#2: Syntax-Guided Synthesis

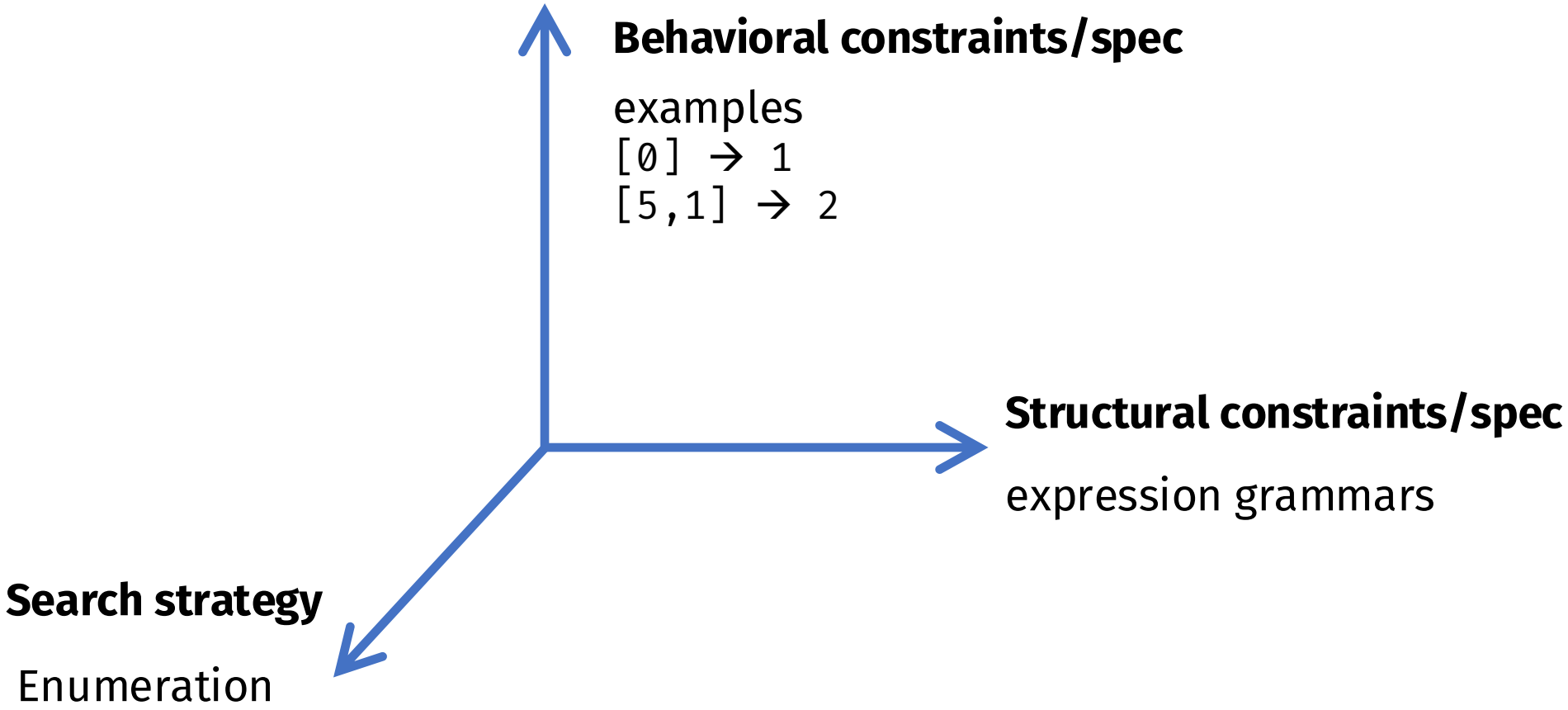
Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

Logistics

- **Office hours time changed**
  - Monday: 4pm – 5pm
- **Other questions?**

This week


Behavioral constraints/spec
examples
[0] → 1
[5,1] → 2
Structural constraints/spec
expression grammars
Search strategy
Enumeration

Today

- **Synthesis from examples**
- **Syntax-guided synthesis**
  - expression grammars as structural constraints
  - the SyGuS project

Synthesis from examples

Synthesis from Examples

=

Programming by Example

=

Inductive Programming /
Inductive Learning

Inductive learning: History

MIT/LCS/TR-76

LEARNING STRUCTURAL DESCRIPTIONS FROM EXAMPLES

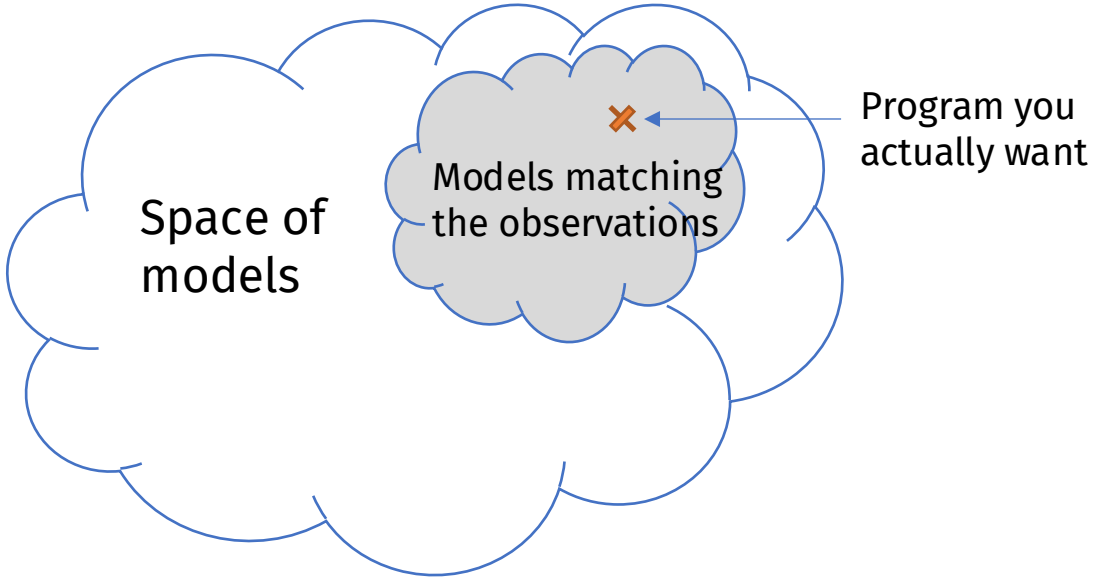
Patrick H. Winston

September 1970

Patrick Winston

- Explored the question of generalizing from a set of observations
- Became the foundation of machine learning

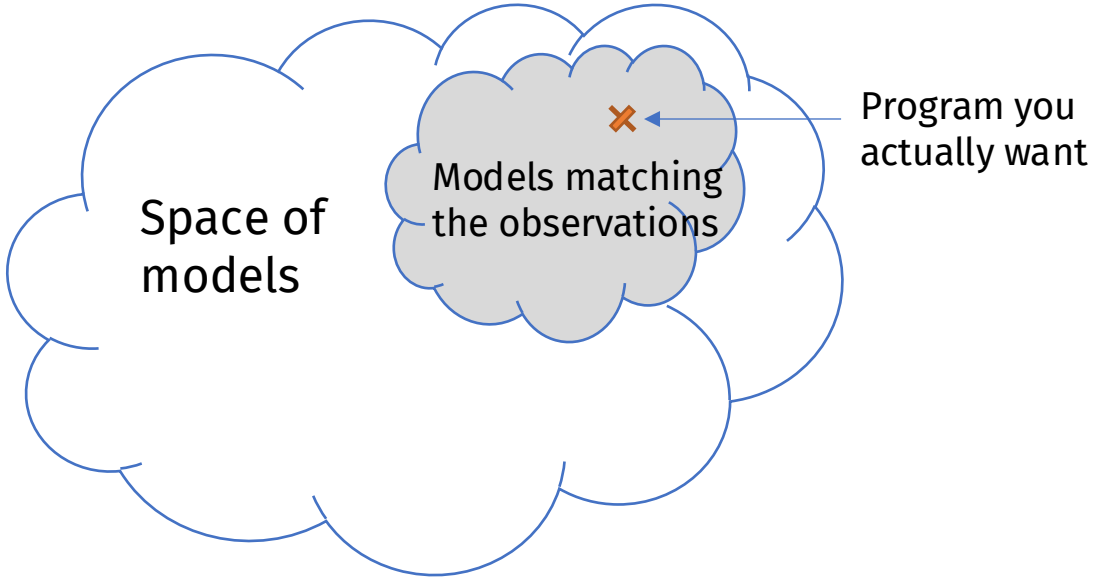
Key issues in inductive learning



- (1) How do you find a model that matches the observations?

- (2) How do you know it is the model you are looking for?

Key issues in inductive learning



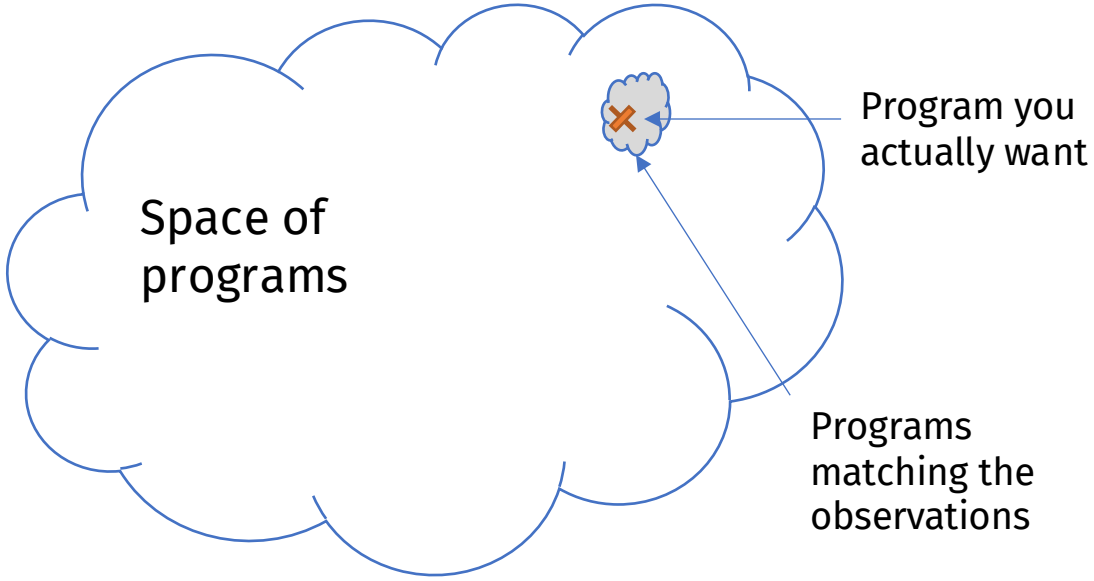
Traditional ML:

- Fix the space so that (1) is easy
- (2) becomes the main challenge

- (1) How do you find a model that matches the observations?

- (2) How do you know it is the model you are looking for?

The synthesis approach



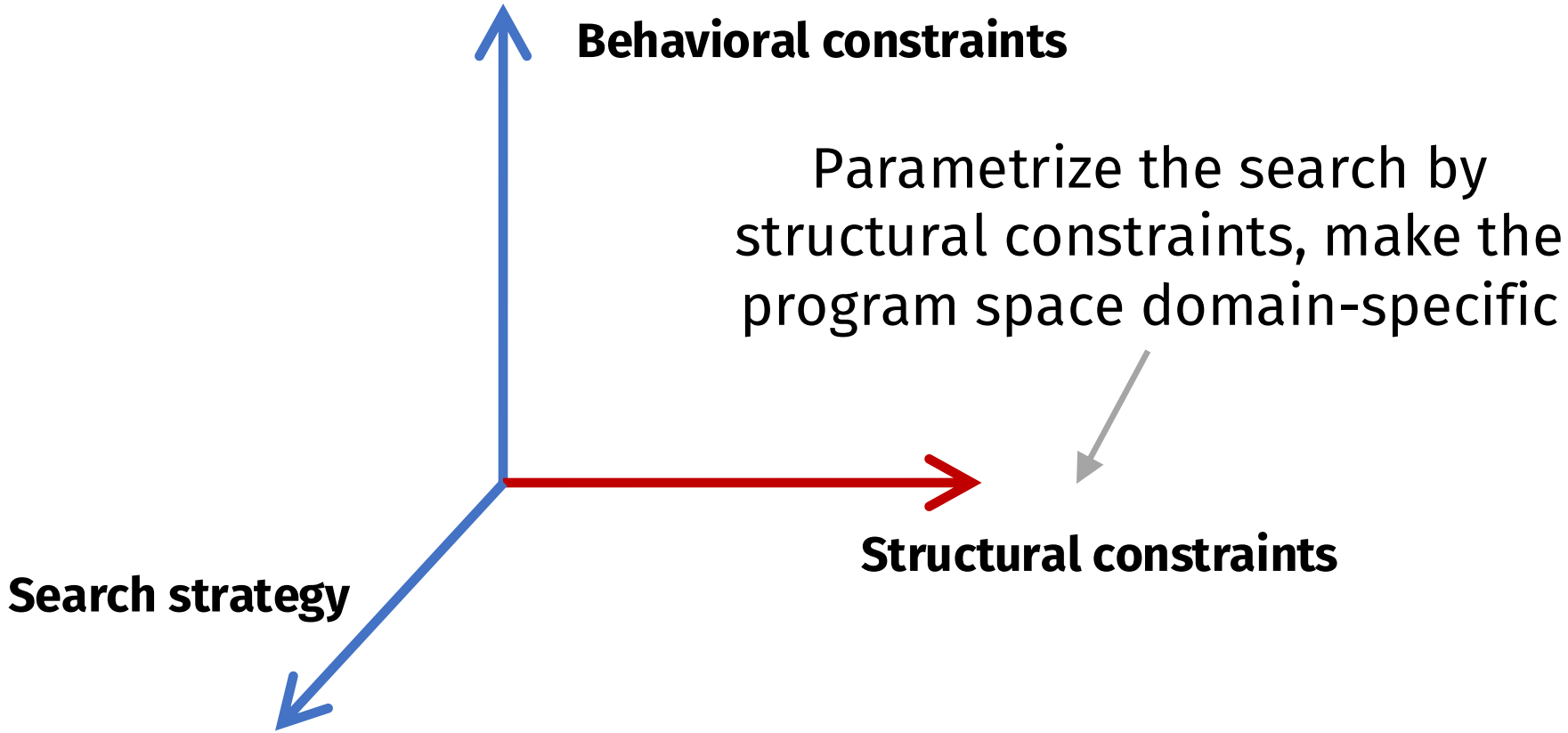
Program synthesis:

- Customize the space so that (2) becomes easier
- (1) is now the main challenge

- (1) How do you find a program that matches the observations?

- (2) How do you know it is the program you are looking for?

Key idea

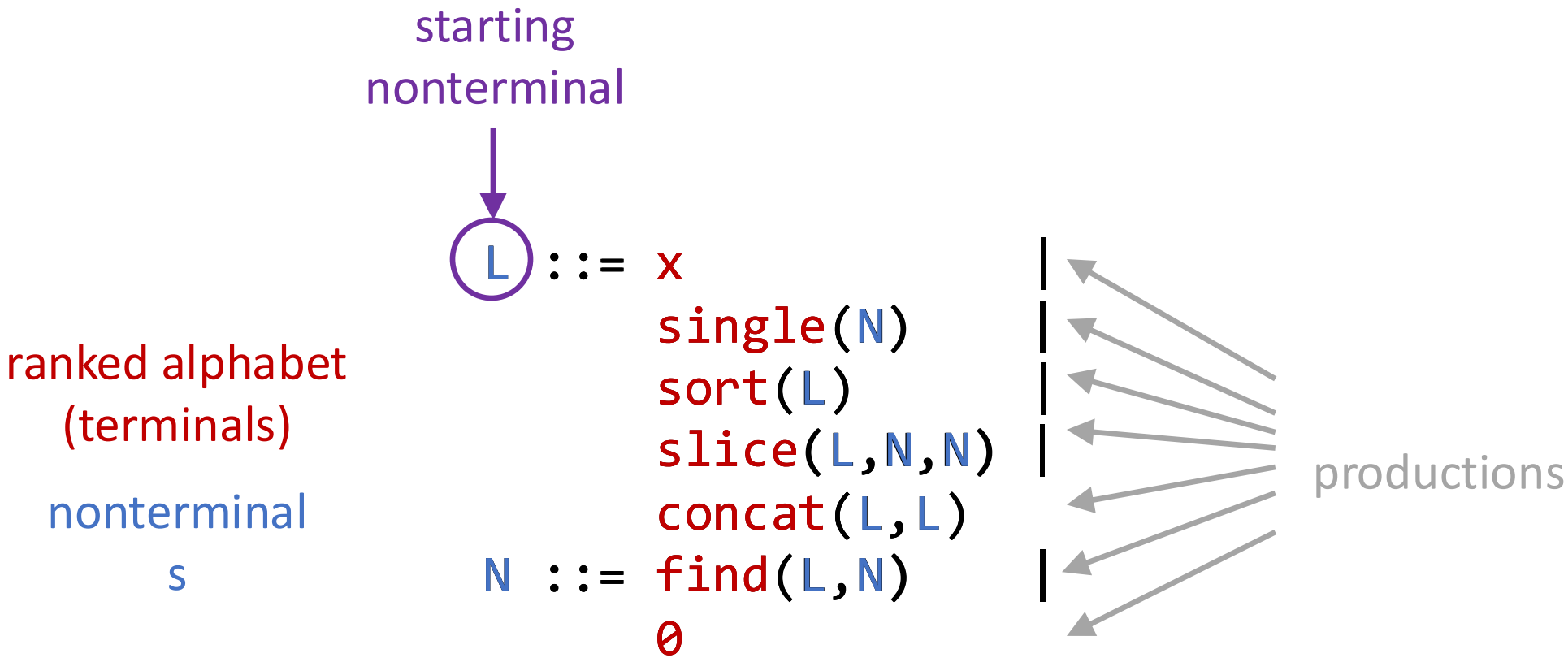

Behavioral constraints
Parametrize the search by structural constraints, make the program space domain-specific
Structural constraints
Search strategy

Syntax-Guided Synthesis

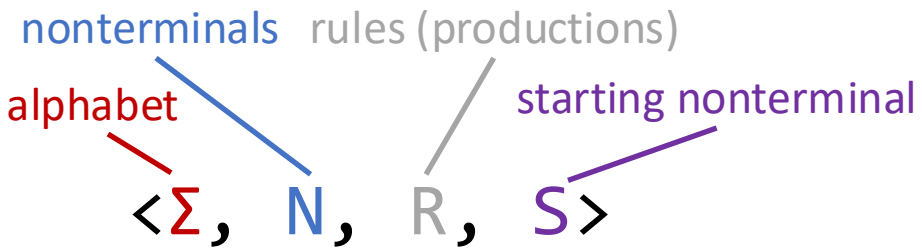
Example

```
[1,4,7,2,0,6,9,2,5,0]  →  [1,2,4,7,0]

L ::= x              |      the input
      single(N)      |      single(1) = [1]
      sort(L)        |      sort([6,9,2,5]) = [2,5,6,9]
      slice(L,N,N)   |      slice([6,9,2,5],0,2) = [6,9]
      concat(L,L)            concat([6,9],[2,5]) = [6,9,2,5]
N ::= find(L,N)      |      find([6,9],9) = 1
      0                     0

f(x) := concat( sort(slice(x,0,find(x,0))),  single(0))
```

Regular tree grammars (RTGs)



Regular tree grammars (RTGs)



- Trees: $\tau \in T_\Sigma(N)$ = all trees made from $N \cup \Sigma$
- Rules are of the form: $A \to \sigma(A_1, \ldots, A_n)$
- Derives in one step: $\mathcal{C}[A] \to \mathcal{C}[t]$ if $(A \to t) \in R$
  A is the leftmost non-terminal in $\mathcal{C}[A]$
- Incomplete terms/programs: $\{\tau \in T_\Sigma(N) | A \to^* \tau\}$
- Complete terms/programs: $\{t \in T_\Sigma | A \to^* t\}$
  = programs without holes
- Whole programs: $\{t \in T_\Sigma | S \to^* t\}$
  = roughly, programs of the right type

```
concat(L,0)
```

$L \to concat(L, L)$

```
concat(L,L) -> concat(x,L)
```

```
find(concat(L,L),N)
```

```
find(concat(x,x),0)
```

```
sort(concat(L,L))
```

RTGs as structural constraints

Space of programs
= the *language* of an RTG $L(G)$
= all complete, whole programs

```
L ::= x
      single(N)
      sort(L)
      slice(L,N,N)
      concat(L,L)
N ::= find(L,N)
      0
```

```
x   sort(x)   concat(x, x)   slice(x,0,0)

...

slice(x,0,find(x,0))

...

concat(sort(slice(x,0,find(x,0))), single(0))

...
```

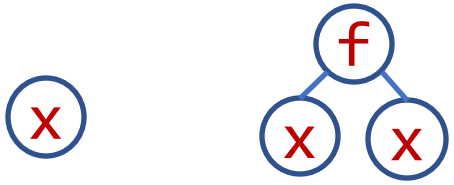
How big is the space?

```
E ::= x | f(E,E)
```

depth <= 0

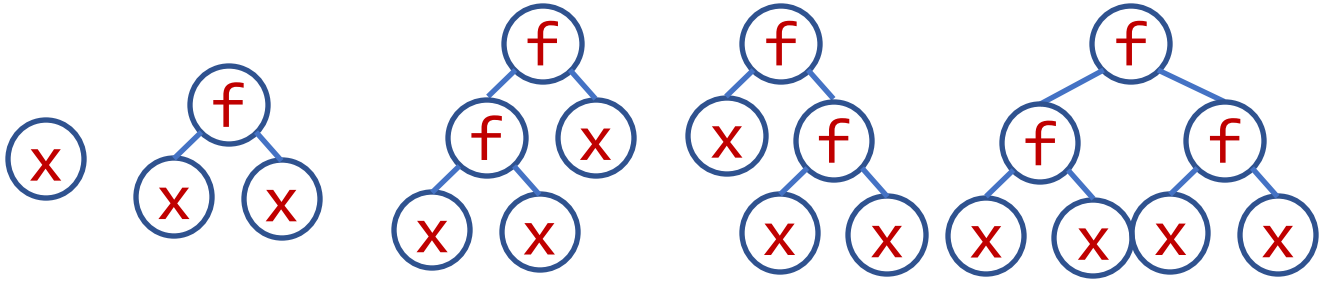
N(0) = 1

depth <= 1



N(1) = 2

depth <= 2



N(2) = 5

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

How big is the space?

$$E ::= x \mid f(E,E)$$

$$N(d) = 1 + N(d - 1)^2 \qquad N(d) \sim c^{2^d} \qquad (c > 1)$$

N(1) = 1
N(2) = 2
N(3) = 5
N(4) = 26
N(5) = 677
N(6) = 458330
N(7) = 210066388901
N(8) = 4412788774590617598780 2
N(9) = 19472704769152964495597034454938489304527912 05
N(10) = 37918623102659260828682350280278932773702331522473885847617341507177682544103411753253520 26

How big is the space?

$$E ::= x_1 \mid \ldots \mid x_k \mid f_1(E,E) \mid \ldots \mid f_m(E,E)$$

$$N(0) = k$$

$$N(d) = k + m * N(d - 1)^2$$

N(1) = 3
N(2) = 30
N(3) = 2703
N(4) = 21918630
N(5) = 1441279023230703
N(6) = 623185566841454795381868562263O
N(7) = 116508075215851596766492219468227024724121520304443212304350703

$k = m = 3$

Syntactic sugar

Instead of this:

```
L ::= x              |
      single(N)      |
      sort(L)        |
      slice(L,N,N)   |
      concat(L,L)
N ::= find(L,N)      |
      0
```

We will often write this:

```
L ::= x              |
      [N]            |
      sort(L)        |
      L[N..N]        |
      L + L
N ::= find(L,N)      |
      0
```

- allow custom syntax for terminal symbols
- not an RTG strictly speaking, but you know what we mean...

Syntactic sugar

```
L ::= sort(L)
      L[N..N]
      L + L
      [N]
      x
N ::= find(L,N)
      0
```

```
x   sort(x)   x + x   x[0..0]

...

x[0..find(x, 0)]

...

sort(x[0..find(x, 0)]) + [0]

...
```

The SyGuS project

https://sygus.org/

- Goal: Unify different syntax-guided approaches
- Collection of synthesis benchmarks + yearly competition
  - 6 competitions since 2013
  - consider writing a SyGuS solver for your project!
- Common input format + supporting tools
  - parser, baseline synthesizers

[Alur et al. 2013]

SyGuS problems

- SyGuS problem = < theory, spec, grammar >

A “library” of types and function symbols

Example: Linear Integer Arithmetic (LIA)

`True, False`
`0,1,2,...`
`∧, ∨, ¬, +, ≤, ite`

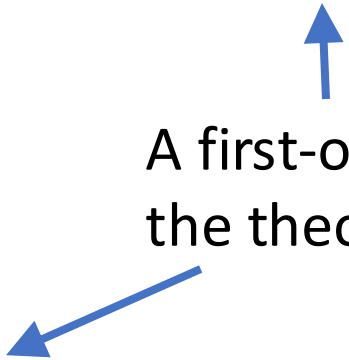
RTG with terminals in the theory
(+ input variables)

Example: Conditional LIA expressions w/o sums

```
E ::= x | 0 | ite C E E
C ::= E ≤ E | C ∧ C | ¬C
```

SyGuS problems

SyGuS problem = < theory, spec, grammar >



A first-order logic formula over the theory

Examples:

```
f(0, 1) = 1 ∧
f(1, 0) = 1 ∧
f(1, 1) = 1 ∧
f(2, 0) = 2
```

SyGuS demo

SyGuS problems

- SyGuS problem = < theory, spec, grammar >

A first-order logic formula over the theory

can inductive synthesis handle these?

Examples:

```
f(0, 1) = 1 ∧
f(1, 0) = 1 ∧
f(1, 1) = 1 ∧
f(2, 0) = 2
```

Formula with free variables:

```
x ≤ f(x, y) ∧
y ≤ f(x, y) ∧
(f(x, y) = x ∨ f(x, y) = y)
```