Products and Sums; Recursion; References; Polymorphism; Records; Subtyping
Section and Practice Problems

Week 7: Tue Mar 6–Fri Mar 10, 2023

---

1 Products and Sums

For these questions, use the lambda calculus with products and sums (Lecture 13§1.1).

(a) Write a program that constructs two values of type $\textbf{int} + (\textbf{int} \rightarrow \textbf{int})$, one using left injection, and one using right injection.

Answer:

$$\textit{let } a : \textbf{int} + (\textbf{int} \rightarrow \textbf{int}) = \textit{inl}_{\textbf{int}+(\textbf{int}\rightarrow\textbf{int})}\ 3 \textit{ in}$$
$$\textit{inr}_{\textbf{int}+(\textbf{int}\rightarrow\textbf{int})}\ \lambda x : \textbf{int}.\ 3$$

(b) Write a function that takes a value of type $\textbf{int} + (\textbf{int} \rightarrow \textbf{int})$ and if the value is an integer, it adds 7 to it, and if the value is a function it applies the function to 42.

Answer:

$$\lambda a : \textbf{int} + (\textbf{int} \rightarrow \textbf{int}).\ \textit{case } a \textit{ of } \lambda y : \textbf{int}.\ y + 7 \mid \lambda f : \textbf{int} \rightarrow \textbf{int}.\ f\ 42$$

(c) Give a typing derivation for the following program.

$$\lambda p : (\textbf{unit} \rightarrow \textbf{int}) \times (\textbf{int} \rightarrow \textbf{int}).\ \lambda x : \textbf{unit} + \textbf{int}.\ \text{case } x \text{ of } \#1\ p \mid \#2\ p$$

Answer: *For brevity, let* $e_1 \equiv \lambda x : \textbf{unit} + \textbf{int}.\ \textit{case } x \textit{ of } \#1\ p \mid \#2\ p$ *and let* $\Gamma = \{p : (\textbf{unit} \rightarrow \textbf{int}) \times (\textbf{int} \rightarrow \textbf{int}), x : \textbf{unit} + \textbf{int}\}$

$$\text{T-Abs}\ \dfrac{\text{T-Abs}\ \dfrac{\text{T-Case}\ \dfrac{\text{T-Var}\ \dfrac{}{\Gamma \vdash x : \textbf{unit} + \textbf{int}} \quad \text{T-LProj}\ \dfrac{\text{T-Var}\ \dfrac{}{\Gamma \vdash p : (\textbf{unit} \rightarrow \textbf{int} \times \textbf{int} \rightarrow \textbf{int})}}{\Gamma \vdash \#1\ p : \textbf{unit} \rightarrow \textbf{int}} \quad \text{T-RProj}\ \dfrac{\text{T-Var}\ \dfrac{}{\Gamma \vdash p : (\textbf{unit} \rightarrow \textbf{int} \times \textbf{int} \rightarrow \textbf{int})}}{\Gamma \vdash \#2\ p : \textbf{int} \rightarrow \textbf{int}}}{\Gamma \vdash \textit{case } x \textit{ of } \#1\ p \mid \#2\ p : \textbf{int}}}{p : (\textbf{unit} \rightarrow \textbf{int}) \times (\textbf{int} \rightarrow \textbf{int}) \vdash \lambda x : \textbf{unit} + \textbf{int}.\ \textit{case } x \textit{ of } \#1\ p \mid \#2\ p : (\textbf{unit} + \textbf{int}) \rightarrow \textbf{int}}}{\vdash \lambda p : (\textbf{unit} \rightarrow \textbf{int}) \times (\textbf{int} \rightarrow \textbf{int}).\ e_1 : ((\textbf{unit} \rightarrow \textbf{int}) \times (\textbf{int} \rightarrow \textbf{int})) \rightarrow (\textbf{unit} + \textbf{int}) \rightarrow \textbf{int}}$$

(d) Write a program that uses the term in part (c) above to produce the value 42.

Answer: *We refer to the term in part (c) above as* f.

$$f\ (\lambda x : \textbf{unit}.\ 42, \lambda x : \textbf{int}.\ 41)\ \textit{inl}_{\textbf{unit}+\textbf{int}}\ ()$$

2 Recursion

(a) Use the $\mu x.\ e$ expression to write a function that takes a natural number n and returns the sum of all even natural numbers less than or equal to n. (You can assume you have appropriate integer comparison operators, and also a modulus operator.)

Answer:

$$\mu f.\ \lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else}\ \textbf{if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + f\ (n-2)\ \textbf{else}\ f\ (n-1)$$

(b) Try executing your program by applying it to the number 5.

Answer: *The program executes correctly and returns 6. For brevity, we will refer to the expression from the answer above as F.*

$$\begin{aligned}
& F\ 5 \\
\longrightarrow & (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 5 \\
\longrightarrow & \textbf{if}\ 5 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (5\ \textit{mod}\ 2) = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1) \\
\longrightarrow & \textbf{if}\ \textit{false}\ \textbf{then}\ 0\ \textbf{else if}\ (5\ \textit{mod}\ 2) = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1) \\
\longrightarrow & \textbf{if}\ (5\ \textit{mod}\ 2) = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1) \\
\longrightarrow & \textbf{if}\ 1 = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1) \\
\longrightarrow & \textbf{if}\ \textit{false}\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1) \\
\longrightarrow & F\ (5-1) \\
\longrightarrow & (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ (5-1) \\
\longrightarrow & (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 4 \\
\longrightarrow & \textbf{if}\ 4 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (4\ \textit{mod}\ 2) = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1) \\
\longrightarrow & \textbf{if}\ \textit{false}\ \textbf{then}\ 0\ \textbf{else if}\ (4\ \textit{mod}\ 2) = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1) \\
\longrightarrow & \textbf{if}\ (4\ \textit{mod}\ 2) = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1) \\
\longrightarrow & \textbf{if}\ 0 = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1) \\
\longrightarrow & \textbf{if}\ \textit{true}\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1) \\
\longrightarrow & 4 + F\ (4-2) \\
\longrightarrow & 4 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ (4-2) \\
\longrightarrow & 4 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 2 \\
\longrightarrow & 4 + (\textbf{if}\ 2 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (2\ \textit{mod}\ 2) = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1)) \\
\longrightarrow & 4 + (\textbf{if}\ \textit{false}\ \textbf{then}\ 0\ \textbf{else if}\ (2\ \textit{mod}\ 2) = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1)) \\
\longrightarrow & 4 + (\textbf{if}\ (2\ \textit{mod}\ 2) = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1)) \\
\longrightarrow & 4 + (\textbf{if}\ 0 = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1)) \\
\longrightarrow & 4 + (\textbf{if}\ \textit{true}\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1)) \\
\longrightarrow & 4 + (2 + F\ (2-2)) \\
\longrightarrow & 4 + (2 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ (2-2)) \\
\longrightarrow & 4 + (2 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 0) \\
\longrightarrow & 4 + (2 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 0) \\
\longrightarrow & 4 + (2 + (\textbf{if}\ 0 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (0\ \textit{mod}\ 2) = 0\ \textbf{then}\ 0 + F\ (0-2)\ \textbf{else}\ F\ (0-1))) \\
\longrightarrow & 4 + (2 + (\textbf{if}\ \textit{true}\ \textbf{then}\ 0\ \textbf{else if}\ (0\ \textit{mod}\ 2) = 0\ \textbf{then}\ 0 + F\ (0-2)\ \textbf{else}\ F\ (0-1)))
\end{aligned}$$

$\longrightarrow 4 + (2 + (0))$
$\longrightarrow^{*} 6$

(c) Give a typing derivation for the following program. What happens if you execute the program?

$$\mu p\colon(\mathbf{int} \to \mathbf{int}) \times (\mathbf{int} \to \mathbf{int}).\ (\lambda n\colon\mathbf{int}.\ n + 1, \#1\ p)$$

Answer: *For brevity, we write τ_p for the type $(\mathbf{int} \to \mathbf{int}) \times (\mathbf{int} \to \mathbf{int})$.*

$$\text{T-REC}\;\dfrac{\text{T-PAIR}\;\dfrac{\text{T-ABS}\;\dfrac{\text{T-SUM}\;\dfrac{\text{T-VAR}\;\dfrac{}{p\colon\tau_p, n\colon\mathbf{int} \vdash n\colon\mathbf{int}} \quad \text{T-INT}\;\dfrac{}{p\colon\tau_p, n\colon\mathbf{int} \vdash 1\colon\mathbf{int}}}{p\colon\tau_p, n\colon\mathbf{int} \vdash n + 1\colon\mathbf{int}}}{p\colon\tau_p \vdash \lambda n\colon\mathbf{int}.\ n+1\colon\mathbf{int} \to \mathbf{int}} \quad \text{T-PROJ}\;\dfrac{\text{T-VAR}\;\dfrac{}{p\colon\tau_p \vdash p\colon\tau_p}}{p\colon\tau_p \vdash \#1\ p\colon\mathbf{int} \to \mathbf{int}}}{p\colon\tau_p \vdash (\lambda n\colon\mathbf{int}.\ n+1, \#1\ p)\colon\tau_p}}{\vdash \mu p\colon\tau_p.\ (\lambda n\colon\mathbf{int}.\ n+1, \#1\ p)\colon\tau_p}$$

Now, if you actually tried to execute this expression under a Call-By-Value semantics, it would unfold the recursive expression to $(\lambda n : \mathbf{int}.\ n + 1, \#1\ P)$, where P is the recursive expression $\mu p : (\mathbf{int} \to \mathbf{int}) \times (\mathbf{int} \to \mathbf{int}).\ (\lambda n : \mathbf{int}.\ n + 1, \#1\ p)$. While the first element of the pair is a value, the second $\#2\ P$ is not, and so we would attempt to evaluate that expression. However, that requires evaluating the expression $P \equiv \mu p\colon(\mathbf{int} \to \mathbf{int}) \times (\mathbf{int} \to \mathbf{int}).\ (\lambda n\colon \mathbf{int}.\ n + 1, \#1\ p)$.

So, under Call-by-Value semantics, the program will not terminate.

3 References

(a) Give a typing derivation for the following program.

$$\begin{array}{l} \text{let } a\colon\mathbf{int\ ref} = \text{ref } 4 \text{ in} \\ \text{let } b\colon(\mathbf{int} \to \mathbf{int})\ \mathbf{ref} = \text{ref } \lambda x\colon\mathbf{int}.\ x + 38 \text{ in} \\ !b\ !a \end{array}$$

Answer: *For brevity, we will write e for the expression above, and e_b for the subexpression **let** $b : (\mathbf{int} \to \mathbf{int})\ \mathbf{ref} = \mathit{ref}\ \lambda x\colon\mathbf{int}.\ x + 38\ \mathit{in}\ !b\ !a$*

$$\text{T-LET}\;\dfrac{\text{T-ALLOC}\;\dfrac{\text{T-INT}\;\dfrac{}{\vdash 4\colon\mathbf{int}}}{\vdash \mathit{ref}\ 4\colon\mathbf{int\ ref}} \quad \text{T-LET}\;\dfrac{\text{T-ALLOC}\;\dfrac{\text{T-ABS}\;\dfrac{\dfrac{\vdots_1}{a\colon\mathbf{int\ ref}, x\colon\mathbf{int} \vdash x + 38\colon\mathbf{int}}}{a\colon\mathbf{int\ ref} \vdash \lambda x\colon\mathbf{int}.\ x + 38\colon\mathbf{int} \to \mathbf{int}}}{a\colon\mathbf{int\ ref} \vdash \mathit{ref}\ \lambda x\colon\mathbf{int}.\ x + 38\colon(\mathbf{int} \to \mathbf{int})\ \mathbf{ref}} \quad \dfrac{\vdots_2}{a\colon\mathbf{int\ ref}, b\colon(\mathbf{int} \to \mathbf{int})\ \mathbf{ref} \vdash\ !b\ !a\colon\mathbf{int}}}{a\colon\mathbf{int\ ref} \vdash e_b\colon\mathbf{int}}}{\vdash e\colon\mathbf{int}}$$

The subderivation marked $\vdots_1$ is:

$$\text{T-ADD}\;\dfrac{\text{T-VAR}\;\dfrac{}{a\colon\mathbf{int\ ref}, x\colon\mathbf{int} \vdash x\colon\mathbf{int}} \quad \text{T-INT}\;\dfrac{}{a\colon\mathbf{int\ ref}, x\colon\mathbf{int} \vdash 38\colon\mathbf{int}}}{a\colon\mathbf{int\ ref}, x\colon\mathbf{int} \vdash x + 38\colon\mathbf{int}}$$

The subderivation marked $\vdots_2$ *is:*

$$\textsc{T-App}\ \dfrac{\textsc{T-Deref}\ \dfrac{\textsc{T-Var}\ \dfrac{}{\Gamma_{ab} \vdash b\colon (\textbf{int} \to \textbf{int})\ \textbf{ref}}}{\Gamma_{ab} \vdash !b\colon \textbf{int} \to \textbf{int}} \qquad \textsc{T-Deref}\ \dfrac{\textsc{T-Var}\ \dfrac{}{\Gamma_{ab} \vdash a\colon \textbf{int ref}}}{\Gamma_{ab} \vdash !a\colon \textbf{int}}}{\Gamma_{ab} \vdash !b\ !a\colon \textbf{int}}$$

where $\Gamma_{ab} = a\colon \textbf{int ref}, b\colon (\textbf{int} \to \textbf{int})\ \textbf{ref}$.

(b) Execute the program above for 4 small steps, to get configuration $\langle e, \sigma\rangle$. What is an appropriate Σ such that $\emptyset, \Sigma \vdash e\colon\tau$ and $\Sigma \vdash \sigma$?

Answer:

$$\begin{aligned}
&\langle \textit{let}\ a\colon \textbf{int ref} = \textit{ref}\ 4\ \textit{in let}\ b\colon (\textbf{int} \to \textbf{int})\ \textbf{ref} = \textit{ref}\ \lambda x\colon \textbf{int}.\ x + 38\ \textit{in}\ !b\ !a, \emptyset\rangle\\
\longrightarrow&\langle \textit{let}\ a\colon \textbf{int ref} = \ell_a\ \textit{in let}\ b\colon (\textbf{int} \to \textbf{int})\ \textbf{ref} = \textit{ref}\ \lambda x\colon \textbf{int}.\ x + 38\ \textit{in}\ !b\ !a, [\ell_a \mapsto 4]\rangle\\
\longrightarrow&\langle \textit{let}\ b\colon (\textbf{int} \to \textbf{int})\ \textbf{ref} = \textit{ref}\ \lambda x\colon \textbf{int}.\ x + 38\ \textit{in}\ !b\ !\ell_a, [\ell_a \mapsto 4]\rangle\\
\longrightarrow&\langle \textit{let}\ b\colon (\textbf{int} \to \textbf{int})\ \textbf{ref} = \ell_b\ \textit{in}\ !b\ !\ell_a, [\ell_a \mapsto 4, \ell_b \mapsto \lambda x\colon \textbf{int}.\ x + 38]\rangle\\
\longrightarrow&\langle !\ell_b\ !\ell_a, [\ell_a \mapsto 4, \ell_b \mapsto \lambda x\colon \textbf{int}.\ x + 38]\rangle
\end{aligned}$$

An appropriate store typing context is $\Sigma = \ell_a \mapsto \textbf{int}, \ell_b \mapsto \textbf{int} \to \textbf{int}$.

(c) Consider a store $\sigma = [\ell_1 \mapsto 42, \ell_2 \mapsto \lambda n\colon\textbf{int}.\ n + 1]$. What is the domain of σ?
Now consider a store type $\Sigma = [\ell_1 \mapsto \textbf{int}, \ell_2 \mapsto \textbf{int} \to \textbf{int}]$. Note that $\text{dom}(\sigma) = \text{dom}(\Sigma)$.
Show that $\emptyset, \Sigma \vdash \sigma$.

Answer: *The domain of* σ *(and of* Σ*) is the set* $\{\ell_1, \ell_2\}$.
$\emptyset, \Sigma \vdash \sigma$ *holds if and only if* $dom(\sigma) = dom(\Sigma)$ *and for all* $\ell \in dom(\sigma)$ *we have* $\emptyset, \Sigma \vdash \sigma(\ell)\colon\tau$ *where* $\Sigma(\ell) = \tau$.
Since $dom(\sigma) = dom(\Sigma) = \{\ell_1, \ell_2\}$, *we need to show that:*

- $\emptyset, \Sigma \vdash 42\colon \textbf{int}$ *and*
- $\emptyset, \Sigma \vdash \lambda n\colon \textbf{int}.\ n + 1\colon \textbf{int} \to \textbf{int}$

Both of these judgments hold, i.e., we can produce derivations for them (which we do not show here).

4 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.)

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

Answer:

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

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

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

Answer:

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

5 Records and Subtyping

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

(i) Write an expression with type
$$\{\ \textit{cell} : \textbf{int ref}, \textit{inc} : \textbf{unit} \to \textbf{int}\ \}$$
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.*

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

(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.)

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

Answer:

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

$$\text{T-APP}\ \dfrac{\dfrac{\vdots_1}{\vdash e_1 : \{dogs : \mathbf{int}, cats : \mathbf{int}\} \to \mathbf{int}} \qquad \dfrac{\vdots_2}{\vdash e_2 : \{dogs : \mathbf{int}, cats : \mathbf{int}\}}}{\vdash e_1\ e_2 : \mathbf{int}}$$

The derivation of e_1 *is straight forward:*

$$\dfrac{\dfrac{\dfrac{}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x:\{dogs:\textbf{int}, cats:\textbf{int}\}}\text{T-VAR}}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x.dogs:\textbf{int}}\text{T-FIELD}}{}$$

$$\dfrac{\dfrac{\dfrac{}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x:\{dogs:\textbf{int}, cats:\textbf{int}\}}\text{T-VAR}}{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x.cats:\textbf{int}}\text{T-FIELD}}{}$$

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

$$\dfrac{x:\{dogs:\textbf{int}, cats:\textbf{int}\} \vdash x.dogs + x.cats:\textbf{int}}{\vdash e_1:\{dogs:\textbf{int}, cats:\textbf{int}\} \rightarrow \textbf{int}}\text{T-ABS}$$

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

$$\Gamma(a) = \{dogs:\textbf{int}, cats:\textbf{int}, mice:\textbf{int}\}$$
$$\Gamma(f) = \{dogs:\textbf{int}, cats:\textbf{int}\} \rightarrow \{apples:\textbf{int}, kiwis:\textbf{int}\}$$

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.

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

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

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

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

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

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

(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 $\tau_1 \rightarrow \tau_2$ to be a subtype of $\tau_1' \rightarrow \tau_2'$, 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., $\{dogs : \textbf{int}\} \rightarrow \{apples : \textbf{int}\} \leq \{dogs : \textbf{int}, cats : \textbf{int}\} \rightarrow \{apples : \textbf{int}\}$. Suppose we have a function f_b of type $\{dogs : \textbf{int}\} \rightarrow \{apples : \textbf{int}\}$, and we want to use it somewhere that wants a function g_a of type $\{dogs : \textbf{int}, cats : \textbf{int}\} \rightarrow \{apples : \textbf{int}\}$. 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:

$$\frac{\dfrac{}{\{dogs : \textbf{int}, cats : \textbf{int}\} \leq \{dogs : \textbf{int}\}} \qquad \dfrac{}{\{apples : \textbf{int}\} \leq \{apples : \textbf{int}\}}}{\{dogs : \textbf{int}\} \rightarrow \{apples : \textbf{int}\} \;\leq\; \{dogs : \textbf{int}, cats : \textbf{int}\} \rightarrow \{apples : \textbf{int}\}}$$

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.*