A Data-Oriented Approach to Program Synthesis

Fritz Obermeyer

2016-04-06

Overview

Motivation:

Given a curated knowledge base,
fill holes in programs subject to tests.

Machine Learning Methods:

Tune the knowledge base's language.
Find common phrases to add to language.
Search for plausible conjectures.

Synthesize programs by sketching

Solar-Lezama (2008)

Fill holes in a program
with terms of a specified language
subject to constraints
modulo simplification.

Synthesize programs by sketching

Solar-Lezama (2008)

Fill holes in a program
with terms of a specified language
subject to constraints
modulo simplification.

How?
Best-first search
modulo simplification
filtered by constraint propagation.

Example program with one hole

$\text{conj} = \lambda s.\ \lambda r.\ \lambda f.\ \text{COMP}\ r\ \text{COMP}\ f\ s$
$\text{conj} = \lambda s.\ \lambda r.\ \lambda f.\ \text{COMP}\ \text{COMP}\ r\ f\ s$
$\text{raise} = \lambda x.\ \lambda y.\ x$
$\text{lower} = \lambda x.\ \text{APP}\ x\ \top$
$\text{pull} = \lambda x.\ \lambda y.\ \text{JOIN}\ x\ \text{APP}\ \text{DIV}\ y$
$\text{push} = \lambda x.\ \text{APP}\ x\ \bot$

$\text{move1} = \langle \text{I},\ \text{I} \rangle$
$\text{move2} = \langle \text{push},\ \text{pull} \rangle$
$\text{move3} = \langle \text{lower},\ \text{raise} \rangle$
$\text{move4} = \Lambda r1, s1.\ \Lambda r2, s2.\ \langle \text{COMP}\ r2\ r1,\ \text{COMP}\ s1\ s2 \rangle$
$\text{move5} = \Lambda r1, s1.\ \Lambda r2, s2.\ \langle \text{APP}\ \text{APP}\ \text{conj}\ s1\ r2,\ \text{APP}\ \text{APP}\ \text{conj}\ r1\ s2 \rangle$

$\text{hole} \sqsubseteq A$
$\text{APP}\ \text{hole}\ B \sqsubseteq I$
$\text{APP}\ \text{hole}\ B = I$
$\text{APP}\ \text{hole}\ CB \not\sqsubseteq I$

$\text{hole} \not\sqsubseteq \text{move1}$	$\text{move1} \sqsubseteq A$
$\text{hole} \not\sqsubseteq \text{move2}$	$\text{move2} \sqsubseteq A$
$\text{hole} \not\sqsubseteq \text{move3}$	$\text{move3} \sqsubseteq A$
$\text{hole} \not\sqsubseteq \text{move4}$	$\text{move4} \sqsubseteq A$
$\text{hole} \not\sqsubseteq \text{move5}$	$\text{move5} \sqsubseteq A$

Example language

$$
\begin{aligned}
&\text{language} = \{\\
&\quad 'APP' :\ 1.0,\\
&\quad \textit{// 'COMP': 1.6,}\\
&\quad 'JOIN' :\ 3.0,\\
&\quad 'B' :\ 1.0,\\
&\quad 'C' :\ 1.3,\\
&\quad 'A' :\ 2.0,\\
&\quad '\bot' :\ 2.0,\\
&\quad '\top' :\ 2.0,\\
&\quad 'I' :\ 2.2,\\
&\quad \textit{// 'Y': 2.3,}\\
&\quad 'K' :\ 2.6,\\
&\quad 'S' :\ 2.7,\\
&\quad 'J' :\ 2.8,\\
&\quad 'DIV' :\ 3.0,\\
&\}
\end{aligned}
$$

Approximate with an abstract interpretation

Given a set P of $\sim$ 30K probe terms, define a 4-tuple of sets

$$
\begin{aligned}
&[\![x]\!] = \{ \\
&\quad \text{above}: \{p \in P \mid x \sqsubseteq p\}, \\
&\quad \text{below}: \{p \in P \mid p \sqsubseteq x\}, \\
&\quad \text{nabove}: \{p \in P \mid x \not\sqsubseteq p\}, \\
&\quad \text{nbelow}: \{p \in P \mid p \not\sqsubseteq x\} \\
&\}
\end{aligned}
$$

Approximate with an abstract interpretation

Given a set P of $\sim$ 30K probe terms, define a 4-tuple of sets

$$
\begin{array}{l}
[\![\mathrm{x}]\!] = \{ \\
\quad \mathrm{above}: \{\mathrm{p} \in \mathrm{P} \mid \mathrm{x} \sqsubseteq \mathrm{p}\}, \\
\quad \mathrm{below}: \{\mathrm{p} \in \mathrm{P} \mid \mathrm{p} \sqsubseteq \mathrm{x}\}, \\
\quad \mathrm{nabove}: \{\mathrm{p} \in \mathrm{P} \mid \mathrm{x} \not\sqsubseteq \mathrm{p}\}, \\
\quad \mathrm{nbelow}: \{\mathrm{p} \in \mathrm{P} \mid \mathrm{p} \not\sqsubseteq \mathrm{x}\} \\
\}
\end{array}
$$

Growing P leads to more accurate approximations $[\![\mathrm{x}]\!]$.

Verify by constraint propagation

Constraints are simple predicates: $=, \quad \sqsubseteq, \quad \not\sqsubseteq$.
This language is Π^2_0-complete.

Verify by constraint propagation

Constraints are simple predicates: $=, \quad \sqsubseteq, \quad \not\sqsubseteq$.
This language is Π^2_0-complete.

$[\![x \sqsubseteq y]\!]$ is satisfied iff
$\text{below}(x) \;\cap\; \text{nbelow}(y) = \emptyset = \text{nabove}(x) \;\cap\; \text{above}(y)$.

Verify by constraint propagation

Constraints are simple predicates: $=, \quad \sqsubseteq, \quad \not\sqsubseteq$.
This language is Π^2_0-complete.

$[\![x \sqsubseteq y]\!]$ is satisfied iff
$\text{below}(x) \ \cap \ \text{nbelow}(y) = \emptyset = \text{nabove}(x) \ \cap \ \text{above}(y)$.

$[\![\text{APP} \;\; x \;\; y = z]\!]$ is a 3-way propagation node

$$\frac{x \ \sqsubseteq \ x' \qquad y \ \sqsubseteq \ y'}{z \ \sqsubseteq \ \text{APP} \ x' \ y'} \qquad \frac{x' \ \sqsubseteq \ x \qquad y' \ \sqsubseteq \ y}{\text{APP} \ x' \ y' \ \sqsubseteq \ z}$$

etc.

And focus on data

1. Invest in a curated knowledge base.

2. Leverage the knowledge base in synthesis.

Invest in a Knowledge Base

$\sim$ 1 engineer year, $\sim$ 10 CPU year,
$\sim$ 1GB compressed, $\sim$ 10GB in memory.

Invest in a Knowledge Base

$\sim$ 1 engineer year, $\sim$ 10 CPU year,
$\sim$ 1GB compressed, $\sim$ 10GB in memory.

Automatically deduce many small facts.

Todd-Coxeter forward-chaining enumeration of $\sim$ 30K equivalence classes.

VM-based with an inference rule compiler, with $\sim$ 100 inference rules.

ML Method 1: Tune the language

Fit a probabilistic grammar to a corpus.

Why?

- to learn facts about *relevant* terms
- to synthesize from *popular* components

How?

EM or Empirical Bayes

ML Method 2: Find common phrases

Suggest new atoms for the grammar.

Why?
"SK is a bad basis"

How?
Greedily suggest APP pairs from among $\sim$ 1M.

ML Method 3: Find conjectures [1/3]

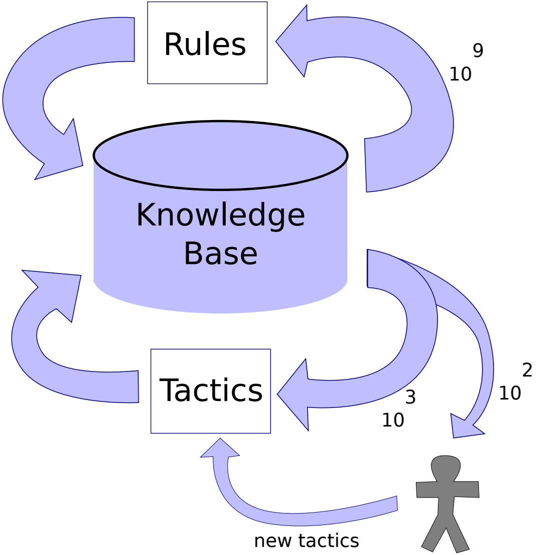
Search for plausible conjectures.

Among millions of statements

$x = y, \quad x \sqsubseteq y, \quad x \not\sqsubseteq y$

- Find 1000 to apply expensive tactics to
- Find 100 to send to a human

ML Method 3: Find conjectures [2/3]



ML Method 3: Find conjectures [3/3]

How?
Quantify evidence based on extensionality

$$\frac{}{\top \not\sqsubseteq \bot} \qquad \frac{f\ x \not\sqsubseteq f\ y}{x \not\sqsubseteq y}$$

Questions?

Implemented at
`http://github.com/fritzo/pomagma`

Language

Combinatory algebra is a simple language:
$\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{AP}(-, -)$.

There are no types, no variables, no binding.

Extensionally collapse to reduce search space.

Add nondeterminism to quotient out implementation.

Implement types-as-closures via nondeterminism.