Obsidian Language Formalism

1 Language Overview

2 Obsidian Core Calculus Grammar

The grammar is shown in Figure **??**. In the grammar, assume $\overline{I}$ means a possibly empty sequence of identifiers I; also assume that all items in the sequence are distinct, so that "x, y, z" is a valid sequence, but "x, y, x" is not.

$x, y, z \in$ VARIABLES $\quad \ell \in$ INDIRECTREFS $\quad o \in$ OBJECTREFS $\quad f \in$ FIELDNAMES

$Ct ::= \texttt{contract}\ C\ \{\ \overline{Ct}\ \ \overline{St}\ \ \overline{\texttt{const}\ \tau\ f}\ \ \overline{\tau\ f}\ \}$

$St ::= \texttt{state}\ S\ \{\ \overline{\tau\ f}\ \ \overline{MethDecl\ MethBody}\ \}$

$p ::= \texttt{owned} \mid \texttt{readonly} \mid \texttt{shared}$

$P ::= x \mid P.f$

$\tau_s ::= C \mid C.S \mid P.C \mid P.C.S$

$\tau ::= p \rhd \tau_s \mid \texttt{pack\_to}[S, \tau]$

$TxLabel ::= \texttt{function} \mid \texttt{transaction}$

$MethDecl ::= TxLabel\ \tau\ m(\overline{\tau\ x}) \hookrightarrow \overline{S}$

$MethBody ::= \{\ e\ \}$

$$\begin{aligned} e ::= {} & x \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \texttt{new}\ C.S(\overline{x})\ \texttt{as}\ p \mid x.m(\overline{x}) \mid f.m(\overline{x}) \\ \mid {} & \texttt{throw} \mid \texttt{try}\ \{\ e\ \}\ \texttt{catch}\ \{\ e\ \} \\ \mid {} & \texttt{pack\_to}\ S(\overline{x})\ \texttt{returns}\ x \mid \texttt{unpack}\ \{\ e\ \} \mid \texttt{assert}\ x : \tau_s\ \texttt{in}\ e \\ \mid {} & \ell \mid \texttt{residual}(x) \mid \texttt{residual}(f) \end{aligned}$$

Fig. 1. Grammar

3 Static Semantics

Figure **??** defines auxiliary relations that are helpful in defining the typing relation; Figure **??** defines the typing relation for expressions; Figure **??** defines extra typing judgments to check fields, methods, and contracts.

The typing relation $\Delta \vdash_b e : \tau \dashv \Delta'$ is flow-sensitive: it outputs a typing context, in addition to giving a type to the expression e. The following invariant also holds for the typing relation: $x \in \mathsf{dom}(\Delta)$ if and only if $x \in \mathsf{dom}(\Delta')$. Informally, this says that the contexts Δ and Δ' have the exact same variables in them, only differing perhaps by the type assigned to those variables.

The boolean b in the typing judgment indicates whether the expression e should be typed as if it were inside an `unpack` statement: inside an `unpack`, $b = \mathsf{t}$, while outside of an unpack, $b = \mathsf{f}$. Some typing judgments are valid for only one or the other case (e.g. T-PACK and T-INV) but others are valid in either case (e.g. T-LET).

Types can either be of contract type or of a special `pack_to` type. Only the former sort of types, however, are allowed in the typing context. This is implicitly enforced by:

- T-LET - this rule requires the bound variable to be of contract type
- the method Ok judgment - this determines the initial value of the typing context when typechecking a method body, and prohibits `pack_to` types in the initial context
- the field Ok judgment - this ensures that fields aren't of type `pack_to`, and thus `pack_to` types cannot appear in the typing context via field unpacking.

It is assumed that the type rules have access to the helper function *lookup*, $fields$, and *methods*. These helper functions retrieve, respectively, the contract signature, the set of fields, and the set of methods of a contract given the contract's name. If the type specifies a specific typestate, all the methods and fields that are available in that state (including those defined in the contract as a whole) are retrieved. There is also $mutableFields$, which retrieves the subset of $fields$ that are not labeled as `const`.

We say that a context Δ is well-formed if each type in the context is well-formed with respect to the context: a context is well-formed if

4 Dynamic Semantics

Auxiliary definitions for the dynamic semantics are shown in Figure **??**. The small-step evaluation relation is defined in Figure **??**. We augment the set of expressions e in the runtime to make it easier to express the desired semantics for exceptions: it is assumed that whole programs do not make use of the new `try-catch` and method call constructs.

$\boxed{res(p)}$ Residual Permission

$res(\texttt{owned}) = \texttt{readonly} \qquad res(\texttt{readonly}) = \texttt{readonly} \qquad res(\texttt{shared}) = \texttt{shared}$

$\boxed{res(\tau)}$ Residual Type

$$res(p \rhd \tau_s) = res(p) \rhd \tau_s$$

$\boxed{res(\overline{x}, \Delta)}$ Residual Context

$$\frac{\Delta' = \Delta[x_i \mapsto res(\Delta(x_i))]}{res(\overline{x}, \Delta) = \Delta'}$$

$\boxed{rm(x, \Delta),\ rm(\overline{x}, \Delta)}$

$$\frac{p \neq \texttt{owned}}{rm(x, (\Delta, x : p \rhd \tau)) = \Delta} \qquad \frac{x \notin \mathsf{dom}(\Delta)}{rm(x, \Delta) = \Delta}$$

$rm(\varnothing, \Delta) = \Delta \qquad rm(\{x_1, \ldots, x_n, x_{n+1}\}, \Delta) = rm(\{x_1, \ldots, x_n\}, rm(x_{n+1}, \Delta))$

$\boxed{\tau <: \tau'}$ Subtyping

$$\tau <: \tau \qquad p \rhd C.S <: p \rhd C \qquad p \rhd P.C.S <: p \rhd P.C$$

$\boxed{trans(\overline{S}, \tau_s)}$

$trans(\{S\}, C) = C.S \qquad trans(\{\}, \tau_s) = \tau_s \qquad trans(\{S_2\}, C.S_1) = C.S_2$

$trans(\{S_1, S_2, \ldots\}, C.S) = C \qquad trans(\{S_1, S_2, \ldots\}, C) = C$

$\boxed{mergeable(\Delta; \Delta_1, \ldots, \Delta_n)}$

$$\frac{mergeable(\Delta; \Delta_1, \ldots, \Delta_n) \qquad \exists \tau'.(\forall i.\Delta_i(x) <: \tau')}{mergeable((\Delta, x : \tau); \Delta_1, \ldots, \Delta_n)} \qquad mergeable(\varnothing; \Delta_1, \ldots, \Delta_n)$$

$\boxed{merge(\Delta; \Delta_1, \ldots, \Delta_n)}$

$$\frac{mergeable(\Delta; \Delta_1, \ldots, \Delta_n)}{merge(\Delta; \Delta_1, \ldots, \Delta_n) = \{(x, \tau) \mid x \in \Delta\ \&\ \Delta_i(x) <: \tau\}}$$

Fig. 2. Auxiliary Relations

$\boxed{\Delta \vdash_b e : \tau \dashv \Delta'}$

$$\frac{\tau <: \tau' \qquad \Delta \vdash_b e : \tau \dashv \Delta'}{\Delta \vdash_b e : \tau' \dashv \Delta'}\text{T-SUB} \qquad \frac{}{\Delta, x : \tau \vdash_b x : \tau \dashv \Delta, x : res(\tau)}\text{T-VAR}$$

$$\frac{x \notin \Delta \qquad \Delta \vdash_b e_1 : p \rhd \tau_s \dashv \Delta_1 \qquad \Delta_1, x : p \rhd \tau_s \vdash_b e_2 : \tau \dashv \Delta_2}{\Delta \vdash_b \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau \dashv rm(x, \Delta_2)}\text{T-LET}$$

$$\frac{\begin{array}{c} managed(C) \Rightarrow \Delta(\texttt{this}) <: p' \rhd manager(C) \\ \overline{\tau\ f} = fields(C.S) \end{array}}{\Delta, \overline{x : \tau} \vdash_b \texttt{new}\ C.S(\overline{x})\ \texttt{as}\ p : p \rhd C.S \dashv \Delta}\text{T-NEW}$$

$$\frac{(\tau\ f) \in fields(\Delta(\texttt{this}))}{\Delta \vdash_{\mathsf{f}} \texttt{residual}(f) : res(\tau) \dashv \Delta}\text{T-RES-1}$$

$$\frac{}{\Delta, x : \tau \vdash_{\mathsf{t}} \texttt{residual}(x) : res(\tau) \dashv \Delta, x : \tau}\text{T-RES-2} \qquad \frac{}{\Delta \vdash_b \texttt{throw} : \tau \dashv \Delta'}\text{T-THR}$$

$$\frac{\big(TL\ \tau''\ m(\overline{\tau\ z}) \to \overline{S}\big) \in methods(\tau') \qquad p = \texttt{readonly} \Rightarrow TL = \texttt{function}}{\Delta, x : p \rhd \tau', \overline{y : \tau[z \mapsto y][\texttt{this} \mapsto x]} \vdash_{\mathsf{f}} x.m(\overline{y}) : \tau'' \dashv \Delta, x : p \rhd trans(\overline{S}, \tau')}\text{T-INV-1}$$

$$\frac{\begin{array}{c} (p \rhd \tau'\ f) \in fields(\Delta(\texttt{this})) \qquad trans(\overline{S}, \tau') <: \tau' \\ \big(TL\ \tau''\ m(\overline{\tau\ z}) \to \overline{S}\big) \in methods(\tau') \qquad p = \texttt{readonly} \Rightarrow TL = \texttt{function} \end{array}}{\Delta, \overline{y : \tau[z \mapsto y][\texttt{this} \mapsto \texttt{this}.f]} \vdash_{\mathsf{f}} f.m(\overline{y}) : \tau'' \dashv \Delta}\text{T-INV-2}$$

$$\frac{\begin{array}{c} \Delta(\texttt{this}) = p \rhd C.S_1 \qquad p \neq \texttt{readonly} \qquad \overline{\tau\ f} = mutableFields(C.S_1) \\ \forall i. f_i \notin \Delta \qquad \Delta, \overline{f : \tau} \vdash_{\mathsf{t}} e : \texttt{pack\_to}[S_2, \tau] \dashv \Delta' \end{array}}{\Delta \vdash_{\mathsf{f}} \texttt{unpack}\ \{e\} : \tau \dashv rm(\overline{f}, \Delta')}\text{T-UNPACK}$$

$$\frac{\overline{\tau\ f} = mutableFields(trans(S, \Delta(\texttt{this})))}{\Delta, y : \tau', \overline{x : \tau} \vdash_{\mathsf{t}} \texttt{pack\_to}\ S(\overline{x})\ \texttt{returns}\ y : \texttt{pack\_to}[S, \tau'] \dashv \Delta}\text{T-PACK}$$

$$\frac{\Delta \vdash_b e_1 : \tau \dashv \Delta_1 \qquad \Delta \vdash_b e_2 : \tau \dashv \Delta_2}{\Delta \vdash_b \texttt{try}\ \{e_1\}\ \texttt{catch}\ \{e_2\} : \tau \dashv merge(\Delta; \Delta_1, \Delta_2)}\text{T-TRY}$$

$$\frac{\Delta, x : p \rhd \tau_s' \vdash_t e \dashv \Delta'}{\Delta, x : p \rhd \tau_s \vdash_t \texttt{assert}\ x : \tau_s'\ \texttt{in}\ e \dashv \Delta'}\text{T-ASSERT}$$

Fig. 3. Statics

$\boxed{(\tau\ f)\ \mathsf{Ok}}$ Field Consistency

$$\frac{}{(\texttt{owned} \rhd \tau_s\ f)\ \mathsf{Ok}} \qquad \frac{}{(p \rhd C\ f)\ \mathsf{Ok}}$$

$\boxed{(\mathit{MethDecl}\ \mathit{MethBody})\ \mathsf{Ok\ in}\ C.S}$ Method Consistency

$$\frac{\mathsf{f}; \varnothing, \texttt{this} : \texttt{owned} \rhd C.S, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau \dashv \Delta \quad \forall x, \tau_s'.\ \Delta(x) = \texttt{owned} \rhd \tau_s' \ \text{ iff } \ x = \texttt{this}}{\texttt{transaction}\ \tau'\ m(\overline{\tau\ x}) \hookrightarrow \overline{S}\ \{e\}\ \mathsf{Ok\ in}\ C.S}$$

$$\frac{\mathsf{f}; \varnothing, \texttt{this} : \texttt{readonly} \rhd C.S, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau \dashv \Delta \quad \forall x, \tau_s'.\ \Delta(x) \neq \texttt{owned} \rhd \tau_s'}{\texttt{function}\ \tau'\ m(\overline{\tau\ x})\ \{e\}\ \mathsf{Ok\ in}\ C.S}$$

Fig. 4. Auxiliary Judgments

$$\mu \in \textsc{Locations} \rightharpoonup \textsc{Objects}$$

$$\rho \in \textsc{Variables} \rightharpoonup \textsc{Locations}$$

$$T ::= C.S \mid \ell.C.S$$

$$\textsc{Objects} = \{(T, f_{map}) \mid f_{map} \in \textsc{FieldNames} \rightharpoonup \textsc{Locations})$$

$$\mathit{field}(f, (T, f_{map})) = f_{map}(f)$$

$$\mathit{type}((T, f_{map})) = T$$

$$(T, f_{map})[f \mapsto \ell] = (T, f_{map}[f \mapsto \ell])$$

$$\ell \in \textsc{Locations}$$

$$e ::= \ldots \mid \texttt{try}(\mu)\ \{e_1\}\ \texttt{catch}\ \{e_2\} \mid \ell \mid \mathsf{Ex} \mid \texttt{call}(\rho)\ \{e\}$$

$$v ::= \ell \mid \texttt{pack\_to}\ S(\overline{\ell})\ \texttt{returns}\ \ell$$

$\boxed{\mathbb{E}, \mathbb{E}[e]}$ Evaluation Context, Substitution

$$\mathbb{E} ::= \square \mid \texttt{let}\ x = \mathbb{E}\ \texttt{in}\ e \mid \texttt{let}\ x = \ell\ \texttt{in}\ \mathbb{E} \mid \texttt{unpack}\ \{\,\mathbb{E}\,\} \mid \texttt{call}(\rho)\ \{\mathbb{E}\}$$

$$\square[e] = e$$

$$(\texttt{let}\ x = \mathbb{E}\ \texttt{in}\ e')[e] = (\texttt{let}\ x = \mathbb{E}[e]\ \texttt{in}\ e')$$

$$(\texttt{let}\ x = \ell\ \texttt{in}\ \mathbb{E})[e] = (\texttt{let}\ x = \ell\ \texttt{in}\ \mathbb{E}[e])$$

$$(\texttt{unpack}\ \{\mathbb{E}\})[e] = (\texttt{unpack}\ \{\mathbb{E}[e]\})$$

$$(\texttt{call}(\rho)\ \{\mathbb{E}\})[e] = (\texttt{call}(\rho)\ \{\mathbb{E}[e]\})$$

$\boxed{\mathit{context}(\mu, \rho, \mathbb{E})}$ Calculates a suitable ρ' for evaluation inside of $\mathbb{E}$

$$\mathit{context}(\mu, \rho, \square) = \rho \qquad \mathit{context}(\mu, \rho, \texttt{let}\ x = \mathbb{E}\ \texttt{in}\ e) = \mathit{context}(\mu, \rho, \mathbb{E})$$

$$\mathit{context}(\mu, \rho, \texttt{let}\ x = \ell\ \texttt{in}\ \mathbb{E}) = \mathit{context}(\mu, \rho[x \mapsto \ell], \mathbb{E})$$

$$\frac{\overline{\tau\ f} = \mathit{fields}(\mathit{type}(\mu(\rho(\texttt{this}))))) \qquad \rho' = \rho[f_1 \mapsto \mathit{field}(f_1, \mu(\rho(\texttt{this})))] \ldots [f_n \mapsto \mathit{field}(f_n, \mu(\rho(\texttt{this})))]}{\mathit{context}(\mu, \rho, \texttt{unpack}\ \{\,\mathbb{E}\,\}) = \mathit{context}(\mu, \rho' \backslash \{\texttt{this}\}, \mathbb{E})}$$

$$\mathit{context}(\mu, \rho, \texttt{call}(\rho')\ \{\mathbb{E}\}) = \mathit{context}(\mu, \rho', \mathbb{E})$$

Fig. 5. Auxiliary Definitions

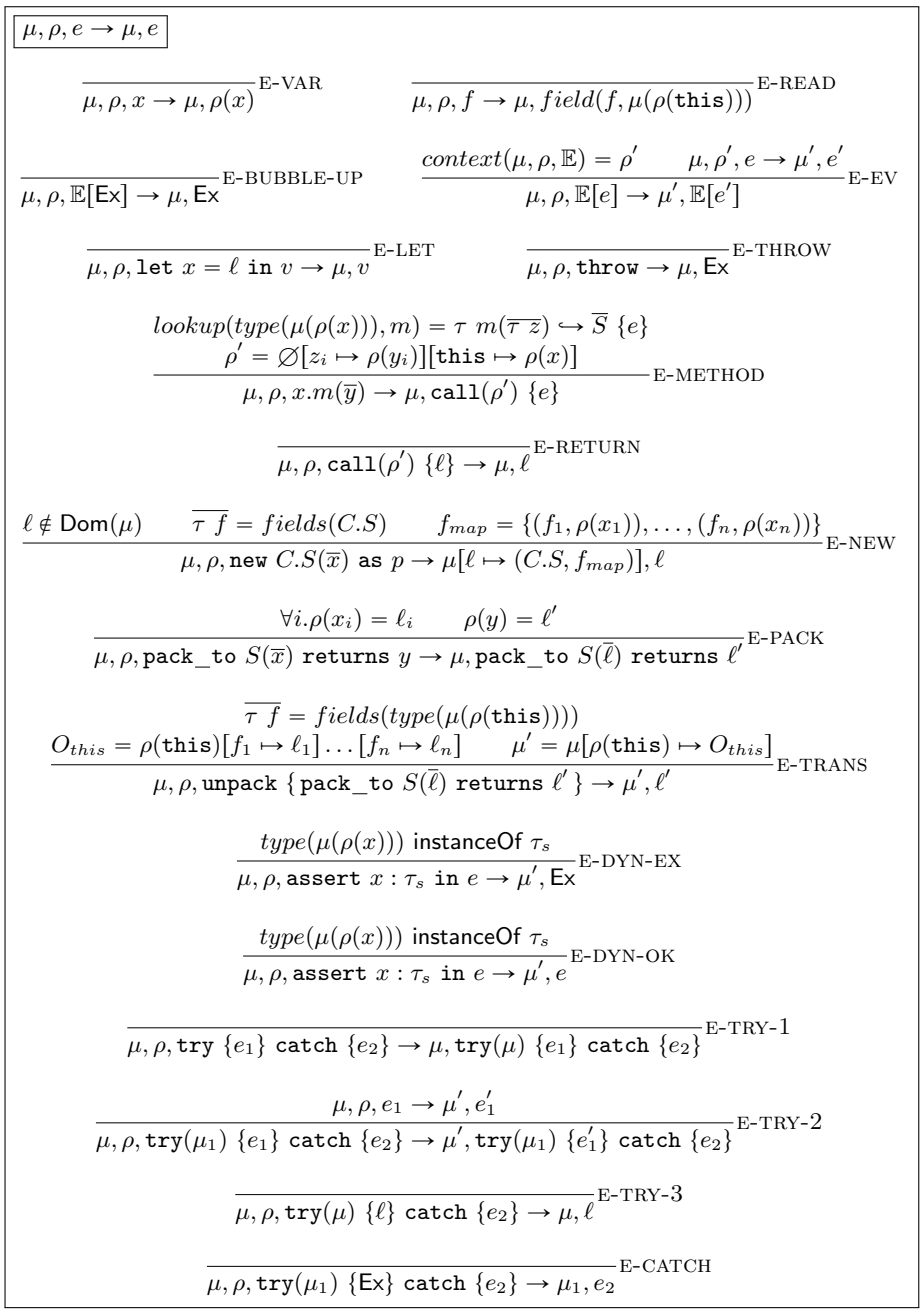

Fig. 6. Dynamics