Obsidian Language Formalism

1 Language Overview

2 Obsidian Core Calculus Grammar

The grammar is shown in Figure 1. Unless stated otherwise, 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.

$x, y, z \in \textsc{LocalVariables}$ $\quad$ $\texttt{this} \in \textsc{SpecialVariables}$ $\quad$ $f, g \in \textsc{FieldVariables}$

$\texttt{this}, a, x, y, z, f, g \in \textsc{Variables}$ $\quad$ $\ell \in \textsc{IndirectRefs}$ $\quad$ $o \in \textsc{ObjectRefs}$

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

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

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

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

$$\tau_p ::= \tau_s \mid a.\tau_p$$

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

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

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

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

$$\begin{aligned} e ::=\ & a \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \texttt{new}\ C.S(\overline{x})\ \texttt{as}\ p \mid a.m(\overline{x}) \\ \mid\ & \texttt{throw} \mid \texttt{try}\ \{\ e\ \}\ \texttt{catch}\ \{\ e\ \} \\ \mid\ & \texttt{pack\_to}\ S(\overline{x})\ \texttt{returns}\ y \mid \texttt{unpack}\ \{\ e\ \} \mid \texttt{if}\ x : \tau_s\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\ \mid\ & \ell \end{aligned}$$

Fig. 1. Grammar

There are several categories of variable names in the formalism. LocalVariables (denoted by x, y, z) are variables defined in the body of a method by a `let` expression, or by method arguments. FieldVariables (denoted by f, g) are variables used for fields, but they can sometimes enter the local context, for example during an unpack block. SpecialVariables includes only the special identifier `this`. Variables (denoted a, b) is a catch-all category for all the aforementioned variable types.

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: $x \in \mathsf{dom}(\Delta)$ 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 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.

$\boxed{\mathsf{res}(p), \mathsf{res}(\tau), \mathsf{res}(\overline{x}, \Delta)}$ Residual Permission/Type/Context

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

$$\mathsf{res}(p \rhd \tau_s) = \mathsf{res}(p) \rhd \tau_s \qquad \frac{\Delta' = \Delta[x_i \mapsto \mathsf{res}(\Delta(x_i))]}{\mathsf{res}(\overline{x}, \Delta) = \Delta'}$$

$\boxed{\mathsf{rm}(x, \Delta),\ \mathsf{rm}(\overline{x}, \Delta)}$ Removes variable x or variables $\overline{x}$ from the context Δ

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

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

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

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

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

Merges contexts $\Delta_1, \ldots, \Delta_n$ after branching from the original context Δ

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

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

$\boxed{\mathsf{wf}(\Delta, \tau), \mathsf{wf}(C, \tau)}$ Assures wellformedness of type τ w.r.t. Δ or C

$$\frac{\neg\mathsf{hasParent}(\mathsf{contractOf}(\tau_s))}{\mathsf{wf}(\Delta, p \rhd \tau_s)} \qquad \frac{\mathsf{contractOf}(\Delta(\texttt{this})) = \mathsf{contractOf}(\tau_s)}{\mathsf{wf}(\Delta, p \rhd \tau_s)}$$

$$\frac{\mathsf{contractOf}(\Delta(a)) = C \qquad \mathsf{wf}(\Delta, C, p \rhd \tau_p)}{\mathsf{wf}(\Delta, p \rhd a.\tau_p)} \qquad \frac{\begin{array}{c}(\tau_f\ f) \in \mathsf{constFields}(C)\\ \mathsf{wf}(\mathsf{contractOf}(\tau_f), p \rhd \tau_p)\end{array}}{\mathsf{wf}(C, p \rhd f.\tau_p)}$$

$$\frac{C = \mathsf{parent}(\mathsf{contractOf}(\tau_s))}{\mathsf{wf}(C, p \rhd \tau_s)}$$

$\boxed{\mathsf{subst}(\tau, \tau', M)}$ Changes perspective of τ using mapping M, with recipient type τ'

$$\frac{M(z) = y}{\mathsf{subst}(p \rhd z.\tau_p, \tau, M) = p \rhd y.\tau_p} \qquad \frac{\neg\mathsf{hasParent}(\mathsf{contractOf}(\tau_s))}{\mathsf{subst}(p \rhd \tau_s, \tau, M) = p \rhd \tau_s}$$

$$\frac{\mathsf{contractOf}(\tau_s) = \mathsf{contractOf}(\tau_{s'})}{\mathsf{subst}(p \rhd \tau_s, a.f_1 \cdots .f_n.\tau_s', M) = p \rhd a.f_1 \cdots .f_n.\tau_s}$$

Fig. 2. Auxiliary Relations

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

$$\frac{\tau <: \tau' \qquad \Delta \vdash_b e : \tau \dashv \Delta'}{\Delta \vdash_b e : \tau' \dashv \Delta'}\,\text{T-SUB} \qquad\qquad \frac{a \neq \texttt{this}}{\Delta, a : \tau \vdash_b a : \tau \dashv \Delta, a : \mathsf{res}(\tau)}\,\text{T-VAR}$$

$$\frac{\Delta(\texttt{this}) = \tau}{\Delta \vdash_{\mathsf{f}} \texttt{this} : \mathsf{res}(\tau) \dashv \Delta}\,\text{T-THIS} \qquad\qquad \frac{(\tau\ \ f) \in \mathsf{fields}(\Delta(\texttt{this}))}{\Delta \vdash_{\mathsf{f}} \texttt{res}(f) : \mathsf{res}(\tau) \dashv \Delta}\,\text{T-FIELD}$$

$$\frac{\begin{array}{c} \forall \tau_p.\ \tau \neq x.\tau_p \qquad \forall \tau.\ (\tau\ x) \notin \mathsf{constFields}(\Delta(\texttt{this})) \\ x \notin \Delta \qquad \Delta \vdash_b e_1 : p \rhd \tau_s \dashv \Delta_1 \qquad \Delta_1, x : p \rhd \tau_s \vdash_b e_2 : \tau \dashv \Delta_2 \end{array}}{\Delta \vdash_b \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau \dashv \mathsf{rm}(x, \Delta_2)}\,\text{T-LET}$$

$$\frac{\mathsf{hasParent}(C) \Rightarrow \exists p'.\ \Delta(\texttt{this}) <: p' \rhd \mathsf{parent}(C) \qquad \overline{\tau\ \ f} = \mathsf{fields}(C.S)}{\Delta, \overline{x : \tau} \vdash_b \texttt{new}\ C.S(\overline{x})\ \texttt{as}\ p : p \rhd C.S \dashv \Delta, \overline{x : \mathsf{res}(\tau)}}\,\text{T-NEW}$$

$$\frac{\begin{array}{c} M = \varnothing\overline{[z \mapsto y]}[\texttt{this} \mapsto x] \\ \left(TL\ \tau_{ret}\ m(\overline{\tau\ z}) \hookrightarrow \overline{S}\right) \in \mathsf{methods}(\tau_{recip}) \qquad \tau'_{ret} = \mathsf{subst}(\tau_{ret}, \tau_{recip}, M) \\ p = \texttt{readonly} \Rightarrow TL = \texttt{function} \qquad \forall i.\ \tau'_i = \mathsf{subst}(\tau_i, \tau_{recip}, M) \end{array}}{\Delta, x : p \rhd \tau_{recip}, \overline{y : \tau'} \vdash_{\mathsf{f}} x.m(\overline{y}) : \tau'_{ret} \dashv \Delta, x : p \rhd \mathsf{trans}(\overline{S}, \tau_{recip}), \overline{y : \mathsf{res}(\tau')}}\,\text{T-INV-1}$$

$$\frac{\begin{array}{c} M = \varnothing\overline{[z \mapsto x]}[\texttt{this} \mapsto \texttt{this}.f] \qquad (p \rhd \tau_{recip}\ \ f) \in \mathsf{fields}(\Delta(\texttt{this})) \\ \left(TL\ \tau_{ret}\ m(\overline{\tau\ z}) \hookrightarrow \overline{S}\right) \in \mathsf{methods}(\tau_{recip}) \\ p = \texttt{readonly} \Rightarrow TL = \texttt{function} \qquad \mathsf{trans}(\overline{S}, \tau_{recip}) <: \tau_{recip} \end{array}}{\Delta, \overline{x : \tau'} \vdash_{\mathsf{f}} f.m(\overline{x}) : \mathsf{subst}(\tau_{ret}, \tau_{recip}, M) \dashv \Delta, \overline{x : \mathsf{res}(\mathsf{subst}(\tau, \tau_{recip}, M))}}\,\text{T-INV-2}$$

$$\frac{\begin{array}{c} \overline{\tau\ f} = \mathsf{mutableFields}(\Delta(\texttt{this})) \\ p \neq \texttt{readonly} \qquad \Delta, \overline{f : \tau} \vdash_{\mathsf{t}} e : \texttt{pack\_to}[S, \tau_{ret}] \dashv \Delta' \end{array}}{\Delta \vdash_{\mathsf{f}} \texttt{unpack}\ \{e\} : \tau_{ret} \dashv \mathsf{rm}(\overline{f}, \Delta'), \texttt{this} : \mathsf{trans}(S, \Delta(\texttt{this}))}\,\text{T-UNPACK}$$

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

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

$$\frac{}{\Delta \vdash_b \texttt{throw} : \tau \dashv \Delta'}\,\text{T-THR}$$

$$\frac{\Delta, x : p \rhd \tau'_s \vdash_b e_1 : \tau \dashv \Delta_1 \qquad \Delta, x : p \rhd \tau_s \vdash_b e_2 : \tau \dashv \Delta_2}{\Delta, x : p \rhd \tau_s \vdash_t \texttt{if}\ x : \tau'_s\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \dashv \mathsf{merge}(\Delta; \Delta_1, \Delta_2)}\,\text{T-DYN}$$

Fig. 3. Statics

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

$$\frac{\tau = p \rhd a.f_1.\cdots.f_n.C \qquad \mathsf{wf}(\varnothing[\mathtt{this} \mapsto \mathtt{readonly} \rhd D], \tau)}{(\tau\ f)\ \mathsf{Ok\ in}\ D}$$

$$\frac{\tau = \mathtt{owned} \rhd a.f_1.\cdots.f_n.C.S \qquad \mathsf{wf}(\varnothing[\mathtt{this} \mapsto \mathtt{readonly} \rhd D], \tau)}{(\tau\ f)\ \mathsf{Ok\ in}\ D}$$

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

$$\frac{\begin{array}{c}\Delta = \varnothing, \mathtt{this} : \mathtt{owned} \rhd C.S, \overline{x : \tau} \qquad \forall i.\ \mathsf{wf}(\Delta, \tau_i) \\ \Delta \vdash_{\mathsf{f}} e : \tau' \dashv \Delta' \qquad \forall a, \tau_p.\ \Delta'(a) = \mathtt{owned} \rhd \tau_p \ \text{ iff } \ a = \mathtt{this} \\ \Delta'(\mathtt{this}) <: \mathsf{trans}(\overline{S}, \mathtt{owned} \rhd C.S)\end{array}}{\mathtt{transaction}\ \tau'\ m(\overline{\tau\ x}) \hookrightarrow \overline{S}\ \{e\}\ \mathsf{Ok\ in}\ C.S}$$

$$\frac{\begin{array}{c}\Delta = \varnothing, \mathtt{this} : \mathtt{readonly} \rhd C.S, \overline{x : \tau} \qquad \forall i.\ \mathsf{wf}(\Delta, \tau_i) \\ \Delta \vdash_{\mathsf{f}} e : \tau' \dashv \Delta' \qquad \forall a, \tau_p.\ \Delta'(a) \neq \mathtt{owned} \rhd \tau_p\end{array}}{\mathtt{function}\ \tau'\ m(\overline{\tau\ x})\ \{e\}\ \mathsf{Ok\ in}\ C.S}$$

$\boxed{\mathtt{state}\ S\ \{\ \ldots\ \}\ \mathsf{Ok\ in}\ C}$ State Consistency

$$\frac{\forall i.\ (\mathit{MethDecl}_i\ \mathit{MethBody}_i)\ \mathsf{Ok\ in}\ C.S \qquad \forall i.\ (\tau_i\ f_i)\ \mathsf{Ok\ in}\ C}{\mathtt{state}\ S\ \{\ \overline{\tau\ f}\ \ \overline{\mathit{MethDecl}\ \mathit{MethBody}}\ \}}$$

$\boxed{\mathtt{contract}\ C\ \{\ \ldots\ \}\ \mathsf{Ok}}$ Contract Consistency

$$\frac{\begin{array}{c}\forall i.\ (\tau_{f_i}\ f_i)\ \mathsf{Ok\ in}\ C \qquad \forall i.\ (\tau_{g_i}\ g_i)\ \mathsf{Ok\ in}\ C \qquad \forall i.\ St_i\ \mathsf{Ok\ in}\ C \qquad \forall i.\ Ct_i\ \mathsf{Ok} \\ \forall \tau, i, j.\ \big((\tau\ f_i) \notin \mathsf{fields}(St_j) \wedge (\tau\ g_i) \notin \mathsf{fields}(St_j) \wedge f_i \neq g_j\big)\end{array}}{\mathtt{contract}\ C\ \{\ \overline{Ct}\ \ \overline{St}\ \ \overline{\mathtt{const}\ \tau_f\ f}\ \ \overline{\tau_g\ g}\ \}}$$

Fig. 4. Auxiliary Judgments

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

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

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

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

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

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

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

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

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

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

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

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

$$\square[e] = e$$

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

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

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

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

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

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

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

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

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

Fig. 5. Auxiliary Definitions

$$\boxed{\mu, \rho, e \rightarrow \mu, e}$$

$$\frac{}{\mu, \rho, x \rightarrow \mu, \rho(x)}\text{E-VAR} \qquad \frac{}{\mu, \rho, f \rightarrow \mu, \mathit{field}(f, \mu(\rho(\texttt{this})))}\text{E-READ}$$

$$\frac{}{\mu, \rho, \mathbb{E}[\mathsf{Ex}] \rightarrow \mu, \mathsf{Ex}}\text{E-BUBBLE-UP} \qquad \frac{\mathit{context}(\mu, \rho, \mathbb{E}) = \rho' \qquad \mu, \rho', e \rightarrow \mu', e'}{\mu, \rho, \mathbb{E}[e] \rightarrow \mu', \mathbb{E}[e']}\text{E-EV}$$

$$\frac{}{\mu, \rho, \texttt{let}\ x = \ell\ \texttt{in}\ v \rightarrow \mu, v}\text{E-LET} \qquad \frac{}{\mu, \rho, \texttt{throw} \rightarrow \mu, \mathsf{Ex}}\text{E-THROW}$$

$$\frac{\begin{array}{c}\mathit{lookup}(\mathit{type}(\mu(\rho(x))), m) = \tau\ m(\overline{\tau\ z}) \hookrightarrow \overline{S}\ \{e\} \\ \rho' = \varnothing[z_i \mapsto \rho(y_i)][\texttt{this} \mapsto \rho(x)]\end{array}}{\mu, \rho, x.m(\overline{y}) \rightarrow \mu, \texttt{call}(\rho')\ \{e\}}\text{E-METHOD}$$

$$\frac{}{\mu, \rho, \texttt{call}(\rho')\ \{\ell\} \rightarrow \mu, \ell}\text{E-RETURN}$$

$$\frac{\ell \notin \mathsf{Dom}(\mu) \qquad \overline{\tau\ f} = \mathit{fields}(C.S) \qquad f_{map} = \{(f_1, \rho(x_1)), \ldots, (f_n, \rho(x_n))\}}{\mu, \rho, \texttt{new}\ C.S(\overline{x})\ \texttt{as}\ p \rightarrow \mu[\ell \mapsto (C.S, f_{map})], \ell}\text{E-NEW}$$

$$\frac{\forall i. \rho(x_i) = \ell_i \qquad \rho(y) = \ell'}{\mu, \rho, \texttt{pack\_to}\ S(\overline{x})\ \texttt{returns}\ y \rightarrow \mu, \texttt{pack\_to}\ S(\overline{\ell})\ \texttt{returns}\ \ell'}\text{E-PACK}$$

$$\frac{\begin{array}{c}\overline{\tau\ f} = \mathit{fields}(\mathit{type}(\mu(\rho(\texttt{this})))) \\ O_{this} = \rho(\texttt{this})[f_1 \mapsto \ell_1] \ldots [f_n \mapsto \ell_n] \qquad \mu' = \mu[\rho(\texttt{this}) \mapsto O_{this}]\end{array}}{\mu, \rho, \texttt{unpack}\ \{\ \texttt{pack\_to}\ S(\overline{\ell})\ \texttt{returns}\ \ell'\ \} \rightarrow \mu', \ell'}\text{E-TRANS}$$

$$\frac{\mathit{type}(\mu(\rho(x)))\ \mathsf{instanceOf}\ \tau_s}{\mu, \rho, \texttt{if}\ x : \tau_s\ \texttt{in}\ e \rightarrow \mu', \mathsf{Ex}}\text{E-DYN-EX} \qquad \frac{\mathit{type}(\mu(\rho(x)))\ \mathsf{instanceOf}\ \tau_s}{\mu, \rho, \texttt{if}\ x : \tau_s\ \texttt{in}\ e \rightarrow \mu', e}\text{E-DYN-OK}$$

$$\frac{}{\mu, \rho, \texttt{try}\ \{e_1\}\ \texttt{catch}\ \{e_2\} \rightarrow \mu, \texttt{try}(\mu)\ \{e_1\}\ \texttt{catch}\ \{e_2\}}\text{E-TRY-1}$$

$$\frac{\mu, \rho, e_1 \rightarrow \mu', e_1'}{\mu, \rho, \texttt{try}(\mu_1)\ \{e_1\}\ \texttt{catch}\ \{e_2\} \rightarrow \mu', \texttt{try}(\mu_1)\ \{e_1'\}\ \texttt{catch}\ \{e_2\}}\text{E-TRY-2}$$

$$\frac{}{\mu, \rho, \texttt{try}(\mu)\ \{\ell\}\ \texttt{catch}\ \{e_2\} \rightarrow \mu, \ell}\text{E-TRY-3}$$

$$\frac{}{\mu, \rho, \texttt{try}(\mu_1)\ \{\mathsf{Ex}\}\ \texttt{catch}\ \{e_2\} \rightarrow \mu_1, e_2}\text{E-CATCH}$$

Fig. 6. Dynamics