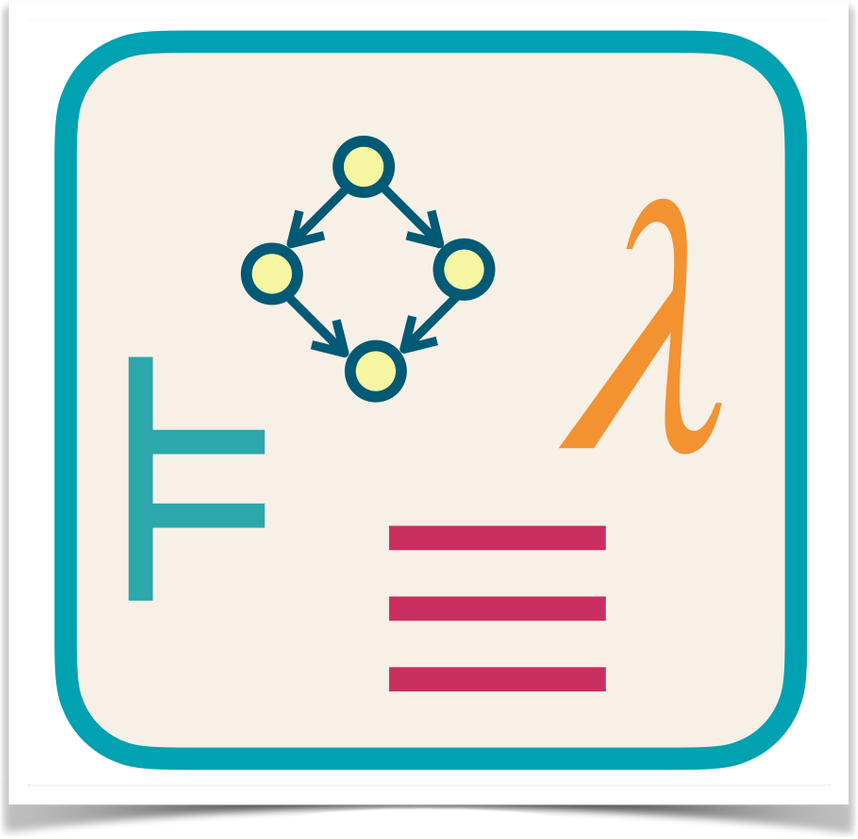


LPP 2026/27 (063AA, 9CFU)

Programming Languages

Fabio Gadducci

and Laboratory

Vincenzo Ciancia

IIa - HOFL Syntax & Types

Lambda notation, again

Bound variables

```
int f(int x) { return x^2 + 2*x + 5 }
int f(int y) { return y^2 + 2*y + 5 }
```

$f \triangleq \lambda x.\ x^2 + 2x + 5$
$f \triangleq \lambda y.\ y^2 + 2y + 5$

```
let f x = x^2 + 2*x + 5
let f y = y^2 + 2*y + 5
```

they are all the same!
local names of bound variables are not important:
it is the principle of alpha-conversion
bound names can be renamed as we like

Free variables

$x^2 + 2x + 5$

$y^2 + 2y + 5$

they are not the same!
names of global variables matter

$\lambda x.\ x^2 + 2x + 5$

$\lambda x.\ y^2 + 2y + 5$



the same enclosing context
can make a difference

$\lambda x.\ t$

we say it binds the free occurrences of x in t

$\lambda x.\ x^2 + 2z + 5$

$\lambda y.\ y^2 + 2z + 5$

$\lambda z.\ z^2 + 2z + 5$

are they all equivalent
(by alpha-conversion)?

Free variables: formally

$$\mathsf{fv}(\lambda x.\ x^2 + 2z + 5) = \{z\}$$

$$\mathsf{fv}(\lambda y.\ y^2 + 2z + 5) = \{z\}$$

$$\mathsf{fv}(\lambda z.\ z^2 + 2z + 5) = \emptyset$$

$$t ::= x \mid \lambda x.\ t \mid t\ t \mid \ldots$$

$$\mathsf{fv} : LTerms \to \wp(\, Var)$$

$$\begin{array}{rcl} \mathsf{fv}(x) & \triangleq & \{x\} \\ \mathsf{fv}(\lambda x.\ t) & \triangleq & \mathsf{fv}(t) \setminus \{x\} \\ \mathsf{fv}(t_1\ t_2) & \triangleq & \mathsf{fv}(t_1) \cup \mathsf{fv}(t_2) \end{array}$$

Alpha-conversion, again

$$\lambda x.\ t \equiv \lambda y.\ (t[{}^{y}/_{x}]) \qquad \text{if } y \notin \mathsf{fv}(\lambda x.\ t)$$

$\lambda x.\ x^2 + 2z + 5 \equiv \lambda y.\ ((x^2 + 2z + 5)[{}^{y}/_{x}]) = \lambda y.\ y^2 + 2z + 5$

$\lambda x.\ x^2 + 2z + 5 \not\equiv \lambda z.\ ((x^2 + 2z + 5)[{}^{z}/_{x}])$ because $z \in \mathsf{fv}(\lambda x.\ x^2 + 2z + 5)$

Beta rule, again

$$(\lambda x.\ t)\ e \equiv t[^{e}/_{x}]$$

we cannot just use plain syntactic substitution that would replace every free occurrence of x with e
we need to use capture-avoiding substitutions

how is (capture-avoiding) substitution defined?
and why is it called “capture-avoiding”?

Capture-avoiding substitution

Substitution, 1st try

$$y[^{e}/_{x}] \triangleq \begin{cases} e & \text{if } y = x \\ y & \text{otherwise} \end{cases}$$

$$(t_1\ t_2)[^{e}/_{x}] \triangleq t_1[^{e}/_{x}]\ (t_2[^{e}/_{x}])$$

$$(\lambda y.\ t)[^{e}/_{x}] \triangleq \begin{cases} \lambda y.\ t & \text{if } y = x \\ \lambda y.\ (t[^{e}/_{x}]) & \text{otherwise} \end{cases}$$

$t_1 \triangleq \lambda x.\ \lambda y.\ x^2 + 2y + 5$ $\qquad$ $t_2 \triangleq y$

$$t_1\ t_2 \equiv (\lambda x.\ \lambda y.\ x^2 + 2y + 5)\ y$$

free

$$\equiv (\lambda y.\ x^2 + 2y + 5)[^{y}/_{x}]$$

$$\equiv \lambda y.\ ((x^2 + 2y + 5)[^{y}/_{x}])$$

$$\equiv \lambda y.\ y^2 + 2y + 5$$

captured variable!

Capture-avoiding

free variables occurring in e
should remain free after the application of $[^e/_x]$

solution: alpha-convert bound names before substituting!

$$(\lambda y.\ x^2 + 2y + 5)[^y/_x] \equiv (\lambda z.\ (x^2 + 2y + 5)[^z/_y])[^y/_x]$$
$$\equiv (\lambda z.\ x^2 + 2z + 5)[^y/_x]$$
$$\equiv \lambda z.\ ((x^2 + 2z + 5)[^y/_x])$$
$$\equiv \lambda z.\ y^2 + 2z + 5$$

free

Substitution, 2nd try

$$y[^{e}/_{x}] \triangleq \begin{cases} e & \text{if } y = x \\ y & \text{otherwise} \end{cases}$$

$$(t_1\ t_2)[^{e}/_{x}] \triangleq t_1[^{e}/_{x}]\ (t_2[^{e}/_{x}])$$

$$(\lambda y.\ t)[^{e}/_{x}] \triangleq \begin{cases} \lambda y.\ t & \text{if } y = x \\ \lambda y.\ (t[^{e}/_{x}]) & \text{otherwise} \end{cases}$$

$$(\lambda y.\ t)[^{e}/_{x}] \triangleq \begin{cases} \lambda y.\ t & \text{if } y = x \\ \lambda z.\ (t[^{z}/_{y}][^{e}/_{x}]) & \text{otherwise, with} \\ & z \notin \mathsf{fv}(e) \cup \mathsf{fv}(\lambda y.\ t) \cup \{x\} \end{cases}$$

superfluous: no free occurrences to replace

Substitution, final

$$y[^{e}/_{x}] \triangleq \begin{cases} e & \text{if } y = x \\ y & \text{otherwise} \end{cases}$$

$$(t_1\ t_2)[^{e}/_{x}] \triangleq t_1[^{e}/_{x}]\ (t_2[^{e}/_{x}])$$

$$(\lambda y.\ t)[^{e}/_{x}] \triangleq \lambda z.\ (t[^{z}/_{y}][^{e}/_{x}]) \quad \text{with } z \notin \mathsf{fv}(e) \cup \mathsf{fv}(\lambda y.\ t) \cup \{x\}$$

HOFL pre-terms

(Higher Order Functional Language)

HOFL Syntax

$n \in \mathbb{Z}$

$x \in Ide$

$\mathsf{op} \in \{+, -, \times\}$

$\mathbf{if}\ t = 0\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1$

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

ordinary ops

pairing, projections

abstraction, application

recursion

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

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

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

guess the meaning of the above pre-term

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

Exercise

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

guess the meaning of the above pre-term

greater or equal than 0

Exercise

$$\lambda f\,.\ \mathbf{rec}\ x\,.\ f\ x$$

guess the meaning of the above pre-term

fixpoint operator

Badge exercise

assuming $true = 0$

$false =$ any $n \neq 0$

fill the dots (in HOFL)

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

Pre-terms

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

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

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

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

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

$\mathcal{T}$ 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

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

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

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

Type judgements

formulas: $t : \tau$ reads “has type”

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

Type system

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

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

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

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

Well-formed terms

$$t \quad ::= \quad x \mid n \mid t_0 \text{ op } t_1 \mid \textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \mid \lambda x.\, t \mid t_0\ t_1 \mid \textbf{rec } x.\, t$$

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

$\mathcal{T}$ 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

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

$$fact : int \to int$$

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

Example

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

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

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

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

Example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$x - 1 : int$

$f\ (x-1) : int$

$x \times (f\ (x-1)) : int$

$\mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times (f\ (x-1)) : int$

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

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

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

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

$x - 1$: 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

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

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

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

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

Example

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

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

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

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

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

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

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

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

$\tau_5=int$ $x : int,\ 2 : int$

$\overset{*}{\widehat{x}=int}$ $\square$

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

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

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

Example

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

more concisely

$$t = \mathbf{rec}\ p.\quad \lambda\ \underline{x}\ .\ (\underline{x}\ , (\quad p\quad (x + \underline{2})))$$
$$\qquad\qquad\qquad

Example

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

more concisely

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

$$\phantom{t = \mathbf{rec}\ p.\quad \lambda\ } \textit{int}\qquad \textit{int}$$

Example

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

more concisely

$$t = \mathbf{rec}\ p.\quad \lambda \underset{\sqcup}{x}.\ (\underset{\sqcup}{x}, (\quad p \quad (\underset{\sqcup}{x} + \underset{\sqcup}{2})))$$

$$\begin{array}{cccc} & & & \\ int & int & int & int \end{array}$$

Example

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

more concisely

$$t = \mathbf{rec}\ p.\quad \lambda \underset{\llcorner\lrcorner}{x}.\ (\underset{\llcorner\lrcorner}{x}, (\quad p \quad (\underset{\llcorner\lrcorner}{x} + \underset{\llcorner\lrcorner}{2})))$$

$$\begin{array}{llll} int & int & int & int \\ & & \multicolumn{2}{c}{\llcorner\!\!\!\!\!\!\!\!\!\!\!\!\lrcorner} \\ & & \multicolumn{2}{c}{int} \end{array}$$

Example

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

more concisely

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

Example

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

more concisely

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

Example

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

more concisely

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

Example

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

more concisely

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

Example

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

more concisely

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

$$int \qquad int \qquad int \qquad int \qquad int$$

$$int$$

$$\tau_4$$

$$int * \tau_4$$

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

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

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

infer the type of the above term

Exercise

$$\lambda f\,.\ \mathbf{rec}\ x\,.\ f\ x$$

infer the type of the above term

Exercise

$$\lambda g\,.\,\lambda f\,.\,\lambda x\,.\,g(f\ x)$$

infer the type of the above term

Exercise

$$\lambda f\,.\;\lambda x\,.\;\lambda y\,.\;f(x,y)$$

infer the type of the above term

Exercise

$$\textbf{rec}\ f\,.\ \lambda x\,.\ \textbf{if}\ x\ \textbf{then}\ f\ x\ \textbf{else}\ x$$

infer the type of the above term

Capture-avoiding substitutions (again)

Free variables

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

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

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

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

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

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

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

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

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

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

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. $\begin{array}{l} x_0 : \tau_0 \\ t_0 : \tau_0 \end{array} \quad t : \tau \quad \Rightarrow \quad t[^{t_0}/_{x_0}] : \tau$

proof details omitted:
by structural induction of the stronger assertion below

$$P(t : \tau) = \forall \widetilde{x}, \widetilde{t}, \widetilde{\tau} \,.\, (\widetilde{x} : \widetilde{\tau} \ \wedge \ \widetilde{t} : \widetilde{\tau}) \Rightarrow t[^{\widetilde{t}}/_{\widetilde{x}}] : \tau$$

base case: $t = n$

therefore $\tau = int$

we have $n[^{\widetilde{t}}/_{\widetilde{x}}] = n$ and thus obviously $n : int$

Types are respected

TH. $\begin{matrix} x_0 : \tau_0 \\ t_0 : \tau_0 \end{matrix} \qquad t : \tau \qquad \Rightarrow \qquad t[^{t_0}/_{x_0}] : \tau$

proof details omitted:

by structural induction of the stronger assertion below

$$P(t : \tau) = \forall \widetilde{x}, \widetilde{t}, \widetilde{\tau} \,.\, (\widetilde{x} : \widetilde{\tau} \ \wedge \ \widetilde{t} : \widetilde{\tau}) \Rightarrow t[^{\widetilde{t}}/_{\widetilde{x}}] : \tau$$

inductive case: $t = (t_1, t_2)$

therefore $\tau = \tau_1 * \tau_2$ where $t_1 : \tau_1$ and $t_2 : \tau_2$

we have $t[^{\widetilde{t}}/_{\widetilde{x}}] = (t_1[^{\widetilde{t}}/_{\widetilde{x}}], t_2[^{\widetilde{t}}/_{\widetilde{x}}])$

by inductive hypothesis $t_1[^{\widetilde{t}}/_{\widetilde{x}}] : \tau_1$ and $t_2[^{\widetilde{t}}/_{\widetilde{x}}] : \tau_2$

therefore $(t_1[^{\widetilde{t}}/_{\widetilde{x}}], t_2[^{\widetilde{t}}/_{\widetilde{x}}]) : \tau_1 * \tau_2 = \tau$

Types are respected

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

proof details omitted:

by structural induction of the stronger assertion below

$$P(t : \tau) = \forall \widetilde{x}, \widetilde{t}, \widetilde{\tau} \,.\, (\widetilde{x} : \widetilde{\tau} \ \wedge \ \widetilde{t} : \widetilde{\tau}) \Rightarrow t[^{\widetilde{t}}/_{\widetilde{x}}] : \tau$$

inductive case: $t = \lambda y \,.\, t'$

therefore $\tau = \tau_y \to \tau'$ where $t' : \tau'$

we have $t[^{\widetilde{t}}/_{\widetilde{x}}] = \lambda z \,.\, (t'[^{z,\widetilde{t}}/_{y,\widetilde{x}}])$

by inductive hypothesis $t'[^{z,\widetilde{t}}/_{y,\widetilde{x}}] : \tau'$

therefore $\lambda z \,.\, (t'[^{z,\widetilde{t}}/_{y,\widetilde{x}}]) : \tau_y \to \tau' = \tau$

Exercise

Check if the HOFL pre-term is typeable

$\lambda x\,.\,\lambda y\,.\,\lambda z\,.$ if z then $(x\ y)$ else $(y\ x)$

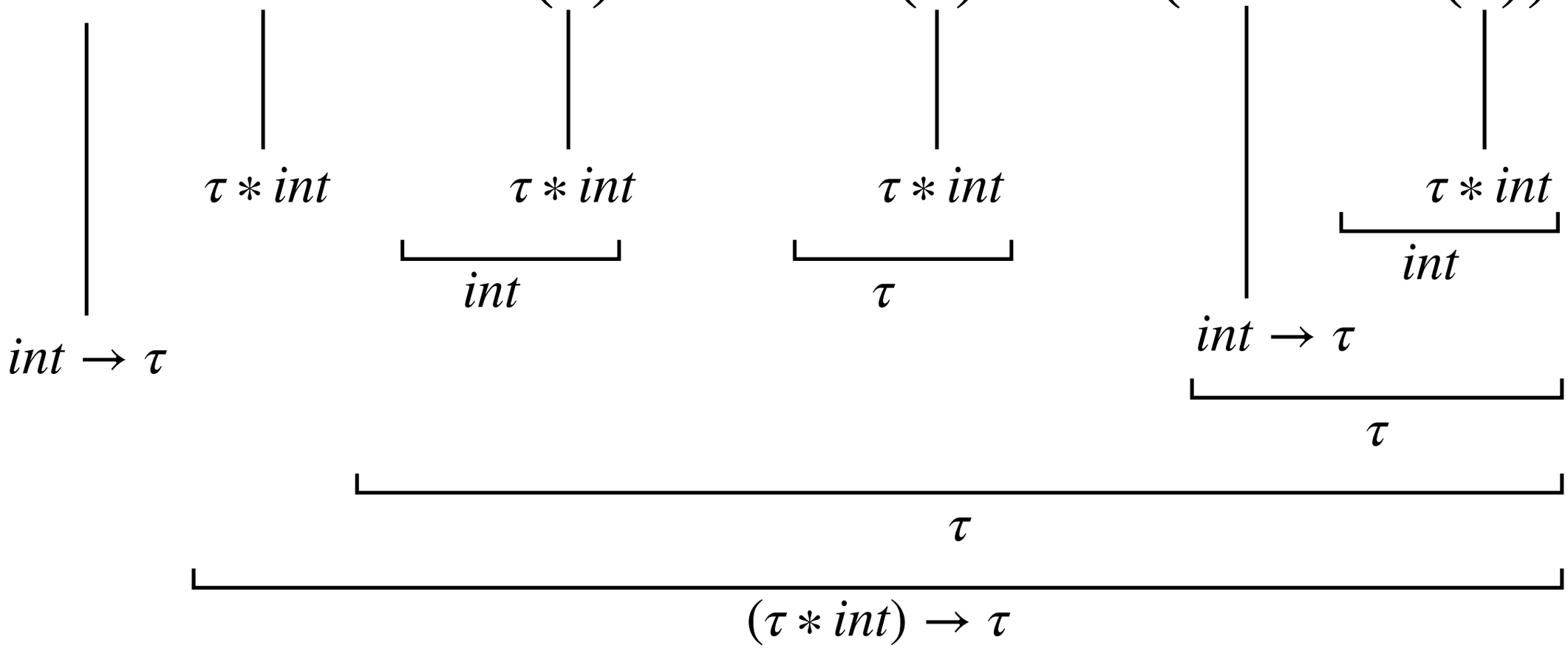
int int τ_1 $\tau_1 \to \tau_2$ $\tau_1 \to \tau_2$

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

Exercise

Check if the HOFL pre-term is typeable

rec $last$. λx . if snd(x) then fst(x) else ($last$ snd(x))



$int \rightarrow \tau = (\tau * int) \rightarrow \tau$

$int = (\tau * int)$