1 Formalization

1.1 Syntax

$$f \in \textsc{TransformerNames} \qquad\qquad t \in \textsc{TypeNames}$$
$$a, x, y, z \in \textsc{Identifiers}$$

$$
\begin{array}{lll}
\mathcal{Q}, \mathcal{R}, \mathcal{S} & ::= & \texttt{!} \mid \texttt{any} \mid \texttt{nonempty} \mid \texttt{empty} \mid \texttt{every} \\
M & ::= & \texttt{fungible} \mid \texttt{unique} \mid \texttt{immutable} \mid \texttt{consumable} \mid \texttt{asset} \quad \text{(type declaration modifiers)} \\
T & ::= & \texttt{bool} \mid \texttt{nat} \mid \texttt{type } t \texttt{ is } \overline{M}\ T \mid \texttt{list } \tau \mid \{\overline{x : \tau}\} \\
\tau, \sigma, \pi & ::= & \mathcal{Q}\ T \\
\mathcal{S} & ::= & x \mid x.y \mid \texttt{true} \mid \texttt{false} \mid n \mid [x] \mid \{\overline{x : \tau \mapsto x}\} \mid \texttt{new}(t, \overline{M}, T) \\
\mathcal{D} & ::= & x \mid x.y \mid \texttt{var } x : T \mid \texttt{consume} \\
\texttt{Decl} & ::= & \texttt{transformer } f(\overline{x : \tau}) \rightarrow x : \tau\ \overline{\{\texttt{Stmt}\}} \\
\texttt{Stmt} & ::= & \texttt{pass} \\
& \mid & \mathcal{S} \rightarrow \mathcal{D} \mid \mathcal{S} \xrightarrow{x} \mathcal{D} \mid \mathcal{S} \xrightarrow{\mathcal{Q} \text{ s.t. } f(\overline{x})} \mathcal{D} \mid \mathcal{S} \rightarrow f(\overline{x}) \rightarrow \mathcal{D} \\
& \mid & \texttt{try } \overline{\{\texttt{Stmt}\}} \texttt{ catch } \overline{\{\texttt{Stmt}\}} \\
\texttt{Prog} & ::= & \overline{\texttt{Decl}}; \overline{\texttt{Stmt}}
\end{array}
$$

[Include flow-by-variable?] [Remove bool type? Can implement the "filter" selectors another way, e.g., by using a transformer returning a pair.]

1.2 Statics

$\boxed{\Gamma \vdash \mathcal{S} : \tau \dashv \Delta}$ $\boxed{\Gamma \vdash \mathcal{D} : \tau \dashv \Delta}$ **Storage Typing**

A *storage* is either a source or a destination.

$$\frac{b \in \{\texttt{true}, \texttt{false}\}}{\Gamma \vdash b :\, ! \texttt{ bool} \dashv \Gamma} \ \textsc{Bool} \qquad \frac{}{\Gamma \vdash n :\, ! \texttt{ nat} \dashv \Gamma} \ \textsc{Nat} \qquad \frac{\neg(\tau \texttt{ immutable})}{\Gamma, x : \tau \vdash x : \tau \dashv \Gamma, x : \tau} \ \textsc{Var}$$

$$\frac{\Gamma \vdash x : \tau \dashv \Delta \qquad \neg(\tau \texttt{ immutable}) \qquad \texttt{fields}(\tau) = \overline{z : \sigma} \qquad y : \mathcal{R}\ T \in \overline{z : \sigma}}{\Gamma \vdash x.y : \mathcal{R}\ T \dashv \Gamma} \ \textsc{Field}$$

$$\frac{}{\Gamma, x : \mathcal{Q}\ T \vdash [x] :\, ! \texttt{ list } \mathcal{Q}\ T \dashv \Gamma, x : \texttt{empty } T} \ \textsc{Single}$$

$$\frac{}{\Gamma, y : \mathcal{Q}\ T \vdash \{x : \mathcal{Q}\ T \mapsto y\} :\, ! \ \{x : \mathcal{Q}\ T\} \dashv \Gamma, y : \texttt{empty } T} \ \textsc{Record}$$

$$\frac{}{\Gamma \vdash \texttt{new}(t, \overline{M}, T) : \texttt{every list } ! \ (\texttt{type } t \texttt{ is } \overline{M}\ T) \dashv \Gamma} \ \textsc{New}$$

$$\frac{}{\Gamma \vdash (\texttt{var } x : T) : \texttt{empty } T \dashv \Gamma, x : \texttt{empty } T} \ \textsc{VarDef} \qquad \frac{\tau \texttt{ consumable}}{\Gamma \vdash \texttt{consume} : \tau \dashv \Gamma} \ \textsc{Consume}$$

$\boxed{\Gamma \vdash S \text{ ok} \dashv \Delta}$ **Statement Well-formedness**

$$\frac{}{\Gamma \vdash \texttt{pass ok} \dashv \Gamma} \text{ Ok-Pass} \qquad \frac{\Gamma \vdash \mathcal{S} : \mathcal{Q}\ T \dashv \Delta \qquad \texttt{update}(\Delta, \mathcal{S}, \Delta(\mathcal{S}) \ominus \mathcal{Q}) \vdash \mathcal{D} : \mathcal{R}\ T \dashv \Xi}{\Gamma \vdash (\mathcal{S} \to \mathcal{D})\ \texttt{ok} \dashv \texttt{update}(\Xi, \mathcal{D}, \Xi(\mathcal{D}) \oplus \mathcal{Q})} \text{ Ok-Flow-Every}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathcal{S} : \mathcal{Q}\ T \dashv \Delta \\ \texttt{transformer}\ f(\overline{x : \sigma}, y : \texttt{demote}(\texttt{elemtype}(T))) \to z : !\,\texttt{bool}\ \{\,\overline{\texttt{Stmt}}\,\} \\ \forall i. \texttt{demote}(\Gamma(a_i)) = \sigma_i \qquad \texttt{update}(\Delta, \mathcal{S}, \Delta(\mathcal{S}) \ominus \mathcal{Q}) \vdash \mathcal{D} : \mathcal{S}\ T \dashv \Xi\end{array}}{\Gamma \vdash (\mathcal{S} \xrightarrow{\ \mathcal{R}\ \text{s.t.}\ f(\overline{a})\ } \mathcal{D})\ \texttt{ok} \dashv \texttt{update}(\Xi, \mathcal{D}, \Xi(\mathcal{D}) \oplus \min(\mathcal{Q}, \mathcal{R}))} \text{ Ok-Flow-Filter}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathcal{S} : \mathcal{Q}\ T_1 \dashv \Delta \\ \texttt{transformer}\ f(\overline{x : \sigma}, y : \texttt{demote}(\texttt{elemtype}(T_1))) \to z : \mathcal{R}\ T_2\ \{\,\overline{\texttt{Stmt}}\,\} \\ \forall i. \texttt{demote}(\Gamma(x_i)) = \sigma_i \qquad \texttt{update}(\Delta, \mathcal{S}, \Delta(\mathcal{S}) \ominus \mathcal{Q}) \vdash \mathcal{D} : \mathcal{S}\ T_2 \dashv \Xi\end{array}}{\Gamma \vdash (\mathcal{S} \to f(\overline{x}) \to \mathcal{D})\ \texttt{ok} \dashv \texttt{update}(\Xi, \mathcal{D}, \Xi(\mathcal{D}) \oplus \mathcal{Q})} \text{ Ok-Flow-Transformer}$$

$$\frac{\Gamma \vdash \overline{S_1}\ \texttt{ok} \dashv \Delta \qquad \Gamma \vdash \overline{S_2}\ \texttt{ok} \dashv \Xi}{\Gamma \vdash (\texttt{try}\ \{\overline{S_1}\}\ \texttt{catch}\ \{\overline{S_2}\})\ \texttt{ok} \dashv \Delta \sqcup \Xi} \text{ Ok-Try}$$

$\boxed{\vdash \texttt{Decl ok}}$ **Declaration Well-formedness**

$$\frac{\overline{x : \tau} \vdash \overline{\texttt{Stmt}}\ \texttt{ok} \dashv \Gamma, y : \sigma \qquad \forall \pi \in \texttt{img}(\Gamma). \neg(\pi\ \texttt{asset})}{\vdash (\texttt{transformer}\ f(\overline{x : \tau}) \to y : \sigma\{\overline{\texttt{Stmt}}\})\ \texttt{ok}} \text{ Ok-Transformer}$$

$\boxed{\texttt{Prog ok}}$ **Program Well-formedness**

$$\frac{\vdash \overline{\texttt{Decl}}\ \texttt{ok} \qquad \emptyset \vdash \overline{\texttt{Stmt}}\ \texttt{ok} \dashv \Gamma \qquad \forall \tau \in \texttt{img}(\Gamma). \neg(\tau\ \texttt{asset})}{(\overline{\texttt{Decl}}; \overline{\texttt{Stmt}})\ \texttt{ok}} \text{ Ok-Prog}$$

1.3 Dynamics

$$\begin{array}{lll}
V & ::= & \texttt{true} \mid \texttt{false} \mid n \mid \{x : \tau \mapsto V\} \\
\mathcal{V} & ::= & \overline{V} \\
\texttt{Stmt} & ::= & \ldots \mid \texttt{put}(\mathcal{V}, \mathcal{D}) \mid \texttt{revert} \mid \texttt{try}(\Sigma, \overline{S}, \overline{S})
\end{array}$$

Definition 1. *An environment Σ is a tuple (μ, ρ) where $\mu : \textsc{IdentifierNames} \rightharpoonup \mathbb{N}$ is the* variable lookup environment, *and $\rho : \mathbb{N} \rightharpoonup \mathcal{V}$ is the* storage environment.

$\boxed{\langle \Sigma, \overline{\texttt{Stmt}} \rangle \to \langle \Sigma, \overline{\texttt{Stmt}} \rangle}$

Note that we abbreviate $\langle \Sigma, \cdot \rangle$ as Σ, which signals the end of evaluation.

The new constructs of $\texttt{resolve}(\Sigma, \mathcal{S})$ and $\texttt{put}(\mathcal{V}, \mathcal{D})$ are used to simplify the process of locating sources and updating destinations, respectively.

$$\frac{\langle \Sigma, S_1 \rangle \to \langle \Sigma', \overline{S_3} \rangle}{\langle \Sigma, S_1 \overline{S_2} \rangle \to \langle \Sigma', \overline{S_3}\ \overline{S_2} \rangle} \text{ Seq} \qquad \frac{}{\langle \Sigma, (\texttt{revert})\ \overline{S} \rangle \to \langle \Sigma, \texttt{revert} \rangle} \text{ Revert} \qquad \frac{}{\langle \Sigma, \texttt{pass} \rangle \to \Sigma} \text{ Pass}$$

Here we give the rules for the new $\texttt{put}(\mathcal{V}, \mathcal{D})$ statement. **[TODO: Need to finalize how $\mathcal{V} + \mathcal{W}$ works; in particular, need to make sure that you can't overwrite things that shouldn't be overwritten (e.g., a nonfungible nat). Probably need to tag types with modifiers or something.]**

$$\frac{}{\langle \Sigma, \mathtt{put}(\mathcal{V},\mathtt{consume})\rangle \to \Sigma}\ \text{Put-Consume} \qquad \frac{}{\langle \Sigma, \mathtt{put}(\mathcal{V},x)\rangle \to \langle \Sigma, \mathtt{put}(\mathcal{V},\mu(x))\rangle}\ \text{Put-Var}$$

$$\frac{\rho(\mu(X)) = \overline{\{x : \tau \mapsto \ell\}} \qquad (y : \sigma \mapsto k) \in \overline{x : \tau \mapsto \ell}}{\langle \Sigma, \mathtt{put}(\mathcal{V},x.y)\rangle \to \langle \Sigma, \mathtt{put}(\mathcal{V},k)\rangle}\ \text{Put-Field}$$

$$\frac{\rho(\ell) = \mathcal{W} \qquad \mathcal{W} + \mathcal{V} \neq \mathtt{revert}}{\langle \Sigma, \mathtt{put}(\mathcal{V},\ell)\rangle \to \Sigma[\rho \mapsto \rho[\ell \mapsto \mathcal{W} + \mathcal{V}]]}\ \text{Put-Loc} \qquad \frac{\rho(\ell) = \mathcal{W} \qquad \mathcal{W} + \mathcal{V} = \mathtt{revert}}{\langle \Sigma, \mathtt{put}(\mathcal{V},\ell)\rangle \to \langle \Sigma, \mathtt{revert}\rangle}\ \text{Put-Loc-Fail}$$

$$\frac{\ell \notin \mathsf{dom}(\rho)}{\langle \Sigma, \mathtt{put}(\mathcal{V},\mathtt{var}\ A : T)\rangle \to \Sigma[\mu \mapsto \mu[A \mapsto \ell], \rho \mapsto \rho[\ell \mapsto \mathcal{V}]]}\ \text{Put-VarDef}$$

$$\frac{\mathtt{resolve}(\Sigma,\mathcal{S}) = (\Sigma',\ell)}{\langle \Sigma, \mathcal{S} \to \mathcal{D}\rangle \to \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto [\,]]], \mathtt{put}(\rho'(\ell),\mathcal{D})\rangle}\ \text{Flow-Every}$$

$$\frac{\mathtt{resolve}(\Sigma,\mathcal{S}) = (\Sigma',\ell) \qquad \rho'(\ell) = \mathcal{V}}{\mathcal{U} = [v \in \mathcal{V} \mid \langle \Sigma', f(\overline{x},v)\rangle \to^* \langle \Sigma'',k\rangle \text{ and } \rho''(k) = \mathtt{true}] \qquad \mathtt{compat}(|\mathcal{U}|,|\mathcal{V}|,\mathcal{Q})}{\Sigma, (\mathcal{S} \xrightarrow{\mathcal{Q}\ \text{s.t.}\ f(\overline{x})} \mathcal{D}) \to \langle \Sigma'[\rho' \mapsto \rho'[\ell \mapsto \rho'(\ell) - \mathcal{U}]], \mathtt{put}(\mathcal{U},\mathcal{D})\rangle}\ \text{Flow-Filter}$$

$$\frac{\mathtt{resolve}(\Sigma,\mathcal{S}) = (\Sigma',\ell) \qquad \rho'(\ell) = \mathcal{V}}{\mathcal{U} = [v \in \mathcal{V} \mid \langle \Sigma', f(\overline{x},v)\rangle \to^* \langle \Sigma'',k\rangle \text{ and } \rho''(k) = \mathtt{true}] \qquad \neg\mathtt{compat}(|\mathcal{U}|,|\mathcal{V}|,\mathcal{Q})}{\Sigma, (\mathcal{S} \xrightarrow{\mathcal{Q}\ \text{s.t.}\ f(\overline{x})} \mathcal{D}) \to \langle \Sigma', \mathtt{revert}\rangle}\ \text{Flow-Filter-Fail}$$

$$\frac{\mathtt{resolve}(\Sigma,\mathcal{S}) = (\Sigma',\ell)}{\rho'(\ell) = v, \mathcal{V} \qquad \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto \mathcal{V}]], f(\overline{x},v)\rangle \to^* \langle \Sigma'',k\rangle}{\langle \Sigma, \mathcal{S} \to f(\overline{x}) \to \mathcal{D}\rangle \to \langle (\mu',\rho''), \mathtt{put}(\rho''(k),\mathcal{D})\ (\mathcal{S} \to f(\overline{x}) \to \mathcal{D})\rangle}\ \text{Flow-Transformer}$$

$$\frac{\mathtt{resolve}(\Sigma,\mathcal{S}) = (\Sigma',\ell) \qquad \rho'(\ell) = [\,]}{\langle \Sigma, \mathcal{S} \to f(\overline{x}) \to \mathcal{D}\rangle \to \Sigma}\ \text{Flow-Transformer-Done}$$

$$\frac{\ell \notin \mathsf{dom}(\rho) \qquad \mathtt{transformer}\ f(\overline{y : \tau}) \to z : \sigma\ \{\ \overline{S}\ \} \qquad \mu' = \overline{y \mapsto \mu(x)}, z \mapsto \ell}{\langle \Sigma, f(\overline{x})\rangle \to \langle (\mu', \rho[\ell \mapsto [\,]]), \overline{S}\ \ell\rangle}\ \text{Call}$$

We introduce a new statement, $\mathtt{try}(\Sigma, \overline{S_1}, \overline{S_2})$, to implement the try-catch statement, which keeps track of the environment that we begin execution in so that we can revert to the original environment in the case of a $\mathtt{revert}$.

$$\frac{}{\langle \Sigma, \mathtt{try}\ \{\overline{S_1}\}\ \mathtt{catch}\ \{\overline{S_2}\}\rangle \to \langle \Sigma, \mathtt{try}(\Sigma, \overline{S_1}, \overline{S_2})\rangle}\ \text{Try-Start}$$

$$\frac{\langle \Sigma, \overline{S_1}\rangle \to \langle \Sigma'', \overline{S_1'}\rangle}{\langle \Sigma, \mathtt{try}(\Sigma', \overline{S_1}, \overline{S_2})\rangle \to \langle \Sigma'', \mathtt{try}(\Sigma', \overline{S_1'}, \overline{S_2})\rangle}\ \text{Try-Step}$$

$$\frac{}{\langle \Sigma, \mathtt{try}(\Sigma', \mathtt{revert}, \overline{S_2})\rangle \to \langle \Sigma', \overline{S_2}\rangle}\ \text{Try-Revert} \qquad \frac{}{\langle \Sigma, \mathtt{try}(\Sigma', \cdot, \overline{S_2})\rangle \to \Sigma}\ \text{Try-Done}$$

$\boxed{\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell)}$ **Storage Resolution**

We use $\texttt{resolve}(\Sigma, \mathcal{S})$ to get the location storing the values of $\mathcal{S}$, which returns an environment because it may need to allocate new memory (e.g., in the case of creating a new record value).

$$\frac{\mu(\mathcal{S}) = \ell}{\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma, \ell)} \text{ \small Resolve-Var} \qquad \frac{\rho(\mu(x)) = \overline{\{z : \tau \mapsto \ell\}} \qquad (y : \sigma \mapsto k) \in \overline{z : \tau \mapsto \ell}}{\texttt{resolve}(\Sigma, x.y) = (\Sigma, k)} \text{ \small Resolve-Field}$$

$$\frac{\ell \notin \mathsf{dom}(\rho)}{\texttt{resolve}(\Sigma, [x]) = (\Sigma[\rho \mapsto \rho[\ell \mapsto \rho(\mu(x)), \mu(x) \mapsto [\,]]], \ell)} \text{ \small Resolve-Single}$$

$$\frac{k \notin \mathsf{dom}(\rho) \cup \overline{\ell} \qquad \Sigma' = \Sigma[\rho \mapsto \rho[\mu(y) \mapsto [\,], \overline{\ell \mapsto \rho(\mu(y))}, k \mapsto \overline{\{x : \tau \mapsto \ell\}}]]}{\texttt{resolve}(\Sigma, \overline{\{x : \tau \mapsto y\}}) = (\Sigma', k)} \text{ \small Resolve-Record}$$

$$\frac{b \in \{\texttt{true}, \texttt{false}\} \qquad \ell \notin \mathsf{dom}(\rho)}{\texttt{resolve}(\Sigma, b) = (\Sigma[\rho \mapsto \rho[\ell \mapsto b]], \ell)} \text{ \small Resolve-Bool}$$

$$\frac{\mu(t) = \ell}{\texttt{resolve}(\Sigma, \texttt{new}(t, \overline{M}, T)) = (\Sigma, \ell)} \text{ \small Resolve-Source}$$

$$\frac{t \notin \mathsf{dom}(\mu) \qquad \ell \notin \mathsf{dom}(\rho)}{\texttt{resolve}(\Sigma, \texttt{new}(t, \overline{M}, T)) = (\Sigma'[\rho \mapsto \rho[\ell \mapsto \texttt{values}(T)], \mu \mapsto \mu[t \mapsto \ell]], \ell)} \text{ \small Resolve-New-Source}$$

1.4 Auxiliaries

Definition 2. *Define* $\textbf{\textit{Quant}} = \{\texttt{empty}, \texttt{any}, !, \texttt{nonempty}, \texttt{every}\}$, *and call any* $\mathcal{Q} \in \textbf{\textit{Quant}}$ *a* type quantity. *Define* $\texttt{empty} < \texttt{any} < ! < \texttt{nonempty} < \texttt{every}$.

$\boxed{\tau \,\texttt{asset}}$ **Asset Types**

$$(\mathcal{Q}\ T) \,\texttt{asset} \Leftrightarrow \mathcal{Q} \neq \texttt{empty} \text{ and } (\texttt{asset} \in \texttt{modifiers}(T) \text{ or}$$
$$(T = \mathcal{C}\ \tau \text{ and } \tau \,\texttt{asset}) \text{ or}$$
$$(T = \{\overline{y : \sigma}\} \text{ and } \exists x : \tau \in \overline{y : \sigma}.(\tau \,\texttt{asset})))$$

$\boxed{\tau \,\texttt{consumable}}$ **Consumable Types**

$$(\mathcal{Q}\ T) \,\texttt{consumable} \Leftrightarrow \texttt{consumable} \in \texttt{modifiers}(T) \text{ or}$$
$$\neg((\mathcal{Q}\ T) \,\texttt{asset}) \text{ or}$$
$$(T = \mathcal{C}\ \tau \text{ and } \tau \,\texttt{consumable}) \text{ or}$$
$$(T = \{\overline{y : \sigma}\} \text{ and } \forall x : \tau \in \overline{y : \sigma}.(\sigma \,\texttt{consumable}))$$

$\mathcal{Q} \oplus \mathcal{R}$ represents the quantity present when flowing $\mathcal{R}$ of something to a storage already containing $\mathcal{Q}$. $\mathcal{Q} \ominus \mathcal{R}$ represents the quantity left over after flowing $\mathcal{R}$ from a storage containing $\mathcal{Q}$.

Definition 3. *Let $\mathcal{Q}, \mathcal{R} \in \textbf{Quant}$. Define the commutative operator $\oplus$, called* combine, *as the unique function $\textbf{Quant}^2 \to \textbf{Quant}$ such that*

$$
\begin{aligned}
\mathcal{Q} \oplus \texttt{empty} &= \mathcal{Q} \\
\mathcal{Q} \oplus \texttt{every} &= \texttt{every} \\
\texttt{nonempty} \oplus \mathcal{R} &= \texttt{nonempty} \quad \textit{if } \texttt{empty} < \mathcal{R} < \texttt{every} \\
! \oplus \mathcal{R} &= \texttt{nonempty} \quad \textit{if } \texttt{empty} < \mathcal{R} < \texttt{every} \\
\texttt{any} \oplus \texttt{any} &= \texttt{any}
\end{aligned}
$$

Define the operator $\ominus$, called split, *as the unique function $\textbf{Quant}^2 \to \textbf{Quant}$ such that*

$$
\begin{aligned}
\mathcal{Q} \ominus \texttt{empty} &= \mathcal{Q} \\
\texttt{empty} \ominus \mathcal{R} &= \texttt{empty} \\
\mathcal{Q} \ominus \texttt{every} &= \texttt{empty} \\
\texttt{every} \ominus \mathcal{R} &= \texttt{every} \quad \textit{if } \mathcal{R} < \texttt{every} \\
\texttt{nonempty} - \mathcal{R} &= \texttt{any} \quad \textit{if } \texttt{empty} < \mathcal{R} < \texttt{every} \\
! - \mathcal{R} &= \texttt{empty} \quad \textit{if } ! \leq \mathcal{R} \\
! - \texttt{any} &= \texttt{any} \\
\texttt{any} - \mathcal{R} &= \texttt{any} \quad \textit{if } \texttt{empty} < \mathcal{R} < \texttt{every}
\end{aligned}
$$

Note that we write $(\mathcal{Q}\ T) \oplus \mathcal{R}$ to mean $(\mathcal{Q} \oplus \mathcal{R})\ T$ and similarly $(\mathcal{Q}\ T) \ominus \mathcal{R}$ to mean $(\mathcal{Q} \ominus \mathcal{R})\ T$.

Definition 4. *We can consider a type environment Γ as a function $\textsc{Identifiers} \to \textsc{Types} \cup \{\bot\}$ as follows:*

$$
\Gamma(x) = \begin{cases} \tau & \textit{if } x : \tau \in \Gamma \\ \bot & \textit{otherwise} \end{cases}
$$

We write $\textbf{dom}(\Gamma)$ to mean $\{x \in \textsc{Identifiers} \mid \Gamma(x) \neq \bot\}$, and $\Gamma|_X$ to mean the environment $\{x : \tau \in \Gamma \mid x \in X\}$ (restricting the domain of Γ).

Definition 5. *Let $\mathcal{Q}$ and $\mathcal{R}$ be type quantities, $T_{\mathcal{Q}}$ and $T_{\mathcal{R}}$ base types, and Γ and Δ type environments. Define the following orderings, which make types and type environments into join-semilattices. For type quantities, define the partial order $\sqsubseteq$ as the reflexive closure of the strict partial order $\sqsubset$ given by*

$$
\mathcal{Q} \sqsubset \mathcal{R} \Leftrightarrow (\mathcal{Q} \neq \texttt{any} \textit{ and } \mathcal{R} = \texttt{any}) \textit{ or } (\mathcal{Q} \in \{!, \texttt{every}\} \textit{ and } \mathcal{R} = \texttt{nonempty})
$$

For types, define the partial order $\leq$ by

$$
\mathcal{Q}\ T_{\mathcal{Q}} \leq \mathcal{R}\ T_{\mathcal{R}} \Leftrightarrow T_{\mathcal{Q}} = T_{\mathcal{R}} \textit{ and } \mathcal{Q} \sqsubseteq \mathcal{R}
$$

For type environments, define the partial order $\leq$ by

$$
\Gamma \leq \Delta \Leftrightarrow \forall x. \Gamma(x) \leq \Delta(x)
$$

Denote the join of Γ and Δ by $\Gamma \sqcup \Delta$.

$\boxed{\texttt{elemtype}(T) = \tau}$

$$
\texttt{elemtype}(T) = \begin{cases} \texttt{elemtype}(T') & \textit{if } T = \texttt{type } t \textit{ is } \overline{M}\ T' \\ \tau & \textit{if } T = \mathcal{C}\ \tau \\ !\ T & \textit{otherwise} \end{cases}
$$

$\boxed{\texttt{modifiers}(T) = \overline{M}}$ **Type Modifiers**

$$\texttt{modifiers}(T) = \begin{cases} \overline{M} & \text{if } T = \texttt{type } t \texttt{ is } \overline{M} \ T \\ \emptyset & \text{otherwise} \end{cases}$$

$\boxed{\text{demote}(\tau) = \sigma}$ $\boxed{\text{demote}_*(T_1) = T_2}$ **Type Demotion** demote and demote$_*$ take a type and "strip" all the asset modifiers from it, as well as unfolding named type definitions. This process is useful, because it allows selecting asset types without actually having a value of the desired asset type. Note that demoting a transformer type changes nothing. This is because a transformer is **never** an asset, regardless of the types that it operators on, because it has no storage.

$$\text{demote}(\mathcal{Q} \ T) = \mathcal{Q} \ \text{demote}_*(T)$$
$$\text{demote}_*(\texttt{bool}) = \texttt{bool}$$
$$\text{demote}_*(\texttt{nat}) = \texttt{nat}$$
$$\text{demote}_*(\{\overline{x : \tau}\}) = \left\{\overline{x : \text{demote}(\tau)}\right\}$$
$$\text{demote}_*(\texttt{type } t \texttt{ is } \overline{M} \ T) = \text{demote}_*(T)$$

$\boxed{\texttt{fields}(T) = \overline{x : \tau}}$ **Fields**

$$\texttt{fields}(T) = \begin{cases} \overline{x : \tau} & \text{if } T = \{\overline{x : \tau}\} \\ \texttt{fields}(T) & \text{if } T = \texttt{type } t \texttt{ is } \overline{M} \ T \\ \emptyset & \text{otherwise} \end{cases}$$

$\boxed{\texttt{update}(\Gamma, x, \tau)}$ **Type environment modification**

$$\texttt{update}(\Gamma, x, \tau) = \begin{cases} \Delta, x : \tau & \text{if } \Gamma = \Delta, x : \sigma \\ \Gamma & \text{otherwise} \end{cases}$$

$\boxed{\texttt{compat}(n, m, \mathcal{Q})}$ The relation $\texttt{compat}(n, m, \mathcal{Q})$ holds when the number of values sent, n, is compatible with the original number of values m, and the type quantity used, $\mathcal{Q}$.

$$\begin{aligned} \texttt{compat}(n, m, \mathcal{Q}) \Leftrightarrow & (\mathcal{Q} = \texttt{nonempty} \text{ and } n \geq 1) \text{ or} \\ & (\mathcal{Q} = ! \text{ and } n = 1) \text{ or} \\ & (\mathcal{Q} = \texttt{empty} \text{ and } n = 0) \text{ or} \\ & (\mathcal{Q} = \texttt{every} \text{ and } n = m) \text{ or} \\ & \mathcal{Q} = \texttt{any} \end{aligned}$$

$\boxed{\texttt{values}(T) = \mathcal{V}}$ The function $\texttt{values}$ gives a list of all of the values of a given base type.

$$\texttt{values}(\texttt{bool}) = [\texttt{true}, \texttt{false}]$$
$$\texttt{values}(\texttt{nat}) = [0, 1, 2, \ldots]$$
$$\texttt{values}(\texttt{list } T) = [L | L \subseteq \texttt{values}(T), |L| < \infty]$$
$$\texttt{values}(\texttt{type } t \texttt{ is } \overline{M} \ T) = \texttt{values}(T)$$
$$\texttt{values}(\{\overline{x : \mathcal{Q} \ T}\}) = [\{\overline{x : \tau \mapsto v}\} | \overline{v \in \texttt{values}(T)}]$$