CS 190I Program Synthesis for the Masses

Lecture 5: Inductive Program Synthesis

Motivated by Armando's CS-6.887 at MIT

Yu Feng
Spring 2021

Summary of previous lecture

- 2nd paper review was out
- Review of propositional logic
- Normal forms (NNF, DNF, CNF)
- A basic SAT solver (DPLL algorithm)
- https://rise4fun.com/z3

Program synthesis

- Deductive synthesis
  - The goal is to **derive** a function/program from a formal specification (e.g., logical formula)
- Inductive synthesis
  - The goal is to **generate** a function/program that matches a given set of input/output examples (observations)

Inductive synthesis

- The goal is to generate a function/program that matches a given set of input/output examples (observations)
- Programming by Example (PBE) v.s. Programming by Demonstration (PBD)
- Synthesize the factorial function based on an example
    - fact(6) = 720
- Synthesize the factorial function based on trace
    - fact(6) = 6*(5*…*1) = 720

Core challenges in PBE

- There are two core challenges in the PBE/PBD paradigm
- C1 (Correctness): How to find a program that matches the observations? Where the observations can be either input/output examples or richer execution traces as in PBD.
- C2 (Generality): How do you know the program you found is the one you were actually looking for? There is a potentially large space of possible programs that match the given observations, so how do we know which one matches the user intent

Core challenges in PBE

- In PL folks focus more on the first challenge
  - Carefully design the space of programs in a way that excludes undesirable solutions from the space and focuses the search on "reasonable" programs.
  - Effective pruning technique to avoid “bad” programs
- Machine learning mainly focuses on the second challenge
  - Pick spaces of programs that are either extremely expressive (e.g. neural networks) so that there are many different ways to match any set of observations and the challenge is how to avoid over-fitting
  - Trade off how many samples you match against other criteria that make it more likely that your solution will work well enough in general

Represent a program

- The most common representation is an Abstract Syntax Tree (AST)
- A tree with different kinds of nodes for different kinds of constructs in the language
- The syntax of such a language can be represented as a context free grammar:

EXPR → **if** *EXPR* **then** *EXPR* **else** *EXPR*

| *EXPR* + *EXPR*

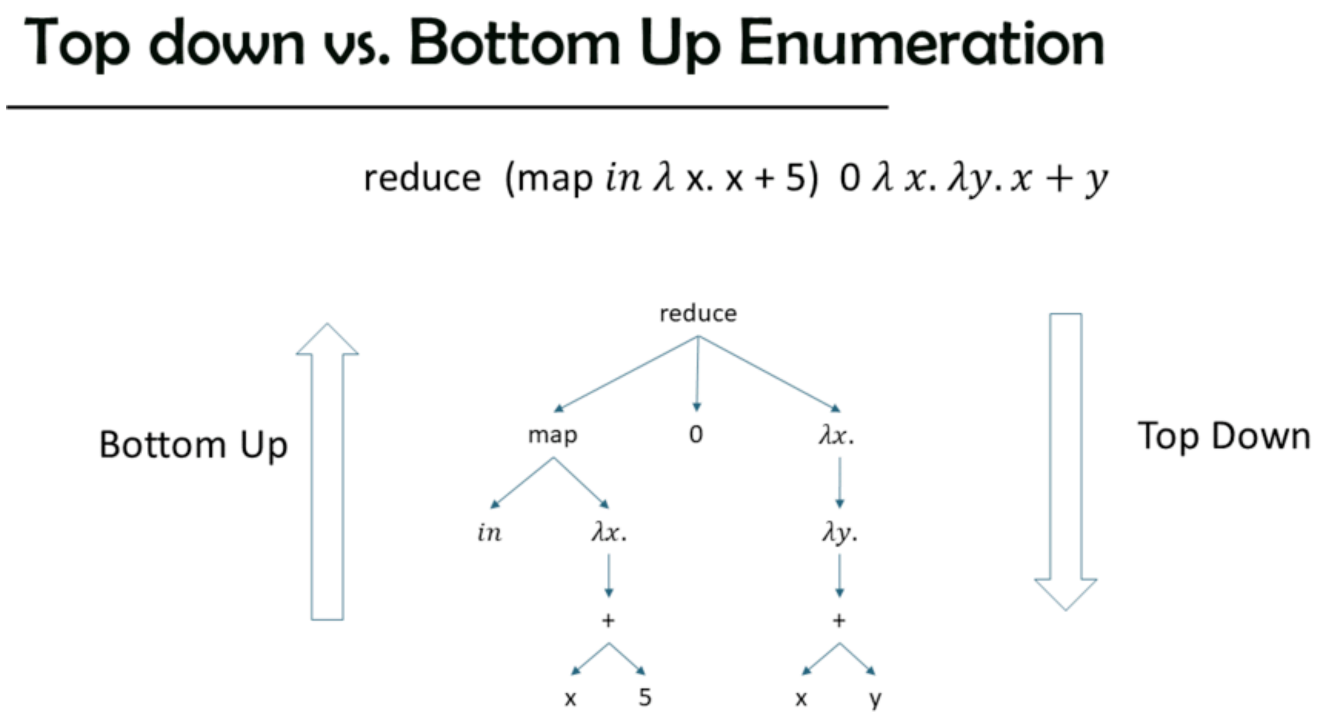
| *ID*

Challenges & Search techniques

- In any program synthesis system, the challenge involves two questions:
    - What is the space of programs
    - How is it going to be searched
- We will explore a few different classes of search techniques

Explicit enumeration

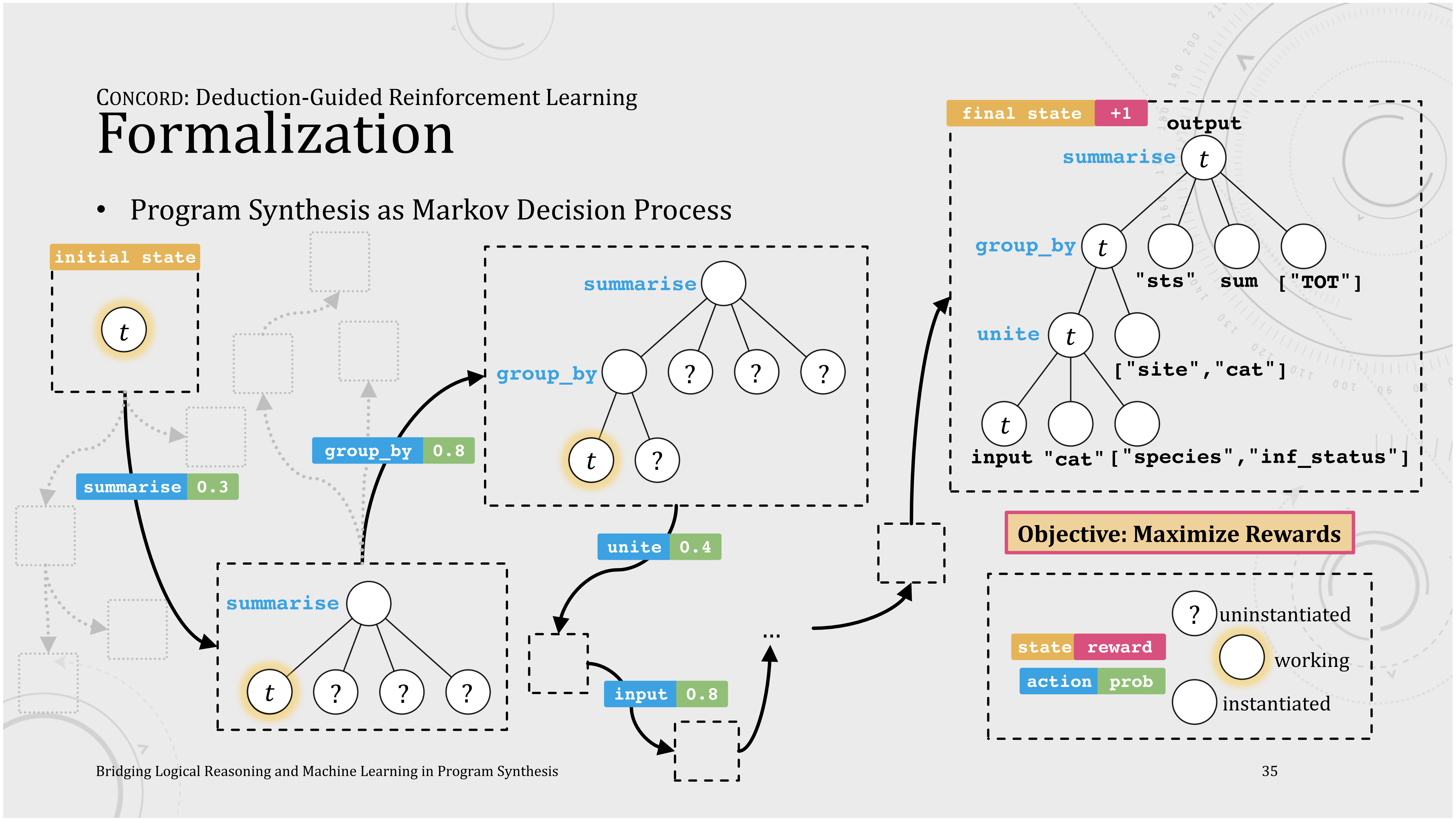
- Explicitly construct different programs until one finds a program that satisfies the specifications.
- The key is how to avoid generating programs that have no hope of satisfying the observations, or which can be shown to be redundant with other programs we have already enumerated.
- BU vs. TD. In bottom up enumeration, the idea is to start by discovering low-level components and then discover how to assemble them together into larger programs. By contrast, top-down enumeration starts by trying to discover the high-level structure of the program first, and from there it tries to enumerate the low-level fragments. Essentially, in both cases we are explicitly constructing ASTs, but in one case we are constructing them from the root down, and in the other case we are constructing them from the leafs up.


Top down vs. Bottom Up Enumeration
reduce (map in λ x. x + 5) 0 $\lambda x. \lambda y. x + y$
reduce
map
0
$\lambda x.$
in
$\lambda x.$
$\lambda y.$
+
+
x
5
x
y
Bottom Up
Top Down

Symbolic search

- In explicit search, the synthesizer always maintains one or more partially constructed programs that it is currently considering.
- Symbolic search techniques maintain a **symbolic representation** of the space of all programs that are considered valid. Different symbolic representations lead to different search algorithms. One popular symbolic representations is **constraint-based system**
- As an analogy, suppose we want to search for an integer value of such that 2+x = 10. Enumerative search would try all the values one by one until it hits the correct one. By contrast, a symbolic search technique may perform some algebraic manipulation to deduce that x = 10-2 = 8

Neural-guided search



CONCORD: Deduction-Guided Reinforcement Learning

Formalization

- Program Synthesis as Markov Decision Process

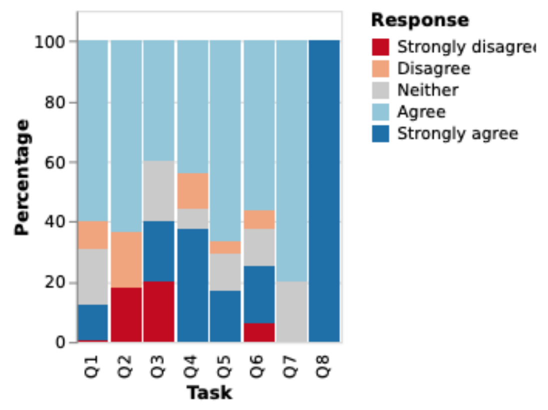
Restrict your search space

- One of the key design decisions is the space of programs that will be considered.
- One way to do this is to define a small domain specific language, and then consider all possible programs within this language.
- It is common to describe the ASTs in the form of a context free grammar.
- It is easy to enumerate all programs in a DSL, or to sample randomly from the space

DSL in Falx

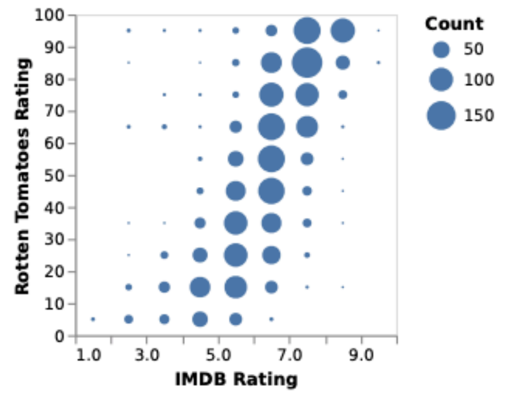
```
Stacked(
  x=Task,
  h=Percent,
  color=Response)
```

Task	Response	Percent
Q1	Disagree	9.1%
Q1	Agree	59.9%
...	...	...



```
Scatter(
  x=IMDB,
  y=RT,
  size=Count)
```

IMDB	RT	Count
1-2	0-10	3
4-5	10-20	76
...	...	...



$$
\begin{array}{rcll}
P_V & = & \mathrm{MultiPlot}(SP, c_{\mathrm{sub}}) \mid SP & \\
SP & = & \mathrm{MultiLayer}(\bar{L}) \mid L & \\
L & = & \mathrm{Scatter}(c_x, c_y, c_{color}, c_{size}) & \text{(Scatter Plot)} \\
 & \mid & \mathrm{Line}(c_x, c_y, c_{color}) & \text{(Line Chart)} \\
 & \mid & \mathrm{Bar}(c_x, c_y, c_{y_2}, c_{color}) & \text{(Bar Chart)} \\
 & \mid & \mathrm{Stacked}(c_x, c_h, c_{color}) & \text{(Stacked Bar Chart)} \\
c & = & column \mid \epsilon &
\end{array}
$$

Fig. 8. The visualization language $\mathcal{L}_V$.

TODOs by next lecture

- Working on R2 and HW1
- Start working the outline of your proposal
- Discuss your final project during office hour!