June 1, 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

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

Notes:

– Declarations (σ-terms) are annotated by what effects they have.
– d-terms do not appear in programs, except as part of σ-terms.
– All methods (and lambda expressions) take exactly one argument. If a method specifies no argument, then the argument is implicitly of type `Unit`.
– Although $e_1.\pi(e_2)$ is a syntactically valid expression, it is only well-formed under the static semantics if e_1 has a resource-type (remembering that π operations can only be performed on resources).

2.2 Rules

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

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

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

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

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

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

$$
\frac{\Gamma,\ x : \tau \vdash e : \tau'\ \texttt{with}\ \varepsilon \quad \sigma = \texttt{def}\ m(x : \tau) : \tau'\ \texttt{with}\ \varepsilon}{\Gamma \vdash \sigma = e\ \texttt{OK}}\ (\varepsilon\text{-}\textsc{ValidImpl}_{\sigma})
$$

Notes:

- The rules ε-Var, ε-Resource, and ε-NewObj have in their consequents an expression typed with no effect: merely having an object or resource is not an effect; you must do something with it, like a call a method on it, in order for it to 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

What happens if we relax the requirement that all methods in an object must be effect-annotated? In the next system we allow objects which have no effect-annotated methods. When an object is annotated we can use the rules from the previous section. When an object has no annotations we use the additional rules introduced here, which give an upper bound on the effects of a program.

3.1 Grammar

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

Notes:

- σ denotes a declaration with effect labels. d denotes a declaration without effect labels.
- There are two new expressions: $\texttt{new}_\sigma$ for objects whose methods are annotated; $\texttt{new}_d$ for objects whose methods aren't.
- $\{\bar{\sigma}\}$ 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 as a consequence of the capture rules. ε is an upper-bound on the possible effects of the object $\{\bar{d}\}$.

3.2 Rules

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

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

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

$$\frac{\Gamma \vdash e_1:\{\bar{\sigma}\},\ \texttt{def}\ m(x:\tau_1):\tau_2\ \texttt{with}\ \varepsilon \in \{\bar{\sigma}\} \quad \Gamma \vdash e_2:\tau_1}{\Gamma \vdash e_1.m(e_2):\tau_2}\ (\text{T-MethCall}_\sigma)$$

$$\frac{\Gamma \vdash e_1:\{\bar{d}\},\ \texttt{def}\ m(x:\tau_1):\tau_2 \in \{\bar{d}\} \quad \Gamma \vdash e_2:\tau_1}{\Gamma \vdash e_1.m(e_2):\tau_2}\ (\text{T-MethCall}_d)$$

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

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

$$\frac{d = \texttt{def}\ m(x : \tau_1) : \tau_2 \quad \Gamma \vdash e : \tau_2}{\Gamma \vdash d = e\ \texttt{OK}}\ (\varepsilon\text{-}\textsc{ValidImpl}_d)$$

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

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

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

Notes:

- Rules with the judgement form $\Gamma \vdash e : \tau$ do standard typing judgements on structural objects, without any effect analysis. These rules are needed to apply the ε-ValidImpl$_d$ rule.
- The ε judgements from the previous section are to be applied to annotated parts of the program; the C from this section are for unannotated parts.
- In applying C-NewObj the variable 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 means we only consider the effects used in the method-body, giving a tighter upper bound on the effects.
- To perform effect analysis on an unannotated object $\{\bar{d}\}$ we give it the type $\{\bar{d}\ \texttt{captures}\ \varepsilon\}$ by the rule C-NewObj, where ε is an upper-bound on the possible effects that object can have. If a method is called on that object, C-NewObj concludes the effects to be those captured in ε.

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.

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

QUESTION: to make `effects` total over the set of types we should define it on types of the form $\{\bar{d}\ \texttt{captures}\ \varepsilon\}$. Otherwise we might be in trouble, since the input Γ could theoretically have these types in it (although I think with these rules it never will in practice). The definition should probably be $\texttt{effects}(\{\bar{d}\ \texttt{captures}\ \varepsilon\}) = \varepsilon$, because if it is already annotated with what it captures then we must have previously called `effects` on it.

4 Dynamic Semantics

4.1 Terminology

- If e is an expression, then $[e_1/x_1, ..., e_n/x_n]e$ is a new expression, the same as e, but with every free occurrence of x_i replaced by e_i.
- $\varnothing$ is the empty set. The empty type is denoted `Unit`. Its single instance is `unit`.
- A configuration is a pair $e \mid \varepsilon$.
- To execute a program e is to perform reduction steps starting from the configuration $e \mid \varnothing$.
- $e_1 \mid \varepsilon_1 \longrightarrow_* e_2 \mid \varepsilon_2$ if $e_2 \mid \varepsilon_2$ can be obtained by applying one or more reduction rules to $e_1 \mid \varepsilon_1$.
- If $e_1 \mid \varepsilon_1 \longrightarrow_* v \mid \varepsilon_2$, for some value v then we say that $e_1 \mid \varepsilon_1$ terminates.

4.2 Grammar

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

$$
\begin{array}{rll}
\tau ::= & \{\bar{\sigma}\} & \textit{types} \\
\mid & \{\bar{r}\} & \\
\\
\Gamma ::= & \varnothing & \textit{contexts} \\
\mid & \Gamma,\ x : \tau &
\end{array}
$$

4.3 Rules

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

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

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

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

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

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

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

5 Theorems

Lemma 5.1. (Atom)

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

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

Theorem 5.1. (Progress)

$\boxed{\text{Statement.}}$ If $e_A : \tau$ `with` ε, then for any configuration $e_A \mid \varepsilon_A$, either e_A is a value or $e_A \mid \varepsilon \longrightarrow e_B \mid \varepsilon_B$.

$\boxed{\text{Proof.}}$ By structural induction on possible derivations of $e_A : \tau$ `with` ε. We consider every rule which could have made this typing judgement.

$\boxed{\text{Case.}}$ ε-Var.
Then $e_A = x$ is a value.

$\boxed{\text{Case.}}$ ε-Resource.
Then $e_A = r$ is a value.

$\boxed{\text{Case.}}$ ε-NewObj.
Then $e_A = new_\sigma\ x \Rightarrow \overline{\sigma = e}$ is a value.

$\boxed{\text{Case.}}$ C-NewObj.
Then $e_A = x$ is a value.

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

- $e_1 : \{\bar{\sigma}\}$ `with` ε_1
- $e_2 : \tau_2$ `with` ε_2
- $\sigma_i = \texttt{def}\ m_i(y : \tau_2) : \tau$ `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 includes the subderivation $e_1 : \{\bar{\sigma}\}$ `with` ε_1, so by the inductive hypothesis. Then $e_1\ \ \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. Then applying E-MethCall1 to $e \mid \varepsilon_A$, we have $e_A \mid \varepsilon_A \longrightarrow e_1'.m_i(e_2) \mid \varepsilon_B$.

<u>Subcase.</u> e_2 is not a value. Then without loss of generality, $e_1 = v_1$ is a value. Also, $e_2 : \tau_2$ `with` ε_2 is a subderivation. By the inductive hypothesis, $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. Then applying E MethCall2$_\sigma$ to $e_A \mid \varepsilon_A$, we have $e_A \mid \varepsilon_A \longrightarrow v_1.m_i(e_2') \mid \varepsilon_B$.

<u>Subcase.</u> $e_1 = v_1$ and $e_2 = v_2$ are values. By the Atom lemma, $e_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$. Also, $\texttt{def}\ m_i(y : \tau_2)\tau$ `with` $\varepsilon_3 = e_i \in \overline{\sigma = e}$. Then applying E-MethCall3$_\sigma$ to $e_A \mid \varepsilon_A$, we have $e_A \mid \varepsilon_A \longrightarrow [v_1/x, v_2/y]e_i \mid \varepsilon_A$

$\boxed{\text{Case.}}$ ε-OperCall.
Then $e_A = e_1.\pi(e_2) : \texttt{Unit}$ `with` $\{r.\pi\} \cup \varepsilon_1 \cup \varepsilon_2$ and the following are known:

- $e_1 : \{\bar{r}\}$ `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, so applying the inductive assumption, we have $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. Then applying E-OperCall1 to $e_A \mid \varepsilon_A$ we have $e_1.\pi(e_2) \mid \varepsilon_A \longrightarrow e_1'.\pi(e_2) \mid \varepsilon_B$.

Subcase. e_2 is not a value. Without loss of generality, $e_1 = v_1$ is a value. $e_2 : \tau_2$ `with` ε_2 is a subderivation, so applying the inductive assumption, we have $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. Then applying E-OperCall2 to $e_A \mid \varepsilon_A$ we have $v_1.\pi(e_2) \mid \varepsilon_A \longrightarrow v_1.\pi(e_2')$.

Subcase. e_1 and e_2 are values. By the Atom lemma, $e_1 = r$ for some $r \in R$. Then applying E-OperCall3 to $e_A \mid \varepsilon_A$, we have $r.\pi(v_2) \mid \varepsilon_A \longrightarrow \texttt{unit} \mid \varepsilon_A \cup \{r.\pi\}$.

$\boxed{\text{Case.}}$ C-MethCall.
Then $e_A = e_1.m_i(e_2)$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup$ $\texttt{effects}(\tau_2) \cup \varepsilon$ and the following are known:

- $e_1 : \{\bar{d}\ \texttt{captures}\ \varepsilon\}\ \texttt{with}\ \varepsilon_1$
- $e_2 : \tau_2\ \texttt{with}\ \varepsilon_2$
- $d_i = \texttt{def}\ m_i(y : \tau_2) : \tau$

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

Subcase. e_1 is not a value. $e_1 : \{\bar{d}\ \texttt{captures}\ \varepsilon\}\ \texttt{with}\ \varepsilon_1$ is a subderivation. By the inductive hypothesis, $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. Then applying E-MethCall1 to $e_A \mid \varepsilon_A$, we have $e_1.m_i(e_2) \mid \varepsilon_A \longrightarrow e_1'.m_i(e_2) \mid \varepsilon_B$.

Subcase. e_2 is not a value. Without loss of generality, $e_1 = v_1$ is a value. Also, $e_2 : \tau_2\ \texttt{with}\ \varepsilon_2$ is a subderivation. By the inductive hypothesis, $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. Then applying E-MethCall2$_d$ to $e_A \mid \varepsilon_A$, we have $v_1.m_i(e_2) \mid \varepsilon_A \longrightarrow v_1.m_i(e_2') \mid \varepsilon_B$.

Subcase. e_1 and e_2 are values. By the Atom lemma, $e_1 = \texttt{new}_d\ x \Rightarrow \overline{d = e}$. Also, $\texttt{def}\ m_i(y : \tau_2) : \tau = e_i \in \overline{d = e}$. Then applying E-MethCall3$_d$ to $e_A \mid \varepsilon_A$, we have $v_1.m_i(v_2) \mid \varepsilon_A \longrightarrow [v_1/x, v_2/y]e_i \mid \varepsilon_A$

□

Theorem 5.2. (Effect Preservation)

$\boxed{\text{Statement.}}$ If $e_A \mid \varepsilon_A \longrightarrow e_B \mid \varepsilon_B$, then $\varepsilon_A \subseteq \varepsilon_B$.

$\boxed{\text{Proof.}}$ We can divide reduction rules into three classes of rules based on what they do to the effect-set of a configuration. We consider each.

$\boxed{\text{Case.}}$ E-MethCall3$_d$, E-MethCall3$_\sigma$.
In these rules $\varepsilon_A = \varepsilon_B$.

$\boxed{\text{Case.}}$ E-MethCall1, E-MethCall2$_\sigma$, E-MethCall2$_d$, E-OperCall1, E-OperCall2.
In these rules the antecedent contains a subreduction of the form $e \mid \varepsilon_A \longrightarrow e' \mid \varepsilon_B$. By the inductive assumption, $\varepsilon_A \subseteq \varepsilon_B$.

$\boxed{\text{Case.}}$ E-OperCall3.
We have $\varepsilon_B = \varepsilon_A \cup \{r.\pi\}$, so $\varepsilon_A \subseteq \varepsilon_B$.

□

Theorem 5.3. (Type Preservation)

$\boxed{\text{Statement.}}$ If $e_A : \tau\ \texttt{with}\ \varepsilon$ and $e_A \mid \varepsilon_A \longrightarrow e_B \mid \varepsilon_B$, then $e_B : \tau\ \texttt{with}\ \varepsilon$.

$\boxed{\text{Proof.}}$ We first induct on possible derivations of $e_A : \tau\ \texttt{with}\ \varepsilon$, and then on the rule used to reduce $e_A \mid \varepsilon_A$ to $e_B \mid \varepsilon_B$.

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

$\boxed{\text{Case.}}$ ε-MethCall$_\sigma$.
Then $e_A = e_1.m_i(e_2) : \tau\ \texttt{with}\ \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon$ and the following are true:

- $e_A : \tau\ \texttt{with}\ \varepsilon_1 \cup \varepsilon_2 \cup \texttt{effects}(\tau_2) \cup \varepsilon$
- $e_1 : \{\overline{\sigma}\}\ \texttt{with}\ \varepsilon_1$

– $e_2 : \tau_2$ `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 $e_1 \mid \varepsilon_A \to e_1' \mid \varepsilon_B$. By the inductive assumption $e_1' : \{\bar{\sigma}\}$ `with` ε. Then by ε-MethCall we have $e_B = e_1'.m_i(e_2) : \tau$ `with` ε.

Subcase. E-MethCall2$_\sigma$ Then $e_1 = v_1 = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$, and $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. By the inductive assumption $e_2' : \tau_2$ `with` ε_2. Then by ε-MethCall we have $e_B = v_1.m_i(e_2) : \tau$ `with` ε.

Subcase. E-MethCall3$_\sigma$ Then $e_1 = v_1 = \texttt{new}_\sigma \Rightarrow \overline{\sigma = e}$, and `def` $m_i(y : \tau_2) : \tau$ `with` $\varepsilon_3 = e' \in \overline{\sigma = e}$, and $e_2 = v_2$ is a value.

Now, since we know $e_1 : \{\overline{\sigma}\}$ `with` ε_1, the only rule with this conclusion is ε-NewObj. Then the premises of that rule must hold. So $\overline{\sigma = e}$ `OK`. The only rule with this conclusion is ε-ValidImpl$_\sigma$. The premises of that rule must hold, so $e' : \tau$ `with` ε_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 $\{\overline{\sigma}\}$ `with` ε_1. $v_2 = e_2$ and y have the same type τ_2 `with` ε_2. So the type of e', which is τ `with` ε_3, is preserved by the substitution. So $e_B : \tau$ `with` ε_3.

$\boxed{\text{Case.}}$ ε-OperCall$_\sigma$.

Then $e_A = e_1.\pi(e_2) :$ `Unit with` $\{r, \pi\} \cup \varepsilon_1 \cup \varepsilon_2$, and we know:
– $e_1 : \{\bar{r}\}$ `with` ε_1
– $e_2 : \tau_2$ `with` ε_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 $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. By the inductive assumption, $e_1' : \{\bar{r}\}$ `with` ε_1. From these we can apply ε-OperCall, giving $e_B = e_1'.\pi(e_2) :$ `Unit with` $\{r.\pi\} \cup \varepsilon_1 \cup \varepsilon_2$.

Subcase. E-OperCall2. Then $e_1 = r$ for some $r \in R$ and $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. By the inductive assumption $e_2' : \tau_2$ `with` ε_2. From these we can apply ε-OperCall, giving $e_B = r.\pi(e_2')$ `with` ε.

Subcase. E-OperCall3. Then $r.\pi(v) \mid \varepsilon_A \longrightarrow$ `unit` $\mid \varepsilon_A \cup \{r.\pi\}$. By the Atom lemma, $\varepsilon_1 = \varepsilon_2 = \varnothing$, so $e_A :$ `Unit with` $\{r.\pi\}$. By a degenerate case of ε-NewObj, `unit` : `Unit with` $\varnothing$. **But this isn't the same type as e_A. After performing an operation you lose the effect from the type information, but gain it in the runtime information. Should the statement really be reworded to say that you don't lose effect annotations as you reduce, except when performing an operation at which point the runtime gains something from the type info (and that type info is allowed to be discarded). We can use the fact that once you call an operation, evaluation (on this particular configuration) must stop, so the only time we discard effect annotations for a configuration is when it's about to terminate**

$\boxed{\text{Case.}}$ C-MethCall.

Then $e_A = e_1.m_i(e_2)$ and the following are known:
– $e_A : \tau$ `with` $\varepsilon_1 \cup \varepsilon_2 \cup \texttt{effects}(\tau_2) \cup \varepsilon$
– $e_1 : \{\bar{d}$ `captures` $\varepsilon\}$ `with` ε_1
– $e_2 : \tau_2$ `with` ε_2
– $d_i =$ `def` $m_i(y : \tau_2) : \tau$

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

Subcase. E-MethCall1 Then $e_1.m_i(e_2) \mid \varepsilon_A \longrightarrow e_1'.m_i(e_2) \mid \varepsilon_B$. By the inductive assumption e_1' types to the same as e_1. Then applying C-MethCall we get $\texttt{type}(e_B) = \texttt{type}(e_A)$.

Subcase. E-MethCall2$_d$ Then $e_1 = v_1$ some value and $v_1.m_i(e_2) \mid \varepsilon_A \longrightarrow v_1.m_i(e_2')$. By the inductive assumption $\texttt{type}(e_2') = \texttt{type}(e_2)$. Then applying C-MethCall we get $\texttt{type}(e_B) = \texttt{type}(e_A)$.

Subcase. E-MethCall3$_d$ Then $v_1.m_i(v_2) \mid \varepsilon_A \longrightarrow [v_1/x, v_2/y]e_i \mid \varepsilon_B$, where e_i is the body of method m_i. Now e_1 has the type $\{\bar{d}$ `captures` $\varepsilon\}$ `with` ε_1, and the only rule matching this judgement is C-NewObj. So the premises of that rule, applied to e_1, must be true.

Firstly this means $\overline{d = e}$ `OK`, so $d_i = e_i$ `OK`.

But what happens to the effects? e_i and τ could be anything and there are no rules

for effect-checking isolated expressions. Is there a smarter way than proceeding by case analysis on e_i and τ?

$\square$

Theorem 5.4 (Soundness Of Effects).

$\boxed{\text{Statement.}}$ If $e_A : \tau$ `with` ε and $e_A \mid \varepsilon_A \longrightarrow e_B \mid \varepsilon_B$, then $\varepsilon_B \setminus \varepsilon_A \subseteq \varepsilon$.

$\boxed{\text{Proof.}}$ By the Effect Preservation Theorem, $\varepsilon_A \subseteq \varepsilon_B$. Now proceed by structural induction on the evaluation rule.

$\boxed{\text{Case.}}$ E-MethCall3_d, E-MethCall3_σ.
In these rules $e_A = e_B$. Then $e_B \setminus e_A = \varnothing$, so $e_B \setminus e_A \subseteq \varepsilon$ is vacuously true.

$\boxed{\text{Case.}}$ E-OperCall1.
Then $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. Theorem holds by the inductive assumption.

$\boxed{\text{Case.}}$ E-OperCall2.
Then $\exists r \in R \mid e_1 = r$ and $r.\pi(e_2) \longrightarrow r.\pi(e_2')$ and $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. Theorem holds by the inductive assumption.

$\boxed{\text{Case.}}$ E-OperCall3.
Then $\varepsilon_B = \varepsilon_A \cup \{r.\pi\}$, so we have to show $r.\pi \in \varepsilon$. From the Atom lemma, $r : \{r\}$ `with` $\varnothing$ and $v : \tau_2$ `with` $\varnothing$, for some τ_2. By applying ε-OperCall then $r.m(\pi) : \{r.\pi\} \cup \varnothing \cup \varnothing$. So $\varepsilon = \{r.\pi\}$.

$\boxed{\text{Case.}}$ E-MethCall1.
Then $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. Theorem holds by the inductive assumption.

$\boxed{\text{Case.}}$ E-MethCall2_σ.
Then $e_1 = v$ some value and $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. Theorem holds by the inductive assumption.

$\boxed{\text{Case.}}$ E-MethCall2_d.
Then $e_1 = v = \texttt{new}_d\ x \Rightarrow \overline{d = e}$ and $e_2 \mid \varepsilon \longrightarrow e_2' \mid \varepsilon_B$. Theorem holds by inductive assumption.

$\square$