Lecture 4
Biasing Enumerative Search

Plan for this week

Today:

- Search space prioritization/biasing

Next lecture:

- Discuss the Euphony paper
- Synthesis frameworks + suggested projects

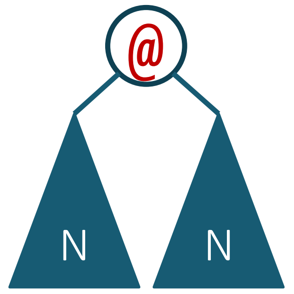
Project:

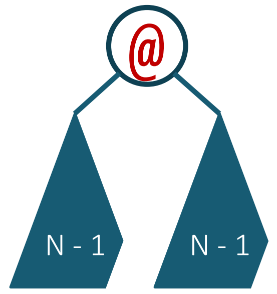
- Proposals due soon
- Talk to me about the topic

Scaling enumerative search

Prune

Discard useless subprograms





`m * N`^{2} `m * (N - 1)`^{2}

Prioritize

Explore more promising candidates first

```
P = { [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

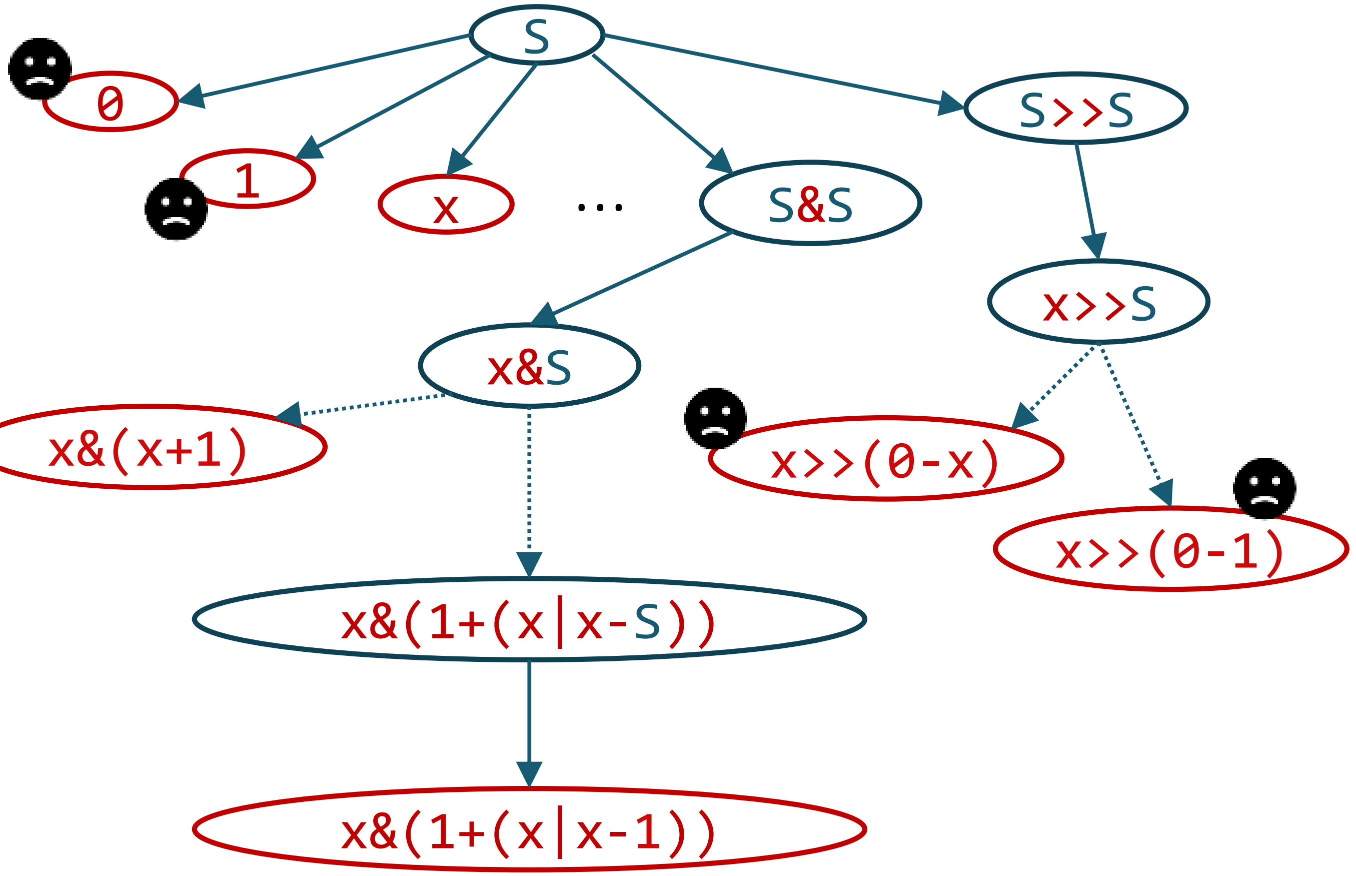
Top-down search (revisited)

Turn off the rightmost sequence of **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?

- hard-code domain knowledge
- learn from a corpus of programs

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

- our focus today!

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

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
```

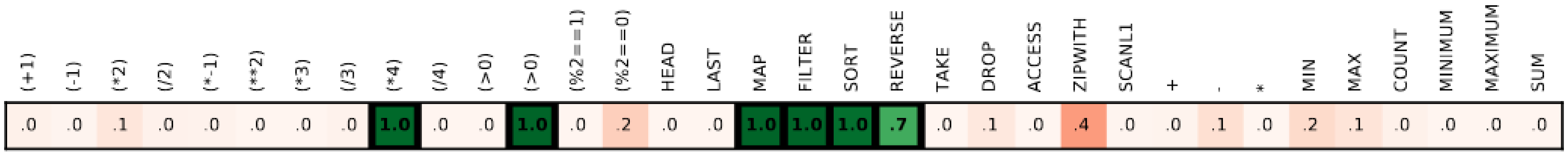
DeepCoder

Input: IO-examples

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

neural network

component likelihoods



(+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

DeepCoder: search strategies

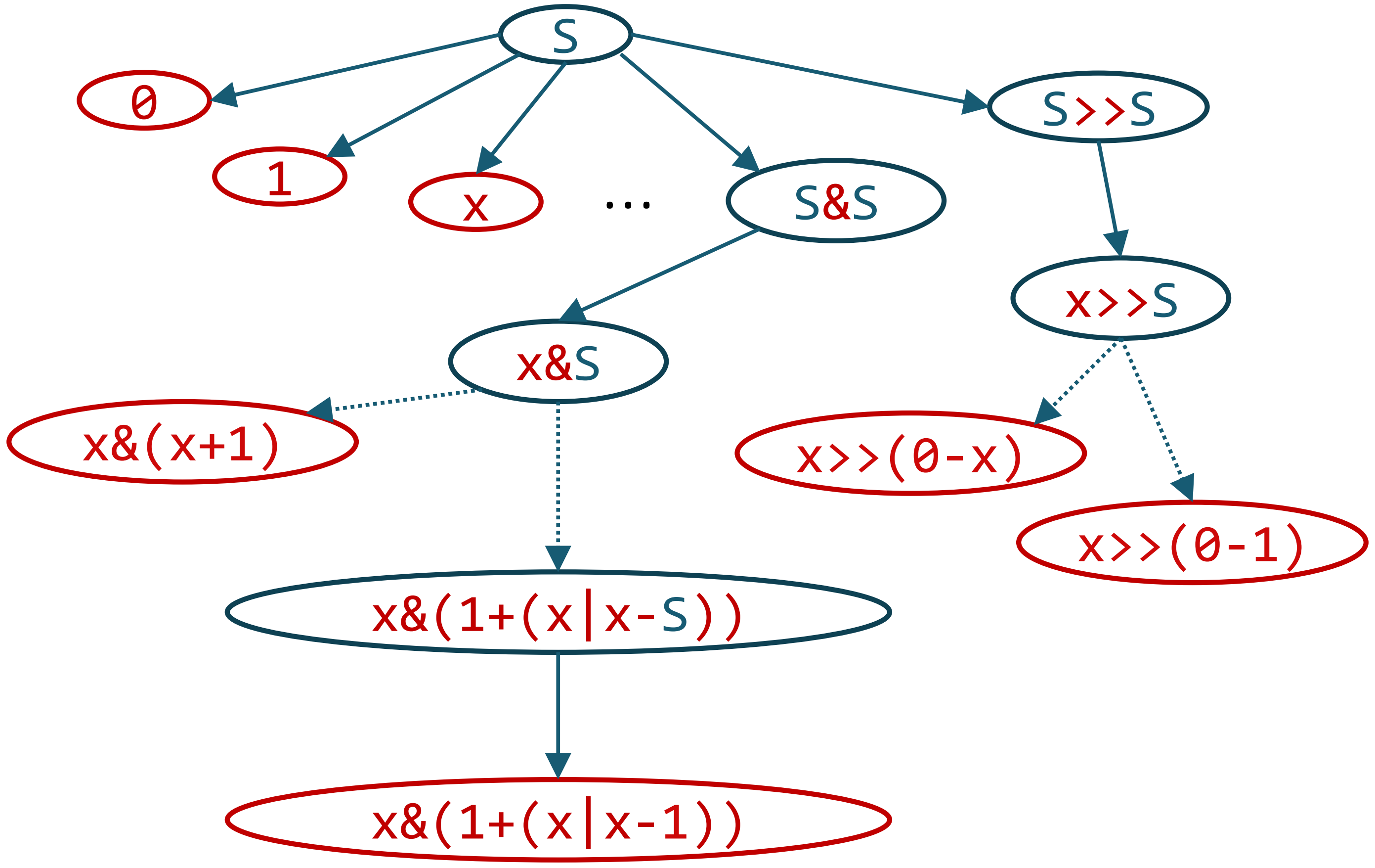
Top-down DFS

- 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: we want to explore programs in the order of likelihood!

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: grammar-based (PCFG, PHOG)

Probabilistic CFG (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

Encodes the popularity of each operation (terminal)

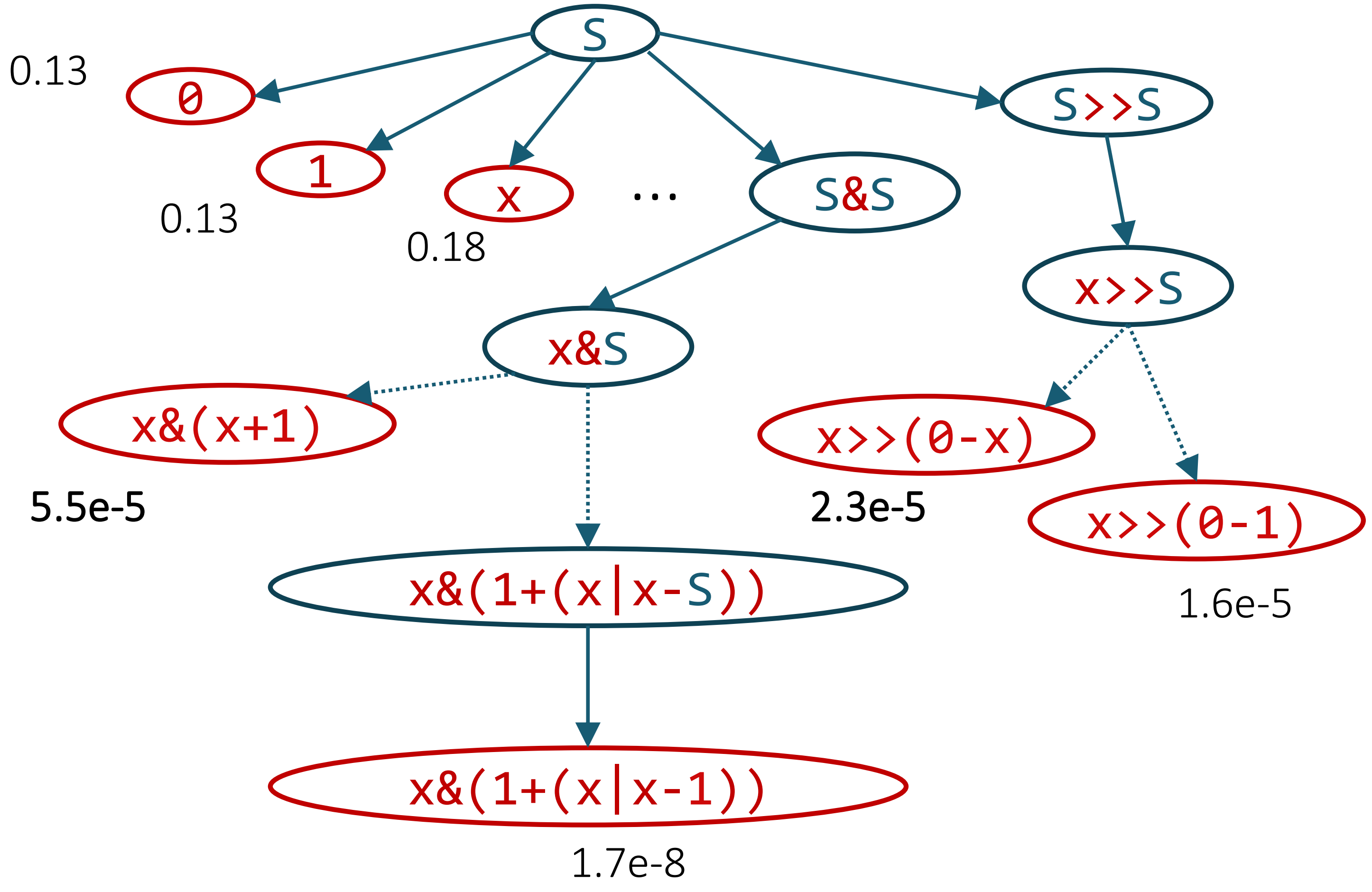
- here: variable more likely than constant, plus more likely than shift

More useful if specific to a spec

Probabilistic CFG (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

$$\wp(p) = \prod_{R \in S \to^* p} \wp(R)$$



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

Encodes context-specific likelihood

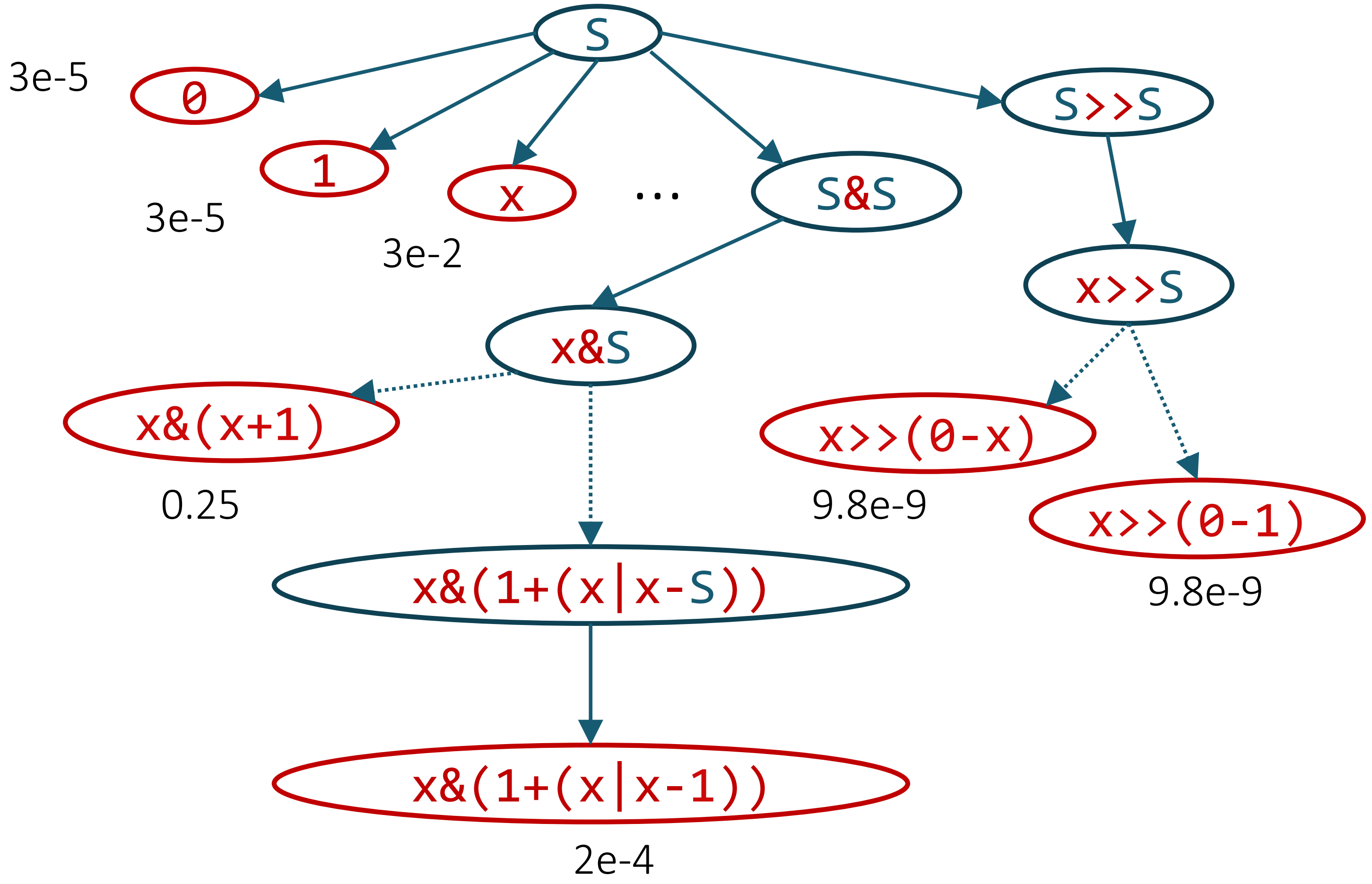
- here: **`x`** is not likely in **`x - ?`** but likely in **`1 + ?`**

Probabilistic Higher-Order Grammar (PHOG)

[Bielik, Raychev, Vechev '16]

```
N[context] -> 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



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. arXiv

Weighted top-down search

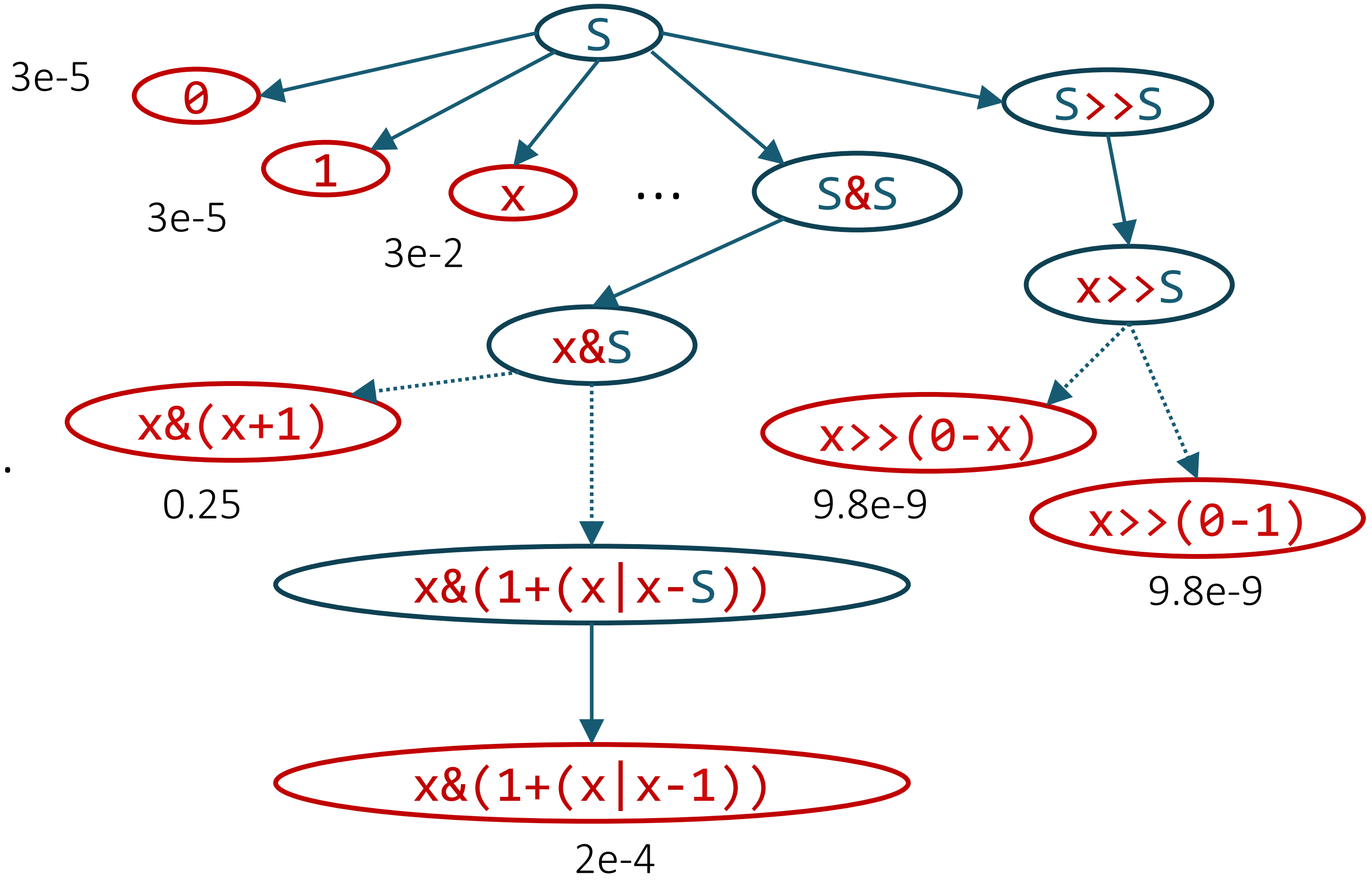
Wanted: explore programs in the order of likelihood

$$\wp(p) = \prod_{R \in S \to^* p} \wp(R)$$

Hard to maximize multiplicative cost…
but easy to minimize additive cost!
= shortest path

$$cost(p) = \sum_{R \in S \to^* p} cost(R)$$

$$-\log_2 \wp(p) = \sum_{R \in S \to^* p} -\log_2 \wp(R)$$



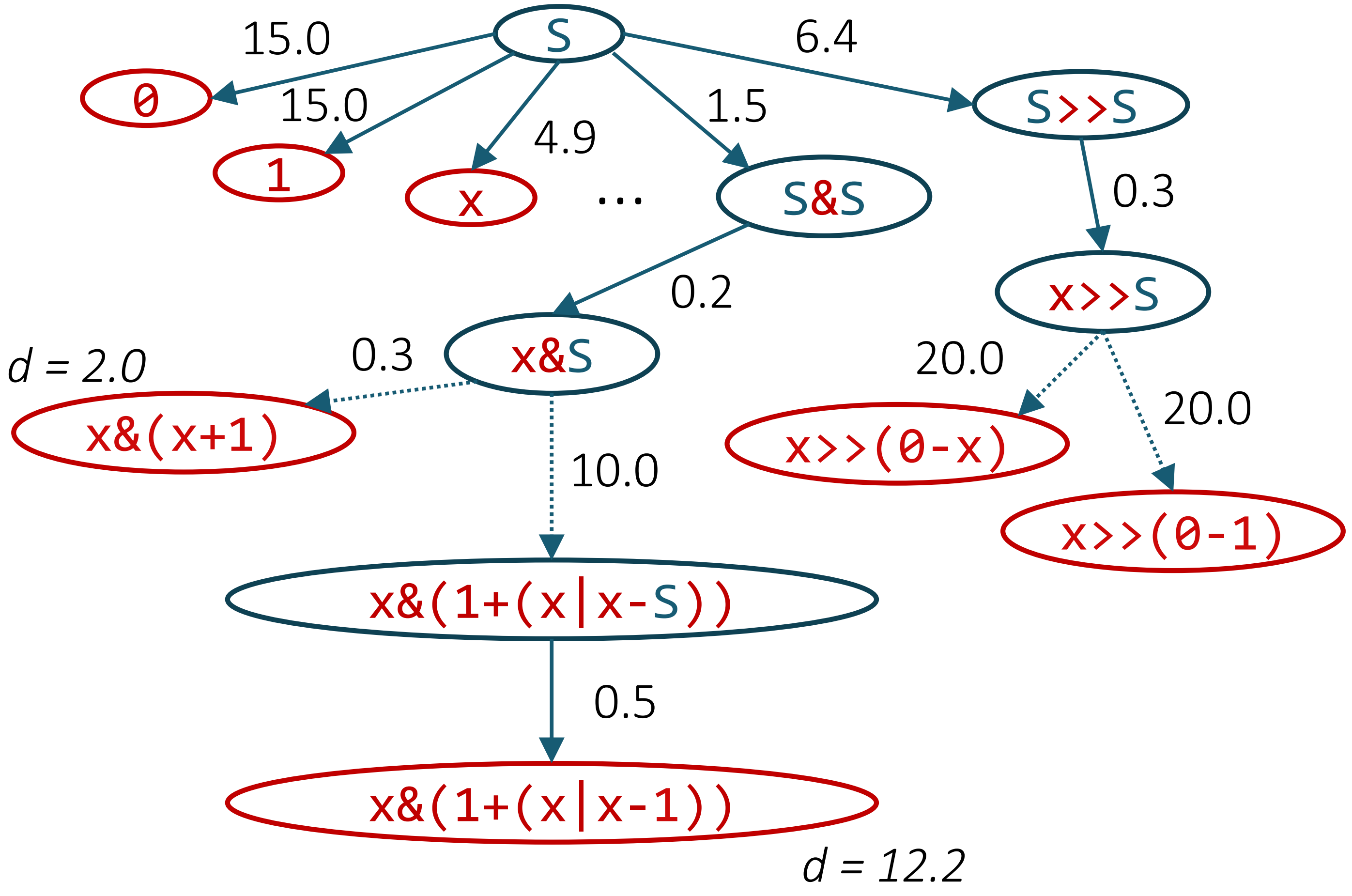
Weighted top-down search

Assigns costs to edges:

$$cost(R) = -\log_2 \wp(R)$$

Now *cost(p)* < *cost(p')*
iff *p* is more likely than *p'*!

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



Weighted top-down search (Dijkstra)

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

unroll(p,c) {
  wl' := []
  A := left-most nonterminal in p
  forall (A → rhs) in R:
    wl' += <p[A -> rhs], c + w(A → rhs)>
  return wl';
}
```

wl now stores candidates (nodes) together with their costs

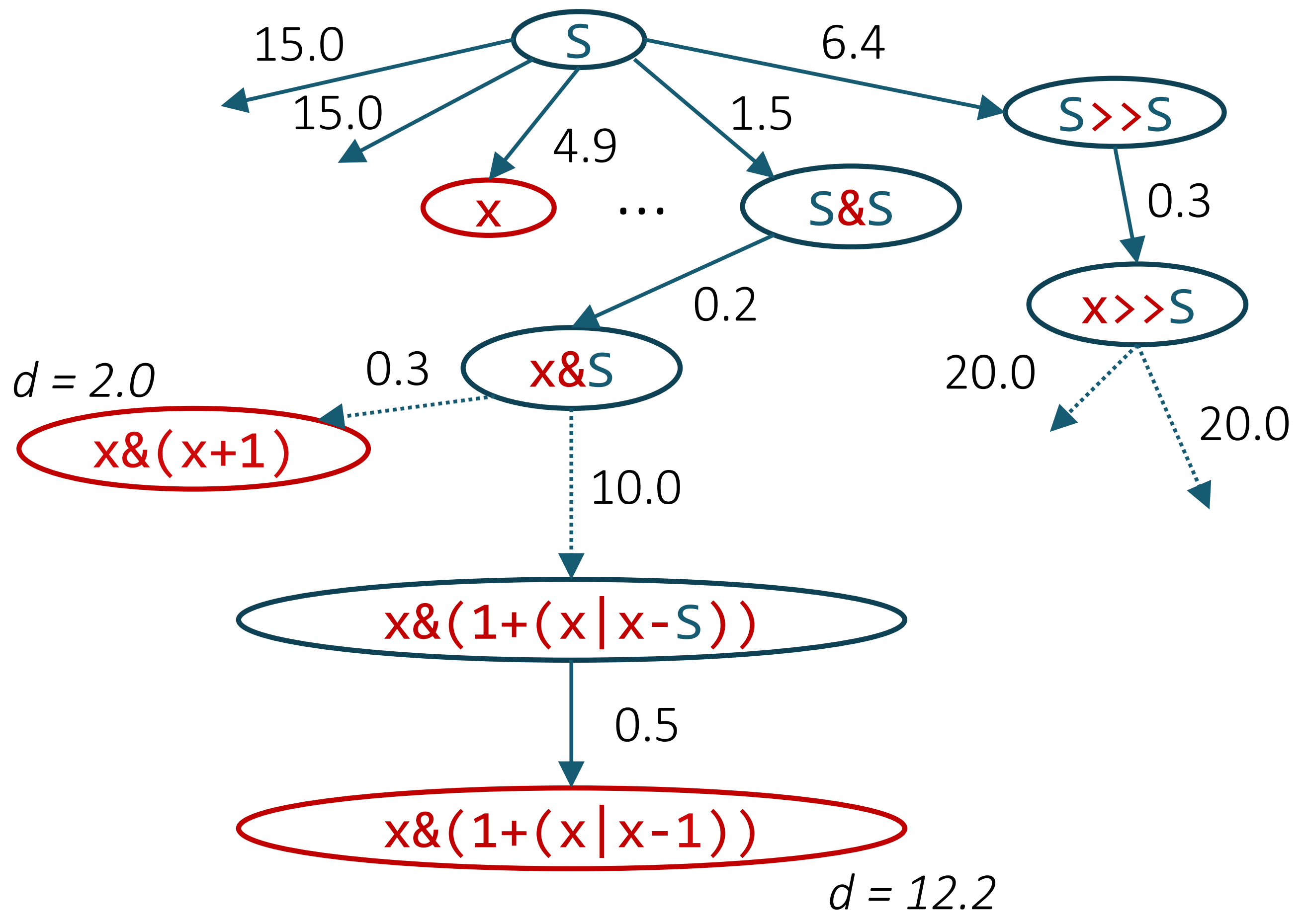
Dequeue the node with minimal cost

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

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

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

Roughly how close is this program to the closest leaf

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

Bottom-up search (revisited)

```
bottom-up (<Σ, N, R, S>, [i → o]):
  bank[A,d] := {} forall A, d
  for d in [0..]:
    forall (A → rhs) in R:
      forall p in new-terms(A→rhs, d, bank):
        if (A = S ∧ p([i]) = [o]):
          return p
        bank[A,d] += p;

new-terms(A → σ(A_1…A_n), d, bank):
 if (d = 0 ∧ n = 0) yield σ
 else forall <d_1,…,d_n> in [0..d-1]^n s.t. max(d_1,…,d_n) = d-1:
        forall <p_1,…,p_n> in bank[A_1,d_1] × … × bank[A_n,d_n]:
          yield σ(p_1,…,p_n)
```

Search by depth

Bottom-up variations

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

by depth

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

by size

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

by cost!

Bottom-up by cost: discussion

What kind of cost functions are supported?

- positive
- integer
- compositional:
  `cost(σ(p₁,…,pₙ)) = C + cost(p₁) + … + cost(pₙ)`

$$\mathtt{cost}(\sigma(p_1,\ldots,p_n)) = C + \mathtt{cost}(p_1) + \ldots + \mathtt{cost}(p_n)$$

Bottom-up: example

```
                          cost
L ::= sort(L)        |    10
      L + L          |    3
      x                   1
```

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)))`
`x + sort(x)`
`sort(x) + x`

s = 5: …

by cost

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

[Barke, Peleg, Polikarpova. OOPSLA’20]

Q: can we guide bottom-up search by a PCFG? by a PHOG?

Weighted search

Top-down

+ Supports real-valued weights: optimal enumeration order

+ Supports context-dependent weights

Bottom-up

+ Inherits benefits of bottom up: fast, supports OE

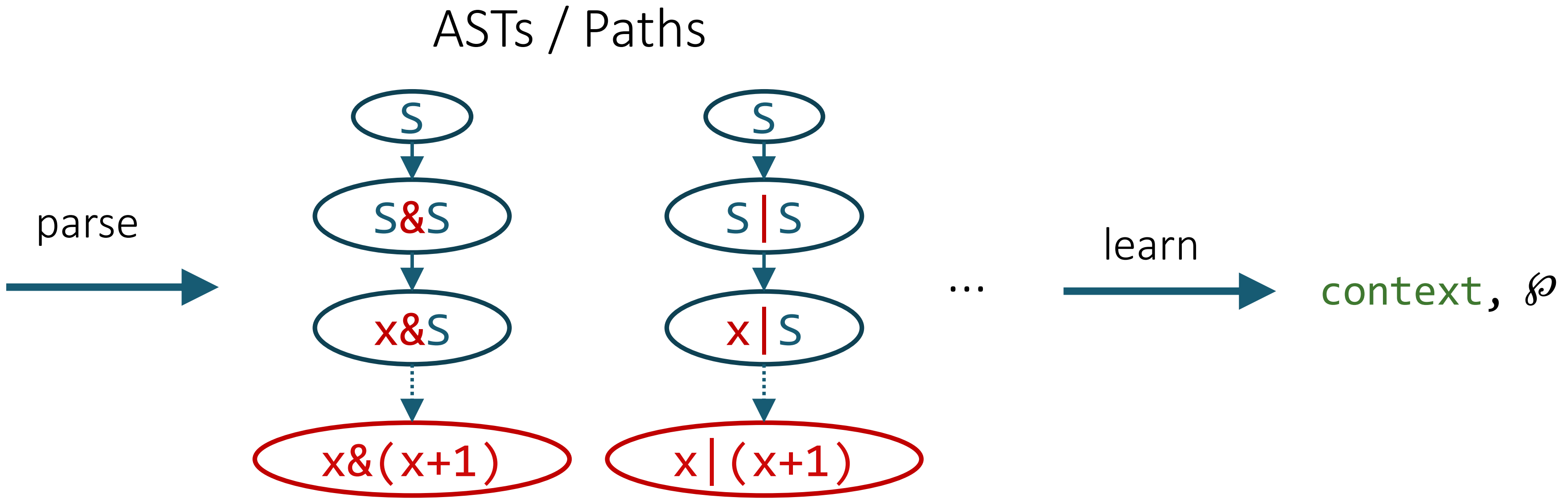
Learning PHOGs

[Bielik, Raychev, Vechev ‘16]

CFG +

Corpus

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



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

Rep x “-” S

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

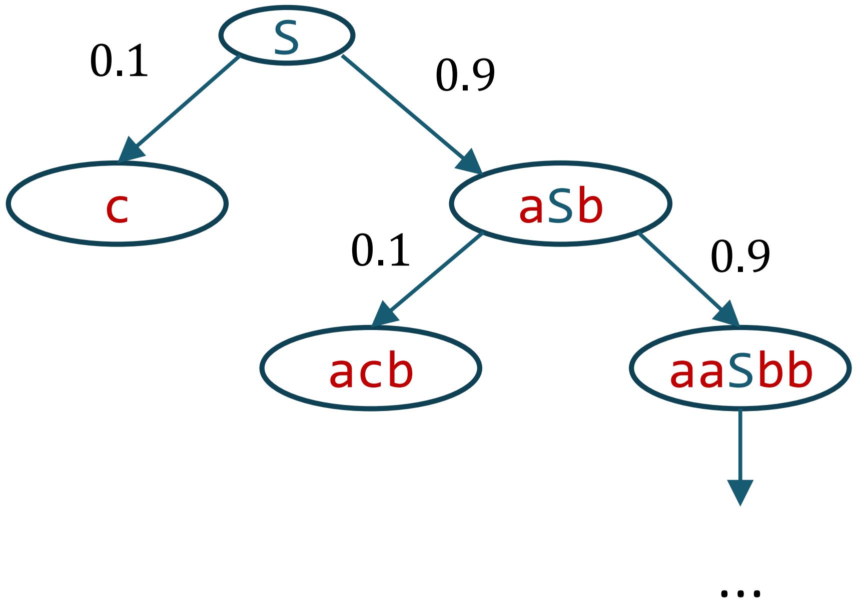
```
S[x,“-”] ->  “.”     0.72
S[x,“-”] ->  “-”     0.001
S[x,“-”] ->  x       0.12
S[x,“-”] ->  S + S   0.02
…
```

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

```
pts = []
```

```
n1 = x + “-” n2 = “-” + x
pts = [“-”, “--”]
```

```
n1 = Rep(x,x,S) n2 = S
```

```
n1 = Rep(S+x,”.”,”-”) n2 = Rep(S,”.”,”-”) + Rep(x,”.”,”-”)
```

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

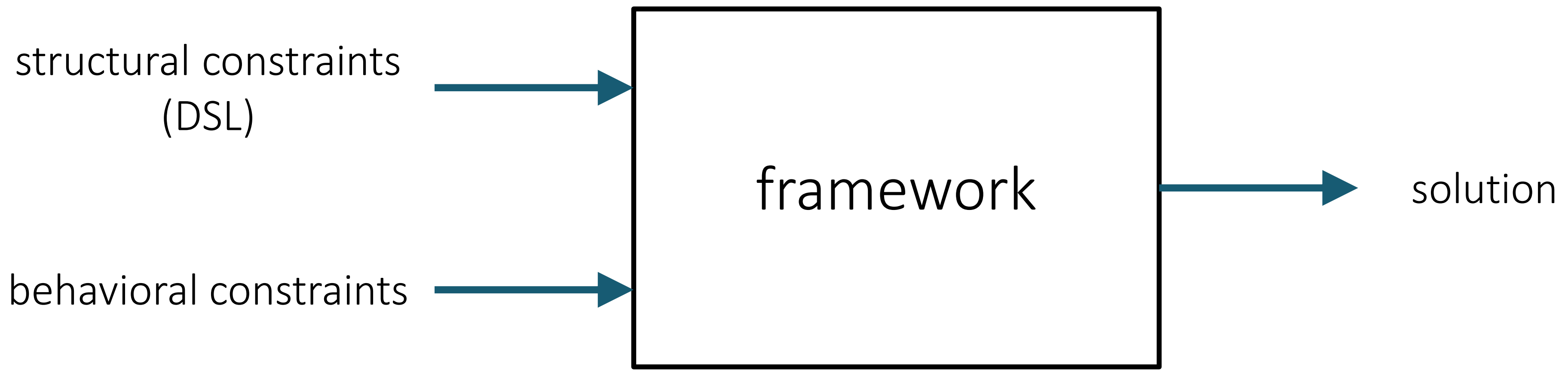
Requires high-quality training data

- 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/)

SemGuS (https://www.semgus.org/)

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)

Let’s see how to solver the same problem in Rosette

PROSE

[Polozov, Gulwani '15]

Ranking function

```
r :: Program -> Int
```

Grammar

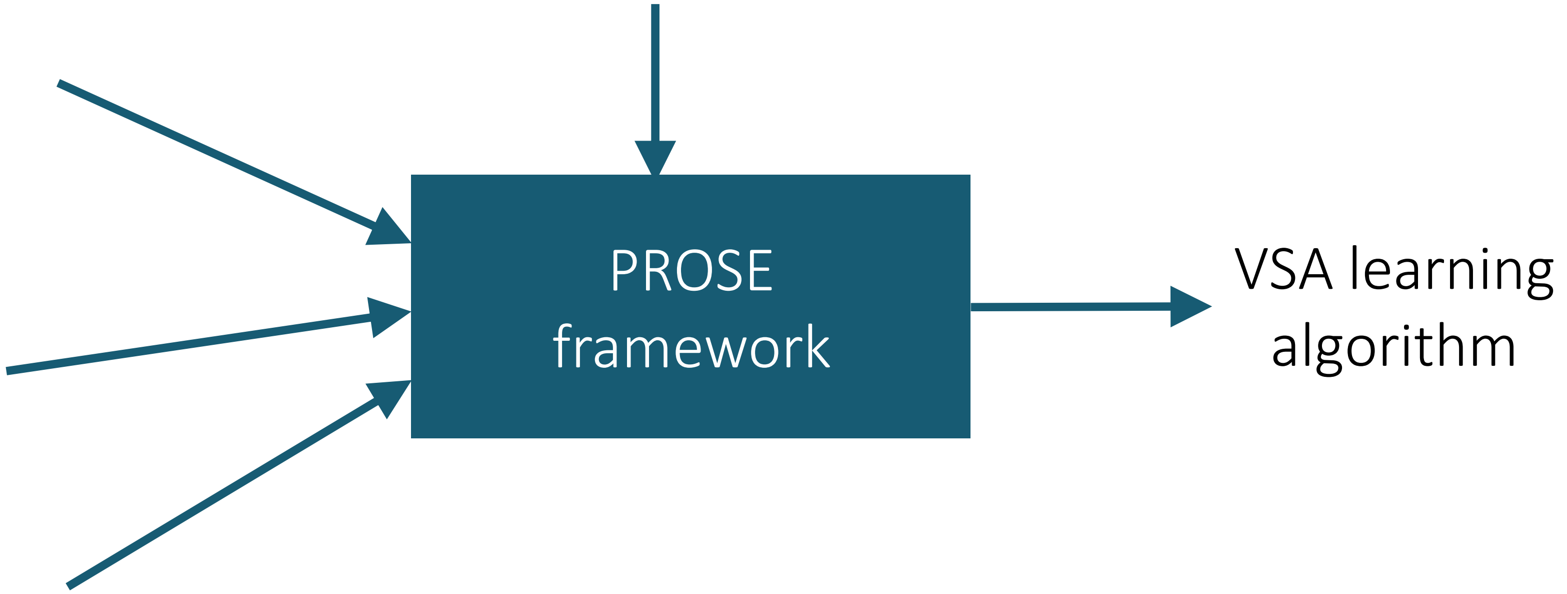
```
A ::= f(B, C) | ...
B ::= g(D, E) | ...
```

Semantics

```
f :: (B, C) -> A
f (b, c) = ...
```

Inverse semantics

```
f⁻¹ :: A -> {(B, C)}
f⁻¹ a = ...
```

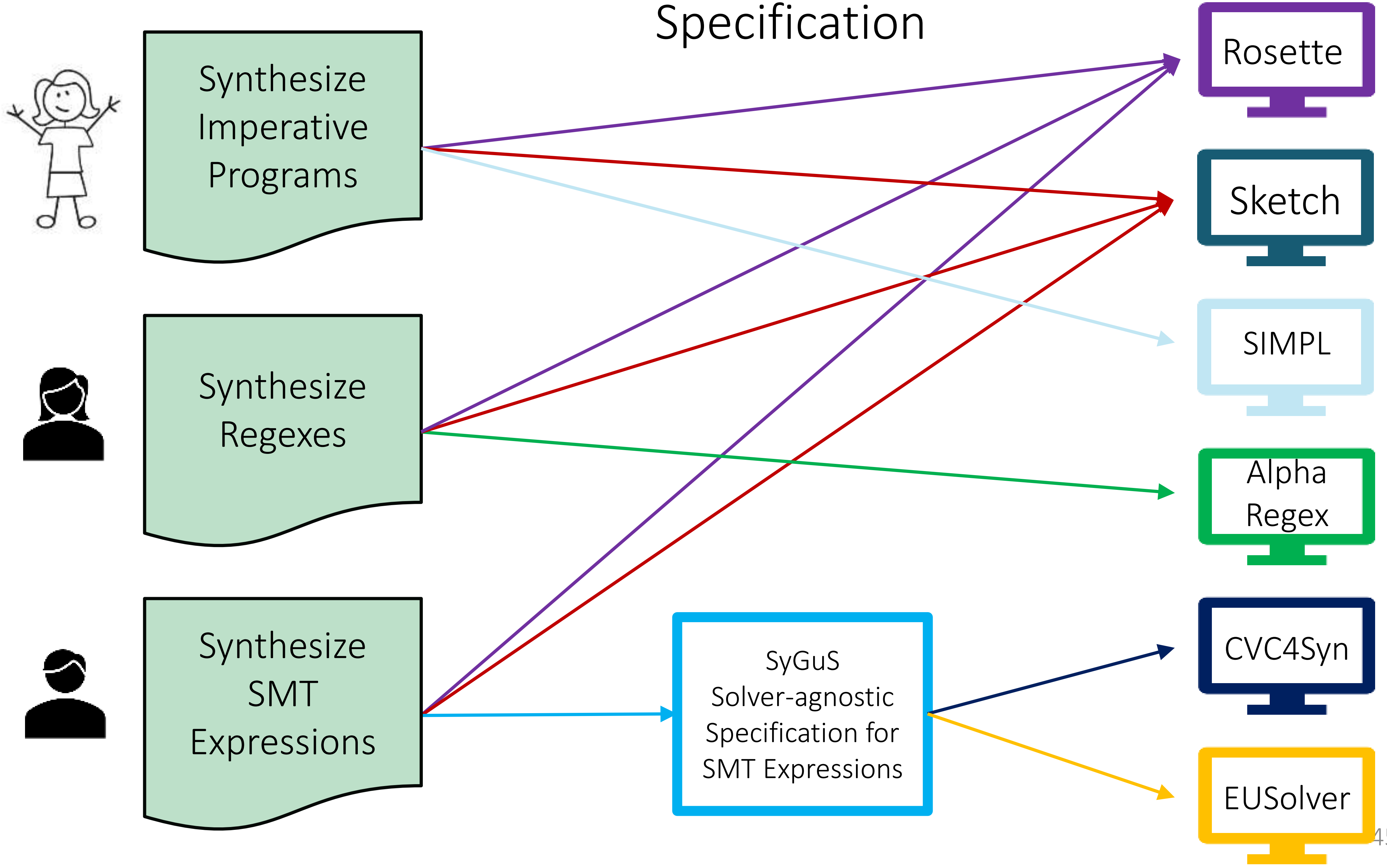


https://microsoft.github.io/prose/

SemGuS

Semantics-guided synthesis

Specification

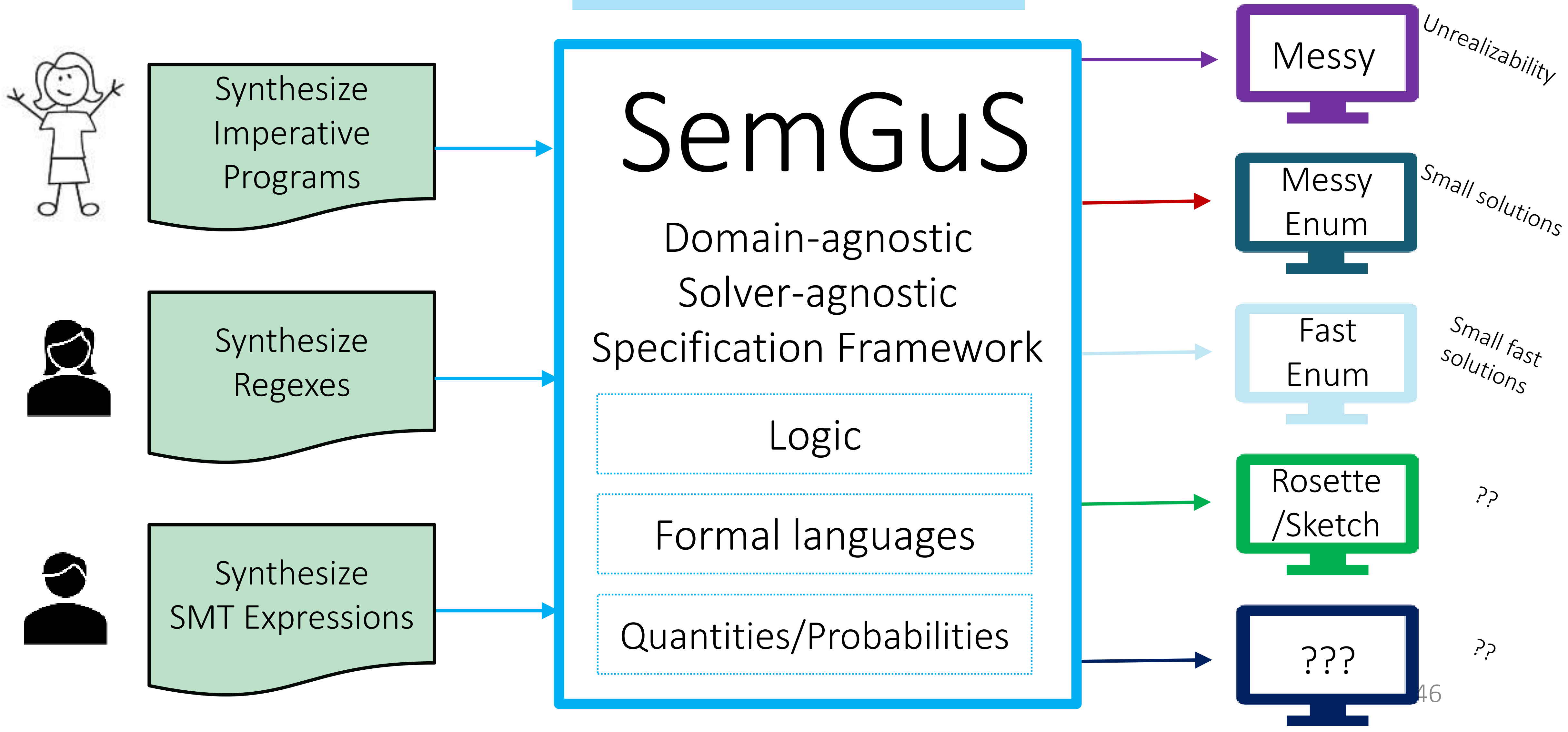


Synthesize Imperative Programs
Synthesize Regexes
Synthesize SMT Expressions
SyGuS Solver-agnostic Specification for SMT Expressions
Rosette
Sketch
SIMPL
Alpha Regex
CVC4Syn
EUSolver

Unified benchmarks!

semgus.org

SemGuS Competition!



Next lectures

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, send me message (one team member)
- If you haven't decided, talk to me after class