Unifying Algebraic with Abstract Effects

November 19, 2020

1 Surface language

1.1 syntax

$$
\begin{array}{lcll|lcll}
 & & & & \varepsilon & ::= & \overline{x.E} & \textit{effects} \\
e & ::= & x & \textit{expressions} & \tau & ::= & \{x \Rightarrow \overline{\sigma}\} & \textit{type} \\
 & \mid & \texttt{new}(x \Rightarrow \overline{d}) & & \sigma & ::= & \texttt{def}\ m(x:\tau) : \{\varepsilon\}\ \tau & \textit{decl. types} \\
 & \mid & e.m(e) & & & \mid & \texttt{effect}\ E = \overline{\texttt{o}} & \\
 & \mid & x.op(e) & & & \mid & \texttt{effect}\ E = \varepsilon & \\
 & \mid & handle\{h\}e & & & \mid & \texttt{effect}\ E = \varepsilon & \\
d & ::= & \texttt{def}\ m(x:\tau) : \varepsilon\ \tau = e & \textit{declarations} & \texttt{o} & ::= & \texttt{def}\ op(x:\tau) : \tau & \textit{operation} \\
 & \mid & \texttt{effect}\ E = \overline{\texttt{o}} & & \texttt{h} & ::= & return\ x \rightarrow e & \textit{handler} \\
 & \mid & \texttt{effect}\ E = \varepsilon & & & \mid & x.op(x) \rightarrow e; h &
\end{array}
$$

Figure 1: Syntax of surface language

The expression of the surface language contains variables, two basic object-oriented expressions: the `new` statement and the method call, the operation call `x.op(e)`, and handled expression `handle{h}e`. Objects are created by `new` statements that contain a variable x representing the current object along with a list of declarations. Declarations come in three kinds: a method declaration, a effect member with operations, and an effect member with subeffects. Method declarations are annotated with a set of effects.

Object types are a collection of declaration types, which include method signatures and the types of effect-member declarations and definitions. Similar to the difference between the modules and their types, effects in an object must always be defined (i.e., has operations or subeffects), whereas effects in object types may or may not have definitions (i.e., be either abstract or concrete).

The signature of effect is a list of operations. Each operation has an input type and an output type. The handler `h` must contain a `return` clause.

1.2 Typing Rules

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \{\}\ \tau}\ (\text{T-Var}) \qquad \frac{\forall i,\ d_i \in \overline{d},\ \sigma_i \in \overline{\sigma},\ \Gamma,\ x : \{x \Rightarrow \overline{\sigma}\} \vdash d_i : \sigma_i}{\Gamma \vdash \texttt{new}(x \Rightarrow \overline{d}) : \{\}\ \{x \Rightarrow \overline{\sigma}\}}\ (\text{T-New})$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : \{\varepsilon_1\}\{x \Rightarrow \overline{\sigma}\} \quad \texttt{def}\ m(y : \tau_2) : \{\varepsilon_3\}\ \tau_1 \in \overline{\sigma} \\ \Gamma \vdash [e_1/x][e_2/y]\varepsilon_3\ wf \quad \Gamma \vdash e_2 : \{\varepsilon_2\}\ [e_1/x]\tau_2 \quad \varepsilon = \varepsilon_1 \cup \varepsilon_2 \cup [e_1/x][e_2/y]\varepsilon_3\end{array}}{\Gamma \vdash e_1.m(e_2) : \{\varepsilon\}\ [e_1/x][e_2/y]\tau_1}\ (\text{T-Method})$$

$$\frac{\Gamma \vdash e : \{\varepsilon_1\}\ \tau_1 \quad \Gamma \vdash \tau_1 <: \tau_2 \quad \Gamma \vdash \varepsilon_1 <: \varepsilon_2}{\Gamma \vdash e : \{\varepsilon_2\}\ \tau_2}\ (\text{T-Sub})$$

$$\frac{\Gamma \vdash x : \{\varepsilon\}\{y \Rightarrow \overline{\sigma}\} \quad \texttt{effect}\ E = \overline{o} \in \overline{\sigma} \quad \texttt{def}\ op(z : \tau_1) : \tau_2 \in \overline{o} \quad \Gamma \vdash e : \{\varepsilon\}\tau_1}{\Gamma \vdash x.op(e) : \{x.E, \varepsilon\}\tau_2}\ (\text{T-Op})$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \{x.E, \varepsilon\}\ \tau \quad \Gamma \vdash findop(x.E) = \{x_1.op_1(y_1), \ldots, x_n.op_n(y_n)\} \quad \Gamma \vdash x_i.op_i(y_i) : \tau_i \to \tau_i' \\ \Gamma, resume : \{\_ \Rightarrow \texttt{def}\ m(x : \tau_i') : \{\varepsilon\}\ \tau_r\}, y_i : \tau_i \vdash e_i : \{\varepsilon\}\ \tau_r \quad \Gamma, z : \tau \vdash e_r : \{\varepsilon\}\ \tau_r\end{array}}{\Gamma \vdash \texttt{handle}\ \{x_1.op_1(y_1) \to e_1, ..., x_n.op_n(y_n) \to e_n, return\ z \to e_r\}e : \{\varepsilon\}\ \tau_r}\ (\text{T-Handle})$$

$$\boxed{\Gamma \vdash d : \sigma}$$

$$\frac{\Gamma,\ x : \tau_1 \vdash e : \{\varepsilon\} \quad \Gamma, x : \tau_1 \vdash \varepsilon wf}{\Gamma \vdash \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2 = e\ :\ \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2}\ (\text{DT-Def})$$

$$\frac{}{\Gamma \vdash \texttt{effect}\ g = \overline{o}\ :\ \texttt{effect}\ g = \overline{o}}\ (\text{DT-Effect-1}) \qquad \frac{\Gamma \vdash \varepsilon\ wf}{\Gamma \vdash \texttt{effect}\ g = \{\varepsilon\}\ :\ \texttt{effect}\ g = \{\varepsilon\}}\ (\text{DT-Effect-2})$$

Figure 2: type system

Rules T-Var, T-New, T-Method, and T-Sub are standard object oriented typing judgments. In rule T-OP, the type of the operation call is given by its signature, which is defined in its parent object. In rule T-Handle, the effect `x.E` is handled if the handler handles every operation in `x.E`. The expression in each handler clause may contain a `resume` function which serves as the continuation of the operation. The `findop` function is defined in Fig. 12.

1.3 Subtyping

$\boxed{\Gamma \vdash \tau <: \tau'}$

$$\frac{}{\Gamma \vdash \tau <: \tau} \ (\text{S-Refl1}) \qquad \frac{\Gamma \vdash \tau_1 <: \tau_2 \quad \Gamma \vdash \tau_2 <: \tau_3}{\Gamma \vdash \tau_1 <: \tau_3} \ (\text{S-Trans})$$

$$\frac{\{x \Rightarrow \sigma_i^{i \in 1..n}\} \ \textit{is a permutation of} \ \{x \Rightarrow \sigma_i'^{i \in 1..n}\}}{\Gamma \vdash \{x \Rightarrow \sigma_i^{i \in 1..n}\} <: \{x \Rightarrow \sigma_i'^{i \in 1..n}\}} \ (\text{S-Perm})$$

$$\frac{}{\Gamma \vdash \{x \Rightarrow \sigma_i^{i \in 1..n+k}\} <: \{x \Rightarrow \sigma_i^{i \in 1..n}\}} \ (\text{S-Width}) \quad \frac{\forall i, \ \Gamma, \ x : \{x \Rightarrow \sigma_i^{i \in 1..n}\} \vdash \sigma_i <: \sigma_i'}{\Gamma \vdash \{x \Rightarrow \sigma_i^{i \in 1..n}\} <: \{x \Rightarrow \sigma_i'^{i \in 1..n}\}} \ (\text{S-Depth})$$

$\boxed{\Gamma \vdash \sigma <: \sigma'}$

$$\frac{}{\Gamma \vdash \sigma <: \sigma} \ (\text{S-Refl2}) \quad \frac{\Gamma \vdash \tau_1' <: \tau_1 \quad \Gamma \vdash \tau_2 <: \tau_2' \quad \Gamma, x : \tau_1 \vdash \varepsilon_1 <: \varepsilon_2}{\Gamma \vdash \texttt{def} \ m(x : \tau_1) : \{\varepsilon_1\} \ \tau_2 <: \texttt{def} \ m(x : \tau_1') : \{\varepsilon_2\} \ \tau_2'} \ (\text{S-Def})$$

$$\frac{}{\Gamma \vdash \texttt{effect} \ g = \{\varepsilon\} <: \texttt{effect} \ g} \ (\text{S-Effect})$$

$\boxed{\Gamma \vdash \varepsilon <: \varepsilon'}$

$$\frac{\varepsilon_1 \subseteq \varepsilon_2}{\Gamma \vdash \varepsilon_1 <: \varepsilon_2} \ (\text{SE-Refl}) \quad \frac{\Gamma \vdash x : \{y \Rightarrow \overline{\sigma}\} \quad \texttt{effect} \ E = \varepsilon'' \in \overline{\sigma} \quad \varepsilon \cup \varepsilon'' <: \varepsilon'}{\Gamma \vdash \varepsilon \cup \{x.E\} <: \varepsilon'} \ (\text{SE-Left})$$

$$\frac{\Gamma \vdash x : \{y \Rightarrow \overline{\sigma}\} \quad \texttt{effect} \ E = \varepsilon'' \in \overline{\sigma} \quad \varepsilon <: \varepsilon' \cup \varepsilon''}{\Gamma \vdash \varepsilon <: \varepsilon' \cup \{x.E\}} \ (\text{SE-Right})$$

Figure 3: Subtyping Rules

The subtyping rules for object types are standard. in rule S-Effect , the effect set declaration is a subtype of an abstract effect declaration. The subtyping rules for effect sets (SE-Refl, SE-Left, SE-Right) states that an effect set is a subeffect of another if it is a subset, or becomes a subeffect by replacing an element in either effect sets according to declarations.

1.4 Abstraction problem

```
type A
  effect E {
    op1() : Unit
  }

module a: A
  effect E {
    op1(): Unit
  }

type B
  effect E
  def handler(Unit -> {this.E} T) : {} T

module b: B
  effect E = {a.E}
  def m() : {this.E} Unit
    a.op1()
  def handler(c: Unit -> {this.E} T) : {} T =
    handle c() with
    | a.op1() -> resume ()

(b >> B).handler(
  () => handle (b >> B).m() with
          | a.op1() -> ()
 )
```

In the last expression, the method call `b.m()` handled by two nested handlers. Since the effect of `b.m` is abstract, the inner handler should not handle the expression because it handles the concrete effect `a.E`. Instead, the operation in `b.m()` should be handled by the outer handler.

In order to solve this problem, we introduce the language with the `wrap` and `unwrap` constructs to ensure the correctness of dynamic semantics.

1.5 Abstraction Problem II

```
type B
  effect E {
   def op() : Unit
  }

module b : B
  ...

type A
  effect E
  def handle[F](c: Unit -> {F, this.E} Unit): {F} Unit
  def m(): {this.E} Unit

module a: A
  effect E = {b.E}
  def handle[F](c: Unit -> {F, this.E} Unit): {F} Unit
     handle
       c()
     with
       b.op() -> resume ()
  def m(): {this.E} Unit
     b.op()

//Client1: Prints nothing
handle
  a.handle[b.E] (
     () => b.op(); a.m()
  )
with
    b.op() -> print "desired output"

//Client2: Prints "desired output"
handle
  a.handle[c.E] (
     () => c.op(); a.m()
  )
with
    c.op() -> print "desired output"
```

We translate module a in the following manner

```
module a: A
  effect E = {b.E}
  def handle(x: {_ => effect E}, c: Unit -> {x.E, this.E} Unit): {x.E} Unit
    unwrap{x.E} (
       handle
         c()
       with
         b.op() -> resume ()
     )
  def m(): {this.E} Unit
     b.op()
```

We translate the polymorphic call

```
a.handle[y.E](c)
```

as

```
val x = new
  effect E = y.E
unwrap{x.E}(a.handle(x:  {_ => effect E}, c))
```

Now we evaluate client1:

```
handle
  a.handle[b.E] (
     () => b.op(); a.m()
  )
with
  b.op() -> print "desired output"
---->
handle
  let b' = b:{_ => effect E{...}}>>{_ => effect E} in
  a.handle(b', () => b'.op(); a.m())
with
  b.op() -> print "desired output"
---->
handle
  unwrap{b.E}(
    handle
      unwrap{a.E}(wrap{b.E}(b.op); wrap{a.E}(b.op))
    with
      b.op() -> resume ()
  )
with
  b.op() -> print "desired output"
```

```
--->
 handle
   unwrap{b.E}(unwrap{a.E}(wrap{b.E}(b.op); ())
 with
   b.op() -> print "desired output"
```

2 Language with wrap

2.1 syntax

$$
\begin{array}{lcll}
v & ::= & l & \textit{values} \\
 & \mid & v : \tau \gg \tau & \textit{coercion} \\
 & \mid & \texttt{unwrap}\ (v) & \\
e & ::= & x & \textit{expressions} \\
 & \mid & v & \\
 & \mid & \texttt{new}(x \Rightarrow \overline{d}) & \\
 & \mid & e.m(e) & \\
 & \mid & x.op(e) & \\
 & \mid & handle\{h\}e & \\
 & \mid & \texttt{wrap}\ _{l.E}(e) & \\
 & \mid & \texttt{unwrap}\ _{l.E}(e) & \\
 & \mid & e : \tau \gg \tau & \textit{coercion} \\
d & ::= & \texttt{def}\ m(x:\tau) : \{\varepsilon\}\ \tau = e & \textit{declarations} \\
 & \mid & \texttt{effect}\ E = \overline{\texttt{o}} & \\
 & \mid & \texttt{effect}\ E = \varepsilon &
\end{array}
\qquad
\begin{array}{lcll}
\varepsilon & ::= & \overline{x.E} & \textit{effects} \\
\tau & ::= & \{x \Rightarrow \overline{\sigma}\} & \textit{type} \\
\sigma & ::= & \texttt{def}\ m(x:\tau) : \{\varepsilon\}\ \tau & \textit{decl. types} \\
 & \mid & \texttt{effect}\ E = \overline{\texttt{o}} & \\
 & \mid & \texttt{effect}\ E = \varepsilon & \\
 & \mid & \texttt{effect}\ E & \\
\texttt{o} & ::= & \texttt{def}\ op(x:\tau) : \tau & \textit{operation} \\
\texttt{h} & ::= & return\ x \to e & \textit{handler} \\
 & \mid & x.op(x) \to e; h & \\
\Gamma & ::= & \emptyset \mid \Gamma, x : \tau & \textit{typing context} \\
\mu & ::= & \emptyset \mid \mu, l \mapsto \{x \to \overline{d}\} & \textit{store} \\
\Sigma & ::= & \emptyset \mid \Sigma, l : \tau & \textit{store context}
\end{array}
$$

Figure 4: Language with wrap

In this language, we add locations l, type coercions $e : \tau \gg \tau$, wraped expression $\texttt{wrap}\ _{l.E}(e)$, and unwrapped expression $\texttt{unwrap}\ _{l.E}(e)$. Type coercions are added by the translation rules in section 2.4. Wraps and unwraps changes the dynamic of the program as shown in section 2.3. Every objects evaluates to a value. A value can be a location, a coerced value, or an unwrapped value.

2.2 Static Semantics

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{l : \tau \in \Sigma}{\Gamma \mid \Sigma \vdash l : \{\}\ \tau}\ (\text{T-Loc'}) \qquad \frac{x : \tau \in \Gamma \mid \Sigma}{\Gamma \mid \Sigma \vdash x : \{\}\ \tau}\ (\text{T-Var'})$$

$$\frac{\forall i,\ d_i \in \overline{d},\ \sigma_i \in \overline{\sigma},\ \Gamma,\ x : \{x \Rightarrow \overline{\sigma}\} \mid \Sigma \vdash d_i : \sigma_i}{\Gamma \mid \Sigma \vdash \texttt{new}(x \Rightarrow \overline{d}) : \{\}\ \{x \Rightarrow \overline{\sigma}\}}\ (\text{T-New'})$$

$$\frac{\begin{array}{c}\Gamma \mid \Sigma \vdash e_1 : \{\varepsilon_1\}\{x \Rightarrow \overline{\sigma}\} \quad \texttt{def}\ m(y : \tau_2) : \{\varepsilon_3\}\ \tau_1 \in \overline{\sigma} \\ \Gamma \mid \Sigma \vdash [e_1/x][e_2/y]\varepsilon_3\ wf \quad \Gamma \mid \Sigma \vdash e_2 : \{\varepsilon_2\}\ [e_1/x]\tau_2 \quad \varepsilon = \varepsilon_1 \cup \varepsilon_2 \cup [e_1/x][e_2/y]\varepsilon_3\end{array}}{\Gamma \mid \Sigma \vdash e_1.m(e_2) : \{\varepsilon\}\ [e_1/x][e_2/y]\tau_1}\ (\text{T-Method'})$$

$$\frac{\Gamma \mid \Sigma \vdash e : \{\varepsilon_1\}\ \tau_1 \quad \Gamma \mid \Sigma \vdash \tau_1 <: \tau_2 \quad \Gamma \mid \Sigma \vdash \varepsilon_1 <: \varepsilon_2}{\Gamma \mid \Sigma \vdash e : \{\varepsilon_2\}\ \tau_2}\ (\text{T-Sub'})$$

$$\frac{\Gamma \mid \Sigma \vdash x : \{x \Rightarrow \overline{\sigma}\} \quad \texttt{effect}\ E = \overline{o} \in \overline{\sigma} \quad \texttt{def}\ op(y : \tau_1) : \tau_2 \in \overline{o} \quad \Gamma \mid \Sigma \vdash e : \{\varepsilon\}\tau_1}{\Gamma \mid \Sigma \vdash x.op(e) : \{x.E, \varepsilon\}\tau_2}\ (\text{T-op'})$$

$$\frac{\begin{array}{c}\Gamma \mid \Sigma \vdash e : \{x.E, \varepsilon\}\ \tau \quad \Gamma \mid \Sigma \vdash findop(x.E) = \{x_1.op_1(y_1), \ldots, x_n.op_n(y_n)\} \quad \Gamma \mid \Sigma \vdash x_i.op_i(y_n) : \tau_i \rightarrow \tau_i' \\ \Gamma, resume\ :\ \{\_ \Rightarrow\ \texttt{def}\ m(z\ :\ \tau_i')\ :\ \{\varepsilon\}\ \tau_r\}, y_i\ :\ \tau_i\ \mid\ \Sigma\ \vdash\ e_i\ :\ \{\varepsilon\}\ \tau_r \quad \Gamma, x\ :\ \tau\ \mid\ \Sigma\ \vdash\ e_r\ :\ \{\varepsilon\}\ \tau_r\end{array}}{\Gamma \mid \Sigma \vdash \texttt{handle}\ \{x_1.op_1(y_1) \rightarrow e_i, ..., x_n.op_n(y_n) \rightarrow e_n, return\ x \rightarrow e_r\}e : \{\varepsilon\}\ \tau_r}\ (\text{T-handle'})$$

$$\frac{\Gamma \mid \Sigma \vdash e : \{\varepsilon\}\ \tau_1 \quad \Gamma \mid \Sigma \vdash \tau_1 <: \tau_2}{\Gamma \mid \Sigma \vdash (e : \tau_1 \gg \tau_2) : \{\varepsilon\}\ \tau_2}\ (\text{T-wrap-coercion'})$$

$$\frac{\Gamma \mid \Sigma \vdash e : \{\varepsilon\}\ \tau}{\Gamma \mid \Sigma \vdash \texttt{wrap}_{\ \varepsilon}(e) : \{\varepsilon\}\ \tau}\ (\text{T-wrap'}) \qquad \frac{\Gamma \mid \Sigma \vdash e : \{\varepsilon\}\ \tau}{\Gamma \mid \Sigma \vdash \texttt{unwrap}_{\ \varepsilon}(e) : \{\varepsilon\}\ \tau}\ (\text{T-unwrap'})$$

$\boxed{\Gamma \vdash d : \sigma}$

$$\frac{\Gamma,\ x : \tau_1 \mid \Sigma \vdash e : \{\varepsilon\} \quad \Gamma, x : \tau_1 \mid \Sigma \vdash \varepsilon\ wf}{\Gamma \mid \Sigma \vdash \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2 = e\ :\ \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2}\ (\text{DT-Def'})$$

$$\frac{}{\Gamma \mid \Sigma \vdash \texttt{effect}\ g = \overline{o}\ :\ \texttt{effect}\ g = \overline{o}}\ (\text{DT-Effect-1'}) \qquad \frac{\Gamma \mid \Sigma \vdash \varepsilon\ wf}{\Gamma \mid \Sigma \vdash \texttt{effect}\ g = \{\varepsilon\}\ :\ \texttt{effect}\ g = \{\varepsilon\}}\ (\text{DT-Effect-2'})$$

$\boxed{\mu : \Sigma}$

$$\frac{\forall l \in \mu, (l \mapsto \{x \rightarrow \overline{d}\} \in \mu \quad l : \{x \rightarrow \overline{\sigma}\} \in \Sigma \quad \forall d_i \in \overline{d},\ x : \{x \rightarrow \sigma_i\} \mid \Sigma \vdash d_i : \sigma_i)}{\mu : \Sigma}\ (\text{T-store})$$

$\boxed{\Gamma \mid \Sigma \vdash findop(x.E) = \{...\}}$

$$\frac{\Gamma \mid \Sigma \vdash x : \{y \Rightarrow \overline{d}\} \quad \texttt{effect}\ E = \overline{o}}{\Gamma \mid \Sigma \vdash findop(x.E) = \{x.o \mid o \in \overline{o}\}}\ (\text{T-find-1})$$

$$\frac{\Gamma \mid \Sigma \vdash x : \{y \Rightarrow \overline{d}\} \quad \texttt{effect}\ E = \{x_1.E_1, \ldots, x_n.E_n\} \quad \forall i \in [1, n], \Gamma \mid \Sigma \vdash findop(x_i.E_i) = s_i}{\Gamma \mid \Sigma \vdash findop(x.E) = s_1 \cup \cdots \cup s_n}\ (\text{T-find-2})$$

Figure 5: Static Semantics

2.3 Dynamic Semantics

$$
\begin{array}{llll}
E & ::= & [\,] & \textit{evaluation context} \\
 & | & E.m(e) & \\
 & | & v.m(E) & \\
 & | & E.op(e) & \\
 & | & v.op(E) & \\
 & | & handle\{E\}e & \\
 & | & handle\{h\}E & \\
 & | & \texttt{wrap}\ _{l.F}(E) & \\
 & | & \texttt{unwrap}\ _{l.F}(E) & \\
 & | & E : \tau \gg \tau' &
\end{array}
\qquad
\begin{array}{llll}
X^{es}_{l.op} & ::= & [\,] & \textit{evaluation context for op} \\
 & | & X^{es}_{l.op}.m(e) & \\
 & | & v.m(X^{es}_{l.op}) & \\
 & | & X^{es}_{l.op}.op(e) & \\
 & | & v.op(X^{es}_{l.op}) & \\
 & | & handle\{h\}(X^{es}_{l.op}) & \textit{if } l.op \to e \notin h \\
 & | & \texttt{wrap}\ _{l'.E}(X^{es}_{l.op}) & \textit{if } l.op \notin l'.E \\
 & | & \texttt{wrap}\ _{l'.E}(X^{es}_{l.op}) & \textit{if } l'.E \in es \\
 & | & \texttt{unwrap}\ _{l'.E}(X^{l'.E::es}_{l.op}) & \\
 & | & X^{es}_{l.op} : \tau \gg \tau' &
\end{array}
$$

Figure 6: Evaluation Contexts

$\boxed{\langle e \mid \mu \rangle \longrightarrow \langle e' \mid \mu' \rangle}$

$$\frac{\langle e \mid \mu \rangle \longrightarrow \langle e' \mid \mu' \rangle}{\langle E[e] \mid \mu \rangle \longrightarrow \langle E[e'] \mid \mu' \rangle}\ \text{(E-Congruence)}$$

$$\frac{l \notin dom(\mu)}{\langle \texttt{new}(x \Rightarrow \overline{d}) \mid \mu \rangle \longrightarrow \langle l \mid \mu, l \mapsto \{x \Rightarrow \overline{d}\} \rangle}\ \text{(E-New)}$$

$$\frac{return\ x \to e \in h}{\langle handle\{h\}(v) \mid \mu \rangle \longrightarrow \langle [v/x]e \mid \mu \rangle}\ \text{(E-Return)}$$

$$\frac{loc(v) = l \quad l.op(y) \to e \in h}{\langle handle\{h\}(X^{es}_{l.op}[v.op(v')]) \mid \mu \rangle \longrightarrow \langle [v'/y][new(\_ \Rightarrow \texttt{def}\ m(x : Unit) : \{\}Unit = handle\{h\}(X^{es}_{l.op}[x]))/resume]e \mid \mu \rangle}\ \text{(E-Handl}$$

$$\frac{l_1 \mapsto \{x \Rightarrow \overline{d}\} \in \mu \quad \texttt{def}\ m(y : \tau_1) : \{\varepsilon\}\ \tau_2 = e \in \overline{d}}{\langle l_1.m(v_2) \mid \mu \rangle \longrightarrow \langle [v_2/y][l_1/x]e \mid \mu \rangle}\ \text{(E-Method)}$$

$$\frac{loc(v_1) = l_1 \quad l_1 \mapsto \{x \Rightarrow \overline{d}\} \in \mu \quad \texttt{def}\ m(y : \tau_1) : \{l_1.E\}\ \tau_2 \in \tau' \quad \texttt{effect}\ E = \{\varepsilon\} \in \tau \quad \texttt{effect}\ E \in \tau'}{\langle (v_1 : \tau \gg \tau').m(v_2) \mid \mu \rangle \longrightarrow \langle \texttt{wrap}\ _{l_1.E}(v_1.m(\texttt{unwrap}\ _{l_1.E}(v_2))) \mid \mu \rangle}\ \text{(E-Method-Wrap)}$$

$$\frac{}{\langle \texttt{unwrap}\ _{l.E}(v_1).m(v_2) \mid \mu \rangle \longrightarrow \langle \texttt{unwrap}\ _{l.E}(v_1.m(v_2)) \mid \mu \rangle}\ \text{(E-Method-Unwrap)}$$

$$\frac{}{\langle \texttt{wrap}\ _{l'.E}(v) \mid \mu \rangle \longrightarrow \langle v \mid \mu \rangle}\ \text{(E-wrap)}$$

$\boxed{loc(v) = l}$

$$
\begin{array}{rl}
loc(l) & = l \\
loc(v : \tau \gg\ \tau) & = loc(v) \\
loc(\texttt{unwrap}\ (v)) & = loc(v)
\end{array}
$$

Figure 7: Wyvern dynamic semantics.

We use two evaluation contexts: E, and $X^{es}_{l.op}$. E is a standard evaluation context. $X^{es}_{l.op}$ is the evaluation context for an operation $l.op$, where $l.op$ is not handled or hidden inside an abstraction. Therefore, in rule E-Handle, we ensure that the current handler is the inner-most handler that handles the operation $v.op$. And more importantly, the operation is not hidden inside abstraction, so it is safe to handle it.

Rules E-Congruence, E-New, E-Return, and E-Method are standard object-oriented dynamic semantics. E-Method-Wrap introduces `wrap` and `unwrap` when there is an effect abstraction in type coercion. E-Method-Unwrap transfers `unwrap` from an object to its member. E-Wrap eliminates `wrap` .

2.4 Translation from surface language

$$\left[\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \{\}\ \tau}\ (\textsc{T-Var})\right] = x$$

$$\left[\frac{\forall i,\ d_i \in \overline{d},\ \sigma_i \in \overline{\sigma},\ D_i :: \Gamma, x : \{x \Rightarrow \overline{\sigma}\} \vdash d_i : \sigma_i}{\Gamma \vdash \texttt{new}(x \Rightarrow \overline{d}) : \{\}\ \{x \Rightarrow \overline{\sigma}\}}\ (\textsc{T-New})\right] = \texttt{new}\ (x \Rightarrow \overline{[D]})$$

$$\left[\frac{\begin{array}{c} E_1 :: \Gamma \vdash e_1 : \{\varepsilon_1\}\{x \Rightarrow \overline{\sigma}\} \quad \texttt{def}\ m(y : \tau_2) : \{\varepsilon_3\}\ \tau_1 \in \overline{\sigma} \\ \Gamma \vdash [e_1/x][e_2/y]\varepsilon_3\ wf \quad E_2 :: \Gamma \vdash e_2 : \{\varepsilon_2\}\ [e_1/x]\tau_2 \quad \varepsilon = \varepsilon_1 \cup \varepsilon_2 \cup [e_1/x][e_2/y]\varepsilon_3 \end{array}}{\Gamma \vdash e_1.m(e_2) : \{\varepsilon\}\ [e_1/x][e_2/y]\tau_1}\ (\textsc{T-Method})\right] = [E_1].m([E_2])$$

$$\left[\frac{E :: \Gamma \vdash e : \{\varepsilon_1\}\ \tau_1 \quad S :: \Gamma \vdash \tau_1 <: \tau_2 \quad \Gamma \vdash \varepsilon_1 <: \varepsilon_2}{\Gamma \vdash e : \{\varepsilon_2\}\ \tau_2}\ (\textsc{T-Sub})\right] = [E] : \tau_1 \gg\ \tau_2$$

$$\left[\frac{E_1 :: \Gamma \vdash e_1 : \{\varepsilon\}\{x \Rightarrow \overline{\sigma}\} \quad \texttt{effect}\ E = \overline{o} \in \overline{\sigma} \quad \texttt{def}\ op(x : \tau_1) : \tau_2 \in \overline{o} \quad E_2 :: \Gamma \vdash e_2 : \{\varepsilon\}\tau_1}{\Gamma \vdash e_1.op(e_2) : \{e_1.E, \varepsilon\}\tau_2}\ (\textsc{T-op})\right] = [E_1].op[E_2]$$

$$\left[\frac{\begin{array}{l} E :: \Gamma \vdash e : \{x.E, \varepsilon\}\ \tau \quad \Gamma \vdash findop(x.E) = \{x_1.op_1(y_1), \ldots, x_n.op_n(y_n)\} \quad \Gamma \vdash x_i.op_i : \tau_i \to \tau_i' \\ \Gamma, resume : \{\_ \Rightarrow \texttt{def}\ m(z : \tau_i') : \{\varepsilon\}\ \tau_r\}, y_i : \tau_i \vdash e_i : \{\varepsilon\}\ \tau_r \quad \Gamma, y : \tau \vdash e_r : \{\varepsilon\}\ \tau_r \end{array}}{\Gamma \vdash \texttt{handle}\ \{x_i.op_i(y_i) \to e_i, ..., x_n.op_n(y_n) \to e_n, return\ y \to e_r\}e : \{\varepsilon\}\ \tau_r}\ (\textsc{T-handle})\right]$$

$$= \texttt{handle}\ \{...\}[E]$$

Figure 8: Translation of terms

$$\left[\frac{E :: \Gamma,\ x : \tau_1 \vdash e : \{\varepsilon\}\ \tau_2 \quad \Gamma,\ x : \tau_1 \vdash \varepsilon\ \mathit{wf}}{\Gamma \vdash \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2 = e\ :\ \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2}\ (\text{DT-Def})\right] = \texttt{def}\ m(x : \tau_1) : \{\varepsilon\}\ \tau_2 = [E]$$

$$\left[\frac{}{\Gamma \vdash \texttt{effect}\ g = \overline{o}\ :\ \texttt{effect}\ g = \overline{o}}\ (\text{DT-Effect-1})\right] = \texttt{effect}\ g = \overline{o}$$

$$\left[\frac{\Gamma \vdash \varepsilon\ \mathit{wf}}{\Gamma \vdash \texttt{effect}\ g = \{\varepsilon\}\ :\ \texttt{effect}\ g}\ (\text{DT-Effect-2})\right] = \texttt{effect}\ g = \{\varepsilon\}$$

Figure 9: Translation of declarations

2.5 Soundness

Theorem 2.1. *(Progress) If $\emptyset \mid \Sigma \vdash e : \{\varepsilon\}\ \tau$ (i.e., e is a closed, well-typed expression), then either*

1. *e is a value*
2. *For all μ such that $\mu : \Sigma$, there exists e', μ' such that $\langle e \mid \mu\rangle \longrightarrow \langle e' \mid \mu'\rangle$*
3. *$e = X_{l.op}[v.op(v')]$, where $loc(v) = l$*

Theorem 2.2. *(Preservation) If $\Gamma \mid \Sigma \vdash e : \{\varepsilon\}\ \tau$, $\mu : \Sigma$, and $\langle e \mid \mu\rangle \longrightarrow \langle e' \mid \mu'\rangle$, then there exists Σ' such that $\Sigma \subseteq \Sigma'$, $\mu' : \Sigma'$, and $\Gamma \mid \Sigma' \vdash e' : \{\varepsilon\}\ \tau$*

3 Fine-grain call-by-value multi-agent calculus

3.1 syntax

(*agents*)	$i, j ::= \{1 \dots n\}$
(*lists*)	$l ::= i \mid il$
(*value types*)	$\tau ::= unit \mid \tau \to \sigma$
(*computation types*)	$\sigma ::= \{\varepsilon\}\tau$
(*effect types*)	$\varepsilon ::= \cdot \mid f, \varepsilon \mid op, \varepsilon$
(*i values*)	$v_i ::= ()_i \mid \lambda x_i : \tau.\ c_i$
(*i expression*)	$e_i ::= x_i \mid v_i \mid [e_j]^{\tau}_{l}$
(*i computation*)	$c_i ::= \texttt{return}\ e_i \mid op(e_i, y.c_i) \mid \texttt{do}\ x \leftarrow c_i\ \texttt{in}\ c'_i \mid e_i\ e'_i \mid \texttt{with}\ h_i\ \texttt{handle}\ c_i$ $\mid [c_j]^{\sigma}_{l} \mid [op]^{\varepsilon}_{l}(e_i, y_i.c_i)$
(*i handler*)	$h_i ::= \texttt{handler}\ \{\texttt{return}\ x_i \mapsto c^r_i, op^1(x^1_i, k^1) \mapsto c^1_i \dots op^n(x^n_i, k^n) \mapsto c^n_i\}$

3.2 Operational Semantics

$\boxed{e \longrightarrow e'}$

$$\frac{e_j \mapsto e'_j}{[e_j]^{\tau}_{l} \longrightarrow [e'_j]^{\tau}_{l}}\ (\text{E-}\textsc{Congruence}) \qquad \frac{}{[()_j]^{unit}_{l} \longrightarrow ()_i}\ (\text{E-}\textsc{Unit})$$

$$\frac{}{[\lambda x_j : \tau'.\ c_j]^{\tau \to \sigma}_{jl} \longrightarrow \lambda x_i : \tau.\ [\{[x_i]^{\tau'}_{irev(l)}/x_j\}c_j]^{\sigma}_{jl}}\ (\text{E-}\textsc{Lambda})$$

$\boxed{c \longrightarrow c'}$

$$\frac{e_i \mapsto e'_i}{\texttt{return}\ e_i \longrightarrow \texttt{return}\ e'_i}\ (\text{E-}\textsc{Ret}) \qquad \frac{e_i \mapsto e'_i}{op(e_i, y_i.c_i) \longrightarrow op(e'_i, y_i, c_i)}\ (\text{E-}\textsc{Op})$$

$$\frac{e_i \longrightarrow e'_i}{[op]^{\varepsilon}_{l}(e_i; y_i.c_i) \longrightarrow [op]^{\varepsilon}_{l}(e'_i; y_i.c_i)}\ (\text{E-}\textsc{EmbedOp1}) \qquad \frac{\overline{\Delta_i}(\varepsilon) = \varepsilon'}{[op]^{\varepsilon}_{l}(v_i; y_i.c_i) \longrightarrow [op]^{\varepsilon'}_{l}(v_i; y_i.c_i)}\ (\text{E-}\textsc{EmbedOp2})$$

$$\frac{\Delta_i(\varepsilon) = \varepsilon \quad op \in \varepsilon}{[op]^{\varepsilon}_{l}(v_i; y_i.c_i) \longrightarrow op(v_i; y_i.c_i)}\ (\text{E-}\textsc{EmbedOp3}) \qquad \frac{\Delta_i(\varepsilon) = \varepsilon \quad op \notin \varepsilon \quad op' \in \varepsilon}{[op]^{\varepsilon}_{l}(v_i; y_i.c_i) \longrightarrow [op]^{\varepsilon \setminus op'}(v_i; y_i.c_i)}\ (\text{E-}\textsc{EmbedOp4})$$

$$\frac{e_i \longrightarrow e''_i}{e_i\ e'_i \longrightarrow e''_i\ e'_i}\ (\text{E-}\textsc{App1}) \qquad \frac{e_i \longrightarrow e'_i}{v_i\ e'_i \longrightarrow v_i\ e'_i}\ (\text{E-}\textsc{App2}) \qquad \frac{}{(\lambda x_i : \tau.\ c_i)\ v_i \longrightarrow \{v_i/x_i\}c_i}\ (\text{E-}\textsc{App3})$$

$$\frac{c_i \longrightarrow c''_i}{\texttt{do}\ x \leftarrow c_i\ \texttt{in}\ c'_i \longrightarrow \texttt{do}\ x \leftarrow c''_i\ \texttt{in}c'_i}\ (\text{E-}\textsc{Seq1}) \qquad \frac{}{\texttt{do}\ x \leftarrow \texttt{return}\ v_i\ \texttt{in}\ c'_i \longrightarrow \{v_i/x\}c'_i}\ (\text{E-}\textsc{Seq2})$$

$$\frac{}{\texttt{do}\ x \leftarrow op_i(v_i; y_i.c_i)\texttt{in}\ c'_i \longrightarrow op_i(v_i; y_i.\ \texttt{do}\ x \leftarrow c_i\ \texttt{in}\ c'_i)}\ (\text{E-}\textsc{Seq3})$$

$$\frac{\Delta_i(\varepsilon) = \varepsilon \quad op \notin \varepsilon}{\texttt{do}\ x \leftarrow [op_j]^{\varepsilon}_{l}(v_i; y_i.c_i)\texttt{in}\ c'_i \longrightarrow [op_j]^{\varepsilon}_{l}(v_i; y_i.\ \texttt{do}\ x \leftarrow c_i\ \texttt{in}\ c'_i)}\ (\text{E-}\textsc{Seq4})$$

$$\frac{c_i \longrightarrow c'_i}{\texttt{with}\ h_i\ \texttt{handle}\ c_i \longrightarrow\ \texttt{with}\ h_i\ \texttt{handle}\ c'_i}\ (\text{E-}\textsc{Handle1}) \qquad \frac{\texttt{return}\ x_i \mapsto c'_i \in h_i}{\texttt{with}\ h_i\ \texttt{handle}\ \ \texttt{return}\ v_i \longrightarrow \{v_i/x_i\}c'_i}\ (\text{E-}\textsc{Handle2})$$

$$\frac{op(x_i; k) \mapsto c'_i \in h_i \quad \Sigma(op) = \tau_A \to \tau_B}{\texttt{with}\ h_i\ \texttt{handle}\ op(v_; y_i.c_i) \longrightarrow \{v_i/x_i\}\{(\lambda y : \tau_B.\ \texttt{with}\ h_i\ \texttt{handle}\ c_i)/k\}c'_i}\ (\text{E-}\textsc{Handle3})$$

$$\frac{\Delta_i(\varepsilon) = \varepsilon \quad op \notin \varepsilon}{\texttt{with}\ h_i\ \texttt{handle}\ [op]^{\varepsilon}_{l}(v_i, y_i.c_i) \longrightarrow [op]^{\varepsilon}_{l}(v_i; y_i.\ \texttt{with}\ h_i\ \texttt{handle}\ c_i))}\ (\text{E-}\textsc{Handle4})$$

$$\frac{c_j \longrightarrow c'_j}{[c_j]^{\sigma}_{l} \longrightarrow [c'_j]^{\sigma}_{l}}\ (\text{E-}\textsc{Embed1}) \qquad \frac{}{[\ \texttt{return}\ v_j]^{\{\varepsilon\}\tau}_{l} \longrightarrow\ \texttt{return}\ [v_j]^{\tau}_{l}}\ (\text{E-}\textsc{Embed2})$$

$$\frac{\Sigma(op) = \tau_A \to \tau_B}{[op_j(v_j; y_j.c_j)]^{\{\varepsilon\}\tau}_{l} \longrightarrow [op_j]^{\varepsilon}_{l}([v_j]^{\tau_A}_{j}; y_i.\{[y_i]^{\tau_B}_{i}/y_j\}[c_j]^{\{\varepsilon\}\tau}_{l})}\ (\text{E-}\textsc{Embed3})$$

$$\frac{\Sigma(op) = \tau_A \to \tau_B \quad \Delta_j(\varepsilon') = \varepsilon' \quad op \notin \varepsilon'}{[[op_k]^{\varepsilon'}_{l'}(v_j; y_j.c_j)]^{\{\varepsilon\}\tau}_{l} \longrightarrow [op_k]^{\varepsilon}_{l'l}([v_j]^{\tau_A}_{j}; y_i.\{[y_i]^{\tau_B}_{i}/y_j\}[c_j]^{\{\varepsilon\}\tau}_{l})}\ (\text{E-}\textsc{Embed4})$$

Figure 10: Operational Semantics

3.3 Static Semantics

$\boxed{\Gamma \vdash e_i : \tau}$

$$\frac{}{\Gamma \vdash ()_i : 1}\ (\text{T-UNIT}) \qquad \frac{}{\Gamma \vdash x_i : \Gamma(x_i)}\ (\text{T-VAR}) \qquad \frac{\Gamma, x_i : \tau \vdash c_i : \sigma}{\Gamma \vdash (\lambda x_i : \tau.\ c_i) : \tau \to \sigma}\ (\text{T-LAM})$$

$$\frac{\Gamma \vdash e_j : \tau' \quad \Gamma \vdash \tau' \leq_{li} \tau}{\Gamma \vdash [e_j]_l^\tau : \tau}\ (\text{T-EMBEDEXP})$$

$\boxed{\Gamma \vdash c_i : \sigma}$

$$\frac{\Gamma \vdash e_i : \tau}{\Gamma \vdash \texttt{return}\ e_i : \{\varepsilon\}\tau}\ (\text{T-RET}) \quad \frac{\Sigma(op) = \tau_A \to \tau_B \quad \Gamma \vdash e_i : \tau_A \quad \Gamma.y_i : \tau_B \vdash c_i : \{\varepsilon\}\tau \quad op \in \Delta_i(\varepsilon)}{\Gamma \vdash op(e_i; y_i.c_i) : \{\varepsilon\}\tau}\ (\text{T-OP})$$

$$\frac{\Gamma \vdash c_i : \{\varepsilon\}\tau \quad \Gamma, x_i : \tau \vdash c_i' : \{\varepsilon\}\tau'}{\Gamma \vdash \texttt{do}\ x_i \leftarrow c_i\ \texttt{in}\ c_i' : \{\varepsilon\}\tau'}\ (\text{T-SEQ}) \quad \frac{\Gamma \vdash e_1 : \tau \to \sigma \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2\ : \sigma}\ (\text{T-APP})$$

$$\frac{\begin{array}{c} h_i = \{\ \texttt{return}\ x \mapsto c^r, op^1(x;k) \mapsto c^1, \ldots, op^n(x;k) \mapsto c^n\} \\ \Gamma, x : \tau_A \vdash c^r : \{\varepsilon'\}\tau_B \quad \big\{\Sigma(op^i) = \tau_i \to \tau_i' \quad \Gamma, x : \tau_i, k : \tau_i' \to \{\varepsilon'\}\tau_B \vdash c^i : \{\varepsilon'\}\tau_B\big\}_{1 \leq i \leq n} \\ \Gamma \vdash c_i : \{\varepsilon\}\tau_A \quad \varepsilon \setminus \{op^i\}_{1 \leq i \leq n} \subseteq \varepsilon' \end{array}}{\Gamma \vdash \texttt{with}\ h_i\ \texttt{handle}\ c_i : \{\varepsilon'\}\tau_B}\ (\text{T-HANDLE})$$

$$\frac{\Gamma \vdash c_j : \sigma' \quad \Gamma \vdash \sigma' \leq_{li} \sigma}{\Gamma \vdash [c_j]_l^\sigma : \sigma}\ (\text{T-EMBED})$$

$$\frac{\Sigma(op) = \tau_A \to \tau_B \quad \Gamma \vdash e_i : \tau_A \quad \Gamma.y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau \quad \overline{\Delta_i}(\varepsilon) \subseteq \overline{\Delta_i}(\varepsilon') \quad \Gamma \vdash op \leq_{li} \varepsilon}{\Gamma \vdash [op]_l^\varepsilon(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau}\ (\text{T-EMBEDOP})$$

Figure 11: Static Semantics

$\boxed{\tau \leq_l \tau'}$

$$\frac{}{\mathit{unit} \leq_l \mathit{unit}}\ (\textsc{R-unit}) \qquad \frac{\tau \leq_l \tau' \quad \sigma \leq_l \sigma'}{\tau \to \sigma \leq_l \tau' \to \sigma'}\ (\textsc{R-arrow})$$

$\boxed{\sigma \leq_l \sigma}$

$$\frac{\varepsilon \leq_l \varepsilon' \quad \tau \leq_l \tau'}{\{\varepsilon\}\tau \leq_l \{\varepsilon'\}\tau'}\ (\textsc{R-sigma})$$

$\boxed{\varepsilon \leq_l \varepsilon}$

$$\frac{\overline{\Delta_i}(\varepsilon) \subseteq \overline{\Delta_i}(\varepsilon')}{\varepsilon \leq_i \varepsilon'}\ (\textsc{R-eff1}) \qquad \frac{\varepsilon \leq_l \varepsilon'' \quad \varepsilon'' \leq_{l'} \varepsilon'}{\varepsilon \leq_{ll'} \varepsilon'}\ (\textsc{R-eff2})$$

Figure 12: Type Relations

3.4 Safety

Lemma 3.1. *(Substitution)*
If $\Gamma, x_j : \tau' \vdash c_i : \sigma$ and $\Gamma \vdash e_j : \tau'$, then $\Gamma \vdash \{e_j/x_j\}c_i : \sigma$

Lemma 3.2. *If $\Gamma \vdash c_i : \{\varepsilon\}\tau$ then $\overline{\Delta_i}(\varepsilon) = \varepsilon$*

Lemma 3.3. *If $op \leq_l \varepsilon$, then $op \leq_l \varepsilon \setminus op'$*

Lemma 3.4. *(Preservation)*
If $\Gamma \vdash c_i : \{\varepsilon\}\tau$ and $c_i \longrightarrow c_i'$, then $\Gamma \vdash c_i' : \{\varepsilon\}\tau$

Proof. (Sketch) By induction on the derivation that $c_i \longrightarrow c_i'$. We proceed by the cases on the last step of the derivation.

1. E-Ret: By inversion, $\Gamma \vdash e_i : \tau$. By preservation of expressions and IH, we have $\Gamma \vdash e_i' : \tau$. Then we can use E-Ret to derive $\Gamma \vdash c_i' : \{\varepsilon\}\tau$

2. E-Op: Follow immediately from inversion and IH

3. E-EmbedOp1: Follow immediately from inversion and IH

4. E-EmbedOp2:

$$\frac{\overline{\Delta_i}(\varepsilon) = \varepsilon''}{[op_j]_l^{\varepsilon}(v_i; y_i.c_i) \longrightarrow [op_j]_l^{\varepsilon''}(v_i; y_i.c_i)}\ (\textsc{E-EmbedOp2})$$

We have the typing rule as follows:

$$\frac{\Sigma(op) = \tau_A \to \tau_B \quad \Gamma \vdash e_i : \tau_A \quad \Gamma.y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau \quad \overline{\Delta_i}(\varepsilon) \subseteq \overline{\Delta_i}(\varepsilon') \quad \Gamma \vdash op \leq_{li} \varepsilon}{\Gamma \vdash [op]_l^{\varepsilon}(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau}\ (\textsc{T-EmbedOp})$$

Since $\overline{\Delta_i}(\varepsilon'') = \varepsilon''$ and $\varepsilon'' = \overline{\Delta_i}(\varepsilon)$, we have $\overline{\Delta_i}(\varepsilon'') \subseteq \overline{\Delta_i}(\varepsilon')$. Then we can use T-EmbedOp to derive $\Gamma \vdash [op]_l^{\varepsilon''}(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau$

5. E-EmbedOp3: We have the typing rule as follows:

$$\frac{\Sigma(op) = \tau_A \to \tau_B \quad \Gamma \vdash e_i : \tau_A \quad \Gamma.y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau \quad \overline{\Delta_i}(\varepsilon) \subseteq \overline{\Delta_i}(\varepsilon') \quad \Gamma \vdash op \leq_{li} \varepsilon}{\Gamma \vdash [op]_l^{\varepsilon}(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau} \text{ (T-EmbedOp)}$$

By E-EmbedOp3, $op \in \overline{\Delta_i}(\varepsilon)$. So $op \in \overline{\Delta_i}(\varepsilon')$. By inversion on the typing rule, we have $\Gamma, y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau$. By lemma 3.2, we have $\Gamma, y_i : \tau_B \vdash c_i : \{\overline{\Delta_i}(\varepsilon')\}\tau$. Then we can use T-Op to derive the designed result $\Gamma \vdash op(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau$

6. E-EmbedOp4: We have the typing rule as follows:

$$\frac{\Sigma(op) = \tau_A \to \tau_B \quad \Gamma \vdash e_i : \tau_A \quad \Gamma.y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau \quad \overline{\Delta_i}(\varepsilon) \subseteq \overline{\Delta_i}(\varepsilon') \quad \Gamma \vdash op \leq_{li} \varepsilon}{\Gamma \vdash [op]_l^{\varepsilon}(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau} \text{ (T-EmbedOp)}$$

By lemma 3.3, we have $op \leq_{li} \varepsilon \setminus op'$. By inversion on the typing rule, we have $\Gamma, y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau$ and $\varepsilon \subseteq \overline{\Delta_i}(\varepsilon')$. So $\varepsilon \setminus op' \subseteq \overline{\Delta_i}(\varepsilon')$. By lemma 3.2, we have $\Gamma, y_i : \tau_B \vdash c_i : \{\overline{\Delta_i}(\varepsilon')\}\tau$. Then we can apply T-EmbedOp again to derive $\Gamma \vdash [op]_l^{\varepsilon \setminus op'}(e_i; y_i.c_i) : \{\overline{\Delta_i}(\varepsilon')\}\tau$

7. E-App1: Follows immediately by T-App

8. E-App2: Follows immediately by T-App

9. E-App3: By inversion of T-App, we $\Gamma \vdash (\lambda x_i : \tau.\ c_i) : \tau \to \sigma$, $\Gamma \vdash v_i : \tau$. By inversion of T-Lam, $\Gamma, x_i : \tau \vdash c_i : \sigma$. By substitution lemma, we have $\Gamma \vdash \{v_i/x_i\}c_i : \sigma$.

10. E-Seq1: Follows immediately by T-Seq and IH.

11. E-Seq2: By inversion on T-Seq, we have $\Gamma \vdash \texttt{return } v_i : \{\varepsilon\}\tau$ and $\Gamma, x_i : \tau \vdash c_i' : \{\varepsilon\}\tau'$. By inversion on T-Ret, we have $\Gamma \vdash v_i : \tau$. Then by substitution lemma we have $\Gamma \vdash \{v_i/x\}c_i' : \{\varepsilon\}\tau'$.

12. E-Seq3:

$$\frac{}{\texttt{do } x \leftarrow op_i(v_i; y_i.c_i)\texttt{in } c_i' \longrightarrow op_i(v_i; y_i.\ \texttt{do } x \leftarrow c_i \texttt{ in } c_i')} \text{ (E-Seq3)}$$

By inversion of T-Seq, we have $\Gamma \vdash op_i(v_i; y_i.c_i) : \{\varepsilon\}\tau$ and $\Gamma, x : \tau \vdash c_i' : \{\varepsilon\}\tau'$. By inversion on T-OP, we have $\Gamma, y_i : \tau_B \vdash c_i : \{\varepsilon\}\tau$ and $op \in \varepsilon$ and $\Gamma \vdash v_i : \tau_A$. Then by T-Seq, we have $\Gamma, y_i : \tau_B \vdash \texttt{do } x \leftarrow c_i \texttt{ in } c_i' : \{\varepsilon\}\tau'$. Then we can use T-Op to derive $\Gamma \vdash op_i(v_i; y_i.\ \texttt{do } x \leftarrow c_i \texttt{ in } c_i') : \{\varepsilon\}\tau'$.

13. E-Seq4

$$\frac{\Delta_i(\varepsilon) = \varepsilon \quad op \notin \varepsilon}{\texttt{do } x \leftarrow [op_j]_l^{\varepsilon}(v_i; y_i.c_i)\texttt{in } c_i' \longrightarrow [op_j]_l^{\varepsilon}(v_i; y_i.\ \texttt{do } x \leftarrow c_i \texttt{ in } c_i')} \text{ (E-Seq4)}$$

$$\frac{\Gamma \vdash c_i : \{\varepsilon'\}\tau \quad \Gamma, x_i : \tau \vdash c_i' : \{\varepsilon'\}\tau'}{\Gamma \vdash \texttt{do } x_i \leftarrow c_i \texttt{ in } c_i' : \{\varepsilon'\}\tau'} \text{ (T-seq)}$$

By inversion on T-Seq, we have $\Gamma \vdash [op_j]^{\varepsilon}_{l}(v_i; y_i.c_i) : \{\varepsilon'\}\tau$ and $\Gamma, x : \tau \vdash c'_i : \{\varepsilon'\}\tau'$. Then by inversion on T-EmbedOp, we have $\Gamma, y_i : \tau_B \vdash c_i : \{\varepsilon'\}\tau$, $\overline{\Delta}_i(\varepsilon) \subseteq \varepsilon'$. Then by T-Seq, we have $\Gamma, y_i : \tau_B \vdash \texttt{ do } x \leftarrow c_i \texttt{ in } c'_i : \{\varepsilon'\}\tau'$. Then by T-EmbedOp, we have $\Gamma \vdash [op]^{\varepsilon}_{l}(v_i; y_i.\texttt{ do } x \leftarrow c_i \texttt{ in } c'_i) : \{\varepsilon'\}\tau'$

14. E-Handle1: Follows immediately by inversion on T-Handle and IH

15. E-Handle2: By T-Handle, we have $\Gamma \vdash \texttt{ with } h_i \texttt{ handle } \texttt{ return } v_i : \{\varepsilon'\}\tau_B$. By inversion on T-Handle, we have $\Gamma, x_i : \tau_A \vdash c'_i : \{\varepsilon\}\tau_B$, and $\Gamma \vdash \texttt{ return } v_i : \{\varepsilon\}\tau_A$. By inversion on T-Ret, we have $\Gamma \vdash v_i : \tau_A$. Then by substitution lemma, we have $\Gamma \vdash \{v_i/x_i\}c'_i : \{\varepsilon'\}\tau_B$.

16. E-Handle3

$$\frac{op(x_i; k) \mapsto c'_i \in h_i \quad \Sigma(op) = \tau_i \to \tau'_i}{\texttt{with } h_i \texttt{ handle } op(v;y_i.c_i) \longrightarrow \{v_i/x_i\}\{(\lambda y_i : \tau'_i.\, \texttt{with } h_i \texttt{ handle } c_i)/k\}c'_i} \text{ (E-HANDLE3)}$$

By T-Handle, we have $\Gamma \vdash \texttt{ with } h_i \texttt{ handle } op(v; y_i.c_i) : \{\varepsilon'\}\tau_B$. By inversion on T-Handle, we have $\Gamma, x_i : \tau_i, k : \tau'_i \to \{\varepsilon'\}\tau_B \vdash c'_i : \{\varepsilon'\}\tau_B$, and $\Gamma \vdash op(v; y_i.c_i) : \{\varepsilon\}\tau_A$. By inversion on T-Op, we have $\Gamma \vdash v_i : \tau_i$ and $\Gamma, y_i : \tau'_i \vdash c_i : \{\varepsilon\}\tau_A$. By T-Handle, we have $\Gamma, y_i : \tau'_i \vdash \texttt{with } h_i \texttt{ handle } c_i : \{\varepsilon'\}\tau_B$. Then by T-Lam, we have $\Gamma \vdash \lambda y_i : \tau'_i.\ \texttt{with } h_i \texttt{ handle } c_i : \tau'_i \to \{\varepsilon'\}\tau_B$. Then, by substitution lemma, we have $\Gamma \vdash \{v_i/x_i\}\{(\lambda y_i : \tau'_i.\, \texttt{with } h_i \texttt{ handle } c_i)/k\}c'_i : \{\varepsilon'\}\tau_B$.

17. E-Handle4:

$$\frac{\Delta_i(\varepsilon) = \varepsilon \quad op \notin \varepsilon}{\texttt{with } h_i \texttt{ handle } [op]^{\varepsilon}_{l}(v_i, y_i.c_i) \longrightarrow [op]^{\varepsilon}_{l}(v_i; y_i.\, \texttt{with } h_i \texttt{ handle } c_i))} \text{ (E-HANDLE4)}$$

$$\frac{\begin{array}{c} h_i = \{ \texttt{ return } x \mapsto c^r, op^1(x;k) \mapsto c^1, \ldots, op^n(x;k) \mapsto c^n\} \\ \Gamma, x : \tau_A \vdash c^r : \{\varepsilon'\}\tau_B \quad \left\{\Sigma(op^i) = \tau_i \to \tau'_i \quad \Gamma, x : \tau_i, k : \tau'_i \to \{\varepsilon'\}\tau_B \vdash c^i : \{\varepsilon'\}\tau_B\right\}_{1 \le i \le n} \\ \Gamma \vdash c_i : \{\varepsilon''\}\tau_A \quad \varepsilon'' \setminus \{op^i\}_{1 \le i \le n} \subseteq \varepsilon' \end{array}}{\Gamma \vdash \texttt{ with } h_i \texttt{ handle } c_i : \{\varepsilon'\}\tau_B} \text{ (T-HANDLE)}$$

By T-Handle, we have $\Gamma \vdash \texttt{ with } h_i \texttt{ handle } [op]^{\varepsilon}_{l}(v; y_i.c_i) : \{\varepsilon'\}\tau_B$. By inversion on T-Handle, we have $\Gamma \vdash [op]^{\varepsilon}_{l}(v; y_i.c_i) : \{\varepsilon''\}\tau_A$ and $\varepsilon'' \setminus \{op^i\} \subseteq \varepsilon'$. By inversion on T-EmbedOp, we have $\Gamma \vdash v_i : \tau_i$, $\Gamma, y_i : \tau'_i \vdash c_i : \{\varepsilon''\}\tau_A$ and $\varepsilon \subseteq \varepsilon''$. Since ε doesn't contain any concrete operation, we have $\varepsilon \subseteq \varepsilon'' \setminus \{op^i\} \subseteq \varepsilon'$. Then by T-Handle, we have $\Gamma, y_i : \tau'_i \vdash \texttt{ with } h_i \texttt{ handle } c_i : \{\varepsilon'\}\tau_B$. Then, we use T-EmbedOp to derive $\Gamma \vdash [op]^{\varepsilon}_{l}(v_i; y_i.\, \texttt{with } h_i \texttt{ handle } c_i) : \{\varepsilon'\}\tau_B$

18. E-Embed1: Follows immediately from Inversion and IH

19. E-Embed2: By typing rule, we have $\Gamma \vdash [\ \texttt{return}\ v_j]_l^{\{\varepsilon\}\tau} : \{\varepsilon\}\tau$. By inversion on the typing rule, we have $\Gamma \vdash\ \texttt{return}\ v_j : \{\varepsilon'\}\tau'$ such that $\{\varepsilon\}\tau' \leq_{li} \{\varepsilon\}\tau$. By inversion on R-Sigma, we have $\tau' \leq_{li} \tau$. Then by T-EmbedExp, we have $\Gamma \vdash [v_j]_l^{\tau} : \tau$. Then by T-Ret, we have $\Gamma \vdash \texttt{return}\ [v_j]_l^{\tau} : \{\varepsilon\}\tau$. $\Gamma \vdash [\ \texttt{return}\ v_j]_l^{\{\varepsilon\}\tau} : \{\varepsilon\}\tau$

20. E-Embed3:

$$\frac{\Sigma(op) = \tau_A \to \tau_B}{[op(v_j; y_j.c_j)]_l^{\{\varepsilon\}\tau} \longrightarrow [op]_l^{\varepsilon}([v_j]_j^{\tau_A}; y_i.\{[y_i]_i^{\tau_B}/y_j\}[c_j]_l^{\{\varepsilon\}\tau})}\ (\text{E-EMBED3})$$

By typing rule, we have $\Gamma \vdash op(v_j; y_j.c_j) : \{\varepsilon'\}\tau'$, where $\{\varepsilon'\}\tau' \leq_{li} \{\varepsilon\}\tau$. By inversion on T-Op, we have $\Gamma \vdash v_j : \tau_A$, and $\Gamma, y_j : \tau_B \vdash c_j : \{\varepsilon'\}\tau'$. Then, by T-EmbedExp, we have $\Gamma \vdash [v_j]_j^{\tau_A} : \tau_A$. By substitution lemma, we have $\Gamma, y_i : \tau_B \vdash \{[y_i]_i^{\tau_B}/y_j\}c_j : \{\varepsilon'\}\tau'$. By T-Embed, we have $\Gamma, y_i : \tau_B \vdash \{[y_i]_i^{\tau_B}/y_j\}[c_j]_l^{\{\varepsilon\}\tau} : \{\varepsilon\}\tau$. Then we can use T-EmbedOp to derive $\Gamma \vdash [op]_l^{\varepsilon}([v_j]_j^{\tau_A}; y_i.\{[y_i]_i^{\tau_B}/y_j\}[c_j]_l^{\{\varepsilon\}\tau}) : \{\varepsilon\}\tau$.

21. E-Embed4:

$$\frac{\Sigma(op_k) = \tau_A \to \tau_B \quad \Delta_j(\varepsilon') = \varepsilon' \quad op \notin \varepsilon'}{[[op_k]_{l'}^{\varepsilon'}(v_j; y_j.c_j)]_l^{\{\varepsilon\}\tau} \longrightarrow [op_k]_{l'jl}^{\varepsilon}([v_j]_j^{\tau_A}; y_i.\{[y_i]_i^{\tau_B}/y_j\}[c_j]_l^{\{\varepsilon\}\tau})}\ (\text{E-EMBED4})$$

By typing rule, we have $\Gamma \vdash [op_K]_{l'}^{\varepsilon'}(v_j; y_j.c_j) : \{\varepsilon''\}\tau''$, where $\{\varepsilon''\}\tau'' \leq_{li} \{\varepsilon\}\tau$. By inversion on T-EmbedOp, we have $\Gamma \vdash v_j : \tau_A$ and $\Gamma, y_j : \tau_B \vdash c_j : \{\varepsilon''\}\tau''$. Then, by T-EmbedExp, we have $\Gamma \vdash [v_j]_j^{\tau_A} : \tau_A$. By substitution lemma, we have $\Gamma, y_i : \tau_B \vdash \{[y_i]_i^{\tau_B}/y_j\}c_j : \{\varepsilon''\}\tau''$. By T-Embed, we have $\Gamma, y_i : \tau_B \vdash \{[y_i]_i^{\tau_B}/y_j\}[c_j]_l^{\{\varepsilon\}\tau} : \{\varepsilon\}\tau$. Then we use T-EmbedOp to derive $\Gamma \vdash [op_k]_{l'jl}^{\varepsilon}([v_j]_j^{\tau_A}; y_i.\{[y_i]_i^{\tau_B}/y_j\}[c_j]_l^{\{\varepsilon\}\tau}) : \{\varepsilon\}\tau$.

□

Lemma 3.5. *(Progress)*
If $\varnothing \vdash c_i : \{\varepsilon\}\tau$ then either

1. $c_i \longrightarrow c_i'$
2. $c_i = \texttt{return}\ v_i$
3. $c_i = op(v_i; y_i.c_i')$
4. $c_i = [op]_l^{\varepsilon}(v_i; y_i.c_i')$

Definition 3.1. A i-computation c is <u>oblivious</u> to effect label f if $f \notin Dom(\delta_i)$, and for all subexpression $[e]_j^{\tau}$ and subcomputation $[c]_j^{\sigma}$, $f \notin Dom(\delta_j)$

Theorem 3.6. *Let c_1 and c_2 be computations that are oblivious to the effect f. If $c_1 \approx c_2$, $c_1 \to c_1'$, $c_2 \to c_2'$, then $c_1' \approx c_2'$. Furthermore, If e_1, e_2 oblivious to f, $e_1 \approx e_2$, $e_1 \to e_1'$, $e_2 \to e_2'$, then $e_1' \approx e_2'$*

The relation $\approx$ is defined as follows:

$\boxed{e \approx e}$

$$\frac{}{x \approx x}\ (\text{R-Var}) \qquad \frac{}{() \approx ()}\ (\text{R-Unit})$$

$$\frac{c \approx c'}{\lambda x : \tau.\ c \approx \lambda x : \tau.\ c'}\ (\text{R-Lam}) \qquad \frac{e \approx e'}{[e]_l^\tau \approx [e']_l^\tau}\ (\text{R-EmbedExp})$$

$\boxed{c \approx c}$

$$\frac{e \approx e'}{\texttt{return}\ e \approx\ \texttt{return}\ e'}\ (\text{R-Ret}) \qquad \frac{e \approx e' \quad c \approx c'}{op''(e; y.c) \approx op''(e'; y.c')}\ (\text{R-op})$$

$$\frac{c \approx c' \quad d \approx d'}{\texttt{do}\ x \leftarrow c\ \texttt{in}\ d \approx\ \texttt{do}\ x \leftarrow c'\ \texttt{in}\ d'}\ (\text{R-Seq}) \qquad \frac{e_1 \approx e_1' \quad e_2 \approx e_2'}{e_1\ e_2 \approx e_1'\ e_2'}\ (\text{R-App})$$

$$\frac{h \approx h' \quad c \approx c'}{\texttt{with}\ h\ \texttt{handle}\ c \approx\ \texttt{with}\ h'\ \texttt{handle}\ c'}\ (\text{R-Handle}) \qquad \frac{c_j \approx c_j'}{[c_j]_l^\sigma \approx [c_j']_l^\sigma}\ (\text{R-Embed})$$

$$\frac{e \approx e' \quad c \approx c'}{[op'']_l^\varepsilon(e; y.c) \approx [op'']_l^\varepsilon(e'; y.c')}\ (\text{R-Embedop1}) \qquad \frac{e \approx e' \quad c \approx c' \quad \exists i \in l, \delta_i(f) = op, op'}{[op]_l^\varepsilon(e; y.c) \approx [op']_l^\varepsilon(e'; y.c')}\ (\text{R-Embedop2})$$

$\boxed{h \approx h}$

$$\frac{c_r \approx c_r' \quad c_1 \approx c_1', \ldots c_n \approx c_n'}{\begin{array}{c}\{\ \texttt{return}\ x \mapsto c_r, op_1(x_1, k_1) \mapsto c_1, \ldots, op_n(x_n, k_n) \mapsto c_n\} \approx \\ \{\ \texttt{return}\ x \mapsto c_r', op_1(x_1, k_1) \mapsto c_1', \ldots, op_n(x_n, k_n) \mapsto c_n'\}\end{array}}\ (\text{R-Handler})$$

Figure 13: Definition of $\approx_{op,op'}$

Proof. (Sketch) By induction on derivation of $c_1 \approx c_2$ and $e_1 \approx e_2$

1. R-Ret: The only reduction rule that applies is E-Ret, so we have $e_1 \longrightarrow e_1'$ and $e_2 \longrightarrow e_2'$. By IH, we have $e_1' \approx e_2'$. Then the result follows by R-Ret

2. R-Op: The only reduction rule that applies is E-Op. The result is immediate by IH.

3. R-Seq: If the reduction rule is E-Seq1, then result is immediate by IH.

   If the reduction rule is E-Seq2. Then we have $c_1 =\ \texttt{do}\ x \leftarrow\ \texttt{return}\ v_1\ \texttt{in}\ d_1$, $c_2 =\ \texttt{do}\ x \leftarrow\ \texttt{return}\ v_2\ \texttt{in}\ d_2$. By inversion, we have $v_1 \approx v_2$ and $d_1 \approx d_2$. So we have $\{v_1/x\}d_1 \approx \{v_2/x\}d_2$.

   If the reduction rule is E-Seq3, then $c_1 =\ \texttt{do}\ x \leftarrow op(v_1; y.k_1)\ \texttt{in}\ d_1$, $c_2 =\ \texttt{do}\ x \leftarrow op(v_2; y.k_2)\ \texttt{in}\ d_2$. By inversion, we have $k_1 \approx k_2$ and $d_1 \approx d_2$. So $\texttt{do}\ x \leftarrow k_1\ \texttt{in}\ d_1 \approx\ \texttt{do}\ x \leftarrow k_2\ \texttt{in}\ d_2$. So $op(v_1; y.\ \texttt{do}\ x \leftarrow$

k_1 `in` $d_1) \approx op(v_2; y.$ `do` $x \leftarrow k_2$ `in` $d_2)$. The proof is similar for rule E-Seq4.

4. R-App: The cases for reduction rules E-App1 and E-App2 follows by IH. If reduction rule is E-App3. Then $c_1 = (\lambda x : \tau.\ d_1)\ v_1$ and $c_2 = (\lambda x : \tau.\ d_2)\ v_2$. By inversion we have $d_1 \approx d_2$ and $v_1 \approx v_2$. So we have $\{v_1/x\}d_1 \approx \{v_2/x\}d_2$.

5. R-Handle: If reduction rule is E-Handle1, then result follows by IH.

   If the reduction rule is E-Handle2. Then $c_1 =$ `with` h_1 `handle return` v_1 and $c_2 =$ `with` h_2 `handle return` v_2. By inversion we have $h_1 \approx h_2$, $v_1 \approx v_2$. Let `return` $c_{r1} \in h_1$ and `return` $c_{r2} \in h_2$. By inversion we have $c_{r1} \approx c_{r2}$. So $\{v_1/x\}c_{r1} \approx \{v_2/x\}c_{r2}$.

   If the reduction rule is E-Handle3. Then $c_1 =$ `with` h_1 `handle` $op(v_1; y.k_1)$ and $c_2 =$ `with` h_2 `handle` $op(v_2; y.k_2)$. By inversion we have $v_1 \approx v_2$, $k_1 \approx k_2$ and $h_1 \approx h_2$. Then by equivalents rules we derive $c_1' \approx c_2'$. The case for E-Handle4 is similar.

6. R-Embed: If reduction is E-Embed1, result is immediate by IH. If reduction rule is E-Embed2, then $c_1 = [$ `return` $v_1]_l^{\{\varepsilon\}\tau}$ and $c_2 = [$ `return` $v_2]_l^{\{\varepsilon\}\tau}$. It is easy to see $[v_1]^\tau \approx [v_2]^\tau$. So the result holds.

   If the reduction rule is E-Embed3, Then $c_1 = [op(v_1; y.k_1)]\{\varepsilon\}\tau_l$ and $c_2 = [op(v_2; y.k_2)]_l^{\{\varepsilon\}\tau}$. By inversion we have $v_1 \approx v_1$, $k_1 \approx k_2$. Then by equivalent rules we have $c_1' \approx c_2'$. Same arguments apply for E-Embed4.

7. R-EmbedOp1: If reduction rule is E-EmbedOp1, then result follows by IH. If reduction rules is E-EmbedOp2 or E-EmbedOp3, reduction does not affect terms except effect annotation, so the equivalence relation still hods after reduction.

8. R-EmbedOp2: Reduction rules E-EmbedOp1 and E-EmbedOp2 are similar to the previous case. If the reduction rule is E-EmbedOp3, then by R-EmbedOp2, the operations op and op' are exported as effect f by some agent, and since current agent is oblivious to f, this case is impossible.

□

3.5 Abstraction Problem I

```
op : 1 -> 1

module b: B
  f = op
  def m() : {f} Unit
    op ()
  def handler(c: 1 -> {f} 1) : {} Int =
    handle c () with
    |  op () -> 1
    | return _ -> 0

// Example program
b.handler(
  () => handle b.m() with
          | op -> resume ()
          | return _ -> ()
 )

// Translation of Example Program
[λc: 1 -> {f} 1.
  handle c() with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{(1\to\{f\}1)\to\{\}int}$
(λ_:1.
  handle [λ_:1. op()]$_b^{1\to\{f\}1}$ () with
  | op(x, k) -> k ()
  | return _ -> return ())

\\steps--->
 (λc: 1 -> {f} 1.
  [handle [c]$_a^{1\to\{f\}1}$ () with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$)
(λ_:1.
  handle [λ_:1. op()]$_b^{1\to\{f\}1}$ () with
  | op(x, k) -> k ()
  | return _ -> return ())

\\steps--->
  [handle
```

```
      [$\lambda$_:1.
       handle [$\lambda$_:1. op()]$_b^{1\to\{f\}1}$ () with
       | op(x, k) -> k ()
       | return _ -> return () ]$_a^{1\to\{f\}1}$ ()
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

 \\steps--->
[handle
     $\lambda$_:1.
       [handle [$\lambda$_:1. op()]$_b^{1\to\{f\}1}$ () with
       | op(x, k) -> k ()
       | return _ -> return () ]$_a^{\{f\}1}$ ()
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

\\steps--->
[handle
       [handle [$\lambda$_:1. op()]$_b^{1\to\{f\}1}$ () with
       | op(x, k) -> k ()
       | return _ -> return () ]$_a^{\{f\}1}$
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$


\\steps--->
[handle
       [handle $\lambda$_:1. [op()]$_b^{\{f\}1}$ () with
       | op(x, k) -> k ()
       | return _ -> return () ]$_a^{\{f\}1}$
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

\\steps--->
[handle
       [handle [op()]$_b^{\{f\}1}$ with
       | op(x, k) -> k ()
       | return _ -> return () ]$_a^{\{f\}1}$
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

\\steps--->
[handle
      [handle op$_b^f$((), y.return y) with
      | op(x, k) -> k ()
      | return _ -> return () ]$_a^{\{f\}1}$
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

\\steps--->
[handle
      [op$_b^f$((), y. handle return y with
                       | op(x, k) -> k ()
                       | return _ -> return ()
         )]$_a^{\{f\}1}$
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

\\steps--->
[handle
      op$_{ba}^f$((), y. [handle return y with
                       | op(x, k) -> k ()
                       | return _ -> return ()]$_a^{\{f\}1}$
         )
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$

\\steps--->
[handle
      op((), y. [handle return y with
                       | op(x, k) -> k ()
                       | return _ -> return ()]$_a^{\{f\}1}$
         )
  with
  | op(x, k) -> return 1
  | return x -> return 0)]$_b^{\{\}int}$
```

3.6 Effect Polymorphism

3.6.1 Syntax

(*agents*)	$i, j ::= \{1 \dots n\}$
(*lists*)	$l ::= i \mid il$
(*value types*)	$\tau ::= unit \mid \tau \to \sigma \mid \forall\alpha.\ \tau$
(*computation types*)	$\sigma ::= \{\varepsilon\}\tau$
(*effect types*)	$\varepsilon ::= \cdot \mid f, \varepsilon \mid op, \varepsilon \mid \alpha, \varepsilon$
(*i values*)	$v_i ::= ()_i \mid \lambda x_i : \tau.\ c_i \mid \Lambda\alpha.\ e$
(*i expression*)	$e_i ::= x_i \mid v_i \mid [e_j]^{\tau}_{l} \mid e\ [\varepsilon]$
(*i computation*)	$c_i ::= \texttt{return}\ e_i \mid op(e_i, y.c_i) \mid \texttt{do}\ x \leftarrow c_i\ \texttt{in}\ c'_i \mid e_i\ e'_i \mid \texttt{with}\ h_i\ \texttt{handle}\ c_i$
	$\mid [c_j]^{\sigma}_{l} \mid [op]^{\varepsilon}_{l}(e_i, y_i.c_i)$
(*i handler*)	$h_i ::= \texttt{handler}\ \{\texttt{return}\ x_i \mapsto c^r_i, op^1(x^1_i, k^1) \mapsto c^1_i \dots op^n(x^n_i, k^n) \mapsto c^n_i\}$

3.6.2 Dynamics

$$\frac{e \longrightarrow e'}{e\ [\varepsilon] \longrightarrow e'\ [\varepsilon]}$$

$$\frac{}{(\Lambda\alpha.\ e)\ [\varepsilon] \longrightarrow \{\varepsilon/\alpha\}e}$$

3.6.3 Static Semantics

$$\frac{\Gamma, \alpha\ effect \vdash e : \tau}{\Gamma \vdash \Lambda\alpha.\ e : \forall\alpha.\ \tau}$$

$$\frac{\Gamma \vdash e : \forall\alpha.\ \tau}{\Gamma \vdash e\ [\varepsilon] : [\varepsilon/\alpha]\tau}$$

3.6.4 Encoding of Abstraction Problem II

```
f = op
Λa. λc : (1 -> {a, f} 1).
  handle
    c ()
  with
  | op(_, k) -> k ()
  | return _ -> return ()
```