#23: Brahma and Type-directed synthesis

Sankha Narayan Guria

EECS 700: Introduction to Program Synthesis

Brahma

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

Brahma encoding: take 1

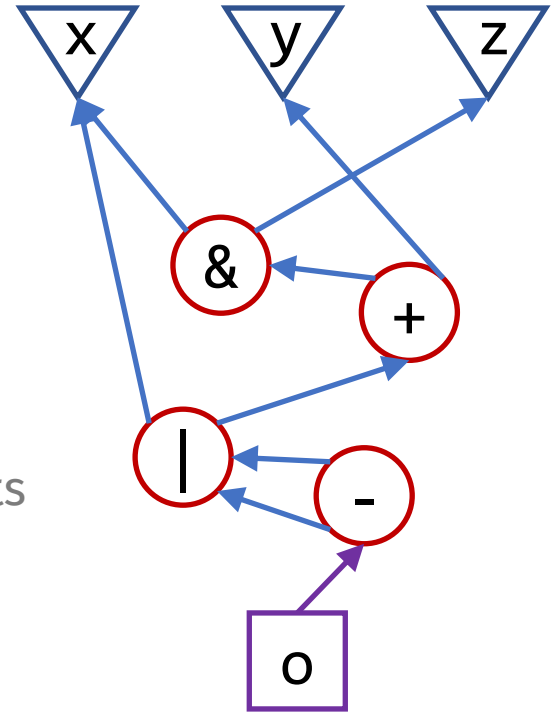
program = DAG

I

inputs

F

components

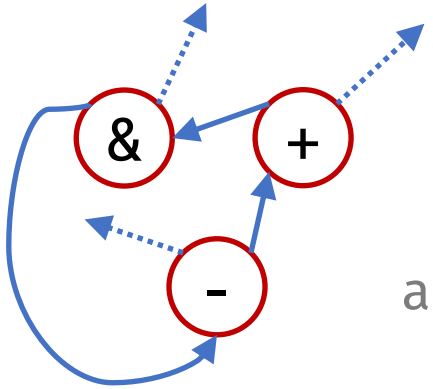


$V = I \cup F$ nodes

parameter space

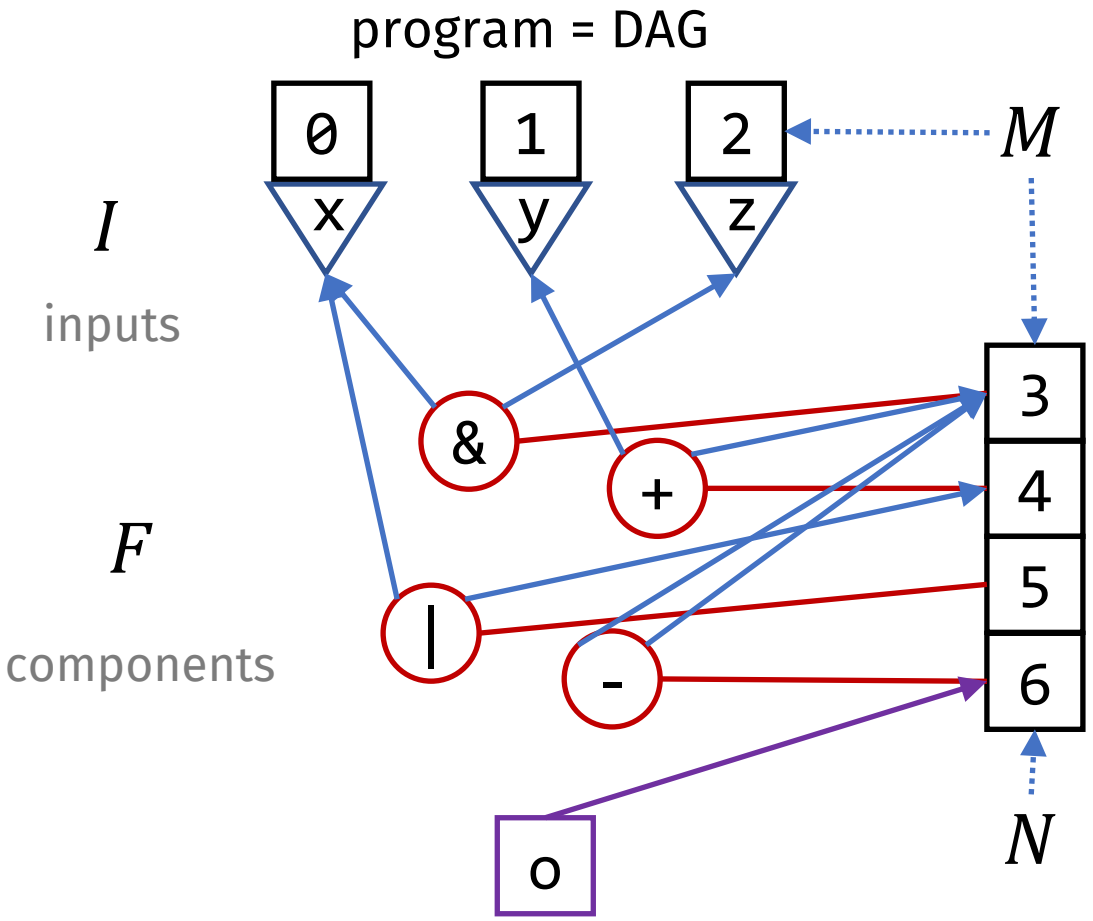
$$C = \{c_o : V\} \cup \bigcup_{f \in F} \{c_1^f, c_2^f : V\}$$

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



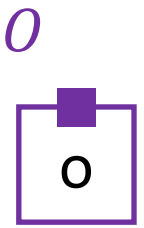
hard to encode acyclicity!

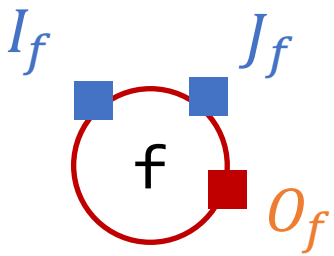
Brahma encoding: take 2



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

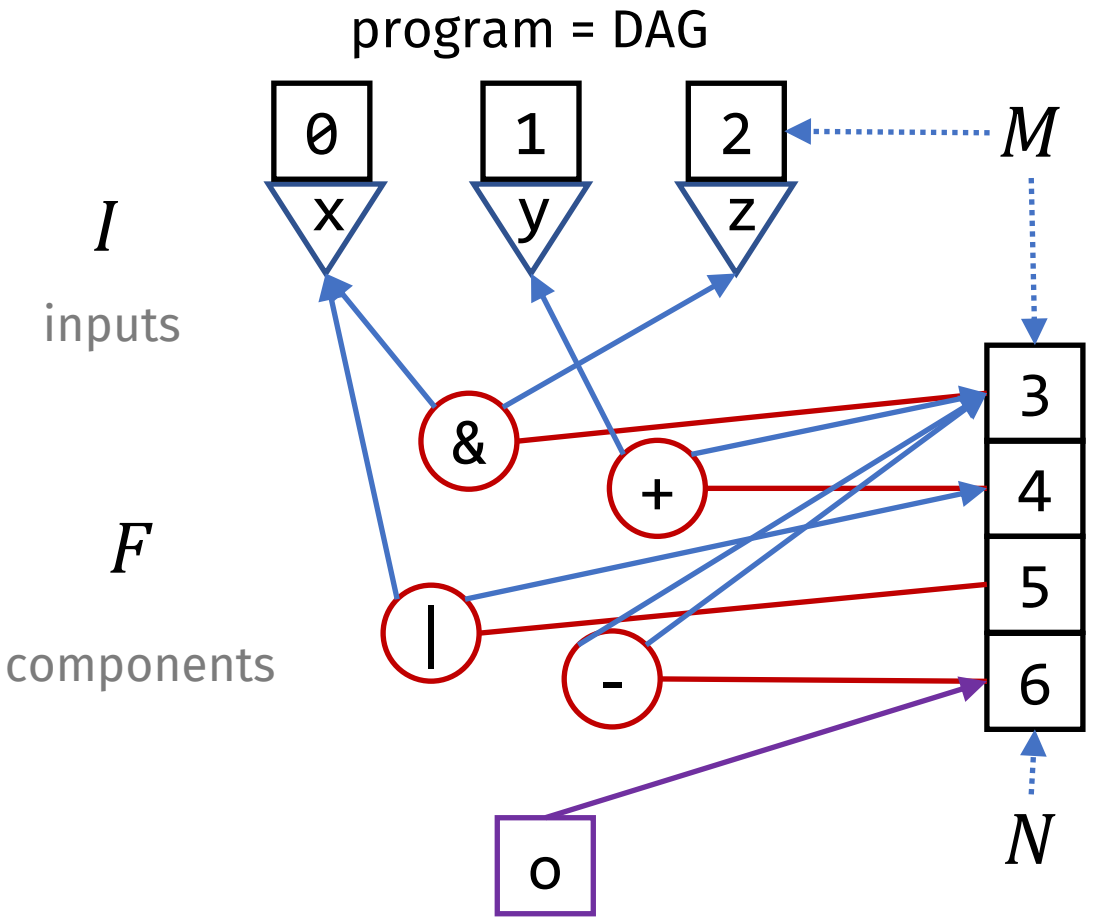
parameter space





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

Brahma encoding: take 2



parameter space

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

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

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

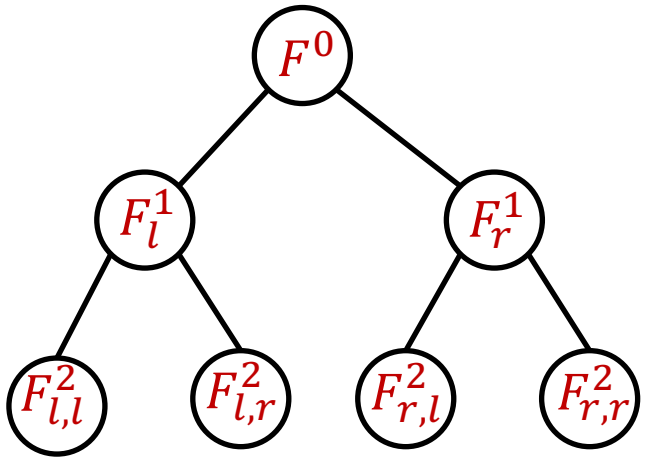
$$\wedge \bigwedge_{x,y \in T \cup I \cup \{O\}} c_x = c_y \Rightarrow x = y$$

Brahma: contributions

- SMT encoding of program space
  - sound? complete? solver-friendly?
  - more compact than alternatives*
- SMT solver can guess constants
  - e.g. 0x55555555 in P23

Alternative encodings

Tree encoding



Linear encoding

$$t_0 = F_0(t_{I0}, t_{J0})$$
$$t_1 = F_1(t_{I1}, t_{J1})$$
$$\dots$$
$$t_N = F_N(t_{IN}, t_{JN})$$

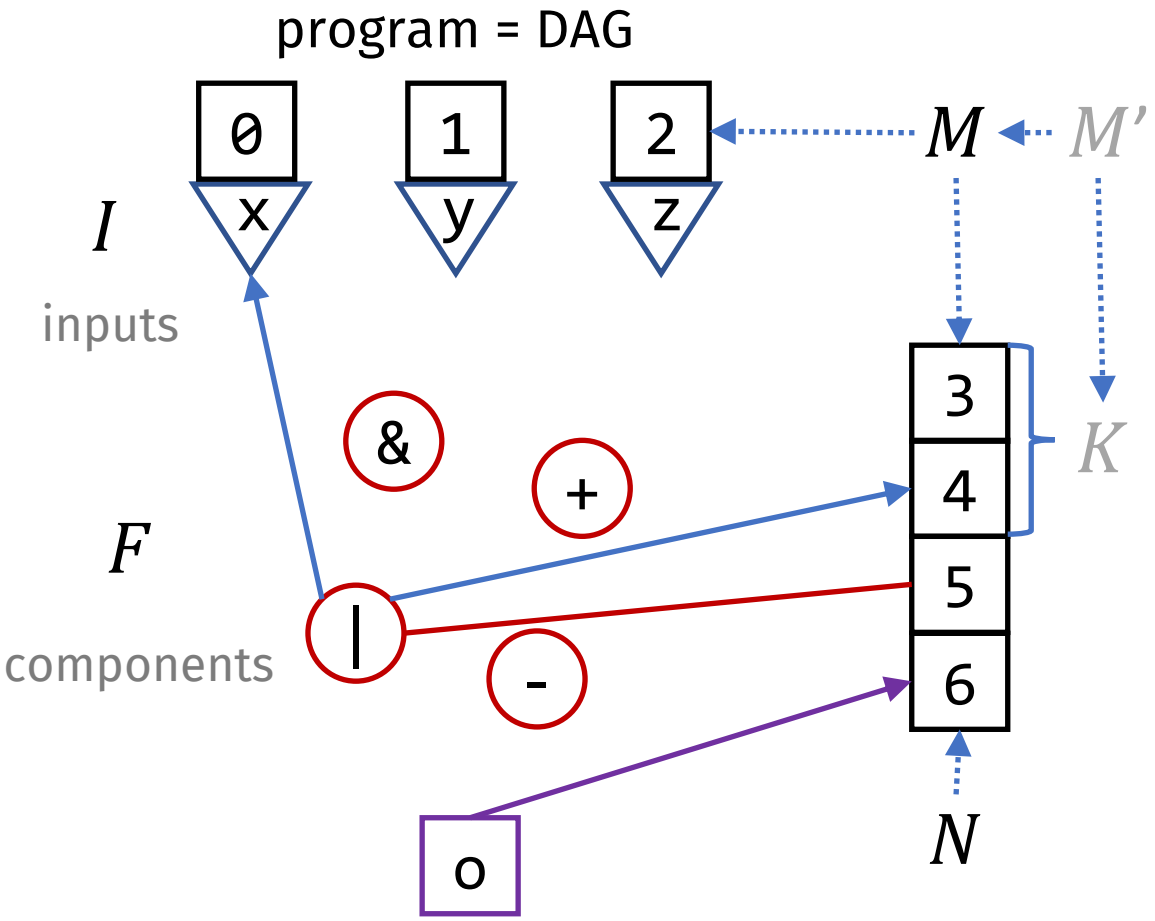
Brahma: limitations

- 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

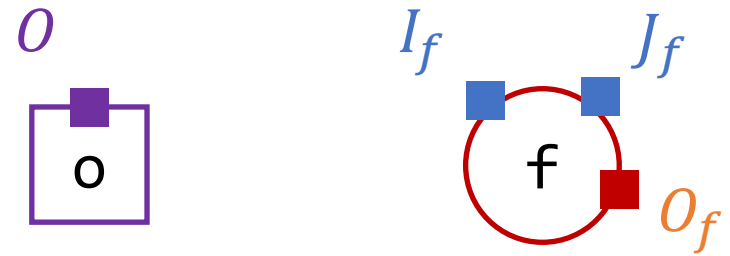
- Behavioral Constraints? Structural Constraints? Search Strategy?
  - First-order formula
  - A multiset of components + straight-line program
  - Constraint based + CEGIS
- Can we represent these structural constraints as a grammar?
  - Yes and no
  - No because grammars cannot encode multiplicities
    - 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

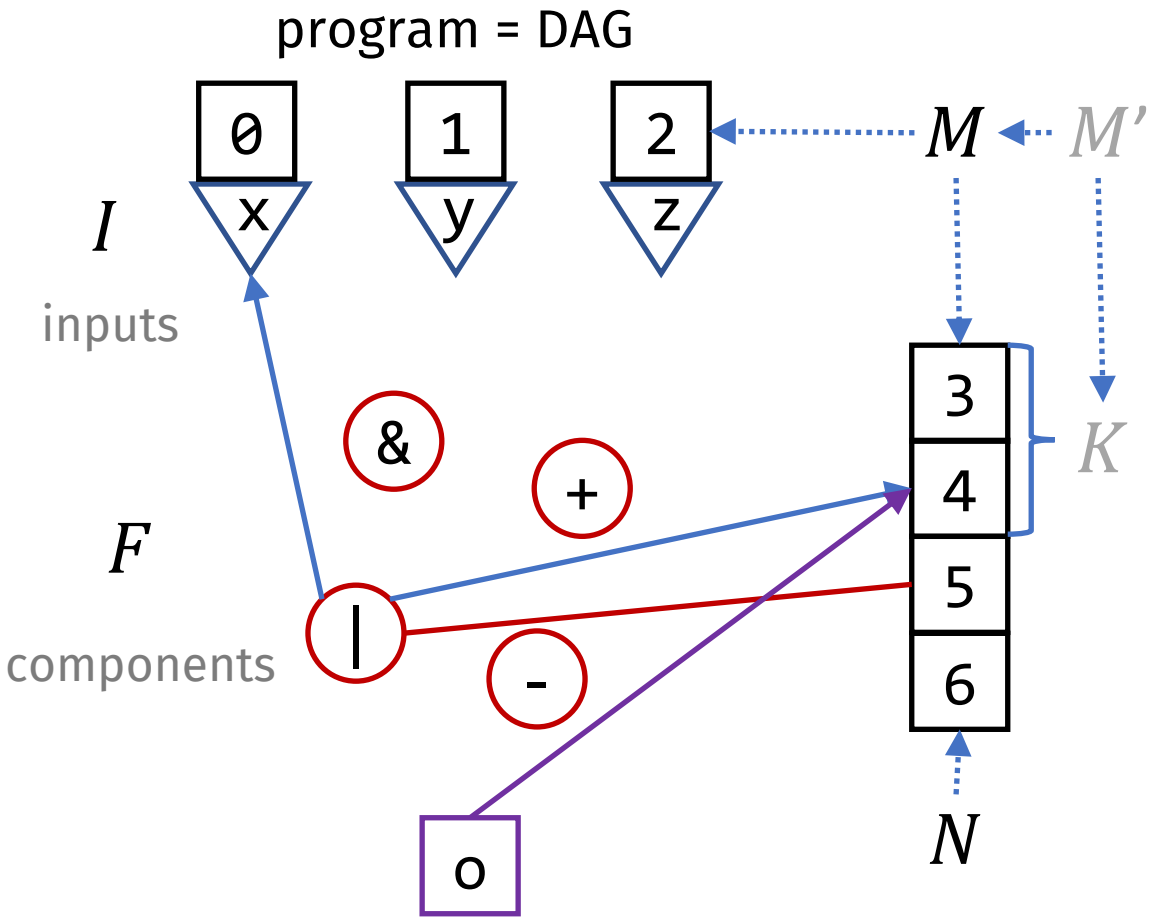
$$C = \{c_o : \text{Int}\} \cup \bigcup_{f \in F} \{c_{O_f}, c_{I_f}, c_{J_f} : \text{Int}\}$$



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

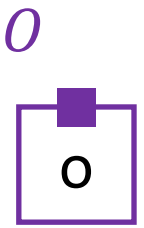
$$\wedge \bigwedge_{f,g \in F, f \neq g} c_{O_f} \neq c_{O_g} \wedge \bigwedge_{f \in F} c_{I/J_f} < c_{O_f}$$

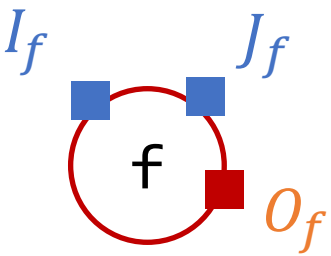
Limit #components to K?



parameter space

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





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

$$\wedge \bigwedge_{f,g \in F, f \neq g} c_{O_f} \neq c_{O_g} \wedge \bigwedge_{f \in F} c_{I/J_f} < c_{O_f}$$