Day 15.

1. Effect Types and Function Types

Now we can do the "obvious" thing for functions.

$$\frac{}{\Gamma \vdash x : \Gamma(x) \;\&\; \emptyset} \quad \frac{\Gamma[x \mapsto t_1] \vdash e : t_2 \;\&\; F}{\Gamma \vdash \lambda x.e : t_1 \to t_2 \;\&\; \emptyset} \quad \frac{\Gamma \vdash e_1 : t_1 \to t_2 \;\&\; F_1 \quad \Gamma \vdash e_2 : t_1 \;\&\; F_2}{\Gamma \vdash e_1\, e_2 : t_2 \;\&\; F_1 \cup F_2}$$

- No effects in defining a function—recall the local example.
- Application combines left and right effects—$(\lambda a.\lambda b.a + b)(\mathsf{put}\,1)\mathsf{get}$ has both get (rhs) and put (lhs) effects.

Let's see it work:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{a \mapsto \mathtt{Int}\} \vdash a : \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{}{\{a \mapsto \mathtt{Int}\} \vdash 1 : \mathtt{Int} \;\&\; \emptyset}}{\{a \mapsto \mathtt{Int}\} \vdash a + 1 : \mathtt{Int} \;\&\; \emptyset}}{\{a \mapsto \mathtt{Int}\} \vdash \mathsf{put}\,(a+1) : \mathtt{Int} \;\&\; \{\mathsf{put}\}}}{\emptyset \vdash \lambda a.\mathsf{put}\,(a+1) : \mathtt{Int} \to \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{}{\emptyset \vdash 1 : \mathtt{Int} \;\&\; \emptyset}}{\emptyset \vdash (\lambda a.\mathsf{put}\,(a+1))\,1 : \mathtt{Int} \;\&\; \emptyset}$$

Something seems to have gone wrong: intuitively, we should expect that evaluating this term will have a put effect. But that's vanished from its type.

Key idea: we've lost track of the effects that happen when executing the *body* of the function—the e above the line in the λ typing rule appears nowhere below the line. That effect shouldn't happen when we *define* the function, but we need to keep track of it for each *use* of the function.

$$\mathcal{T} \ni t ::= \mathtt{Int} \mid t \xrightarrow{F} t \quad (F \subseteq \mathcal{F})$$

Now we can restate the typing rules for functions:

$$\frac{\Gamma[x \mapsto t_1] \vdash e : t_2 \;\&\; F}{\Gamma \vdash \lambda x.e : t_1 \xrightarrow{F} e_2 \;\&\; \emptyset} \quad \frac{\Gamma \vdash e_1 : t_1 \xrightarrow{F_3} t_2 \;\&\; F_1 \quad \Gamma \vdash e_2 : t_1 \;\&\; F_2}{\Gamma \vdash e_1\, e_2 : t_2 \;\&\; F_1 \cup F_2 \cup F_3}$$

And our example should work:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{a \mapsto \mathtt{Int}\} \vdash a : \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{}{\{a \mapsto \mathtt{Int}\} \vdash 1 : \mathtt{Int} \;\&\; \emptyset}}{\{a \mapsto \mathtt{Int}\} \vdash a + 1 : \mathtt{Int} \;\&\; \emptyset}}{\{a \mapsto \mathtt{Int}\} \vdash \mathsf{put}\,(a+1) : \mathtt{Int} \;\&\; \{\mathsf{put}\}}}{\emptyset \vdash \lambda a.\mathsf{put}\,(a+1) : \mathtt{Int} \xrightarrow{\{\mathsf{put}\}} \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{}{\emptyset \vdash 1 : \mathtt{Int} \;\&\; \emptyset}}{\emptyset \vdash (\lambda a.\mathsf{put}\,(a+1))\,1 : \mathtt{Int} \;\&\; \{\mathsf{put}\}}$$

2. Putting It All Together

Let's continue our adventures with functions and effects. We'll round out our language:

$$\mathcal{E} \ni e ::= \cdots \mid \mathsf{ifz}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \mathsf{fix}\, e$$

The type-and-effect rule for ifz is going to be the "obvious" extension of its pure typing rule

$$\frac{\Gamma \vdash e_1 : \mathtt{Int}\ \&\ F_1 \quad \Gamma \vdash e_2 : T\ \&\ F_2 \quad \Gamma \vdash e_3 : T\ \&\ F_3}{\Gamma \vdash \mathsf{ifz}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : t\ \&\ F_1 \cup F_2 \cup F_3}$$

Because we don't know which branch of the ifz we're going to follow, we include both in the effect signature. Of course, we can also have fixed points with side-effects.

$$\frac{\Gamma \vdash e : t \to t\ \&\ F}{\Gamma \vdash \mathsf{fix}\, e : t\ \&\ F}$$

Finally, I'm going to introduce some syntactic sugar:

$$e_1\ ;\ e_2 \triangleq \mathtt{let}\ z = e_1\ \mathtt{in}\ e_2$$

The typing rule follows from the typing rule for its expansion:

$$\frac{\Gamma \vdash e_1 : t_1\ \&\ F_1 \quad \Gamma \vdash e_2 : t_2\ \&\ F_2}{\Gamma \vdash e_1\ ;\ e_2 : t_2\ \&\ F_1 \cup F_2}$$

Now we can put it all together:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash n : \mathtt{Int}\ \&\ \emptyset} \quad \dfrac{}{\Gamma \vdash \mathsf{get} : \mathtt{Int}\ \&\ \{\mathsf{get}\}} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash \mathsf{get} : \mathtt{Int}\ \&\ \{\mathsf{get}\}} \quad \dfrac{}{\Gamma \vdash n : \mathtt{Int}\ \&\ \emptyset}}{\Gamma \vdash \mathsf{put}\,(\mathsf{get} \times n) : \mathtt{Int}\ \&\ \{\mathsf{get}, \mathsf{put}\}} \quad \Delta}{\Gamma \vdash \mathsf{put}\,(\mathsf{get} \times n)\,;f\,(n-1) : \mathtt{Int}\ \&\ \{\mathsf{get}, \mathsf{put}\}}}{\Gamma \vdash \mathsf{ifz}\ n\ \mathsf{then\ get\ else\ put}\,(\mathsf{get} \times n)\,;f\,(n-1) : \mathtt{Int}\ \&\ \{\mathsf{get}, \mathsf{put}\}}}{\{f \mapsto \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int}\} \vdash \lambda n.\mathsf{ifz}\ n\ \mathsf{then\ get\ else\ put}\,(\mathsf{get} \times n)\,;f\,(n-1) : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int}\ \&\ \emptyset}}{\emptyset \vdash \lambda f.\lambda n.\mathsf{ifz}\ n\ \mathsf{then\ get\ else\ put}\,(\mathsf{get} \times n)\,;f\,(n-1) : (\mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int}) \to (\mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int})\ \&\ \emptyset}}{\emptyset \vdash \mathsf{fix}\,(\lambda f.\lambda n.\mathsf{ifz}\ n\ \mathsf{then\ get\ else\ put}\,(\mathsf{get} \times n)\,;f\,(n-1)) : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int}\ \&\ \emptyset}$$

where Δ is

$$\frac{\dfrac{}{\Gamma \vdash f : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int}\ \&\ \emptyset} \quad \dfrac{\dfrac{}{\Gamma \vdash n : \mathtt{Int}\ \&\ \emptyset} \quad \dfrac{}{\Gamma \vdash 1 : \mathtt{Int}\ \&\ \emptyset}}{\Gamma \vdash n - 1 : \mathtt{Int}\ \&\ \emptyset}}{\Gamma \vdash f\,(n-1) : \mathtt{Int}\ \&\ \{\mathsf{get}, \mathsf{put}\}}$$

and $\Gamma = \{f \mapsto \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int}, n \mapsto \mathtt{Int}\}$.

- We use the effects we assumed for f in the derivation. Could we have concluded that $f : \mathtt{Int} \xrightarrow{\{\mathsf{get}\}} \mathtt{Int}$? What about $f : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put},\mathsf{throw}\}} \mathtt{Int}$?

3. Subsumption

Something's gone a bit funny while we weren't paying attention. On the one hand:

$$\dfrac{\dfrac{}{\Gamma \vdash x : \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{}{\Gamma \vdash \mathsf{get} : \mathtt{Int} \;\&\; \{\mathsf{get}\}} \quad \dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{Int} \;\&\; \emptyset}}{\Gamma \vdash \mathsf{put}\, y : \mathtt{Int} \;\&\; \{\mathsf{put}\}}}{\Gamma \vdash \mathsf{ifz}\ x\ \mathsf{then\ get\ else\ put}\ y : \mathtt{Int} \;\&\; \{\mathsf{get}, \mathsf{put}\}}$$

where $\Gamma = \{x \mapsto \mathtt{Int}, y \mapsto \mathtt{Int}\}$. But now suppose we try to push abstraction over y down into the branches:

$$\dfrac{\dfrac{}{\Gamma' \vdash x : \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{\dfrac{}{\Gamma \vdash \mathsf{get} : \mathtt{Int} \;\&\; \{\mathsf{get}\}}}{\Gamma' \vdash \lambda y.\mathsf{get} : \mathtt{Int} \xrightarrow{\{\mathsf{get}\}} \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{Int} \;\&\; \emptyset}}{\Gamma \vdash \mathsf{put}\, y : \mathtt{Int} \;\&\; \{\mathsf{put}\}}}{\Gamma' \vdash \lambda y.\mathsf{put}\, y : \mathtt{Int} \xrightarrow{\{\mathsf{put}\}} \mathtt{Int} \;\&\; \emptyset}}{\Gamma' \vdash \mathsf{ifz}\ x\ \mathsf{then}\ \lambda y.\mathsf{get}\ \mathsf{else}\ \lambda y.\mathsf{put}\, y : \lightning \;\&\; \emptyset}$$

where $\Gamma' = \{x \mapsto \mathtt{Int}\}$. What's gone wrong? The two branches of the ifz have different types—because they have different effects—and so we can't come up with a uniform result type.

The first thing we can do is to introduce a typing rule called *subsumption*. Subsumption basically captures that effects in type-and-effect systems are optional: they may happen, but the type system doesn't guarantee that they do. As a result, we can also claim that more effects happen than we have actually found in the term.

$$\frac{\Gamma \vdash e : t \;\&\; F}{\Gamma \vdash e : t \;\&\; F'}\ (F \subseteq F')$$

- Subsumption is the first rule we've seen that isn't syntax directed. That is, all our other rules only apply to particular terms—even if multiple rules might apply to that term. Subsumption *always* applies, and it's up to us to figure out when it's necessary.
- We now have a different way to conclude multiple types for the same term. For any given type T, the term $\lambda a.a$ has types $\mathcal{E} \xrightarrow{\emptyset} T$, $T \xrightarrow{\{\mathsf{get}\}} T$, $T \xrightarrow{\{\mathsf{put},\mathsf{throw}\}} T$, and so forth. We might prefer the type that claims the fewest effects—i.e., the type that fits the term most precisely, or that can be used in as many typings as possible. This idea of a *most general* type is quite important in the development of functional programming, but we're not going to go any further into it for now.

With subsumption in hand, we can now make our problematic derivation go through:

$$\dfrac{\dfrac{}{\Gamma' \vdash x : \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{\dfrac{}{\Gamma \vdash \mathsf{get} : \mathtt{Int} \;\&\; \{\mathsf{get}\}}}{\Gamma \vdash \mathsf{get} : \mathtt{Int} \;\&\; \{\mathsf{get}, \mathsf{put}\}} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{Int} \;\&\; \emptyset}}{\Gamma \vdash \mathsf{put}\, y : \mathtt{Int} \;\&\; \{\mathsf{put}\}}}{\Gamma \vdash \mathsf{put}\, y : \mathtt{Int} \;\&\; \{\mathsf{get}, \mathsf{put}\}}}{\Gamma' \vdash \lambda y.\mathsf{get} : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int} \;\&\; \emptyset} \quad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{Int} \;\&\; \emptyset}}{\Gamma \vdash \mathsf{put}\, y : \mathtt{Int} \;\&\; \{\mathsf{put}\}}}{\Gamma' \vdash \lambda y.\mathsf{put}\, y : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int} \;\&\; \emptyset}}{\Gamma' \vdash \mathsf{ifz}\ x\ \mathsf{then}\ \lambda y.\mathsf{get}\ \mathsf{else}\ \lambda y.\mathsf{put}\, y : \mathtt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \mathtt{Int} \;\&\; \emptyset}$$

with Γ and Γ' as above. But you ought to find this derivation a little unsatisfying. The crux of the problem is that subsumption seems to break modularity. We need subsumption to make the branches of the ifz have the same type, but subsumption actually needs to be applied some distance

up the typing derivation to make this happen. Problems with modularity are usually brought into sharper contrast when we introduce variables. Here's a somewhat contrived example:

$$\texttt{let}\ f = \lambda a.a\ \texttt{in}\ (\mathsf{ifz}\ x\ \mathsf{then}\ f\ \mathsf{else}\ \lambda y.\mathsf{get}, \mathsf{ifz}\ x\ \mathsf{then}\ f\ \mathsf{else}\ \lambda y.\mathsf{put}\ y)$$

What type should we give f? The first component requires that its type includes the get effect, while the second component requires the put effect. To satisfy both, we have to give f the type $\texttt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \texttt{Int}$, and so the pair has type $(\texttt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \texttt{Int}, \texttt{Int} \xrightarrow{\{\mathsf{get},\mathsf{put}\}} \texttt{Int})$. But, if we inline f:

$$(\mathsf{ifz}\ x\ \mathsf{then}\ \lambda y.y\ \mathsf{else}\ \lambda y.\mathsf{get}, \mathsf{ifz}\ x\ \mathsf{then}\ \lambda y.y\ \mathsf{else}\ \lambda y.\mathsf{put}\ y)$$

the resulting term can be given the type $(\texttt{Int} \xrightarrow{\{\mathsf{get}\}} \texttt{Int}, \texttt{Int} \xrightarrow{\{\mathsf{put}\}} \texttt{Int})$. So how can we get this term for the original type? We need to generalize subsumption to a relation on terms called *subtyping*.