Synthesizing Data Structure Transformations from Input-Output Examples

John Feser, Swarat Chaudhuri, and Isil Dillig

Problem: Synthesis of functional programs from examples

Specification

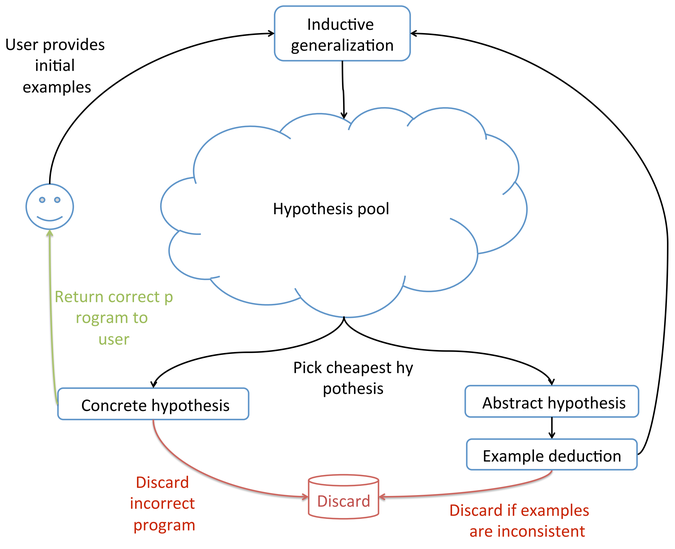
$$[\,] \to [\,]$$
$$[0] \to [0]$$
$$[0\ 1] \to [1\ 0]$$

- User provides examples
- Nested data structures

Result

$\lambda x.\ \texttt{foldl}\ x\ (\lambda z y.\ y : z)[\,]$

- Higher-order functional program
- Recursion encapsulated using combinators



Search process

What is a hypothesis?

A program that may contain holes

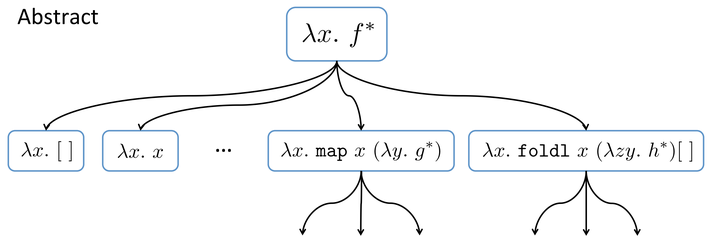
$\lambda x.\ x$ — Concrete

$\lambda x.\ \texttt{map}\ x\ (\lambda y.\ g^*)$ — Abstract

Key Idea: Inductive generalization

Generate lazy set of hypotheses from set of examples

Concrete hypotheses are generated lazily by composing operators with function arguments and constants.



Refinement tree

Abstract hypotheses drawn from the following list:

$\lambda x.\ \texttt{map}\ f\ x$	Map a function over a list
$\lambda x.\ \texttt{mapt}\ f\ x$	Map a function over a tree
$\lambda x.\ \texttt{filter}\ f\ x$	Filter a list with a predicate f
$\lambda x.\ \texttt{foldl}\ f\ e\ x$	Fold a function over a list from left to right
$\lambda x.\ \texttt{foldr}\ f\ e\ x$	Fold a function over a list from right to left
$\lambda x.\ \texttt{foldt}\ f\ e\ x$	Fold a function over a tree from bottom up
$\lambda x.\ \texttt{recl}\ f\ e\ x$	General recursion over a list

Key Idea: Deduction

For an abstract hypothesis and a set of examples, generate examples for the holes in the hypothesis

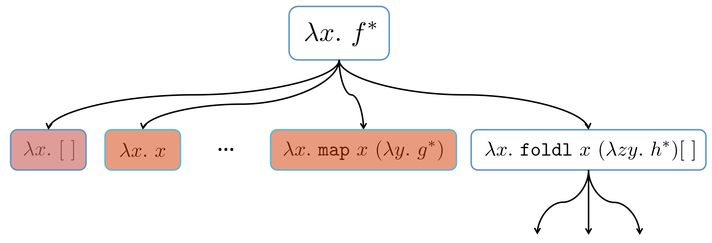
Examples for f*

$$[\,] \to [\,]$$
$$[2] \to [2]$$
$$[2\ 1] \to [1\ 2]$$
$$[2\ 1\ 3] \to [3\ 1\ 2]$$

Child hypothesis: $\lambda x.\ \texttt{map}\ x\ (\lambda y.\ g^*)$

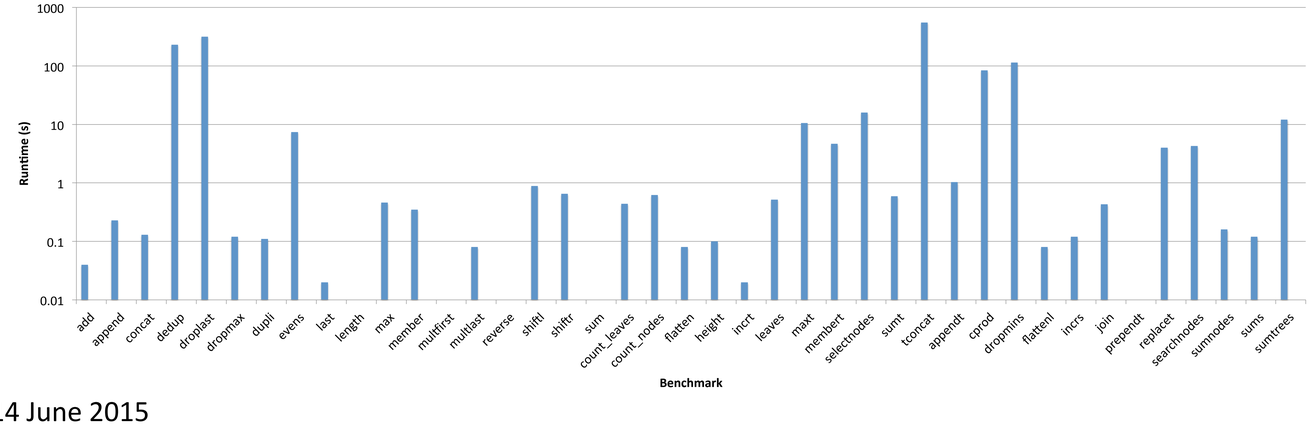
Examples for g*

$$1 \to 1$$
$$1 \to 2$$
$$2 \to 1$$
$$2 \to 2$$
$$2 \to 3$$
$$3 \to 2$$



When examples are inconsistent, prune subtree

Results:



- 75% of problems require ≤ 5 examples
- 88% of problems synthesized in ≤ 5 minutes

Random specification testing

- Generate random input, pass to solution program to generate random example
- Determine how many examples are needed to successfully synthesize a program in 90% of runs
- Number of random examples needed is comparable with number of expert examples, sometimes smaller
- Runtimes on random examples are comparable