Inductive learning with SMT Solvers and Beyond

Armando Solar-Lezama
Framing the problem

Appears simple: learn from examples

- So what is an example?

Functional Program
- Input Output Pair

Reactive Program
- Input/Output Trace?

Concurrent Program
- Trace?
Framing the problem

Additional Concerns when defining Examples

- Partial vs. Complete Examples
- Positive vs. Negative Examples
- Noisy vs. Sanitized Examples
Framing the problem

Interaction model: Active vs. Passive learning

Passive Learning
User provides examples as they come to mind

Active Learning
Machine drives the example selection process
Basic approach

1) Chose a parameterized family of possible solutions
   - This is standard in machine learning too

2) Search the space of parameters for good instances
   - instances that match the examples

Key questions
   - How to represent the search space
   - How to eliminate more than one candidate at a time
Inductive Synthesis with SAT/SMT

Given:
- a parameterized program $P[c],$
- a set of input output pairs $(in_i, out_i)$

Set of potentially valid candidates represented as a constraint $Q(c)$.
- Initially it is the universal set $Q_0(c) = true$
- Every example introduces new constraints $Q_i(c) = Q_{i-1}(c) \land P[c](in_i) = out_i$
Inductive Synthesis with SAT/SMT

\[ P[c](in_1) = out_1 \]

\[ P[c](in_0) = out_0 \]

\[ P[c](in_2) = out_2 \]

\[ P[c](in_3) = out_3 \]
Active learning in Classification

Given: Set of labeled examples and a learned boundary

Idea: Sample closest to the boundary is the best
Active learning with SAT/SMT

Problem: Boolean predicates don’t offer good metric
  - How do you know which solution is “closest to boundary”

Solution: Generate many predicates and vote
  \[ P_1(in) + P_2(in) + P_3(in) + P_4(in) > 0 \]
  - “Closest to the boundary” means maximum disagreement

Simpler solution (by Jha, Gulwani, Seshia and Twari):
  - Find two predicates that disagree on an input, use that input
  - Generalizes easily to functions
Other approaches

Version spaces

- Factor search space $S = S_1 \times S_2 \times S_3$
- When a solution $(s_1, s_2, s_3)$ fails, identify which factors contributed to failure, eliminate an entire subspace of solutions.

- Very efficient for many domains
Synthesis with abstract examples

\[
\begin{align*}
\text{next} & \quad \text{next} & \quad \text{next} & \quad \text{next} \\
\text{prev} & \quad \text{prev} & \quad \text{prev} & \quad \text{prev}
\end{align*}
\]
A scenario is more powerful than an example
- each scenario represents an infinite set of examples

It is also less powerful
- The picture is ambiguous regarding the fate of ...
Challenge

Scenario is really a quantified constraint

∀ inputs

\[
\text{input satisfies input frame} \implies \text{output satisfies output frame}
\]

Many tricks are required for this to solve efficiently
Beyond user provided examples

Learning from sample code

- Another mechanism to acquire domain knowledge

Programming by imitation

- A lot of programming is done by “Monkey see Monkey do” rule
- With proper guidance, machines can do this better