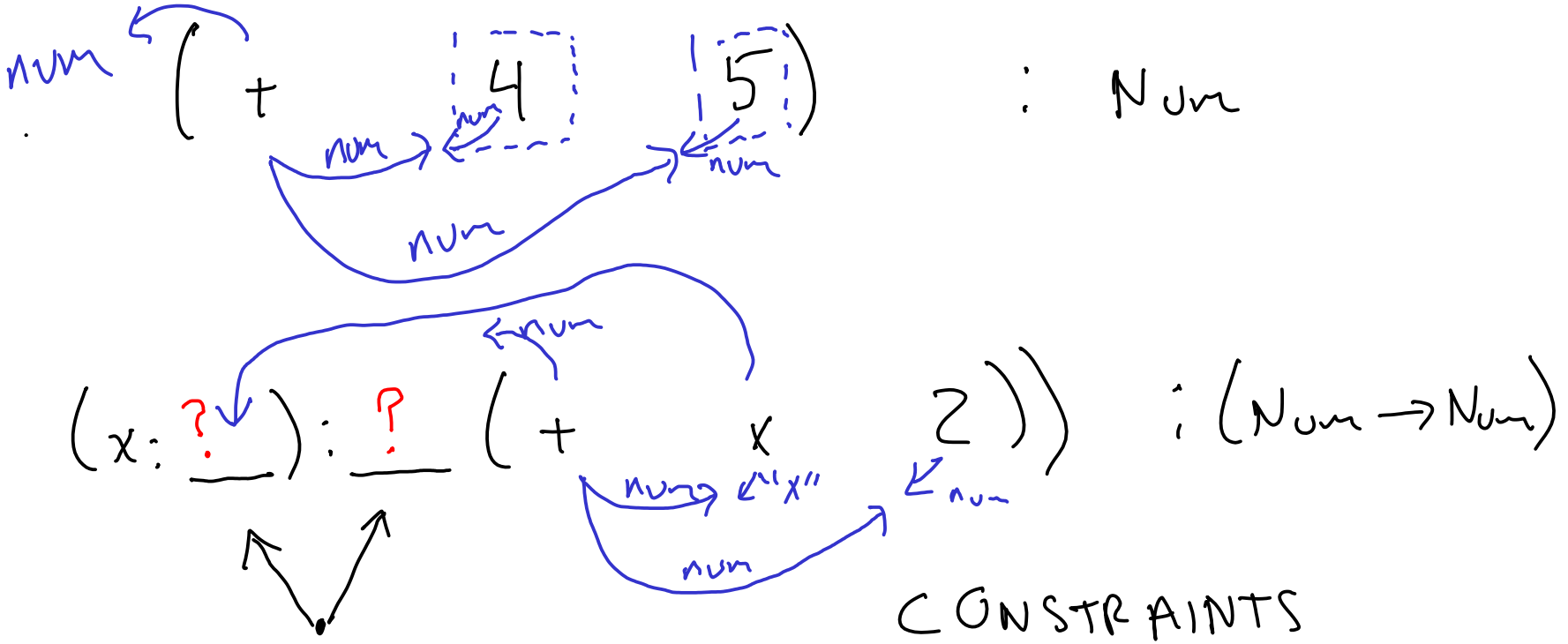
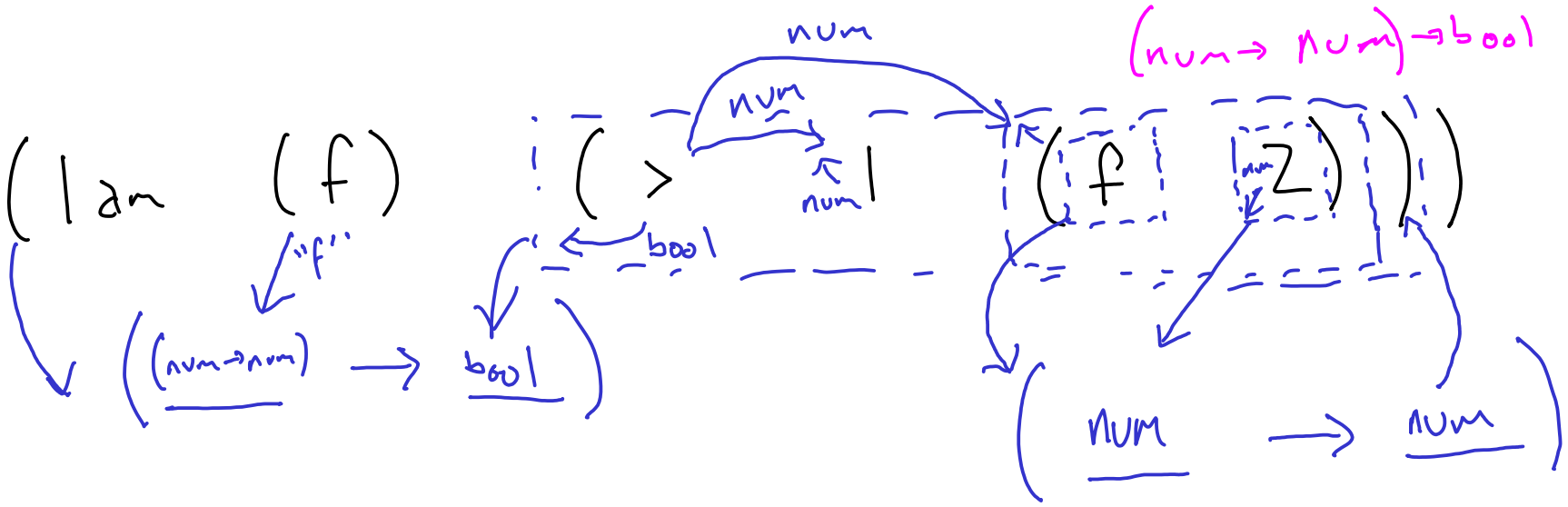


type-infer :: UntypedExpr → TypedExpr

Step 1: constraint generation

Step 2: solve constraints


num
num
(>
4
bool
num
(+ num num 1 num 2))
num


num
(num→num)→bool
num
(lam (f)
num
(> num 1
(f num 2)))
"f"
bool
((num→num) → bool)
(num → num)

gen-constraints :: Expr → Set<Constraints>

→ e = e-num(n)

e = e-bool(b)

e = e-id(x)

e = e-plus(e1, e2)

e = (+ 4 5)

false

type(e) = Num
type(4) = Num
type(5) = Num
type(4) = Num
type(5) = Num

① type(e) = Num ②

type(e) = Bool

type(e) = id-type(x) ③

type(e) = Num

type(e1) = Num

type(e2) = Num

type(false) = Num

type(false) = Bool

e = e-lam(x, b)

type(e) =

(id-type(x) → type(b)) ④

e = e-app(f, a)

type(f) = (type(a) → type(c))

1. Type of expr
2. Base types (num, bool, str, ...)
3. Type of ids
4. Compound types ($\tau \to \tau$, (List τ), (Pair τ_1 τ_2) ....)

Solve for:

- Any contradictions (in which expr(s))
- Type of each expr

solve-constraints :: Set<Constraint> $\to$ Map<Expr, Type>*

$e = (\text{lam}\ \ (x)\ \ 4)$ $\quad type(e) = (id\text{-}type("x") \rightarrow type(4))$

$type(4) = Num$