EnerJ: Approximate Data Types for Safe and General Low-Power Computation — Full Proofs

Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman
University of Washington, Department of Computer Science and Engineering

June 5, 2011

1 Type System

This report formalizes EnerJ, a programming language for supporting approximate computation [2]. The paper describing EnerJ gives an overview of the FEnerJ formalism, but here we describe the language in detail and prove a series of properties over the language.

This section introduces the core type system, which is made up of type qualifiers that extend Featherweight Java [1]. Section 2 describes the big-step operational semantics that define the language's runtime system. Section 3 proves a number of properties about the language, the most important of which is *non-interference* (intuitively, that the precise part of the program is unaffected by the approximate part). The appendices contain complete listings, generated by the Ott tool, of the language's grammar and definitions.[1]

1.1 Ordering

We introduce a strict ordering on the language's type qualifiers:

$\boxed{q <:_{\mathrm{q}} q'}$ ordering of precision qualifiers

$$\frac{q \neq \texttt{top}}{q <:_{\mathrm{q}} \texttt{lost}} \quad \text{QQ\_LOST}$$

$$\frac{}{q <:_{\mathrm{q}} \texttt{top}} \quad \text{QQ\_TOP}$$

$$\frac{}{q <:_{\mathrm{q}} q} \quad \text{QQ\_REFL}$$

Subclassing is standard:

$\boxed{C \sqsubseteq C'}$ subclassing

$$\frac{\texttt{class}\ \mathit{Cid}\ \texttt{extends}\ C'\ \{\ \_\ \_\ \} \in \mathit{Prg}}{\mathit{Cid} \sqsubseteq C'} \quad \text{SC\_DEF}$$

$$\frac{\texttt{class}\ C \ldots \in \mathit{Prg}}{C \sqsubseteq C} \quad \text{SC\_REFL}$$

$$\frac{C \sqsubseteq C_1 \qquad C_1 \sqsubseteq C'}{C \sqsubseteq C'} \quad \text{SC\_TRANS}$$

Subtyping combines these two and add a special case for primitives:

$\boxed{T <: T'}$ subtyping

[1] Ott: http://www.cl.cam.ac.uk/~pes20/ott/

$$\frac{q <:_{\mathrm{q}} q' \qquad C \sqsubseteq C'}{q\ C <:\ q'\ C'} \quad \text{ST\_REFT}$$

$$\frac{q <:_{\mathrm{q}} q'}{q\ P <:\ q'\ P} \quad \text{ST\_PRIMT1}$$

$$\frac{}{\texttt{precise}\ P <:\ \texttt{approx}\ P} \quad \text{ST\_PRIMT2}$$

We use the method ordering to express that we can replace a call of the sub-method by a call to the super-method, i.e. for our static method binding:

$\boxed{ms <:\ ms'}$ invocations of method ms can safely be replaced by calls to ms'

$$\frac{T' <:\ T \qquad \overline{T_k}^k <:\ \overline{T'_k}^k}{T\ m(\overline{T_k\ pid}^k)\ \texttt{precise} <:\ T'\ m(\overline{T'_k\ pid}^k)\ \texttt{approx}} \quad \text{MST\_DEF}$$

1.2 Adaptation

The `context` qualifier depends on the context and we need to adapt it, when the receiver changes, i.e. for field accesses and method calls.

We need to be careful and decide whether we can represent the new qualifier. If not, we use `lost`.

$\boxed{q \rhd q' = q''}$ combining two precision qualifiers

$$\frac{q'{=}\texttt{context}\ \wedge\ (q \in \{\texttt{approx}, \texttt{precise}, \texttt{context}\})}{q \rhd q' = q} \quad \text{QCQ\_CONTEXT}$$

$$\frac{q'{=}\texttt{context}\ \wedge\ (q \in \{\texttt{top}, \texttt{lost}\})}{q \rhd q' = \texttt{lost}} \quad \text{QCQ\_LOST}$$

$$\frac{q' {\neq} \texttt{context}}{q \rhd q' = q'} \quad \text{QCQ\_FIXED}$$

To combine whole types, we just need to adapt the qualifiers:

$\boxed{q \rhd T = T'}$ precision qualifier - type combination

$$\frac{q \rhd q' = q''}{q \rhd q'\ C = q''\ C} \quad \text{QCT\_REFT}$$

$$\frac{q \rhd q' = q''}{q \rhd q'\ P = q''\ P} \quad \text{QCT\_PRIMT}$$

Same for methods:

$\boxed{q \rhd ms = ms'}$ precision qualifier - method signature combination

$$\frac{q \rhd T = T' \qquad q \rhd \overline{T_k}^k = \overline{T'_k}^k}{q \rhd T\ m(\overline{T_k\ pid}^k)\ q' = T'\ m(\overline{T'_k\ pid}^k)\ q'} \quad \text{QCMS\_DEF}$$

1.3 Look-up Functions

The declared type of a field can be looked-up in the class declaration:

$\boxed{\text{FType}(C, f) = T}$ look up field f in class C

$$\frac{\texttt{class}\ \mathit{Cid}\ \texttt{extends}\ \_\ \{\ \_T\ f\texttt{;}\ \_\ \_\ \} \in \mathit{Prg}}{\text{FType}(\mathit{Cid}, f) = T}\ \text{SFTC\_DEF}$$

For a qualified class type, we also need to adapt the type:

$\boxed{\text{FType}(qC, f) = T}$ look up field f in reference type qC

$$\frac{\text{FType}(C, f) = T_1 \qquad q \rhd T_1 = T}{\text{FType}(q\ C, f) = T}\ \text{SFTT\_DEF}$$

Note that subsumption in the type rule will be used to get to the correct class that declares the field. Same for methods.

$\boxed{\text{MSig}(C, m, q) = ms}$ look up signature of method m in class C

$$\frac{\begin{array}{l}\texttt{class}\ \mathit{Cid}\ \texttt{extends}\ \_\ \{\ \_\ \_\ ms\ \{\ e\ \}\ \_\ \} \in \mathit{Prg}\\ \text{MName}(ms) = m\ \wedge\ \text{MQual}(ms) = q\end{array}}{\text{MSig}(\mathit{Cid}, m, q) = ms}\ \text{SMSC\_DEF}$$

$\boxed{\text{MSig}(qC, m) = ms}$ look up signature of method m in reference type qC

$$\frac{\text{MSig}(C, m, q) = ms \qquad q \rhd ms = ms'}{\text{MSig}(q\ C, m) = ms'}\ \text{SMST\_DEF}$$

1.4 Well-formedness

A well-formed expression:

$\boxed{{}^s\Gamma \vdash e : T}$ expression typing

$$\frac{{}^s\Gamma \vdash e : T_1 \qquad T_1 <: T}{{}^s\Gamma \vdash e : T}\ \text{TR\_SUBSUM}$$

$$\frac{qC\ \ \mathsf{OK}}{{}^s\Gamma \vdash \texttt{null} : qC}\ \text{TR\_NULL}$$

$$\frac{}{{}^s\Gamma \vdash \mathcal{L} : \texttt{precise}\ P}\ \text{TR\_LITERAL}$$

$$\frac{{}^s\Gamma(x) = T}{{}^s\Gamma \vdash x : T}\ \text{TR\_VAR}$$

$$\frac{\begin{array}{l}q\ C\ \ \mathsf{OK}\\ q \in \{\texttt{precise}, \texttt{approx}, \texttt{context}\}\end{array}}{{}^s\Gamma \vdash \texttt{new}\ q\ C\texttt{()} : T}\ \text{TR\_NEW}$$

$$\frac{{}^s\Gamma \vdash e_0 : q\ C \qquad \text{FType}(q\ C, f) = T}{{}^s\Gamma \vdash e_0.f : T}\ \text{TR\_READ}$$

$$\frac{\begin{array}{ll}{}^s\Gamma \vdash e_0 : q\ C & \text{FType}(q\ C, f) = T\\ \texttt{lost} \notin T & {}^s\Gamma \vdash e_1 : T\end{array}}{{}^s\Gamma \vdash e_0.f := e_1 : T}\ \text{TR\_WRITE}$$

$$\frac{\begin{array}{l} {}^s\Gamma \vdash e_0 : q\ C \qquad q \in \{\texttt{precise}, \texttt{context}, \texttt{top}\} \\ \text{MSig}(\texttt{precise}\ C, m) = T\ m(\overline{T_i\ pid}^{\,i})\ \texttt{precise} \\ \texttt{lost} \notin \overline{T_i}^{\,i} \qquad {}^s\Gamma \vdash \overline{e_i}^{\,i} : \overline{T_i}^{\,i} \end{array}}{{}^s\Gamma \vdash e_0 . m(\overline{e_i}^{\,i}) : T} \quad \text{TR\_CALL1}$$

$$\frac{\begin{array}{l} {}^s\Gamma \vdash e_0 : \texttt{approx}\ C \\ \text{MSig}(\texttt{approx}\ C, m) = T\ m(\overline{T_i\ pid}^{\,i})\ \texttt{approx} \\ \texttt{lost} \notin \overline{T_i}^{\,i} \qquad {}^s\Gamma \vdash \overline{e_i}^{\,i} : \overline{T_i}^{\,i} \end{array}}{{}^s\Gamma \vdash e_0 . m(\overline{e_i}^{\,i}) : T} \quad \text{TR\_CALL2}$$

$$\frac{\begin{array}{l} {}^s\Gamma \vdash e_0 : \texttt{approx}\ C \\ \text{MSig}(\texttt{approx}\ C, m) = \mathit{None} \\ \text{MSig}(\texttt{precise}\ C, m) = T\ m(\overline{T_i\ pid}^{\,i})\ \texttt{precise} \\ \texttt{lost} \notin \overline{T_i}^{\,i} \qquad {}^s\Gamma \vdash \overline{e_i}^{\,i} : \overline{T_i}^{\,i} \end{array}}{{}^s\Gamma \vdash e_0 . m(\overline{e_i}^{\,i}) : T} \quad \text{TR\_CALL3}$$

$$\frac{{}^s\Gamma \vdash e : \_ \qquad q\ C\ \ \mathsf{OK}}{{}^s\Gamma \vdash (q\ C)\ e : T} \quad \text{TR\_CAST}$$

$$\frac{{}^s\Gamma \vdash e_0 : q\ P \qquad {}^s\Gamma \vdash e_1 : q\ P}{{}^s\Gamma \vdash e_0 \oplus e_1 : q\ P} \quad \text{TR\_PRIMOP}$$

$$\frac{{}^s\Gamma \vdash e_0 : \texttt{precise}\ P \qquad {}^s\Gamma \vdash e_1 : T \qquad {}^s\Gamma \vdash e_2 : T}{{}^s\Gamma \vdash \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} : T} \quad \text{TR\_COND}$$

Note how `lost` is used to forbid invalid field updates and method calls.
Well-formed types:

$\boxed{T\ \ \mathsf{OK}}$ well-formed type

$$\frac{\texttt{class}\ C \ldots \in \mathit{Prg}}{q\ C\ \ \mathsf{OK}} \quad \text{WFT\_REFT}$$

$$\frac{}{q\ P\ \ \mathsf{OK}} \quad \text{WFT\_PRIMT}$$

Well-formed classes just propagate the checks and ensure the superclass is valid:

$\boxed{\mathit{Cls}\ \ \mathsf{OK}}$ well-formed class declaration

$$\frac{\begin{array}{l} {}^s\Gamma = \{\texttt{this} \mapsto \texttt{context}\ \mathit{Cid}\} \\ {}^s\Gamma \vdash \overline{\mathit{fd}}\ \ \mathsf{OK} \qquad {}^s\Gamma, \mathit{Cid} \vdash \overline{\mathit{md}}\ \ \mathsf{OK} \\ \texttt{class}\ C \ldots \in \mathit{Prg} \end{array}}{\texttt{class}\ \mathit{Cid}\ \texttt{extends}\ C\ \{\ \overline{\mathit{fd}}\ \overline{\mathit{md}}\ \}\ \ \mathsf{OK}} \quad \text{WFC\_DEF}$$

$$\frac{}{\texttt{class Object}\ \{\}\ \ \mathsf{OK}} \quad \text{WFC\_OBJECT}$$

Fields just check their types:

$\boxed{{}^s\Gamma \vdash T\ f;\ \ \mathsf{OK}}$ well-formed field declaration

$$\frac{T\ \ \mathsf{OK}}{{}^s\Gamma \vdash T\ f;\ \ \mathsf{OK}} \quad \text{WFFD\_DEF}$$

Methods check their type, the body expression, overriding, and the method qualifier:

$\boxed{{}^s\Gamma, C \vdash \mathit{md}\ \ \mathsf{OK}}$ well-formed method declaration

$$\frac{\begin{array}{l} {}^s\Gamma = \{\texttt{this} \mapsto \texttt{context}\ C\} \\ {}^s\Gamma' = \left\{\texttt{this} \mapsto \texttt{context}\ C,\ \overline{pid \mapsto T_i}^{\,i}\right\} \\ T,\ \overline{T_i}^{\,i}\ \ \mathsf{OK} \qquad {}^s\Gamma' \vdash e : T \qquad C \vdash m\ \ \mathsf{OK} \\ q \in \{\texttt{precise}, \texttt{approx}\} \end{array}}{{}^s\Gamma, C \vdash T\ m(\overline{T_i\ pid}^{\,i})\ q\ \{\ e\ \}\ \ \mathsf{OK}} \quad \text{WFMD\_DEF}$$

Overriding checks for all supertypes C' that a helper judgment holds:

$\boxed{C \vdash m\ \ \mathsf{OK}}$ method overriding OK

$$\frac{C \sqsubseteq C' \Longrightarrow C, C' \vdash m\ \ \mathsf{OK}}{C \vdash m\ \ \mathsf{OK}} \quad \text{OVR\_DEF}$$

This helper judgment ensures that if both methods are of the same precision, the signatures are equal. For a precise method we allow an approximate version that has relaxed types:

$\boxed{C, C' \vdash m\ \ \mathsf{OK}}$ method overriding OK auxiliary

$$\frac{\begin{array}{l} \text{MSig}(C, m, \texttt{precise}) = ms_0 \ \wedge\ \text{MSig}(C', m, \texttt{precise}) = ms_0' \ \wedge\ (ms_0' = None \ \vee\ ms_0 = ms_0') \\ \text{MSig}(C, m, \texttt{approx}) = ms_1 \ \wedge\ \text{MSig}(C', m, \texttt{approx}) = ms_1' \ \wedge\ (ms_1' = None \ \vee\ ms_1 = ms_1') \\ \text{MSig}(C, m, \texttt{precise}) = ms_2 \ \wedge\ \text{MSig}(C', m, \texttt{approx}) = ms_2' \ \wedge\ (ms_2' = None \ \vee\ ms_2 <:\ ms_2') \end{array}}{C, C' \vdash m\ \ \mathsf{OK}} \quad \text{OVRA\_DEF}$$

An environment simply checks all types:

$\boxed{{}^s\Gamma\ \ \mathsf{OK}}$ well-formed static environment

$$\frac{\begin{array}{l} {}^s\Gamma = \left\{\texttt{this} \mapsto q\ C,\ \overline{pid \mapsto T_i}^{\,i}\right\} \\ q\ C,\ \overline{T_i}^{\,i}\ \ \mathsf{OK} \end{array}}{{}^s\Gamma\ \ \mathsf{OK}} \quad \text{SWFE\_DEF}$$

Finally, a program checks the contained classes, the main expression and type, and ensures that the subtyping hierarchy is a-cyclic:

$\boxed{\vdash Prg\ \ \mathsf{OK}}$ well-formed program

$$\frac{\begin{array}{l} Prg = \overline{Cls_i}^{\,i},\ C,\ e \\ \quad \overline{Cls_i\ \ \mathsf{OK}}^{\,i} \qquad\qquad \texttt{context}\ C\ \ \mathsf{OK} \\ \{\texttt{this} \mapsto \texttt{context}\ C\} \vdash e : \_ \\ \forall C', C''.\ ((C' \sqsubseteq C''\ \wedge\ C'' \sqsubseteq C') \Longrightarrow C' = C'') \end{array}}{\vdash Prg\ \ \mathsf{OK}} \quad \text{WFP\_DEF}$$

2 Runtime System

2.1 Helper Functions

$\boxed{h + o\ =\ (h', \iota)}$ add object o to heap h resulting in heap h' and fresh address ι

$$\frac{\iota \notin \text{dom}(h) \qquad h' = h \oplus (\iota \mapsto o)}{h + o\ =\ (h', \iota)} \quad \text{HNEW\_DEF}$$

$\boxed{h[\iota.f\ :=\ v] = h'}$ field update in heap

$$\frac{\begin{array}{c} v{=}\texttt{null}_a \;\lor\; (v{=}\iota' \;\land\; \iota' \in \mathrm{dom}(h)) \\ h(\iota) = \left(T, \overline{fv}\right) \qquad f \in \mathrm{dom}\left(\overline{fv}\right) \qquad \overline{fv}'{=}\overline{fv}[f \mapsto v] \\ h'{=}h \oplus \left(\iota \mapsto \left(T, \overline{fv}'\right)\right) \end{array}}{h[\iota.f \;:=\; v] \;=\; h'} \quad \text{HUP\_REFT}$$

$$\frac{\begin{array}{c} h(\iota) = \left(T, \overline{fv}\right) \qquad \overline{fv}(f) = (q', {}^r\mathcal{L}') \\ \overline{fv}'{=}\overline{fv}[f \mapsto (q', {}^r\mathcal{L})] \qquad h'{=}h \oplus \left(\iota \mapsto \left(T, \overline{fv}'\right)\right) \end{array}}{h[\iota.f \;:=\; (q, {}^r\mathcal{L})] \;=\; h'} \quad \text{HUP\_PRIMT}$$

2.2 Runtime Typing

In the runtime system we only have `precise` and `approx`. The `context` qualifier is substituted by the correct concrete qualifiers. The `top` and `lost` qualifiers are not needed at runtime.

This function replaces `context` qualifier by the correct qualifier from the environment:

$\boxed{\mathrm{sTrT}(h, \iota, T) \;=\; T'}$ convert type T to its runtime equivalent T'

$$\frac{\begin{array}{c} q{=}\texttt{context} \implies q'{=}\mathrm{TQual}(h(\iota)\downarrow_1) \\ q{\neq}\texttt{context} \implies q'{=}q \end{array}}{\mathrm{sTrT}(h, \iota, q\ C) \;=\; q'\ C} \quad \text{sTrT\_REFT}$$

$$\frac{\begin{array}{c} q{=}\texttt{context} \implies q'{=}\mathrm{TQual}(h(\iota)\downarrow_1) \\ q{\neq}\texttt{context} \implies q'{=}q \end{array}}{\mathrm{sTrT}(h, \iota, q\ P) \;=\; q'\ P} \quad \text{sTrT\_PRIMT}$$

We can assign a type to a value, relative to a current object ι. For a reference type, we look up the concrete type in the heap, determine the runtime representation of the static type, and ensure that the latter is a subtype of the former. The null value can be assigned an arbitrary type. And for primitive values we ensure that the runtime version of the static type is a supertype of the concrete type.

$\boxed{h, \iota \vdash v :\ T}$ type T assignable to value v

$$\frac{\begin{array}{c} \mathrm{sTrT}(h, \iota_0, q\ C) \;=\; q'\ C \\ h(\iota)\downarrow_1 = T_1 \qquad T_1 <:\ q'\ C \end{array}}{h, \iota_0 \vdash \iota :\ q\ C} \quad \text{RTT\_ADDR}$$

$$\frac{}{h, \iota_0 \vdash \texttt{null}_a :\ q\ C} \quad \text{RTT\_NULL}$$

$$\frac{\begin{array}{c} \mathrm{sTrT}(h, \iota_0, q'\ P) \;=\; q''\ P \\ {}^r\mathcal{L} \in P \qquad q\ P <:\ q''\ P \end{array}}{h, \iota_0 \vdash (q, {}^r\mathcal{L}) :\ q'\ P} \quad \text{RTT\_PRIMT}$$

2.3 Look-up Functions

Look-up a field of an object at a given address. Note that subtyping allows us to go to the class that declares the field:

$\boxed{\mathrm{FType}(h, \iota, f) \;=\; T}$ look up type of field in heap

$$\frac{h, \iota \vdash \iota :\ q\ C \qquad \mathrm{FType}(q\ C, f) \;=\; T}{\mathrm{FType}(h, \iota, f) \;=\; T} \quad \text{RFT\_DEF}$$

Look-up the method signature of a method at a given address. Subtyping again allows us to go to any one of the possible multiple definitions of the methods. In a well-formed class, all these methods are equal:

$\boxed{\mathrm{MSig}(h, \iota, m) \;=\; ms}$ look up method signature of method m at ι

$$\frac{h, \iota \vdash \iota :\ q\ C \qquad \text{MSig}(q\ C, m)\ =\ ms}{\text{MSig}(h, \iota, m)\ =\ ms} \quad \text{RMS\_DEF}$$

For the method body, we need the most concrete implementation. This first function looks for a method with the given name and qualifier in the given class and in sequence in all super classes:

$\boxed{\text{MBody}(C, m, q)\ =\ e}$ look up most-concrete body of m, q in class C or a superclass

$$\frac{\begin{array}{l}\texttt{class}\ Cid\ \texttt{extends}\ \_\ \{\ \_\ \_\ ms\ \{\ e\ \}\ \_\ \} \in Prg \\ \text{MName}(ms) = m\ \wedge\ \text{MQual}(ms) = q\end{array}}{\text{MBody}(Cid, m, q)\ =\ e} \quad \text{SMBC\_FOUND}$$

$$\frac{\begin{array}{l}\texttt{class}\ Cid\ \texttt{extends}\ C_1\ \{\ \_\ \overline{ms_n\ \{\ e_n\ \}}^{\,n}\ \} \in Prg \\ \overline{\text{MName}(ms_n) \neq m}^{\,n} \qquad \text{MBody}(C_1, m, q)\ =\ e\end{array}}{\text{MBody}(Cid, m, q)\ =\ e} \quad \text{SMBC\_INH}$$

To look up the most concrete implementation for a method at a given address, we have three cases to consider. If it's a precise method, look it up. If it's an approximate method, try to find an approximate method. If you are looking for an approximate method, but couldn't find one, try to look for a precise methods:

$\boxed{\text{MBody}(h, \iota, m)\ =\ e}$ look up most-concrete body of method m at ι

$$\frac{h(\iota)\downarrow_1 = \texttt{precise}\ C \qquad \text{MBody}(C, m, \texttt{precise})\ =\ e}{\text{MBody}(h, \iota, m)\ =\ e} \quad \text{RMB\_CALL1}$$

$$\frac{h(\iota)\downarrow_1 = \texttt{approx}\ C \qquad \text{MBody}(C, m, \texttt{approx})\ =\ e}{\text{MBody}(h, \iota, m)\ =\ e} \quad \text{RMB\_CALL2}$$

$$\frac{\begin{array}{l}h(\iota)\downarrow_1 = \texttt{approx}\ C \qquad \text{MBody}(C, m, \texttt{approx})\ =\ None \\ \text{MBody}(C, m, \texttt{precise})\ =\ e\end{array}}{\text{MBody}(h, \iota, m)\ =\ e} \quad \text{RMB\_CALL3}$$

Get the field values corresponding to a given reference type. For fields of reference type, just use the null value. For fields of a primitive type, we need to look up the declared type of the field in order to determine the correct qualifier for the value.

$\boxed{\text{FVsInit}(qC)\ =\ \overline{fv}}$ initialize the fields for reference type qC

$$\frac{\begin{array}{l}q \in \{\texttt{precise}, \texttt{approx}\} \\ \forall f \in \text{refFields}(C)\,.\ \overline{fv}(f) = \texttt{null}_a \\ \forall f \in \text{primFields}(C)\,.\ (\text{FType}(q\ C, f)\ =\ q'\ P\ \wedge\ \overline{fv}(f) = (q', 0))\end{array}}{\text{FVsInit}(q\ C)\ =\ \overline{fv}} \quad \text{FVSI\_DEF}$$

2.4 Semantics

The standard semantics of our programming language:

$\boxed{{}^{r}\Gamma \vdash h, e\ \rightsquigarrow\ h', v}$ big-step operational semantics

$$\frac{}{{}^{r}\Gamma \vdash h, \texttt{null}\ \rightsquigarrow\ h, \texttt{null}_a} \quad \text{OS\_NULL}$$

$$\frac{}{{}^{r}\Gamma \vdash h, \mathcal{L}\ \rightsquigarrow\ h, (\texttt{precise}, {}^{r}\mathcal{L})} \quad \text{OS\_LITERAL}$$

$$\frac{{}^{r}\Gamma(x) = v}{{}^{r}\Gamma \vdash h, x\ \rightsquigarrow\ h, v} \quad \text{OS\_VAR}$$

$$\frac{\begin{array}{l}\text{sTrT}(h, {}^{r}\Gamma(\texttt{this}), q\ C) = q'\ C \\ \text{FVsInit}(q'\ C) = \overline{fv} \\ h + \left(q'\ C, \overline{fv}\right) = (h', \iota)\end{array}}{{}^{r}\Gamma \vdash h, \texttt{new}\ q\ C\texttt{()} \rightsquigarrow h', \iota} \quad \text{OS\_NEW}$$

$$\frac{{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h', \iota_0 \qquad h'(\iota_0\texttt{.}f) = v}{{}^{r}\Gamma \vdash h, e_0\texttt{.}f \rightsquigarrow h', v} \quad \text{OS\_READ}$$

$$\frac{\begin{array}{l}{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, \iota_0 \qquad {}^{r}\Gamma \vdash h_0, e_1 \rightsquigarrow h_1, v \\ h_1[\iota_0\texttt{.}f := v] = h'\end{array}}{{}^{r}\Gamma \vdash h, e_0\texttt{.}f := e_1 \rightsquigarrow h', v} \quad \text{OS\_WRITE}$$

$$\frac{\begin{array}{l}{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, \iota_0 \qquad {}^{r}\Gamma \vdash h_0, \overline{e_i}^{\,i} \rightsquigarrow h_1, \overline{v_i}^{\,i} \\ \text{MBody}(h_0, \iota_0, m) = e \qquad \text{MSig}(h_0, \iota_0, m) = \_\ m\texttt{(}\overline{\_\ pid}^{\,i}\texttt{)}\ q \\ {}^{r}\Gamma' = \left\{\texttt{precise}; \texttt{this} \mapsto \iota_0, \overline{pid \mapsto v_i}^{\,i}\right\} \\ {}^{r}\Gamma' \vdash h_1, e \rightsquigarrow h', v\end{array}}{{}^{r}\Gamma \vdash h, e_0\texttt{.}m\texttt{(}\overline{e_i}^{\,i}\texttt{)} \rightsquigarrow h', v} \quad \text{OS\_CALL}$$

$$\frac{\begin{array}{l}{}^{r}\Gamma \vdash h, e \rightsquigarrow h', v \\ h', {}^{r}\Gamma(\texttt{this}) \vdash v : q\ C\end{array}}{{}^{r}\Gamma \vdash h, \texttt{(}q\ C\texttt{)}\ e \rightsquigarrow h', v} \quad \text{OS\_CAST}$$

$$\frac{\begin{array}{c}{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, (q, {}^{r}\mathcal{L}_0) \\ {}^{r}\Gamma \vdash h_0, e_1 \rightsquigarrow h', (q, {}^{r}\mathcal{L}_1)\end{array}}{{}^{r}\Gamma \vdash h, e_0 \oplus e_1 \rightsquigarrow h', (q, {}^{r}\mathcal{L}_0 \oplus {}^{r}\mathcal{L}_1)} \quad \text{OS\_PRIMOP}$$

$$\frac{\begin{array}{l}{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, (q, {}^{r}\mathcal{L}) \qquad {}^{r}\mathcal{L} \neq 0 \\ {}^{r}\Gamma \vdash h_0, e_1 \rightsquigarrow h', v\end{array}}{{}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \rightsquigarrow h', v} \quad \text{OS\_COND\_T}$$

$$\frac{{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, (q, 0) \qquad {}^{r}\Gamma \vdash h_0, e_2 \rightsquigarrow h', v}{{}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \rightsquigarrow h', v} \quad \text{OS\_COND\_F}$$

$$\frac{{}^{r}\Gamma \vdash h, e \rightsquigarrow h', v \qquad h' \cong \tilde{h}' \qquad v \cong \tilde{v}}{{}^{r}\Gamma \vdash h, e \rightsquigarrow \tilde{h}', \tilde{v}} \quad \text{OS\_APPROX}$$

A program is executed by instantiating the main class and then evaluating the main expression in a suitable heap and environment:

$\boxed{\vdash Prg \rightsquigarrow h, v}$ big-step operational semantics of a program

$$\frac{\begin{array}{l}\text{FVsInit}(\texttt{precise}\ C) = \overline{fv} \\ \emptyset + \left(\texttt{precise}\ C, \overline{fv}\right) = (h_0, \iota_0) \\ {}^{r}\Gamma_0 = \{\texttt{precise}; \texttt{this} \mapsto \iota_0\} \qquad {}^{r}\Gamma_0 \vdash h_0, e \rightsquigarrow h, v\end{array}}{\vdash \overline{Cls}, C, e \rightsquigarrow h, v} \quad \text{OSP\_DEF}$$

We provide a checked version of the semantics that ensures that we do not have an interference between approximate and precise parts:

$\boxed{{}^{r}\Gamma \vdash h, e \rightsquigarrow_c h', v}$ checked big-step operational semantics

$$\frac{{}^{r}\Gamma \vdash h, \texttt{null} \rightsquigarrow h, \texttt{null}_a}{{}^{r}\Gamma \vdash h, \texttt{null} \rightsquigarrow_c h, \texttt{null}_a} \quad \text{COS\_NULL}$$

$$\frac{{}^{r}\Gamma \vdash h, \mathcal{L} \rightsquigarrow h, (\texttt{precise}, {}^{r}\mathcal{L})}{{}^{r}\Gamma \vdash h, \mathcal{L} \rightsquigarrow_c h, (\texttt{precise}, {}^{r}\mathcal{L})} \quad \text{COS\_LITERAL}$$

$$\frac{{}^r\Gamma \vdash h, x \ \rightsquigarrow \ h, v}{{}^r\Gamma \vdash h, x \ \rightsquigarrow_c \ h, v} \quad \text{COS\_VAR}$$

$$\frac{{}^r\Gamma \vdash h, \texttt{new}\ q\ C\texttt{()} \ \rightsquigarrow \ h', \iota}{{}^r\Gamma \vdash h, \texttt{new}\ q\ C\texttt{()} \ \rightsquigarrow_c \ h', \iota} \quad \text{COS\_NEW}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h', \iota_0 \\ {}^r\Gamma \vdash h, e_0\texttt{.}f \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, e_0\texttt{.}f \ \rightsquigarrow_c \ h', v} \quad \text{COS\_READ}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, \iota_0 \qquad h(\iota_0)\downarrow_1 = q\ C \\ {}^r\Gamma\downarrow_1 = q' \qquad (q{=}q' \ \vee \ q'{=}\texttt{precise}) \\ {}^r\Gamma \vdash h_0, e_1 \ \rightsquigarrow_c \ h_1, v \\ {}^r\Gamma \vdash h, e_0\texttt{.}f \ \texttt{:=} \ e_1 \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, e_0\texttt{.}f \ \texttt{:=} \ e_1 \ \rightsquigarrow_c \ h', v} \quad \text{COS\_WRITE}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, \iota_0 \qquad {}^r\Gamma \vdash h_0, \overline{e_i}^{\,i} \ \rightsquigarrow_c \ h_1, \overline{v_i}^{\,i} \\ \text{MBody}(h_0, \iota_0, m) \ = \ e \qquad \text{MSig}(h_0, \iota_0, m) \ = \ \_\ m\texttt{(}\overline{\_\ pid}^{\,i}\texttt{)}\ q \\ {}^r\Gamma' = \left\{\texttt{precise}; \texttt{this} \mapsto \iota_0, \ \overline{pid \mapsto v_i}^{\,i}\right\} \\ {}^r\Gamma' \vdash h_1, e \ \rightsquigarrow_c \ h', v \\ {}^r\Gamma \vdash h, e_0\texttt{.}m\texttt{(}\overline{e_i}^{\,i}\texttt{)} \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, e_0\texttt{.}m\texttt{(}\overline{e_i}^{\,i}\texttt{)} \ \rightsquigarrow_c \ h', v} \quad \text{COS\_CALL}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e \ \rightsquigarrow_c \ h', v \\ {}^r\Gamma \vdash h, \texttt{(}q\ C\texttt{)}\ e \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, \texttt{(}q\ C\texttt{)}\ e \ \rightsquigarrow_c \ h', v} \quad \text{COS\_CAST}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, (q, {}^r\mathcal{L}_0) \\ {}^r\Gamma \vdash h_0, e_1 \ \rightsquigarrow_c \ h', (q, {}^r\mathcal{L}_1) \\ {}^r\Gamma \vdash h, e_0 \oplus e_1 \ \rightsquigarrow \ h', (q, {}^r\mathcal{L}_0 \oplus {}^r\mathcal{L}_1) \end{array}}{{}^r\Gamma \vdash h, e_0 \oplus e_1 \ \rightsquigarrow_c \ h', (q, {}^r\mathcal{L}_0 \oplus {}^r\mathcal{L}_1)} \quad \text{COS\_PRIMOP}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, (q, {}^r\mathcal{L}) \qquad {}^r\mathcal{L} \neq 0 \\ {}^r\Gamma' = {}^r\Gamma\texttt{(}q\texttt{)} \qquad {}^r\Gamma' \vdash h_0, e_1 \ \rightsquigarrow_c \ h', v \\ {}^r\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow_c \ h', v} \quad \text{COS\_COND\_T}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, (q, {}^r\mathcal{L}) \qquad {}^r\mathcal{L} = 0 \\ {}^r\Gamma' = {}^r\Gamma\texttt{(}q\texttt{)} \qquad {}^r\Gamma' \vdash h_0, e_2 \ \rightsquigarrow_c \ h', v \\ {}^r\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow_c \ h', v} \quad \text{COS\_COND\_F}$$

2.5 Well-formedness

A heap is well formed if all field values are correctly typed and all types are valid:

$\boxed{h \ \ \mathsf{OK}}$ well-formed heap

$$\frac{\begin{array}{c} \forall \iota \in \text{dom}(h)\,, f \in h(\iota)\downarrow_2 . \ (\text{FType}(h, \iota, f) \ = T \ \wedge \ h, \iota \vdash h(\iota\texttt{.}f) \ : \ T) \\ \forall \iota \in \text{dom}(h) . \ (h(\iota)\downarrow_1 \ \ \mathsf{OK} \ \wedge \ \text{TQual}(h(\iota)\downarrow_1) \in \{\texttt{precise}, \texttt{approx}\}) \end{array}}{h \ \ \mathsf{OK}} \quad \text{WFH\_DEF}$$

This final judgment ensures that the heap and runtime environment correspond to a static environment. It makes sure that all pieces match up:

$\boxed{h, {}^r\Gamma : {}^s\Gamma \ \ \mathsf{OK}}$ runtime and static environments correspond

$$\frac{\begin{array}{l} {}^{r}\Gamma = \left\{ \texttt{precise}; \texttt{this} \mapsto \iota,\ \overline{pid \mapsto v_i}^{\,i} \right\} \\ {}^{s}\Gamma = \left\{ \texttt{this} \mapsto \texttt{context}\ C,\ \overline{pid \mapsto T_i}^{\,i} \right\} \\ \quad h\ \ \mathsf{OK} \qquad\qquad\qquad {}^{s}\Gamma\ \ \mathsf{OK} \\ h, \iota \vdash \iota :\ \texttt{context}\ C \\ h, \iota \vdash \overline{v_i}^{\,i} :\ \overline{T_i}^{\,i} \end{array}}{h, {}^{r}\Gamma : {}^{s}\Gamma\ \ \mathsf{OK}} \quad \text{WFRSE\_DEF}$$

3 Proofs

The principal goal of formalizing EnerJ is to prove a *non-interference* property (Theorem 3.3). The other properties listed in this section support that proof.

3.1 Type Safety

Theorem 3.1 (Type Safety)

$$\left.\begin{array}{ll} 1. & \vdash Prg\ \ \mathit{OK} \\ 2. & h, {}^{r}\Gamma : {}^{s}\Gamma\ \ \mathit{OK} \\ 3. & {}^{s}\Gamma \vdash e : T \\ 4. & {}^{r}\Gamma \vdash h, e \rightsquigarrow h', v \end{array}\right\} \Longrightarrow \left\{ \begin{array}{ll} I. & h', {}^{r}\Gamma : {}^{s}\Gamma\ \ \mathit{OK} \\ II. & h', {}^{r}\Gamma(\texttt{this}) \vdash v :\ T \end{array}\right.$$

We prove this by rule induction on the operational semantics.

Case 1: e**=**`null`
The heap is not modified so $I.$ trivially holds.
The null literal statically gets assigned an arbitrary reference type. The null value can be assigned an arbitrary reference type.

Case 2: e**=**$\mathcal{L}$
The heap is not modified so $I.$ trivially holds.
A primitive literal statically gets assigned type precise or a supertype. The evaluation of a literal gives a precise value which can be assigned any primitive type.

Case 3: e**=**x
The heap is not modified so $I.$ trivially holds.
We know that 2. that the environments correspond and therefore that the static type of the variable can be assigned to the value of the variable.

Case 4: e**=**`new` qC`()`
For $I.$ we only have to show that the newly created object is valid. The initialization with the null or zero values ensures that all fields are correctly typed.
The type of the new object is the result of sTrT on the static type.

Case 5: e**=**$e_0.f$
The heap is not modified so $I.$ trivially holds.
We know from 2. that the heap is well formed. In particular, we know that the values stored for fields are subtypes of the field types.
We perform induction on e_0 and then use Lemma 3.4 to adapt the declared field, which is checked by the well-formed heap, to the adapted field type T.

Case 6: e**=**$e_0.f\ := \ e_1$

We perform induction on e_0 and e_1. We know from 3. that the static type of e_1 is a subtype of the adapted field type. We use Lemma 3.5 to adapt the type to the declaring class to re-establish that the heap is well formed.

Case 7: e=e_0 . m ($\overline{e}$)
A combination of cases 6 and 7.

Case 8: e=(qC) e
By induction we know that the heap is still well formed.
4. performs a runtime check to ensure that the value has the correct type.

Case 9: e=$e_0 \oplus e_1$
By induction we know that the heap is still well formed.
The type matches trivially.

Case 10: e=`if`(e_0) {e_1} `else` {e_2}
By induction we know that the heap is still well formed.
The type matches by induction. □

3.2 Equivalence of Checked Semantics

We prove that an execution under the unchecked operational semantics has an equivalent execution under the checked semantics.

Theorem 3.2 (Equivalence of Checked Semantics)

$$\left.\begin{array}{ll} 1. & \vdash \mathit{Prg}\ \ \mathsf{OK} \\ 2. & h, {}^{r}\Gamma : {}^{s}\Gamma\ \ \mathsf{OK} \\ 3. & {}^{s}\Gamma \vdash e : T \\ 4. & {}^{r}\Gamma \vdash h, e \ \rightsquigarrow\ h', v \end{array}\right\} \Longrightarrow\ I.\quad {}^{r}\Gamma \vdash h, e \ \rightsquigarrow_c\ h', v$$

We prove this by rule induction on the operational semantics.
The checked operational semantics is only different from the unchecked semantics for the field write, method call, and conditional cases. The other cases trivially hold.

Case 1: e=`if`(e_0) {e_1} `else` {e_2}
We know from 3. that the static type of the condition is always precise. Therefore, ${}^{r}\Gamma'$ is well formed and we can apply the induction hypothesis on e_1 and e_2.

Case 2: e=e_0 . m ($\overline{e}$)
From the proof of type safety we know that the values in ${}^{r}\Gamma'$ are well formed. We are using `precise` as the approximate environment. Therefore, ${}^{r}\Gamma'$ is well formed and we can apply the induction hypothesis on e.

Case 3: e=e_0 . f := e_1
We know from 2. that q'=`precise`. Therefore, the additional check passes. □

3.3 Non-Interference

The express a non-interference property, we first define a relation $\cong$ on values, heaps, and environments. Intuitively, $\cong$ denotes an equality that disregards approximate values. The relation only holds for values, heaps, and environments with identical types.

Where v and $\tilde{v}$ are primitive values, $v \cong \tilde{v}$ iff the values have the same type qP and either q = `approx` or $v = \tilde{v}$. For objects, $\iota \cong \tilde{\iota}$ iff $\iota = \tilde{\iota}$. For heaps, $h \cong \tilde{h}$ iff the two heaps contain the same set of addresses ι and, for each such ι and each respective field f, $h(\iota.f) \cong \tilde{h})(\iota.f)$. Similarly, for environments, ${}^{r}\Gamma \cong {}^{r}\tilde{\Gamma}$ iff ${}^{r}\Gamma(\texttt{this}) \cong {}^{r}\tilde{\Gamma}(\texttt{this})$ and, for every parameter identifier pid, ${}^{r}\Gamma(pid) \cong {}^{r}\tilde{\Gamma}(pid)$.

We can now state our desired non-interference property.

Theorem 3.3 (Non-Interference)

$$\left.\begin{array}{ll} 1. & \vdash Prg\ \texttt{OK}\ \wedge\ \vdash h, {}^{r}\Gamma : {}^{s}\Gamma \\ 2. & {}^{s}\Gamma \vdash e : T \\ 3. & {}^{r}\Gamma \vdash h, e \rightsquigarrow h', v \\ 4. & h \cong \tilde{h}\ \wedge\ {}^{r}\Gamma \cong {}^{r}\tilde{\Gamma} \\ 5. & \vdash \tilde{h}, {}^{r}\tilde{\Gamma} : {}^{s}\Gamma \end{array}\right\} \Longrightarrow \left\{\begin{array}{ll} I. & {}^{r}\tilde{\Gamma} \vdash \tilde{h}, e \rightarrow \tilde{h}', \tilde{v} \\ II. & h' \cong \tilde{h}' \\ III. & v \cong \tilde{v} \end{array}\right.$$

The non-interference property follows from the definition of the checked semantics, which are shown to hold in Theorem 3.2 given premises 1, 2, and 3. That is, via Theorem 3.2, we know that ${}^{r}\Gamma \vdash h, e \rightsquigarrow_c h', v$. The proof proceeds by rule induction on the *checked* semantics.

Case 1: e=`null`
The heap is unmodified, so $h = h'$ and $\tilde{h}' = \tilde{h}$. Because $h \cong \tilde{h}$, trivially $h' \cong \tilde{h}'$ (satisfying $II.$).
Both $v =$ `null` and $\tilde{v} =$ `null`, so $III.$ also holds.

Case 2: e=$\mathcal{L}$
As above, the heap is unmodified and $v = \tilde{v}$ because literals are assigned precise types.

Case 3: e=x
Again, the heap is unmodified. If x has precise type, then $v = \tilde{v}$ and $III.$ holds. Otherwise, both v and $\tilde{v}$ have approximate type so $v \cong \tilde{v}$ vacuously. (That is, $v \cong \tilde{v}$ holds for any such pair of values when their type is approximate.)

Case 4: e=`new` qC`()`
In this case, a new object o is created with address v and $h' = h \oplus (v \mapsto o)$. Because v has a reference type and $\tilde{v}$ has the same type, $v \cong \tilde{v}$. Furthermore, $\tilde{h}' = h \oplus (\tilde{v} \mapsto o)$, so $h \cong \tilde{h}$.

Case 5: e=e_0`.`f
The heap is unmodified in field lookup, so $II.$ holds by induction. Also by induction, e_0 resolves to the same address ι under h as under $\tilde{h}$ due to premise 4. If $h(\iota.f)$ has approximate type, then $III.$ holds vacuously; otherwise $v = \tilde{v}$.

Case 6: e=e_0`.`f `:=` e_1
Apply induction to both subexpressions (e_0 and e_1). Under either heap h or $\tilde{h}$, the first expression e_0 resolves to the same object o. By type safety, e_1 resolves to a value with a dynamic type compatible with the static type of o's field f.
If the value is approximate, then the field must have approximate type and the conclusions hold vacuously. If the value is precise, then induction implies that the value produced by e_1 must be $v = \tilde{v}$, satisfying $III.$ Similarly, the heap update to h is identical to the one to $\tilde{h}$, so $\tilde{h} \cong \tilde{h}'$.

Case 7: e=e_0`.`m`(`$\overline{e}$`)`
As in Case 5, let e_0 map to o in both h and $\tilde{h}$. The same method body is therefore looked up by MBody and, by induction on the evaluation of the method body, the conclusions all hold.

Case 8: e=`(`qC`)` e
Induction applies directly; the expression changes neither the output heap nor the value produced.

Case 9: e=$e_0 \oplus e_1$
The expression does not change the heap. If the type of $e_0 \oplus e_1$ is approximate, then $III.$ hold vacuously. If it is precise, then both $e0$ and $e1$ also have precise type, and, via induction, each expression produces the same literal under h and ${}^{r}\Gamma$ as under $\tilde{h}$ and ${}^{r}\tilde{\Gamma}$. Therefore, $v = \tilde{v}$, satisfying $III.$

Case 10: e=`if(`e_0`)` `{`e_1`}` `else` `{`e_2`}`
By type safety, e_0 resolves to a value with precise type. Therefore, by induction, the expression produces the same value under heap h and environment ${}^{r}\Gamma$ as under the equivalent structures $\tilde{h}$ and ${}^{r}\tilde{\Gamma}$. The rule applied for ${}^{r}\Gamma \vdash h, e \rightsquigarrow h', v$ (either COS_COND_T or COS_COND_F) also applies for ${}^{r}\tilde{\Gamma} \vdash \tilde{h}, e \rightarrow \tilde{h}', \tilde{v}$ because the value in the condition is the same in either case. That is, either e_1 is evaluated in bot settings or else e_2 is; induction applies in either case. $\square$

3.4 Adaptation from a Viewpoint

Lemma 3.4 (Adaptation from a Viewpoint)

$$\left.\begin{array}{ll} 1. & h, \iota_0 \vdash \iota : q\ C \\ 2. & h, \iota \vdash v : T \end{array}\right\} \Longrightarrow \begin{array}{l} \exists T'.\ q \triangleright T = T' \wedge \\ \quad h, \iota_0 \vdash v : T' \end{array}$$

This lemma justifies the type rule TR_READ and the method result in TR_CALL.

Case analysis of T:

Case 1: T**=**$q'\ C'$ **or** T**=**$q'\ P$ **where** $q' \in \{\texttt{precise}, \texttt{approx}, \texttt{top}\}$

In this case we have that T'=T and the viewpoint is irrelevant.

Case 2: T**=**`context` C' **or** T**=**`context` P

Case 2a: $q \in \{\texttt{precise}, \texttt{approx}\}$

We have that T'=$q\ C'$ or T'=$q\ P$, respectively.

2. uses the precision of ι to substitute `context`. 1. gives us the type for ι. Together, they give us the type of v relative to ι_0.

Case 2b: $q \in \{\texttt{lost}, \texttt{top}\}$

We have that T'=`lost` C' or T'=`lost` P, respectively.

Such a T' is a valid type for any value. □

3.5 Adaptation to a Viewpoint

Lemma 3.5 (Adaptation to a Viewpoint)

$$\left.\begin{array}{ll} 1. & h, \iota_0 \vdash \iota : q\ C \\ 2. & q \triangleright T = T' \\ 3. & \texttt{lost} \notin T' \\ 4. & h, \iota_0 \vdash v : T' \end{array}\right\} \Longrightarrow h, \iota \vdash v : T$$

This lemma justifies the type rule TR_WRITE and the requirements for the types of the parameters in TR_CALL.

Case analysis of T:

Case 1: T**=**$q'\ C'$ **or** T**=**$q'\ P$ **where** $q' \in \{\texttt{precise}, \texttt{approx}, \texttt{top}\}$

In this case we have that T'=T and the viewpoint is irrelevant.

Case 2: T**=**`context` C' **or** T**=**`context` P

We have that T'=$q\ C'$ or T'=$q\ P$, respectively. 3. forbids lost from occurring.

1. gives us the precision for ι and 4. for v, both relative to ι_0. From 2. and 3. we get the conclusion. □

References

[1] A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight Java: a minimal core calculus for Java and GJ. *TOPLAS*, 23(3), 2001. 1

[2] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI*, 2011. 1

4 Complete Grammar

We use the tool Ott to formalize EnerJ and used the generated LaTeX code throughout this document.

We define the following Ott meta-variables:

i, j, k, n	index variables as arbitrary elements
f	field identifier
mid	method identifier
pid	parameter identifier
Cid	derived class identifier
$RAId$	raw address identifier
$PrimV$	primitive value

The grammar of EnerJ is as follows:

terminals	::=		
	|	:=	
	|	`class`	keyword: class declaration
	|	`extends`	keyword: super type declaration
	|	`new`	keyword: object creation
	|	`if`	keyword: if
	|	`else`	keyword: else
	|	`this`	keyword: current object
	|	`null`	keyword: null value
	|	$\oplus$	syntax: primitive operation
	|	{	syntax: start block
	|	}	syntax: end block
	|	`(`	syntax: start parameters
	|	`)`	syntax: end parameters
	|	`;`	syntax: separator
	|	`.`	syntax: selector
	|	`=`	syntax: assignment
	|	`Object`	name of root class
	|	`int`	name of primitive type
	|	$\mathcal{L}$	primitive literal
	|	$<:_{\mathrm{q}}$	ordering of precision qualifiers
	|	$<:$	subtyping
	|	$\in$	containment judgement
	|	$\notin$	non-containment judgement
	|	$\vdash$	single element judgement
	|	$\vdash$	multiple element judgement
	|	:	separator
	|	$\mapsto$	maps-to
	|	OK	well-formedness judgement
	|	$=$	alias
	|	$=$	option alias
	|	$\neq$	not alias
	|	$=$	multiple alias

	|	$\vee$		logical or
	|	$\wedge$		logical and
	|	**AND**		top-level logical and
	|	$\Longrightarrow$		logical implication
	|	$\texttt{null}_a$		special null address
	|	ρ		primitive values
	|	0		zero value
formula	::=			formulas
	|	otherwise		none of the previous rules applied
	|	*judgement*		judgement
	|	$formula_1 \ , \ .. \ , \ formula_k$		sequence
	|	$(formula)$		bracketed
	|	$!formula$		negation
	|	$formula \ \vee \ formula'$		logical or
	|	$formula \ \wedge \ formula'$		logical and
	|	$formula \qquad formula'$		top-level logical and
	|	$formula \Longrightarrow formula'$		implies
	|	$\overline{{}^sfml}$		static formulas
	|	$\overline{{}^rfml}$		runtime formulas
	|	$\forall f \in \overline{f}.\ formula$		for all f in $\overline{f}$ holds *formula*
	|	$\forall C, C'.\ formula$		for all C and C' holds *formula*
	|	$\forall \iota \in \bar{\iota}.\ formula$		for all ι in $\bar{\iota}$ holds *formula*
	|	$\forall \iota \in \bar{\iota}, f \in \overline{fv}.\ formula$		for all ι in $\bar{\iota}$ and fields f in $\overline{fv}$ holds *formula*
	|	$h \cong h'$		two heaps are equal in their precise part
	|	$v \cong v'$		two values are equal in their precise parts
C	::=			class name
	|	*Cid*		derived class identifier
	|	`Object`		name of base class
	|	_	M	some class name
P	::=			primitive type name
	|	`int`		integers
	|	**float**		floating-point numbers
q	::=			precision qualifier
	|	`precise`		precise
	|	`approx`		approximate
	|	`top`		top
	|	`context`		context
	|	`lost`		lost
	|	$\mathrm{TQual}(T)$	M	extract precision qualifier from type
	|	$\mathrm{MQual}(ms)$	M	extract method qualifier from signature
	|	${}^r\Gamma{\downarrow_1}$	M	extract environment qualifier

$\overline{q}$	$::=$			precision qualfiers
	$\mid$	$q_1, .., q_n$		precision qualifier list
	$\mid$	$\{\overline{q}\}$	M	notation
qC	$::=$			qualified class name
	$\mid$	$q\ C$		definition
qP	$::=$			qualified primitive type
	$\mid$	$q\ P$		definition
T	$::=$			type
	$\mid$	qC		reference type
	$\mid$	qP		primitive type
	$\mid$	$\_$	M	some type
	$\mid$	${}^{s}\Gamma(x)$	M	look up parameter type
	$\mid$	$h(\iota)\downarrow_1$	M	look up type in heap
$\overline{T}$	$::=$			types
	$\mid$	$T_1, .., T_n$		type list
	$\mid$	$\overline{T}_1,\ \overline{T}_2$		two type lists
	$\mid$	$\emptyset$		no types
	$\mid$	$\_$	M	some types
Prg	$::=$			program
	$\mid$	$\overline{Cls},\ C,\ e$		
Cls	$::=$			class declaration
	$\mid$	`class` Cid `extends` C $\{\ \overline{fd}\ \overline{md}\ \}$		class declaration
	$\mid$	`class Object` $\{\}$		declaration of base class
$\overline{Cls}$	$::=$			class declarations
	$\mid$	$Cls_1 .. Cls_n$		class declaration list
$\overline{fd}$	$::=$			field declarations
	$\mid$	$T\ f\,;$		type T and field name f
	$\mid$	$\overline{fd}_1 .. \overline{fd}_n$		field declaration list
	$\mid$	$\_$	M	some field declarations
$\overline{f}$	$::=$			list of field identifiers
	$\mid$	$f_1 .. f_n$		field identifier list
	$\mid$	$\mathrm{refFields}(C)$	M	recursive reference type fields look-up
	$\mid$	$\mathrm{primFields}(C)$	M	recursive primitive type fields look-up
e	$::=$			expression
	$\mid$	`null`		null expression

	|	$\mathcal{L}$		primitive literal
	|	x		variable read
	|	$\texttt{new}\ qC\texttt{()}$		object construction
	|	$e\,.\,f$		field read
	|	$e_0\,.\,f\ :=\ e_1$		field write
	|	$e_0\,.\,m\texttt{(}\overline{e}\texttt{)}$		method call
	|	$\texttt{(}qC\texttt{)}\ e$		cast
	|	$e_0 \oplus e_1$		primitive operation
	|	$\texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\}$		conditional
	|	*None*	M	no expression defined
$\overline{e}$	::=			expressions
	|	$e_1, .., e_k$		list of expressions
	|	$\emptyset$		empty list
md	::=			method declaration
	|	$ms\ \{\ e\ \}$		method signature and method body
$\overline{md}$	::=			method declarations
	|	md		method declaration
	|	$\overline{md}_1 .. \overline{md}_n$		method declaration list
	|	_	M	some method declarations
ms	::=			method signature
	|	$T\ m\texttt{(}\overline{mpd}\texttt{)}\ q$		method signature definition
	|	*None*	M	no method signature defined
m	::=			method name
	|	mid		method identifier
	|	$\mathrm{MName}(ms)$	M	extract method name from signature
$\overline{mpd}$	::=			method parameter declarations
	|	$T\ pid$		type and parameter name
	|	$\overline{mpd}_1, .., \overline{mpd}_n$		list
	|	_	M	some method parameter declarations
x	::=			parameter name
	|	pid		parameter identifier
	|	$\texttt{this}$		name of current object
${}^s\Gamma$	::=			static environment
	|	$\{{}^s\delta\}$		composition
${}^s\delta_p$	::=			static variable parameter environment
	|	$pid \mapsto T$		variable pid has type T

${}^s\delta_t$	::=			static variable environment for `this`
	|	$\texttt{this} \mapsto T$		variable `this` has type T
${}^s\delta$	::=			static variable environment
	|	${}^s\delta_t$		mapping for `this`
	|	${}^s\delta_t, \_$		mapping for `this` and some others
	|	${}^s\delta_t, {}^s\delta_{p_1}, .., {}^s\delta_{p_i}$		mappings list
$\overline{{}^sfml}$	::=			static formulas
	|	$Prg{=}Prg'$		program alias
	|	$C{=}C'$		class alias
	|	$T{=}T'$		type alias
	|	$T{=}T'$		option type alias
	|	$q{=}q'$		qualifier alias
	|	$q{\neq}q'$		qualifier not alias
	|	$q \in \overline{q}$		qualifier in set of qualifiers
	|	$q \notin \overline{T}$		qualifier in set of types
	|	$m{=}m'$		method name alias
	|	$m{\neq}m'$		method name not alias
	|	$ms{=}ms'$		method signature alias
	|	${}^s\Gamma{=}{}^s\Gamma'$		static environment alias
	|	$e{=}e'$		expression alias
	|	$Cls \in Prg$		class definition in program
	|	$\texttt{class}\ C \ldots \in Prg$		partial class definition in program
ι	::=			address identifier
	|	$RAId$		raw address identifier
	|	${}^r\Gamma(\texttt{this})$	M	currently active object look-up
	|	$\_$	M	some address identifier
${}^r\mathcal{L}$	::=			primitive value
	|	0		zero value
	|	$PrimV$		primitive value
	|	${}^r\mathcal{L}_0 \oplus {}^r\mathcal{L}_1$	M	binary operation
ρ_q	::=			qualified primitive value
	|	$(q, {}^r\mathcal{L})$		qualified primitive value
$\overline{\iota}$	::=			address identifiers
	|	$\iota_1, .., \iota_n$		address identifier list
	|	$\emptyset$		empty list
	|	$\_$	M	some address identifier list
	|	$\mathrm{dom}(h)$	M	domain of heap
v	::=			value
	|	ι		address identifier

	|	$\texttt{null}_a$		null value
	|	ρ_q		qualified primitive value
	|	$\_$	M	some value
	|	$\tilde{v}$		similarity
	|	$h(\iota.f)$	M	field value look-up
	|	${}^r\Gamma(x)$	M	argument value look-up
	|	$\overline{fv}(f)$	M	field value look-up
$\overline{v}$	::=			values
	|	$v_1, .., v_n$		value list
	|	$\emptyset$		empty list
$\overline{fv}$	::=			field values
	|	$f \mapsto v$		field f has value v
	|	$\overline{fv}_1, .., \overline{fv}_n$		field value list
	|	$\_$	M	some field values
	|	$h(\iota){\downarrow}_2$	M	look up field values in heap
	|	$\overline{fv}[f \mapsto v]$	M	update existing field f to v
o	::=			object
	|	$\left(T, \overline{fv}\right)$		type T and field values $\overline{fv}$
	|	$h(\iota)$	M	look up object in heap
he	::=			heap entry
	|	$(\iota \mapsto o)$		address ι maps to object o
h	::=			heap
	|	$\emptyset$		empty heap
	|	$h \oplus he$		add he to h, overwriting existing mappings
	|	$\tilde{h}$		similarity
${}^r\Gamma$	::=			runtime environment
	|	$\{q; {}^r\delta\}$		composition
	|	${}^r\Gamma(q)$	M	update the precision in environment ${}^r\Gamma$
${}^r\delta_p$	::=			runtime variable environment parameter entry
	|	$pid \mapsto v$		variable pid has value v
${}^r\delta_t$	::=			runtime variable environment entry for `this`
	|	$\texttt{this} \mapsto \iota$		variable `this` has address ι
${}^r\delta$	::=			runtime variable environment
	|	${}^r\delta_t$		mapping for `this`
	|	${}^r\delta_t, \_$		mapping for `this` and some others
	|	${}^r\delta_t, {}^r\delta_{p_1}, .., {}^r\delta_{p_k}$		mappings list

$\overline{{}^{r}fml}$	::=		runtime formulas
	|	$h{=}h'$	heap alias
	|	${}^{r}\mathcal{L}{=}{}^{r}\mathcal{L}'$	primitive value alias
	|	${}^{r}\mathcal{L}{\neq}{}^{r}\mathcal{L}'$	primitive value not alias
	|	${}^{r}\mathcal{L} \in P$	primitive value has primitive type
	|	$v{=}v'$	value alias
	|	$v{\neq}v'$	value not alias
	|	$v{=}v'$	value alias
	|	$o{=}o'$	object alias
	|	$\iota \in \overline{\iota}$	address in addresses
	|	$\overline{\iota}{\neq}\overline{\iota}'$	addresses not aliased
	|	$\iota \notin \overline{\iota}$	address not in addresses
	|	$f \in \mathrm{dom}(\overline{fv})$	field identifier f contained in domain of $\overline{fv}$
	|	$x \in {}^{r}\Gamma$	parameter in runtime environment
	|	${}^{r}\Gamma{=}{}^{r}\Gamma'$	runtime environment alias
	|	$\overline{fv}{=}\overline{fv}'$	fields alias
stsubxing	::=		
	|	$q <:_{\mathrm{q}} q'$	ordering of precision qualifiers
	|	$C \sqsubseteq C'$	subclassing
	|	$T <: T'$	subtyping
	|	$\overline{T} <: \overline{T}'$	subtypings
	|	$ms <: ms'$	invocations of method ms can safely be replaced by calls to ms'
qcombdef	::=		
	|	$q \rhd q' = q''$	combining two precision qualifiers
	|	$q \rhd T = T'$	precision qualifier - type combination
	|	$q \rhd \overline{T} = \overline{T}'$	precision qualifier - types combination
	|	$q \rhd ms = ms'$	precision qualifier - method signature combination
st_helpers	::=		
	|	$\mathrm{FType}(C, f) = T$	look up field f in class C
	|	$\mathrm{FType}(qC, f) = T$	look up field f in reference type qC
	|	$\mathrm{MSig}(C, m, q) = ms$	look up signature of method m in class C
	|	$\mathrm{MSig}(qC, m) = ms$	look up signature of method m in reference type qC
typerules	::=		
	|	${}^{s}\Gamma \vdash e : T$	expression typing
	|	${}^{s}\Gamma \vdash \overline{e} : \overline{T}$	expression typings
wfstatic	::=		
	|	T OK	well-formed type
	|	$\overline{T}$ OK	well-formed types
	|	Cls OK	well-formed class declaration
	|	${}^{s}\Gamma \vdash T\ f;$ OK	well-formed field declaration
	|	${}^{s}\Gamma \vdash \overline{fd}$ OK	well-formed field declarations

	|	${}^s\Gamma, C \vdash md$ OK	well-formed method declaration
	|	${}^s\Gamma, C \vdash \overline{md}$ OK	well-formed method declarations
	|	$C \vdash m$ OK	method overriding OK
	|	$C, C' \vdash m$ OK	method overriding OK auxiliary
	|	${}^s\Gamma$ OK	well-formed static environment
	|	$\vdash Prg$ OK	well-formed program
rt_helpers	::=		
	|	$h + o = (h', \iota)$	add object o to heap h resulting in heap h' and fresh address ι
	|	$h[\iota.f := v] = h'$	field update in heap
	|	$\mathrm{sTrT}(h, \iota, T) = T'$	convert type T to its runtime equivalent T'
	|	$h, \iota \vdash v : T$	type T assignable to value v
	|	$h, \iota \vdash \overline{v} : \overline{T}$	types $\overline{T}$ assignable to values $\overline{v}$
	|	$\mathrm{FType}(h, \iota, f) = T$	look up type of field in heap
	|	$\mathrm{MSig}(h, \iota, m) = ms$	look up method signature of method m at ι
	|	$\mathrm{MBody}(C, m, q) = e$	look up most-concrete body of m, q in class C or a superclass
	|	$\mathrm{MBody}(h, \iota, m) = e$	look up most-concrete body of method m at ι
	|	$\mathrm{FVsInit}(qC) = \overline{fv}$	initialize the fields for reference type qC
semantics	::=		
	|	${}^r\Gamma \vdash h, e \rightsquigarrow h', v$	big-step operational semantics
	|	${}^r\Gamma \vdash h, \overline{e} \rightsquigarrow h', \overline{v}$	sequential big-step operational semantics
	|	$\vdash Prg \rightsquigarrow h, v$	big-step operational semantics of a program
	|	${}^r\Gamma \vdash h, e \rightsquigarrow_c h', v$	checked big-step operational semantics
	|	${}^r\Gamma \vdash h, \overline{e} \rightsquigarrow_c h', \overline{v}$	checked sequential big-step operational semantics
wfruntime	::=		
	|	h OK	well-formed heap
	|	$h, {}^r\Gamma : {}^s\Gamma$ OK	runtime and static environments correspond

5 Complete Definitions

$\boxed{q <:_{\mathrm{q}} q'}$ ordering of precision qualifiers

$$\frac{q \neq \texttt{top}}{q <:_{\mathrm{q}} \texttt{lost}} \quad \text{QQ\_LOST}$$

$$\frac{}{q <:_{\mathrm{q}} \texttt{top}} \quad \text{QQ\_TOP}$$

$$\frac{}{q <:_{\mathrm{q}} q} \quad \text{QQ\_REFL}$$

$\boxed{C \sqsubseteq C'}$ subclassing

$$\frac{\texttt{class } Cid \texttt{ extends } C' \; \{ \_ \_ \} \in Prg}{Cid \sqsubseteq C'} \quad \text{SC\_DEF}$$

$$\frac{\texttt{class } C \ldots \in Prg}{C \sqsubseteq C} \quad \text{SC\_REFL}$$

$$\frac{C \sqsubseteq C_1 \qquad C_1 \sqsubseteq C'}{C \sqsubseteq C'} \quad \text{SC\_TRANS}$$

$\boxed{T <: T'}$ subtyping

$$\frac{q <:_{\mathrm{q}} q' \qquad C \sqsubseteq C'}{q\ C <: q'\ C'} \quad \text{ST\_REFT}$$

$$\frac{q <:_{\mathrm{q}} q'}{q\ P <: q'\ P} \quad \text{ST\_PRIMT1}$$

$$\frac{}{\texttt{precise}\ P <: \texttt{approx}\ P} \quad \text{ST\_PRIMT2}$$

$\boxed{\overline{T} <: \overline{T}'}$ subtypings

$$\frac{\overline{T_i <: T'_i}^{\,i}}{\overline{T_i}^{\,i} <: \overline{T'_i}^{\,i}} \quad \text{STS\_DEF}$$

$\boxed{ms <: ms'}$ invocations of method ms can safely be replaced by calls to ms'

$$\frac{T' <: T \qquad \overline{T_k}^{\,k} <: \overline{T'_k}^{\,k}}{T\ m(\overline{T_k\ pid}^{\,k})\ \texttt{precise} <: T'\ m(\overline{T'_k\ pid}^{\,k})\ \texttt{approx}} \quad \text{MST\_DEF}$$

$\boxed{q \rhd q' = q''}$ combining two precision qualifiers

$$\frac{q' {=} \texttt{context} \ \wedge\ (q \in \{\texttt{approx}, \texttt{precise}, \texttt{context}\})}{q \rhd q' = q} \quad \text{QCQ\_CONTEXT}$$

$$\frac{q' {=} \texttt{context} \ \wedge\ (q \in \{\texttt{top}, \texttt{lost}\})}{q \rhd q' = \texttt{lost}} \quad \text{QCQ\_LOST}$$

$$\frac{q' {\neq} \texttt{context}}{q \rhd q' = q'} \quad \text{QCQ\_FIXED}$$

$\boxed{q \rhd T = T'}$ precision qualifier - type combination

$$\frac{q \rhd q' = q''}{q \rhd q'\ C = q''\ C} \quad \text{QCT\_REFT}$$

$$\frac{q \rhd q' = q''}{q \rhd q'\ P = q''\ P} \quad \text{QCT\_PRIMT}$$

$\boxed{q \rhd \overline{T} = \overline{T}'}$ precision qualifier - types combination

$$\frac{\overline{q \rhd T_k = T'_k}^{\,k}}{q \rhd \overline{T_k}^{\,k} = \overline{T'_k}^{\,k}} \quad \text{QCTS\_DEF}$$

$\boxed{q \rhd ms = ms'}$ precision qualifier - method signature combination

$$\frac{q \rhd T = T' \qquad q \rhd \overline{T_k}^{\,k} = \overline{T'_k}^{\,k}}{q \rhd T\ m(\overline{T_k\ pid}^{\,k})\ q' = T'\ m(\overline{T'_k\ pid}^{\,k})\ q'} \quad \text{QCMS\_DEF}$$

$\boxed{\mathrm{FType}(C, f) = T}$ look up field f in class C

$$\frac{\texttt{class}\ Cid\ \texttt{extends}\ \_\ \{\ \_T\ f\texttt{;}\ \_\ \_\ \}\in Prg}{\text{FType}(Cid, f)\ =T}\ \text{SFTC\_DEF}$$

$\boxed{\text{FType}(qC, f)\ =T}$ look up field f in reference type qC

$$\frac{\text{FType}(C, f)\ =T_1 \qquad q\ \triangleright\ T_1\ =\ T}{\text{FType}(q\ C, f)\ =T}\ \text{SFTT\_DEF}$$

$\boxed{\text{MSig}(C, m, q)\ = ms}$ look up signature of method m in class C

$$\frac{\begin{array}{l}\texttt{class}\ Cid\ \texttt{extends}\ \_\ \{\ \_\ \_\ ms\ \{\ e\ \}\ \_\ \}\in Prg\\ \text{MName}(ms)=m\ \wedge\ \text{MQual}(ms)=q\end{array}}{\text{MSig}(Cid, m, q)\ = ms}\ \text{SMSC\_DEF}$$

$\boxed{\text{MSig}(qC, m)\ = ms}$ look up signature of method m in reference type qC

$$\frac{\text{MSig}(C, m, q)\ = ms \qquad q\ \triangleright\ ms\ =\ ms'}{\text{MSig}(q\ C, m)\ = ms'}\ \text{SMST\_DEF}$$

$\boxed{{}^s\Gamma \vdash e : T}$ expression typing

$$\frac{{}^s\Gamma \vdash e : T_1 \qquad T_1 <:\ T}{{}^s\Gamma \vdash e : T}\ \text{TR\_SUBSUM}$$

$$\frac{qC\ \ \mathsf{OK}}{{}^s\Gamma \vdash \texttt{null} : qC}\ \text{TR\_NULL}$$

$$\frac{}{{}^s\Gamma \vdash \mathcal{L} : \texttt{precise}\ P}\ \text{TR\_LITERAL}$$

$$\frac{{}^s\Gamma(x)=T}{{}^s\Gamma \vdash x : T}\ \text{TR\_VAR}$$

$$\frac{\begin{array}{l}q\ C\ \ \mathsf{OK}\\ q\in\{\texttt{precise}, \texttt{approx}, \texttt{context}\}\end{array}}{{}^s\Gamma \vdash \texttt{new}\ q\ C\texttt{()} : T}\ \text{TR\_NEW}$$

$$\frac{{}^s\Gamma \vdash e_0 : q\ C \qquad \text{FType}(q\ C, f)\ =T}{{}^s\Gamma \vdash e_0\texttt{.}f : T}\ \text{TR\_READ}$$

$$\frac{\begin{array}{ll}{}^s\Gamma \vdash e_0 : q\ C & \text{FType}(q\ C, f)\ =T\\ \texttt{lost}\notin T & {}^s\Gamma \vdash e_1 : T\end{array}}{{}^s\Gamma \vdash e_0\texttt{.}f\ :=\ e_1 : T}\ \text{TR\_WRITE}$$

$$\frac{\begin{array}{ll}{}^s\Gamma \vdash e_0 : q\ C & q\in\{\texttt{precise}, \texttt{context}, \texttt{top}\}\\ \multicolumn{2}{l}{\text{MSig}(\texttt{precise}\ C, m)\ = T\ m(\overline{T_i\ pid}^{\,i})\ \texttt{precise}}\\ \texttt{lost}\notin\overline{T_i}^{\,i} & {}^s\Gamma \vdash \overline{e_i}^{\,i} : \overline{T_i}^{\,i}\end{array}}{{}^s\Gamma \vdash e_0\texttt{.}m(\overline{e_i}^{\,i}) : T}\ \text{TR\_CALL1}$$

$$\frac{\begin{array}{ll}{}^s\Gamma \vdash e_0 : \texttt{approx}\ C & \\ \multicolumn{2}{l}{\text{MSig}(\texttt{approx}\ C, m)\ = T\ m(\overline{T_i\ pid}^{\,i})\ \texttt{approx}}\\ \texttt{lost}\notin\overline{T_i}^{\,i} & {}^s\Gamma \vdash \overline{e_i}^{\,i} : \overline{T_i}^{\,i}\end{array}}{{}^s\Gamma \vdash e_0\texttt{.}m(\overline{e_i}^{\,i}) : T}\ \text{TR\_CALL2}$$

$$\frac{\begin{array}{l} {}^s\Gamma \vdash e_0 : \texttt{approx}\ C \\ \text{MSig}(\texttt{approx}\ C, m) = None \\ \text{MSig}(\texttt{precise}\ C, m) = T\ m(\overline{T_i\ pid}^{\,i})\ \texttt{precise} \\ \texttt{lost} \notin \overline{T_i}^{\,i} \qquad {}^s\Gamma \vdash \overline{e_i}^{\,i} : \overline{T_i}^{\,i} \end{array}}{{}^s\Gamma \vdash e_0.m(\overline{e_i}^{\,i}) : T} \quad \text{TR\_CALL3}$$

$$\frac{{}^s\Gamma \vdash e : \_ \qquad q\ C\ \ \mathsf{OK}}{{}^s\Gamma \vdash (q\ C)\ e : T} \quad \text{TR\_CAST}$$

$$\frac{{}^s\Gamma \vdash e_0 : q\ P \qquad {}^s\Gamma \vdash e_1 : q\ P}{{}^s\Gamma \vdash e_0 \oplus e_1 : q\ P} \quad \text{TR\_PRIMOP}$$

$$\frac{{}^s\Gamma \vdash e_0 : \texttt{precise}\ P \qquad {}^s\Gamma \vdash e_1 : T \qquad {}^s\Gamma \vdash e_2 : T}{{}^s\Gamma \vdash \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} : T} \quad \text{TR\_COND}$$

$\boxed{{}^s\Gamma \vdash \overline{e} : \overline{T}}$ expression typings

$$\frac{\overline{{}^s\Gamma \vdash e_k : T_k}^{\,k}}{{}^s\Gamma \vdash \overline{e_k}^{\,k} : \overline{T_k}^{\,k}} \quad \text{TRM\_DEF}$$

$\boxed{T\ \ \mathsf{OK}}$ well-formed type

$$\frac{\texttt{class}\ C \ldots \in Prg}{q\ C\ \ \mathsf{OK}} \quad \text{WFT\_REFT}$$

$$\frac{}{q\ P\ \ \mathsf{OK}} \quad \text{WFT\_PRIMT}$$

$\boxed{\overline{T}\ \ \mathsf{OK}}$ well-formed types

$$\frac{\overline{T_k\ \ \mathsf{OK}}^{\,k}}{\overline{T_k}^{\,k}\ \ \mathsf{OK}} \quad \text{WFTS\_DEF}$$

$\boxed{Cls\ \ \mathsf{OK}}$ well-formed class declaration

$$\frac{\begin{array}{l} {}^s\Gamma = \{\texttt{this} \mapsto \texttt{context}\ Cid\} \\ {}^s\Gamma \vdash \overline{fd}\ \ \mathsf{OK} \qquad {}^s\Gamma, Cid \vdash \overline{md}\ \ \mathsf{OK} \\ \texttt{class}\ C \ldots \in Prg \end{array}}{\texttt{class}\ Cid\ \texttt{extends}\ C\ \{\ \overline{fd}\ \overline{md}\ \}\ \ \mathsf{OK}} \quad \text{WFC\_DEF}$$

$$\frac{}{\texttt{class Object}\ \{\}\ \ \mathsf{OK}} \quad \text{WFC\_OBJECT}$$

$\boxed{{}^s\Gamma \vdash T\ f;\ \ \mathsf{OK}}$ well-formed field declaration

$$\frac{T\ \ \mathsf{OK}}{{}^s\Gamma \vdash T\ f;\ \ \mathsf{OK}} \quad \text{WFFD\_DEF}$$

$\boxed{{}^s\Gamma \vdash \overline{fd}\ \ \mathsf{OK}}$ well-formed field declarations

$$\frac{\overline{{}^s\Gamma \vdash T_i\ f_i;\ \ \mathsf{OK}}^{\,i}}{{}^s\Gamma \vdash \overline{T_i\ f_i;}^{\,i}\ \ \mathsf{OK}} \quad \text{WFFDS\_DEF}$$

$\boxed{{}^s\Gamma, C \vdash md\ \ \mathsf{OK}}$ well-formed method declaration

$$\frac{\begin{array}{l} {}^s\Gamma = \{\texttt{this} \mapsto \texttt{context}\ C\} \\ {}^s\Gamma' = \left\{\texttt{this} \mapsto \texttt{context}\ C,\ \overline{pid \mapsto T_i}^{\,i}\right\} \\ T,\ \overline{T_i}^{\,i}\ \ \mathsf{OK} \qquad {}^s\Gamma' \vdash e : T \qquad C \vdash m\ \ \mathsf{OK} \\ q \in \{\texttt{precise}, \texttt{approx}\} \end{array}}{{}^s\Gamma, C \vdash T\ m(\overline{T_i\ pid}^{\,i})\ q\ \{\ e\ \}\ \ \mathsf{OK}} \quad \text{WFMD\_DEF}$$

$\boxed{{}^s\Gamma, C \vdash \overline{md}\ \ \mathsf{OK}}$ well-formed method declarations

$$\frac{\overline{{}^s\Gamma, C \vdash md_k\ \ \mathsf{OK}}^{\,k}}{{}^s\Gamma, C \vdash \overline{md_k}^{\,k}\ \ \mathsf{OK}} \quad \text{WFMDS\_DEF}$$

$\boxed{C \vdash m\ \ \mathsf{OK}}$ method overriding OK

$$\frac{C \sqsubseteq C' \Longrightarrow C, C' \vdash m\ \ \mathsf{OK}}{C \vdash m\ \ \mathsf{OK}} \quad \text{OVR\_DEF}$$

$\boxed{C, C' \vdash m\ \ \mathsf{OK}}$ method overriding OK auxiliary

$$\frac{\begin{array}{l} \mathrm{MSig}(C, m, \texttt{precise}) = ms_0\ \wedge\ \mathrm{MSig}(C', m, \texttt{precise}) = ms'_0\ \wedge\ (ms'_0 = \mathit{None}\ \vee\ ms_0 = ms'_0) \\ \mathrm{MSig}(C, m, \texttt{approx}) = ms_1\ \wedge\ \mathrm{MSig}(C', m, \texttt{approx}) = ms'_1\ \wedge\ (ms'_1 = \mathit{None}\ \vee\ ms_1 = ms'_1) \\ \mathrm{MSig}(C, m, \texttt{precise}) = ms_2\ \wedge\ \mathrm{MSig}(C', m, \texttt{approx}) = ms'_2\ \wedge\ (ms'_2 = \mathit{None}\ \vee\ ms_2 <:\ ms'_2) \end{array}}{C, C' \vdash m\ \ \mathsf{OK}} \quad \text{OVRA\_DEF}$$

$\boxed{{}^s\Gamma\ \ \mathsf{OK}}$ well-formed static environment

$$\frac{\begin{array}{l} {}^s\Gamma = \left\{\texttt{this} \mapsto q\ C,\ \overline{pid \mapsto T_i}^{\,i}\right\} \\ q\ C,\ \overline{T_i}^{\,i}\ \ \mathsf{OK} \end{array}}{{}^s\Gamma\ \ \mathsf{OK}} \quad \text{SWFE\_DEF}$$

$\boxed{\vdash \mathit{Prg}\ \ \mathsf{OK}}$ well-formed program

$$\frac{\begin{array}{l} \mathit{Prg} = \overline{\mathit{Cls}_i}^{\,i},\ C,\ e \\ \overline{\mathit{Cls}_i\ \ \mathsf{OK}}^{\,i} \qquad \texttt{context}\ C\ \ \mathsf{OK} \\ \{\texttt{this} \mapsto \texttt{context}\ C\} \vdash e : \_ \\ \forall C', C''.\ ((C' \sqsubseteq C''\ \wedge\ C'' \sqsubseteq C') \Longrightarrow C' = C'') \end{array}}{\vdash \mathit{Prg}\ \ \mathsf{OK}} \quad \text{WFP\_DEF}$$

$\boxed{h + o\ =\ (h', \iota)}$ add object o to heap h resulting in heap h' and fresh address ι

$$\frac{\iota \notin \mathrm{dom}(h) \qquad h' = h \oplus (\iota \mapsto o)}{h + o\ =\ (h', \iota)} \quad \text{HNEW\_DEF}$$

$\boxed{h[\iota.f\ :=\ v]\ = h'}$ field update in heap

$$\frac{\begin{array}{l} v = \texttt{null}_a\ \vee\ (v = \iota'\ \wedge\ \iota' \in \mathrm{dom}(h)) \\ h(\iota) = \left(T, \overline{fv}\right) \qquad f \in \mathrm{dom}\left(\overline{fv}\right) \qquad \overline{fv}' = \overline{fv}[f \mapsto v] \\ h' = h \oplus \left(\iota \mapsto \left(T, \overline{fv}'\right)\right) \end{array}}{h[\iota.f\ :=\ v]\ = h'} \quad \text{HUP\_REFT}$$

$$\frac{\begin{array}{l} h(\iota) = \left(T, \overline{fv}\right) \qquad \overline{fv}(f) = (q', {}^r\mathcal{L}') \\ \overline{fv}' = \overline{fv}[f \mapsto (q', {}^r\mathcal{L})] \qquad h' = h \oplus \left(\iota \mapsto \left(T, \overline{fv}'\right)\right) \end{array}}{h[\iota.f\ :=\ (q, {}^r\mathcal{L})]\ = h'} \quad \text{HUP\_PRIMT}$$

$\boxed{\text{sTrT}(h, \iota, T) = T'}$ convert type T to its runtime equivalent T'

$$\frac{\begin{array}{c} q{=}\texttt{context} \Longrightarrow q'{=}\text{TQual}(h(\iota){\downarrow_1}) \\ q{\neq}\texttt{context} \Longrightarrow q'{=}q \end{array}}{\text{sTrT}(h, \iota, q\ C) = q'\ C} \quad \text{sTrT\_refT}$$

$$\frac{\begin{array}{c} q{=}\texttt{context} \Longrightarrow q'{=}\text{TQual}(h(\iota){\downarrow_1}) \\ q{\neq}\texttt{context} \Longrightarrow q'{=}q \end{array}}{\text{sTrT}(h, \iota, q\ P) = q'\ P} \quad \text{sTrT\_primT}$$

$\boxed{h, \iota \vdash v : T}$ type T assignable to value v

$$\frac{\begin{array}{c} \text{sTrT}(h, \iota_0, q\ C) = q'\ C \\ h(\iota){\downarrow_1} = T_1 \qquad T_1 <:\ q'\ C \end{array}}{h, \iota_0 \vdash \iota :\ q\ C} \quad \text{rtt\_addr}$$

$$\frac{}{h, \iota_0 \vdash \texttt{null}_a :\ q\ C} \quad \text{rtt\_null}$$

$$\frac{\begin{array}{c} \text{sTrT}(h, \iota_0, q'\ P) = q''\ P \\ {}^{r}\mathcal{L} \in P \qquad q\ P <:\ q''\ P \end{array}}{h, \iota_0 \vdash (q, {}^{r}\mathcal{L}) :\ q'\ P} \quad \text{rtt\_primT}$$

$\boxed{h, \iota \vdash \overline{v} : \overline{T}}$ types $\overline{T}$ assignable to values $\overline{v}$

$$\frac{\overline{h, \iota \vdash v_i :\ T_i}^{\,i}}{h, \iota \vdash \overline{v_i}^{\,i} :\ \overline{T_i}^{\,i}} \quad \text{rtts\_def}$$

$\boxed{\text{FType}(h, \iota, f) = T}$ look up type of field in heap

$$\frac{h, \iota \vdash \iota :\ q\ C \qquad \text{FType}(q\ C, f) = T}{\text{FType}(h, \iota, f) = T} \quad \text{rft\_def}$$

$\boxed{\text{MSig}(h, \iota, m) = ms}$ look up method signature of method m at ι

$$\frac{h, \iota \vdash \iota :\ q\ C \qquad \text{MSig}(q\ C, m) = ms}{\text{MSig}(h, \iota, m) = ms} \quad \text{rms\_def}$$

$\boxed{\text{MBody}(C, m, q) = e}$ look up most-concrete body of m, q in class C or a superclass

$$\frac{\begin{array}{c} \texttt{class}\ Cid\ \texttt{extends}\ \_\ \{\ \_\ \_\ ms\ \{\ e\ \}\ \_\ \} \in Prg \\ \text{MName}(ms){=}m \ \wedge\ \text{MQual}(ms){=}q \end{array}}{\text{MBody}(Cid, m, q) = e} \quad \text{smbc\_found}$$

$$\frac{\begin{array}{c} \texttt{class}\ Cid\ \texttt{extends}\ C_1\ \{\ \_\ \overline{ms_n\ \{\ e_n\ \}}^{\,n}\ \} \in Prg \\ \overline{\text{MName}(ms_n){\neq}m}^{\,n} \qquad \text{MBody}(C_1, m, q) = e \end{array}}{\text{MBody}(Cid, m, q) = e} \quad \text{smbc\_inh}$$

$\boxed{\text{MBody}(h, \iota, m) = e}$ look up most-concrete body of method m at ι

$$\frac{h(\iota){\downarrow_1} = \texttt{precise}\ C \qquad \text{MBody}(C, m, \texttt{precise}) = e}{\text{MBody}(h, \iota, m) = e} \quad \text{rmb\_call1}$$

$$\frac{h(\iota){\downarrow_1} = \texttt{approx}\ C \qquad \text{MBody}(C, m, \texttt{approx}) = e}{\text{MBody}(h, \iota, m) = e} \quad \text{rmb\_call2}$$

$$\frac{\begin{array}{l} h(\iota)\downarrow_1 = \texttt{approx}\ C \qquad \mathrm{MBody}(C, m, \texttt{approx}) = \mathit{None} \\ \mathrm{MBody}(C, m, \texttt{precise}) = e \end{array}}{\mathrm{MBody}(h, \iota, m) = e} \quad \text{RMB\_CALL3}$$

$\boxed{\mathrm{FVsInit}(qC) = \overline{fv}}$ initialize the fields for reference type qC

$$\frac{\begin{array}{l} q \in \{\texttt{precise}, \texttt{approx}\} \\ \forall f \in \mathrm{refFields}(C)\,.\ \overline{fv}(f) = \texttt{null}_a \\ \forall f \in \mathrm{primFields}(C)\,.\ (\mathrm{FType}(q\ C, f) = q'\ P \ \wedge\ \overline{fv}(f) = (q', 0)) \end{array}}{\mathrm{FVsInit}(q\ C) = \overline{fv}} \quad \text{FVSI\_DEF}$$

$\boxed{{}^{r}\Gamma \vdash h, e \rightsquigarrow h', v}$ big-step operational semantics

$$\frac{}{{}^{r}\Gamma \vdash h, \texttt{null} \rightsquigarrow h, \texttt{null}_a} \quad \text{OS\_NULL}$$

$$\frac{}{{}^{r}\Gamma \vdash h, \mathcal{L} \rightsquigarrow h, (\texttt{precise}, {}^{r}\mathcal{L})} \quad \text{OS\_LITERAL}$$

$$\frac{{}^{r}\Gamma(x) = v}{{}^{r}\Gamma \vdash h, x \rightsquigarrow h, v} \quad \text{OS\_VAR}$$

$$\frac{\begin{array}{l} \mathrm{sTrT}(h, {}^{r}\Gamma(\texttt{this}), q\ C) = q'\ C \\ \mathrm{FVsInit}(q'\ C) = \overline{fv} \\ h + \left(q'\ C, \overline{fv}\right) = (h', \iota) \end{array}}{{}^{r}\Gamma \vdash h, \texttt{new}\ q\ C\texttt{()} \rightsquigarrow h', \iota} \quad \text{OS\_NEW}$$

$$\frac{{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h', \iota_0 \qquad h'(\iota_0 . f) = v}{{}^{r}\Gamma \vdash h, e_0 . f \rightsquigarrow h', v} \quad \text{OS\_READ}$$

$$\frac{\begin{array}{l} {}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, \iota_0 \qquad {}^{r}\Gamma \vdash h_0, e_1 \rightsquigarrow h_1, v \\ h_1[\iota_0 . f := v] = h' \end{array}}{{}^{r}\Gamma \vdash h, e_0 . f := e_1 \rightsquigarrow h', v} \quad \text{OS\_WRITE}$$

$$\frac{\begin{array}{l} {}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, \iota_0 \qquad {}^{r}\Gamma \vdash h_0, \overline{e_i}^{\,i} \rightsquigarrow h_1, \overline{v_i}^{\,i} \\ \mathrm{MBody}(h_0, \iota_0, m) = e \qquad \mathrm{MSig}(h_0, \iota_0, m) = \_\ m(\overline{\_\ pid}^{\,i})\ q \\ {}^{r}\Gamma' = \left\{\texttt{precise}; \texttt{this} \mapsto \iota_0,\ \overline{pid \mapsto v_i}^{\,i}\right\} \\ {}^{r}\Gamma' \vdash h_1, e \rightsquigarrow h', v \end{array}}{{}^{r}\Gamma \vdash h, e_0 . m(\overline{e_i}^{\,i}) \rightsquigarrow h', v} \quad \text{OS\_CALL}$$

$$\frac{\begin{array}{l} {}^{r}\Gamma \vdash h, e \rightsquigarrow h', v \\ h', {}^{r}\Gamma(\texttt{this}) \vdash v : q\ C \end{array}}{{}^{r}\Gamma \vdash h, \texttt{(}q\ C\texttt{)}\ e \rightsquigarrow h', v} \quad \text{OS\_CAST}$$

$$\frac{\begin{array}{l} {}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, (q, {}^{r}\mathcal{L}_0) \\ {}^{r}\Gamma \vdash h_0, e_1 \rightsquigarrow h', (q, {}^{r}\mathcal{L}_1) \end{array}}{{}^{r}\Gamma \vdash h, e_0 \oplus e_1 \rightsquigarrow h', (q, {}^{r}\mathcal{L}_0 \oplus {}^{r}\mathcal{L}_1)} \quad \text{OS\_PRIMOP}$$

$$\frac{\begin{array}{l} {}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, (q, {}^{r}\mathcal{L}) \qquad {}^{r}\mathcal{L} \neq 0 \\ {}^{r}\Gamma \vdash h_0, e_1 \rightsquigarrow h', v \end{array}}{{}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \rightsquigarrow h', v} \quad \text{OS\_COND\_T}$$

$$\frac{{}^{r}\Gamma \vdash h, e_0 \rightsquigarrow h_0, (q, 0) \qquad {}^{r}\Gamma \vdash h_0, e_2 \rightsquigarrow h', v}{{}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \rightsquigarrow h', v} \quad \text{OS\_COND\_F}$$

$$\frac{{}^{r}\Gamma \vdash h, e \rightsquigarrow h', v \qquad h' \cong \tilde{h}' \qquad v \cong \tilde{v}}{{}^{r}\Gamma \vdash h, e \rightsquigarrow \tilde{h}', \tilde{v}} \quad \text{OS\_APPROX}$$

$\boxed{{}^r\Gamma \vdash h, \overline{e} \ \rightsquigarrow \ h', \overline{v}}$ sequential big-step operational semantics

$$\frac{\begin{array}{c} {}^r\Gamma \vdash h, e \ \rightsquigarrow \ h_0, v \\ {}^r\Gamma \vdash h_0, \overline{e_i}^{\,i} \ \rightsquigarrow \ h', \overline{v_i}^{\,i} \end{array}}{{}^r\Gamma \vdash h, e, \overline{e_i}^{\,i} \ \rightsquigarrow \ h', v, \overline{v_i}^{\,i}} \quad \text{OSS\_DEF}$$

$$\frac{}{{}^r\Gamma \vdash h, \emptyset \ \rightsquigarrow \ h, \emptyset} \quad \text{OSS\_EMPTY}$$

$\boxed{\vdash Prg \ \rightsquigarrow \ h, v}$ big-step operational semantics of a program

$$\frac{\begin{array}{l} \text{FVsInit}(\texttt{precise}\ C) = \overline{fv} \\ \emptyset + \left(\texttt{precise}\ C, \overline{fv}\right) \ = \ (h_0, \iota_0) \\ {}^r\Gamma_0 {=} \{\texttt{precise}; \texttt{this} \mapsto \iota_0\} \qquad {}^r\Gamma_0 \vdash h_0, e \ \rightsquigarrow \ h, v \end{array}}{\vdash \overline{Cls}, \ C, \ e \ \rightsquigarrow \ h, v} \quad \text{OSP\_DEF}$$

$\boxed{{}^r\Gamma \vdash h, e \ \rightsquigarrow_c \ h', v}$ checked big-step operational semantics

$$\frac{{}^r\Gamma \vdash h, \texttt{null} \ \rightsquigarrow \ h, \texttt{null}_a}{{}^r\Gamma \vdash h, \texttt{null} \ \rightsquigarrow_c \ h, \texttt{null}_a} \quad \text{COS\_NULL}$$

$$\frac{{}^r\Gamma \vdash h, \mathcal{L} \ \rightsquigarrow \ h, (\texttt{precise}, {}^r\mathcal{L})}{{}^r\Gamma \vdash h, \mathcal{L} \ \rightsquigarrow_c \ h, (\texttt{precise}, {}^r\mathcal{L})} \quad \text{COS\_LITERAL}$$

$$\frac{{}^r\Gamma \vdash h, x \ \rightsquigarrow \ h, v}{{}^r\Gamma \vdash h, x \ \rightsquigarrow_c \ h, v} \quad \text{COS\_VAR}$$

$$\frac{{}^r\Gamma \vdash h, \texttt{new}\ q\ C\texttt{()} \ \rightsquigarrow \ h', \iota}{{}^r\Gamma \vdash h, \texttt{new}\ q\ C\texttt{()} \ \rightsquigarrow_c \ h', \iota} \quad \text{COS\_NEW}$$

$$\frac{\begin{array}{c} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h', \iota_0 \\ {}^r\Gamma \vdash h, e_0.f \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, e_0.f \ \rightsquigarrow_c \ h', v} \quad \text{COS\_READ}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, \iota_0 \qquad h(\iota_0){\downarrow_1} = q\ C \\ {}^r\Gamma{\downarrow_1} = q' \qquad (q{=}q' \ \vee \ q'{=}\texttt{precise}) \\ {}^r\Gamma \vdash h_0, e_1 \ \rightsquigarrow_c \ h_1, v \\ {}^r\Gamma \vdash h, e_0.f \ := \ e_1 \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, e_0.f \ := \ e_1 \ \rightsquigarrow_c \ h', v} \quad \text{COS\_WRITE}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, \iota_0 \qquad\qquad {}^r\Gamma \vdash h_0, \overline{e_i}^{\,i} \ \rightsquigarrow_c \ h_1, \overline{v_i}^{\,i} \\ \text{MBody}(h_0, \iota_0, m) \ = \ e \qquad \text{MSig}(h_0, \iota_0, m) \ = \ \_\ m(\overline{\_\ pid}^{\,i})\ q \\ {}^r\Gamma' {=} \left\{\texttt{precise}; \texttt{this} \mapsto \iota_0, \ \overline{pid \mapsto v_i}^{\,i}\right\} \\ {}^r\Gamma' \vdash h_1, e \ \rightsquigarrow_c \ h', v \\ {}^r\Gamma \vdash h, e_0.m(\overline{e_i}^{\,i}) \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, e_0.m(\overline{e_i}^{\,i}) \ \rightsquigarrow_c \ h', v} \quad \text{COS\_CALL}$$

$$\frac{\begin{array}{c} {}^r\Gamma \vdash h, e \ \rightsquigarrow_c \ h', v \\ {}^r\Gamma \vdash h, \texttt{(}q\ C\texttt{)}\ e \ \rightsquigarrow \ h', v \end{array}}{{}^r\Gamma \vdash h, \texttt{(}q\ C\texttt{)}\ e \ \rightsquigarrow_c \ h', v} \quad \text{COS\_CAST}$$

$$\frac{\begin{array}{l} {}^r\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, (q, {}^r\mathcal{L}_0) \\ {}^r\Gamma \vdash h_0, e_1 \ \rightsquigarrow_c \ h', (q, {}^r\mathcal{L}_1) \\ {}^r\Gamma \vdash h, e_0 \oplus e_1 \ \rightsquigarrow \ h', (q, {}^r\mathcal{L}_0 \oplus {}^r\mathcal{L}_1) \end{array}}{{}^r\Gamma \vdash h, e_0 \oplus e_1 \ \rightsquigarrow_c \ h', (q, {}^r\mathcal{L}_0 \oplus {}^r\mathcal{L}_1)} \quad \text{COS\_PRIMOP}$$

$$\frac{\begin{array}{c} {}^{r}\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, (q, {}^{r}\mathcal{L}) \qquad {}^{r}\mathcal{L}{\neq}0 \\ {}^{r}\Gamma'{=}{}^{r}\Gamma(q) \qquad {}^{r}\Gamma' \vdash h_0, e_1 \ \rightsquigarrow_c \ h', v \\ {}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow \ h', v \end{array}}{{}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow_c \ h', v} \quad \text{COS\_COND\_T}$$

$$\frac{\begin{array}{c} {}^{r}\Gamma \vdash h, e_0 \ \rightsquigarrow_c \ h_0, (q, {}^{r}\mathcal{L}) \qquad {}^{r}\mathcal{L}{=}0 \\ {}^{r}\Gamma'{=}{}^{r}\Gamma(q) \qquad {}^{r}\Gamma' \vdash h_0, e_2 \ \rightsquigarrow_c \ h', v \\ {}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow \ h', v \end{array}}{{}^{r}\Gamma \vdash h, \texttt{if}(e_0)\ \{e_1\}\ \texttt{else}\ \{e_2\} \ \rightsquigarrow_c \ h', v} \quad \text{COS\_COND\_F}$$

$\boxed{{}^{r}\Gamma \vdash h, \overline{e} \ \rightsquigarrow_c \ h', \overline{v}}$ checked sequential big-step operational semantics

$$\frac{\begin{array}{c} {}^{r}\Gamma \vdash h, e \ \rightsquigarrow_c \ h_0, v \\ {}^{r}\Gamma \vdash h_0, \overline{e_i}^{\,i} \ \rightsquigarrow_c \ h', \overline{v_i}^{\,i} \end{array}}{{}^{r}\Gamma \vdash h, e, \overline{e_i}^{\,i} \ \rightsquigarrow_c \ h', v, \overline{v_i}^{\,i}} \quad \text{COSS\_DEF}$$

$$\frac{}{{}^{r}\Gamma \vdash h, \emptyset \ \rightsquigarrow_c \ h, \emptyset} \quad \text{COSS\_EMPTY}$$

$\boxed{h \ \ \mathsf{OK}}$ well-formed heap

$$\frac{\begin{array}{c} \forall \iota \in \mathrm{dom}(h), f \in h(\iota){\downarrow_2} .\ (\mathrm{FType}(h, \iota, f) = T \ \wedge \ h, \iota \vdash h(\iota.f) \ : \ T) \\ \forall \iota \in \mathrm{dom}(h) .\ (h(\iota){\downarrow_1} \ \ \mathsf{OK} \ \wedge \ \mathrm{TQual}(h(\iota){\downarrow_1}) \in \{\texttt{precise}, \texttt{approx}\}) \end{array}}{h \ \ \mathsf{OK}} \quad \text{WFH\_DEF}$$

$\boxed{h, {}^{r}\Gamma : {}^{s}\Gamma \ \ \mathsf{OK}}$ runtime and static environments correspond

$$\frac{\begin{array}{l} {}^{r}\Gamma{=}\left\{\texttt{precise}; \texttt{this} \mapsto \iota, \overline{pid \mapsto v_i}^{\,i}\right\} \\ {}^{s}\Gamma{=}\left\{\texttt{this} \mapsto \texttt{context}\ C, \overline{pid \mapsto T_i}^{\,i}\right\} \\ \quad h \ \ \mathsf{OK} \qquad\qquad {}^{s}\Gamma \ \ \mathsf{OK} \\ h, \iota \vdash \iota : \ \texttt{context}\ C \\ h, \iota \vdash \overline{v_i}^{\,i} : \ \overline{T_i}^{\,i} \end{array}}{h, {}^{r}\Gamma : {}^{s}\Gamma \ \ \mathsf{OK}} \quad \text{WFRSE\_DEF}$$