Programming by Example

Lecture #9

Spring, 2018

236608

Software Synthesis and Automated Reasoning
Reasoning

- \(\lambda\)-calculus
- Dependent Types

Type Theory (basics)

Axiomatic Semantics

Satisfiability Modulo Theory

Programming by Example

Syntax Guided Synthesis

Counterexample Guided Inductive Synthesis

Type Directed Synthesis

Refinement Types
Synthesis!

MIT & NASA, 1957

"Code" (~165cm)
Synthesis!

- Automatic programming?
  - but I have to tell the computer what I want...

• Level of abstraction

???
Python, Haskell, ...
C
assembly
machine code

Synthesis = an unusually concise / intuitive programming language + a compiler that sometimes doesn’t work 😊
Programming by Example

= 

Synthesis from Examples

= 

Inductive Synthesis

(sometimes: inductive learning)

Slides adapted from Nadia Polikarpova
Ancient Synthesis: Zendo

What is the secret of the Buddha-Nature?

- Zen master provides a set of positive (white) and negative (black) examples.

- Students have to guess the general rule.
Programming by Example

• Synthesize a program whose behavior satisfies a set of examples

  - Wait... doesn’t machine learning do that?
Programming by Example

Traditional Machine Learning

- Learn a function from a set of examples
- Millions of data points
  Scalability is a challenge
- Data is noisy
  Need to avoid overfitting, but approximate solutions are also good enough
- Search space is parametrized
  (vector)

Inductive Synthesis

- Learn a function from a set of examples
- Small numbers of examples
  Ambiguity is a challenge
- Data is clean
  It would be annoying if user said $f(x)=5$ and the system assumed the user is wrong and decided that $f(x)=6$
- Search space has complex structure
  (syntax tree)

Optimization-based search (fast)

Combinatorial search (slow)
Programming by Example

• Motivation

Algorithm Designers

Software Developers

End-Users
(including: students and teachers)
Inductive Learning: Key Issues

1. How do you find a program that matches the observations?
2. How do you know it is the program you are looking for?

Space of programs

Programs matching the observations

Program you actually want

Traditional ML emphasizes

This one we’ll manage
Inductive Learning: Key Issues

1. How do you find a program that matches the observations?

2. How do you know it is the program you are looking for?

Modern research focuses more on 1.

But we shouldn’t forget this one.
Introduction to Sketching

- **Idea**: write the program but leave **holes**.
  - Simple hole construct: “??”
    Sketch will replace it with an **integer constant**.

```
harness void doubleSketch(int x) {
    int t = x * ??;
    assert t == x + x;
}
```
Theseus and the Minotaur

• Rules:
  ‣ Theseus can move in any direction (but not through walls).
  ‣ Minotaur obeys the holy laws of King Minos:
    ○ Only moves if can get closer to Theseus
    ○ Horizontal takes precedence over vertical
    ○ Makes two moves for every move of Theseus
Introduction to Sketching

• Control — conditional

```java
if ( ?? )
    t = doOneThing( ?? );
else
    t = doAnotherThing( ?? );
```

```java
int sel = ?? ;
if (sel == 0)
    t = doOneThing( ?? );
else if (sel == 1)
    t = doAnotherThing( ?? );
else
    assert false;
```
Introduction to Sketching

• Control – repeat

```
repeat (??) {
    t = someFunc(t, ??);
    t = t + 1;
}
≠
```

```
for (int i = 0; i < ??; i++) {
    t = someFunc(t, ??);
    t = t + 1;
}
```

one hole per instance

same value in all iterations
Introduction to Sketching

• Generators

```c
 generator int affine(int a, int b) {
   return ?? + a * ?? + b * ?? ;
 }
```

• Without `generator`: same value for all invocations (like a normal function)
Lab #8

Create a list of moves for Theseus using Sketch.

Q1. Ignore Minotaur, find the exit.

Q2. Model Minotaur moves as well, avoid capture.

• Use: ??, assert, repeat

At THESEUS = new At(x=1, y=2);
At MINOTAUR = new At(x=1, y=0);
At EXIT = new At(x=2, y=1);

harness escape() {
    At theseus = copy(THESEUS),
    minotaur = copy(MINOTAUR);
    ...
    left, right, right, up ...

    assert (success)
}