July 14, 2016

1 Effects

Fix some set of resources R. A resource is some language primitive that has the authority to directly perform I/O operations. Elements of the set R are denoted by r. Π is a fixed set of operations on resources. Its members are denoted π. An effect is a member of the set of pairs $R \times \Pi$. A set of effects is denoted by ε. In this system we cannot dynamically create resources or resource-operations.

Throughout we refer to the notions of effects and captures. A piece of code C has the effect (r, π) if operation π is performed on resource r during execution of C. C captures the effect (r, π) if it has the authority to perform operation π on resource r at some point during its execution.

We use $r.\pi$ as syntactic sugar for the effect (r, π). For example, $FileIO.append$ instead of $(FileIO, append)$.

Types are either resources or structural. Structural types have a set of method declarations. An object of a particular structural type $\{\bar{\sigma}\}$ can have any of the methods defined by σ invoked on it. The structural type $\varnothing$ with no methods is called `Unit`.

We assume there are constructions of the familiar types using the basic structural type $\varnothing$ and method declarations (for example, $\mathbb{N}$ could be made using $\varnothing$ and a `successor` function, Peano-style).

Note the distinction between methods (usually denoted m) and operations (usually denoted π). An operation can only be invoked on a resource; resources can only have operations invoked on them. A method can only be invoked on an object; objects can only have methods invoked on them.

We make a simplifying assumption that every method/lambda takes exactly one argument. Invoking some operation π on a resource returns $\varnothing$.

2 Static Semantics For Fully-Annotated Programs

In this first system every method in the program is explicitly annotated with its set of effects.

2.1 Grammar

$$
\begin{array}{lll}
e & ::= & x \qquad\qquad\qquad\qquad\qquad\qquad \textit{expressions} \\
& | & r \\
& | & \texttt{new}\ x \Rightarrow \overline{\sigma = e} \\
& | & e.m(e) \\
& | & e.\pi(e) \\
\\
\tau & ::= & \{\bar{\sigma}\} \mid \{\bar{r}\} \qquad\qquad\qquad\qquad \textit{types} \\
\\
\sigma & ::= & \texttt{def}\ m(x : \tau) : \tau\ \texttt{with}\ \varepsilon\ \textit{labeled decls.} \\
\\
\Gamma & ::= & \varnothing \\
& | & \Gamma,\ x : \tau
\end{array}
$$

Notes:

- All declarations (σ-terms) are annotated by what effects they have.
- All methods (and lambda expressions) take exactly one argument. If a method specifies no argument the argument is assumed to be of type `Unit`.
- π is the name of an operation from the set Π; m is the name of a method.
- The type $\{\bar{r}\}$ is an (indeterminate) set of resources; there will only be one actual resource at run-time, and it will be one of the resources in the set.

2.2 Rules

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

$$\frac{}{\Gamma,\ x : \tau \vdash x : \tau\ \texttt{with}\ \varnothing}\ (\varepsilon\text{-VAR}) \qquad \frac{r \in R}{\Gamma,\ r : \{r\} \vdash r : \{r\}\ \texttt{with}\ \varnothing}\ (\varepsilon\text{-RESOURCE})$$

$$\frac{\Gamma,\ x : \{\bar{\sigma}\} \vdash \overline{\sigma = e}\ \texttt{OK}}{\Gamma \vdash \texttt{new}\ x \Rightarrow \overline{\sigma = e} : \{\bar{\sigma}\}\ \texttt{with}\ \varnothing}\ (\varepsilon\text{-NEWOBJ})$$

$$\frac{\Gamma \vdash e_1 : \{\bar{r}\}\ \texttt{with}\ \varepsilon_1 \quad \Gamma \vdash e_2 : \tau_2\ \texttt{with}\ \varepsilon_2 \quad \pi \in \Pi}{\Gamma \vdash e_1.\pi(e_2) : \texttt{Unit}\ \texttt{with}\ \{\bar{r}.\pi\} \cup \varepsilon_1 \cup \varepsilon_2}\ (\varepsilon\text{-OPERCALL})$$

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

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

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

Notes:

– In ε-Var, ε-Resource, and ε-NewObj the consequent has an expression typed with no effect; merely possessing a (transitive) capability for an object is not a effect. You must do something with it in order to possibly have an effect.
– ε-ValidImpl says that the return type and effects of the body of a method must agree with what its signature says.

3 Static Semantics For Partly-Annotated Programs

In the next system we allow objects which have no effect-annotated methods. If an object has no annotations on its methods, the extra rules below can give a conservative effect inference on what it captures. If an object is fully annotated, the rules from the previous section can be used. There is no inbetween; partially-annotated objects are not allowed.

3.1 Grammar

$$
\begin{array}{lll}
e ::= x & & \textit{expressions} \\
\quad | \quad r & & \\
\quad | \quad \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} & & \\
\quad | \quad \texttt{new}_d\ x \Rightarrow \overline{d = e} & & \\
\quad | \quad e.m(e) & & \\
\quad | \quad e.\pi(e) & &
\end{array}
$$

$$
\begin{array}{lll}
\tau ::= \{\bar{\sigma}\} & & \textit{types} \\
\quad | \quad \{\bar{r}\} & & \\
\quad | \quad \{\bar{d}\} & & \\
\quad | \quad \{\bar{d}\ \texttt{captures}\ \varepsilon\} & &
\end{array}
$$

$$
\sigma ::= d\ \texttt{with}\ \varepsilon \qquad \textit{labeled decls.}
$$

$$
d ::= \texttt{def}\ m(x : \tau) : \tau \quad \textit{unlabeled decls.}
$$

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 $\{\bar{d}\}$.

3.2 Rules

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

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

$$\frac{\Gamma \vdash r : \{\bar{r}\} \quad \Gamma \vdash e : \tau \quad m \in M}{\Gamma \vdash r.\pi(e_1) : \texttt{Unit}}\ \text{(T-MethCall}_r\text{)}$$

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

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

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

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

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

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

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

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

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

Notes:

- The system includes the rules from the fully-annotated system.
- Rules with the judgement form do standard typing of objects, without any effect analysis. Their sole use is in the ε-ValidImpl$_d$ rule.
- In applying C-NewObj, Γ is the current context. The variable Γ' is some sub-context. A good choice of sub-context is Γ restricted to the free variables in the method-body being typechecked. This tightens the upper-bound to exclude resources never used in the program.
- To perform effect analysis on an unannotated object $\{\bar{d}\}$, figure out what it captures using C-NewObj, yielding the type $\{\bar{d} \texttt{ captures } \varepsilon\}$. Then, when a method is called on that object, C-MethCall can be applied.
- C-ConservativeTyping is used to effect-type unannotated portions of code. It essentially says that in the absence of any effect information, you can assume its effects as $\texttt{effects}(\texttt{Gamma}')$, a safe upper-bound for some relevant choice of Γ'.

3.3 Effects Function

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

A method m can return a resource r (directly or via some enclosing object). Returning a resource isn't an effect but it means any unannotated program using m also captures r. To account for this, when the `effects` function is operating on a type τ it must analyse the return type of the method declarations in τ. Since the resource might be itself enclosed by an object, we do a recursive analysis.

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

Notes:

– In the last case, it is not necessary to recurse to sub-declarations; the type $\{\bar{d}\ \texttt{captures}\ \varepsilon\}$ can only result from C-NewObj. In that case, ε is already an estimate of what the object captures in some context Γ_0; furthermore, the current Γ may be bigger than Γ_0, in which case re-annotating the object would introduce more effects and lose precision.

4 Dynamic Semantics

4.1 Terminology

- The runtime effects are what actually happens when the program is executed; the static effect annotations are an estimate of what will happen at runtime (our eventual aim is to show the static annotations are safe).
- If e is an expression then $[e_1/x_1]e$ is a new expression, the same as e, with every free occurrence of x_1 replaced by e_1. $[e_1/x_1, ..., e_n/x_n]e$ is syntactic sugar for repeated one-variable substitution: $[e_1/x_1]...[e_n/x_n]e$.
- $\varnothing$ is the empty set. The empty type is denoted `Unit`. Its single instance is `unit`.

4.2 Grammar

$$
\begin{array}{lll}
e ::= x & & \textit{expressions} \\
\quad | \quad e.m(e) & & \\
\quad | \quad e.\pi(e) & & \\
\quad | \quad v & & \\
 & & \\
v ::= r & & \textit{values} \\
\quad | \quad \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} & & \\
\quad | \quad \texttt{new}_d\ x \Rightarrow \overline{d = e} & &
\end{array}
\qquad
\begin{array}{lll}
d ::= \texttt{def}\ m(x : \tau) : \tau & & \textit{unlabeled decls.} \\
 & & \\
\sigma ::= d\ \texttt{with}\ \varepsilon & & \textit{labeled decls.} \\
 & & \\
\tau ::= \{\bar{\sigma}\} & & \textit{types} \\
\quad | \quad \{\bar{r}\} & & \\
 & & \\
\Gamma ::= \varnothing & & \textit{contexts} \\
\quad | \quad \Gamma,\ x : \tau & &
\end{array}
$$

4.3 Rules

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

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

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

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

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

$$\frac{e_1 \longrightarrow e_1' \mid \varepsilon}{e_1.\pi(e_2) \longrightarrow e_1'.\pi(e_2) \mid \varepsilon}\ \text{(E-OperCall1)} \qquad \frac{e_2 \longrightarrow e_2' \mid \varepsilon}{r.\pi(e_2) \longrightarrow r.\pi(e_2') \mid \varepsilon}\ \text{(E-OperCall2)}$$

$$\frac{r \in R \quad \pi \in \Pi}{r.\pi(v) \longrightarrow \texttt{unit} \mid \{r.\pi\}}\ \text{(E-OperCall3)}$$

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

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

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

Notes:

- A multi-step involves *zero* or more applications of a small-step.
- Multi-step rules accumulate the run-time effects produced by the individual small-steps.
- The only rule which produces effects is E-OperCall3 (the rule for evaluating operations on resources).
- Method calls are evaluated by performing substitution on the body of the method, and evaluating that.

5 Theorems

Lemma 5.1. (Canonical Forms)

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

- If $\Gamma \vdash e : \{\bar{r}\}$ `with` ε, then $\exists r \in R \mid e = r$.
- If $\Gamma \vdash e : \{\bar{\sigma}\}$`with` ε, then $e = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$.
- If $\Gamma \vdash e : \{\bar{d}$ `captures` $\varepsilon_1\}$ `with` ε, 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.

□

Definition. (Substitution)

$[e'/z]e$ means 'substitute every free occurrence of x in e for e'. A definition over the grammar is as follows.

- $[e'/z]z = e'$
- $[e'/z]r = r$
- $[e'/z](e_1.m(e_2)) = ([e'/z]e_1).m([e'/z]e_2)$
- $[e'/z](e_1.\pi(e_2)) = ([e'/z]e_1).\pi([e'/z]e_2)$
- $[e'/z](\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}) = ...$

(But that last one—-if z and x happen to be the same variables we don't want to sub out x because it's the 'this' variable. Needs a bit more thought, consult the almighty TAPL)

Lemma 5.2. (Substitution)

Statement. If $\Gamma, z : \tau' \vdash e : \tau$ `with` ε, and $\Gamma \vdash e' : \tau'$ `with` ε', then $\Gamma \vdash [e'/z]e : \tau$ `with` ε.

Proof. By structural induction on possible derivations of $\Gamma \vdash e : \tau_1$ `with` ε_1. First, if z does not appear in e then $[e'/z]e = e$, so the statement holds vacuously. So without loss of generality we assume z appears somewhere in e and consider the last rule used in the derivation, and then the location of z.

Case. ε-Var.
Then $[e'/z]z = e_1$. By assumption $\Gamma \vdash e' : \tau$ `with` ε, so $\Gamma \vdash [e'/z]z = e$.

Case. ε-Resource.
Then $\exists r \in R \mid e = r$, however $\texttt{freevars}(r) = \varnothing$, so the statement holds vacuously.

Case. ε-NewObj.
Then $e = \texttt{new}\ x \Rightarrow \overline{\sigma = e}$.

z appears in some method body e_i. By inversion we know $\Gamma, x : \{\bar{\sigma}\} \vdash \overline{\sigma = e}$ `OK`. The only rule with this conclusion is ε-ValidImpl$_\sigma$; by inversion on that we have:
- $\Gamma, y : \tau_1 \vdash e_i : \tau_2$ `with` ε
- $\sigma = \texttt{def}\ m_i(y : \tau_1) : \tau_2$ `with` ε

$\Gamma, z : \tau$ can be used to prove $\sigma_i = e_i$ `OK` via the inductive assumption. By an application of ε-ValidImpl$_\sigma$ we may type the substituted object declaration. Then by ε-NewObj we type $[e'/z]e$ to the same as the original.

Case. ε-OperCall.
Then $e = e_1.\pi(e_2)$. The variable z must appear in e_1 or e_2. By rule inversion we have a sub-derivation for the type of both sub-expressions. If we perform substitution on both, the inductive assumption applies, yielding the same types for $[e'/z]e_1$ and $[e'/z]e_2$. Then ε-OperCall types the substituted operation call to the same as the original.

Case. ε-MethCall$_\sigma$.
Then $e = e_1.m_i(e_2)$. The variable z must appear in e_1 or $e-2$. By rule inversion we have a sub-derivation for both; we may type the substituted sub-expressions to the same as the original. Then by applying ε-MethCall on the substituted sub-expressions we type $[e'/z]e_2$ to the same as the original.

Case. C-MethCall.
Same as above case; it's not important that the receiver's declarations are unannotated because substitution only applies to expressions.

Case. C-NewObj.
Same as for the rule ε-NewObj; it's not important that the receiver's declarations are unannotated because substitution only applies to expressions.

□

Corollary. If $\Gamma, z_i : \tau_i' \vdash e : \tau$ `with` ε, and $\Gamma \vdash e_i' : \tau_i'$ `with` ε_i', then $\Gamma \vdash [e_1'/z_1, ..., e_n'/z_n]e : \tau$ `with` ε. This follows by the definition $[e_1'/z_1, ..., e_n'/z_n]e = [e_1']...[e_n']e$ and induction on the length n.

Theorem 5.3. (Progress)

Statement. If $\Gamma \vdash e_A : \tau_A$ `with` ε_A, either e_A is a value or a small-step $e_A \longrightarrow e_B \mid \varepsilon$ can be applied.

Proof. By structural induction on possible derivations of $\Gamma \vdash e_A : \tau_A$ `with` ε_A. We consider the last rule used.

Case. ε-Var, ε-Resource, ε-NewObj, C-NewObj.
Then e_A is a value.

Case. ε-MethCall.
Then $e_A = e_1.m_i(e_2)$ and the following are known:
- $e_A : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $e_1 : \{\overline{\sigma}\}$ `with` ε_1
- $e_2 : \tau_2$ `with` ε_2
- $\sigma_i =$ `def` $m_i(y : \tau_2) : \tau_3$ `with` ε_3

We look at the cases for when e_1 and e_2 are values.

Subcase. e_1 is not a value. The derivation of $e_A : \tau$ `with` ε_A includes the subderivation $e_1 : \{\bar{\sigma}\}$ `with` ε_1. By the inductive hypothesis $e_1 \longrightarrow e_1' \mid \varepsilon$. Then E-MethCall1 gives the reduction $e_A \longrightarrow e_1'.m_i(e_2) \mid \varepsilon$.

Subcase. e_2 is not a value. Without loss of generality, $e_1 = v_1$ is a value. Also, $e_2 : \tau_2$ `with` ε_2 is a subderivation. By inductive hypothesis, $e_2 \longrightarrow e_2' \mid \varepsilon$. Then E MethCall2$_\sigma$ gives the reduction $e_A \longrightarrow v_1.m_i(e_2') \mid \varepsilon$.

Subcase. $e_1 = v_1$ and $e_2 = v_2$ are values. By Canonical Forms, $e_1 =$ `new`$_\sigma$ $x \Rightarrow \overline{\sigma = e}$. Also, `def` $m_i(y : \tau_2) : \tau_3$ `with` $\varepsilon_3 = e_i \in \overline{\sigma = e}$. By the assumption of the typing rule used, the receiver and argument are well-typed for the method m_i. Then E-MethCall3$_\sigma$ gives the reduction $e_1.m_i(e_2) \longrightarrow [v_1/x, v_2/y]e_i \mid \varnothing$.

Case. ε-OperCall.
Then $e_A = e_1.\pi(e_2)$ and the following are known:
- $e_A :$ `Unit with` $\{r.\pi\} \cup \varepsilon_1 \cup \varepsilon_2$
- $e_1 : \{\bar{r}\}$ `with` ε_1
- $e_2 : \tau_2$ `with` ε_2
- $\pi \in \Pi$

We look at the cases for when e_1 and e_2 are values.

Subcase. e_1 is not a value. $e_1 : \{\bar{r}\}$ `with` ε_1 is a subderivation. Applying inductive assumption, we have $e_1 \longrightarrow e_1' \mid \varepsilon$. Then E-OperCall1 gives the reduction $e_1.\pi(e_2) \longrightarrow e_1'.\pi(e_2) \mid \varepsilon$.

Subcase. e_2 is not a value. Without loss of generality, $e_1 = v_1$ is a value. Also, $e_2 : \tau_2$ `with` ε_2 is a subderivation, so applying inductive assumption we get $e_2 \longrightarrow e_2' \mid \varepsilon$. Then E-OperCall2 gives the reduction $v_1.\pi(e_2) \longrightarrow v_1.\pi(e_2') \mid \varepsilon$.

Subcase. e_1 and e_2 are values. By Canonical Forms, $\exists r \in R \mid e_1 = r$. Then E-OperCall3 gives the reduction $r.\pi(v_2) \longrightarrow$ `unit` $\mid \{r.\pi\}$.

Case. C-MethCall.
Then $e_A = e_1.m_i(e_2)$ and the following are known:

- $e_A : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup$ `effects`$(\tau_2) \cup \varepsilon_c$
- $e_1 : \{\bar{d}$ `captures` $\varepsilon_c\}$ `with` ε_1
- $e_2 : \tau_2$ `with` ε_2
- $d_i =$ `def` $m_i(y : \tau_2) : \tau_3$

We look at the cases for when e_1 and e_2 are values.

Subcase. e_1 is not a value. Also, $e_1 : \{\bar{d}$ `captures` $\varepsilon_c\}$ `with` ε_1 is a subderivation. By inductive hypothesis, $e_1 \longrightarrow e_1' \mid \varepsilon$. Then E-MethCall1 gives the reduction $e_1.m_i(e_2) \longrightarrow e_1'.m_i(e_2) \mid \varepsilon$.

Subcase. e_2 is not a value. Without loss of generality, $e_1 = v_1$ is a value. Also, $e_2 : \tau_2$ `with` ε_2 is a subderivation. By inductive hypothesis, $e_2 \longrightarrow e_2' \mid \varepsilon$. Then E-MethCall2$_d$ gives the reduction $v_1.m_i(e_2) \longrightarrow v_1.m_i(e_2') \mid \varepsilon$.

Subcase. e_1 and e_2 are values. By Canonical Forms, $e_1 =$ `new`$_d\ x \Rightarrow \overline{d = e}$. Also, `def` $m_i(y : \tau_2) : \tau_3 = e_i \in \overline{d = e}$. By assumption of the typing rule used, the receiver and argument are well-typed for method m_i. Then E-MethCall3$_d$ gives the reduction $v_1.m_i(v_2) \longrightarrow [v_1/x, v_2/y]e_i \mid \varnothing$

□

Theorem 5.4. (Preservation for Mini Calculus)

Statement. Work in the mini calculus for fully-annotated programs. Suppose the following hold:

- $\Gamma_A \vdash e_A : \tau_A$ `with` ε_A
- $e_A \longrightarrow e_B \mid \varepsilon$
- $\Gamma_B \vdash e_B : \tau_B$ `with` ε_B

Then $\varepsilon_B \subseteq \varepsilon_A$ and $\forall r.\pi \in \varepsilon_A \setminus \varepsilon_B \mid r.\pi \in \varepsilon$. Furthermore, if e_A is not of the form $e_1.\pi(e_2)$, then $\tau_A = \tau_B$.

Intuitively: during reduction, effects are only lost from the static information when they are added to the runtime information, so every effect gets "accounted for". You cannot gain static information after reducing an expression.

Proof. By structural induction on the derivation of $\Gamma_A \vdash e_A : \tau_A$ `with` ε_A and then on the reduction rule used.

Case. ε-Resource, ε-Var, ε-NewObj, C-NewObj.
e_A is a value, so no reduction rules can be applied to it. The theorem statement is vacuously satisfied.

Case. ε-MethCall$_\sigma$.
Then $e_A = e_1.m_i(e_2)$ and the following are true:

- $e_A : \tau$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $e_1 : \{\overline{\sigma}\}$ `with` ε_1
- $e_2 : \tau_2$ `with` ε_2
- $\sigma_i =$ `def` $m_i(y : \tau_2) : \tau$ `with` ε_3

We do a case analysis on the reduction rules applicable to $e_1.m_i(e_2)$, for m_i an annotated method.

Subcase. E-MethCall1 Then $e_1 \longrightarrow e_1' \mid \varepsilon$. By inductive assumption $e_1' : \{\overline{\sigma}\}$ `with` ε_1'. Then by ε-MethCall we have $e_B = e_1'.m_i(e_2) : \tau_3$ `with` $\varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_3$. Then $\varepsilon_B = \varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_3$ and $\varepsilon_B \setminus \varepsilon_A = \varepsilon_1' \setminus \varepsilon_1$. Any lost effect info is accounted for by inductive assumption.

Subcase. E-MethCall2$_\sigma$ Then $e_1 = v_1 =$ `new`$_\sigma\ x \Rightarrow \overline{\sigma = e}$, and $e_2 \longrightarrow e_2' \mid \varepsilon$. By inductive assumption $e_2' : \tau_2$ `with` ε_2. Then by ε-MethCall we have $e_B = v_1.m_i(e_2) : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2' \cup \varepsilon_3$. Then $\varepsilon_B = \varepsilon_1 \cup \varepsilon_2' \cup \varepsilon_3$ and $\varepsilon_B \setminus \varepsilon_A = \varepsilon_2' \setminus \varepsilon_2$. Any lost effect info is accounted for by inductive assumption.

Subcase. E-MethCall3$_\sigma$ Then $e_1 = v_1 = \texttt{new}_\sigma \Rightarrow \overline{\sigma = e}$, and $\texttt{def}\ m_i(y : \tau_2) : \tau_3\ \texttt{with}\ \varepsilon_3 = e' \in \overline{\sigma = e}$, and $e_2 = v_2$ is a value, and $v_1.m_i(v_2) \longrightarrow [v_1/x, v_2/y]e' \mid \varnothing$.
We already know $e_1 : \{\overline{\sigma}\}\ \texttt{with}\ \varepsilon_1$. The only rule with this conclusion is ε-NewObj. By inversion, $\overline{\sigma = e}\ \texttt{OK}$. The only rule with this conclusion is ε-ValidImpl$_\sigma$. By inversion, $e' : \tau_3\ \texttt{with}\ \varepsilon_3$.
Now $e_B = [v_1/x, v_2/y]e'$, since the rule E-MethCall3 was used. We know $v_1 = e_1$ and x have the same type, which is $\{\overline{\sigma}\}\ \texttt{with}\ \varepsilon_1$. We also know $v_2 = e_2$ and y have the same type, which is $\tau_2\ \texttt{with}\ \varepsilon_2$. By the substitution lemma, the type of e' is preserved under substitution. So $e_B : \tau_3\ \texttt{with}\ \varepsilon_3$.
Since $e_1 = v_1$ and $e_2 = v_2$ are values, by Canonical Forms $\varepsilon_1 = \varepsilon_2 = \varnothing$. So $\varepsilon_A = \varepsilon_3$. Then $\varepsilon_B \setminus \varepsilon_A = \varnothing$ and there are no lost effects to account for.

Case. ε-OperCall$_\sigma$.
Then $e_A = e_1.\pi(e_2) : \texttt{Unit}$, and we know:
- $e_A : \{r, \pi\} \cup \varepsilon_1 \cup \varepsilon_2$
- $e_1 : \{\bar{r}\}\ \texttt{with}\ \varepsilon_1$
- $e_2 : \tau_2\ \texttt{with}\ \varepsilon_2$
- $\pi \in \Pi$

There are three reduction rules applicable to terms of the form $e_1.\pi(e_2)$ for π an operation. We consider each.

Subcase. E-OperCall. Then $e_1 \longrightarrow e_1' \mid \varepsilon$. By inductive assumption, $e_1' : \{\bar{r}\}\ \texttt{with}\ \varepsilon_1'$. From these we can apply ε-OperCall, giving $e_B = e_1'.\pi(e_2) : \texttt{Unit with}\ \{r.\pi\} \cup \varepsilon_1' \cup \varepsilon_2$. Then $\varepsilon_B = \{r.\pi\} \cup \varepsilon_1' \cup \varepsilon_2$ and $\varepsilon_A \setminus \varepsilon_B = \varepsilon_1'$. Any lost effect info from ε_1' is accounted for by inductive hypothesis.

Subcase. E-OperCall2. Then $e_1 = r$ for some $r \in R$ and $e_2 \longrightarrow e_2' \mid \varepsilon$. By inductive assumption $e_2' : \tau_2\ \texttt{with}\ \varepsilon_2'$. From these we can apply ε-OperCall, giving $e_B = r.\pi(e_2')\ \texttt{with}\ \varepsilon$. Then $\varepsilon_B = r.\pi \cup \varepsilon_1 \cup \varepsilon_2'$ and $\varepsilon_A \setminus \varepsilon_B = \varepsilon_2'$. Any lost effect info from ε_2' is accounted for by inductive hypothesis.

Subcase. E-OperCall3. Then $r.\pi(v) \longrightarrow \texttt{unit} \mid \{r.\pi\}$. By Canonical Forms, $\varepsilon_1 = \varepsilon_2 = \varnothing$, so $e_A : \texttt{Unit with}\ \{r.\pi\}$. By a degenerate case of ε-NewObj, $\varepsilon_B = \texttt{unit} : \texttt{Unit with}\ \varnothing$. Then $\varepsilon_A \setminus \varepsilon_B = \{r.\pi\}$. We can see that this is exactly the set of runtime effects ε, so we have accounted for the only lost effect.

□

Theorem 5.5. (Small-Step Soundness For Mini Calculus)

Statement. Work in the mini calculus for fully-annotated programs. If $\Gamma \vdash e_A : \tau_A\ \texttt{with}\ \varepsilon_A$ and $e_A \longrightarrow e_B \mid \varepsilon$, then $\varepsilon \subseteq \varepsilon_A$.

Proof. By structural induction on the rule used to derive $\Gamma \vdash e_A : \tau\ \texttt{with}\ \varepsilon_A$, and then on the reduction rule used.

Case. ε-Resource, ε-Var, ε-NewObj.
e_A is a value, so no reduction rules can be applied to it. The theorem statement is vacuously satisfied.

Case. ε-MethCall$_\sigma$.
Then $e_A = e_1.m_i(e_2)$ and the following are known:
- $e_A : \tau_A\ \texttt{with}\ \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $e_1 : \{\bar{\sigma}\}\ \texttt{with}\ \varepsilon_1$
- $e_2 : \tau_2\ \texttt{with}\ \varepsilon_2$
- $\sigma_i = \ \texttt{def}\ m_i(y : \tau_2) : \tau_3\ \texttt{with}\ \varepsilon_3$

Consider the possible reduction rules.

Subcase. E-MethCall1. Then $e_1 \longrightarrow e_1' \mid \varepsilon$. By inductive assumption, $\varepsilon \subseteq \varepsilon_1 \subseteq \varepsilon_A$.

Subcase. E-MethCall2. Then $e_2 \longrightarrow e_2' \mid \varepsilon$. By inductive assumption, $\varepsilon \subseteq \varepsilon_2 \subseteq \varepsilon_A$.

Subcase. E-MethCall3. Then $v_1.m_i(v_2) \longrightarrow [v_1/x, v_2/y]e' \mid \varnothing$. We're done as $\varnothing \subseteq e_A$.

Case. ε-OperCall.
Then $e_A = e_1.m_i(e_2)$ and the following are known:

- $e_A : \texttt{unit with } \varepsilon_1 \cup \varepsilon_2 \cup \{r.\pi\}$
- $e_1 : \{\bar{r}\} \texttt{ with } \varepsilon_1$
- $e_2 : \tau_2 \texttt{ with } \varepsilon_2$
- $\pi \in \Pi$

Consider the possible reduction rules.

Subcase. E-OperCall1 Then $e_1 \longrightarrow e_1' \mid \varepsilon$. By induction assumption $\varepsilon \subseteq \varepsilon_1 \subseteq \varepsilon_A$.

Subcase. E-OperCall2 Then $e_2 \longrightarrow e_2' \mid \varepsilon$. By induction assumption $\varepsilon \subseteq \varepsilon_2 \subseteq \varepsilon_A$.

Subcase. E-OperCall3 Then $\exists r \mid e_1 = r$ and $e_2 = v_2$ for some value and $r.\pi_i(v_2) \longrightarrow \texttt{unit} \mid \{r.\pi\}$. By Canonical Forms, $\varepsilon_1 = \varepsilon_2 = \varnothing$. Then $\varepsilon_A = \{r.\pi\}$. We can see this is exactly the set of runtime effects.

□

Theorem 5.6. (Multi-Step Soundness For Mini Calculus)

Statement. Work in the mini calculus for fully-annotated programs. If $\Gamma \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$ and $e \longrightarrow_* e_B \mid \varepsilon$ then $\varepsilon \subseteq \varepsilon_A$.

Proof. If the multi-step involves zero steps then the theorem is vacuously satisfied. Otherwise the multi-step involves more than one step. Induct on the number of steps.

Base Case. The length is 1. The theorem holds by Small-Step Soundness For Mini Calculus.

Inductive Case. If there is a big-step of length n+1 then by E-MultiStep3 it can be decomposed into a multi-step of length n; $e_1 \longrightarrow_* e_n \mid \varepsilon_n$; and a multi-step of length 1; $e_n \longrightarrow_* e_{n+1} \mid \varepsilon$. Then $e_1 \longrightarrow_* e_{n+1} \mid \varepsilon_n \cup \varepsilon$ is the entire multi-step.

Let the type derivations be $\Gamma \vdash e_1 : \tau_1 \texttt{ with } \varepsilon_1$ and $\Gamma \vdash e_n \texttt{ with } \varepsilon_n$

By inductive assumption on the smaller multi-steps, we have that $\varepsilon \subseteq \varepsilon_n$ and $\varepsilon_n \subseteq \varepsilon_1$. Then $\varepsilon_1 \cup \varepsilon_n \subseteq \varepsilon_1$. ε_1 is the static effect information at the start of the multi-step, so we're done.

□

Theorem 5.7. (Single-Method Extension)

Statement. If the following are true:

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

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

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 $\Gamma' = \texttt{freevars}(e_i) \cap \Gamma$. Let $\varepsilon_i = \texttt{effects}(\Gamma)$. Let $\sigma_i = d_i \texttt{ with } \varepsilon_i$.

With this choice of Γ' and the assumption $\Gamma \vdash d_i = e_i \texttt{ OK}$ we may type e_i using C-ConservativeTyping. Then $e_i : \tau_i \texttt{ with } \varepsilon_i$, where $\varepsilon_i = \texttt{effects}(\Gamma')$. Then ε-ValidImpl$_\sigma$ gives $\sigma_i = e_i \texttt{ OK}$.

By the substitution lemma, $[v_1/x, v_2/y]e_i : \tau_i \texttt{ with } \varepsilon_i$. By application of multi-step soundness to the assumed multi-step reduction we have $\varepsilon \subseteq \varepsilon_i = \texttt{effects}(\Gamma') \subseteq \texttt{effects}(\Gamma)$.

□

Theorem 5.8. (Extension)

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

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

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

Proof. From Single-Method Extension we know that $\forall i \exists \varepsilon_i \mid \varepsilon \subseteq \varepsilon_i \subseteq \texttt{effects}(\Gamma')$. Let $\Gamma' = \bigcup_{i=1}^{n} \Gamma'_i$. Then $\bigcup_{i=1}^{n} \varepsilon_i \subseteq \texttt{effects}(\Gamma') \subseteq \texttt{effects}(\Gamma)$.

From Single-Method Extension we know $\sigma_i = e_i\ \texttt{OK}$, so $\overline{\sigma = e}\ \texttt{OK}$. Then by ε-NewObj we have the claimed typing judgement.

□

Definition. (Label)

Using the results of the previous theorem we define a program-transforming function called `label`, which transforms an unannotated program into an annotated one.

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

Notes:

- `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 $\hat{e}$ as the labeled program. Likewise, we will use $\hat{\tau}$ and $\hat{\varepsilon}$ to refer to the type and effects of $\hat{e}$.

Theorem 5.9. (Refinement)

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

Proof. We show the property holds for each case of the `label` function by inducting on the complexity of e.

Case. $\texttt{label}(r)$, $\texttt{label}(x)$.
Then $e = \hat{e}$ so it holds immediately.

Case. $\texttt{label}(\texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e}), \texttt{label}(\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e})$.
Typing judgements for objects only give $\varnothing$ as the set of effects so we're done automatically. Although an object can have capabilities for effects bu those are encoded in its methods (in the case of a labeled object) or in its type (in the case of an unlabeled object having type $\{\bar{d}\ \texttt{captures}\ \varepsilon_c\}$).

Case. $\texttt{e}_1.\texttt{m}(\texttt{e}_2)$.
Then $\texttt{label}(e_1.m(e_2)) = \hat{e}_1.m(\hat{e}_2)$ and the inductive assumption holds for $\hat{e}_1$ and $\hat{e}_2$. Since $\hat{e}_1.m(\hat{e}_2)$ is a labeled object we may type it with ε-MethCall$_\sigma$. By inspecting that rule we see the set of effects after labeling is $\hat{\varepsilon} = \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \hat{\varepsilon}_3$. By inductive assumption on subexpressions, each $\hat{\varepsilon}_i \subseteq \varepsilon_i$.

Now what is the form of $e_1.m(e_2)$? If e_i is already labeled then $\hat{\varepsilon}_i = \varepsilon$. So without loss of generality, let's assume e_1 and e_2 are both unlabeled. There are two judgments that might have typed $e_1.m(e_2)$ so let's examine them.

Subcase. C-ConservativeTyping Then $\varepsilon = \texttt{effects}(\Gamma')$. Any effect in $\hat{\varepsilon}$ must be captured in the environment at large; therefore $\hat{\varepsilon} \subseteq \varepsilon$.
Subcase. C-MethCall. Then $\varepsilon = \varepsilon_1 \cup \varepsilon_2 \cup \texttt{effects}(\tau_2) \cup \texttt{effects}(\Gamma')$ for some sub-context. Since $\hat{\varepsilon}_i \subseteq \varepsilon_i$, we just need to show that $\hat{\varepsilon}_3 \subseteq \varepsilon$.

Note that $\hat{\varepsilon}_3$ represents the effects captured by executing method m. Let $r.\pi \in \hat{\varepsilon}_3$. If $r.\pi$ is captured then someone must be able to call $r.\pi$ from somewhere as a result of the method call. From the perspective of the method call $e_1.m(e_2)$ there are two ways $r.\pi$ could be invoked:
- During execution of method m the operation $r.\pi$ is invoked, either directly or indirectly, before control-flow returns to the call-site.
- As a result of executing method m a capability for r is gained at the call-site. r may then be passed back up the call-chain and someone later may invoke $r.\pi$, after $e_1.m(e_2)$ has finished executing.

In the first case, the resource r must be visible in the current environment, i.e. $r \in \Gamma'$. Then $r.\pi \in \texttt{effects}(\Gamma') \subseteq \varepsilon$.

In the second case, if r is currently in the environment then $r.\pi \in \texttt{effects}(\Gamma')$ again. Otherwise r might not be in the environment; the only way for the call-site to gain a capability then is for the method m to have returned it. Therefore $r \in \texttt{effects}(\tau_2) \subseteq \varepsilon$, where τ_2 is the return type of m.

Case. $\texttt{e}_1.\pi(\texttt{e}_2)$.
Then $\hat{e} = \hat{e}_1.\pi(\hat{e}_2)$. By inductive assumption the labeled expressions will be $\hat{\varepsilon}_1$ and $\hat{\varepsilon}_2$. The only rule which can be used to type $\hat{e}$ and e is ε-OperCall; then the effect-set for e is $\varepsilon = \{r.\pi\} \cup \varepsilon_1 \cup \varepsilon_2$ and for $\hat{e}$ is $\hat{\varepsilon} = \{r.\pi\} \cup \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2$. $\hat{\varepsilon} \subseteq \varepsilon$ by inductive assumption on each ε_i.

□

Intuition. This theorem says that labeling a program can only make the effect inference more precise (never less precise).

Theorem 5.10. (Preservation Under Labeling)

Statement. If $\Gamma \vdash e : \tau$ then $\Gamma \vdash \hat{e} : \tau$.

Proof. By induction on the complexity of the expression e and considering the possible forms of e. If $e = r$ or $e = x$ then it holds trivially. Consider the other cases.

Case. $e_1.m(e_2)$.
By induction we know $\texttt{type}(e_1) = \texttt{type}(\hat{e}_1)$ and $\texttt{type}(e_2) = \texttt{type}(\hat{e}_2)$. The result holds by applying T-MethCall$_\sigma$.

Case. $e_1.\pi(e_2)$.
Same as above but use T-MethCall$_r$.

Case. $\texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$.
From induction we know that since $\Gamma \vdash \overline{\sigma = e}\ \texttt{OK}$ then $\Gamma \vdash \overline{\sigma = \hat{e}}\ \texttt{OK}$. The result holds by applying T-NewObj$_\sigma$.

Case. $\texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e}$.
Same as above, but $\Gamma \vdash \overline{d = \hat{e}}$ lets us apply T-NewObj$_d$.

□

Theorem 5.11. (Small-Step Soundness Of Labeled Programs)

Statement. If $e_A \longrightarrow e_B \mid \varepsilon_B$ and $e_A : \tau_A$ then $\hat{e}_A : \hat{\tau}\ \texttt{with}\ \hat{\varepsilon}$ where $\varepsilon_B \subseteq \hat{\varepsilon}$.

Proof. By induction on the evaluation rule used. In all cases, the type of $\hat{e}_A$ is the same as e_A by the previous theorem so we omit its derivation.

Case. E-MethCall1.
Then the following are known.
- $e_A = e_1.m(e_2)$

– $e_1 \to e_1' \mid \varepsilon$
– $e_B = e_1'.m(e_2) \mid \varepsilon$

By inductive assumption $\hat{e}_1 : \tau$ `with` $\hat{\varepsilon}_1$ where $\varepsilon \subseteq \hat{\varepsilon}_1$. By applying the rule ε-MethCall$_\sigma$ we type $\hat{e}_A$ to something with the effect-set $\hat{\varepsilon} = \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \hat{\varepsilon}_3 \supseteq \hat{\varepsilon}_1 \supseteq \varepsilon$.

Case. E-OperCall1.
Same as previous case, but use the rule ε-OperCall.

Case. E-MethCall2$_\sigma$, E-MethCall2$_d$.
Then the following are known.
– $e_A = v_1.m(e_2)$
– $e_2 \longrightarrow e_2' \mid \varepsilon$

By inductive assumption $\hat{e}_2 : \tau$ `with` $\hat{\varepsilon}_2$ where $\varepsilon \subseteq \hat{\varepsilon}_2$. By applying the rule ε-MethCall$_\sigma$ we type $\hat{e}_1.m(\hat{e}_2)$ to something with the effect-set $\hat{\varepsilon} = \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \hat{\varepsilon}_3 \supseteq \hat{\varepsilon}_2 \supseteq \varepsilon$.

Case. E-OperCall2.
Same as previous case, but use the rule

Case. E-MethCall3$_\sigma$, E-MethCall3$_d$.
In this case $\varepsilon = \varnothing$ so the statement holds trivially.

Case. E-OperCall3.
Then $e_A = r.\pi(v)$. $\hat{r} = r$ so $\hat{e}_A = r.\pi(\hat{v})$. By application of ε-OperCall we have $\hat{e}_A$: `Unit with` $\{r.\pi\} \cup \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2$, where $\hat{\varepsilon}_2$ is the set of effects for $\hat{v}$ and $\hat{\varepsilon}_1 = \varnothing$ for $\hat{r} = r$. By inspection of the rule for E-OperCall3, $\varepsilon = r.\pi \subseteq \{r.\pi\} \cup \hat{\varepsilon}_2 = \hat{\varepsilon}_A$.

□

Theorem 5.12. (Soundness Theoerm)

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 and $\varepsilon \subseteq \varepsilon_A$.

Proof. Induct on the typing judgement for $\Gamma \vdash e_A : \tau_A$ `with` ε_A and then on the evaluation rule used for $e_A \longrightarrow e_B \mid \varepsilon$. First note that the theorem statement has already been proven for typing judgements from the calculus for fully-labeled programs. Therefore we'll only consider the extra typing rules in the calculus that allows unlabeled objects.

Case. C-NewObj.
Then e_A is of the form $\texttt{new}_d$ so it is a value. Then no reduction can be applied; the theorem statement holds vacuously.

Case. C-MethCall.
Then $e_A = e_1.m(e_2)$ where $e_1 = \texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e} : \{\bar{d}$ `captures` $\varepsilon_c\}$ `with` $\varnothing$, where $\varepsilon_c = \texttt{effects}(\Gamma_A')$, for $\Gamma_A' \subseteq \Gamma_A$. Consider the extension $\hat{e}_A = \texttt{label}(e_A)$. Since `label` only changes type-information, the runtime effects of e_A are invariant under the `label` function. Furthermore, e_A is a value if and only if $\hat{e}_A$ is a value.

Now, $\hat{e}_A$ is not a value because e_A is not a value. Therefore we can reduce $\hat{e}_A$. Then $\hat{e}_A \longrightarrow \hat{e}_B \mid \varepsilon$, the same ε we get when reducing e_A.

By small-step soundness of labeled programs, $\varepsilon \subseteq \hat{\varepsilon}_A$. By refinement, $\hat{\varepsilon}_A \subseteq \varepsilon_A$. So $\varepsilon \subseteq \varepsilon_A$.

Case. C-ConservativeTyping.
Then $\varepsilon_A = \texttt{effects}(\Gamma')$ for some Γ'. Runtime effects must be captured by the environment; therefore $\varepsilon \subseteq \varepsilon_A$.

□