Programming Paradigms, session 11: Polymorphism

Exercises

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?

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.

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) $\texttt{let}\ f = \texttt{fun}\ x \to x\ \texttt{in}\ (f\ 3,\ f\ \texttt{true})$

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)

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 $\texttt{fun x} \to \texttt{x x}$ 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.

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).

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

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

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.