May 27, 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

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

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

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

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

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

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

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

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

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

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

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

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.

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

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

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

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

4.3 Rules

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

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

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

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

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

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

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

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

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

5 Theorems

Lemma 1 (Atom). *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* $e = \texttt{new}_d\ x \Rightarrow \overline{d = e}$.

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 1 (Progress). *Suppose the following holds:*

- $e_A : \tau$ `with` ε

Then for any configuration $e_A \mid \varepsilon_A$ one of the following is true:

- e_A *is a value.*
- $e_A \mid \varepsilon_A \longrightarrow e_B \mid \varepsilon_B$

Proof. By structural induction on possible derivations of $e_A : \tau$ `with` ε.

Case ε-Var*:* Then $e_A = x$ is a value.

Case ε-Resource*:* Then $e_A = r$ is a value.

Case ε-NewObj*:* Then $e_A = \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e}$ is a value.

Case C-NewObj*:* Then $e_A = \texttt{new}_d\ x \Rightarrow \overline{\sigma = e}$, which is a value.

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.

Subcase 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 $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. Then applying E-MethCall1 to $e \mid \varepsilon_A$, we have $e_A \mid \varepsilon_A \longrightarrow e_1'.m_i(e_2) \mid \varepsilon_B$.

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 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 $e_A \mid \varepsilon_A \longrightarrow v_1.m_i(e_2') \mid \varepsilon_B$.

Subcase $e_1 = v_1$ 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 $e_A \mid \varepsilon_A \longrightarrow [v_1/x, v_2/y]e_i \mid \varepsilon_A$.

So we're done.

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:
- $e_1 : \{\bar{\sigma}\}$ `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: $e_1 : \{\bar{\sigma}\}$ `with` ε_1 is a subderivation. By the inductive hypothesis $e_1 \mid \varepsilon_A \longrightarrow e_1' \mid \varepsilon_B$. 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. Furthermore, it is some resource r by the Atom lemma. Now, $e_2 : \tau_2$ `with` ε_2 is a subderivation. By the inductive hypothesis $e_2 \mid \varepsilon_A \longrightarrow e_2' \mid \varepsilon_B$. Then applying E-OperCall2 to $e_A \mid \varepsilon_A$, we have $r.\pi(e_2) \mid \varepsilon_A \longrightarrow r.\pi(e_2') \mid \varepsilon_B$.

Subcase e_1 *and* $e_2 = v_2$ *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 $r.\pi(v_2) \mid \varepsilon_A \longrightarrow \texttt{unit} \mid \varepsilon_A \cup \{r.\pi\}$.

So we're done.

Case C-MethCall*:* Then $e_A = e_1.m_i(e_2)$ $\texttt{with}$ $\varepsilon_1 \cup \varepsilon_2 \cup$ $\texttt{effects}(\tau_2) \cup \varepsilon$ and the following are known:
– $e_1 : \{\bar{d}$ $\texttt{captures}$ $\varepsilon\}$ $\texttt{with}$ ε_1
– $e_2 : \tau_2$ $\texttt{with}$ ε_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*: $e_1 : \{\bar{d}$ $\texttt{captures}$ $\varepsilon\}$ $\texttt{with}$ ε_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, $e_2 : \tau_2$ $\texttt{with}$ ε_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 $e_1 = v_1$ *and* $e_2 = v_2$ *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 $v_1.m_i(v_2) \mid \varepsilon_A \longrightarrow [v_1/x, v_2/y]e_i \mid \varepsilon_A$.
So we're done.

This concludes all the cases. So either e is a value, or a single reduction step can be made on $e_A \mid \varepsilon_A$ to give a new configuration $e_B \mid \varepsilon_B$. □

Theorem 2 (Preservation). *If the following holds:*

– $e_A : \tau$ $\texttt{with}$ ε
– $e_A \mid \varepsilon_A \longrightarrow e_B \mid \varepsilon_B$

Then $e_B : \tau$ $\texttt{with}$ ε*.*

Proof. By structural induction on possible derivations of $e_A : \tau$ $\texttt{with}$ ε. First, if the rule used was ε-Resource, ε-Var, ε-NewObj, or C-NewObj, then e_A is a value, so no reduction can be applied to it. This means the theorem is vacuously satisfied. Otherwise we consider the remaining rules and then induct on possible derivations of $e_A \mid \varepsilon_A \longrightarrow e_B \mid \varepsilon_B$.

Case ε-MethCall$_\sigma$*:* Then $e_A = e_1.m_i(e_2) : \tau$ $\texttt{with}$ $\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon$, and we know:
– $e_1 : \{\overline{\sigma}\}$ $\texttt{with}$ ε_1
– $e_2 : \tau_2$ $\texttt{with}$ ε_2
– $\sigma_i = \texttt{def}$ $m_i(y : \tau_2) : \tau$ $\texttt{with}$ 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 $e_1' : \{\bar{\sigma}\}$ $\texttt{with}$ ε. Then by ε-MethCall we have $e_B = e_1'.m_i(e_2) : \tau$ $\texttt{with}$ 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 $e_2' : \tau_2$ $\texttt{with}$ ε_2. Then by ε-MethCall we have $e_B = e_1'.m_i(e_2) : \tau$ $\texttt{with}$ ε.

Subcase E-METHCALL3$_\sigma$: Then $e_1 = v_1 = \texttt{new}_\sigma \Rightarrow \overline{\sigma = e}$, and $\texttt{def}\ m_i(y : \tau_2) : \tau\ \texttt{with}\ \varepsilon_3 = e' \in \overline{\sigma = e}$, and $e_2 = v_2$ is a value. Furthermore, since we know $e_1 : \{\overline{\sigma}\}\ \texttt{with}\ \varepsilon_1$, the only rule with this conclusion is ε-NEWOBJ. Then its antecedent must hold. of ε-NEWOBJ holds, so $\overline{\sigma = e}\ \texttt{OK}$. The only rule with this conclusion is ε-VALIDIMPL$_\sigma$. Then its antecedent must hold, so $e' : \tau\ \texttt{with}\ \varepsilon_3$.

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 $\{\overline{\sigma}\}\ \texttt{with}\ \varepsilon_1$. $v_2 = e_2$ and y have the same type $\tau_2\ \texttt{with}\ \varepsilon_2$. So the type of e', which is $\tau\ \texttt{with}\ \varepsilon_3$, is preserved by the substitution. So $e_B : \tau\ \texttt{with}\ \varepsilon_3$.

Theorem 3 (Monotonicity). *If* $e_A \mid \varepsilon_A \longrightarrow_* e_B \mid \varepsilon_B$, *then* $\varepsilon_A \subseteq \varepsilon_B$.

Proof. Consider the degenerate case where $e_A \mid \varepsilon_A \longrightarrow_* e_B \mid \varepsilon_B$ consists of a single reduction. We induct on that reduction, considering three classes of rules.

Case E-METHCALL3$_d$, E-METHCALL3$_\sigma$*:* In these rules $\varepsilon_A = \varepsilon_B$.

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, $\varepsilon_A \subseteq \varepsilon_B$.

Case E-OPERCALL3*:* We have $\varepsilon_B = \varepsilon_A \cup \{r.\pi\}$, so $\varepsilon_A \subseteq \varepsilon_B$.

Therefore the theorem statement holds for single-step reductions. If $e_A \mid \varepsilon_A \longrightarrow_* e_B \mid \varepsilon_B$ is a sequence of reduction steps then it holds by induction on the length of the sequence. □

Theorem 4 (Soundness). *If the following holds:*

- $e_A : \tau\ \texttt{with}\ \varepsilon$
- $e_A \mid \varnothing \longrightarrow_* e_B \mid \varepsilon_B$

Then $\varepsilon_B \subseteq \varepsilon$

Note 1. A more general form of soundness is that if $e_A \mid \varepsilon_A \longrightarrow_* e_B \mid \varepsilon_B$, then $\varepsilon_B \setminus \varepsilon_A \subseteq \varepsilon$.

Proof.