1 Darya's Example 1

```
// Γ_0 = {{FileIO}}
let logger1 = new
    def log(entry : string) : Unit with FileIO.append
       FileIO.append('/logs/mylog.txt', entry)

 // Γ_1 = {{FileIO}, logger1}
in new
   def main() : Unit
      logger1.log('Hello, World!')
```

Start with $\Gamma_0\{\{FileIO\}\}$. After execution of line 2, we obtain $\Gamma_1 = \{\{FileIO\},\ logger1\}$. Line 7 declares an unannotated object type so we want to apply C-NewObj.

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

We show the antecedents hold. Firstly $effects(\Gamma_1) = effects(\{FileIO\}) = \{(r, m) \mid r \in \bar{r}, m \in M\}$. By expanding this out we get $effects(\Gamma_1) = \{(FileIO, FileIO.append), (FileIO, FileIO.read), (FileIO, FileIO.write), (FileIO, logger1.log)\}$.

Secondly the body of the new object is well-typed (according to standard typing rules). Apply ε-ValidImpl$_d$. Then we prove $\overline{d = e}$ OK (for the body of the main object).

Apply C-NewObj. Then $\Gamma_1 \vdash new\ x \Rightarrow \overline{d = e} : \{\texttt{def main() : Unit captures}\ \varepsilon\}$, where $\varepsilon = \{(FileIO, FileIO.append), (FileIO, FileIO.read), (FileIO, FileIO.write), (FileIO, logger1.log)\}$.

2 Darya's Example 2

```
// Γ_0 = {{FileIO}}
let logger2 = new
   def log(entry : String) : Unit with FileIO.append
      FileIO.append('/logs/mylog.txt', entry)
   def expose() : { FileIO } with ∅
      FileIO

 // Γ_1 = {{FileIO}, logger2}
in new
   def main() : Unit
      logger2.expose().read('/etc/passwd')  // has a read effect that is not captured
```

Very similar to example 1 but the set of effects computed is different. $effects(\Gamma_1) = effects(\{FileIO\}) \cup effects(logger2)$. $effects(\{FileIO\}) = \{(FileIO, FileIO.write), (FileIO, FileIO.read), (FileIO, FileIO.write), (FileIO, logger2.log), (FileIO, logger2.expose)\}$.

$logger2$ matches $\{\bar{\sigma}\}$ so we take the union over $effects(\sigma)$, for $\sigma \in \bar{\sigma}$. This is $effects(logger2.log) \cup effects(logger2.expose) = \{(FileIO, FileIO.append)\}$.

$effects(logger2) \subseteq effects(\{FileIO\})$, so $effects(\Gamma_1) = effects(\{FileIO\})$.

Then by the same process as before we conclude that $new\ x \Rightarrow \overline{d = e} : \{\texttt{def main() : Unit captures}\ \varepsilon\}$, where $\varepsilon = \{(FileIO, FileIO.write), (FileIO, FileIO.read), (FileIO, FileIO.write), (FileIO, logger2.log), (FileIO, logger2.expose)\}$.

3 Darya's Example 3

```
// $\Gamma_0 = \{\{FileIO\}\}$
let logger3 = new
  def log(entry : String) : Unit with FileIO.append
    FileIO.append('/logs/mylog.txt', entry)
  def createExpose() : SigFoo with ∅
    new
      def getIO() : { FileIO } with ∅
        FileIO
in new
  def main() : Unit
    logger3.createExpose().io().read('/etc/passwd')
```

4 Darya's Example 4

```
// $\Gamma_0 = \{\{FileIO\}\}$
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 } */
```