Types and Type Inference

Edward Z. Yang

What is a type?

Hindley-Milner type inference
and polymorphism

What is a type?

What is a type?

True :: Bool

What is a type?

expr :: type

What is a type?

expr :: $\boxed{\text{Bool}}$

$\boxed{\text{Int}}$

$\boxed{\text{Int} \to \text{Bool}}$

What is a type?

$$
\begin{aligned}
\tau \;::=\;& \mathrm{Int} \\
\mid\;& \mathrm{Bool} \\
\mid\;& \tau_1 \rightarrow \tau_2 \\
\mid\;& \ldots
\end{aligned}
$$

What is a type ... really?

A TYPE IS: A Way to Prevent Errors

```
print(100 + "bob")
```

A TYPE IS: A Way to Prevent Errors

```
function apply(f, x) {
    return f(x);
}
```

A TYPE IS: A Way to Prevent Errors

The world's MOST POPULAR lightweight formal method!

A TYPE IS: A method of program organization

2 degrees Farenheit

2 degrees Celsius

A TYPE IS: A method of program organization

```haskell
-- This function takes two integers
-- and returns their sum.
plus :: Int -> Int -> Int
plus a b = a+b
```

A TYPE IS: A method of program organization

```
-- This function takes a function
-- in its first argument and a value
-- in its second argument.
apply :: (a -> b) -> a -> b
apply f x = f x
```

A TYPE IS: A method of program organization

```
data Set k

empty :: Set k
insert :: k -> Set k -> Set k
delete :: k -> Set k -> Set k
member :: k -> Set k -> Bool
```

(Modularity later this quarter!)

A TYPE IS: A Hint to the Compiler

```
x = record["key"]
```

A TYPE IS: A Hint to the Compiler

x = hashTableLookup(record, "key")

A TYPE IS: A Hint to the Compiler

$$x = *(\text{record} + \text{keyOffset})$$

A TYPE IS:

The central organizing principle of the theory of programming languages.

—Bob Harper

Types → Type Errors

Type Errors are language dependent

size 10 array

arr[200]

Type Errors are language dependent

size 10 array

arr[200]

Segfault

C/C++

Type Errors are language dependent

size 10 array

arr[200]

Out of bounds!

Haskell / Java

Type Errors are language dependent

null pointer

arr[200]

Segfault

C/C++

Type Errors are language dependent

null pointer

arr[200]

Null pointer derefence

Java

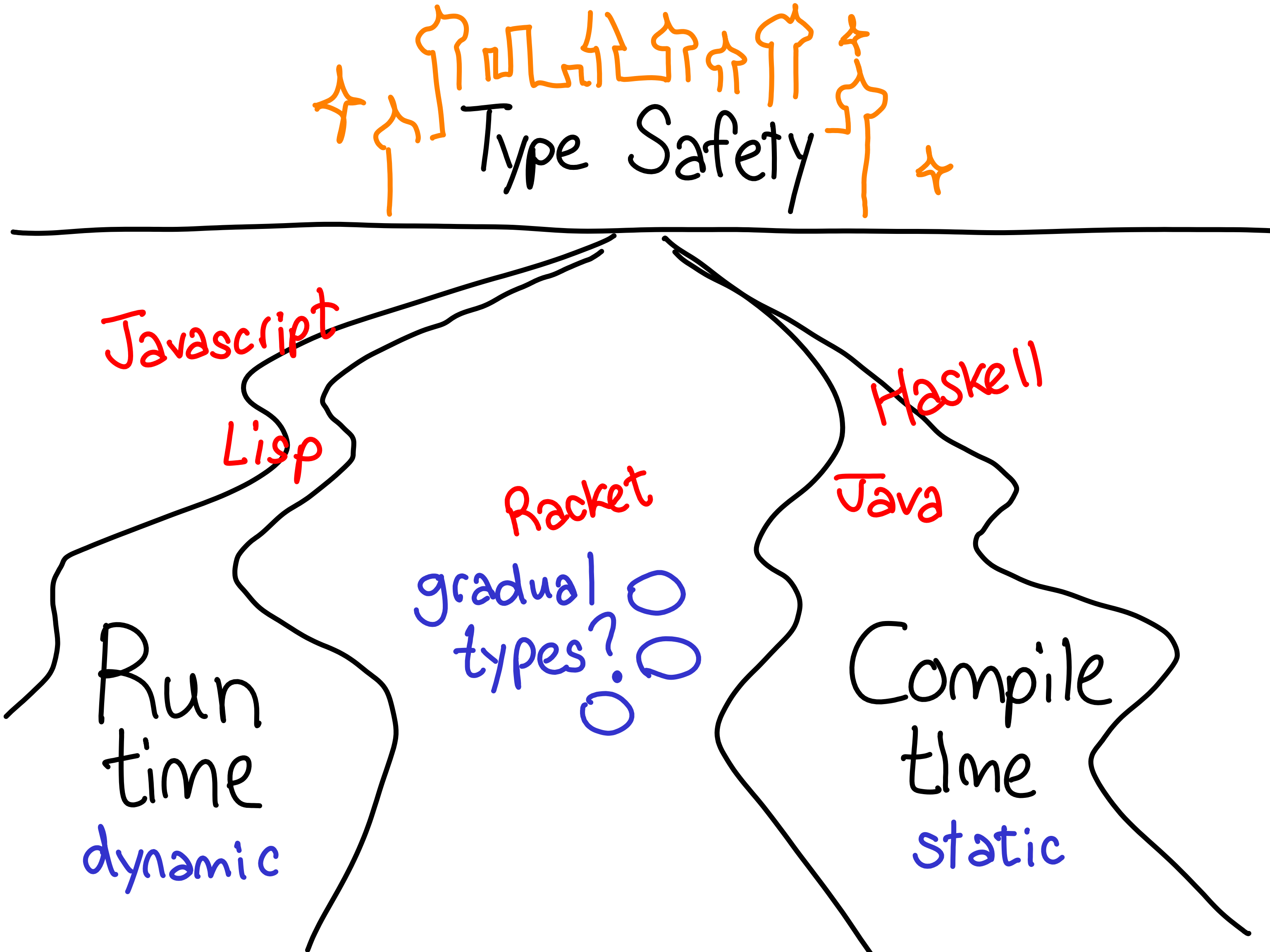
Type Errors are language dependent

maybe type

~~arr ! 2ØØ~~

Cannot unify Maybe Array
with expected Array

Haskell

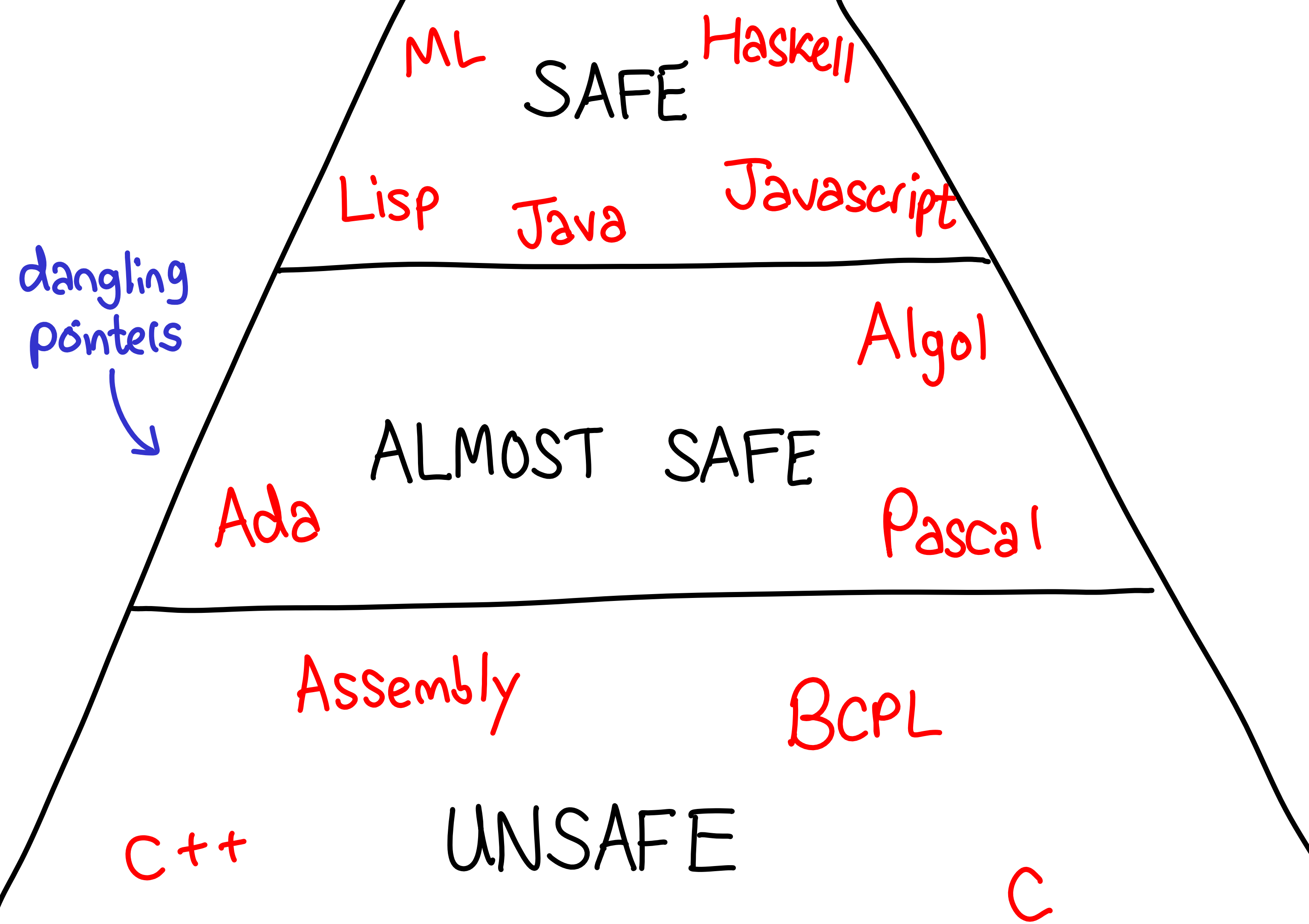


Type Safety
Javascript
Lisp
Racket
gradual types?
Haskell
Java
Run time
dynamic
Compile time
static

expressivity versus information

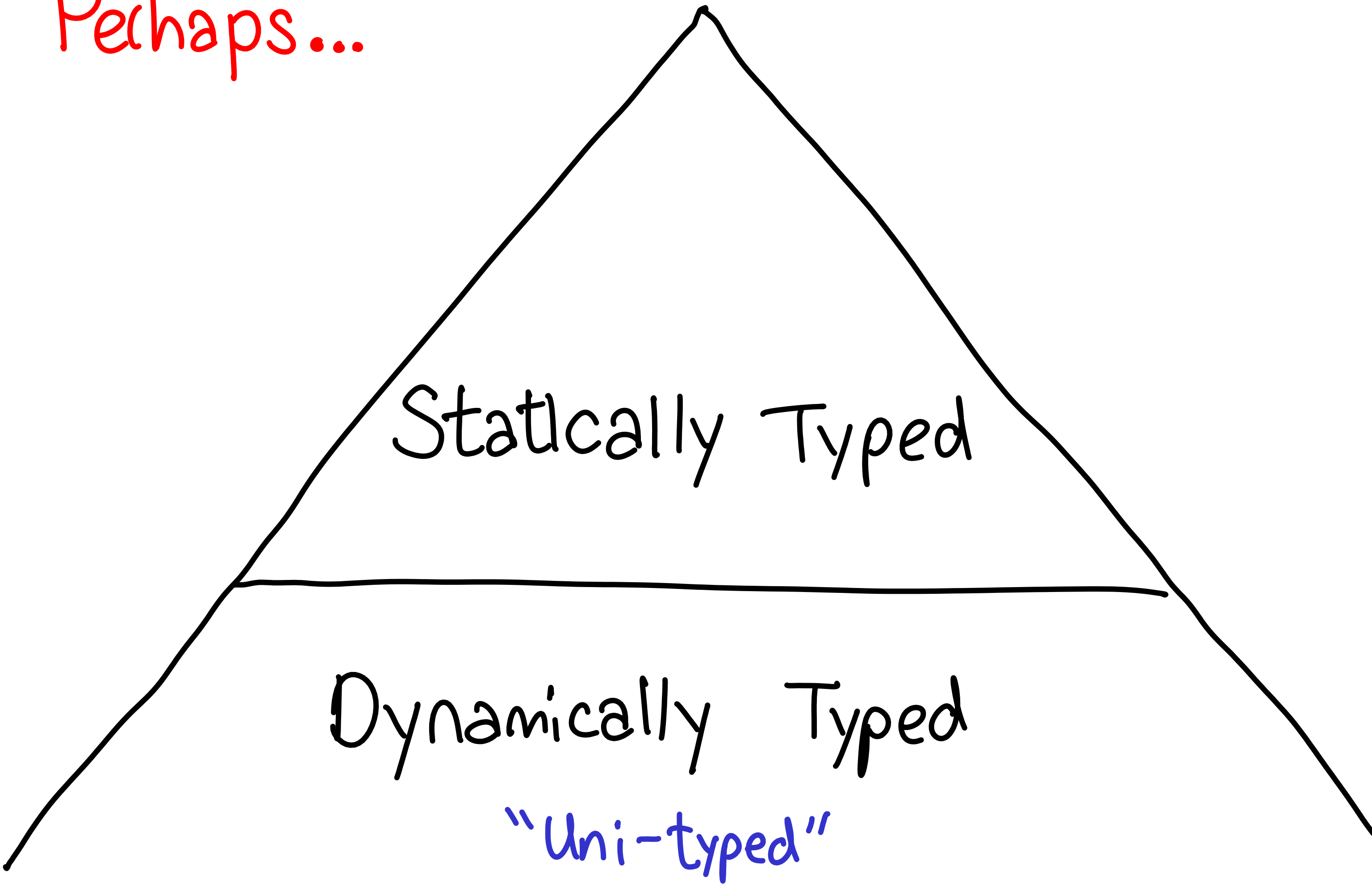
```
function f(x) {
    return x<10 ? x : x();
}
```

(dependent types?)



ML
SAFE
Haskell
Lisp
Java
Javascript
dangling pointers
Algol
ALMOST SAFE
Ada
Pascal
Assembly
BCPL
UNSAFE
C++
C

Perhaps...



(pause)

Hindley-Milner type inference

What is type inference?

```
int f(int x) { return x+1;}
```

What is type inference?

```
f(   x) { return x+1;}
```

I don't have to annotate all
my types? Sweet!

uHaskell

Haskell Subset

decl ::= name pat = exp

pat ::= id | (pat, pat) | pat : pat | []

exp ::= n | True | False | [] | id | (exp)

| exp op exp | exp exp | (exp, exp)

| if exp then exp else exp

type ::= type → type | [type] | (type, type) | Bool | Int

Lists, Booleans, Pairs, Integers

Type Inference by Example

$\boxed{\text{Ex 1}}$ The Basics

$\boxed{\text{Ex 2}}$ Polymorphism — the important one!

$\boxed{\text{Ex 3}}$ Data Types

$\boxed{\text{Ex 4}}$ Type Error: Cannot Unify

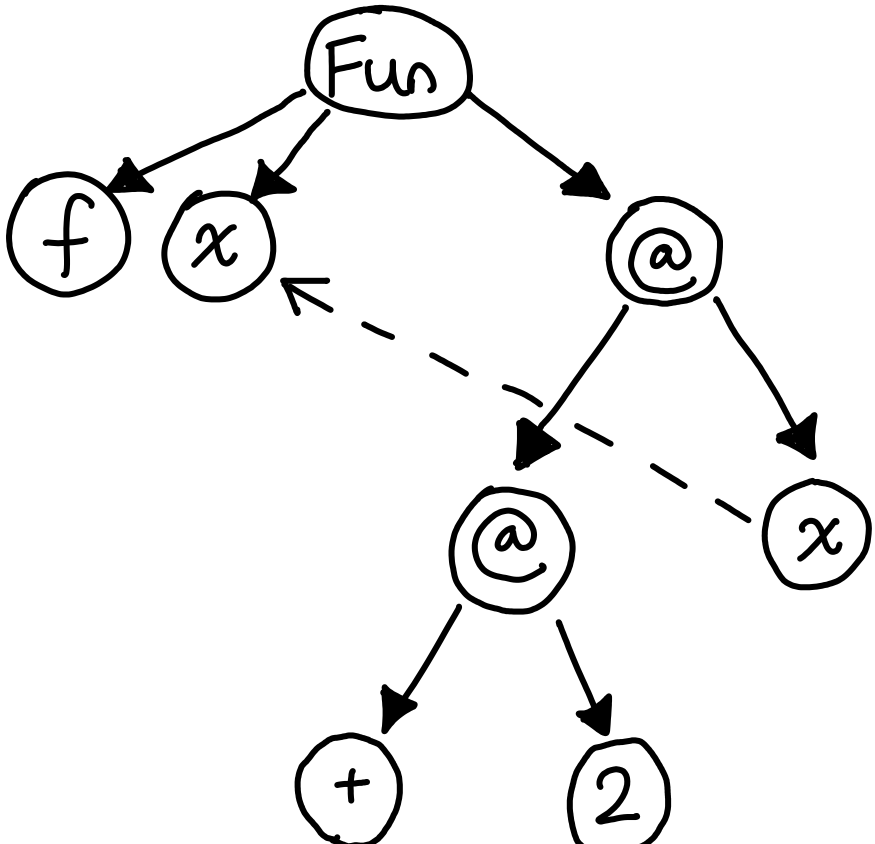
$\boxed{\text{Ex 5}}$ Type Error: Occurs Check

$$f\ x = 2 + x$$

$\boxed{\text{Ex 1}}$

1. Parsing

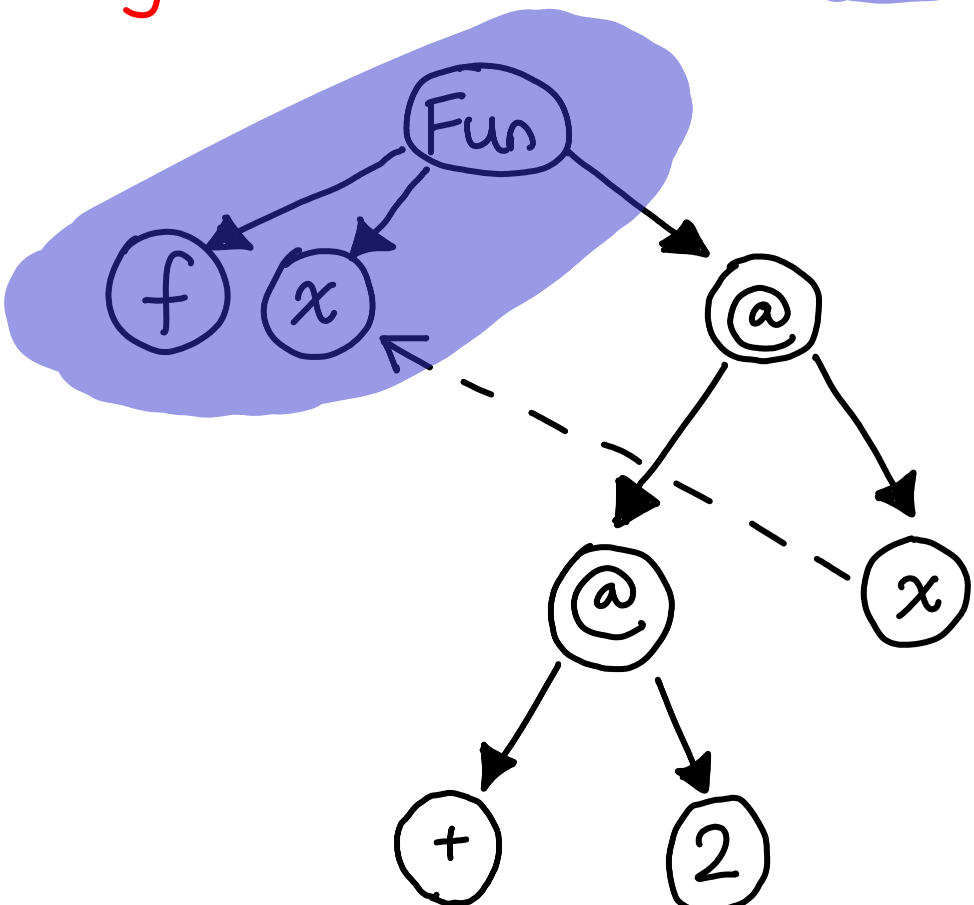
$$f\ x = 2 + x$$



Ex 1

1. Parsing

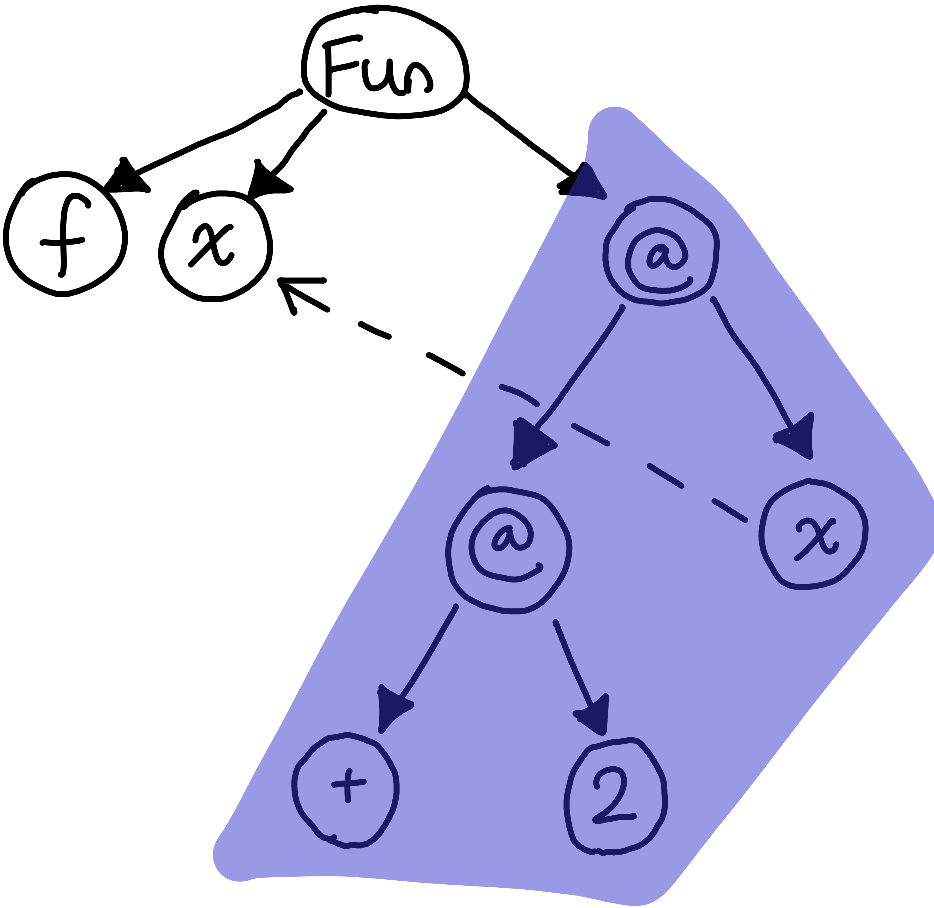
$f\ x = 2 + x$



Ex 1

1. Parsing

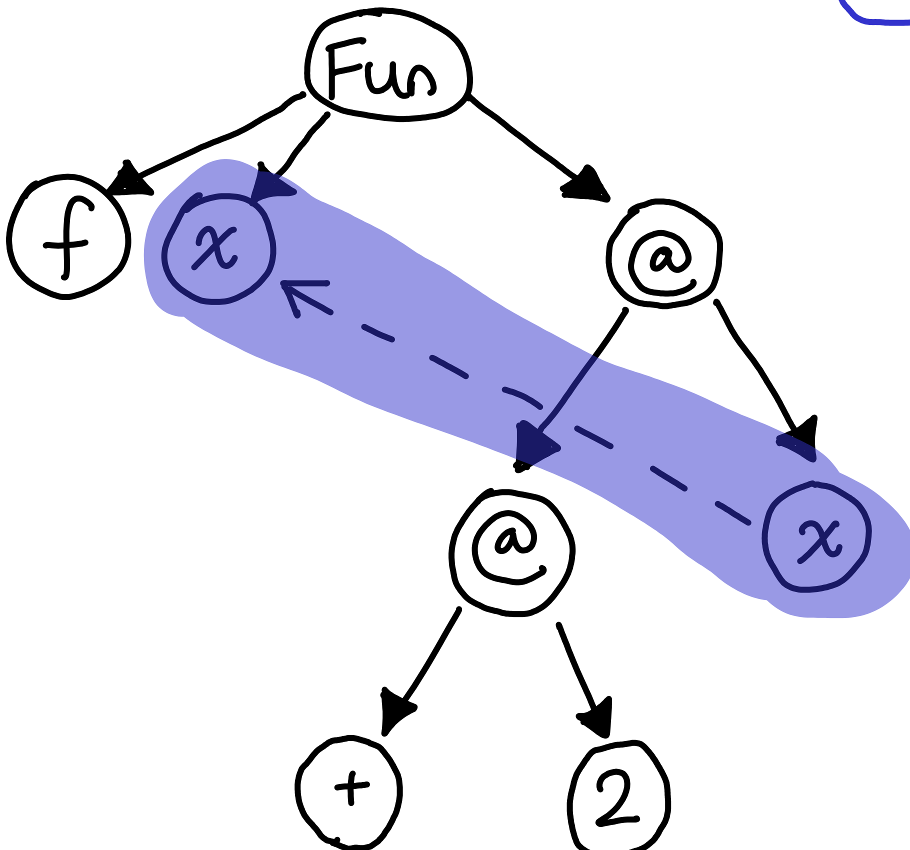
$f\ x = 2 + x$



Ex 1

1. Parsing

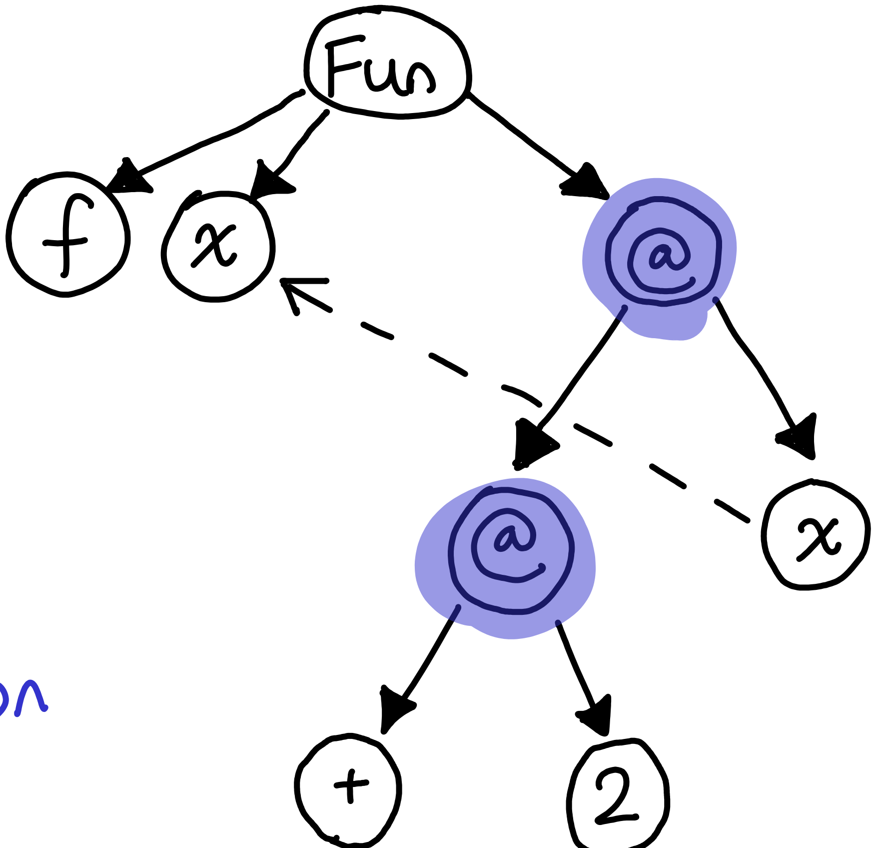
$$f\ x = 2 + x$$



Ex 1

1. Parsing

$f\ x = 2 + x$

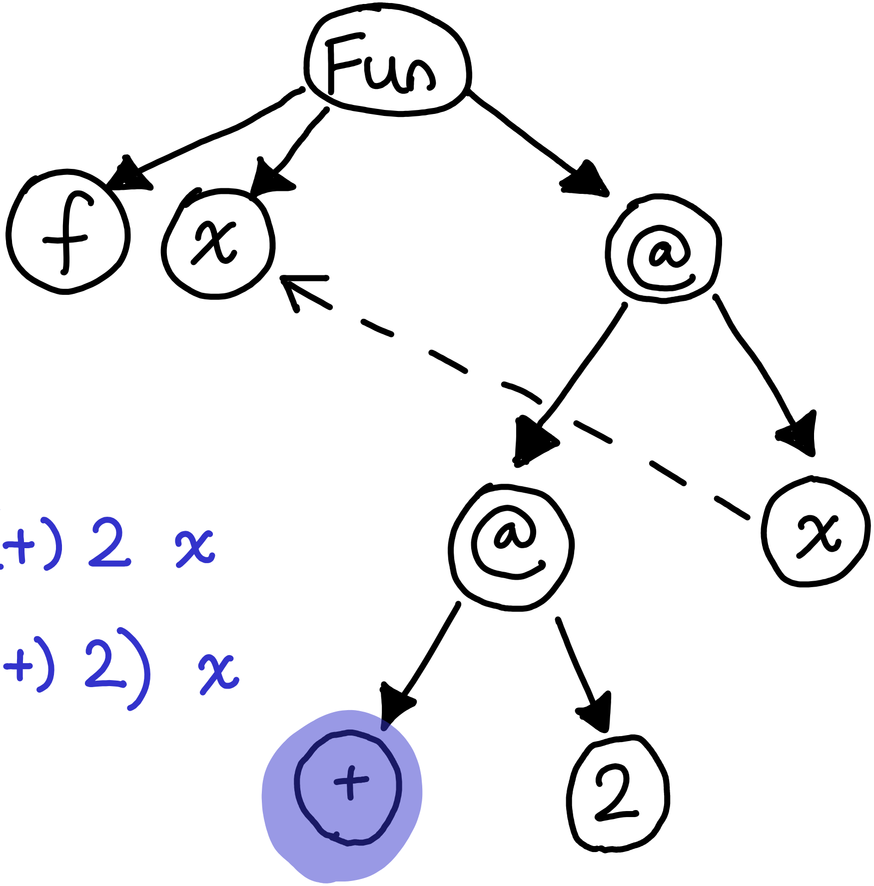


Curried Function Application

Ex 1

1. Parsing

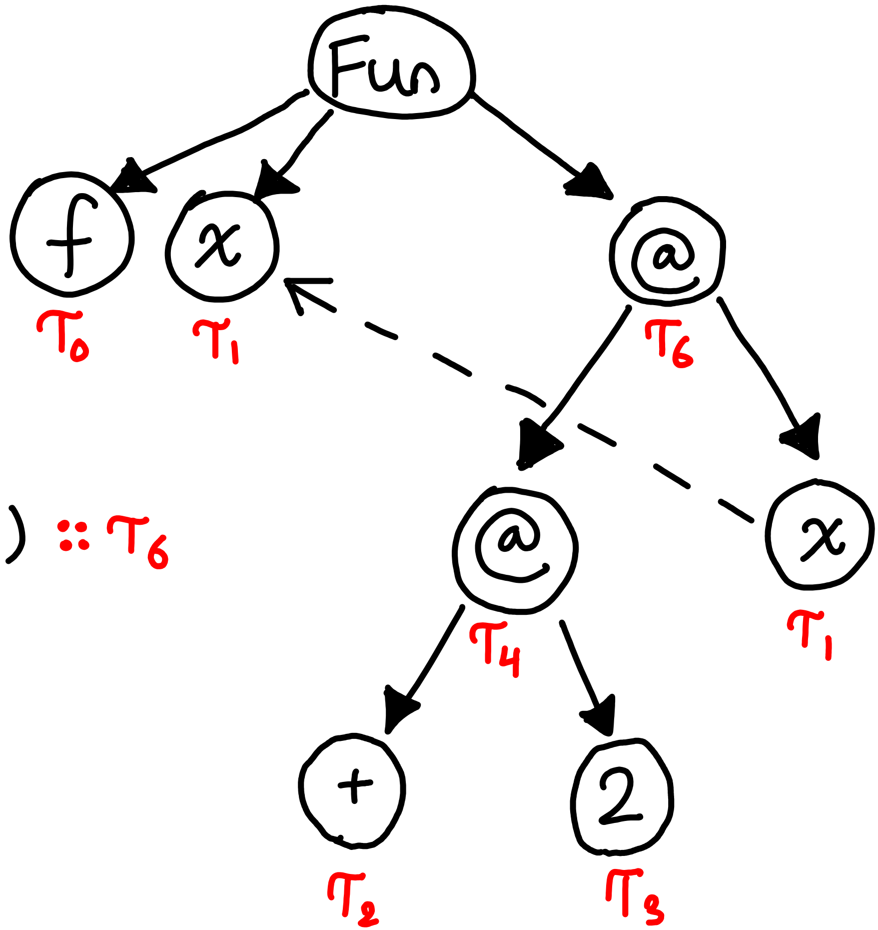
$$f\ x = 2 + x$$



$$2 + x \cong (+)\ 2\ \ x$$
$$\cong ((+)\ 2)\ \ x$$

$\boxed{\text{Ex 1}}$

2. Assign Type Variables $f\ x = 2 + x$

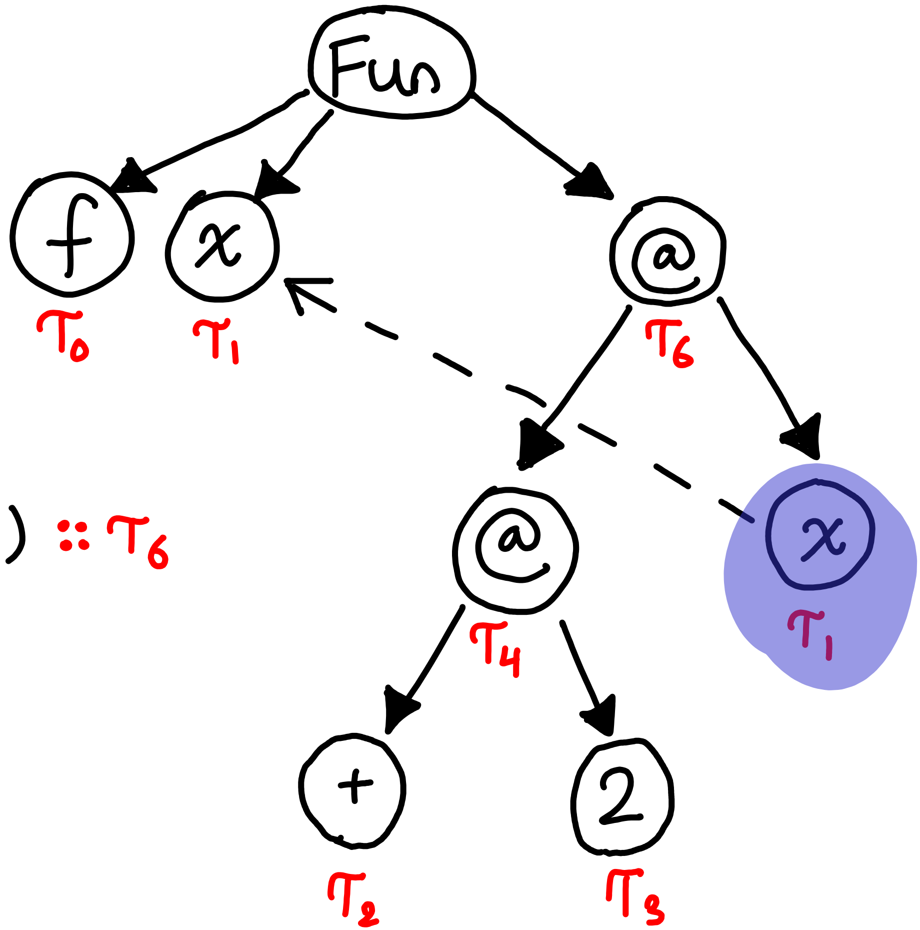


$f :: \tau_0$

$(((+)\ 2)\ x) :: \tau_6$

Ex 1

2. Assign Type Variables $f\ x = 2 + x$



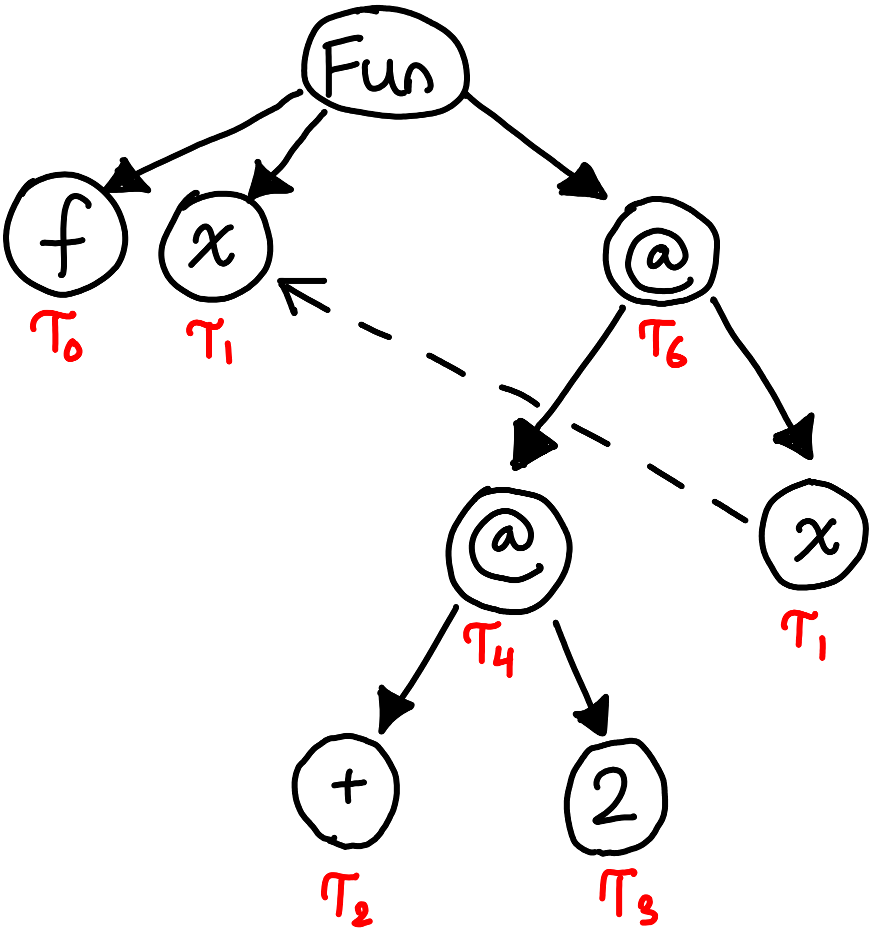
$f :: \tau_0$

$(((+)\ 2)\ x) :: \tau_6$

Ex 1

3. Add Constraints

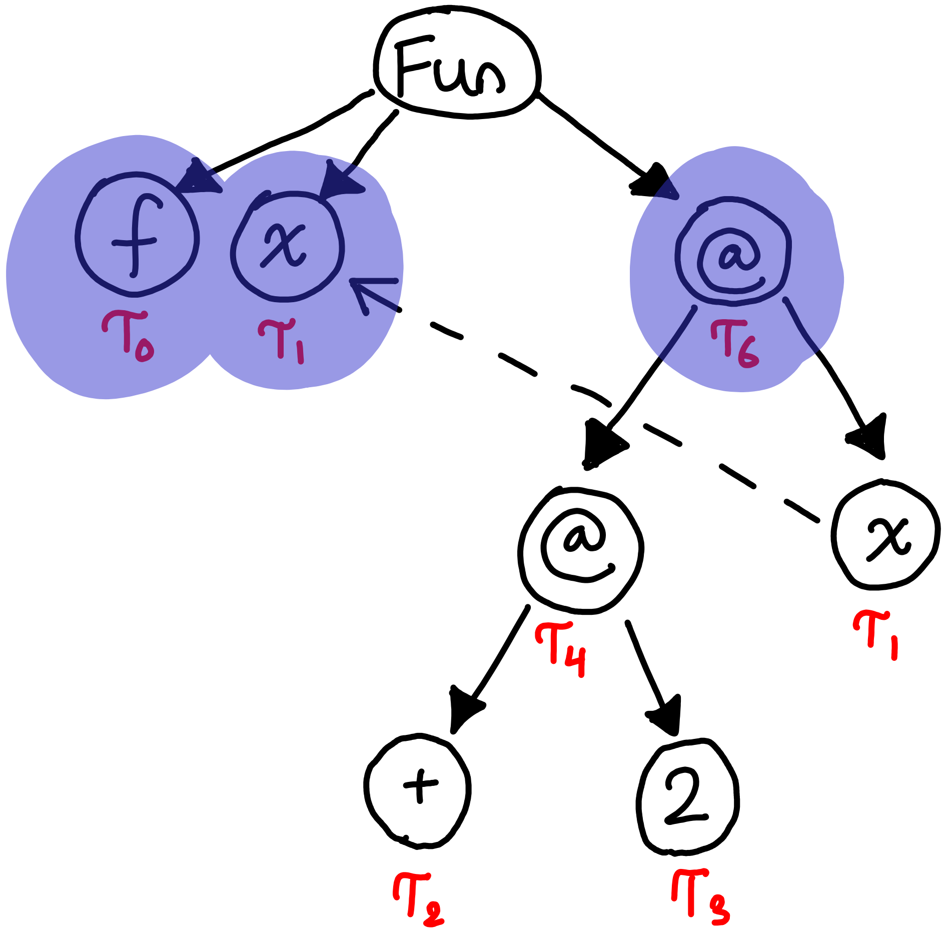
$f\ x = 2 + x$



Ex 1

3. Add Constraints

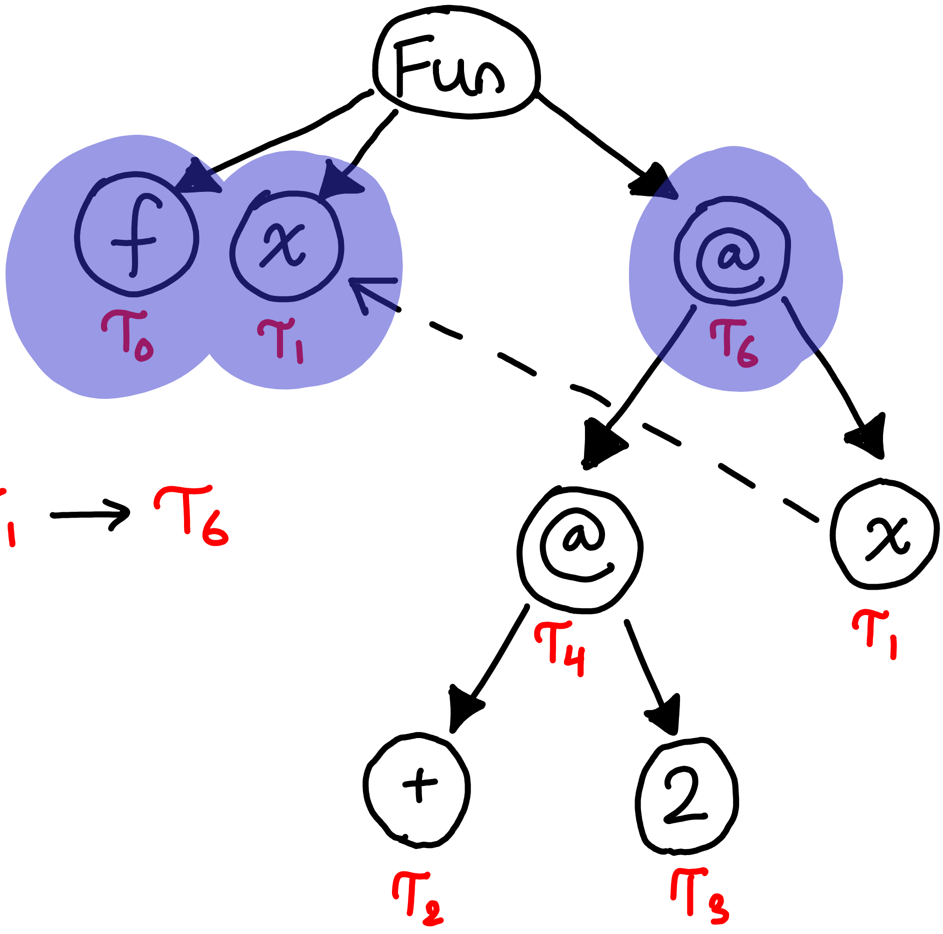
$f\ x = 2 + x$



Ex 1

3. Add Constraints

$f\ x = 2 + x$

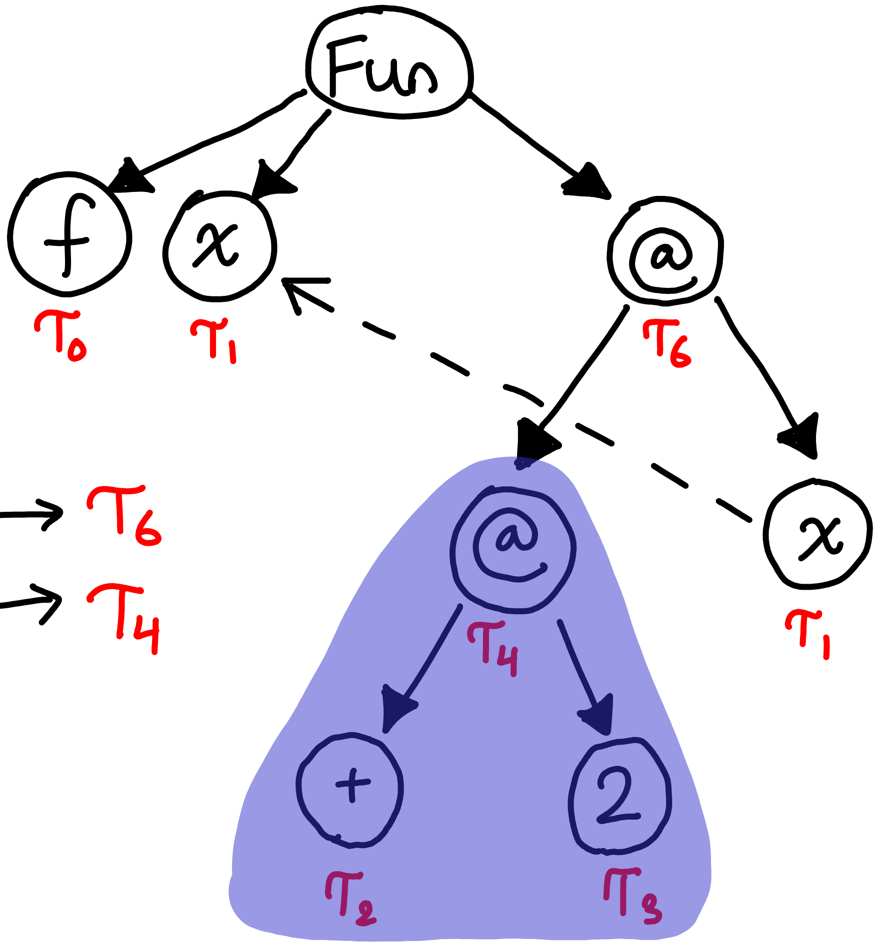


$\tau_0 = \tau_1 \rightarrow \tau_6$

Ex 1

3. Add Constraints

$f\ x = 2 + x$



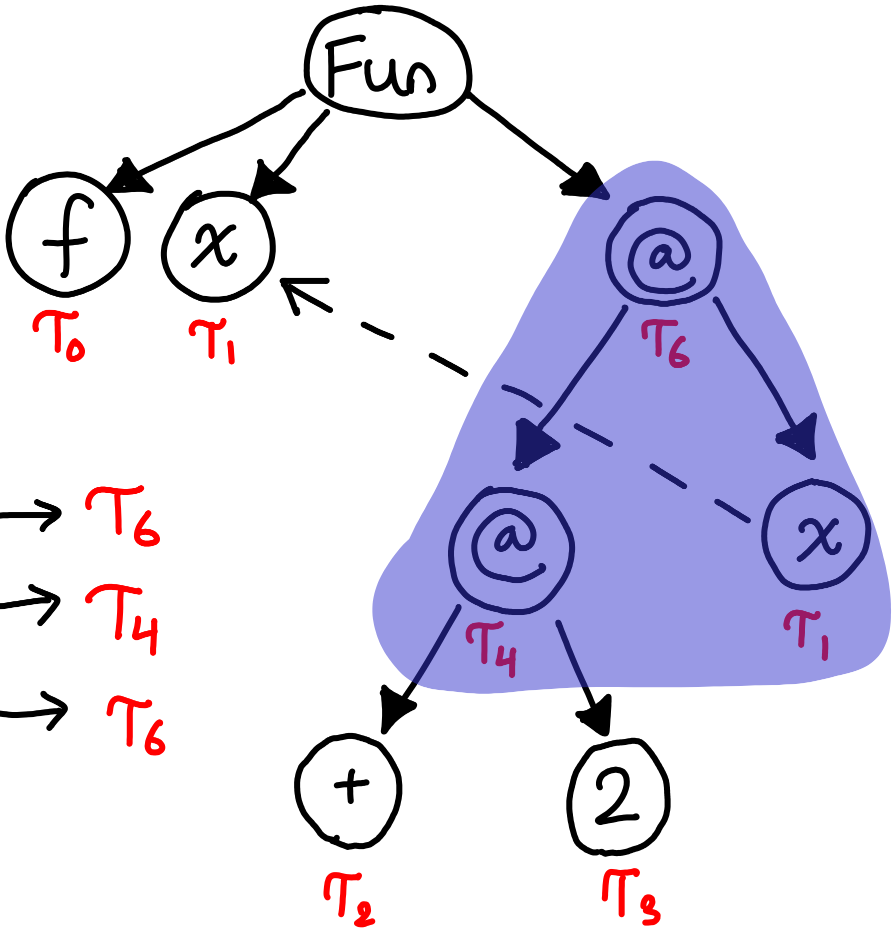
$\tau_0 = \tau_1 \to \tau_6$

$\tau_2 = \tau_3 \to \tau_4$

Ex 1

3. Add Constraints

$f\ x = 2 + x$



$\tau_0 = \tau_1 \to \tau_6$

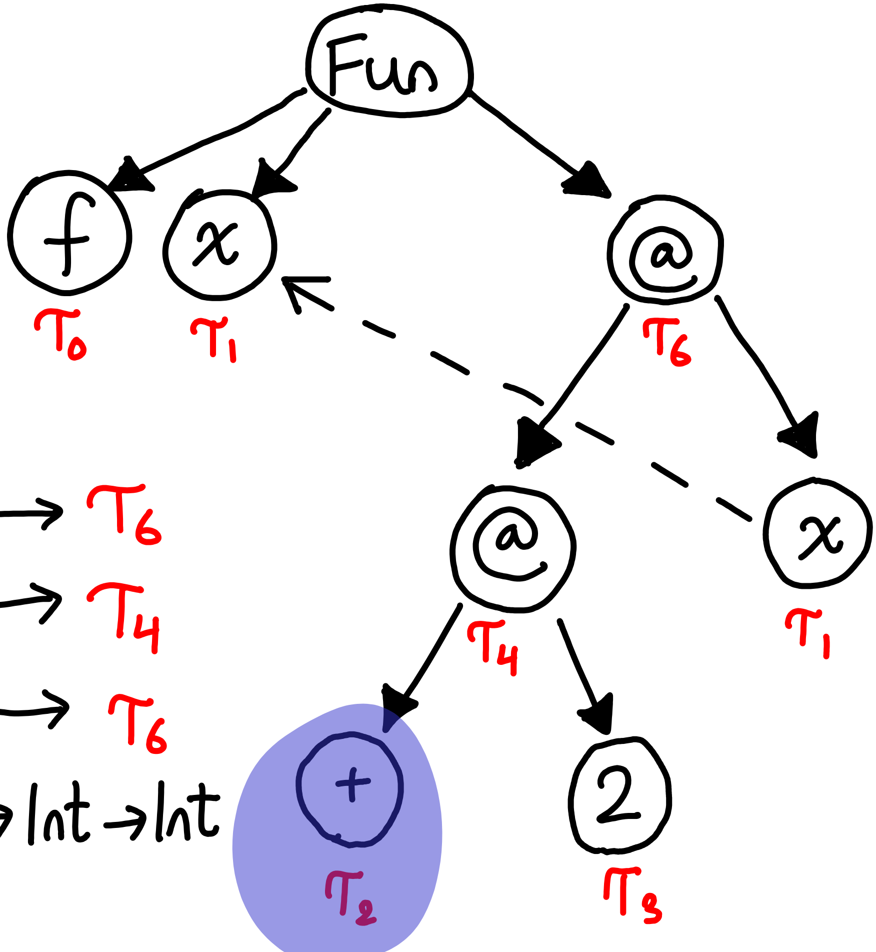
$\tau_2 = \tau_3 \to \tau_4$

$\tau_4 = \tau_1 \to \tau_6$

Ex 1

3. Add Constraints

$f\ x = 2 + x$



$\tau_0 = \tau_1 \to \tau_6$

$\tau_2 = \tau_3 \to \tau_4$

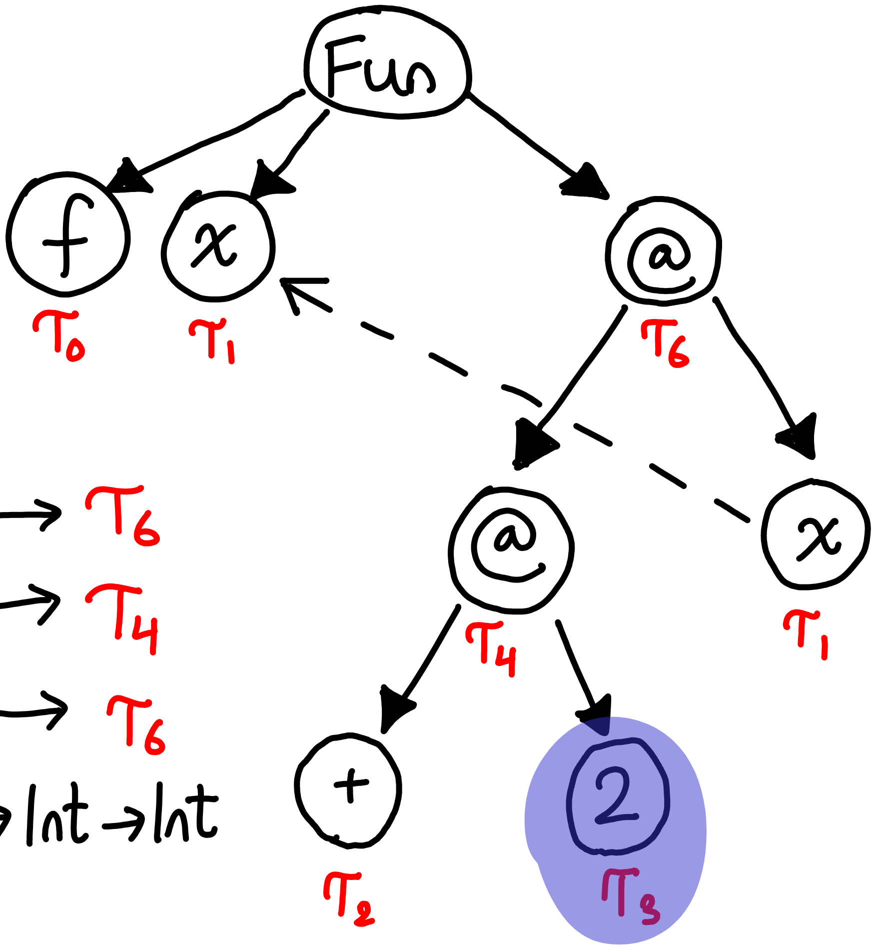
$\tau_4 = \tau_1 \to \tau_6$

$\tau_2 = \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int}$

$\boxed{\text{Ex 1}}$

3. Add Constraints

$f\ x = 2 + x$



$\tau_0 = \tau_1 \rightarrow \tau_6$

$\tau_2 = \tau_3 \rightarrow \tau_4$

$\tau_4 = \tau_1 \rightarrow \tau_6$

$\tau_2 = \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$

$\tau_3 = \text{Int}$

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$$T_0 = T_1 \to T_6$$
$$T_2 = T_3 \to T_4$$
$$T_4 = T_1 \to T_6$$
$$T_2 = Int \to Int \to Int$$
$$T_3 = Int$$

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

$$T_0 = T_1 \rightarrow T_6$$
$$T_2 = T_3 \rightarrow T_4$$
$$T_4 = T_1 \rightarrow T_6$$
$$T_2 = Int \rightarrow Int \rightarrow Int$$
$$T_3 = Int$$

process a constraint

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

"finished" constraints

$$T_0 = T_1 \to T_6$$
$$T_2 = T_3 \to T_4$$
$$T_4 = T_1 \to T_6$$
$$T_2 = Int \to Int \to Int$$
$$T_3 = Int$$

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$T_0 = T_1 \to T_6$

$T_2 = T_3 \to T_4$

$T_4 = T_1 \to T_6$

$T_3 \to T_4 = Int \to Int \to Int$

$T_3 = Int$

substitute

Ex 1

4. Solve constraints

$$f\ x = 2 + x$$

$$\tau_0 = \tau_1 \to \tau_6$$

$$\tau_2 = \tau_3 \to \tau_4$$

$$\tau_4 = \tau_1 \to \tau_6$$

$$\tau_3 \to \tau_4 = Int \to Int \to Int$$

$$\tau_3 = Int$$

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$$\tau_0 = \tau_1 \to \tau_6$$

$$\tau_2 = \tau_3 \to (\tau_1 \to \tau_6)$$

$$\tau_4 = \tau_1 \to \tau_6$$

$$\tau_3 \to (\tau_1 \to \tau_6) = \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int}$$

$$\tau_3 = \mathrm{Int}$$

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

$$T_0 = T_1 \to T_6$$

$$T_2 = T_3 \to T_1 \to T_6$$

$$T_4 = T_1 \to T_6$$

$$T_3 \to T_1 \to T_6 = Int \to Int \to Int$$

$$T_3 = Int$$

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$T_0 = T_1 \to T_6$

$T_2 = T_3 \to T_1 \to T_6$

$T_4 = T_1 \to T_6$

$T_3 \to T_1 \to T_6 = Int \to Int \to Int$

$T_3 = Int$

no variable to substitute

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$$T_0 = T_1 \rightarrow T_6$$

$$T_2 = T_3 \rightarrow T_1 \rightarrow T_6$$

$$T_4 = T_1 \rightarrow T_6$$

$$T_3 \rightarrow T_1 \rightarrow T_6 = \mathrm{Int} \rightarrow \mathrm{Int} \rightarrow \mathrm{Int}$$

$$T_3 = \mathrm{Int}$$

unification...

$\boxed{\text{Ex 1}}$

4. Solve constraints

$f\ x = 2 + x$

$$T_0 = T_1 \to T_6$$
$$T_2 = T_3 \to T_1 \to T_6$$
$$T_4 = T_1 \to T_6$$
$$T_3 = Int$$
$$T_1 \to T_6 = Int \to Int$$
$$T_3 = Int$$

... splitting an equality up!

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

$$\tau_0 = \tau_1 \to \tau_6$$

$$\tau_2 = \tau_3 \to \tau_1 \to \tau_6$$

$$\tau_4 = \tau_1 \to \tau_6$$

$$\tau_3 = \mathrm{Int}$$

$$\tau_1 \to \tau_6 = \mathrm{Int} \to \mathrm{Int}$$

$$\tau_3 = \mathrm{Int}$$

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$$T_0 = T_1 \to T_6$$

$$T_2 = Int \to T_1 \to T_6$$

$$T_4 = T_1 \to T_6$$

$$T_3 = Int$$

$$T_1 \to T_6 = Int \to Int$$

$$\cancel{Int = Int}$$

pointless constraint

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$$T_0 = T_1 \to T_6$$

$$T_2 = \mathrm{Int} \to T_1 \to T_6$$

$$T_4 = T_1 \to T_6$$

$$T_3 = \mathrm{Int}$$

$$T_1 \to T_6 = \mathrm{Int} \to \mathrm{Int}$$

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

$$\tau_0 = \tau_1 \to \tau_6$$

$$\tau_2 = Int \to \tau_1 \to \tau_6$$

$$\tau_4 = \tau_1 \to \tau_6$$

$$\tau_3 = Int$$

$$\tau_1 = Int$$

$$\tau_6 = Int$$

Ex 1

4. Solve constraints

$f\ x = 2 + x$

$\tau_0 = \tau_1 \rightarrow \tau_6$

$\tau_2 = Int \rightarrow \tau_1 \rightarrow \tau_6$

$\tau_4 = \tau_1 \rightarrow \tau_6$

$\tau_3 = Int$

$\tau_1 = Int$

$\tau_6 = Int$

Ex 1

4. Solve constraints

$$f\ x = 2 + x$$

$$T_0 = Int \to T_6$$

$$T_2 = Int \to Int \to T_6$$

$$T_4 = Int \to T_6$$

$$T_3 = Int$$

$$T_1 = Int$$

$$T_6 = Int$$

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

$$\tau_0 = Int \to \tau_6$$
$$\tau_2 = Int \to Int \to \tau_6$$
$$\tau_4 = Int \to \tau_6$$
$$\tau_3 = Int$$
$$\tau_1 = Int$$
$$\tau_6 = Int$$

$\boxed{\text{Ex 1}}$

4. Solve constraints

$$f\ x = 2 + x$$

$$T_0 = Int \to Int$$
$$T_2 = Int \to Int \to Int$$
$$T_4 = Int \to Int$$
$$T_3 = Int$$
$$T_1 = Int$$
$$T_6 = Int$$

$\boxed{\text{Ex 1}}$

5. Read out type

$f\ x = 2 + x$

$$T_0 = Int \to Int$$
$$T_1 = Int$$
$$T_2 = Int \to Int \to Int$$
$$T_3 = Int$$
$$T_4 = Int \to Int$$
$$T_6 = Int$$

$$f :: T_0$$
$$\Downarrow$$
$$f :: Int \to Int$$

$\boxed{\text{Ex 1}}$

Hindley-Milner type inference

(1. Parse the program)

2. Assign type variables to all nodes

3. Generate constraints

4. Solve constraints (via Unification)

(5. Read out top-level types)

Hindley-Milner type inference

(1. Parse the program)

2. Assign type variables to all nodes

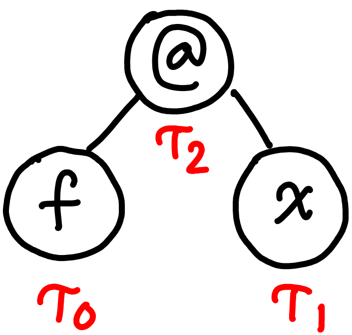
3. Generate constraints

4. Solve constraints (via Unification)

(5. Read out top-level types)

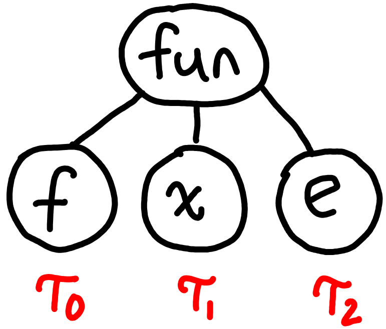
Generating constraints

application



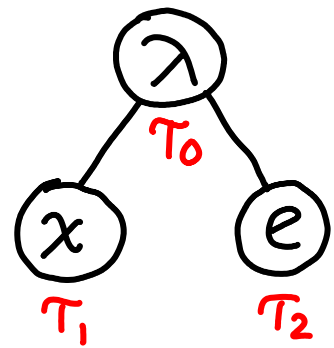
$$\tau_0 = \tau_1 \rightarrow \tau_2$$

f^{n} declaration



$$\tau_0 = \tau_1 \rightarrow \tau_2$$

lambda



$$\tau_0 = \tau_1 \rightarrow \tau_2$$

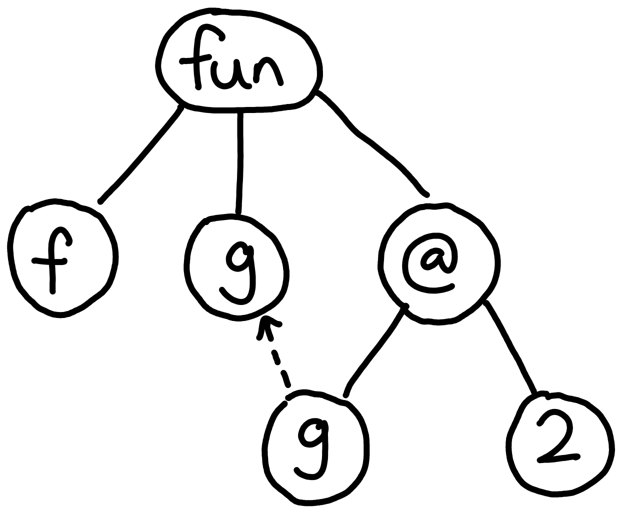
Polymorphism

$$f\ g = g\ 2$$

$\boxed{\text{Ex 2}}$

Polymorphism

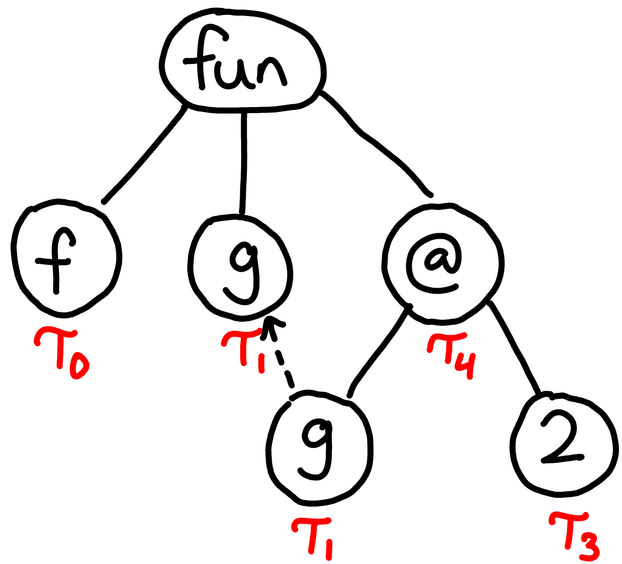
$f\ g = g\ 2$



Ex 2

Polymorphism

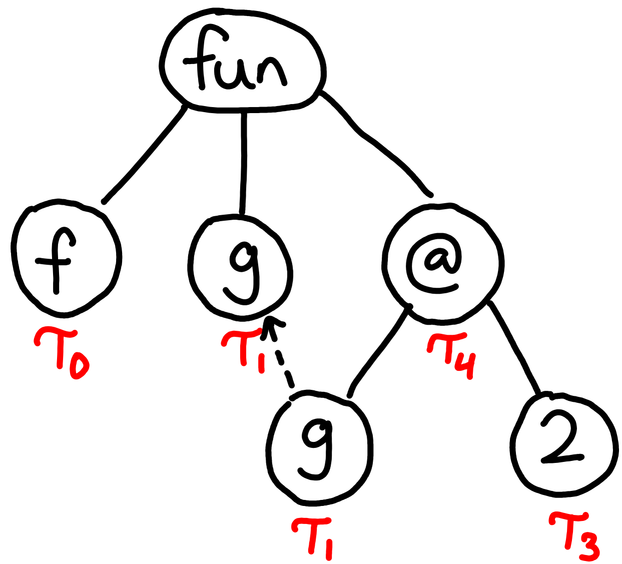
$f\ g = g\ 2$



Ex 2

Polymorphism

$f\ g = g\ 2$



$\tau_0 = \tau_1 \to \tau_4$

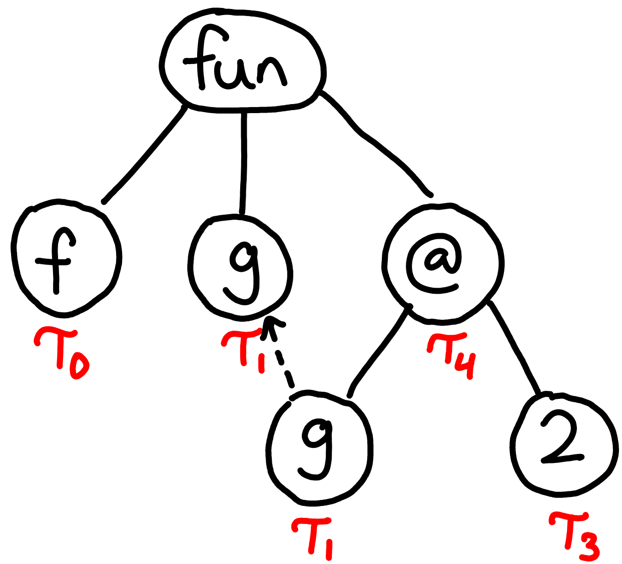
$\tau_1 = \tau_3 \to \tau_4$

$\tau_3 = \mathrm{Int}$

Ex 2

Polymorphism

$f\ g = g\ 2$



$\tau_0 = \tau_1 \to \tau_4$

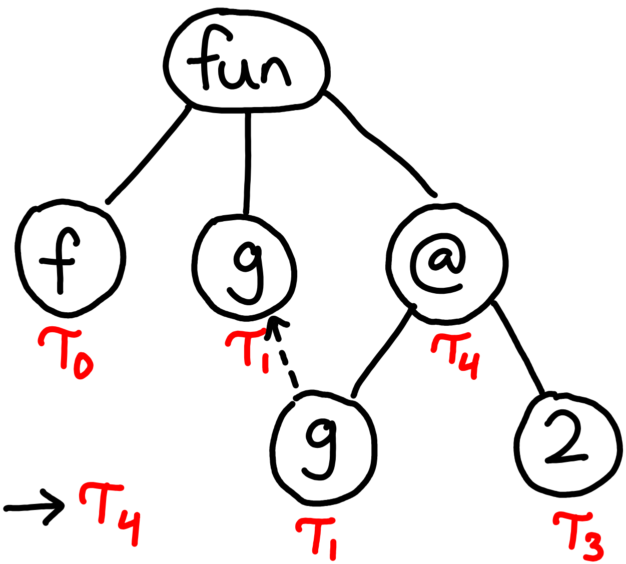
$\tau_1 = Int \to \tau_4$

$\tau_3 = Int$

Ex 2

Polymorphism

$f\ g = g\ 2$



$\tau_0 = (\text{Int} \to \tau_4) \to \tau_4$

$\tau_1 = \text{Int} \to \tau_4$

$\tau_3 = \text{Int}$

Ex 2

Polymorphism

$$f\ g = g\ 2$$

$$f :: (Int \to \tau_4) \to \tau_4$$

polymorphic types

$$\tau_0 = (Int \to \tau_4) \to \tau_4$$

$$\tau_1 = Int \to \tau_4$$

$$\tau_3 = Int$$

$\boxed{\text{Ex 2}}$

Polymorphism

$f\ g = g\ 2$

$f_{Int} :: (Int \to Int) \to Int$

$f_{Int}\ (+2)$

$\boxed{\text{Ex } 2}$

Polymorphism

$$f\ g = g\ 2$$

$$f_{Int} :: (Int \to Int) \to Int \qquad f_{Int}\ (+2)$$

$$f_{Bool} :: (Int \to Bool) \to Bool \qquad f_{Bool}\ (==2)$$

$\boxed{\text{Ex 2}}$

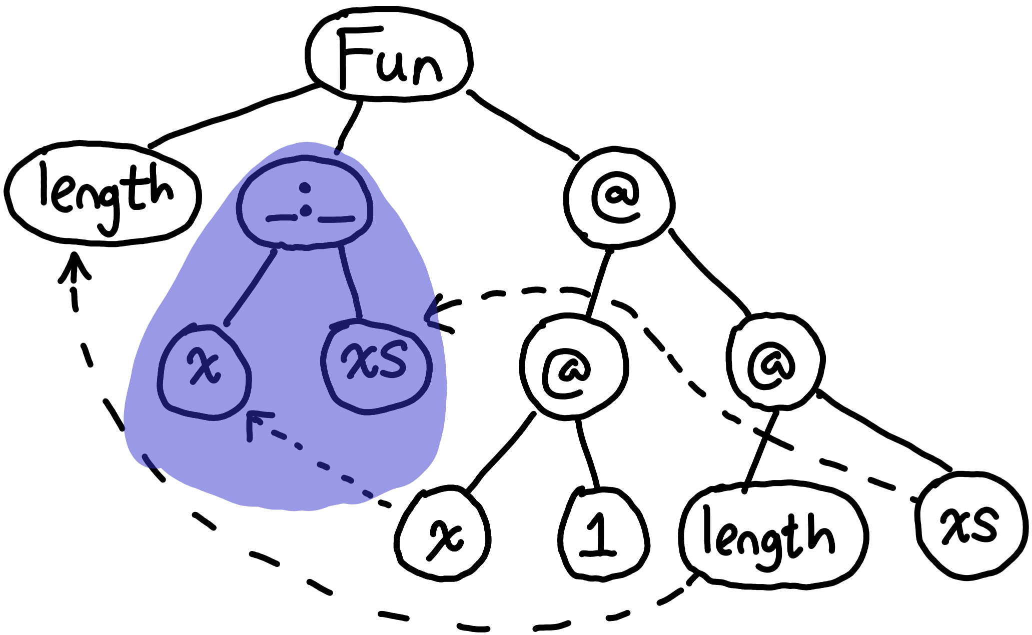
Data Types

```
length []     = 0
length (x:xs) = 1 + length xs
```

Ex 3

Data Types

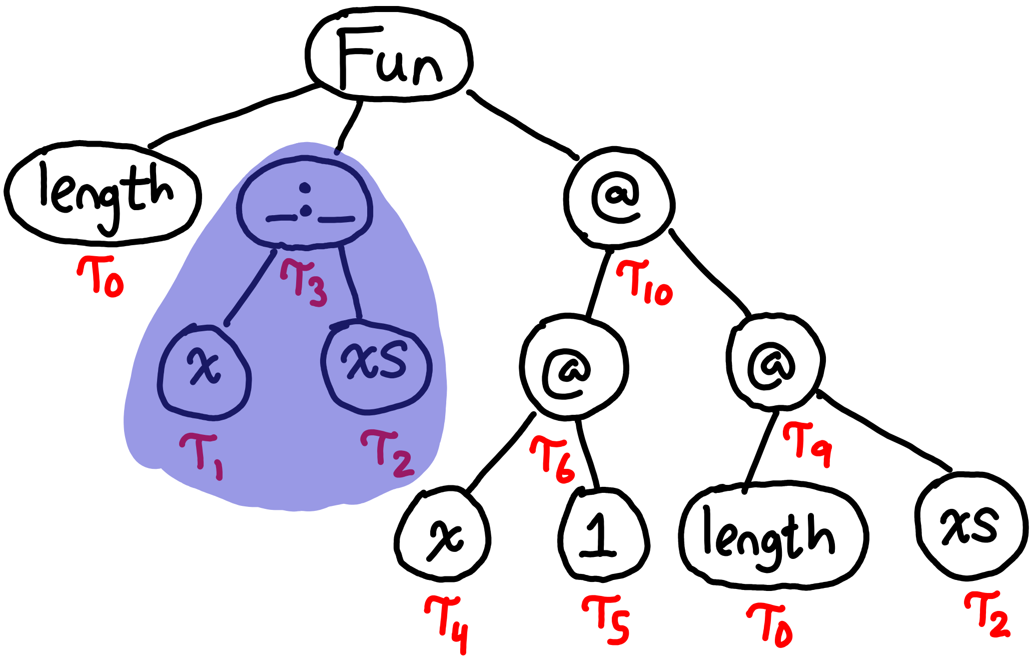
length (x:xs) = 1 + length xs



Ex 3

Data Types

length (x:xs) = 1 + length xs



Ex 3

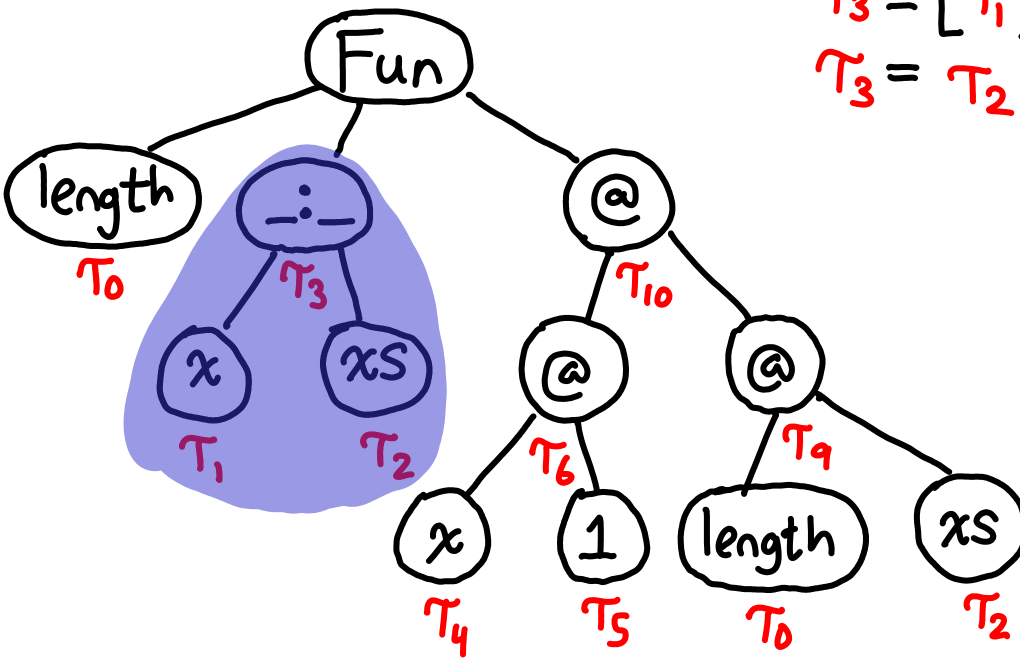
Data Types

length (x:xs) = 1 + length xs

$T_0 = T_3 \to T_{10}$

$T_3 = [T_1]$

$T_3 = T_2$



Ex 3

Data Types

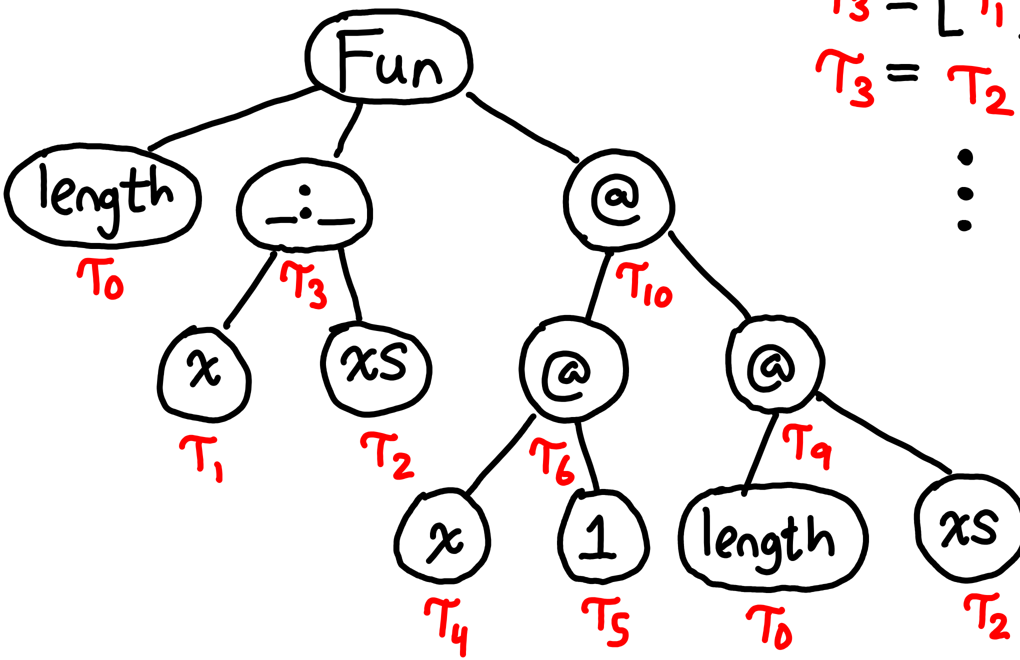
length (x:xs) = 1 + length xs

$T_0 = T_3 \to T_{10}$

$T_3 = [T_1]$

$T_3 = T_2$

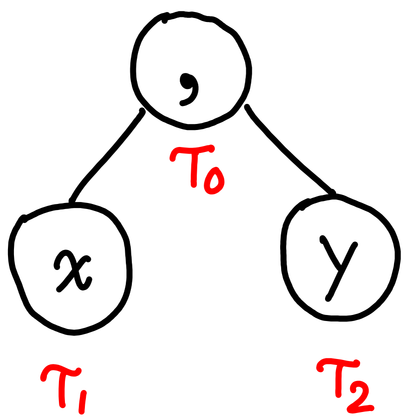
$\vdots$



length :: $[T_1] \to$ Int

$\boxed{\text{Ex 3}}$

Exercise: What are the constraints generated by products?

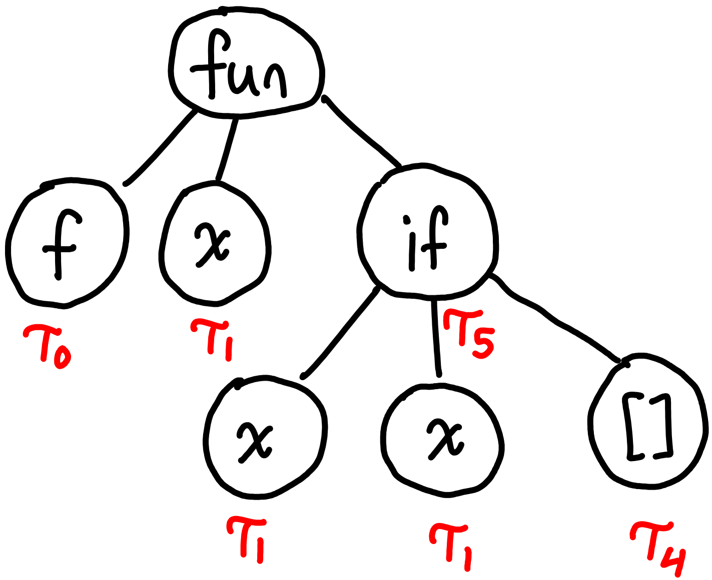


Type errors: Cannot unify □ and □

```
f x = if x then x else []
```

$\boxed{\text{Ex 4}}$

Type errors: Cannot unify □ and □



$$\tau_0 = \tau_1 \rightarrow \tau_5$$

$$\tau_5 = \tau_1$$

$$\tau_1 = \tau_4$$

$$\tau_1 = \text{Bool}$$

$$\tau_4 = [\tau_5]$$

Ex 4

Type errors: Cannot unify □ and □

$$\tau_1 = \text{Bool} \neq [\tau_5] = \tau_4$$

$\tau_0 = \tau_1 \to \tau_5$

$\tau_5 = \tau_1$

$\tau_1 = \tau_4$

$\tau_1 = \text{Bool}$

$\tau_4 = [\tau_5]$

$\boxed{\text{Ex 4}}$

Type errors: Occurs check

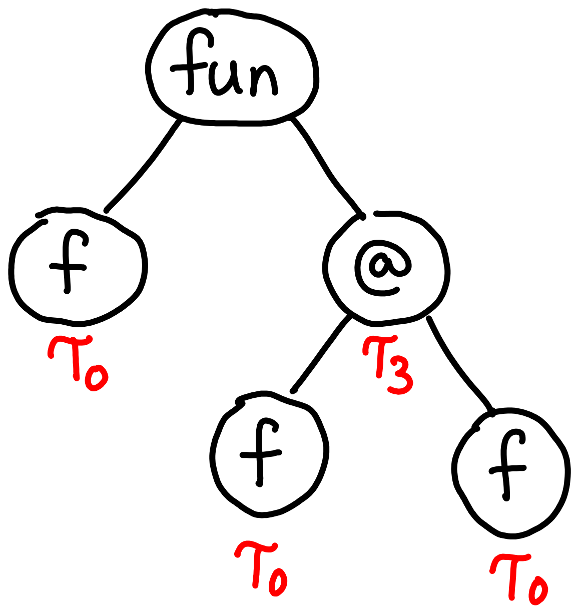
$$f = f\ f$$

remember Ω?

$\boxed{\text{Ex 5}}$

Type errors: Occurs check

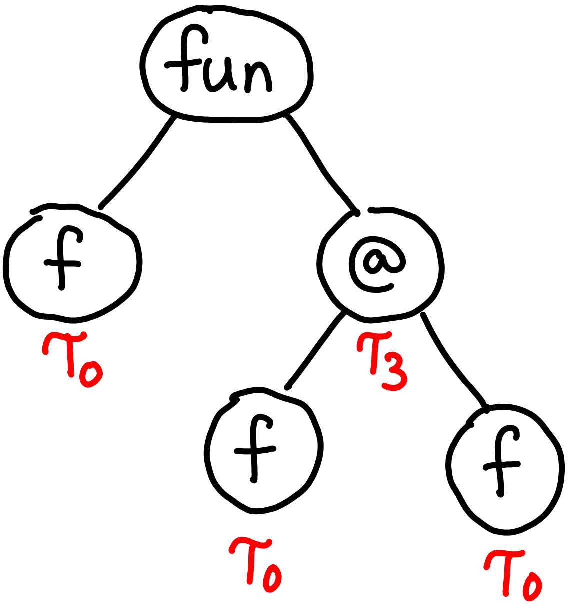
$$f = f\ f$$



Ex 5

Type errors: Occurs check

$$f = f\ f$$



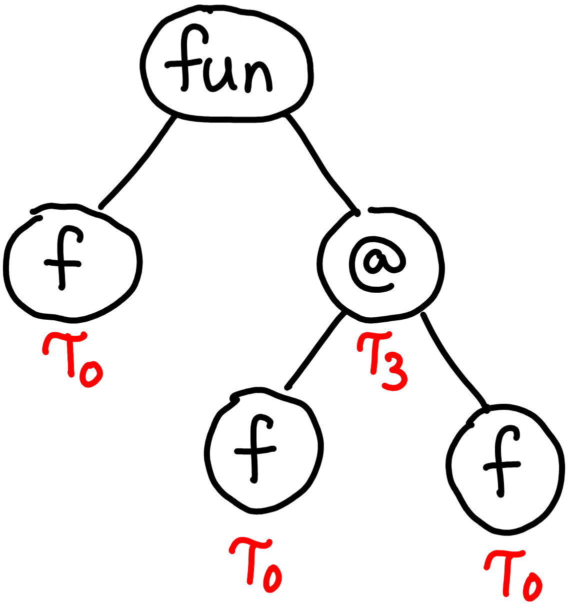
$$\tau_0 = \tau_3$$

$$\tau_0 = \tau_0 \to \tau_3$$

Ex 5

Type errors: Occurs check

$$f = f\ f$$



$$\tau_0 = \tau_3$$

$$\tau_0 = \tau_0 \rightarrow \tau_3$$

$\boxed{\text{Ex 5}}$

$$T_0 = T_0 \longrightarrow T_3$$

$$T_0 = (T_0 \longrightarrow T_3) \longrightarrow T_3$$

$$T_0 = ((T_0 \longrightarrow T_3) \longrightarrow T_3) \longrightarrow T_3$$

$$\vdots$$

Type errors: Occurs check

if e contains x and $e \neq x$

then $\text{unify}(x, e)$ fails

e.g. $\text{unify}(\tau_0, \tau_0 \to \tau_3)$ fails

Left out:

- let-bindings

$$\text{let } f\,x = x$$
$$\text{in } (f\,2,\ f\,\text{True})$$

these need distinct type variables

- the "deductive system"

$$\frac{\Gamma, x:\tau \vdash e : \tau'}{\Gamma \vdash \lambda x.\,e : \tau \to \tau'}$$

more inference rules!

Fun fact: Hindley-Milner type inference is DEXPTIME-complete

[Kanellakis, Mairson, Mitchell '89]

$$
\begin{aligned}
\text{pair}\ x\ f &= f\ x\ x \\
f_1\ x &= \text{pair}\ x \\
f_2\ x &= f_1\ (f_1\ x) \\
f_3\ x &= f_2\ (f_2\ x) \\
g\ z &= f_3\ (\lambda x.\ x)\ z
\end{aligned}
$$

Fun fact: Hindley-Milner type inference infers a **unique most general type** for all expressions (**principal typing**)

$$
\begin{tikzcd}
 & a \to b \to a & \\
\mathrm{Int} \to b \to \mathrm{Int} \arrow[ur] & & a \to \mathrm{Int} \to a \arrow[ul] \\
 & \mathrm{Int} \to \mathrm{Int} \to \mathrm{Int} \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Comparison: C++ templates

```
template <typename T>
void swap(T &x, T&y) {
    T tmp = x;
    x = y;
    y = tmp;
}
```

Comparison: C++ templates

```
void swap(Dog &x, Dog &y) {
    Dog tmp = x;
    x = y;
    y = tmp;
}
```

```
void swap(Cat &x, Cat &y) {
    Cat tmp = x;
    x = y;
    y = tmp;
}
```

Comparison: Go "type inference"

```
var int y;
x := 2 + y;
```

int

no polymorphism & annotations

Hindley-Milner Type Inference

+ No more annotations
+ Polymorphism
+ Technique generalizes
− Non-local errors
− Mutable assignment
− Implementation requires boxing
− Not what Haskell or ML uses