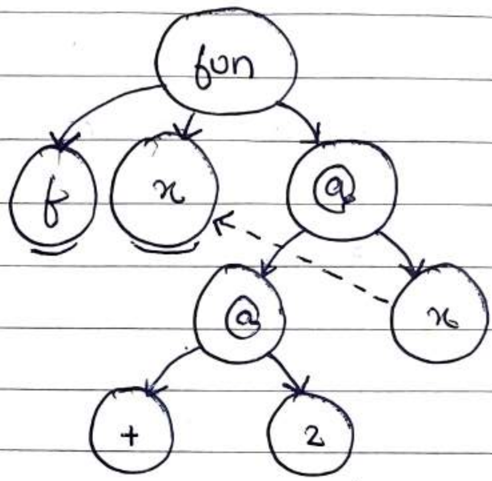
TYPE INFERENCE ALGORITHM

- parse program to build parse tree
- assign type variables to nodes in tree
- generate constraints
  - from environment: literals (2), built-in operators (+), known functions (tail).
  - from form of parse tree: application, abstraction nodes.
- solve constraints using unification
- determine types of top-level declarations.

STEP 1: PARSE PROGRAM

- parse program text to construct parse tree.

let $f\ x = 2 + x$

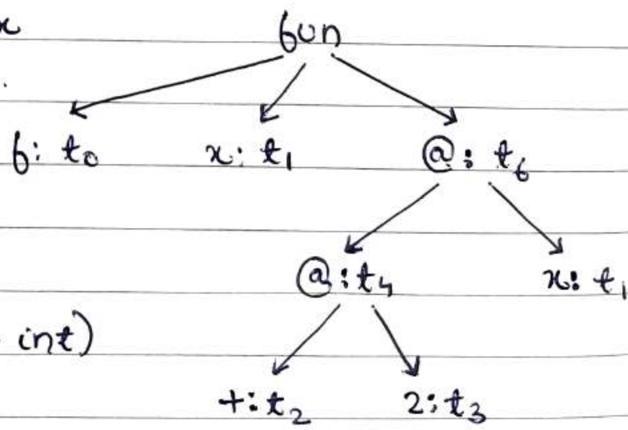


infix operators are converted to curried function application during parsing:

$2 + x \rightarrow (+)\ 2\ x$

STEP 3: ADD CONSTRAINTS

let $f\ x = 2 + x$



$t_3 = int$

$t_2 = int \rightarrow (int \rightarrow int)$

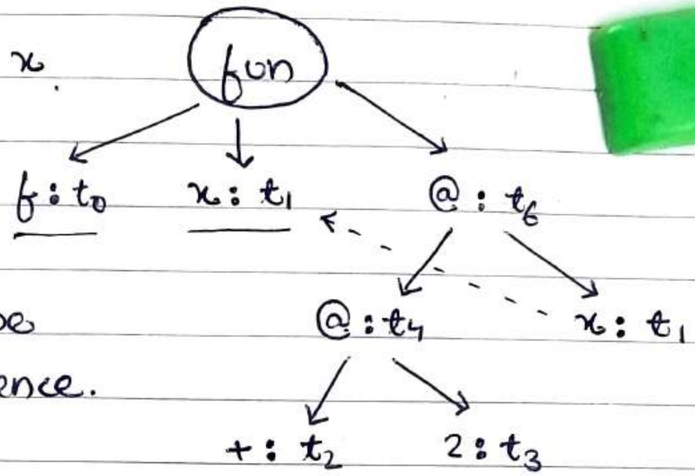
$t_2 = t_3 \rightarrow t_4$

$t_4 = t_1 \rightarrow t_6$

$t_0 = t_1 \rightarrow t_6$

STEP 2: ASSIGN TYPE VARIABLES TO NODES.

let f x = 2 + x.



variables are given same type as binding ocorrence.

STEP 4: SOLVE CONSTRAINTS.

(1) $t_0 = t_1 \to t_6$

$$\left\{\begin{array}{l} t_4 = t_1 \to t_6 \\ t_2 = t_3 \to t_4 \\ t_2 = int \to (int \to int) \end{array}\right.$$

$t_3 = int$

$t_3 \to t_4$
$= int \to (int \to int)$
$\Downarrow$
$t_3 = int$
$t_4 = int \to int.$

(2) $t_0 = t_1 \to t_6$

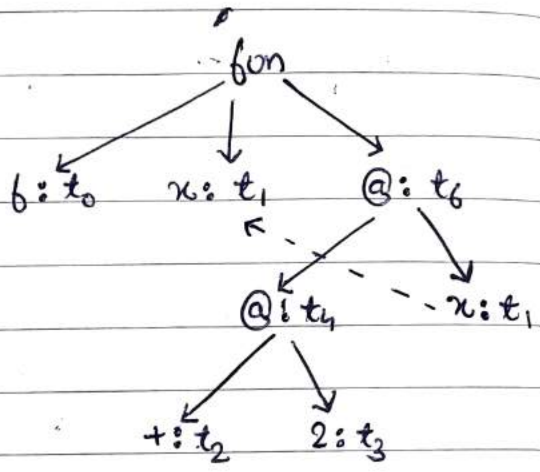
$$\left\{\begin{array}{l} t_4 = t_1 \to t_6 \\ t_4 = int \to int \end{array}\right.$$

$t_1 \to t_6$
$= int \to int$
$\Rightarrow t_1 = int, \quad t_6 = int$

(3) $t_0 = int \to int$

STEP5: DETERMINE TYPE OF DECLARATION

let $f\ x = 2 + x$



$t_0 = int \to int$

$t_1 = int$

$t_2 = int \to int \to int$

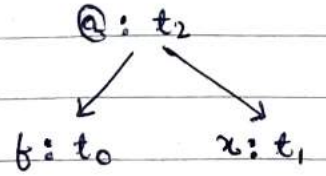
$t_3 = int$

$t_4 = int \to int$

$t_6 = int \to int$

val f : $int \to int$ = <fun>

CONSTRAINTS FROM APPLICATION NODES



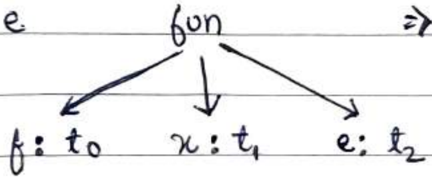
$f\ x$

$\Rightarrow\ t_0 = t_1 \to t_2$

domain range

CONSTRAINTS FROM ABSTRACTION NODES.

let $f\ x = e$



$\Rightarrow\ t_0 = t_1 \to t_2$

domain range