Lecture 4
Search Bias

Nadia Polikarpova

Plan for this week

Today:

- Search space prioritization/biasing

Tomorrow:

- Discuss the Euphony paper
- Synthesis framework demos

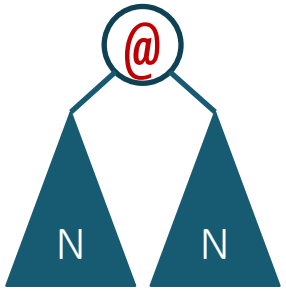
Project:

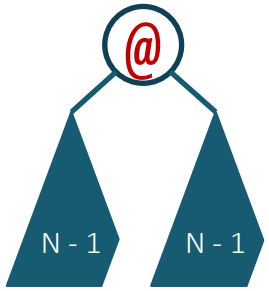
- Proposals due in ten days
- Talk to me about the topic: Thursday or next week after class

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
- If spec is insufficient: plays monkey's paw

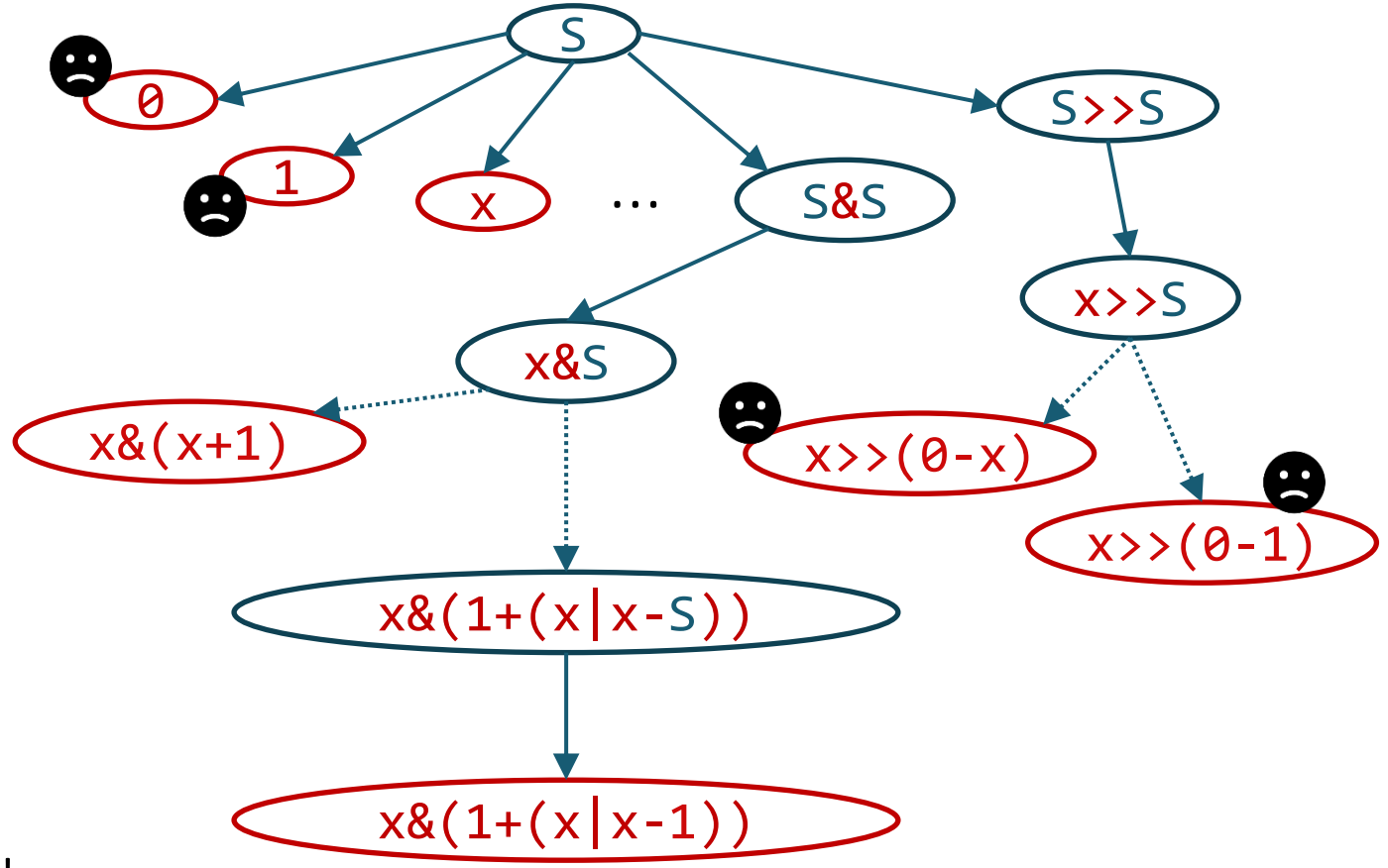
Top-down search (revisited)

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!



Biasing the search

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

Q1: how do we know which programs are likely?

- learn a statistical (probabilistic) model from a corpus of programs!

Q2: how do we use this information to guide search?

Statistical 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
- and also that can be learned from the data we have

Statistical Models in Synthesis

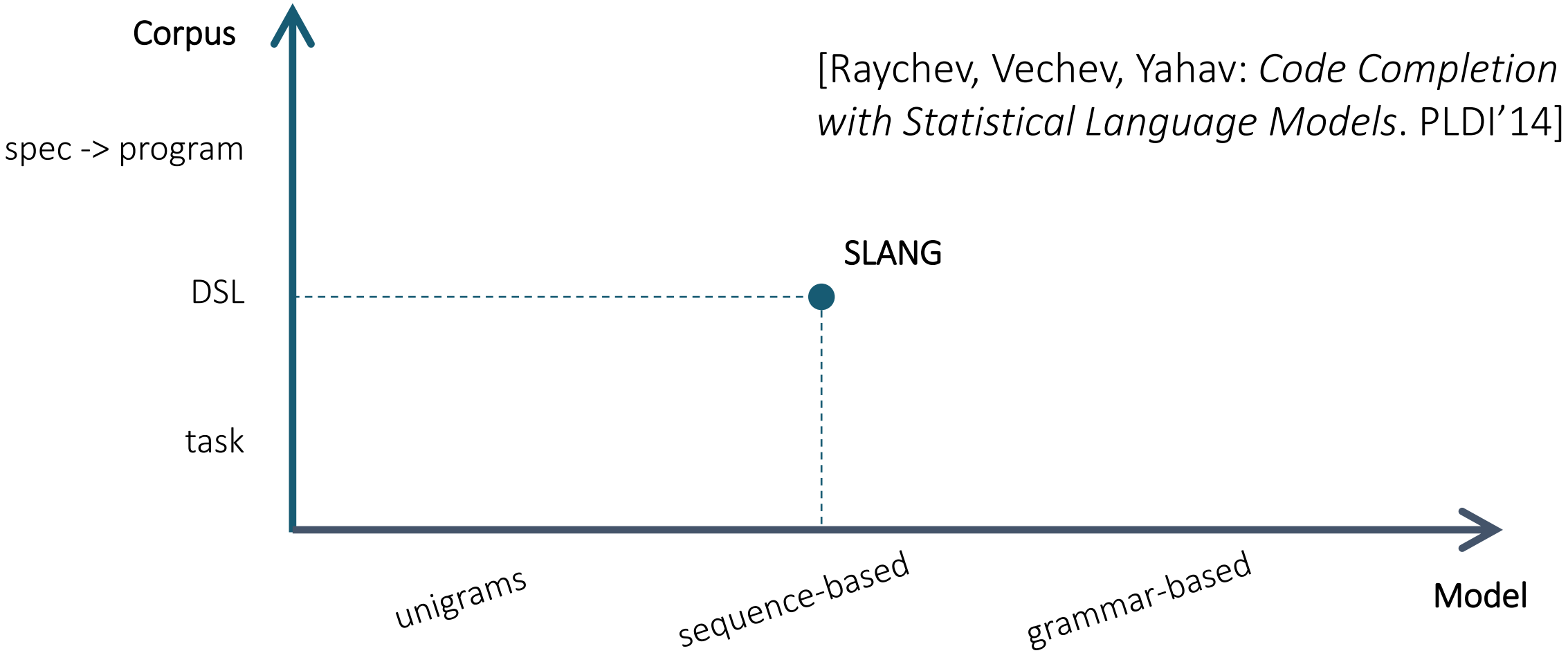
Kinds of corpora:

- All programs from DSL: what are natural programs in this DSL?
- Solutions to specific task (e.g. for MOOCs)
- Spec-program pairs: what are likely programs for this spec?

Kinds of models:

- Likely components (aka unigrams)
- Sequence-based: n-grams, RNN (LSTM)
- Grammar-based: PCFG, PHOG

Statistical Models in Synthesis



[Raychev, Vechev, Yahav: *Code Completion with Statistical Language Models*. PLDI'14]

SLANG

Input: code snippet with holes

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  ? {smsMgr, msgList}  // (H1)
} else {
  ? {smsMgr, message}  // (H2)
}
```

SLANG

Output: holes completed with (sequences) of method calls

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  smsMgr.sendMultipartTextMessage(...msgList...);
} else {
  smsMgr.sendTextMessage(...message...);
}
```

SLANG: inference phase

code snippet with holes

```
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  ArrayList<String> msgList =
      smsMgr.divideMsg(message);
  ? {smsMgr, msgList}  // (H1)
} else {
  ? {smsMgr, message}  // (H2)
}
```

static analysis

abstract histories of objects

$$\begin{aligned}
\textbf{smsMgr} &\mapsto \{\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \textbf{H2}\rangle\ , \\
&\quad\ \langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \textbf{H1}\rangle\} \\
\textbf{message} &\mapsto \{\langle \texttt{length}, 0\rangle\ ,\ \ \langle \texttt{length}, 0\rangle \cdot \langle \textbf{H2}\rangle\} \\
\textbf{msgList} &\mapsto \{\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \textbf{H1}\rangle\}
\end{aligned}$$

learned generative model:

- bigrams suggest candidates
- n-grams / RNNs rank them

Partial History	Id	Candidate Completions	Pr
$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \textbf{H2}, \textbf{smsMgr}\rangle$	11	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{sendTextMessage}, 0\rangle$	0.0073
	12	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 0\rangle$	0.0010
$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \textbf{H1}, \textbf{smsMgr}\rangle$	21	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 0\rangle$	0.0033
	22	$\langle \texttt{getDefault}, \texttt{ret}\rangle \cdot \langle \texttt{divideMsg}, 0\rangle \cdot \langle \texttt{sendTextMessage}, 0\rangle$	0.0016
$\langle \texttt{length}, 0\rangle \cdot \langle \textbf{H2}, \textbf{message}\rangle$	31	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{length}, 0\rangle$	0.0132
	32	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{split}, 0\rangle$	0.0080
	33	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{sendTextMessage}, 3\rangle$	0.0017
	34	$\langle \texttt{length}, 0\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 1\rangle$	0.0001
$\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \textbf{H1}, \textbf{msgList}\rangle$	41	$\langle \texttt{divideMsg}, \texttt{ret}\rangle \cdot \langle \texttt{sendMultipartTextMessage}, 3\rangle$	0.0821

SLANG

Predicts completions for sequences of API calls

Treats programs as (sets of) abstract histories

- Performs static analysis to abstract programs into finite histories

Training: learns bigrams, n-grams, RNNs on histories

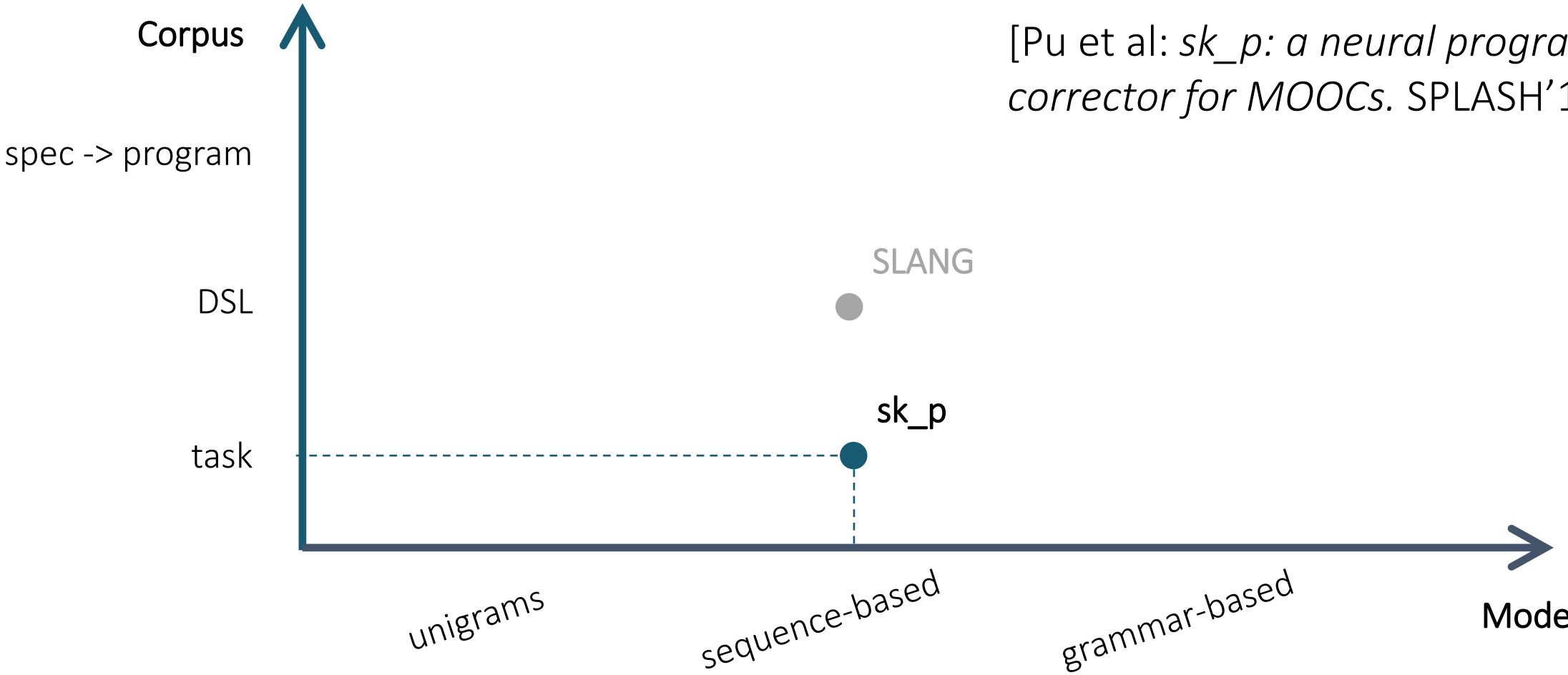
Inference: given a history with holes

- Uses bigrams to get possible completions
- Uses n-grams / RNN to rank them
- Combines history completions into a coherent program

Features: fast (very little search)

Limitations: all invocation pairs must appear in training set

Statistical Models in Synthesis



[Pu et al: *sk_p: a neural program corrector for MOOCs.* SPLASH'16]

sk_p

Input: incorrect program
+ test suite

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  for a in range(0, len(poly) - 1):
    f = poly[a]*x**a+f
    a += 1
  return f
```

sk_p

Output: corrected program

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  while a < len(poly):
    f = poly[a]*x**a+f
    a += 1
  return f
```

sk_p

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  for a in range(0, len(poly) - 1):
    f = poly[a]*x**a+f
    a += 1
  return f
```

normalize variables

```
_start_
  x2 = 0
  x3 = 0.0
  for x2 in range ( 0 , len ( x0 ) - 1 ) :
    x3 = x0 [ x2 ] * x1 ** x2 + x3
    x2 += 1
  return x3
_end_
```

extract partial fragments

Partial Fragment 1:

```
_start_
[          ]
  x3 = 0.0
```

Partial Fragment 2:

```
  x2 = 0
[          ]
  for x2 in range ( 0 , len ( x0 ) - 1 ) :
```

Partial Fragment 3:

```
  x3 = 0.0
[          ]
  x3 = x0 [ x2 ] * x1 ** x2 + x3
```

neural net (seq2seq)

```
0.141, while x2 < len ( x0 ):
0.007, for x4 in range ( len ( x0 ) ) :
0.0008, for x4 in range ( 0 ) :
```

beam search

```
def evaluatePoly(poly, x):
  a = 0
  f = 0.0
  while a < len(poly):
    f = poly[a]*x**a+f
    a += 1
  return f
```

sk_p

Program corrections for MOOCs

Treats programs as a sequence of tokens

- Abstracts away variables names

Uses the skipgram model to predict which statement is most likely to occur between the two

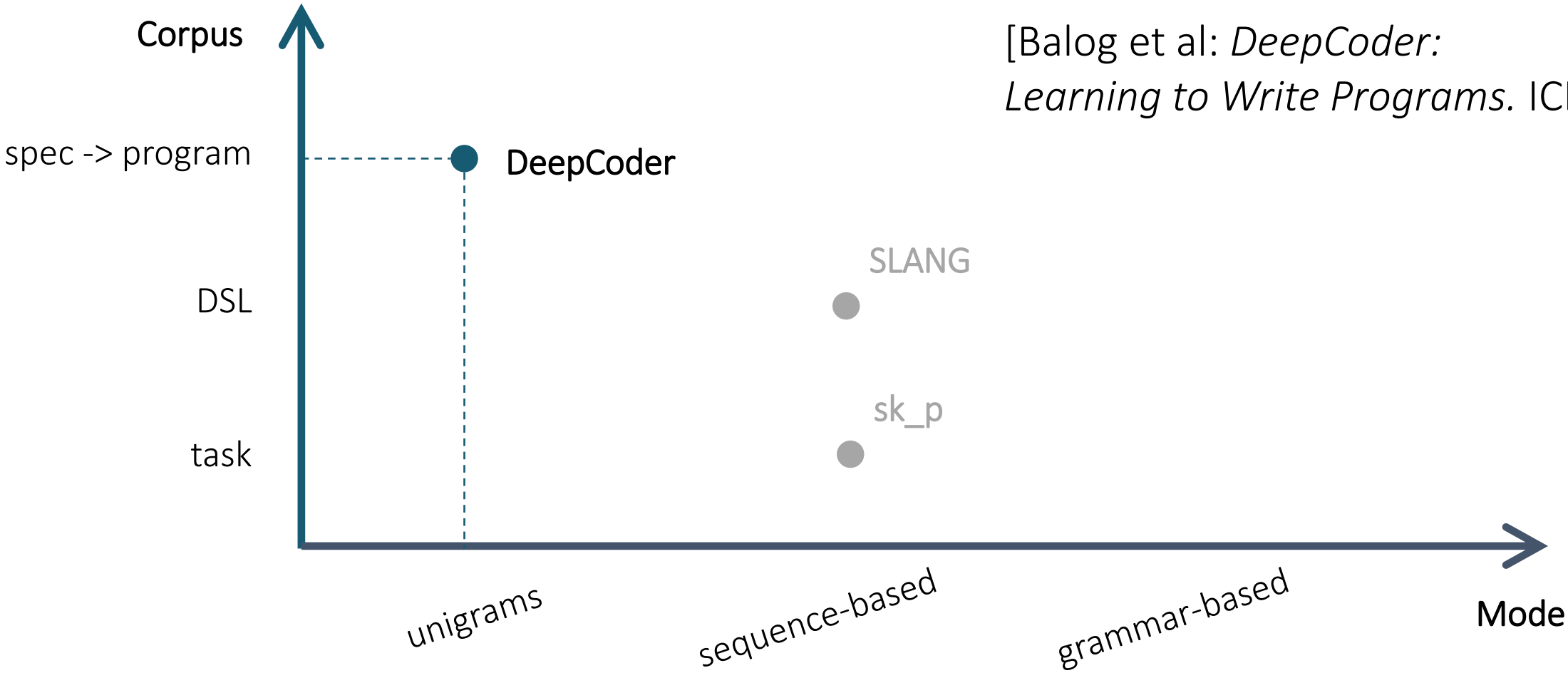
Features

- Can repair syntax errors

Limitations

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

Statistical Models in Synthesis



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

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
```

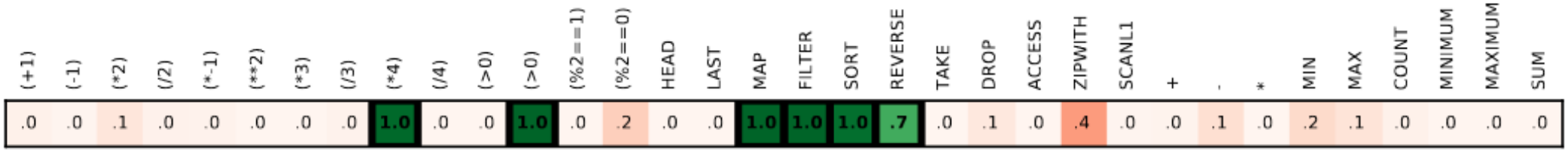
DeepCoder

Input: IO-examples

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

neural network

component likelihoods



existing search technique + sort-and-add

Output: Program in a list DSL

DeepCoder

Predicts likely components from IO examples

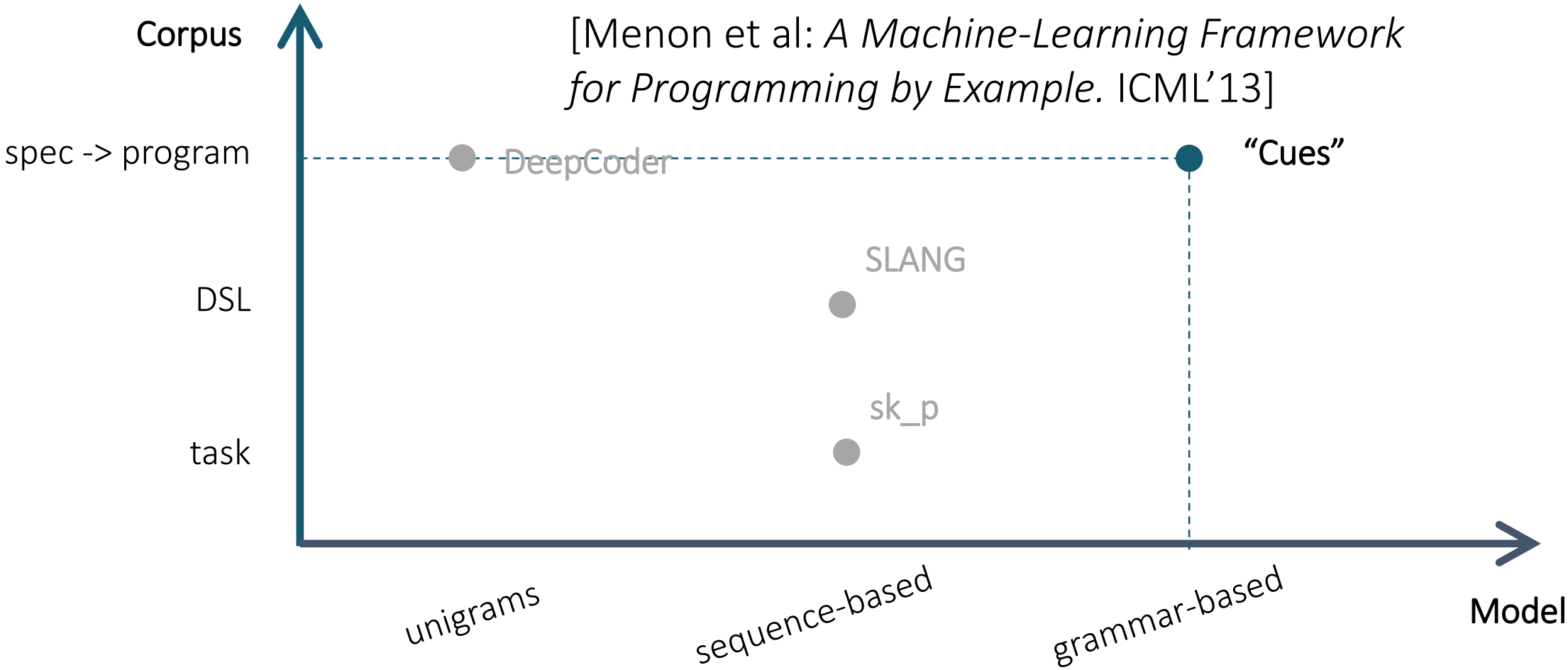
Features

- Trained on synthetic data
- Can be easily combined with any enumerative search
- Significant speedups for a small list DSL

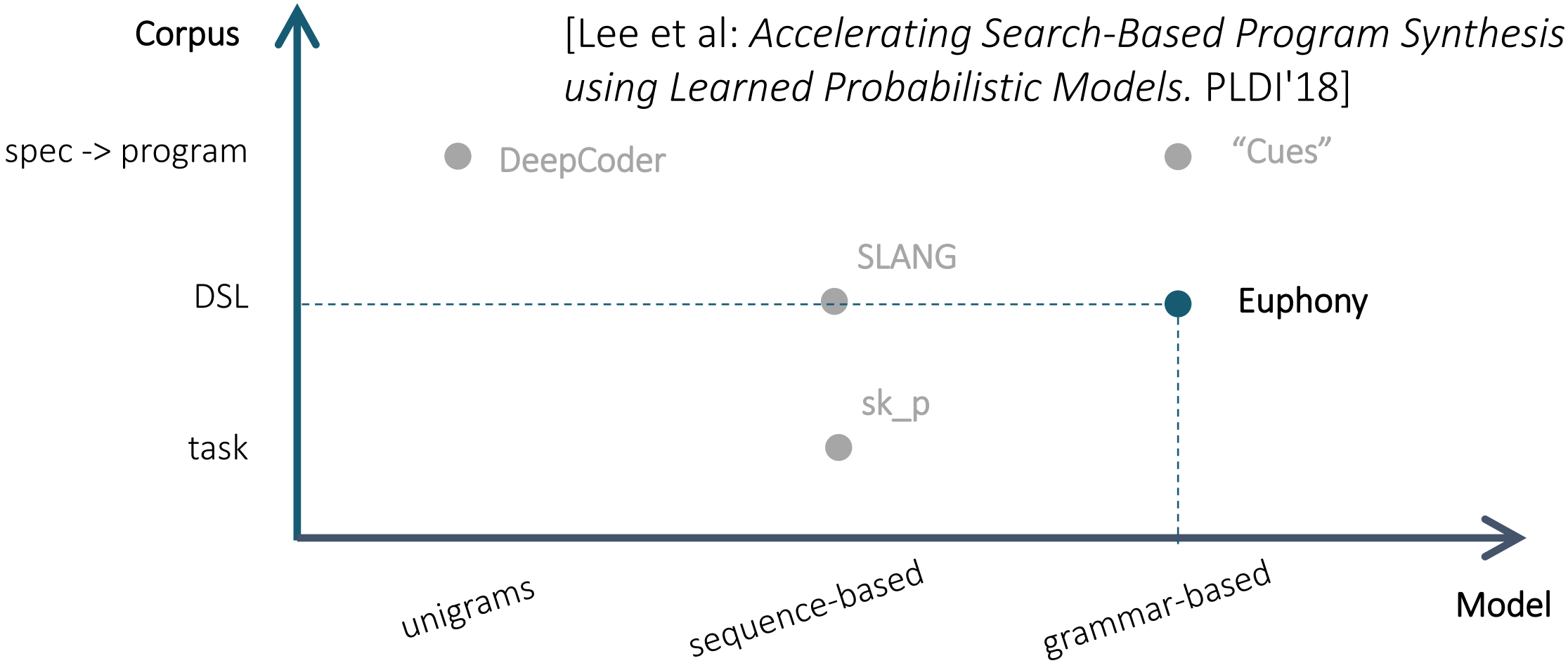
Limitations

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

Statistical Models in Synthesis

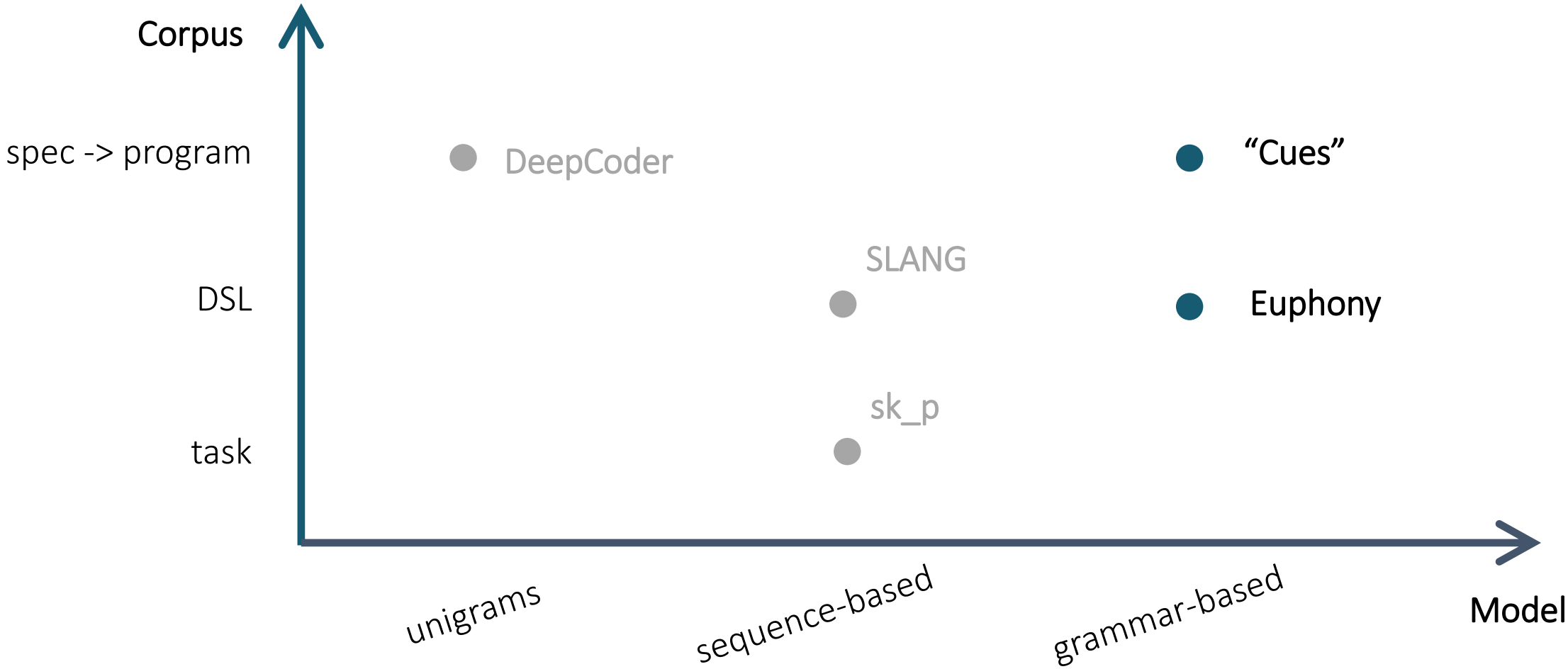


Statistical Models in Synthesis



[Lee et al: *Accelerating Search-Based Program Synthesis using Learned Probabilistic Models.* PLDI'18]

Grammar-based models: guiding the search



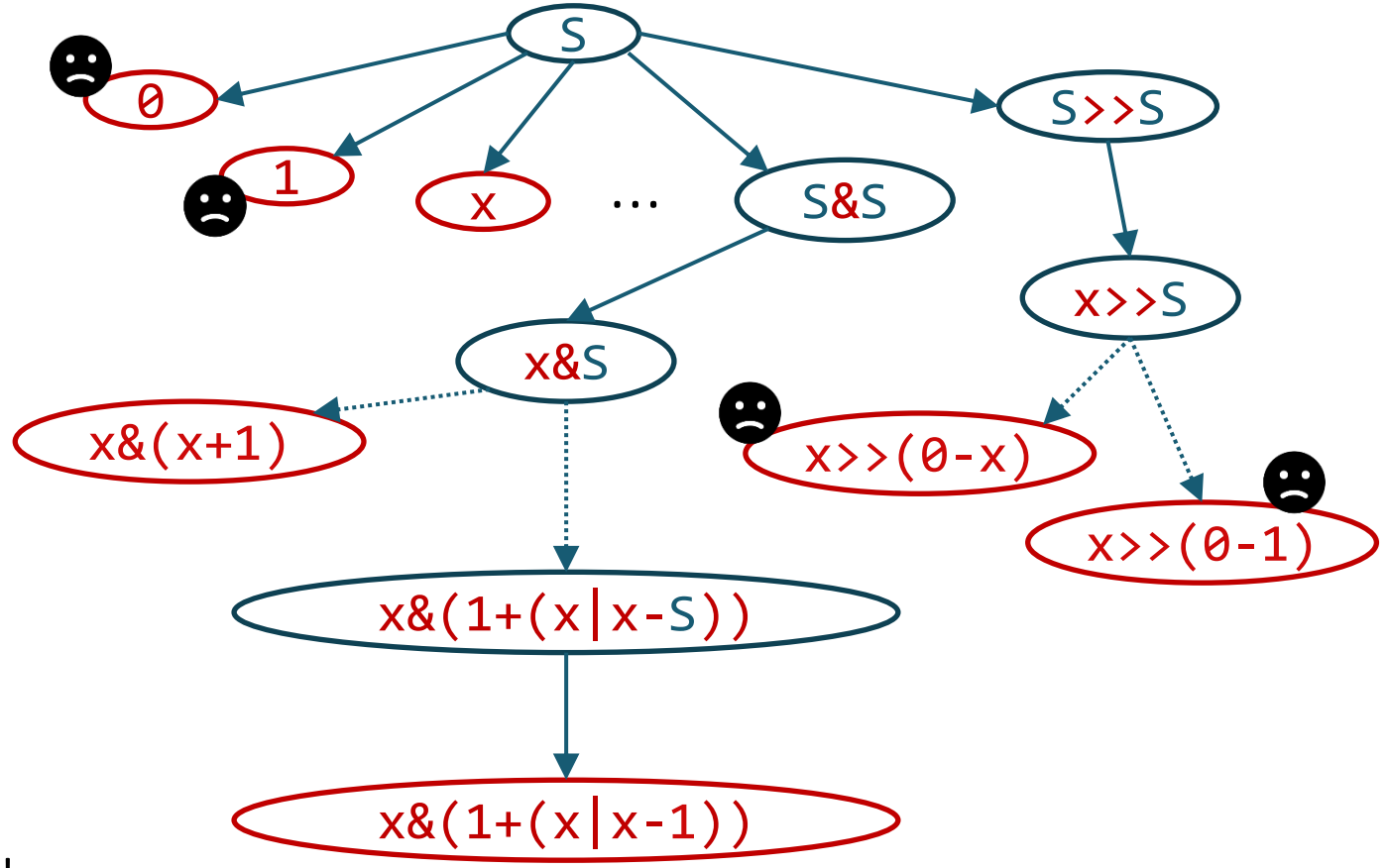
Top-down search (revisited)

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!



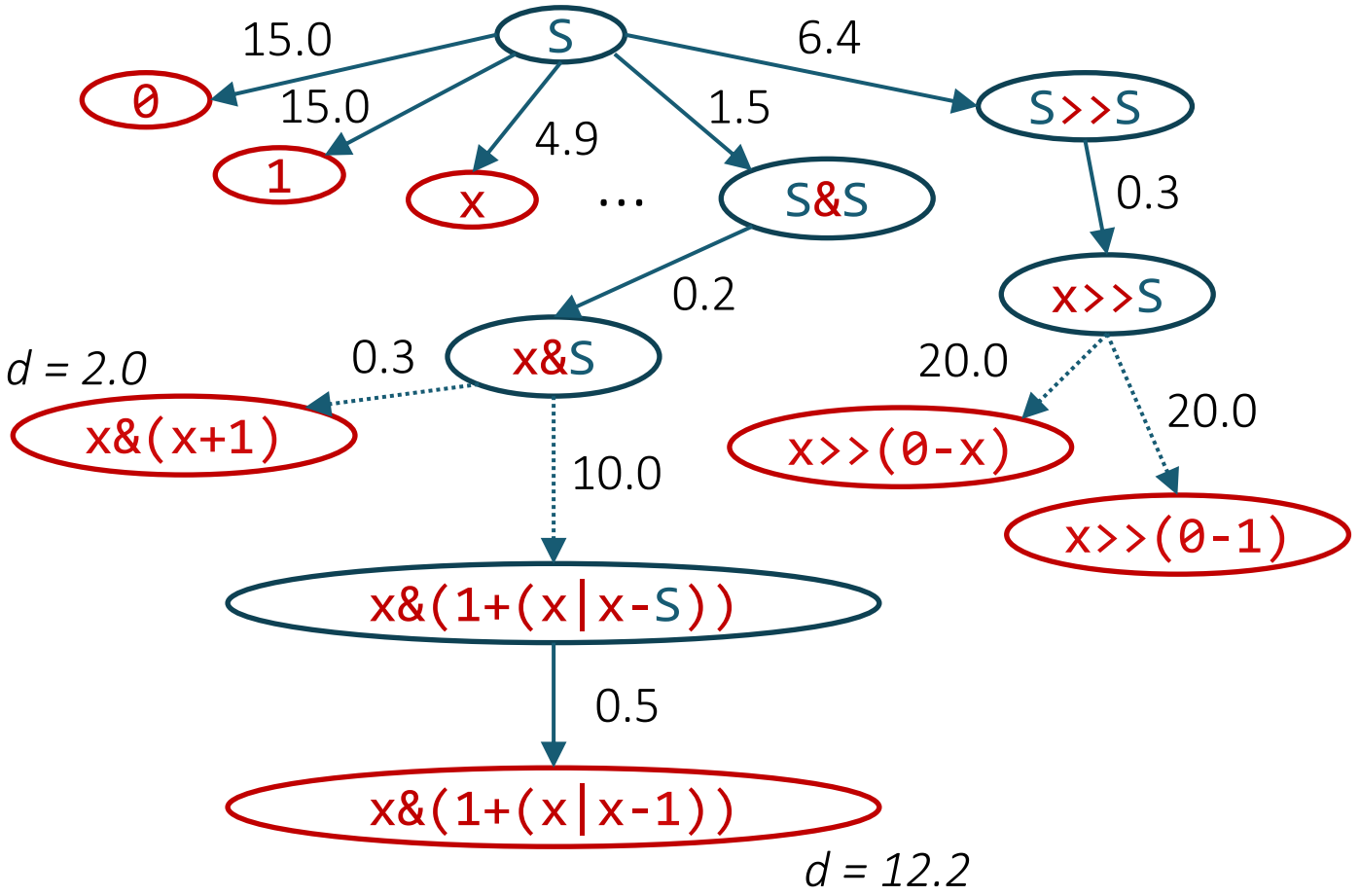
Weighted top-down search

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]) {
  wl := [<S,0>]
  while (wl != [])
    <p,d> := wl.dequeue_min(d);
    if (ground(p) && p([i]) = [o])
      return p;
    wl.enqueue(unroll(p,d));
}

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

`wl` now stores candidates (nodes) together with their distances

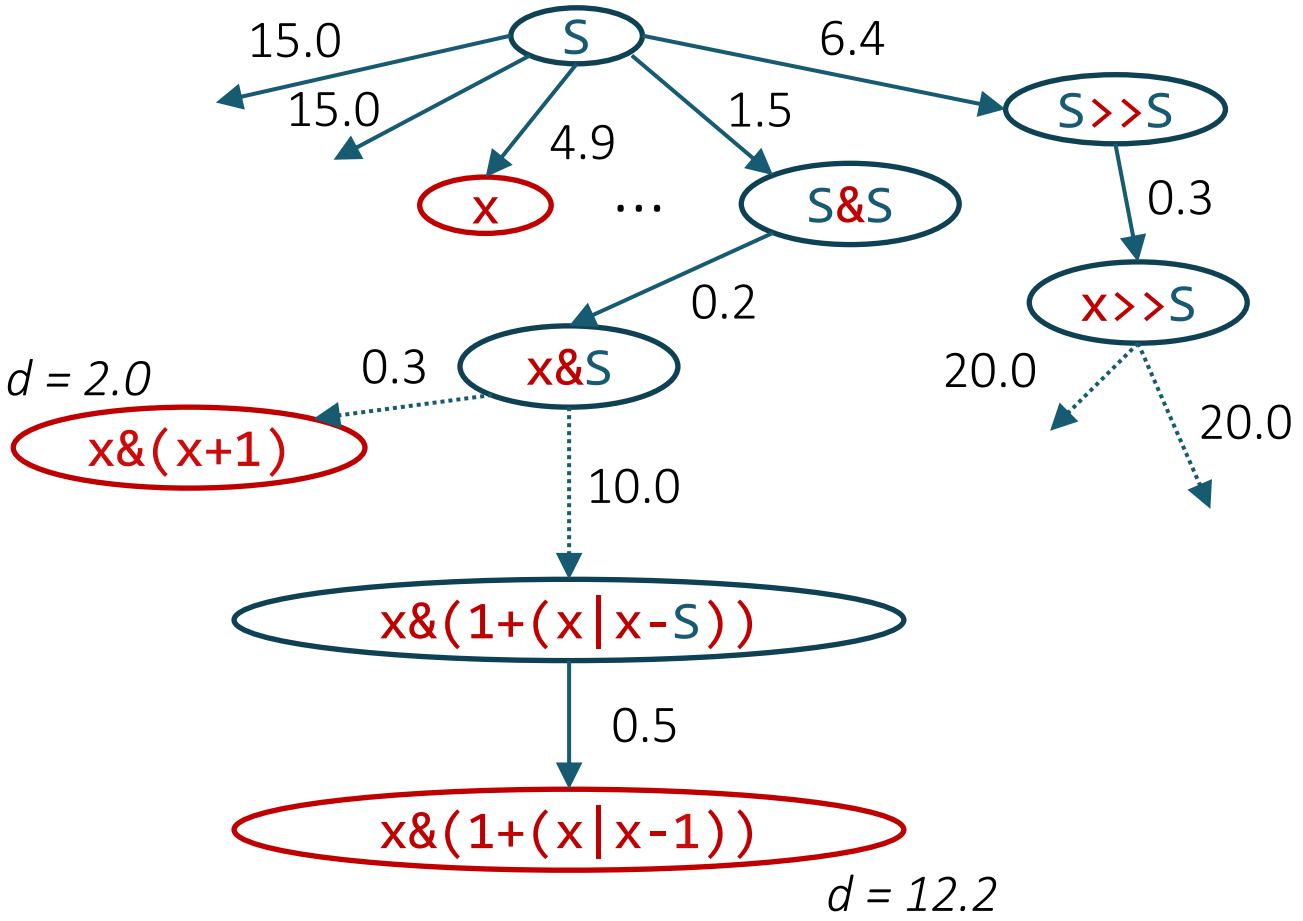
Dequeue the node with the shortest distance from the root

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*)

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

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

Roughly how close is this program to the closest leaf

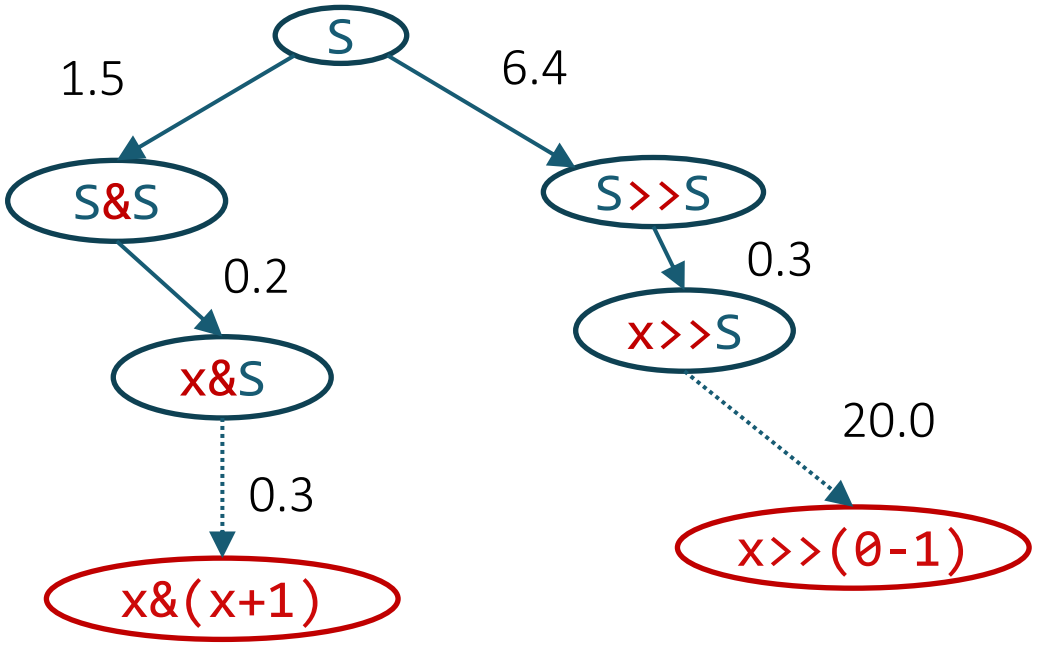
So, where do these come from?

Assigning weights to edges

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

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

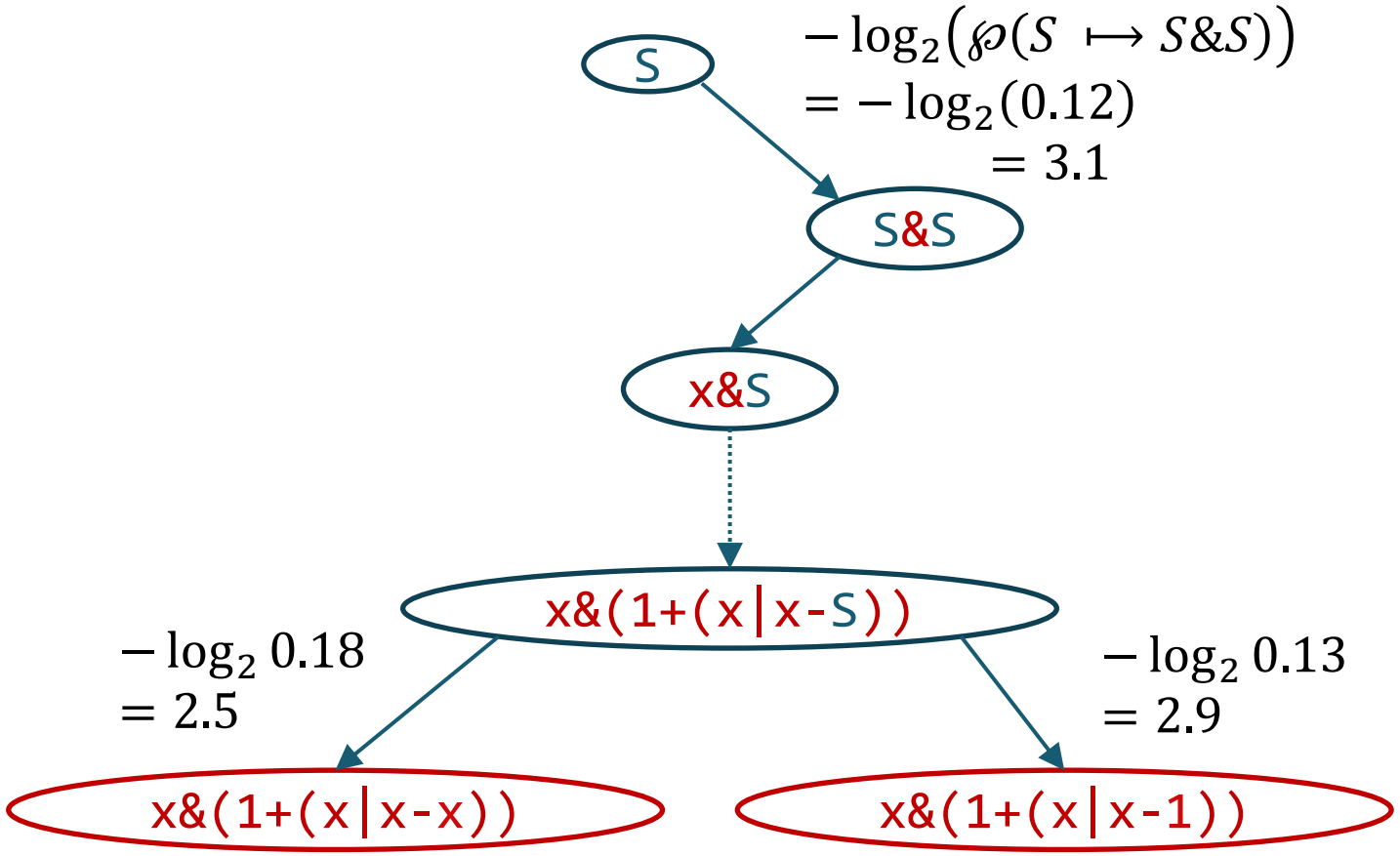
$$\wp(p) = \prod_{e \in S \to p} \wp(e)$$



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

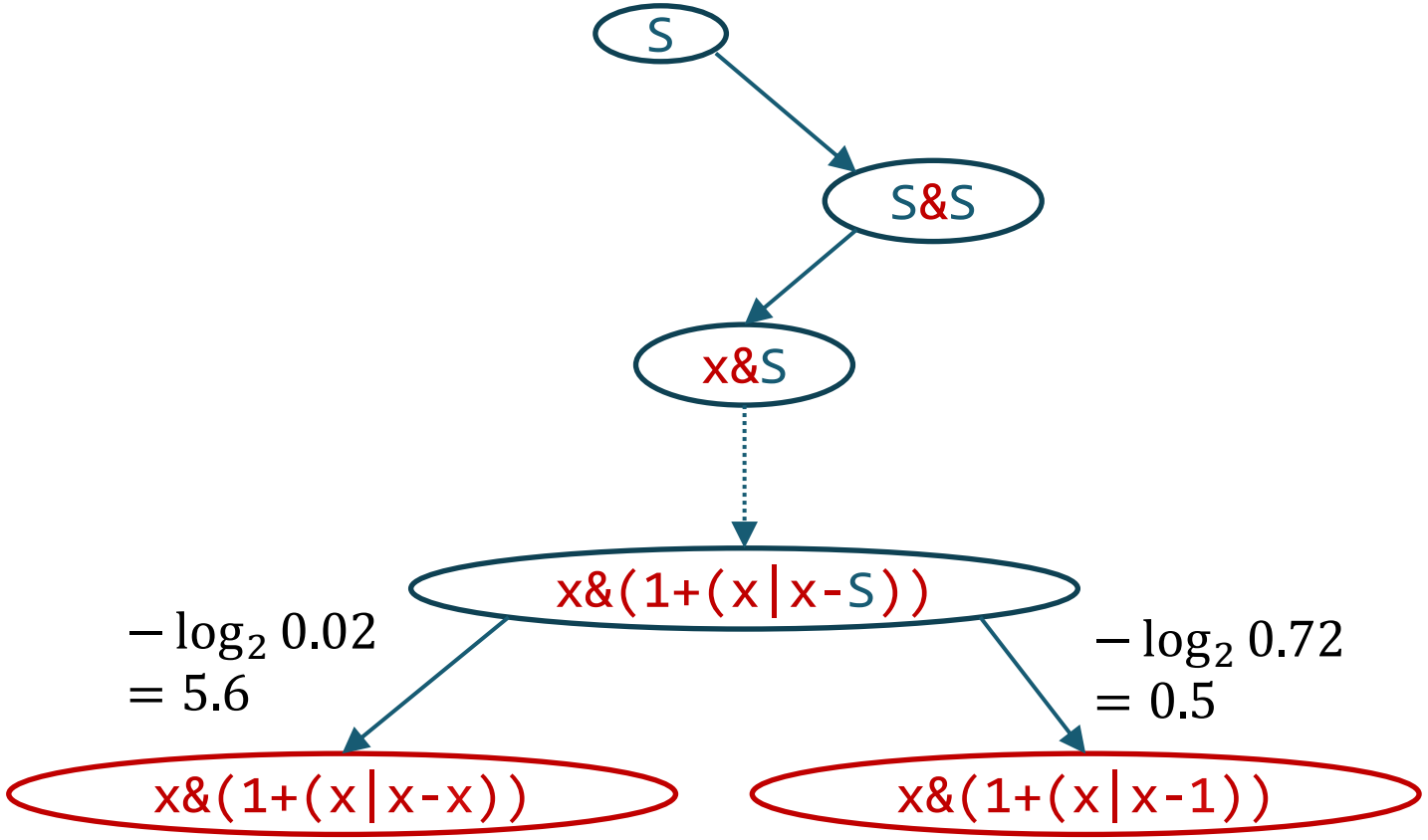


Probabilistic Higher-Order Grammar (PHOG)

[Bielik, Raychev, Vechev '16]

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

		$\wp$
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

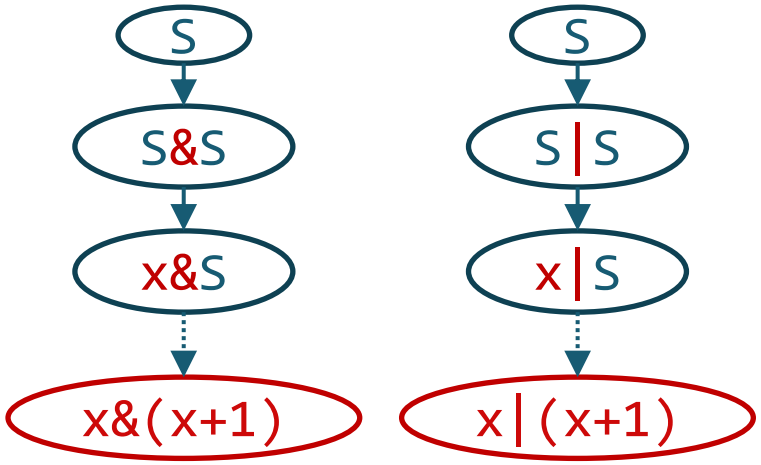
[Bielik, Raychev, Vechev '16]

CFG +

Corpus

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

parse



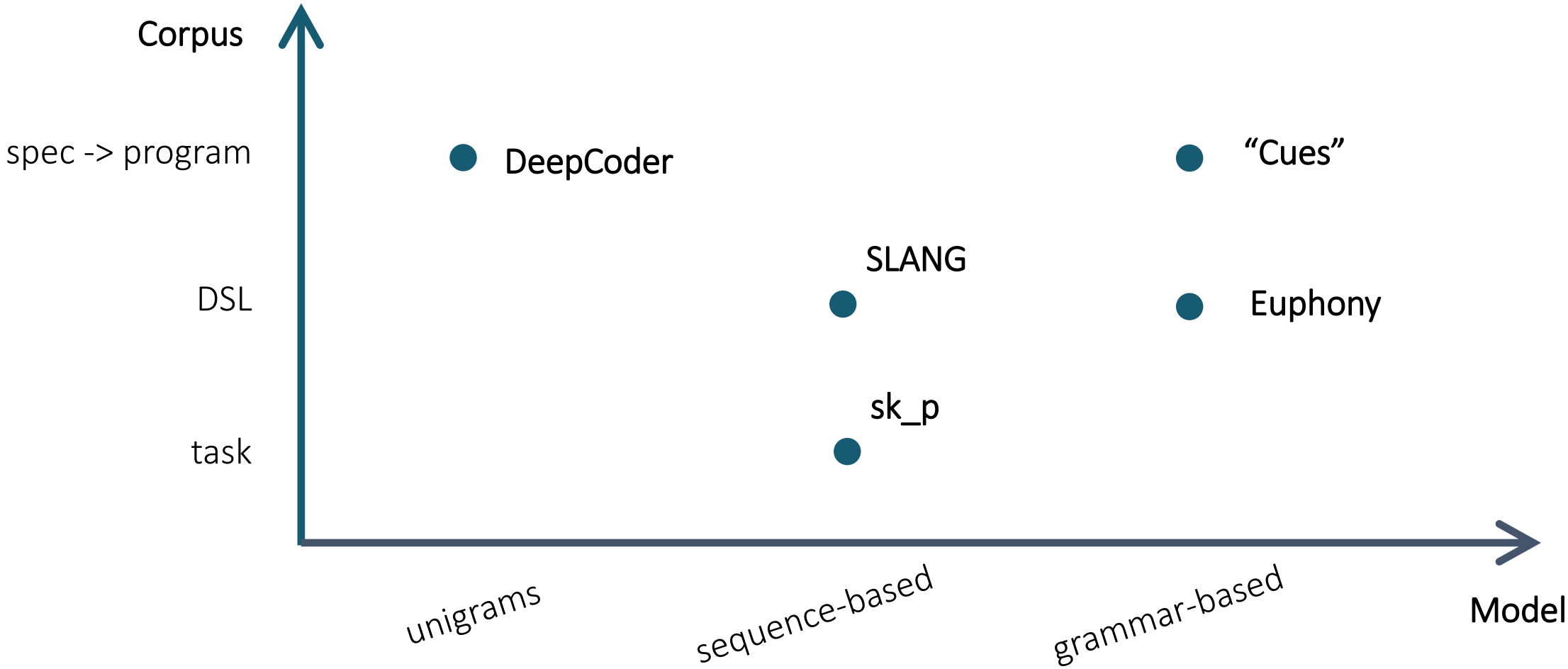
learn

`context, ℘`

PHOGs useful for:

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

How do they compare?



Euphony

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

- IO Examples (or first-order formula via CEGIS)
- PHOG
- Weighted enumerative search via A*

Euphony

Q2: What would these productions look like if we replaced the PHOG with a PCGF? With 3-grams?

PHOG:

```
S[“-”,Rep] ->  “.”     0.72
S[“-”,Rep] ->  “-”     0.001
S[“-”,Rep] ->  x       0.12
S[“-”,Rep] ->  S + S   0.02
…
```

PCFG:

```
S ->  “.”     0.2
S ->  “-”     0.2
S ->  x       0.3
S ->  S + S   0.2
…
```

3-grams:

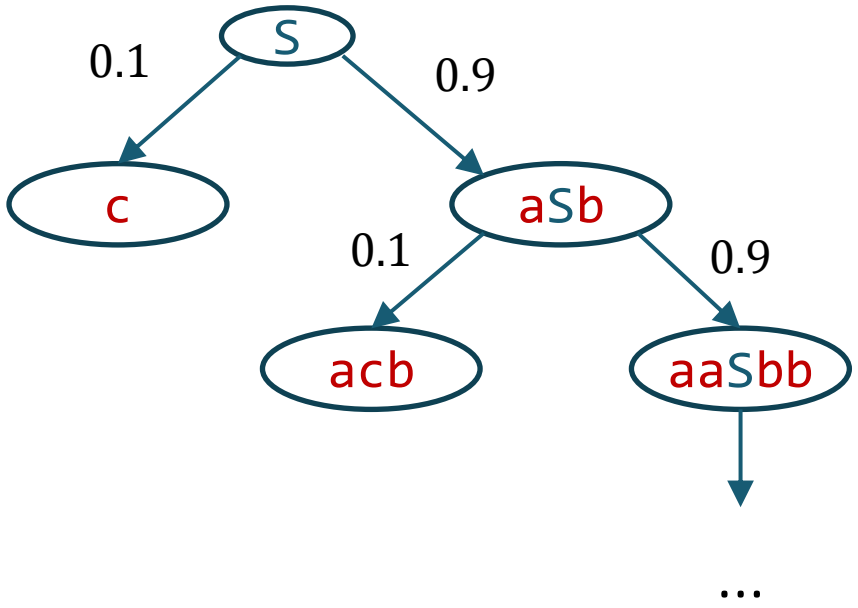
```
S[x,“-”] ->  “.”     0.72
S[x,“-”] ->  “-”     0.001
S[x,“-”] ->  x       0.12
S[x,“-”] ->  S + S   0.02
…
```

Do you think these other probabilistic models would work as well as a PHOG?

Euphony

Q3: What does h(S) = 0.1 mean? Why is it the case?

```
S ->  a S b  0.9
S ->  c      0.1
```



Euphony

Q4: Give an example of sentential forms n_i, n_j and set of points *pts* such that n_i and n_j are equivalent on *pts* but not weakly equivalent

```
S ->  S + S
S ->  x
```

```
n1 = S + S
n2 = x
```

```
pts = [(“” -> “”)]
```

Euphony: strengths

Efficient way to guide search by a probabilistic grammar

- Much better than DeepCoder's sort-and-add
- First to use A* and propose a sound heuristic

Transfer learning for PHOGs

- Remember: abstraction is key to learning models of code!

Extend observational equivalence to top-down search

Euphony: weaknesses

Requires high-quality training data

- for each problem domain!

Transfer learning requires manually designed features

Next week

Topics:

- Representation-based search
- Stochastic search

Paper: Rishabh Singh: BlinkFill: Semisupervised Programming By Example for Syntactic String Transformations. VLDB'16

Projects:

- Proposals due Friday
- Should demonstrate that you started working on the project or at least researched the area
- Once you have decided on the topic, put it on the Google sheet next to any of the team members
- If you haven't decided, talk to me after class or in OH