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

NSF ExCAPE Summer School
June 23-25, 2015
Formal Synthesis

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

- **Find** \( f \in C \) that satisfies \( \phi \)

- **Example:**
  - \( C \): all affine functions \( f \) of \( x \in \mathbb{R} \)
  - \( \phi \): \( \forall x. f(x) \geq x + 42 \)
Reactive Synthesis: An Example of Formal Synthesis

- **Given:**
  - Class C: all finite-state transducers with input alphabet I and output alphabet O
  - Linear temporal logic (LTL) Specification $\phi$

- Find transducer in C that satisfies $\phi$
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 $x \in \mathbb{R}$
  - $E = \{(0, 42), (1, 43), (2, 45)\}$
  - $\Psi$ -- 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 $\Psi$?
"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)}
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

- Given:
  - Class of Artifacts C
  - Formal specification $\phi$
  - Set of (labeled) examples $E$ (or source of E)

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

- Example:
  - C: all affine functions $f$ of $x \in \mathbb{R}$
  - $E = \{(0,42), (1, 43), (2, 44)\}$
  - $\phi: \forall x. f(x) \geq x + 42$
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”
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?
- How does inductive synthesis compare with (traditional) machine learning?
- What are the common themes amongst various inductive synthesis efforts?
- Is there a complexity/computability theory for 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

- How does inductive synthesis compare with machine learning? What are the common themes amongst various inductive synthesis efforts?
  - Oracle-Guided Inductive Synthesis (OGIS) Framework

- Is there a complexity/computability theory for inductive synthesis?
  - Yes! A first step: Theoretical analysis of counterexample-guided inductive synthesis (CEGIS)
Outline for this Lecture Sequence

- 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

- Demo: Requirement Mining for Cyber-Physical Systems
Further Reading

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

  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
- ...

Example Verification Problem

- Transition System
  - Init: I
    
    \[ x = 1 \land y = 1 \]
  - Transition Relation: \( \delta \)
    
    \[ x' = x + y \land y' = y + x \]

- Property: \( \Psi = G (y \geq 1) \)

- 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 \( \phi \) s.t.
  
  \[ x = 1 \land y = 1 \Rightarrow \phi \]
  
  \[ \phi \land y \geq 1 \land x' = x + y \land y' = y + x \Rightarrow \phi' \land y' \geq 1 \]
Example Verification Problem

- **Transition System**
  - **Init:** \( I \)
    \[
    x = 1 \land y = 1
    \]
  - **Transition Relation:** \( \delta \)
    \[
    x' = x+y \land y' = y+x
    \]

- **Property:** \( \Psi = G (y \geq 1) \)

- **Attempted Proof by Induction:**
  \[
  y \geq 1 \land x' = x+y \land y' = y+x \implies y' \geq 1
  \]

  - Fails. Need to Strengthen Invariant: Find \( \phi \) s.t.
    \[
    x \geq 1 \land y \geq 1 \land x' = x+y \land y' = y+x \implies x' \geq 1 \land y' \geq 1
    \]

- **Safety Verification \( \rightarrow \) Invariant Synthesis**
One Reduction from Verification to Synthesis

NOTATION
Transition system $M = (I, \delta)$
Safety property $\Psi = G(\psi)$

VERIFICATION PROBLEM
Does $M$ satisfy $\Psi$?

SYNTHESIS PROBLEM
Synthesize $\phi$ s.t.

$\begin{align*}
I & \Rightarrow \phi \land \psi \\
\phi \land \psi \land \delta & \Rightarrow \phi' \land \psi'
\end{align*}$
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.
\[ I \Rightarrow \phi \land \psi \]
\[ \phi \land \psi \land \delta \Rightarrow \phi' \land \psi' \]

SYNTHESIS PROBLEM #2
Synthesize \( \alpha : S \to \hat{S} \) where
\[ \alpha(M) = (\hat{I}, \hat{\delta}) \]
s.t.
\[ \alpha(M) \text{ satisfies } \Psi \]
iff
\[ M \text{ satisfies } \Psi \]
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)

1. **INITIALIZE**
   - Structure Hypothesis ("Syntax-Guidance"), Initial Examples

2. **SYNTHESIZE**
   - Candidate Artifact
   - Counterexample
   - Synthesis Fails

3. **VERIFY**
   - Verification Succeeds
SyGuS vs. CEGIS

- SyGuS --- problem classes
- CEGIS --- solution classes
Lazy SMT Solving performs Inductive Synthesis (of Lemmas)

**SYNTHESIS**
- SMT Formula
- Initial Boolean Abstraction
- Generate SAT Formula
- Blocking Clause/Lemma
- Proof Analysis
- SAT Formula
- “Spurious Model”

**VERIFICATION**
- Invoke SAT Solver
- SAT (model)
  - (“Counter-example”)
- Invoke Theory Solver
- SAT
  - Done
- UNSAT
  - Done
Other Examples

- Invariant Generation via ICE Learning [P. Garg & M. Parthasarathy]

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, …
Two Award-Winning Dissertations, circa 2000

- **Michael Ernst**: “Dynamically Discovering Likely Program Invariants”
  - Learning (likely) Invariants from Tests

- **William Chan**: “Symbolic Model Checking for Large Software Specifications”
  - “Temporal logic queries”
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?
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. $G(\_ \Rightarrow X \_)$
Specification Mining

- Inductive Synthesis of Specifications


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 (Lecture 2)

- 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 $\rightarrow$ more easily satisfied
  - Non-negative satisfaction value $\equiv$ Boolean satisfaction

- Example: "For all time points between 60 and 100, the absolute value of $x$ is below 0.1"

$$\Box_{[60,100]}(|x| < 0.1)$$
CounterExample Guided Inductive Synthesis

[Jin, Donze, Deshmukh, Seshia, HSCC 2013]
CounterExample Guided Inductive Synthesis

Experimental Engine Control Model

Find “Tightest” Properties

Settling Time is ??
Overshoot is ??
Upper Bound on x is ??

1. m
2.

Counterexamples

Settling Time is ... ms
Overshoot is ... KPa
Upper Bound on x is ...

Are there behaviors that do NOT satisfy these requirements?
CounterExample Guided Inductive Synthesis

Experimental Engine Control Model

Find "Tightest" Properties

Settling Time is ??
Overshoot is ??
Upper Bound on x is ??

Counterexamples

Are there behaviors that do NOT satisfy these requirements?

Mined Requirement

Settling Time is 6.3 ms
Overshoot is 5.6 KPa
Upper Bound on x is 4.1

NO

Experimental Engine Control Model
Experimental Results on Industrial Airpath Controller

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

![Pressure diff. vs Time](image-url)
Mining can expose deep bugs

- 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
Summary of Part 1

- Basic Terminology
  - Formal Synthesis
  - Inductive Synthesis
  - Formal Inductive Synthesis
  - Notions from Machine Learning

- Reductions to Synthesis
  - Verification artifacts
  - Specifications
  - Case Study: requirement mining for closed-loop control systems ➔ Demo on Thu morning