Advanced Software Security

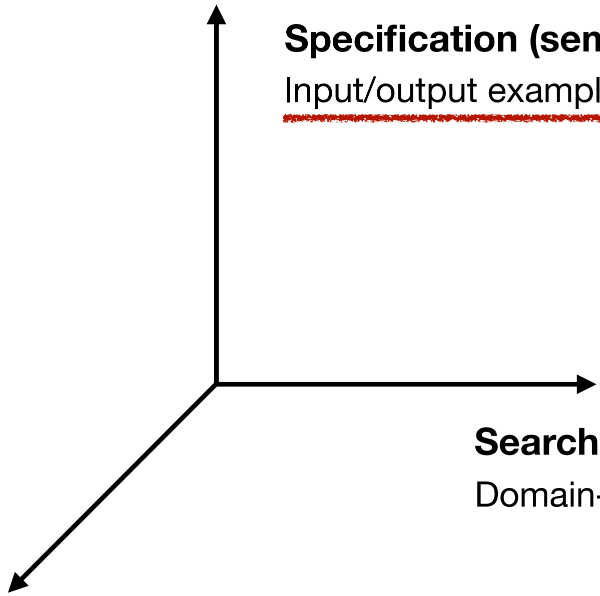
3. Inductive Synthesis and Enumerative Search

Kihong Heo

KAIST

Dimensions in Program Synthesis

Specification (semantic constraint)
Input/output examples



Search strategy
Enumerative search

Search space (syntactic constraint)
Domain-specific language

Inductive Synthesis

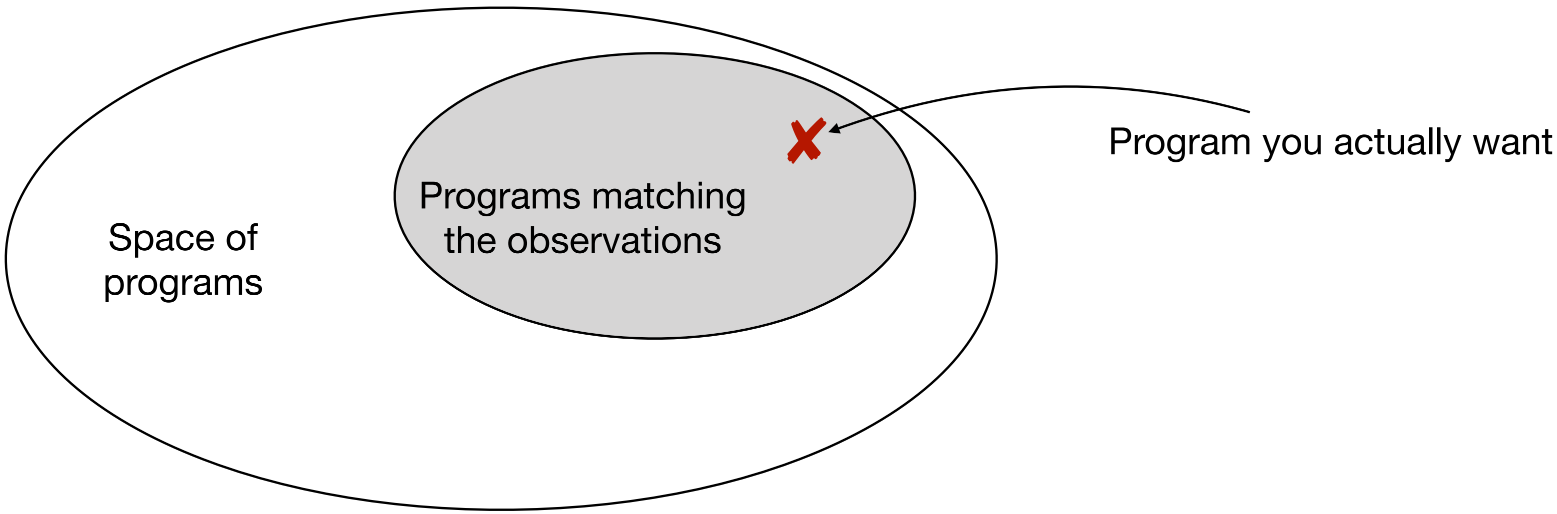
- Given a set of examples, find a program consistent with the examples
  - “Programming by example (PBE)”

x	f(x)
1	1
2	3
3	5
4	7

- Long-standing problem: inductive learning*
  - Problem of generalizing from a set of observations
  - Foundation of modern machine learning

*P. Winston. Learning structural descriptions from examples. 1970

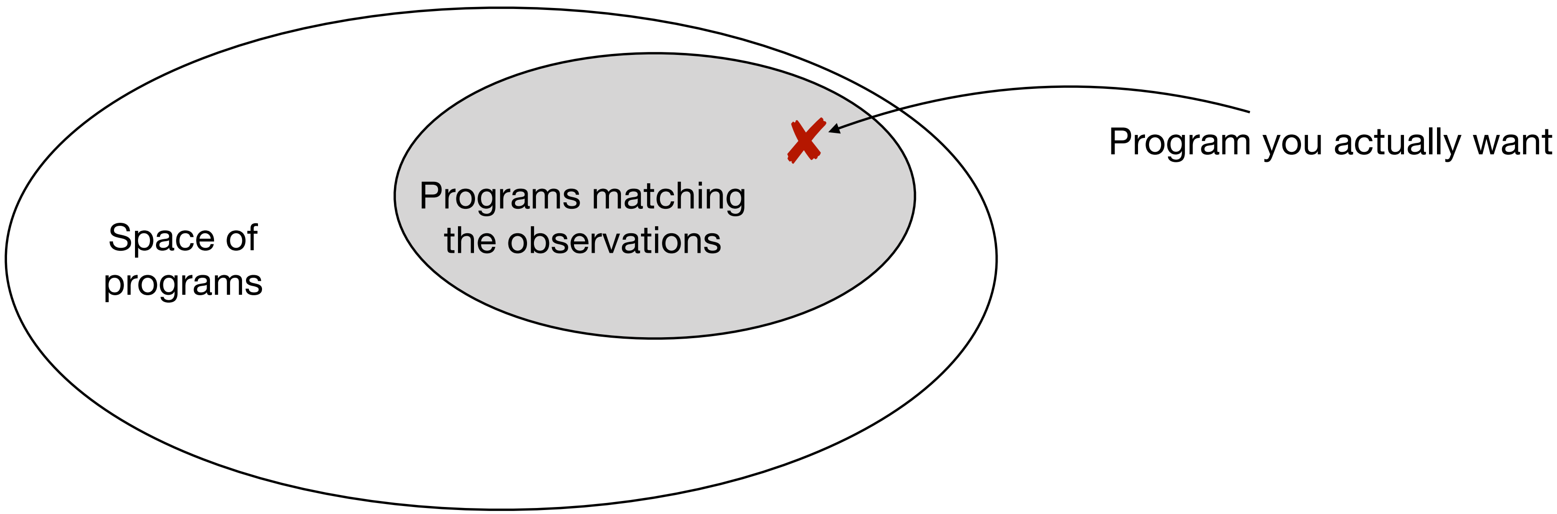
Key Issues in Inductive Learning



1. How to find a program that matches the observations?

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

Key Issues in Inductive Learning

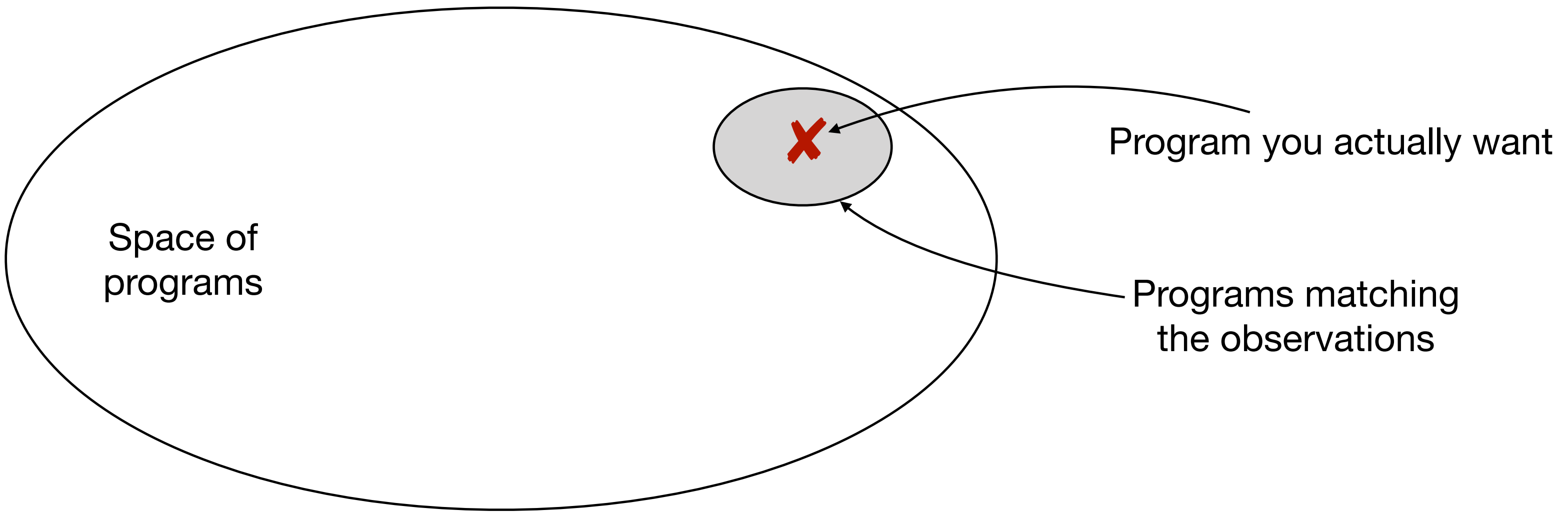


Traditional ML

1. How to find a program that matches the observations? Easy. Fix the space

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

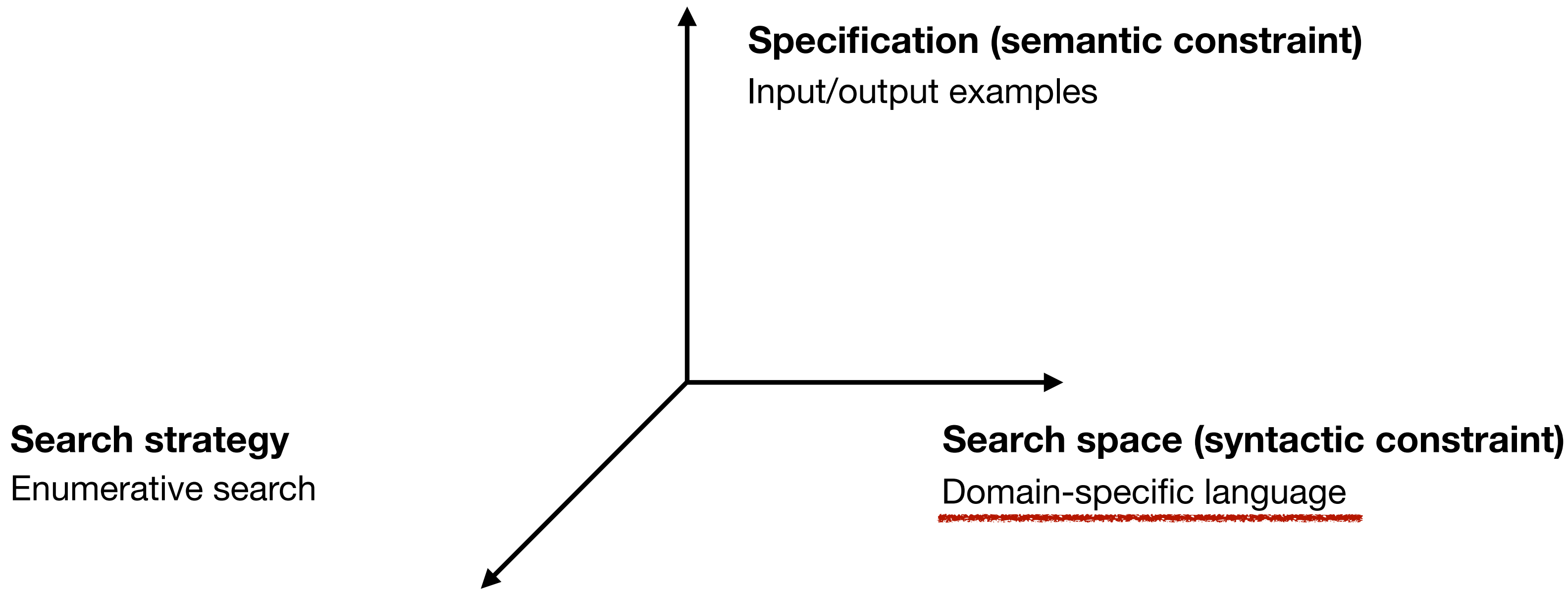
Key Issues in Inductive Learning



Program synthesis

1. How to find a program that matches the observations? — Main challenge.
2. How do you know it is the program you are looking for? — Easy. Customize the space.

Dimensions in Program Synthesis



Program Space

- Should strike a good balance between **expressiveness** and **efficiency**
- Usually described as a context-free grammar of a domain-specific language
  - E.g., restrictions on operators or control structures

$$G = \langle \Sigma, N, R, S \rangle$$

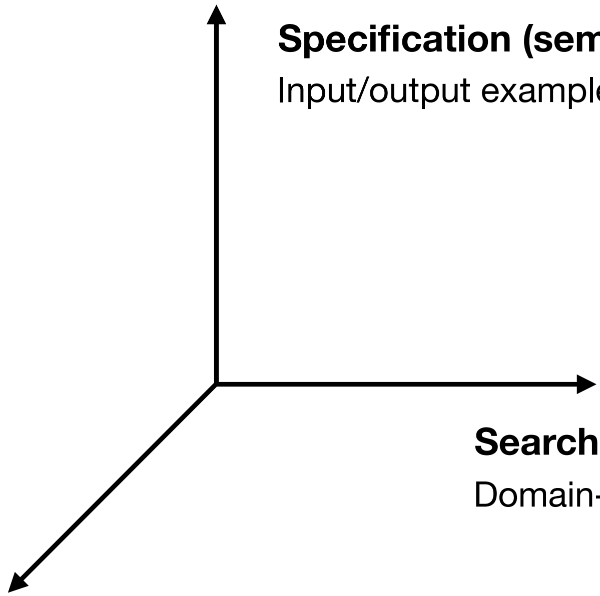
Σ: alphabet N: nonterminals R: production rules S: starting nonterminal

- Example

$$S \to x \mid y \mid 0 \mid 1 \mid S + S \mid S - S \mid \texttt{if}\ B\ S\ S$$
$$B \to S \leq S \mid S = S$$

$$L \to x \mid \texttt{single}(N) \mid \texttt{sort}(L)$$
$$\mid \texttt{slice}(L, N, N) \mid \texttt{concat}(L, L)$$
$$N \to \texttt{find}(L, N) \mid 0$$

Dimensions in Program Synthesis



Specification (semantic constraint)
Input/output examples

Search strategy
Enumerative search

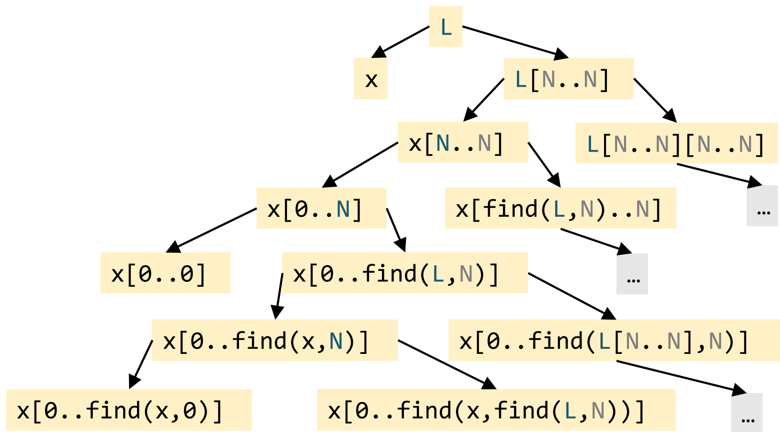
Search space (syntactic constraint)
Domain-specific language

Enumerative Search

- **Explicitly and exhaustively enumerate** programs in the search space until finding a solution
- Idea:
  - Sample programs from the grammar one by one
  - Test them on the examples
- How to **systematically** enumerate?
  - Top-down: starting from the start non-terminal
  - Bottom-up: starting from terminals

Top-down Enumeration

- Search space: tree
  - nodes: incomplete programs
  - edges: left-most productions
- General algorithm:
  - Start from the **start non-terminal**
  - **Expand** left-most non-terminals using all production rules



Example

Specification

Find a function $f(x, y)$ where $f(0,1) = 1 \ \wedge \ f(1,2) = 3$

Grammar

$$S \to x \mid y \mid S + S \mid S - S \mid \texttt{if} \ B \ S \ S$$
$$B \to S \leq S \mid S = S$$

Enumeration

iter 0	S					
iter 1	x	y	S + S	S - S	if B S S	
iter 2	x + S	y + S	x - S	y - S	`if` $(S \leq S)$ S S	…
iter 3	x + x	x + y	y + x	y - y	`if` $(x \leq S)$ S S	…

Top-down Enumeration Algorithm

```
top-down(G = <Σ, N, R, S>, ϕ):
  Q := {S}
  while Q != {}:
    p := dequeue(Q)
    if ground(p) ∧ ϕ(p): return p
    P’ := unroll(G, p)
    forall p’ ∈ P’:
      Q := enqueue(Q, p’)


unroll(G = <Σ, N, R, S>, p):
  Q’ := {}
  A := left-most non-terminal in p
  forall (A → B) in R:
    p’ := p[B/A]
    Q’ := Q’ ∪ {p’}
  return Q’
```

Bottom-up Enumeration

- Generate larger programs using smaller programs (similar to dynamic programming)
- Enumerate in increasing order of program size
- General algorithm:
  - Start from **terminals**
  - **Combine** sub-programs into larger ones using production rules

Example

Specification

Find a function $f(x, y)$ where $f(3,1) = 3 \ \wedge \ f(1,2) = 3$

Grammar

$$S \rightarrow x \mid y \mid S + S \mid S - S \mid \texttt{if}\ B\ S\ S$$
$$B \rightarrow S \leq S \mid S = S$$

Enumeration

iter 1	x	y		
iter 2	$x + y$	$x - y$	$x \leq y$	$x = y$
iter 3	$x + x + y$	$x + x - y$	...	if $(x \leq y)\ y\ x$
iter 4	$x + x + x + y$		...	if $(x \leq y)\ (y + x)\ x$
...				

Bottom-up Enumeration Algorithm

```
bottom-up(G = <Σ, N, R, S>, ϕ):
  Q := set of all terminals in G
  while true:
    forall p in Q:
      if ϕ(p): return p
    Q += grow(G, Q)

grow(G, Q):
  Q' := {}
  forall (A → B) in G:
    Q' += { B[C → p] | p ∈ Q, C →* p }
  return Q'
```

Top-down vs Bottom-up

- Bottom-up:
  - Enumerate **complete** programs
  - Each candidate = executable program
- Top-down: enumerate partial programs
  - Enumerate **incomplete** programs
  - Each candidate = overall structure of future candidates
- Optimization?

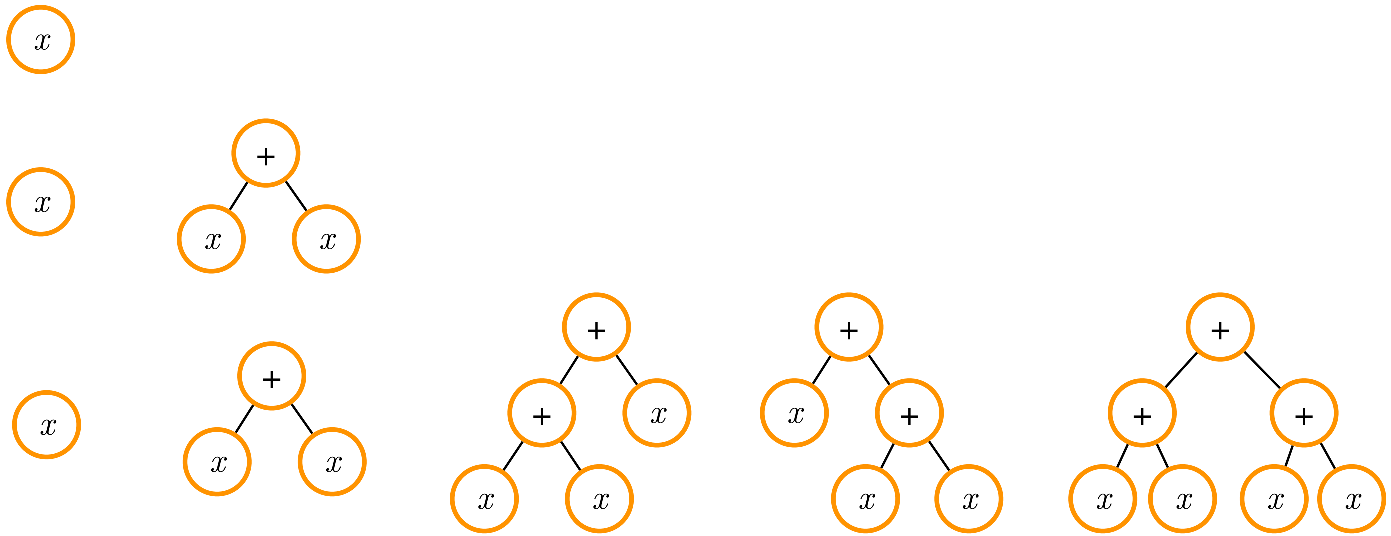
Size of the Problem Space

$$S \rightarrow x \mid S + S$$

depth ≤ 1

depth ≤ 2

depth ≤ 3



N(1) = 1

N(2) = 2

N(3) = 5

$N(d) = 1 + N(d - 1)^2$

*Examples from Nadia Polikarpova’s slides

How Big is the Space?

$$S \to x \mid S + S$$

N(d) = 1 + N(d - 1)²

N(1) = 1
N(2) = 2
N(3) = 5
N(4) = 26
N(5) = 677
N(6) = 458330
N(7) = 210066388901
N(8) = 44127887745906175987802
N(9) = 1947270476915296449559703445493848930452791205
N(10) = 3791862310265926082868235028027893277370233152247388584761734150717768254410341175325352026

‼

*Examples from Nadia Polikarpova's slides

Summary

- Inductive synthesis = programming by example
- Enumerative search: systematically search for solutions
- Challenge: **huge search space**
- **How to optimize the search?**