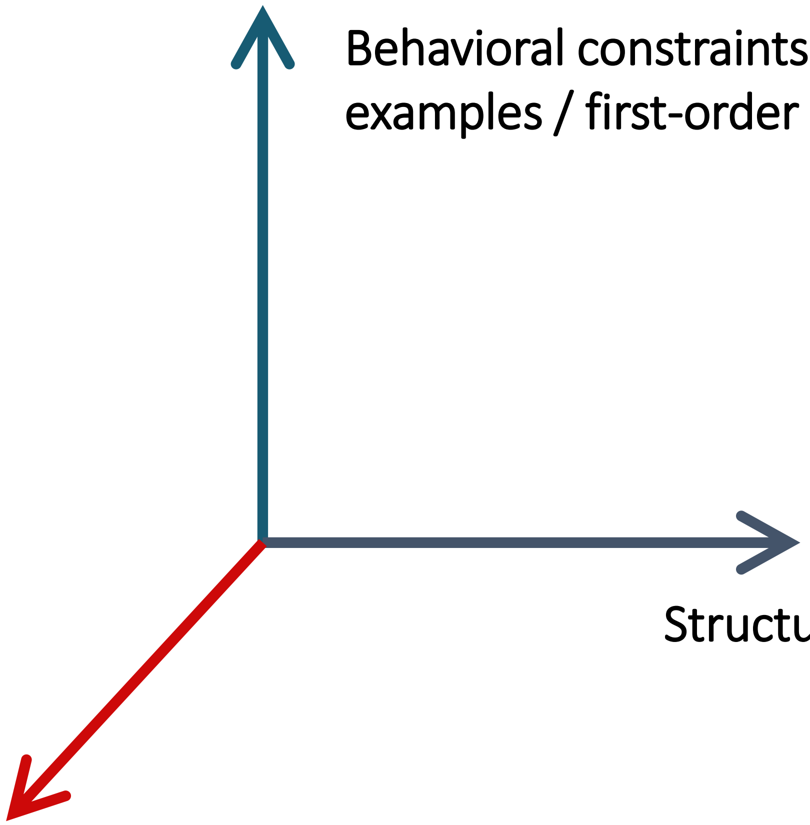
Lecture 8
Constraint-based search

Nadia Polikarpova

The problem statement

Behavioral constraints =
examples / first-order formula



Structural constraints = we'll see...

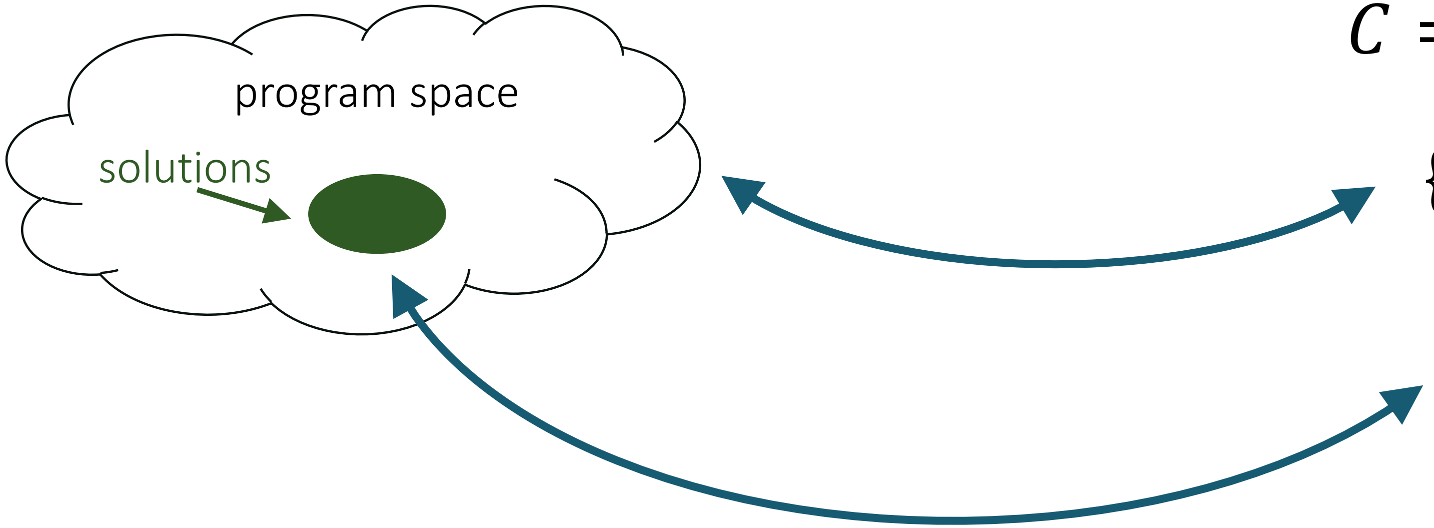
Search strategy?

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?



parameter space

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

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

well-formedness constraint

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

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}`

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

`x` is a two-bit word

`(x = x`$_h$`x`$_l$`)`

`E = [11 → 01]`

parameter space

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

```
decode[0] → x
decode[1] → x & 1
```

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

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

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

SAT solver → `Model {c→1}`

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

SMT encoding: example

program space

x + N | x * N

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

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
```

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

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

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

SMT solver →

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

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

What is a good encoding?

Sound

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

Complete

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

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

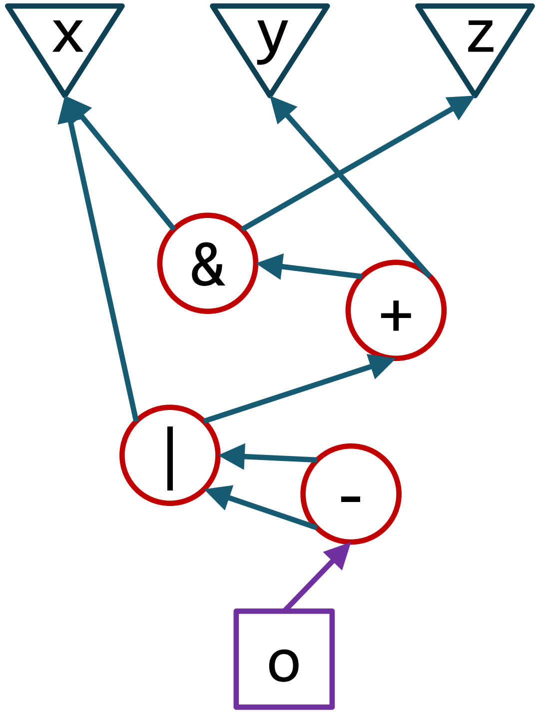
program = DAG

I

inputs

F

components

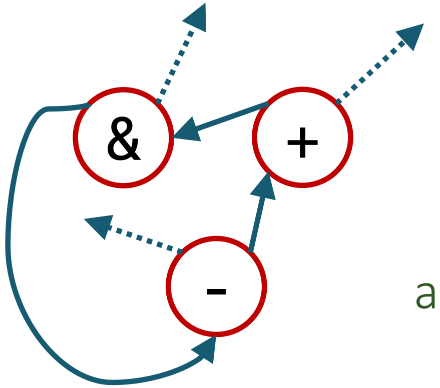


$V = I \cup F$ nodes

parameter space

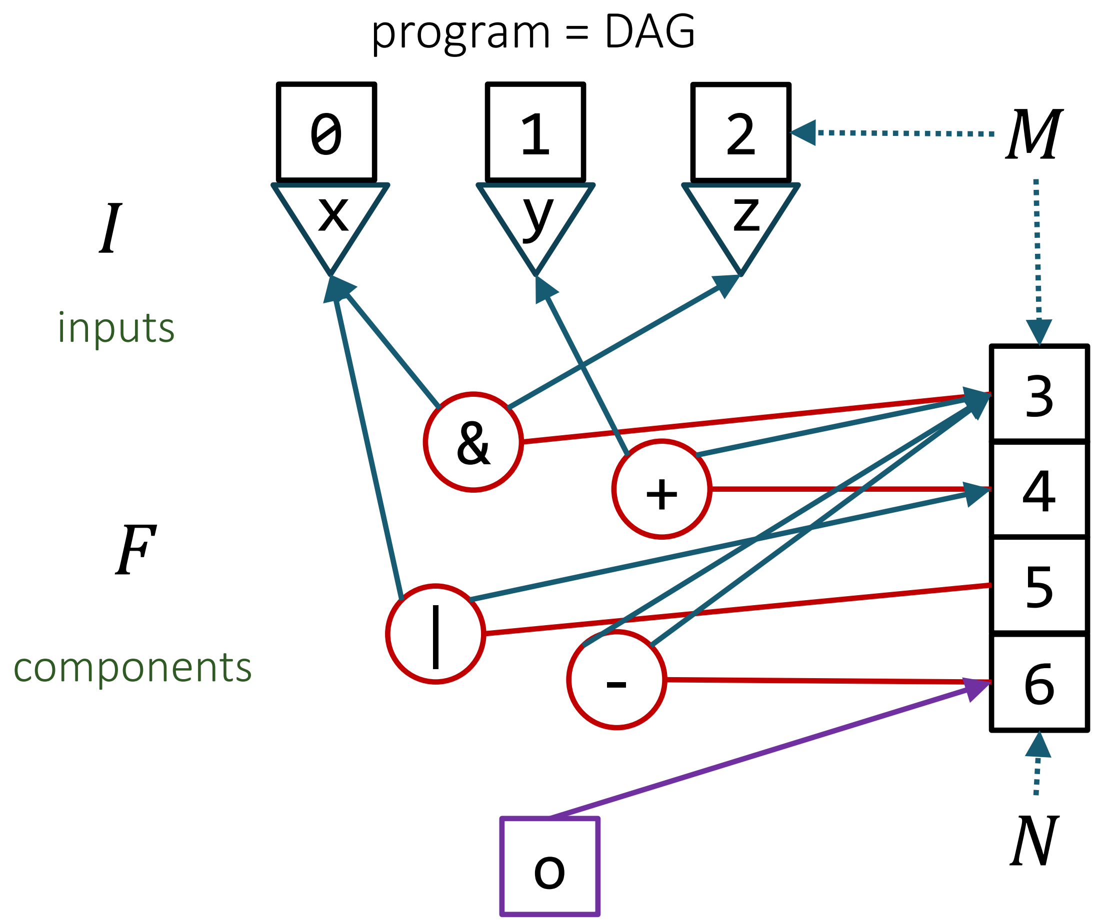
$$C = \{out : V\} \cup \bigcup_{f \in F} \{inl_f, inr_f : V\}$$

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



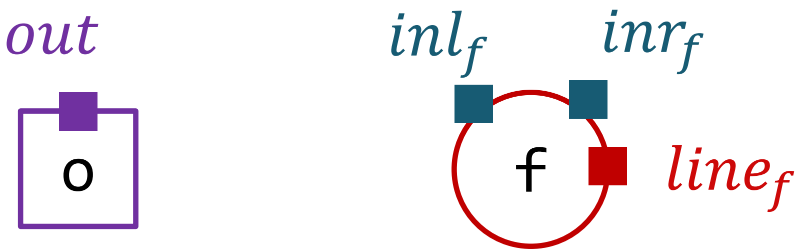
hard to encode acyclicity!

Brahma encoding: take 2



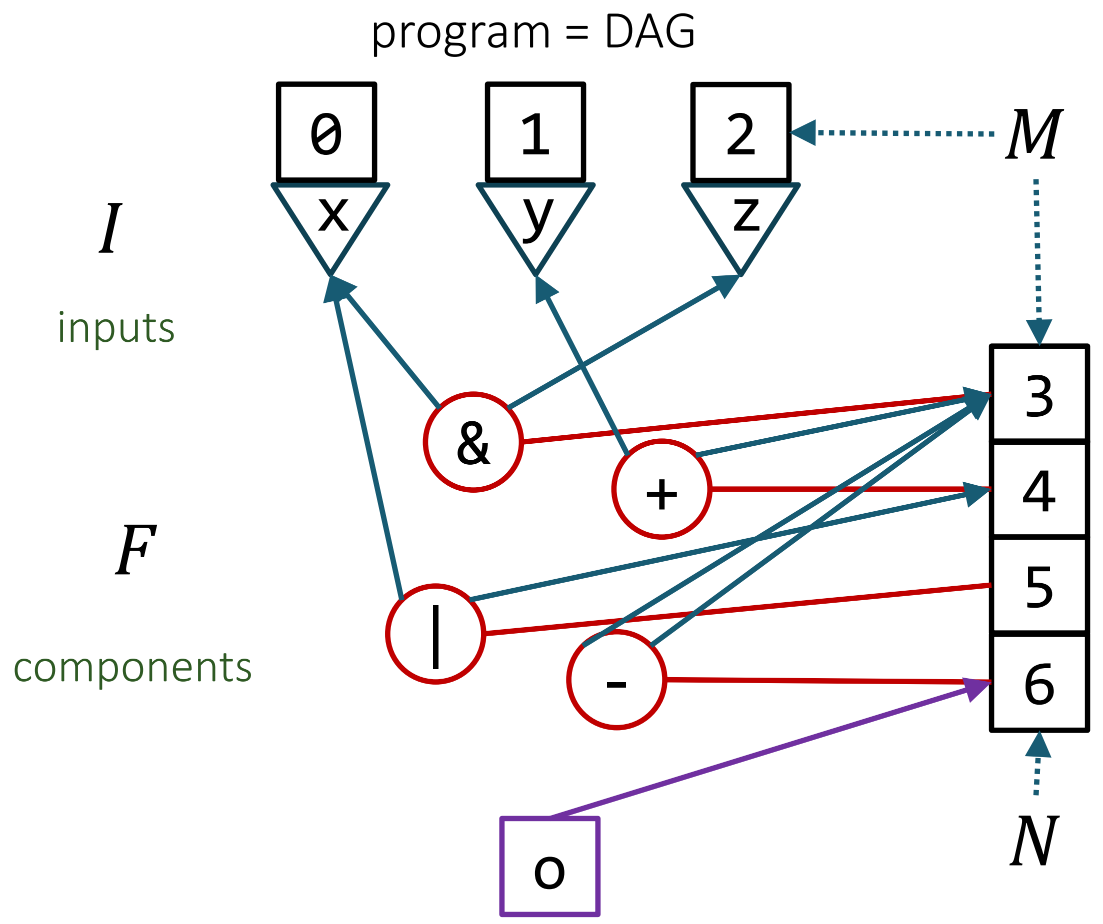
parameter space

$$C = \{out\colon \mathrm{Int}\} \cup \bigcup_{f \in F} \{line_f, inl_f, inr_f\colon \mathrm{Int}\}$$



$$\mathrm{wf}(C) \equiv out \in M \;\wedge \bigwedge_{f \in F} line_f \in N \wedge in^{l/r}{}_f \in M$$

Brahma encoding: take 2



parameter space

$$C = \{out\colon \mathrm{Int}\} \cup \bigcup_{f \in F} \{line_f, inl_f, inr_f\colon \mathrm{Int}\}$$

$$T = \bigcup_{f \in F} \{L_f, R_f, O_f\colon \mathrm{BitVector}\}$$

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

$$\wedge \bigwedge_{f,g \in F \cup I} line_f = in^{l/r}{}_g \Rightarrow O_f = L/R_f$$

$$\wedge\ line_f = out \Rightarrow O_f = O$$

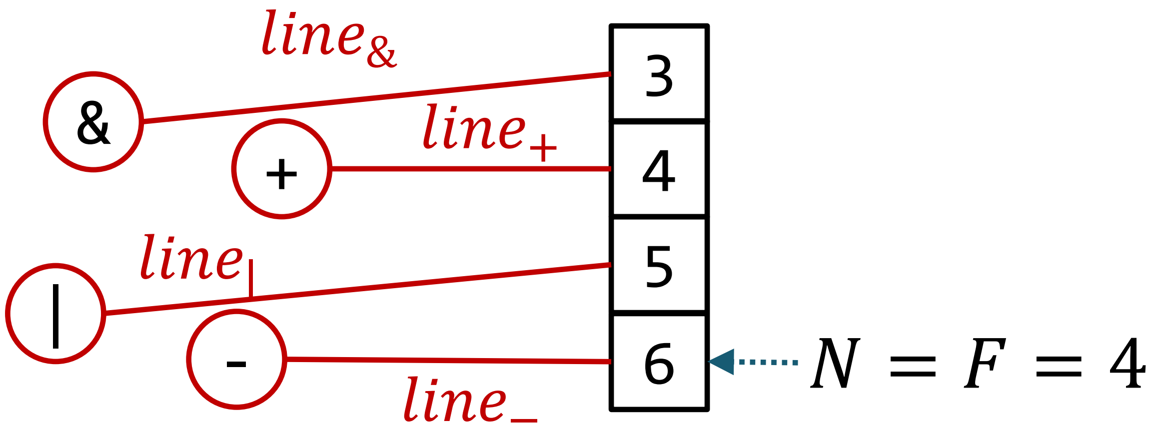
Brahma: questions

Behavioral Constraints? Structural Constraints? Search Strategy?

- First-order formula
- Expressions over a multiset of components
- Constraint based + CEGIS

Alternative encodings

Brahma encoding



Linear encoding

$$t_3 = F_3(inl_3, inr_3)$$

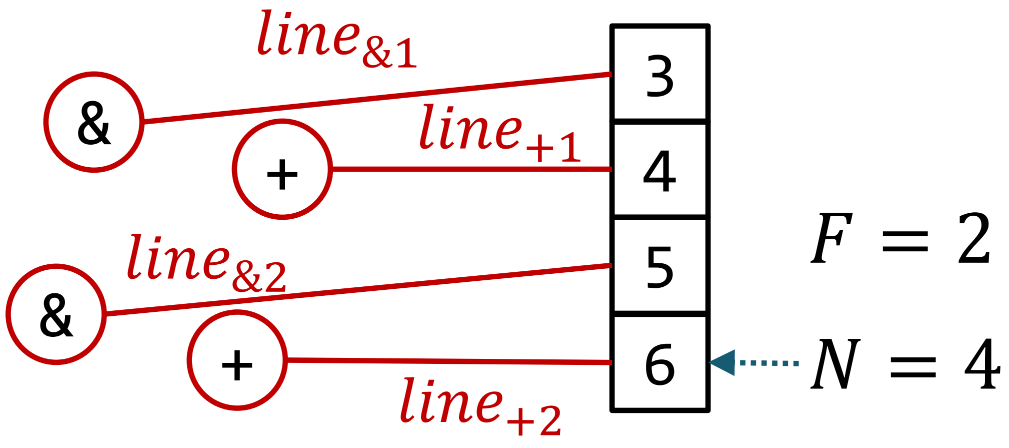
$$t_4 = F_4(inl_4, inr_4)$$

$$t_5 = F_4(inl_5, inr_5)$$

$$t_6 = F_6(inl_6, inr_6)$$

Alternative encodings

Brahma encoding



Linear encoding

$$t_3 = F_3(inl_3, inr_3)$$

$$t_4 = F_4(inl_4, inr_4)$$

$$t_5 = F_4(inl_5, inr_5)$$

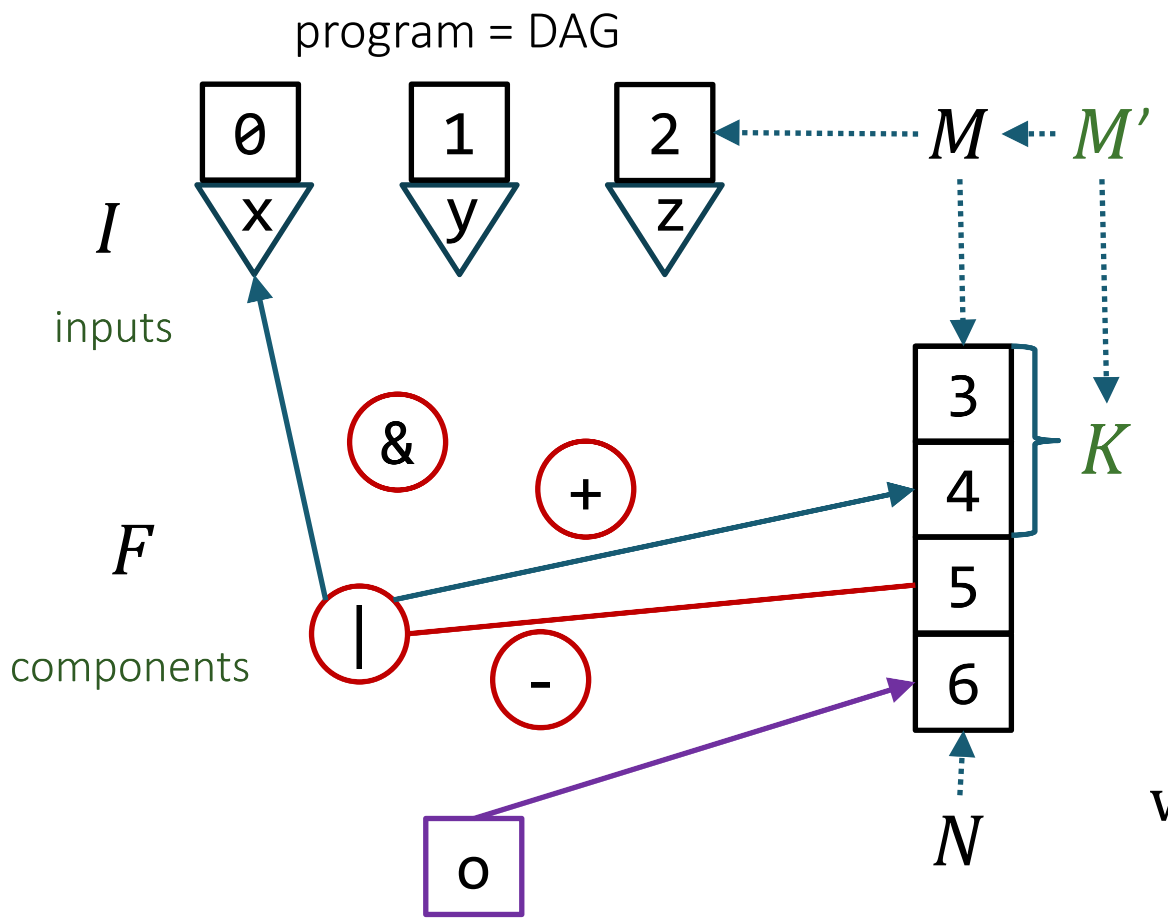
$$t_6 = F_6(inl_6, inr_6)$$

$N!$ assignmetns to $line$ vars

F^N assignmetns to F vars

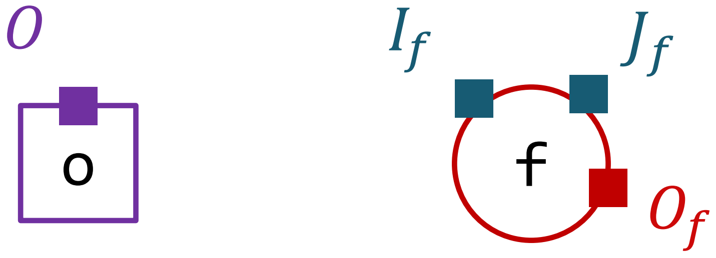
Brahma encoding is worse is component multiplicities are high! (N >> F)

Limit #components to K?



parameter space

$$C = \{out\!: \mathrm{Int}\} \cup \bigcup_{f \in F} \{line_f, inl_f, inr_f\!: \mathrm{Int}\}$$

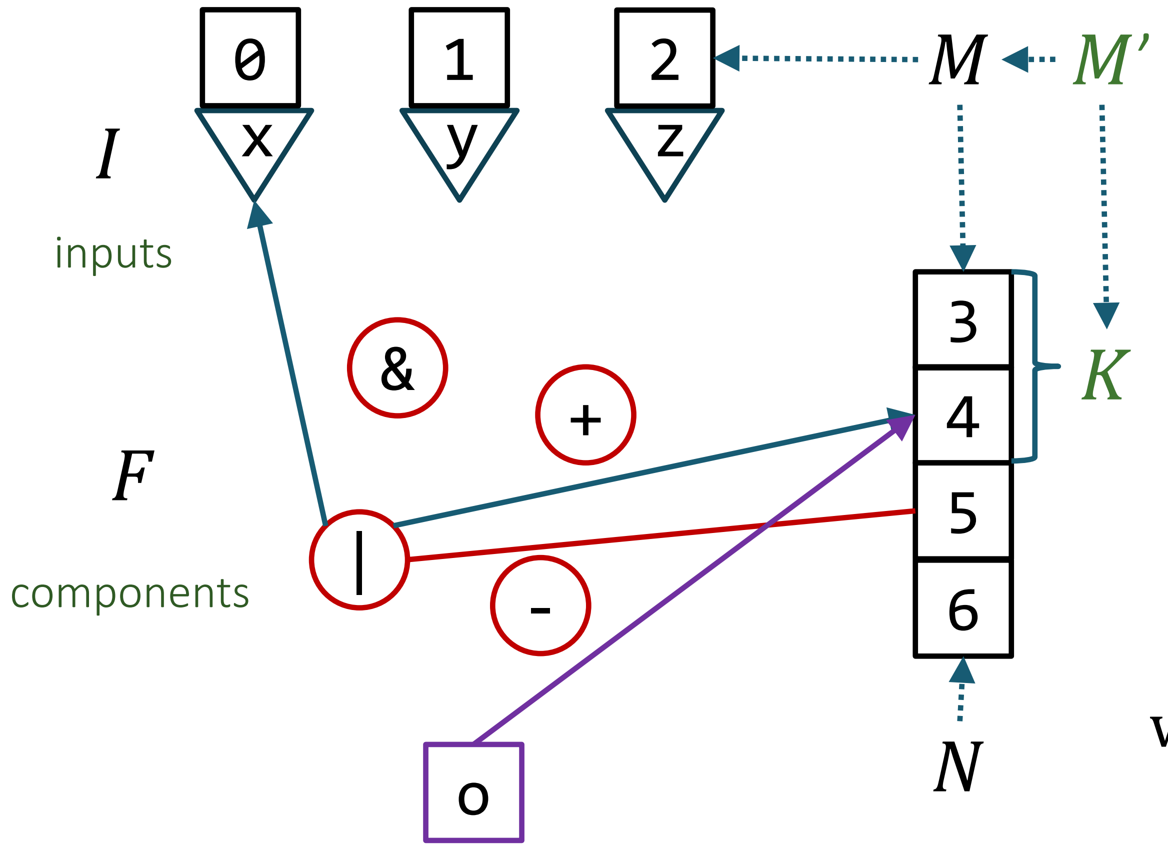


$$\mathrm{wf}(C) \equiv out \in \overset{M'}{\cancel{M}} \wedge \bigwedge_{f \in F} line_f \in \overset{K}{\cancel{N}} \wedge in^{l/r}{}_f \in \overset{M'}{\cancel{M}}$$

$$\wedge \bigwedge_{f,g \in F, f \not\equiv g} line_f \neq line_g \wedge \bigwedge_{f \in F} in^{l/r}{}_f < line_f$$

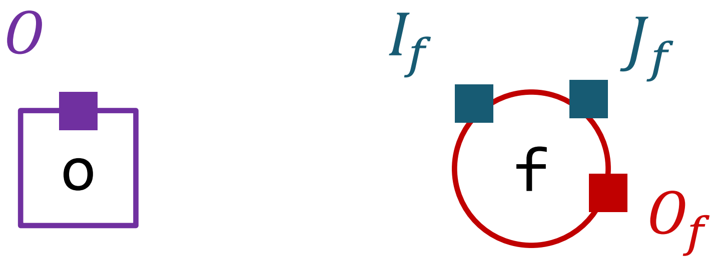
Limit #components to K?

program = DAG



parameter space

$$C = \{out: \text{Int}\} \cup \bigcup_{f \in F} \{line_f, inl_f, inr_f: \text{Int}\}$$



$$\text{wf}(C) \equiv out \in \overset{M'}{\cancel{M}} \wedge \bigwedge_{f \in F} line_f \in N \wedge in^{l/r}{}_f \in M$$

$$\wedge \bigwedge_{f,g \in F, f \not\equiv g} line_f \neq line_g \wedge \bigwedge_{f \in F} in^{l/r}{}_f < line_f$$

Imprecise component specs

ExAllSolver($\psi_{\mathtt{wfp}}$, $\phi_{\mathtt{lib}}$, $\psi_{\mathtt{conn}}$, $\phi_{\mathtt{spec}}$):

// $\exists L \forall \vec{I}, O, T : \psi_{\mathtt{wfp}} \wedge (\phi_{\mathtt{lib}} \wedge \psi_{\mathtt{conn}} \Rightarrow \phi_{\mathtt{spec}})$
// is a synthesis constraint
// Output: synthesis failed or values for L
$\mathcal{S} := \{\vec{I}_0\}$ // $\vec{I}_0$ is an arbitrary input
while (1) {
model := T-SAT($\exists L, O_1, \ldots, O_n, T_1, \ldots, T_n : \psi_{\mathtt{wfp}}(L) \wedge$
$\bigwedge_{\vec{I}_i \in \mathcal{S}} (\phi_{\mathtt{lib}}(T_i) \wedge \psi_{\mathtt{conn}}(\vec{I}_i, O_i, T_i, L)$
$\wedge \phi_{\mathtt{spec}}(\vec{I}_i, O_i))$);
if (model $\neq \perp$) {
curr$L$:= model$|_L$
} else {
return("synthesis failed");
}
model := T-SAT($\exists \vec{I}, O, T : \psi_{\mathtt{conn}}(\vec{I}, O, T, \mathtt{curr}L) \wedge$
$\phi_{\mathtt{lib}}(T) \wedge \neg \phi_{\mathtt{spec}}(\vec{I}, O)$);
if (model $\neq \perp$) {
$\vec{I}_1$:= model$|_{\vec{I}}$; $\mathcal{S}$:= $\mathcal{S} \cup \{\vec{I}_1\}$;
} else {
return(curr$L$);
}
}

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

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

Comparison of search strategies