#8: Weighted Enumerative Search - II

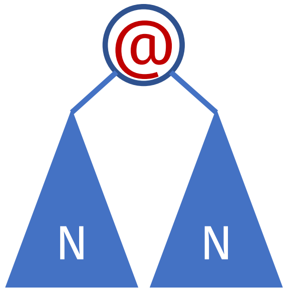
Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

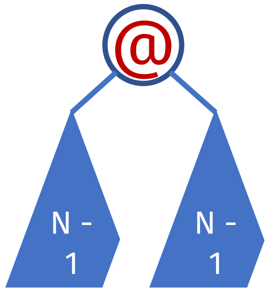
Scaling enumerative search

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

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

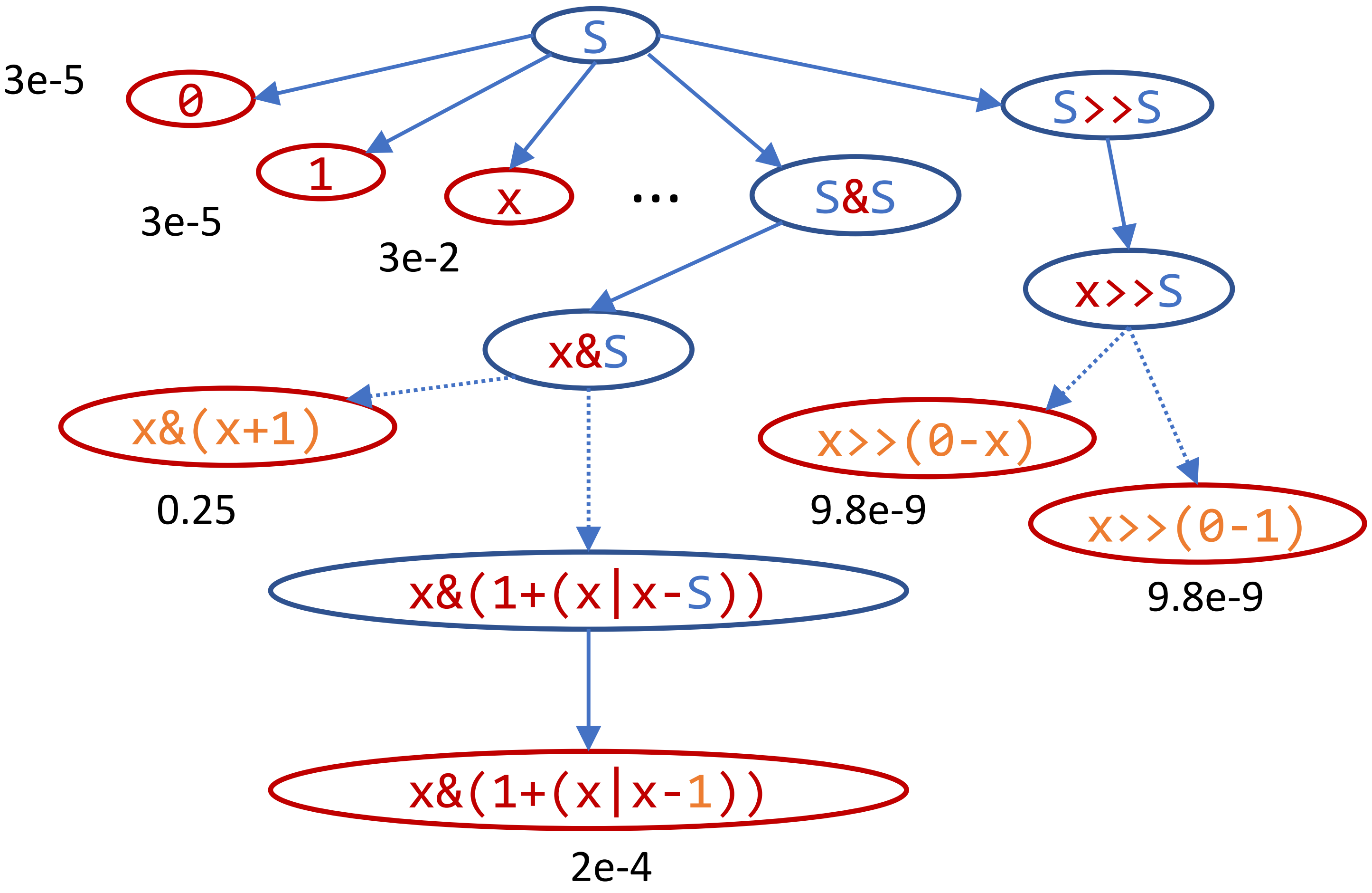
$$\wp(t) = \prod_{(r_i,\tau_i)\in S\to^* t} \wp(r_i \mid \tau_i)$$

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

= shortest path

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

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



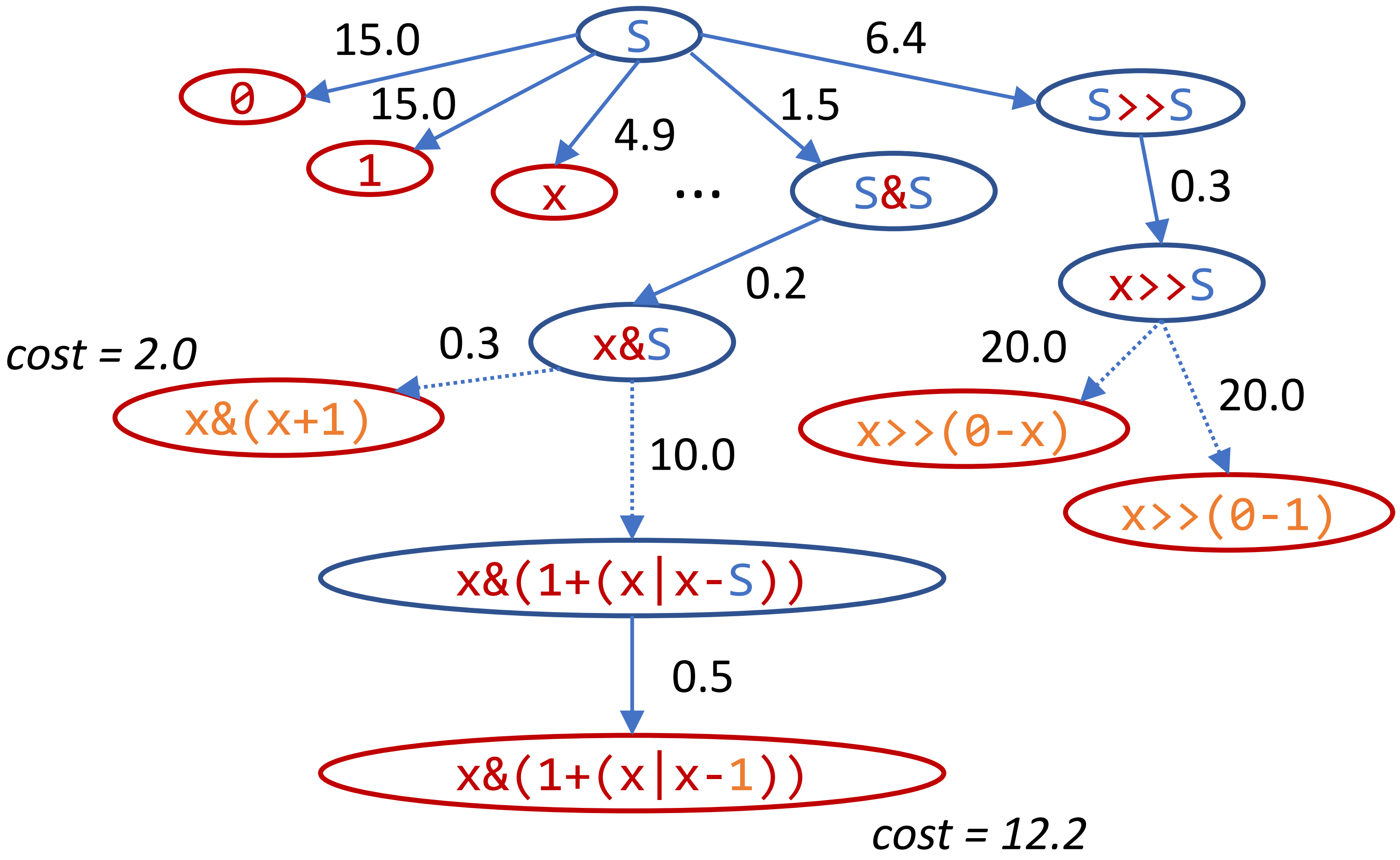
Weighted top-down search

Assigns weights to edges:

$$weight(r_i \mid \tau_i) = -\log_2 \wp(r_i \mid \tau_i)$$

Now *cost(t)* < *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(<Σ, N, R, S>, [i → o]) {
  wl := [<S,0>]
  while (wl != [])
    <τ,c> := wl.dequeue_min(c);
    if (complete(τ) && τ([i]) = [o])
      return τ;
    wl.enqueue(unroll(τ,c));
}

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

wl now stores candidates (nodes) together with their costs

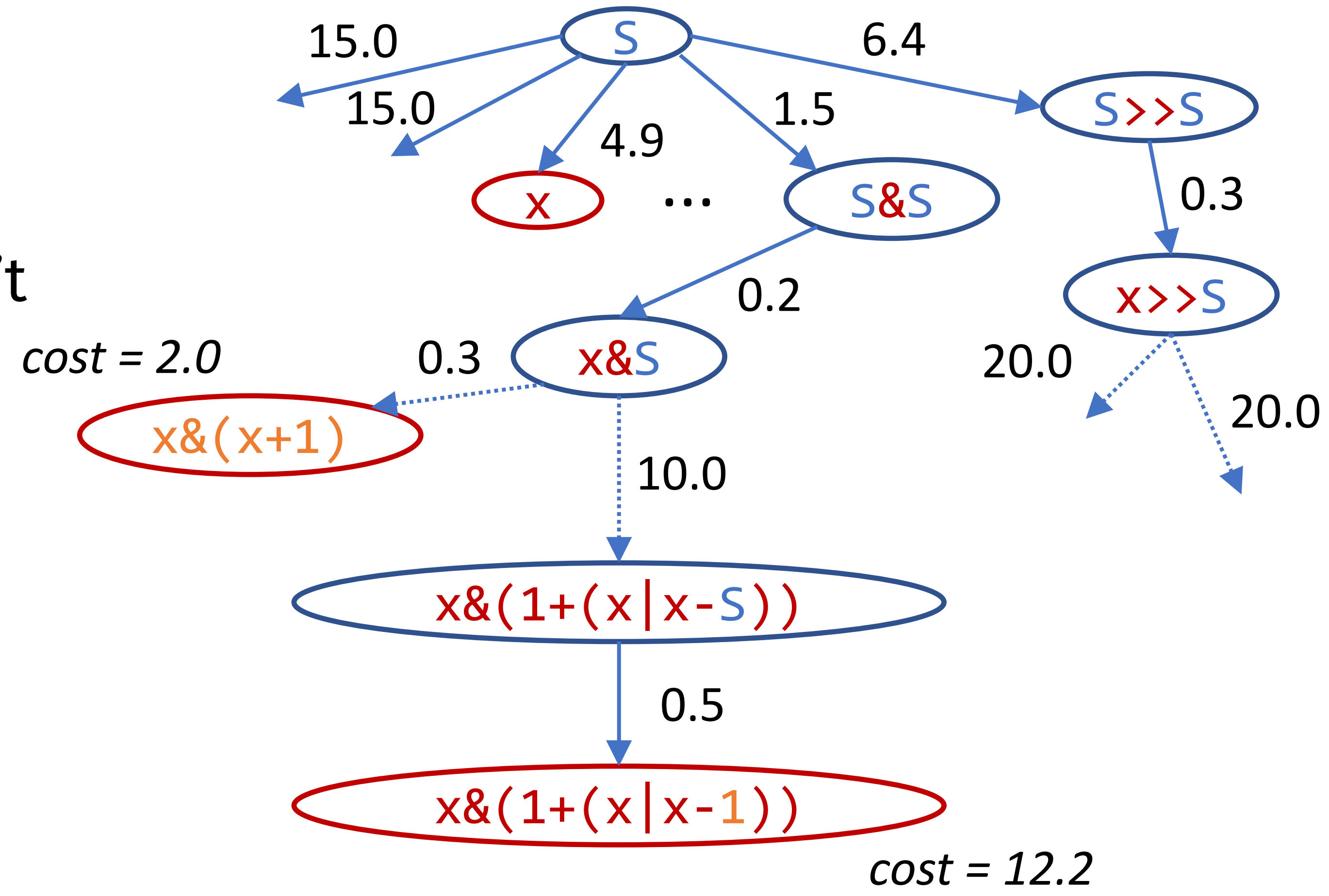
Dequeue the node with minimal cost

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 != [])
    <τ,c,h> := wl.dequeue_min(c + h);
    if (complete(τ) && τ([i]) = [o])
      return τ;
    wl.enqueue(unroll(τ,c));
}

unroll(τ,c) {
  wl' := []
  A := leftmost nonterminal in τ
  forall (A → rhs) in R:
    wl' += <τ[A -> rhs], c + w(A → rhs|τ),
                          h(τ[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’22

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(A₁…Aₙ), d, bank):
 if (d = 0 ∧ n = 0) yield σ
 else forall <d₁,…,dₙ> in [0..d-1]ⁿ s.t. max(d₁,…,dₙ) = d-1:
       forall <p₁,…,pₙ> in bank[A₁,d₁] × … × bank[Aₙ,dₙ]:
         yield σ(p₁,…,pₙ)
```

Search by depth

Bottom-up variations

```
new-terms(A → σ(A₁…Aₙ), d, bank):
 if (d = 0 ∧ n = 0) yield σ
 else forall <d₁,…,dₙ> in [0..d-1]ⁿ s.t. max(d₁,…,dₙ) = d-1:
        forall <p₁,…,pₙ> in bank[A₁,d₁] × … × bank[Aₙ,dₙ]:
          yield σ(p₁,…,pₙ)
```

by depth

```
new-terms(A → σ(A₁…Aₙ), s, bank):
 if (s = 1 ∧ n = 0) yield σ
 else forall <s₁,…,sₙ> in [0..s-1]ⁿ s.t. sum(s₁,…,sₙ) = s-1:
        forall <p₁,…,pₙ> in bank[A₁,s₁] × … × bank[Aₙ,sₙ]:
          yield σ(p₁,…,pₙ)
```

by size

```
new-terms(A → σ(A₁…Aₙ), c, bank):
 budget = c - w(A → σ(A₁…Aₙ))
 if (budget = 0 ∧ n = 0) yield σ
 else forall <c₁,…,cₙ> in [0.. budget]ⁿ s.t. sum(c₁,…,cₙ) = budget:
        forall <p₁,…,pₙ> in bank[A₁,c₁] × … × bank[Aₙ,cₙ]:
          yield σ(p₁,…,pₙ)
```

by cost!

Bottom-up by cost: discussion

- What kind of cost functions are supported?
  - positive
  - integer
  - context-free

Bottom-up: example

cost

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

by depth		by size		by cost	
d= 0:	`x`	s= 1:	`x`	c= 1:	`x`
d =1:	`sort(x)`	s =2:	`sort(x)`	c =2,3,4:	
	`x + x`	s = 3:	`x + x`	c = 5:	`x + x`
d = 2:	`sort(sort(x))`		`sort(sort(x))`	c =6,7,8:	
	`sort(x + x)`	s = 4:	`sort(x + x)`	c = 9:	`x + (x + x)`
	`x + sort(x)`		`sort(sort(sort(x)))`		`(x + x) + x`
	`sort(x) + x`		`x + sort(x)`	c = 10:	
	`x + (x + x)`		`sort(x) + x`	c = 11:	`sort(x)`
	`(x + x) + x`	s = 5:	…	c = 12:	
d = 3:	…			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, observational equivalence