#7: Weighted Enumerative Search - I

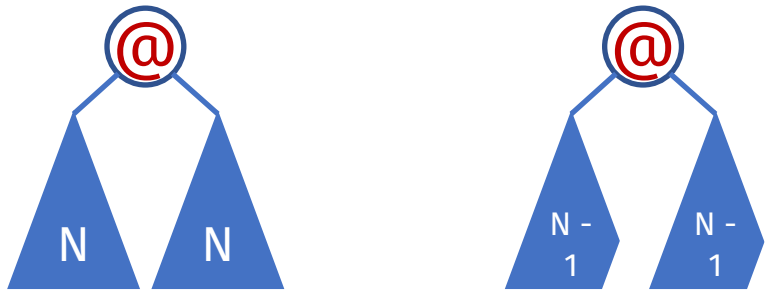
Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

Scaling enumerative search

Prune

- Discard useless subprograms



$m * N^2$ $m * (N - 1)^2$

Prioritize

- Explore more promising candidates first

```
P = { [0][N..N] ,
      x[N..N]   ,
      ... }
```

dequeue this first

Order of search

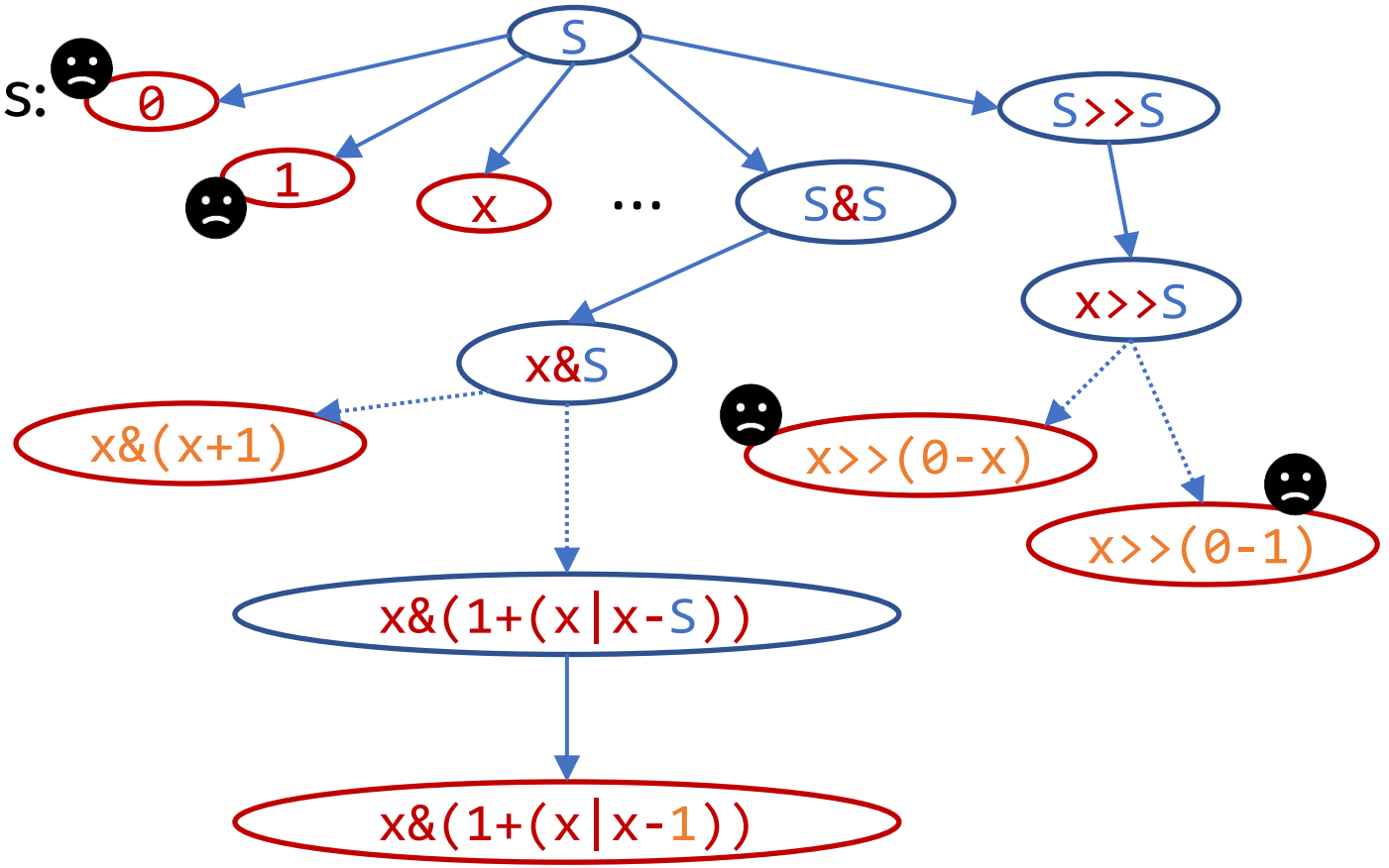
- Enumerative search explores programs by depth / size
  - Good default bias: small solution is likely to generalize
  - But far from perfect
- Result:
  - Scales poorly with the size of the smallest solution to a given spec

Top-down search (revisited)

Turn off the rightmost sequence of 1s:

00101 → 00100
01010 → 01000
10110 → 10000

```
S ->  0 | 1 | x |
      S + S     |
      S - S     |
      S & S     |
      S | S     |
      S << S    |
      S >> S
```



Explores many unlikely programs!

Biasing the search

- **Idea:** explore programs in the order of likelihood, not size
- **Q1:** how do we know which programs are likely?
  - hard-code domain knowledge
  - learn from a corpus of programs
  - learn on the fly
- **Q2:** how do we use this information to guide search?
  - our focus today!

Weighted enumerative search

Example: DeepCoder

Balog et al. DeepCoder: Learning to Write Programs. ICLR'17

Probabilistic Grammars

Weighted top-down search

Weighted bottom-up search

DeepCoder

Input: IO-examples

```
[-17 -3 4 11 0 -5 -9 13 6 6 -8 11]
→ [-12 -20 -32 -36 -68]
```

DeepCoder

Output: Program in a list DSL

```
a <- [int]
b <- Filter (<0) a
c <- Map (*4) b
d <- Sort c
e <- Reverse d
```

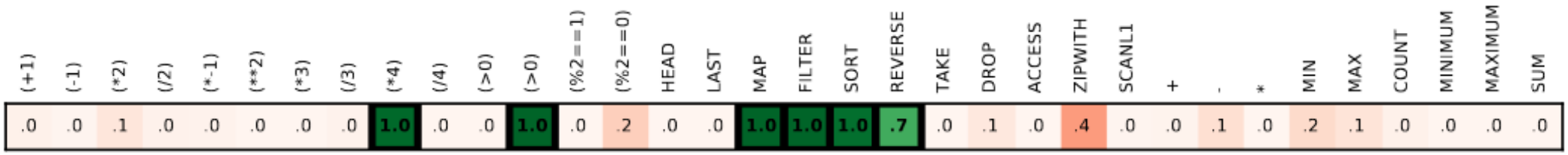
DeepCoder

Input: IO-examples

```
[-17 -3 4 11 0 -5 -9 13 6 6 -8 11]
→ [-12 -20 -32 -36 -68]
```

neural network

component weights

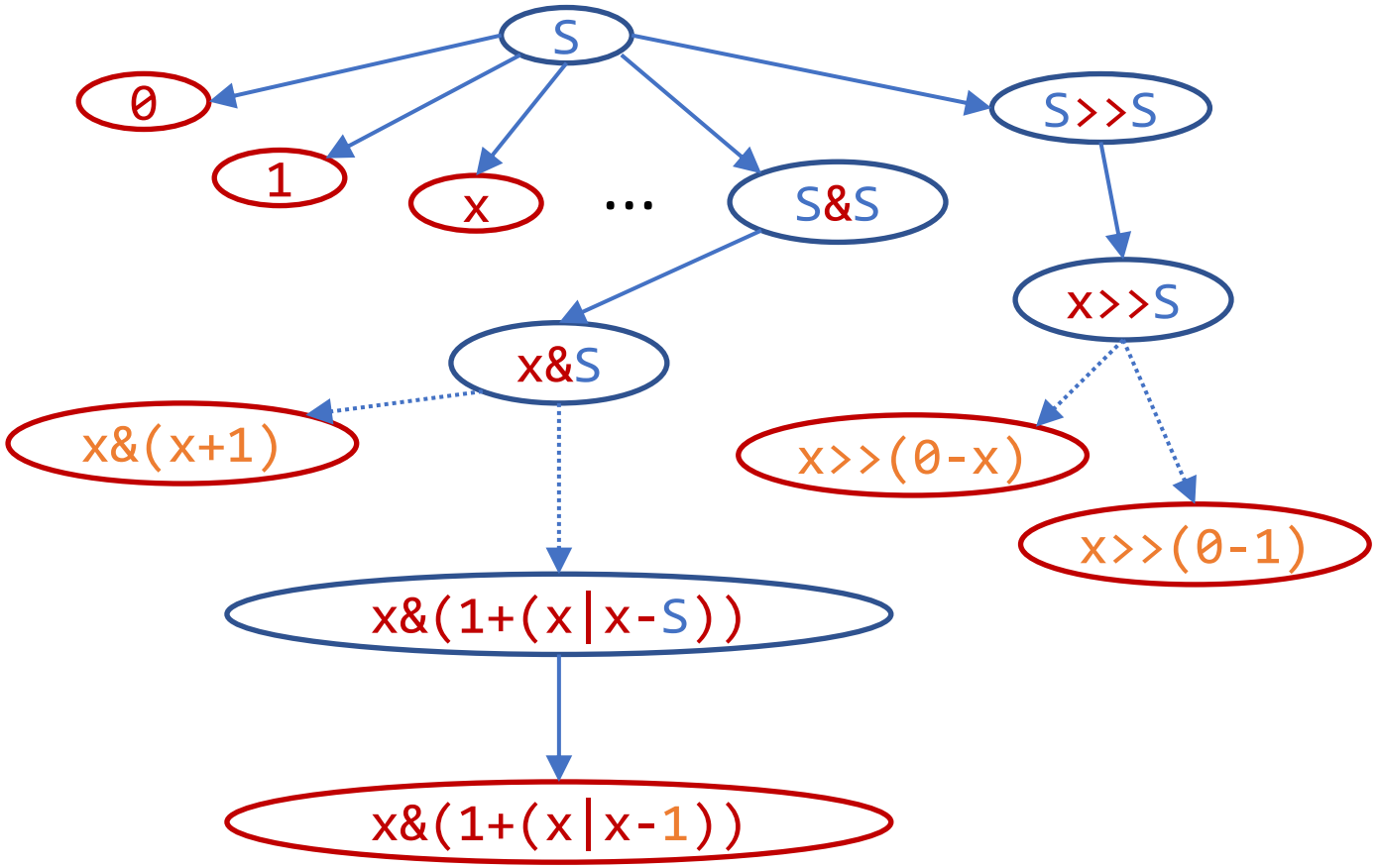


weighted search

Output: Program in a list DSL
Goal: Minimize sum of component weights

DeepCoder: search strategies

- Top-down DFS
  - Picks expansions for the current non-terminal in the order of probability
- Sort-and-add
  - start with N most probable functions
  - when search fails, add next N functions
- Pros and cons?



Recall: goal is to explore programs in the order of total weight!

Weighted enumerative search

DeepCoder

Probabilistic Grammars

Weighted top-down search

Weighted bottom-up search

Probabilistic Language Models

- Originated in Natural Language Processing
- In general: a probability distribution over sentences in a language
  - $P(s)$ for $s \in L$
- In practice:
  - must be in a form that can be used to guide search
  - for enumerative search: probabilistic (or weighted) grammars

Probabilistic (Tree) Grammar

regular tree grammar

production probability (given context)

$\langle G, \wp \rangle$

- Production probability: $\wp: R \times T_\Sigma(N) \rightarrow [0,1]$
  - for example: $\wp(S \rightarrow x \mid S) = 0.3 \quad \wp(S \rightarrow x \mid x\ -S) = 0.0001$
  - only defined for contexts where rule's LHS is the leftmost non-terminal
  - probabilities of all productions in the same context add up to 1:

$$\forall \tau. S \rightarrow^* \tau \wedge \tau \notin T_\Sigma \Rightarrow \sum_{r \in dom(P(.|\tau))} P(r \mid \tau) = 1$$

- Term probability:
  - let $S = \tau_0 \rightarrow^{r_1} \tau_1 \rightarrow^{r_2} \ldots \rightarrow^{r_n} \tau_n = \tau$ be the unique derivation of partial program τ

$$\wp(\tau) = \prod_{i=1}^{n} \wp(r_i \mid \tau_i)$$

Types of context

$$\wp : \mathrm{R} \times T_{\Sigma}(N) \rightarrow [0,1]$$

- In general, can depend on any part of the context term!
- But this is unwieldy
  - bad for learning
  - bad for (some) search algorithms
- In practice we want to restrict the context
  - PCFG
  - n-grams
  - PHOG

Probabilistic Context-Free Grammars (PCFG)

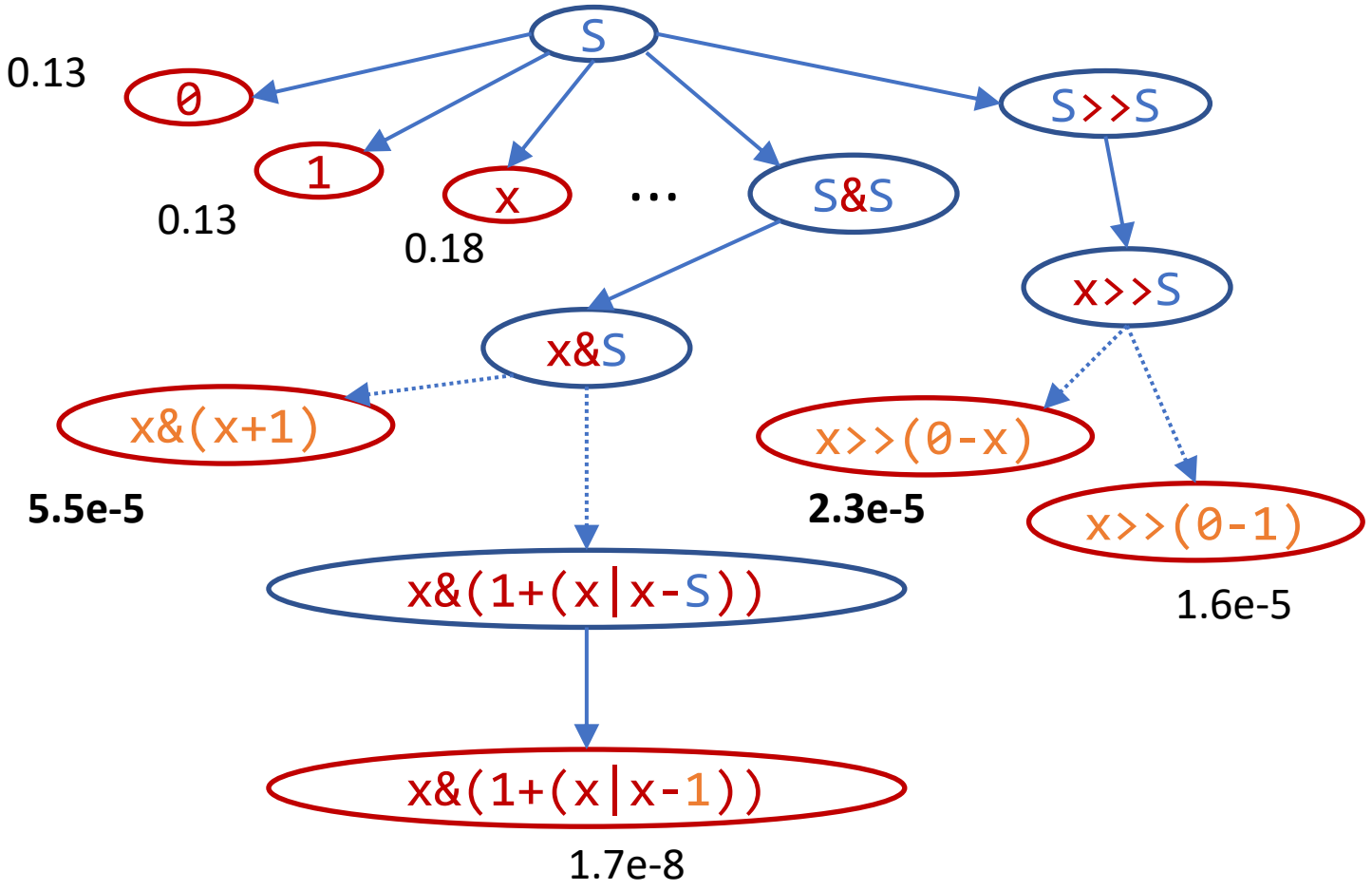
	$\wp(R)$
`S -> 0`	0.13
`S -> 1`	0.13
`S -> x`	0.18
`S -> S + S`	0.11
`S -> S - S`	0.11
`S -> S & S`	0.12
`S -> S | S`	0.12
`S -> S << S`	0.05
`S -> S >> S`	0.05

$$\wp: R \rightarrow [0,1]$$

- Encodes the popularity of each production (operation)
  - here: variable more likely than constant, plus more likely than shift

Probabilistic Context-Free Grammars (PCFG)

		$\wp(R)$
S ->	0	0.13
S ->	1	0.13
S ->	x	0.18
S ->	S + S	0.11
S ->	S - S	0.11
S ->	S & S	0.12
S ->	S | S	0.12
S ->	S << S	0.05
S ->	S >> S	0.05

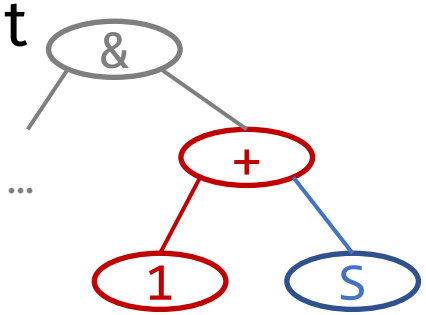


N-grams

`N[left sibling, parent] -> rhs`

		℘
`S[x,-] ->`	`1`	**0.72**
`S[x,-] ->`	`x`	0.02
`S[x,-] ->`	`S + S`	0.12
`S[x,-] ->`	`S - S`	0.12
`...`		
`S[1,+] ->`	`1`	0.26
`S[1,+] ->`	`x`	0.25
`S[1,+] ->`	`S + S`	0.19
`S[1,+] ->`	`S - S`	0.08

- Encodes likelihood of a production in a fixed context
  - fixed set of AST nodes determined relative to the focus nonterminal
  - e.g. left sibling and parent

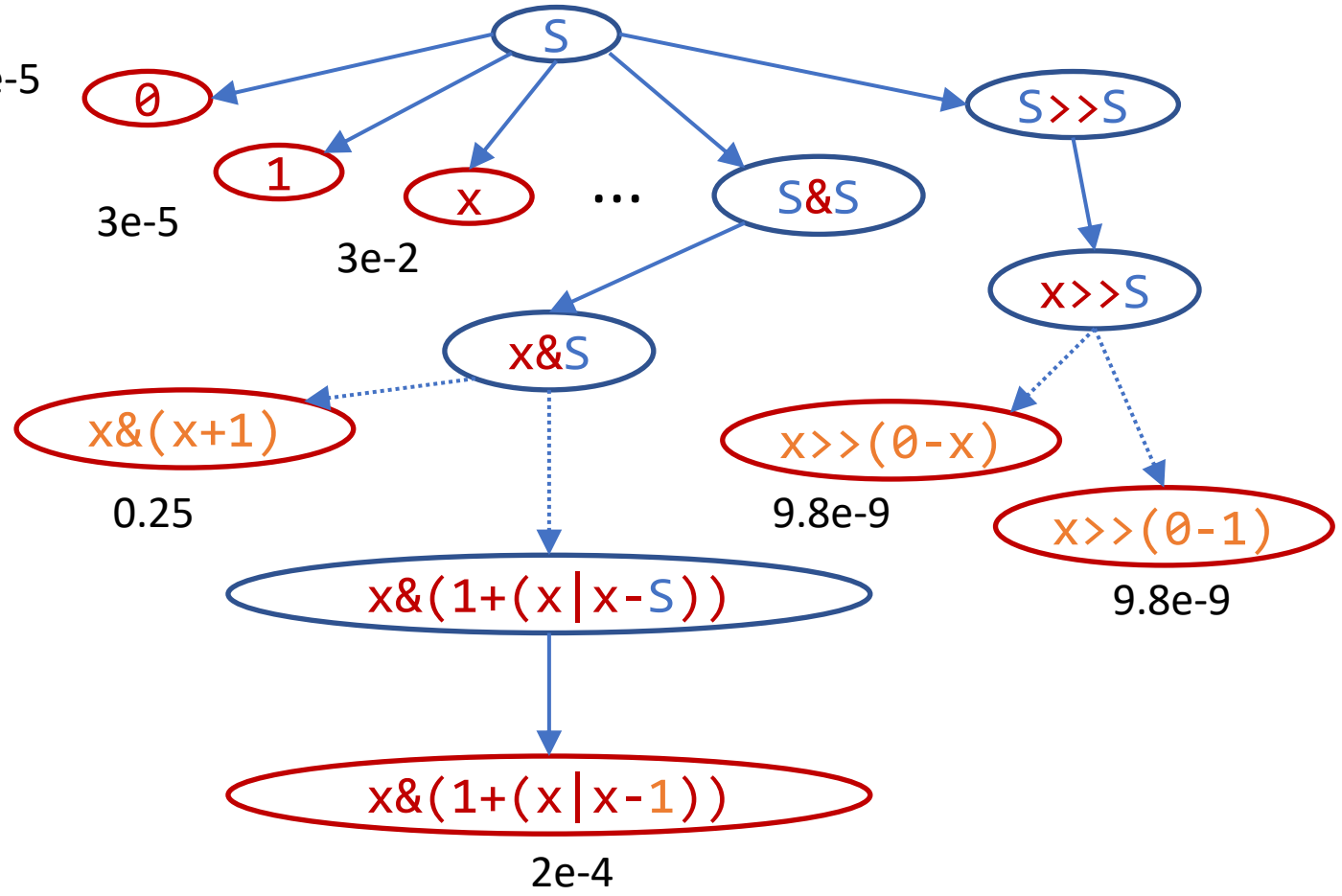


  - here: x is not likely in x - S? but likely in 1 + S?

N-grams

`N[left sibling, parent] -> rhs`

		℘
`S[x,-] -> 1`		**0.72**
`S[x,-] -> x`		0.02
`S[x,-] -> S + S`		0.12
`S[x,-] -> S - S`		0.12
...		
`S[1,+] -> 1`		0.26
`S[1,+] -> x`		0.25
`S[1,+] -> S + S`		0.19
`S[1,+] -> S - S`		0.08



Probabilistic Higher-Order Grammar (PHOG)

- The same fixed context might not work for every problem
- Idea:

1. 1. define context as a program that traverses the AST
2. 2. learn the best context together with probabilities

Bielik, Raychev, Vechev. PHOG: Probabilistic Model for Code. ICML'16

Conditional models

- **Unconditional model**
- Which programs are more natural in this DSL?

+ easier to get data / learn

- need more context to capture interesting properties

- **Conditional model**
- Which programs are more likely to solve a given spec?

- harder to get data / learn

- can get away with less context