1 Examples

1.1 Safe Logger

```
1 import(File.append)
2    log = λx:Unit . File.append
3    in log unit
```

Firstly $\vdash \lambda x : \texttt{Unit} \,.\, \texttt{File.append} : \texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit with} \varnothing$ by ε-Abs. Call this type $\hat{\tau}$. $\texttt{effects}(\hat{\tau}) = \{\texttt{File.append}\}$ and its erasure is $\texttt{Unit} \to \texttt{Unit}$. Now $\texttt{log} : \texttt{Unit} \to \texttt{Unit} \vdash \texttt{log unit} : \texttt{Unit}$ by T-Abs.

By definition $\texttt{ho-safe}(\texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}, \{\texttt{File.append}\})$ iff $\texttt{safe}(\texttt{Unit}, \{\texttt{File.append}\})$ and $\texttt{ho-safe}(\texttt{Unit}, \{\texttt{File.append}\})$. The two conjuncts are true by Safe-Unit and HOSafe-unit.

1.2 Logger Uses Undefined Capability

```
1 import(File.append)
2    log = λx:Unit . File.write
3    in log unit
```

Firstly $\vdash \lambda \texttt{x} : \texttt{Unit}. \texttt{File.write} : \texttt{Unit} \to_{\texttt{File.write}} \texttt{Unit with} \varnothing$ by ε-Abs. Now:

$$
\begin{aligned}
&\texttt{effects}(\texttt{Unit} \to_{\texttt{File.write}} \texttt{Unit}) \\
&= \{\texttt{File.write}\} \cup \texttt{effects}(\texttt{Unit}) \cup \texttt{ho-effects}(\texttt{Unit}) \\
&= \{\texttt{File.write}\}
\end{aligned}
$$

But $\{\texttt{File.write}\} \neq \{\texttt{File.append}\}$, the set of capabilities declared by the module. Hence this program doesn't typecheck.

1.3 Higher-Order Safe

```
1 import(File.append)
2    getLogger = λx:Unit. (λy:Unit. File.append)
3 in (getLogger unit) unit
```

Firstly, $\texttt{x} : \texttt{Unit} \vdash \lambda \texttt{y} : \texttt{Unit}. \texttt{File.append} : \texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit with} \varnothing$ by ε-Abs. Then by ε-Abs again, $\vdash \lambda \texttt{x} : \texttt{Unit}. (\lambda \texttt{y} : \texttt{Unit}. \texttt{File.append}) : \texttt{Unit} \to_{\varnothing} \texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit with} \varnothing$. This is our $\hat{\tau}$.

The set of effects declared by the module is $\varepsilon = \{\texttt{File.append}\}$. This needs to be the same as $\texttt{effects}(\hat{\tau})$.

$$
\begin{aligned}
&\texttt{effects}(\hat{\tau}) \\
&= \texttt{effects}(\texttt{Unit} \to_{\varnothing} \texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}) \\
&= \texttt{ho-effects}(\texttt{Unit}) \cup \varnothing \cup \texttt{effects}(\texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}) \\
&= \texttt{effects}(\texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}) \\
&= \texttt{ho-effects}(\texttt{Unit}) \cup \{\texttt{File.append}\} \cup \texttt{effects}(\texttt{Unit}) \\
&= \{\texttt{File.append}\}
\end{aligned}
$$

We also need higher-order safety.

$$
\begin{aligned}
&\texttt{ho-safe}(\texttt{Unit} \to_{\varnothing} \texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}, \{\texttt{File.append}\}) \\
&\equiv \texttt{safe}(\texttt{Unit}, \{\texttt{File.append}\}) \wedge \texttt{ho-safe}(\texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}, \{\texttt{File.append}\}) \\
&\equiv \texttt{ho-safe}(\texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}, \{\texttt{File.append}\}) \\
&\equiv \texttt{safe}(\texttt{Unit}, \{\texttt{File.append}\}) \wedge \texttt{ho-safe}(\texttt{Unit}, \{\texttt{File.append}\}) \\
&\equiv \texttt{True}
\end{aligned}
$$

Lastly, $\texttt{erase}(\texttt{Unit} \to_{\varnothing} \texttt{Unit} \to_{\texttt{File.append}} \texttt{Unit}) = \texttt{Unit} \to \texttt{Unit} \to \texttt{Unit}$. By using T-App twice we have $\texttt{getLogger} : \texttt{Unit} \to \texttt{Unit} \to \texttt{Unit} \vdash (\texttt{getLogger unit})\ \texttt{unit} : \texttt{Unit}$.

1.4 Higher-Order Unsafe

In this example the module leaks a capability for appending to a file, violating its signature.

```
import(∅)
   getLogger = λx:Unit . (λy:Unit . File.append)
in (getLogger unit) unit
```

By ε-Abs, $\vdash \lambda \mathtt{x} : \mathtt{Unit}.\ (\lambda \mathtt{y} : \mathtt{Unit}.\ \mathtt{File.append}) : \mathtt{Unit} \to_{\varnothing} \mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}\ \mathtt{with}\ \varnothing$. This is our $\hat{\tau}$. The set of effects declared by the module is $\varnothing$. This needs to be the same as $\mathtt{effects}(\hat{\tau})$.

$\mathtt{effects}(\hat{\tau})$
$= \mathtt{effects}(\mathtt{Unit} \to_{\varnothing} \mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit})$
$= \mathtt{ho\text{-}effects}(\mathtt{Unit}) \cup \varnothing \cup \mathtt{effects}(\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit})$
$= \mathtt{effects}(\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit})$
$= \mathtt{ho\text{-}effects}(\mathtt{Unit}) \cup \{\mathtt{File.append}\} \cup \mathtt{effects}(\mathtt{Unit})$
$= \{\mathtt{File.append}\} \neq \varnothing$

So the example fails to typecheck.

1.5 Higher-Order Unsafe 2

In this example we pass in a function which writes and appends to a file. However, the signature expects it to only be appending.

```
import(File.append)
    logger = λf:Unit →File.append Unit. f unit
in logger (λx:Unit. let y = File.append in File.write)
```

It desugars into this program.

```
import(File.append)
    logger = λf:Unit →File.append Unit. f unit
in logger (λx:Unit. (λy:Unit. File.append) File.write)
```

By ε-App we have $\mathtt{f} : \mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit} \vdash \mathtt{f\ unit} : \mathtt{Unit}\ \mathtt{with}\ \{\mathtt{File.append}\}$. Then by ε-Abs we have $\vdash \mathtt{logger} : (\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}) \to_{\mathtt{varnothing}} \mathtt{Unit}\ \mathtt{with}\ \varnothing$. This is our $\hat{\tau}$.

The set of effects declared by this module is $\{\mathtt{File.append}\}$. We need this to be the same as $\mathtt{effects}(\hat{\tau})$. By definition,

$\mathtt{effects}((\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}) \to_{\mathtt{varnothing}} \mathtt{Unit})$
$= \mathtt{ho\text{-}effects}(\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}) \cup \varnothing \cup \mathtt{effects}(\mathtt{Unit})$
$= \mathtt{ho\text{-}effects}(\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit})$
$= \mathtt{effects}(\mathtt{Unit}) \cup \mathtt{ho\text{-}effects}(\mathtt{Unit})$
$= \varnothing \subseteq \{\mathtt{File.append}\}$

Here is the derivation of higher-order safety.
$\mathtt{ho\text{-}safe}((\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}) \to_{\mathtt{varnothing}} \mathtt{Unit}, \{\mathtt{File.append}\})$
$\equiv \mathtt{safe}(\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}, \{\mathtt{File.append}\}) \wedge \mathtt{ho\text{-}safe}(\mathtt{Unit}, \{\mathtt{File.append}\})$
$\equiv \mathtt{safe}(\mathtt{Unit} \to_{\mathtt{File.append}} \mathtt{Unit}, \{\mathtt{File.append}\})$
$\equiv \{\mathtt{File.write}\} \subseteq \{\mathtt{File.write}\} \wedge \mathtt{safe}(\mathtt{Unit}, \{\mathtt{File.write}\}) \wedge \mathtt{ho\text{-}safe}(\mathtt{Unit}, \{\mathtt{File.write}\})$
$\equiv \mathtt{True}$

Now by T-Abs we have $\mathtt{logger} : \mathtt{Unit} \to \mathtt{Unit}, \mathtt{x} : \mathtt{Unit} \vdash \lambda \mathtt{y} : \mathtt{Unit}.\ \mathtt{File.append} : \mathtt{Unit} \to \mathtt{Unit}$. By T-Abs again we have $\mathtt{logger} : \mathtt{Unit} \to \mathtt{Unit} \vdash \lambda \mathtt{x} : \mathtt{Unit}.(\lambda \mathtt{y} : \mathtt{Unit}.\ \mathtt{File.write}) : \mathtt{Unit} \to \mathtt{Unit} \to \mathtt{Unit}$.

So this example typechecks (but it shouldn't)

1.6 Higher-Order Unsafe 3

In this example we pass in a function which returns a function which writes to a file (not allowed).

```
1 import(∅)
2     logger = λf:Unit →∅ Unit →∅ Unit. f unit unit
3 in logger (λx:Unit. (λy:Unit. File.write))
```

Firstly by ε-App we have $\texttt{f} : \texttt{Unit} \to_\varnothing \texttt{Unit} \to_\varnothing \texttt{Unit} \vdash \texttt{f unit unit} : \texttt{Unit with} \varnothing$. Then by ε-Abs we have $\vdash \texttt{logger} : \texttt{Unit} \to_\varnothing \texttt{Unit} \to_\varnothing \texttt{Unit} \to_\varnothing \texttt{Unit}$. This is our $\hat{\tau}$.

Secondly $\texttt{effects}(\hat{\tau}) = \varnothing$ which is the same as the set of effects declared by this module. This is also trivially higher-order safe.

Now $\texttt{erase}(\texttt{Unit} \to_\varnothing \texttt{Unit} \to_\varnothing \texttt{Unit}) = \texttt{Unit} \to \texttt{Unit} \to \texttt{Unit}$ and by using T-App and T-Abs we have $\texttt{logger} : \texttt{Unit} \to \texttt{Unit} \to \texttt{Unit} \vdash \texttt{logger}(\lambda \texttt{x} : \texttt{Unit}.(\lambda \texttt{y} : \texttt{Unit}.\texttt{File.write})) : \texttt{Unit}$.

So this example typechecks (but it shouldn't)