

Lecture 4

Weighted Enumerative Search

Nadia Polikarpova

Plan for this week

Today:

- Weighted enumerative search

Thursday:

- Discuss the Euphony paper
- Synthesis frameworks + suggested projects

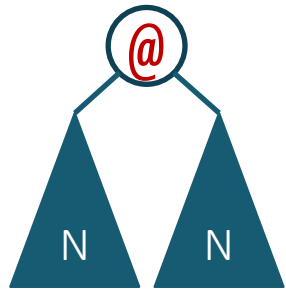
Project:

- Proposals due in ten days
- Talk to me about the topic

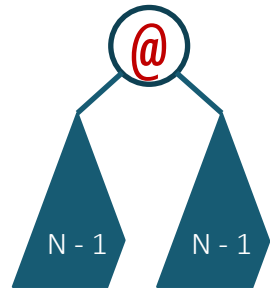
Scaling enumerative search

Prune

Discard useless subprograms



$$m * N^2$$



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

Prioritize

Explore more promising candidates first

$$P = \{ \begin{array}{l} [0][N..N] \\ x[N..N] \\ \dots \end{array} , \quad \leftarrow \begin{array}{l} \text{dequeue} \\ \text{this first} \end{array}$$

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:

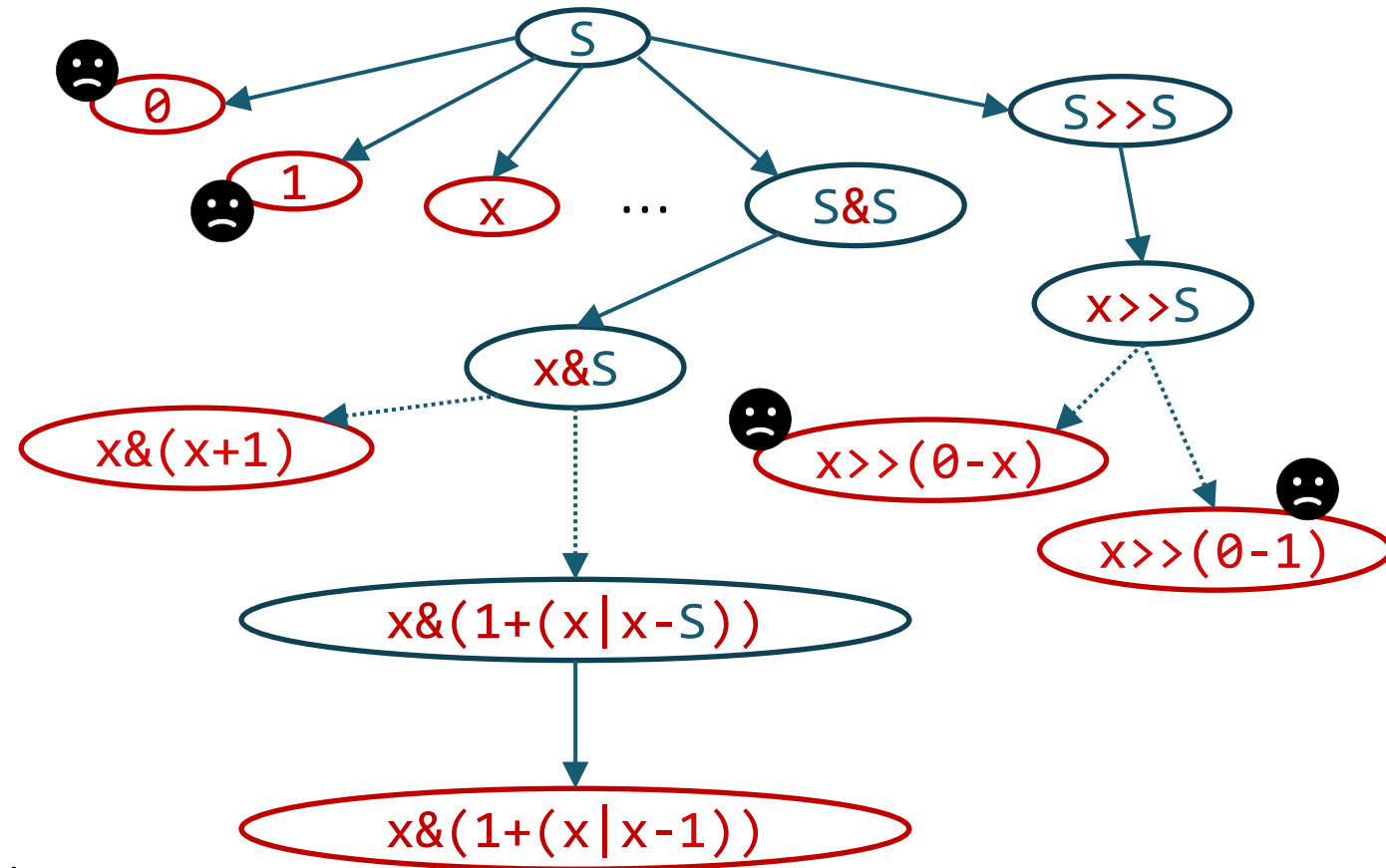
00101 → 00100

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

(+1)	(-1)	(*2)	(/2)	(*1)	(**2)	(*3)	(/3)	(*4)	(/4)	(>0)	(>0)	(%2==1)	(%2==0)	HEAD	LAST	MAP	FILTER	SORT	REVERSE	TAKE	DROP	ACCESS	ZIPWITH	SCANL1	+	.	*	MIN	MAX	COUNT	MINIMUM	MAXIMUM	SUM
.0	.0	.1	.0	.0	.0	.0	.0	1.0	.0	.0	1.0	.0	.2	.0	.0	1.0	1.0	1.0	.7	.0	.1	.0	.4	.0	.0	.1	.0	.2	.1	.0	.0	.0	.0

↓
weighted search

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

DeepCoder: search strategies

Top-down DFS

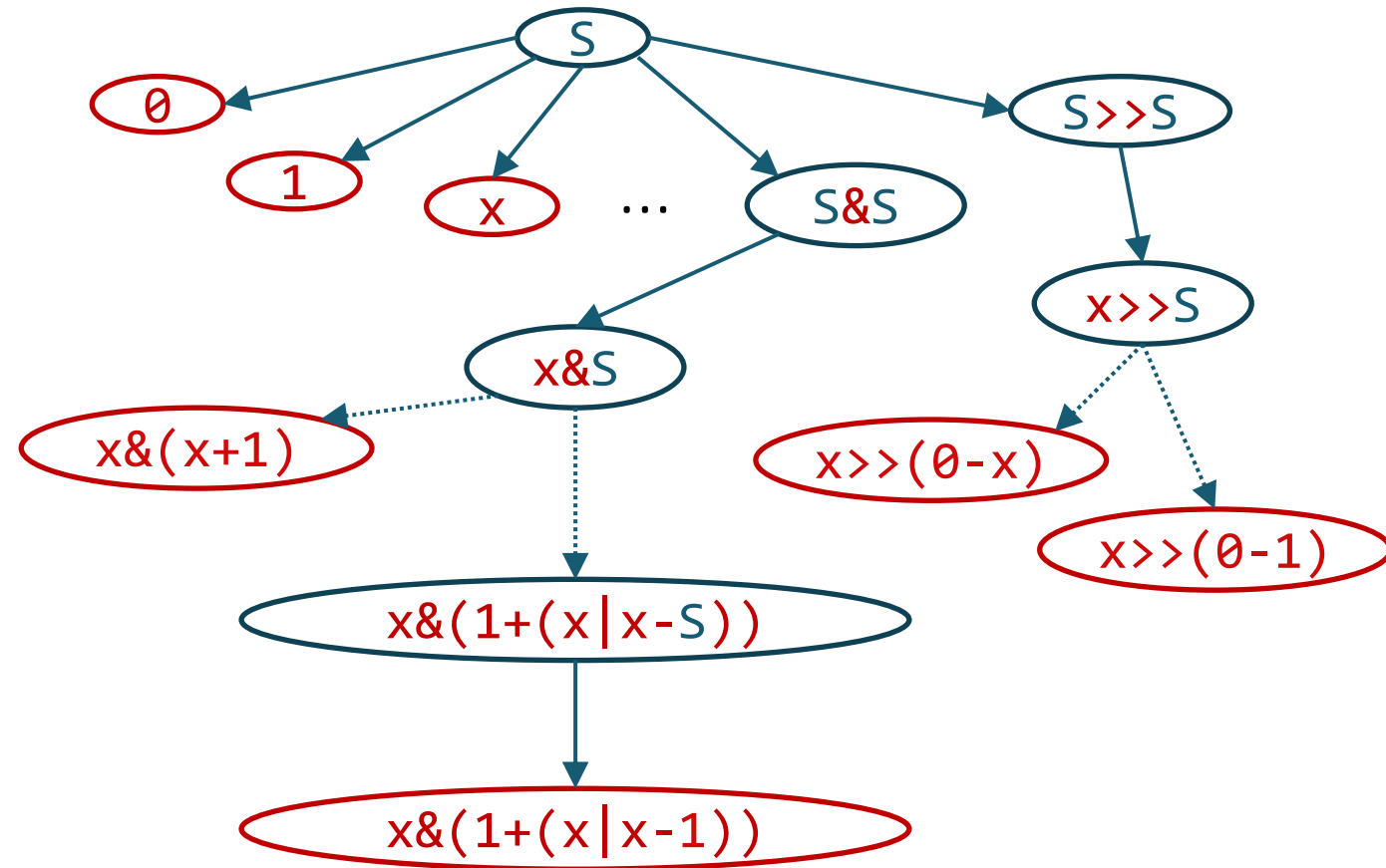
- Picks expansions for the current non-terminal in the order of probability

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

- for example: $\wp(S \rightarrow x \mid S) = 0.3$ $\wp(S \rightarrow x \mid x - S) = 0.0001$
- 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:

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

Term probability:

- let $S = \tau_0 \xrightarrow{r_1} \tau_1 \xrightarrow{r_2} \dots \xrightarrow{r_n} \tau_n = \tau$ be the unique derivation of partial program τ

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

Types of context

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


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)

	$\wp(R)$
$S \rightarrow 0$	0.13
$S \rightarrow 1$	0.13
$S \rightarrow x$	0.18
$S \rightarrow S + S$	0.11
$S \rightarrow S - S$	0.11
$S \rightarrow S \& S$	0.12
$S \rightarrow S S$	0.12
$S \rightarrow S \ll S$	0.05
$S \rightarrow S \gg S$	0.05

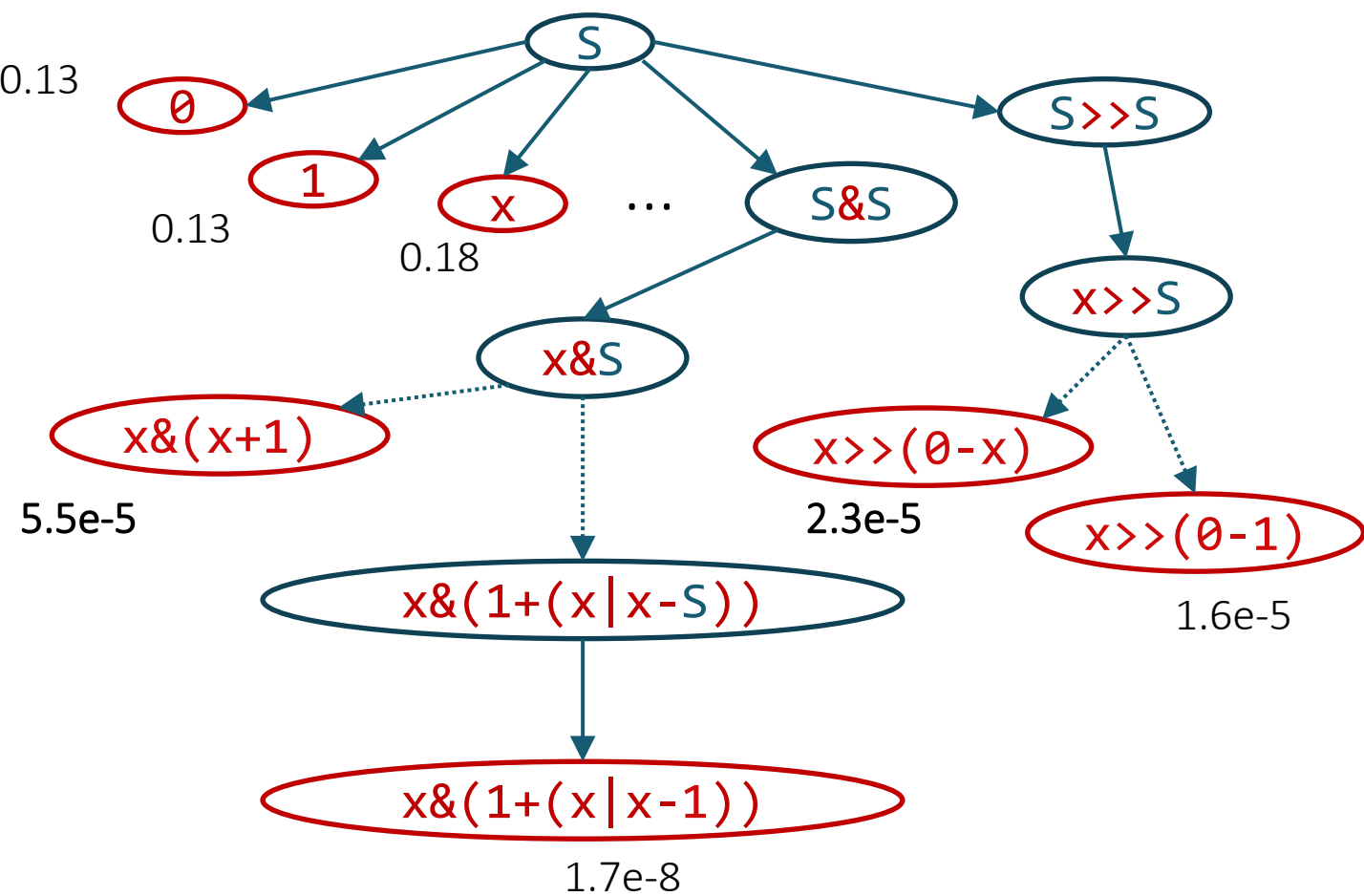
$$\wp: R \rightarrow [0,1]$$

Encodes the popularity of each production (operation)

- here: variable more likely than constant, plus more likely than shift

Probabilistic Context-Free Grammars (PCFG)

		$\mathcal{P}(R)$
$S \rightarrow$	θ	0.13
$S \rightarrow$	1	0.13
$S \rightarrow$	x	0.18
$S \rightarrow$	$S + S$	0.11
$S \rightarrow$	$S - S$	0.11
$S \rightarrow$	$S \& S$	0.12
$S \rightarrow$	$S S$	0.12
$S \rightarrow$	$S \ll S$	0.05
$S \rightarrow$	$S \gg S$	0.05



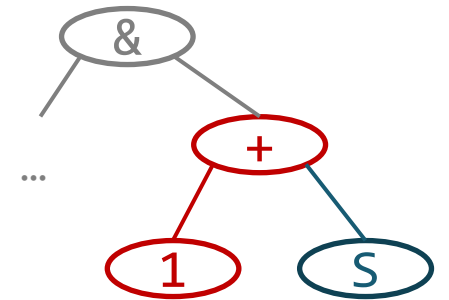
N-grams

$N[\text{left sibling, parent}] \rightarrow \text{rhs}$

		\wp
$S[x, -]$	$\rightarrow 1$	0.72
$S[x, -]$	$\rightarrow x$	0.02
$S[x, -]$	$\rightarrow S + S$	0.12
$S[x, -]$	$\rightarrow S - S$	0.12
...		
$S[1, +]$	$\rightarrow 1$	0.26
$S[1, +]$	$\rightarrow x$	0.25
$S[1, +]$	$\rightarrow S + S$	0.19
$S[1, +]$	$\rightarrow S - S$	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

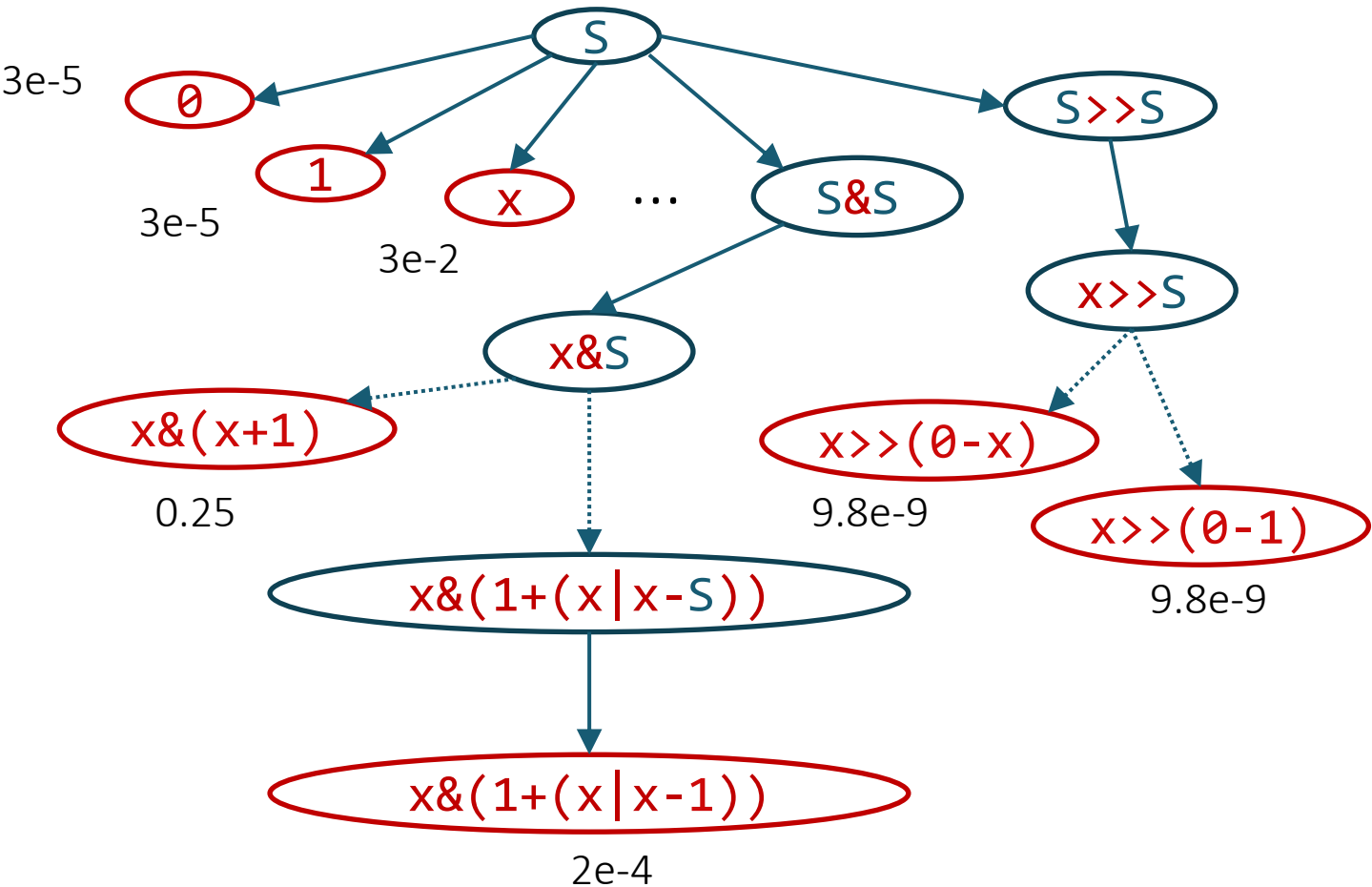


- here: x is not likely in $x - S$ but likely in $1 + S$

N-grams

N[left sibling, parent] -> rhs

		ϕ
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. define context as a program that traverses the AST
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

Weighted enumerative search

DeepCoder

Probabilistic Grammars

Weighted top-down search

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

Weighted bottom-up search

Barke, Peleg, Polikarpova. Just-in-Time Learning for Bottom-Up Enumerative Synthesis. OOPSLA'20

Shi, Bieber, Singh. TF-Coder: Program Synthesis for Tensor Manipulations. arXiv

Weighted top-down search

Wanted: explore programs in the order of **probability**

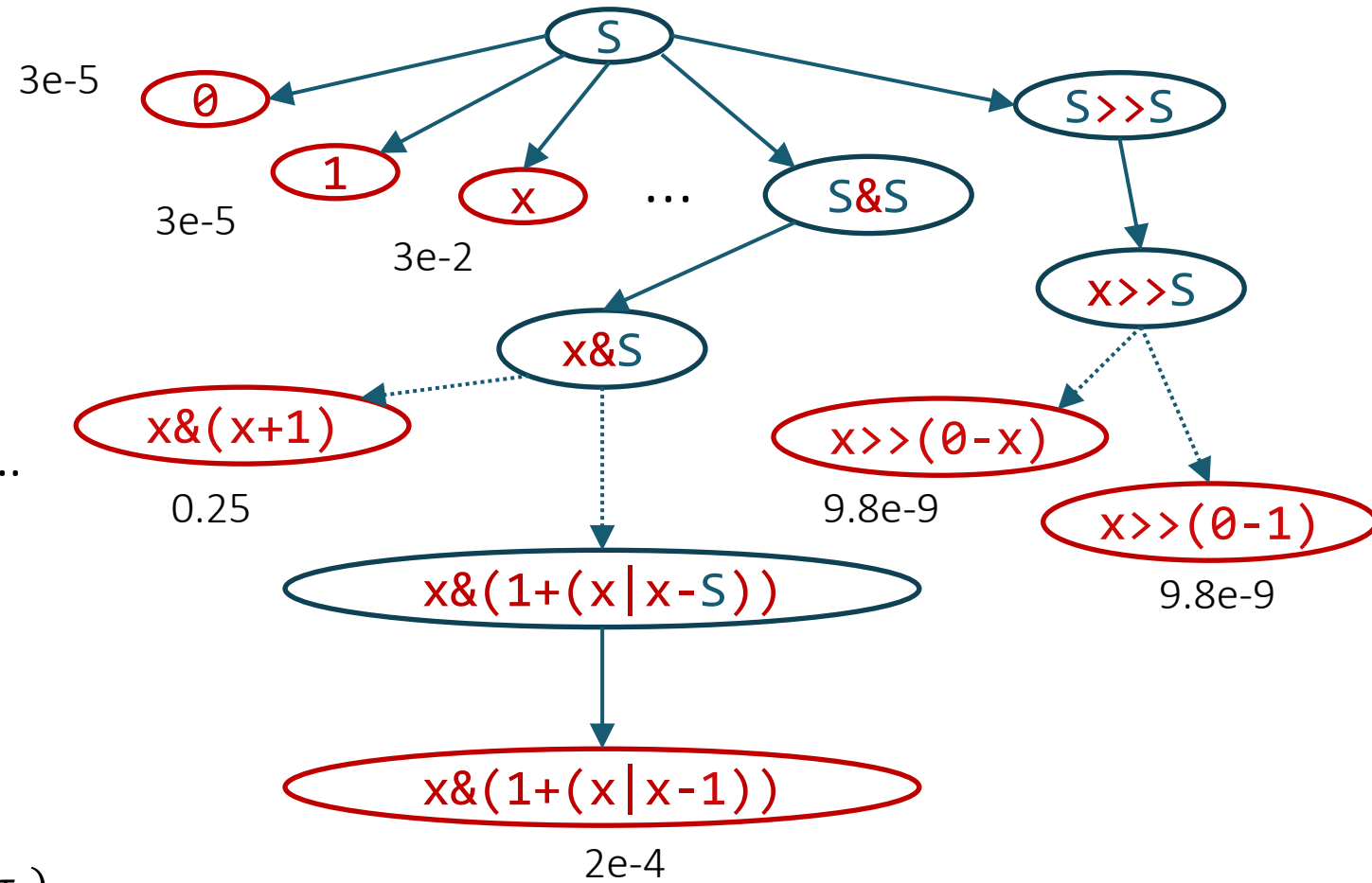
$$\wp(t) = \prod_{(r_i, \tau_i) \in S \rightarrow^* t} \wp(r_i \mid \tau_i)$$

Hard to maximize multiplicative cost...
but easy to minimize additive cost!

= **shortest path**

$$cost(t) = \sum_{(r_i, \tau_i) \in S \rightarrow^* t} weight(r_i \mid \tau_i)$$

$$-\log_2 \wp(t) = \sum_{(r_i, \tau_i) \in S \rightarrow^* t} -\log_2 \wp(r_i \mid \tau_i)$$



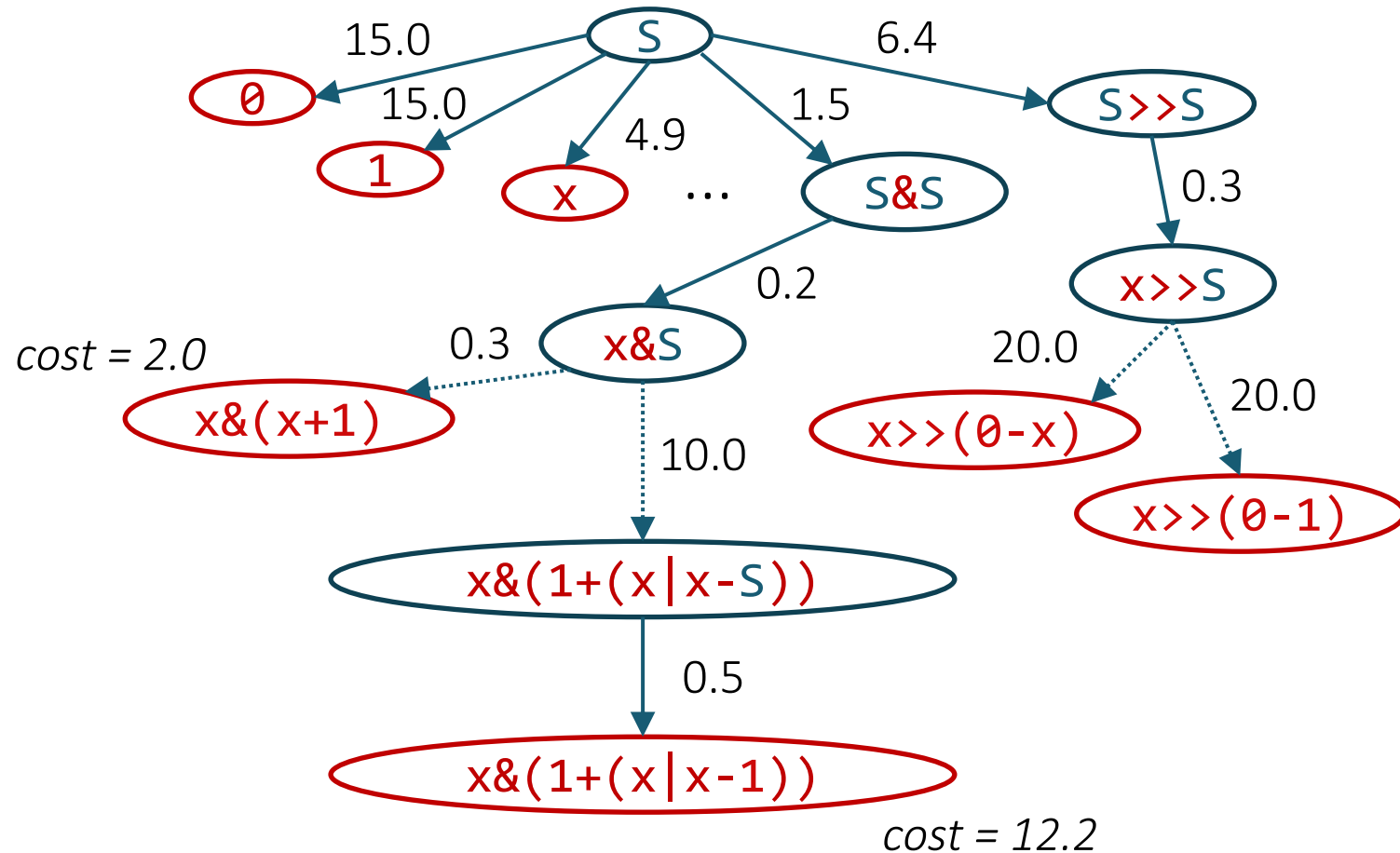
Weighted top-down search

Assigns weights to edges:





$$\text{weight}(r_i | \tau_i) = -\log_2 \wp(r_i | \tau_i)$$

Now $\text{cost}(t) < \text{cost}(t')$
iff t is more likely than t' !

We can use shortest path algo
(e.g. Dijkstra) to search by cost!



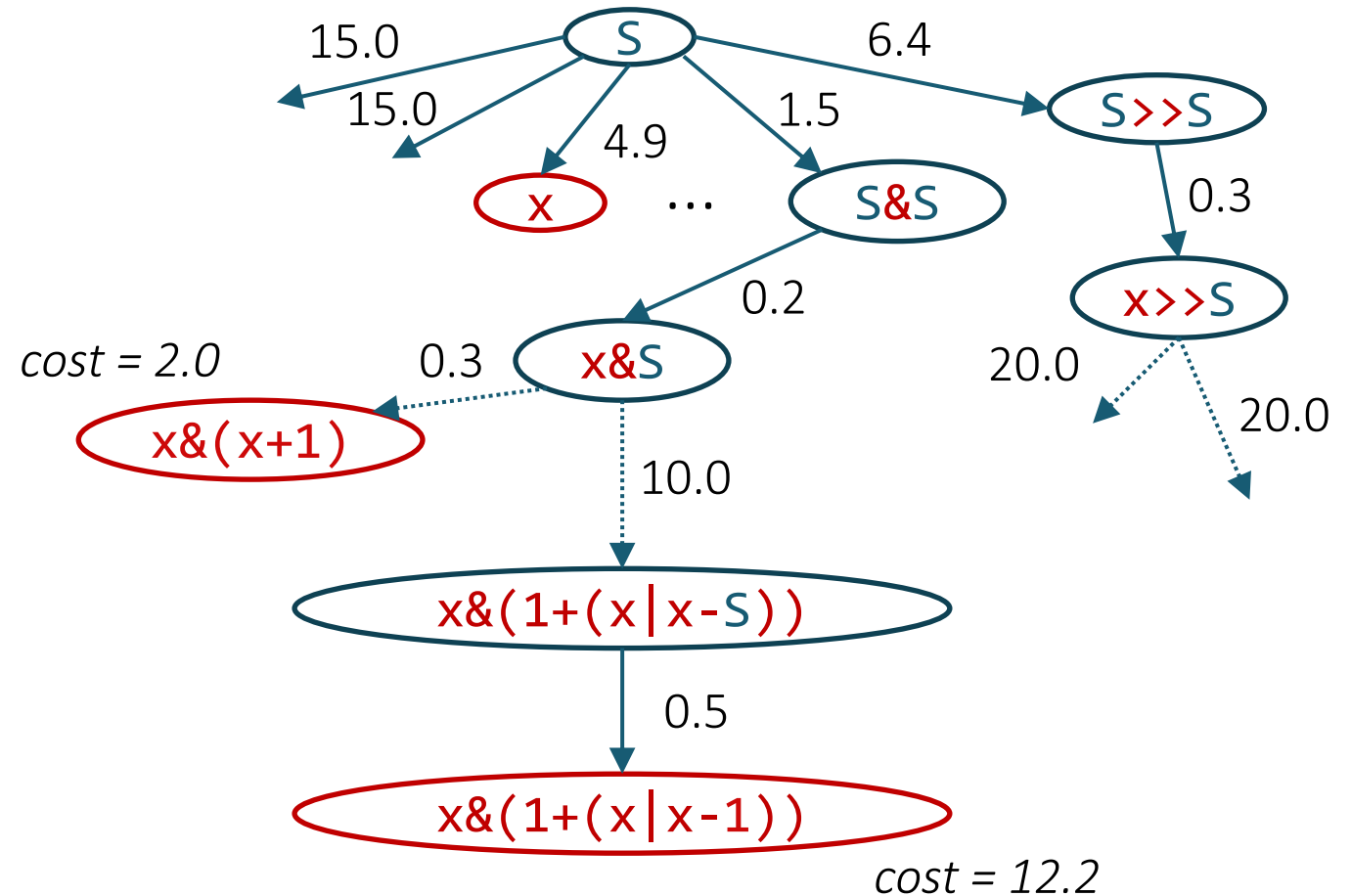
Weighted top-down search (Dijkstra)

```
top-down(< $\Sigma$ , N, R, S>, [i  $\rightarrow$  o]) {  
  w1 := [<S, 0>]  w1 now stores candidates (nodes)  
  while (w1 != [])  together with their costs  
    < $\tau$ , c> := w1.dequeue_min(c);  Dequeue the node with minimal cost  
    if (complete( $\tau$ ) &&  $\tau$ ([i]) = [o])  
      return  $\tau$ ;  
    w1.enqueue(unroll( $\tau$ , c));  
}  
  
unroll( $\tau$ , c) {  
  w1' := []  
  A := left-most nonterminal in  $\tau$   
  forall (A  $\rightarrow$  rhs) in R:  
    w1' += < $\tau$ [A  $\rightarrow$  rhs], c + w(A  $\rightarrow$  rhs |  $\tau$ )>  Distance to a new node: add the w(R)  
  return w1';  
}
```


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(< $\Sigma$ , N, R, S>, [ $i \rightarrow o$ ]) {  
  w1 := [<S, 0, h(S)>]  
  while (w1 != [])  
    < $\tau$ , c, h> := w1.dequeue_min(c + h);  
    if (complete( $\tau$ ) &&  $\tau$ ( $i$ ) =  $o$ )  
      return  $\tau$ ;  
    w1.enqueue(unroll( $\tau$ , c));  
}
```

Roughly how close is this
program to the closest leaf

```
unroll( $\tau$ , c) {  
  w1' := []  
  A := leftmost nonterminal in  $\tau$   
  forall (A  $\rightarrow$  rhs) in R:  
    w1' += < $\tau$ [A  $\rightarrow$  rhs], c + w(A  $\rightarrow$  rhs |  $\tau$ ),  
          h( $\tau$ [A  $\rightarrow$  rhs])>  
  return w1';  
}
```

Weighted enumerative search

DeepCoder

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

Weighted top-down search

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

Weighted bottom-up search

Barke, Peleg, Polikarpova. Just-in-Time Learning for Bottom-Up Enumerative Synthesis. OOPSLA'20

Shi, Bieber, Singh. TF-Coder: Program Synthesis for Tensor Manipulations. TOPLAS'22

Bottom-up search (revisited)

```
bottom-up (< $\Sigma$ , N, R, S>, [ $i \rightarrow o$ ]):
```

```
  bank[A,d] := {} forall A, d
```

```
  for d in [0..]:
```

```
    forall (A  $\rightarrow$  rhs) in R:
```

```
      forall p in new-terms(A $\rightarrow$ rhs, d, bank):
```

```
        if (A = S  $\wedge$  p([i]) = [o]):
```

```
          return p
```

```
        bank[A,d] += p;
```

```
new-terms(A  $\rightarrow$   $\sigma$ (A1...An), d, bank):
```

```
  if (d = 0  $\wedge$  n = 0) yield  $\sigma$ 
```

```
  else forall <d1,...,dn> in [0..d-1]n s.t. max(d1,...,dn) = d-1:
```

```
    forall <p1,...,pn> in bank[A1,d1]  $\times$  ...  $\times$  bank[An,dn]:
```

```
      yield  $\sigma$ (p1,...,pn)
```

Search by depth



Bottom-up variations

```
new-terms( $A \rightarrow \sigma(A_1 \dots A_n)$ , d, bank):  
  if ( $d = 0 \wedge n = 0$ ) yield  $\sigma$   
  else forall  $\langle d_1, \dots, d_n \rangle$  in  $[0..d-1]^n$  s.t.  $\max(d_1, \dots, d_n) = d-1$ :  
    forall  $\langle p_1, \dots, p_n \rangle$  in  $\text{bank}[A_1, d_1] \times \dots \times \text{bank}[A_n, d_n]$ :  
      yield  $\sigma(p_1, \dots, p_n)$ 
```

by depth

```
new-terms( $A \rightarrow \sigma(A_1 \dots A_n)$ , s, bank):  
  if ( $s = 1 \wedge n = 0$ ) yield  $\sigma$   
  else forall  $\langle s_1, \dots, s_n \rangle$  in  $[0..s-1]^n$  s.t.  $\text{sum}(s_1, \dots, s_n) = s-1$ :  
    forall  $\langle p_1, \dots, p_n \rangle$  in  $\text{bank}[A_1, s_1] \times \dots \times \text{bank}[A_n, s_n]$ :  
      yield  $\sigma(p_1, \dots, p_n)$ 
```

by size

```
new-terms( $A \rightarrow \sigma(A_1 \dots A_n)$ , c, bank):  
  budget =  $c - w(A \rightarrow \sigma(A_1 \dots A_n))$   
  if (budget = 0  $\wedge$  n = 0) yield  $\sigma$   
  else forall  $\langle c_1, \dots, c_n \rangle$  in  $[0.. \text{budget}]^n$  s.t.  $\text{sum}(c_1, \dots, c_n) = \text{budget}$ :  
    forall  $\langle p_1, \dots, p_n \rangle$  in  $\text{bank}[A_1, c_1] \times \dots \times \text{bank}[A_n, c_n]$ :  
      yield  $\sigma(p_1, \dots, p_n)$ 
```

by cost!

Bottom-up by cost: discussion

What kind of cost functions are supported?

- positive
- integer
- context-free

Bottom-up: example

by depth

d= 0: x

d =1: sort(x)
x + x

d = 2: sort(sort(x))
sort(x + x)
x + sort(x)
sort(x) + x
x + (x + x)
(x + x) + x

d = 3: ...

by size

s= 1: x

s =2: sort(x)

s = 3: x + x
sort(sort(x))

s = 4: sort(x + x)
sort(sort(sort(x)))

s = 5: x + sort(x)
sort(x) + x

s = 5: ...

L ::= sort(L)
L + L
x
by cost

cost
10
3
1

c= 1: x

c =2,3,4:

c = 5: x + x

c =6,7,8:

c = 9: x + (x + x)
(x + x) + x

c = 10:

c = 11: sort(x)

c = 12:

c = 13: x + (x + (x + x))
(x + x) + (x + x)
(x + (x + x)) + x

Weighted search

Top-down

- + Supports real-valued weights: optimal enumeration order
- + Supports context-dependent weights

Bottom-up

- + Inherits benefits of bottom up: dynamic programming, OE

Euphony

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

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

Euphony

Rep x “_” S

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

PHOG:

$S[\text{“_”}, \text{Rep}] \rightarrow \text{“.”} \quad 0.72$
 $S[\text{“_”}, \text{Rep}] \rightarrow \text{“_”} \quad 0.001$
 $S[\text{“_”}, \text{Rep}] \rightarrow x \quad 0.12$
 $S[\text{“_”}, \text{Rep}] \rightarrow S + S \quad 0.02$

...

PCFG:

$S \rightarrow \text{“.”} \quad 0.2$
 $S \rightarrow \text{“_”} \quad 0.2$
 $S \rightarrow x \quad 0.3$
 $S \rightarrow S + S \quad 0.2$

...

Sequential 3-grams:

$S[x, \text{“_”}] \rightarrow \text{“.”} \quad 0.72$
 $S[x, \text{“_”}] \rightarrow \text{“_”} \quad 0.001$
 $S[x, \text{“_”}] \rightarrow x \quad 0.12$
 $S[x, \text{“_”}] \rightarrow S + S \quad 0.02$

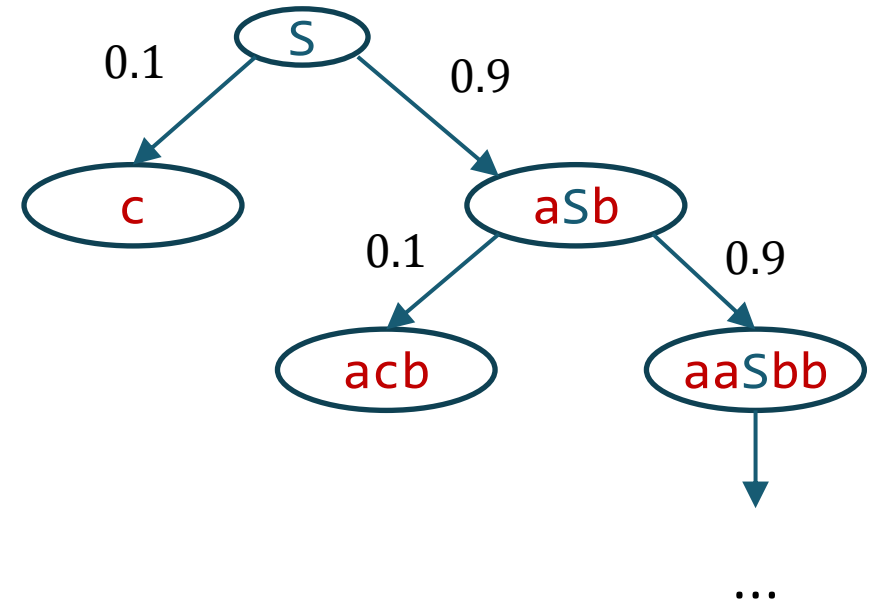
...

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?

$S \rightarrow a S b \quad 0.9$
 $S \rightarrow c \quad 0.1$



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

$n1 = \text{Rep}(\text{"-"}, \text{"."}, x)$ $n2 = \text{"-"} \quad pts = [\text{"-."}]$

$n1 = x + 1 + S$ $n1 = x + 2 + S$ $pts = [1]$

$n1 = \text{"-"} + \text{"."}$ $n1 = \text{"-"} + S$ $pts = [\text{"-."}]$

$n1 = \text{Rep}(\text{"."}, \text{"."}, S)$ $n2 = S$

$n1 = 5 + S$ $n2 = 3 + S + 2$

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

- Abstraction is key to learning models of code!

Extend observational equivalence to top-down search

Euphony: weaknesses

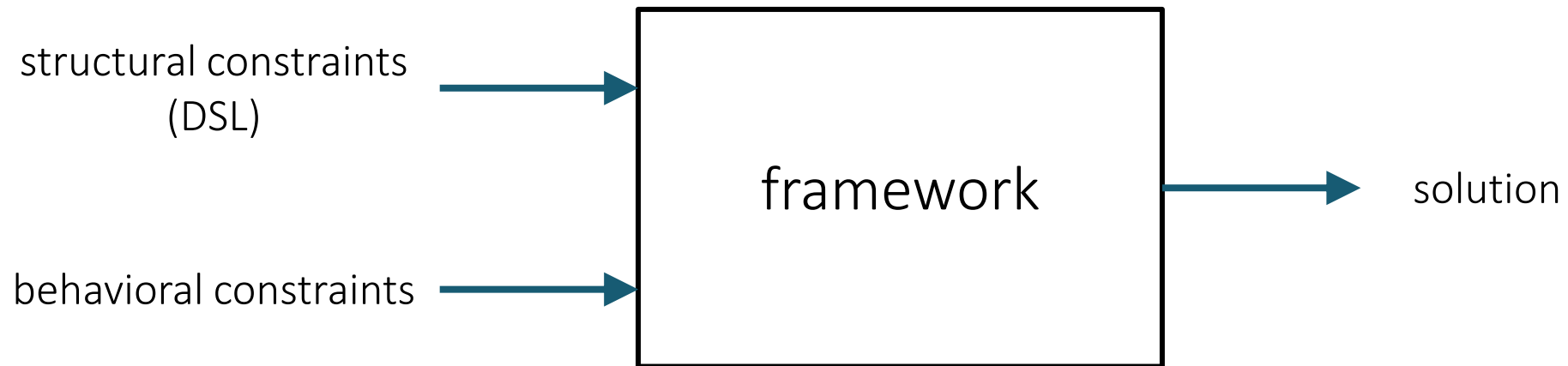
Requires high-quality training data

- for each problem domain!

Transfer learning requires manually designed features

Synthesis frameworks

synthesis framework = a highly-configurable synthesizer



Synthesis frameworks

Sketch (<https://people.csail.mit.edu/asolar/>)

Rosette (<https://emina.github.io/rosette/>)

- see also: <https://www.cs.utexas.edu/~bornholt/post/building-synthesizer.html>

PROSE (<https://www.microsoft.com/en-us/research/project/prose-framework/>)

Sketch

[Solar-Lezama 2013]

Problem: isolate the least significant zero bit in a word

- example: 0010 0101 → 0000 0010

Easy to implement with a loop

```
int W = 32;
bit[W] isolate0 (bit[W] x) {      // W: word size
    bit[W] ret = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { ret[i] = 1; return ret; }
}
```

Can this be done more efficiently with bit manipulation?

- Trick: adding 1 to a string of ones turns the next zero to a 1
- i.e. 000111 + 1 = 001000

Sketch: space of possible implementations

```
/**
 * Generate the set of all bit-vector expressions
 * involving +, &, xor and bitwise negation (~).
 */

generator bit[W] gen(bit[W] x){
    if(??) return x;
    if(??) return ??;
    if(??) return ~gen(x);
    if(??){
        return { | gen(x) (+ | & | ^) gen(x) | };
    }
}
```

Sketch: synthesis goal

```
generator bit[W] gen(bit[W] x, int depth){
    assert depth > 0;
    if(??) return x;
    if(??) return ??;
    if(??) return ~gen(x, depth-1);
    if(??){
        return { | gen(x, depth-1) (+ | & | ^) gen(x, depth-1) | };
    }
}

bit[W] isolate0fast (bit[W] x) implements isolate0 {
    return gen(x, 3);
}
```

Sketch: output

```
bit[W] isolate0fast (bit[W] x) {  
    return (~x) & (x + 1);  
}
```

Rosette

[Torlak, Bodik 2014]

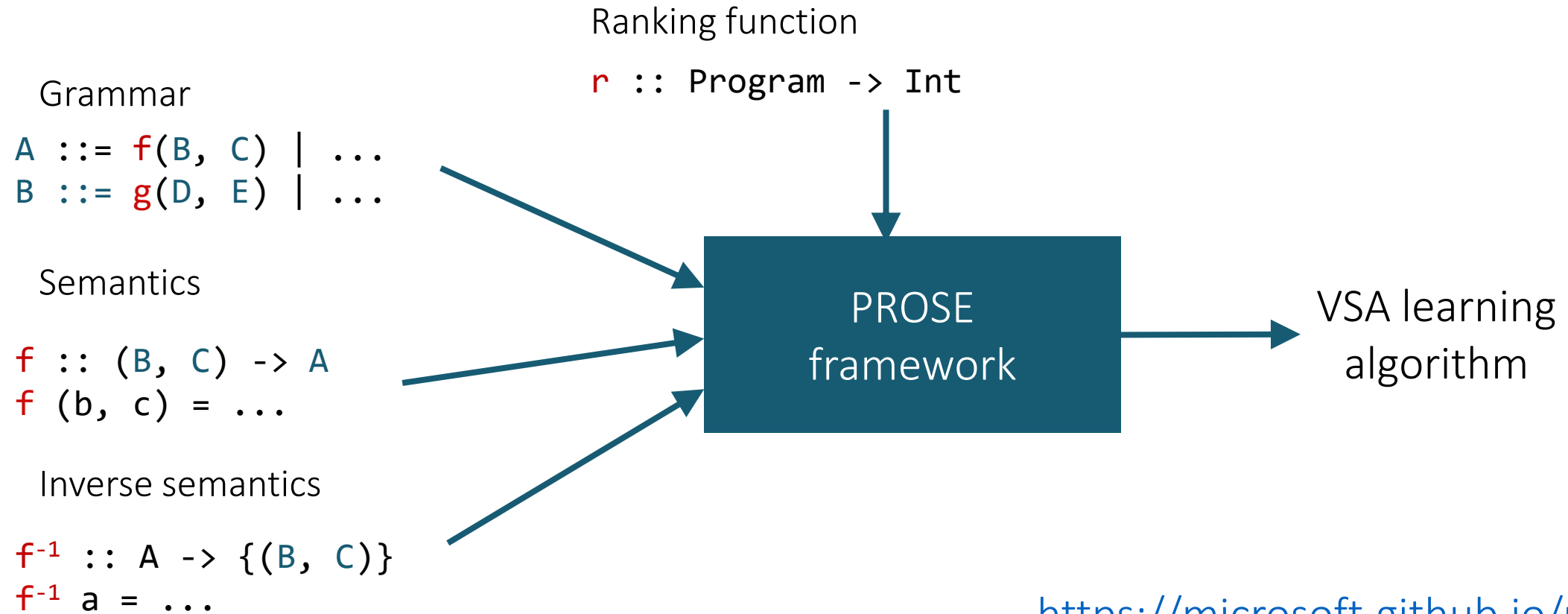
A solver-aided language on top of Racket

- Racket's metaprogramming + symbolic variables + solver queries
- Can define full-fledged SDSLs (Solver-aided DSLs)

<https://emina.github.io/rosette/>

PROSE

[Polozov, Gulwani '15]



Next week

Topics:

- Representation-based search
- Stochastic search

Paper: Rishabh Singh: [BlinkFill: Semisupervised Programming By Example for Syntactic String Transformations](#). VLDB'16

Projects:

- 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