

# Obsidian Language Formalism

## 1 Language Overview

## 2 Obsidian Core Calculus Grammar

The grammar is shown in Figure 1. In the grammar, assume  $\bar{I}$  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.

$x, y, z, f \in \text{VARIABLES}$	$\ell \in \text{INDIRECTREFS}$	$o \in \text{OBJECTREFS}$
$Ct ::= \text{contract } C \{ \overline{Ct} \ \overline{St} \ \overline{\text{const } \tau f} \ \overline{\tau f} \}$		
$St ::= \text{state } S \{ \overline{\tau f} \ \overline{\text{MethDecl } \text{MethBody}} \}$		
$p ::= \text{owned} \mid \text{readonly} \mid \text{shared}$		
$P ::= x.P \mid x$		
$\tau_s ::= C \mid C.S \mid P.C \mid P.C.S$		
$\tau ::= p \triangleright \tau_s \mid \text{pack\_to}[S, \tau]$		
$TxLabel ::= \text{function} \mid \text{transaction}$		
$\text{MethDecl} ::= TxLabel \ \tau \ m(\overline{\tau x}) \hookrightarrow \overline{S}$		
$\text{MethBody} ::= \{ e \}$		
$e ::= x \mid \text{let } x = e \text{ in } e \mid \text{new } C.S(\overline{x}) \text{ as } p \mid x.m(\overline{x}) \mid \text{res}(f)$		
$\mid \text{throw} \mid \text{try } \{ e \} \text{ catch } \{ e \}$		
$\mid \text{pack\_to } S(\overline{x}) \text{ returns } x \mid \text{unpack } \{ e \} \mid \text{if } x : \tau_s \text{ in } e$		
$\mid \ell$		

Fig. 1. Grammar

## 3 Static Semantics

Figure 2 defines auxiliary relations that are helpful in defining the typing relation; Figure 3 defines the typing relation for expressions; Figure 4 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:  $x \in \text{dom}(\Delta)$  if and only if  $x \in \text{dom}(\Delta')$ . Informally, this says that the contexts  $\Delta$  and  $\Delta'$  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**,  $b = \text{t}$ , while outside of an **unpack**,  $b = \text{f}$ . 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  $\Delta$  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 5. The small-step evaluation relation is defined in Figure 6. 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.

$\boxed{\text{res}(p), \text{res}(\tau), \text{res}(\bar{x}, \Delta)}$	Residual Permission/Type/Context
$\text{res}(\text{owned}) = \text{readonly} \quad \text{res}(\text{readonly}) = \text{readonly} \quad \text{res}(\text{shared}) = \text{shared}$	
$\text{res}(p \triangleright \tau_s) = \text{res}(p) \triangleright \tau_s \quad \frac{\Delta' = \Delta[x_i \mapsto \text{res}(\Delta(x_i))]}{\text{res}(\bar{x}, \Delta) = \Delta'}$	
$\boxed{\text{rm}(x, \Delta), \text{rm}(\bar{x}, \Delta)}$	
$\frac{p \neq \text{owned}}{\text{rm}(x, (\Delta, x : p \triangleright \tau)) = \Delta} \quad \frac{x \notin \text{dom}(\Delta)}{\text{rm}(x, \Delta) = \Delta}$	
$\text{rm}(\emptyset, \Delta) = \Delta \quad \text{rm}(\{x_1, \dots, x_n, x_{n+1}\}, \Delta) = \text{rm}(\{x_1, \dots, x_n\}, \text{rm}(x_{n+1}, \Delta))$	
$\boxed{\tau <: \tau'}$	Subtyping
$\tau <: \tau \quad p \triangleright C.S <: p \triangleright C \quad p \triangleright P.C.S <: p \triangleright P.C$	
$\boxed{\text{mergeable}(\Delta; \Delta_1, \dots, \Delta_n), \text{merge}(\Delta; \Delta_1, \dots, \Delta_n)}$	
$\frac{\text{mergeable}(\Delta; \Delta_1, \dots, \Delta_n) \quad \exists \tau'. (\forall i. \Delta_i(x) <: \tau')}{\text{mergeable}((\Delta, x : \tau); \Delta_1, \dots, \Delta_n)} \quad \text{mergeable}(\emptyset; \Delta_1, \dots, \Delta_n)$	
$\frac{\text{mergeable}(\Delta; \Delta_1, \dots, \Delta_n)}{\text{merge}(\Delta; \Delta_1, \dots, \Delta_n) = \{(x, \tau) \mid x \in \Delta \ \& \ \Delta_i(x) <: \tau\}}$	
$\boxed{\text{wf}(\Delta, C, \tau_s), \text{wf}(\Delta, C, \tau)}$	Wellformedness of types w.r.t. $\Delta$ and $C$
$\frac{\text{parent}(C) = D \quad \tau_b = C \vee \tau_b = C.S}{\text{wf}(\Delta, D, \tau_b)}$	
$\frac{\neg \text{hasParent}(C) \quad \tau_b = C \vee \tau_b = C.S}{\text{wf}(\Delta, D, \tau_b)} \quad \frac{\tau_b = C \vee \tau_b = C.S}{\text{wf}(\Delta, C, \tau_b)}$	
$\frac{\Delta(x) = C' \quad \text{wf}(\emptyset, C', P.\tau_b) \quad \tau_b = C \vee \tau_b = C.S}{\text{wf}(\Delta, D, x.P.\tau_b)}$	
$\frac{(\tau_f \ f) \in \text{constFields}(D) \quad \text{wf}(\emptyset, \text{contractOf}(\tau_f), P.\tau_b) \quad \tau_b = C \vee \tau_b = C.S}{\text{wf}(\Delta, D, f.P.\tau_b)}$	
$\boxed{\text{subst}(\tau, x : \tau', M)}$	Substitution for Types
$\frac{M(z) = y}{\text{subst}(p \triangleright z.\tau, x : \tau', M) = p \triangleright y.\tau} \quad \frac{z \notin \text{dom}(M)}{\text{subst}(p \triangleright z.\tau, x : \tau', M) = p \triangleright x.z.\tau}$	
$\frac{\neg \text{hasParent}(C) \quad \tau_b = C \vee \tau_b = C.S}{\text{subst}(p \triangleright \tau_b, x : \tau, M) = p \triangleright \tau_b}$	
$\frac{\text{hasParent}(C) \quad \text{parent}(C) = \text{contractOf}(\tau) \quad \tau_b = C \vee \tau_b = C.S}{\text{subst}(p \triangleright \tau_b, x : \tau, M) = p \triangleright x.\tau_b}$	
$\frac{\text{contractOf}(\tau_b) = C}{\text{subst}(p \triangleright C, x : P.\tau_b, M) = p \triangleright P.C} \quad \frac{\text{contractOf}(\tau_b) = C}{\text{subst}(p \triangleright C.S, x : P.\tau_b, M) = p \triangleright P.C.S}$	

Fig. 2. Auxiliary Relations

$\Delta \vdash_b e : \tau \dashv \Delta'$	
$\frac{\tau <: \tau' \quad \Delta \vdash_b e : \tau \dashv \Delta'}{\Delta \vdash_b e : \tau' \dashv \Delta'} \text{T-SUB}$	$\frac{}{\Delta, x : \tau \vdash_b x : \tau \dashv \Delta, x : \text{res}(\tau)} \text{T-VAR}$
$\frac{\forall P, \tau_b. \tau \neq x.P.\tau_b \quad \forall \tau. (\tau \ x) \notin \text{constFields}(\Delta(\text{this})) \quad x \notin \Delta \quad \Delta \vdash_b e_1 : p \triangleright \tau_s \dashv \Delta_1 \quad \Delta_1, x : p \triangleright \tau_s \vdash_b e_2 : \tau \dashv \Delta_2}{\Delta \vdash_b \text{let } x = e_1 \text{ in } e_2 : \tau \dashv \text{rm}(x, \Delta_2)} \text{T-LET}$	
$\frac{\text{hasParent}(C) \Rightarrow \exists p'. \Delta(\text{this}) <: p' \triangleright \text{parent}(C) \quad \overline{\tau} \ f = \text{fields}(C.S)}{\Delta, \bar{x} : \bar{\tau} \vdash_b \text{new } C.S(\bar{x}) \text{ as } p : p \triangleright C.S \dashv \Delta, x : \text{res}(\tau)} \text{T-NEW}$	
$\frac{(\tau \ f) \in \text{fields}(\Delta(\text{this}))}{\Delta \vdash_b \text{res}(f) : \text{res}(\tau) \dashv \Delta} \text{T-READ}$	$\frac{}{\Delta \vdash_b \text{throw} : \tau \dashv \Delta'} \text{T-THR}$
$\frac{M = \emptyset[\bar{z} \mapsto \bar{y}] \quad (TL \ \tau_{ret} \ m(\bar{\tau} \bar{z}) \rightarrow \bar{S}) \in \text{methods}(\tau_{recip}) \quad \tau'_{ret} = \text{subst}(\tau_{ret}, x : \tau_{recip}, M) \quad p = \text{readonly} \Rightarrow TL = \text{function} \quad \forall i. \tau'_i = \text{subst}(\tau_i, x : \tau_{recip}, M)}{\Delta, x : p \triangleright \tau_{recip}, \bar{y} : \bar{\tau}' \vdash_f x.m(\bar{y}) : \tau'_{ret} \dashv \Delta, x : p \triangleright \text{trans}(\bar{S}, \tau_{recip}), \bar{y} : \text{res}(\tau')} \text{T-INV}$	
$\frac{\overline{\tau} \ f = \text{fields}(C.S_1) \quad p \neq \text{readonly} \quad \forall i. f_i \notin \Delta \quad \Delta, \bar{f} : \tau \vdash_t e : \text{pack\_to}[S_2, \tau_{ret}] \dashv \Delta'}{\Delta, \text{this} : p \triangleright C.S_1 \vdash_f \text{unpack } \{e\} : \tau_{ret} \dashv \text{rm}(\bar{f}, \Delta'), \text{this} : p \triangleright C.S_2} \text{T-UNPACK}$	
$\frac{\overline{\tau_c} \ \bar{f}_c = \text{constFields}(\Delta(\text{this})) \quad \overline{\tau_m} \ \bar{f}_m = \text{mutableFields}(\text{trans}(S, \Delta(\text{this}))) \quad \Delta(f_{c_i}) <: \tau_{c_i} \quad \Delta' = \text{res}(\bar{f}_c, \Delta), \bar{y} : \text{res}(\tau'), \bar{x} : \text{res}(\tau_m)}{\Delta, \bar{y} : \tau', \bar{x} : \bar{\tau}_m \vdash_t \text{pack\_to } S(\bar{x}) \text{ returns } \bar{y} : \text{pack\_to}[S, \tau'] \dashv \Delta'} \text{T-PACK}$	
$\frac{\Delta \vdash_b e_1 : \tau \dashv \Delta_1 \quad \Delta \vdash_b e_2 : \tau \dashv \Delta_2}{\Delta \vdash_b \text{try } \{e_1\} \text{ catch } \{e_2\} : \tau \dashv \text{merge}(\Delta; \Delta_1, \Delta_2)} \text{T-TRY}$	
$\frac{\Delta, x : p \triangleright \tau'_s \vdash_b e_1 : \tau \dashv \Delta_1 \quad \Delta, x : p \triangleright \tau_s \vdash_b e_2 : \tau \dashv \Delta_2}{\Delta, x : p \triangleright \tau_s \vdash_t \text{if } x : \tau'_s \text{ then } e_1 \text{ else } e_2 \dashv \text{merge}(\Delta; \Delta_1, \Delta_2)} \text{T-DYN}$	

Fig. 3. Statics

$(\tau \ f) \text{ Ok}$	Field Consistency
$\overline{(\text{owned} \triangleright_{\tau_s} f) \text{ Ok}}$	$\overline{(p \triangleright_C f) \text{ Ok}}$
$(\text{MethDecl } \text{MethBody}) \text{ Ok in } C.S$	Method Consistency
$\frac{\text{f}; \emptyset, \text{this} : \text{owned} \triangleright C.S, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau \dashv \Delta}{\forall x, \tau'_s. \Delta(x) = \text{owned} \triangleright \tau'_s \text{ iff } x = \text{this}}$	
$\text{transaction } \tau' \ m(\overline{\tau \ x}) \leftrightarrow \overline{S} \ \{e\} \text{ Ok in } C.S$	
$\frac{\text{f}; \emptyset, \text{this} : \text{readonly} \triangleright C.S, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e : \tau \dashv \Delta}{\forall x, \tau'_s. \Delta(x) \neq \text{owned} \triangleright \tau'_s}$	
$\text{function } \tau' \ m(\overline{\tau \ x}) \ \{e\} \text{ Ok in } C.S$	

Fig. 4. Auxiliary Judgments

$\mu \in \text{LOCATIONS} \rightarrow \text{OBJECTS}$
$\rho \in \text{VARIABLES} \rightarrow \text{LOCATIONS}$
$T ::= C.S \mid \ell.C.S$
$\text{OBJECTS} = \{(T, f_{\text{map}}) \mid f_{\text{map}} \in \text{FIELDNAMES} \rightarrow \text{LOCATIONS}\}$
$\text{field}(f, (T, f_{\text{map}})) = f_{\text{map}}(f)$
$\text{type}((T, f_{\text{map}})) = T$
$(T, f_{\text{map}})[f \mapsto \ell] = (T, f_{\text{map}}[f \mapsto \ell])$
$\ell \in \text{LOCATIONS}$
$e ::= \dots \mid \text{try}(\mu) \{e_1\} \text{ catch } \{e_2\} \mid \ell \mid \text{Ex} \mid \text{call}(\rho) \{e\}$
$v ::= \ell \mid \text{pack\_to } S(\bar{\ell}) \text{ returns } \ell$
$\boxed{\mathbb{E}, \mathbb{E}[e]}$ Evaluation Context, Substitution
$\mathbb{E} ::= \square \mid \text{let } x = \mathbb{E} \text{ in } e \mid \text{let } x = \ell \text{ in } \mathbb{E} \mid \text{unpack } \{\mathbb{E}\} \mid \text{call}(\rho) \{\mathbb{E}\}$
$\square[e] = e$
$(\text{let } x = \mathbb{E} \text{ in } e')[e] = (\text{let } x = \mathbb{E}[e] \text{ in } e')$
$(\text{let } x = \ell \text{ in } \mathbb{E})[e] = (\text{let } x = \ell \text{ in } \mathbb{E}[e])$
$(\text{unpack } \{\mathbb{E}\})[e] = (\text{unpack } \{\mathbb{E}[e]\})$
$(\text{call}(\rho) \{\mathbb{E}\})[e] = (\text{call}(\rho) \{\mathbb{E}[e]\})$
$\boxed{\text{context}(\mu, \rho, \mathbb{E})}$ Calculates a suitable $\rho'$ for evaluation inside of $\mathbb{E}$
$\text{context}(\mu, \rho, \square) = \rho \quad \text{context}(\mu, \rho, \text{let } x = \mathbb{E} \text{ in } e) = \text{context}(\mu, \rho, \mathbb{E})$
$\text{context}(\mu, \rho, \text{let } x = \ell \text{ in } \mathbb{E}) = \text{context}(\mu, \rho[x \mapsto \ell], \mathbb{E})$
$\overline{\tau f} = \text{fields}(\text{type}(\mu(\rho(\text{this}))))$
$\frac{\rho' = \rho[f_1 \mapsto \text{field}(f_1, \mu(\rho(\text{this}))) \dots [f_n \mapsto \text{field}(f_n, \mu(\rho(\text{this})))]}{\text{context}(\mu, \rho, \text{unpack } \{\mathbb{E}\}) = \text{context}(\mu, \rho' \setminus \{\text{this}\}, \mathbb{E})}$
$\text{context}(\mu, \rho, \text{call}(\rho') \{\mathbb{E}\}) = \text{context}(\mu, \rho', \mathbb{E})$

Fig. 5. Auxiliary Definitions

$\mu, \rho, e \rightarrow \mu, e$	
$\frac{}{\mu, \rho, x \rightarrow \mu, \rho(x)} \text{E-VAR}$	$\frac{}{\mu, \rho, f \rightarrow \mu, \text{field}(f, \mu(\rho(\mathbf{this})))} \text{E-READ}$
$\frac{}{\mu, \rho, \mathbb{E}[\mathbf{Ex}] \rightarrow \mu, \mathbf{Ex}} \text{E-BUBBLE-UP}$	$\frac{\text{context}(\mu, \rho, \mathbb{E}) = \rho' \quad \mu, \rho', e \rightarrow \mu', e'}{\mu, \rho, \mathbb{E}[e] \rightarrow \mu', \mathbb{E}[e']} \text{E-EV}$
$\frac{}{\mu, \rho, \mathbf{let } x = \ell \text{ in } v \rightarrow \mu, v} \text{E-LET}$	$\frac{}{\mu, \rho, \mathbf{throw} \rightarrow \mu, \mathbf{Ex}} \text{E-THROW}$
$\frac{\text{lookup}(\text{type}(\mu(\rho(x))), m) = \tau \quad m(\bar{\tau} \bar{z}) \hookrightarrow \bar{S} \{e\} \quad \rho' = \emptyset[z_i \mapsto \rho(y_i)][\mathbf{this} \mapsto \rho(x)]}{\mu, \rho, x.m(\bar{y}) \rightarrow \mu, \text{call}(\rho') \{e\}} \text{E-METHOD}$	
$\frac{}{\mu, \rho, \text{call}(\rho') \{ \ell \} \rightarrow \mu, \ell} \text{E-RETURN}$	
$\frac{\ell \notin \text{Dom}(\mu) \quad \bar{\tau} \bar{f} = \text{fields}(C.S) \quad f_{\text{map}} = \{(f_1, \rho(x_1)), \dots, (f_n, \rho(x_n))\}}{\mu, \rho, \mathbf{new } C.S(\bar{x}) \text{ as } p \rightarrow \mu[\ell \mapsto (C.S, f_{\text{map}})], \ell} \text{E-NEW}$	
$\frac{\forall i. \rho(x_i) = \ell_i \quad \rho(y) = \ell'}{\mu, \rho, \mathbf{pack\_to } S(\bar{x}) \text{ returns } y \rightarrow \mu, \mathbf{pack\_to } S(\bar{\ell}) \text{ returns } \ell'} \text{E-PACK}$	
$\frac{\bar{\tau} \bar{f} = \text{fields}(\text{type}(\mu(\rho(\mathbf{this})))) \quad O_{\text{this}} = \rho(\mathbf{this})[f_1 \mapsto \ell_1] \dots [f_n \mapsto \ell_n] \quad \mu' = \mu[\rho(\mathbf{this}) \mapsto O_{\text{this}}]}{\mu, \rho, \mathbf{unpack } \{ \mathbf{pack\_to } S(\bar{\ell}) \text{ returns } \ell' \} \rightarrow \mu', \ell'} \text{E-TRANS}$	
$\frac{\text{type}(\mu(\rho(x))) \text{ instanceOf } \tau_s}{\mu, \rho, \mathbf{if } x : \tau_s \text{ in } e \rightarrow \mu', \mathbf{Ex}} \text{E-DYN-EX}$	$\frac{\text{type}(\mu(\rho(x))) \text{ instanceOf } \tau_s}{\mu, \rho, \mathbf{if } x : \tau_s \text{ in } e \rightarrow \mu', e} \text{E-DYN-OK}$
$\frac{}{\mu, \rho, \mathbf{try } \{e_1\} \text{ catch } \{e_2\} \rightarrow \mu, \mathbf{try}(\mu) \{e_1\} \text{ catch } \{e_2\}} \text{E-TRY-1}$	
$\frac{\mu, \rho, e_1 \rightarrow \mu', e'_1}{\mu, \rho, \mathbf{try}(\mu_1) \{e_1\} \text{ catch } \{e_2\} \rightarrow \mu', \mathbf{try}(\mu_1) \{e'_1\} \text{ catch } \{e_2\}} \text{E-TRY-2}$	
$\frac{}{\mu, \rho, \mathbf{try}(\mu) \{ \ell \} \text{ catch } \{e_2\} \rightarrow \mu, \ell} \text{E-TRY-3}$	
$\frac{}{\mu, \rho, \mathbf{try}(\mu_1) \{ \mathbf{Ex} \} \text{ catch } \{e_2\} \rightarrow \mu_1, e_2} \text{E-CATCH}$	

Fig. 6. Dynamics