April 27, 2016

# 1 Example 1

This example is a fully-annotated program. We can check it using rules from the fully-annotated system.

```
1  // $\Gamma_0 = \{FileIO : \{FileIO\}\}$
2  let logger1 = new
3      def log(entry : String) : Unit with FileIO.append
4         FileIO.append('/logs/mylog.txt', entry)
5
6  // $\Gamma_1 = \{FileIO : \{FileIO\},\ logger1 : \{log : String \to Unit\}\}$
7  in new
8     def main() : Unit with FileIO.append
9        logger1.log('Hello, World!')
```

Start with $\Gamma_0$. After execution of line 2, we obtain $\Gamma_1$. Line 7 declares an unannotated object type so we want to match it with the consequent in $\varepsilon$-NewObj.

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

To prove $\overline{\sigma = e}$ $\texttt{OK}$ we need the following rules.

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

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

$\texttt{logger1.log}(''\texttt{Hello,world!}'')$ can be checked with $\varepsilon$-MethCallResource. In this case, $logger1 : \{...\}\ \texttt{with}\ \varnothing$ by $\varepsilon$-Var and $''\texttt{Hello,world!}'' : String\ \texttt{with}\ \varnothing$ (but there's no rule for constants). The definition of `log` says that it has the effect `FileIO.append`, so the effect set for $\texttt{logger1.log}(''\texttt{Hello,world!}'')$ is the singleton $\{\texttt{FileIO.append}\}$.

With the body of `main` typechecked we can apply $\varepsilon$-ValidImpl$_\sigma$, because the annotation for `main` matches the effect we computed for its body. Then we know that the method implementations for the new object are well-formed.

Finally we may apply $\varepsilon$-NewObj. We conclude that the type is $\{\texttt{main} : \texttt{Unit} \to \texttt{Unit with}\ \{\texttt{FileIO.append}\}\}$ .

# 2 Example 2

This example is like the previous one but the main object is not annotated. So we need to use the capture-rules from the partially-annotated system.

```
1  // $\Gamma_0 = \{FileIO : \{FileIO\}\}$
2  let logger1 = new
3      def log(entry : string) : Unit with FileIO.append
4         FileIO.append('/logs/mylog.txt', entry)
5
6  // $\Gamma_1 = \{FileIO : \{FileIO\},\ logger1 : \{log : String \to Unit\}\}$
7  in new
8     def main() : Unit
9        logger1.log('Hello, World!')
```

Start with $\Gamma_0$. After execution of line 2, we obtain $\Gamma_1$. Line 7 declares an unannotated object type so we want to match that with the consequent of C-NewObj.

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

### Typechecking

We must type the body of the `main` method. First we type `logger1`. As $\texttt{logger1} \in \Gamma$ we can apply T-Var.

There is no rule for typechecking string constants but it should typecheck to `String`.

Then we can typecheck $\texttt{logger1.log}(''\texttt{Hello,world}'')$ with T-$\text{MethCall}_\sigma$. All the types match up, so this expression types to `Unit`. The body of `main` matches the signature, so we're good.

### Effect-Checking

We need the $effects$ function and a choice of $\Gamma'$. We choose $\Gamma' = \{\texttt{logger1} : \{\texttt{log} : \texttt{Str}- > \texttt{Unit}\}\}$ , because `logger1` is the only free-variable appearing in the body of `main`.

- $effects(d \texttt{ with } \varepsilon) = \varepsilon$
- $effects(\{\bar{\sigma}\}) = \bigcup_{\sigma \in \bar{\sigma}}\ effects(\sigma)$

By applying the above cases of the $effects$ function we see that:
$\texttt{effects}(\Gamma') = \texttt{effects}(\texttt{logger1}) = \texttt{effects}(\texttt{logger1.log})$.

$\texttt{escapes}(\texttt{def log}(\texttt{entry} : \texttt{String}) : \texttt{Unit}) = \texttt{escapes}(\texttt{Unit})$. `Unit` is equivalent to the structural type $\varnothing$, so the `escapes` function evaluates to $\varnothing$ by a degenerate case of the rule Escapes-Composite.

### Conclusion

Now we've satisfied the antecedents of C-NewObj. We label the new object with the following type:
$\texttt{main} : \texttt{Unit} \to \texttt{Unit captures } \{\texttt{FileIO.append}\}$.

## 3 Example 3

In this example the logger exposes the `FileIO` resource through a method, so anyone who calls that resource will capture every effect on `FileIO`.

```
// $\Gamma_0 = \{FileIO : \{FileIO\}\}$
let logger2 = new
   def log(entry : String) : Unit with FileIO.append
      FileIO.append('/logs/mylog.txt', entry)
   def expose() : { FileIO } with ∅
      FileIO

 // $\Gamma_1 = \{FileIO : \{FileIO\}, logger2 : \{log : String \to Unit,\ \ expose : Unit \to FileIO\}\}$
in new
   def main() : Unit
      logger2.expose().read('/etc/passwd')  // has a read effect that is not captured
```

Similar to example 2. Again we want to apply C-NewObj.

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

### Type-Checking

To type the body of `main` we apply T-MethCall$_\sigma$ to $\texttt{logger2.expose}()$, which types to $\{\texttt{FileIO}\}$. Then we can type $\texttt{logger2.expose}().\texttt{read}(''\texttt{/etc/passwd}'')$ by applying T-MethCall$_r$, which says that it types to $\{r\}$ (because $\{\texttt{r}\} <: \varnothing = \texttt{Unit}$).

### Effect-Checking

Our choice of $\Gamma'$ will be `logger2`, as this is the set of free variables in the body of `main`. We use the following cases of the `effects` function.

- $\texttt{effects}(d\ \texttt{with}\ \varepsilon) = \varepsilon \cup \texttt{escapes}(d)$
- $\texttt{effects}(\{\bar{\sigma}\}) = \bigcup_{\sigma \in \bar{\sigma}}\ \texttt{effects}(\sigma)$

$\texttt{effects}(\texttt{logger2.log}) = \texttt{effects}(\texttt{def log}(\texttt{entry} : \texttt{String}) : \texttt{Unit with FileIO.append})$.

Which is $\{\texttt{FileIO.append}\} \cup \texttt{escapes}(\texttt{def log}(\texttt{entry} : \texttt{String}) : \texttt{Unit})$. Because `Unit` is equivalent to the composite type with no declarations $\varnothing$, the `escapes` function returns $\varnothing$ (a degenerate case of the rule Escapes-Composite).

Now for $\texttt{effects}(\texttt{logger2.expose})$. This is the same as $\texttt{effects}(\texttt{def expose}() : \{\texttt{FileIO}\}\ \texttt{with}\ \varnothing)$. This is $\varnothing \cup escapes\{FileIO\}$. By the rule Escapes-Resource, this is the set of all possible effects on *FileIO*.

### Conclusion

Finally we can apply C-NewObj. The object we created types to the following.

$\{\texttt{main} : \texttt{Unit} \to \texttt{Unit captures}\ \{\texttt{FileIO.append}, \texttt{FileIO.read}, \texttt{FileIO.write}\}\ \texttt{with}\ \varnothing\}$

## 4 Example 4

In this example the resource is exposed by returning an object which has an authority for it.

```
type SigFoo
  def getIO() : { FileIO } with ∅

let logger3 = new
  def log(entry : String) : Unit with FileIO.append
    FileIO.append('/logs/mylog.txt', entry)
  def expose() : SigFoo with ∅
    new
      def getIO() : { FileIO } with ∅
        FileIO

in new
  def main() : Unit
    logger3.expose().getIO().read('/etc/passwd')
```

As in previous examples we want to apply C-NewObj.

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

### Typechecking

First we'll typecheck the body of `main`. $\texttt{logger3} : \{\texttt{log} : \texttt{String} \to \texttt{Unit...}, \texttt{expose} : \texttt{Unit} \to \texttt{SigFoo...}\}$ by the rule T-Var, as $\texttt{logger3} \in \Gamma$.

We apply T-MethCall$_\sigma$ to `logger3.expose()`. The argument is of type `Unit` (need a rule for this?). The return type of `expose` is `SigFoo`, so `logger3.expose() : SigFoo`.

We apply T-MethCall$_\sigma$ to `logger3.expose().getIO()`. This typechecks to $\{FileIO\}$.

We apply T-MethCall$_r$ to `logger3.expose().getIO().read(″/etc/passwd″)`. This typechecks to $\{FileIO\}$, and $\{FileIO\}$ <: `Unit`, so the body matches the signature.

### Effect-Checking

The free variables of `main` is `logger3`, so we choose $\Gamma'$ containing only `logger3`. Here are the relevant cases for the `effects` function.

- $\texttt{effects}(d\ \texttt{with}\ \varepsilon) = \varepsilon \cup \texttt{escapes}(d)$
- $\texttt{effects}(\{\bar{\sigma}\}) = \bigcup_{\sigma \in \bar{\sigma}}\ \texttt{effects}(\sigma)$

$\texttt{effects}(\texttt{logger3}) = \texttt{effects}(\texttt{logger3.log}) \cup \texttt{effects}(\texttt{logger3.expose})$

First, $\texttt{effects}(\texttt{logger3.log}) = \{\texttt{FileIO.append}\} \cup \texttt{escapes}(\texttt{def log(entry : String) : Unit})$. By Escapes-Composite, `escapes` returns ∅ here.

Second, $\texttt{effects}(\texttt{logger3.expose}) = \varnothing \cup \texttt{escapes}(\texttt{def expose() : SigFoo})$. `SigFoo` is a structural type so we apply Escapes-Structural. Then $\texttt{escapes}(\texttt{logger3.expose}) = \texttt{escapes}(\texttt{SigFoo})$.

`SigFoo` has the form $\{\bar{\sigma}\}$, so we apply Escapes$_{\bar{\sigma}}$ The only declaration is `getIO`, which has the form $\sigma$, so we apply Escapes$_\sigma$. This gives us $\texttt{escapes}(\texttt{SigFoo}) = \texttt{escapes}(\texttt{def getIO() : \{FileIO\}})$.

Finally we can apply Escapes-Resource. We have `escapes(SigFoo)` is the set of all effects on `FileIO`. Then `effects(logger3)` is the set of all effects on `FileIO`.

### Conclusion

Now we know $\texttt{effects}(\Gamma')$ to be the set of all effects on `FileIO`, we may finally apply C-NewObj. We conclude that the new object types to:

$$\{\texttt{main} : \texttt{Unit} \to \texttt{Unit captures } \{\texttt{FileIO.read, FileIO.write, FileIO.append}\}\}\ \texttt{with}\ \varnothing$$

## 5 Example 5

This is an example with parametricity.

```
// $\Gamma_0 = \{\{FileIO\}\}$
type SigPasswordReader
  def readPasswords(fileio : { FileIO }) : String with FileIO.read
let passwordReader = new
  def readPasswords(fileio : { FileIO }) : String with FileIO.read
    fileio. read('/etc/passwd')
in
  let logger4 = new
    def log(entry : String) : Unit with FileIO.append
      FileIO.append('/log/mylog.txt', entry)
    def enablePasswordReading(pr : SigPasswordReader) : Unit
      pr.readPasswords(FileIO)
  in new
    def main() : Unit
      logger4.enablePasswordReading(passwordReader)
/* This example also illustrates parametricity: passwordReader accepts any resources of type { FileIO } */
```

Want to apply C-NewObj.

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

### Type-Checking

Same as previous sections.

### Effect-Checking

The body of the new expression contains the free variables `logger4` and `passwordReader`, so we choose $\Gamma'$ containing those objects.

Then $\texttt{effects}(\texttt{Gamma}') = \texttt{effects}(\texttt{logger3}) \cup \texttt{effects}(\texttt{passwordReader})$.

First, $\texttt{effects}(\texttt{logger4}) = \texttt{effects}(\texttt{logger4.log}) \cup \texttt{effects}(\texttt{logger4.enablePasswordReading})$.

Then $\texttt{effects}(\texttt{logger4.log} = \{\texttt{FileIO.append}\} \cup \texttt{escapes}(\texttt{def log}(\texttt{entry} : \texttt{String}) : \texttt{Unit}$.

This expands to $\{\texttt{FileIO.append}\} \cup \varnothing = \{\texttt{FileIO.append}\}$ by applying Escapes-Structural.

Second, $\texttt{effects}(\texttt{passwordReader}) = \texttt{effects}(\texttt{readPasswords})$.

Which expands to $\{\texttt{FileIO.read}\} \cup \texttt{escapes}(\texttt{def readPasswords}(\texttt{fileio} : \{\texttt{FileIO}\}))$.

We apply Escapes-Resource. The set of effects escaping is everything on `FileIO`. Therefore $\texttt{effects}(\Gamma')$ is everything on `FileIO`.

### Conclusion

We apply C-NewObj. The object created has the following type.

$\{\texttt{main} : \texttt{Unit} \to \texttt{Unit captures}\ \{\texttt{FileIO.read}, \texttt{FileIO.append}, \texttt{FIleIO.write}\}\}\ \texttt{with}\ \varnothing$

## 6 Example 6

This has partially-labeled declarations.

```
// Γ0 = {{FileIO}}
let logger2 = new
   def log(entry : String) : Unit with FileIO.append
      FileIO.append('/logs/mylog.txt', entry)
   def expose() : { FileIO }
      FileIO

 // Γ1 = {{FileIO}, logger2}
in new
   def main() : Unit
      logger2.expose().read('/etc/passwd')  // has a read effect that is not captured
```