Lecture 20: Oracle Guided Program Synthesis

17-355/17-655/17-819: Program Analysis

Rohan Padhye and Jonathan Aldrich

April 13, 2021

* Course materials developed with Claire Le Goues

Synthesis Approaches We've Seen So Far

- Deductive Synthesis
  - Explore the space of programs equivalent to some spec, choose the best one
  - E.g. Denali for superoptimization

- Inductive Synthesis with Sketching
  - User specifies a "sketch" – a program with holes
  - Alternates generating a program from inputs, and generating counterexample inputs to force a better program, until convergence on a correct program

- What if we want to do inductive synthesis, but don't have a sketch idea?

Today: Oracle-Guided Inductive Synthesis

1. Generalize CEGIS (counterexample-guided inductive synthesis)
   - From sketches to arbitrary programs

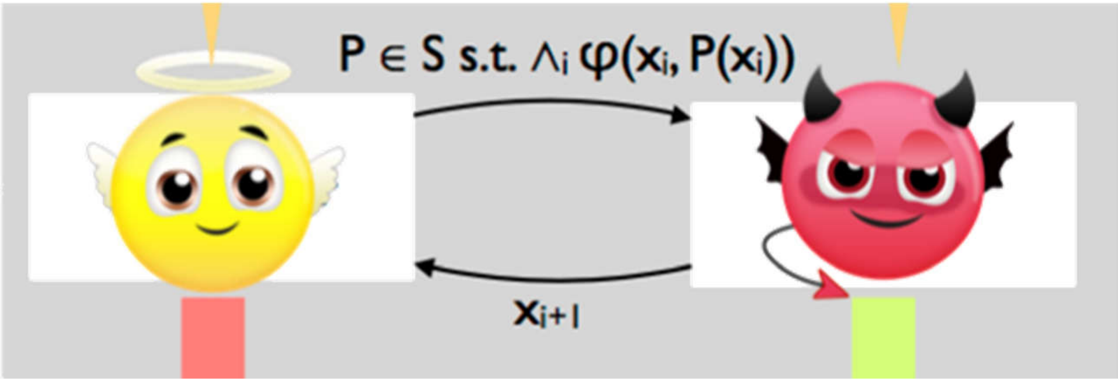
2. Synthesize programs from components

CEGIS: A Mathematical View

- Let's formalize Counterexample-Guided Inductive Synthesis (CEGIS)
- Consider a formalization of synthesizing a max function for lists

$$\exists P_{max} \forall l, m : P_{max}(l) = m \Rightarrow (m \in l) \wedge (\forall x \in l : m \geq x)$$

- CEGIS iterates between synthesis from examples and counterexample generation



- How do we generate a counterexample?

Counterexample generation, formalized

- Let's say we have a candidate program P_{max}. Does it meet the spec?
  - Here's how that can be formalized:

$$\forall l, m : P_{max}(l) = m \Rightarrow (m \in l) \wedge (\forall x \in l : m \geqslant x)$$

- By De Morgan's Law, this is equivalent to disproving the negation:

$$\exists l, m : (P_{max}(l) = m) \wedge (m \notin l \vee \exists x \in l : m < x)$$

- This finds a list l and a corresponding incorrect output m
- Let's tweak this to generate the correct output, $m*$:

$$\exists l, m^* : (P_{max}(l) \neq m^*) \wedge (m^* \in l) \wedge (\forall x \in l : m^* \geqslant x)$$

- We can use this to help generate the next version of P_{max}

Oracle-Guided Component-Based Program Synthesis

- Goal: given a set of N components $f_1 \dots f_N$ and a set of n input/output pairs $< \alpha_0, \beta_0 > \dots < \alpha_n, \beta_n >$, synthesize a function f such that $\forall \alpha_i . f(\alpha_i) = \beta_i$
- We search for programs of a particular form:

0	$z_0 := \texttt{input}^0$
1	$z_1 := \texttt{input}^1$
$\dots$	$\dots$
m	$z_m := \texttt{input}^m$
$m+1$	$z_{m+1} := f_?(z_?, \dots, z_?)$
$m+2$	$z_{m+2} := f_?(z_?, \dots, z_?)$
$\dots$	$\dots$
$m+N$	$z_{m+N} := f_?(z_?, \dots, z_?)$
$m+N+1$	$\texttt{return}\ z_?$

Put inputs in variables

Compute N functions, each of which has arguments

Choices: fill in the ?s
- What order are the functions in?
- What variables are passed to each function?
- What variable is returned?

The program is defined by a set of variables

- Program input variables $\vec{Y}$
- Input to each component $Q := \bigcup_{i=1}^{N} \vec{\chi}_i$
- Output of each component $R := \bigcup_{i=1}^{N} r_i$
- Output of the program r

0	$z_0 := \texttt{input}^0$
1	$z_1 := \texttt{input}^1$
$\ldots$	$\ldots$
m	$z_m := \texttt{input}^m$
$m+1$	$z_{m+1} := f_?(z_?, \ldots,$
$m+2$	$z_{m+2} := f_?(z_?, \ldots,$
$\ldots$	$\ldots$
$m+N$	$z_{m+N} := f_?(z_?, \ldots,$
$m+N+1$	$\texttt{return}\ z_?$

Program variables are specified by location variables

- Location variable l_x specifies where x is defined
- L is the set of location variables

$$L \quad := \quad \{l_x | x \in Q \cup R \cup \overrightarrow{Y} \cup r\}$$

(again: component inputs, component results, program inputs, and program result)

0	$z_0 := \texttt{input}^0$
1	$z_1 := \texttt{input}^1$
$\ldots$	$\ldots$
m	$z_m := \texttt{input}^m$
$m+1$	$z_{m+1} := f_?(z_?, \ldots,$
$m+2$	$z_{m+2} := f_?(z_?, \ldots,$
$\ldots$	$\ldots$
$m+N$	$z_{m+N} := f_?(z_?, \ldots,$
$m+N+1$	$\texttt{return}\ z_?$

Example of Location Variables

- Imagine we have one input and one component, +
- Here's a sample program:

$$\begin{array}{ll} 0 & z_0 := \texttt{input}^0 \\ 1 & z_1 := z_0 + z_0 \\ 2 & \texttt{return}\ z_1 \end{array}$$

- This can be specified by the location variables

$$\{l_{r_+} \mapsto 1, l_{\chi_+^1} \mapsto 0, l_{\chi_+^2} \mapsto 0, l_r \mapsto 1, l_Y \mapsto 0\}$$

Practice with Location Variable Encodings

Assume two components, * and <<, each of which takes two inputs and produces a single output. Provide a map which assigns values to location variables that describe the following straight-line code. For your reference, the variables are: $\overrightarrow{Y}$ r $\overrightarrow{X}_i$ r_i

z_0 = $input_0$

z_1 = $input_1$

z_2 = z_0 << z_1 *// component <<*

z_3 = z_2 * z_2 *// component **

return z_2

Well-formedness constraints on the generated program

- Component inputs come from locations 0... M
  - M = number of inputs $|\overrightarrow{Y}|$ + number of functions N

$$\bigwedge_{x \in Q} (0 \leqslant l_x < M)$$

- Component outputs defined after program inputs

$$\bigwedge_{x \in R} (|\overrightarrow{Y}| \leqslant l_x < M)$$

- One component per line

$$\bigwedge_{x,y \in R, x \not\equiv y} (l_x \neq l_y)$$

- Component inputs are defined before use

$$\bigwedge_{i=1}^{N} \bigwedge_{x \in \overrightarrow{\chi}_i} l_x < l_{r_i}$$

0	$z_0 := \texttt{input}^0$
1	$z_1 := \texttt{input}^1$
$\ldots$	$\ldots$
m	$z_m := \texttt{input}^m$
$m+1$	$z_{m+1} := f_?(z_?, \ldots, z_?)$
$m+2$	$z_{m+2} := f_?(z_?, \ldots, z_?)$
$\ldots$	$\ldots$
$m+N$	$z_{m+N} := f_?(z_?, \ldots, z_?)$
$m+N+1$	$\texttt{return}\ z_?$

Functionality constraints

- Variables defined at the same location are the same (have the same value)
  - Basically: define value flow from definition to use

$$\bigwedge_{x,y \in Q \cup R \cup \overrightarrow{Y} \cup \{r\}} (l_x = l_y \Rightarrow x = y)$$

- The program inputs and outputs match a test case pair
  - We repeat this for all test cases

$$(\alpha = \overrightarrow{Y}) \wedge (\beta = r)$$

- Functional components obey their specification

$$(\bigwedge_{i=1}^{N} \phi_i(\overrightarrow{\chi}_i, r_i))$$

Component-Based Synthesis, Overall

- We conjoin the well-formedness and functionality constraints into one big formula
- We have an SMT solver solve that formula
- The result is a witness, assigning integer values to each location variable
    - We can then convert the witness into a program
    - Line i of the program:

$$z_i = f_j(z_{\sigma_1}, \dots, z_{\sigma_\eta}) \text{ when } l_{r_j} == i \text{ and } \bigwedge_{k=1}^{\eta} (l_{\chi_j^k} == \sigma_k)$$

- We can then put this into a CEGIS loop: