Programming Paradigms, session 11: Polymorphism

Exercises with solutions

Léon Gondelman · Aalborg University Copenhagen · autumn 2026

Background

The exercises use the functional language **mini-ML** of the lecture, whose syntax is:

e ::=		
	x	identifier
|	c	constant $(1, 2, \ldots, \texttt{true}, \ldots)$
|	op	primitive $(+, \times, \texttt{fst}, \ldots)$
|	$\texttt{fun}\ x \to e$	function
|	$e\ e$	application
|	$(e,\ e)$	pair
|	$\texttt{let}\ x = e\ \texttt{in}\ e$	local binding

Simple (monomorphic) types are given by:

$$\tau \ ::= \ \texttt{int} \mid \texttt{bool} \mid \cdots \mid \tau \to \tau \mid \tau \times \tau$$

The typing rules are those of slide 21 of the lecture ("the Hindley–Milner system"), without its last two rules, generalization and instantiation, which belong to the polymorphic system.

Monomorphic Typing of mini-ML

Exercise 1. (Types of simple functions.)

Give the (monomorphic) type of each of the following functions.

a) A function that increments an integer.
b) A function that takes a pair of integers and returns `true` iff both components are equal.
c) A function that takes a pair of integers and returns their sum.
d) A function that takes an integer m and returns a *function* that takes an integer n and returns $m+n$.

What can you observe about functions of several arguments? Can a function that takes a tuple be represented as a function with only one argument at a time?

> **Solution.**
>
> **a)** $\texttt{fun}\ x \to +(x, 1)$ has type $\texttt{int} \to \texttt{int}$.
> **b)** $\texttt{fun}\ p \to \texttt{fst}\ p = \texttt{snd}\ p$ has type $\texttt{int} \times \texttt{int} \to \texttt{bool}$.
> **c)** $\texttt{fun}\ p \to +(\texttt{fst}\ p, \texttt{snd}\ p)$ has type $\texttt{int} \times \texttt{int} \to \texttt{int}$.
> **d)** $\texttt{fun}\ m \to \texttt{fun}\ n \to +(m, n)$ has type $\texttt{int} \to (\texttt{int} \to \texttt{int})$.
>
> Observation: the function in (d) takes its arguments *one at a time* (this is called **currying**). Conversely, a function that takes a pair (as in (c)) can always be replaced by a curried version, and vice versa. In general, a function of type $\tau_1 \times \tau_2 \to \tau$ can be represented as $\tau_1 \to (\tau_2 \to \tau)$.

Exercise 2. (Typing `((fun x → x) (fun x → x)) 42`.)

Show that the expression $(\texttt{fun}\ x \to x)\ (\texttt{fun}\ x \to x)\ 42$ is typable in the monomorphic system. Build a complete typing derivation.

Solution. We proceed from the leaves:

1. The inner identity $\texttt{fun}\ x \to x$ can be given the type $\texttt{int} \to \texttt{int}$:

$$\frac{x : \texttt{int} \vdash x : \texttt{int}}{\vdash \texttt{fun}\ x \to x\ :\ \texttt{int} \to \texttt{int}}$$

2. The outer identity is applied to a value of type $\texttt{int} \to \texttt{int}$, so we choose its type as $(\texttt{int} \to \texttt{int}) \to (\texttt{int} \to \texttt{int})$:

$$\frac{x : \texttt{int} \to \texttt{int} \ \vdash\ x : \texttt{int} \to \texttt{int}}{\vdash \texttt{fun}\ x \to x\ :\ (\texttt{int} \to \texttt{int}) \to (\texttt{int} \to \texttt{int})}$$

3. Apply the outer to the inner:

$$\frac{\vdash \texttt{fun}\ x \to x\ :\ (\texttt{int}\to\texttt{int}) \to (\texttt{int}\to\texttt{int}) \qquad \vdash \texttt{fun}\ x \to x\ :\ \texttt{int}\to\texttt{int}}{\vdash (\texttt{fun}\ x \to x)\ (\texttt{fun}\ x \to x)\ :\ \texttt{int} \to \texttt{int}}$$

4. Apply the result to 42:

$$\frac{\vdash (\texttt{fun}\ x \to x)\ (\texttt{fun}\ x \to x)\ :\ \texttt{int} \to \texttt{int} \qquad \vdash 42 : \texttt{int}}{\vdash ((\texttt{fun}\ x \to x)\ (\texttt{fun}\ x \to x))\ 42\ :\ \texttt{int}}$$

The key insight is that the *two occurrences* of $\texttt{fun}\ x \to x$ receive *different* monomorphic types. This is possible because they are syntactically separate sub-expressions.

Exercise 3. (Untypable expressions.)

For each of the following mini-ML expressions, either give its type or explain why it cannot be typed in the monomorphic system.

a) $\texttt{fun}\ x \to +(x,\ x)$
b) $\texttt{fun}\ f \to f\ (f\ 3)$
c) $\texttt{fun}\ f \to (f\ 3,\ f\ \texttt{true})$
d) $\texttt{let}\ f = \texttt{fun}\ x \to x\ \texttt{in}\ (f\ 3,\ f\ \texttt{true})$

Solution.

a) Typable. The primitive $+$ expects $\texttt{int} \times \texttt{int}$, so x must have type $\texttt{int}$. The type is $\texttt{int} \to \texttt{int}$.
b) Typable. Since f is applied to $3 : \texttt{int}$, we need $f : \texttt{int} \to \tau$. Since $f\ 3$ is also given to f, we need $\tau = \texttt{int}$, so $f : \texttt{int} \to \texttt{int}$. The overall type is $(\texttt{int} \to \texttt{int}) \to \texttt{int}$.
c) **Not typable.** The application $f\ 3$ forces $f : \texttt{int} \to \tau_1$, while $f\ \texttt{true}$ forces $f : \texttt{bool} \to \tau_2$. In the monomorphic system f has a single type, so we would need $\texttt{int} = \texttt{bool}$, which is impossible.
d) **Not typable** (in the monomorphic system). The monomorphic `let` rule gives f a single type, the type of $\texttt{fun}\ x \to x$: here f receives one type $\tau \to \tau$, used for both occurrences of f in the pair. The same clash $\texttt{int} = \texttt{bool}$ arises. The monomorphic `let` is therefore *not* substitution: substituting $\texttt{fun}\ x \to x$ for f gives $((\texttt{fun}\ x \to x)\ 3,\ (\texttt{fun}\ x \to x)\ \texttt{true})$, which is typable, since each occurrence of $\texttt{fun}\ x \to x$ receives its own type. (This *will* become typable in the polymorphic system, where f can receive the scheme $\forall\alpha.\ \alpha \to \alpha$.)

Exercise 4. (Building derivation trees.)

Build a complete typing derivation for each of the following expressions.

a) $\texttt{let}\ f = \texttt{fun}\ x \to +(x, 1)\ \texttt{in}\ f\ 2$
b) $\texttt{fun}\ f \to \texttt{fun}\ x \to f\ (f\ x)$ (give its type and the full tree)

Solution.

a) We write $\Gamma_f = f : \texttt{int} \to \texttt{int}$.

First, type the body of f:

$$\dfrac{\dfrac{}{x : \texttt{int} \vdash + : \texttt{int}\times\texttt{int} \to \texttt{int}} \quad \dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}} \quad \dfrac{}{x : \texttt{int} \vdash 1 : \texttt{int}}}{x : \texttt{int} \vdash (x, 1) : \texttt{int}\times\texttt{int}}}{x : \texttt{int} \vdash +(x, 1) : \texttt{int}}$$

Then:

$$\dfrac{\dfrac{\text{(above)}}{\vdash \texttt{fun}\ x \to +(x, 1) : \texttt{int} \to \texttt{int}} \quad \dfrac{\dfrac{}{\Gamma_f \vdash f : \texttt{int} \to \texttt{int}} \quad \dfrac{}{\Gamma_f \vdash 2 : \texttt{int}}}{\Gamma_f \vdash f\ 2 : \texttt{int}}}{\vdash \texttt{let}\ f = \texttt{fun}\ x \to +(x, 1)\ \texttt{in}\ f\ 2\ :\ \texttt{int}}$$

b) Let $\Gamma = f : \tau \to \tau,\ x : \tau$. Then:

$$\dfrac{\dfrac{}{\Gamma \vdash f : \tau \to \tau} \quad \dfrac{\dfrac{}{\Gamma \vdash f : \tau \to \tau} \quad \dfrac{}{\Gamma \vdash x : \tau}}{\Gamma \vdash f\ x : \tau}}{\Gamma \vdash f\ (f\ x) : \tau}$$

Closing up:

$$\dfrac{\dfrac{\text{(above)}}{f : \tau\to\tau\ \vdash\ \texttt{fun}\ x \to f\ (f\ x)\ :\ \tau \to \tau}}{\vdash \texttt{fun}\ f \to \texttt{fun}\ x \to f\ (f\ x)\ :\ (\tau \to \tau) \to (\tau \to \tau)}$$

The type is $(\tau \to \tau) \to (\tau \to \tau)$ for any choice of τ (e.g. $(\texttt{int} \to \texttt{int}) \to (\texttt{int} \to \texttt{int})$).

Parametric Polymorphism (System F and Hindley–Milner)

The exercises below use the extended type algebra with type variables and $\forall$-quantification (slides 6–30).

Exercise 5. (Typing `fun x → x x` in System F.)

Show that in System F (slide 11) one can build a derivation of:

$$\vdash \texttt{fun}\ x \to x\ x\ :\ (\forall\alpha.\ \alpha \to \alpha) \to (\forall\alpha.\ \alpha \to \alpha)$$

Hint. In the body, x has the polymorphic type $\forall\alpha.\ \alpha \to \alpha$, which can be instantiated differently at each use.

Solution. Let $\Gamma = x : \forall\alpha.\ \alpha \to \alpha$.

First, we instantiate x with the type $(\forall\alpha.\ \alpha \to \alpha) \to (\forall\alpha.\ \alpha \to \alpha)$:

$$\dfrac{\dfrac{}{\Gamma \vdash x : \forall\alpha.\ \alpha \to \alpha}}{\Gamma \vdash x : (\forall\alpha.\ \alpha \to \alpha) \to (\forall\alpha.\ \alpha \to \alpha)}\ (\text{instantiate } \alpha := \forall\alpha.\ \alpha \to \alpha)$$

Second, x is also used as the argument, with its original type:

$$\dfrac{}{\Gamma \vdash x : \forall\alpha.\ \alpha \to \alpha}$$

Now apply:

$$\dfrac{\Gamma \vdash x : (\forall\alpha.\ \alpha \to \alpha) \to (\forall\alpha.\ \alpha \to \alpha) \qquad \Gamma \vdash x : \forall\alpha.\ \alpha \to \alpha}{\Gamma \vdash x\ x : \forall\alpha.\ \alpha \to \alpha}$$

Close with the function rule:

$$\dfrac{\Gamma \vdash x\ x : \forall\alpha.\ \alpha \to \alpha}{\vdash \texttt{fun}\ x \to x\ x\ :\ (\forall\alpha.\ \alpha \to \alpha) \to (\forall\alpha.\ \alpha \to \alpha)}$$

Note: this type is *not* a valid Hindley–Milner type (the $\forall$ appears in a non-prenex position), which is why $\texttt{fun}\ x \to x\ x$ is not typable in OCaml/Haskell.

Exercise 6. (Generalization and the side condition.)

Recall the generalization rule of System F:

$$\frac{\Gamma \vdash e : \tau \qquad \alpha \notin \mathcal{L}(\Gamma)}{\Gamma \vdash e : \forall \alpha.\ \tau}$$

a) Without the side condition $\alpha \notin \mathcal{L}(\Gamma)$, show that one could derive $\vdash \texttt{fun}\ x \to x : \forall \alpha.\ \alpha \to \forall \alpha.\ \alpha$ and explain why this is unsound.

b) In the correct system, explain why we *can* derive $\vdash \texttt{fun}\ x \to x : \forall \alpha.\ \alpha \to \alpha$ (the side condition is satisfied).

Solution.

a) Without the side condition, starting from $x : \alpha \vdash x : \alpha$, we generalize over α even though $\alpha \in \mathcal{L}(\{x : \alpha\})$:

$$\frac{x : \alpha \vdash x : \alpha}{x : \alpha \vdash x : \forall \alpha.\ \alpha} \qquad \Longrightarrow \qquad \vdash \texttt{fun}\ x \to x : \alpha \to \forall \alpha.\ \alpha$$

Generalizing the remaining free α: $\vdash \texttt{fun}\ x \to x : \forall \alpha.\ \alpha \to \forall \alpha.\ \alpha$.
This is unsound because we could then instantiate twice: with $\alpha := \texttt{int}$, the identity function receives the type $\texttt{int} \to \forall \alpha.\ \alpha$, so that $(\texttt{fun}\ x \to x)\ 1 : \forall \alpha.\ \alpha$, a type that can in turn be instantiated to any type. In particular, $(\texttt{fun}\ x \to x)\ 1$ could be given type $\texttt{bool}$, or $\texttt{int} \to \texttt{int}$, and the program $(\texttt{fun}\ x \to x)\ 1\ 2$ would type-check but get stuck at runtime.

b) In the correct derivation, we first close with the function rule:

$$\frac{x : \alpha \vdash x : \alpha}{\vdash \texttt{fun}\ x \to x : \alpha \to \alpha}$$

Now the environment is $\emptyset$, so $\mathcal{L}(\emptyset) = \emptyset$ and $\alpha \notin \mathcal{L}(\emptyset)$, so we may generalize:
$\vdash \texttt{fun}\ x \to x : \forall \alpha.\ \alpha \to \alpha$.

Exercise 7. (Free type variables and substitution.)

Recall the definitions from slides 9–10. The set of free type variables $\mathcal{L}(\tau)$ is defined by:

$$\mathcal{L}(\texttt{int}) = \emptyset, \quad \mathcal{L}(\alpha) = \{\alpha\}, \quad \mathcal{L}(\tau_1 \to \tau_2) = \mathcal{L}(\tau_1) \cup \mathcal{L}(\tau_2),$$

$$\mathcal{L}(\tau_1 \times \tau_2) = \mathcal{L}(\tau_1) \cup \mathcal{L}(\tau_2), \quad \mathcal{L}(\forall \alpha.\ \tau) = \mathcal{L}(\tau) \setminus \{\alpha\}.$$

The substitution $\tau[\alpha \leftarrow \tau']$ replaces every free occurrence of α in τ by τ'.

Part A. Compute the set of free type variables for each of the following types.

a) $\alpha \to \texttt{int}$
b) $\alpha \times \beta \to \alpha$
c) $\forall \alpha.\ \alpha \to \beta$
d) $\forall \alpha.\ \forall \beta.\ \alpha \times \beta \to \alpha$
e) $(\forall \alpha.\ \alpha \to \alpha) \to \beta$

Part B. Compute the following substitutions.

f) $(\alpha \to \texttt{int})[\alpha \leftarrow \texttt{bool}]$
g) $(\alpha \times \beta \to \alpha)[\alpha \leftarrow \texttt{int} \times \beta]$

h) $(\forall\alpha.\ \alpha \to \beta)[\beta \leftarrow \alpha \to \alpha]$ What do you observe about the resulting type? Are $(\forall\alpha.\ \alpha \to \beta)$ and $(\forall\gamma.\ \gamma \to \beta)$ essentially the same type? What would then happen if we do substitution $(\forall\gamma.\ \gamma \to \beta)[\beta \leftarrow \alpha \to \alpha]$? Compare resulting types of both substitutions: are they equivalent?

i) $(\forall\alpha.\ \alpha \to \beta)[\alpha \leftarrow \texttt{int}]$

Solution.

Part A.

a) $\mathcal{L}(\alpha \to \texttt{int}) = \mathcal{L}(\alpha) \cup \mathcal{L}(\texttt{int}) = \{\alpha\} \cup \emptyset = \{\alpha\}$.

b) $\mathcal{L}(\alpha \times \beta \to \alpha) = \mathcal{L}(\alpha \times \beta) \cup \mathcal{L}(\alpha) = \{\alpha, \beta\} \cup \{\alpha\} = \{\alpha, \beta\}$.

c) $\mathcal{L}(\forall\alpha.\ \alpha \to \beta) = \mathcal{L}(\alpha \to \beta) \setminus \{\alpha\} = \{\alpha, \beta\} \setminus \{\alpha\} = \{\beta\}$. The variable α is bound by the $\forall$, so only β is free.

d) $\mathcal{L}(\forall\alpha.\ \forall\beta.\ \alpha \times \beta \to \alpha) = (\mathcal{L}(\alpha \times \beta \to \alpha) \setminus \{\beta\}) \setminus \{\alpha\} = (\{\alpha, \beta\} \setminus \{\beta\}) \setminus \{\alpha\} = \emptyset$. Both variables are bound; this is a *closed* type (no free variables).

e) $\mathcal{L}((\forall\alpha.\ \alpha \to \alpha) \to \beta) = \mathcal{L}(\forall\alpha.\ \alpha \to \alpha) \cup \mathcal{L}(\beta) = \emptyset \cup \{\beta\} = \{\beta\}$.

Part B.

f) $(\alpha \to \texttt{int})[\alpha \leftarrow \texttt{bool}] = \texttt{bool} \to \texttt{int}$.

g) $(\alpha \times \beta \to \alpha)[\alpha \leftarrow \texttt{int} \times \beta] = (\texttt{int} \times \beta) \times \beta \to (\texttt{int} \times \beta)$. Note that β in the substituted type $\texttt{int} \times \beta$ refers to the same free variable β already present in the type.

h) $(\forall\alpha.\ \alpha \to \beta)[\beta \leftarrow \alpha \to \alpha] = \forall\alpha.\ \alpha \to (\alpha \to \alpha)$. We substitute β (which is free) by $\alpha \to \alpha$. Since $\alpha \neq \beta$, the definition lets the substitution go under $\forall\alpha$. In the result, all three occurrences of α are bound by $\forall\alpha$, including the two that come from $\alpha \to \alpha$, which were free before the substitution.

The types $\forall\alpha.\ \alpha \to \beta$ and $\forall\gamma.\ \gamma \to \beta$ are essentially the same type: they differ only by the name of the bound variable. Yet $(\forall\gamma.\ \gamma \to \beta)[\beta \leftarrow \alpha \to \alpha] = \forall\gamma.\ \gamma \to (\alpha \to \alpha)$, where both occurrences of α are free. The two results are *not* equivalent: $\mathcal{L}(\forall\alpha.\ \alpha \to (\alpha \to \alpha)) = \emptyset$, while $\mathcal{L}(\forall\gamma.\ \gamma \to (\alpha \to \alpha)) = \{\alpha\}$.

Remark. This is a case of *variable capture*: the free α in the substituted term $\alpha \to \alpha$ becomes accidentally bound by $\forall\alpha$. A clean implementation avoids this by α-renaming the bound variable before substituting: rename to $\forall\gamma.\ \gamma \to \beta$, then substitute to get $\forall\gamma.\ \gamma \to (\alpha \to \alpha)$.

i) $(\forall\alpha.\ \alpha \to \beta)[\alpha \leftarrow \texttt{int}] = \forall\alpha.\ \alpha \to \beta$. The substitution has no effect because α is bound by $\forall\alpha$ (the definition says $(\forall\alpha.\ \tau)[\alpha \leftarrow \tau'] = \forall\alpha.\ \tau$).

Exercise 8. (Instantiation and generalization.)

Recall from slide 27 that $\tau \leq \sigma$ ("τ is an instance of σ") means $\tau \leq \forall\alpha_1 \ldots \alpha_n.\ \tau'$ iff there exist types $\tau_1, \ldots, \tau_n$ such that $\tau = \tau'[\alpha_1 \leftarrow \tau_1, \ldots, \alpha_n \leftarrow \tau_n]$.

And from slide 28, generalization is defined by $\mathit{Gen}(\tau, \Gamma) = \forall\alpha_1 \ldots \alpha_n.\ \tau$ where $\{\alpha_1, \ldots, \alpha_n\} = \mathcal{L}(\tau) \setminus \mathcal{L}(\Gamma)$.

a) Is $\texttt{int} \to \texttt{int}$ an instance of $\forall\alpha.\ \alpha \to \alpha$? If yes, give the instantiation.

b) Is $\texttt{int} \to \texttt{bool}$ an instance of $\forall\alpha.\ \alpha \to \alpha$?

c) Is $\texttt{int} \times \texttt{bool} \to \texttt{int}$ an instance of $\forall\alpha.\ \forall\beta.\ \alpha \times \beta \to \alpha$?

d) Compute $\mathit{Gen}(\alpha \to \alpha,\ \emptyset)$.

e) Compute $\mathit{Gen}(\alpha \to \beta,\ \{x : \alpha\})$.

Solution.

a) Yes. $(\alpha \to \alpha)[\alpha \leftarrow \texttt{int}] = \texttt{int} \to \texttt{int}$. So the instantiation is $\alpha := \texttt{int}$.

b) No. We would need $\alpha = \texttt{int}$ (from the domain) and $\alpha = \texttt{bool}$ (from the codomain), which is a contradiction.

c) Yes. $(\alpha \times \beta \to \alpha)[\alpha \leftarrow \texttt{int},\ \beta \leftarrow \texttt{bool}] = \texttt{int} \times \texttt{bool} \to \texttt{int}$.

d) $\mathcal{L}(\alpha \to \alpha) = \{\alpha\}$ and $\mathcal{L}(\emptyset) = \emptyset$, so the variables to generalize are $\{\alpha\} \setminus \emptyset = \{\alpha\}$. Therefore $\mathit{Gen}(\alpha \to \alpha,\ \emptyset) = \forall\alpha.\ \alpha \to \alpha$.

e) $\mathcal{L}(\alpha \to \beta) = \{\alpha, \beta\}$ and $\mathcal{L}(\{x : \alpha\}) = \{\alpha\}$, so the variables to generalize are $\{\alpha, \beta\} \setminus \{\alpha\} = \{\beta\}$. Therefore $\mathit{Gen}(\alpha \to \beta,\ \{x : \alpha\}) = \forall\beta.\ \alpha \to \beta$. Note that α is *not* generalized because it appears free in the environment.

Exercise 9. (Derivation of `let f = fun x → x in (f 1, f true)` in the syntax-directed Hindley–Milner system.)

We saw in Exercise 3(d) that $\texttt{let}\ f = \texttt{fun}\ x \to x\ \texttt{in}\ (f\ 1,\ f\ \texttt{true})$ is **not** typable in the monomorphic system. Using the syntax-directed Hindley–Milner rules (slides 25–28), build a complete typing derivation showing that this expression **is** typable. State the type scheme assigned to f and the overall type of the expression.

Solution.

Step 1: Type $\texttt{fun}\ x \to x$.

Using the function rule with a type variable α:

$$\frac{x : \alpha \vdash x : \alpha}{\emptyset \vdash \texttt{fun}\ x \to x\ :\ \alpha \to \alpha}$$

The inferred type is $\alpha \to \alpha$.

Step 2: Generalize.

The let rule uses generalization: $\mathit{Gen}(\alpha \to \alpha,\ \emptyset) = \forall\alpha.\ \alpha \to \alpha$ (since $\mathcal{L}(\alpha \to \alpha) \setminus \mathcal{L}(\emptyset) = \{\alpha\}$). So we extend the environment with $f : \forall\alpha.\ \alpha \to \alpha$.

Let $\Gamma = f : \forall\alpha.\ \alpha \to \alpha$.

Step 3: Type $f\ 1$.

We instantiate the scheme of f with $\alpha := \texttt{int}$:

$$\frac{\dfrac{\texttt{int} \to \texttt{int}\ \le\ \forall\alpha.\ \alpha \to \alpha}{\Gamma \vdash f : \texttt{int} \to \texttt{int}} \qquad \dfrac{}{\Gamma \vdash 1 : \texttt{int}}}{\Gamma \vdash f\ 1 : \texttt{int}}$$

Step 4: Type $f\ \texttt{true}$.

We instantiate the scheme of f again, this time with $\alpha := \texttt{bool}$:

$$\frac{\dfrac{\texttt{bool} \to \texttt{bool}\ \le\ \forall\alpha.\ \alpha \to \alpha}{\Gamma \vdash f : \texttt{bool} \to \texttt{bool}} \qquad \dfrac{}{\Gamma \vdash \texttt{true} : \texttt{bool}}}{\Gamma \vdash f\ \texttt{true} : \texttt{bool}}$$

Step 5: Type the pair.

$$\frac{\Gamma \vdash f\ 1 : \texttt{int} \qquad \Gamma \vdash f\ \texttt{true} : \texttt{bool}}{\Gamma \vdash (f\ 1,\ f\ \texttt{true}) : \texttt{int} \times \texttt{bool}}$$

Step 6: Assemble with let.

$$\frac{\emptyset \vdash \texttt{fun}\ x \to x : \alpha \to \alpha \qquad \Gamma \vdash (f\ 1,\ f\ \texttt{true}) : \texttt{int} \times \texttt{bool}}{\emptyset \vdash \texttt{let}\ f = \texttt{fun}\ x \to x\ \texttt{in}\ (f\ 1,\ f\ \texttt{true})\ :\ \texttt{int} \times \texttt{bool}}$$

where $\Gamma = f : \mathit{Gen}(\alpha \to \alpha,\ \emptyset) = f : \forall\alpha.\ \alpha \to \alpha$.

The key is that `let` introduces a *polymorphic* type scheme $\forall\alpha.\ \alpha \to \alpha$ for f, which can be instantiated to $\texttt{int} \to \texttt{int}$ for $f\ 1$ and to $\texttt{bool} \to \texttt{bool}$ for $f\ \texttt{true}$ *independently*. This is exactly what was impossible in the monomorphic system.