Synthesis and Inductive Learning – Part 2

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

NSF ExCAPE Summer School
June 23-25, 2015
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
Comparison with Machine Learning
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 $\phi$
Counterexample-Guided Inductive Synthesis (CEGIS)

- **INITIALIZE**
  - Structure Hypothesis (“Syntax-Guidance”), Initial Examples

- **SYNTHESIZE**
  - Candidate Artifact
  - Synthesis Fails

- **VERIFY**
  - Verification Succeeds
  - Counterexample
CEGIS = Learning from Examples & Counterexamples

- **INITIALIZE**
  - “Concept Class”, Initial Examples

- **LEARNING ALGORITHM**
  - Candidate Concept
  - Learning Fails

- **VERIFICATION ORACLE**
  - Counterexample
  - Learning Succeeds
CEGIS is an instance of Active Learning

1. **Search Strategy**: How to search the space of candidate concepts?
2. **Example Selection**: Which examples to learn from?
Some Instances of CEGIS you’ve seen (will see soon) (see [Alur et al., FMCAD’13])

<table>
<thead>
<tr>
<th>Instance</th>
<th>Concept Class</th>
<th>Learner</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKETCH</td>
<td>Programs in SKETCH</td>
<td>SAT/SMT solver</td>
<td>SAT/SMT solver</td>
</tr>
<tr>
<td>Enumerative SyGuS</td>
<td>Defined by Grammar</td>
<td>Enumerative Search</td>
<td>SMT solver</td>
</tr>
<tr>
<td>Stochastic SyGuS</td>
<td>Defined by Grammar</td>
<td>Stochastic Search</td>
<td>SMT solver</td>
</tr>
<tr>
<td>Constraint SyGuS</td>
<td>Defined by Grammar</td>
<td>SMT solver</td>
<td>SMT solver</td>
</tr>
<tr>
<td>Req. Mining for STL</td>
<td>Parametric STL</td>
<td>Parameter Search</td>
<td>Simulation-based falsifier</td>
</tr>
</tbody>
</table>
# Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Formal Inductive Synthesis</th>
<th>Machine Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept/Program Classes</td>
<td>Programmable, Complex</td>
<td>Fixed, Simple</td>
</tr>
<tr>
<td>Learning Algorithms</td>
<td>General-Purpose Solvers</td>
<td>Specialized</td>
</tr>
<tr>
<td>Learning Criteria</td>
<td>Exact, w/ Formal Spec</td>
<td>Approximate, w/ Cost Function</td>
</tr>
<tr>
<td>Oracle-Guidance</td>
<td>Common (can control Oracle)</td>
<td>Rare (black-box oracles)</td>
</tr>
</tbody>
</table>

* Between typical inductive synthesizer and machine learning algo
Oracle-Guided Inductive Synthesis
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 $\phi$**
Common Oracle Query Types

Positive Witness
\[ x \in \phi, \text{if one exists, else } \perp \]

Negative Witness
\[ x \notin \phi, \text{if one exists, else } \perp \]

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: \( f, X \subseteq f \)
\[ f' \text{ s.t. } f' \neq f \land X \subseteq f', \text{if it exists; o.w. } \perp \]
Examples of OGIS

- L* algorithm to learn DFAs:
  - Membership + Equivalence queries

- CEGIS used in SKETCH/SyGuS solvers
  - (positive) Witness + Equivalence/Subsumption queries

- CEGIS used in Reactive Model Predictive Control
  - covered in Vasu Raman’s lecture

- 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]
Reverse Engineering Malware by Program Synthesis

Obfuscated code:
Input: y  Output: modified value of y

```c
{ a=1; b=0; z=1; c=0;
  while(1) {
    if (a == 0) {
      if (b == 0) { y=z+y; a =~a; b=~b; c=~c; if (~c) break; } 
      else { z=z+y; a=~a; b=~b; c=~c; if (~c) break; } 
    }
    else if (b == 0) { z=y << 2; a=~a; }
    else { z=y << 3; a=~a; b=~b; }
  } }
```

What it does:
y = y * 45

We solve this using program synthesis.

FROM CONFICKER WORM

Class of Programs: “Loop-Free”

- Programs implementing functions: $I \rightarrow O$

\[ P(I): \]
\[ O_1 = f_1 (V_1) \]
\[ O_2 = f_2 (V_2) \]
\[ \ldots \]
\[ O_n = f_n (V_n) \]

where

$f_1, f_2, \ldots, f_n$ are functions from a given component library

Functions could be **if-then-else** definitions and hence, the above represents any loop-free code.
Program Learning as Set Cover

[Goldman & Kearns, 1995]

Space of all possible programs
Each dot represents semantically unique program
Program Learning as Set Cover

Space of all possible programs
Each dot represents
semantically unique program
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

\[(i_1, o_1)\]

Example $\equiv$ Set of programs \textit{ruled out} by that example
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

(i₁, o₁) - E₁
(i₂, o₂) - E₂
.........
(iₙ, oₙ) - Eₙ

Theorem: [Goldman & Kearns, ’95]
Smallest set of I/O examples to learn correct program

IS

Minimum size subset of {E₁, E₂, ......., Eₙ} that covers all the incorrect programs
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

(i₁, o₁) - E₁
(i₂, o₂) - E₂

........

(iₙ, oₙ) - Eₙ

Smallest set of I/O examples to learn correct design

Practical challenge: can’t enumerate all inputs and find set Eᵢ for each

Minimum size subset of {E₁, E₂, ...., Eₙ} that cover all the incorrect programs

IS
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

(i_1, o_1) - E_1
(i_2, o_2) - E_2
........
(i_n, o_n) - E_n

ONLINE set-cover:

In each step,
• choose some (i_j,o_j) pair
• eliminated incorrect programs E_j disclosed
Program Learning as Set Cover

Space of all possible programs
Each dot represents semantically unique program

(i₁, o₁) - E₁
(i₂, o₂) - E₂
........
(iₙ, oₙ) - Eₙ

Our heuristic:
|Eⱼ| ≥ 1: at least one incorrect program identified

ONLINE set-cover:
In each step,
• choose some (iⱼ,oⱼ) pair
• eliminated incorrect programs Eⱼ disclosed
Approach: Learning based on Distinguishing Inputs

Space of all possible programs
Each dot represents semantically unique program

[Jha et al., ICSE’10]
Learning from Distinguishing Inputs

Make (positive) Witness query
Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs
Learning from Distinguishing Inputs

Example I/O set $E := \{(i_1,o_1)\}$

Learner synthesizes $P_1$ using SMT solver

Space of all possible programs
Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs

Learning from Distinguishing Inputs
Learning from Distinguishing Inputs

Example I/O set $E := \{(i_1, o_1)\}$

Space of all possible programs
Learning from Distinguishing Inputs

Example I/O set $E := \{(i_1, o_1)\}$

Implemented as a single Distinguishing input query:
- Given $P_1$,
- Finds $P_2$ s.t. $P_1 \neq P_2$ and $(i_2, o_2)$ is dist. input
Example I/O set $E := E \cup \{(i_2, o_2)\}$

Space of all possible programs
Example I/O set $E := E \cup \{(i_j, o_j)\}$

Space of all possible programs
Example I/O set $E := E \cup \{(i_k, o_k)\}$

Space of all possible programs
Learning from Distinguishing Inputs

Example I/O set $E := E \cup \{(i_n, o_n)\}$

Space of all possible programs

Correct Program?
Soundness

Conditional on Validity of Structure Hypothesis

- Library of components is sufficient?
  - YES
    - Correct design
  - NO
    - I/O pairs show infeasibility?
      - YES
        - Infeasibility reported
      - NO
        - Incorrect design

Can put this learning approach within an outer CEGIS loop
Example 2: Assumption Synthesis
Reactive Synthesis from LTL

Often due to incomplete environment assumptions!
Example

• Inputs: request $r$ and cancel $c$
• Outputs: grant $g$
• System specification $\varphi_s$:
  - $G (r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$
• Environment assumption $\varphi_e$:
  - True
• Is it realizable?
• Not realizable because the environment can force $c$ to be high all the time
Counter-strategy and counter-trace

- Counter-strategy is a strategy for the environment to force violation of the specification.
- Counter-trace is a fixed input sequence such that the specification is violated regardless of the outputs generated by the system.

- System $\varphi_s$:
  - $G (r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$

- A counter-trace:
  - $r$: 1 1 (1)
  - $c$: 1 1 (1)
CounterStrategy-Guided Environment Assumption Synthesis [Li et al., MEMOCODE 2011]

Diagram:
- Start
- Formal Specification
  - Add
  - Mine Assumptions
- Synthesis Tool
  - Unrealizable
  - Compute Counterstrategy
- Realizable
- Done
Instance of CEGIS

- Concept Class: all LTL formulas over I and O “constraining I”
- Domain of Examples: All finite-state transducers with input O and output I (environments)
- Formal Spec: set $\phi$ of env transducers for which there exists an implementation of Sys satisfying $\phi_s$
- Oracle supports subsumption query:
  - Does the current assumption rule out environments outside $\phi$? If not, give one such counterexample.
Assumption Learning Algorithm

Specification Templates

User Scenarios

Counter-strategy

generate Candidate $\phi$
e.g. $G F$?

Eliminate or Accept Candidate

Eliminate $\phi$

Accept $\phi$

Done

(check consistency with existing assumptions)
Assumption Synthesis Algorithm

**Specification Templates**

- Example: $G \bigwedge F$

**User Scenarios**

**Counter-strategy $E$**

**Generate Candidate $\phi$**

- Example: $G \bigwedge F p$

**Invoke Model Checker:** Does $E$ satisfy $\neg \phi$?

**VERSION SPACES:**
- Retain candidates consistent with examples
- Traverse weakest to strongest

- **NO:** Eliminate $\phi$
- **YES:** Accept $\phi$
  (check consistency with existing assumptions)

**Done**
Version Space Learning

Originally due to [Mitchell, 1978]

STRONGEST (most specific)

WEAKEST (most general)
Example

- System $\Phi_s$:
  - $G (r \rightarrow X F g)$
  - $G(c \lor g \rightarrow X \neg g)$

- A counter-trace:
  - $r$: 1 1 (1)
  - $c$: 1 1 (1)

- Test assumption candidates by checking its negation:
  - $G (F c)$
  - $G (F \neg c)$

- Realizable!
Theoretical Results

- **Theorem 1: [Completeness]** If there exist environment assumptions under our structure hypothesis that make the spec realizable, then the procedure finds them (terminates successfully).
  - “conditional completeness” guarantee

- **Theorem 2: [Soundness]** The procedure never adds inconsistent environment assumptions.
Summary of Lecture 2

- Differences between Formal Inductive Synthesis and Machine Learning
- Common features across various inductive synthesis methods
- Oracle-Guided Inductive Synthesis framework
- Two Instances
  - Learning Programs from Distinguishing Inputs
  - Learning Assumptions from Counterstrategies
- Brief Introduction to the Theory (more tomorrow)