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


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


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

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

\[ \text{move1} = \langle I, I \rangle \]
\[ \text{move2} = \langle \text{push}, \text{pull} \rangle \]
\[ \text{move3} = \langle \text{lower}, \text{raise} \rangle \]
\[ \text{move4} = \Lambda r_1, s_1. \Lambda r_2, s_2. \langle \text{COMP} \ r_2 \ r_1, \ \text{COMP} \ s_1 \ s_2 \rangle \]
\[ \text{move5} = \Lambda r_1, s_1. \Lambda r_2, s_2. \langle \text{APP} \ \text{APP} \ \text{conj} \ s_1 \ r_2, \ \text{APP} \ \text{APP} \ \text{conj} \ r_1 \ s_2 \rangle \]

\[ \text{hole} \not\sqsubseteq A \]
\[ \text{APP} \ \text{hole} \ B \not\sqsubseteq I \]
\[ \text{APP} \ \text{hole} \ B = I \]
\[ \text{APP} \ \text{hole} \ CB \not\subseteq I \]

\[ \text{hole} \not\subseteq \text{move1} \quad \text{move1} \subseteq A \]
\[ \text{hole} \not\subseteq \text{move2} \quad \text{move2} \subseteq A \]
\[ \text{hole} \not\subseteq \text{move3} \quad \text{move3} \subseteq A \]
\[ \text{hole} \not\subseteq \text{move4} \quad \text{move4} \subseteq A \]
\[ \text{hole} \not\subseteq \text{move5} \quad \text{move5} \subseteq A \]
Example language

language = {
    'APP': 1.0,
    // 'COMP': 1.6,
    'JOIN': 3.0,
    'B': 1.0,
    'C': 1.3,
    'A': 2.0,
    '⊥': 2.0,
    '⊤': 2.0,
    'I': 2.2,
    // 'Y': 2.3,
    'K': 2.6,
    'S': 2.7,
    'J': 2.8,
    'DIV': 3.0,
}
Approximate with an abstract interpretation

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

$$\lfloor x \rceil = \{$$
- above: \( \{ p \in P \mid x \sqsubseteq p \} \),
- below: \( \{ p \in P \mid p \sqsubseteq x \} \),
- nabove: \( \{ p \in P \mid x \nsubseteq p \} \),
- nbelow: \( \{ p \in P \mid p \nsubseteq x \} \)

$$\}$$

Growing $P$ leads to more accurate approximations
Approximate with an abstract interpretation

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

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

Growing \( P \) leads to more accurate approximations \([x]\).
Verify by constraint propagation

Constraints are simple predicates: $=, \subseteq, \not\subseteq$. This language is $\Pi^2_0$-complete.
Verify by constraint propagation

Constraints are simple predicates: $=$, $\subseteq$, $\not\subseteq$. This language is $\Pi_0^2$-complete.

$[x \subseteq y]$ 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: $=,$ $\subseteq,$ $\not\subseteq.$ This language is $\Pi^2_0$-complete.

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

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

\[
\frac{x \subseteq x' \quad y \subseteq y'}{z \subseteq \text{APP } x' y'}
\quad
given

\[
\frac{x' \subseteq x \quad y' \subseteq y}{\text{APP } x' y' \subseteq z}
\]

etc.
And focus on data

1. Invest in a curated knowledge base.

2. Leverage the knowledge base in synthesis.
Invest in a Knowledge Base

\~ 1 engineer year, \~ 10 CPU year,
\~ 1GB compressed, \~ 10GB in memory.
Invest in a Knowledge Base

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

Automatically deduce many small facts.

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

VM-based with an inference rule compiler, 
with \~ 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 $\sim 1M$. 
Search for plausible conjectures.

Among millions of statements

\[ x = y, \ x \sqsubseteq y, \ x \not\sqsubseteq y \]

- 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

\[
\begin{align*}
T & \not\subseteq \bot \\
\frac{f \ x \ not \subseteq \ f \ y}{x \ not \subseteq \ y}
\end{align*}
\]
Questions?

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

Combinatory algebra is a simple language: $S, K, I, AP(−, −)$.

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.