



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

**PSC 2020/21** (375AA, 9CFU)

Principles for Software Composition

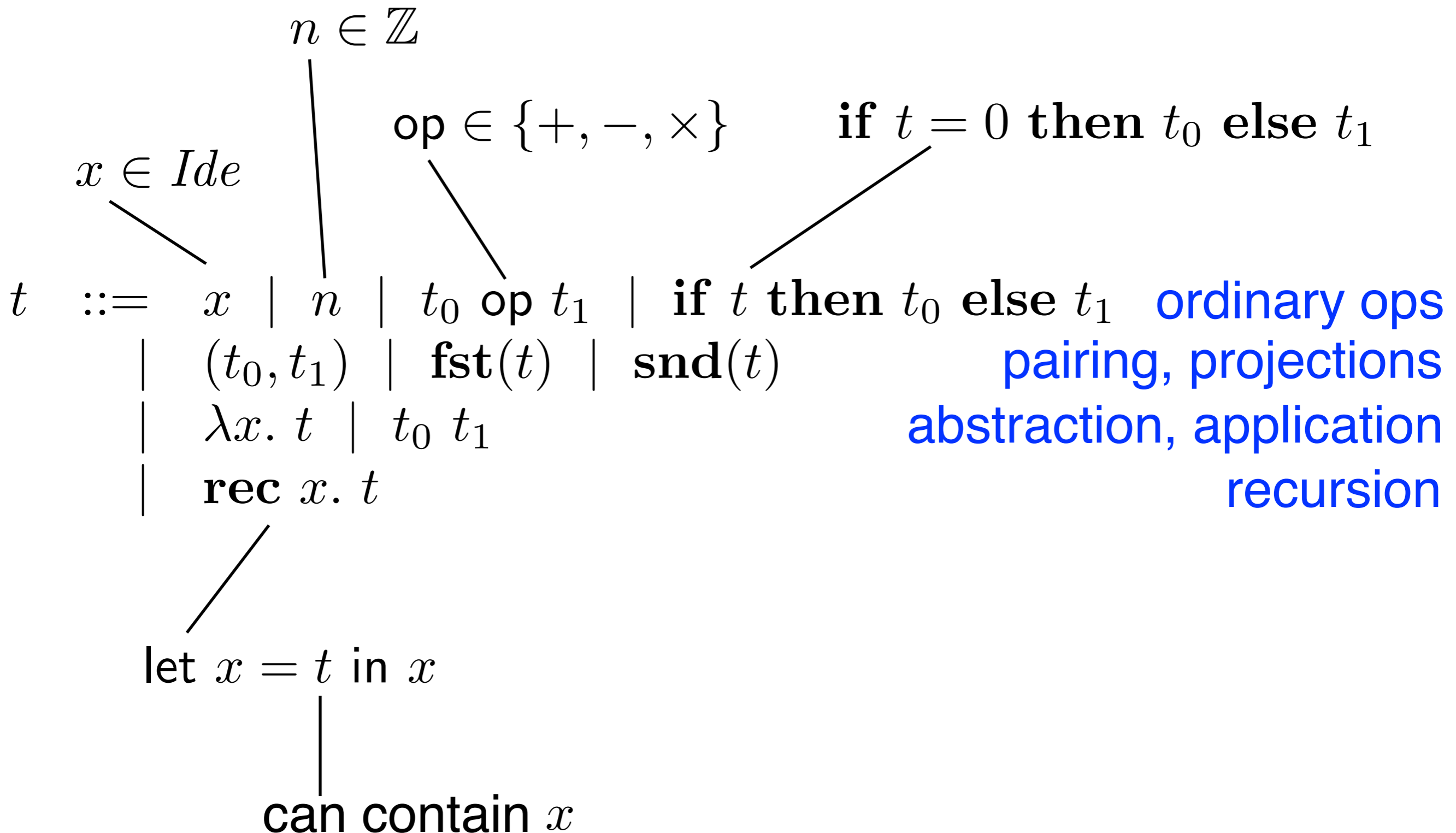
Roberto Bruni

<http://www.di.unipi.it/~bruni/>

12a - HOFL Syntax & Types

# HOFFL pre-terms (Higher Order Functional Language)

# HOFLL Syntax





# Exercise

**rec**  $f$ .  $\lambda x$ . **if**  $x$  **then** 1 **else**  $x \times (f (x - 1))$

guess the meaning of the above pre-term

factorial



# Exercise

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

guess the meaning of the above pre-term

$$\text{rep } n \ f \ x = f^n \ x$$



# Exercise

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

guess the meaning of the above pre-term

greater or equal than 0

# Badge exercise



assuming  $true = 0$

$false = \text{any } n \neq 0$

fill the dots (in HOFL)

*or*  $\triangleq$   $\lambda n. \lambda m. \dots$

*and*  $\triangleq$   $\lambda n. \lambda m. \dots$

*not*  $\triangleq$   $\lambda m. \dots$

*implies*  $\triangleq$   $\lambda n. \lambda m. \dots$

*iff*  $\triangleq$   $\lambda n. \lambda m. \dots$

# Pre-terms

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

they are called pre-terms, why?

$x + 1$  ✓

✗  $1 + (0, 5)$

**if**  $x$  **then**  $x + 1$  **else**  $x - 1$  ✓

✗  $2 \times \lambda x. x$

$(0, \lambda x. x)$  ✓

**we need a  
type system**

✗  $3 \lambda x. x + 1$

$\text{fst}(0, \lambda x. x)$  ✓

✗  $\text{fst}(3)$

$(\lambda x. x + 1) 3$  ✓

✗ **if**  $x$  **then**  $\lambda x. x$  **else**  $(x, x)$

**rec**  $f. \lambda x. x + (f 0)$  ✓

✗ **rec**  $f. \lambda x. f + x$



# HOFL types

# Types

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

which types?

infinitely many combinations!

**pairs**

*int*

*int \* int*

*int \* (int → int)*

*int \* (int \* int)*

*(int → int) \* (int → int)*

**functions**

*int → int*

*(int \* int) → int*

*(int → int) → int*

*(int → int) → (int → (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

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

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

$\hat{x}$  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

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

$$\frac{t_0 : \tau_0 \quad t_1 : \tau_1}{(t_0, t_1) : \tau_0 * \tau_1}$$

$$\frac{t : \tau_0 * \tau_1}{\text{fst}(t) : \tau_0}$$

$$\frac{t : \tau_0 * \tau_1}{\text{snd}(t) : \tau_1}$$

$$\frac{x : \tau_0 \quad t : \tau_1}{\lambda x. t : \tau_0 \rightarrow \tau_1}$$

$$\frac{t_1 : \tau_0 \rightarrow \tau_1 \quad t_0 : \tau_0}{t_1 t_0 : \tau_1}$$

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

# Well-formed terms

$$t ::= x \mid n \mid t_0 \text{ op } t_1 \mid \text{if } t \text{ then } t_0 \text{ else } t_1$$
$$\mid (t_0, t_1) \mid \text{fst}(t) \mid \text{snd}(t)$$
$$\mid \lambda x. t \mid t_0 t_1$$
$$\mid \text{rec } x. t$$
$$\tau ::= \text{int} \mid \tau_0 * \tau_1 \mid \tau_0 \rightarrow \tau_1 \quad \mathcal{T} \text{ set of all types}$$

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*

$T_\tau$  set of all well-formed terms of type  $\tau$

# Type checking

# Example

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

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

---

$fact : int \rightarrow int$



# Example

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

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

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

# Example

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

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

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

# Example

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

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

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

# Example

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

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

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

# Example

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

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

$$\begin{array}{c}
 \frac{\hat{x} = int \quad f : int \rightarrow int \quad x - 1 : int}{f(x - 1) : int} \\
 \frac{\hat{x} = int \quad \frac{\hat{x} = int \quad 1 : int}{x : int \quad 1 : int} \quad (x \times (f(x - 1))) : int}{\mathbf{if} x \mathbf{then} 1 \mathbf{else} (x \times (f(x - 1))) : int} \\
 \frac{\hat{f} = int \rightarrow int \quad \frac{\hat{x} = int \quad \mathbf{if} x \mathbf{then} 1 \mathbf{else} (x \times (f(x - 1))) : int}{x : int}}{f : int \rightarrow int \quad \lambda x. \mathbf{if} x \mathbf{then} 1 \mathbf{else} (x \times (f(x - 1))) : int \rightarrow int} \\
 \hline
 fact : int \rightarrow int
 \end{array}$$

# Example

variables are tagged with (declared) types

we deduce the type of terms by structural recursion

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

$$\begin{array}{c}
 \frac{\frac{\frac{\frac{\widehat{f} = int \rightarrow int}{f : int \rightarrow int} \quad \frac{\widehat{x} = int}{x : int} \quad \frac{1 : int}{1 : int}}{\mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ (x \times (f(x - 1))) : int}}{\widehat{x} = int \quad x : int \quad 1 : int} \quad \frac{\frac{\widehat{x} = int}{x : int} \quad \frac{\frac{\widehat{f} = int \rightarrow int}{f : int \rightarrow int} \quad \frac{x : int \quad 1 : int}{x - 1 : int}}{f(x - 1) : int}}{(x \times (f(x - 1))) : int}}{\widehat{f} = int \rightarrow int \quad \widehat{x} = int \quad x : int \quad 1 : int} \quad \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ (x \times (f(x - 1))) : int}{\widehat{f} = int \rightarrow int \quad \widehat{x} = int \quad x : int} \quad \lambda x. \ \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ (x \times (f(x - 1))) : int \rightarrow int}{f : int \rightarrow int \quad \lambda x. \ \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ (x \times (f(x - 1))) : int \rightarrow int} \\
 \hline
 fact : int \rightarrow int
 \end{array}$$

# Example

variables are tagged with (declared) types

we deduce the type of terms by structural recursion

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

$$\begin{array}{c}
 \frac{\frac{\frac{\frac{\frac{\widehat{x} = int}{x : int} \quad \frac{\widehat{x} = int}{1 : int}}{x : int \quad 1 : int} \quad \frac{\widehat{x} = int}{x : int} \quad \frac{\widehat{x} = int}{f(x-1) : int}}{(x \times (f(x-1))) : int} \quad \frac{\widehat{x} = int}{x : int} \quad \frac{\widehat{f} = int \rightarrow int}{f : int \rightarrow int}}{\mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ (x \times (f(x-1))) : int} \quad \frac{\widehat{f} = int \rightarrow int}{x : int}}{\lambda x. \ \mathbf{if} \ x \ \mathbf{then} \ 1 \ \mathbf{else} \ (x \times (f(x-1))) : int \rightarrow int} \quad \frac{\widehat{f} = int \rightarrow int}{f : int \rightarrow int}}{fact : int \rightarrow int}
 \end{array}$$

# Example

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

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

more concisely

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



# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

$\underbrace{\hspace{15em}}_{int}$



# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# Example

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

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

more concisely

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

# 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

$$t \ 0 \equiv (0, (t \ 2)) \equiv (0, (2, (t \ 4))) \equiv \dots \equiv (0, (2, (4, \dots)))$$

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:  $b \sim (t, b)$   
Expected type:  $t \rightarrow b$   
Actual type:  $t \rightarrow (t, b)$
- Relevant bindings include  
 $p :: t \rightarrow b$  (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

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

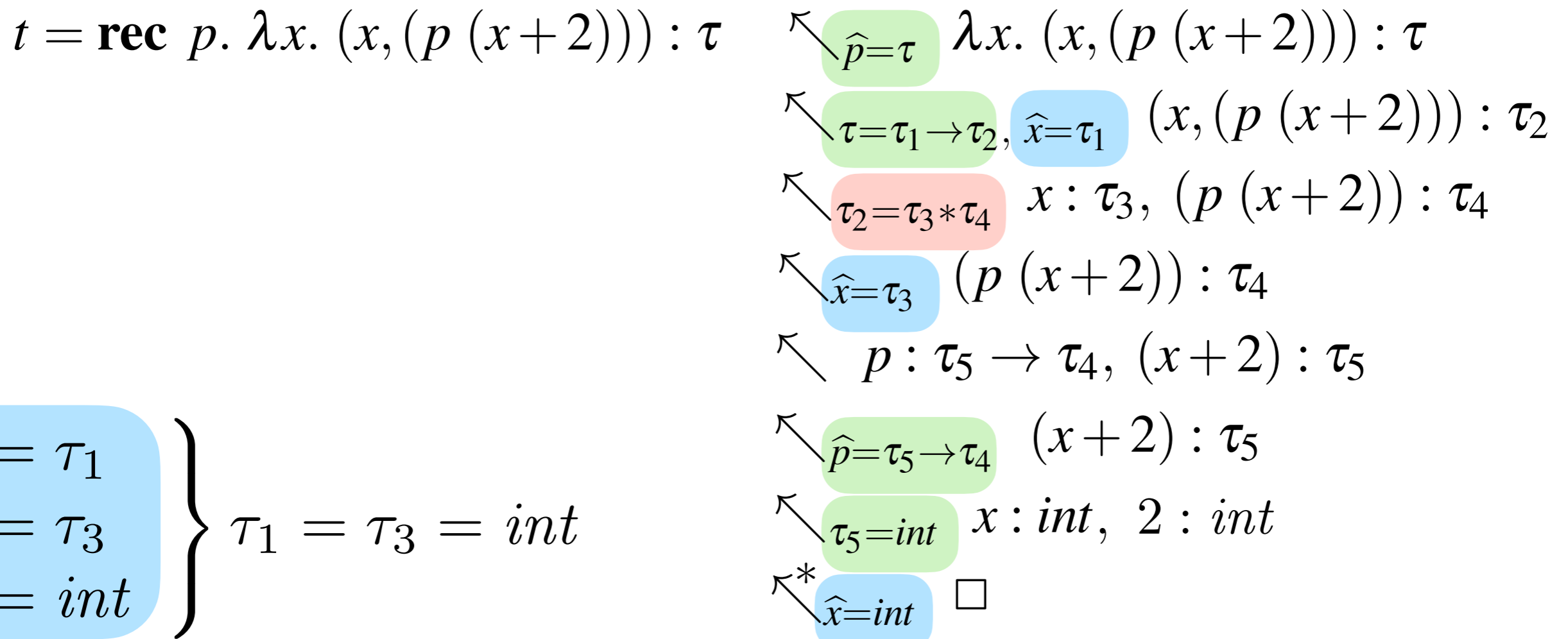
## Haskell

```
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

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



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

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

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

# Example

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

more concisely

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

$\square$   
*int*

# Example

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

more concisely

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

# Example

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

more concisely

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

# Example

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

more concisely

$$t = \mathbf{rec} \ p. \ \lambda \underset{\substack{\square \\ int}}{x}. \ (\underset{\substack{\square \\ int}}{x}, (p \ (\underset{\substack{\square \\ int}}{x} + \underset{\substack{\square \\ int}}{2})))$$

$\underbrace{\hspace{10em}}_{int}$

# Example

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

more concisely

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

# Example

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

more concisely

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

# Example

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

more concisely

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

$\tau_4$



# Example

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

more concisely

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

$\tau_4$

$\text{int} * \tau_4$

# Example

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

more concisely

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

$\underbrace{\hspace{10em}}_{\text{int}}$   
 $\underbrace{\hspace{12em}}_{\tau_4}$   
 $\underbrace{\hspace{14em}}_{\text{int} * \tau_4}$

$$(\text{int} \rightarrow (\text{int} * \tau_4)) = (\text{int} \rightarrow \tau_4) \Rightarrow \tau_4 = (\text{int} * \tau_4)$$

fail (occur check)



# Exercise

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

infer the type of the above term



# Exercise

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

infer the type of the above term

# Capture-avoiding substitutions (again)

# Free variables

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

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

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

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

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

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

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

$$\text{fv}(\lambda x. t) \stackrel{\text{def}}{=} \text{fv}(t) \setminus \{x\}$$

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

$$\text{fv}(\mathbf{rec } x. t) \stackrel{\text{def}}{=} \text{fv}(t) \setminus \{x\}$$

# Substitutions

$$n[t/x] = n$$

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

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

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

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

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

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

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

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

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

# Types are respected

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

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