Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Parametric Polymorphism; Records and Subtyping; Curry-Howard Isomorphism; Existential Types
Section and Practice Problems

Apr 2-5, 2019

---

1 Parametric polymorphism

(a) For each of the following System F expressions, is the expression well-typed, and if so, what type does it have? (If you are unsure, try to construct a typing derivation. Make sure you understand the typing rules.)

- $\Lambda A.\, \lambda x\!:\!A \rightarrow \textbf{int}.\, 42$
- $\lambda y\!:\!\forall X.\, X \rightarrow X.\, (y\ [\textbf{int}])\ 17$
- $\Lambda Y.\, \Lambda Z.\, \lambda f\!:\!Y \rightarrow Z.\, \lambda a\!:\!Y.\, f\ a$
- $\Lambda A.\, \Lambda B.\, \Lambda C.\, \lambda f\!:\!A \rightarrow B \rightarrow C.\, \lambda b\!:\!B.\, \lambda a\!:\!A.\, f\ a\ b$

Answer:

- $\Lambda A.\, \lambda x\!:\!A \rightarrow \textbf{int}.\, 42$ *has type*
$$\forall A.\, (A \rightarrow \textbf{int}) \rightarrow \textbf{int}$$
- $\lambda y\!:\!\forall X.\, X \rightarrow X.\, (y\ [\textbf{int}])\ 17$ *has type*
$$(\forall X.\, X \rightarrow X) \rightarrow \textbf{int}$$
- $\Lambda Y.\, \Lambda Z.\, \lambda f\!:\!Y \rightarrow Z.\, \lambda a\!:\!Y.\, f\ a$ *has type*
$$\forall Y.\, \forall Z.\, (Y \rightarrow Z) \rightarrow Y \rightarrow Z$$
- $\Lambda A.\, \Lambda B.\, \Lambda C.\, \lambda f\!:\!A \rightarrow B \rightarrow C.\, \lambda b\!:\!B.\, \lambda a\!:\!A.\, f\ a\ b$ *has type*
$$\forall A.\, \forall B.\, \forall C.\, (A \rightarrow B \rightarrow C) \rightarrow B \rightarrow A \rightarrow C$$

(b) For each of the following types, write an expression with that type.

- $\forall X.\, X \rightarrow (X \rightarrow X)$
- $(\forall C.\, \forall D.\, C \rightarrow D) \rightarrow (\forall E.\, \textbf{int} \rightarrow E)$
- $\forall X.\, X \rightarrow (\forall Y.\, Y \rightarrow X)$

Answer:

- $\forall X.\, X \rightarrow (X \rightarrow X)$ *is the type of*
$$\Lambda X.\, \lambda x\!:\!X.\, \lambda y\!:\!X.\, y$$
- $(\forall C.\, \forall D.\, C \rightarrow D) \rightarrow (\forall E.\, \textbf{int} \rightarrow E)$ *is the type of*
$$\lambda f\!:\!\forall C.\, \forall D.\, C \rightarrow D.\, \Lambda E.\, \lambda x\!:\!\textbf{int}.\, (f\ [\textbf{int}]\ [E])\ x$$

- $\forall X.\, X \to (\forall Y.\, Y \to X)$ *is the type of*

$$\Lambda X.\, \lambda x\!:\!X.\, \Lambda Y.\, \lambda y\!:\!Y.\, x$$

2 Records and Subtyping

(a) Assume that we have a language with references and records.

(i) Write an expression with type

$$\{\ cell : \textbf{int ref}, inc : \textbf{unit} \to \textbf{int}\ \}$$

such that invoking the function in the field *inc* will increment the contents of the reference in the field *cell*.

Answer: *The following expression has the appropriate type.*

$$\begin{array}{l} \textit{let}\ x = \textit{ref}\ 14\ \textit{in} \\ \{\ \ cell = x,\ inc = \lambda u\!:\!\textbf{unit}.\ x := (!x + 1)\ \ \} \end{array}$$

(ii) Assuming that the variable y is bound to the expression you wrote for part (i) above, write an expression that increments the contents of the cell twice.

Answer:

$$\textit{let}\ z = y.inc\ ()\ \textit{in}\ y.inc\ ()$$

(b) The following expression is well-typed (with type **int**). Show its typing derivation. (Note: you will need to use the subsumption rule.)

$$(\lambda x\!:\!\{dogs : \textbf{int}, cats : \textbf{int}\}.\ x.dogs + x.cats)\ \{dogs = 2, cats = 7, mice = 19\}$$

Answer:
For brevity, let $e_1 \equiv \lambda x\!:\!\{dogs : \textbf{int}, cats : \textbf{int}\}.\ x.dogs + x.cats)$ *and let* $e_2 \equiv \{dogs = 2, cats = 7, mice = 19\}$. *The derivation has the following form.*

$$\textsc{T-App}\ \dfrac{\dfrac{\vdots_1}{\vdash e_1 : \{dogs : \textbf{int}, cats : \textbf{int}\} \to \textbf{int}} \qquad \dfrac{\vdots_2}{\vdash e_2 : \{dogs : \textbf{int}, cats : \textbf{int}\}}}{\vdash e_1\ e_2 : \textbf{int}}$$

The derivation of e_1 *is straight forward:*

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x:\{dogs:\textbf{int}, cats:\textbf{int}\}}\ \text{T-VAR}}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x.dogs:\textbf{int}}\ \text{T-FIELD} \qquad \dfrac{\dfrac{}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x:\{dogs:\textbf{int}, cats:\textbf{int}\}}\ \text{T-VAR}}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x.cats:\textbf{int}}\ \text{T-FIELD}}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x.dogs + x.cats:\textbf{int}}\ \text{T-ADD}}{\vdash e_1:\{dogs:\textbf{int}, cats:\textbf{int}\} \to \textbf{int}}\ \text{T-ABS}
$$

The derivation of e_2 requires the use of subsumption, since we need to show that $e_2 \equiv \{dogs = 2, cats = 7, mice = 19\}$ has type $\{dogs : \textbf{int}, cats : \textbf{int}\}$.

$$
\dfrac{\dfrac{\vdash 2:\textbf{int} \quad \vdash 7:\textbf{int} \quad \vdash 19:\textbf{int}}{\vdash \{dogs = 2, cats = 7, mice = 19\}:\{dogs : \textbf{int}, cats : \textbf{int}, mice : \textbf{int}\}} \qquad \dfrac{}{\{dogs : \textbf{int}, cats : \textbf{int}, mice : \textbf{int}\} \leq \{dogs : \textbf{int}, cats : \textbf{int}\}}}{\vdash \{dogs = 2, cats = 7, mice = 19\}:\{dogs : \textbf{int}, cats : \textbf{int}\}}
$$

(c) Suppose that Γ is a typing context such that

$$
\begin{aligned}
\Gamma(a) &= \{dogs:\textbf{int}, cats:\textbf{int}, mice:\textbf{int}\} \\
\Gamma(f) &= \{dogs:\textbf{int}, cats:\textbf{int}\} \to \{apples:\textbf{int}, kiwis:\textbf{int}\}
\end{aligned}
$$

Write an expression e that uses variables a and f and has type $\{apples : \textbf{int}\}$ under context Γ, i.e., $\Gamma \vdash e : \{apples : \textbf{int}\}$. Write a typing derivation for it.

Answer: *A suitable expressions is $f\ a$. Note that f is a function that expects an expression of type $\{dogs : \textbf{int}, cats : \textbf{int}\}$ as an argument. Variable a is of type $\{dogs : \textbf{int}, cats : \textbf{int}, mice : \textbf{int}\}$, which is a subtype, so we can use a as an argument to f.*

Function f returns a value of type $\{apples : \textbf{int}, kiwis : \textbf{int}\}$ but our expression e needs to return a value of type $\{apples : \textbf{int}\}$. But $\{apples : \textbf{int}, kiwis : \textbf{int}\}$ is a subtype of $\{apples : \textbf{int}\}$, so it works out.

Here is a typing derivation for it. We abbreviate type $\{dogs : \textbf{int}, cats : \textbf{int}, mice : \textbf{int}\}$ to DCM and abbreviate type $\{dogs : \textbf{int}, cats : \textbf{int}\}$ to DC.

Which of the inference rules are uses of subsumption? Some of the derivations have been elided. Fill them in.

$$\dfrac{\dfrac{\Gamma \vdash f : \{dogs : \textbf{int}, cats : \textbf{int}\} \to \{apples : \textbf{int}, kiwis : \textbf{int}\} \qquad \dfrac{\Gamma \vdash a : \mathrm{DCM} \qquad \dfrac{\vdots}{\mathrm{DCM} \leq \mathrm{DC}}}{\Gamma \vdash a : \mathrm{DC}}}{\Gamma \vdash f\ a : \{apples : \textbf{int}, kiwis : \textbf{int}\}} \qquad \dfrac{\vdots}{\{apples : \textbf{int}, kiwis : \textbf{int}\} \leq \{apples : \textbf{int}\}}}{\Gamma \vdash f\ a : \{apples : \textbf{int}\}}$$

(d) Which of the following are subtypes of each other?

(a) $\{dogs : \textbf{int}, cats : \textbf{int}\} \to \{apples : \textbf{int}\}$

(b) $\{dogs : \textbf{int}\} \to \{apples : \textbf{int}\}$

(c) $\{dogs : \textbf{int}\} \to \{apples : \textbf{int}, kiwis : \textbf{int}\}$

(d) $\{dogs : \textbf{int}, cats : \textbf{int}, mice : \textbf{int}\} \to \{apples : \textbf{int}, kiwis : \textbf{int}\}$

(e) $(\{apples : \textbf{int}\})$ **ref**

(f) $(\{apples : \textbf{int}, kiwis : \textbf{int}\})$ **ref**

(g) $(\{kiwis : \textbf{int}, apples : \textbf{int}\})$ **ref**

For each such pair, make sure you have an understanding of *why* one is a subtype of the other (and for pairs that aren't subtypes, also make sure you understand).

Answer: *Of the function types:*

- *(b) is a subtype of (a)*
- *(c) is a subtype of (b)*
- *(c) is a subtype of (d)*
- *(c) is a subtype of (a)*
- *(d) is* not *a subtype of either (a) or (b), or vice versa*

The key thing is that for $\tau_1 \to \tau_2$ to be a subtype of $\tau_1' \to \tau_2'$, we must be contravariant in the argument type and covariant in the result type, i.e., $\tau_1' \leq \tau_1$ and $\tau_2 \leq \tau_2'$.

Let's consider why (b) is a subtype of (a), i.e., $\{dogs : \textbf{int}\} \to \{apples : \textbf{int}\} \leq \{dogs : \textbf{int}, cats : \textbf{int}\} \to \{apples : \textbf{int}\}$. Suppose we have a function f_b of type $\{dogs : \textbf{int}\} \to \{apples : \textbf{int}\}$, and we want to use it somewhere that wants a function g_a of type $\{dogs : \textbf{int}, cats : \textbf{int}\} \to \{apples : \textbf{int}\}$. Let's think about how g_a could be used: it could be given an argument of type $\{dogs : \textbf{int}, cats : \textbf{int}\}$, and so f_b had better be able to handle any record that has the fields dogs and cats. Indeed, f_b can be given any value of type $\{dogs : \textbf{int}\}$, i.e., any record that has a field dogs. So f_b can take any argument that g_b can be given The other way that a function can be used is by taking the result of applying it. The result types of the functions are the same, so we have no problem there. Here is a derivation showing the subtyping relation:

$$\dfrac{\dfrac{}{\{dogs : \textbf{int}, cats : \textbf{int}\} \leq \{dogs : \textbf{int}\}} \qquad \dfrac{}{\{apples : \textbf{int}\} \leq \{apples : \textbf{int}\}}}{\{dogs : \textbf{int}\} \to \{apples : \textbf{int}\} \quad \leq \quad \{dogs : \textbf{int}, cats : \textbf{int}\} \to \{apples : \textbf{int}\}}$$

Let's consider why (d) is not a subtype of (a) and (a) is not a subtype of (d). (d) is not a subtype of (a) since they are not contravariant in the argument type (i.e., the argument type of (a) is not a subtype of the argument type of (d)). (a) is not a subtype of (d) since the result type of (a) is not a subtype of the result type of (d) (i.e., they are not covariant in the result type).

For the ref types:

- *(f) is a subtype of (g) (and vice versa) assuming the more permissive subtyping rule for records that allows the order of fields to be changed.*
- *(e) is not a subtype of either (f) or (g), or vice versa.*

3 Curry-Howard isomorphism

The following logical formulas are tautologies, i.e., they are true. For each tautology, state the corresponding type, and come up with a term that has the corresponding type.

For example, for the logical formula $\forall \phi. \phi \implies \phi$, the corresponding type is $\forall X.\ X \to X$, and a term with that type is $\Lambda X.\ \lambda x\!:\!X.\, x$. Another example: for the logical formula $\tau_1 \wedge \tau_2 \implies \tau_1$, the corresponding type is $\tau_1 \times \tau_2 \to \tau_1$, and a term with that type is $\lambda x\!:\!\tau_1 \times \tau_2.\ \#1\ x$.

You may assume that the lambda calculus you are using for terms includes integers, functions, products, sums, universal types and existential types.

(a) $\forall \phi.\ \forall \psi.\ \phi \wedge \psi \implies \psi \vee \phi$

Answer: *The corresponding type is*

$$\forall X.\ \forall Y.\ X \times Y \to Y + X$$

A term with this type is

$$\Lambda X.\ \Lambda Y.\ \lambda x\!:\!X \times Y.\ \textsf{inl}_{Y+X}\ \#2\ x$$

(b) $\forall \phi.\ \forall \psi.\ \forall \chi.\ (\phi \wedge \psi \implies \chi) \implies (\phi \implies (\psi \implies \chi))$

Answer: *The corresponding type is*

$$\forall X.\ \forall Y.\ \forall Z.\ (X \times Y \to Z) \to (X \to (Y \to Z))$$

A term with this type is

$$\Lambda X.\ \Lambda Y.\ \Lambda Z. \lambda f\!:\!X \times Y \to Z.\ \lambda x\!:\!X.\ \lambda y\!:\!Y.\ f\ (x, y)$$

Note that this term uncurries the function. It is the opposite of the currying we saw in class.

(c) $\exists \phi.\ \forall \psi.\ \psi \implies \phi$

Answer: *The corresponding type is*

$$\exists X.\ \forall Y.\ Y \to X$$

A term with this type is

$$\textsf{pack}\ \{\textbf{\textsf{int}}, \Lambda Y.\ \lambda y\!:\!Y.\ 42\}\ \textsf{as}\ \exists X.\ \forall Y.\ Y \to X$$

(d) $\forall \psi.\ \psi \implies (\forall \phi.\ \phi \implies \psi)$

Answer: *The corresponding type is*

$$\forall Y.\ Y \to (\forall X.\ X \to Y)$$

A term with this type is

$$\Lambda Y.\ \lambda a\!:\!Y.\ \Lambda X.\ \lambda x\!:\!X.\ a$$

Primitive propositions in logic correspond

(e) $\forall\psi.\ (\forall\phi.\ \phi \implies \psi) \implies \psi$

Answer: *A corresponding type is*

$$\forall Y.\ (\forall X.\ X \to Y) \to Y$$

A term with this type is

$$\Lambda Y.\ \lambda f\!:\!\forall X.\ X \to Y.\ f\ [\textbf{\textit{int}}]\ 42$$

4 Existential types

(a) Write a term with type $\exists C.\ \{\ \mathit{produce} : \textbf{int} \to C, \mathit{consume} : C \to \textbf{bool}\ \}$. Moreover, ensure that calling the function *produce* will produce a value of type C such that passing the value as an argument to *consume* will return true if and only if the argument to *produce* was 42. (Assume that you have an integer comparison operator in the language.)

Answer:
In the following solution, we use ***int*** *as the witness type, and implement produce using the identity function, and implement consume by testing whether the value of type C (i.e., of witness type* ***int****) is equal to 42.*

$\textit{pack}\ \{\textbf{\textit{int}}, \{\ \mathit{produce} = \lambda a\!:\!\textbf{\textit{int}}.\ a,\ \mathit{consume} = \lambda a\!:\!\textbf{\textit{int}}.\ a = 42\ \}\}$
$\textit{as}\ \exists C.\ \{\ \mathit{produce} : \textbf{\textit{int}} \to C, \mathit{consume} : C \to \textbf{\textit{bool}}\ \}$

(b) Do the same as in part (a) above, but now use a different witness type.

Answer: *Here's another solution where instead we use* ***bool*** *as the witness type, and implement produce by comparing the integer argument to 42, and implement consume as the identity function.*

$\textit{pack}\ \{\textbf{\textit{bool}}, \{\ \mathit{produce} = \lambda a\!:\!\textbf{\textit{int}}.\ a = 42,\ \mathit{consume} = \lambda a\!:\!\textbf{\textit{bool}}.\ a\ \}\}$
$\textit{as}\ \exists C.\ \{\ \mathit{produce} : \textbf{\textit{int}} \to C, \mathit{consume} : C \to \textbf{\textit{bool}}\ \}$

(c) Assuming you have a value v of type $\exists C.\ \{\ \mathit{produce} : \textbf{int} \to C, \mathit{consume} : C \to \textbf{bool}\ \}$, use v to "produce" and "consume" a value (i.e., make sure you know how to use the unpack $\{X, x\} = e_1$ in e_2 expression.

Answer: $\textit{unpack}\ \{D, r\} = v\ \textit{in}$
$\textit{let}\ d = r.\mathit{produce}\ 19\ \textit{in}$
$r.\mathit{consume}\ d$