Day 15.

1. Type and Effect Systems

Let's consider a language with state and exceptions. (Exact semantics unimportant—we'll do with an intuitive semantics for now.)

$$t ::= z \mid t \odot t \mid x \mid \lambda x.t \mid t\ t \mid \mathsf{get} \mid \mathsf{put}\ t \mid \mathsf{throw}\ t \mid \mathsf{try}\ t\ \mathsf{catch}\ t$$

Now, we want to design a type system—that is, a static approximation of its (intuitive) dynamic semantics—for this language.

- Initial intuition: what does $\Gamma \vdash t : T_1 \to T_2$ mean? It means that t defines a term that, given a T_1 shaped argument, produces a T_2 shaped result. That is, it approximates the observable behavior of t.
- Just types sufficient for *pure* functional programming: intuitively, nothing but the free variables of a term (i.e., Γ) determine the meaning of the term.
- Insufficient for *impure* functional programming... but why would we care?
  - Correctness: can we do two things in parallel?
  - Compiler transformations: can we combine subexpressions? Omit dead code? Etc.

Goal: a *type and effect* system which characterizes both *what* a term produces and *how* it produces it.

一种类型和效果系统，它表征术语产生的内容及其产生方式。

$$\mathcal{Y} \ni T ::= \texttt{Int} \mid T \to T$$
$$\mathcal{E} \ni e ::= \mathsf{get} \mid \mathsf{put} \mid \mathsf{throw}$$

Our typing relation will now be a 4-place relation, associating a term and its context with both a type ($T \in \mathcal{Y}$) and a *set of* effects ($E \subseteq \mathcal{E}$).

$$\cdot \vdash \cdot : \cdot \,\&\, \cdot \;\subseteq\; (\mathcal{X} \rightharpoonup \mathcal{Y}) \times \mathcal{T} \times \mathcal{Y} \times \mathcal{P}(\mathcal{E})$$

Let's try to write some typing rules.

We'll start with integers.

$$\frac{}{\Gamma \vdash z : \texttt{Int} \,\&\, \emptyset} \qquad \frac{\Gamma \vdash t_1 : \texttt{Int} \,\&\, e_1 \quad \Gamma \vdash t_2 : \texttt{Int} \,\&\, e_2}{\Gamma \vdash t_1 \odot t_2 : \texttt{Int} \,\&\, e_1 \cup e_2}$$

- Integer constants "obviously" have no effect.
- Binary operations have as many effects as their operands do... for example, $\mathsf{get} + 1$ must have the effects that get does, but it doesn't add any more effects of its own.

Now let's look at some side-effecting operations:

$$\frac{}{\Gamma \vdash \mathsf{get} : \texttt{Int} \,\&\, \{\mathsf{get}\}} \qquad \frac{\Gamma \vdash t : \texttt{Int} \,\&\, e}{\Gamma \vdash \mathsf{put}\ t : \texttt{Int} \,\&\, e \cup \{\mathsf{put}\}}$$

- We're making a simplifying assumption here: that the state is always an integer. We'll do the same for throw/catch. This isn't necessary, but it is a significant simplification at this point—otherwise, we would have to track changes in the type of the state through a program.
- get has a side effect—it reads the state—so we reflect that in its effects.
- put has a side effect—it writes the state—so we reflect that in its effects. But it *also* has any side effect that its argument term t would have. For example, $\mathsf{put}\,(\mathsf{get} + 1)$ both reads and writes the state.
- get and put effects accumulate, but never go away. (We don't have any idea of a "local" state invisible to the outside world. But we could do... what might that look like?)

How about exceptions?

$$\frac{\Gamma \vdash t : \mathtt{Int} \mathbin{\&} e}{\Gamma \vdash \mathsf{throw}\, t : T \mathbin{\&} e \cup \{\mathsf{throw}\}} \qquad \frac{\Gamma \vdash t_1 : T \mathbin{\&} e_1 \quad \Gamma \vdash t_2 : \mathtt{Int} \to T \mathbin{\&} e_2}{\Gamma \vdash \mathsf{try}\ t_1\ \mathsf{catch}\ t_2 : T \mathbin{\&} (e_1 \setminus \{\mathsf{throw}\}) \cup e_2}$$

- Again, we assume that the thrown value is an `Int`; this simplifies the typing of try ... catch .... (Although it is much easier here to imagine how to adapt the effect system to thrown values of any type. How would you do it?)
- $\mathsf{throw}\, t$ has any effects that t has: $\mathsf{throw}\,\mathsf{get}$, for example, both reads the state and thrown an exception.
- $\mathsf{throw}\, t$ has an *arbitrary* return type. Why is this justified? Why is this necessary?
- In $\mathsf{try}\ t_1\ \mathsf{catch}\ t_2$, we don't know whether t_2 will execute, so we include its effects regardless. But, we can filter throw from the effects of t_1, since if t_1 did throw then it would be caught. (This does *not* mean that the effects of $\mathsf{try}\ t_1\ \mathsf{catch}\ t_2$ may not include throw. Why?)

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

$$\frac{}{\Gamma \vdash x : \Gamma(x) \mathbin{\&} \emptyset} \qquad \frac{\Gamma[x \mapsto T_1] \vdash t : T_2 \mathbin{\&} e}{\Gamma \vdash \lambda x.t : T_1 \to T_2 \mathbin{\&} \emptyset} \qquad \frac{\Gamma \vdash t_1 : T_1 \to T_2 \mathbin{\&} e_1 \quad \Gamma \vdash t_2 : T_1 \mathbin{\&} e_2}{\Gamma \vdash t_1\, t_2 : T_2 \mathbin{\&} e_1 \cup e_2}$$

- 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:

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

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.

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

Now we can restate the typing rules for functions:

$$\frac{\Gamma[x \mapsto T_1] \vdash t : T_2 \ \&\ e}{\Gamma \vdash \lambda x.t : T_1 \xrightarrow{e} T_2 \ \&\ \emptyset} \quad \frac{\Gamma \vdash t_1 : T_1 \xrightarrow{e_3} T_2 \ \&\ e_1 \quad \Gamma \vdash t_2 : T_1 \ \&\ e_2}{\Gamma \vdash t_1\, t_2 : T_2 \ \&\ e_1 \cup e_2 \cup e_3}$$

And our example should work:

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