1 Grammar

$$
\begin{array}{lll}
e ::= & x & \textit{expressions} \\
\mid & r & \\
\mid & \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} & \\
\mid & \texttt{new}_d\ x \Rightarrow \overline{d = e} & \\
\mid & e.m(e) & \\
\mid & e.\pi & \\
 & & \\
\tau ::= & \{\bar{\sigma}\} & \textit{types} \\
\mid & \{\bar{r}\} & \\
\mid & \{\bar{d}\} & \\
\mid & \{\bar{d}\ \texttt{captures}\ \varepsilon\} & \\
 & & \\
\sigma ::= & d\ \texttt{with}\ \varepsilon & \textit{labeled decls.} \\
 & & \\
d ::= & \texttt{def}\ m(x : \tau) : \tau & \textit{unlabeled decls.}
\end{array}
$$

Notes:

- σ denotes a declaration with effect labels; d a declaration without effect labels.
- $\texttt{new}_\sigma$ is for creating annotated objects; $\texttt{new}_d$ for unannotated objects.
- $\{\bar{\sigma}\}$ is the type of an annotated object. $\{\bar{d}\}$ is the type of an unannotated object.
- $\{\bar{d}\ \texttt{captures}\ \varepsilon\}$ is a special kind of type that doesn't appear in source programs but may be assigned by the new rules in this section. Intuitively, ε is an upper-bound on the effects captured by $\{\bar{d}\}$.

2 Semantics

2.1 Static Semantics

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau}\ \text{(T-Var)} \qquad \frac{}{\Gamma,\ r : \{\bar{r}\} \vdash r : \{\bar{r}\}}\ \text{(T-Resource)}$$

$$\frac{\Gamma \vdash e_1 : \{\bar{r}\}}{\Gamma \vdash e_1.\pi : \texttt{Unit}}\ \text{(T-OperCall)}$$

$$\frac{\Gamma \vdash e_1 : \{\bar{\sigma}\} \quad \texttt{def}\ m(y : \tau_2) : \tau_3\ \texttt{with}\ \varepsilon_3 \in \{\bar{\sigma}\} \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1.m(e_2) : \tau_3}\ \text{(T-MethCall}_\sigma\text{)}$$

$$\frac{\Gamma \vdash e_1 : \{\bar{d}\} \quad \texttt{def}\ m(y : \tau_2) : \tau_3 \in \{\bar{d}\} \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1.m(e_2) : \tau_3}\ \text{(T-MethCall}_d\text{)}$$

$$\frac{\Gamma \vdash \sigma_i = e_i\ \texttt{OK}}{\Gamma \vdash\ \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} : \{\bar{\sigma}\}}\ \text{(T-New}_\sigma\text{)} \qquad \frac{\Gamma \vdash d_i = e_i\ \texttt{OK}}{\Gamma \vdash\ \texttt{new}_d\ x \Rightarrow \overline{d = e} : \{\bar{d}\}}\ \text{(T-New}_d\text{)}$$

$\boxed{\Gamma \vdash d = e\ \texttt{OK}}$

$$\frac{d = \texttt{def}\ m(y : \tau_2) : \tau_3 \quad \Gamma, y : \tau_2 \vdash e : \tau_3}{\Gamma \vdash d = e\ \texttt{OK}}\ \text{(}\varepsilon\text{-ValidImpl}_d\text{)}$$

$\boxed{\Gamma \vdash \sigma = e \texttt{ OK}}$

$$\frac{\Gamma,\ y : \tau_2 \vdash e : \tau_3 \texttt{ with } \varepsilon_3 \quad \sigma = \texttt{def } m(y : \tau_2) : \tau_3 \texttt{ with } \varepsilon_3}{\Gamma \vdash \sigma = e \texttt{ OK}}\ (\varepsilon\text{-ValidImpl}_\sigma)$$

$\boxed{\Gamma \vdash e : \tau \texttt{ with } \varepsilon}$

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau \texttt{ with } \varnothing}\ (\varepsilon\text{-Var}) \qquad \frac{}{\Gamma,\ r : \{\bar{r}\} \vdash r : \{\bar{r}\} \texttt{ with } \varnothing}\ (\varepsilon\text{-Resource})$$

$$\frac{\Gamma,\ x : \{\bar{\sigma}\} \vdash \overline{\sigma = e} \texttt{ OK}}{\Gamma \vdash \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} : \{\bar{\sigma}\} \texttt{ with } \varnothing}\ (\varepsilon\text{-NewObj}) \qquad \frac{\Gamma \vdash e_1 : \{\bar{r}\} \texttt{ with } \varepsilon_1}{\Gamma \vdash e_1.\pi : \texttt{Unit with } \{\bar{r}.\pi\} \cup \varepsilon_1}\ (\varepsilon\text{-OperCall})$$

$$\frac{\Gamma \vdash e_1 : \{\bar{\sigma}\} \texttt{ with } \varepsilon_1 \quad \Gamma \vdash e_2 : \tau_2 \texttt{ with } \varepsilon_2 \quad \sigma_i = \texttt{def } m_i(y : \tau_2) : \tau_3 \texttt{ with } \varepsilon_3}{\Gamma \vdash e_1.m_i(e_2) : \tau_3 \texttt{ with } \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3}\ (\varepsilon\text{-MethCall})$$

$$\frac{\varepsilon_c = \texttt{effects}(\Gamma') \quad \Gamma' \subseteq \Gamma \quad \Gamma', x : \{\bar{d} \texttt{ captures } \varepsilon_c\} \vdash \overline{d = e} \texttt{ OK}}{\Gamma \vdash \ \texttt{new}_d\ x \Rightarrow \overline{d = e} : \{\bar{d} \texttt{ captures } \varepsilon_c\} \texttt{ with } \varnothing}\ (\text{C-NewObj})$$

$$\frac{\Gamma \vdash e_1 : \{\bar{d} \texttt{ captures } \varepsilon_c\} \texttt{ with } \varepsilon_1 \quad \Gamma \vdash e_2 : \tau_2 \texttt{ with } \varepsilon_2 \quad d_i = \ \texttt{def } m_i(y : \tau_2) : \tau_3}{\Gamma \vdash e_1.m_i(e_2) : \tau_3 \texttt{ with } \varepsilon_1 \cup \varepsilon_2 \cup \texttt{effects}(\tau_2) \cup \varepsilon_c}\ (\text{C-MethCall})$$

$$\frac{\Gamma' \subseteq \Gamma \quad \Gamma' \vdash e : \tau}{\Gamma \vdash e : \tau \texttt{ with effects}(\Gamma')}\ (\text{C-Inference})$$

Notes:

- This system includes all the rules from the fully-annotated system.
- The T rules do standard typing of objects, without any effect analysis. Their sole purpose is so ε-ValidImpl$_d$ can be applied. **We are assuming the T-rules on their own are sound**.
- In C-NewObj, Γ' is intended to be some subcontext of the current Γ. The object is labelled as capturing the effects in Γ' (exact definition in the next section).
- In C-NewObj we must add $\texttt{effects}(\tau_2)$ to the static effects of the object, because the method body will have access to the resources captured by τ_2 (the type of the argument passed into the method).
- A good choice of Γ' would be Γ restricted to the free variables in the object definition.
- The purpose of C-Inference is to ascribe static effects to unannotated portions of code (for instance, the body of an unlabeled method).
- As a useful convention we'll often use ε_c to denote the output of the `effects` function.

2.2 `effects` Function

The `effects` function returns the set of effects captured in a particular context.

- $\texttt{effects}(\varnothing) = \varnothing$
- $\texttt{effects}(\Gamma, x : \tau) = \texttt{effects}(\Gamma) \cup \texttt{effects}(\tau)$
- $\texttt{effects}(\{\bar{r}\}) = \{(r, \pi) \mid r \in \bar{r}, \pi \in \Pi\}$
- $\texttt{effects}(\{\bar{\sigma}\}) = \bigcup_{\sigma \in \bar{\sigma}} \texttt{effects}(\sigma)$
- $\texttt{effects}(\{\bar{d}\}) = \bigcup_{d \in \bar{d}} \texttt{effects}(d)$

- $\texttt{effects}(d\ \texttt{with}\ \varepsilon) = \varepsilon \cup \texttt{effects}(d)$
- $\texttt{effects}(\texttt{def}\ \texttt{m}(x : \tau_1) : \tau_2) = \texttt{effects}(\tau_2)$
- $\texttt{effects}(\{\bar{d}\ \texttt{captures}\ \varepsilon_c\}) = \varepsilon_c$

Notes:

- Since a method can return a capability for a resource r we need to figure out what the return type of a method captures. This requires a recursive crawl through the definitions and types inside it.
- In the last case we don't want to recurse to sub-declarations because the effects have already been captured previously (this is ε_c) by a potentially different context.

2.3 Dynamic Semantics

$\boxed{e \longrightarrow e \mid \varepsilon}$

$$\frac{e_1 \longrightarrow e_1' \mid \varepsilon}{e_1.m(e_2) \longrightarrow e_1'.m(e_2) \mid \varepsilon}\ \text{(E-MethCall1)}$$

$$\frac{v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} \quad e_2 \longrightarrow e_2' \mid \varepsilon}{v_1.m(e_2) \longrightarrow v_1.m(e_2') \mid \varepsilon}\ \text{(E-MethCall2}_\sigma\text{)} \qquad \frac{v_1 = \texttt{new}_d\ x \Rightarrow \overline{d = e} \quad e_2 \longrightarrow e_2' \mid \varepsilon}{v_1.m(e_2) \longrightarrow v_1.m(e_2') \mid \varepsilon}\ \text{(E-MethCall2}_d\text{)}$$

$$\frac{v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} \quad \texttt{def}\ \texttt{m}(y : \tau_1) : \tau_2\ \texttt{with}\ \varepsilon = e \in \overline{\sigma = e}}{v_1.m(v_2) \longrightarrow [v_1/x, v_2/y]e \mid \varnothing}\ \text{(E-MethCall3}_\sigma\text{)}$$

$$\frac{v_1 = \texttt{new}_d\ x \Rightarrow \overline{d = e} \quad \texttt{def}\ \texttt{m}(y : \tau_1) : \tau_2 = e \in \overline{d = e}}{v_1.m(v_2) \longrightarrow [v_1/x, v_2/y]e \mid \varnothing}\ \text{(E-MethCall3}_d\text{)}$$

$$\frac{e_1 \longrightarrow e_1' \mid \varepsilon}{e_1.\pi \longrightarrow e_1'.\pi \mid \varepsilon}\ \text{(E-OperCall1)} \qquad \frac{}{r.\pi \longrightarrow \texttt{unit} \mid \{r.\pi\}}\ \text{(E-OperCall2)}$$

$\boxed{e \longrightarrow_* e \mid \varepsilon}$

$$\frac{}{e \longrightarrow_* e \mid \varnothing}\ \text{(E-MultiStep1)} \qquad \frac{e \longrightarrow e' \mid \varepsilon}{e \longrightarrow_* e' \mid \varepsilon}\ \text{(E-MultiStep2)}$$

$$\frac{e \longrightarrow_* e' \mid \varepsilon_1 \quad e' \longrightarrow_* e'' \mid \varepsilon_2}{e \longrightarrow_* e'' \mid \varepsilon_1 \cup \varepsilon_2}\ \text{(E-MultiStep3)}$$

Notes:

- E-MethCall2$_d$ and E-MethCall2$_\sigma$ are really doing the same thing, but one applies to labeled objects (the σ version) and the other on unlabeled objects. Same goes for E-MethCall3$_\sigma$ and E-MethCall3$_d$.
- E-MethCall1 can be used for both labeled and unlabeled objects.

2.4 Substitution Function

We extend our Substitution function from the previous system in a straightforward way by adding a new case for unlabeled objects.

- $[e'/z]z = e'$
- $[e'/z]y = y$, if $y \neq z$
- $[e'/z]r = r$
- $[e'/z](e_1.m(e_2)) = ([e'/z]e_1).m([e'/z]e_2)$
- $[e'/z](e_1.\pi) = ([e'/z]e_1).\pi$
- $[e'/z](\texttt{new}_d\ x \Rightarrow \overline{d = e}) = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = [e'/z]e}$, if $z \neq x$ and $z \notin \texttt{freevars}(e_i)$
- $[e'/z](\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}) = \texttt{new}_\sigma\ x \Rightarrow \sigma = [e'/z]e$, if $z \neq x$ and $z \notin \texttt{freevars}(e_i)$

3 Proofs

Lemma 3.1. (Canonical Forms)

Statement. Suppose e is a value. The following are true:

- If $\Gamma \vdash e : \{\bar{r}\}\ \texttt{with}\ \varepsilon$, then $e = r$ for some resource r.
- If $\Gamma \vdash e : \{\bar{\sigma}\}\ \texttt{with}\ \varepsilon$, then $e = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$.
- If $\Gamma \vdash e : \{\bar{d}\ \texttt{captures}\ \varepsilon_c\}\ \texttt{with}\ \varepsilon$, then $e = \texttt{new}_d\ x \Rightarrow \overline{d = e}$.

Furthermore, $\varepsilon = \varnothing$ in each case.

Proof. These typing judgements each appear exactly once in the conclusion of different rules. The result follows by inversion of ε-Resource, ε-NewObj, and C-NewObj respectively.

□

Lemma 3.2. (Substitution Lemma)

Statement. If $\Gamma, z : \tau' \vdash e : \tau\ \texttt{with}\ \varepsilon$, and $\Gamma \vdash e' : \tau'\ \texttt{with}\ \varepsilon'$, then $\Gamma \vdash [e'/z]e : \tau\ \texttt{with}\ \varepsilon$.

Intuition If you substitute z for something of the same type, the type of the whole expression stays the same after substitution.

Proof. We've already proven the lemma by structural induction on the ε rules. The new case is defined on a form not in the grammar for the fully-annotated system. So all that remains is to induct on derivations of $\Gamma \vdash e : \tau\ \texttt{with}\ \varepsilon$ using the new C rules.

Case. C-MethCall.
Then $e = e_1.m(e_2)$ and $[e'/z]e = ([e'/z]e_1).m([e'/z]e_2)$. By inductive assumption we know that e_1 and $[e'/z]e_1$ have the same types, and that e_2 and $[e'/z]e_2$ have the same types. Since e and $[e'/z]e$ have the same syntactic struture, and their corresponding subexpressions have the same types, then Γ can use C-MethCall to type $[e'/z]e$ the same as e.

Case. C-Inference.
Then $\Gamma \vdash e : \tau\ \texttt{with}\ \texttt{effects}(\Gamma')$, where $\Gamma' \subseteq \Gamma$. By inversion $\Gamma' \vdash e : \tau$. Applying the inductive hypothesis (and our assumption that the T rules are sonud) $\Gamma' \vdash [e'/z]e : \tau$. Since $\Gamma' \subseteq \Gamma'$ we have $\Gamma' \vdash [e'/z]e : \tau\ \texttt{with}\ \texttt{effects}\ (\Gamma')$ under C-Inference. Because $\Gamma' \subseteq \Gamma$ then $\Gamma \vdash [e'/z]e : \tau\ \texttt{with}\ \texttt{effects}\ (\Gamma')$.

Case. C-NewObj.
Then $e = \texttt{new}_d\ x \Rightarrow \overline{d = e}$. z appears in some method body e_i. By inversion we know $\Gamma, x : \{\bar{\sigma}\} \vdash \overline{d = e}\ \texttt{OK}$. The only rule with this conclusion is ε-ValidImpl$_d$; by inversion on that we know for each i that:
- $d_i = \texttt{def}\ m_i(y : \tau_1) : \tau_2\ \texttt{with}\ \varepsilon$
- $\Gamma, y : \tau_1 \vdash e_i : \tau_2\ \texttt{with}\ \varepsilon$

If z appears in the body of e_i then $\Gamma, z : \tau \vdash d_i = e_i\ \texttt{OK}$ by inductive assumption. Then we can use ε-ValidImpl$_d$ to conclude $\overline{d = [e'/z]e}\ \texttt{OK}$. This tells us that the types and static effects of all the methods are unchanged under substitution. By choosing the same $\Gamma' \subseteq \Gamma$ used in the original application of C-NewObj, we can apply C-NewObj to the expression after substitution. The types and static effects the methods are the same, and the same Γ' has been chosen, so $[e'/z]e$ will be ascribed the same type as e.

□

Lemma 3.3. (Monotonicity of `effects`)

Statement. If $\Gamma_1 \subseteq \Gamma_2$ then $\texttt{effects}(\Gamma_1) \subseteq \texttt{effects}(\Gamma_2)$

Proof. Because $\texttt{effects}(\Gamma_1)$ is the union of $\texttt{effects}(\tau)$, for every $(x, \tau) \in \Gamma_1 \subseteq \Gamma_2$. Then $\texttt{effects}(\Gamma_1) \subseteq \texttt{effects}(\Gamma_2)$.

□

Lemma 3.4. (Use Principle)

Statement. If $\Gamma \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$, and $e_A \longrightarrow_* e'_A \mid \varepsilon$, then $\forall r.\pi \in \varepsilon \mid (r, \{r\}) \in \Gamma$.

Proof. The only reduction that can add effects to ε is $r.\pi$. So at some point, an expression of the form $r.\pi$ must have been evaluated. In the source program it must have had the form $e.\pi$. Since the entire program typechecked under Γ, e must have been typed to $\{r\}$ at some point. Since resources cannot be dynamically created, $(r, \{r\}) \in \Gamma$.

Intuition. If you typecheck e with Γ, if an effect can happen on r when executing e then r must be in Γ.

□

Lemma 3.5. (Tightening Lemma)

Statement. If $\Gamma \vdash e : \tau \texttt{ with } \varepsilon$ then $\Gamma \cap \texttt{freevars}(e) \vdash e : \tau \texttt{ with } \varepsilon$.

Proof. The typing judgements operate on the form of e, so don't consider any variables external to e.

□

Note. We'll use $\texttt{freevars}(e) \cap \Gamma$ to mean Γ, where the pair (x, τ) is thrown out if $x \notin \texttt{freevars}(e)$.

Intuition. If you can typecheck e in Γ, you can throw out the parts in Γ not relevant to e and still typecheck it.

Theorem 3.6. (Extension Lemma)

Statement. If the following are true:

- $v_1 = \texttt{new } x \Rightarrow d_i = e_i$
- $d_i = \texttt{def } m_i(y : \tau_2) : \tau_3$
- $\Gamma \vdash d_i = e_i \texttt{ OK}$
- $\Gamma \vdash v_2 : \tau_2 \texttt{ with } \varepsilon_2$
- $[v_1/x, v_2/y]e_i \longrightarrow_* e'_i \mid \varepsilon$

Then $\exists \varepsilon_i \mid \varepsilon \subseteq \varepsilon_i \subseteq \texttt{effects}(\Gamma)$. Letting $\sigma_i = d_i \texttt{ with } \varepsilon_i$, then also $\sigma_i = e_i \texttt{ OK}$.

Intuition. This lemma says that we can take an unlabeled object v_1 with one method and produce a labeled object with one method, whose static effects contain all of the possible runtime effects. This ε_i is just going to be everything captured by the method body.

Note. In this theorem we only consider objects with a single method m_i. Later we generalise the result to objects with any number of methods.

Proof. Let $\Gamma' = (\texttt{freevars}(e_i) \cup \texttt{freevars}(v_1)) \cap \Gamma$. Our ε_i will be $\texttt{effects}(\Gamma')$.

By the Tightening Lemma, $\Gamma' \vdash d_i = e_i \texttt{ OK}$. The only rule with this conclusion is ε-ValidImpl$_d$. By inversion , $\Gamma', y : \tau_2 \vdash e_i : \tau_3$. This lets us apply C-Inference, with the entirety of Γ' as our subcontext. Then

$\Gamma' \vdash e_i : \tau_3$ `with effects` (Γ'). By the Tightening Lemma again, $\Gamma' \vdash v_2 : \tau_2$ `with` ε_2. By the Substitution Lemma, $\Gamma' \vdash [v_1/x, v_2/y]e_i : \tau_3$ `with effects` (Γ').

Since $[v_1/x, v_2/y]e_i \longrightarrow_* e_i' \mid \varepsilon$ and Γ' can typecheck the expression before reduction, then $\varepsilon \subseteq \texttt{effects}(\Gamma')$. Since $\Gamma' \subseteq \Gamma$, by monotonicity, $\varepsilon \subseteq \texttt{effects}(\Gamma') \subseteq \texttt{effects}(\Gamma)$. So $\texttt{effects}(\Gamma')$ is a witness to the ε_i in the statement of the lemma.
Finally, $\Gamma' \vdash \sigma_i = e_i$ under ε_σ. Since $\Gamma' \subseteq \Gamma$, then $\Gamma \vdash \sigma_i = e_i$ also.

□

Lemma 3.7. (Extension Theorem)

Statement. If $v_1 = \texttt{new}\ x \Rightarrow \overline{d = e}$ and $\forall i$ the following are true:

- $d_i = \texttt{def}\ m_i(y : \tau_2) : \tau_3$
- $\Gamma \vdash d_i = e_i$ `OK`
- $\Gamma \vdash v_2 : \tau_2$ `with` ε_2
- $[v_1/x, v_2/y]e_i \longrightarrow_* v \mid \varepsilon$

Then $\exists \varepsilon_1, ..., \varepsilon_n \mid \varepsilon \subseteq \bigcup_{i=1}^{n} \varepsilon_i \subseteq \texttt{effects}(\Gamma)$. Letting $\sigma_i = d_i$ `with` ε_i then also $\Gamma \vdash \texttt{new}\ x \Rightarrow \overline{\sigma = e}$ `with` $\varnothing$

Intuition. Any unlabeled object can be extended to a labeled object, whose labels contain every possible runtime effect.

Proof. From Single-Method Extension we know that $\forall i \exists \varepsilon_i \mid \varepsilon \subseteq \varepsilon_i \subseteq \texttt{effects}(\Gamma)$, and that the statement holds for each i. Then $\bigcup_{i=1}^{n} \varepsilon_i \subseteq \texttt{effects}(\Gamma)$. It also tells us that $\sigma_i = e_i$ `OK`, so $\overline{\sigma = e}$ `OK`. Then by ε-NewObj we have the claimed typing judgement.

□

Definition 3.8. (`label`)

The Extension Theorem essentially says that any unlabeled program can be extended to a fully-labeled one, whose labels give a conservative upper-bound on the possible runtime effects. We define a program-transforming function called `label` which does this.

- $\texttt{label}(r) = \text{r}$
- $\texttt{label}(x) = \text{x}$
- $\texttt{label}(e_1.m(e_2)) = \texttt{label}(e_1).m(\texttt{label}(e_2))$
- $\texttt{label}(e_1.\pi(e_2)) = \texttt{label}(e_1).\pi(\texttt{label}(e_2))$
- $\texttt{label}(\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}) = \texttt{new}_\sigma x \Rightarrow \texttt{label-helper}(\overline{\sigma = e})$
- $\texttt{label}(\texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e}) = \texttt{new}_\sigma\ x \Rightarrow \texttt{label-helper}(\overline{d = e})$
- $\texttt{label-helper}(\sigma = e) = \sigma = \texttt{label}(e)$
- $\texttt{label-helper}(\texttt{def}\ m(y : \tau_2) : \tau_3 = e) = \texttt{def}\ m(y : \tau_2) : \tau_3\ \texttt{with}\ \Gamma \cap \texttt{freevars}(e) = \texttt{label}(e)$

Notes:

- The program after labeling will be fully-labeled and contain terms entirely from the grammar for fully-labeled programs. Hence we can appeal to the soundness of that system.
- `label` is defined on expressions; `label-helper` on declarations. This is just for clarity; everywhere other than this section we'll only use `label`.
- Initially it seems like `label` on a $\texttt{new}_\sigma$ object should just be the identity function; but the body of the methods of such an object may instantiate unlabeled objects and/or call methods on unlabeled objects, so we must recursively label those.
- From here on out we will use $\hat{e}$ to refer to a fully-labeled program. We may sometimes say $\texttt{labels}(e) = \hat{e}$, and from then on refer to the labeled version of e. At least until we've established this transformation preserves types, we'll use $\hat{\tau}$ and $\hat{\varepsilon}$ to refer to the types and effects of $\hat{e}$.

Theorem 5.9. (Refinement Theorem)

Statement. If $\Gamma \vdash e : \tau$ `with` ε and `label`$(e) = \hat{e}$ and $\Gamma \vdash \hat{e} : \hat{\tau}$ `with` $\hat{\varepsilon}$, then $\hat{\varepsilon} \subseteq \varepsilon$ and $\tau = \hat{\tau}$.

Intuition. Labels can only make the static effects more precise; never less precise.

Proof. By induction on the judgement $\Gamma \vdash e : \tau$ `with` ε.

Case. ε-Resource, ε-Var.
If e is a resource or a variable then $e = \hat{e}$ so the statement is automatically fulfilled.

Case. ε-OperCall.
Then $e = e_1.\pi$ and we know:
- $\Gamma \vdash e :$ `Unit with` $\{r.\pi\} \cup \varepsilon_1$
- $\Gamma \vdash e_1 : \{\bar{r}\}$ `with` ε_1

Applying definitions, $\hat{e} =$ `label`$(e_1.\pi) =$ (`label`$(e_1)).\pi = \hat{e}_1.\pi$. By inductive assumption, $\Gamma \vdash \hat{e}_1 : \{\bar{r}\}$ `with` $\hat{\varepsilon}_1$, where $\hat{\varepsilon}_1 \subseteq \varepsilon_1$. Then $\Gamma \vdash \hat{e} :$ `Unit with` $\{r.\pi\} \cup \hat{\varepsilon}_1$ by ε-OperCall. Importantly, $\{r.\pi\} \cup \hat{\varepsilon}_1 \subseteq \{r.\pi\} \cup \varepsilon_1$ as claimed.

Case. ε-MethCall.
Then $e = e_1.m_i(e_2)$ and we know:
- $\Gamma \vdash e : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $\Gamma \vdash e_1 : \{\bar{\sigma}\}$ `with` ε_1
- $\Gamma \vdash e_2 : \tau_2$ `with` ε_2
- $\sigma_i =$ `def` $m_i(y : \tau_2) : \tau_3$ `with` ε_3

Applying definitions, $\hat{e} =$ `label`$(e_1.m_i(e_2)) =$ (`label`$(e_1)).m_i($`label`$(e_2)) = \hat{e}_1.m_i(\hat{e}_2)$. By inductive assumption, $\Gamma \vdash \hat{e}_1 : \{\bar{\sigma}\}$ `with` $\hat{\varepsilon}_1$ and $\Gamma \vdash \hat{e}_2 : \tau_2$ `with` $\hat{\varepsilon}_2$, where $\hat{\varepsilon}_1 \subseteq \varepsilon_1$ and $\hat{\varepsilon}_2 \subseteq \varepsilon_2$. Then $\Gamma \vdash \hat{e} : \tau_3$ `with` $\hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \varepsilon_3$ under ε-MethCall. Importantly, $\hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \varepsilon_3 \subseteq \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$ as claimed.

Case. C-MethCall.
Then $e = e_1.m_i(e_2)$ and we know:
- $\Gamma \vdash e : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $\Gamma \vdash e_1 : \{\bar{d}$ `captures` $\varepsilon_c\}$ `with` ε_1
- $\Gamma \vdash e_2 : \tau_2$ `with` ε_2
- $d_i =$ `def` $m_i(y : \tau_2) : \tau_3$

The reasoning is the same as the above case, but use C-MethCall instead of ε-MethCall.

Case. C-Inference.
We know:
- $\Gamma' \subseteq \Gamma$
- $\Gamma' \vdash e : \tau$
- $\Gamma \vdash e : \tau$ `with effects`(Γ')

This one's kind of interesting. There aren't any judgements of the form $e : \tau$ `with` ε so we can't use the induction hypothesis, right?

Case. ε-NewObj.
Then $e = e_1.m_i(e_2)$ and we know:
- $\Gamma \vdash e : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $\Gamma \vdash e_1 : \{\bar{d}$ `captures` $\varepsilon_c\}$ `with` ε_1
- $\Gamma \vdash e_2 : \tau_2$ `with` ε_2
- $d_i =$ `def` $m_i(y : \tau_2) : \tau_3$

The reasoning is the same as the above case, but use C-MethCall instead of ε-MethCall.

□