

#4: Equivalence Reduction

Sankha Narayan Guria

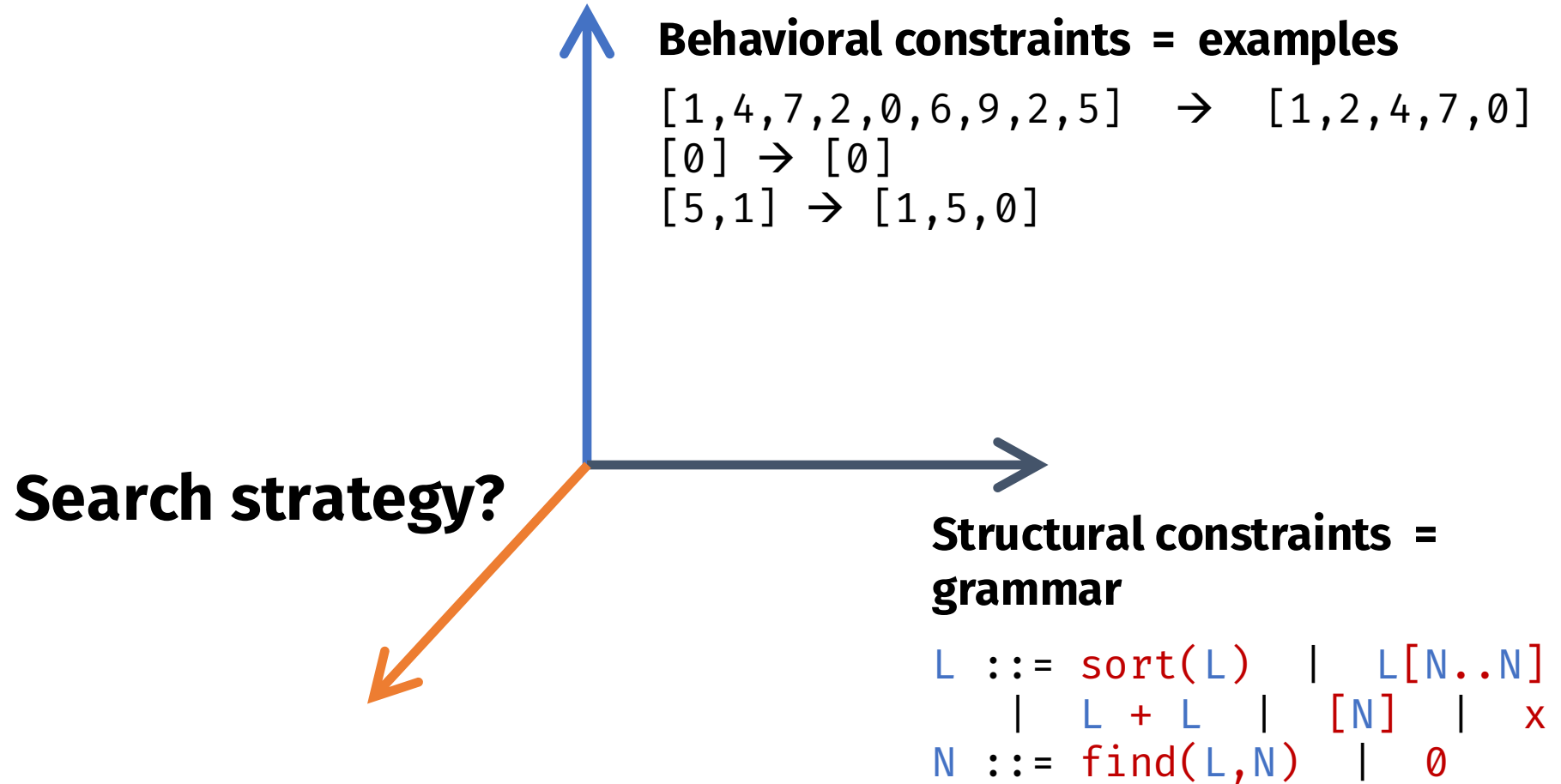
EECS 700: Introduction to Program Synthesis



Logistics

- Paper 2 assignment released!
- Demo code available at: <https://github.com/ku-progsys/eecs700-demos>

The problem statement



Enumerative search

=

Explicit / Exhaustive Search

Idea: Enumerate programs from the grammar one by one and test them on the examples

bottom-up

L ::= sort(L)

L[N..N]

L + L

[N]

x

N ::= find(L, N)

0

top-down

x 0

sort(x) x[0..0] x + x [0]

find(x, 0)

sort(sort(x)) sort(x[0..0])

sort(x + x) sort([0])

x[0..find(x, 0)] ...

L

x sort(L) L[N..N] L + L [N]

sort(x) sort(sort(L)) sort([N])

sort(L[N..N]) sort(L + L)

x[N..N] (sort L)[N..N] ...

Bottom-up vs top-down

Top-down

Bottom-up

Smaller to larger depth

- Has to explore between $3 \cdot 10^9$ and 10^{23} programs to find `sort(x[0..find(x, 0)]) + [0]` (depth 6)

Candidates are **whole** but might not be **complete**

- Cannot always run on inputs
- Can always relate to outputs (?)

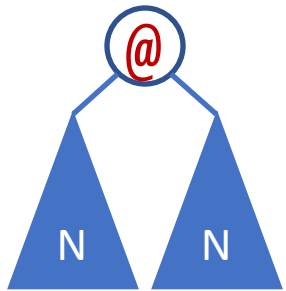
Candidates are **complete** but might not be **whole**

- Can always run on inputs
- Cannot always relate to outputs

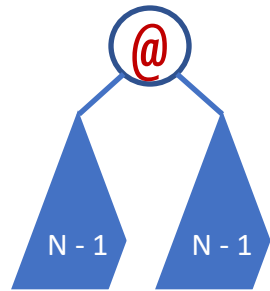
How to make it scale

Prune

- Discard useless subprograms



$$m * N^2$$



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

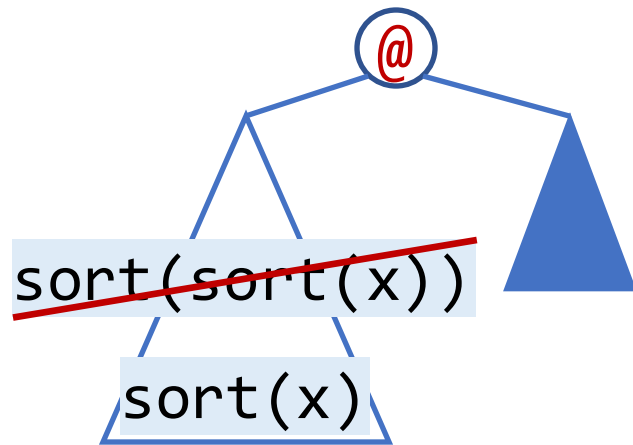
Prioritize

- Explore promising candidates first

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

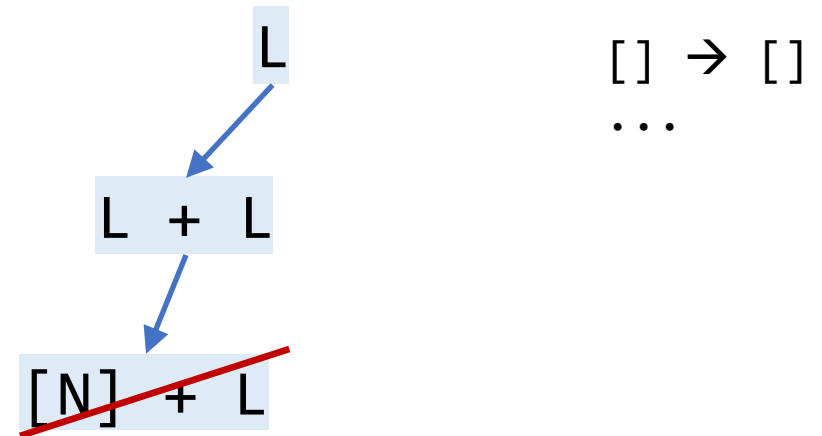
When can we discard a subprogram?

redundant



Equivalence reduction
(also: symmetry breaking)

infeasible



Top-down propagation

Equivalent programs

```

L ::= sort(L)
    L[N..N]
    L + L
    [N]
N ::= x
    find(L,N)
    0
  
```

bottom_up
→

```

x  0
sort(x)  x[0..0]  x + x  [0]  find(x,0)

sort(sort(x))  sort(x + x)  sort(x[0..0])
sort([0])  x[0..find(x,0)]  x[find(x,0)..0]
x[find(x,0)..find(x,0)]  sort(x)[0..0]
x[0..0][0..0]  (x + x)[0..0]  [0][0..0]
x + (x + x)  x + [0]  sort(x) + x  x[0..0] + x
(x + x) + x  [0] + x  x + x[0..0]  x + sort(x)
...
  
```

Equivalent programs

```

L ::= sort(L)
    L[N..N]
    L + L
    [N]
x
N ::= find(L,N)
    0
  
```

bottom_up
→

x 0

sort(x) x[0..0] x + x [0] find(x,0)

sort(sort(x)) sort(x + x) sort(x[0..0])

sort([0]) x[0..find(x,0)] x[find(x,0)..0]

x[find(x,0)..find(x,0)] sort(x)[0..0]

x[0..0][0..0] (x + x)[0..0] [0][0..0]

x + (x + x) x + [0] sort(x) + x x[0..0] + x

(x + x) + x [0] + x x + x[0..0] x + sort(x)

...

Equivalent programs

```

L ::= sort(L)
    L[N..N]
    L + L
    [N]
    x
N ::= find(L,N)
    0
  
```

bottom_up
→

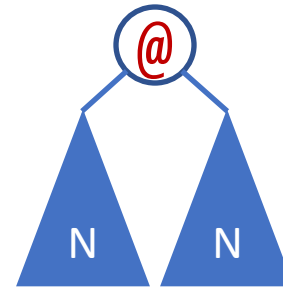
```

x  0
sort(x)  x[0..0]  x + x  [0]  find(x,0)
      sort(x + x)
      x[0..find(x,0)]
x + (x + x)  x + [0]  sort(x) + x
              [0] + x          x + sort(x)
...
  
```

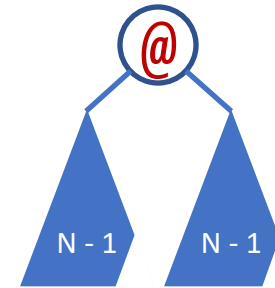
Bottom-up + equivalence reduction

```
bottom-up (<T, N, R, S>, [i → o]) {  
  bank := [t | A ::= t in R]  
  while (true)  
    forall (p in bank)  
      if (p([i]) = [o])  
        return p;  
    bank += grow(bank);  
}
```

```
grow (bank) {  
  bank' := []  
  forall (A ::= rhs in R)  
    bank' += [rhs[B -> p] | p in bank, B →* p]  
  return [p' in bank' | forall p in bank: !equiv(p, p')];  
}
```



$$m * N^2$$



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

Bottom-up + equivalence reduction

```
bottom-up (<T, N, R, S>, [i → o]) {  
  bank := [t | A ::= t in R]  
  while (true)  
    forall (p in bank)  
      if (p([i]) = [o])  
        return p;  
  bank += grow(bank);  
}
```

```
grow (bank) {  
  bank' := []  
  forall (A ::= rhs in R)  
    bank' += [rhs[B -> p] | p in bank, B →* p]  
  return [p' in bank' | forall p in bank: !equiv(p, p')];  
}
```

- How do we implement equiv?
 - In general, undecidable
 - For SyGuS problems: expensive
 - Doing expensive checks on every candidate defeats the purpose of pruning the space!

Observational equivalence

bottom-up ($\langle T, N, R, S \rangle, [i \rightarrow o]$)
{ ... }

```
equiv(p, p') {  
  return p([i]) = p'([i])  
}
```

$[[\theta] \rightarrow [\theta]]$

x θ

sort(x) x[$\theta..0$] x + x [θ] find(x, θ)

- In PBE, all we care about is equivalence on the given inputs!
 - easy to check efficiently
 - even more programs are equivalent

sort(x + x)

x[$\theta..find(x, \theta)$]

x + (x + x) x + [θ] sort(x) + x

[θ] + x

x + sort(x)

Observational equivalence

bottom-up ($\langle T, N, R, S \rangle, [i \rightarrow o]$)
{ ... }

```
equiv(p, p') {  
  return p([i]) = p'([i])  
}
```

$[[\theta] \rightarrow [\theta]]$

$x \quad \theta$

$\text{sort}(x) \quad x[\theta..\theta] \quad x + x \quad [\theta] \quad \text{find}(x, \theta)$

$\text{sort}(x + x)$

$x[\theta..\text{find}(x, \theta)]$

$x + (x + x) \quad x + [\theta] \quad \text{sort}(x) + x$

$[\theta] + x$

$x + \text{sort}(x)$

Observational equivalence

bottom-up ($\langle T, N, R, S \rangle, [i \rightarrow o]$)
{ ... }

```
equiv(p, p') {  
  return p([i]) = p'([i])  
}
```

$[[\theta] \rightarrow [\theta]]$

$\boxed{x} \quad \boxed{\theta}$

$x[\theta..0]$

$x + x$

$x + (x + x)$

Observational equivalence

- Proposed simultaneously in two papers:
 - Udupa, Raghavan, Deshmukh, Mador-Haim, Martin, Alur: [TRANSIT: specifying protocols with concolic snippets](#). PLDI'13
 - Albarghouthi, Gulwani, Kincaid: [Recursive Program Synthesis](#). CAV'13
- Variations used in most bottom-up PBE tools:
 - **ESolver** (baseline SyGuS enumerative solver)
 - **EUSolver** [Alur et al. TACAS'17]
 - **Probe** [Barke et al. OOPSLA'20]
 - **TFCoder** [Shi et al. TOPLAS'22]

User-specifies equations

[Smith, Albarghouthi: VMCAI'19]

Equations

$\text{sort}(\text{sort}(1)) = \text{sort}(1)$

$(11 + 12) + 13 = 11 + (12 + 13)$

$n = n + 0$

$n + m = m + n$

derived
automatically



Term-rewriting system (TRS)

1. $\text{sort}(\text{sort}(1)) \rightarrow \text{sort}(1)$

2. $(11 + 12) + 13 \rightarrow 11 + (12 + 13)$

3. $n + 0 \rightarrow n$

4. $n + m \rightarrow_{(n > m)} m + n$

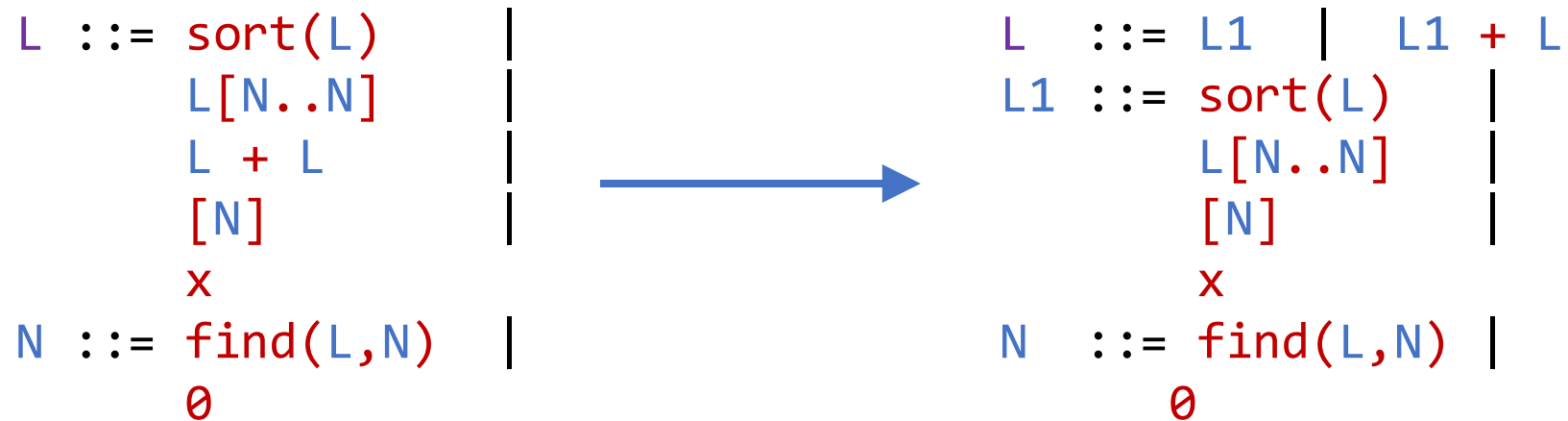
x 0

$\text{sort}(x)$ $x[0..0]$ $x + x$ $[0]$ $\text{find}(x, 0)$

~~$\text{sort}(\text{sort}(x))$~~ rule 1 applies, not in *normal form*

Built-in equivalences

- For a predefined set of operations, equivalence reduction can be hard-coded in the tool or built into the grammar



Built-in equivalences

- Used by:
 - λ^2 [Feser et al.'15]
 - **Leon** [Kneuss et al.'13]
- Leon's implementation using *attribute grammars* described in:
 - Koukoutos, Kneuss, Kuncak: An Update on Deductive Synthesis and Repair in the Leon tool [SYNT'16]

Equivalence reduction: comparison

- Observational
 - Very general, no user input required
 - Finds more equivalences
 - Can be costly (with many examples, large outputs)
 - If new examples are added, has to restart the search
- User-specified
 - Fast
 - Requires equations
- Built-in
 - Even faster
 - Restricted to built-in operators
 - Only certain symmetries can be eliminated by modifying the grammar
- Q1: Can any of them apply to top-down?
- Q2: Can any of them apply beyond PBE?