PSC 2023/24 (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start

12a - HOFL Syntax & Types

HOFL pre-terms
(Higher Order Functional Language)

HOFL Syntax

$n \in \mathbb{Z}$

$\mathsf{op} \in \{+, -, \times\}$ $\quad$ $\textbf{if } t = 0 \textbf{ then } t_0 \textbf{ else } t_1$

$x \in Ide$

$$
\begin{array}{rcll}
t & ::= & x \mid n \mid t_0 \ \mathsf{op}\ t_1 \mid \textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 & \text{ordinary ops} \\
& \mid & (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) & \text{pairing, projections} \\
& \mid & \lambda x.\ t \mid t_0\ t_1 & \text{abstraction, application} \\
& \mid & \textbf{rec } x.\ t & \text{recursion}
\end{array}
$$

$\mathsf{let}\ x = t\ \mathsf{in}\ x$

can contain x

Exercise

$$\mathbf{rec}\ f.\ \lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

guess the meaning of the above pre-term

factorial

Exercise

$$\textbf{rec}\ rep.\ \lambda n.\ \lambda f.\ \lambda x.\ \textbf{if}\ n\ \textbf{then}\ x$$
$$\textbf{else}\ f\ (rep\ (n-1)\ f\ x)$$

guess the meaning of the above pre-term

$$rep\ n\ f\ x = f^n\ x$$

Exercise

$$\lambda x.\ \left(\left(\begin{array}{l} \textbf{rec}\ f.\ \lambda y.\ \textbf{if}\ (x-y)\ \textbf{then}\ 0 \\ \qquad\qquad\quad \textbf{else if}\ (x+y)\ \textbf{then}\ 1 \\ \qquad\qquad\quad \textbf{else}\ f\ (y+1) \end{array} \right)\ 0 \right)$$

guess the meaning of the above pre-term

greater or equal than 0

Badge exercise

assuming $true = 0$
$false =$ any $n \neq 0$

fill the dots (in HOFL)

$$
\begin{array}{rcl}
or & \triangleq & \lambda n.\ \lambda m.\ \cdots \\
and & \triangleq & \lambda n.\ \lambda m.\ \cdots \\
not & \triangleq & \lambda m.\ \cdots \\
implies & \triangleq & \lambda n.\ \lambda m.\ \cdots \\
iff & \triangleq & \lambda n.\ \lambda m.\ \cdots
\end{array}
$$

Pre-terms

$$
\begin{array}{lcl}
t & ::= & x \;\mid\; n \;\mid\; t_0 \text{ op } t_1 \;\mid\; \textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \\
 & \mid & (t_0, t_1) \;\mid\; \textbf{fst}(t) \;\mid\; \textbf{snd}(t) \\
 & \mid & \lambda x.\ t \;\mid\; t_0\ t_1 \\
 & \mid & \textbf{rec } x.\ t
\end{array}
$$

they are called pre-terms, why?

- $x + 1$ ✓
- $\textbf{if } x \textbf{ then } x + 1 \textbf{ else } x - 1$ ✓
- $(0, \lambda x.\ x)$ ✓
- $\textbf{fst}(0, \lambda x.\ x)$ ✓
- $(\lambda x.\ x + 1)\ 3$ ✓
- $\textbf{rec } f.\ \lambda x.\ x + (f\ 0)$ ✓

we need a type system

- ✗ $1 + (0, 5)$
- ✗ $2 \times \lambda x.\ x$
- ✗ $3\ \lambda x.\ x + 1$
- ✗ $\textbf{fst}(3)$
- ✗ $\textbf{if } x \textbf{ then } \lambda x.\ x \textbf{ else } (x, x)$
- ✗ $\textbf{rec } f.\ \lambda x.\ f + x$

HOFL types

Types

$$
\begin{array}{rcl}
t & ::= & x \mid n \mid t_0 \ \mathsf{op}\ t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
& \mid & (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid & \lambda x.\ t \mid t_0\ t_1 \\
& \mid & \textbf{rec}\ x.\ t
\end{array}
$$

which types?

infinitely many combinations!

pairs

functions

int

$int * int$

$int \to int$

$int * (int \to int)$

$(int * int) \to int$

$int * (int * int)$

$(int \to int) \to int$

$(int \to int) * (int \to int)$

$(int \to int) \to (int \to (int * int))$

Types Syntax

$$\tau \ ::= \ int \mid \tau_0 * \tau_1 \mid \tau_0 \rightarrow \tau_1$$

$\mathcal{T}$ set of all types

why not lists?

for the same reason we avoid division
head and tail are not total functions

assume variables are typed

$$Ide = \{Ide_\tau\}_{\tau \in \mathcal{T}}$$

$$\widehat{\cdot} : Ide \rightarrow \mathcal{T}$$

$\widehat{x}$ denotes the type of x

Type judgements

reads “has type”

formulas: $t : \tau$

types are assigned to pre-terms
using a set of inference rules
(structural induction of HOFL syntax)

Type system

$$\frac{}{x : \widehat{x}} \qquad \frac{}{n : int} \qquad \frac{t_0 : int \quad t_1 : int}{t_0 \text{ op } t_1 : int} \qquad \frac{t : int \quad t_0 : \tau \quad t_1 : \tau}{\mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1 : \tau}$$

$$\frac{t_0 : \tau_0 \quad t_1 : \tau_1}{(t_0, t_1) : \tau_0 * \tau_1} \qquad \frac{t : \tau_0 * \tau_1}{\mathbf{fst}(t) : \tau_0} \qquad \frac{t : \tau_0 * \tau_1}{\mathbf{snd}(t) : \tau_1}$$

$$\frac{x : \tau_0 \quad t : \tau_1}{\lambda x.\ t : \tau_0 \to \tau_1} \qquad \frac{t_1 : \tau_0 \to \tau_1 \quad t_0 : \tau_0}{t_1\ t_0 : \tau_1}$$

$$\frac{x : \tau \quad t : \tau}{\mathbf{rec}\ x.\ t : \tau}$$

Well-formed terms

$$
\begin{array}{rcl}
t & ::= & x \mid n \mid t_0 \,\mathsf{op}\, t_1 \mid \mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1 \\
& \mid & (t_0, t_1) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \\
& \mid & \lambda x.\ t \mid t_0\ t_1 \\
& \mid & \mathbf{rec}\ x.\ t
\end{array}
$$

$$\tau \ ::= \ int \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

$\mathcal{T}$ set of all types

a pre-term t is *well formed* if $\exists \tau \in \mathcal{T}.\ t : \tau$

i.e. if we can assign a type to it

also called *well-typed* or *typeable*

T_τ set of all well-formed terms of type τ

Type checking

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x - 1))$$

$$\frac{}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \textbf{rec}\ f : int \to int.\ \lambda x : int.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (f\ (x-1))$$

$$\frac{f : int \to int \qquad \lambda x.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ (x \times (f(x-1))) : int \to int}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x - 1))$$

$$\dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{x : int \qquad \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int}{\lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int \to int}}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

$$\frac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{x : int \quad 1 : int \quad (x \times (f(x-1))) : int}{\mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int}}{\lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int \to int}}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

$$\dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \quad \dfrac{}{1 : int} \quad \dfrac{x : int \qquad f(x-1) : int}{(x \times (f(x-1))) : int}}{\mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int}}{\lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int \to int}}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \textbf{rec}\ f : int \to int.\ \lambda x : int.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (f\ (x-1))$$

$$\dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{}{1 : int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{f : int \to int \qquad x - 1 : int}{f(x-1) : int}}{(x \times (f(x-1))) : int}}{\textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ (x \times (f(x-1))) : int}}{\lambda x.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ (x \times (f(x-1))) : int \to int}}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

$$\dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{}{1 : int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{x : int \qquad 1 : int}{x - 1 : int}}{f(x-1) : int}}{(x \times (f(x-1))) : int}}{\mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int}}{\lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int \to int}}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

$$\dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \qquad \dfrac{\dfrac{\widehat{x} = int}{x : int} \quad \dfrac{}{1 : int} \quad \dfrac{\dfrac{\widehat{x} = int}{x : int} \quad \dfrac{\dfrac{\widehat{f} = int \to int}{f : int \to int} \quad \dfrac{\dfrac{\widehat{x} = int}{x : int} \quad \dfrac{}{1 : int}}{x - 1 : int}}{f(x-1) : int}}{(x \times (f(x-1))) : int}}{\mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int}}{\lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ (x \times (f(x-1))) : int \to int}}{fact : int \to int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int \to int} .\lambda \underbrace{x}_{int} .\mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (\ f\ \ (x-1))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int \to int} .\lambda \underbrace{x}_{int} .\mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (\ f\ (x-1))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \textbf{rec}\ f : int \to int.\ \lambda x : int.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \textbf{rec}\ \underbrace{f}_{int \to int} .\lambda \underbrace{x}_{int} .\textbf{if}\ \underbrace{x}_{int}\ \textbf{then}\ \underbrace{1}_{int}\ \textbf{else}\ x \times (\ f\ (x-1))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int\to int}\ .\lambda \underbrace{x}_{int}.\mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ \underbrace{1}_{int}\ \mathbf{else}\ \underbrace{x}_{int} \times (\ f\ \ (x-1))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int \to int} .\lambda \underbrace{x}_{int} .\mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ \underbrace{1}_{int}\ \mathbf{else}\ \underbrace{x}_{int} \times (\underbrace{f}_{int \to int} (x - 1))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int\to int}.\ \lambda \underbrace{x}_{int}.\ \mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ \underbrace{1}_{int}\ \mathbf{else}\ \underbrace{x}_{int} \times (\underbrace{f}_{int\to int}\ (\underbrace{x}_{int} - 1))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int\to int}.\lambda \underbrace{x}_{int}.\mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ \underbrace{1}_{int}\ \mathbf{else}\ \underbrace{x}_{int} \times (\underbrace{f}_{int\to int}\ (\underbrace{x}_{int} - \underbrace{1}_{int}))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int\to int}.\ \lambda\ \underbrace{x}_{int}.\ \mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ \underbrace{1}_{int}\ \mathbf{else}\ \underbrace{x}_{int} \times (\underbrace{f}_{int\to int}\ \underbrace{(\underbrace{x}_{int} - \underbrace{1}_{int})}_{int}))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \textbf{rec}\ f : int \to int.\ \lambda x : int.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (f\ (x-1))$$

more concisely

$$fact \stackrel{\text{def}}{=} \textbf{rec}\ \underbrace{f}_{int\to int}.\lambda\ \underbrace{x}_{int}.\textbf{if}\ \underbrace{x}_{int}\ \textbf{then}\ \underbrace{1}_{int}\ \textbf{else}\ \underbrace{x}_{int} \times (\underbrace{\underbrace{f}_{int\to int}\ (\underbrace{\underbrace{x}_{int} - \underbrace{1}_{int}}_{int})}_{int}))$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1))$$

more concisely

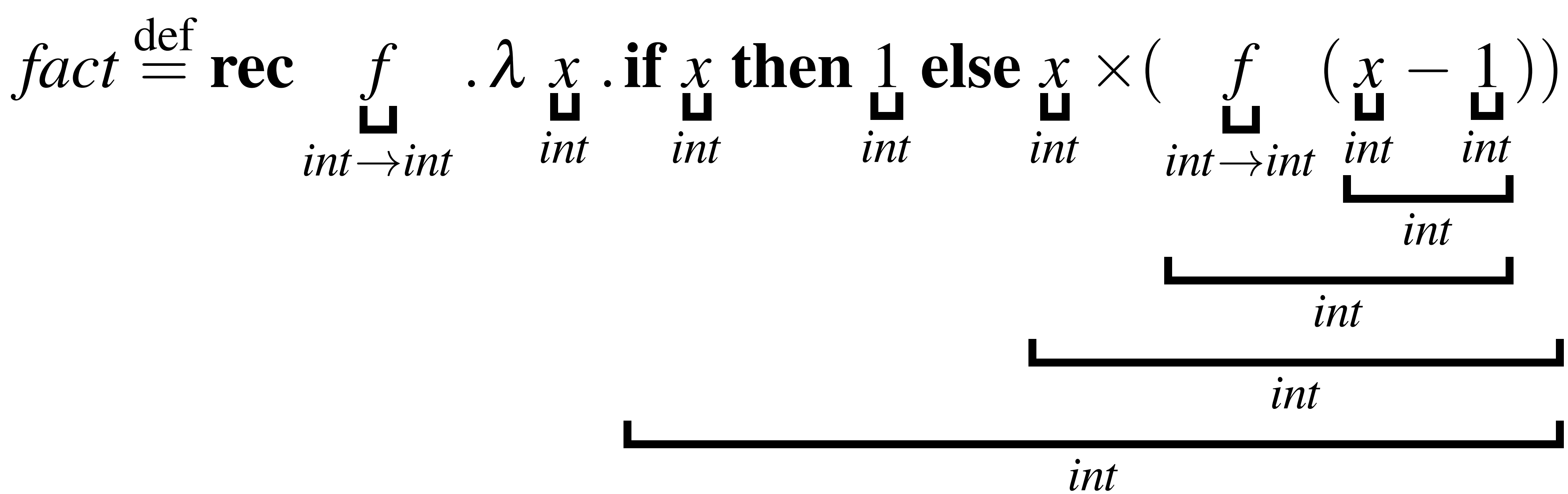
$$fact \stackrel{\text{def}}{=} \mathbf{rec}\ \underbrace{f}_{int\to int}\ .\lambda\ \underbrace{x}_{int}\ .\mathbf{if}\ \underbrace{x}_{int}\ \mathbf{then}\ \underbrace{1}_{int}\ \mathbf{else}\ \underbrace{\underbrace{x}_{int} \times (\underbrace{\underbrace{f}_{int\to int}\ (\underbrace{\underbrace{x}_{int} - \underbrace{1}_{int}}_{int})}_{int})}_{int}$$

Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \textbf{rec}\ f : int \to int.\ \lambda x : int.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (f\ (x-1))$$

more concisely

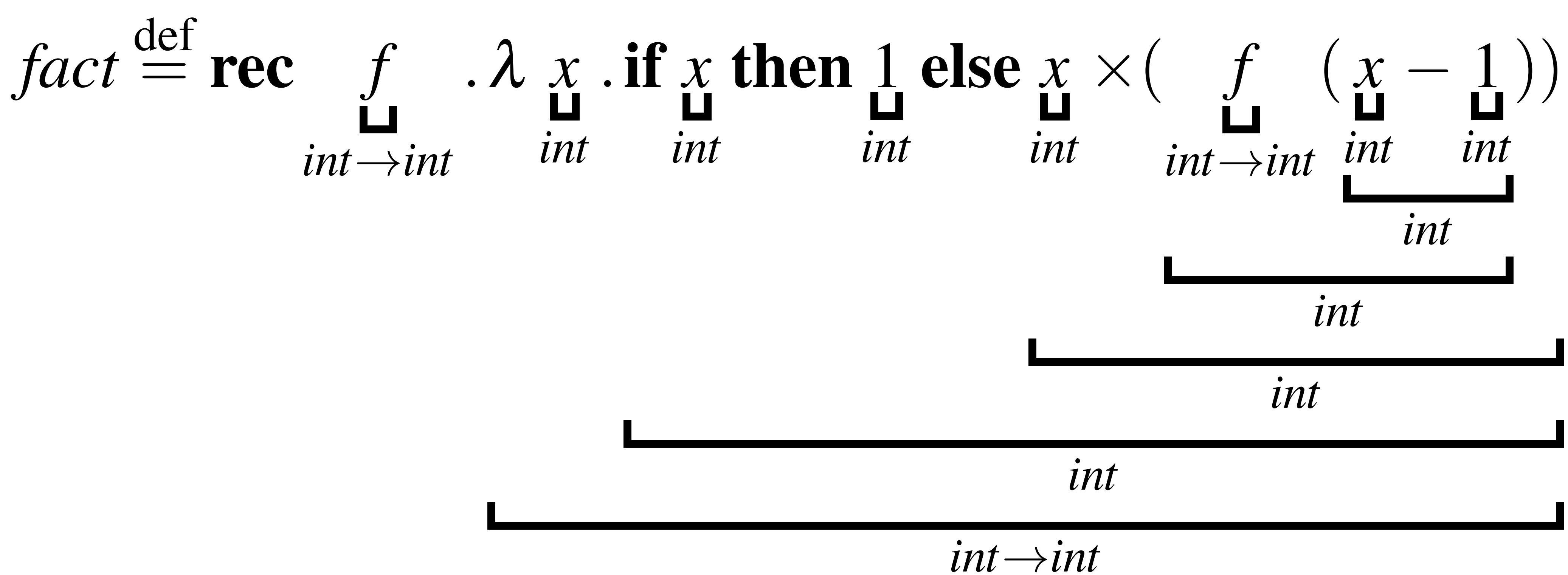


Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \textbf{rec}\ f : int \to int.\ \lambda x : int.\ \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (f\ (x-1))$$

more concisely

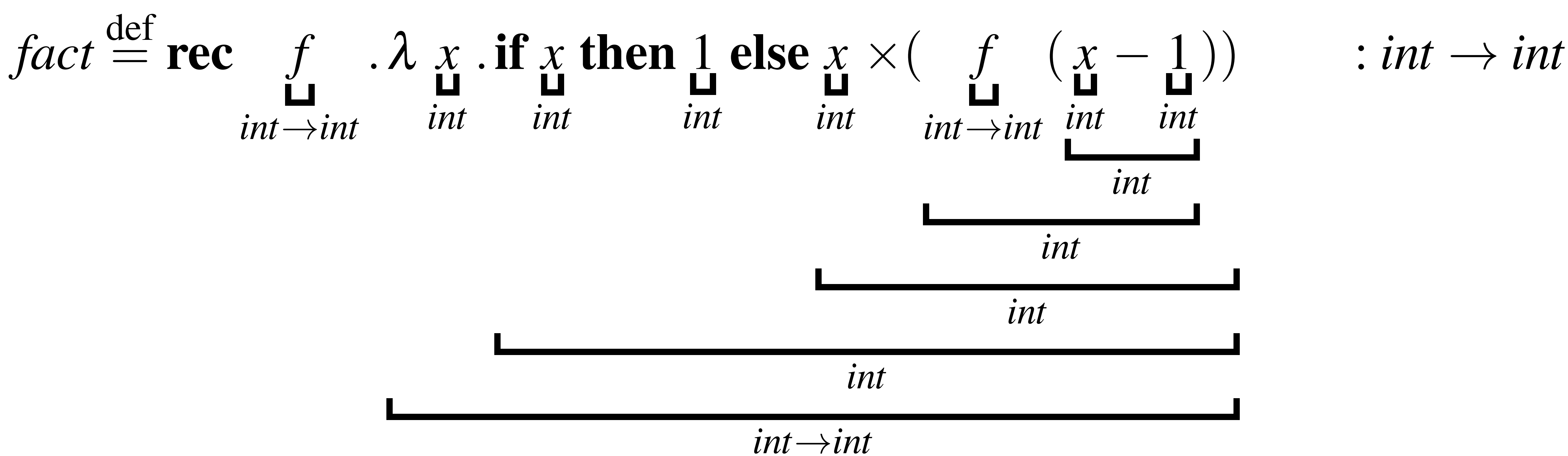


Example

variables are tagged with (declared) types
we deduce the type of terms by structural recursion

$$fact \triangleq \mathbf{rec}\ f : int \to int.\ \lambda x : int.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x - 1))$$

more concisely



Type inference

Example

types of variables are not given
type rules are used to derive type constraints (type equations)
whose solutions (via unification) define the principal type

$$t \stackrel{\text{def}}{=} \mathbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

intuitively

$$t\ 0 \equiv (0, (t\ 2)) \equiv (0, (2, (t\ 4))) \equiv \cdots \equiv (0, (2, (4, \ldots)))$$

sequence of all even numbers

we can type sequence of integers of fixed length
we have no type for sequences of any/infinite length

Example

types of variables are not given
type rules are used to derive type constraints (type equations)
whose solutions (via unification) define the principal type

$$t \stackrel{\text{def}}{=} \mathbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

Haskell

```
Prelude> let p x = (x, p (x+2))

<interactive>:…:5: error:
    • Occurs check: cannot construct the infinite type: b ~ (t, b)
      Expected type: t -> b
        Actual type: t -> (t, b)
    • Relevant bindings include
        p :: t -> b (bound at <interactive>:…:5)
```

Example

types of variables are not given
type rules are used to derive type constraints (type equations)
whose solutions (via unification) define the principal type

$$t \stackrel{\text{def}}{=} \mathbf{rec}\ p.\, \lambda x.\, (x, (p\ (x+2)))$$

Haskell

```
Prelude> let p x = x:(p (x+2))
p :: Num t => t -> [t]
Prelude> take 10 $ p 0
[0,2,4,6,8,10,12,14,16,18]
```

Example

$$t \stackrel{\text{def}}{=} \textbf{rec } p.\ \lambda x.\ (x, (p\ (x+2)))$$

$$t = \textbf{rec } p.\ \lambda x.\ (x, (p\ (x+2))) : \tau$$

$\nwarrow_{\hat{p}=\tau}\ \lambda x.\ (x, (p\ (x+2))) : \tau$

$\nwarrow_{\tau=\tau_1\to\tau_2,\ \hat{x}=\tau_1}\ (x, (p\ (x+2))) : \tau_2$

$\nwarrow_{\tau_2=\tau_3*\tau_4}\ x : \tau_3,\ (p\ (x+2)) : \tau_4$

$\nwarrow_{\hat{x}=\tau_3}\ (p\ (x+2)) : \tau_4$

$\nwarrow\ p : \tau_5 \to \tau_4,\ (x+2) : \tau_5$

$\nwarrow_{\hat{p}=\tau_5\to\tau_4}\ (x+2) : \tau_5$

$\nwarrow_{\tau_5=int}\ x : int,\ 2 : int$

$\nwarrow^{*}_{\hat{x}=int}\ \square$

$$\left.\begin{array}{l}\hat{x} = \tau_1\\ \hat{x} = \tau_3\\ \hat{x} = int\end{array}\right\}\ \tau_1 = \tau_3 = int$$

$$\left.\begin{array}{l}\hat{p} = \tau = \tau_1 \to \tau_2\\ \hat{p} = \tau_5 \to \tau_4\end{array}\right\}\ \begin{array}{l}\tau_1 = \tau_5 = int\\ \tau_2 = \tau_4\end{array}$$

$$\left.\begin{array}{l}\tau_2 = \tau_4\\ \tau_2 = \tau_3 * \tau_4\end{array}\right\}\ \text{fail! (occur check)}$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ p.\quad \lambda\ x\ .\ (\ x\ ,(\quad p\quad (\ x + \underbrace{2}_{int})))$$

Example

$$t \stackrel{\text{def}}{=} \mathbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \mathbf{rec}\ p.\quad \lambda\ x\ .\ (\ x\ , (\quad p\quad (\underbrace{x}_{int} + \underbrace{2}_{int})))$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ p.\quad \lambda \underbrace{x}_{int}.\ (\underbrace{x}_{int}, (\quad p \quad (\underbrace{x}_{int} + \underbrace{2}_{int})))$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ p.\quad \lambda \underbrace{x}_{int}.\ (\underbrace{x}_{int}, (\quad p \quad \underbrace{(\underbrace{x}_{int} + \underbrace{2}_{int})}_{int})))$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ p.\quad \lambda \underbrace{x}_{int}.\ (\underbrace{x}_{int}, (\underbrace{p}_{int \to \tau_4}\ \underbrace{(\underbrace{x}_{int} + \underbrace{2}_{int})}_{int})))$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ \underbrace{p}_{int\to\tau_4}.\quad \lambda\ \underbrace{x}_{int}\ .\ (\underbrace{x}_{int}, (\ \underbrace{p}_{int\to\tau_4}\ \ \underbrace{(\underbrace{x}_{int} + \underbrace{2}_{int})}_{int})))$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ \underbrace{p}_{int\to\tau_4}.\quad \lambda\ \underbrace{x}_{int}\ .\ (\underbrace{x}_{int}, \underbrace{(\ \underbrace{p}_{int\to\tau_4}\quad \underbrace{(\underbrace{x}_{int} + \underbrace{2}_{int})}_{int}}_{\tau_4}))$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec}\ p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec}\ \underbrace{p}_{int\to\tau_4}.\ \lambda\ \underbrace{x}_{int}.\ \underbrace{(\underbrace{x}_{int}, \underbrace{(\underbrace{p}_{int\to\tau_4}\ \underbrace{(\underbrace{x}_{int}+\underbrace{2}_{int})}_{int})}_{\tau_4})}_{int*\tau_4}$$

Example

$$t \stackrel{\text{def}}{=} \textbf{rec } p.\ \lambda x.\ (x, (p\ (x+2)))$$

more concisely

$$t = \textbf{rec } p.\ \lambda x.\ (x, (p\ (x+2)))$$

p : $int \to \tau_4$; x : int; x : int; p : $int \to \tau_4$; x : int; 2 : int

$x+2$: int

$p\ (x+2)$: τ_4

$(x, (p\ (x+2)))$: $int * \tau_4$

$$(int \to (int * \tau_4)) = (int \to \tau_4) \Rightarrow \tau_4 = (int * \tau_4)$$

fail (occur check)

Exercise

$$\textbf{rec}\ rep.\ \lambda n.\ \lambda f.\ \lambda x.\ \textbf{if}\ n\ \textbf{then}\ x$$
$$\textbf{else}\ f\ (rep\ (n-1)\ f\ x)$$

infer the type of the above term

Exercise

$$\lambda x.\ \left(\left(\begin{array}{ll} \textbf{rec}\ f.\ \lambda y. & \textbf{if}\ (x - y)\ \textbf{then}\ 0 \\ & \textbf{else if}\ (x + y)\ \textbf{then}\ 1 \\ & \textbf{else}\ f\ (y + 1) \end{array} \right)\ 0 \right)$$

infer the type of the above term

Capture-avoiding substitutions (again)

Free variables

$$\mathrm{fv}(n) \stackrel{\mathrm{def}}{=} \varnothing$$

$$\mathrm{fv}(x) \stackrel{\mathrm{def}}{=} \{x\}$$

$$\mathrm{fv}(t_0 \text{ op } t_1) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}(\mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t) \cup \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}((t_0, t_1)) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}(\mathbf{fst}(t)) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t)$$

$$\mathrm{fv}(\mathbf{snd}(t)) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t)$$

$$\mathrm{fv}(\lambda x.\ t) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t) \backslash \{x\}$$

$$\mathrm{fv}((t_0\ t_1)) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}(\mathbf{rec}\ x.\ t) \stackrel{\mathrm{def}}{=} \mathrm{fv}(t) \backslash \{x\}$$

Substitutions

$$n[{}^{t}/_{x}] = n$$

$$y[{}^{t}/_{x}] \stackrel{\text{def}}{=} \begin{cases} t & \text{if } y = x \\ y & \text{if } y \neq x \end{cases}$$

$$(t_0 \text{ op } t_1)[{}^{t}/_{x}] \stackrel{\text{def}}{=} t_0[{}^{t}/_{x}] \text{ op } t_1[{}^{t}/_{x}] \quad \text{with op} \in \{+, -, \times\}$$

$$(\textbf{if } t' \textbf{ then } t_0 \textbf{ else } t_1)[{}^{t}/_{x}] \stackrel{\text{def}}{=} \textbf{if } t'[{}^{t}/_{x}] \textbf{ then } t_0[{}^{t}/_{x}] \textbf{ else } t_1[{}^{t}/_{x}]$$

$$(t_0, t_1)[{}^{t}/_{x}] \stackrel{\text{def}}{=} (t_0[{}^{t}/_{x}], t_1[{}^{t}/_{x}])$$

$$\textbf{fst}(t')[{}^{t}/_{x}] \stackrel{\text{def}}{=} \textbf{fst}(t'[{}^{t}/_{x}])$$

$$\textbf{snd}(t')[{}^{t}/_{x}] \stackrel{\text{def}}{=} \textbf{snd}(t'[{}^{t}/_{x}])$$

$$(t_0\ t_1)[{}^{t}/_{x}] \stackrel{\text{def}}{=} (t_0[{}^{t}/_{x}]\ t_1[{}^{t}/_{x}])$$

$$(\lambda y.\ t')[{}^{t}/_{x}] \stackrel{\text{def}}{=} \lambda z.\ (\ t'[{}^{z}/_{y}][{}^{t}/_{x}]\ ) \quad \text{for } z \notin \text{fv}(\lambda y.\ t') \cup \text{fv}(t) \cup \{x\}$$

$$(\textbf{rec } y.\ t')[{}^{t}/_{x}] \stackrel{\text{def}}{=} \textbf{rec } z.\ (t'[{}^{z}/_{y}][{}^{t}/_{x}]) \quad \text{for } z \notin \text{fv}(\textbf{rec } y.\ t') \cup \text{fv}(t) \cup \{x\}$$

Types are respected

TH. $x_0 : \tau_0$ $\quad t_0 : \tau_0$ $\quad t : \tau \quad \Rightarrow \quad t[{}^{t_0}/_{x_0}] : \tau$

proof omitted
(by structural induction
of the stronger assertion $t[{}^{\widetilde{t}}/_{\widetilde{x}}] : \tau$)