On the Importance of Common Sense in Program Synthesis

Hila Peleg, Technion

The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement n° 615688 – ERC- COG-PRIME.
Program Synthesis

Specifications

Candidate solution
Programming by Example

\[ \mathcal{E} = \{ "abdfibfcfdeb" \mapsto "bd" \} \]

\[ \mathcal{E} = \{ "abdfibfcfdeb" \mapsto "bd" \} \]

\[ p \ s.t. \ \forall (\iota, \omega) \in \mathcal{E}. \left[ p \right](\iota) = \omega \]
Why examples?

• The process of writing code:
  1. Problem
  2. Intent
  3. Solution (which might have bugs, go to #1)

• Programmers use examples:
  1. To understand the problem
  2. To formulate intent
  3. To test the solution

9. Write a function that concatenates two lists.  
\[ [a, b, c], [1, 2, 3] \rightarrow [a, b, c, 1, 2, 3] \]

10. Write a function that combines two lists by alternatingly taking elements, e.g.  
\[ [a, b, c], [1, 2, 3] \rightarrow [a, 1, b, 2, c, 3]. \]

https://adriann.github.io/programming_problems.html
ML has used this since the 70s

• Version Spaces [Mitchell ‘77]
  • Generalize using a set of “concepts” that are ranked for generality
Version Spaces and Concept Learning

• Generalize the set of cards you’re shown
ML has used this since the 70s

• Version Spaces [Mitchell ‘77]
  • Generalize using a set of “concepts” that are ranked for generality
• ML mimicking how humans generalize from multiple examples
  • Given n examples, remember what they have in common
  • If updating with example n+1, see what it has in common with the list for n examples
• (Under a specific set of circumstances, we would call this \( \sqcup \))
Bias, our savior

An unbiased language: one that can describe any subset of concrete examples.
Bias in ML

• A ML algorithm is biased by its language of classifiers.
• This is not a bad thing!
  • Bias is what gives us generalization to examples we haven’t seen
• Desirable biases:
  • Domain knowledge on samples (“samples are English letters, it doesn’t matter what we decide for Klingon”)
  • Domain knowledge on use of the result (“will flag suspicious transactions for human review, ok to have false positives on one parameter”)
Bias in synthesis

• Synthesis (often powered by ML) is biased by the search space and the search algorithm

• Search space: possible programs
  • We can’t represent what we can’t represent
  • In combination with the algorithm, programs “mask” each other

• Search algorithm: will lead to overfitting
  • Input: "abdfibfcfdebdfdebdihgfkjfdebd"
  • Output: "bd"
  • Resulting program: `input.takeRight(2)`
Common sense vs. bias

• Human common sense is (supposedly) able to give us *unbiased* learning that can generalize past existing examples.

• Let’s try it out:
  • \( f(1,1) = 1 \)
  • \( f(1,2) = 2 \)
  • \( f(3,0) = 3 \)

Three examples is a very small number to generalize from (if you’re a computer)! Well done!
Generalize better with multiple biases?

• One way to try to be less biased is to have multiple competing biases

• Example: let’s look for \( f(x, y, z): \text{bool} \) where
  
  • \( f(2.3, 5.7, 4.0) = f(1.0, 1.0, 1.0) = f(2.0, 3.0, 2.5) = \text{true} \)
  
  • \( f(2.0, 2.0, 2.1) = \text{false} \)

\[
z = a \cdot x + b \cdot y \\
z \geq |a \cdot x - b \cdot y| \\
z \leq |a \cdot x - b \cdot y|
\]

\[
z = \frac{1}{2} \cdot x + \frac{1}{2} \cdot y \\
\text{No consistent } a, b \\
z \leq |\frac{2}{5} \cdot x + \frac{3}{5} \cdot y|
Generalize better with multiple biases

• What do you do when there’s more than one answer?
• We can rank them and pick the highest rank
• Ranking is domain specific
  • Human-directed intervention in a less-biased system
What happens when we change domains?

• Will our ranking still fit?
JARVIS, a test synthesizer [VMCAI18]

```java
int a = foo();
assert(a == 8);
```

Multiple biases here

- **Group**: compatible tests
- **Abstract**: generalized behavior
- **Sample**: generate values
What happens when we change domains?

- Will our ranking still fit?
- JARVIS ranking of numerical domains is well suited to testing numerical libraries
- When we wanted something different (e.g. geometry) required re-doing the ranking to fit new domain

- What could we have done differently?
What could we have done differently?

• We suggested the user might occasionally help JARVIS out.

• Why not? The user can bring a moment of common sense to the mix:
  • Choose between generalizations
  • Manually generalize the result more

• While the synthesizer still does the brunt of the work
But for any more we want to involve a human

JARVIS, FlashFill

Search space with domain specific bias

A less biased search space

Injecting common sense into the system
Test Driven Development (Nature’s PBE)

• An iterative process
  • introducing a failing test
  • writing the minimal amount of code to make that test pass
Let’s program a calculator!

Test code:

```java
@Test
public void calculatorEmptyString() {
    Calculator c = new Calculator();
    int res = c.evaluate("");
}
```

System code:

```java
public class Calculator {
    public int evaluate(String expr) {
        return -1;
    }
}
```
@Test
public void calculatorEmptyString() {
    Calculator c = new Calculator();
    int res = c.evaluate(""');
    Assert.assertEquals(res, 0);
}

public class Calculator {
    public int evaluate(String expr) {
        return -1;
    }
}
Fix the code to match

Test code:

```java
@Test
class Calculator {
    @Test
    public void calculatorEmptyString() {
        Calculator c = new Calculator()
        int res = c.evaluate("")
        Assert.assertEquals(res, 0)
    }
}
```

System code:

```java
public class Calculator {
    public int evaluate(String expr) {
        return 0;
    }
}
```
Test Driven Development (Nature’s PBE)

• An iterative process
  • introducing a failing test
  • writing the minimal amount of code to make that test pass

• This is just what we do in PBE:
  1. Differentiating input-output example (failed test)
  2. Find next program that matches (make all tests pass)

• So what’s the difference? The lack of common sense.
TDD vs. PBE

• Test 1: isPrime(5) == true
  • TDD: return true;
  • PBE: return true;

• Test 2: isPrime(4) == false
  • TDD: return x % 2 == 1
  • PBE: return x == 5

The less bias in our search space, the more the result will lean toward an overfitted result
We need bias (even though we don’t like it)

• The human can create bias for us

Rule out parts of the search space

Rank candidates

Bring domain knowledge to a generic synthesizer

\[ z = \frac{1}{2} \cdot x + \frac{1}{2} \cdot y \]

\[ z \leq \left| \frac{2}{5} \cdot x + \frac{3}{5} \cdot y \right| \]
Demo time!
Inversion of control

• Current program synthesis: user repeatedly queries the synthesizer

• Inversion of control:
  • the synthesizer uses the human in order to get the best result
  • ask the best questions
  • give the best feedback tools
Help the user (help us)

• Because the user isn’t perfect, either
Extreme programming, the computer aided version

• PBE is (kind of) like TDD

• More generally, programming with a synthesizer is like pair programming

• This is our ideal
  • The machine brings the knowledge
  • The human brings the common sense