

$$\lambda \alpha. e : \forall \alpha. t \leftarrow \lambda x. e$$

$$e \leftarrow t' : t[\alpha \rightarrow t'] \leftarrow e[x \rightarrow v]$$

$\forall \alpha. t$

$$\text{List}(A) \text{ append}(A) (\text{List}(A), \text{List}(A)) \{-.\} ;$$

$$\downarrow$$

$$\forall A. (\text{List}(A), \text{List}(A)) \rightarrow \text{List}(A)$$

$$\text{let append } \underline{x} : \underline{'a \text{ list}} \text{ (} \underline{y} : \underline{'a \text{ list}} \text{)}$$

$$\downarrow$$

$$: \underline{'a \text{ list}} = \{-.\} ;$$

$$\underline{'a} . \underline{'a \text{ list}} \rightarrow \underline{'a \text{ list}} \rightarrow \underline{'a \text{ list}}$$

$$\underline{\forall a.} \text{ list}(a) \rightarrow \text{list}(a) \rightarrow \text{list}(a)$$

True

$$\text{append}(x, y) \rightarrow$$

$$\text{append}(\text{int})(x, y)$$

Diagram illustrating the type inference process for the expression $\text{append}(\text{int})(x, y)$:

- The expression $\text{append}(x, y)$ is typed as App (Application), with append as the function and x, y as arguments.
- The expression $\text{append}(\text{int})(x, y)$ is typed as App , with append as the function and x, y as arguments.
- The expression $\text{append}(\text{int})$ is typed as App , with append as the function and int as the argument.

append : 'a list \rightarrow ---

append x : 'b list \rightarrow 'b list
 int list
 'b list
 ...

append ('b) x

let append x y = ---

\downarrow
 let append = λ 'a λ x (x y) (---)

$\Gamma; \Delta \vdash e : \forall \alpha. \textcircled{t}$

$\Gamma; \Delta \vdash e \langle t' \rangle : t[\alpha \rightarrow t']$

\downarrow
 a list of
 available types

$\lambda x. x + 5$

$\lambda \textcircled{x}. y + 5$

let f = $\lambda \alpha. \lambda x : \alpha. x$ in f(int)

$\alpha \rightarrow \alpha$
 \downarrow
 $\lambda x. e'$ \vee $\forall \alpha. \alpha \rightarrow \alpha$

$(\lambda x \alpha. x) 3$

int \rightarrow int

$$\Gamma; \Delta, \alpha \vdash e : t$$

$$\Gamma; \Delta \vdash \underbrace{\Delta \alpha}_{\uparrow} . e : \forall \alpha . t$$

$$\underbrace{\lambda x . x} : \underbrace{\forall \alpha . \alpha \rightarrow \alpha}_a$$

$$\text{let } f = \underbrace{\lambda \alpha . \lambda \alpha_1 . \dots}_{\uparrow}$$

$$\text{let } f \stackrel{t_1}{\leftarrow} x \stackrel{t_2}{=} y \stackrel{t_3}{=} \text{match } x \text{ with } \begin{cases} a :: b \rightarrow f_1 b & (a :: y) \\ [] \rightarrow y_1 \end{cases}$$

$t_1 = [t_2]$

$t_2 = [t_2]$

t_3

$\forall \alpha . [\alpha] \rightarrow [\alpha] \rightarrow [\alpha]$

$$\forall t_2 . \underbrace{[t_2]}_{\uparrow} \rightarrow \underbrace{[t_2]}_{\uparrow} \rightarrow \underbrace{[t_2]}_{\uparrow}$$

add

~~append~~ $\langle T \text{ extends Arith} \rangle (\underbrace{T a, T b}_{\uparrow})$

append $\langle T \rangle$ ($\text{List} \langle T \rangle$, $\text{List} \langle T \rangle$) $\rightarrow \text{List} \langle T \rangle$

$\forall T. \text{List} T \rightarrow \text{List} T \rightarrow \text{List} T$

Union {
 int n; -
 student s; -
 } type Sort =
 n of int
 | s of student

matcher -- with

| int i: int) -
 | s s' -> []

list $\langle D \rangle$ -

T -

$\widehat{A} \rightarrow B$

$\{x:t\}$

Context \leftarrow Env

constraint

$\overset{\text{int}}{\downarrow} \quad \overset{\text{List(int)}}{\downarrow}$
 $(\lambda x. x @ ((\lambda x. x) (x @ \text{Nil}))) \ 3$

\Downarrow
 $3 @ 3 @ \text{Nil}$