UCLID5's Elements: Formal Modeling, Verification, Synthesis, and Learning

Sanjit A. Seshia

Professor

EECS, UC Berkeley

Joint work with
Kevin Cheang, Pranav Gaddamadugu, Adwait Godbole, Federico Mora,
Kevin Lauefer, Shaokai Lin, Yatin Manerkar, Rohit Sinha, Elizabeth Polgreen,
Cameron Rasmussen, Jonathan Shi, Pramod Subramanyan

https://github.com/uclid-org/uclid

SAT-SMT Winter School @ FSTTCS
December 16-17, 2022

Overview of this Tutorial

An **introduction to UCLID5**, a system for formal modeling, verification and synthesis of computational systems

- ✓ Motivation – Verification of Trusted Computing Platforms
- ✓ Multi-Modal Modeling with UCLID5
- Verification by Reduction to Synthesis
- Syntax-Guided Synthesis
- Formal Inductive Synthesis & Oracle-Guided Inductive Synthesis
- Satisfiability and Synthesis Modulo Oracles

Formal Synthesis

- Given:
  - Class of Artifacts C
  - Formal (mathematical) Specification ϕ
- Find $f \in C$ that satisfies ϕ
- Example 1:
  - C: all affine functions f of $x \in R$
  - ϕ: $\forall x.\ f(x) \geq x + 42$
- Example 2: SyGuS
- Example 3: Reactive synthesis (from LTL)

C → defined by grammar
ϕ → SMT formula

ϕ → LTL property
C → set of all FSMs

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
- **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 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 programs/designs/controllers, etc.)

Verification by Reduction 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, Frames in IC3/PDR
- Ranking functions (for proofs of termination)
- Intermediate lemmas for compositional proofs
- Simulation/Bisimulation Relations
- Theory lemma instances in SMT solving
- Patterns for Quantifier Instantiation in SMT solving
- …

One Reduction from Verification to Synthesis

NOTATION

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

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

VERIFICATION PROBLEM

Does M satisfy Ψ?

↓

SYNTHESIS PROBLEM

Synthesize ϕ s.t.

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

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

Two Reductions from Verification to Synthesis

NOTATION

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

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

VERIFICATION PROBLEM

Does M satisfy Ψ?

SYNTHESIS PROBLEM #1

Synthesize ϕ s.t.

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

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

SYNTHESIS PROBLEM #2

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

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

s.t.

$\alpha(M)$ satisfies Ψ

iff

M satisfies Ψ

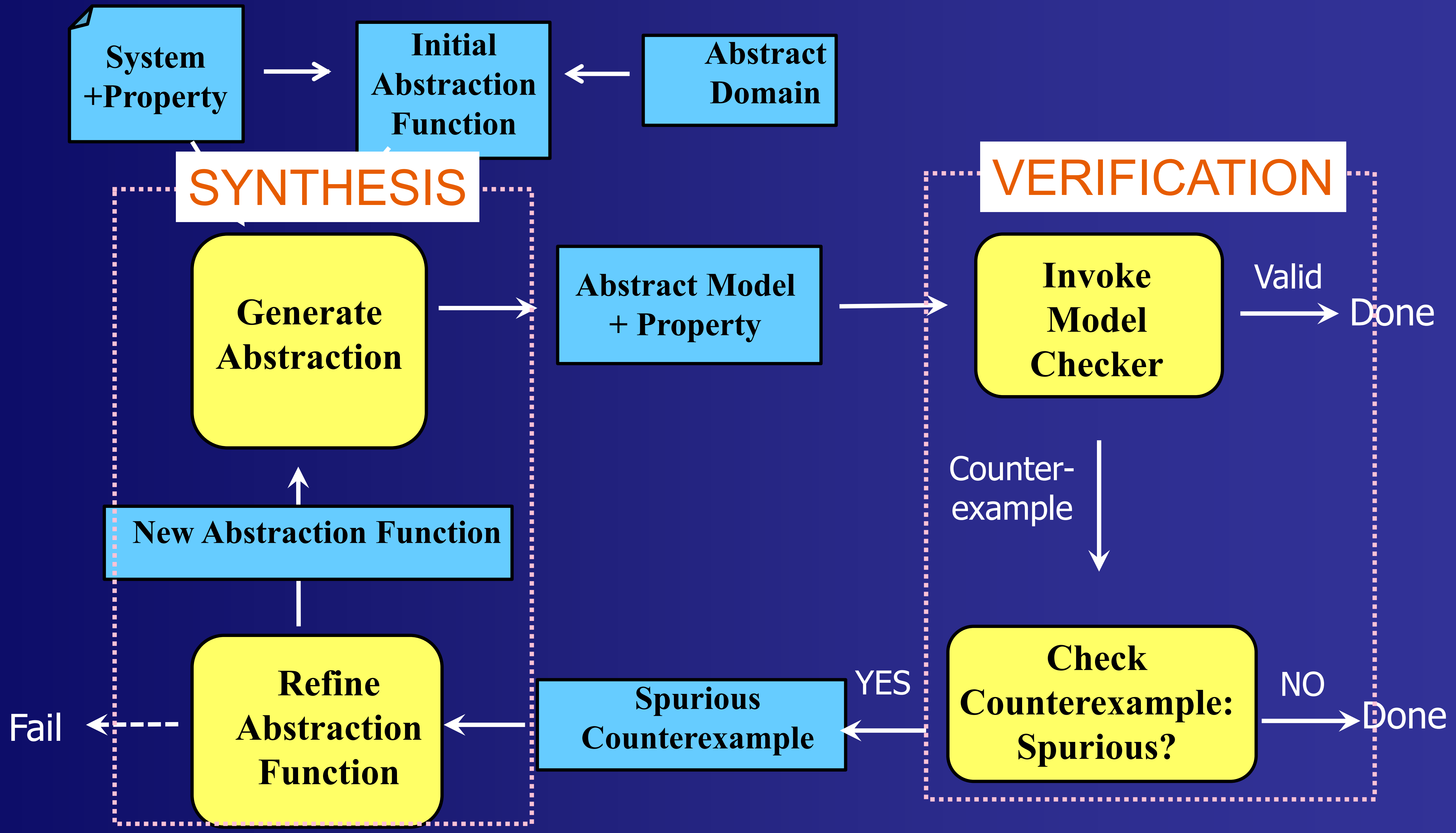
Common Approach for both: Inductive Synthesis

Synthesis of:-

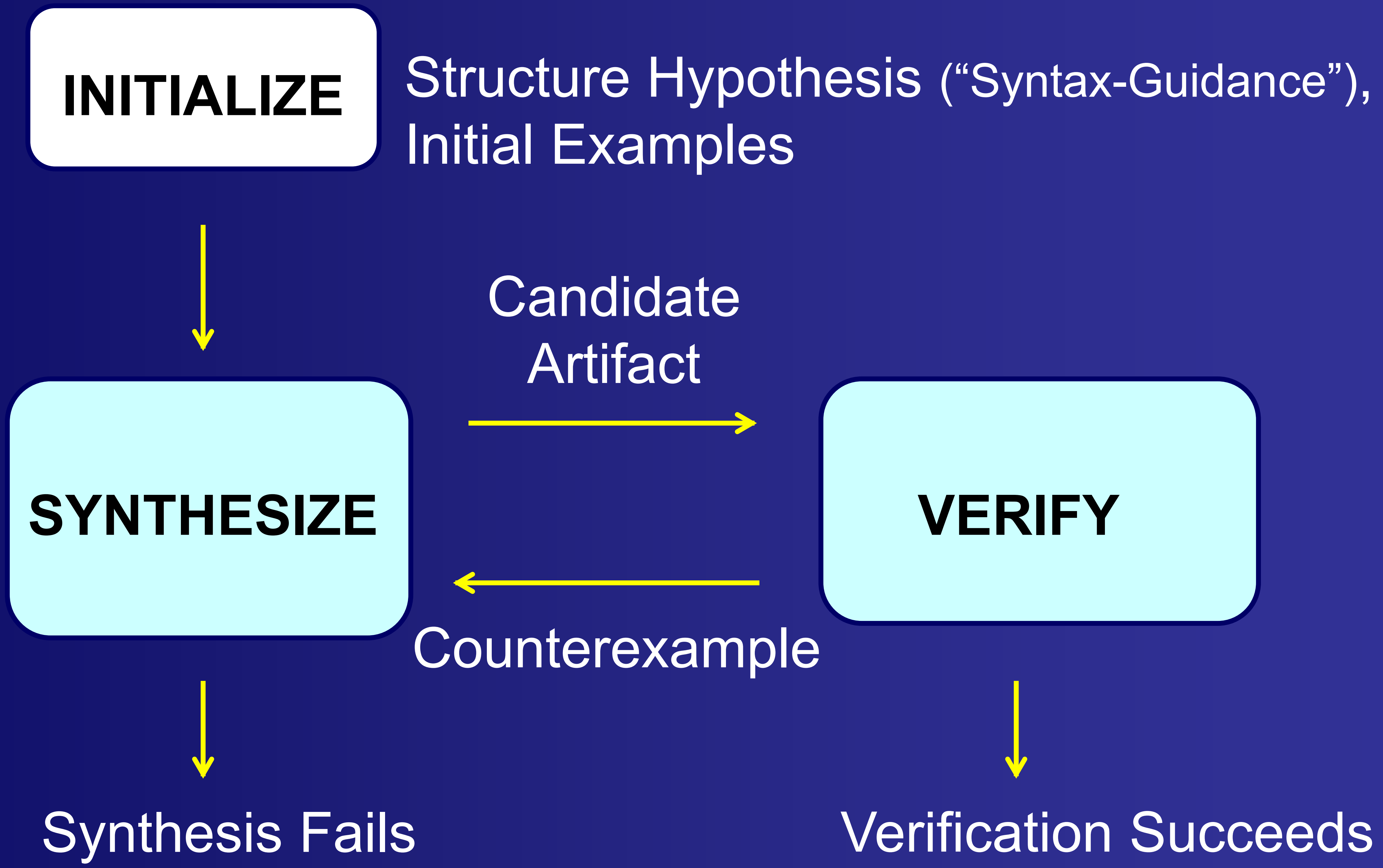
- Inductive Invariants
  - Choose templates for invariants
  - Infer likely invariants from tests (examples)
  - Check if any are true inductive invariants, possibly iterate
- Abstraction Functions
  - Choose an abstract domain
  - Use Counter-Example Guided Abstraction Refinement (CEGAR)

Counterexample-Guided Abstraction Refinement (CEGAR) is Inductive Synthesis/Learning

[Anubhav Gupta, '06]



CEGAR = Counterexample-Guided Inductive Synthesis (of Abstractions)



Syntax-Guided Synthesis

Definition

(FMCAD 2013)

Given:

1. Set of functions $f_1, f_2, \ldots, f_k$ to be synthesized
2. Set of grammars $G_1, G_2, \ldots, G_k$ – each f_i is to be synthesized from $L(G_i)$
3. Specification ϕ – SMT formula in $T \cup (EUF)$ — $\phi(\vec{f}, \vec{X})$

Find: Expressions $e_i \in L(G_i)$ s.t. $\phi[f_i \leftarrow e_i,\ i=1,\ldots,k]$ is valid in T

If these exist, realizable.

if not, unrealizable.

$$\exists \vec{f} \in L(\vec{G})\ \forall \vec{X}\ .\ \phi(\vec{f}, \vec{X})$$

Example

$\phi(f, x, y)$ f : integer x integer → integer

Specification: (x ≤ f(x,y)) & (y ≤ f(x,y)) & (f(x,y) = x | f(x,y) = y) — QF_LIA

Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else

L(G)

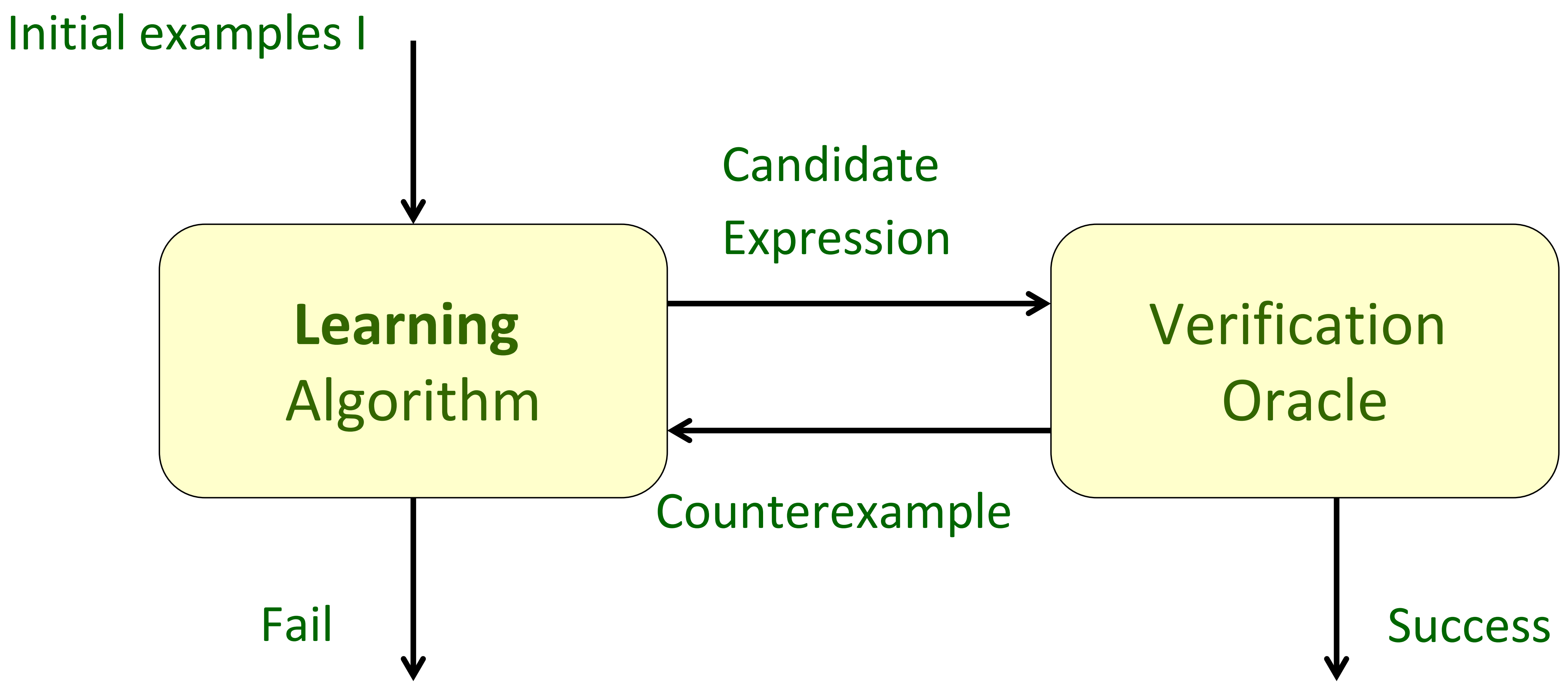
$f(x,y) = \text{ITE}(x \geq y, x, y)$

$x > y$

max

$(x \leq 0) \land (y \leq 0) \land (0 = x \mid 0 = y)$ X

SyGuS solved through Counterexample-Guided Inductive Synthesis (Counterexample-Guided Learning)



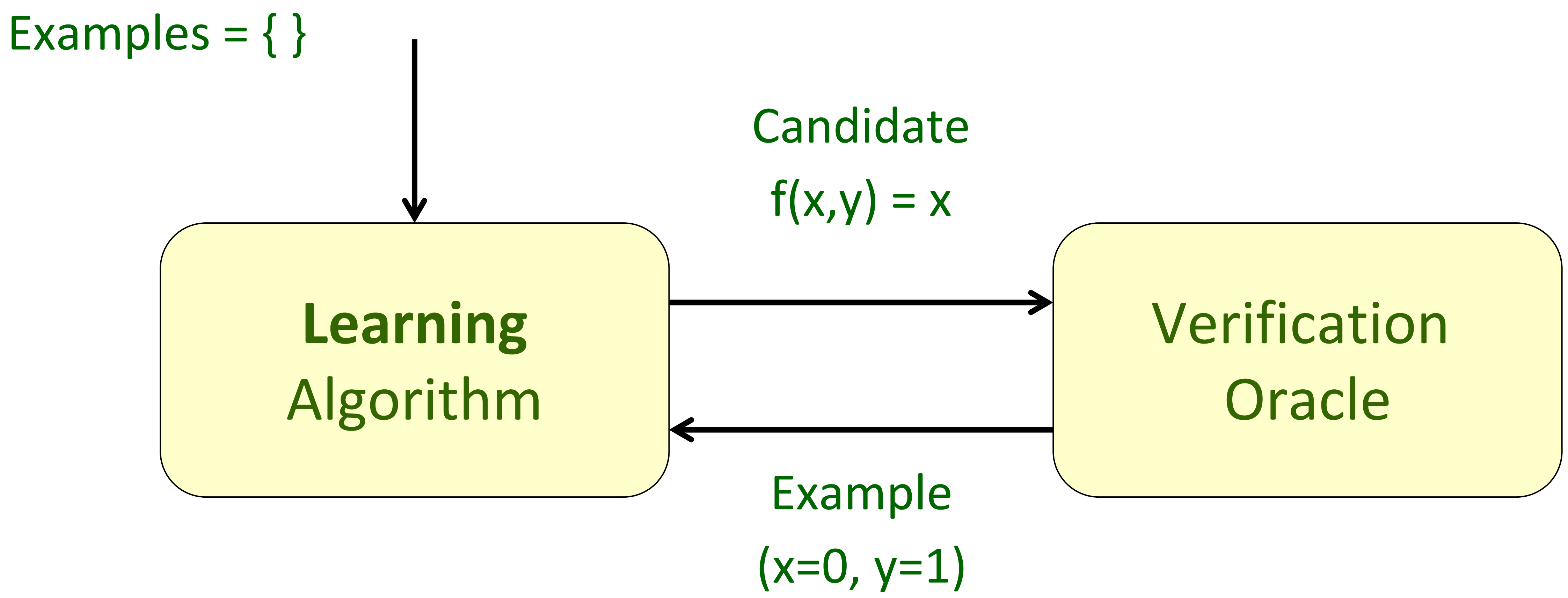
Concept class: Set E of expressions

Examples: Concrete input values

(slide adapted from one by R. Alur)

CEGIS Example

- Specification: $(x \le f(x,y))$ & $(y \le f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$
- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else

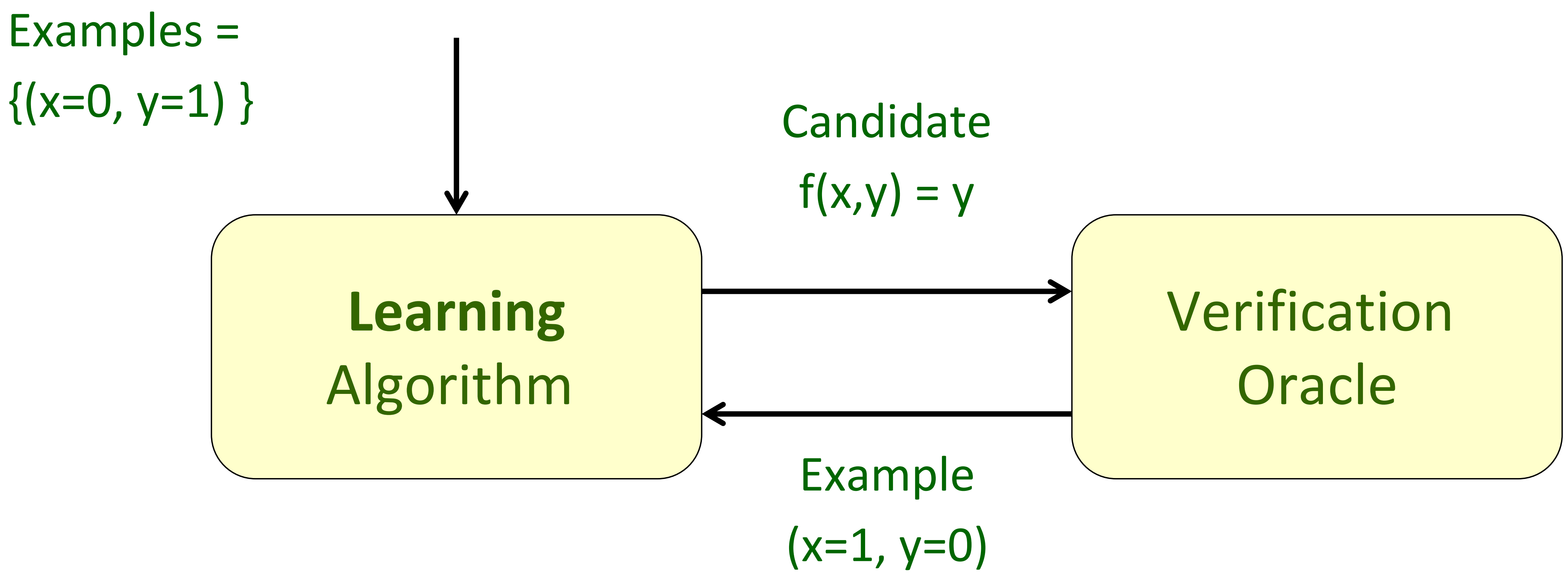


(slide adapted from one by R. Alur)

CEGIS Example

- Specification: $(x \le f(x,y))$ & $(y \le f(x,y))$ & $(f(x,y) = x \mid f(x,y) = y)$

- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else



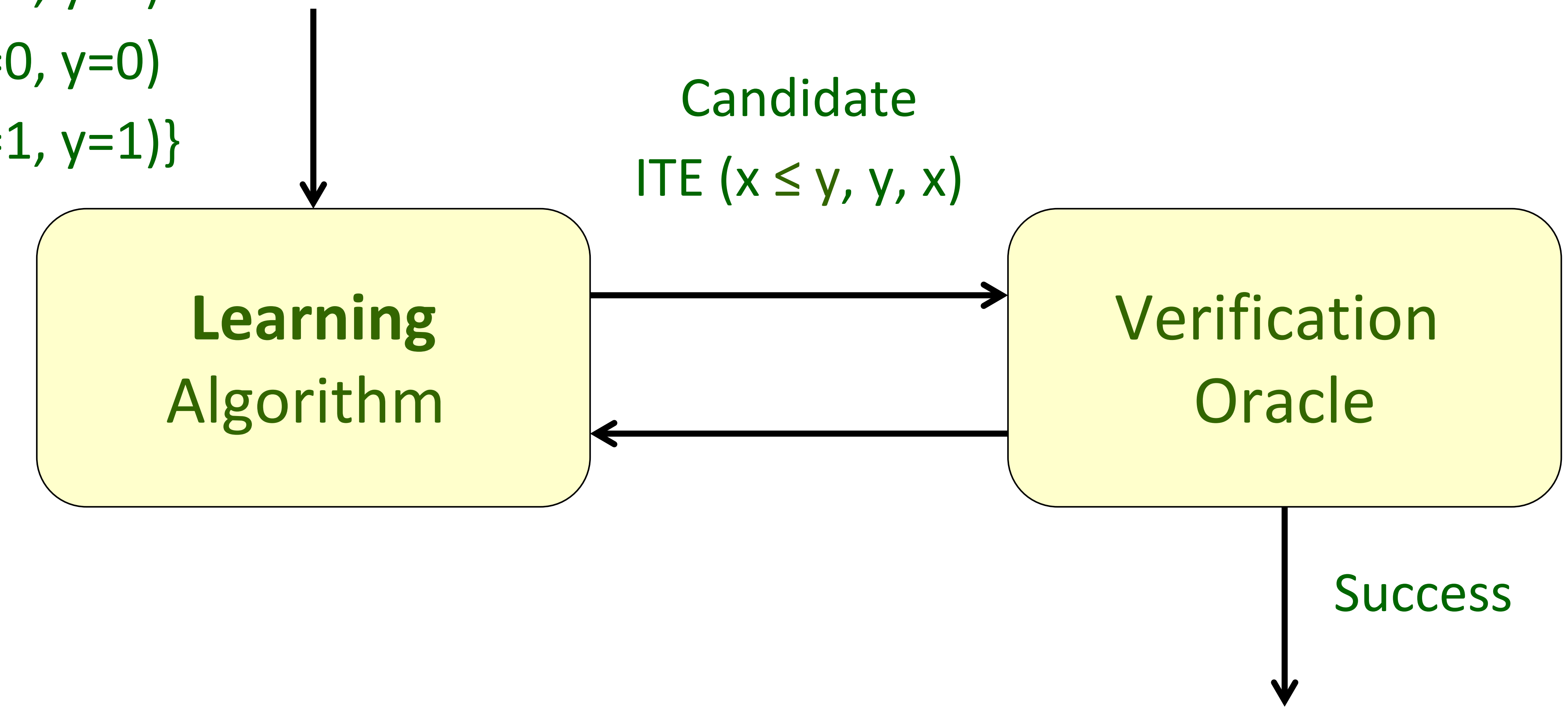
(slide adapted from one by R. Alur)

CEGIS Example

- Specification: (x ≤ f(x,y)) & (y ≤ f(x,y)) & (f(x,y) = x | f(x,y) = y)

- Set E: All expressions built from x,y,0,1, Comparison, +, If-Then-Else

Examples =
{(x=0, y=1)
(x=1, y=0)
(x=0, y=0)
(x=1, y=1)}



(slide adapted from one by R. Alur)

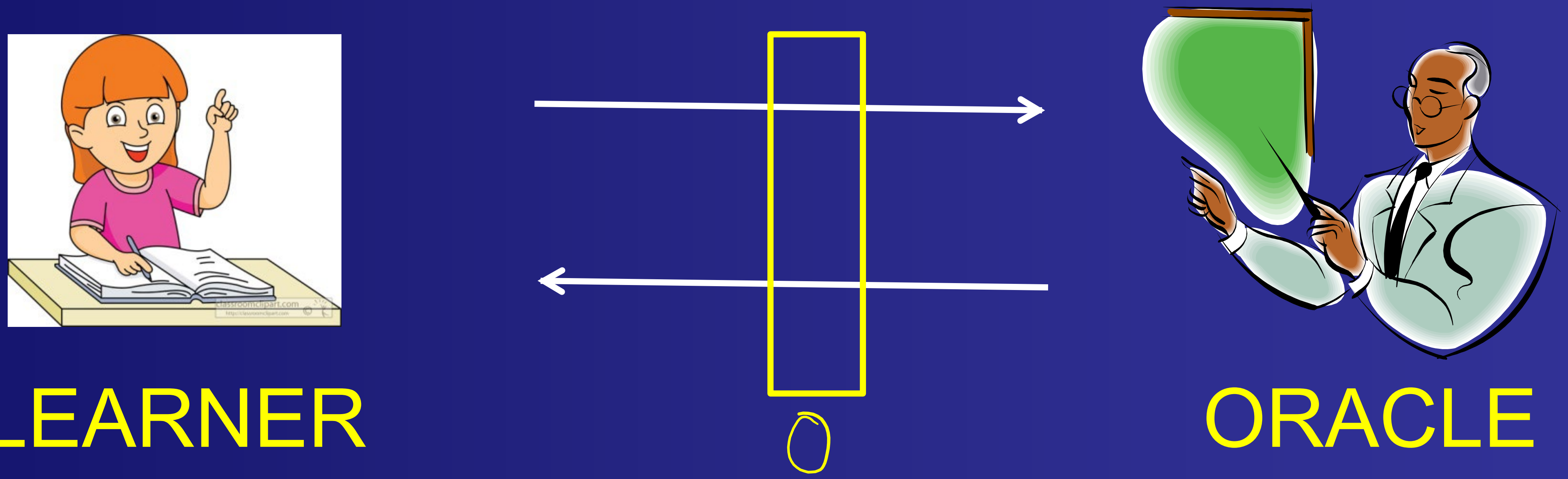
Formal Inductive Synthesis & Oracle-Guided Inductive Synthesis

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

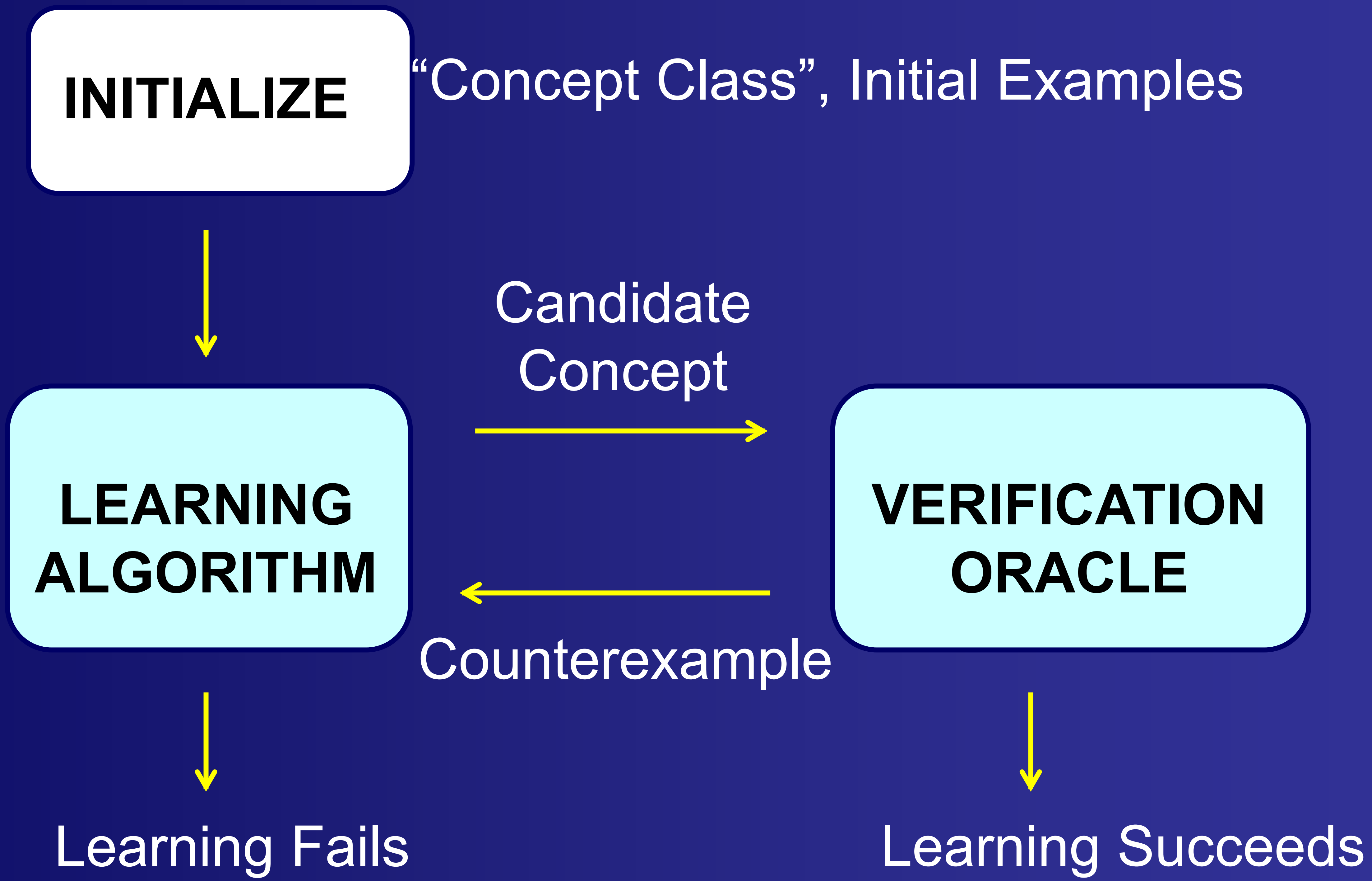
Oracle Interface

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

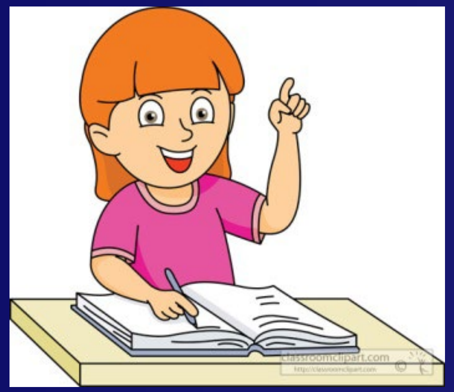


- Specifies WHAT the learner and oracle do
- Does *not* specify HOW the oracle/learner is implemented

CEGIS = Learning from Examples & Counterexamples



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. $\perp$

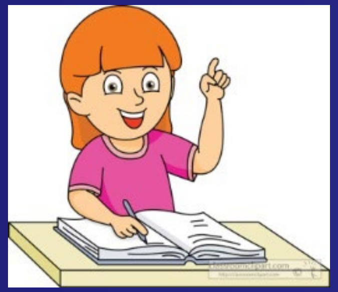
LEARNER

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

[See Jha & Seshia, Acta Informatica 2017 for details]

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** [Jin et al., HSCC 2013]
  - **Reactive Model Predictive Control** [Raman et al., 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]

More Examples

- CEGIS(T) [3]

Is my program “x + 5” correct?

No, but if you replace 5 with 1, it would be

- ICE learning [4]

Is my invariant “x > 5 ” correct?

No, and $inv(6) \implies inv(5)$

No, and $inv(100) = false$

No, and $inv(0) = true$

[3] Counterexample Guided Inductive Synthesis modulo Theories - Abate et al
[4] ICE: A robust framework for learning invariants - Garg et al

(slide due to E. Polgreen)

Satisfiability and Synthesis Modulo Oracles

(some slide material due to E. Polgreen)

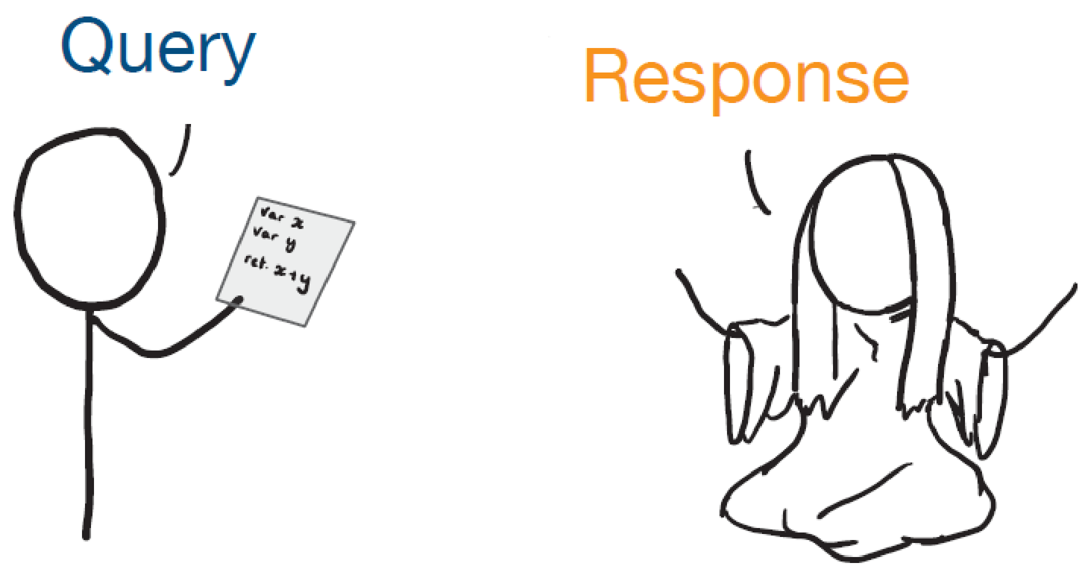
Logic Constraint Solvers → Oracle-Based Solvers

- Current SMT solvers require *all constraints to be encoded as logical formulas*
- Limiting for *complex* components, or those that may only be available as *executables* or via interaction with *humans*
- Our Contribution: [Polgreen et al., VMCAI'22]
  - Satisfiability Modulo Theories *and Oracles* (SMTO)
  - Synthesis Modulo *Oracles* (SMO)
  - Key idea: Oracle Interface expanded by oracle using "assumption generator" and "constraint generators"

$$I = \begin{cases} Q & : (y_1\, \sigma_1), \ldots, (y_j\, \sigma_j) \\ R & : (z_1\, \sigma'_1), \ldots, (z_k\, \sigma'_k) \\ \alpha_{gen} & : \textit{assumption generator} \\ \beta_{gen} & : \textit{constraint generator} \end{cases}$$

Formalized Oracle Interface

- We define how the oracle is queried by defining an interface



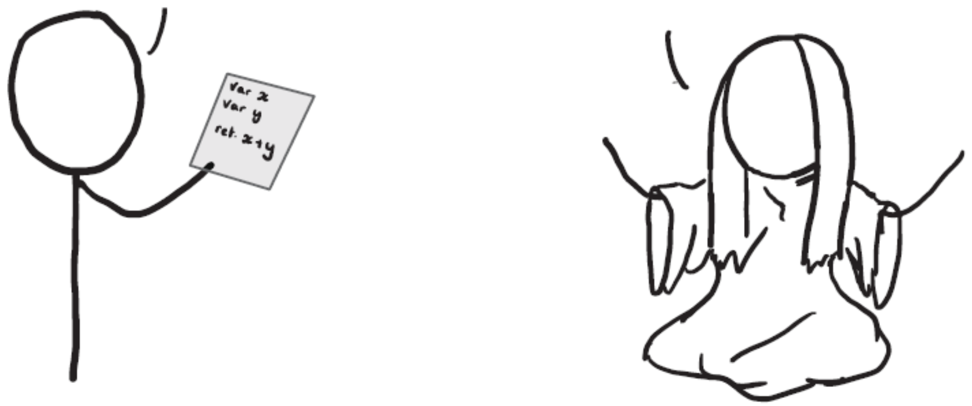
$\vec{y}$: query domain
$\vec{z}$: response co-domain
α_{gen} : assumption generator
β_{gen} : constraint generator

- and assumption and constraint generators, which generate:
  - assumptions the solver is allowed to make
  - and constraints the solver must abide by

Oracle Function Symbols

Is this number y prime?

No, z=false, it is not prime.



An oracle function symbol is a symbol whose behaviour is defined to be the same as an external oracle.

Note: oracle must be functional

$$\vec{y} : (y : integer)$$
$$\vec{z} : (z : bool)$$
$$\alpha_{gen} : prime(y) = z$$
$$\beta_{gen} : \emptyset$$

prime is an oracle function symbol

Satisfiability Modulo Theories and Oracles (SMTO)

An SMTO problem is a tuple:

- $\vec{f}$: a set of ordinary function symbols
- $\vec{\theta}$: a set of oracle function symbols
- ρ : a formula in a background theory
- $\vec{\mathcal{O}}$: a set of oracle interfaces

$\vec{f} : \{f_1, f_2\}$

$\vec{\theta} : \{prime\}$

$\rho : prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$

$\vec{\mathcal{O}} : \{\mathcal{O}_{prime}\}$

$\mathcal{O}_{prime}$

$\vec{y}$: $(y : integer)$

$\vec{z}$: $(z : bool)$

α_{gen} : $prime(y) = z$

β_{gen} : $\emptyset$

Is this satisfiable? What is a valid assignment to f_1 and f_2?

Satisfiability Modulo Theories and Oracles (SMTO)

SAT?

$$prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$$

$\mathcal{O}_{prime}$

$\vec{y}$: $(y : integer)$
$\vec{z}$: $(z : bool)$
α_{gen} : $prime(y) = z$
β_{gen} : $\varnothing$

Conjunction of assumptions. True if no assumptions

Satisfiable iff $\exists f_1, f_2 \,.\, \forall prime \,.\, A \implies \rho$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 \,.\, \exists prime \,.\, A \wedge \rho$ is unsatisfiable

Unknown otherwise

Restrict to *Definitional SMTO*

Satisfiability Modulo Theories and Oracles (SMTO)

SAT?

$$prime(f_1) \wedge prime(f_2) \wedge (f_1 * f_2 = 24)$$

$\mathcal{O}_{prime}$

$\overrightarrow{y}$: $(y : integer)$
$\vec{z}$: $(z_1 : bool, z_2 : integer)$
α_{gen} : $prime(y) = z$
β_{gen} : $f_1 < z_2$

Constraints must be obeyed by the solver:

Conjunction of constraints. True if no constraints.

Satisfiable iff $\exists f_1, f_2 . \forall prime . A \implies (\rho \wedge B)$ is satisfiable

Unsatisfiable iff $\exists f_1, f_2 . \exists prime . A \wedge \rho \wedge B$ is unsatisfiable

Unknown otherwise

Synthesis Modulo Oracles (SyMO)

A SyMO problem is a tuple:

$\vec{f}$: a tuple of functions to synthesise

$\vec{\theta}$: a set of oracle function symbols

$\forall \vec{x} \,.\, \phi$: a formula in a background theory, where ϕ is quantifier-free

$\vec{\mathcal{O}}$: a set of oracle interfaces

Generalizes SyGuS with richer oracle interfaces

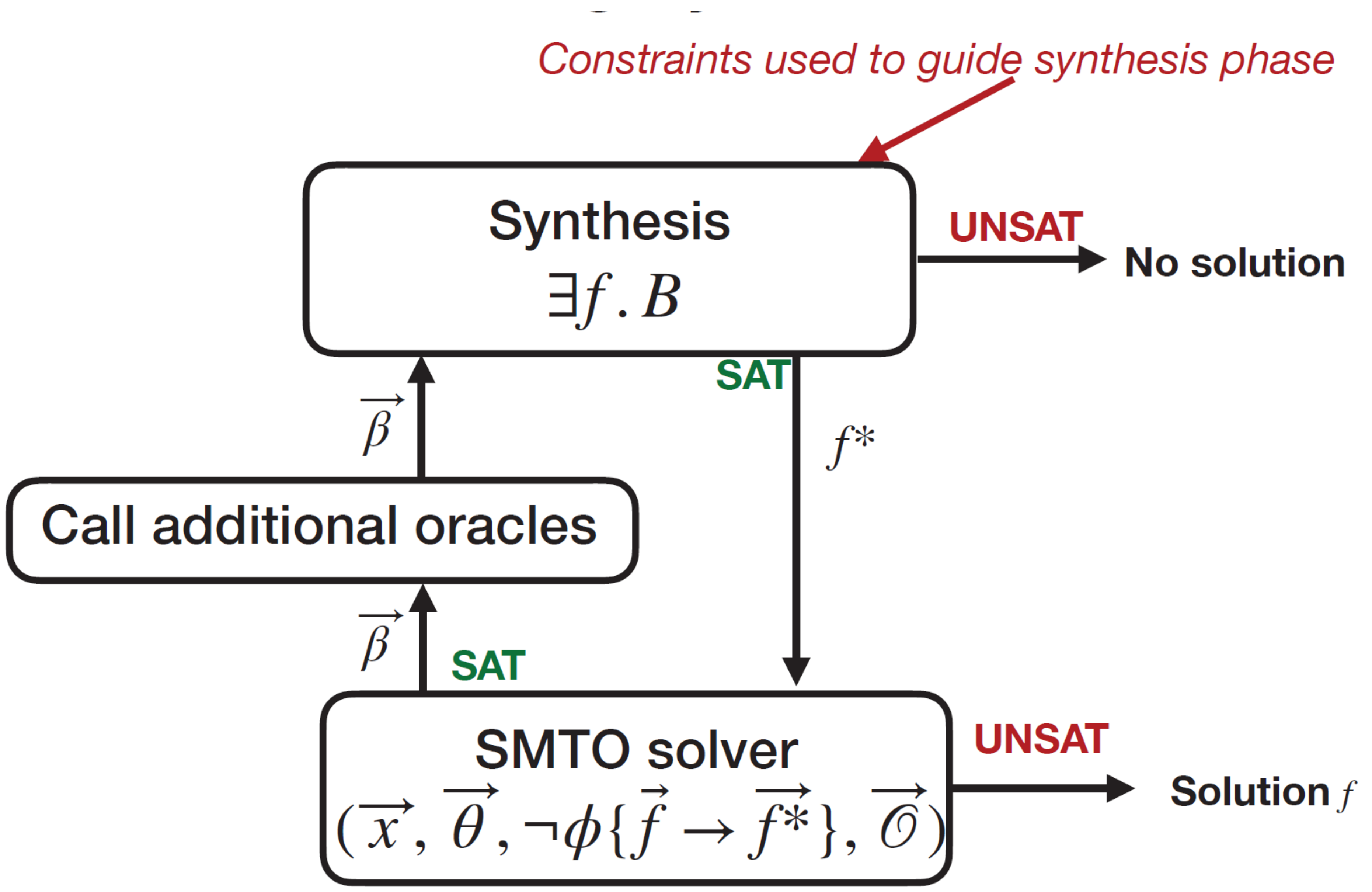
And:

- All **assumption** generators define **oracle function symbols**
- All oracles are **functional**

$\Longrightarrow$ checking $\vec{f}$ is valid is now **definitional SMTO**

Synthesis Modulo Oracles (SyMO)

- Synthesis solver calls SMTO solver to check correctness of the synthesized functions
- It can additionally invoke other oracles to guide the search
  - E.g. answering membership queries, provide labeled examples, demonstrations, preferences, etc.



Some Experimental Results with SyMO/SMTO

Approximate model

	Problem		Delphi (oracles)		CVC5 (no oracles)	
		#	#	s	#	s
SyMO	Images	10	9	21.6s	0	
	Control stability	112	104	29.3s	16	19.4s
	Control safety	112	31	59.9s	0	
	PBE	150	148	0.5s	150	<0.5s
SMTO	Math	12	9	<0.5s	5	2.2s

Oracle-Guided Reasoning with UCLID5

Latest version of UCLID5 has support for Satisfiability and Synthesis Modulo Oracles

Used it for several tasks including algorithmically synthesizing a stabilizing controller

```
module main {
  var x0, x1: float;
  group states : float = {x0, x1};
  <..LTI system spec vars decls..>
  oracle function [isstable] isStable
    (s00:float, s01:float, s10:float, s11:float) : boolean;
  synthesis function k0 (): float;
  synthesis function k1 (): float;

  // LTI system spec values
  axiom A:(a00==0.901224922471 && a01==0.000000013429 && a10==0.000000007451 &&
     a11==0.0);
  axiom B: (b0==128.0 && b1==0.0);
  axiom ax1: ABK00 == a00 - b0*k0());
  <...>
  axiom ax4: ABK11 == a11 - b1*k1());

  init { // bound initial states
    assume (finite_forall (s: float) in states :: s<0.1 && s>-0.1);
  }
  next { // step the system
    x0' = ABK00*x0 + ABK01*x1;
    x1' = ABK10*x0 + ABK11*x1;
  }
  // the safety condition
  invariant stability: isStable(ABK00, ABK01, ABK10, ABK11);
  invariant safety: finite_forall (s: float) in states :: s < 1.0&&s > -1.0;

  control {
    unroll(10); // fix safety bound
    check;
  }
}
```

Summary

- Formal Synthesis
- Verification by Reduction to Synthesis
- Syntax-Guided Synthesis
- Formal Inductive Synthesis
  - Counterexample-guided inductive synthesis (CEGIS)
  - General framework for solution methods: Oracle-Guided Inductive Synthesis (OGIS)
  - Theoretical analysis (see Jha & Seshia, 2017)
- Satisfiability and Synthesis Modulo Oracles
  - A generic approach to solve OGIS problems
- Lots of potential for future work!