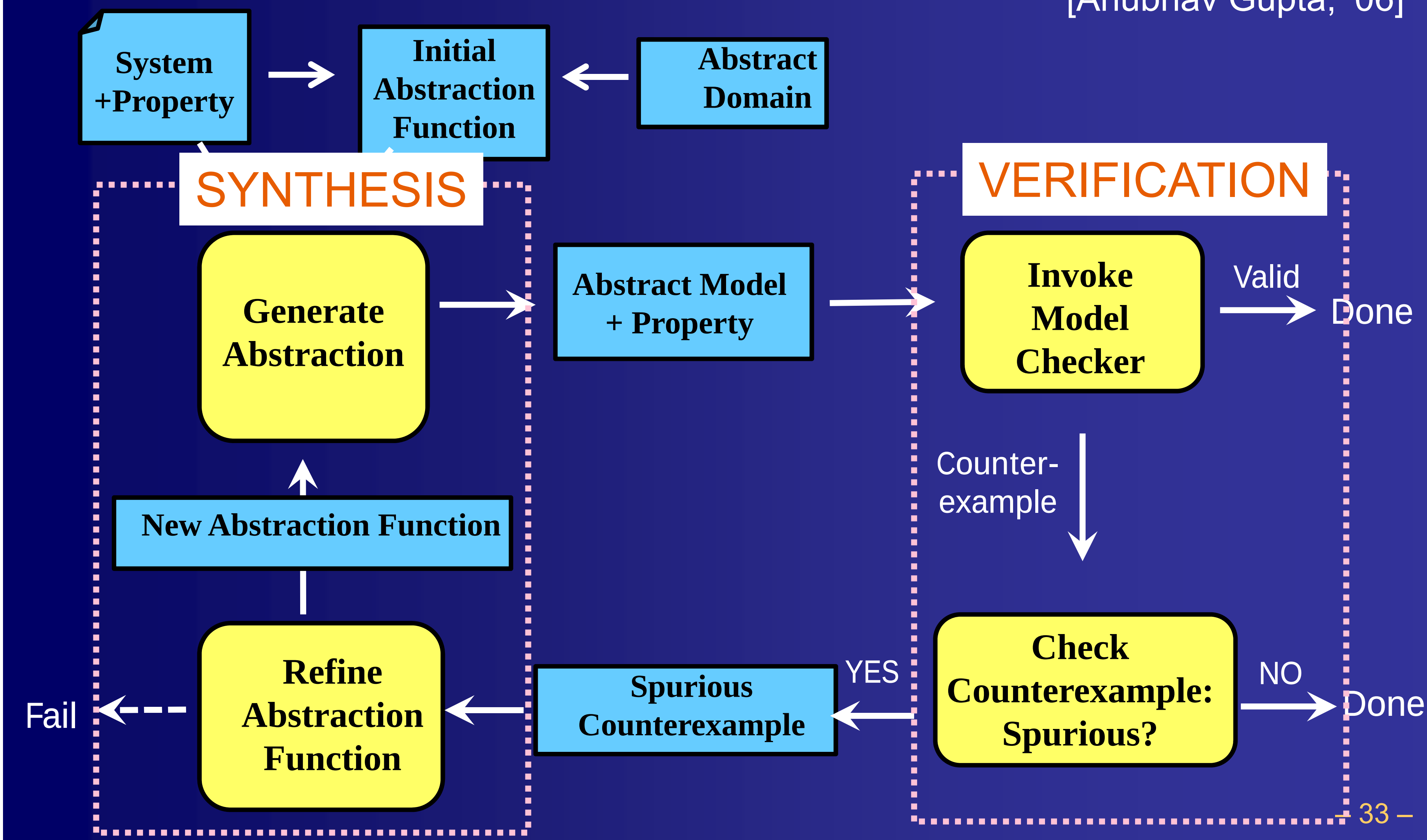


CEGIS vs. SyGuS

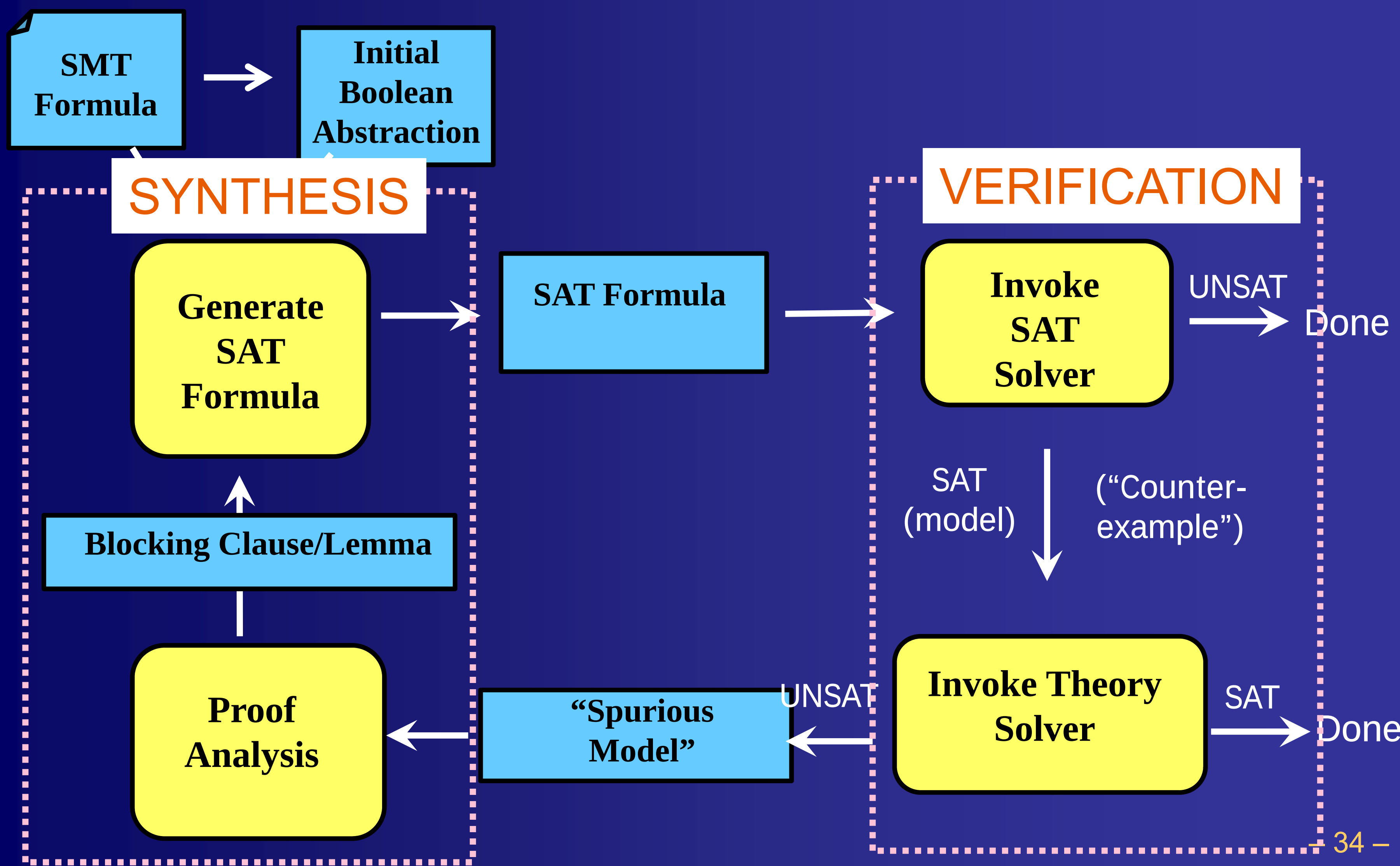
- **SyGuS is a family of PROBLEMS**
- **CEGIS is a family of SOLUTIONS**
- **All SyGuS solvers (available today) use some form of CEGIS**

Counterexample-Guided Abstraction Refinement is CEGIS (for abstractions)

[Anubhav Gupta, '06]



Lazy SMT Solving performs CEGIS (of Lemmas)



Example: CEGIS for SyGuS

- **Specification:**

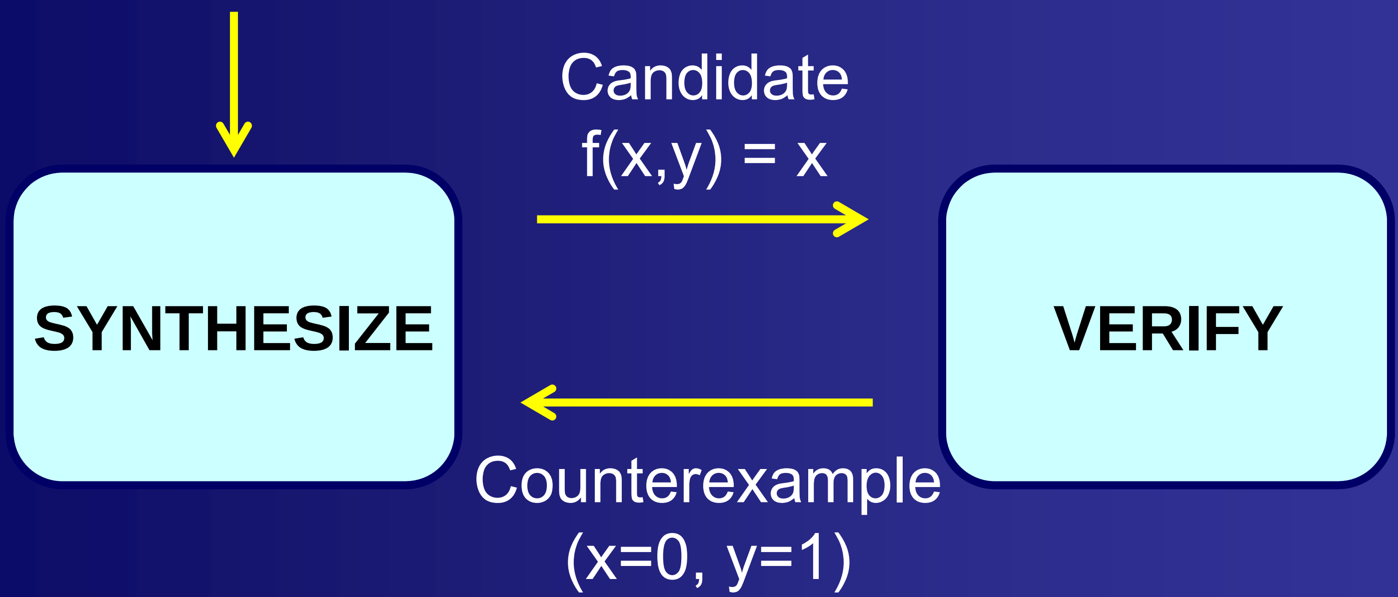
$$x \le f(x,y) \land y \le f(x,y) \land (f(x,y) = x \lor f(x,y) = y)$$

- **Grammar**

Term := x | y | 0 | 1 | If-Then-Else (Cond, Term, Term)

Cond := Term <= Term | Cond & Cond | ~Cond | (Cond)

Examples: { }



Example: CEGIS for SyGuS

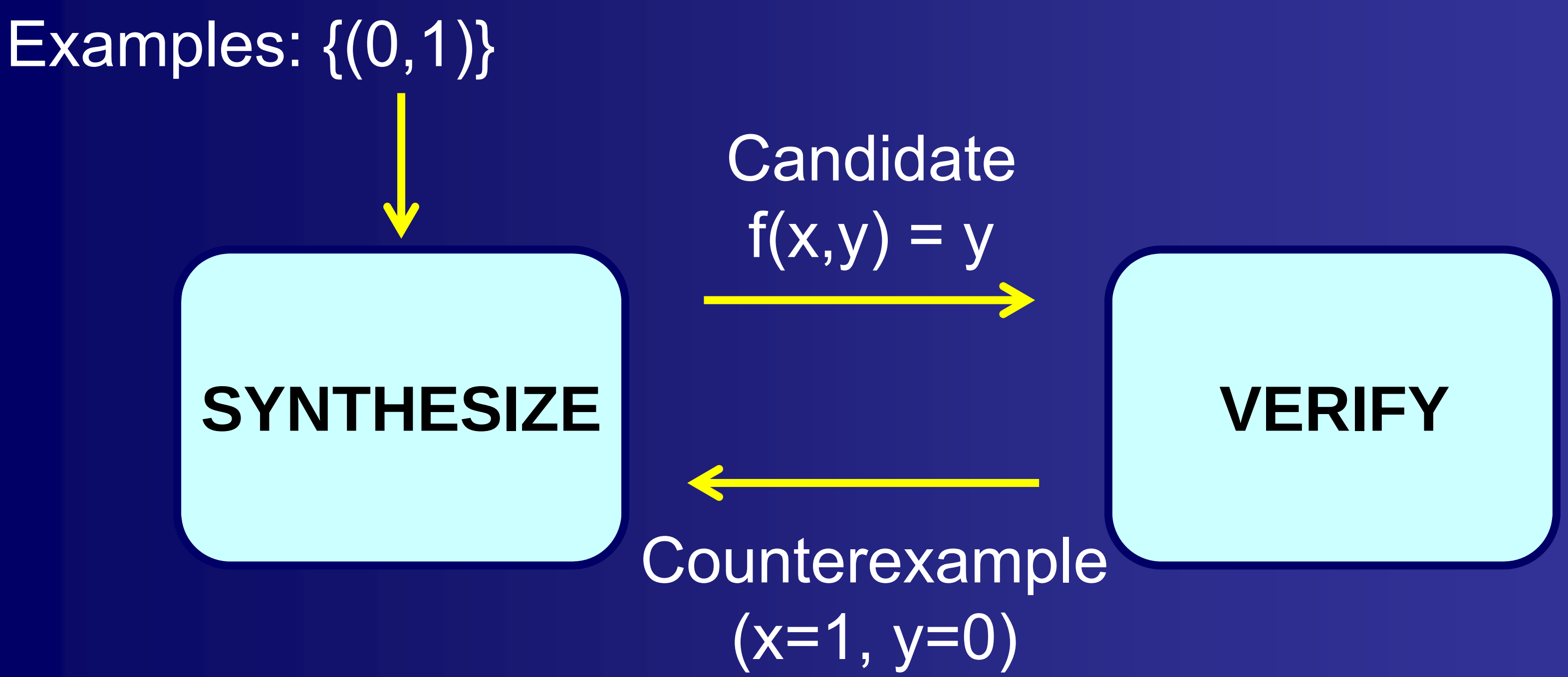
- **Specification:**

  $$x \leq f(x,y) \wedge \ y \leq f(x,y) \wedge (\ f(x,y) = x \vee \ f(x,y) = y \ )$$

- **Grammar**

  Term := x | y | 0 | 1 | If-Then-Else (Cond, Term, Term)

  Cond := Term <= Term | Cond & Cond | ~Cond | (Cond)



Example: CEGIS for SyGuS

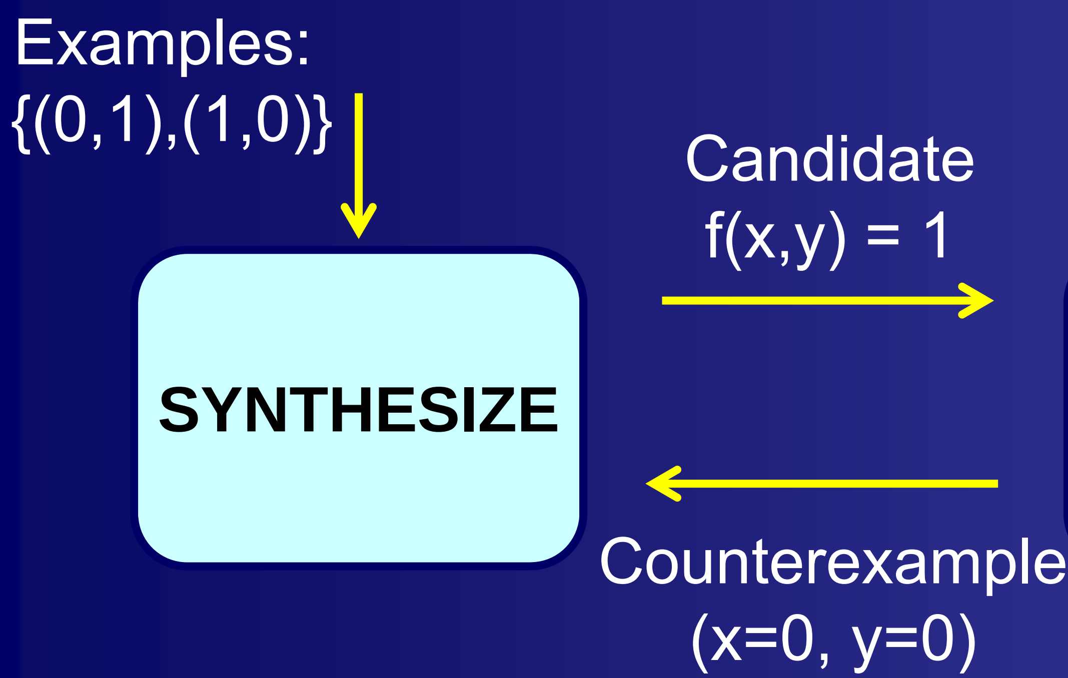
- **Specification:**

  $$x \le f(x,y) \ \wedge \ y \le f(x,y) \ \wedge \ (\, f(x,y) = x \ \vee \ f(x,y) = y \,)$$

- **Grammar**

  Term := x | y | 0 | 1 | If-Then-Else (Cond, Term, Term)

  Cond := Term <= Term | Cond & Cond | ~Cond | (Cond)



Example: CEGIS for SyGuS

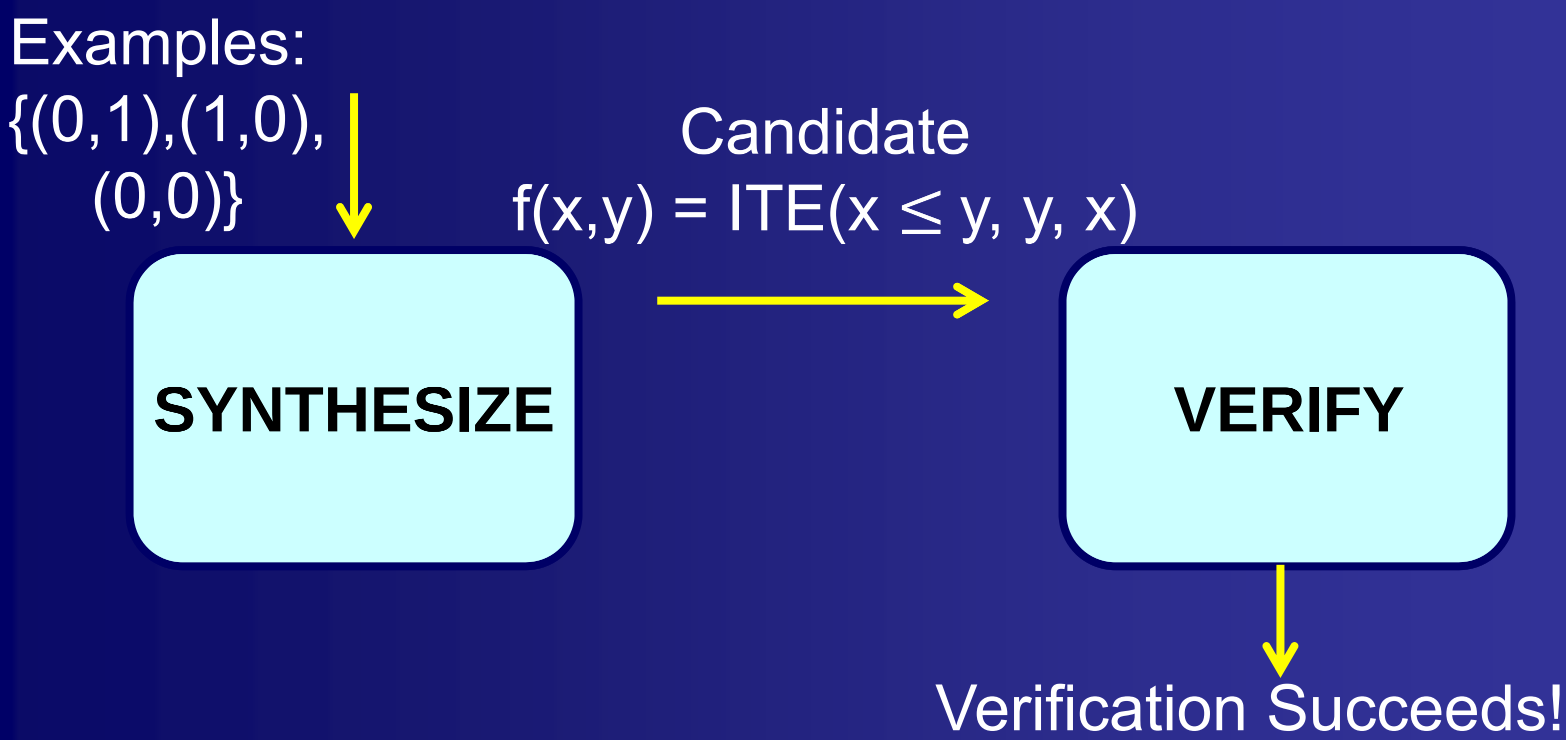
- **Specification:**

  $$x \le f(x,y) \wedge y \le f(x,y) \wedge (f(x,y) = x \vee f(x,y) = y)$$

- **Grammar**

  Term := x | y | 0 | 1 | If-Then-Else (Cond, Term, Term)

  Cond := Term <= Term | Cond & Cond | ~Cond | (Cond)

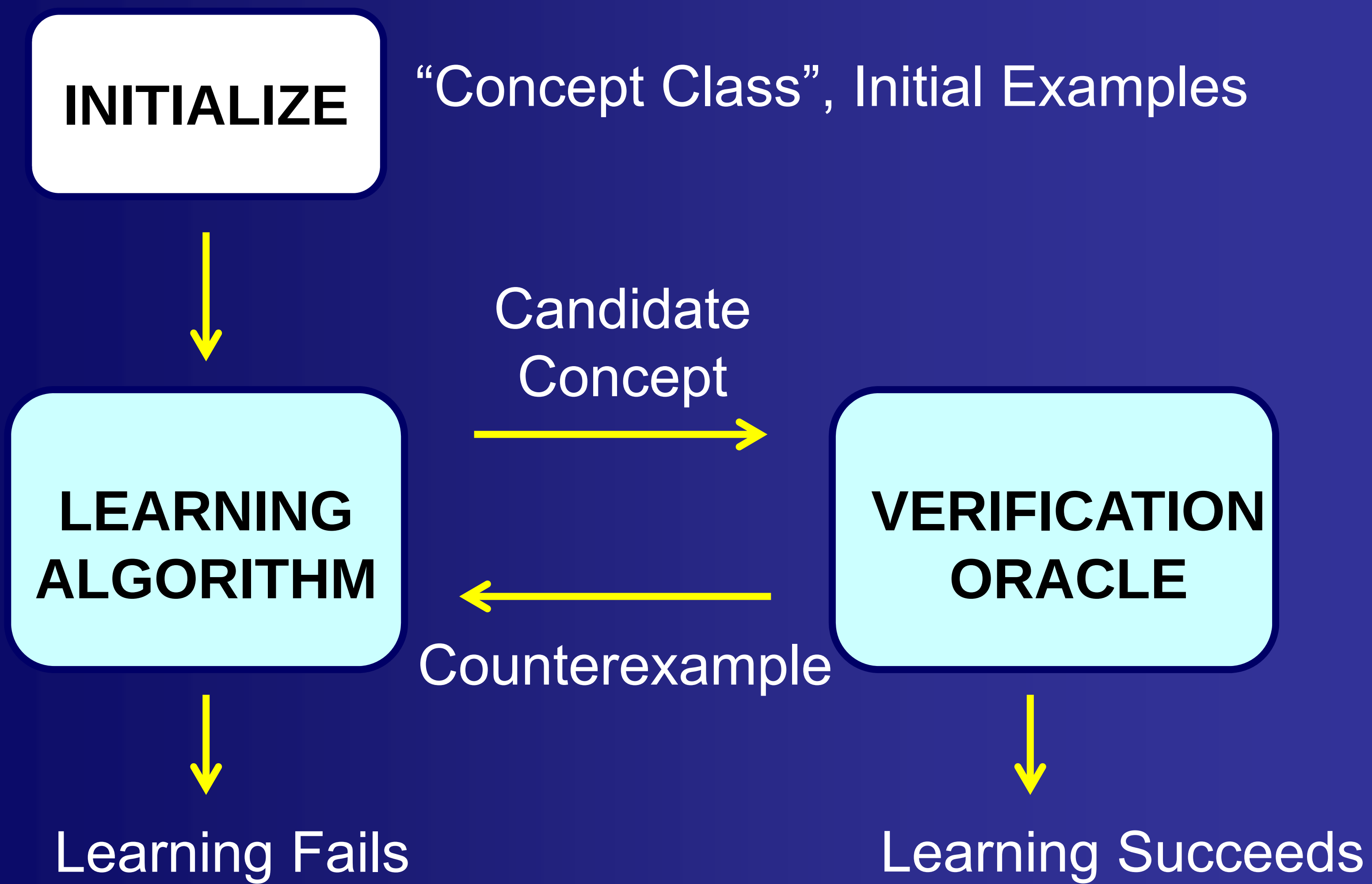


Three Flavors of SyGuS Solvers

- All use CEGIS, differ in implementation of "Synthesis" step
- **Enumerative** [Udupa et al., PLDI 2013]
  - Enumerate expressions in increasing order of "syntactic simplicity" with heuristic optimizations
- **Symbolic** [Jha et al., ICSE 2010, PLDI 2011]
  - Encode search for expressions as SMT problem
  - Similar approach used in SKETCH [Solar-Lezama'08]
- **Stochastic** [Schkufza et al., ASPLOS 2013]
  - Markov Chain Monte Carlo search method over space of expressions
- See [Alur et al., FMCAD 2013] paper for more details

Theoretical Aspects of Inductive Synthesis

CEGIS = Learning from Examples & Counterexamples



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 control Oracle)	Rare (black-box oracles)

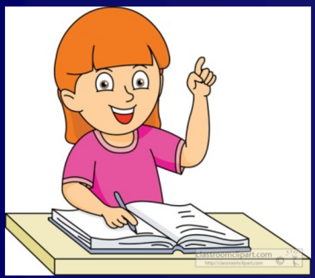
* Between typical inductive synthesizer and machine learning algo

Oracle-Guided Inductive Synthesis

- **Given:**
  - **Domain of Examples D**
  - **Concept Class C**
  - **Formal Specification $\phi \subseteq D$**
  - **Oracle O that can answer queries of type Q**

- **Find, by only querying O, an $f \in C$ that satisfies ϕ**

Common Oracle Query Types



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

Examples of OGIS

- **L* algorithm to learn DFAs: counterexample-guided**
  - **Membership + Equivalence queries**
- **CEGIS used in SKETCH/SyGuS solvers**
  - **(positive) Witness + Equivalence/Subsumption 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

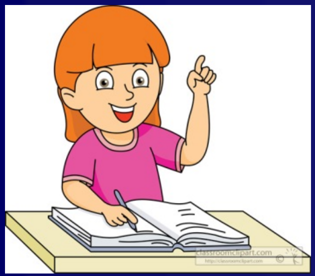
Feature	Formal Inductive Synthesis	Machine Learning
Concept/Program		imple
		alized
Lea		mate, w/ unction
Ora		lack-box les)

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 $\perp$

Equivalence: Is $f = \phi$?

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

Subsumption: Is $f \subseteq \phi$?

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

ORACLE

- 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?

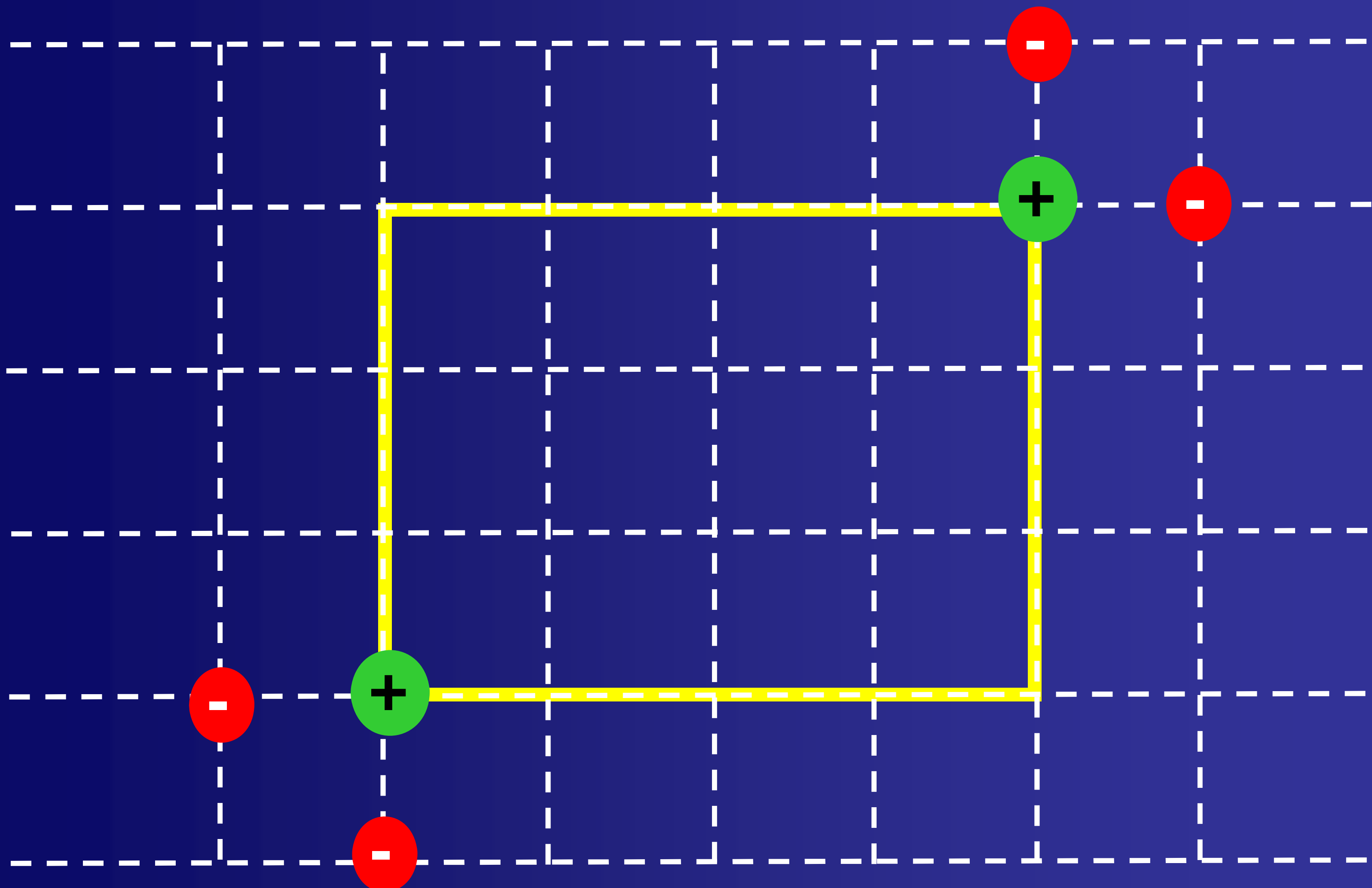
Problem 1: Bounds on Sample Complexity

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

σ is a teaching sequence (uniquely identifies concept c)

Σ is the set of all teaching sequences

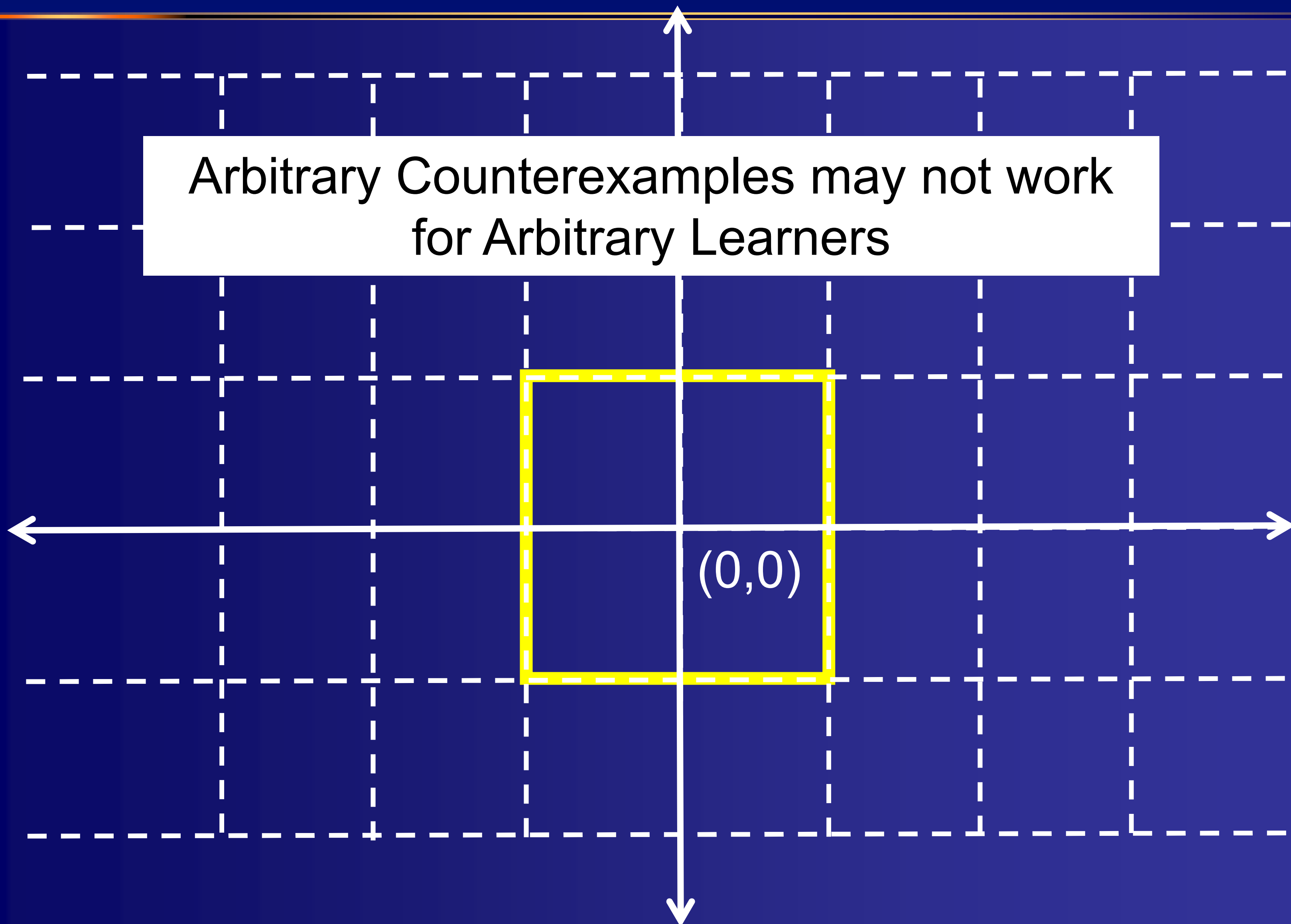
Theorem: *TD*(*C*) is lower bound on Sample Complexity

- CEGIS: 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")
- Open Problems: Compute TD for common classes of SyGuS problems

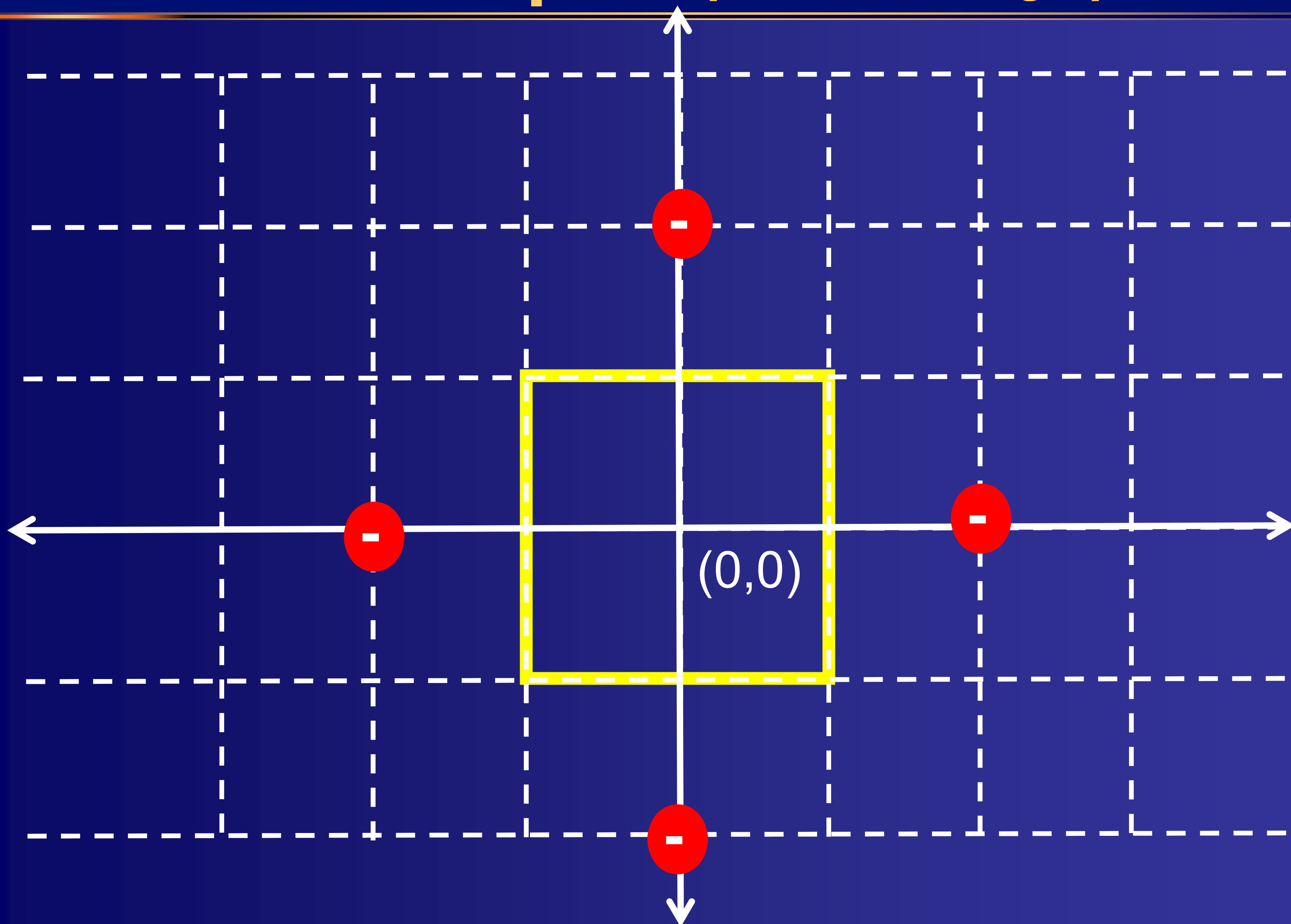
[see Jha & Seshia, 2015]

Problem 2:
Convergence of Counterexample-guided loop with positive witness and membership/subsumption queries

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



Learning $-1 \le x,y \le 1$ 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 ⊕ ϕ
- **MinCEGIS**: Minimal counterexamples
  - A least element of f ⊕ ϕ according to **size**
  - Motivated by debugging methods that seek to find small counterexamples to explain errors & repair

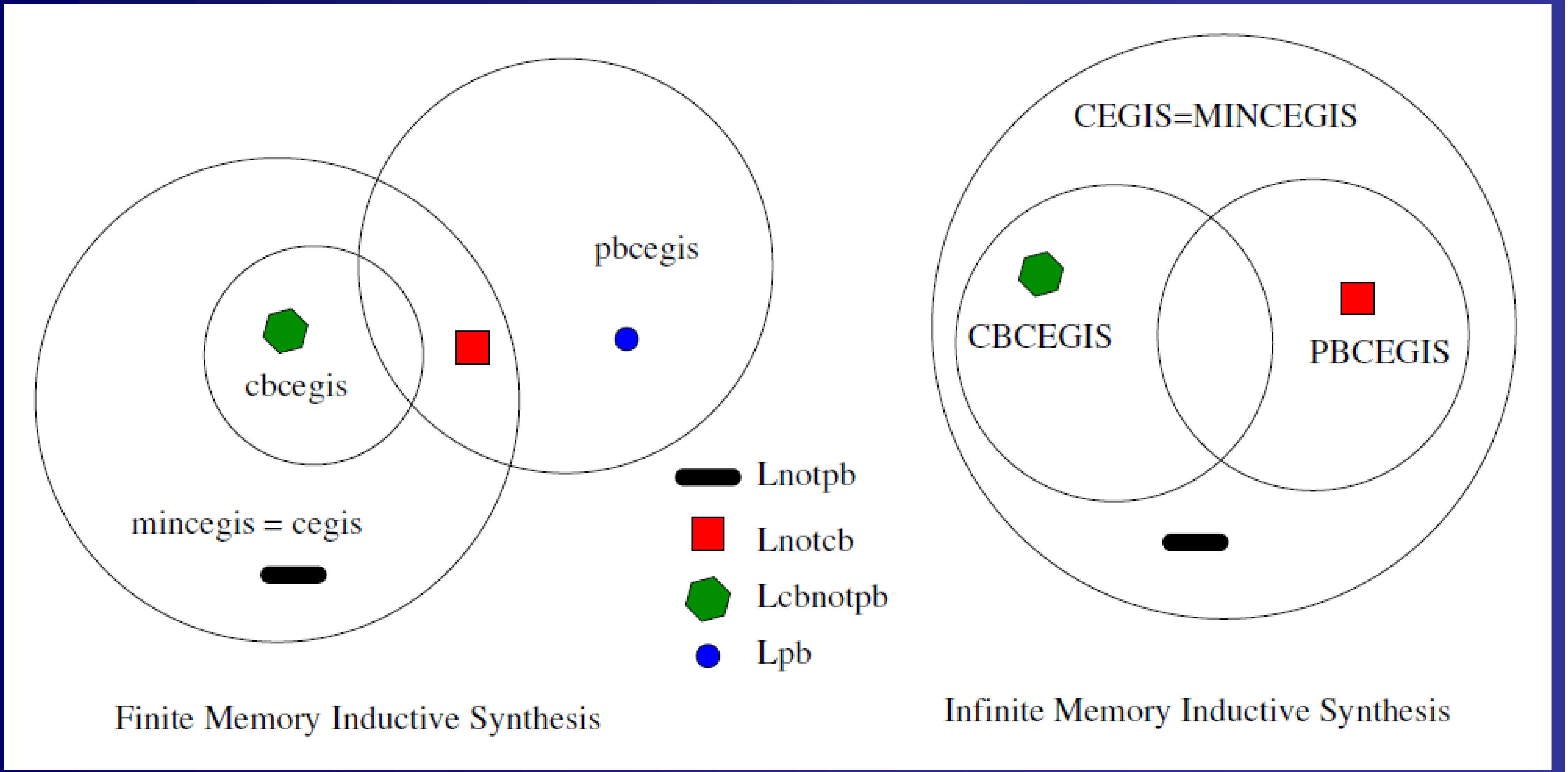
Types of Counterexamples

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



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 and its Applications
- Syntax-Guided Synthesis
  - Problem Definition
  - Demo
- Inductive Synthesis
  - Counterexample-guided inductive synthesis
  - General framework: Oracle-Guided Inductive Synthesis
  - Theoretical analysis
- Lots of potential for future work!