1 Formalization

1.1 Locators

$$\begin{array}{lll}
\mathcal{L} & ::= & x \mid \texttt{var}\ x : T \mid \mathcal{L} \to @x \mid [\overline{\mathcal{L}}] \mid \{\overline{x : \tau \mapsto \mathcal{L}}\} \mid \mathcal{L} \xrightarrow{\mathcal{L}} \mid \mathcal{L} \xrightarrow{\mathcal{Q}\text{ s.t. } f[\overline{T}](\overline{L})} & \text{(locators)} \\
\text{Flow} & ::= & \mathcal{L} \to \mathcal{L} \mid \mathcal{L} \to f[\overline{T}](\overline{x}) \to \mathcal{L} & \text{(flows)}
\end{array}$$

$$\frac{}{\langle \Sigma, x \rangle \to \langle \Sigma, \mu(x) \rangle}\ \textsc{Loc-Id} \qquad \frac{\langle \Sigma, \mathcal{L} \rangle \to \langle \Sigma', \mathcal{L}' \rangle}{\langle \Sigma, \mathcal{L} \to @x \rangle \to \langle \Sigma', \mathcal{L}' \to @x \rangle}\ \textsc{Loc-Field-Congr}$$

$$\frac{\overline{\rho(\ell).x = k}}{\left\langle \Sigma, \overline{\ell} \to @x \right\rangle \to \left\langle \Sigma', \overline{k} \right\rangle}\ \textsc{Loc-Field} \qquad \frac{\ell \notin \mathsf{dom}(\rho)}{\langle \Sigma, \texttt{var}\ x : T \rangle \to \langle \Sigma[\mu \mapsto \mu[x \mapsto \ell], \rho \mapsto \rho[\ell \mapsto []]], \ell \rangle}\ \textsc{Loc-VarDef}$$

$$\frac{\langle \Sigma, \mathcal{L} \rangle \to \langle \Sigma'', \mathcal{L}'' \rangle}{\left\langle \Sigma, \mathcal{L} \xrightarrow{\mathcal{L}'} \right\rangle \to \left\langle \Sigma'', \mathcal{L}'' \xrightarrow{\mathcal{L}'} \right\rangle}\ \textsc{Loc-Val-Src-Congr} \qquad \frac{\langle \Sigma, \mathcal{L}' \rangle \to \langle \Sigma'', \mathcal{L}'' \rangle}{\left\langle \Sigma, \mathcal{L} \xrightarrow{\mathcal{L}'} \right\rangle \to \left\langle \Sigma'', \mathcal{L} \xrightarrow{\mathcal{L}'} \right\rangle}\ \textsc{Loc-Val-Sel-Congr}$$

$$\frac{}{\left\langle \Sigma, \overline{\ell} \xrightarrow{\overline{k}} \right\rangle \to \left\langle \Sigma, \texttt{select}(\rho, \overline{\ell}, \overline{k}) \right\rangle}\ \textsc{Loc-Val}$$

$$\texttt{select}(\rho, \overline{\ell}, \overline{k}) = \begin{cases} [] & \text{if } \overline{k} = [] \\ \texttt{revert} & \text{if } \overline{\ell} = [] \text{ and } \overline{k} \neq [] \\ j, \texttt{select}(\rho, \overline{\ell} \setminus j, \overline{k'}) & \text{if } \overline{k} = (i, \overline{k'}) \text{ and } j \in \overline{\ell} \text{ and } \rho(i) = \rho(j) \\ \texttt{select}(\rho, \overline{\ell}, \overline{k'}) & \text{if } \overline{k} = (i, \overline{k'}) \text{ and } \forall j \in \overline{\ell}. \rho(i) \neq \rho(j) \end{cases}$$

1.2 Syntax

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

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

1.3 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.

$$\frac{b \in \{\texttt{true}, \texttt{false}\}}{\Gamma \vdash b : !\ \texttt{bool} \dashv \Gamma}\ \textsc{Bool} \qquad \frac{}{\Gamma \vdash n : !\ \texttt{nat} \dashv \Gamma}\ \textsc{Nat}$$

$$\frac{}{\Gamma, x : \tau \vdash \texttt{demote}(x) : \text{demote}(\tau) \dashv \Gamma, x : \tau}\ \textsc{Demote} \qquad \frac{\texttt{immutable} \notin \text{modifiers}(\tau)}{\Gamma, x : \tau \vdash x : \tau \dashv \Gamma, x : \tau}\ \textsc{Var}$$

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash \mathcal{S} : \mathcal{Q}\ T \dashv \Delta \qquad \Delta \vdash x : \text{demote}(\mathcal{R}\ T) \dashv \Delta \qquad \text{update}(\Delta, \mathcal{S}, \Delta(\mathcal{S}) \ominus \mathcal{Q}) \vdash \mathcal{D} : \mathcal{S}\ T \dashv \Xi}{\Gamma \vdash (\mathcal{S} \xrightarrow{x} \mathcal{D})\ \textbf{ok} \dashv \text{update}(\Xi, \mathcal{D}, \Xi(\mathcal{D}) \oplus \mathcal{R})}\ \textsc{Ok-Flow-Var}$$

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

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

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

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

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

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

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

1.4 Dynamics

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

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

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

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. **[Can probably just make put into a function as well, like resolve.]**

$$\frac{\langle \Sigma, S_1 \rangle \to \left\langle \Sigma', \overline{S_3} \right\rangle}{\left\langle \Sigma, S_1\, \overline{S_2} \right\rangle \to \left\langle \Sigma', \overline{S_3}\ \overline{S_2} \right\rangle}\ \textsc{Seq} \qquad \frac{}{\left\langle \Sigma, (\texttt{revert})\ \overline{S} \right\rangle \to \langle \Sigma, \texttt{revert} \rangle}\ \textsc{Revert}$$

Here we give the rules for the new $\texttt{put}(\mathcal{V}, \mathcal{D})$ statement. Here $\mathcal{V} + \mathcal{W}$ refers to the combine operation for relevant values. **[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.] [Here we have an issue if we want to do this thing that actually allows working with infinite lists/sets of values, because we can't concatenate two infintie lists (or at least, I'm not sure how...). But we could use multisets and dodge all the issues (also resolves the issue of how to deal with duplicates).]**

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

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

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

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

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

$$\frac{\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \qquad \rho'(\ell) = \mathcal{V} \qquad \rho'(\mu'(x)) = \mathcal{W} \qquad \mathcal{W} \le \mathcal{V}}{\left\langle \Sigma, \mathcal{S} \xrightarrow{x} \mathcal{D} \right\rangle \to \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto \mathcal{V} - \mathcal{W}]], \texttt{put}(\mathcal{W}, \mathcal{D}) \rangle} \text{ FLOW-VAR}$$

$$\frac{\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \qquad \rho'(\ell) = \mathcal{V} \qquad \rho'(\mu'(x)) = \mathcal{W} \qquad \mathcal{W} \not\le \mathcal{V}}{\left\langle \Sigma, \mathcal{S} \xrightarrow{x} \mathcal{D} \right\rangle \to \langle \Sigma', \texttt{revert} \rangle} \text{ FLOW-VAR-FAIL}$$

$$\frac{\begin{array}{c}\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \qquad \rho'(\ell) = \mathcal{V} \\ \mathcal{U} = [v \in \mathcal{V} \mid \langle \Sigma', f(\overline{x}, v) \rangle \to^* \langle \Sigma'', k \rangle \text{ and } \rho''(k) = \texttt{true}] \qquad \text{compat}(|\mathcal{U}|, |\mathcal{V}|, \mathcal{Q})\end{array}}{\left\langle \Sigma, \mathcal{S} \xrightarrow{\mathcal{Q} \textbf{ s.t. } f(\overline{x})} \mathcal{D} \right\rangle \to \langle \Sigma'[\rho' \mapsto \rho'[\ell \mapsto \rho'(\ell) - \mathcal{U}]], \texttt{put}(\mathcal{U}, \mathcal{D}) \rangle} \text{ FLOW-FILTER}$$

$$\frac{\begin{array}{c}\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \qquad \rho'(\ell) = \mathcal{V} \\ \mathcal{U} = [v \in \mathcal{V} \mid \langle \Sigma', f(\overline{x}, v) \rangle \to^* \langle \Sigma'', k \rangle \text{ and } \rho''(k) = \texttt{true}] \qquad \neg\text{compat}(|\mathcal{U}|, |\mathcal{V}|, \mathcal{Q})\end{array}}{\left\langle \Sigma, \mathcal{S} \xrightarrow{\mathcal{Q} \textbf{ s.t. } f(\overline{x})} \mathcal{D} \right\rangle \to \langle \Sigma', \texttt{revert} \rangle} \text{ FLOW-FILTER-FAIL}$$

$$\frac{\begin{array}{c}\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \\ \rho'(\ell) = v, \mathcal{V} \qquad \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto \mathcal{V}]], f(\overline{x}, v) \rangle \to^* \langle \Sigma'', k \rangle\end{array}}{\langle \Sigma, \mathcal{S} \to f(\overline{x}) \to \mathcal{D} \rangle \to \langle (\mu', \rho''), \texttt{put}([\rho''(k)], \mathcal{D})\ (\mathcal{S} \to f(\overline{x}) \to \mathcal{D}) \rangle} \text{ FLOW-TRANSFORMER}$$

$$\frac{\texttt{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \qquad \rho'(\ell) = []}{\langle \Sigma, \mathcal{S} \to f(\overline{x}) \to \mathcal{D} \rangle \to \langle \Sigma, \texttt{put}([], \mathcal{D}) \rangle} \text{ FLOW-TRANSFORMER-DONE}$$

[NOTE: It is important that we flow an empty list in the FLOW-TRANSFORMER-DONE rule, otherwise we may fail to allocate a variable as expected.]

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

We introduce a new statement, $\texttt{try}(\Sigma, \overline{S_1}, \overline{S_2})$, 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 $\texttt{revert}$.

$$\frac{}{\left\langle \Sigma, \texttt{try}\ \{\overline{S_1}\}\ \texttt{catch}\ \{\overline{S_2}\} \right\rangle \to \left\langle \Sigma, \texttt{try}(\Sigma, \overline{S_1}, \overline{S_2}) \right\rangle} \text{ TRY-START}$$

$$\frac{\left\langle \Sigma, \overline{S_1} \right\rangle \to \left\langle \Sigma'', \overline{S_1'} \right\rangle}{\left\langle \Sigma, \texttt{try}(\Sigma', \overline{S_1}, \overline{S_2}) \right\rangle \to \left\langle \Sigma'', \texttt{try}(\Sigma', \overline{S_1'}, \overline{S_2}) \right\rangle} \text{ TRY-STEP}$$

$$\frac{}{\left\langle \Sigma, \texttt{try}(\Sigma', \texttt{revert}, \overline{S_2}) \right\rangle \to \left\langle \Sigma', \overline{S_2} \right\rangle} \text{ TRY-REVERT} \qquad \frac{}{\left\langle \Sigma, \texttt{try}(\Sigma', \cdot, \overline{S_2}) \right\rangle \to \Sigma} \text{ TRY-DONE}$$

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

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

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

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

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

$$\frac{\ell \notin \mathsf{dom}(\rho)}{\texttt{resolve}(\Sigma, n) = (\Sigma[\rho \mapsto \rho[\ell \mapsto n]], \ell)}\ \textsc{Resolve-Nat}$$

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

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

[TODO: Need to be sure to handle uniqueness correctly; could do this in Resolve-New-Source, or in the various flow rules.]

1.5 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* $\texttt{empty} < \texttt{any} < ! < \texttt{nonempty} < \texttt{every}$.

$\boxed{\mathrm{isAsset}(\overline{T_V}, \tau)}$ **Asset Types**

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

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

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

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

$$
\begin{array}{rcll}
\mathcal{Q} \oplus \mathtt{empty} & = & \mathcal{Q} & \\
\mathcal{Q} \oplus \mathtt{every} & = & \mathtt{every} & \\
\mathtt{nonempty} \oplus \mathcal{R} & = & \mathtt{nonempty} & \text{if } \mathtt{empty} < \mathcal{R} < \mathtt{every} \\
! \oplus \mathcal{R} & = & \mathtt{nonempty} & \text{if } \mathtt{empty} < \mathcal{R} < \mathtt{every} \\
\mathtt{any} \oplus \mathtt{any} & = & \mathtt{any} &
\end{array}
$$

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

$$
\begin{array}{rcll}
\mathcal{Q} \ominus \mathtt{empty} & = & \mathcal{Q} & \\
\mathtt{empty} \ominus \mathcal{R} & = & \mathtt{empty} & \\
\mathcal{Q} \ominus \mathtt{every} & = & \mathtt{empty} & \\
\mathtt{every} \ominus \mathcal{R} & = & \mathtt{every} & \text{if } \mathcal{R} < \mathtt{every} \\
\mathtt{nonempty} - \mathcal{R} & = & \mathtt{any} & \text{if } \mathtt{empty} < \mathcal{R} < \mathtt{every} \\
! - \mathcal{R} & = & \mathtt{empty} & \text{if } ! \leq \mathcal{R} \\
! - \mathtt{any} & = & \mathtt{any} & \\
\mathtt{any} - \mathcal{R} & = & \mathtt{any} & \text{if } \mathtt{empty} < \mathcal{R} < \mathtt{every}
\end{array}
$$

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:*

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

We write $\mathbf{dom}(\Gamma)$ 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*

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

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

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

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 $\Gamma \sqcup \Delta$.

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

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

$\boxed{\text{modifiers}(\overline{T_V}, T) = \overline{M}}$ **Type Modifiers**

$$\text{modifiers}(\overline{T_V}, T) = \begin{cases} \overline{M} & \text{if } T = \texttt{type}\ t\ \texttt{is}\ \overline{M}\ T' \\ \overline{M} & \text{if } (T\ \texttt{is}\ \overline{M}) \in \overline{T_V} \\ \emptyset & \text{otherwise} \end{cases}$$

$\boxed{\text{demote}(\tau) = \sigma}$ $\boxed{\text{demote}_*(T_1) = T_2}$ **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.

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

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

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

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

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

$\boxed{\text{compat}(n, m, \mathcal{Q})}$ The relation 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, $\mathcal{Q}$.

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

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

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