June 5, 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:

- 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 (as π operations can only be performed on resources).

2.2 Rules

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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/sequence. The empty type is denoted `Unit`. Its single instance is `unit`.
- A small-step reduction has the form $e \mid \Sigma \longrightarrow e' \mid \Sigma' \mid \varepsilon'$. Σ and Σ' are the stores before and after; e and e' are the expression to be evaluated before and after; ε is the set of effects during execution of e.
- A store Σ is a mapping from variables to value-type pairs (v, τ).
- To execute a program e is to perform reduction steps starting from the configuration $e \mid \varnothing$.
- $e_1 \mid \Sigma_1 \longrightarrow_* e_2 \mid \Sigma_2 \mid \varepsilon_2$ if multiple small-step rules can be applied to $e_1 \mid \Sigma_1$, with ε_2 as the set of accumulated effects.
- If $e_1 \mid \Sigma_1 \longrightarrow_* v \mid \Sigma_2 \mid \varepsilon_2$, for some value v then we say that $e_1 \mid \Sigma$ terminates.

4.2 Grammar

$$
\begin{array}{lll}
e ::= x & & \textit{expressions} \\
\quad \mid \quad e.m(e) & & \\
\quad \mid \quad e.\pi(e) & & \\
\quad \mid \quad v & &
\end{array}
$$

$$
\begin{array}{lll}
v ::= r & & \textit{values} \\
\quad \mid \quad \texttt{new}_\sigma\ x \Rightarrow \overline{\sigma = e} & & \\
\quad \mid \quad \texttt{new}_d\ x \Rightarrow \overline{d = e} & &
\end{array}
$$

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

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

$$
\begin{array}{lll}
\tau ::= \{\bar{\sigma}\} & & \textit{types} \\
\quad \mid \quad \{\bar{r}\} & &
\end{array}
$$

$$
\begin{array}{lll}
\Gamma ::= \varnothing & & \textit{context} \\
\quad \mid \quad \Gamma,\ x : \tau & &
\end{array}
$$

$$
\begin{array}{lll}
\Sigma ::= \varnothing & & \textit{store} \\
\quad \mid \quad \Sigma,\ x \mapsto (v, \tau) & &
\end{array}
$$

4.3 Rules

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

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

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

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

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

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

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

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

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

$$\frac{e_1 \mid \Sigma_1 \longrightarrow e_2 \mid \Sigma_2 \mid \varepsilon_2 \quad e_2 \mid \Sigma_2 \longrightarrow e_3 \mid \Sigma_3 \mid \varepsilon_3}{e_1 \mid \Sigma_1 \longrightarrow_* e_3 \mid \Sigma_3 \mid \varepsilon_2 \cup \varepsilon_3} \text{ (E-BigStep1)}$$

$$\frac{e_1 \mid \Sigma_1 \longrightarrow_* e_2 \mid \Sigma_2 \mid \varepsilon_2 \quad e_2 \mid \Sigma_2 \longrightarrow_* e_3 \mid \Sigma_3 \mid \varepsilon_3}{e_1 \mid \Sigma_1 \longrightarrow_* e_3 \mid \Sigma_3 \mid \varepsilon_3} \text{ (E-BigStep2)}$$