1 Well-Behaved and Annotated Logger

This program instantiates a well-behaved logger and uses it to append some text to a file. In this example the only resource is `FileIO` and the only operation is `append`. Our initial Γ should contain `FileIO` : {`FileIO`}.

```
let logger = new
   def log(x : Str) : Unit with FileIO.append
        FileIO.log(x)

in logger.log(``hello, world!'')
```

This program desugars as below. Let's note a few things about it:

- We assume there is some notion of a `Str` and that any Γ can type a string literal to `Str with` $\varnothing$.
- Because our system does not model arguments to operations, the desugared version calls `FileIO.append` instead of `FileIO.append`("`hello, world!`")
- A function called `m` that takes $\mathtt{t}_1$ as input and outputs something of type $\mathtt{t}_2$ is said to have the type $\mathtt{m} : \mathtt{t}_1 \to \mathtt{t}_2$.

```
new_σ x ⇒ {
   def m(y : { log : Str -> Unit with FileIO.append }) : Unit with FileIO.append = y.log(``hello, world!'')
}.m(new_σ x ⇒ { def log(x : Str) : Unit with FileIO.append = FileIO.append })
```

To typecheck we match the form of the program to the conclusion of ε-MethCall. To apply that rule we must first type the sub-expressions.

Typing Outermost Receiver

The expression being typed is:
$\mathtt{new}_\sigma\ \mathtt{x} \Rightarrow \{$
$\quad \mathtt{def\ m(y : \{log : Str \to Unit\ with\ FileIO.append\})} : \mathtt{Unit\ with\ FileIO.append}\ =$
$\quad\quad \mathtt{y.log(\text{“}hello, world!''})$
$\}$

This is an object so we want to apply ε-NewObj. To do that we need to show $\sigma_i = e_i$ `OK` by using ε-ValidImpl$_\sigma$. In this case there's only one method, `m`.

If we take the environment $\Gamma, \mathtt{y} : \{\mathtt{log} : ....\})$ then y has a `log` method which takes a `Str` as an argument. $\Gamma \vdash$ "`hello, world!`'' : `Str with` $\varnothing$ by assumption. The declared return type of `log` is `Unit` and its set of effets of `FileIO.append`. So now using ε-MethCall, we know $\Gamma \vdash \mathtt{y.log(\text{“}hello, world!''})$: `Unit with` $\varnothing \cup \varnothing \cup$ `FileIO.append`.

Now that we have typed the body of the method `m` i.e. $\Gamma \vdash \mathtt{y.log(\text{“}hello, world!''})$: `Unit with FileIO.append`, we can see this aligns with the declared type/effects of `m`. By an application of ε-ValidImpl$_\sigma$, we conclude $\Gamma \vdash$ `def m....` $= \mathtt{y.log(\text{“}hello, world!''})$ `OK`. This is the only method in the object so $\Gamma \vdash \overline{\sigma = \mathtt{e}}$ `OK`.

Finally by applying ε-NewObj we may conclude:
$\Gamma \vdash \mathtt{new}_\sigma \mathtt{x}\ \Rightarrow \{\mathtt{def\ m... = y.log(\text{“}hello, world!''})\} : \{\mathtt{m} : \{\mathtt{log} : ...\} \to \mathtt{Unit}\}$ `with` $\varnothing$

Typing Outermost Argument

The expression being typed is: $\mathtt{new}_\sigma\ \mathtt{x} \Rightarrow \{\mathtt{def\ log(x : str)} : \mathtt{Unit\ with\ FileIO.append} = \mathtt{FileIO.append}\}$

This is an object so we want to apply ε-NewObj. To do that we need to show $\sigma_i = e_i$ `OK` by using ε-ValidImpl$_\sigma$. In this case there's only one method `log`.

The body of `log` is `FileIO.append`. The receiver is `FileIO` which is a resource in Γ, so $\Gamma \vdash$ `FileIO` : {`FileIO`}. By using ε-OperCall we can type the body $\Gamma \vdash$ `FileIO.append` : `Unit with FileIO.append`.

The declared type of `log` is $\texttt{log} : \texttt{Str} \to \texttt{Unit with FileIO.append}$. We can see the return type and effect-set matches the return-type and effect-set of the body. Therefore $\Gamma \vdash \sigma = \texttt{e OK}$ for this method. This is the only method, so $\Gamma \vdash \overline{\sigma = \texttt{e}}\ \texttt{OK}$.

Finally by applying ε-NewObj we may conclude:
$\Gamma \vdash \texttt{new}_\sigma\ \texttt{x} \Rightarrow \{\texttt{def log.... = FileIO.append}\} : \{\texttt{log} : \texttt{Str} \to \texttt{Unit with FileIO.append}\}\ \texttt{with}\ \varnothing$

Typing Outermost Method Call

Since we've typed the receiver and the argument, we can see that the receiver has a method called `m`. `m` has a formal parameter type $\{\texttt{log} : \texttt{Str} \to \texttt{Unit with FileIO.append}\}$, which is the same as our actual argument.

Let `e` refer to the whole program. By an application of ε-MethCall we may conclude:
$\Gamma \vdash \texttt{e} : \texttt{Unit with}\ \varnothing \cup \varnothing \cup \texttt{FileIO.append}$