

# 1 Formalization

## 1.1 Syntax

$$\begin{aligned} f &\in \text{TRANSFORMER NAMES} & t &\in \text{TYPE NAMES} \\ a, x, y, z &\in \text{IDENTIFIERS} \end{aligned}$$

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

[Remove bool type? Can implement the “filter” selectors another way, e.g., by using a transformer returning a pair.]

## 1.2 Statics

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

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

$$\begin{aligned} &\frac{b \in \{\text{true}, \text{false}\}}{\Gamma \vdash b : ! \text{bool} \dashv \Gamma} \text{ BOOL} & \frac{}{\Gamma \vdash n : ! \text{nat} \dashv \Gamma} \text{ NAT} \\ &\frac{}{\Gamma, x : \tau \vdash \text{demote}(x) : \text{demote}(\tau) \dashv \Gamma, x : \tau} \text{ DEMOTE} & \frac{\text{immutable} \notin \text{modifiers}(\tau)}{\Gamma, x : \tau \vdash x : \tau \dashv \Gamma, x : \tau} \text{ VAR} \\ &\frac{\Gamma \vdash x : ! T \dashv \Delta \quad \text{immutable} \notin \text{modifiers}(\tau) \quad \text{fields}(T) = \overline{z} : \overline{\sigma} \quad y : \tau \in \overline{z} : \overline{\sigma}}{\Gamma \vdash x.y : \tau \dashv \Gamma} \text{ FIELD} \\ &\frac{}{\Gamma, x : \mathcal{Q} T \vdash [x] : ! \text{list } \mathcal{Q} T \dashv \Gamma, x : \text{empty } T} \text{ SINGLE} \\ &\frac{}{\Gamma, \overline{y} : \mathcal{Q} \overline{T} \vdash \{x : \mathcal{Q} \overline{T} \mapsto \overline{y}\} : ! \{x : \mathcal{Q} \overline{T}\} \dashv \Gamma, \overline{y} : \text{empty } \overline{T}} \text{ RECORD} \\ &\frac{}{\Gamma \vdash \text{new}(t, \overline{M}, T) : \text{every list } ! (\text{type } t \text{ is } \overline{M} T) \dashv \Gamma} \text{ NEW} \\ &\frac{}{\Gamma \vdash (\text{var } x : T) : \text{empty } T \dashv \Gamma, x : \text{empty } T} \text{ VARDEF} & \frac{\tau \text{ consumable}}{\Gamma \vdash \text{consume} : \tau \dashv \Gamma} \text{ CONSUME} \end{aligned}$$

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

$$\frac{}{\Gamma \vdash \text{skip ok} \dashv \Gamma} \text{OK-SKIP} \quad \frac{\Gamma \vdash S : \mathcal{Q} \ T \dashv \Delta \quad \text{update}(\Delta, S, \Delta(S) \ominus \mathcal{Q}) \vdash \mathcal{D} : \mathcal{R} \ T \dashv \Xi}{\Gamma \vdash (S \rightarrow \mathcal{D}) \text{ ok} \dashv \text{update}(\Xi, \mathcal{D}, \Xi(\mathcal{D}) \oplus \mathcal{Q})} \text{OK-FLOW-EVERY}$$

$$\frac{\Gamma \vdash S : \mathcal{Q} \ T \dashv \Delta \quad \Delta \vdash x : \text{demote}(\mathcal{R} \ T) \dashv \Delta \quad \text{update}(\Delta, S, \Delta(S) \ominus \mathcal{Q}) \vdash \mathcal{D} : S \ T \dashv \Xi}{\Gamma \vdash (S \xrightarrow{x} \mathcal{D}) \text{ ok} \dashv \text{update}(\Xi, \mathcal{D}, \Xi(\mathcal{D}) \oplus \mathcal{R})} \text{OK-FLOW-VAR}$$

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

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

$$\frac{\Gamma \vdash \overline{S_1} \text{ ok} \dashv \Delta \quad \Gamma \vdash \overline{S_2} \text{ ok} \dashv \Xi}{\Gamma \vdash (\text{try } \{ \overline{S_1} \} \text{ catch } \{ \overline{S_2} \}) \text{ ok} \dashv \Delta \sqcup \Xi} \text{OK-TRY}$$

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

$$\frac{\overline{x} : \overline{\tau} \vdash \overline{\text{Stmt}} \text{ ok} \dashv \Gamma, y : \sigma \quad \forall \pi \in \text{img}(\Gamma). \neg(\pi \text{ asset})}{\vdash (\text{transformer } f(\overline{x} : \overline{\tau}) \rightarrow y : \sigma \{ \overline{\text{Stmt}} \}) \text{ ok}} \text{OK-TRANSFORMER}$$

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

$$\frac{\vdash \overline{\text{Decl}} \text{ ok} \quad \emptyset \vdash \overline{\text{Stmt}} \text{ ok} \dashv \Gamma \quad \forall \tau \in \text{img}(\Gamma). \neg(\tau \text{ asset})}{(\overline{\text{Decl}}; \overline{\text{Stmt}}) \text{ ok}} \text{OK-PROG}$$

### 1.3 Dynamics

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

**Definition 1.** An environment  $\Sigma$  is a tuple  $(\mu, \rho)$  where  $\mu : \text{IDENTIFIERNAMES} \rightarrow \mathbb{N}$  is the variable lookup environment, and  $\rho : \mathbb{N} \rightarrow \mathcal{V}$  is the storage environment.

$$\boxed{\langle \Sigma, \overline{\text{Stmt}} \rangle \rightarrow \langle \Sigma, \overline{\text{Stmt}} \rangle}$$

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

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

$$\frac{\langle \Sigma, S_1 \rangle \rightarrow \langle \Sigma', \overline{S_3} \rangle}{\langle \Sigma, S_1 \overline{S_2} \rangle \rightarrow \langle \Sigma', \overline{S_3} \overline{S_2} \rangle} \text{SEQ} \quad \frac{}{\langle \Sigma, (\text{revert}) \overline{S} \rangle \rightarrow \langle \Sigma, \text{revert} \rangle} \text{REVERT} \quad \frac{}{\langle \Sigma, \text{skip} \rangle \rightarrow \Sigma} \text{SKIP}$$

Here we give the rules for the new  $\text{put}(\mathcal{V}, \mathcal{D})$  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.]**

$$\begin{array}{c}
\frac{}{\langle \Sigma, \text{put}(\mathcal{V}, \text{consume}) \rangle \rightarrow \Sigma} \text{PUT-CONSUME} \qquad \frac{}{\langle \Sigma, \text{put}(\mathcal{V}, x) \rangle \rightarrow \langle \Sigma, \text{put}(\mathcal{V}, \mu(x)) \rangle} \text{PUT-VAR} \\
\\
\frac{\rho(\mu(X)) = \overline{\{x : \tau \mapsto \ell\}} \quad (y : \sigma \mapsto k) \in \overline{x : \tau \mapsto \ell}}{\langle \Sigma, \text{put}(\mathcal{V}, x.y) \rangle \rightarrow \langle \Sigma, \text{put}(\mathcal{V}, k) \rangle} \text{PUT-FIELD} \\
\\
\frac{\rho(\ell) = \mathcal{W} \quad \mathcal{W} + \mathcal{V} \neq \text{revert}}{\langle \Sigma, \text{put}(\mathcal{V}, \ell) \rangle \rightarrow \Sigma[\rho \mapsto \rho[\ell \mapsto \mathcal{W} + \mathcal{V}]]} \text{PUT-LOC} \qquad \frac{\rho(\ell) = \mathcal{W} \quad \mathcal{W} + \mathcal{V} = \text{revert}}{\langle \Sigma, \text{put}(\mathcal{V}, \ell) \rangle \rightarrow \langle \Sigma, \text{revert} \rangle} \text{PUT-LOC-FAIL} \\
\\
\frac{\ell \notin \text{dom}(\rho)}{\langle \Sigma, \text{put}(\mathcal{V}, \text{var } A : T) \rangle \rightarrow \Sigma[\mu \mapsto \mu[A \mapsto \ell], \rho \mapsto \rho[\ell \mapsto \mathcal{V}]]} \text{PUT-VARDEF} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell)}{\langle \Sigma, \mathcal{S} \rightarrow \mathcal{D} \rangle \rightarrow \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto []]], \text{put}(\rho'(\ell), \mathcal{D}) \rangle} \text{FLOW-EVERY} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \quad \rho'(\ell) = \mathcal{V} \quad \rho'(\mu'(x)) = \mathcal{W} \quad \mathcal{W} \leq \mathcal{V}}{\langle \Sigma, \mathcal{S} \xrightarrow{x} \mathcal{D} \rangle \rightarrow \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto \mathcal{V} - \mathcal{W}]], \text{put}(\mathcal{W}, \mathcal{D}) \rangle} \text{FLOW-VAR} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \quad \rho'(\ell) = \mathcal{V} \quad \rho'(\mu'(x)) = \mathcal{W} \quad \mathcal{W} \not\leq \mathcal{V}}{\langle \Sigma, \mathcal{S} \xrightarrow{x} \mathcal{D} \rangle \rightarrow \langle \Sigma', \text{revert} \rangle} \text{FLOW-VAR-FAIL} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \quad \rho'(\ell) = \mathcal{V} \quad \mathcal{U} = [v \in \mathcal{V} \mid \langle \Sigma', f(\bar{x}, v) \rangle \rightarrow^* \langle \Sigma'', k \rangle \text{ and } \rho''(k) = \text{true}] \quad \text{compat}(|\mathcal{U}|, |\mathcal{V}|, \mathcal{Q})}{\langle \Sigma, \mathcal{S} \xrightarrow{\mathcal{Q} \text{ s.t. } f(\bar{x})} \mathcal{D} \rangle \rightarrow \langle \Sigma'[\rho' \mapsto \rho'[\ell \mapsto \rho'(\ell) - \mathcal{U}]], \text{put}(\mathcal{U}, \mathcal{D}) \rangle} \text{FLOW-FILTER} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \quad \rho'(\ell) = \mathcal{V} \quad \mathcal{U} = [v \in \mathcal{V} \mid \langle \Sigma', f(\bar{x}, v) \rangle \rightarrow^* \langle \Sigma'', k \rangle \text{ and } \rho''(k) = \text{true}] \quad \neg \text{compat}(|\mathcal{U}|, |\mathcal{V}|, \mathcal{Q})}{\langle \Sigma, \mathcal{S} \xrightarrow{\mathcal{Q} \text{ s.t. } f(\bar{x})} \mathcal{D} \rangle \rightarrow \langle \Sigma', \text{revert} \rangle} \text{FLOW-FILTER-FAIL} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \quad \rho'(\ell) = v, \mathcal{V} \quad \langle \Sigma'[\rho \mapsto \rho'[\ell \mapsto \mathcal{V}]], f(\bar{x}, v) \rangle \rightarrow^* \langle \Sigma'', k \rangle}{\langle \Sigma, \mathcal{S} \rightarrow f(\bar{x}) \rightarrow \mathcal{D} \rangle \rightarrow \langle (\mu', \rho''), \text{put}([\rho''(k)], \mathcal{D}) (\mathcal{S} \rightarrow f(\bar{x}) \rightarrow \mathcal{D}) \rangle} \text{FLOW-TRANSFORMER} \\
\\
\frac{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma', \ell) \quad \rho'(\ell) = []}{\langle \Sigma, \mathcal{S} \rightarrow f(\bar{x}) \rightarrow \mathcal{D} \rangle \rightarrow \langle \Sigma, \text{put}([], \mathcal{D}) \rangle} \text{FLOW-TRANSFORMER-DONE}
\end{array}$$

**[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.]**

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

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

$$\begin{aligned} & \frac{}{\langle \Sigma, \text{try } \{\overline{S_1}\} \text{ catch } \{\overline{S_2}\} \rangle \rightarrow \langle \Sigma, \text{try}(\Sigma, \overline{S_1}, \overline{S_2}) \rangle} \text{TRY-START} \\ & \frac{\langle \Sigma, \overline{S_1} \rangle \rightarrow \langle \Sigma'', \overline{S'_1} \rangle}{\langle \Sigma, \text{try}(\Sigma', \overline{S_1}, \overline{S_2}) \rangle \rightarrow \langle \Sigma'', \text{try}(\Sigma', \overline{S'_1}, \overline{S_2}) \rangle} \text{TRY-STEP} \\ & \frac{}{\langle \Sigma, \text{try}(\Sigma', \text{revert}, \overline{S_2}) \rangle \rightarrow \langle \Sigma', \overline{S_2} \rangle} \text{TRY-REVERT} \quad \frac{}{\langle \Sigma, \text{try}(\Sigma', \cdot, \overline{S_2}) \rangle \rightarrow \Sigma} \text{TRY-DONE} \end{aligned}$$

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

We use  $\text{resolve}(\Sigma, \mathcal{S})$  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).

$$\begin{aligned} & \frac{\mu(\mathcal{S}) = \ell}{\text{resolve}(\Sigma, \mathcal{S}) = (\Sigma, \ell)} \text{RESOLVE-VAR} \quad \frac{\rho(\mu(x)) = \{z:\tau \mapsto \ell\} \quad (y:\sigma \mapsto k) \in \overline{z:\tau \mapsto \ell}}{\text{resolve}(\Sigma, x.y) = (\Sigma, k)} \text{RESOLVE-FIELD} \\ & \frac{\ell \notin \text{dom}(\rho)}{\text{resolve}(\Sigma, [x]) = (\Sigma[\rho \mapsto \rho[\ell \mapsto \rho(\mu(x)), \mu(x) \mapsto []]], \ell)} \text{RESOLVE-SINGLE} \\ & \frac{k \notin \text{dom}(\rho) \cup \overline{\ell} \quad \overline{\ell \notin \text{dom}(\rho)} \quad \Sigma' = \Sigma[\rho \mapsto \rho[\mu(y) \mapsto [], \ell \mapsto \rho(\mu(y)), k \mapsto \{x:\tau \mapsto \ell\}]]}{\text{resolve}(\Sigma, \{\overline{x:\tau \mapsto y}\}) = (\Sigma', k)} \text{RESOLVE-RECORD} \\ & \frac{b \in \{\text{true}, \text{false}\} \quad \ell \notin \text{dom}(\rho)}{\text{resolve}(\Sigma, b) = (\Sigma[\rho \mapsto \rho[\ell \mapsto b]], \ell)} \text{RESOLVE-BOOL} \\ & \frac{\ell \notin \text{dom}(\rho)}{\text{resolve}(\Sigma, n) = (\Sigma[\rho \mapsto \rho[\ell \mapsto n]], \ell)} \text{RESOLVE-NAT} \\ & \frac{\mu(t) = \ell}{\text{resolve}(\Sigma, \text{new}(t, \overline{M}, T)) = (\Sigma, \ell)} \text{RESOLVE-SOURCE} \\ & \frac{t \notin \text{dom}(\mu) \quad \ell \notin \text{dom}(\rho)}{\text{resolve}(\Sigma, \text{new}(t, \overline{M}, T)) = (\Sigma'[\rho \mapsto \rho[\ell \mapsto \text{values}(T)], \mu \mapsto \mu[t \mapsto \ell]], \ell)} \text{RESOLVE-NEW-SOURCE} \end{aligned}$$

**[TODO: Need to be sure to handle uniqueness correctly; could do this in RESOLVE-NEW-SOURCE, or in the various flow rules.]**

## 1.4 Auxiliaries

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

### $\tau \text{ asset}$ Asset Types

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

### $\tau \text{ consumable}$ Consumable Types

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

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

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

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

Define the operator  $\ominus$ , called split, as the unique function  $\mathbf{Quant}^2 \rightarrow \mathbf{Quant}$  such that

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

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

**Definition 4.** We can consider a type environment  $\Gamma$  as a function  $\text{IDENTIFIERS} \rightarrow \text{TYPES} \cup \{\perp\}$  as follows:

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

We write  $\text{dom}(\Gamma)$  to mean  $\{x \in \text{IDENTIFIERS} \mid \Gamma(x) \neq \perp\}$ , and  $\Gamma|_X$  to mean the environment  $\{x : \tau \in \Gamma \mid x \in X\}$  (restricting the domain of  $\Gamma$ ).

**Definition 5.** Let  $\mathcal{Q}$  and  $\mathcal{R}$  be type quantities,  $T_{\mathcal{Q}}$  and  $T_{\mathcal{R}}$  base types, and  $\Gamma$  and  $\Delta$  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

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

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  $\Gamma$  and  $\Delta$  by  $\Gamma \sqcup \Delta$ .

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

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

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

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

$$\boxed{\text{demote}(\tau) = \sigma}$$

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

**Type Demotion** demote and demote<sub>\*</sub> 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.

$$\text{demote}(\mathcal{Q} T) = \mathcal{Q} \text{demote}_*(T)$$

$$\text{demote}_*(\text{bool}) = \text{bool}$$

$$\text{demote}_*(\text{nat}) = \text{nat}$$

$$\text{demote}_*({\overline{\{x : \tau\}}}) = {\overline{\{x : \text{demote}(\tau)\}}}$$

$$\text{demote}_*(\text{type } t \text{ is } \overline{M} T) = \text{demote}_*(T)$$

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

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

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

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

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

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

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

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