

May 8, 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 \emptyset with no methods is called **Unit**.

We assume there are constructions of the familiar types using the basic structural type \emptyset and method declarations (for example, \mathbb{N} could be made using \emptyset 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 \emptyset .

2 Fully-Annotated Programs

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

2.1 Grammar

$$\begin{array}{ll}
 e ::= x & \text{expressions} \\
 \mid r & \\
 \mid \mathbf{new} \ x \Rightarrow \overline{\sigma = e} & \\
 \mid e.m(e) & \\
 \mid e.\pi(e) & \\
 \\
 \tau ::= \{\bar{\sigma}\} \mid \{\bar{r}\} & \text{types} \\
 \\
 \sigma ::= d \text{ with } \varepsilon & \text{labeled decls.} \\
 \\
 d ::= \mathbf{def} \ m(x : \tau) : \tau & \text{unlabeled decls.} \\
 \\
 \Gamma ::= \emptyset & \\
 \mid \Gamma, x : \tau &
 \end{array}$$

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 if e_1 is a resource (so e_1 is only a resource in well-typed programs).

2.2 Rules

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

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

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

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

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

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

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

Notes:

- Every expression in the program must be explicitly annotated; either as σ -terms or by what they capture.
- 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 be an effect.
- ε -VALIDIMPL says that the return type and effects of the body of a method must agree with what its signature says.
- In ε -METHCALLRESOURCE, we may only call a method m on a resource r if m is a predefined operation in the set M . Invoking m returns the resource r you called it upon (which has potentially different state afterwards).

3 Partially-Annotated Programs

In this second system methods may either be fully labeled with their effects or have no labels. When they have no labels a conservative effect inference is performed using rules which provide an upper-bound (not necessarily tight) on the effects of the code when executed.

3.1 Grammar

$e ::= x$	<i>expressions</i>
r	
$\mathbf{new}_\sigma x \Rightarrow \overline{\sigma = e}$	
$\mathbf{new}_d x \Rightarrow \overline{d = e}$	
$e.m(e)$	
$e.\pi(e)$	
$\tau ::= \{\bar{\sigma}\}$	<i>types</i>
$\{\bar{r}\}$	
$\{\bar{d}\}$	
$\{\bar{d} \text{ captures } \varepsilon\}$	
$\sigma ::= d \text{ with } \varepsilon$	<i>labeled decls.</i>
$d ::= \mathbf{def } m(x : \tau) : \tau$	<i>unlabeled decls.</i>

Notes:

- σ denotes a declaration with effect labels. d denotes a declaration without effect labels.
- There are two new expressions: \mathbf{new}_σ for objects whose declarations are annotated; \mathbf{new}_d for objects whose declarations aren't.
- $\{\bar{d} \text{ captures } \varepsilon\}$ is a special kind of type that doesn't appear in the source program, but may be assigned as a consequence of the capture rules.

3.2 Rules

In addition to the rules from the previous system, the partially-annotated system has the following rules.

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

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

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

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

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

$$\frac{\Gamma \vdash \sigma_i = e_i \text{ OK}}{\Gamma \vdash \mathbf{new}_\sigma x \Rightarrow \overline{\sigma = e} : \{\bar{\sigma}\}} \text{ (T-NEW}_\sigma\text{)}$$

$$\frac{\Gamma \vdash d_i = e_i \text{ OK}}{\Gamma \vdash \mathbf{new}_d x \Rightarrow \overline{d = e} : \{\bar{d}\}} \text{ (T-NEW}_d\text{)}$$

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

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

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

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

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

Notes:

- The ε judgements are to be applied to annotated parts of the program; the C rules for unannotated parts.
- The rules ε -VAR, ε -RESOURCE, and ε -NEWOBJ have in their antecedents 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 your program to have effects.
- The T judgements before standard typechecking, but they operate on annotated terms. They are needed to apply the ε -VALIDIMPL_d rule.
- 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 and gives a better approximation of its effects.
- When an unannotated d -declaration is encountered it is first assigned a γ -type by C-NEWOBJ. This annotates it as capturing a certain set of effects. C-METHCALL can then conclude its effects to be what it captures.

3.3 Effects Function

The **effects** function returns the set of effects in a particular typing 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 .

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