Formal Inductive Synthesis -- Theory and Applications

Sanjit A. Seshia

Professor

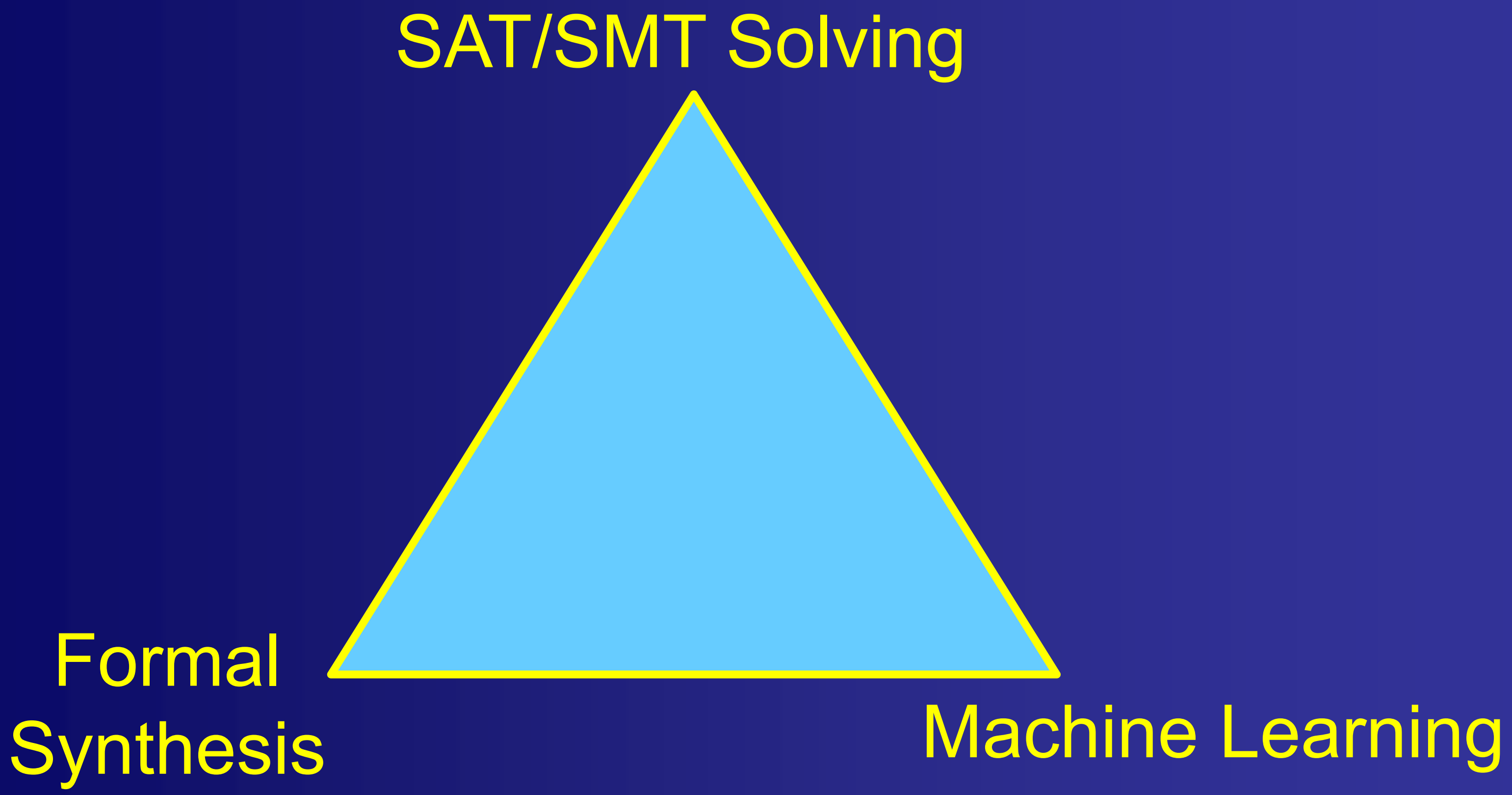
EECS Department

UC Berkeley

Acknowledgments to several Ph.D. students, postdoctoral researchers, and collaborators, and to the students of EECS 219C, Spring 2015/16, UC Berkeley

SAT/SMT Winter School @ TIFR
December 8, 2016

Connections in this Lecture



Examples of Synthesis

Program Synthesis: Example 1

Compute the MAX of two 32-bit integers *without* using conditional statements!

```
int max(int x, int y) {
    if (x >= y)
        return x;
    else
        return y;
}
```

```
int max_no_cond(int x, int y) {
    t1= x ^ y;
    t2 = - (x < y);
    t3 = t1 & t2;
    return t3 ^ x;
}
```

Program Synthesis: Example 2

Turn off rightmost contiguous 1 bits

10110 → 10000, 11010 → 11000

Naïve implementation:

```
i = length(x) – 1;
while( x[i] == 0 ){
  i--; if (i < 0) return x;
 }
x[i] = 0; i--;
while( x[i] == 1 ){
 x[i] = 0; i--; if (i < 0) return x;
 }
return x;
```

Bit-wise implementation:

```
    t1= x - 1;
    t2 = x | t1;
    t3 = t2 + 1;
    return t3 & x;
```

Program Synthesis problem

- Given a reference implementation R, and a restricted program space S, find a program P in S that is equivalent to the reference R.

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

Example Verification Problem

- Transition System
  - Init: I

    $x = 1 \wedge y = 1 \qquad\qquad x, y \in \mathbf{Z}$
  - Transition Relation: δ

    $x' = x+y \;\wedge\; y' = y+x$
- Temporal Logic Property: Ψ = G ($y \geq 1$)
  - "always, $y \geq 1$"
- Attempted Proof by Induction:
  - Base Case: $x = 1 \wedge y = 1 \Rightarrow \; y \geq 1$
  - Inductive Step:

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

Example Verification Problem

- Transition System
  - Init: 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 Fails:

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

➢ Need to Strengthen Invariant: Find ϕ s.t.

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

- **Safety Verification → Invariant Synthesis**

Safety Verification as Invariant Synthesis

- Transition System
  - Init: I

    $x = 1 \wedge y = 1$
  - Transition Relation: δ

    $x' = x+y \;\wedge\; y' = y+x$
- Property: Ψ = G ($y \geq 1$)

➢ Following Strengthened Invariant works: ϕ = $x \geq 1$

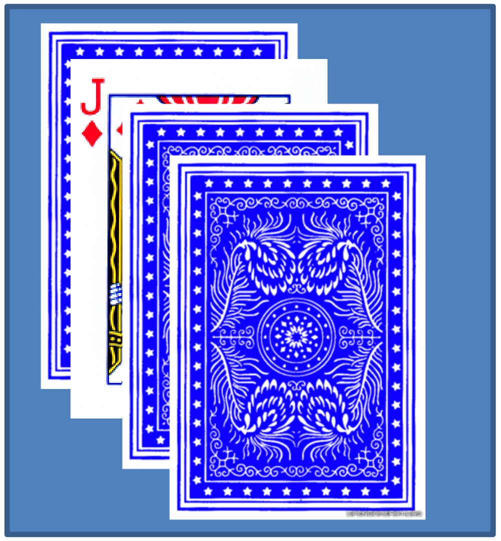
$$x \geq 1 \wedge y \geq 1 \wedge\; x' = x+y \;\wedge\; y' = y+x \;\Rightarrow\; x' \geq 1 \wedge y' \geq 1$$

- How can we automate this process?

Another (Fun) Synthesis Problem: Inventing Card Tricks

[S. Jha, V. Raman, and S. A. Seshia, FMCAD 2016]

Transformations such as
Moving card to front/back,
Flipping card over,
Cutting the deck (with audience choice),
Repeating some number of times (audience chosen)



J – audience card

Configuration where only front-facing card is the one chosen initially by audience

$\exists$ sequence of transformations $\forall$ audience choices
(we reach the desired final configuration)

Similar format for problems in AI planning

Formal Synthesis and Machine Learning

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: $\forall x.\ f(x) \geq x + 42$

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 Ψ
    - May or may not be formally described

- Find, using only E, an $f \in C$ that meets Ψ

- Example:
  - C: all affine functions f of $x \in R$
  - E = {(0,42), (1, 43), (2, 44)}
  - Ψ -- find consistent f

Inductive Synthesis

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

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

Inductive Synthesis

- Example:
  - C: all predicates of the form $ax + by \geq c$
  - $E = \{(0,42), (1, 43), (2, 45)\}$
  - Ψ -- find consistent f
- One such: $-x + y \geq 42$
- Another: $-x + y \geq 0$
- Which one to pick: need to augment Ψ?

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

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 $ax + by \geq c$

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

Ψ -- 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 ϕ
  - Domain of examples D
- Find $f \in C$ that satisfies ϕ using only elements of D
  - i.e. no direct access to ϕ, only to elements of D representing ϕ
- Example:
  - C: all affine functions f of $x \in R$
  - $D = R^2$
  - ϕ: $\forall x.\ f(x) \geq x + 42$

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”

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 Methods ≈ Computational Proof methods
  - Temporal logic / Assertions
  - Boolean reasoning: SAT solving & Binary Decision Diagrams
  - Equivalence checking
  - Model checking
  - Automated theorem proving, SMT solving
  - …

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 for this Tutorial

- How can inductive synthesis be used to solve other (non-synthesis) problems?
- What is a core computational problem for formal synthesis (aka SAT or SMT)?
- 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 for this Tutorial

- How can inductive synthesis be used to solve other (non-synthesis) problems?
  - Reducing a Problem to Synthesis
- What is a core computational problem for formal synthesis (aka SAT or SMT)?
  - Syntax-Guided Synthesis (SyGuS)
- 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 oracles/learners

Outline for this Lecture Sequence

- **Examples of Reduction to Synthesis**
  - **Specification**
  - **Verification**
- **Demo: Requirement Mining for Cyber-Physical Systems**
- **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

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

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

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
- …

Recall: Example Verification Problem

- **Transition System**
  - **Init: 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 #2

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

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

s.t.

$\alpha(M)$ satisfies Ψ

iff

M satisfies Ψ

SYNTHESIS PROBLEM #1

Synthesize ϕ s.t.

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

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

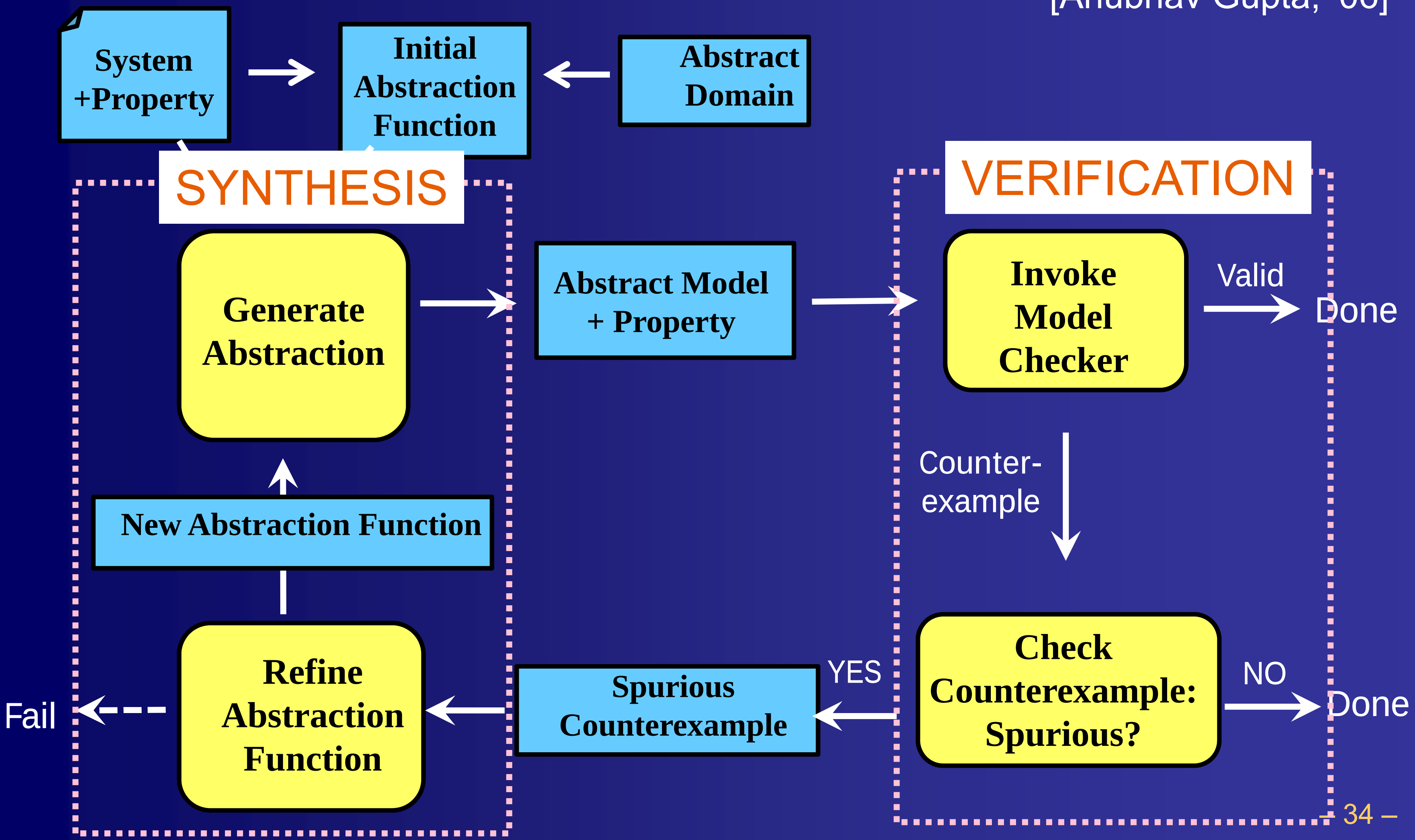
Common Approach for both: Inductive Synthesis

Synthesis of:-

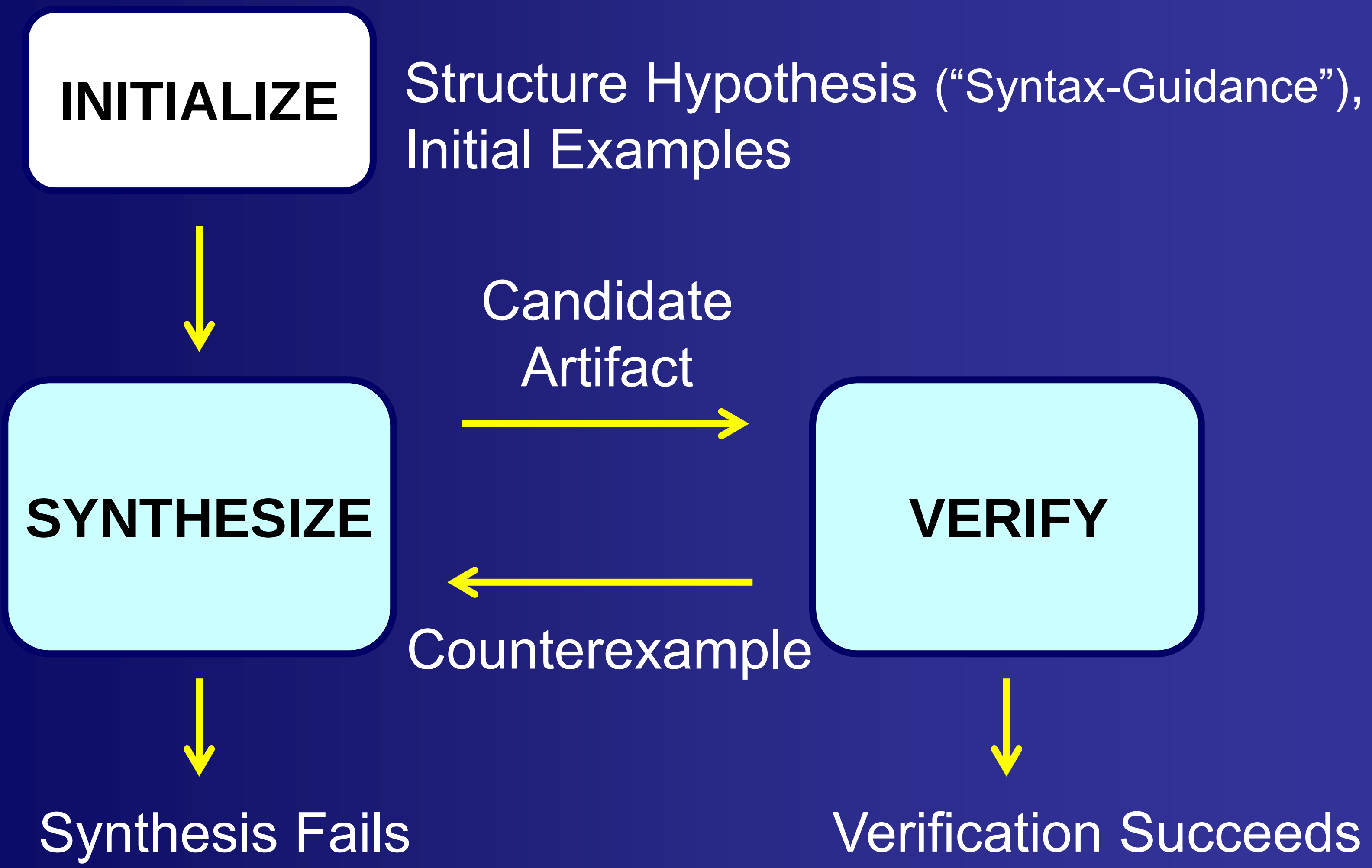
- Inductive Invariants
  - Choose templates for invariants
  - Infer likely invariants from tests (examples)
  - Check if any are true inductive invariants, possibly iterate
- Abstraction Functions
  - Choose an abstract domain
  - Use Counter-Example Guided Abstraction Refinement (CEGAR)

Counterexample-Guided Abstraction Refinement is Inductive Synthesis

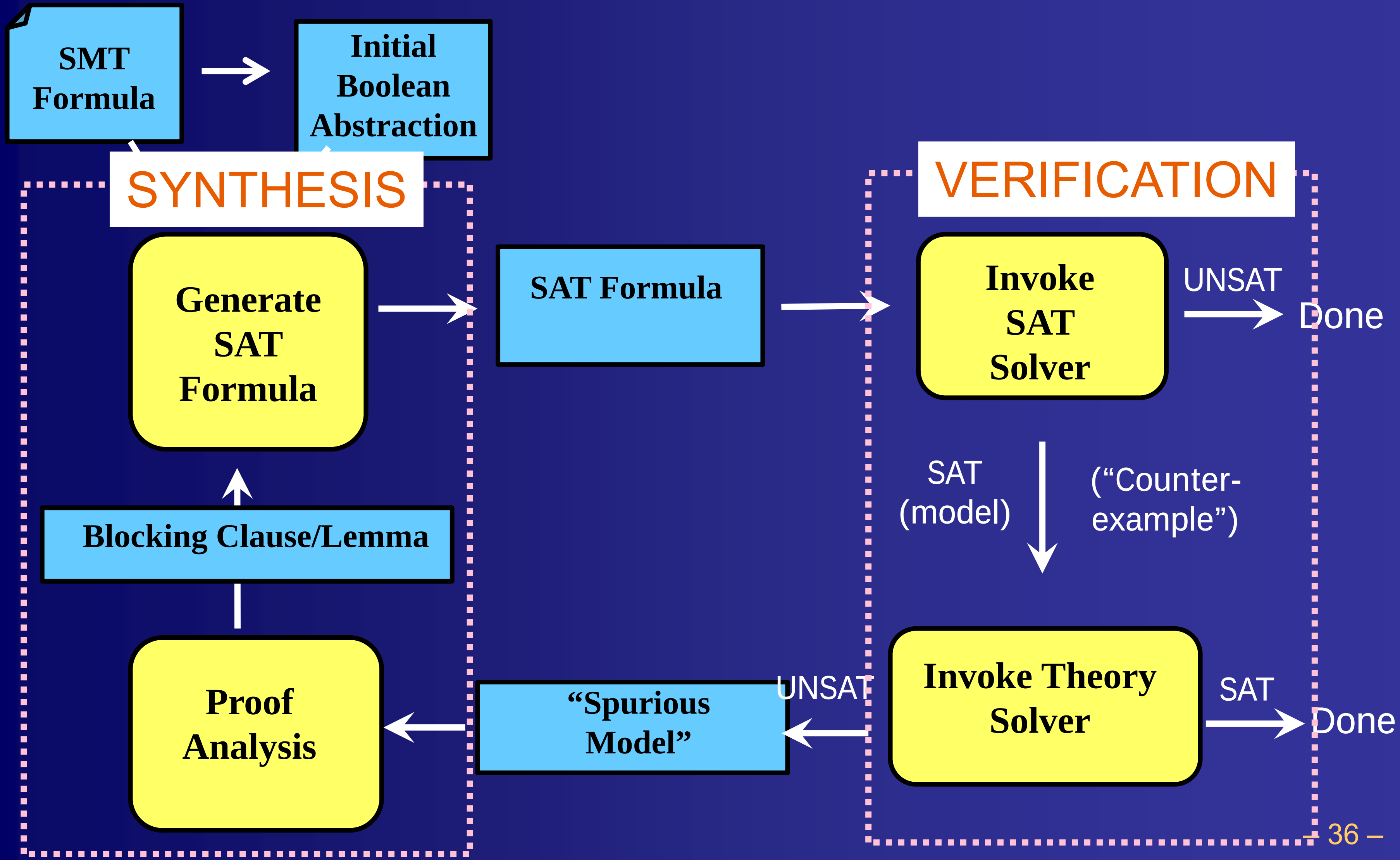
[Anubhav Gupta, '06]



CEGAR = Counterexample-Guided Inductive Synthesis (of Abstractions)



Lazy SMT Solving performs Inductive Synthesis (of Lemmas)



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!*"

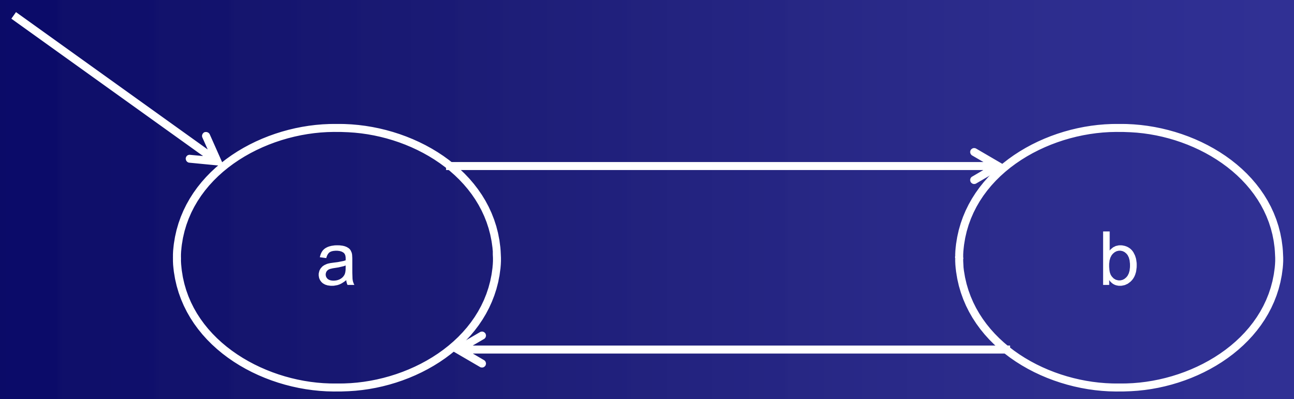
– paraphrased from [Young et al., 1985]

- 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?**
- **Suppose we don't have (a good enough) ϕ.**
- **SYNTHESIS PROBLEM: Given (closed) system M, find specification ϕ such that M satisfies ϕ.**
  - **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 ϕ, does M satisfy ϕ?**

- **SYNTHESIS PROBLEM: Given (closed) system M and *class of specifications C*, find specification ϕ in C such that M satisfies ϕ.**
  - **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 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
  - Based on Counterexample-Guided Inductive Synthesis (CEGIS)
  - Implemented in Breach toolbox by A. Donze
  - An enabler for Toyota to apply formal verification to software in a cyber-physical system [see Yamaguchi et al., FMCAD 2016]

Specification Mining

- Inductive Synthesis of Specifications

- See survey of the topic in 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

  – Environment Assumptions for Reactive Synthesis (with application to planning in robotics)

Summary of Part 1

- Basic Terminology
  - Formal Synthesis
  - Inductive Synthesis
  - Formal Inductive Synthesis (basic intro)
  - Notions from Machine Learning
- Reductions to Synthesis
  - Verification artifacts
  - Specifications
  - Industry tech transfer: requirement mining for closed-loop automotive control systems

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.

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

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

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

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)
```

Invariant Synthesis via SyGuS

- Find ϕ s.t.

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

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

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

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

- **Homework: Try encoding this in SyGuS and solve it using one of the reference solvers available at sygus.org**

Recall Program Synthesis Example 2

Turn off rightmost contiguous 1 bits

10110 → 10000, 11010 → 11000

Naïve implementation:

```
i = length(x) – 1;
while( x[i] == 0 ){
  i--; if (i < 0) return x;
 }
x[i] = 0; i--;
while( x[i] == 1 ){
 x[i] = 0; i--; if (i < 0) return x;
 }
return x;
```

Bit-wise implementation:

```
t1= x - 1;
t2 = x | t1;
t3 = t2 + 1;
return t3 & x;
```

More Demos (time permitting)

- Impact of Grammar definition
  - Small changes to grammar can affect the run time in unpredictable ways!

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

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

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)

SyGuS vs. CEGIS

- SyGuS --- problem classes
- CEGIS --- solution classes

Example: CEGIS for SyGuS

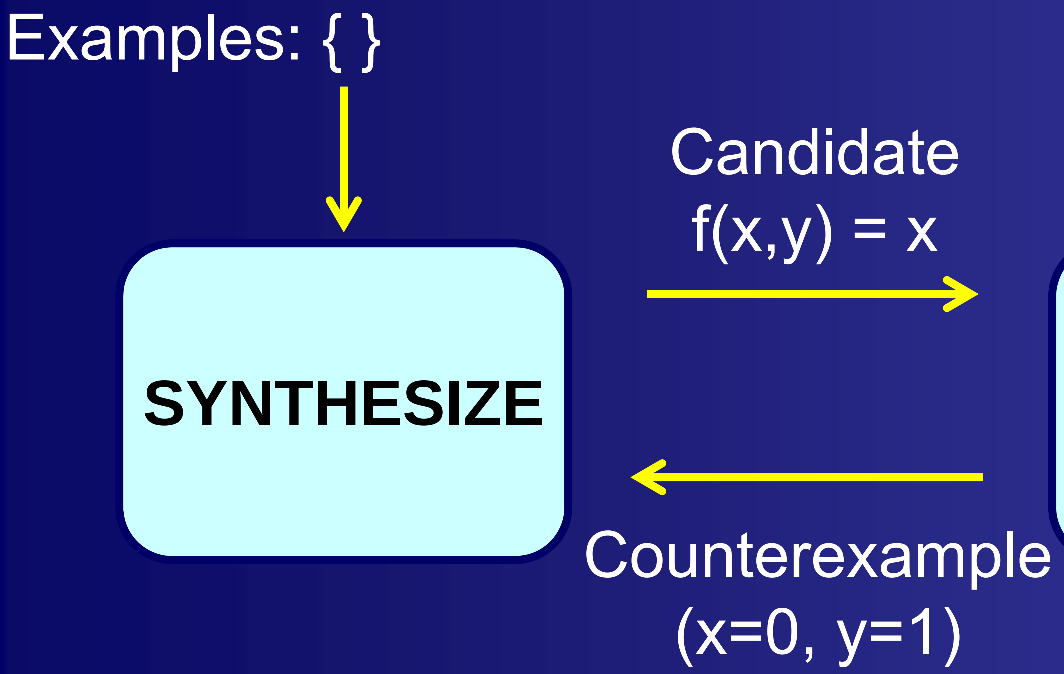
- **Specification:**

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

- **Grammar**

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

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



Example: CEGIS for SyGuS

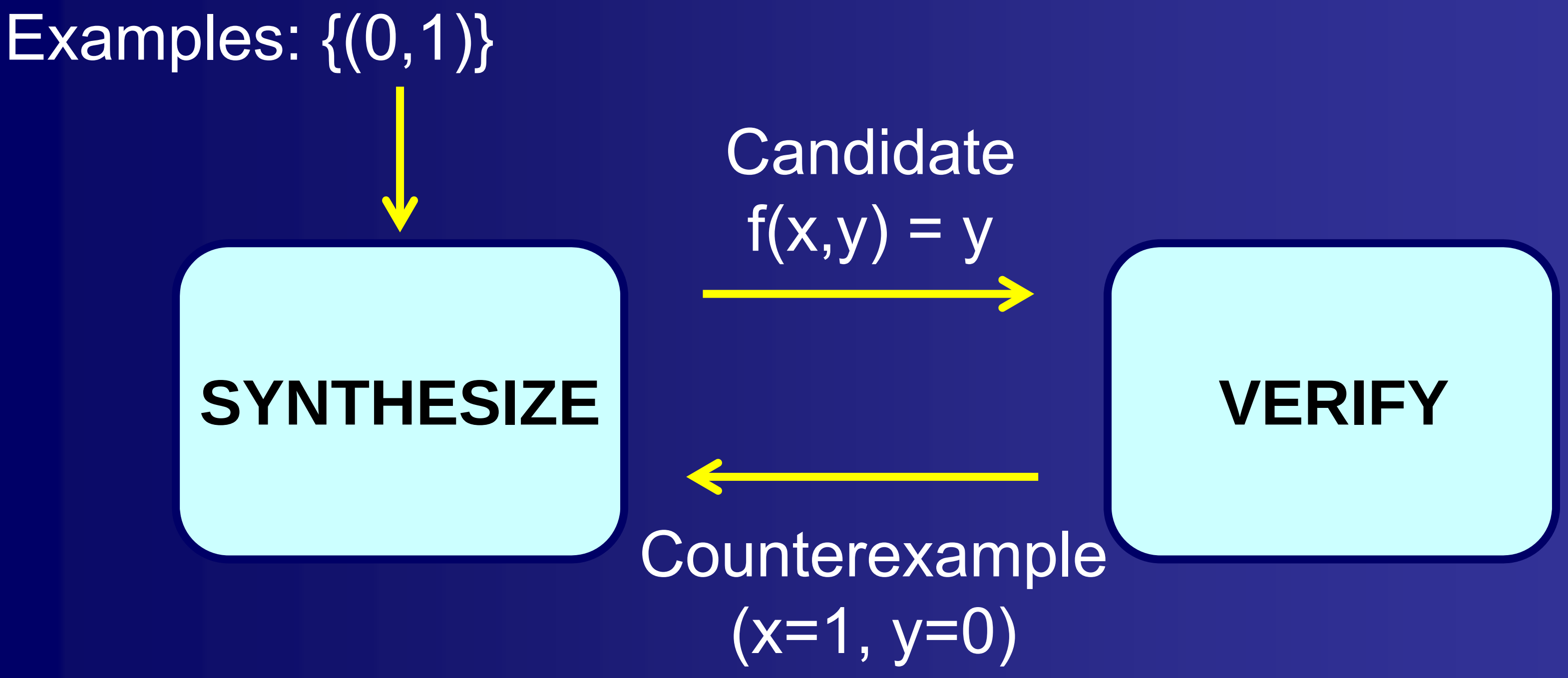
- **Specification:**

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

- **Grammar**

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

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



Example: CEGIS for SyGuS

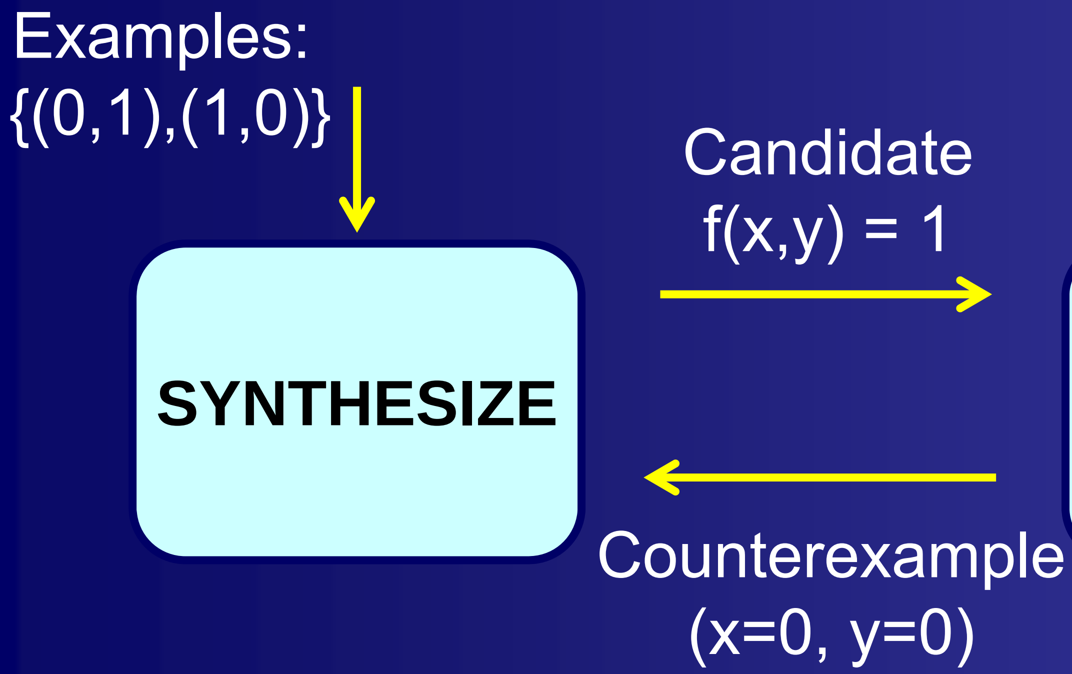
- **Specification:**

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

- **Grammar**

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

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



Example: CEGIS for SyGuS

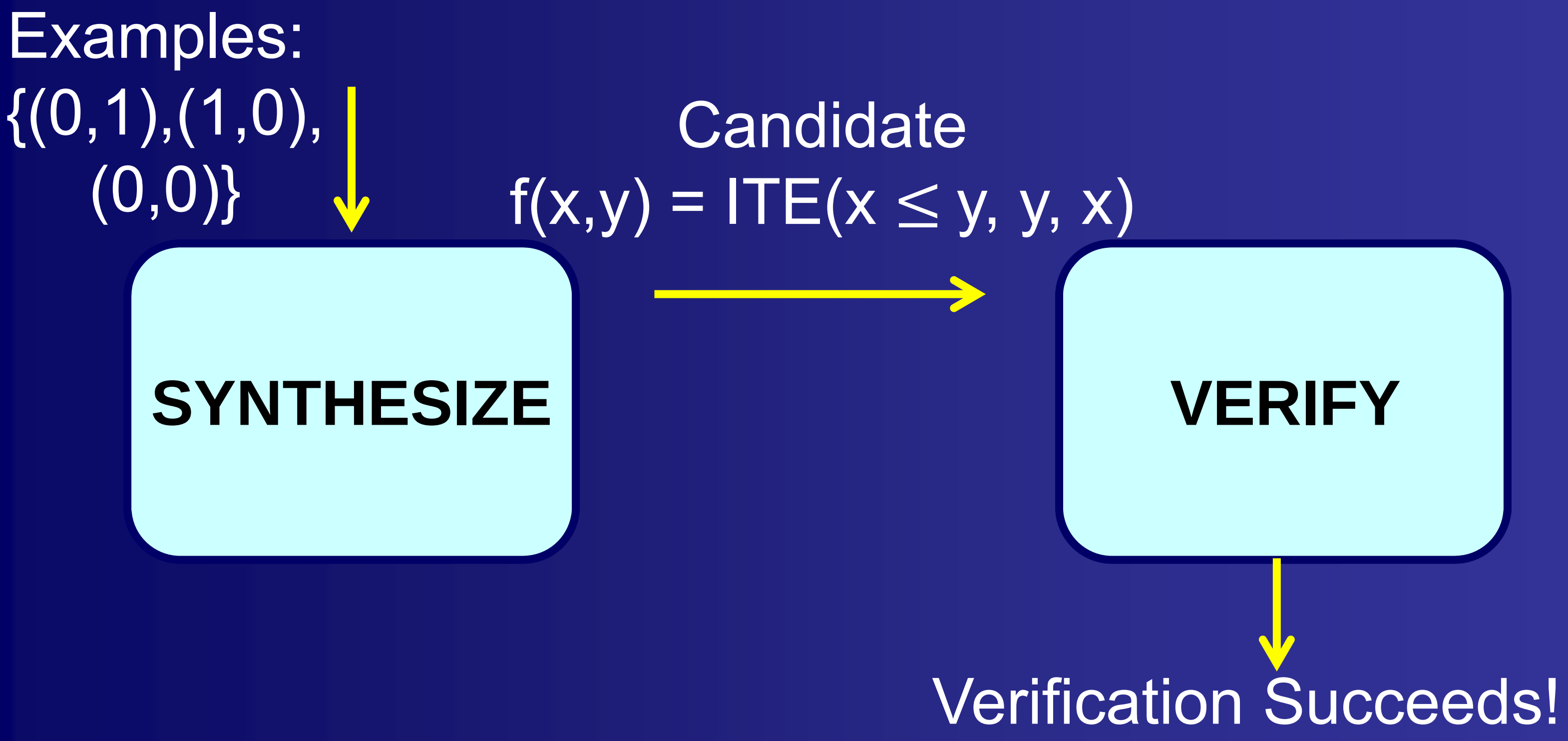
- **Specification:**

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

- **Grammar**

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

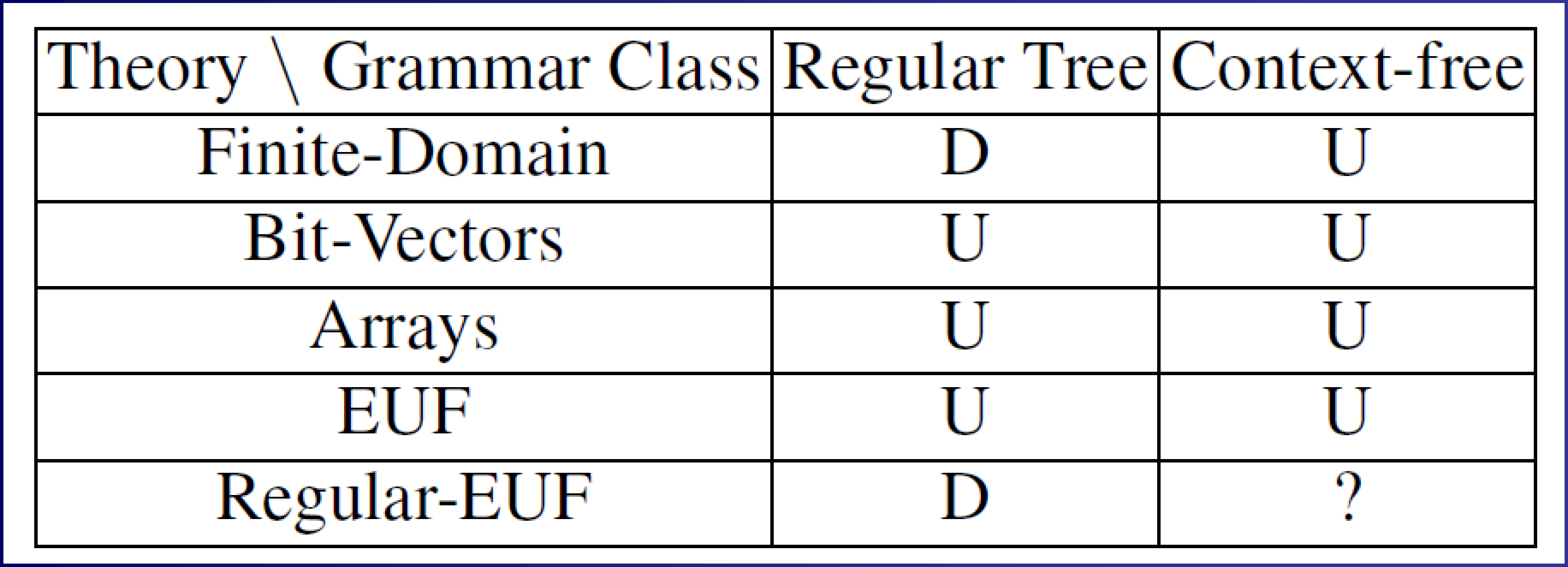
Cond := Term <= Term | Cond & Cond | ~Cond | (Cond)



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**

Decidability of SyGuS Problems



Theory \ Grammar Class	Regular Tree	Context-free
Finite-Domain	D	U
Bit-Vectors	U	U
Arrays	U	U
EUF	U	U
Regular-EUF	D	?

[B. Caulfield, M. Rabe, S. A. Seshia, S. Tripakis,
"What's Decidable about Syntax-Guided Synthesis, 2016]

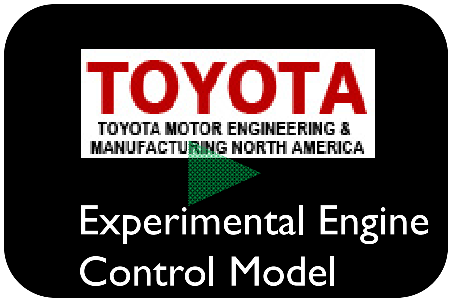
An Industrial Application of Inductive Synthesis of Specifications

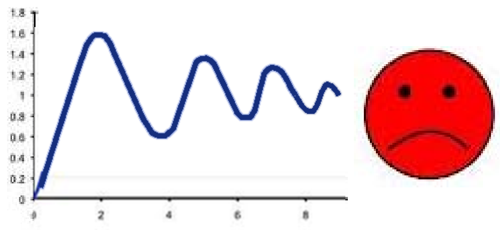
Requirements Mining for Closed-Loop Automotive Control Systems

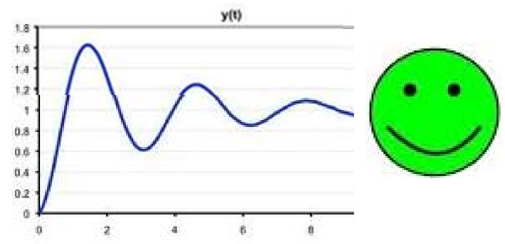
Used internally by Toyota production groups

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"


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





Solution: Requirements Mining

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

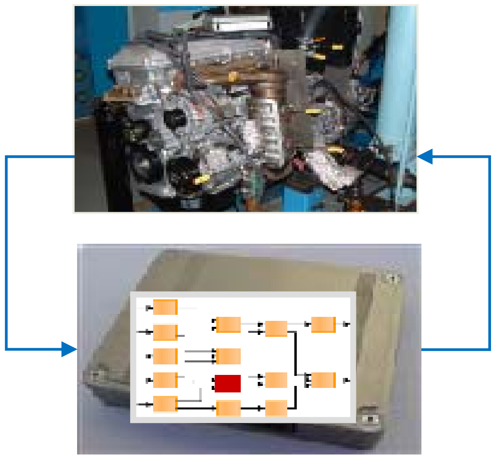
Value added by mining:

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

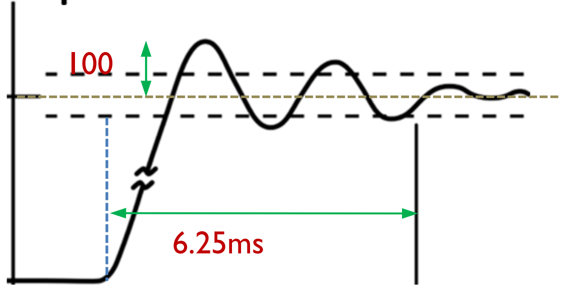
It's working, but I don't understand why!

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]

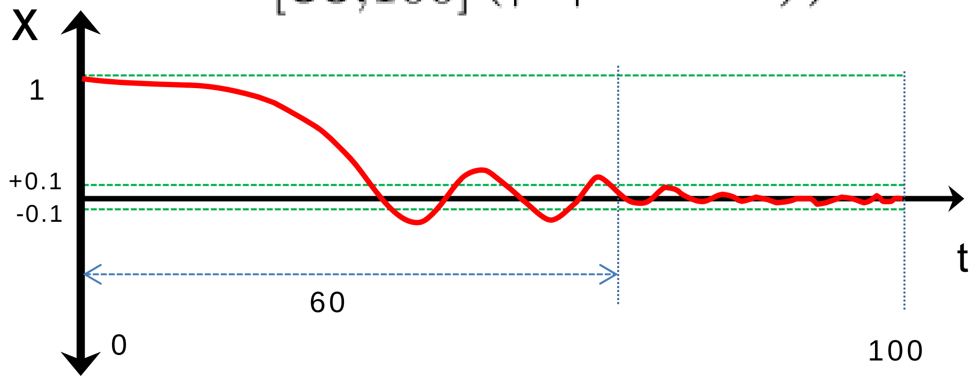


OKAY, LET'S START BY DOCUMENTING YOUR MARKET REQUIREMENTS.
NO, LET'S START BY YOU TELLING ME ALL THE THINGS YOU CAN DESIGN. THEN I'LL TELL YOU WHICH ONE I LIKE.
S. Adams E-mail: SCOTTADAMS@AOL.COM

Signal Temporal Logic (STL)

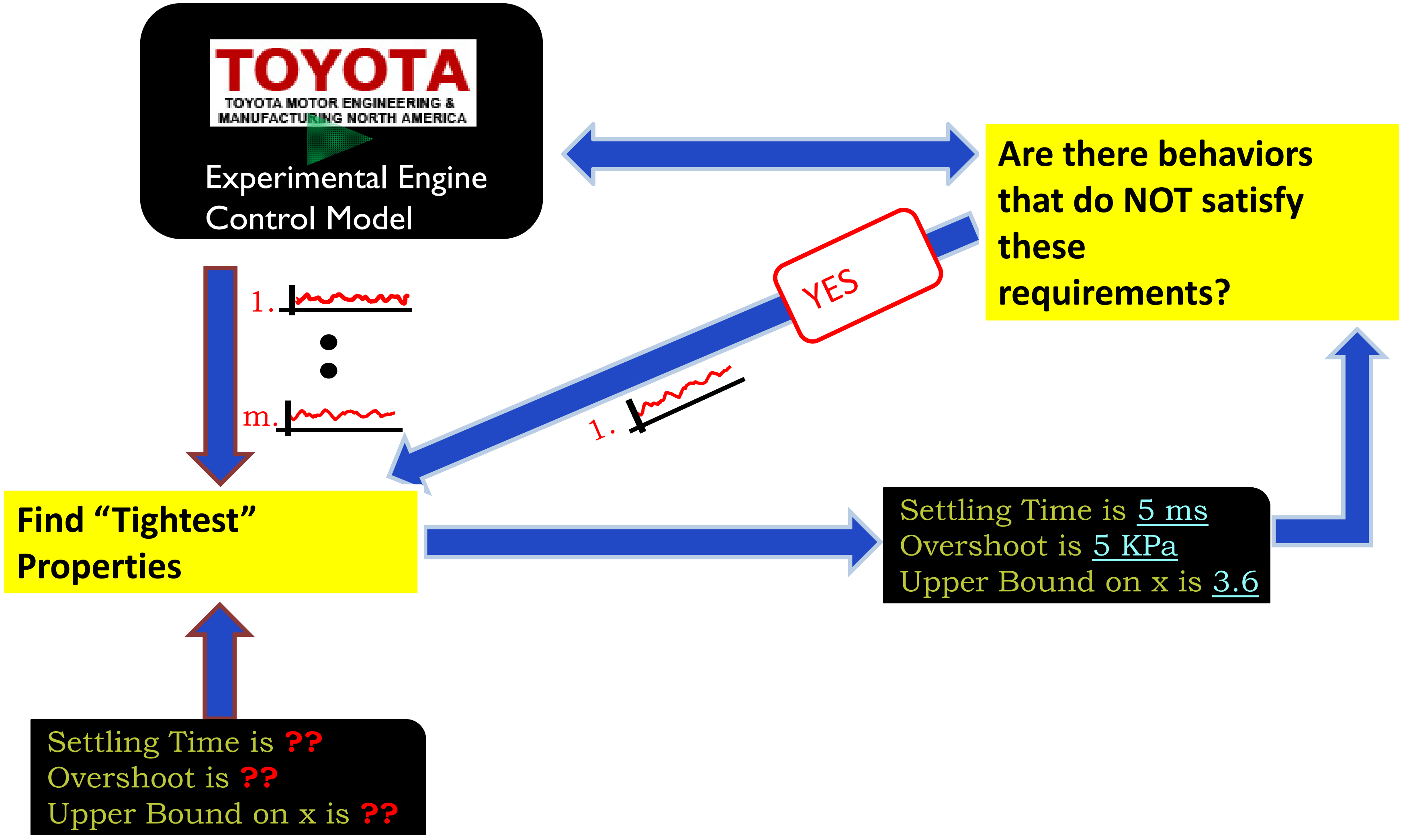
- Extension of Linear Temporal Logic (LTL) and Variant of Metric Temporal Logic (MTL)
  - Quantitative semantics: satisfaction of a property over a trace given real-valued interpretation
  - Greater value → more easily satisfied
  - Non-negative satisfaction value ≡ Boolean satisfaction
- Example: *"For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

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

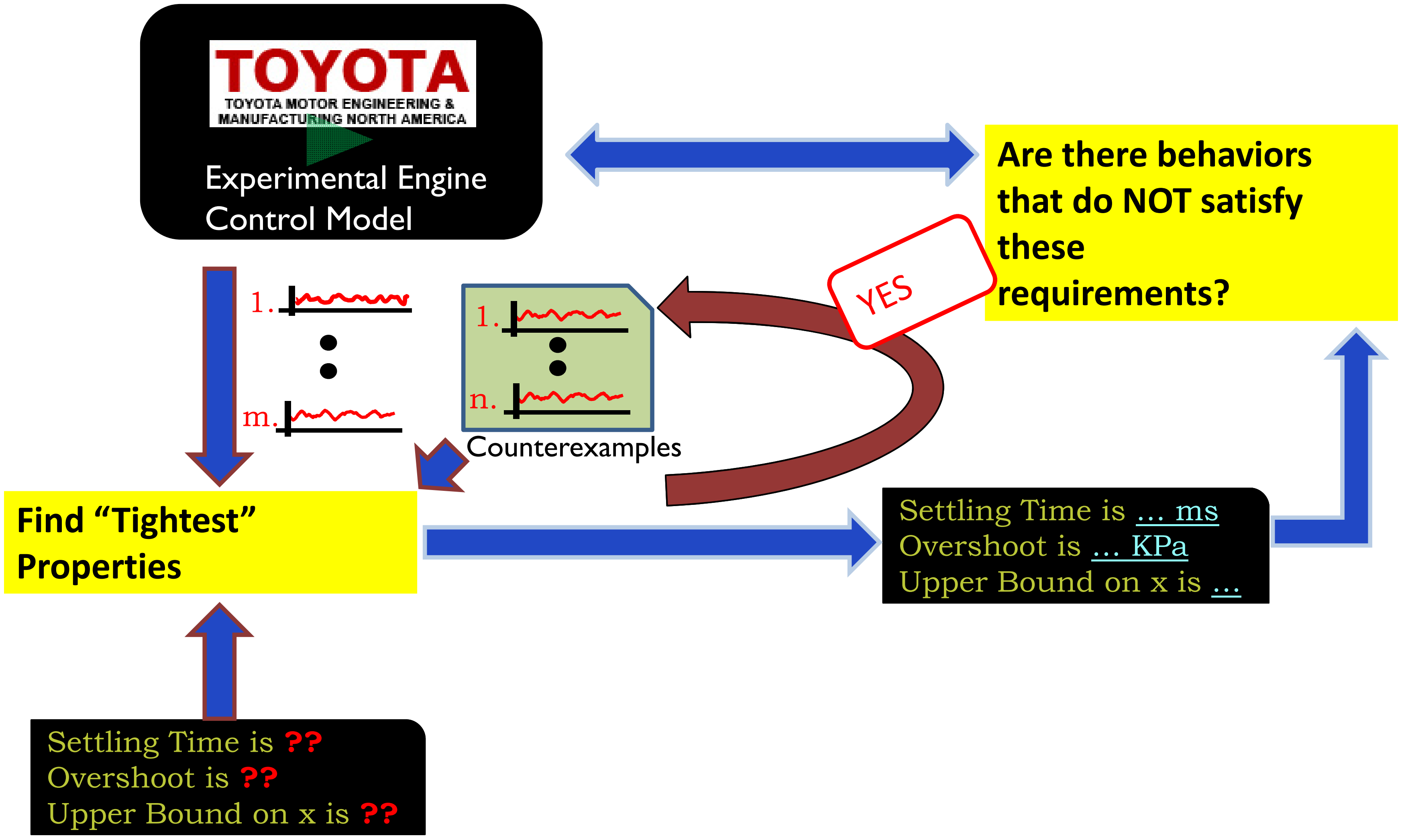


CounterExample Guided Inductive Synthesis

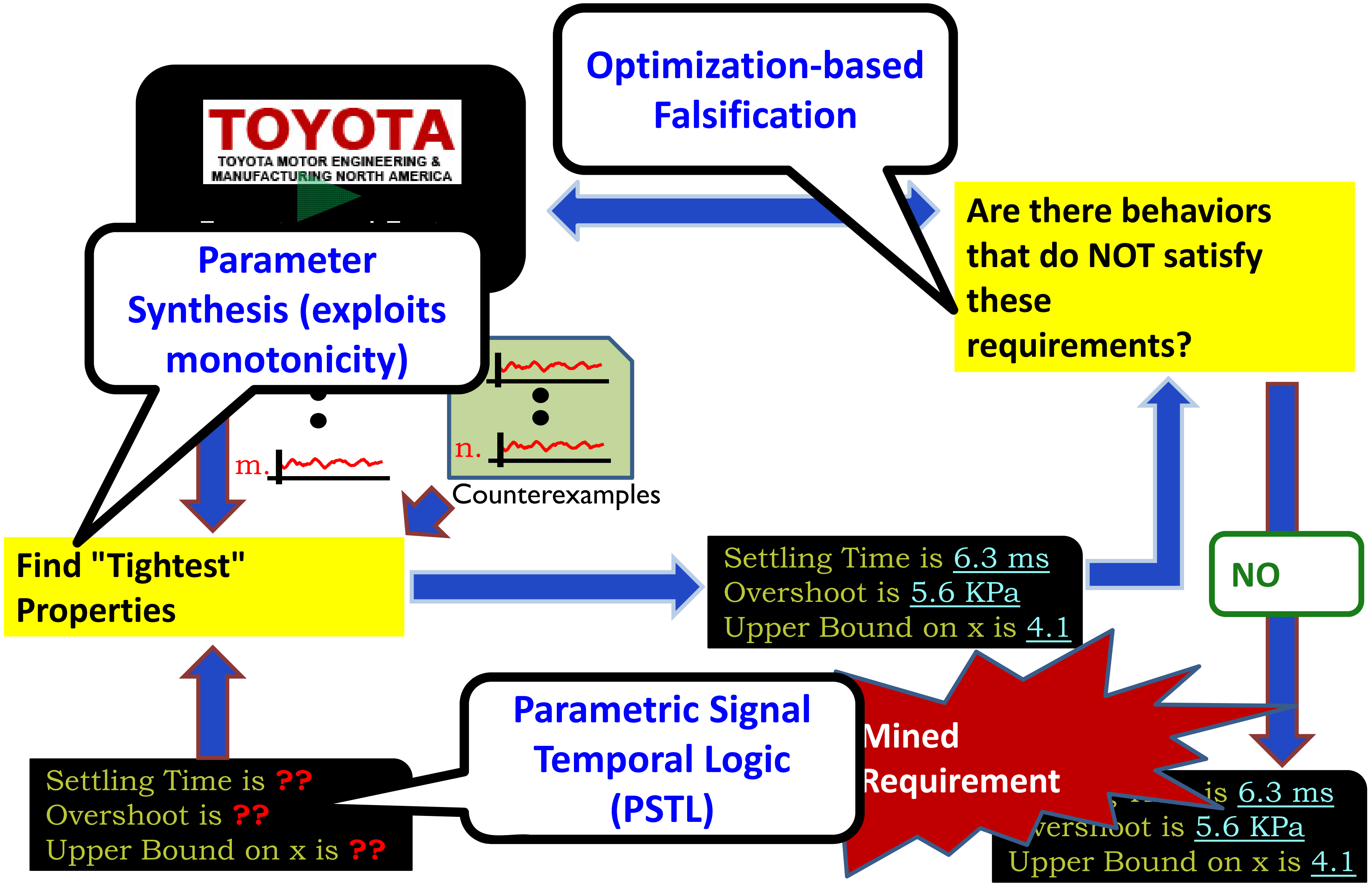
[Jin, Donze, Deshmukh, Seshia, HSCC 2013]


TOYOTA
TOYOTA MOTOR ENGINEERING & MANUFACTURING NORTH AMERICA
Experimental Engine Control Model
Are there behaviors that do NOT satisfy these requirements?
YES
1.
m.
Find "Tightest" Properties
Settling Time is 5 ms
Overshoot is 5 KPa
Upper Bound on x is 3.6
Settling Time is ??
Overshoot is ??
Upper Bound on x is ??

CounterExample Guided Inductive Synthesis


TOYOTA
TOYOTA MOTOR ENGINEERING & MANUFACTURING NORTH AMERICA
Experimental Engine Control Model
Are there behaviors that do NOT satisfy these requirements?
YES
1.
m.
1.
n.
Counterexamples
Find "Tightest" Properties
Settling Time is … ms
Overshoot is … KPa
Upper Bound on x is …
Settling Time is ??
Overshoot is ??
Upper Bound on x is ??

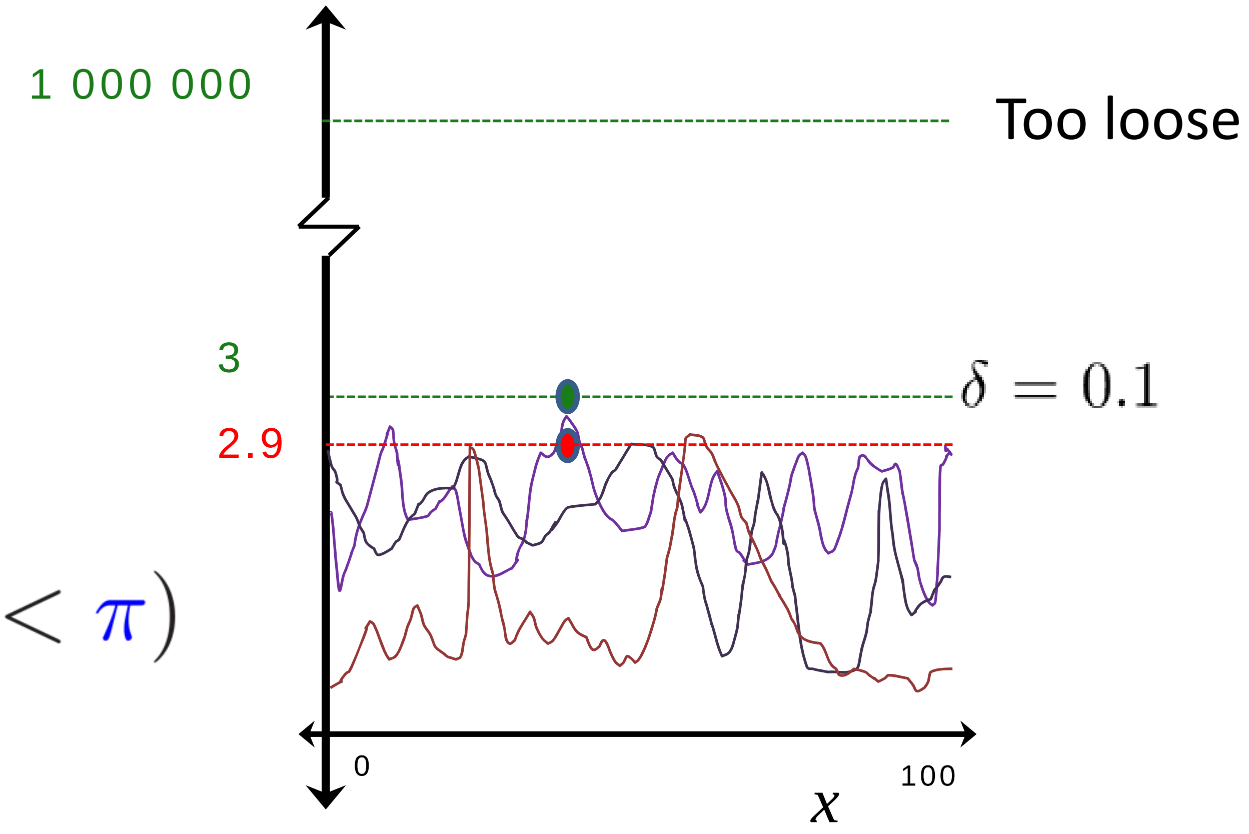
CounterExample Guided Inductive Synthesis


TOYOTA
TOYOTA MOTOR ENGINEERING & MANUFACTURING NORTH AMERICA
Optimization-based Falsification
Are there behaviors that do NOT satisfy these requirements?
Parameter Synthesis (exploits monotonicity)
m.
n.
Counterexamples
Find "Tightest" Properties
Settling Time is 6.3 ms
Overshoot is 5.6 KPa
Upper Bound on x is 4.1
NO
Parametric Signal Temporal Logic (PSTL)
Mined Requirement
Settling Time is ??
Overshoot is ??
Upper Bound on x is ??
Settling Time is 6.3 ms
Overshoot is 5.6 KPa
Upper Bound on x is 4.1

Parameter Synthesis = Find δ-tight values of params (for suitably small δ)

Find "Tightest" Properties

$\square(x < \pi)$

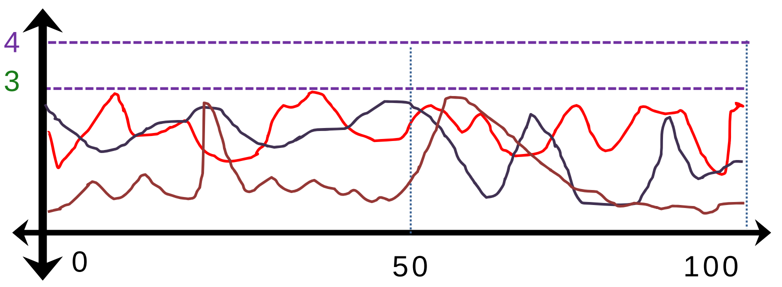


Want the value of π corresponding to the "tightest" satisfaction over a set of traces

Satisfaction Monotonicity

Find "Tightest" Properties

- Satisfaction function monotonic in parameter value
- Example: $\square(x < \pi)$



If upper bound of all signals is 3, any number > 3 is also an upper bound

- $\rho(\pi, x) = \inf_t (\pi - x(t))$
- For all x, $\rho(\pi, x)$ is a monotonic function of π
- Advantage: If monotonic, use binary search over param space, otherwise exhaustive search

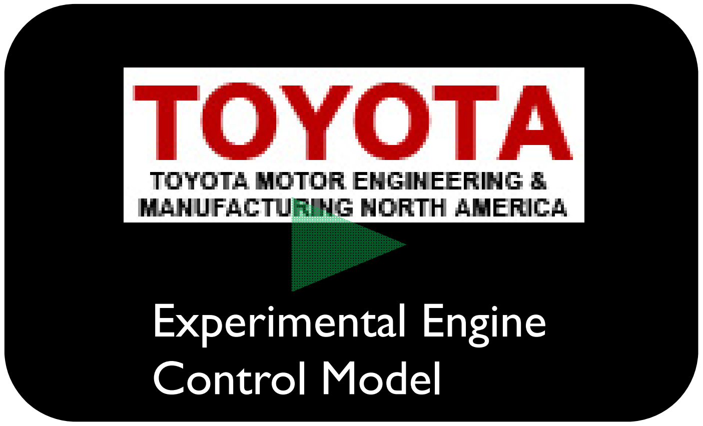
Deciding Satisfaction Monotonicity -- use SMT solving!

Find "Tightest" Properties

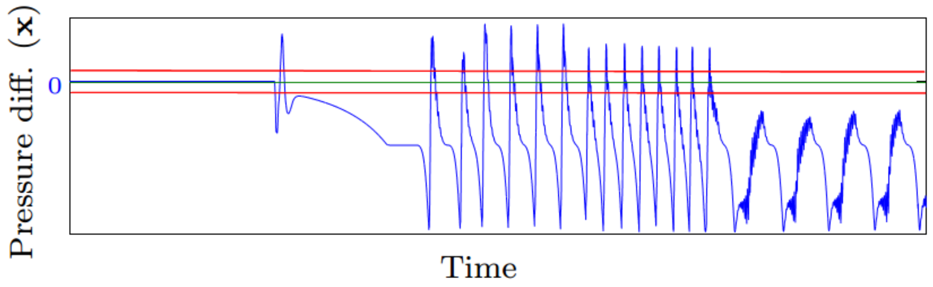
- Need to decide whether:

  For all x, $\rho(\pi, x)$ is a monotonic function of π
- Theorem: Deciding monotonicity of a PSTL formula is undecidable
- Use an encoding to satisfiability modulo theories (SMT) solving
  - Quantified formulas involving uninterpreted functions, and arithmetic over reals → linear arithmetic if predicates are linear

Experimental Results on Industrial Airpath Controller [Jin, Donze, Deshmukh, Seshia, HSCC 2013]

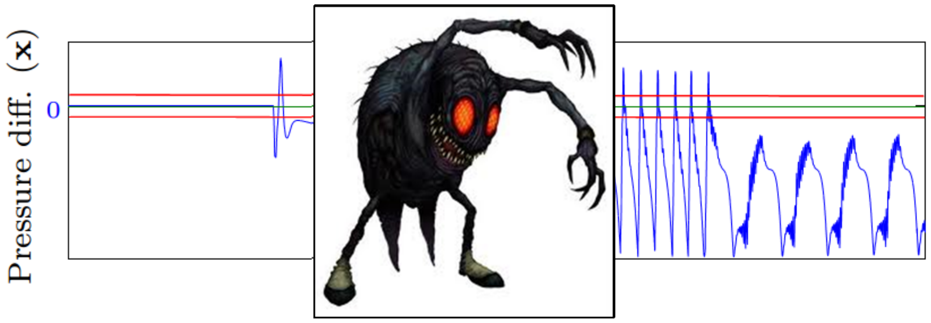

TOYOTA
TOYOTA MOTOR ENGINEERING & MANUFACTURING NORTH AMERICA
Experimental Engine Control Model

- 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)



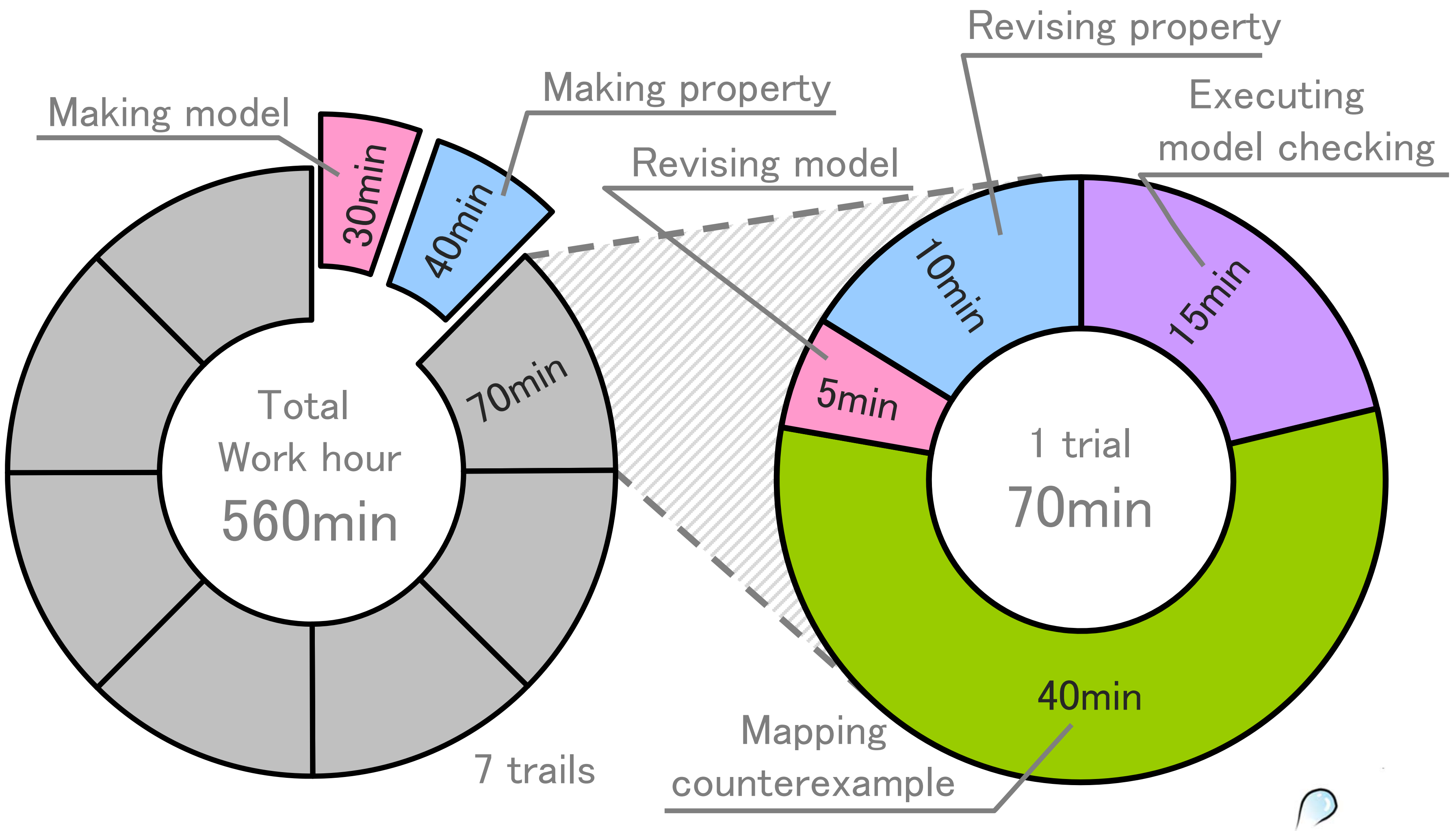
Mining can expose deep bugs

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



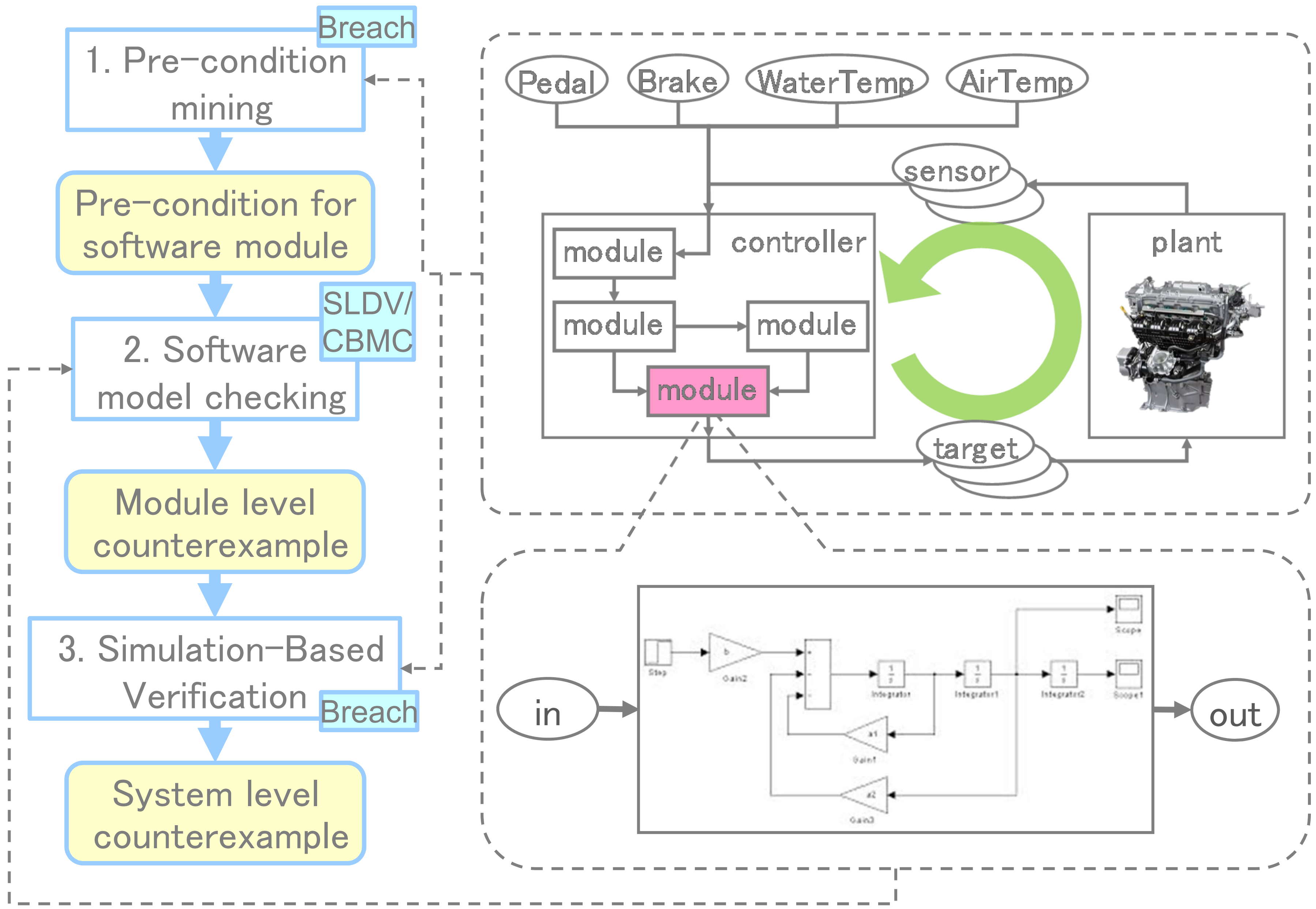
- Uncovered a tricky bug
  - Discussion with control designer revealed it to be a real bug
  - Root cause identified as wrong value in a look-up table, bug was fixed
- Duality between spec mining and bug-finding:
  - Synthesizing “tightest” spec could uncover corner-case bugs
  - Looking for bugs ≈ Mine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

Toyota Unit's Experience with Model Checking
[FMCAD'16]

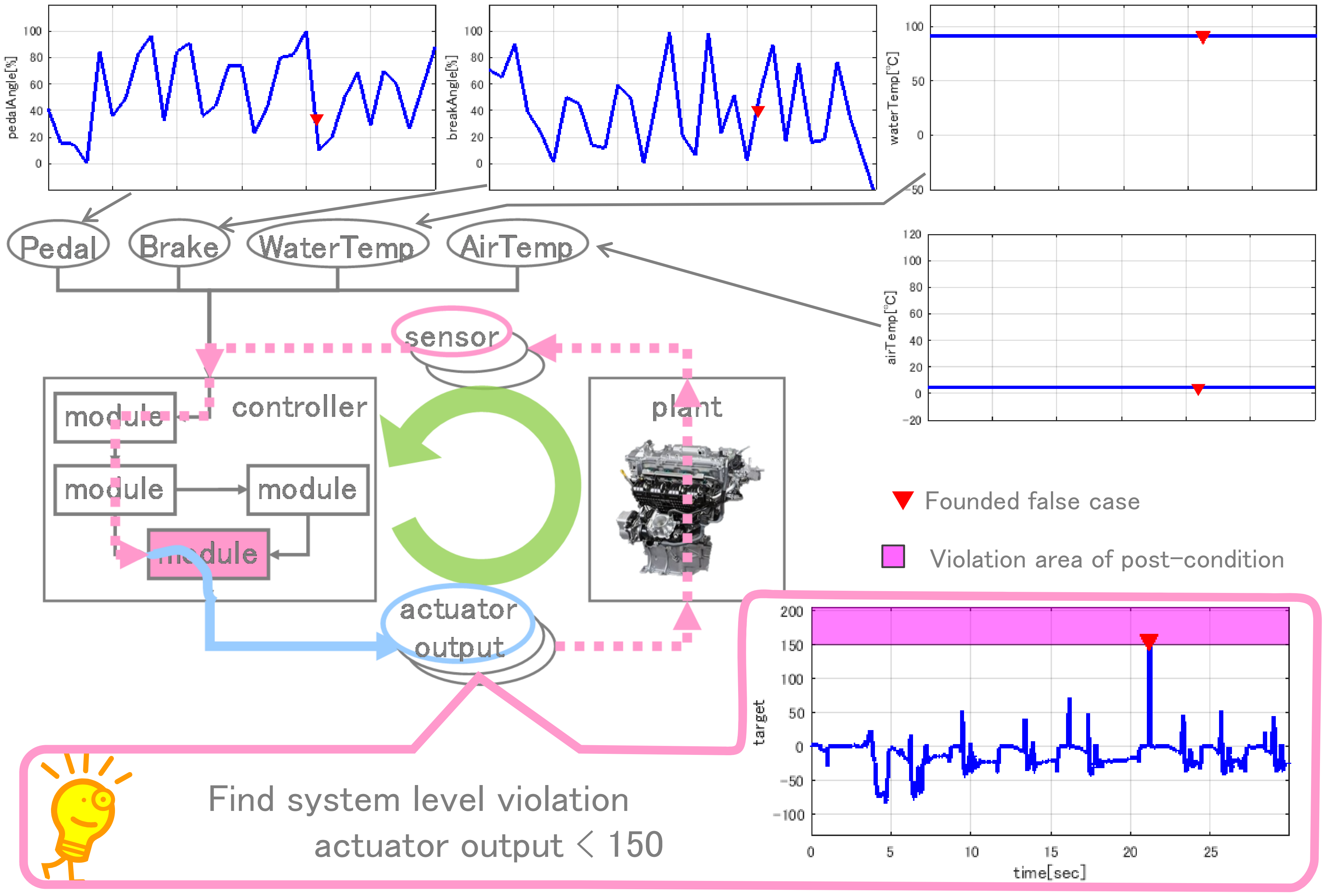


Making/revising property: 110 min
Mapping counterexample: 280 min for just 1 module

Overview of FMCAD'16 methodology



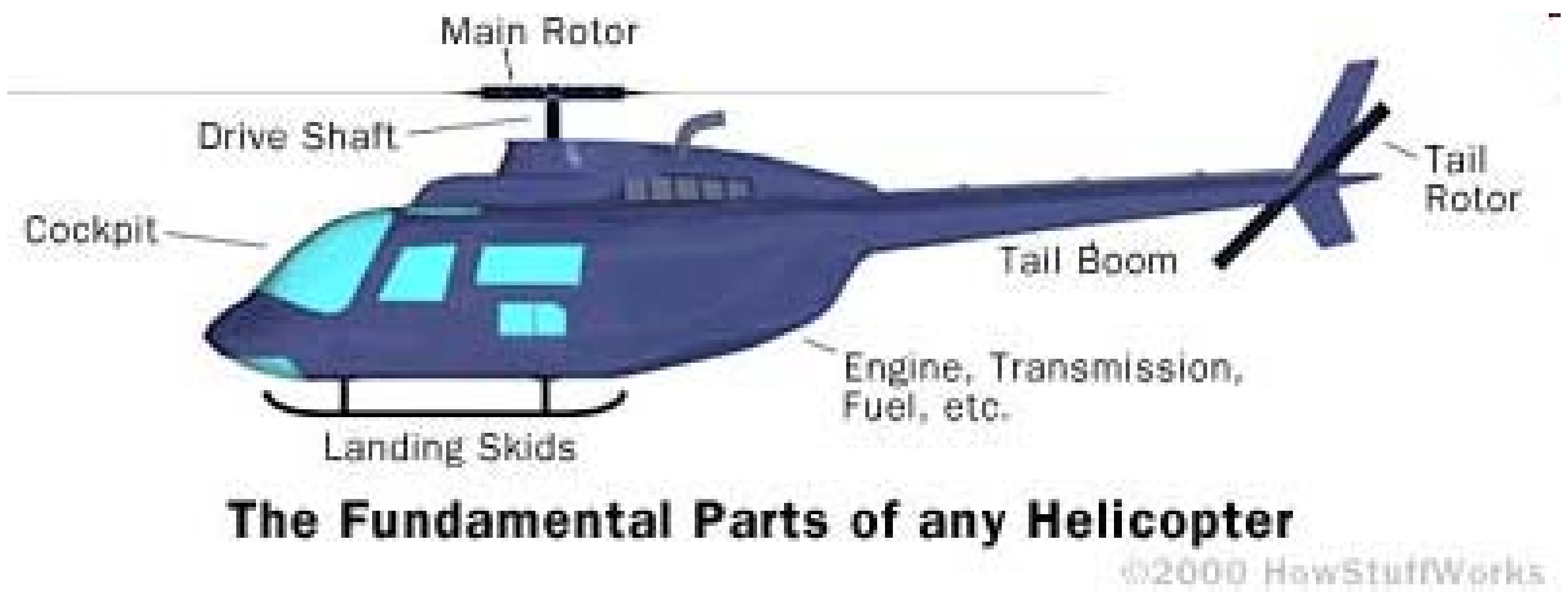
Requirement Mining In Toyota Case Study [FMCAD'16]



Find system level violation
actuator output < 150

Demo: Requirement Mining for a Helicopter Control Model

An Example: Modeling Helicopter Dynamics (taken from textbook available at http://LeeSeshia.org, Chapter 2)

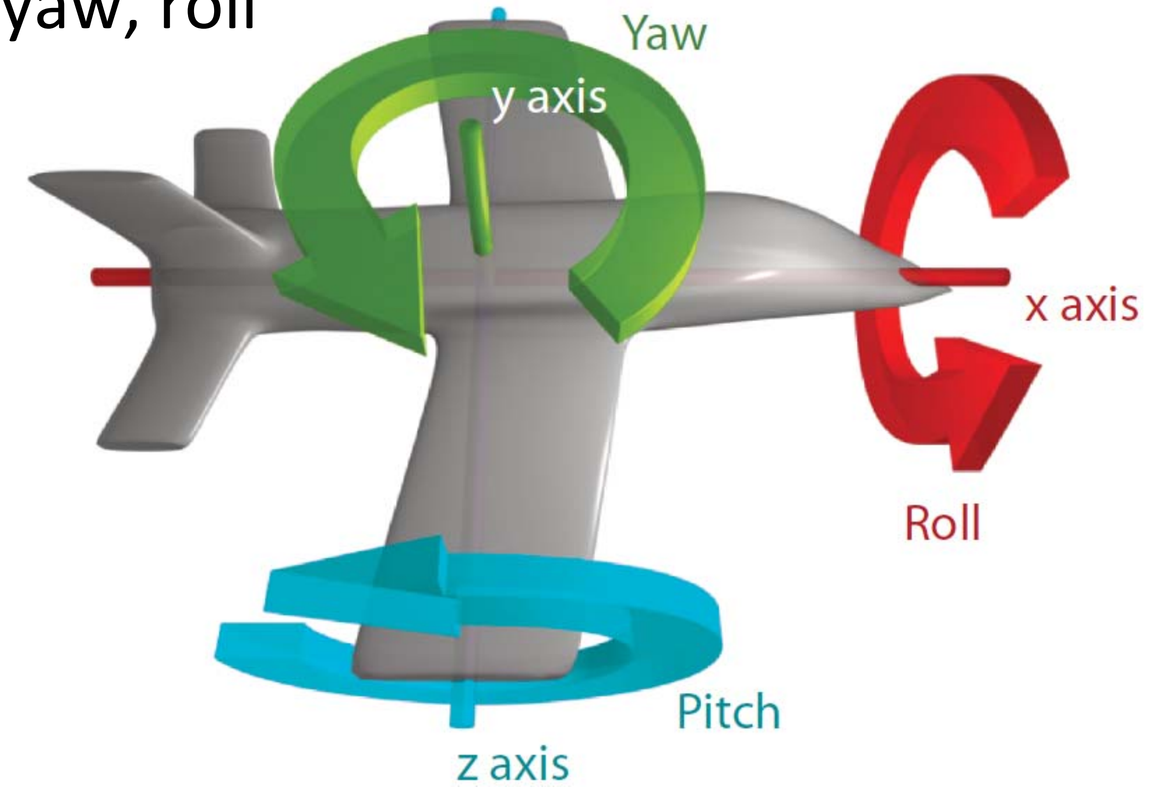


Go to **Breach** webpage for this example:
http://www.eecs.berkeley.edu/~donze/mining_example.html

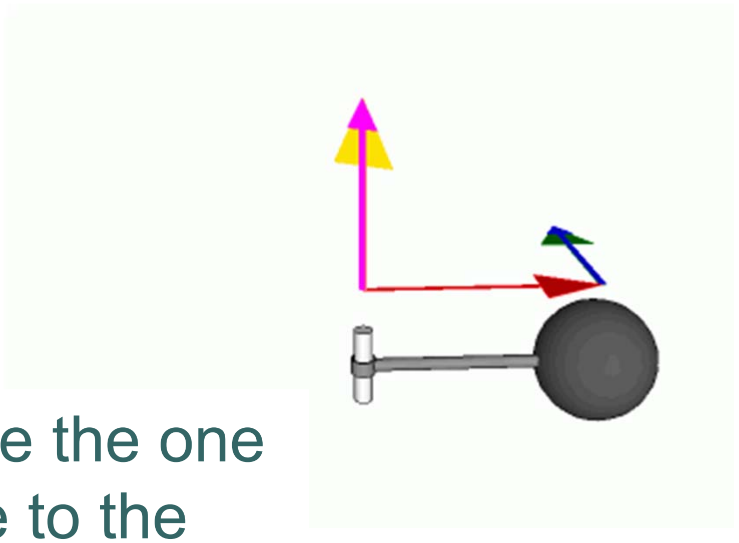
Modeling Physical Motion

Six degrees of freedom:

- Position: x, y, z
- Orientation: pitch, yaw, roll



Feedback Control Problem

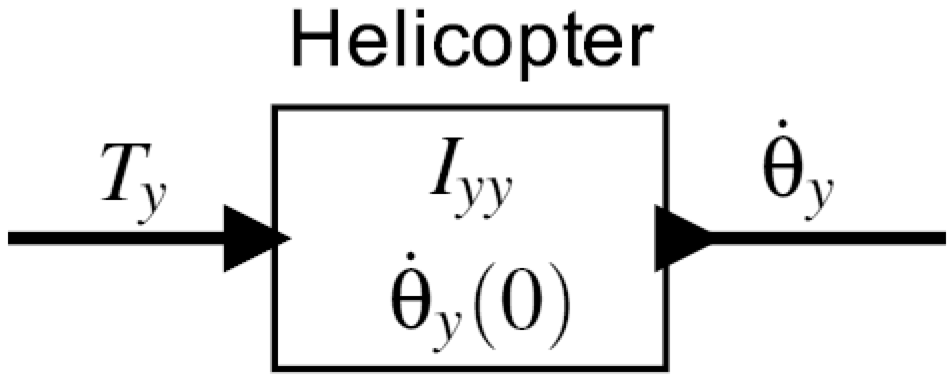


A helicopter without a tail rotor, like the one below, will spin uncontrollably due to the torque induced by friction in the rotor shaft.

Control system problem:
Apply torque using the tail rotor to counterbalance the torque of the top rotor.

Model of the helicopter

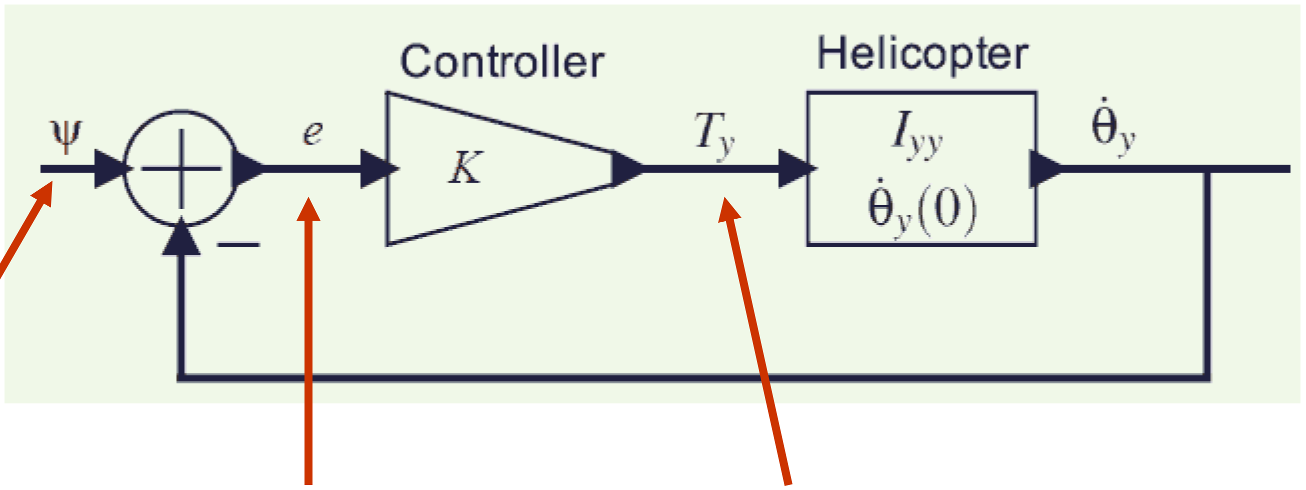
Input is the net torque of the tail rotor and the top rotor. Output is the angular velocity around the *y* axis.



Parameters of the model are shown in the box. The input and output relation is given by the equation to the right.

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

Proportional controller



desired angular velocity

error signal

$$e(t) = \psi(t) - \dot{\theta}_y(t)$$

net torque

$$T_y(t) = Ke(t)$$

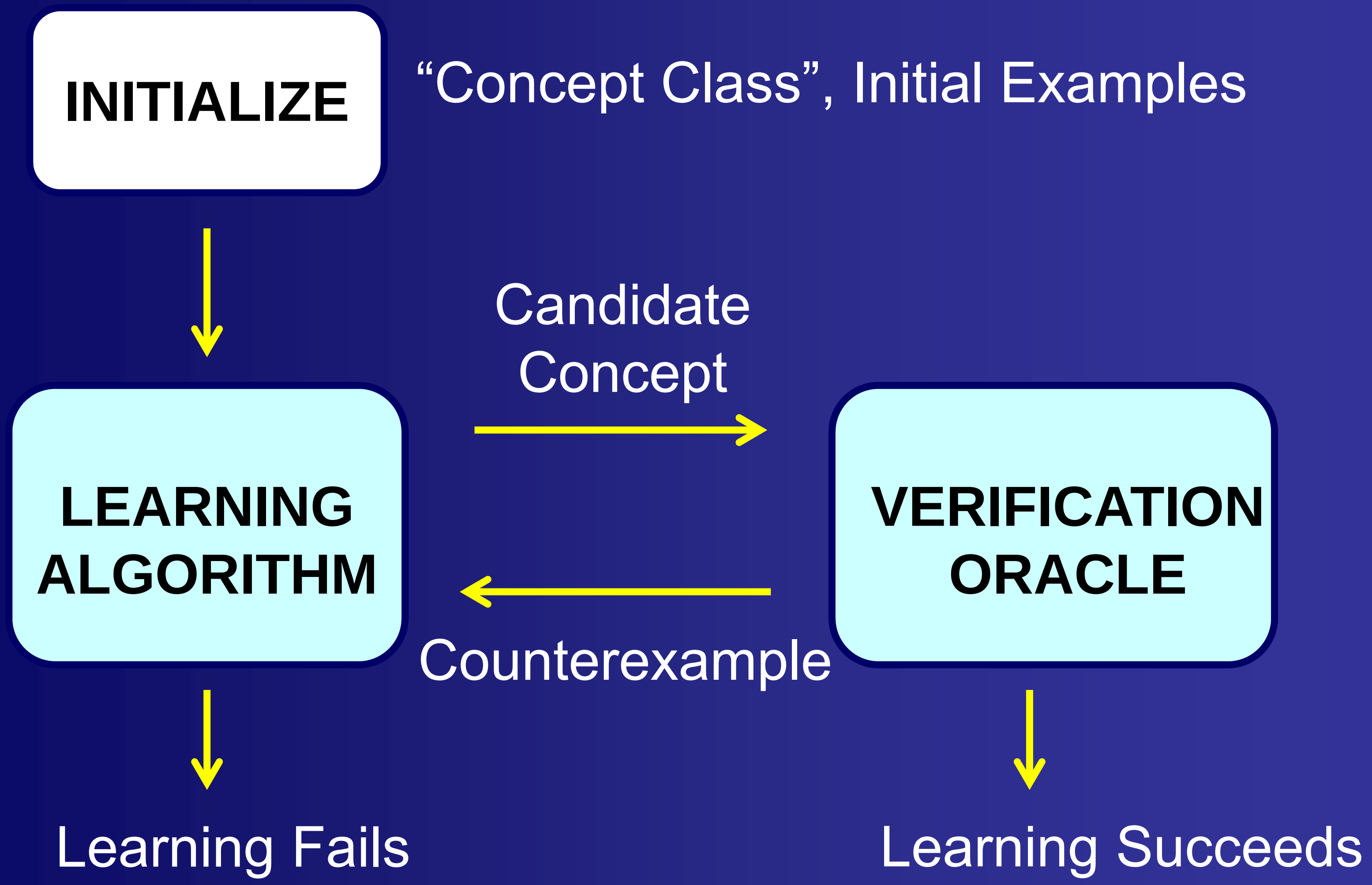
How long does it take for theta_dot to reach desired psi?

Summary of Part 2

- Syntax-Guided Synthesis
  – Problem Definition
  – Examples
  – CEGIS
  – Decidability
- Requirement Mining for Closed-Loop Control Systems
  – Use of CEGIS, but for synthesis of STL properties
  – SMT solving used to determine satisfaction monotonicity

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/design Oracle)*	*Rare (black-box oracles)*

* Between typical inductive synthesizer and machine learning algo

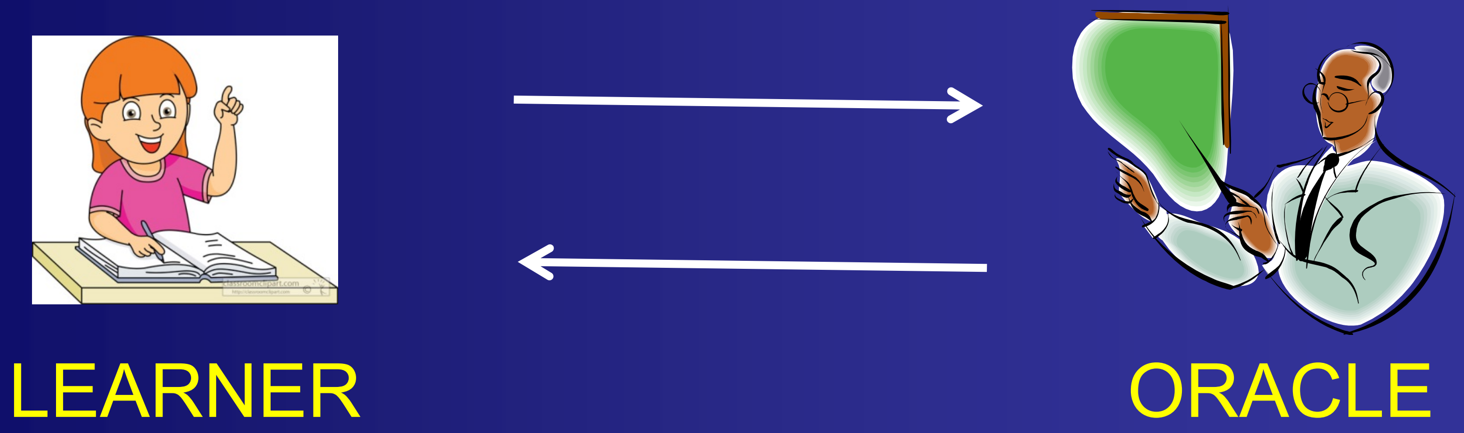
Formal Inductive Synthesis

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

- Example:
  - C: all affine functions f of $x \in R$
  - ϕ: $\forall x.\ f(x) \geq x + 42$
  - O = {(pos-witness, (x, f(x)) satisfying ϕ)}

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 ϕ)

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 $f = \phi$?

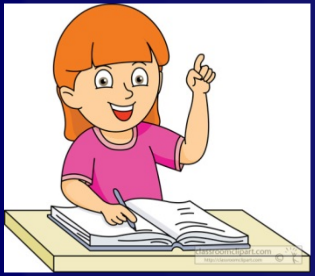
Yes / No + $x \in \phi \oplus f$

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. $\perp$



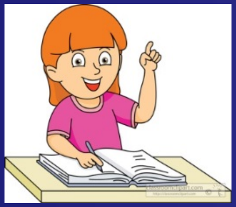
LEARNER

ORACLE

Formal Inductive Synthesis

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

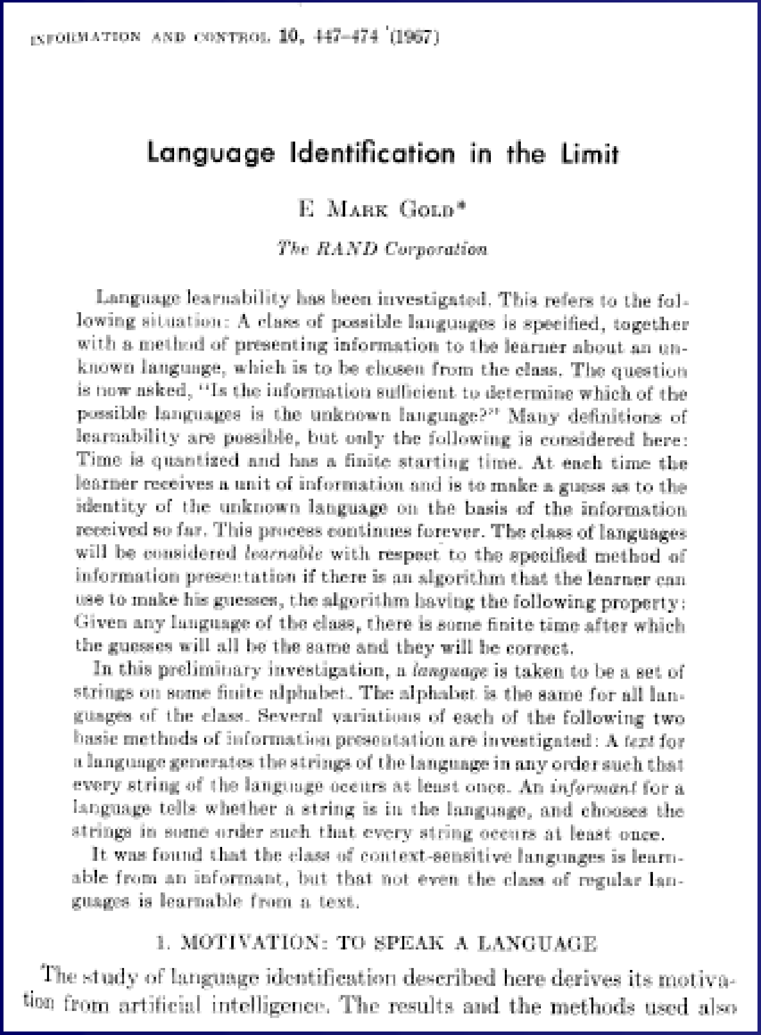
Design/Select:



Oracle-Guided Inductive Synthesis (OGIS)

- A dialogue is a sequence of (query, response) conforming 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 ϕ

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**
- **CEGAR**
- **CEGIS used in Program Synthesis/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

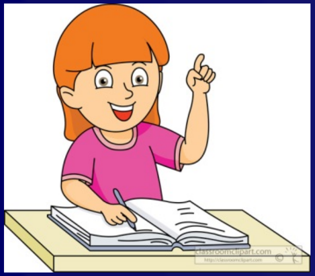
Feature	Formal Inductive Synthesis	Machine Learning
Concept/Program		imple
		alized
Lea		mate, w/ unction
Ora		lack-box cles)

What can we prove about convergence/complexity 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

$$\begin{tikzcd}[column sep=huge] \text{LEARNER} \arrow[r, leftrightarrow, "\text{Positive Witness}", "x \in \phi\text{, if one exists, else } \perp"'] & \text{ORACLE} \\ {} \arrow[r, leftrightarrow, "\text{Counterexample to } f?", "\text{Yes + counterexample } x \,/\, \perp"'] & {} \end{tikzcd}$$



- 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?

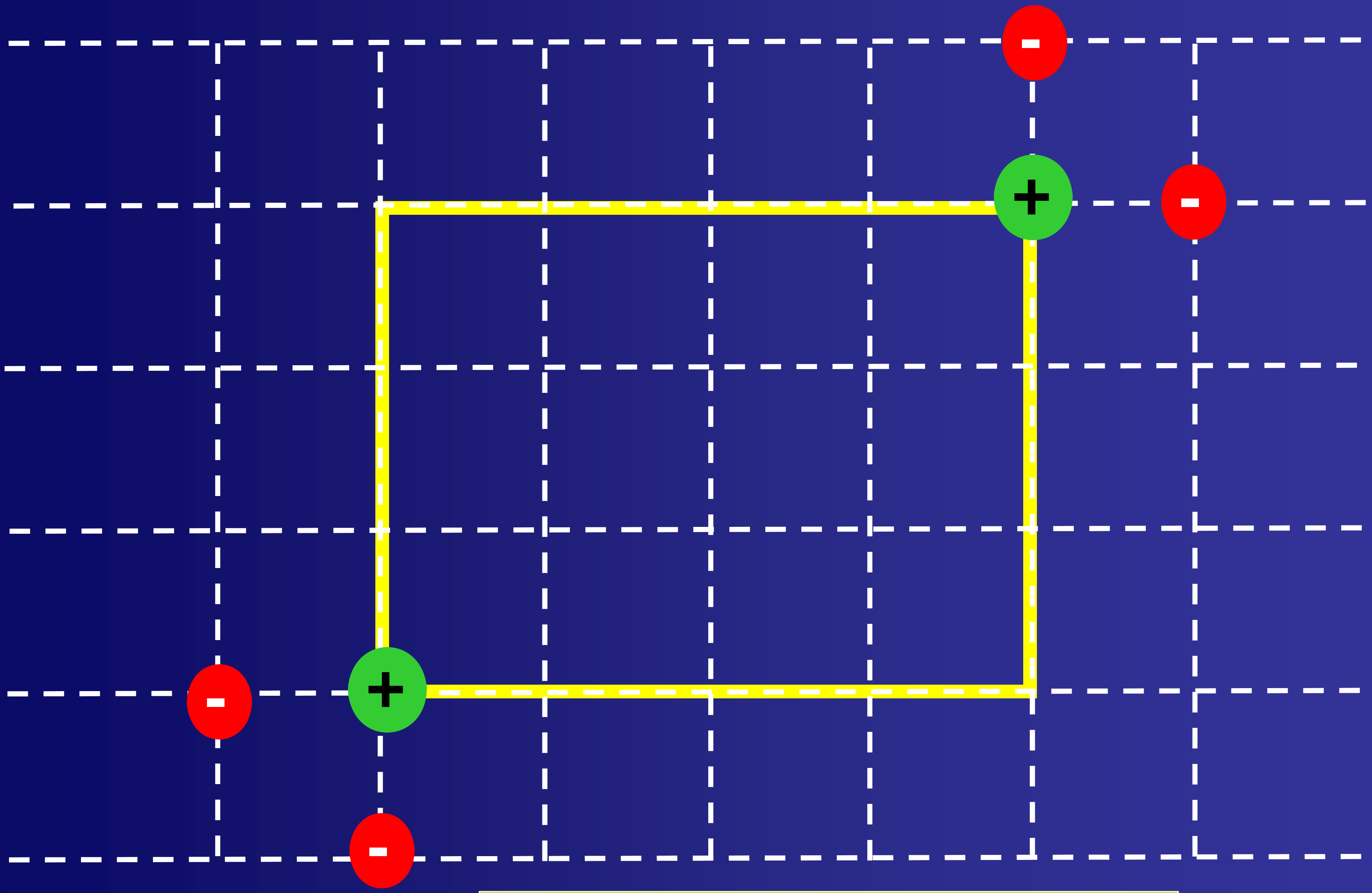
Problem 1: Bounds on Sample Complexity

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

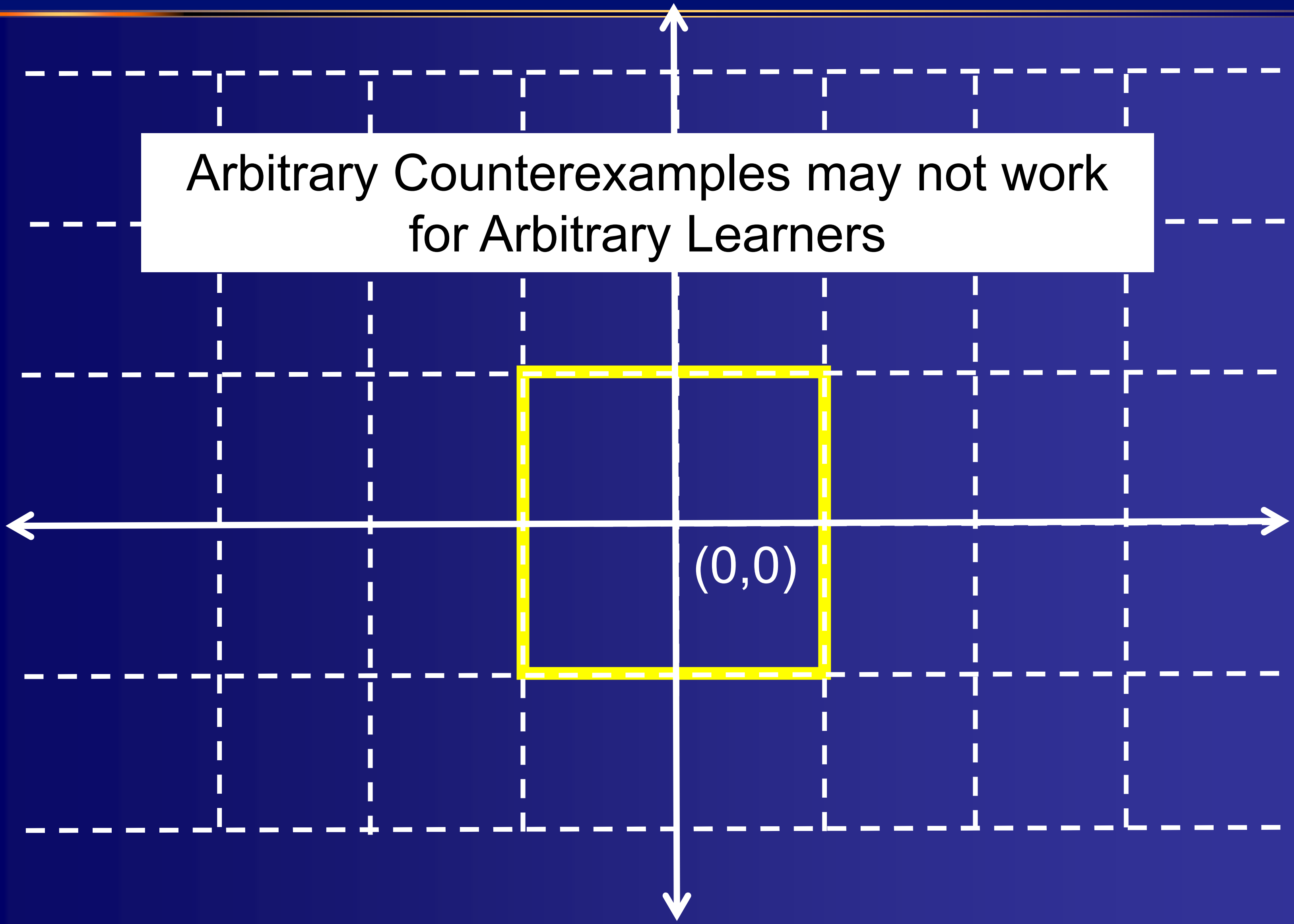
Theoretical Results: Num. of Queries needed for Finite Program Classes with CEGIS

- Thm 1: NP-hard to find minimum number of queries for CEGIS oracle interface
  - CEGIS int. = {counterexample, positive witness}
- Thm 2: Teaching Dimension of Program Class is lower bound on query complexity
  - TD: min number of queries needed to uniquely identify any program in the class
- Thm 3: Teaching Dimension of Octagons is $O(d^2)$ where d is the dimension of the space
  - Relevant for Synthesis of "Octagon" Invariants

[Jha & Seshia '15; Jha, Seshia, Zhu, SYNT'16]

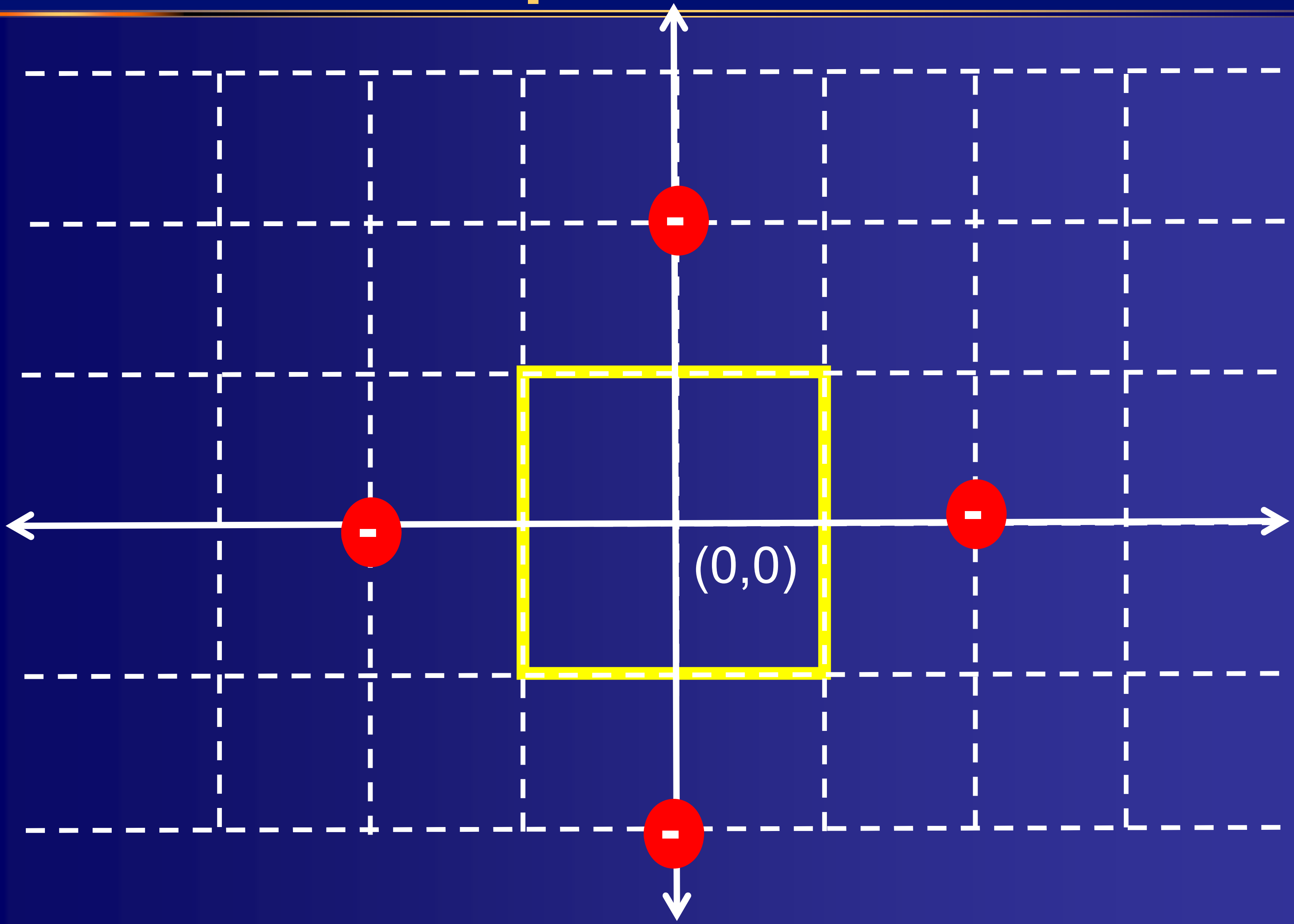
Problem 2: Convergence of CEGIS for Infinite-Sized Program Classes

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



Arbitrary Counterexamples may not work for Arbitrary Learners

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



Types of Counterexamples

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

– 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

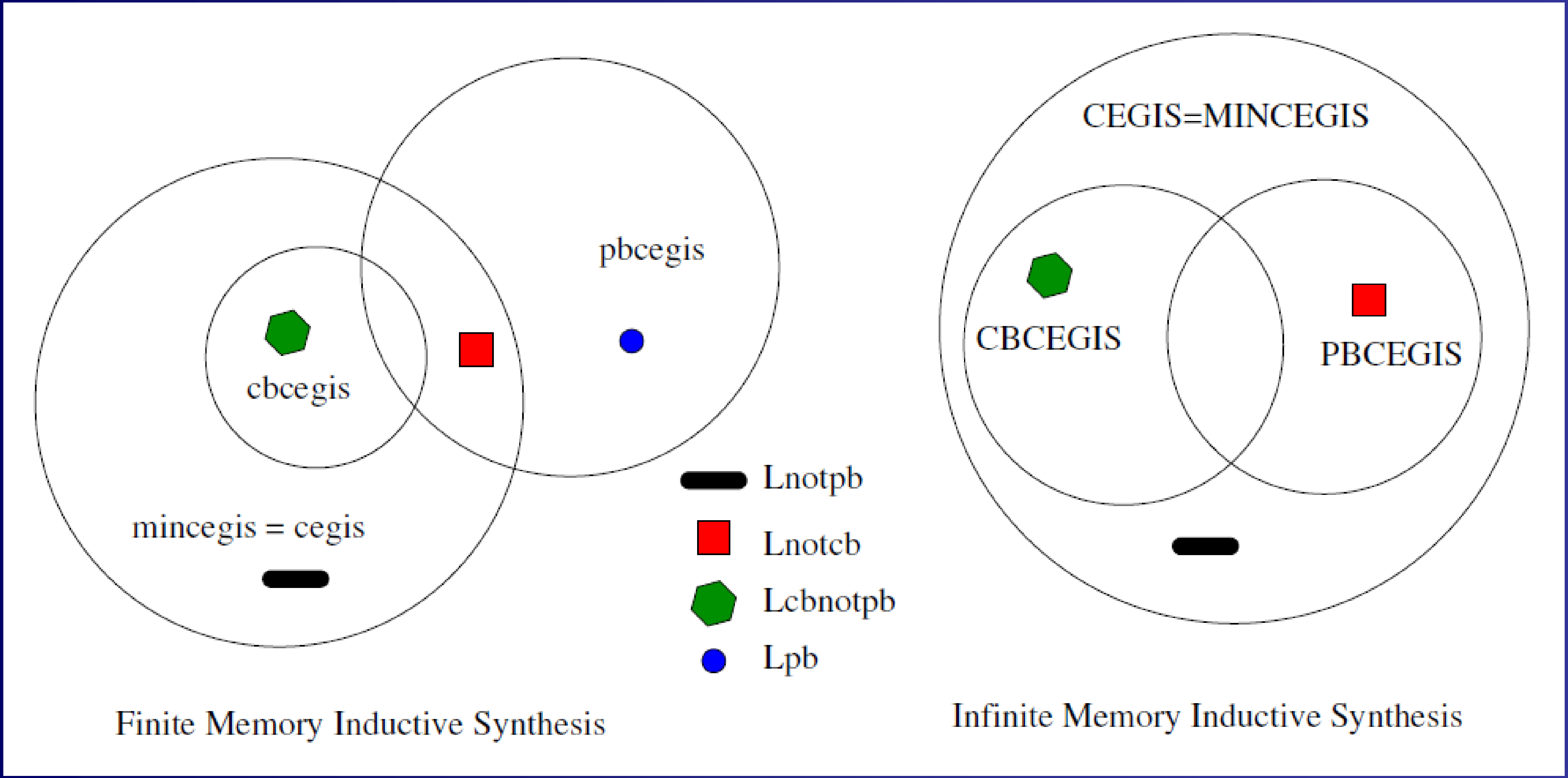
Assume there is a function $size: D \rightarrow N$

- **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; '15]



Open Problems

- For Finite Domains: Prove results 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?
- Broaden Program Classes & Properties of Learner/Verifier considered

Summary

- Formal Synthesis and its Applications
  - *Reduction* to Synthesis
  - Solve via *Inductive* Synthesis
- Syntax-Guided Synthesis
- Industrial Case Study: Synthesis of Specifications
- 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!