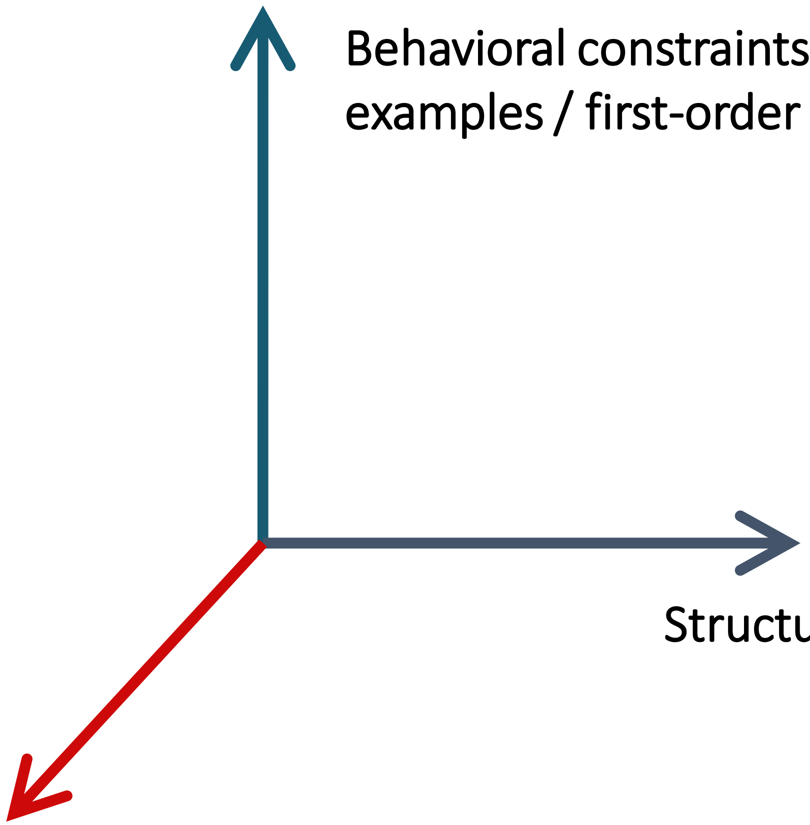
Lecture 8
Constraint-based search

The problem statement

Behavioral constraints =
examples / first-order formula



Search strategy?

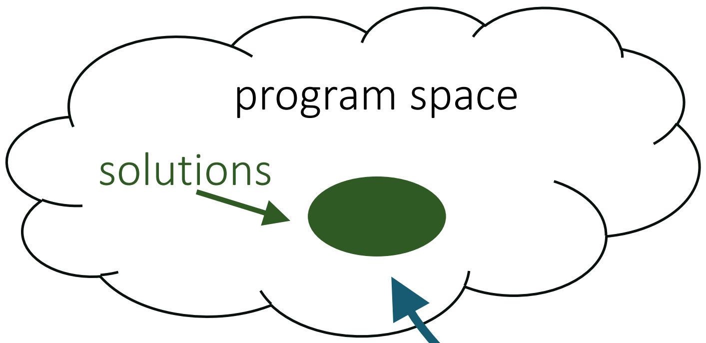
Enumerative
Representation-based
Stochastic
Constraint-based

Structural constraints = we'll see…

Constraint-based search

Idea: encode the synthesis problem as a SAT/SMT problem and let a solver deal with it

What is an encoding?



parameter space

$$C = \{c_1, \dots, c_n\}$$

$$\{C \mid \mathrm{wf}(C)\}$$

well-formedness constraint

$$\{C \mid \mathrm{wf}(C) \land \mathrm{sem}(C)\}$$

semantic correctness constraint

$$\mathrm{sem}(C) = \bigwedge_{\langle i,o\rangle \in E} \varphi(C, i, o)$$

C denotes a program that gives output *o* on input *i*

How to define an encoding

Define the parameter space $C = \{c_1, \ldots, c_n\}$

- `decode : C → Prog` (might not be defined for all C)

Define a formula $\mathrm{wf}(c_1, \ldots, c_n)$

- that holds iff `decode[C]` is defined

Define a formula $\varphi(c_1, \ldots, c_n, i, o)$

- that holds iff `(decode[C])(i) = o`

Constraint-based search

```
constraint-based (wf, φ, E = [i → o]) {
  match SAT(wf(C) ∧ ⋀⟨i,o⟩∈E φ(C, i, o)) with
    Unsat -> return "No solution"
    Model C* -> return decode[C*]
}
```

Find a satisfying assignment for $c_1, \ldots, c_n$ (**i** and **o** are fixed)

SAT encoding: example

program space

`{x, x & 1}`

`x` is a two-bit word
(`x = x`$_h$`x`$_l$)

`E = [11 → 01]`

parameter space

$$C = \{c : \mathrm{Bool}\}$$

`decode[0] → x`
`decode[1] → x & 1`

$$\mathrm{wf}(c) \equiv \top$$

$$\varphi(c, i_h, i_l, o_h, o_l) \equiv (\neg c \;\Rightarrow\; o_h = i_h \wedge o_l = i_l)$$
$$\wedge\ (c \;\Rightarrow\; o_h = 0 \wedge o_l = i_l)$$

$$\mathrm{SAT}(\varphi(c, 1,1,0,1))$$

$$\mathrm{SAT}((\neg c \;\Rightarrow\; 0 = 1 \wedge 1 = 1) \wedge (c \;\Rightarrow 0 = 0 \wedge 1 = 1))$$

SAT solver →

`Model {c→1}`

`return` `decode[1]` i.e. `x & 1`

SMT encoding: example

program space

`x + N | x * N`

N is an in integer literal
x is an integer input

`E = [2 → 9]`

parameter space

$$C = \{c_{op}: \text{Bool}, c_N: \text{Int}\}$$

```
decode[0,N] → x + N
decode[1,N] → x * N
```

$$\text{wf}(c_{op}, c_N) \equiv \top$$

$$\varphi(c_{op}, c_N, i, o) \equiv (\neg c_{op} \Rightarrow o = i + c_N) \wedge (c_{op} \Rightarrow o = i * c_N)$$

$$\text{SAT}(\varphi(c_{op}, c_N, 2, 9))$$

$$\text{SAT}((\neg c_{op} \Rightarrow 9 = 2 + c_N) \wedge (c_{op} \Rightarrow 9 = 2 * c_N))$$

SMT solver →

Model $\{c_{op} \to 0, c_N \to 7\}$

`return decode[0,7]` i.e. `x + 7`

What is a good encoding?

Sound

- if $\mathrm{wf}(C) \land \mathrm{sem}(C)$ then `decode[C]` is a solution

Complete

- if `decode[C]` is a solution then $\mathrm{wf}(C) \land \mathrm{sem}(C)$

Small parameter space

- avoid symmetries

Solver-friendly

- decidable logic, compact constraint

DSL limitations

Program space can be parameterized with a finite set of parameters

- Counterexample:

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

- Workaround

```
L0 ::= x    L1 ::= sort(L0)  |  L0[N0..N0]
N0 ::= 0         |  L0 + L0  |  [N0]  |  L0
            N1 ::= find(L0,N0)  |  N0
```

Program semantics $\varphi(C, i, o)$ is expressible as a (decidable) SAT/SMT formula

- Counterexample

Brahma

Idea: encode the space of loop-free (bit-vector) programs as an SMT constraint

Brahma encoding: take 1

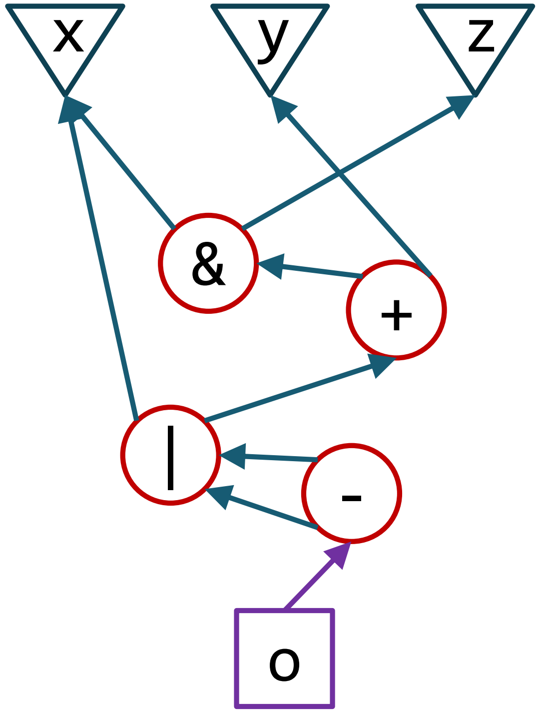
program = DAG

I

inputs

F

components

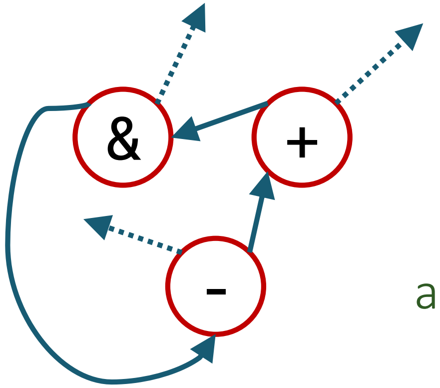


$V = I \cup F$ nodes

parameter space

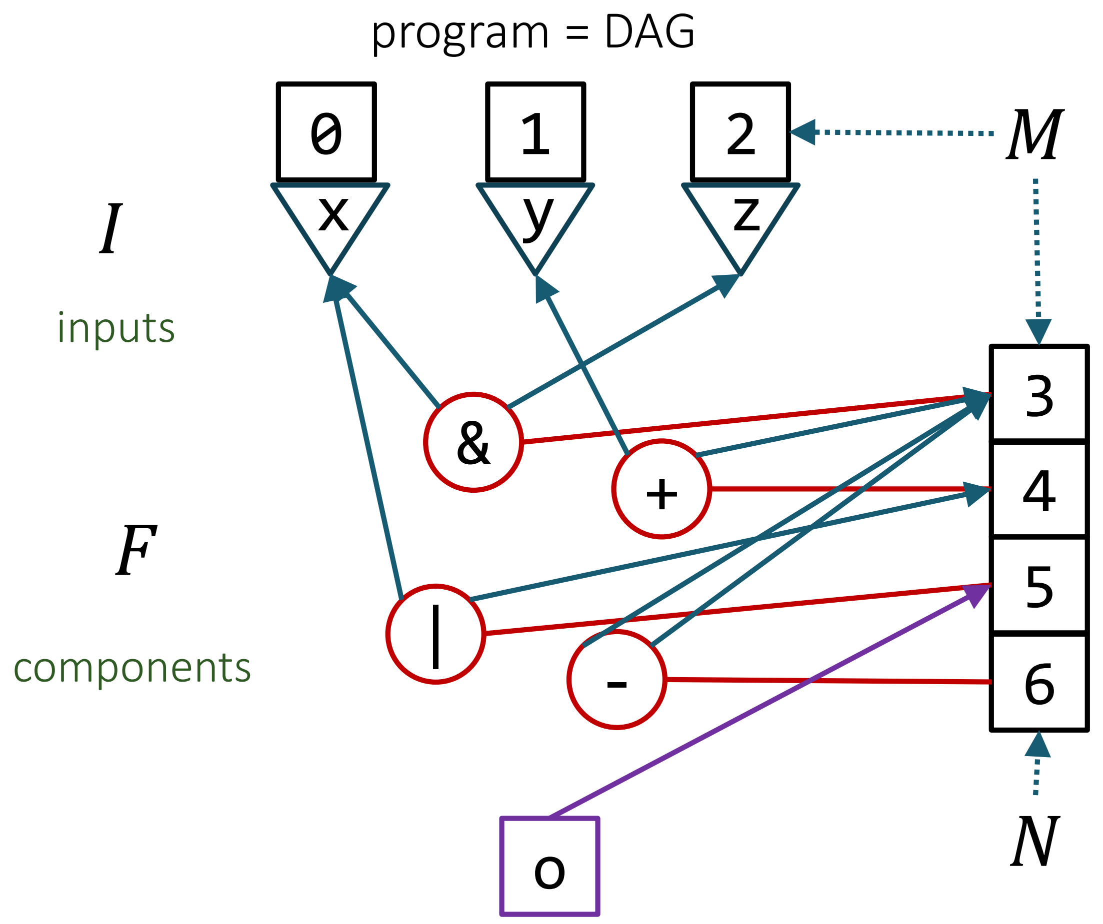
$$C = \{c_o : V\} \cup \bigcup_{f \in F} \{c_1^f, c_2^f : V\}$$

$\mathrm{wf}(C) \equiv ?$



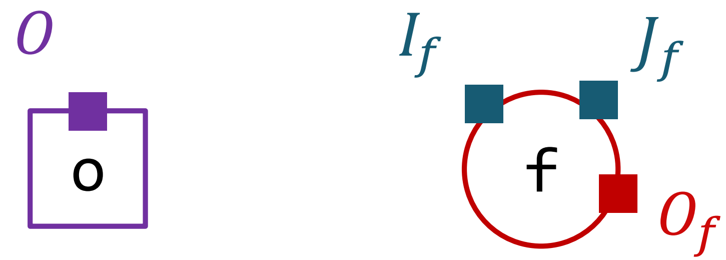
hard to encode acyclicity!

Brahma encoding: take 2



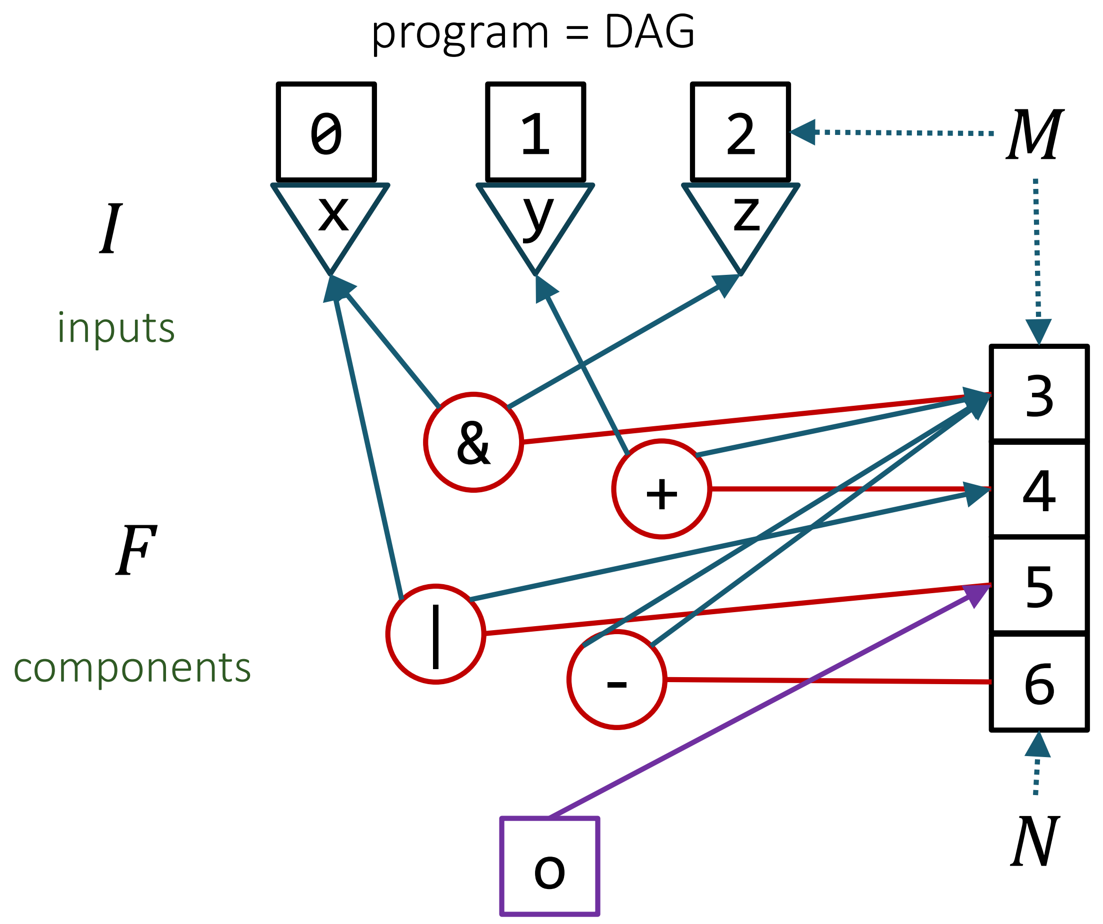
parameter space

$$C = \{c_O : \mathrm{Int}\} \cup \bigcup_{f \in F} \{c_{O_f}, c_{I_f}, c_{J_f} : \mathrm{Int}\}$$



$$\mathrm{wf}(C) \equiv c_O \in M \ \wedge \bigwedge_{f \in F} c_{O_f} \in N \wedge c_{I/J_f} \in M$$

Brahma encoding: take 2



parameter space

$$C = \{c_o : \mathrm{Int}\} \cup \bigcup_{f \in F} \{c_{O_f}, c_{I_f}, c_{J_f} : \mathrm{Int}\}$$

$$T = \bigcup_{f \in F} \{I_f, J_f. O_f\}$$

$$\varphi(C, I, O) \equiv \exists T. \bigwedge_{f \in F} O_f = F(I_f, J_f)$$

$$\wedge \bigwedge_{x,y \in T \cup I \cup \{O\}} c_x = c_y \Rightarrow x = y$$

Brahma: contributions

SMT encoding of program space

- sound?
- complete?
- solver-friendly?

SMT solver can guess constants

- e.g. 0x55555555 in P23

Brahma: limitations

Requires component multiplicities

- If we didn’t have multiplicities, where would their encoding break? How could we fix it?
- What happens if user provides too many? too few?
- What’s the alternative to including dead code?

Requires *precise* SMT specs for components

- What happens if we give an over-approximate spec?

No loops, no types, no ranking

Brahma: questions

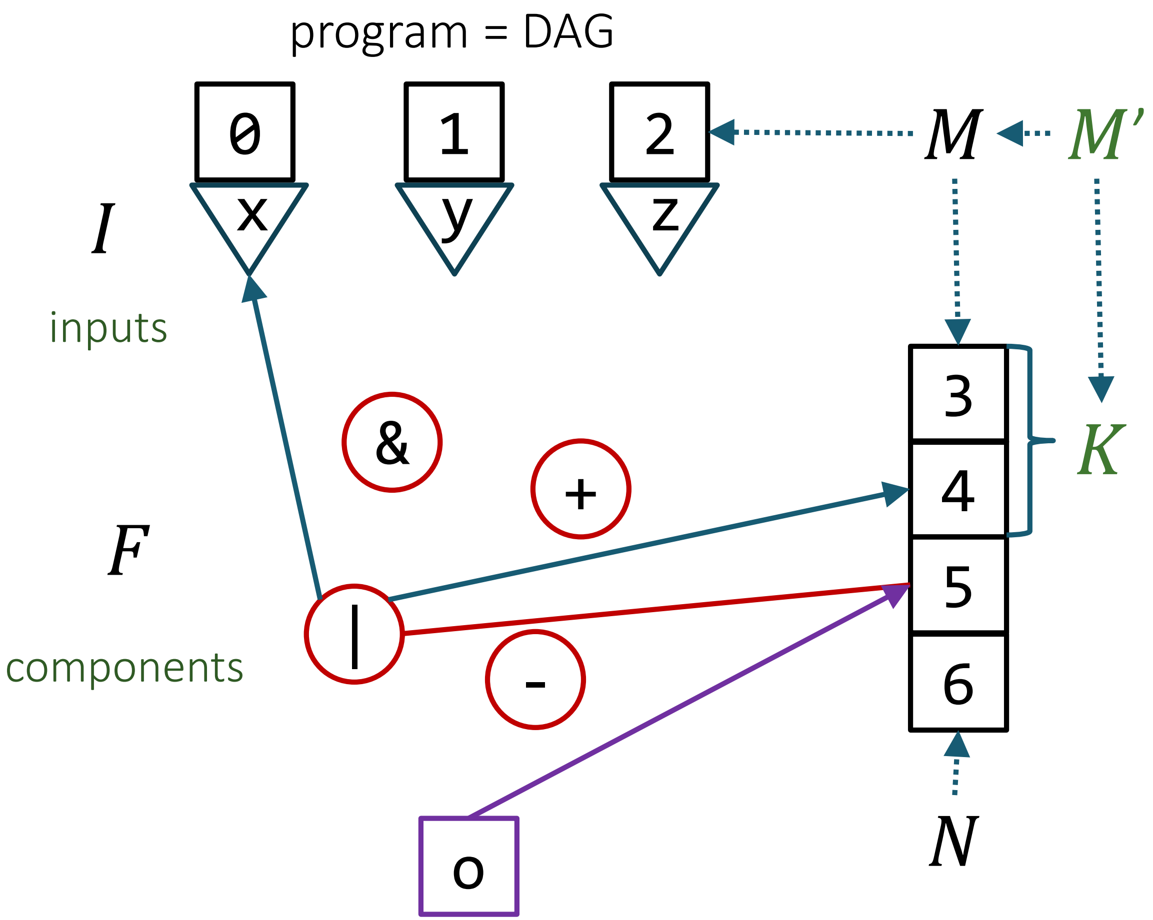
Behavioral Constraints? Structural Constraints? Search Strategy?

- First-order formula
- A multiset of components + straight-line program
- Constraint based + CEGIS

Can we represent these structural constraints as a grammar?

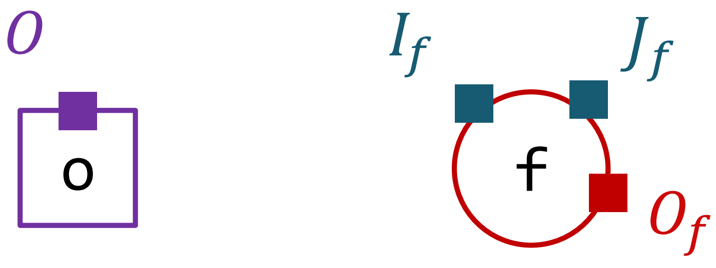
- Yes and no
- No because grammars cannot encode multiplicities
  - also: you can have let-bindings in SyGuS but CFG cannot encode well-formedness
- Yes because the set is finite, so we can simply enumerate all possible programs
  - but this is not useful for synthesis

Limit #components to K?



parameter space

$$C = \{c_O : \mathrm{Int}\} \cup \bigcup_{f \in F} \{c_{O_f}, c_{I_f}, c_{J_f} : \mathrm{Int}\}$$

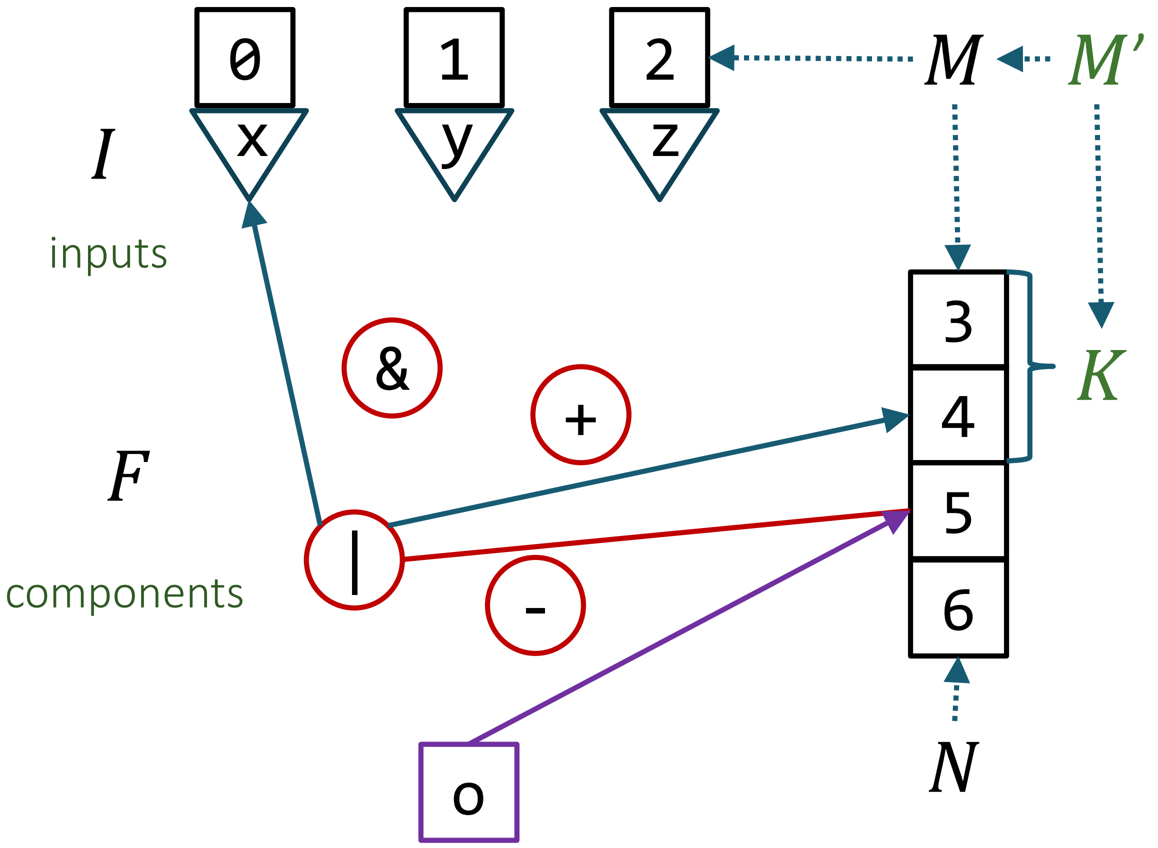


$$\mathrm{wf}(C) \equiv c_O \in \overset{M'}{\cancel{M}} \wedge \bigwedge_{f \in F} c_{O_f} \in \overset{K}{\cancel{N}} \wedge c_{I/J_f} \in \overset{M'}{\cancel{M}}$$

$$\wedge \bigwedge_{f,g \in F, f \not\equiv g} c_{O_f} \neq c_{O_g} \wedge \bigwedge_{f \in F} c_{I/J_f} < c_{O_f}$$

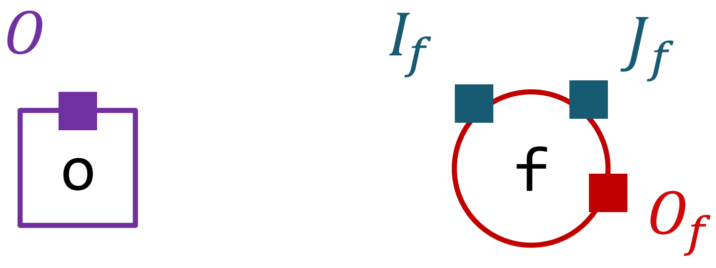
Limit #components to K?

program = DAG



parameter space

$$C = \{c_O : \mathrm{Int}\} \cup \bigcup_{f \in F} \{c_{O_f}, c_{I_f}, c_{J_f} : \mathrm{Int}\}$$



$$\mathrm{wf}(C) \equiv c_O \in \overset{M'}{\cancel{M}} \wedge \bigwedge_{f \in F} c_{O_f} \in N \wedge c_{I/J_f} \in M$$

$$\wedge \bigwedge_{f,g \in F, f \not\equiv g} c_{O_f} \neq c_{O_g} \wedge \bigwedge_{f \in F} c_{I/J_f} < c_{O_f}$$

A linear encoding with uninterpreted functions

$$t_0 = \mathtt{0}$$

$$t_1 = \mathtt{1}$$

$$t_2 = \mathtt{bvadd}(t_1, t_0)$$

$$t_3 = \mathtt{bvadd}(t_2, t_1)$$

	Production	1st child	2nd child	Nonterm	Output value on 10010101	
$line$	$\mathbf{p}^l(line)$	$\mathbf{c}(line, 0)$	$\mathbf{c}(line, 1)$	$\mathbf{n}^l(line)$	$\mathbf{v}_l^\epsilon(line)$	$\ldots$
0	$\mathtt{0}$	$*$	$*$	B	0000	
1	$\mathtt{1}$	$*$	$*$	B	0001	
2	$\mathtt{bvadd}(B, B)$	1	0	B	0001	
3	$\mathtt{bvadd}(B, B)$	2	1	B	0010	

- Line $L-1$ should be assigned the initial non-terminal:

$$\mathtt{n}^l(L-1) = S \tag{5.3}$$

- For each line, if that line is assigned a certain production, its children are appropriate non-terminals and appear at lower numbered lines.[4]

$$\begin{aligned}\forall i < L.\ \mathtt{p}^l(i) = op(N_0, \ldots, N_j) \Rightarrow& \\ &\mathtt{n}^l(\mathtt{c}(i,0)) = N_0 \land \ldots \land \mathtt{n}^l(\mathtt{c}(i,j)) = N_j \\ &\mathtt{c}(i,0) < i \land \ldots \land \mathtt{c}(i,j) < i\end{aligned}$$

- If a line is assigned a certain non-terminal, it can only contain one of the corresponding productions:

$$\forall i < L.\ \mathtt{n}^l(i) = N \Rightarrow \bigvee_{r \in \delta(N)} \mathtt{p}^l(i) = r$$

Behavioral constraints:

- For each example, the value at line $L-1$ should be the correct output on that example:

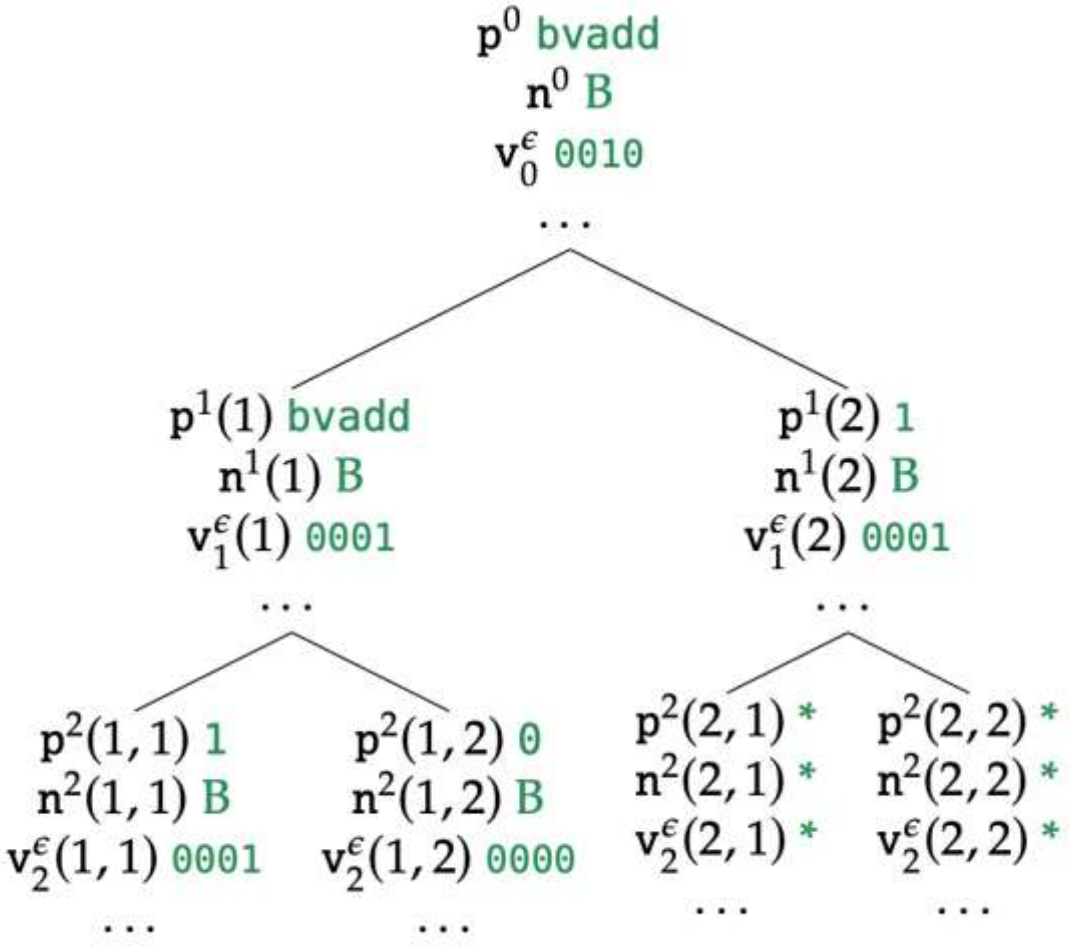
$$\forall \epsilon \in \mathcal{E}.\ \mathbf{v}_l^{\epsilon}(L-1) = \mathsf{M}(\epsilon) \tag{5.4}$$

- for each example $\epsilon \in \mathcal{E}$, the value of the line $\mathbf{v}_f^{\epsilon}$ is computed consistently with $\mathbf{p}^l$ and the values of the children. The following constraint is stated for each example $\epsilon \in \mathcal{E}$ and for each line $i < L$:

$$\forall i < L. \bigwedge_{op(N_1,\ldots,N_j)\in\delta} \mathbf{p}^l(i) = op(N_1, \ldots, N_j) \Rightarrow$$
$$\mathbf{v}_l^{\epsilon}(i) = [\![op]\!](\mathbf{v}_l^{\epsilon}(\mathsf{c}(i,0)), \ldots, \mathbf{v}_l^{\epsilon}(\mathsf{c}(i,j)))$$

A tree encoding with uninterpreted functions

bvadd(bvadd(1, 0), 1)



- The root node should be the initial non-terminal of the grammar

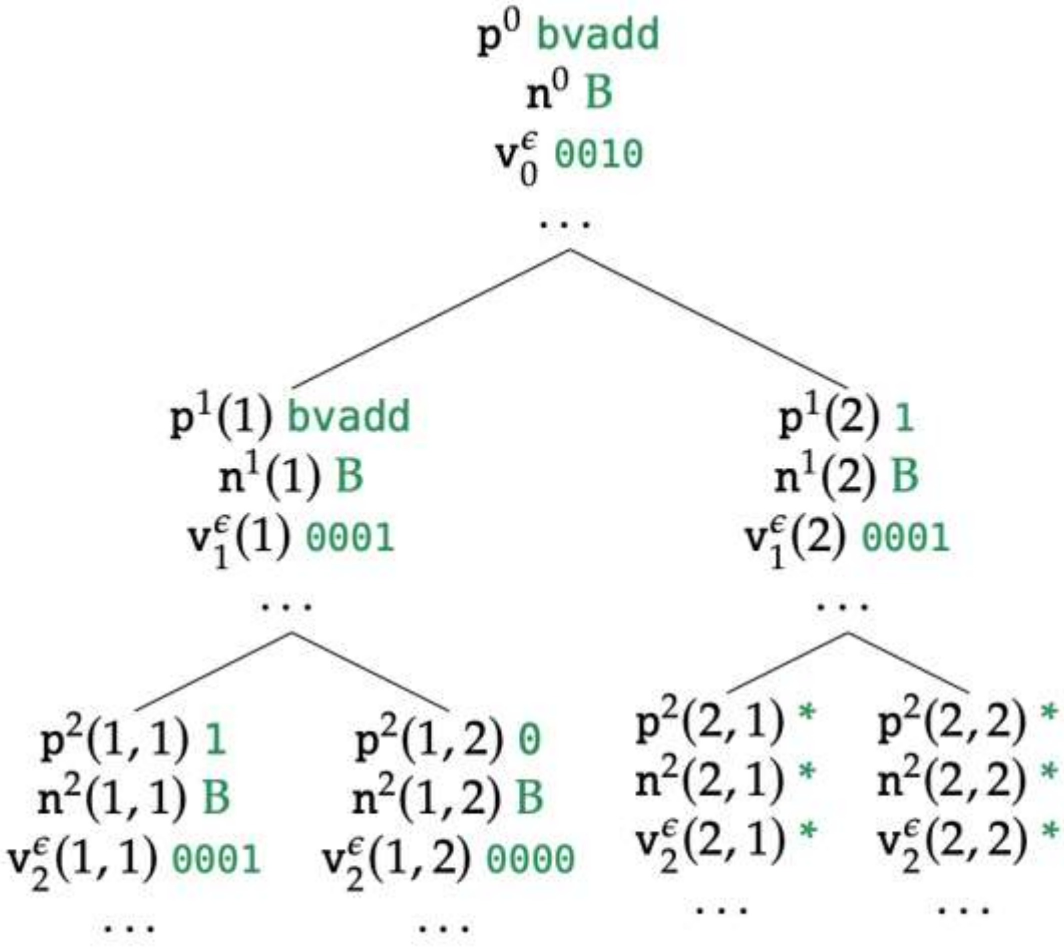
$$\mathbf{n}^0 = S \tag{5.1}$$

- For each path, if a node at that path is assigned a certain production, its children are appropriate non-terminals. Because we are interested in the children of a node, we should state this constraint only for non-leaf nodes. Thus the following constraint is added for any depth $d < D$:

$$\begin{aligned}&\forall i_1, \ldots, i_d < K.\\ &\quad \mathbf{p}^d(i_1, \ldots, i_d) = op(N_0, \ldots, N_j) \Rightarrow\\ &\qquad \mathbf{n}^{d+1}(i_1, \ldots, i_d, 0) = N_0 \wedge \ldots \wedge \mathbf{n}^{d+1}(i_1, \ldots, i_d, j) = N_j\end{aligned}$$

A tree encoding with uninterpreted functions

bvadd(bvadd(1, 0), 1)



- If a node is a certain non-terminal, it can only be one of the corresponding productions (at the last level, only 0-arity productions are allowed). We write $r \in \delta(N)$ to denote a production r with left nonterminal N and $r \in \delta^0(N)$ to denote a 0-arity production r

  with left nonterminal N. We start with the first case and state the following constraint for all $d < D$:

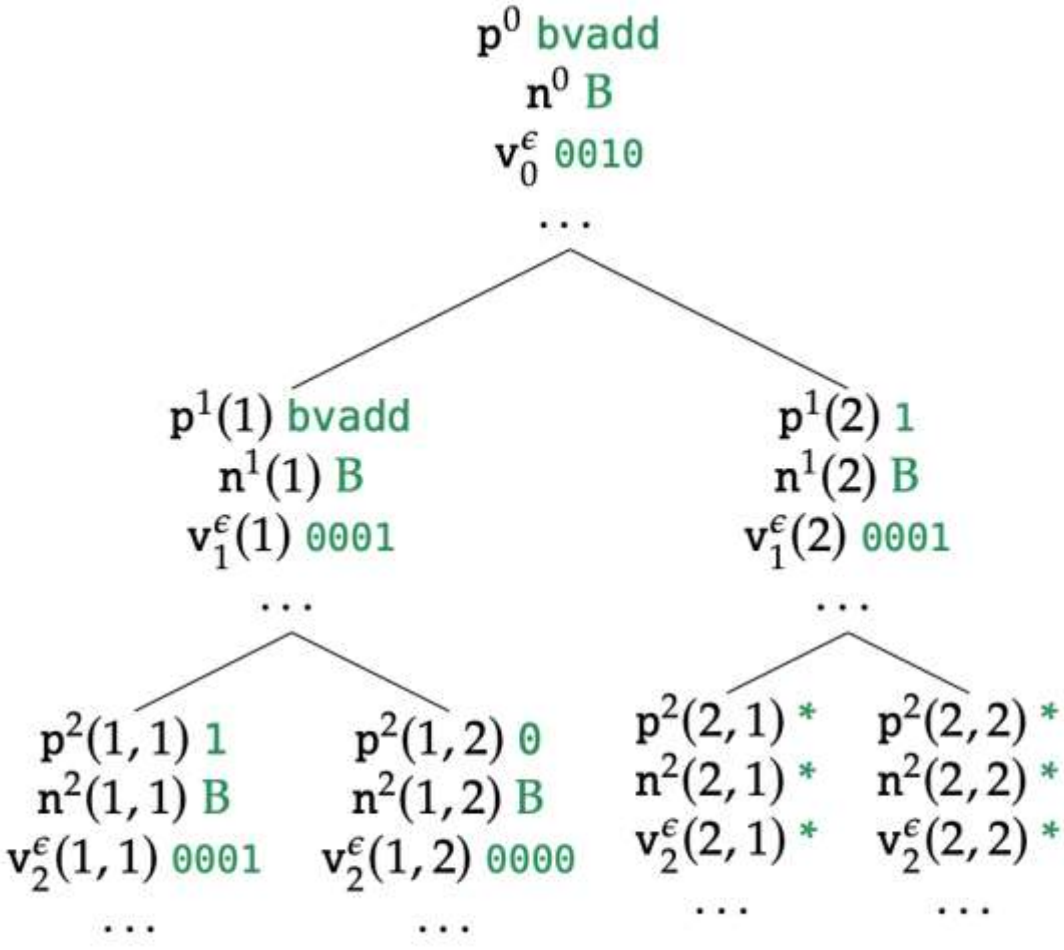
$$\forall i_1, \ldots, i_d < K.$$
$$\mathrm{n}^d(i_1, \ldots, i_d) = N \Rightarrow \bigvee_{r \in \delta(N)} \mathrm{p}^d(i_1, \ldots, i_d) = r$$

The constraint for the last level D is very similar but only allows 0-arity productions.

$$\forall i_1, \ldots, i_D < K.$$
$$\mathrm{n}^D(i_1, \ldots, i_D) = N \Rightarrow \bigvee_{r \in \delta^0(N)} \mathrm{p}^D(i_1, \ldots, i_D) = r$$

A tree encoding with uninterpreted functions

bvadd(bvadd(1, 0), 1)



Behavioral constraints: ensure that the generated tree is correct on all examples.

- for each IO example the value of the root node on that example should be the correct output

$$\forall \epsilon \in \mathcal{E}.\ \mathsf{v}_0^\epsilon = \mathsf{M}(\epsilon) \tag{5.2}$$

- for each example $\epsilon \in \mathcal{E}$, the v_d^ϵ is computed consistently with p^d and the values of the children. The following constraint is stated for each example $\epsilon \in \mathcal{E}$ and for each depth $d < D$:

$$\forall i_1, \ldots, i_d < K.$$
$$\bigwedge_{op(N_1,\ldots,N_j)\in\delta} \mathsf{p}^d(i_1,\ldots,i_d) = op(N_1,\ldots,N_j) \Rightarrow$$
$$\mathsf{v}_d^\epsilon(i_1,\ldots,i_d) = [\![op]\!](\mathsf{v}_{d+1}^\epsilon(i_1,\ldots,i_d,0),\ldots,\mathsf{v}_{d+1}^\epsilon(i_1,\ldots,i_d,j))$$

- Again a little bit of care has to go for the last depth in making sure we only use 0-arity productions:

$$\forall i_1, \ldots, i_D < K.$$
$$\bigwedge_{op()\in\delta^0} \mathsf{p}^D(i_1,\ldots,i_D) = op() \Rightarrow \mathsf{v}_D^\epsilon(i_1,\ldots,i_D) = [\![op]\!]()$$

What are the differences of each encoding?

Brahma

- Bounds number of components
- Mostly uses bit-vector variables

Linear

- Bounds size (sort of, allows reusing lines)
- Uninterpreted functions

Tree

- Bounds depth
- Uninterpreted functions

Comparison of search strategies