July 23, 2016

1 Effects

R is a fixed set of resources. We define a resource as a language primitive with the authority to perform I/O operations. Π is the set of I/O operations. In this document we cannot dynamically create resources or operations on resources.

Members of R are denoted r; members of Π are denoted π. $r.\pi$ is syntactic sugar for (r, π) (for example, `FileIO.append` instead of $(\texttt{FileIO}, \texttt{append})$). An effect is a member of the pairs $R \times \Pi$. A set of effects is denoted by ε.

We say a piece of code C has the runtime effect $r.\pi$ if $r.\pi$ is called during the execution of C. C captures $r.\pi$ if it has the authority to call $r.\pi$ at some point during its execution. The static effects of C is an approximation of the runtime effects by our typing system. Later on we show the static effects of C give a conservative upper-bound on the runtime effects.

Types in our system are either resources or structural. Structural types are distinguished by what method declarations they have. The type with no methods is called `Unit`; its unique instance of denoted `unit`. Although they look similar in form, operations and methods are distinct. Methods can only be invoked by objects; operations can only be invoked by resources.

We make some simplifying assumptions about methods and operations. Methods take exactly one argument. If the argument is not specified it is assumed to be `unit`. Invoking some operation $r.\pi$ returns $\varnothing$. We don't model arguments to operations, so all operations are null-ary.

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

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

Notes:

- All declarations (σ-terms) are annotated by what effects they have.
- All methods take exactly one argument. If a method specifies no argument the argument is assumed to be of type `Unit`.
- The type $\{\bar{r}\}$ is a set of resources; there will only be one actual resource at run-time, and it will be one of the resources in the set. This covers the case where e.g. a conditional returns a different resource on either branch.

2.2 Static Semantics

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

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

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

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

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

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

Notes:

– In ε-Var, ε-Resource, and ε-NewObj the consequent has an expression typed with no effect. In these rules we may be gaining an authority for an effect but we must use it in some code for that effect to happen.
– ε-ValidImpl says that the return type and effects of the body of a method must be exactly the same as its declaration.

2.3 Dynamic Semantics

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

$$\frac{e_1 \longrightarrow e_1' \mid \varepsilon}{e_1.m(e_2) \longrightarrow e_1'.m(e_2) \mid \varepsilon}\ (\text{E-MethCall1}_\sigma) \qquad \frac{v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} \quad e_2 \longrightarrow e_2' \mid \varepsilon}{v_1.m(e_2) \longrightarrow v_1.m(e_2') \mid \varepsilon}\ (\text{E-MethCall2}_\sigma)$$

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

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

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

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

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

Notes:

- 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-OperCall2 (the rule for evaluating operations on resources).
- Method calls are evaluated by performing substitution on the body of the method. There is no store.

2.4 Substitution Function

Definition 2.4.1. (Substitution)
$[e'/z]e$ means 'substitute every free occurrence of z in e for e'. Here's a definition over rules in the grammar.

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

We use the convention of *alpha-conversion* to make substitution capture-avoiding. In practice, this means that whenever substitution is undefined because of variable capture, we rename variables to meet the side conditions (see Benjamin Pierce, "Types and Programming Languages" p. 71 for more details).

Substitution of multiple variables is written $[e_1/z_1, ..., e_n/z_n]e$, which is shorthand for $[e_n/z_n]...[e_1/z_1]e$

Lemma 2.4.2. (Substitution Lemma)

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

$\boxed{\text{Intuition}}$ If you substitute z for something of the same type, the type of the whole expression stays the same after substitution.

$\boxed{\text{Proof.}}$ By structural induction on possible derivations of $\Gamma \vdash e : \tau\ \texttt{with}\ \varepsilon$. 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.

$\boxed{\text{Case.}}$ ε-Var.
Then $[e'/z]z = e_1$. By assumption $\Gamma \vdash e' : \tau\ \texttt{with}\ \varepsilon$, so $\Gamma \vdash [e'/z]z = e$.

$\boxed{\text{Case.}}$ ε-Resource.
Then $e = r$. $\texttt{freevars}(r) = \varnothing$, so the statement holds vacuously.

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

$\Gamma, z : \tau \vdash \sigma_i = e_i\ \texttt{OK}$ via the inductive assumption. We can use ε-ValidImpl$_\sigma$ to conclude $\overline{\sigma = [e'/z]e}\ \texttt{OK}$. Then by ε-NewObj we type $[e'/z]e$ to the same as the original.

$\boxed{\text{Case.}}$ ε-OperCall.
Then $e = e_1.\pi$. The variable z must appear in e_1. By rule inversion we have a sub-derivation for the type of both sub-expressions so applying the inductive assumption we know e_1 and $[e'/z]e_1$ have the same types. Since e and $[e'/z]e$ have the same syntactic structure, and their corresponding subexpressions have the same types, then ε-OperCall will type $[e'/z]e$ to the same thing as e.

$\boxed{\text{Case.}}$ 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 so applying the inductive hypothesis we know the types of e_1 and $[e'/z]e_1$ are the same, and the types of e_2 and $[e'/z]e_2$ are the same. Since e and $[e'/z]e$ have the same syntactic structure, and their corresponding subexpressions have the same types, then ε-MethCall types $[e'/z](e_1.m_i(e_2))$ to the same as $e_1.m_i(e_2)$.

□

$\boxed{\text{Corollary.}}$ 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 $[e_1'/z_1, ..., e_n'/z_n]e = [e_n'/z_n]...[e_1'/z_1]e$ and induction on the length n.

2.5 Soundness Theorem

In this section we build up to the soundness theorem for our typing system. We do this by proving progress and two preservation theorems about types and static effects. The two preservation theorems directly give us the soundness statement.

Lemma 2.5.1. (Canonical Forms)

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

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

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 and ε-NewObj respectively.

□

Theorem 2.5.2. (Progress Theorem)

$\boxed{\text{Statement.}}$ 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.

$\boxed{\text{Proof.}}$ By induction on possible derivations of $\Gamma \vdash e_A : \tau_A$ `with` ε_A. Consider the last rule used.

$\boxed{\text{Case.}}$ ε-Var, ε-Resource, ε-NewObj.
Then e_A is a value.

$\boxed{\text{Case.}}$ ε-MethCall$_\sigma$.
Then $e_A = e_1.m_i(e_2)$ and the following are known:

- $e_A : \tau_3$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $e_1 : \{\bar{\sigma}\}$ `with` ε_1
- $e_2 : \tau_2$ `with` ε_2
- $\sigma_i = \texttt{def}\ m_i(y : \tau_2) : \tau_3$ `with` ε_3

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

<u>Subcase.</u> 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 $e_A \longrightarrow e_1'.m_i(e_2) \mid \varepsilon$.

<u>Subcase.</u> 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 $e_A \longrightarrow v_1.m_i(e_2') \mid \varepsilon$.

<u>Subcase.</u> $e_1 = v_1$ and $e_2 = v_2$ are values. By Canonical Forms, $e_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$. Also, $\texttt{def}\ m_i(y : \tau_2) : \tau_3$ `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 $e_1.m_i(e_2) \longrightarrow [v_1/x, v_2/y]e_i \mid \varnothing$.

$\boxed{\text{Case.}}$ ε-OperCall.
Then $e_A = e_1.\pi$ and the following are known:

- e_A : $\texttt{Unit with } \{r.\pi\} \cup \varepsilon_1$
- $e_1 : \{\overline{r}\}$ $\texttt{with}$ ε_1

We look at the cases for when e_1 is a value.

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

Subcase. e_1 is a value. By Canonical Forms, $\exists r \in R \mid e_1 = r$. Then E-OperCall3 gives the reduction $r.\pi \longrightarrow \texttt{unit} \mid \{r.\pi\}$.

□

Theorem 2.5.3. (Preservation Theorem)

$\boxed{\text{Statement.}}$ Suppose the following hold:

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

Then $\tau_B = \tau_A$. Also, $\varepsilon \subseteq \varepsilon_A$ and $\varepsilon_B \subseteq \varepsilon_A$ and $\forall r.\pi \in \varepsilon_A \setminus \varepsilon_B \mid r.\pi \in \varepsilon$.

$\boxed{\text{Intuition.}}$ Reduction preserves the relevant static effects in the sense that you only lose static effects during a computation if they actually happen. So you can't gain static effects during reduction, and every lost static effect is "accounted for".

$\boxed{\text{Proof.}}$ By induction on possible derivations of $\Gamma \vdash e_A : \tau_A$ $\texttt{with}$ ε_A. Consider the last rule used.

$\boxed{\text{Case.}}$ ε-Resource, ε-Var, ε-NewObj.
e_A is a value so no reduction can be applied to it. The theorem statement is vacuously satisfied.

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

- $e_A : \tau_3$ $\texttt{with}$ $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$
- $e_1 : \{\overline{\sigma}\}$ $\texttt{with}$ ε_1
- $e_2 : \tau_2$ $\texttt{with}$ ε_2
- $\sigma_i = \texttt{def}\ m_i(y : \tau_2) : \tau_3$ $\texttt{with}$ ε_3

The type to be preserved is τ_3, so we have to show Γ_B must have ascribed the type τ_3 to e_B in its judgement. We do a case analysis on the reductions applicable to $e_1.m_i(e_2)$.

Subcase. E-MethCall1$_\sigma$. Then $e_1 \longrightarrow e_1' \mid \varepsilon$. By inductive assumption $e_1' : \{\bar{\sigma}\}$ $\texttt{with}$ ε_1'. Then by ε-MethCall we have $e_B = e_1'.m_i(e_2) : \tau_3$ $\texttt{with}$ $\varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_3$. Then $\varepsilon_B = \varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_3$ and $\varepsilon_A \setminus \varepsilon_B = \varepsilon_1 \setminus \varepsilon_1'$. For every $r.\pi \in \varepsilon_1 \setminus \varepsilon_1'$ we know $r.\pi \in \varepsilon$ by the inductive assumption. Since e_B has the type τ_3 the type is preserved.

Subcase. E-MethCall2$_\sigma$. Then $e_1 = v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$, and $e_2 \longrightarrow e_2' \mid \varepsilon$. By inductive assumption $e_2' : \tau_2$ $\texttt{with}$ ε_2. Then by ε-MethCall we have $e_B = v_1.m_i(e_2) : \tau_3$ $\texttt{with}$ $\varepsilon_1 \cup \varepsilon_2' \cup \varepsilon_3$. Then $\varepsilon_B = \varepsilon_1 \cup \varepsilon_2' \cup \varepsilon_3$ and $\varepsilon_A \setminus \varepsilon_B = \varepsilon_2 \setminus \varepsilon_2'$. For every $r.\pi \in \varepsilon_2 \setminus \varepsilon_2'$ we know $r.\pi \in \varepsilon$ by the inductive assumption. Since e_B has the type τ_3 the type is preserved.

Subcase. E-MethCall3$_\sigma$ Then $e_1 = v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$, 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 $e_1 : \{\overline{\sigma}\}$ $\texttt{with}$ ε_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, $e' : \tau_3$ $\texttt{with}$ ε_3.

Now $e_B = [v_1/x, v_2/y]e'$ because the rule E-MethCall3 was used. **We know $v_1 = e_1$ and x have the same type, which is $\{\overline{\sigma}\}$ $\texttt{with}$ ε_1. We also know $v_2 = e_2$ and y have the same type, which is τ_2 $\texttt{with}$ ε_2.** By the substitution lemma, the type of e' and $e_B = [v_1/x, v_2/y]e'$ is the same. So $e_B : \tau_3$ $\texttt{with}$ ε_3, and namely $\varepsilon_B = \varepsilon_3$.

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 $\varepsilon_A \setminus \varepsilon_B = \varnothing$, so there are no lost effects to account for. Since e_B has the type τ_3 the type is preserved.

$\boxed{\text{Case.}}$ ε-OperCall.
Then $e_A = e_1.\pi : \texttt{Unit}$, and we know:
- $e_A : \{r, \pi\} \cup \varepsilon_1$
- $e_1 : \{\bar{r}\}$ with ε_1

The type being preserved is Unit, so we have to show Γ_B must have ascribed Unit to e_B. There are two reduction rules applicable to terms of the form $e_1.\pi$.

Subcase. E-OperCall1. Then $e_1 \longrightarrow e_1' \mid \varepsilon$. By inductive assumption, $e_1' : \{\bar{r}\}$ with ε_1'. Then by ε-OperCall we have $e_B = e_1'.\pi : \texttt{Unit with } \{r.\pi\} \cup \varepsilon_1'$. Then $\varepsilon_B = \{r.\pi\} \cup \varepsilon_1'$ and $\varepsilon_A \setminus \varepsilon_B = \varepsilon_1 \setminus \varepsilon_1'$. For every $r.\pi \in \varepsilon_1 \setminus \varepsilon_1'$ we know $r.\pi \in \varepsilon$ by the inductive assumption. Since e_B has the type Unit the type is preserved.

Subcase. E-OperCall2. Then $r.\pi \longrightarrow \texttt{unit} \mid \{r.\pi\}$. By Canonical Forms, $\varepsilon_1 = \varnothing$, so $e_A : \texttt{Unit with } \{r.\pi\}$. By a degenerate case of ε-NewObj, $e_B = \texttt{unit} : \texttt{Unit with } \varnothing$. Then $\varepsilon_A \setminus \varepsilon_B = \{r.\pi\}$, which is exactly the set of runtime effects ε; our only lost effect is accounted for. Since e_B has the type Unit the type is preserved.

□

Theorem 2.5.4. (Soundness Theorem)

$\boxed{\text{Statement.}}$ If $\Gamma_A \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$ either e_A is a value or $e_A \longrightarrow e_B \mid \varepsilon$ and $\exists \Gamma_B \mid \Gamma_B \vdash e_B : \tau_B \texttt{ with } \varepsilon_B$, where $\tau_A = \tau_B$ and $\varepsilon \subseteq \varepsilon_A$.

$\boxed{\text{Proof.}}$ By the progress theorem we know either e_A is a value or a small-step $e_A \longrightarrow e_B \mid \varepsilon$ can be applied. If a typing judgement $\Gamma_B \vdash e_B : \tau_B \texttt{ with } \varepsilon_B$ then we know $\tau_B = \tau_A$ and $\varepsilon \subseteq \varepsilon_A$. It is sufficient to show the existence of Γ_B. We proceed by induction on the typing judgement for $\Gamma_A \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$.

$\boxed{\text{Case.}}$ ε-OperCall.
Then $e_A = e_1.\pi$, and we know:
- $e_A : \{r, \pi\} \cup \varepsilon_1$
- $e_1 : \{\bar{r}\}$ with ε_1

Consider the reduction rule used.

Subcase. E-OperCall1. Then $e_1 \longrightarrow e_1' \mid \varepsilon$. Since $\Gamma_A \vdash e_1 : \tau_1 \texttt{ with } \varepsilon_1$, then by inductive assumption $\exists \Gamma_A' \mid \Gamma_A' \vdash e_1' : \tau_1 \texttt{ with } \varepsilon_1'$.

Subcase. E-OperCall2. Then $r.\pi \longrightarrow \texttt{unit} \mid \{r.\pi\}$. Then $\Gamma_A \vdash \texttt{unit} : \texttt{Unit with } \varnothing$ by a degenerate case of the C-NewObj rule, so choose $\Gamma_B = \Gamma_A$.

□

Theorem 2.5.4. (Soundness Theorem)

$\boxed{\text{Statement.}}$ If $\Gamma_A \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$ and $e \longrightarrow_* e_B \mid \varepsilon$ then $\exists \Gamma_B \mid \Gamma_B \vdash e_B : \tau_B \texttt{ with } \varepsilon_B$, where $\tau_A = \tau_B$ and $\varepsilon \subseteq \varepsilon_A$.

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

$\boxed{\text{Base Case.}}$ The length is 1. The theorem holds by Small-Step Soundness For Mini Calculus.

$\boxed{\text{Inductive Case.}}$ If there is a multi-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.

□

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 in-between; either every method of an object is annotated, or none are.

3.1 Grammar

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

Notes:

- σ 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

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

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

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

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

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

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

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

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

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

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

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

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

Notes:

- This system also includes 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.
- In applying C-NewObj, Γ is the current context. The variable Γ' is some sensible choice of sub-context. A good choice of sub-context is Γ restricted to the free variables in the method-body being type-checked. 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-ConservativeInference 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 are $\texttt{effects}(\Gamma')$, 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

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

$$\begin{array}{rcll} d & ::= & \texttt{def}\ m(x : \tau) : \tau & \textit{unlabeled decls.} \\ \\ \sigma & ::= & d\ \texttt{with}\ \varepsilon & \textit{labeled decls.} \\ \\ \tau & ::= & \{\bar{\sigma}\} & \textit{types} \\ & | & \{\bar{r}\} & \\ \\ \Gamma & ::= & \varnothing & \textit{contexts} \\ & | & \Gamma,\ x : \tau & \end{array}$$

4.3 Rules

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

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

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

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

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

$$\frac{e_1 \longrightarrow e_1' \mid \varepsilon}{e_1.\pi \longrightarrow e_1'.\pi \mid \varepsilon}\ \text{(E-OperCall1)} \qquad \frac{}{r.\pi \longrightarrow \texttt{unit} \mid \{r.\pi\}}\ \text{(E-OperCall3)}$$

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

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

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

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)

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

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

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.

□

Theorem 5.10. (Preservation Under Labeling)

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

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

$\boxed{\text{Case.}}$ $e_1.m(e_2)$.
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$_\sigma$.

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

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

$\boxed{\text{Case.}}$ $\texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e}$.
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)

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

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

$\boxed{\text{Case.}}$ 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 we type $\hat{e}_A$ to something with the effect-set $\hat{\varepsilon} = \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \hat{\varepsilon}_3 \supseteq \hat{\varepsilon}_1 \supseteq \varepsilon$.

$\boxed{\text{Case.}}$ E-OperCall1.
Same as previous case, but use the rule ε-OperCall.

$\boxed{\text{Case.}}$ 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 we type $\hat{e}_1.m(\hat{e}_2)$ to something with the effect-set $\hat{\varepsilon} = \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2 \cup \hat{\varepsilon}_3 \supseteq \hat{\varepsilon}_2 \supseteq \varepsilon$.

$\boxed{\text{Case.}}$ E-OperCall2.
Same as previous case, but use the rule

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

$\boxed{\text{Case.}}$ 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 $\hat{e}_A : \texttt{Unit with } \{r.\pi\} \cup \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2$, 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)

$\boxed{\text{Statement.}}$ If $\Gamma \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$ and $e_A \longrightarrow e_B \mid \varepsilon$ then $\Gamma \vdash e_B : \tau_B \texttt{ with } \varepsilon_B$ and $\varepsilon \subseteq \varepsilon_A$.

$\boxed{\text{Proof.}}$ Induct on the typing judgement for $\Gamma \vdash e_A : \tau_A \texttt{ with } \varepsilon_A$ 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.

$\boxed{\text{Case.}}$ 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.

$\boxed{\text{Case.}}$ C-MethCall.
Then $e_A = e_1.m(e_2)$ where $e_1 = \texttt{new}_\texttt{d}\ x \Rightarrow \overline{d = e} : \{\bar{d} \texttt{ captures } \varepsilon_c\} \texttt{ with } \varnothing$, 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 $\varepsilon \subseteq \varepsilon_A$.

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

□