

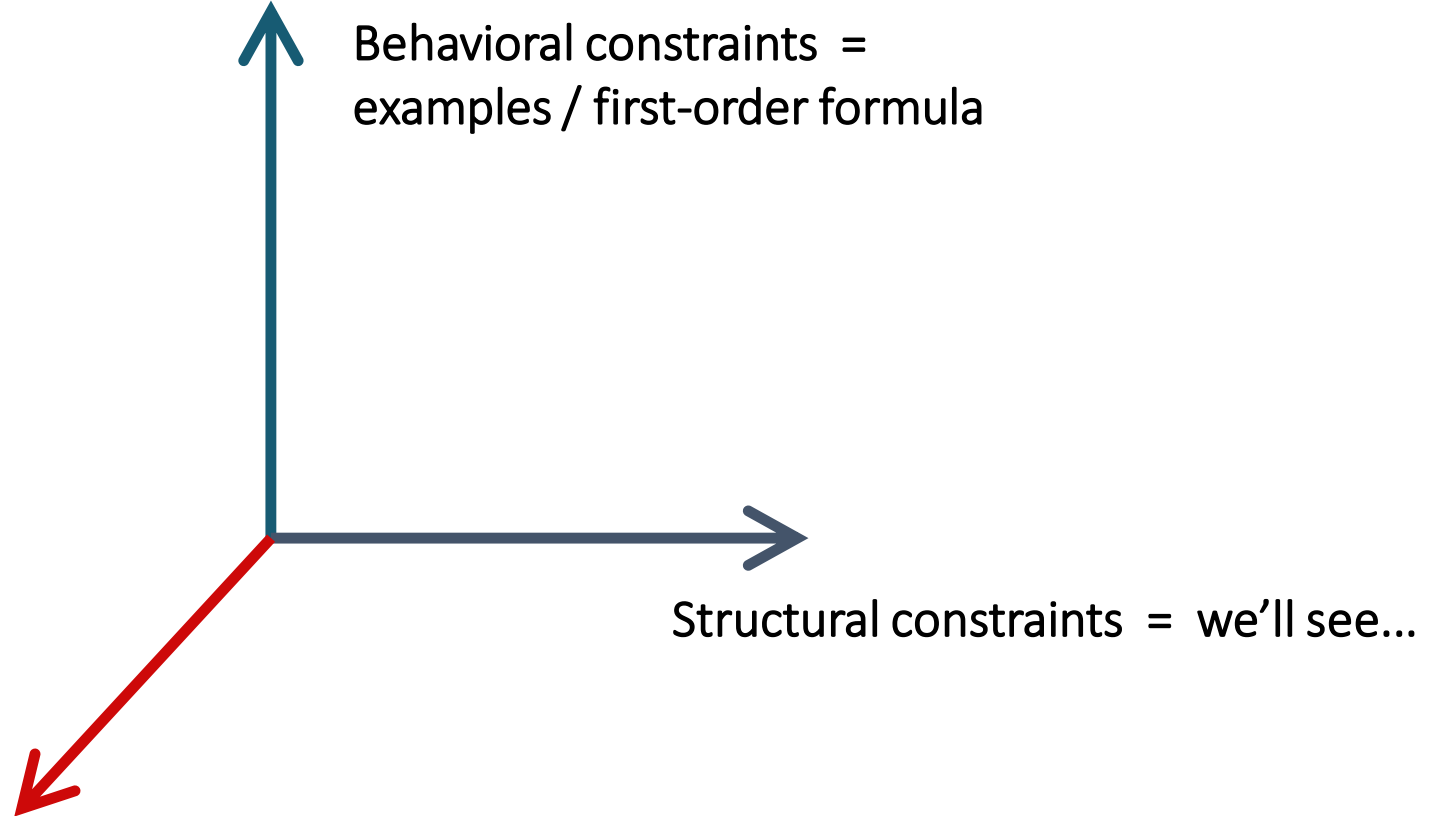
Lecture 8

Constraint-based search

The problem statement

Search strategy?

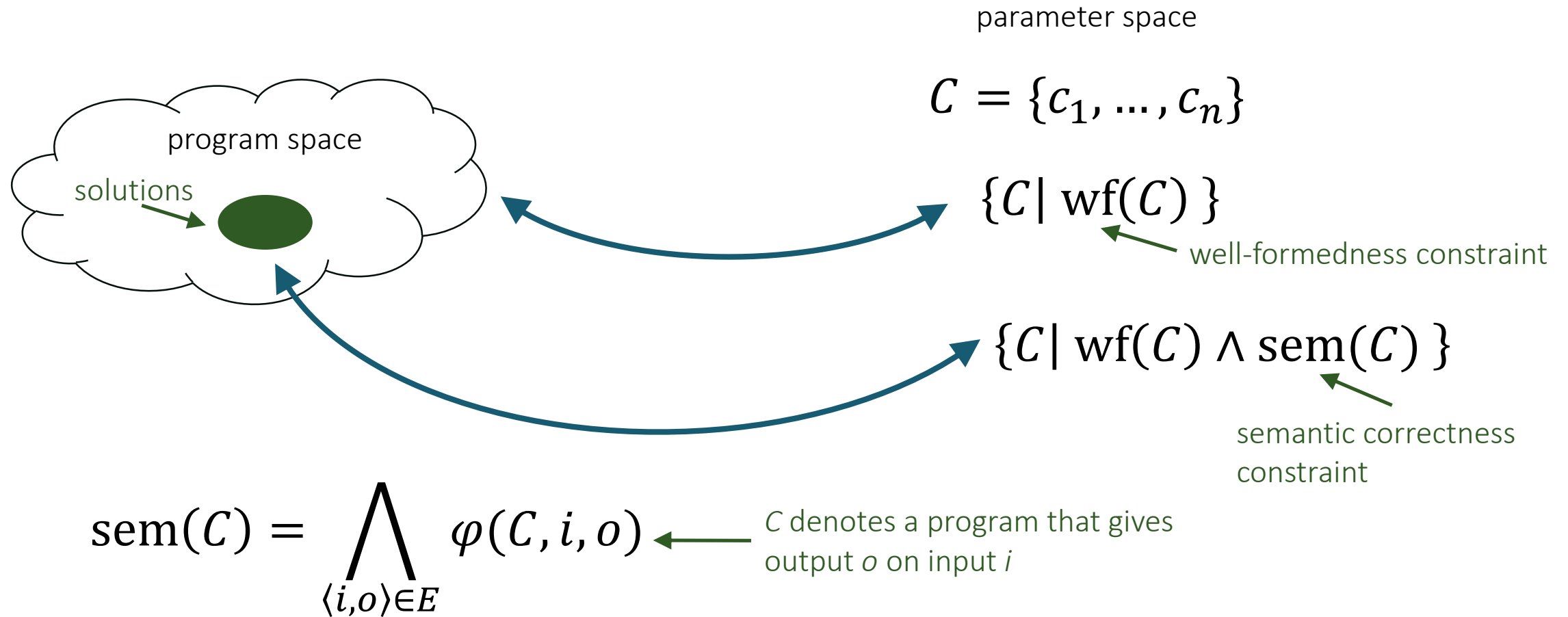
Enumerative
Representation-based
Stochastic
Constraint-based



Constraint-based search

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

What is an encoding?



How to define an encoding

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

- `decode` : $\mathcal{C} \rightarrow \text{Prog}$ (might not be defined for all C)


Define a formula $\text{wf}(c_1, \dots, c_n)$

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

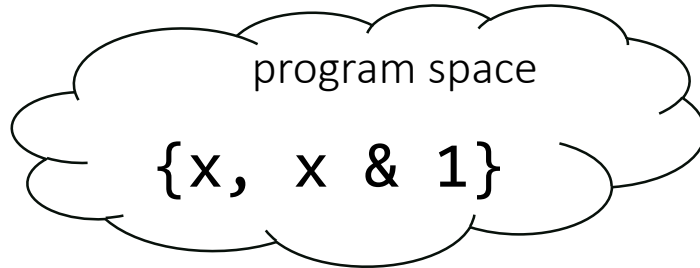
Define a formula $\varphi(c_1, \dots, c_n, i, o)$

- that holds iff $(\text{decode}[C])(i) = o$

Constraint-based search

```
constraint-based (wf,  $\varphi$ ,  $E = [i \rightarrow o]$ ) {  
  match SAT(wf( $C$ )  $\wedge \bigwedge_{\langle i,o \rangle \in E} \varphi(C, i, o)$ ) with  Find a satisfying assignment  
    Unsatisfiable -> return "No solution" for  $c_1, \dots, c_n$   
    Model  $C^*$  -> return decode[ $C^*$ ] (i and o are fixed)  
}
```

SAT encoding: example



$$\text{wf}(c) \equiv \text{T}$$

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

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

$$\text{SAT}((\neg c \Rightarrow 0 = 1 \wedge 1 = 1) \wedge (c \Rightarrow 0 = 0 \wedge 1 = 1)) \xrightarrow{\text{SAT solver}} \text{Model } \{c \rightarrow 1\}$$

x is a two-bit word
($x = x_h x_l$)

$$E = [11 \rightarrow 01]$$

parameter space

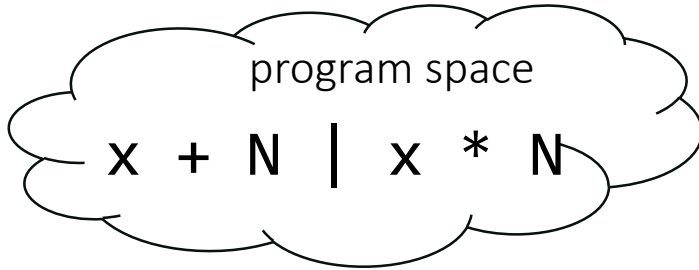
$$C = \{c: \text{Bool}\}$$

$$\text{decode}[0] \rightarrow x$$

$$\text{decode}[1] \rightarrow x \ \& \ 1$$

return decode[1] i.e. $x \ \& \ 1$

SMT encoding: example



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

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

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

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

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

N is an integer literal
 x is an integer input

$$E = [2 \rightarrow 9]$$

parameter space

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

$$\text{decode}[0, N] \rightarrow x + N$$

$$\text{decode}[1, N] \rightarrow x * N$$

SMT solver



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

What is a good encoding?

Sound

- if $\text{wf}(C) \wedge \text{sem}(C)$ then $\text{decode}[C]$ is a solution

Complete

- if $\text{decode}[C]$ is a solution then $\text{wf}(C) \wedge \text{sem}(C)$

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:
$$\begin{array}{lcl} L & ::= & \text{sort}(L) \quad | \quad L[N..N] \\ & & | \quad L + L \quad | \quad [N] \quad | \quad x \\ N & ::= & \text{find}(L, N) \quad | \quad \emptyset \end{array}$$

- Workaround
$$\begin{array}{lcl} L0 & ::= & x \quad L1 ::= \text{sort}(L0) \quad | \quad L0[N0..N0] \\ N0 & ::= & \emptyset \quad | \quad L0 + L0 \quad | \quad [N0] \quad | \quad L0 \\ & & N1 ::= \text{find}(L0, N0) \quad | \quad N0 \end{array}$$

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

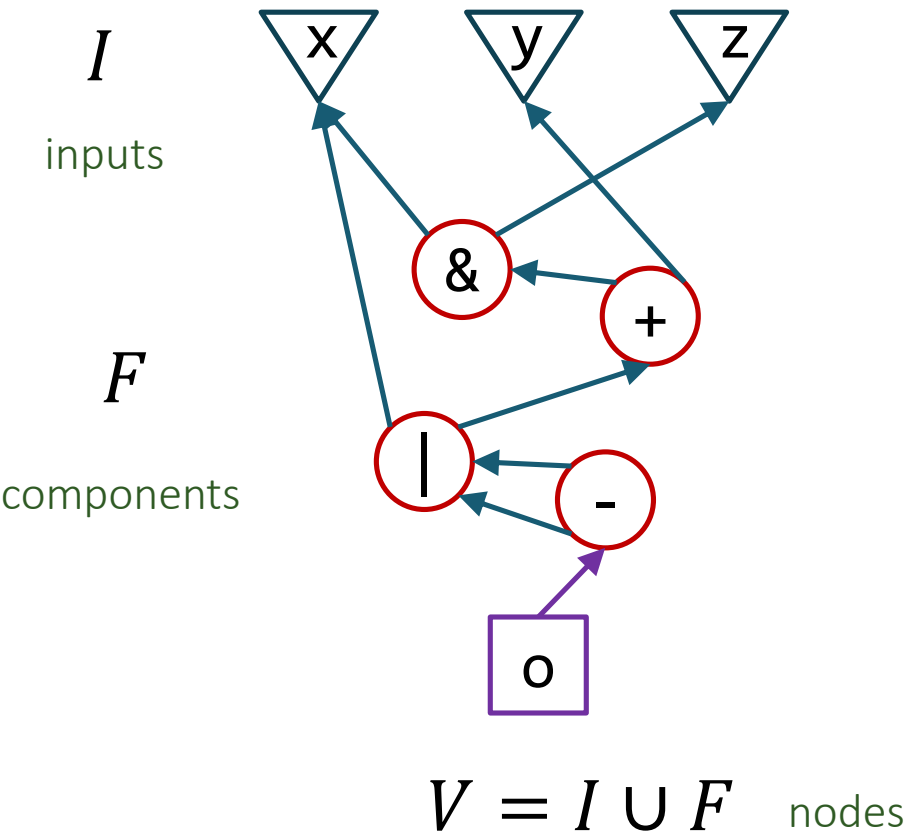
- Counterexample

Brahma

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

Brahma encoding: take 1

program = DAG

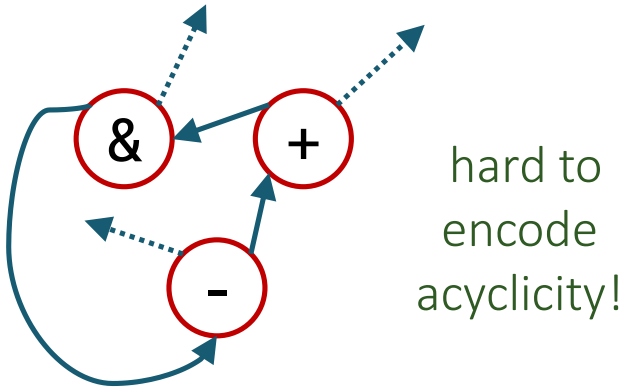


parameter space

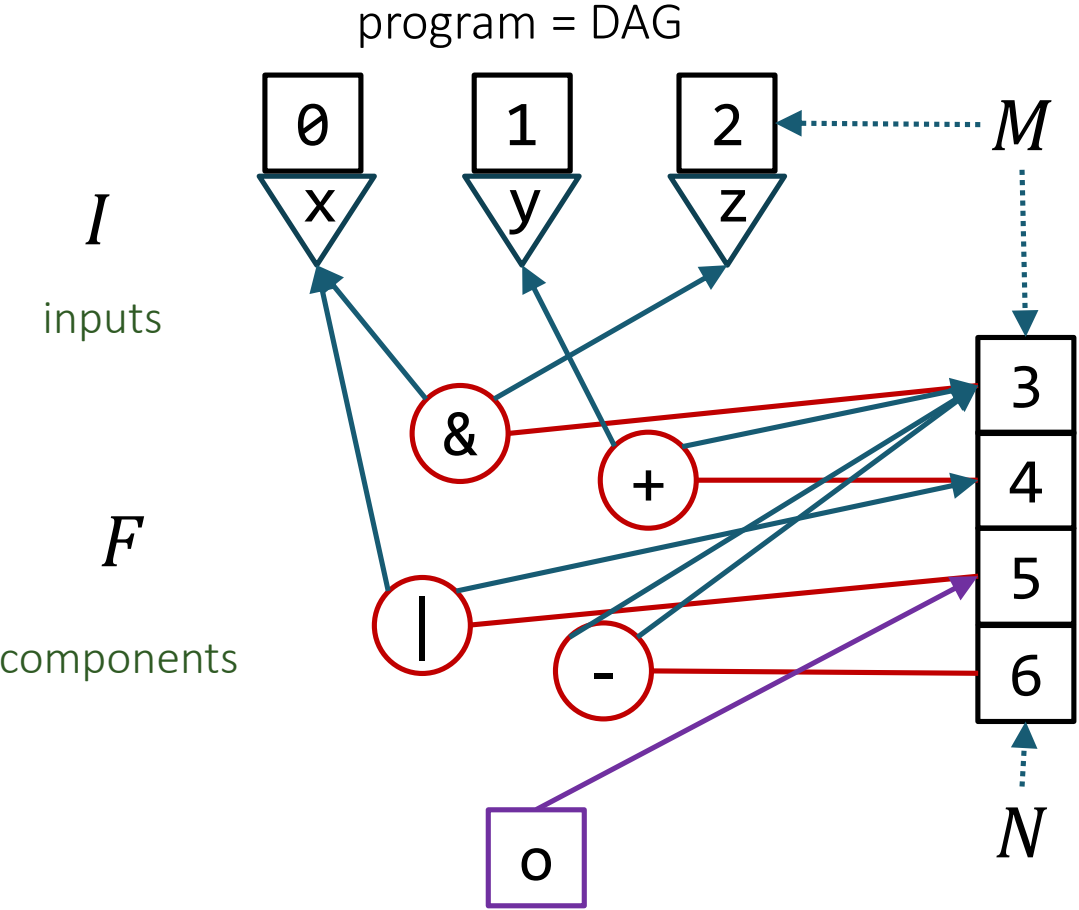
$$C = \{c_o:V\} \cup \bigcup_{f \in F} \{c_1^f, c_2^f:V\}$$



$wf(C) \equiv ?$

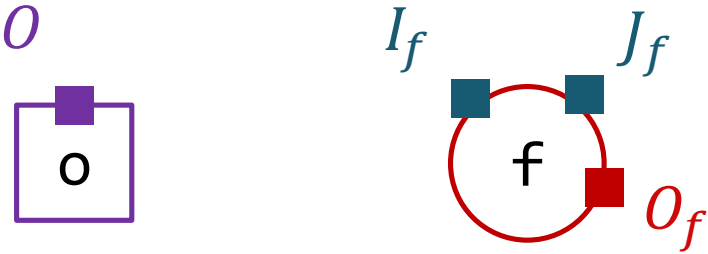


Brahma encoding: take 2



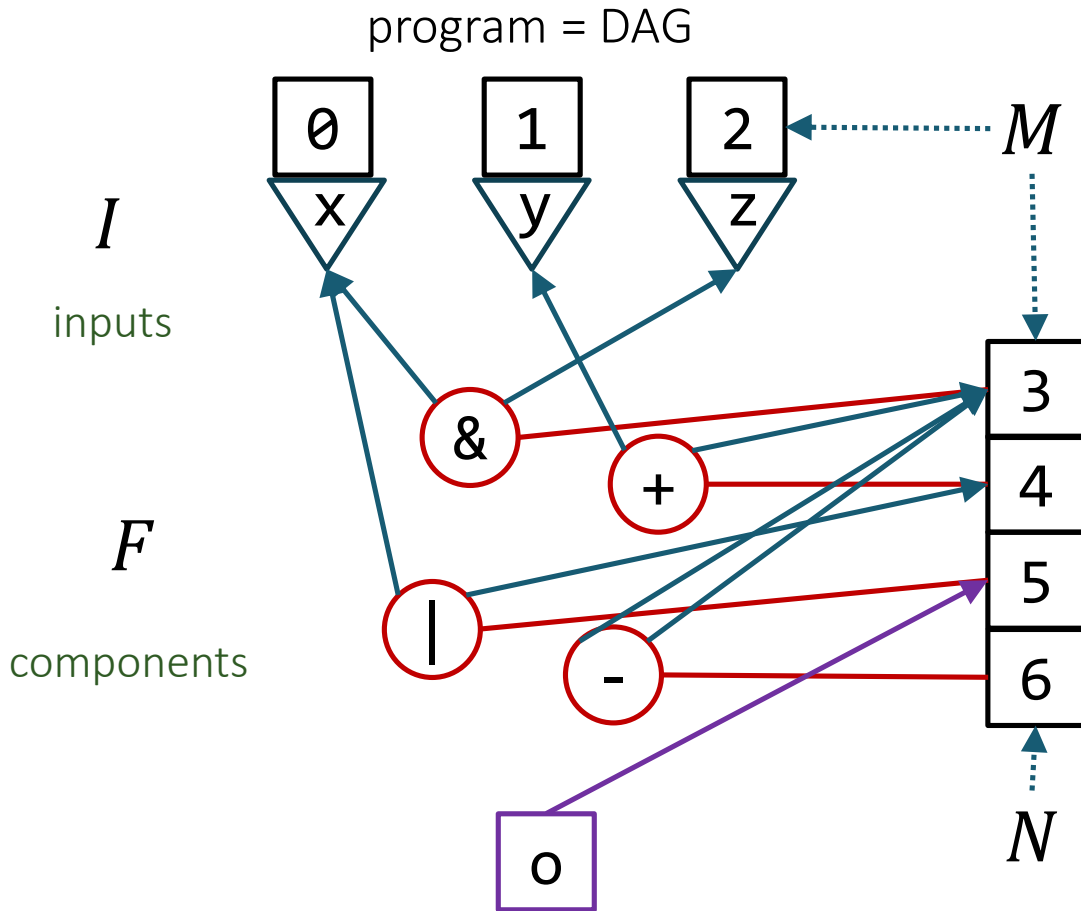
parameter space

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



$$\text{wf}(C) \equiv c_o \in M \wedge \bigwedge_{f \in F} c_{o_f} \in N \wedge c_{I_f/J_f} \in M$$

Brahma encoding: take 2



parameter space

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

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

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

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

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

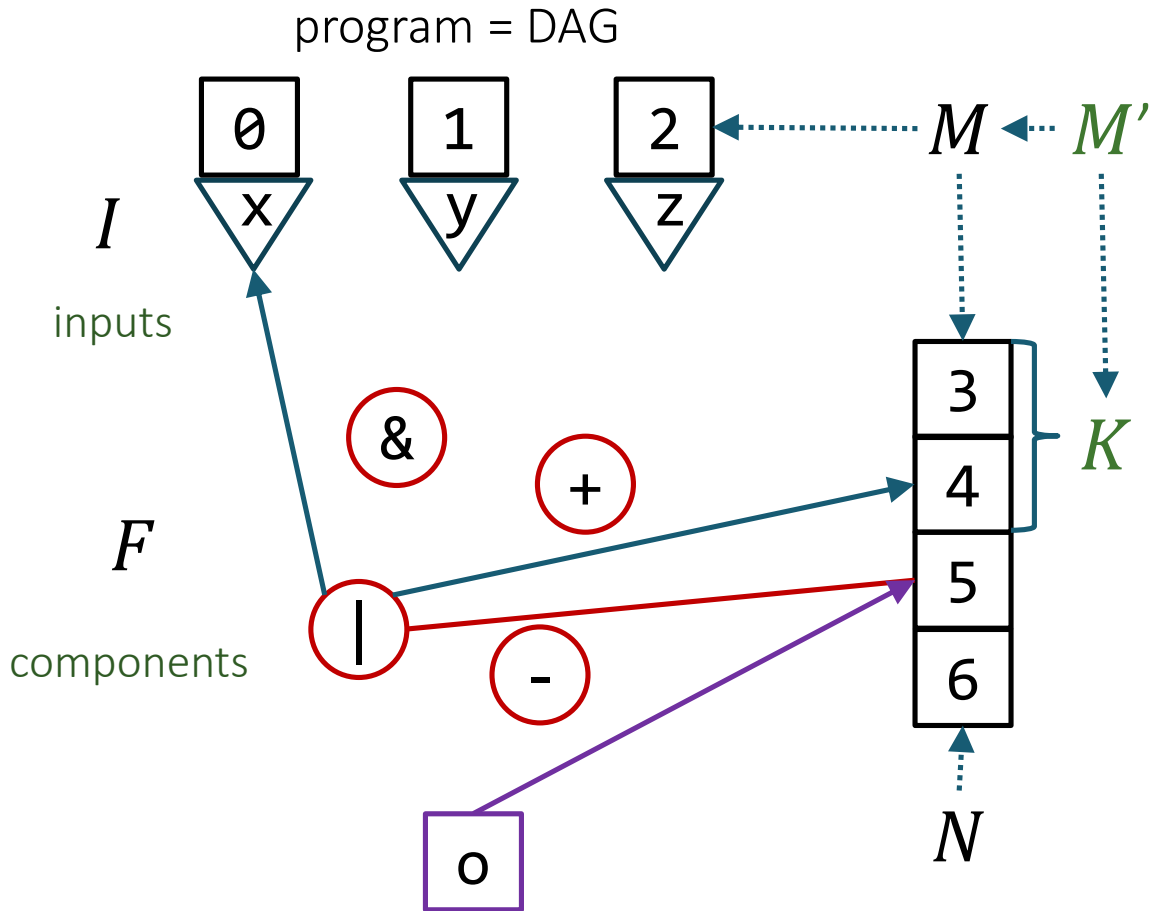
Behavioral Constraints? Structural Constraints? Search Strategy?

- 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?



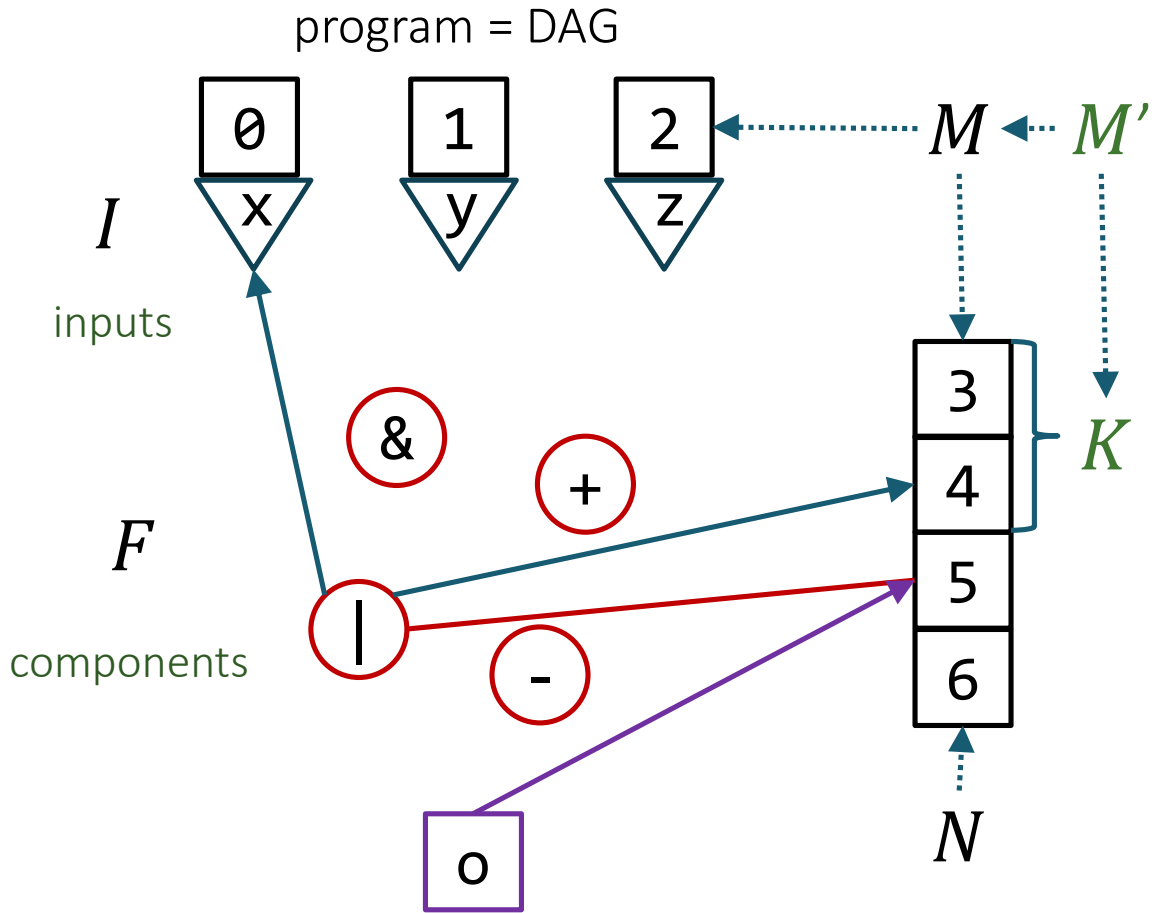
parameter space

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

$$\text{wf}(C) \equiv c_o \in \cancel{M} \wedge \bigwedge_{f \in F} c_{o_f} \in \cancel{N} \wedge c_{I_f/J_f} \in \cancel{M}$$

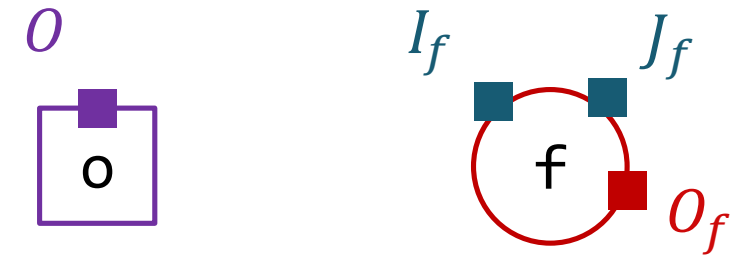
$$\wedge \bigwedge_{f, g \in F, f \neq g} c_{o_f} \neq c_{o_g} \wedge \bigwedge_{f \in F} c_{I_f/J_f} < c_{o_f}$$

Limit #components to K?



parameter space

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



$$\begin{aligned} \text{wf}(C) \equiv & c_o \in \cancel{M} \wedge \bigwedge_{f \in F} c_{o_f} \in N \wedge c_{I/J_f} \in M \\ & \wedge \bigwedge_{f, g \in F, f \neq g} c_{o_f} \neq c_{o_g} \wedge \bigwedge_{f \in F} c_{I/J_f} < c_{o_f} \end{aligned}$$

A linear encoding with uninterpreted functions

$$t_0 = 0$$

$$t_1 = 1$$

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

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

	Production	1st child	2nd child	Nonterm	Output value on 10010101	
<i>line</i>	$p^l(\text{line})$	$c(\text{line}, 0)$	$c(\text{line}, 1)$	$n^l(\text{line})$	$v_l^\epsilon(\text{line})$...
0	0	*	*	<i>B</i>	0000	
1	1	*	*	<i>B</i>	0001	
2	$\text{bvadd}(B, B)$	1	0	<i>B</i>	0001	
3	$\text{bvadd}(B, B)$	2	1	<i>B</i>	0010	

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

$$\mathbf{n}^l(L - 1) = S \quad (5.3)$$

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

$$\begin{aligned} \forall i < L. \mathbf{p}^l(i) = op(N_0, \dots, N_j) \Rightarrow \\ \mathbf{n}^l(\mathbf{c}(i, 0)) = N_0 \wedge \dots \wedge \mathbf{n}^l(\mathbf{c}(i, j)) = N_j \\ \mathbf{c}(i, 0) < i \wedge \dots \wedge \mathbf{c}(i, j) < i \end{aligned}$$

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

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

Behavioral constraints:

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

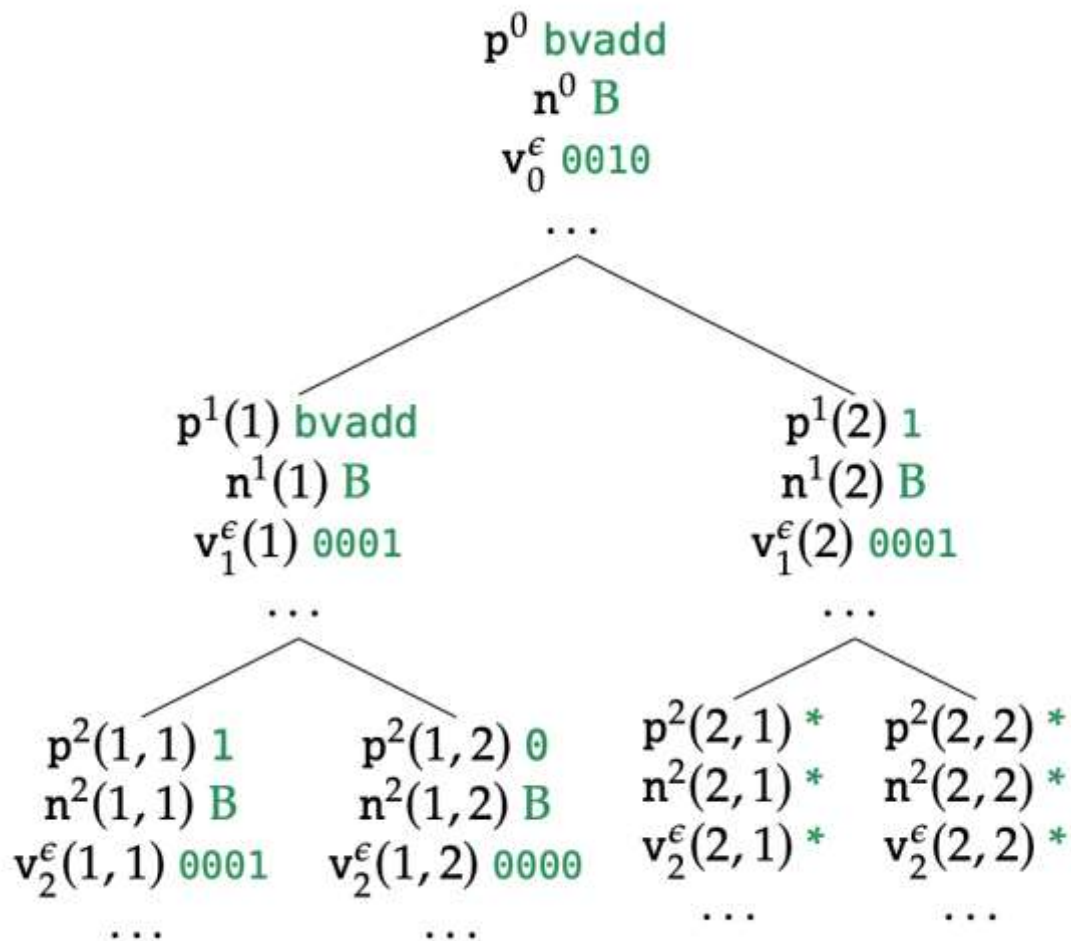
$$\forall \epsilon \in \mathcal{E}. v_l^\epsilon(L - 1) = M(\epsilon) \quad (5.4)$$

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

$$\forall i < L. \bigwedge_{op(N_1, \dots, N_j) \in \delta} p^l(i) = op(N_1, \dots, N_j) \Rightarrow \\ v_l^\epsilon(i) = \llbracket op \rrbracket(v_l^\epsilon(c(i, 0)), \dots, v_l^\epsilon(c(i, j)))$$

A tree encoding with uninterpreted functions

bvadd(bvadd(1, 0), 1)



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

$$n^0 = S \quad (5.1)$$

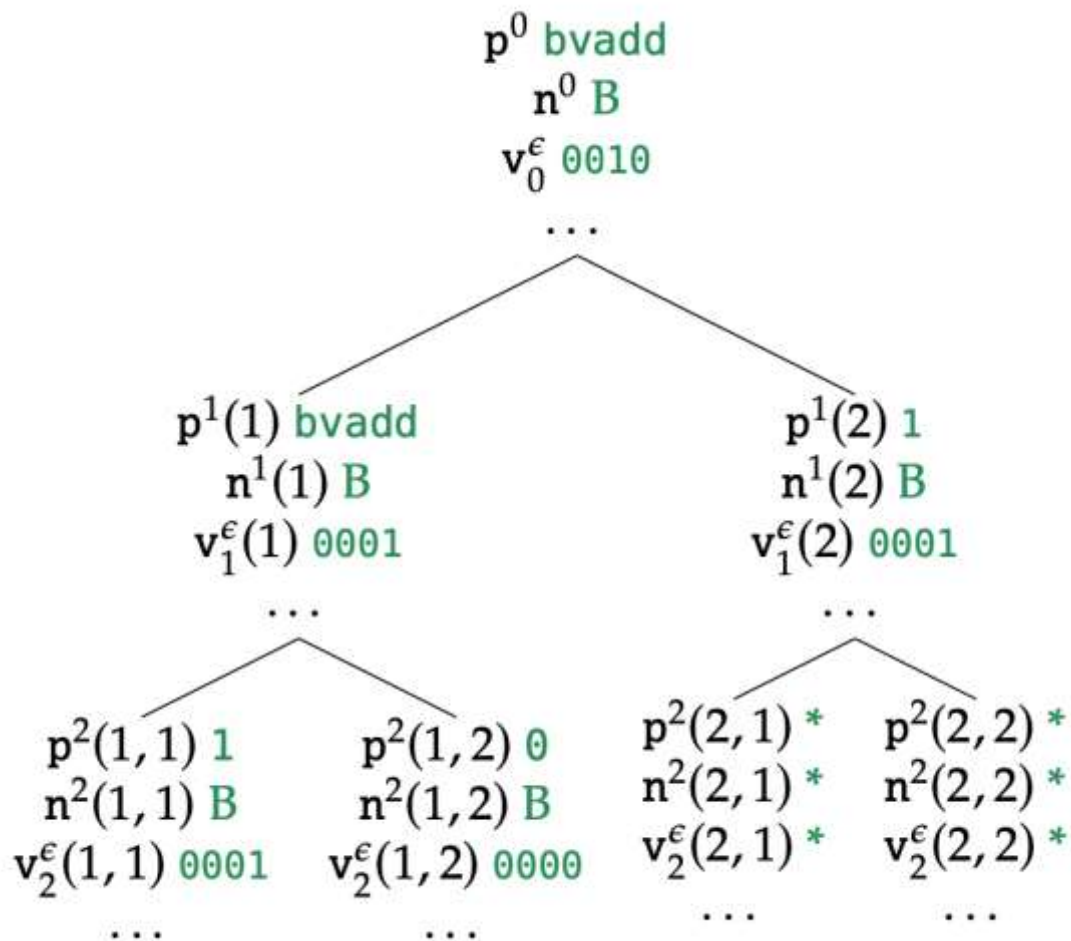
- For each path, if a node at that path is assigned a certain production, its children are appropriate non-terminals. Because we are interested in the children of a node, we should state this constraint only for non-leaf nodes. Thus the following constraint is added for any depth $d < D$:

$$\forall i_1, \dots, i_d < K.$$

$$p^d(i_1, \dots, i_d) = op(N_0, \dots, N_j) \Rightarrow \\ n^{d+1}(i_1, \dots, i_d, 0) = N_0 \wedge \dots \wedge n^{d+1}(i_1, \dots, i_d, j) = N_j$$

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 $d < D$:

$$\forall i_1, \dots, i_d < K.$$

$$n^d(i_1, \dots, i_d) = N \Rightarrow \bigvee_{r \in \delta(N)} p^d(i_1, \dots, i_d) = r$$

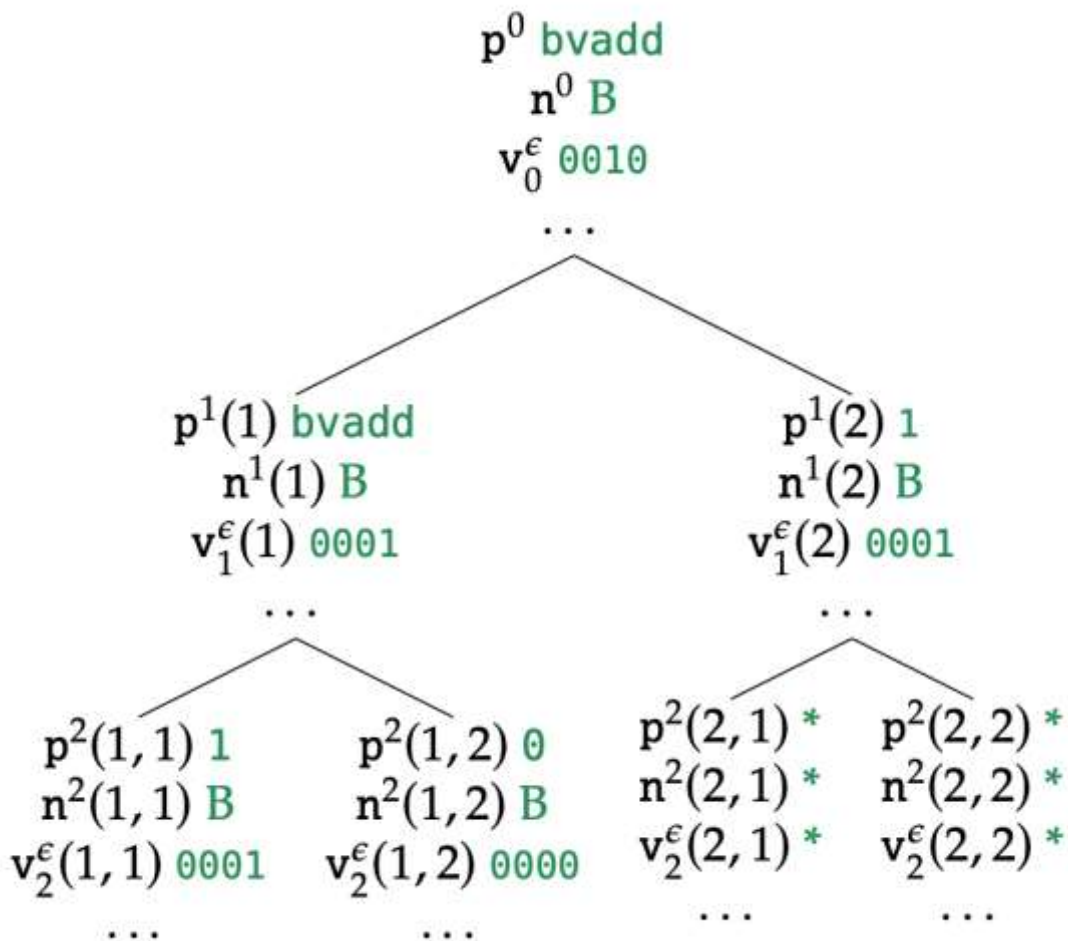
The constraint for the last level D is very similar but only allows 0-arity productions.

$$\forall i_1, \dots, i_D < K.$$

$$n^D(i_1, \dots, i_D) = N \Rightarrow \bigvee_{r \in \delta^0(N)} p^D(i_1, \dots, i_D) = r$$

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

$$\forall \epsilon \in \mathcal{E}. v_0^\epsilon = M(\epsilon) \quad (5.2)$$

- 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 $d < D$:

$$\forall i_1, \dots, i_d < K.$$

$$\bigwedge_{op(N_1, \dots, N_j) \in \delta} p^d(i_1, \dots, i_d) = op(N_1, \dots, N_j) \Rightarrow$$

$$v_d^\epsilon(i_1, \dots, i_d) = \llbracket op \rrbracket(v_{d+1}^\epsilon(i_1, \dots, i_d, 0), \dots, v_{d+1}^\epsilon(i_1, \dots, i_d, j))$$

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

$$\forall i_1, \dots, i_D < K.$$

$$\bigwedge_{op() \in \delta^0} p^D(i_1, \dots, i_D) = op() \Rightarrow v_D^\epsilon(i_1, \dots, i_D) = \llbracket op \rrbracket()$$

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

