1 Grammar

$$
\begin{array}{lcll}
\rho & ::= & x & \textit{primitives} \\
 & | & r & \\
 & & & \\
\tau_\rho & ::= & \{r\} & \textit{primitive types} \\
 & & & \\
e_u & ::= & \rho & \textit{deeply unlabeled progs.} \\
 & | & \texttt{new}_d\ x \Rightarrow \overline{d = e_u} & \\
 & | & e_u.m(e_u) & \\
 & | & e_u.\pi & \\
 & & & \\
d & ::= & \texttt{def}\ m(y : \tau_u) : \tau_u & e_u\textit{-prog decls.} \\
 & & & \\
\tau_u & ::= & \{\bar{d}\} & e_u\textit{-prog types} \\
 & | & \tau_\rho & \\
 & & & \\
e_l & ::= & \rho & \textit{deeply labeled progs.} \\
 & | & \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma_l = e_l} & \\
 & | & l.m(l) & \\
 & | & l.\pi & \\
 & & & \\
\sigma_l & ::= & \texttt{def}\ m(y : \tau_l) : \tau_l\ \texttt{with}\ \varepsilon & e_l\textit{-prog decls.} \\
 & & & \\
\tau_l & ::= & \{\bar{\sigma}_l\} & e_l\textit{-prog types} \\
 & | & \tau_\rho & \\
 & & & \\
e & ::= & \rho & \textit{progs.} \\
 & | & e.m(e) & \\
 & | & e.\pi & \\
 & | & \texttt{new}_d\ x \Rightarrow \overline{d = e_u} & \\
 & | & \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} & \\
 & & & \\
\sigma & ::= & \texttt{def}\ m(y : \tau) : \tau\ \texttt{with}\ \varepsilon & e_l\textit{-prog decls.} \\
 & & & \\
\tau & ::= & \tau_l & \textit{types} \\
 & | & \tau_u & \\
 & | & \{\bar{\sigma}\} &
\end{array}
$$

Notes:

- e_u programs are *deeply unlabeled* programs: no labels appear in the source code (though label inference may be done by the type system).
- e_l programs are *deeply labeled* programs: everything in the source code is labeled.
- e programs are the general form of a syntactically-correct program. They may contain a mixture of labeled and unlabeled parts. Any unlabeled parts must be deeply unlabeled, but labeled parts need not be deeply labeled. This means you can have unlabeled parts appearing inside labeled parts, but not vice versa.
- Any e_l or e_u term is also an e term.

2 Static Semantics

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

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

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

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

$\boxed{\Gamma \vdash e_u : \tau_u}$

$$\frac{\Gamma, x : \{\overline{d}\} \vdash \overline{d = e_u}\ \texttt{OK}}{\Gamma \vdash\ \texttt{new}_d\ x \Rightarrow \overline{d = e_u} : \{\overline{d}\}}\ (e_u\text{-}\textsc{New}) \qquad \frac{\Gamma \vdash e_u : \{r\}}{\Gamma \vdash e_u.\pi : \texttt{Unit}}\ (e_u\text{-}\textsc{OperCall})$$

$$\frac{\Gamma \vdash e_{u,1} : \{\overline{d}\} \qquad \texttt{def}\ m(y : \tau_{u,2}) : \tau_{u,3} \in \{\overline{d}\} \qquad \Gamma \vdash e_{u,2} : \tau_{u,2}}{\Gamma \vdash e_{u,1}.m(e_{u,2}) : \tau_{u,3}}\ (e_u\text{-}\textsc{MethCall})$$

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

$$\frac{d = \texttt{def}\ m(y : \tau_{u,2}) : \tau_{u,3} \quad \Gamma, y : \tau_{u,2} \vdash e_u : \tau_{u,3}}{\Gamma \vdash d = e_u\ \texttt{OK}}\ (e_u\text{-}\textsc{ValidImpl})$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Notes:

- A good choice of Γ' for e_u-$\textsc{New}_\varepsilon$ is the intersection of Γ with the free variables in the object.
- By convention we use ε_c to denote the output of the `effects` function.

3 Definition: `effects` Function

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

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

– $\texttt{effects}(\{\bar{d}\ \texttt{captures}\ \varepsilon_c\}) = \varepsilon_c$

Notes:

1. The function is monotonic: if $\Gamma_1 \subseteq \Gamma_2$, then $\texttt{effects}(\Gamma_1) \subseteq \texttt{effects}(\Gamma_2)$.

4 Dynamic Semantics

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

$$\frac{e_{u,1} \longrightarrow e'_{u,1} \mid \varepsilon}{e_{u,1}.m(e_{u,2}) \longrightarrow e'_{u,1}.m(e_{u,2}) \mid \varepsilon} \text{ (E-MethCall1)}$$

$$\frac{v_{u,1} = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = l} \quad e_{u,2} \longrightarrow e'_{u,2} \mid \varepsilon}{v_{u,1}.m(e_{u,2}) \longrightarrow v_{u,1}.m(e'_{u,2}) \mid \varepsilon} \text{ (E-MethCall2)}$$

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

$$\frac{e_{u,1} \longrightarrow e'_{u,1} \mid \varepsilon}{e_{u,1}.\pi \longrightarrow e'_{u,1}.\pi \mid \varepsilon} \text{ (E-OperCall1)} \qquad \frac{}{r.\pi \longrightarrow \texttt{unit} \mid \{r.\pi\}} \text{ (E-OperCall2)}$$

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

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

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

Notes:

– The runtime only operates on (deeply) unlabeled expressions. You may think of a compiler as stripping all the effect labels from a program before execution.

5 Lemma (Canonical Forms)

TODO

6 Definition (`substitution`)

TODO

7 Lemma (Substitution)

Lemma. Suppose the following is true:

1. $\Gamma, z : \tau' \vdash e : \tau \texttt{ with } \varepsilon$
2. $\Gamma \vdash e' : \tau' \texttt{ with } \varepsilon'$

Then $\Gamma \vdash [e'/z]e : \tau \texttt{ with } \varepsilon$.

Proof. TODO (Should be same as the proof in previous grammar, just need to convert everything to new grammar)

8 Definition (`label`)

A program may be converted into a fully-labeled program. This is a function from e-terms to e_l-terms. It is always defined relative to some Γ, which is usually clear from context. The process is well-defined on e if $\Gamma \vdash e : \tau \texttt{ with } \varepsilon$. Then `label` is defined below.

1. $\texttt{label}(\rho) = \rho$
2. $\texttt{label}(e_1.\pi) = \texttt{label}(e_1).\pi$
3. $\texttt{label}(e_1.m(e_2)) = \texttt{label}(e_1).m(\texttt{label}(e_2))$
4. $\texttt{label}(\texttt{new}_d\ x \Rightarrow \overline{d = e_u}) = \texttt{new}_\sigma\ x \Rightarrow \overline{\texttt{label}_{\texttt{decl}}(d) = \texttt{label}(e_u)}$
5. $\texttt{label}(\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e})\ = \texttt{new}_\sigma\ x \Rightarrow \overline{\texttt{label}_{\texttt{decl}}(\sigma) = \texttt{label}(e)}$

The helper function `label-decl` works by labeling each declaration with what it captures in the context Γ. We abbreviate this as $\texttt{effects}(\Gamma \cap \texttt{freevars}(e))$. The helper is defined below.

5. $\texttt{label}_{\texttt{decl}}(\texttt{def}\ m(y : \tau_A) : \tau_B,\ e_{body}) = \texttt{def}\ m(y : \texttt{label}_{\texttt{type}}(\tau_A)) : \texttt{label}_{\texttt{type}}(\tau_B)\ \texttt{with}\ \Gamma \cap \texttt{freevars}(e_{body})$

If you label a type it should produce the labeled version.

6. $\texttt{label}_{\texttt{type}}(\{r\}) = \{r\}$
7. $\texttt{label}_{\texttt{type}}(\{\bar{\sigma}\}) = \{\texttt{label}_{\texttt{decl}}(\sigma)\}$
8. $\texttt{label}_{\texttt{type}}(\{\bar{d}\}) = \{\texttt{label}_{\texttt{decl}}(d)\}$

Notes:

– The image of $\texttt{label}(e_u)$ is an e_l-term (proof by induction on definition).
– e_u is a value $\iff$ $\texttt{label}(e_u)$ is a value.
– We can define $\varnothing \cap \texttt{freevars}(e)$ as $\varnothing$, and $(\Gamma, x : \tau) \cap \texttt{freevars}(e)$ as $(\Gamma \cap \texttt{freevars}(e)) \cup (\{x\} \cap \texttt{freevars}(e))$.

9 Definition (`unlabel`)

The inverse of `label`. TODO

10 Theorem (`label` and `sub` Commute)

TODO

11 Theorem (Soundness)

Theorem. Suppose $\Gamma \vdash e_A : \tau_A$ and $e_A \longrightarrow e_B \mid \varepsilon$. The following are true:

1. $\Gamma \vdash e_B : \tau_B$
2. $\tau_B <: \tau_A$
3. $\Gamma \vdash \texttt{label}(e_A) : \hat{\tau}_A \texttt{ with } \varepsilon_A$
4. $\Gamma \vdash \texttt{label}(e_B) : \hat{\tau}_B \texttt{ with } \varepsilon_B$
5. $\varepsilon \cup \varepsilon_B = \varepsilon_A$

Proof.

From refinement we know $\Gamma \vdash \texttt{label}(e_A) : \hat{\tau}_A \texttt{ with } \varepsilon_A$, where $\hat{\tau}_A <: \tau_A$. Choose $\hat{\tau}_A = \tau_A$.

Because $\texttt{unlabel}(e_A) = \texttt{unlabel}(\texttt{label}(e_A))$, then $\texttt{label}(e_A) \longrightarrow e_B \mid \varepsilon$.

By soundness of reduction on the e_l-term $\texttt{label}(e_A)$, we know $\Gamma \vdash \texttt{label}(e_B) : \hat{\tau}_B \texttt{ with } \varepsilon_B$, where $\texttt{label}(\tau_B) <: \texttt{label}(\tau_A)$. Choose $\hat{\tau}_B = \hat{\tau}_A$. Then we know $\hat{\tau}_B = \hat{\tau}_A = \tau_A$.

Stuff below needs formal justification we haven't explored: Because labels only make types more restrictive, the range of possible types for e_B is contained in the range of possible types $\texttt{label}(e_B)$. For example:

```
def m₁(y: τ_A): τ_B with ε
```

Is a subtype of:

```
def m₂(y: τ_A): τ_B
```

Because ε is an upper-bound on the effects, m_1 is not allowed to have any effect $r.\pi \notin \varepsilon$, but m_2 is allowed because it has no upper-bound. Therefore the second can (should be able to) be typed to the first in any situation.

Then since we already have a typing judgement for $\texttt{label}(e_B)$ with type $\hat{\tau}_B$ we know $\Gamma \vdash e_B : \tau_B$ (progress theorem). Then choose $\tau_B = \hat{\tau}_B$.