1 Grammar

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

Notes:

- σ is a declaration with its effects labeled; d a declaration without.
- $\texttt{new}_\sigma$ is for creating annotated objects; $\texttt{new}_d$ for unannotated objects.
- $\{\bar{\sigma}\}$ is the type of an annotated object; $\{\bar{d}\}$ of unannotated objects.
- $\{\bar{d}\ \texttt{captures}\ \varepsilon\}$ is a special kind of type that doesn't appear in source programs. ε is an upper-bound on the effects captured by $\{\bar{d}\}$.

2 Semantics

2.1 Static Semantics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{}{\Gamma \vdash \tau <: \tau}\ (\textsc{St-Reflexive}) \qquad \frac{\Gamma \vdash \tau_1 <: \tau_2 \quad \Gamma \vdash \tau_2 <: \tau_3}{\Gamma \vdash \tau_1 <: \tau_3}\ (\textsc{St-Transitive})$$

$$\frac{\Gamma \vdash e : \tau_1 \quad \Gamma \vdash \tau_1 <: \tau_2}{\Gamma \vdash e : \tau_2}\ (\textsc{St-Subsumption}) \qquad \frac{\Gamma \vdash \tau_1 <: \tau_2 \qquad \varepsilon_1 \subseteq \varepsilon_2}{\Gamma \vdash \tau_1\ \texttt{with}\ \varepsilon_1 <: \tau_2\ \texttt{with}\ \varepsilon_2}\ (\textsc{St-EffectTypes})$$

$$\frac{\Gamma \vdash \{\bar{\sigma}\}_1 \text{ is a permutation of } \{\bar{\sigma}\}_2}{\Gamma \vdash \{\bar{\sigma}\}_1 <: \{\bar{\sigma}\}_2}\ (\textsc{St-Permutation}_\sigma) \qquad \frac{\Gamma \vdash \{\bar{d}\}_1 \text{ is a permutation of } \{\bar{d}\}_2}{\Gamma \vdash \{\bar{d}\}_1 <: \{\bar{d}\}_2}\ (\textsc{St-Permutation}_d)$$

$$\frac{\Gamma \vdash \sigma_i <:: \sigma_j}{\Gamma \vdash \{\sigma_i\ {}^{i \in 1..n}\} <: \{\sigma_j\ {}^{j \in 1..n}\}}\ (\textsc{St-Depth}_\sigma) \qquad \frac{\Gamma \vdash d_i <:: d_j}{\Gamma \vdash \{d_i\ {}^{i \in 1..n}\} <: \{d_j\ {}^{j \in 1..n}\}}\ (\textsc{St-Depth}_d)$$

$$\frac{n, k \geq 0}{\Gamma \vdash \{\sigma_i\ {}^{i \in 1..n+k}\} <: \{\sigma_i\ {}^{i \in 1..n}\}}\ (\textsc{St-Width}_\sigma) \qquad \frac{n, k \geq 0}{\Gamma \vdash \{d_i\ {}^{i \in 1..n+k}\} <: \{d_i\ {}^{i \in 1..n}\}}\ (\textsc{St-Width}_d)$$

$\boxed{\Gamma \vdash \sigma <:: \sigma}$

$$\frac{\sigma_i = \texttt{def}\ m_A(y:\tau_1):\tau_2\ \texttt{with}\ \varepsilon_A \qquad \sigma_j = \texttt{def}\ m_B(y:\tau_1'):\tau_2'\ \texttt{with}\ \varepsilon_B \qquad \Gamma \vdash \tau_1' <: \tau_1 \qquad \Gamma \vdash \tau_2 <: \tau_2' \qquad \varepsilon_A \subseteq \varepsilon_B}{\Gamma \vdash \sigma_i <:: \sigma_j} \text{ (St-Method}_\sigma\text{)}$$

$\boxed{\Gamma \vdash d <:: d}$

$$\frac{d_i = \texttt{def}\ m_A(y:\tau_1):\tau_2 \qquad d_j = \texttt{def}\ m_B(y:\tau_1'):\tau_2' \qquad \Gamma \vdash \tau_1' <: \tau_1 \qquad \Gamma \vdash \tau_2 <: \tau_2'}{\Gamma \vdash d_i <:: d_j} \text{ (St-Method}_d\text{)}$$

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 of `effects` 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 authority over 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.
- By convention we'll 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.

- $\texttt{effects}(\varnothing) = \varnothing$
- $\texttt{effects}(\Gamma, x:\tau) = \texttt{effects}(\Gamma) \cup \texttt{effects}(\tau)$
- $\texttt{effects}(\{\bar{r}\}) = \{(r,\pi) \mid r \in \bar{r}, \pi \in \Pi\}$
- $\texttt{effects}(\{\bar{\sigma}\}) = \bigcup_{\sigma \in \bar{\sigma}} \texttt{effects}(\sigma)$
- $\texttt{effects}(\{\bar{d}\}) = \bigcup_{d \in \bar{d}} \texttt{effects}(d)$
- $\texttt{effects}(d\ \texttt{with}\ \varepsilon) = \varepsilon \cup \texttt{effects}(d)$
- $\texttt{effects}(\texttt{def m}(x:\tau_1):\tau_2) = \texttt{effects}(\tau_2)$
- $\texttt{effects}(\{\bar{d}\ \texttt{captures}\ \varepsilon_c\}) = \varepsilon_c$

3 Dynamic Semantics

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

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

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

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

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

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

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

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

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

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 to unlabeled objects. Same goes for E-MethCall3$_\sigma$ and E-MethCall3$_d$.
- E-MethCall1 can be used for both labeled and unlabeled objects.

4 Proofs

In this section we work towards a proof of soundness.

Lemma 4.1. (Canonical Forms)

$\boxed{\text{Statement.}}$ Suppose e is a value. The following are true:

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

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

$\boxed{\text{Proof.}}$ 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.

□

Definition 4.2. (Substitution)

- $[e'/z]z = e'$

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

Lemma 4.2. (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-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.

□

Soundness Strategy

The previous proofs were straightforward grammatical consequences. In the next few proofs we build up to the soundness theorem. Our approach is to show the following:

1. For any program containing unlabeled terms, there is a labeled version of that program which contains the runtime effects (this process is called labeling).
2. After labeling the program will only contain labeled terms, and typing judgements will be sound (from soundness of fully-labeled programs).
3. The presence of labels can only make static effect information more precise (Refinement theorem).

Because the labeled version of a program has more precise type information (3) and is type-and-effect sound (2), then weaker reasoning about the unlabeled version must also be type-and-effect sound.

Definition 4.3. (label)

A program can have its unlabeled terms labeled in a particular context Γ. We will define $\texttt{label}(e)$. It will be well-defined if $\Gamma \vdash e : \tau$; then we say $\texttt{label}(e, \Gamma) = \hat{e}$.

- $\texttt{label}(r, \Gamma) = \mathrm{r}$
- $\texttt{label}(x, \Gamma) = \mathrm{x}$
- $\texttt{label}(e_1.m(e_2), \Gamma) = \texttt{label}(e_1, \Gamma).m(\texttt{label}(e_2), \Gamma)$
- $\texttt{label}(e_1.\pi(e_2), \Gamma) = \texttt{label}(e_1, \Gamma).\pi(\texttt{label}(e_2), \Gamma)$

- $\texttt{label}(\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}, \Gamma) = \texttt{new}_\sigma\ x \Rightarrow \texttt{label-helper}(\overline{\sigma = e}, \Gamma)$
- $\texttt{label}(\texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e}, \Gamma) = \texttt{new}_\sigma\ x \Rightarrow \texttt{label-helper}(\overline{d = e}, \Gamma)$
- $\texttt{label-helper}(\sigma = e, \Gamma) = \sigma = \texttt{label}(e, \Gamma)$
- $\texttt{label-helper}(\texttt{def}\ m(y : \tau_2) : \tau_3 = e, \Gamma) = \texttt{def}\ m(y : \tau_2) : \tau_3\ \texttt{with}\ \texttt{effects}(\Gamma \cap \texttt{freevars}(e)) = \texttt{label}(e, \Gamma)$

Notes:

- $\Gamma \cap \texttt{freevars}(e)$ is the set of pairs $x : \tau \in \Gamma$, such that $x \in \texttt{freevars}(e)$.
- $\texttt{label}(e, \Gamma)$ is read as: "the labeling of e in Γ". When the Γ is obvious in context we will write $\texttt{label}(e)$ instead of $\texttt{label}(e, \Gamma)$.
- Beware of confusing notation: there are two types of equality in the above definitions. One is the equality which defines `label`, and the other is the equality $\sigma = e$ of declarations in the programming language.
- `label` is defined on expressions; `label-helper` on declarations. Everywhere other than this section we'll only use `label`.
- The body of a $\texttt{new}_\sigma$ may contain unlabeled objects so those must be recursively labeled too.
- We may sometimes say $\texttt{labels}(e) = \hat{e}$, and from then on refer to the labeled version of e as $\hat{e}$. We'll use $\hat{\tau}$ and $\hat{\varepsilon}$ to refer to the type and static effects of the labeled version.

Observation 4.4.

Statement. If $\Gamma \vdash e : \tau$, then $\texttt{label}(e, \Gamma)$ only contains terms from the fully-labeled system defined in `effects.pdf`.

Proof. By inspecting the definition, the right-hand side of $\texttt{label}(e) = \hat{e}$ contains only such terms.

$\square$

Property 4.5. (Commutativity Between `label` and `sub`)

Statement. Fix Γ and define $\texttt{label}(e) = \texttt{label}(e, \Gamma)$. Then $\texttt{label}([e'/z]e) = [\texttt{label}(e')/z](\texttt{label}(e))$

Intuition. If perform substitution and labeling on an expression, the order in which you do things doesn't matter.

Proof. Induction on the form of e. In each case, "left-hand side" refers to $\texttt{label}([e'/z]e)$ while "right-hand side" refers to $[\texttt{label}(e')/z](\texttt{label}(e))$.

Case. $e = r$.
By definition, $\texttt{label}(r) = r$ and $[e'/z]r = r$, for any e'. Both sides are equivalent to r because `sub` and `label` act like the identity function.

Case. $e = x$.
By definition, $\texttt{label}(x) = x$. $[e'/z]x$ has two definitions, depending on if $x = z$; consider each case.

<u>Subcase.</u> $x \neq z$. Then $[e'/z]x = x$. Both sides are equivalent to x because `sub` and `label` act like the identity function.

<u>Subcase.</u> $x = z$. Then $[e'/z]x = z$. On the left-hand side, $\texttt{label}([e'/z]x) = \texttt{label}(e')$. On the right-hand side, $[\texttt{label}(e')/z]x = \texttt{label}(e')$.

Case. $e = e_1.\pi$.
On the left-hand side.

$$
\begin{aligned}
&\texttt{label}([e'/z](e_1.\pi)) \\
&= \texttt{label}(([e'/z]e_1).\pi) && \text{(definition of } \texttt{sub}\text{)} \\
&= (\texttt{label}([e'/z]e_1)).\pi && \text{(definition of } \texttt{label}\text{)} \\
&= ([\texttt{label}(e')/z](\texttt{label}(e_1))).\pi && \text{(inductive assumption on } e_1\text{)}
\end{aligned}
$$

On the right-hand side.

$$
\begin{array}{ll}
[\texttt{label}(e')/z](\texttt{label}(e_1.\pi)) & \\
= [\texttt{label}(e')/z](\texttt{label}(e_1).\pi) & \text{(definition of } \texttt{label}\text{)} \\
= ([\texttt{label}(e')/z](\texttt{label}(e_1))).\pi & \text{(definition of } \texttt{sub}\text{)}
\end{array}
$$

$\boxed{\text{Case.}}$ $e = e_1.m(e_2)$.
On the left-hand side.

$$
\begin{array}{ll}
\texttt{label}([e'/z](e_1.m(e_2))) & \\
= \texttt{label}(([e'/z]e_1).m([e'/z]e_2)) & \text{(definition of } \texttt{sub}\text{)} \\
= (\texttt{label}([e'/z]e_1)).m(\texttt{label}([e'/z]e_2)) & \text{(definition of } \texttt{label}\text{)} \\
= ([\texttt{label}(e')/z](\texttt{label}(e_1)).m(\texttt{label}([e'/z]e_2)) & \text{(inductive assumption on } e_1\text{)} \\
= ([\texttt{label}(e')/z](\texttt{label}(e_1)).m([\texttt{label}(e')/z](\texttt{label}(e_2))) & \text{(inductive assumption on } e_2\text{)}
\end{array}
$$

On the right-hand side.

$$
\begin{array}{ll}
[\texttt{label}(e')/z](\texttt{label}(e_1.m(e_2))) & \\
= [\texttt{label}(e')/z]((\texttt{label}(e_1)).m(\texttt{label}(e_2))) & \text{(definition of } \texttt{label}\text{)} \\
= ([\texttt{label}(e')/z](\texttt{label}(e_1))).m([\texttt{label}(e')/z](\texttt{label}(e_2))) & \text{(definition of } \texttt{sub}\text{)}
\end{array}
$$

$\boxed{\text{Case.}}$ $e = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$.
On the left-hand side.

$$
\begin{array}{ll}
\texttt{label}([e'/z](\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = e_i})) & \\
= \texttt{label}(\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = [e'/z]e_i}) & \text{(definition of } \texttt{sub}\text{)} \\
= \texttt{new}_\sigma\ x \Rightarrow \overline{\texttt{label-helper}(\sigma_i = [e'/z]e_i)} & \text{(definition of } \texttt{label}\text{)} \\
= \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = \texttt{label}([e'/z]e_i)} & \text{(definition of } \texttt{label-helper} \text{ on each } \sigma_i = [e'/z]e_i\text{)}
\end{array}
$$

On the right-hand side.

$$
\begin{array}{ll}
[\texttt{label}(e')/z](\texttt{label}(\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = e_i})) & \\
= [\texttt{label}(e')/z](\texttt{new}_\sigma\ x \Rightarrow \overline{\texttt{label-helper}(\overline{\sigma_i = e_i})}) & \text{(definition of } \texttt{label}\text{)} \\
= [\texttt{label}(e')/z](\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = \texttt{label}(e_i)}) & \text{(definition of } \texttt{label-helper} \text{ on each } \sigma_i = e_i\text{)} \\
= \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = [\texttt{label}(e')/z](\texttt{label}(e_i)} & \text{(definition of } \texttt{sub}\text{)} \\
= \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = \texttt{label}([e'/z]e_i)} & \text{(inductive assumption on each } e_i\text{)}
\end{array}
$$

$\boxed{\text{Case.}}$ $e = \texttt{new}_d\ x \Rightarrow \overline{d = e}$.
The proof of this is quite similar to previous case for labeled objects. The main difference is that when labeling an unlabeled object, each $d_i = e_i$ turns into a $\sigma_i = e_i$. For clarity we will define $\varepsilon_i = \texttt{effects}(\Gamma \cap \texttt{freevars}(e_i))$, and $\sigma_i = d_i\ \texttt{with}\ \varepsilon_i$ (these are from the definition of `label-helper`).

On the left-hand side.

$$
\begin{array}{ll}
\texttt{label}([e'/z](\texttt{new}_d\ x \Rightarrow \overline{d_i = e_i})) & \\
= \texttt{label}(\texttt{new}_d\ x \Rightarrow \overline{d_i = [e'/z]e_i}) & \text{(definition of } \texttt{sub}\text{)} \\
= \texttt{new}_d\ x \Rightarrow \overline{\texttt{label-helper}(d_i = [e'/z]e_i)} & \text{(definition of } \texttt{label}\text{)} \\
= \texttt{new}_d\ x \Rightarrow \overline{d_i\ \texttt{with}\ \varepsilon_i = \texttt{label}([e'/z]e_i)} & \text{(definition of } \texttt{label-helper}\text{)} \\
= \texttt{new}_d\ x \Rightarrow \overline{\sigma_i = \texttt{label}([e'/z]e_i)} & (\sigma_i = d_i\ \texttt{with}\ \varepsilon_i)
\end{array}
$$

On the right-hand side.

$$
\begin{array}{ll}
[\texttt{label}(e')/z](\texttt{label}(\texttt{new}_d\ x \Rightarrow \overline{d_i = e_i})) & \\
= [\texttt{label}(e')/z](\texttt{new}_d\ x \Rightarrow \overline{\texttt{label-helper}(\overline{d_i = e_i})}) & \text{(definition of } \texttt{label}\text{)} \\
= [\texttt{label}(e')/z](\texttt{new}_\sigma\ x \Rightarrow \overline{d_i\ \texttt{with}\ \varepsilon_i = \texttt{label}(e_i)}) & \text{(definition of } \texttt{label-helper} \text{ on each } d_i = e_i\text{)} \\
= [\texttt{label}(e')/z](\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = \texttt{label}(e_i)}) & (\sigma_i = d_i\ \texttt{with}\ \varepsilon_i) \\
= \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = [\texttt{label}(e')/z](\texttt{label}(e_i))} & \text{(definition of } \texttt{sub}\text{)} \\
= \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_i = \texttt{label}([e'/z]e_i)} & \text{(inductive assumption on each } e_i\text{)}
\end{array}
$$

$\square$

Property 4.6. (Runtime Invariance Under `label`)

Statement. If the following are true:

- $\Gamma \vdash e_A : \tau_A$ `with` ε_A
- $e_A \longrightarrow e_B \mid \varepsilon$
- $\hat{e}_A = \texttt{label}(e_A, \Gamma)$

Then $\hat{e}_A \longrightarrow \hat{e}_B \mid \varepsilon$ and $\hat{e}_B = \texttt{label}(e_B, \Gamma)$.

Intuition If you label a program and then reduce, or reduce and then label, you get the same thing.

Proof. Induct on the form of e_A and then on the reduction rule $e_A \longrightarrow e_B \mid \varepsilon$. Throughout this proof there is only a single context Γ, so we'll write $\texttt{label}(e)$ instead of $\texttt{label}(e, \Gamma)$ as a notational short-hand.

Case. $e = r$, $e = x$, $e = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$, $e = \texttt{new}_d\ x \Rightarrow \overline{d = e}$.
Then e is a value and the theorem statement holds automatically.

Case. $e = e_1.\pi$.
The only typing rule which applies is ε-OperCall, which tells us:
- $\Gamma \vdash e_1 : \{r\}$ `with` ε_1
- $\Gamma \vdash e_1.\pi : \texttt{Unit with } \varepsilon_1 \cup \{r.\pi\}$

There are two possible reductions.

Subcase. E-OperCall1. We also know $e_1 \longrightarrow e_1' \mid \varepsilon$, and $e_1.\pi \longrightarrow e_1'.\pi \mid \varepsilon$. By inductive assumption, $\hat{e}_1 \longrightarrow \hat{e}_1' \mid \varepsilon$, and $\hat{e}_1' = \texttt{label}(e_1')$. Applying definitions, $\hat{e}_A = \texttt{label}(e_1.\pi) = (\texttt{label}(e_1)).\pi = \hat{e}_1.\pi$. Because $\hat{e}_1 \longrightarrow \hat{e}_1' \mid \varepsilon$, we may apply the reduction E-OperCall1 to obtain $\hat{e}_1.\pi \longrightarrow \hat{e}_1'.\pi \mid \varepsilon$. Lastly, $\hat{e}_B = \texttt{label}(e_1'.\pi) = (\texttt{label}(e_1')).\pi$, which we know to be $\hat{e}_1'.\pi$ by inductive assumption.

Subcase. E-OperCall2. We also know $e_1 = r$ and $r.\pi \longrightarrow \texttt{Unit} \mid \{r.\pi\}$. Applying definitions, $\hat{e}_A = \texttt{label}(r.\pi) = (\texttt{label}(r)).\pi = r.\pi = e_A$. The theorem holds immediately.

Case. $e = e_1.m_i(e_2)$.
There are five possible reductions.

Subcase. E-MethCall1. We also know $e_1 \longrightarrow e_1' \mid \varepsilon$ and $e_1.m_i(e_2) \longrightarrow e_1'.m_i(e_2) \mid \varepsilon$. By inductive assumption, $\hat{e}_1 \longrightarrow \hat{e}_1' \mid \varepsilon$, and $\texttt{label}(e_1') = \hat{e}_1'$. Applying definitions $\hat{e}_A = \texttt{label}(e_1.m_i(e_2)) = (\texttt{label}(e_1)).m_i(\texttt{label}(e_2)) = \hat{e}_1.m_i(\hat{e}_2)$. Because $\hat{e}_1 \longrightarrow \hat{e}_1' \mid \varepsilon$, we may apply the reduction E-MethCall1 to obtain $\hat{e}_1.m_i(\hat{e}_2) \longrightarrow \hat{e}_1'.m_i(\hat{e}_2) \mid \varepsilon$. Lastly, $\hat{e}_B = \texttt{label}(e_1'.m_i(\hat{e}_2)) = (\texttt{label}(e_1')).m_i(\texttt{label}(e_2))$, which we know to be $\hat{e}_1'.m_i(\hat{e}_2) = \hat{e}_B$ by assumptions.

Subcase. E-MethCall2$_\sigma$. We also know $e_1 = v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$, and $e_2 \longrightarrow e_2' \mid \varepsilon$ and $v_1.m_i(e_2) \longrightarrow v_1.m_i(e_2') \mid \varepsilon$. By inductive assumption, $\hat{e}_2 \longrightarrow \hat{e}_2' \mid \varepsilon$, and $\texttt{label}(e_2') = \hat{e}_2'$. Applying definitions, $\hat{e}_A = \texttt{label}(v_1.m_i(e_2)) = (\texttt{label}(v_1)).m_i(\texttt{label}(e_2)) = \hat{v}_1.m_i(\hat{e}_2)$. Because $\hat{e}_2 \longrightarrow \hat{e}_2' \mid \varepsilon$, we may apply the reduction E-MethCall$_\sigma$ to obtain $\hat{v}_1.m_i(\hat{e}_2) \longrightarrow \hat{v}_1.m_i(\hat{e}_2')$. Lastly, $\hat{e}_B = \texttt{label}(v_1.m_i(e_2')) = (\texttt{label}(v_1)).m_i(\texttt{label}(e_2'))$, which we know to be $\hat{v}_1.m_i(\hat{e}_2')$ by assumptions.

Subcase. E-MethCall2$_d$. Identical to the above subcase, but $e_1 = v_1 = \texttt{new}_d\ x \Rightarrow \overline{d = e}$, and we apply the reduction rule E-MethCall$_d$ instead.

Subcase. E-MethCall3$_\sigma$. We also know the following:
- $e_1 = v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$
- $e_2 = v_2$
- $\texttt{def}\ m_i(y : \tau_2) : \tau_3\ \texttt{with}\ \varepsilon_3 = e_{body} \in \{\bar{\sigma}\}$
- $v_1.m_i(v_2) \longrightarrow [v_1/x, v_2/y]e_{body} \mid \varnothing$.

Applying definitions, $\texttt{label}(v_1.m_i(v_2)) = (\texttt{label}(v_1)).m_i(\texttt{label}(v_2)) = \hat{v}_1.m_i(\hat{v}_2)$, where we define $\hat{v}_1 = \texttt{label}(v_1)$ and $\hat{v}_2 = \texttt{label}(v_2)$. Before labeling, the object v_1 has method m_i with body e_{body}. The labeled version, $\hat{v}_1$, has method m_i with body $\texttt{label}(e_{body}) = \hat{e}_{body}$. Because v_1 and v_2 are values, so are $\hat{v}_1$ and $\hat{v}_2$. Therefore we can apply E-MethCall3$_\sigma$ to $\hat{v}_1.m_i(\hat{v}_2)$, giving us $\hat{v}_1.m_i(\hat{v}_2) \longrightarrow [\hat{v}_1/x, \hat{v}_2/y]\hat{e}_{body} \mid \varnothing$. Because `label` and `sub` commute, $\texttt{label}(e_B) = \texttt{label}([v_1/x, v_2/y]e_{body}) = [\texttt{label}(v_1)/x, \texttt{label}(v_2)/y](\texttt{label}(e_{body})))$, which is $[\hat{v}_1/x, \hat{v}_2/y]\hat{e}_{body} = \hat{e}_B$, by how we defined $\hat{v}_1$, $\hat{v}_2$, and $\hat{e}_{body}$.

Subcase. E-MethCall3$_d$. This case is identical to the previous one, except $e_1 = v_1 = \texttt{new}_d\ x \Rightarrow \overline{d = e}$. The same reasoning applies though.

□

Theorem 4.7. (Refinement Theorem)

Statement. If $\Gamma \vdash e : \tau$ `with` ε and $\texttt{label}(e) = \hat{e}$, then one of the following is true:

– $\Gamma \vdash \hat{e} : \hat{\tau}$ `with` $\hat{\varepsilon}$, where $\hat{\varepsilon} \subseteq \varepsilon$ and $\hat{\tau} <: \tau$
– e has the form $\texttt{new}_d\ x \Rightarrow \overline{d = e}$ and $\Gamma \vdash \hat{e} : \overline{d_i \texttt{ with } \varepsilon_i = e_i}$, where $\varepsilon_i = \texttt{effects}\ (\Gamma \cap \texttt{freevars}(e_i))$

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

Proof.

□

Lemma 4.8. (Extension Lemma)

Statement. If $\Gamma \vdash e : \tau$ and $\hat{e} = \texttt{label}(e, \Gamma)$ then $\Gamma \vdash e : \hat{\tau}$ `with` $\hat{\varepsilon}$, where $\hat{\tau} <: \tau$. If $e \longrightarrow e' \mid \varepsilon$, then $\varepsilon \subseteq \hat{\varepsilon}$.

Intuition. If Γ can type e without an effect, there is a way to label e with $\hat{\varepsilon}$ which contains the possible runtime effects of e (so $\hat{\varepsilon}$ is an upper-bound).

Proof.

□

Theorem 4.9. (Soundness Theorem)

Statement. If $\Gamma \vdash e_A : \tau_A$ `with` ε_A and $e_A \longrightarrow e_B \mid \varepsilon$ then $\Gamma \vdash e_B : \tau_B$ `with` ε_B, where:

1. $\tau_B <: \tau_A$ and $\varepsilon_B \subseteq \varepsilon_A$
2. $\varepsilon \subseteq \varepsilon_A$

Proof. □