Lecture 4
Probabilistic Models and Stochastic Search

Nadia Polikarpova

Announcement

Consider applying to the *Programming Languages Mentoring Workshop* (Jan 15, Lisbon)

https://popl19.sigplan.org/track/PLMW-2019-papers

Deadline: October 30

Today

Topics:

- Go over suggested project topics
- Discuss EUSolver
- Search space prioritization
- Stochastic Search (maybe)

Feedback on reviews

More discussion of the technique/eval and less of the writing:

- good: “A major weakness of the this work is its restrictive scope: it only applies to synthesis of conditional expressions.”

Substantiate your evaluations:

- bad: “This technique is fantastic.”
- good: “This technique significantly reduces synthesis times as compared to ESolver.”

Be concrete (avoid vague sentences):

- bad: “This paper’s core contributions are in combining various methodologies in computer science.”
- good: “This paper presents a divide and conquer variation of the enumerative algorithm for solving Syntax-Guided Synthesis (SyGus) problems.”

EUSover

Q1: What does EUSolver use as behavioral constraints? Structural constraint? Search strategy?

- First-order formula
- Conditional expression grammar
- Bottom-up enumerative + pruning

Why do they need the specification to be pointwise?

- Example of a non-pointwise spec?
- How would it break the enumerative solver?

EUSover

Q2: What are pruning/decomposition techniques EUSolver uses to speed up the search?

- Condition abduction + special form of equivalence reduction

Why does EUSolver keep generating additional terms when all inputs are covered?

How is the EUSolver equivalence reduction differ from observational equivalence we saw in class?

- How do they overcome the problem that it's not robust to adding new points?

Branch-wise verification: are more counter-examples always better?

EUSover

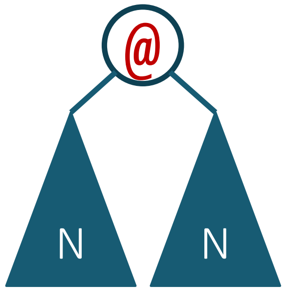
Q3: What would be a naive alternative to decision tree learning for synthesizing branch conditions?

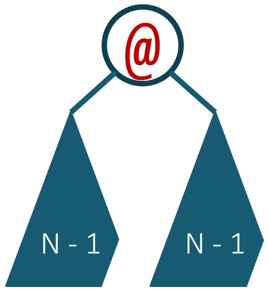
- Learn atomic predicates that precisely classify points
  - why is this worse?
  - is it as bad as ESolver?
- Next best thing is decision tree learning w/o heuristics
  - why is this worse?

Scaling enumerative search

Prune

Discard useless subprograms





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

Prioritize

Explore more promising candidates first

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

dequeue this first

Enumerative search

Explores smaller programs before larger programs

- Small solution is likely to generalize
- Scales poorly with the size of the smallest solution

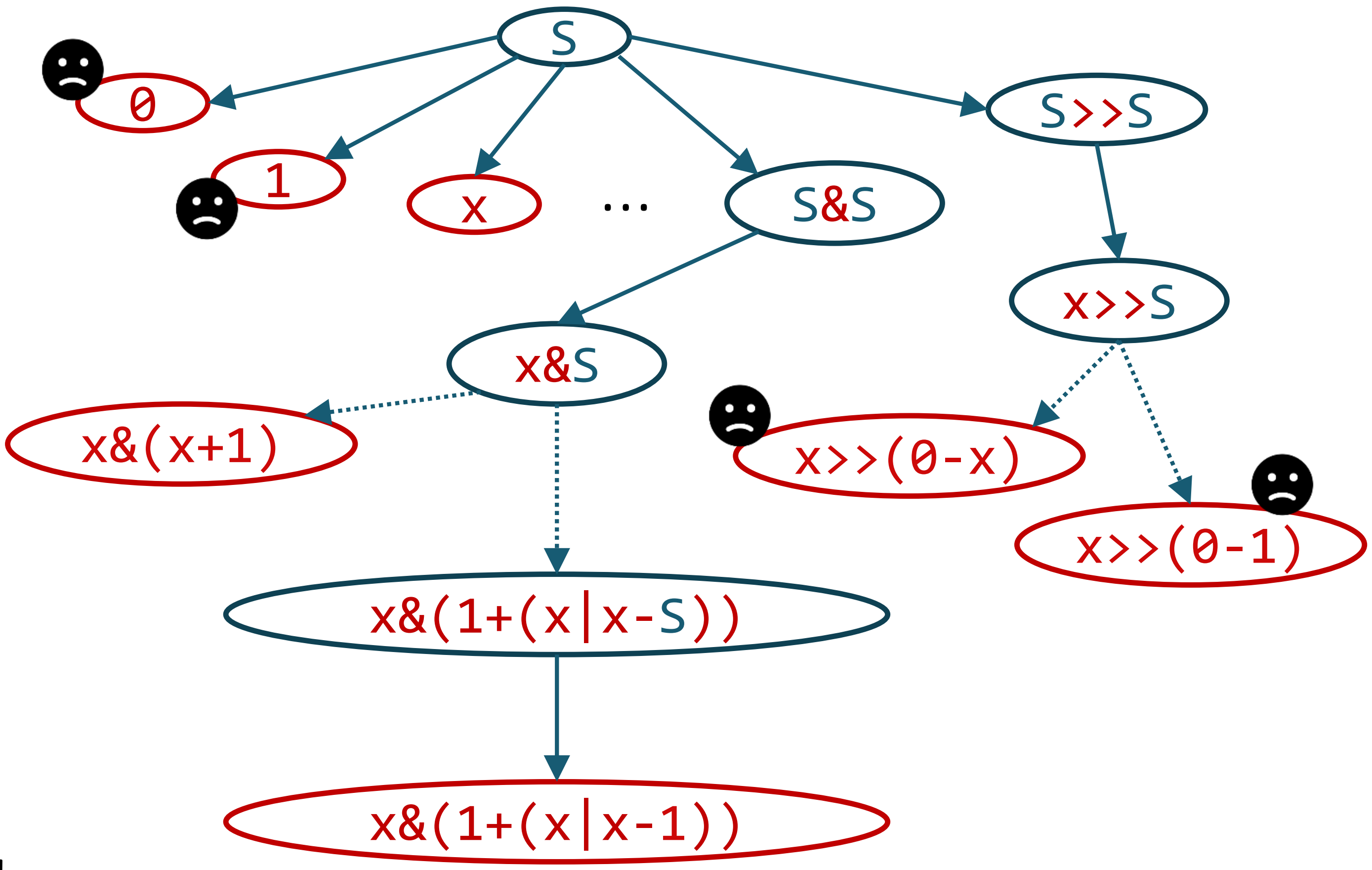
Top-down search (revisited)

Turn off the rightmost sequence of **1**s:

```
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!



Weighted top-down search

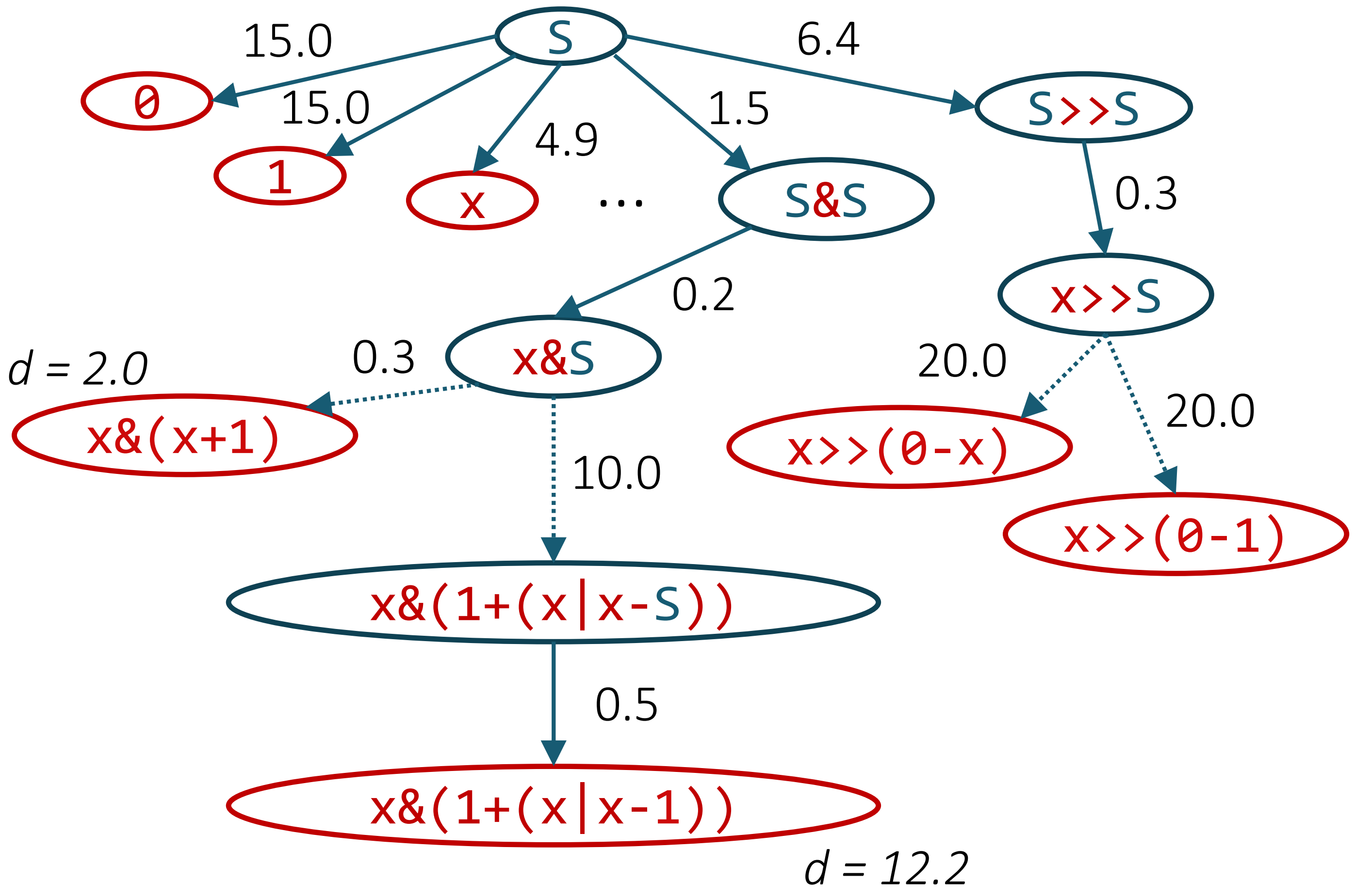
[Lee, Kihong, Alur, Naik PLDI'16]

Idea: explore programs in the order of likelihood, not size

1. Assign weights *w(e)* to edges such that *d(p)* < *d(p')* iff *p* is more likely than *p'*

$$d(p) = \sum_{e \in S \to p} w(e)$$

2. Use Dijkstra's algorithm to find closest leaves



Weighted top-down search (Dijkstra)

```
top-down(<T, N, R, S>, [i → o]) {
  P := [<S,0>]
  while (P != [])
    <p,d> := P.dequeue_min(d);
    if (ground(p) && p([i]) = [o])
      return p;
    P.enqueue(unroll(p,d));
}

unroll(p,d) {
  P' := []
  N := leftmost nonterminal in p
  forall (N ::= rhs in R)
    P' += <p[N -> rhs], d + w(rhs, p)>
  return P';
}
```

P now stores candidates (nodes) together with their distances

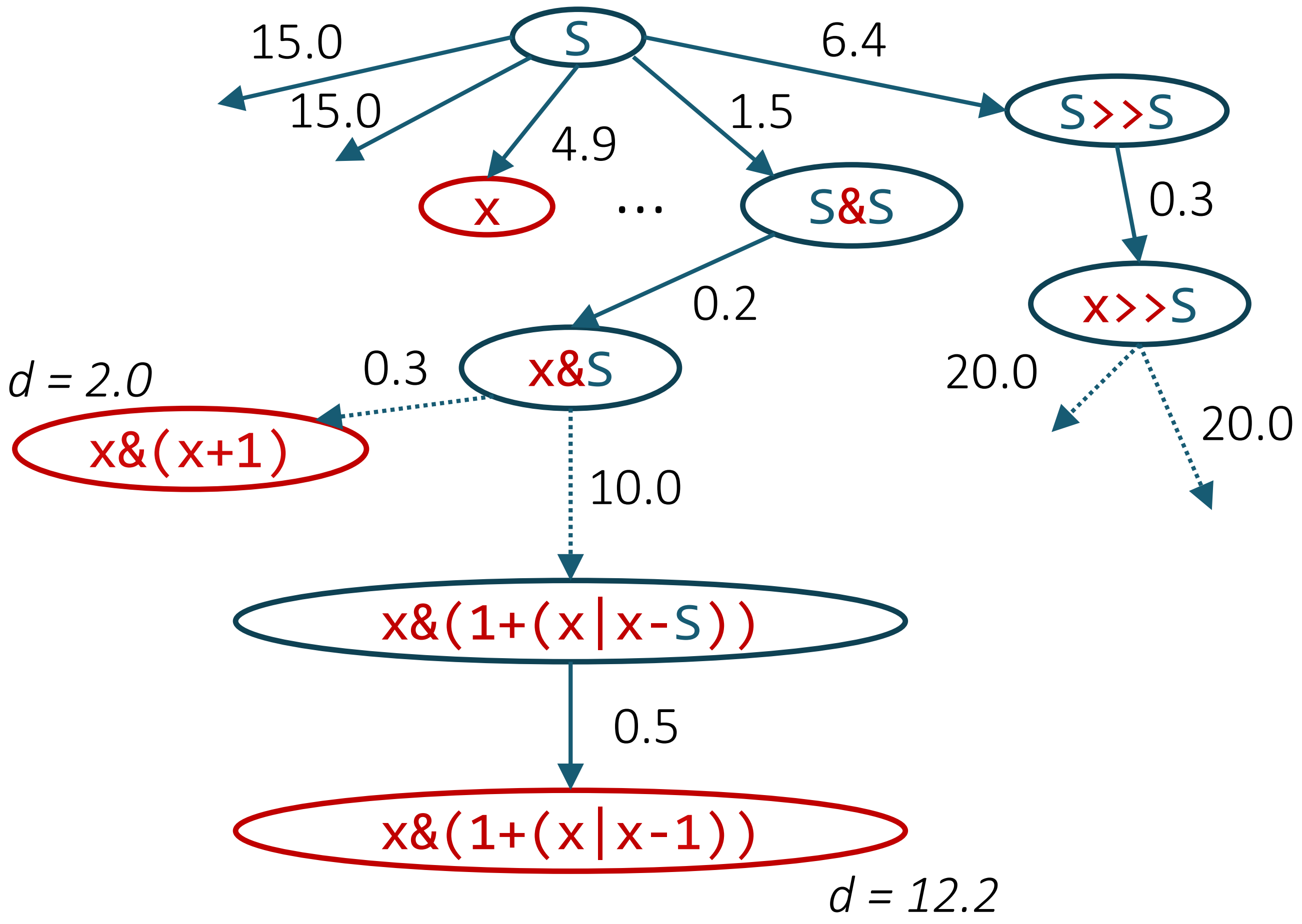
Dequeue the node with the shortest distance from the root

Distance to a new node: add the *w(e)*

Can we do better?

Dijkstra: explores a lot of intermediate nodes that don't lead to any cheap leaves

A*: introduce heuristic function *h(p)* that estimates how close we are to the closest leaf



Weighted top-down search (A*)

[Lee, Kihong, Alur, Naik PLDI'16]

```
top-down(<T, N, R, S>, [i → o]) {
  P := [<S,0,h(S)>]
  while (P != [])
    <p,d,h> := P.dequeue_min(d + h);
    if (ground(p) && p([i]) = [o])
      return p;
    P.enqueue(unroll(p,d));
}

unroll(p,d) {
  P' := []
  N := leftmost nonterminal in p
  forall (N ::= rhs in R)
    P' += <p[N -> rhs], d + w(rhs, p),
                        h(p[N -> rhs])>
  return P';
}
```

Roughly how close is this program to the closest leaf

So, where do these come from?

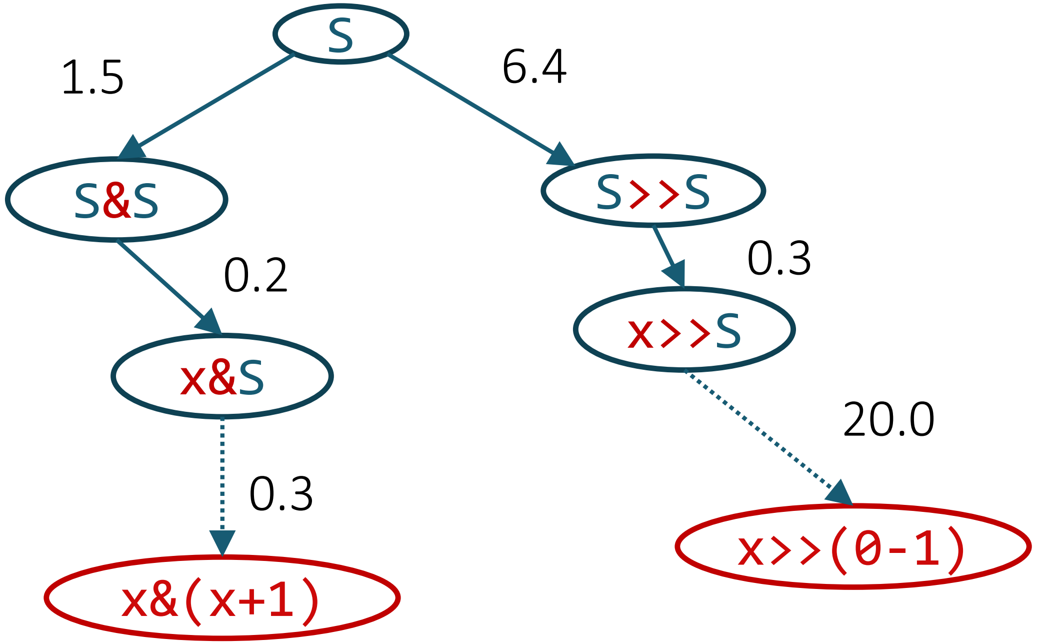
Assigning weights to edges

$$d(p) = \sum_{e \in S \to p} w(e)$$

$$2^{-d(p)} = \prod_{e \in S \to p} 2^{-w(e)}$$

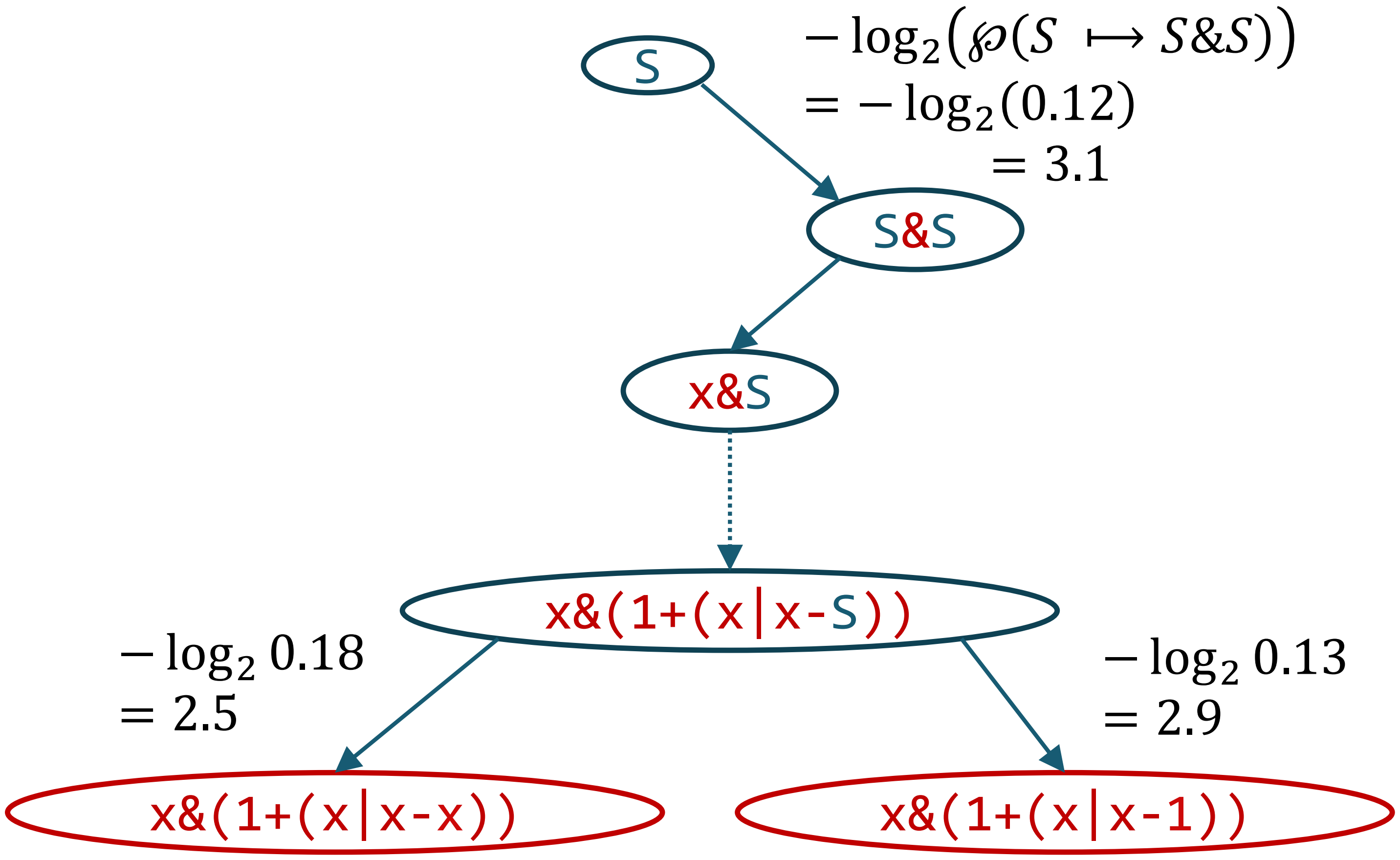
$$\wp(p) = \prod_{e \in S \to p} \wp(e)$$

So, we should decide what is the probability of taking each edge $\wp(e)$ and then set $w(e) = -\log_2 \wp(e)$



Probabilistic CFG (PCFG)

	$\wp$
`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

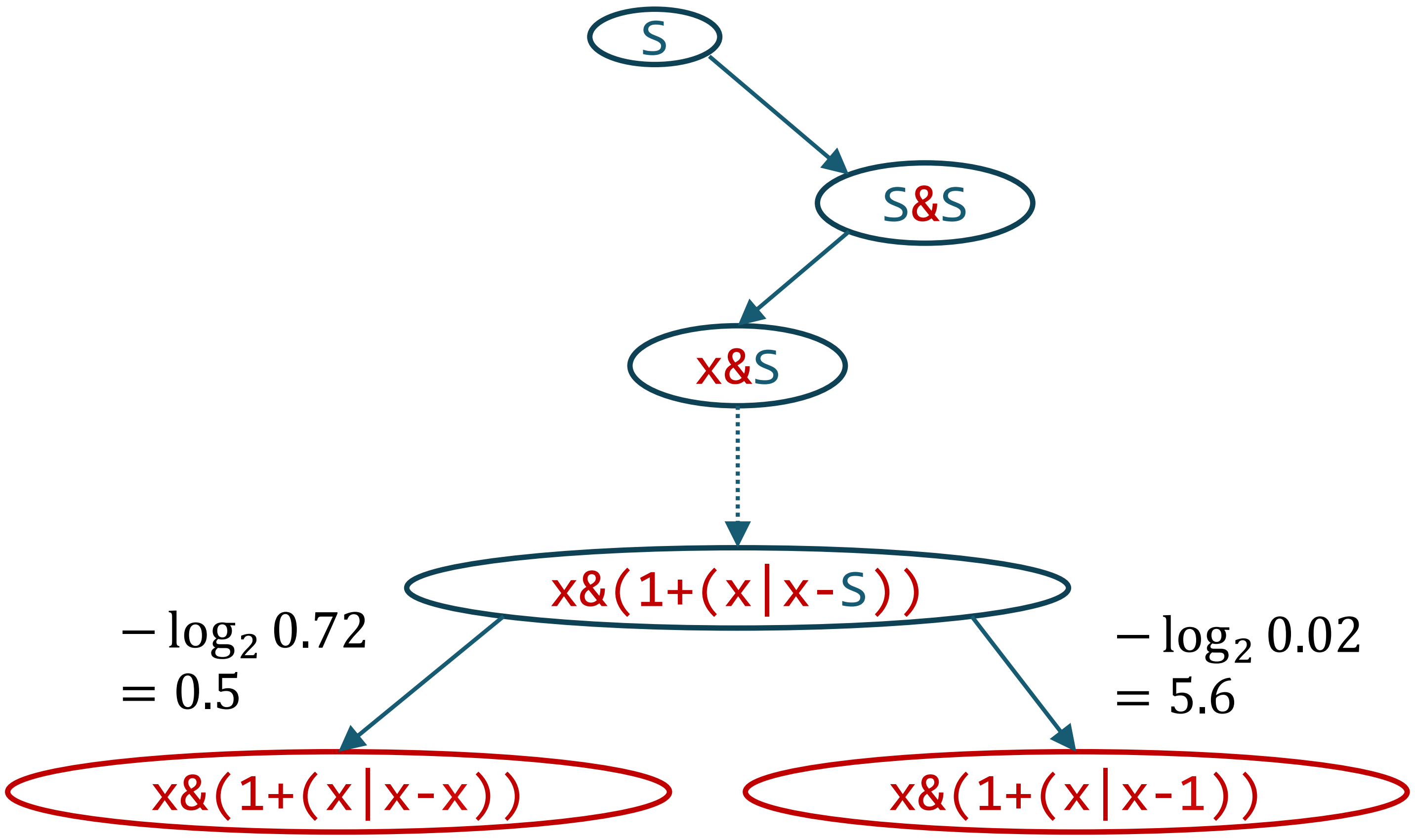


Probabilistic Higher-Order Grammar (PHOG)

[Bielik, Raychev, Vechev '16]

```
N[context] -> rhs
```

Rule	$\wp$
`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



Learning PHOGs

[Bielik, Raychev, Vechev '16]

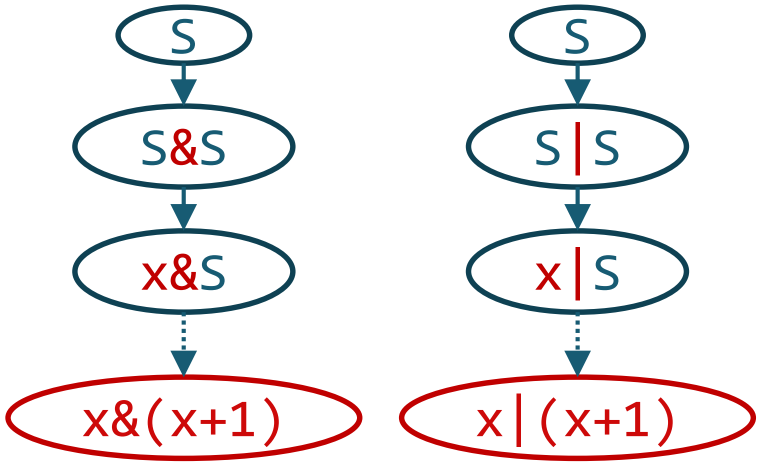
CFG +

Corpus

```
x&(x+1)
x|(x-1)
x
x&(x+x)
x&(1+(x|x-1))
...
```

parse

ASTs / Paths



...

learn

`context`, $\wp$

PHOGs useful for:

- code completion
- deobfuscation
- programming language translation
- statistical bug detection

Probabilistic models: overview

Learn natural programs

Learn solutions for particular problem

- useful for MOOCs

Learn mapping from spec to code

- or features of code

sk_p

[Pu et al. ‘16]

Program corrections for MOOCs

Treats programs as text

- Modulo concrete variable names etc.
- Uses the skipgram model to predict which statement is most likely to occur between the two

Features

- Can repair syntax errors

Limitations

- Needs all algorithmically distinct solutions to appear in the training set

DeepCoder

[Balog et al. ICLR‘17]

Neural programming: neural nets that write programs

Predicts likely components from IO examples:

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

`*4`	(1.0)	`filter`	(1.0)
`>0`	(1.0)	`sort`	(1.0)
`map`	(1.0)	`reverse`	(0.7)

Features

- Can be combined with any enumerative search
- Significant speedups for a small list DSL

Limitations

- Unclear whether it scales to larger DSLs or more complex data structures

Next week

Topics:

- Stochastic Search
- Representation-Based Search

Paper: Gulwani: Automating string processing in spreadsheets using input-output examples

- Review due Wednesday
- Link to PDF on the course wiki
- Submit through EasyChair

Project: come talk to me about the topic!

- Tuesday 5-6pm