1 Conservative Example

1.1 Sugared

In this example, calls to a particular class of functions are first routed through a logging function `loggerProxy` which records their use. For simplicity we just assume that the user wants to apply 0 to every function (practically you'd want to have `loggerProxy` take an extra argument for the input to `f`, but this complicates desugaring and isn't essential to the point being made).

Note that although `id` does not have a capability for `File.log`, and `loggerProxy` does, the implementation of `loggerProxy` is being sensible (so it is type-and-effect safe).

```
def id(a: Int): Int with ∅ =
    a

def loggerProxy(f: Int → Int with ∅): Unit with File.log =
    File.log(``Log: called function '' + f)
    id(0)

def main(): Unit =
   proxy(id)
```

1.2 Desugared

A multi-variable function is desugared into an object with a method which returns an object with a method.

```
let x1 = newσx ⇒ {
    def id(a: Int): Int with ∅ =
       a
} in

let x2 = newσx ⇒ {
    def fix(obj: {id: Int → Int with ∅}): Unit with Sock.read =
        newσx ⇒ {
            def m(): Unit with ∅ =
               obj.id(0)
        }.m(File.log)
} in

let x3 = newdx ⇒ {
    def main(): Unit =
       x2.fix(x1)
} in

x3.main()
```

1.3 Typing

To type x_3 using C-NewObj you need a Γ' containing x_2 and x_1. $\texttt{capture}(x_2) = File.log$ and $\texttt{capture}(x_1) = \varnothing$. So $\varepsilon_c = \texttt{capture}(\Gamma') = \texttt{File.log}$. To meet the premises of C-NewObj we need $\texttt{capture}(\tau) \supset File.log$, for every τ which is the argument of a higher-order function in scope.

The only higher-order function in scope is `x2.fix`. Its formal parameter has the type `id : Int → Int with ∅`. The capture of this type is $\varnothing$ and $\varnothing \not\supset$ `File.log`, so this program is rejected.

2 arg-types

2.1 `arg-types` Function

This function examines the declaration of every method which could be (directly) invoked inside a particular Γ. It returns a set of the types of the arguments of those methods.

- $\texttt{arg-types}(\varnothing) = \varnothing$
- $\texttt{arg-types}(\Gamma, x : \tau) = \texttt{arg-types}(\Gamma) \cup \texttt{arg-types}(\tau)$
- $\texttt{arg-types}(\{r\}) = \varnothing$
- $\texttt{arg-types}(\{\bar{\sigma}\}) = \bigcup_{\sigma \in \bar{\sigma}} \texttt{arg-types}(\sigma)$
- $\texttt{arg-types}(\{\bar{d}\}) = \bigcup_{d \in \bar{d}} \texttt{arg-types}(d)$
- $\texttt{arg-types}(\{\bar{d} \texttt{ captures } \varepsilon_c\}) = \texttt{arg-types}(\{\bar{d}\})$
- $\texttt{arg-types}(d \texttt{ with } \varepsilon) = \texttt{arg-types}(d)$
- $\texttt{arg-types}(\texttt{def } m(y : \tau_2) : \tau_3) = \{\tau_2\} \cup \texttt{arg-types}(\tau_3) \cup \texttt{arg-types}(\tau_2)$ (is $\texttt{arg-types}(\tau_2)$ necessary?)

2.2 `higher-order-args` Function

$$\frac{\tau \in \texttt{arg-types}(\Gamma) \quad \texttt{is-higher-order}(\tau)}{\tau \in \texttt{higher-order-args}(\Gamma)} \text{ (HigherOrderArgs)}$$

2.3 `is-obj` Predicate

The `is-obj` predicate says whether or not a particular type τ is an object.

$$\frac{}{\texttt{is-obj}(\{\bar{d}\})} \text{ (IsObj}_d\text{)} \qquad \frac{}{\texttt{is-obj}(\{\bar{\sigma}\})} \text{ (IsObj}_\sigma\text{)} \qquad \frac{}{\texttt{is-obj}(\{\bar{d} \texttt{ captures } \varepsilon_c\})} \text{ (IsObjSummary)}$$

2.4 `is-higher-order` Predicate

A type is higher-order if it has a method accepting another object as an argument.

$$\frac{d_i = \texttt{def } m(y : \tau_2) : \tau_3 \quad \texttt{is-obj}(\tau_2)}{\texttt{is-higher-order}(\{\bar{d}\})} \text{ (HigherOrder}_d\text{)}$$

$$\frac{\sigma_i = \texttt{def } m(y : \tau_2) : \tau_3 \texttt{ with } \varepsilon \quad \texttt{is-obj}(\tau_2)}{\texttt{is-higher-order}(\{\bar{\sigma}\})} \text{ (HigherOrder}_\sigma\text{)}$$