Effect Handlers, Evidently

NINGNING XIE, Microsoft Research, USA
JONATHAN IMMANUEL BRACHTHÄUSER, University of Tübingen, Germany
DANIEL HILLERSTRÖM, The University of Edinburgh, United Kingdom
PHILIPP SCHUSTER, University of Tübingen, Germany
DAAN LEIJEN, Microsoft Research, USA

Algebraic effect handlers are a powerful way to incorporate effects in a programming language. Sometimes perhaps even *too* powerful. In this article we define a restriction of general effect handlers with *scoped resumptions*. We argue one can still express all important effects, while improving local reasoning about effect handlers. Using the newly gained guarantees, we define a sound and coherent evidence translation for effect handlers which directly passes the handlers as evidence to each operation. We prove full soundness and coherence of the translation into plain lambda calculus. The evidence in turn enables efficient implementations of effect operations; in particular, we show we can execute tail-resumptive operations *in place* (without needing to capture the evaluation context), and how we can replace the runtime search for a handler by indexing with a constant offset.

1 INTRODUCTION

Algebraic effects [Plotkin and Power 2003] and the extension with handlers [Plotkin and Pretnar 2013], are a powerful way to incorporate effects in programming languages. Algebraic effect handlers can express any free monad in a concise and composable way, and can be used to express complex control-flow, like exceptions, asynchronous I/O, local state, backtracking, and much more.

Even though there are many language implementations of algebraic effects, like Koka [Leijen 2014], Eff [Pretnar 2015], Frank [Lindley et al. 2017], Links [Lindley and Cheney 2012], and Multicore OCaml [Dolan et al. 2015], the implementations may not be as efficient as one might hope. Generally, handling effect operations requires a linear search at runtime to the innermost handler. This is a consequence of the core operational rule for algebraic effect handlers:

$$\mathsf{handle}_m\ h\ \mathsf{E}[\mathsf{perform}\ op\ v] \quad \longrightarrow \quad f\ v\ k$$

requiring that $(op \to f)$ is in the handler h and that op is not in the bound operations in the evaluation context E (so the innermost handler gets to handle the operation). The operation clause f gets passed the operation argument v and the resumption $k = \lambda x.\ \mathsf{handle}_m\ h\ \mathsf{E}[x]$. Inspecting this rule, we can see that implementations need to search through the evaluation context to find the innermost handler, capture the context up to that point as the resumption, and can only then invoke the actual operation clause f. This search often is linear in the size of the stack, or in the number of intermediate handlers in the context E.

In prior work, it has been shown that the vast majority of operations can be implemented much more efficiently, often in time constant in the stack size. Doing so, however, requires an intricate runtime system [Dolan et al. 2015; Leijen 2017a] or explicitly passing handler implementations, instead of dynamically searching for them [Brachthäuser et al. 2018; Schuster et al. 2019; Zhang and Myers 2019]. While the latter appears to be an attractive alternative to implement effect handlers, a correspondence between handler passing and dynamic handler search has not been formally established in the literature.

In this article, we make this necessary connection and thereby open up the way to efficient compilation of effect handlers. We identify a simple restriction of general effect handlers, called *scoped resumptions*, and we show that under this restriction we can perform a sound and coherent *evidence translation* for effect handlers. In particular:

- The ability of effect handlers to capture resumptions k as a first-class value is very powerful – perhaps *too* powerful as it can interfere with the ability to do local reasoning. We define the notion of *scoped resumptions* (Section 2.2) as a restriction of general effect handlers where resumptions can only be applied under the scope of their original handler context. We believe all reasonable effect handlers can be written with scoped resumptions, while at the same time ruling out many "wild" applications that have non-intuitive semantics. In particular, it no longer allows handlers that change semantics of *other* operations than the ones it handles itself. This improves the ability to use local reasoning over effects, and the coherence of evidence translation turns out to only be preserved under scoped resumptions (more precisely: an evidence translated program does not get stuck if resumptions are scoped). In this paper, we focus on the evidence translation and use a dynamic check in our formalism. We show various designs on how to check this property statically, but leave full exploration of such a check to future work.
- To open up the way to more efficient implementations, we define a type directed *evidence translation* (Section 4) where the handlers are passed down as an implicit parameter to all operation invocations; similar to the dictionary translation in Haskell for type classes [Jones 1992], or capability passing in Effekt [Brachthäuser et al. 2020]. This turns out to be surprisingly tricky to get right, and we describe various pitfalls in Section 4.2. We prove that our translation is sound (Theorem 4 and 7) and coherent (Theorem 8), and that the evidence provided at runtime indeed always corresponds exactly to the dynamic innermost handler in the evaluation context (Theorem 5). In particular, on an evaluation step:

$$\mathsf{handle}_m\ h\ \mathsf{E}[\mathsf{perform}\ op\ ev\ v] \quad\longrightarrow\quad f\ v\ k \quad \text{with } op \notin \mathsf{bop}(\mathsf{E})\ \wedge (op \to f)\ \in h$$

  the provided evidence ev will be exactly the pair (m, h), uniquely identifying the actual (dynamic) handler m and its implementation h. This is the essence to enabling further optimizations for efficient algebraic effect handlers.

Building on the coherent evidence translation, we describe various techniques for more efficient implementations (Section 6):

- In practice, the majority of effects is *tail resumptive*, that is, their operation clauses have the form $op \to \lambda x.\lambda k.\ k\ e$ with $k \notin e$. That is, they always resume once in the end with the operation result. We can execute such tail resumptive operation clauses *in place*, e.g.

$$\mathsf{perform}\ op\ (m, h)\ v \quad\longrightarrow\quad f\ v\ (\lambda x.\ x) \quad (op_{\mathsf{tail}} \to f)\ \in h$$

  This is of course an important optimization that enables truly efficient effect operations at a cost similar to a virtual method call (since we can implement handlers h as a vector of function pointers where op is at a constant offset such that $f\ =\ h.op$).
- Generally, evidence is passed as an *evidence vector* w where each element is the evidence for a specific effect. That means we still need to select the right evidence at run-time which can be a linear time operation (much like the dynamic search for the innermost handler in the evaluation context). We show that by keeping the evidence vectors in canonical form, we can index the evidence in the vector at a *constant offset* for any context where the effect is non-polymorphic.
- Since the evidence provides the handler implementation directly, it is no longer needed in the context. We can follow Brachthäuser and Schuster [2017] and use an implementation based

on multi-prompt delimited continuations [Dyvbig et al. 2007; Gunter et al. 1995] instead. Given evidence (m, h), we directly yield to a specific prompt m:

$$\mathsf{handle}_m\ h\ \mathsf{E}[\mathsf{perform}\ op\ (m, h)\ v]$$
$$\rightsquigarrow$$
$$\mathsf{prompt}_m\ \mathsf{E}[\mathsf{yield}_m\ (\lambda k.\ (h.op)\ v\ k)]$$

We define a *monadic multi-prompt translation* (Section 5) from an evidence translated program (in F^{ev}) into standard call-by-value polymorphic lambda calculus (F^{v}) where the monad implements the multi-prompt semantics, and we prove that this monadic translation is sound (Theorem 10) and coherent (Theorem 11). Such translation is very important, as it provides the missing link between traditional implementations based on dynamic search for the handler [Dolan et al. 2015; Leijen 2014; Lindley et al. 2017] and implementations of lexical effect handlers using multi-prompt delimited control [Biernacki et al. 2019; Brachthäuser and Schuster 2017; Zhang and Myers 2019]. It also means we can use a standard compilation backend where all usual optimizations apply that would not hold under algebraic effect semantics directly (since all effects become explicit now). For example, as all handlers become regular data types, and evidence is a regular parameter, standard optimizations like inlining can often completely inline the operation clauses at the call site without any special optimization rules for effect handlers [Pretnar et al. 2017]. Moreover, no special runtime system for capturing the evaluation context is needed anymore, like split-stacks [Dolan et al. 2015] or stack copying [Leijen 2017a], and we can generate code directly for any host platform (including C or WebAssembly). In particular, recent advances in compilation guided reference counting [Ullrich and Moura 2019] can readily be used. Such reference counting transformations cannot be applied to traditional effect handler semantics since any effect operation may not resume (or resume more than once), making it impossible to track the reference counts directly.

We start by giving an overview of algebraic effects and handlers and their semantics in an untyped calculus λ^{ϵ} (Section 2), followed by a typed polymorphic formalization F^{ϵ} (Section 3) for which we prove various theorems like soundness, preservation, and the meaning of effect types. In Section 4 we define an extension of F^{ϵ} with explicit evidence vector parameters, called F^{ev}, define a formal evidence passing translation, and prove this translation is coherent and preserves the original semantics. Using the evidence translated programs, we define a coherent monadic translation in Section 5 (based on standard multi-prompt semantics) that translates into standard call-by-value polymorphic lambda-calculus (called F^{v}). Section 6 discusses various immediate optimization techniques enabled by evidence passing, in particular tail-resumption optimization, effect-selective monadic translation, and bind-inlining to avoid explicit allocation of continuations.

For space reasons, we put all evaluation context type rules and the full proofs of all stated lemmas and theorems in the supplemental Appendix which also includes further discussion of possible extensions.

2 UNTYPED ALGEBRAIC EFFECT HANDLERS

We begin by formalizing a minimal calculus of untyped algebraic effect handlers, called λ^{ϵ}. The formalization helps introduce the background, sets up the notations used throughout the paper, and enables us to discuss examples in a more formal way.

The formalization of λ^{ϵ} is given in Figure 1. It is essentially standard call-by-value lambda calculus extended with a rule to perform operations and a rule to handle them. It corresponds closely to the untyped semantics of Forster et al. [2019], and the effect calculus presented by Leijen [2017c]. Sometimes, effect handler semantics are given in a form that does not use evaluation contexts, e.g.

$$
\begin{array}{llll}
\text{Expressions} & & \text{Values} & \\
e ::= v & \text{(value)} & v ::= x & \text{(variables)} \\
\quad\mid e\ e & \text{(application)} & \quad\mid \lambda x.\ e & \text{(functions } f\text{)} \\
\quad\mid \mathsf{handle}\ h\ e & \text{(handler instance)} & \quad\mid \mathsf{handler}\ h & \text{(effect handler)} \\
 & & \quad\mid \mathsf{perform}\ op & \text{(operation)}
\end{array}
$$

$$
\begin{array}{llll}
\text{Handlers} & h & ::= \{ op_1 \to f_1, \ldots, op_n \to f_n \} & \text{(operation clauses)} \\
\text{Evaluation Context} & \mathsf{F} & ::= \square \mid \mathsf{F}\ e \mid v\ \mathsf{F} & \text{(pure evaluation)} \\
 & \mathsf{E} & ::= \square \mid \mathsf{E}\ e \mid v\ \mathsf{E} \mid \mathsf{handle}\ h\ \mathsf{E} & \text{(effectful computation)}
\end{array}
$$

$$
\begin{array}{llll}
(app) & (\lambda x.\ e)\ v & \longrightarrow & e[x{:=}v] \\
(handler) & (\mathsf{handler}\ h)\ v & \longrightarrow & \mathsf{handle}\ h \cdot v\ () \\
(return) & \mathsf{handle}\ h \cdot v & \longrightarrow & v \\
(perform) & \mathsf{handle}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ v & \longrightarrow & f\ v\ k \quad \text{iff } op \notin \mathsf{bop}(\mathsf{E}) \wedge (op \to f) \in h \\
 & & & \quad \text{where } k = \lambda x.\ (\mathsf{handle}\ h \cdot \mathsf{E} \cdot x)
\end{array}
$$

$$
\frac{e \longrightarrow e'}{\mathsf{E} \cdot e \longmapsto \mathsf{E} \cdot e'}\ [\textsc{step}]
\qquad
\begin{array}{lcl}
\mathsf{bop}(\square) & = & \varnothing \\
\mathsf{bop}(\mathsf{E}\ e) & = & \mathsf{bop}(\mathsf{E}) \\
\mathsf{bop}(v\ \mathsf{E}) & = & \mathsf{bop}(\mathsf{E}) \\
\mathsf{bop}(\mathsf{handle}\ h\ \mathsf{E}) & = & \mathsf{bop}(\mathsf{E}) \cup \{ op \mid (op \to f) \in h \}
\end{array}
$$

Fig. 1. λ^{ϵ}: Untyped Algebraic Effect Handlers

[Kammar and Pretnar 2017; Pretnar 2015], but in the end both formulations are equivalent (except that using evaluation contexts turns out to be convenient for our proofs).

There are two differences to earlier calculi: we leave out return clauses (for simplicity) and instead of one handle h expression we distinguish between handle $h\ e$ (as an expression) and handler h (as a value). A handler $h\ v$ evaluates to handle $h\ (v\ ())$ and just invokes its given function v with a unit value under a handle h frame. As we will see later, handler is generative and instantiates handle frames with a unique marker. As such, we treat handle as a strictly internal frame that only occurs during evaluation.

The evaluation contexts consist of *pure* evaluation contexts F and effectful evaluation contexts E that include handle h E frames. We assume a set of operation names op. The perform $op\ v$ construct calls an effect operation op by passing it a value v. Operations are handled by handle $h\ e$ expressions, which can be seen in the (*perform*) rule. Here, the condition $op \notin \mathsf{bop}(\mathsf{E})$ ensures that the *innermost* handle frame handles an operation. To evaluate an operation call, evaluation continues with the body of the *operation clause* ($op \to f$), passing the argument value v and the *resumption* k to f. Note that $f\ v\ k$ is not evaluated under the handler h, while the resumption always resumes under the handler h again; this describes the semantics of *deep* handlers and correspond to a *fold* in a categorical sense (as opposed to *shallow* handlers that are more like a *case*) [Kammar et al. 2013].

For conciseness, we often use the *dot notation* to decompose and compose evaluation contexts, which also conveys more clearly that an evaluation context essentially corresponds to a runtime stack. For example, we would write $v \cdot \mathsf{handle}\ h \cdot \mathsf{E} \cdot e$ as a shorthand for $v\ (\mathsf{handle}\ h\ (\mathsf{E}[e]))$. The dot notation can be defined as:

$$
\begin{array}{lcl}
\mathsf{E} \cdot e & \doteq & \mathsf{E}[e] \\
\square\ e \cdot \mathsf{E} & \doteq & \mathsf{E}\ e
\end{array}
\qquad
\begin{array}{lclcl}
v\ \square \cdot \mathsf{E} & \doteq & v \cdot \mathsf{E} & \doteq & v\ \mathsf{E} \\
\mathsf{handle}\ h\ \square \cdot \mathsf{E} & \doteq & \mathsf{handle}\ h \cdot \mathsf{E} & \doteq & \mathsf{handle}\ h\ \mathsf{E}
\end{array}
$$

2.1 Examples

Here are some examples of common effect handlers. Almost all practical uses of effect handlers are a variation of these.

Exceptions: Assuming we have data constructors just and nothing, we can define a handler for exceptions that converts any exceptional computation e to either just v on success, or nothing on an exception:

$$\text{handler}\ \{\ \mathit{throw} \to \lambda x.\ \lambda k.\ \text{nothing}\ \}\ (\lambda\_.\ \text{just}\ e)$$

For example using $e\ =\ $ perform throw () evaluates to nothing while $e\ =\ 1$ evaluates to just 1.

Reader: In the exception example we just ignored the argument and the resumption of the operation but the *reader* effect uses the resumption to resume with a result:

$$\text{handler}\ \{\ \mathit{get} \to \lambda x.\ \lambda k.\ k\ 1\ \}\ (\lambda\_.\ \text{perform}\ \mathit{get}\ ()\ +\ \text{perform}\ \mathit{get}\ ())$$

Here we handle the *get* operation to always return 1 so the evaluation proceeds as:

$$\begin{array}{ll}
 & \text{handler}\ \{\ \mathit{get} \to \lambda x.\ \lambda k.\ k\ 1\ \}\ (\lambda_.\ \text{perform}\ \mathit{get}\ ()\ +\ \text{perform}\ \mathit{get}\ ()) \\
\longmapsto^* & \text{handle}\ h \cdot \text{perform}\ \mathit{get}\ ()\ +\ \text{perform}\ \mathit{get}\ () \\
\longmapsto^* & (\lambda x.\ \text{handle}\ h \cdot (\square +\ \text{perform}\ \mathit{get}\ ()) \cdot x)\ 1 \\
\longmapsto & \text{handle}\ h \cdot (\square +\ \text{perform}\ \mathit{get}\ ()) \cdot 1 \\
\longmapsto^* & \text{handle}\ h \cdot (1\ +\ \square) \cdot 1 \\
\longmapsto^* & 2
\end{array}$$

State: The *state* effect is more involved with pure effect handlers as we need to return functions from the operation clauses (essentially as a state monad) (variant **1**):

$$\begin{array}{l}
h\ =\ \{\ \mathit{get} \to \lambda x.\ \lambda k.\ (\lambda y.\ k\ y\ y),\ \ \mathit{set} \to \lambda x.\ \lambda k.\ (\lambda y.\ k\ ()\ x)\ \} \\
(\text{handler}\ h\ (\lambda_.\ (\text{perform}\ \mathit{set}\ 21;\ x \leftarrow \text{perform}\ \mathit{get}\ ();\ (\lambda y.\ x\ +\ x)))\ 0
\end{array}$$

where we assume $x \leftarrow e_1;\ e_2$ as a shorthand for $(\lambda x.\ e_2)\ e_1$, and $e_1; e_2$ for $(\_ \leftarrow e_1; e_2)$.

The evaluation of an operation clause now always return directly with a function that takes the current state as its input; which is then used to resume with:

$$\begin{array}{ll}
 & (\text{handler}\ h\ (\lambda_.\ \text{perform}\ \mathit{set}\ 21;\ x \leftarrow \text{perform}\ \mathit{get}\ ();\ (\lambda y.\ y\ +\ x)\ ))\ 0 \\
\longmapsto^* & (\square\ 0) \cdot \text{handle}\ h \cdot (\square;\ x \leftarrow \text{perform}\ \mathit{get}\ ();\ (\lambda y.\ x\ +\ x)) \cdot \text{perform}\ \mathit{set}\ 21 \\
\longmapsto^* & (\square\ 0) \cdot (\lambda y.\ k\ ()\ 21) \quad \text{with}\ k\ =\ \lambda x.\ \text{handle}\ h \cdot (\square;\ x \leftarrow \text{perform}\ \mathit{get}\ ();\ (\lambda y.\ y\ +\ x)) \cdot x \\
= & (\lambda y.\ k\ ()\ 21)\ 0 \\
\longmapsto & k\ ()\ 21 \\
\longmapsto & (\text{handle}\ h \cdot (\square;\ \text{perform}\ \mathit{get}\ ()) \cdot ())\ 21 \\
= & (\square\ 21) \cdot \text{handle}\ h \cdot ((\,)\ ;\ \text{perform}\ \mathit{get}\ ()) \\
\longmapsto & 42
\end{array}$$

Clearly, defining local state as a function is quite cumbersome, so usually one allows for *parameterized handlers* [Leijen 2016; Plotkin and Pretnar 2013] that keep a local parameter p with their handle frame, where the evaluation rules become:

$$\begin{array}{lll}
\text{phandler}\ h\ v'\ v & \longrightarrow \text{phandle}\ h\ v' \cdot v\ () & \\
\text{phandle}\ h\ v' \cdot \mathsf{E} \cdot \text{perform}\ op\ v & \longrightarrow f\ \ v'\ \ v\ k & \text{iff}\ op \notin \text{bop}(\mathsf{E})\ \wedge\ (op \to f)\ \in h
\end{array}$$

where $k\ =\ \lambda y\ x.\ (\text{handle}\ h\ y \cdot \mathsf{E} \cdot x)$. Here the handler parameter v' is passed to the operation clause f and later restored in the resumption which now takes a fresh parameter y besides the result value x. With a parameterized handler the state effect can be concisely defined as (variant **2**):

$$\begin{array}{l}
h\ =\ \{\ \mathit{get} \to \lambda y\ x\ k.\ k\ y\ y,\ \ \mathit{set} \to \lambda y\ x\ k.\ k\ x\ ()\ \} \\
\text{phandler}\ h\ 0\ (\lambda_.\ \text{perform}\ \mathit{set}\ 21;\ x \leftarrow \text{perform}\ \mathit{get}\ ();\ x\ +\ x)
\end{array}$$

Another important advantage in this implementation is that the state effect is now *tail resumptive* which is very beneficial for performance (as shown in the introduction).

There as yet another elegant way to implement local state by Biernacki et al. [2017], where the *get* and *set* operations are defined in separate handlers (variant **3**):

$$h \;=\; \{\; set \rightarrow \lambda x\; k.\; \mathsf{handler}\; \{\; get \rightarrow \lambda\_\; k.\; k\; x \;\}\; (\lambda\_.\; k\; ())\;\}$$
$$\mathsf{handler}\; h\; (\lambda\_.\; \mathsf{perform}\; set\; 42;\;\; x \leftarrow \mathsf{perform}\; get\; ();\;\; x \;+\; x)$$

The trick here is that every *set* operation installs a fresh handler for the *get* operation and resumes under that (so the innermost *get* handler always contains the latest state). Even though elegant, there are some drawbacks to this encoding: a naive implementation may use n handler frames for n set operations, typing this example is tricky and usually requires *masking* [Biernacki et al. 2017; Hillerström and Lindley 2016], and, as we will see, it does not use *scoped resumptions* and thus cannot be used with evidence translation.

Backtracking: By resuming more than once, we can implement backtracking using algebraic effects. For example, the *amb* effect handler collects all all possible results in a list by resuming the *flip* operation first with a true result, and later again with a false result:

$$\mathsf{handler}\; \{\; flip \rightarrow \lambda\_\; k.\; xs \leftarrow k\; \mathsf{true};\;\; ys \leftarrow k\; \mathsf{false};\;\; xs \mathbin{++} ys \;\}$$
$$(\lambda\_.\; x \leftarrow \mathsf{perform}\; flip\; ();\;\; y \leftarrow \mathsf{perform}\; flip\; ();\;\; [x\; \&\&\; y])$$

returning the list [false, false, false, true] in our example. This technique can also be used for probabilistic programming [Kiselyov and Shan 2009].

Async: We can use resumptions k as first class values and for example store them into a queue to implement cooperative threads [Dolan et al. 2017] or asynchronous I/O [Leijen 2017b]. Assuming we have a state handler h_{queue} that maintains a queue of pending resumptions, we can implement a mini-scheduler as:

$$\begin{aligned} h_{async} \;=\; \{\; & fork \rightarrow \lambda f\; k.\; \mathsf{perform}\; enqueue\; k;\;\; schedule\; f\; () \\ & yield \rightarrow \lambda\_\; k.\; \mathsf{perform}\; enqueue\; k;\;\; k' \leftarrow \mathsf{perform}\; dequeue\; ();\;\; k'\; ()\;\} \end{aligned}$$

where *enqueue* enqueues a resumption k, and *dequeue* () resumes one, or returns unit () if the queue is empty. The *schedule* function runs a new action f under the scheduler handler:

$$\begin{aligned} schedule &= \lambda f\; \_.\; \mathsf{handler}\; h_{async}\; (\lambda\_.\; f\; ();\;\; \mathsf{perform}\; dequeue\; ()) \\ async &= \lambda f.\; \mathsf{handler}\; h_{queue}\; (\lambda\_.\; schedule\; f\; ()) \end{aligned}$$

The main wrapper *async* schedules an action under a fresh scheduler queue handler h_{queue}, which is shared by all forked actions under it.

2.2 Scoped Resumptions

The ability of effect handlers to capture the resumption as a first-class value is very powerful – and can be considered as perhaps *too* powerful. In particular, it can be (ab)used to define handlers that change the semantics of *other* handlers that were defined and instantiated orthogonally. Take for example an operation op_1 that is expected to always return the same result, say 1. We can now define another operation op_{evil} that changes the return value of op_1 after it is invoked! Consider the following program where we leave f and h_{evil} undefined for now:

$$\begin{aligned} & h_1 \;=\; \{\; op_1 \rightarrow \lambda x\; k.\; k\; 1\;\} \\ & e \;=\; \mathsf{perform}\; op_1\; ();\; \mathsf{perform}\; op_{evil}\; ();\; \mathsf{perform}\; op_1\; () \\ & f\; (\mathsf{handler}\; h_1\; (\lambda\_.\; \mathsf{handler}\; h_{evil}\; (\lambda\_.\; e))) \end{aligned}$$

Even though h_1 is defined as a pure reader effect and defined orthogonal to any other effect, the op_{evil} operation can still cause the second invocation of op_1 to return 2 instead of 1! In particular, we can define f and h_{evil} as [1]:

$$h_2 \;=\; \{\ op_1 \rightarrow \lambda x\ k.\ k\ 2\ \}$$
$$h_{evil} \;=\; \{\ op_{evil} \rightarrow \lambda x\ k.\ k\ \}$$
$$f \;=\; \lambda k.\ \mathsf{handler}\ h_2\ (\lambda\_.\ k\ ())$$

The trick is that the handler h_{evil} does not directly resume but instead returns the resumption k *as is*, after unwinding through h_1 it is passed to f which now invokes the resumption k under a fresh handler h_2 for op_1 causing all subsequent op_1 operations to be handled by h_2 instead.

We consider this behavior undesirable in practice as it limits the ability to do local reasoning. In particular, a programmer may not expect that calling op_{evil} changes the semantics of op_1. Yet there is no way to forbid it. Moreover, it also affects static analysis and it turns out for example that efficient evidence translation (with its subsequent performance benefits) is not possible if we allow resumptions to be this dynamic.

The solution we propose in this paper is to limit resumptions to be *scoped* only: that is, *a resumption can only be applied under the same handler context as it was captured*. The handler context is the evaluation context where we just consider the handler frames, e.g. for any evaluation context E of the form $\mathsf{F}_0 \cdot \mathsf{handle}\ h_1 \cdot \mathsf{F}_1 \cdot \ldots \cdot \mathsf{handle}\ h_n \cdot \mathsf{F}_n$, the handler context, $\mathsf{hctx}(\mathsf{E})$, is $h_1 \cdot h_2 \cdot \ldots \cdot h_n$. In particular, the evil example is rejected as it does not use a scoped resumption: k is captured under h_1 but applied under h_2.

Our definition of scoped resumption is *minimal* in the sense that it is the minimal requirement needed in the proofs to maintain coherence of evidence translation. In this paper, we guarantee scoped resumptions using a dynamic runtime check in evidence translated programs (called *guard*), but it is also possible to check it statically. It is beyond the scope of this paper to give a particular design, but some ways of doing this are:

- Lexical scoping: a straightforward approach is to syntactically restrict the use of the resumption to be always in the lexical scope of the handler: i.e. fully applied within the operation clause and no occurrences under a lambda (so it cannot escape or be applied in nested handler). This can perhaps already cover all reasonable effects in practice, especially in combination with parameterized handlers[2].
- A more sophisticated solution could use generative types for handler names, together with a check that those types do not escape the lexical scope as described by Zhang and Myers [2019] and also used by Biernacki et al. [2019] and Brachthäuser et al. [2020]. Another option could be to use rank-2 types to prevent the resumption from escaping the lexical scope in which the handler is defined [Leijen 2014; Peyton Jones and Launchbury 1995].

It turns out that the seminal work on algebraic effect handlers by Plotkin and Pretnar [2013] also used a similar restriction as scoped resumptions, and as such, we believe that scoped resumptions are closer to the original categorical interpretation of effect handlers. Plotkin and Pretnar use the first technique to syntactically restrict the use of resumptions under the scope of the operation clause. Resumptions variables k are in a separate syntactic class, always fully applied, and checked under a context K separate from Γ. However, they still allow occurrences under a lambda, allowing a resumption to escape, although in that case the evaluation would no longer type check (i.e. there is no preservation of typings under evaluation). As such it is not quite the same as scoped resumptions:

[1]Note that this example is fine in λ^{ϵ} but cannot be typed in F^{ϵ} as is – we discuss a properly typed version in Section 4.5.
[2]The lexical approach could potentially be combined with an "unsafe" resumption that uses a runtime check as done this article to cover any remaining situations.

Expressions

$$
\begin{array}{lcll}
e & ::= & v & \text{(value)} \\
 & | & e\ e & \text{(application)} \\
 & | & e[\sigma] & \text{(type application)} \\
 & | & \mathsf{handle}\ h\ e & \text{(handler instance)}
\end{array}
$$

Values

$$
\begin{array}{lcll}
v & ::= & x & \text{(variables)} \\
 & | & \lambda^{\epsilon} x : \sigma.\ e & \text{(abstraction)} \\
 & | & \Lambda \alpha^{k}.\ v & \text{(type abstraction)} \\
 & | & \mathsf{handler}^{\epsilon}\ h & \text{(effect handler)} \\
 & | & \mathsf{perform}^{\epsilon}\ op\ \overline{\sigma} & \text{(operation)}
\end{array}
$$

$$
\begin{array}{llcl}
\text{Handlers} & h & ::= & \{ op_1 \to f_1, \ldots, op_n \to f_n \} \\
\text{Evaluation Context} & \mathsf{F} & ::= & \square \mid \mathsf{F}\ e \mid v\ \mathsf{F} \mid \mathsf{F}\ [\sigma] \\
 & \mathsf{E} & ::= & \square \mid \mathsf{E}\ e \mid v\ \mathsf{E} \mid \mathsf{E}\ [\sigma] \mid \mathsf{handle}^{\epsilon}\ h\ \mathsf{E}
\end{array}
$$

$$
\begin{array}{llcl}
(app) & (\lambda^{\epsilon} x : \sigma.\ e)\ v & \longrightarrow & e[x{:=}v] \\
(tapp) & (\Lambda \alpha^{k}.\ v)\ [\sigma] & \longrightarrow & v[\alpha{:=}\sigma] \\
(handler) & (\mathsf{handler}^{\epsilon}\ h)\ v & \longrightarrow & \mathsf{handle}^{\epsilon}\ h \cdot v\ () \\
(return) & \mathsf{handle}^{\epsilon}\ h \cdot v & \longrightarrow & v \\
(perform) & \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v & \longrightarrow & f[\overline{\sigma}]\ v\ k \quad \text{iff } op \notin \mathsf{bop}(\mathsf{E}) \ \wedge\ (op \to f) \in h \\
 & & & \text{where} \quad op : \forall \overline{\alpha}.\ \sigma_1 \to \sigma_2 \ \in \Sigma(l) \\
 & & & \qquad\qquad k \ = \ \lambda^{\epsilon} x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x
\end{array}
$$

Fig. 2. System F^{ϵ}: explicitly typed algebraic effect handlers. Figure 3 defines the types.

it is both more restrictive as it needs k to occur fully applied under an operation clause; but also more liberal as it allows the separate handler state example (since k can occur under a lambda).

2.3 Expressiveness

Scoped resumptions bring easier-to-reason control flow, and, as we will see, open up new design space for algebraic effects compilation. However, one might worry about the expressiveness of scoped resumptions. We believe that all important effect handlers in practice can be defined in terms of scoped resumptions. In particular, note that it is still allowed for a handler to grow its context with applicative forms, for example:

$$\mathsf{handler}\ \{\ tick \to \lambda x\ k.\ 1\ +\ k\ ()\ \}\ (\lambda\_.\ tick\ ();\ tick\ ();\ 1)$$

evaluates to 3 by keeping $(1\ +\ \square)$ frames above the resumption. In this example, even though the full context has grown, k is still a scoped resumption as it resumes under the same (empty) handler context. Similarly, the async scheduler example that stores resumptions in a stateful queue is also accepted since each resumption still resumes under the same handler context (with the state queue handler on top). Multiple resumptions as in the backtracking example are also fine.

There are two main exceptions we know of. First, the state variant 3 based on two separate handlers does not use scoped resumptions since the *set* resumption resumes always under a handler context extended with a *get* handler. However, we can always use, and due to the reasons we have mentioned we may actually prefer, the normal state effect or the parameterized state effect. Second, shallow handlers do not resume under their own handler and as a result generally resume under a different handler context than they captured. Fortunately, any program with shallow handler can be expressed with deep handlers as well [Hillerström and Lindley 2018; Kammar et al. 2013] and thus avoid the unscoped resumptions.

3 EXPLICITLY TYPED EFFECT HANDLERS IN SYSTEM F^{ϵ}

Types			Kinds		
σ ::=	α^k	(type variables of kind k)	k ::=	$*$	(value type)
|	$c^k\ \sigma \ldots \sigma$	(type constructor of kind k)	|	$k \to k$	(type constructors)
|	$\sigma \to \epsilon\ \sigma$	(function type)	|	eff	(effect type (μ,ϵ))
|	$\forall \alpha^k.\ \sigma$	(quantified type)	|	lab	(basic effect (l))

Effect signature	sig	::=	$\{\ op_1 : \forall \overline{\alpha}_1.\ \sigma_1 \to \sigma'_1,\ \ldots,\ op_n : \forall \overline{\alpha}_n.\ \sigma_n \to \sigma'_n \}$
Effect signatures	Σ	::=	$\{ l_1 : sig_1,\ \ldots,\ l_n : sig_n \}$

Type Constructors	$\langle\rangle$	:	eff	empty effect row (total)
	$\langle \_ \mid \_ \rangle$	:	lab $\to$ eff $\to$ eff	effect row extension
	marker	:	eff $\to * \to *$	handler instance marker (m)
	evv	:	eff $\to *$	evidence vector (w,z)
	ev	:	lab $\to *$	evidence (ev)
	$\_ \to \_\ \_$	:	$* \to$ eff $\to *$	function arrow

Syntax

$$\langle l_1, \ldots, l_n \rangle \doteq \langle l_1 \mid \ldots \mid \langle l_n \mid \langle\rangle\rangle \ldots \rangle$$

$$\langle l_1, \ldots, l_n \mid \mu \rangle \doteq \langle l_1 \mid \ldots \mid \langle l_n \mid \mu \rangle \ldots \rangle$$

$$\epsilon ::= \sigma^{\mathsf{eff}}, \quad \mu ::= \alpha^{\mathsf{eff}}, \quad l ::= c^{\mathsf{lab}}$$

Fig. 3. System F^ϵ: types

$$\frac{}{\epsilon \equiv \epsilon}\ [\textsc{refl}] \qquad \frac{\epsilon_1 \equiv \epsilon_2 \quad \epsilon_2 \equiv \epsilon_3}{\epsilon_1 \equiv \epsilon_3}\ [\textsc{eq-trans}]$$

$$\frac{l_1 \neq l_2 \quad \epsilon_1 \equiv \epsilon_2}{\langle l_1, l_2 \mid \epsilon_1 \rangle \equiv \langle l_2, l_1 \mid \epsilon_2 \rangle}\ [\textsc{eq-swap}] \qquad \frac{\epsilon_1 \equiv \epsilon_2}{\langle l \mid \epsilon_1 \rangle \equiv \langle l \mid \epsilon_2 \rangle}\ [\textsc{eq-head}]$$

Fig. 4. Equivalence of row-types.

To prepare for a type directed evidence translation, we first define a typed version of the untyped calculus λ^ϵ called System F^ϵ – a call-by-value effect handler calculus extended with (higher-rank impredicative) polymorphic types and higher kinds à la System F_ω, and row based effect types. Figure 2 defines the extended syntax and evaluation rules with the syntax of types and kinds in Figure 3. System F^ϵ serves as an explicitly typed calculus that can be the target language of compilers and, for this article, serves as the basis for type directed evidence translation.

Being explicitly typed, we now have type applications $e[\sigma]$ and abstractions $\Lambda\alpha^k.\ v$. Also, $\lambda^\epsilon\ x\!:\!\sigma.e$, handle$^\epsilon$ $h\ e$, handler$^\epsilon$ h, and perform$^\epsilon$ $op\ \overline{\sigma}$ all carry an effect type ϵ. Effect types are (extensible) rows of effect labels l (like exn or state). In the types, every function arrow $\sigma_1 \to \epsilon\ \sigma_2$ takes three arguments: the input type σ_1, the output type σ_2, and its effects ϵ when it is evaluated.

Since we have effect rows, effect labels, and regular value types, we use a basic kind system to keep them apart and to ensure well-formedness ($\vdash_{\mathsf{wf}}$) of types (as defined in the Appendix).

3.1 Effect Rows

An effect row is either empty $\langle\rangle$ (the *total* effect), a type variable μ (of kind eff), or an extension $\langle l \mid \epsilon \rangle$ where ϵ is extended with effect label l. We call effects that end in an empty effect *closed*, i.e.

$\langle l_1, \ldots, l_n \rangle$; and effects that end in a polymorphic tail *open*, i.e. $\langle l_1, \ldots, l_n \mid \mu \rangle$. Following Biernacki et al. [2017] and Leijen [2014], we use *simple* effect rows where labels can be duplicated, and where an effect $\langle l, l \rangle$ is not equal to $\langle l \rangle$. We consider rows equivalent up to the order of the labels as defined in Figure 4. There exists a complete and sound unification algorithm for these row types [Leijen 2005] and thus these are also very suitable for Hindley-Milner style type inference.

We consider using simple row-types with duplicate labels a suitable choice for a core calculus since it extends System F typing seamlessly as we only extend the notion of equality between types. There are other approaches to typing effects but all existing approaches depart from standard System F typing in significant ways. Row typing without duplicate labels leads to the introduction of type constraints, as in T-REX for example [Gaster and Jones 1996], or kinds with presence variables (Rémy style rows) as in Links for example [Hillerström and Lindley 2016; Rémy 1994]. Another approach is using effect subtyping [Bauer and Pretnar 2014] but that requires a subtype relation between types instead of simple equality.

The reason we need equivalence between row types up to order of effect labels is due to polymorphism. Suppose we have two functions that each use different effects:

$$f_1 : \forall \mu.\ () \rightarrow \langle l_1 \mid \mu \rangle\ () \qquad\qquad f_2 : \forall \mu.\ () \rightarrow \langle l_2 \mid \mu \rangle\ ()$$

We would still like to be able to express *choose* f_1 f_2 where *choose*: $\forall \alpha.\ \alpha \rightarrow \alpha \rightarrow \alpha$. Using row types we can type this naturally as:

$$\Lambda \mu.\ \mathit{choose}[() \rightarrow \langle l_1, l_2 \mid \mu \rangle\ ()]\ (f_1[\langle l_2 \mid \mu \rangle])\ (f_2[\langle l_1 \mid \mu \rangle])$$

where the types of the arguments are now equivalent $\langle l_1 \mid \langle l_2 \mid \mu \rangle \rangle \equiv \langle l_2 \mid \langle l_1 \mid \mu \rangle \rangle$ (without needing subtype constraints or polymorphic label flags).

Similarly, duplicate labels can easily arise due to type instantiation. For example, a *catch* handler for exceptions can have type:

$$\mathit{catch} : \forall \mu\ \alpha.\ (() \rightarrow \langle \mathsf{exn} \mid \mu \rangle\ \alpha) \rightarrow (\mathit{string} \rightarrow \mu\ \alpha) \rightarrow \mu\ \alpha$$

where *catch* takes an action that can raise exceptions, and a handler function that is called when an exception is caught. Suppose though an exception handler raises itself an exception, and has type $h : \forall \mu.\ \mathit{string} \rightarrow \langle \mathsf{exn} \mid \mu \rangle\ \mathit{int}$. The application *catch action h* is then explicitly typed as:

$$\Lambda \mu.\ \mathit{catch}[\langle \mathsf{exn} \mid \mu \rangle, \mathit{int}]\ \mathit{action}\ h[\mu]$$

where the type application gives rise to the type:

$$\mathit{catch}[\langle \mathsf{exn} \mid \mu \rangle, \mathit{int}] : (() \rightarrow \langle \mathsf{exn}, \mathsf{exn} \mid \mu \rangle\ \mathit{int}) \rightarrow (\mathit{string} \rightarrow \langle \mathsf{exn} \mid \mu \rangle\ \mathit{int}) \rightarrow \langle \mathsf{exn} \mid \mu \rangle\ \mathit{int}$$

naturally leading to duplicate labels in the type. As we will see, simple row types also correspond naturally to the shape of the runtime evidence vectors that we introduce in Section 4.1 (where duplicated labels correspond to nested handlers).

3.2 Operations

We assume the every effect l has a unique set of operations op_1 to op_n with a signature *sig* that gives every operation its input and output types, $op_i : \forall \overline{\alpha}_i.\ \sigma_i \rightarrow \sigma_i'$. There is a global map Σ that maps each effect l to its signature. Since we assume that each *op* is uniquely named, we use the notation $op : \forall \overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2 \ \in \Sigma(l)$ to denote the type of *op* that belongs to effect l, and also $op \ \in \Sigma(l)$ to signify that *op* is part of effect l.

Note that we allow operations to be polymorphic. Therefore perform $op\ \overline{\sigma}\ v$ contains the instantiation types $\overline{\sigma}$ which are passed to the operation clause f in the evaluation rule for (*perform*) (Figure 2). This means that operations can be used polymorphically, but the handling clause itself must be polymorphic in the operation types (and use them as abstract types).

3.3 Quantification and Equivalence to the Untyped Dynamic Semantics

We would like the property that if we do type erasure on the newly defined System F^ϵ we have the same semantics as with the untyped dynamic semantics, i.e.

Theorem 1. (*System F^ϵ has untyped dynamic semantics*)
If $e_1 \longrightarrow e_2$ in System F^ϵ, then either $e_1^* \longrightarrow e_2^*$ or $e_1^* \;=\; e_2^*$.

where e^* stands for the term e with all types, type abstractions, and type applications removed. This seem an obvious property but there is a subtle interaction with quantification. Suppose we (wrongly) allow quantification over expressions instead of values, like $\Lambda\alpha.\ e$, then consider:

$$h \;=\; \{\ tick \rightarrow \lambda x\!:\!()\ k\!:\!(() \rightarrow \langle\rangle\ int).\ 1 \;+\; k\,()\ \}$$
$$\text{handle}\ h\,((\lambda x\!:\!\forall\alpha.\ int.\ x[int] \;+\; x[bool])\ (\Lambda\alpha.\ tick\,();\ 1))$$

In the typed semantics, this would evaluate the argument x at each instantiation (since the whole $\Lambda\alpha.\ tick\,();\ 1$ is passed as a value), resulting in 4. On the other hand, if we do type erasure, the untyped dynamic semantics evaluates to 3 instead (evaluating the argument before applying). Not only do we lose untyped dynamic semantics, but we also break parametricity (as we can observe instantiations). So, it is quite important to only allow quantification over values, much like the ML value restriction [Kammar and Pretnar 2017; Pitts 1998; Wright 1995]. In the proof of Theorem 1 we use in particular the following (seemingly obvious) Lemma:

Lemma 1. (*Type erasure of values*)
If v is a value in System F^ϵ then v^* is a value in λ^ϵ.

Not all systems in the literature adhere to this restriction; for example Biernacki et al. [2017] and Leijen [2016] allow quantification over expressions as $\Lambda\alpha.\ e$, where both ensure soundness of the effect type system by disallowing type abstraction over effectful expressions. However, we believe that this remains a risky affair since Lemma 1 does not hold; and thus a typed evaluation may take more reduction steps than the type-erased term, i.e. seemingly shared argument values may be computed more than once.

3.4 Type Rules for System F^ϵ

Figure 5 defines the typing rules for System F^ϵ. The rules are of the form $\Gamma; w \;\vdash\; e \,:\, \sigma \mid \epsilon \;\rightsquigarrow e'$ for expressions where the variable context Γ and the effect ϵ are given ($\uparrow$), and σ is synthesized ($\downarrow$). The gray parts define the evidence translation which we describe in Section 4 and these can be ignored for now. Values are not effectful, and are typed as $\Gamma \vdash_{\mathsf{val}}\ v \,:\, \sigma \;\rightsquigarrow v'$. Since the effects are inherited, the lambda needs an effect annotation that is passed to the body derivation (ABS). In the rule APP we use standard equality between types and require that all effects match. The VAL rule goes from a value to an expression (opposite of ABS) and allows any inherited effect. The HANDLER rule takes an action with effect $\langle l \mid \epsilon\rangle$ and handles l leaving effect ϵ. The HANDLE rule is similar, but is defined over an expression e and types e under an extended effect $\langle l \mid e\rangle$ in the premise.

In the Appendix, there are type rules for evaluation contexts where $\Gamma \vdash_{\mathsf{ec}}\ \mathsf{E} \,:\, \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ signifies that a context E can be typed as a function from a term of type σ_1 to σ_2 where the resulting expression has effect ϵ. These rules are not needed to check programs but are very useful in proofs and theorems. In particular,

Lemma 2. (*Evaluation context typing*)
If $\varnothing \vdash_{\mathsf{ec}}\ \mathsf{E} \,:\, \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ and $\varnothing \;\vdash\; e \,:\, \sigma_1 \mid \langle\lceil \mathsf{E}\rceil^l \mid \epsilon\rangle$, then $\varnothing \;\vdash\; \mathsf{E}[e] \,:\, \sigma_2 \mid \epsilon$.

where $\lceil \mathsf{E}\rceil^l$ extracts all labels l from a context in reverse order:

$$\lceil \mathsf{F}_0 \cdot \text{handle}\ h_1 \cdot \mathsf{F}_1 \cdot \ldots \cdot \text{handle}\ h_n \cdot \mathsf{F}_n\rceil^l \;=\; \langle l_n,\ \ldots,\ l_1\rangle$$

$$\underset{\uparrow}{\Gamma};\underset{\uparrow\,F^{ev}}{w} \vdash \underset{\uparrow\,F^{\epsilon}}{e} : \underset{\downarrow}{\sigma} \mid \underset{\uparrow}{\epsilon} \rightsquigarrow \underset{\downarrow\,F^{ev}}{e'} \qquad \underset{\uparrow}{\Gamma} \vdash_{\mathsf{val}} \underset{\uparrow\,F^{\epsilon}}{v} : \underset{\downarrow}{\sigma} \rightsquigarrow \underset{\downarrow\,F^{ev}}{v'} \qquad \underset{\uparrow}{\Gamma} \vdash_{\mathsf{ops}} \underset{\uparrow\,F^{\epsilon}}{h} : \underset{\downarrow}{\sigma} \mid \underset{\downarrow}{l} \mid \underset{\uparrow}{\epsilon} \rightsquigarrow \underset{\downarrow\,F^{ev}}{h'}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash_{\mathsf{val}} x : \sigma \rightsquigarrow x}\,[\textsc{var}] \qquad \frac{\Gamma, x : \sigma_1;\ z \vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e' \quad \text{fresh } z}{\Gamma \vdash_{\mathsf{val}} \lambda^{\epsilon}\, x : \sigma_1.\ e : \sigma_1 \to^{\epsilon} \sigma_2 \rightsquigarrow \lambda^{\epsilon} z : \mathsf{evv}\ \epsilon, x : \lceil \sigma_1 \rceil.\ e'}\,[\textsc{abs}]$$

$$\frac{\Gamma \vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'}{\Gamma; w \vdash v : \sigma \mid \epsilon \rightsquigarrow v'}\,[\textsc{val}] \qquad \frac{\Gamma \vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'}{\Gamma \vdash_{\mathsf{val}} \Lambda \alpha^k.\ v : \forall \alpha^k.\ \sigma \rightsquigarrow \Lambda \alpha^k.\ v'}\,[\textsc{tabs}]$$

$$\frac{\Gamma; w \vdash e_1 : \sigma_1 \to \epsilon\ \sigma \mid \epsilon \rightsquigarrow e_1' \qquad \Gamma; w \vdash e_2 : \sigma_1 \mid \epsilon \rightsquigarrow e_2'}{\Gamma; w \vdash e_1\ e_2 : \sigma \mid \epsilon \rightsquigarrow e_1'\ w\ e_2'}\,[\textsc{app}] \qquad \frac{\Gamma; w \vdash e : \forall \alpha^k.\ \sigma_1 \mid \epsilon \rightsquigarrow e' \quad \vdash_{\mathsf{wf}} \sigma : k}{\Gamma; w \vdash e[\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow e'[\lceil \sigma \rceil]}\,[\textsc{tapp}]$$

$$\frac{op : \forall \overline{\alpha}.\ \sigma_1 \to \sigma_2 \in \Sigma(l) \qquad \overline{\alpha} \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash_{\mathsf{val}} \mathsf{perform}^{\epsilon}\ op\ \overline{\sigma} : \sigma_1[\overline{\alpha}{:=}\overline{\sigma}] \to \langle l \mid \epsilon \rangle\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightsquigarrow \mathsf{perform}^{\epsilon}\ op\ \overline{\sigma}}\,[\textsc{perform}]$$

$$\frac{op_i : \forall \overline{\alpha}_i.\ \sigma_1 \to \sigma_2 \in \Sigma(l) \qquad \overline{\alpha} \not\cap \mathsf{ftv}(\epsilon\ \sigma) \qquad \Gamma \vdash_{\mathsf{val}} f_i : \forall \overline{\alpha}.\ \sigma_1 \to \epsilon\ ((\sigma_2 \to \epsilon\ \sigma) \to \epsilon\ \sigma) \rightsquigarrow f_i'}{\Gamma \vdash_{\mathsf{ops}} \{\ op_1 \to f_1, \ldots, op_n \to f_n\ \} : \sigma \mid l \mid \epsilon \rightsquigarrow \{\ op_i \to f_i'\ \}}\,[\textsc{ops}]$$

$$\frac{\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'}{\Gamma \vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h : (() \to \langle l \mid \epsilon \rangle\ \sigma) \to \epsilon\ \sigma \rightsquigarrow \mathsf{handler}^{\epsilon}\ h'}\,[\textsc{handler}]$$

$$\frac{\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h' \qquad \Gamma; \langle\!\langle l : (m, h') \mid w \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'}{\Gamma;\ w \vdash \mathsf{handle}^{\epsilon}\ h\ e : \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}^{w}_{m}\ h'\ e'}\,[\textsc{handle}]$$

Fig. 5. Type Rules for System F^{ϵ} combined with type directed evidence translation to F^{ev} (in gray.)

with l_i corresponding to each h_i (for any $op \in h_i$, $op \in \Sigma(l_i)$). The above lemma shows we can plug well-typed expressions in a suitable context. The next lemma uses this to show the correspondence between the dynamic evaluation context and the static effect type:

Lemma 3. (*Effect corresponds to the evaluation context*)
If $\varnothing \vdash \mathsf{E}[e] : \sigma \mid \epsilon$ then there exists σ_1 such that $\varnothing \vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \to \sigma \mid \epsilon$, and $\varnothing \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^{l} \mid \epsilon \rangle$.

Here we see that the rules guarantee that exactly the effects $\lceil \mathsf{E} \rceil^{l}$ in e are handled by the context E.

3.5 Progress and Preservation

We establish two essential lemmas about the meaning of effect types. First, in any well-typed total System F^{ϵ} expression, all operations are handled (and thus, evaluation cannot get stuck):

Lemma 4. (*Well typed operations are handled*)
If $\varnothing \vdash \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v] : \sigma \mid \langle\rangle$ then E has the form $\mathsf{E}_1 \cdot \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E}_2$ with $op \notin \mathsf{bop}(\mathsf{E}_2)$ and $op \to f \in h$.

Moreover, effect types are meaningful in the sense that an effect type fully reflects all possible effects that may happen during evaluation:

$$
\begin{array}{llll@{\qquad}llll}
\multicolumn{4}{l}{\text{Expressions}} & \multicolumn{4}{l}{\text{Values}} \\
e & ::= & v & \text{(value)} & v & ::= & x & \text{(variables)} \\
 & | & e[\sigma] & \text{(type application)} & & | & \lambda^{\epsilon} z : \mathsf{evv}\ \epsilon, x : \sigma.\ e & \text{(evidence abstraction)} \\
 & | & e\ w\ e & \text{(evidence application)} & & | & \Lambda \alpha^{k}.\ v & \text{(type abstraction)} \\
 & | & \mathsf{handle}^{w}_{m}\ h\ e & \text{(handler instance)} & & | & \mathsf{handler}^{\epsilon}\ h & \text{(effect handler)} \\
 & & & & & | & \mathsf{perform}^{\epsilon}\ op\ \overline{\sigma} & \text{(operation)} \\
 & & & & & | & \mathsf{guard}^{w}\ \mathsf{E}\ \sigma & \text{(guarded abstraction)}
\end{array}
$$

$$
\begin{array}{lll}
(app) & (\lambda^{\epsilon} z : \mathsf{evv}\ \epsilon, x : \sigma.\ e)\ w\ v & \longrightarrow\ e[z {:=} w, x {:=} v] \\
(tapp) & (\Lambda \alpha^{k}.\ v)\ [\sigma] & \longrightarrow\ v[\alpha {:=} \sigma] \\
(handler) & (\mathsf{handler}^{\epsilon}\ h)\ w\ v & \longrightarrow\ \mathsf{handle}^{w}_{m}\ h\ (v\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle\ ()) \\
 & & \qquad \text{where}\quad m \text{ is unique and } h \in \Sigma(l) \\
(return) & \mathsf{handle}^{w}_{m}\ h \cdot v & \longrightarrow\ v \\
(perform) & \mathsf{handle}^{w}_{m}\ h \cdot \mathsf{E} \cdot \mathsf{perform}^{\epsilon}\ op\ \overline{\sigma}\ w'\ v & \longrightarrow\ f[\overline{\sigma}]\ w\ v\ w\ k \quad \text{iff } op \notin \mathsf{bop}(\mathsf{E}) \wedge (op \to f) \in h \\
 & & \qquad \text{where}\quad op : \forall \overline{\alpha}.\ \sigma_1 \to \sigma_2 \in \Sigma(l) \\
 & & \qquad\qquad k = \mathsf{guard}^{w}\ (\mathsf{handle}^{w}_{m}\ h \cdot \mathsf{E})\ (\sigma_2[\overline{\alpha} {:=} \overline{\sigma}]) \\
(guard) & (\mathsf{guard}^{w}\ \mathsf{E}\ \sigma)\ w\ v & \longrightarrow\ \mathsf{E}[v]
\end{array}
$$

Fig. 6. System F^{ev} Typed operational semantics with evidence

Lemma 5. (*Effects types are meaningful*)
If $\varnothing \vdash \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v] : \sigma \mid \epsilon$ with $op \notin \mathsf{bop}(\mathsf{E})$, then $op \in \Sigma(l)$ and $l \in \epsilon$, i.e. effect types cannot be discarded without a handler.

Using these Lemmas, we can show that evaluation can always make progress and that the typings are preserved during evaluation.

Theorem 2. (*Progress*)
If $\varnothing \vdash e_1 : \sigma \mid \langle\rangle$ then either e_1 is a value, or $e_1 \longmapsto e_2$.

Theorem 3. (*Preservation*)
If $\varnothing \vdash e_1 : \sigma \mid \langle\rangle$ and $e_1 \longmapsto e_2$, then $\varnothing \vdash e_2 : \sigma \mid \langle\rangle$.

4 POLYMORPHIC EVIDENCE TRANSLATION TO SYSTEM F^{ev}

Having established a sound explicitly typed core calculus, we can now proceed to do evidence translation. The goal of evidence translation is to pass down evidence *ev* of the handlers in the evaluation context to place where operations are performed. This will in turn enable other optimizations (as described in Section 6) since we can now locally inspect the evidence instead of searching in the dynamic evaluation context.

Following Brachthäuser and Schuster [2017], we represent evidence *ev* for an effect l as a pair (m, h), consisting of a unique *marker* m and its corresponding handler implementation h. The markers uniquely identify each handler frame in the context which is now marked as $\mathsf{handle}_m\ h$. The reason for introducing the separate handler h construct is now apparent: it *instantiates* $\mathsf{handle}_m\ h$ frames with a unique m. This representation of evidence allows for two important optimizations: (1) We can change the operational rule for perform to directly yield to a particular handler identified by m (instead of needing to search for the innermost one), shown in Section 5.1, and (2) It allows local inspection of the actual handler h so we can evaluate tail resumptive operations in place, shown in Section 6.

However, passing the evidence down to each operation turns out to be surprisingly tricky to get right and we took quite a few detours before arriving at the current solution. At first, we thought we could represent evidence for each label l in the effect of a function as separate argument ev_l. For example,

$$f_1 \ : \ \forall\mu.\ int \rightarrow \langle l_1 \mid \mu\rangle\ int \ = \ \Lambda\mu.\ \lambda x.\ \mathsf{perform}\ op_1\ x$$

would be translated as:

$$f_1 \ : \ \forall\mu.\ \mathsf{ev}\ l \rightarrow int \rightarrow \langle l_1 \mid \mu\rangle\ int \ = \ \Lambda\mu.\ \lambda ev.\ \lambda x.\ \mathsf{perform}\ op_1\ ev\ x$$

This does not work though as type instantiation can now cause the runtime representation to change as well! For example, if we instantiate μ to $\langle l_2\rangle$ as $f[\langle l_2\rangle]$ the type becomes $int \rightarrow \langle l_1, l_2\rangle\ int$ which now takes *two* evidence parameters. Even worse, such instantiation can be inside arbitrary types, like a list of such functions, where we cannot construct evidence transformers in general.

Another design that does not quite work is to regard evidence translation as an instance of qualified types [Jones 1992] and use a dictionary passing translation. In essence, in the theory of qualified types, the qualifiers are scoped over monomorphic types, which does not fit well with effect handlers. Suppose we have a function *foo* with a qualified evidence types as:

$$foo \ : \ Ev\ l_1 \Rightarrow \ (int \rightarrow \langle l_1\rangle\ int) \rightarrow \langle l_1\rangle\ int$$

Note that even though *foo* is itself qualified, the argument it takes is a plain function $int \rightarrow \langle l_1\rangle\ int$ *and has already resolved its own qualifiers*. That is too eager for our purposes. For example, if we apply $foo\ (f_1[\langle\rangle])$, under dictionary translation we would get $foo\ ev_1\ (f_1[\langle\rangle]\ ev_1)$. However, it may well be that *foo* itself applies f_1 under a new handler for the l_1 effect and thus needs to pass different evidence than ev_1! Effectively, dictionary translation may partially apply functions with their dictionaries which is not legal for handler evidence. The qualified type we really require for *foo* uses higher-ranked qualifiers, something like $Ev\ l_1 \Rightarrow \ (Ev\ l_1 \Rightarrow \ int \rightarrow \langle l_1\rangle\ int) \rightarrow \langle l_1\rangle\ int$.

4.1 Evidence Vectors

The design we present here instead passes all evidence as a single *evidence vector* to each (effectful) function: this keeps the runtime representations stable under type instantiation, and we can ensure syntactically that functions are never partially applied to evidence.

Figure 6 defines our target language F^{ev} as an explicitly typed calculus with evidence passing. All applications are now of the form $e_1\ w\ e_2$ where we always pass an evidence vector w with the original argument e_2. Therefore, all lambdas are of the form $\lambda^{\epsilon}\ z : \mathsf{evv}\ \epsilon, x : \sigma.\ e$ and always take an evidence vector z besides their regular parameter x. We also extend application forms in the evaluation context to take evidence parameters. The double arrow notation is used do denote the type of these "tupled" lambdas:

$$\sigma_1 \Rightarrow \ \epsilon\ \sigma_2 \ \doteq \ \mathsf{evv}\ \epsilon \rightarrow \sigma_1 \rightarrow \epsilon\ \sigma_2$$

During evidence translation, every effect type ϵ on an arrow is translated to an explicit runtime evidence vector of type $\mathsf{evv}\ \epsilon$, and we translate type annotations as:

$$\begin{array}{llll}
\lceil\cdot\rceil : \sigma \rightarrow \sigma & & & \\
\lceil\forall\alpha.\sigma\rceil = \forall\alpha.\ \lceil\sigma\rceil & & \lceil\alpha\rceil & = \alpha \\
\lceil\tau_1 \rightarrow \epsilon\ \tau_2\rceil = \lceil\tau_1\rceil \Rightarrow \ \epsilon\ \lceil\tau_2\rceil & & \lceil c\ \tau_1 \ldots \tau_n\rceil & = c\ \lceil\tau_1\rceil \ldots \lceil\tau_n\rceil
\end{array}$$

Evidence vectors are essentially a map from effect labels to evidence. During evaluation we need to be able to select evidence from an evidence vector, and to insert new evidence when a handler is instantiated, and we define the following three operations:

$\langle\!\langle\ \rangle\!\rangle$	$: \mathsf{evv}\ \langle\rangle$	(empty evidence vector)
$\_.l$	$: \forall\mu.\ \mathsf{evv}\ \langle l \mid \mu\rangle \rightarrow \mathsf{ev}\ l$	(select evidence from a vector)
$\langle\!\langle l : ev \mid w\rangle\!\rangle$	$: \forall\mu.\ \mathsf{ev}\ l \rightarrow \mathsf{evv}\ \mu \rightarrow \mathsf{evv}\ \langle l \mid \mu\rangle$	(evidence insertion)

Where we assume the following two laws that relate selection and insertion:

$$\begin{array}{lll} \langle\!\langle l : ev \mid w \rangle\!\rangle.l = & ev & \\ \langle\!\langle l' : ev \mid w \rangle\!\rangle.l = & w.l & \text{iff } l \neq l' \end{array}$$

Later we want to be able to select evidence from a vector with a constant offset instead of searching for the label, so we are going to keep them in a canonical form ordered by the effect types l, written as $\langle\!\langle l_1 : ev_1, \ldots, l_n : ev_n \rangle\!\rangle$ with every $l_i \leqslant l_{i+1}$. We can now define $\langle\!\langle l : ev \mid w \rangle\!\rangle$ as notation for a vector where evidence ev was inserted in an ordered way, i.e.

$$\begin{array}{lll} \langle\!\langle l : \_ \mid \_ \rangle\!\rangle \;:\; \forall \mu.\; ev\; l \rightarrow \text{evv}\; \mu \rightarrow \text{evv}\; \langle l \mid \mu \rangle & & \\ \langle\!\langle l : ev \mid \langle\!\langle\rangle\!\rangle \rangle\!\rangle & = \langle\!\langle l : ev \rangle\!\rangle & \\ \langle\!\langle l : ev \mid \langle\!\langle l' : ev',\; w \rangle\!\rangle \rangle\!\rangle & = \langle\!\langle l' : ev',\; \langle\!\langle l : ev \mid w \rangle\!\rangle \rangle\!\rangle & \text{iff } l > l' \\ \langle\!\langle l : ev \mid \langle\!\langle l' : ev',\; w \rangle\!\rangle \rangle\!\rangle & = \langle\!\langle l : ev,\; l' : ev',\; w \rangle\!\rangle & \text{iff } l \leqslant l' \end{array}$$

Note how the dynamic representation as vectors of labeled evidence nicely corresponds to the static effect row-types, in particular with regard to duplicate labels, which correspond to nested handlers at runtime. Here we see why we cannot swap the position of equal effect labels as we need the evidence to correspond to their actual order in the evaluation context. Inserting all evidence in a vector w_1 into another vector w_2 is defined inductively as:

$$\begin{array}{ll} \langle\!\langle \langle\!\langle\rangle\!\rangle \mid w_2 \rangle\!\rangle & = w_2 \\ \langle\!\langle \langle\!\langle l : ev, w_1 \rangle\!\rangle \mid w_2 \rangle\!\rangle & = \langle\!\langle l : ev \mid \langle\!\langle w_1 \mid w_2 \rangle\!\rangle \rangle\!\rangle \end{array}$$

and evidence selection can be defined as:

$$\begin{array}{lll} \_.l \;:\; \forall \mu.\; \text{evv}\; \langle l \mid \mu \rangle \rightarrow \text{ev}\; l & & \\ \langle\!\langle l : ev, \_ \rangle\!\rangle.l & = ev & \\ \langle\!\langle l' : ev, w \rangle\!\rangle.l & = w.l & \text{iff } l \neq l' \\ \langle\!\langle\;\rangle\!\rangle.l & = (\textit{cannot happen}) & \end{array}$$

4.2 Evidence Translation

The evidence translation is already defined in Figure 5, in the gray parts of the rules. The full rules for expressions are of the form $\Gamma; w \;\vdash\; e : \sigma \mid \epsilon \;\rightsquigarrow e'$ where given a context Γ, the expression e has type σ with effect ϵ. The rules take the current evidence vector w for the effect ϵ, of type evv ϵ, and translate to an expression e' of System F^{ev}.

The translation in itself is straightforward where we only need to ensure extra evidence is passed during applications and abstracted again on lambdas. The ABS rule abstracts fully over all evidence in a function as $\lambda^{\epsilon} z : \text{evv}\; \epsilon,\; x : \sigma_1.\; e'$, where the evidence vector is abstracted as z and passed to its premise. Note that since we are translating, z is not part of Γ here (which scopes over F^{ϵ} terms). The type rules for F^{ev}, discussed below, do track such variables in the context. The dual of this is rule APP which passes the effect evidence w as an extra argument to every application as $e_1'\; w\; e_2'$.

To prove preservation and coherence of the translation, we also include a translation rule for handle, even though we assume these are internal. Otherwise there are no surprises here and the main difficulty lies in the operational rules, which we discuss in detail in Section 4.4.

To prove additional properties about the translated programs, we define a more restricted set of typing rules directly over System F^{ev} in Figure 9 of the form $\Gamma; w \;\Vdash\; e : \sigma \mid \epsilon$ (ignoring the gray parts), such that $\Gamma \;\vdash\; w : \text{evv}\; \epsilon$, and where the rules are a subset of the general typing rules for F^{ϵ}. Using this, we prove that the translation is sound:

Theorem 4. (*Evidence translation is Sound in* F^{ev})
If $\varnothing; \langle\!\langle\rangle\!\rangle \;\vdash\; e : \sigma \mid \langle\rangle \;\rightsquigarrow e'$ then $\varnothing; \langle\!\langle\rangle\!\rangle \;\Vdash\; e' : \lceil \sigma \rceil \mid \langle\rangle$.

4.3 Correspondence

The evidence translation maintains a strong correspondence between the effect types, the evidence vectors, and the evaluation contexts. To make this precise, we define the (reverse) extraction of all handlers in a context E as $\lceil \mathsf{E} \rceil$ where:

$$\begin{aligned}
\lceil \mathsf{F}_1 \cdot \mathsf{handle}_{m_1}\ h_1 \cdot \ldots \cdot \mathsf{F}_n \cdot \mathsf{handle}_{m_n}\ h_n \cdot \mathsf{F} \rceil \;&=\; \langle\!\langle l_n : (m_n, h_n) \mid \ldots \mid l_1 : (m_1, h_1) \mid \langle\!\langle\rangle\!\rangle \rangle\!\rangle \\
\lceil \mathsf{F}_1 \cdot \mathsf{handle}_{m_1}\ h_1 \cdot \ldots \cdot \mathsf{F}_n \cdot \mathsf{handle}_{m_n}\ h_n \cdot \mathsf{F} \rceil^{l} \;&=\; \langle l_n,\ \ldots,\ l_1 \rangle \\
\lceil \mathsf{F}_1 \cdot \mathsf{handle}_{m_1}\ h_1 \cdot \ldots \cdot \mathsf{F}_n \cdot \mathsf{handle}_{m_n}\ h_n \cdot \mathsf{F} \rceil^{m} \;&=\; \{ m_n,\ \ldots,\ m_1 \}
\end{aligned}$$

With this we can characterize the correspondence between the evaluation context and the evidence used at perform:

Lemma 6. (*Evidence corresponds to the evaluation context*)
If $\varnothing; w \Vdash \mathsf{E}[e] : \sigma \mid \epsilon$ then for some σ_1 we have $\varnothing; \langle\!\langle \lceil \mathsf{E} \rceil \mid w \rangle\!\rangle \Vdash e : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^{l} \mid \epsilon \rangle$, and $\varnothing; w \Vdash \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \epsilon$.

Lemma 7. (*Well typed operations are handled*)
If $\varnothing; \langle\!\langle\rangle\!\rangle \Vdash \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v] : \sigma \mid \langle\rangle$ then E has the form $\mathsf{E}_1 \cdot \mathsf{handle}^{w}_{m}\ h \cdot \mathsf{E}_2$ with $op \notin \mathrm{bop}(\mathsf{E}_2)$ and $op \rightarrow f\ \in h$.

These brings us to our main theorem which states that the evidence passed to an operation corresponds exactly to the innermost handler for that operation in the dynamic evaluation context:

Theorem 5. (*Evidence Correspondence*)
If $\varnothing; \langle\!\langle\rangle\!\rangle \Vdash \mathsf{E}[\mathit{perform}\ op\ \overline{\sigma}\ w\ v] : \sigma \mid \langle\rangle$ then E has the form $\mathsf{E}_1 \cdot \mathsf{handle}^{w'}_{m}\ h \cdot \mathsf{E}_2$ with $op \notin \mathrm{bop}(\mathsf{E}_2)$, $op \rightarrow f\ \in h$, and the evidence corresponds exactly to dynamic execution context such that $w.l\ =\ (m, h)$.

4.4 Operational Rules of System F^{ev}

The operational rules for System F^{ev} are defined in Figure 6. Since every application now always takes an evidence vector argument w the new (*app*) and (*handler*) rules now only reduce when both arguments are present (and the syntax does not allow for partial evidence applications).

The (*handler*) rule differs from System F^{ϵ} in two significant ways. First, it saves the current evidence in scope (passed as w) in the handle frame itself as handle^{w}_{m}. Secondly, the evidence vector it passes on to its action is now extended with its own unique evidence, as $\langle\!\langle l : (m, h) \mid w \rangle\!\rangle$.

In the (*perform*) rule, the operation clause $(op \rightarrow f)\ \in h$ is now translated itself, and we need to pass evidence to f. Since it takes two arguments, the operation payload x and its resumption k, the application becomes $(f[\overline{\sigma}]\ w\ x)\ w\ k$. The evidence we pass to f is the evidence of *the original handler context* saved as handle^{w} in the (*handler*) rule. In particular, we should not pass the evidence w' of the operation, as that is the evidence vector of the context in which the operation itself evaluates (and an extension of w). In contrast, we evaluate each clause under their original context and need the evidence vector corresponding to that. In fact, we can even ignore the evidence vector w' completely for now as we only need to use it later for implementing optimizations.

4.5 Guarded Context Instantiation and Scoped Resumptions

The definition of the resumption k in the (*perform*) rule differs quite a bit from the original definition in System F^{ϵ} (Figure 2), which was:

$$k \;=\; \lambda^{\epsilon} x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x$$

while the F^{ev} definition now uses:

$$k \;=\; \mathsf{guard}^{w}\ (\mathsf{handle}^{w}_{m}\ h \cdot \mathsf{E})\ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])$$

where we use a the new F^{ev} value term $\mathsf{guard}^w\ \mathsf{E}\ \sigma$. Since k has a regular function type, it now needs to take an extra evidence vector, and we may have expected a more straightforward extension without needing a new guard rule, something like:

$$k\ =\ \lambda^{\epsilon} z : \mathsf{evv}\ \epsilon, x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x$$

but then the question becomes what to do with that passed in evidence z? This is the point where it becomes more clear that resumptions are special and not quite like a regular lambda since they restore a captured context. In particular, *the context* E *that is restored has already captured the evidence of the original context in which it was captured (as* w*), and thus may not match the evidence of the context in which it is resumed (as* z*)*!

The new guarded application rule makes this explicit and only restores contexts that are resumed under the exact same evidence, in other words, only scoped resumptions are allowed:

$$(\mathsf{guard}^w\ \mathsf{E}\ \sigma)\ w\ v \longrightarrow \mathsf{E}[v]$$

If the evidence does not match, the evaluation is stuck in F^{ev}.

As an example of how this can happen, we return to our *evil* example in Section 2.2 which uses non-scoped resumptions to change the meaning of op_1. Since we are now in a typed setting, we modify the example to return a data type of results to make everything well-typed:

$$\begin{aligned}&\mathsf{data}\ res\ =\ \mathsf{again}\ :\ (() \to \langle one\rangle\ res) \to res\\ &\qquad\qquad \mid\ \mathsf{done}\ :\ int \to res\\ &\Sigma\ =\ \{\ one\ :\ \{\ op_1\ :\ () \to int\ \},\ evil\ :\ \{\ op_{evil}\ :\ () \to ()\ \}\ \}\end{aligned}$$

with the following helper definitions:

$$\begin{aligned}&h_1\ =\ \{\ op_1 \to \lambda x\ k.\ k\ 1\ \} && f\ (\mathsf{again}\ k) =\ \mathsf{handler}\ h_2\ (\lambda\_.\ k\ ());\ 0\\ &h_2\ =\ \{\ op_1 \to \lambda x\ k.\ k\ 2\ \} && f\ (\mathsf{done}\ x) =\ x\\ &h_{evil} = \ \{\ op_{evil} \to \lambda x\ k.\ (\mathsf{again}\ k)\ \}\end{aligned}$$

$$body\ =\ \mathsf{perform}\ op_1\ ();\ \mathsf{perform}\ op_{evil}\ ();\ \mathsf{perform}\ op_1\ ();\ \mathsf{done}\ 0$$

and where the main expression is evidence translated as:

$$\begin{aligned}&f\ (\mathsf{handler}\ h_1\ (\lambda\_.\ \mathsf{handler}\ h_{evil}\ (\lambda\_.\ body)))\\ &\rightsquigarrow f\ \langle\!\langle\rangle\!\rangle\ (\mathsf{handler}\ h_1\ \langle\!\langle\rangle\!\rangle\ (\lambda z, \_.\ \mathsf{handler}\ h_{evil}\ z\\ &\qquad (\lambda z : \mathsf{evv}\ \langle one, evil\rangle, \_.\ \mathsf{perform}\ op_1\ z\ ();\ \mathsf{perform}\ op_{evil}\ z\ ();\ \mathsf{perform}\ op_1\ z\ ();\ \mathsf{done}\ 0)))\end{aligned}$$

Starting evaluation in the translated expression, we can now derive:

$$\begin{aligned}&\longmapsto^* f\ \langle\!\langle\rangle\!\rangle \cdot \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_1}\ h_1 \cdot \mathsf{handle}^{w_1}_{m_2}\ h_{evil} \cdot (\square; \mathsf{perform}\ op_1\ w_2\ ();\ \mathsf{done}\ 0) \cdot \mathsf{perform}\ op_{evil}\ w_2\ ()\\ &\qquad \mathsf{with}\ w_1\ =\ \langle\!\langle one : (m_1, h_1)\rangle\!\rangle,\ w_2\ =\ \langle\!\langle evil : (m_2, h_{evil}), one : (m_1, h_1)\rangle\!\rangle\\ &\longmapsto^* f\ \langle\!\langle\rangle\!\rangle \cdot \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_1}\ h_1 \cdot (\lambda z\ x\ z\ k.\ (\mathsf{again}\ k))\ w_1\ ()\ w_1\ k\\ &\qquad \mathsf{with}\ k\ =\ \mathsf{guard}^{w_1}\ (\mathsf{handle}^{w_1}_{m_2}\ h_{evil} \cdot (\square; \mathsf{perform}\ op_1\ w_2\ ();\ \mathsf{done}\ 0))\\ &\longmapsto^* f\ \langle\!\langle\rangle\!\rangle \cdot \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_1}\ h_1 \cdot (\mathsf{again}\ k)\\ &\longmapsto^* f\ \langle\!\langle\rangle\!\rangle\ (\mathsf{again}\ k)\\ &\longmapsto\ \ \mathsf{handler}\ h_2\ \langle\!\langle\rangle\!\rangle\ (\lambda z\ \_.\ k\ z\ ())\\ &\longmapsto^*\ \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_3}\ h_2 \cdot k\ w_3\ () \qquad \mathsf{with}\ w_3\ =\ \langle\!\langle one : (m_3, h_2)\rangle\!\rangle\\ &=\ \ \ \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_3}\ h_2 \cdot \mathsf{guard}^{w_1}\ (\mathsf{handle}^{w_1}_{m_2}\ h_{evil} \cdot \square; \mathsf{perform}\ op_1\ w_2\ (); \mathsf{done}\ 0)\ w_3\ ()\end{aligned}$$

At this point, the guard rule gets stuck as we have captured the context originally under evidence w_1, but we try to resume with evidence w_3, and $w_1\ =\ \langle\!\langle one : (m_1, h_1)\rangle\!\rangle \neq \langle\!\langle one : (m_3, h_2)\rangle\!\rangle\ =\ w_3$.

If we allow the guarded context instantiation anyways we get into trouble when we try to perform op_1 again:

$$\begin{aligned}&\longmapsto\ \ \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_3}\ h_2 \cdot \mathsf{handle}^{w_1}_{m_2}\ h_{evil} \cdot (()\ ;\ \mathsf{perform}\ op_1\ w_2\ ();\ \mathsf{done}\ 0)\\ &\longmapsto^* \mathsf{handle}^{\langle\!\langle\rangle\!\rangle}_{m_3}\ h_2 \cdot \mathsf{handle}^{w_1}_{m_2}\ h_{evil} \cdot (\square;\ \mathsf{done}\ 0) \cdot \mathsf{perform}\ op_1\ w_2\ ()\end{aligned}$$

in that case the innermost handler for op_1 is now h_2 while the evidence $w_2.l$ is (m_1, h_1) and it no longer corresponds to the dynamic context! (and that would void our main correspondence Theorem 5 and in turn invalidate optimizations based on this).

4.6 Uniqueness of Handlers

It turns out that to guarantee coherence of the translation to plain polymorphic lambda calculus, as discussed in Section 5, we need to ensure that all m's in an evaluation context are always unique. This is a tricky property; for example, uniqueness of markers does *not* hold for arbitrary F^{ev} expressions: markers may be duplicated inside lambdas outside of the evaluation context, and we can also construct an expression manually with duplicated markers, e.g. $\mathsf{handle}_m^w \cdot \mathsf{handle}_m^w \cdot e$. However, we can prove that if we only consider initial F^{ev} expressions without handle_m^w, or any expressions reduced from that during evaluation, then it is guaranteed that all m's are always unique in the evaluation context – even though the (*handler*) rule introduces handle_m^w during evaluation, and the (*app*) rule may duplicate markers.

Definition 1. (*Handle-safe expressions*)
A *handle-safe* F^{ev} expression is a well-typed closed expression that either (1) contains no handle_m^w term; or (2) is itself reduced from a handle-safe expression.

Theorem 6. (*Uniqueness of handlers*)
For any handle-safe F^{ev} expression e, if $e = \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2 \cdot \mathsf{handle}_{m_2}^{w_2}\ h \cdot e_0$, then $m_1 \neq m_2$.

4.7 Preservation and Coherence

As exemplified above, the guard rule is also essential to prove the preservation of evidence typings under evaluation. In particular, we can show:

Theorem 7. (*Preservation of evidence typing*)
If $\varnothing; \langle\!\langle\rangle\!\rangle \Vdash e_1 : \sigma \mid \langle\rangle$ and $e_1 \longmapsto e_2$, then $\varnothing; \langle\!\langle\rangle\!\rangle \Vdash e_2 : \sigma \mid \langle\rangle$.

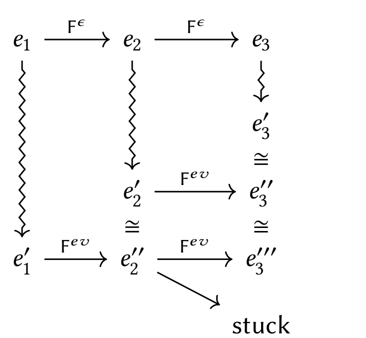

Fig. 7. Coherence

Even more important though is to show that our translation is *coherent*, that is, if we take an evaluation step in System F^ϵ, the evidence translated expression will take a similar step such that the resulting expression is again a translation of the reduced F^ϵ expression:

Theorem 8. (*Evidence translation is coherent*)
If $\varnothing; \langle\!\langle\rangle\!\rangle \vdash e_1 : \sigma \mid \langle\rangle \rightsquigarrow e_1'$ and $e_1 \longmapsto e_2$, and (due to preservation) $\varnothing; \langle\!\langle\rangle\!\rangle \vdash e_2 : \sigma \mid \langle\rangle \rightsquigarrow e_2'$, then exists a e_2'', such that $e_1' \longmapsto e_2''$ and $e_2'' \cong e_2'$.

Interestingly, the theorem states that the translated e_2' is only coherent under an equivalence relation $\cong$ relation to the reduced expression e_2', as illustrate in Figure 7. The reason that e_2' and e_2'' are not directly equal is due to guard expressions only being generated by reduction. In particular, if we have a F^ϵ reduction of the form:

$$\mathsf{handle}^\epsilon\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v \longrightarrow f\ \overline{\sigma}\ v\ k \quad \text{with } k = \lambda^\epsilon x\!:\!\sigma'.\ handle^\epsilon\ h \cdot \mathsf{E} \cdot x$$

then the translation takes the following F^{ev} reduction:

$$\mathsf{handle}_m^w\ h \cdot \mathsf{E}' \cdot \mathsf{perform}\ op\ \lceil\overline{\sigma}\rceil\ w'\ v' \longrightarrow f'\ \lceil\overline{\sigma}\rceil\ w\ v'\ w\ k' \text{ with } k' = \mathsf{guard}^w\ (handle_m^w\ h' \cdot \mathsf{E}')\ \sigma'$$

At this point the translation of $f\ \overline{\sigma}\ v\ k$ will be of the form $f'\ \lceil\overline{\sigma}\rceil\ w'\ v'\ w'\ k''$ where

$$k'' = \lambda^\epsilon z\!:\!\mathsf{evv}\ \epsilon, x.\ handle^\epsilon\ h' \cdot \mathsf{E}'' \cdot x$$

$$\begin{array}{llcl}\text{Expressions} & e & ::= & v \mid e\;e \mid e[\sigma] \\ \text{Values} & v & ::= & x \mid \lambda x:\sigma.\;e \mid \Lambda\alpha^k.\;v\end{array} \qquad \begin{array}{llcl}\text{Context} & \mathsf{F} & ::= & \square \mid \mathsf{F}\;e \mid v\;\mathsf{F} \mid \mathsf{F}\;[\sigma] \\ & \mathsf{E} & ::= & \mathsf{F}\end{array}$$

$$\begin{array}{ll}(app) & (\lambda^\epsilon x:\sigma.\;e)\;v \longrightarrow e[x{:=}v] \\ (tapp) & (\Lambda\alpha^k.\;v)\;[\sigma] \longrightarrow v[\alpha{:=}\sigma]\end{array}$$

$$\frac{x:\sigma \in \Gamma}{\Gamma \vdash_{\mathsf{F}} x^\sigma : \sigma}\;[\textsc{fvar}] \quad \frac{\Gamma \vdash_{\mathsf{F}} v : \sigma}{\Gamma \vdash_{\mathsf{F}} \Lambda\alpha^k.\;v : \forall\alpha^k.\;\sigma}\;[\textsc{ftabs}] \quad \frac{\Gamma, x:\sigma_1 \vdash_{\mathsf{F}} e : \sigma_2}{\Gamma \vdash_{\mathsf{F}} \lambda x:\sigma_1.\;e : \sigma_1 \rightarrow \sigma_2}\;[\textsc{fabs}]$$

$$\frac{\Gamma \vdash_{\mathsf{F}} e_1 : \sigma_1 \rightarrow \sigma \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash_{\mathsf{F}} e_1\;e_2 : \sigma}\;[\textsc{fapp}] \quad \frac{\Gamma \vdash_{\mathsf{F}} e : \forall\alpha^k.\;\sigma_1 \quad \vdash_{\mathsf{wf}} \sigma : k}{\Gamma \vdash_{\mathsf{F}} e[\sigma] : \sigma_1[\alpha{:=}\sigma]}\;[\textsc{ftapp}]$$

Fig. 8. System F^v: explicitly typed (higher kinded) polymorphic lambda calculus with strict evaluation. Types as in Figure 3 with no effects on the arrows.

i.e. the resumption k is translated as a regular lambda now and not as guard! Also, since E is translated now under a lambda, the resulting E'' differs in all evidence terms w in E' which will be z instead.

However, we know that if the resumption k' is ever applied, the argument is either exactly w, in which case $\mathsf{E}''[z{:=}w] = \mathsf{E}'$, or not equal to w in which case the evidence translated program gets stuck. This is captured by $\cong$ relation which is the smallest transitive and reflexive congruence among well-typed F^{ev} expressions, up to renaming of unique markers, satisfying the EQ-GUARD rule, which captures the notion of guarded context instantiation.

$$\frac{e[z{:=}w] \cong \mathsf{E}[x]}{\lambda^\epsilon z, x:\sigma.\;e \cong \mathsf{guard}^w\;\mathsf{E}\;\sigma}\;[\textsc{eq-guard}]$$

Now, is this definition of equivalence strong enough? Yes, because we can show that if two translated expressions are equivalent, then they stay equivalent under reduction (or get stuck):

Lemma 8. (*Operational semantics preserves equivalence, or gets stuck*)
If $e_1 \cong e_2$, and $e_1 \longrightarrow e_1'$, then either e_2 is stuck, or we have e_2' such that $e_2 \longrightarrow e_2'$ and $e_1' \cong e_2'$.

This establishes the full coherence of our evidence translation: if a translated expression reduces under F^{ev} without getting stuck, the final value is equivalent to the value reduced under System F^ϵ. Moreover, the only way an evidence translated expression can get stuck is if it uses non-scoped resumptions.

Note that the evidence translation never produces guard terms, so the translated expression can always take an evaluation step; however, subsequent evaluation steps may lead to guard terms, so after the first step, it may get stuck if a resumption is applied under a different handler context than where it was captured.

5 TRANSLATION TO CALL-BY-VALUE POLYMORPHIC LAMBDA CALCULUS

Now that we have a strong correspondence between evidence and the dynamic handler context, we can translate System F^{ev} expressions all the way to the call-by-value polymorphic lambda calculus, System F^v. This is important in practice as it removes all the special evaluation and type rules of algebraic effect handlers; this in turn means we can apply all the optimizations that regular compilers perform, like inlining, known case expansion, common sub-expression elimination etc. as usual with needing to keep track of effects. Moreover, it means we can compile directly to most

$$\underset{\uparrow}{\Gamma};\ \underset{\uparrow F^{ev}}{w};\ \underset{\uparrow F^{v}}{w'} \ \Vdash\ \underset{\uparrow F^{ev}}{e} : \underset{\downarrow}{\sigma} \mid \underset{\uparrow}{\epsilon} \ \rightsquigarrow \underset{\downarrow F^{v}}{e'} \qquad \underset{\uparrow}{\Gamma} \Vdash_{\mathsf{val}} \underset{\uparrow F^{ev}}{v} : \underset{\downarrow}{\sigma} \ \rightsquigarrow \underset{\downarrow F^{v}}{v'} \qquad \underset{\uparrow}{\Gamma} \Vdash_{\mathsf{ops}} \underset{\uparrow F^{ev}}{h} : \underset{\downarrow}{\sigma} \mid \underset{\downarrow}{l} \mid \underset{\uparrow}{\epsilon} \ \rightsquigarrow \underset{\downarrow F^{v}}{h'}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \Vdash_{\mathsf{val}} x : \sigma \rightsquigarrow x}\ [\text{MVAR}] \qquad \frac{(\Gamma, z : \mathsf{evv}\ \epsilon, x : \sigma_1);\ z;\ z \Vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e'}{\Gamma \Vdash_{\mathsf{val}} \lambda^{\epsilon} z : \mathsf{evv}\ \epsilon, x : \sigma_1.\ e : \sigma_1 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow \lambda z\ x.\ e'}\ [\text{MABS}]$$

$$\frac{\Gamma \Vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'}{\Gamma \Vdash_{\mathsf{val}} \Lambda\alpha.\ v : \forall\alpha.\ \sigma \rightsquigarrow \Lambda\alpha.\ v'}\ [\text{MTABS}] \qquad \frac{\Gamma \Vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'}{\Gamma; w; w' \Vdash v : \sigma \mid \epsilon \rightsquigarrow \mathit{pure}[\lfloor\sigma\rfloor]\ v'}\ [\text{MVAL}]$$

$$\frac{\Gamma; w; w' \Vdash e : \forall\alpha.\ \sigma_1 \mid \epsilon \rightsquigarrow e'}{\Gamma; w; w' \Vdash e[\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow e' \triangleright (\lambda x.\ \mathit{pure}\ (x[\lfloor\sigma\rfloor]))}\ [\text{MTAPP}]$$

$$\frac{\Gamma; w; w' \Vdash e_1 : \sigma_2 \Rightarrow \epsilon\ \sigma \mid \epsilon \rightsquigarrow e_1' \qquad \Gamma; w; w' \Vdash e_2 : \sigma_2 \mid \epsilon \rightsquigarrow e_2'}{\Gamma; w; w' \Vdash e_1\ w\ e_2 : \sigma \mid \epsilon \rightsquigarrow e_1' \triangleright (\lambda f.\ (e_2' \triangleright f\ w'))}\ [\text{MAPP}]$$

$$\frac{\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow e' \qquad \Gamma \Vdash_{\mathsf{val}} w : \mathsf{evv}\ \epsilon \rightsquigarrow w'}{\Gamma \Vdash_{\mathsf{val}} \mathsf{guard}^{w}\ \mathsf{E}\ \sigma_1 : \sigma_1 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow \mathit{guard}\ w'\ e'}\ [\text{MGUARD}]$$

$$\frac{op : \forall\overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2 \in \Sigma(l)}{\Gamma \Vdash_{\mathsf{val}} \mathsf{perform}^{\epsilon}\ op\ \overline{\sigma} : \sigma_1[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow \langle l \mid \epsilon\rangle\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightsquigarrow \mathit{perform}^{op}[\langle l \mid \epsilon\rangle, \lfloor\overline{\sigma}\rfloor]}\ [\text{MPERFORM}]$$

$$\frac{op_i : \forall\overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2 \in \Sigma(l) \qquad \overline{\alpha} \not\in \mathrm{ftv}(\epsilon\ \sigma) \qquad \Gamma \Vdash_{\mathsf{val}} f_i : \forall\overline{\alpha}.\ \sigma_1 \Rightarrow \epsilon\ (\sigma_2 \Rightarrow \epsilon\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow f_i'}{\Gamma \Vdash_{\mathsf{ops}} \{\ op_1 \rightarrow f_1, \ldots, op_n \rightarrow f_n\ \} : \sigma \mid l \mid \epsilon \rightsquigarrow \{\ op_1 \rightarrow f_1', \ldots, op_n \rightarrow f_n'\ \}}\ [\text{MOPS}]$$

$$\frac{\Gamma \Vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'}{\Gamma \Vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h : (() \Rightarrow \langle l \mid \epsilon\rangle\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow \mathit{handler}^{l}[\epsilon, \lfloor\sigma\rfloor]\ h'}\ [\text{MHANDLER}]$$

$$\frac{\Gamma \Vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h' \qquad \Gamma; \langle\!\langle l : (m, h) \mid w\rangle\!\rangle; \langle\!\langle l : (m, h') \mid w'\rangle\!\rangle \Vdash e : \sigma \mid \langle l \mid \epsilon\rangle \rightsquigarrow e'}{\Gamma; w; w' \Vdash \mathsf{handle}^{w}_{m}\ h\ e : \sigma \mid \epsilon \rightsquigarrow \mathit{prompt}[\epsilon, \lfloor\sigma\rfloor]\ m\ w'\ e'}\ [\text{MHANDLE}]$$

Fig. 9. Monadic translation to System-F^{v}. (($\triangleright$) is monadic bind).

common host platforms, like C or WebAssembly without needing a special runtime system to support capturing the evaluation context.

There has been previous work that performs such translation [Forster et al. 2019; Hillerström et al. 2017; Leijen 2017c], as well as various libraries that embed effect handlers as monads [Kammar et al. 2013; Wu et al. 2014] but without evidence translation such embeddings require either a sophisticated runtime system [Dolan et al. 2017 2015; Leijen 2017a], or are not quite as efficient as one might hope. The translation presented here allows for better optimization as it maintains evidence and has no special runtime requirements (it is just F!).

5.1 Translating to Multi-Prompt Delimited Continuations

As a first step, we show that we do not need explicit handle frames anymore that carry around the handler operations h, but can translate to multi-prompt delimited continuations [Brachthäuser and Schuster 2017]. Gunter, Rémy, and Riecke [1995] present the set and cupto operators for named prompts m with the following "control-upto" rule:

$$\mathsf{set}\ m\ \mathsf{in} \cdot \mathsf{E} \cdot \mathsf{cupto}\ m\ \mathsf{as}\ x\ \mathsf{in}\ e \longrightarrow (\lambda k.\ e)\ (\lambda x.\ \mathsf{E} \cdot x) \qquad m \notin \lceil \mathsf{E} \rceil^m$$

This effectively exposes "shallow" multi-prompts: for our purposes, we always need "deep" handling where the resumption evaluates under the same prompt again and we define:

$$\begin{array}{ll} \mathsf{prompt}_m\ e \doteq & \mathsf{set}\ m\ \mathsf{in}\ e \\ \mathsf{yield}_m\ f \doteq & \mathsf{cupto}\ m\ \mathsf{as}\ k\ \mathsf{in}\ (f\ (\lambda x.\ \mathsf{set}\ m\ \mathsf{in}\ (k\ x))) \end{array}$$

which gives us the following derived evaluation rule:

$$\mathsf{prompt}_m \cdot \mathsf{E} \cdot \mathsf{yield}_m\ f \longrightarrow f\ (\lambda x.\ \mathsf{prompt}_m \cdot \mathsf{E} \cdot x) \qquad m \notin \lceil \mathsf{E} \rceil^m$$

This is almost what we need, except that we need a multi-prompt that also takes evidence w into account and uses guard instead of a plain lambda to apply the resumption, i.e.:

$$\mathsf{prompt}_m^w \cdot \mathsf{E} \cdot \mathsf{yield}_m\ f \longrightarrow f\ \ w\ (\mathsf{guard}^w\ (\mathsf{prompt}_m^w \cdot \mathsf{E})) \qquad m \notin \lceil \mathsf{E} \rceil^m$$

Using the correspondence property (Theorem 5), we can use the evidence to inspect the handler locally at the perform and no longer need to keep it in the handle frame. We can now translate both perform $op\ v\ w$ and $\mathsf{handle}_m^w\ h$ in terms of the simpler yield_m and prompt_m^w, as:

$$\begin{array}{ll} \lceil \mathsf{handle}_m^w\ h \rceil & = \mathsf{prompt}_m^w \\ \lceil \mathsf{perform}\ op\ w'\ v \rceil & = \mathsf{yield}_m\ (\lambda w\ k.\ f\ \ w\ v\ w\ k) \quad \text{with } (m, h) = w'.l \text{ and } (op \to f) \in h \end{array}$$

We prove that this is a sound interpretation of effect handling:

Theorem 9. (*Evidence Translation to Multi-Prompt Delimited Continuations is Sound*)
For any evaluation step $e_1 \longmapsto e_2$ we also have $\lceil e_1 \rceil \longmapsto^* \lceil e_2 \rceil$.

Dolan et al. [2015] describe the multi-core OCaml runtime system with *split stacks*; in such setting we could use the pointers to a split point as markers m, and directly yield to the correct handler with constant time capture of the context. Even with such runtime, it may still be beneficial to do a monadic translation as well in order to be able to use standard compiler optimizations.

5.2 Monadic Multi-Prompt Translation to System F^v

With the relation to multi-prompt delimited control established, we can now translate F^{ev} to F^v in a monadic style, where we use standard techniques [Dyvbig et al. 2007] to implement the delimited control as a monad. Assuming notation for data types and matching, we can define a multi-prompt monad mon as follows:

$$\begin{array}{l} \mathsf{data\ mon}\ \mu\ \alpha\ \ = \\ \quad \mid \mathsf{pure}\ :\ \alpha \to \mathsf{mon}\ \mu\ \alpha \\ \quad \mid \mathsf{yield}\ :\ \forall \beta\ r\ \mu'.\ \mathsf{marker}\ \mu'\ r \to ((\beta \to \mathsf{mon}\ \mu'\ r) \to \mathsf{mon}\ \mu'\ r) \to (\mathsf{mon}\ \mu\ \beta \to \mathsf{mon}\ \mu\ \alpha) \to \mathsf{mon}\ \mu\ \alpha \end{array}$$

$$\begin{array}{ll} \mathit{pure}\ x & = \mathsf{pure}\ x \\ \mathit{yield}\ m\ \mathit{clause} & = \mathsf{yield}\ m\ \mathit{clause}\ \mathit{id} \end{array}$$

The pure case is used for value results, while the yield implements yielding to a prompt. A yield $m\,f\ \mathit{cont}$ has three arguments, (1) the marker $m\ :\ \mathsf{marker}\ \mu'\ r$ bound to a prompt in some context with effect μ' and *answer type* r; (2) the operation clause which receives the resumption (of type $\beta \to \mathsf{mon}\ \mu'\ r$) where β is the type of the operation result; and finally (3) the current continuation *cont* which is the runtime representation of the context. When binding a yield, the continuation keeps being extended until the full context is captured:

$$
\begin{array}{lcll}
(f \circ g)\ x & = & f\ (g\ x) & \text{(function composition)} \\
(f \bullet g)\ x & = & g\ x \rhd f & \text{(Kleisli composition)} \\
(\mathsf{pure}\ x) \rhd g & = & g\ x & \text{(monadic bind)} \\
(\mathsf{yield}\ m\ f\ cont) \rhd g & = & \mathsf{yield}\ m\ f\ (g \bullet cont) &
\end{array}
$$

The hoisting of yields corresponds closely to operation hoisting as described by Bauer and Pretnar [2015]. The *prompt* operation has three cases to consider:

$$
\begin{array}{lcll}
prompt & : & \forall \mu\ \alpha.\ \mathsf{marker}\ \mu\ \alpha \to \mathsf{evv}\ \mu \to \mathsf{mon}\ \langle l \mid \mu \rangle\ \alpha \to \mathsf{mon}\ \mu\ \alpha & \\
prompt\ m\ w\ (\mathsf{pure}\ x) & = & \mathsf{pure}\ x & \\
prompt\ m\ w\ (\mathsf{yield}\ m'\ f\ cont) & = & \mathsf{yield}\ m'\ f\ (prompt\ m\ w \circ cont) & \text{if } m \neq m' \\
prompt\ m\ w\ (\mathsf{yield}\ m\ f\ cont) & = & f\ w\ (guard\ w\ (prompt\ m\ w \circ cont)) &
\end{array}
$$

In the pure case, we are at the (*value*) rule and return the result as is. If we find a yield that yields to another prompt we also keep yielding but remember to restore our prompt when resuming in its current continuation, as ($prompt\ m\ w \circ cont$). The final case is when we yield to the prompt itself, in that case we are in the (*yield*) transition and continue with f passing the context evidence w and a guarded resumption[3].

The *guard* operation simply checks if the evidence matches and either continues or gets stuck:

$$
guard\ w_1\ cont\ w_2\ x = \ \mathsf{if}\ (w_1 == w_2)\ \mathsf{then}\ cont\ (pure\ x)\ \mathsf{else\ stuck}
$$

Note that due to the uniqueness property (Theorem 6) we can check the equality $w_1 == w_2$ efficiently by only comparing the markers m (and ignoring the handlers). The handle and perform can be translated directly into *prompt* and *yield* as shown in the previous section, where we generate a $handler^l$ definition per effect l, and a $perform^{op}$ for every operation:

$$
\begin{array}{lcl}
handler^l & : & \forall \mu\ \alpha.\ \mathsf{hnd}^l\ \mu\ \alpha \to \mathsf{evv}\ \mu \to (\mathsf{evv}\ \langle l \mid \mu \rangle \to () \to \mathsf{mon}\ \langle l \mid \mu \rangle\ \alpha) \to \mathsf{mon}\ \mu\ \alpha \\
perform^{op} & : & \forall \mu\ \overline{\alpha}.\ \mathsf{evv}\ \langle l \mid \mu \rangle \to \sigma_1 \to \mathsf{mon}\ \langle l \mid \mu \rangle\ \sigma_2 \qquad \text{with } op : \forall \overline{\alpha}.\ \sigma_1 \to \sigma_2\ \in \Sigma(l) \\
handler^l\ h\ w\ f & = & freshm\ (\lambda m \to prompt\ m\ w\ (f\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle\ ())) \\
perform^{op}\ w\ x & = & \mathsf{let}\ (m, h)\ =\ w.l\ \mathsf{in}\ yield\ m\ (\lambda w\ k.\ ((h.op)\ w\ x \rhd (\lambda f.\ f\ w\ k)))
\end{array}
$$

The *handler* creates a fresh marker and passes on new evidence under a new *prompt*. The *perform* can now directly select the evidence (m, h) from the passed evidence vector and *yield* to m directly. The function passed to yield is a bit complex since each operation clause is translated normally and has a nested monadic type, i.e. $\mathsf{evv}\ \epsilon \to \mathsf{mon}\ \epsilon\ ((\beta \to \mathsf{mon}\ \epsilon\ r) \to \mathsf{mon}\ \epsilon\ r)$, so we need to bind the first partial application to x before passing the continuation k.

Finally, for every effect signature $l\ :\ sig\ \in \Sigma$ we declare a corresponding data type $\mathsf{hnd}^l\ \epsilon\ r$ that is a record of operation clauses:

$$
\begin{array}{l}
l\ :\ \{\ op_1\ :\ \forall \overline{\alpha}_1.\ \sigma_1 \to \sigma_1',\ \ldots,\ op_n\ :\ \forall \overline{\alpha}_n.\ \sigma_n \to \sigma_n'\ \} \\
\ \rightsquigarrow \mathsf{data\ hnd}^l\ \mu\ r\ =\ \mathsf{hnd}^l\ \{\ op_1\ :\ \forall \overline{\alpha}_1.\ \mathsf{op}\ \sigma_1\ \sigma_1'\ \mu\ r,\ \ldots,\ op_n\ :\ \forall \overline{\alpha}_n.\ \mathsf{op}\ \sigma_n\ \sigma_n'\ \mu\ r\ \}
\end{array}
$$

where operations op are a type alias defined as:

$$
\mathsf{alias\ op}\ \alpha\ \beta\ \mu\ r\ \doteq\ \mathsf{evv}\ \mu \to \alpha \to \mathsf{mon}\ (\mathsf{evv}\ \mu \to (\mathsf{evv}\ \mu \to \beta \to \mathsf{mon}\ \mu\ r) \to \mathsf{mon}\ \mu\ r)
$$

[3]Typing the third case needs a dependent match on the markers $m'\ :\ \mathsf{marker}\ \mu'\ r$ and $m\ =\ \mathsf{marker}\ \mu\ \alpha$ where their equality implies $\mu\ =\ \mu'$ and $r\ =\ \alpha$. This can be done in Haskell with the *Equal* GADT, or encoded in F^v using explicit equality witnesses [Baars and Swierstra 2002].

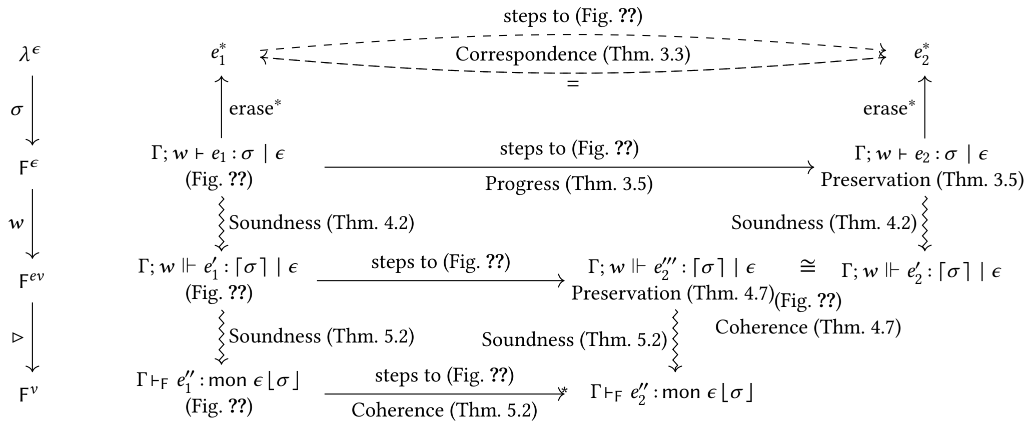

Fig. 10. An overview how the various theorems establish the relations between the different calculi

With these definitions in place, we can do a straightforward type directed translation from F^{ev} to F^{v} by just lifting all operations into the prompt monad, as shown in Figure 9. Types are translated by making all effectful functions monadic:

$$
\begin{array}{llll}
\lfloor \forall \overline{\alpha}.\ \sigma \rfloor = \forall \overline{\alpha}.\ \lfloor \sigma \rfloor & & \lfloor \sigma_1 \Rightarrow\ \epsilon\ \sigma_2 \rfloor = \mathsf{evv}\ \epsilon \rightarrow \lfloor \sigma_1 \rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor \sigma_2 \rfloor \\
\lfloor \alpha \rfloor = \alpha & & \lfloor c\ \sigma_1 \ldots \sigma_n \rfloor = c\ \lfloor \sigma_1 \rfloor \ldots \lfloor \sigma_n \rfloor
\end{array}
$$

We prove that these definitions are correct, and that the resulting translation is fully coherent, where a monadic program evaluates to the same result as a direct evaluation in F^{ev}.

Theorem 10. (*Monadic Translation is Sound*)
If $\varnothing; \langle\!\langle\rangle\!\rangle; \langle\!\langle\rangle\!\rangle \ \Vdash\ e : \sigma \mid \langle\rangle \ \rightsquigarrow e'$, then $\varnothing \vdash_{\mathsf{F}} e' : \mathsf{mon}\ \langle\rangle\ \lfloor \sigma \rfloor$.

Theorem 11. (*Coherence of the Monadic Translation*)
If $\varnothing; \langle\!\langle\rangle\!\rangle; \langle\!\langle\rangle\!\rangle \ \Vdash\ e_1 : \sigma \mid \langle\rangle \ \rightsquigarrow e_1'$ and $e_1 \longrightarrow e_2$, then also $\varnothing; \langle\!\langle\rangle\!\rangle; \langle\!\langle\rangle\!\rangle \ \Vdash\ e_2 : \sigma \mid \langle\rangle \ \rightsquigarrow e_2'$ where $e_1' \longrightarrow^* e_2'$.

Together with earlier results we establish full soundness and coherence from the original typed effect handler calculus F^{ϵ} to the evidence based monadic translation into plain call-by-value polymorphic lambda calculus F^{v}. See Figure 10 for how our theorems relate these systems to each other.

6 OPTIMIZATIONS

With a fully coherent evidence translation to plain polymorphic lambda calculus in hand, we can now apply various transformations in that setting to optimize the resulting programs.

6.1 Partially Applied Handlers

In the current $\mathit{perform}^{op}$ implementation, we yield with a function that takes evidence w to pass on to the operation clause f, as:

$$\lambda w\ k.\ ((h.op)\ w\ x \rhd (\lambda f.\ f\ w\ k))$$

However, the w that is going to be passed in is always that of the handle^w frame. When we instantiate the handle^w we can in principle map the w in advance over all operation clause so these can be partially evaluated over the evidence vector:

$$\begin{aligned} &\mathit{handler}^l\ h\ w\ f &&= \mathit{freshm}\ (\lambda m \rightarrow \mathit{prompt}\ m\ w\ (f\ \langle\!\langle l : (m, \mathit{pmap}^l\ h\ w \mid w\rangle\!\rangle\ ()))) \\ \\ &\mathit{pmap}^l\ (\text{hnd}^l\ f_1 \ldots f_n)\ w &&= \text{phnd}^l\ (\mathit{partial}\ w\ f_1) \ldots (\mathit{partial}\ w\ f_n) \\ &\mathit{partial}\ w\ f &&= \lambda x\ k.\ (f\ w\ x \triangleright (\lambda f'.\ f'\ w\ k)) \end{aligned}$$

The pmap^l function creates a new handler data structure phnd^l where every operation is now partially applied to the evidence which results in simplified type for each operation (as expressed by the pop type alias):

$$\text{alias pop}\ \alpha\ \beta\ \mu\ r \ \doteq\ \alpha \rightarrow (\beta \rightarrow \text{mon}\ \mu\ r) \rightarrow \text{mon}\ \mu\ r$$

The *perform* is now simplified as well as it no longer needs to bind the intermediate application:

$$\mathit{perform}^{op}\ w\ x = \text{let}\ (m, h) \ =\ w.l\ \text{in}\ \mathit{yield}\ m\ (\lambda k.\ (h.op)\ x\ k)$$

Finally, the *prompt* case where the marker matches no longer needs to pass evidence as well:

$$\begin{aligned} &\ldots \\ &\mathit{prompt}\ m\ w\ (\text{yield}\ m\ f\ \mathit{cont}) \ =\ f\ (\mathit{guard}\ w\ (\mathit{prompt}\ m\ w \circ \mathit{cont})) \end{aligned}$$

By itself, the impact of this optimization will be modest, just allowing inlining of evidence in f clauses, and inlining the monadic bind over the partial application, but it opens up the way to do tail resumptive operations in-place.

6.2 Evaluating Tail Resumptive Operations In Place

In practice, almost all important effects are tail-resumptive. The main exceptions we know of are asynchronous I/O (but that is dominated by I/O anyways) and the ambiguity effect for resuming multiple times. As such, we expect the vast majority of operations to be tail-resumptive, and being able to optimize them well is extremely important. We can extend the partially evaluated handler approach to optimize tail resumptions as well. First we extend the pop type to be a data type that signifies if an operation clause is tail resumptive or not:

$$\begin{aligned} \text{data pop}\ \alpha\ \beta\ \mu\ r\ &= \text{tail}\ :\ (\alpha \rightarrow \text{mon}\ \mu\ \beta) \rightarrow \text{pop}\ \alpha\ \beta\ \mu\ r \\ &\mid\ \text{normal}\ :\ (\alpha \rightarrow (\beta \rightarrow \text{mon}\ \mu\ r) \rightarrow \text{mon}\ \mu\ r) \rightarrow \text{pop}\ \alpha\ \beta\ \mu\ r \end{aligned}$$

The *partial* function now creates tail terms for any clause f that the compiler determined to be tail resumptive (i.e. of the form $\lambda x\ k.\ k\ e$) with $k \notin \text{fv}(e)$):

$$\begin{aligned} &\mathit{partial}\ w\ f = \text{tail}\ (\lambda x.\ (f\ w\ x \triangleright (\lambda f'.\ f'\ w\ \mathit{pure}))) && \text{if } f \text{ is tail resumptive} \\ &\mathit{partial}\ w\ f = \text{normal}\ (\lambda x\ k.\ (f\ w\ x \triangleright (\lambda f'.\ f'\ w\ k)) && \text{otherwise} \end{aligned}$$

Instead of passing in an "real" resumption function k, we just pass *pure* directly, leading to $\lambda x.\ (e \triangleright \mathit{pure})$ – and such clause we can now evaluate in-place without needing to yield and capture our resumption context explicitly. The perform^{op} can directly inspect the form of the operation clause from its evidence, and evaluate in place when possible:

$$\begin{aligned} \mathit{perform}^{op}\ w\ x\ = \text{let}\ (m, h) \ =\ w.l\ \text{in case}\ h.op\ \text{of}\ &\mid\ \text{tail}\ f \rightarrow f\ x \\ &\mid\ \text{normal}\ f \rightarrow \mathit{yield}\ m\ (f\ x) \end{aligned}$$

Ah, beautiful! Moreover, if a known handler is applied over some expression, regular optimizations like inlining and known-case evaluation, can often inline the operations fully. As everything has been translated to regular functions and regular data types without any special evaluation rules, there is no need for special optimization rules for handlers either.

6.3 Using Constant Offsets in Evidence Vectors

The $perform^{op}$ operation is now almost as efficient as a virtual method call for tail resumptive operations (just check if it is tail and do in indirect call), except that it still needs to do a dynamic lookup for the evidence as $w.l$.

The trick here is to take advantage of the canonical order of the evidence in a vector, where the location of the evidence in a vector of a closed effect type is fully determined. In particular, for any evidence vector w of type evv $\langle l \mid \epsilon \rangle$ where ϵ is closed, we can replace $w.l$ by a direct index $w[ofs]$ where $(l \text{ in } \epsilon) \;=\; ofs$, defined as:

$$\begin{array}{lll} l \text{ in } \langle\rangle & = 0 & \\ l \text{ in } \langle l' \mid \epsilon \rangle & = l \text{ in } \epsilon & \text{iff } l \leqslant l' \\ l \text{ in } \langle l' \mid \epsilon \rangle & = 1 + (l \text{ in } \epsilon) & \text{iff } l > l' \end{array}$$

This means for any functions with a closed effect, the offset of all evidence is constant. Only functions that are polymorphic in the effect tail need to index dynamically. It is beyond the scope of this paper to discuss this in detail but we believe that even in those cases we can index by a direct offset: following the same approach as TREX [Gaster and Jones 1996], we can use qualified types internally to propagate $(l \text{ in } \mu)$ constraints where the "dictionary" is simply the offset in the evidence vector (and these constraints can be hidden from the user as we can always solve them).

6.4 Reducing Continuation Allocation

The monadic translation still produces inefficiencies as it captures the continuation at every point where an operation may yield. For example, when calling an effectful function *foo*, as in $x \leftarrow foo\,();\;\; e$, the monadic translation produces a bind which takes an allocated lambda as a second argument to represent the continuation e explicitly, as $foo\,() \triangleright (\lambda x.\; e)$.

First of all, we can do a *selective* monadic translation [Leijen 2017c] where we leave out the binds if the effect of a function can be guaranteed to never produce a yield, e.g. total functions (like arithmetic), all effects provided by the host platform (like I/O), and all effects that are statically guaranteed to be tail resumptive (called *linear* effects). It turns out that many (leaf) functions satisfy this property so this removes the vast majority of binding.

Secondly, since we expect the vast majority of operations to be tail resumptive, almost always the effectful functions will not yield at all. It therefore pays off to always inline the bind operation and perform a direct match on the result and inline the continuation directly, e.g. expand to:

$$\begin{array}{ll} \text{case } foo\,() \text{ of} & \mid \text{yield } m\,f\;\, cont \rightarrow \text{yield } m\,f\;((\lambda x.\; e) \bullet cont) \\ & \mid \text{pure } x \rightarrow e \end{array}$$

This can be done very efficiently, and is close to what a C or Go programmer would write: returning a (yielding) flag from every function and checking the flag before continuing. Of course, this is also a dangerous optimization as it duplicates the expression e, and more research is needed to evaluate the impact of code duplication and finding a good inlining heuristic.

As a closing remark, the above optimization is why we prefer the monadic approach over continuation passing style (CPS). With CPS, our example would pass the continuation directly as $foo\,()\,(\lambda x.\; e)$. This style may be more efficient if one often yields (as the continuation is more efficiently composed versus bubbling up through the binds [Ploeg and Kiselyov 2014]) but it prohibits our optimization where we can inspect the result of *foo* (without knowing its implementation) and to only allocate a continuation if it is really needed.

7 RELATED WORK

In the paper, we compare to related work mostly inline when we discuss various aspects. In what follows, we briefly discuss more related work.

The work by Forster et al. [2019] is close to our work as it shows how delimited control, monads, and effect handlers can express each other. They show in particular a monadic semantics for effect handlers, but also prove that there does not exist a typed translation in their monomorphic setting. They conjecture a polymorphic translation may exist, and this paper proves that such translation is indeed possible.

Recent work by Biernacki et al. [2019] introduces labeled effect handlers where a handler can be referred to by name; the generative semantics with global labels is similar to our runtime markers m, but these labels are not guaranteed to be unique in the evaluation context (and they use the innermost handler in such case). Similar to this work they also distinguish between the generative handler (as handle_a), and the expression form handle_m (as handle_l).

Brachthäuser et al. [2020] use capability passing to perform operations directly on a specific handler [Brachthäuser and Schuster 2017; Brachthäuser et al. 2018]. This is also similar to the work of Zhang and Myers [2019] where handlers are passed by name as well. Both of these approaches can be viewed as programming with explicit evidence (capabilities) and we can imagine extending our calculus to allow a programmer to refer explicitly to the evidence (name) of a handler.

8 CONCLUSION

We have shown a full formal and coherent translation from a polymorphic core calculus for effect handlers (F^{ϵ}) to a polymorphic lambda calculus (F^{v}) based on evidence translation (through F^{ev}), and we have characterized the relation to multi-prompt delimited continuations precisely. Besides giving a new framework to reason about semantics of effect handlers, we are also hopeful that these techniques will be used to create efficient implementations of effect handlers in practice. Moreover, from a language design perspective, we expect that the restriction to scoped resumptions will be more widely adopted. As future work, we would like to implement these techniques and integrate them into real world languages.

REFERENCES

Arthur I. Baars, and S. Doaitse Swierstra. 2002. Typing Dynamic Typing. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*, 157–166. ICFP'02. Pittsburgh, PA, USA. doi:10.1145/581478.581494.

Andrej Bauer, and Matija Pretnar. 2014. An Effect System for Algebraic Effects and Handlers. *Logical Methods in Computer Science* 10 (4).

Andrej Bauer, and Matija Pretnar. 2015. Programming with Algebraic Effects and Handlers. *J. Log. Algebr. Meth. Program.* 84 (1): 108–123. doi:10.1016/j.jlamp.2014.02.001.

Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Dec. 2017. Handle with Care: Relational Interpretation of Algebraic Effects and Handlers. *Proc. ACM Program. Lang.* 2 (POPL'17 issue): 8:1–8:30. doi:10.1145/3158096.

Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Dec. 2019. Binders by Day, Labels by Night: Effect Instances via Lexically Scoped Handlers. *Proc. ACM Program. Lang.* 4 (POPL). Association for Computing Machinery, New York, NY, USA. doi:10.1145/3371116.

Jonathan Immanuel Brachthäuser, and Philipp Schuster. Oct. 2017. Effekt: Extensible Algebraic Effects in Scala. In *Scala'17*. Vancouver, CA.

Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. Oct. 2018. Effect Handlers for the Masses. *Proc. ACM Program. Lang.* 2 (OOPSLA). Association for Computing Machinery, New York, NY, USA. doi:10.1145/3276481.

Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. 2020. Effekt: Capability-Passing Style for Type- and Effect-Safe, Extensible Effect Handlers in Scala. *Journal of Functional Programming*. Cambridge University Press.

Lukas Convent, Sam Lindley, Conor McBride, and Craig McLaughlin. Jan. 2020. Doo Bee Doo Bee Doo. *In the Journal of Functional Programming*, January. To appear in the special issue on algebraic effects and handlers.

Stephen Dolan, Spiros Eliopoulos, Daniel Hillerström, Anil Madhavapeddy, KC Sivaramakrishnan, and Leo White. May 2017. Concurrent System Programming with Effect Handlers. In *Proceedings of the Symposium on Trends in Functional Programming*. TFP'17.

Stephen Dolan, Leo White, KC Sivaramakrishnan, Jeremy Yallop, and Anil Madhavapeddy. Sep. 2015. Effective Concurrency through Algebraic Effects. In *OCaml Workshop*.

R Kent Dyvbig, Simon Peyton Jones, and Amr Sabry. 2007. A Monadic Framework for Delimited Continuations. *Journal of Functional Programming* 17 (6). Cambridge University Press: 687–730. doi:10.1017/S0956796807006259.

Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. 2019. On the Expressive Power of User-Defined Effects: Effect Handlers, Monadic Reflection, Delimited Control. *Journal of Functional Programming* 29. Cambridge University Press: 15. doi:10.1017/S0956796819000121.

Ben R. Gaster, and Mark P. Jones. 1996. *A Polymorphic Type System for Extensible Records and Variants*. NOTTCS-TR-96-3. University of Nottingham.

Jean-Yves Girard. 1986. The System F of Variable Types, Fifteen Years Later. *TCS*.

Jean-Yves Girard, Paul Taylor, and Yves Lafont. 1989. *Proofs and Types*. Cambridge University Press.

Carl A. Gunter, Didier Rémy, and Jon G. Riecke. 1995. A Generalization of Exceptions and Control in ML-like Languages. In *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture*, 12–23. FPCA '95. ACM, La Jolla, California, USA. doi:10.1145/224164.224173.

Daniel Hillerström, and Sam Lindley. 2016. Liberating Effects with Rows and Handlers. In *Proceedings of the 1st International Workshop on Type-Driven Development*, 15–27. TyDe 2016. Nara, Japan. doi:10.1145/2976022.2976033.

Daniel Hillerström, and Sam Lindley. 2018. Shallow Effect Handlers. In *APLAS'18*.

Daniel Hillerström, Sam Lindley, Bob Atkey, and KC Sivaramakrishnan. Sep. 2017. Continuation Passing Style for Effect Handlers. In *Proceedings of the Second International Conference on Formal Structures for Computation and Deduction*. FSCD'17.

Mark P. Jones. Feb. 1992. A Theory of Qualified Types. In *4th. European Symposium on Programming (ESOP'92)*, 582:287–306. Lecture Notes in Computer Science. Springer-Verlag, Rennes, France. doi:10.1007/3-540-55253-7_17.

Ohad Kammar, Sam Lindley, and Nicolas Oury. 2013. Handlers in Action. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, 145–158. ICFP '13. ACM, New York, NY, USA. doi:10.1145/2500365.2500590.

Ohad Kammar, and Matija Pretnar. Jan. 2017. No Value Restriction Is Needed for Algebraic Effects and Handlers. *Journal of Functional Programming* 27 (1). Cambridge University Press. doi:10.1017/S0956796816000320.

Oleg Kiselyov, and Chung-chieh Shan. 2009. Embedded Probabilistic Programming. In *Domain-Specific Languages*. doi:10.1007/978-3-642-03034-5_17.

Daan Leijen. 2005. Extensible Records with Scoped Labels. In *Proceedings of the 2005 Symposium on Trends in Functional Programming*, 297–312.

Daan Leijen. 2014. Koka: Programming with Row Polymorphic Effect Types. In *MSFP'14, 5th Workshop on Mathematically Structured Functional Programming*. doi:10.4204/EPTCS.153.8.

Daan Leijen. Aug. 2016. *Algebraic Effects for Functional Programming*. MSR-TR-2016-29. Microsoft Research technical report. Extended version of [Leijen 2017c].

Daan Leijen. 2017a. Implementing Algebraic Effects in C: Or Monads for Free in C. Edited by Bor-Yuh Evan Chang. *Programming Languages and Systems*, LNCS, 10695 (1). Springer International Publishing, Suzhou, China: 339–363. APLAS'17, Suzhou, China.

Daan Leijen. 2017b. Structured Asynchrony with Algebraic Effects. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Type-Driven Development*, 16–29. TyDe 2017. Oxford, UK. doi:10.1145/3122975.3122977.

Daan Leijen. Jan. 2017. Type Directed Compilation of Row-Typed Algebraic Effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17)*, 486–499. Paris, France. doi:10.1145/3009837.3009872.

Sam Lindley, and James Cheney. 2012. Row-Based Effect Types for Database Integration. In *Proceedings of the 8th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, 91–102. TLDI'12. doi:10.1145/2103786.2103798.

Sam Lindley, Connor McBride, and Craig McLaughlin. Jan. 2017. Do Be Do Be Do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17)*, 500–514. Paris, France. doi:10.1145/3009837.3009897.

Simon L Peyton Jones, and John Launchbury. 1995. State in Haskell. *Lisp and Symbolic Comp.* 8 (4): 293–341. doi:10.1007/BF01018827.

Andrew M. Pitts. 1998. Existential Types: Logical Relations and Operational Equivalence. In *In Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, 309–326. Springer-Verlag.

Atze van der Ploeg, and Oleg Kiselyov. 2014. Reflection without Remorse: Revealing a Hidden Sequence to Speed up Monadic Reflection. In *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, 133–144. Haskell'14. Göthenburg, Sweden. doi:10.1145/2633357.2633360.

Gordon D. Plotkin, and John Power. 2003. Algebraic Operations and Generic Effects. *Applied Categorical Structures* 11 (1): 69–94. doi:10.1023/A:1023064908962.

Gordon D. Plotkin, and Matija Pretnar. 2013. Handling Algebraic Effects. In *Logical Methods in Computer Science*, volume 9. 4. doi:10.2168/LMCS-9(4:23)2013.

Matija Pretnar. Dec. 2015. An Introduction to Algebraic Effects and Handlers. Invited Tutorial Paper. *Electron. Notes Theor. Comput. Sci.* 319 (C). Elsevier Science Publishers: 19–35. doi:10.1016/j.entcs.2015.12.003.

Matija Pretnar, Amr Hany Shehata Saleh, Axel Faes, and Tom Schrijvers. 2017. *Efficient Compilation of Algebraic Effects and Handlers*. CW Reports. Department of Computer Science, KU Leuven; Leuven, Belgium. `https://lirias.kuleuven.be/retrieve/472230`.

Didier Rémy. 1994. Type Inference for Records in Natural Extension of ML. In *Theoretical Aspects of Object-Oriented Programming*, 67–95. doi:10.1.1.48.5873.

Philipp Schuster, Jonathan Immanuel Brachthäuser, and Klaus Ostermann. 2019. *Staging Effects and Handlers – A Shift to Zero-Cost Effect Handlers*. Technical Report. University of Tübingen, Germany. `http://ps.informatik.uni-tuebingen.de/publications/schuster19zero`.

Sebastian Ullrich, and Leonardo de Moura. Sep. 2019. Counting Immutable Beans – Reference Counting Optimized for Purely Functional Programming. In *Proceedings of the 31st Symposium on Implementation and Application of Functional Languages (IFL'19)*. Singapore.

Philip Wadler. Jul. 1990. Recursive Types for Free! University of Glasgow. `http://homepages.inf.ed.ac.uk/wadler/papers/free-rectypes/free-rectypes.txt`.

Andrew Wright. 1995. Simple Imperative Polymorphism. In *LISP and Symbolic Computation*, 343–356.

Nicolas Wu, Tom Schrijvers, and Ralf Hinze. 2014. Effect Handlers in Scope. In *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, 1–12. Haskell '14. Göthenburg, Sweden. doi:10.1145/2633357.2633358.

Yizhou Zhang, and Andrew C. Myers. Jan. 2019. Abstraction-Safe Effect Handlers via Tunneling. *Proc. ACM Program. Lang.* 3 (POPL). ACM. doi:10.1145/3290318.

$$\frac{}{\vdash_{\text{wf}}\ \alpha^k\ :\ k}\ [\textsc{kind-var}] \qquad \frac{\vdash_{\text{wf}}\ \sigma\ :\ *\qquad k \neq \text{lab}}{\vdash_{\text{wf}}\ \forall \alpha^k.\,\sigma\ :\ *}\ [\textsc{kind-quant}]$$

$$\frac{}{\vdash_{\text{wf}}\ c^k\ :\ k}\ [\textsc{kind-con}] \qquad \frac{\vdash_{\text{wf}}\ \sigma_1\ :\ k_2 \rightarrow k \qquad \vdash_{\text{wf}}\ \sigma_2\ :\ k_2}{\vdash_{\text{wf}}\ \sigma_1\ \sigma_2\ :\ k}\ [\textsc{kind-app}]$$

$$\frac{}{\vdash_{\text{wf}}\ \langle\rangle\ :\ \text{eff}}\ [\textsc{kind-total}] \qquad \frac{\vdash_{\text{wf}}\ \epsilon\ :\ \text{eff} \qquad \vdash_{\text{wf}}\ l\ :\ \text{lab}}{\vdash_{\text{wf}}\ \langle l \mid \epsilon\rangle\ :\ \text{eff}}\ [\textsc{kind-row}]$$

$$\frac{\vdash_{\text{wf}}\ \sigma_1\ :\ * \qquad \vdash_{\text{wf}}\ \sigma_2\ :\ * \qquad \vdash_{\text{wf}}\ \epsilon\ :\ \text{eff}}{\vdash_{\text{wf}}\ \sigma_1 \rightarrow \epsilon\ \sigma_2}\ [\textsc{kind-arrow}]$$

Fig. 11. Well-formedness of types.

APPENDICES

A FULL RULES

This section contains the rules for well-formed types, and for typing and translating the evaluation contexts.

A.1 Well Formed Types

The kinding rules for types are shown in Figure 11. The rules are standard mostly standard except we do not allow type abstraction over effect labels – or otherwise equivalence between types cannot be decided statically. The rules KIND-TOTAL, KIND-ROW, and KIND-ARROW are not strictly necessary and can be derived from KIND-APP.

A.2 Evaluation Context Typing and Translation

$$\underset{\uparrow}{\Gamma}; \underset{\uparrow\, \mathsf{F}^{ev}}{w} \vdash_{ec} \underset{\uparrow\, \mathsf{F}^{\epsilon}}{\mathsf{E}} : \sigma \rightarrow \underset{\downarrow}{\sigma'} \mid \underset{\uparrow}{\epsilon} \rightsquigarrow \underset{\downarrow\, \mathsf{F}^{ev}}{\mathsf{E}'}$$

$$\frac{}{\Gamma; w \vdash_{ec} \square : \sigma \rightarrow \sigma \mid \epsilon \rightsquigarrow \square}\ [\textsc{cEmpty}]$$

$$\frac{\Gamma; w \vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e' \qquad \Gamma; w \vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow (\sigma_2 \rightarrow \epsilon\ \sigma_3) \mid \epsilon \rightsquigarrow \mathsf{E}'}{\Gamma; w \vdash_{ec} \mathsf{E}\ e : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow \mathsf{E}'\ w\ e'}\ [\textsc{cApp1}]$$

$$\frac{\Gamma \vdash_{val} v : \sigma_2 \rightarrow \epsilon\ \sigma_3 \rightsquigarrow v' \qquad \Gamma; w \vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'}{\Gamma; w \vdash_{ec} v\ \mathsf{E} : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow v'\ w\ \mathsf{E}'}\ [\textsc{cApp2}]$$

$$\frac{\Gamma; w \vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \forall \alpha.\ \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'}{\Gamma; w \vdash_{ec} \mathsf{E}\ [\sigma] : \sigma_1 \rightarrow \sigma_2\ [\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow \mathsf{E}'\ [\lceil \sigma \rceil]}\ [\textsc{cTApp}]$$

$$\frac{\Gamma \vdash_{ops} h : \sigma \mid l \mid \epsilon \rightsquigarrow h' \qquad \Gamma; \langle\!\langle l : (m, h') \mid w \rangle\!\rangle \vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow \mathsf{E}'}{\Gamma; w \vdash_{ec} \mathsf{handle}^{\epsilon}\ h\ \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}^{w}_{m}\ h'\ \mathsf{E}'}\ [\textsc{cHandle}]$$

Fig. 12. Evaluation context typing with evidence translation

B PROOFS

This section contains all the proofs for the Lemmas and Theorems in the main paper organized by system.

B.1 System F^{ϵ}

B.1.1 Type Erasure.

Proof. (*Of Lemma 1*) By straightforward induction. Note $(\Lambda\alpha.\ v)^* = v^*$ is a value by I.H.. ☐

Lemma 9. (*Substitution of Type Erasure*)
1. $(e[x{:=}v])^* = e^*[x{:=}v^*]$.
2. $(v_0[x{:=}v])^* = v_0^*[x{:=}v^*]$.
3. $(h[x{:=}v])^* = h^*[x{:=}v^*]$.

Proof. (*Of Lemma 9*) **Part 1** By induction on e.
case $e = v_0$. Follows from Part 2.
case $e = e_1\ e_2$.

$((e_1\ e_2)[x{:=}v])^*$	
$= ((e_1[x{:=}v])\ (e_2[x{:=}v]))^*$	by substitution
$= (e_1[x{:=}v])^*\ (e_2[x{:=}v])^*$	by erasure
$= (e_1^*[x{:=}v^*])\ (e_2^*[x{:=}v^*])$	I.H.
$= (e_1^*\ e_2^*)\ [x{:=}v]$	by substitution
$= (e_1\ e_2)^*[x{:=}v]$	by erasure

$$
\begin{array}{cccccccccc}
\Gamma; & w; & w' & \Vdash_{ec} & \mathsf{E} & : \sigma \rightarrow & \sigma' & \mid \epsilon & \rightsquigarrow & e' \\
\uparrow & \uparrow & \uparrow & & \uparrow & & \downarrow & \uparrow & & \downarrow \\
 & & \mathsf{F}^{v} & & \mathsf{F}^{ev} & & & & & \mathsf{F}^{v}
\end{array}
$$

$$
\frac{}{\Gamma; w; w' \Vdash_{ec} \Box : \sigma \rightarrow \sigma \mid \epsilon \rightsquigarrow id} \text{[MON-CEMPTY]}
$$

$$
\frac{\Gamma; w; w' \Vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e' \qquad \Gamma; w; w' \Vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow (\sigma_2 \Rightarrow \epsilon\ \sigma_3) \mid \epsilon \rightsquigarrow g}{\Gamma; w; w' \Vdash_{ec} \mathsf{E}\ w\ e : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow (\lambda f.\ e' \triangleright f\ w') \bullet g} \text{[MON-CAPP1]}
$$

$$
\frac{\Gamma; w; w' \Vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \forall \alpha.\ \sigma_2 \mid \epsilon \rightsquigarrow g}{\Gamma; w; w' \Vdash_{ec} \mathsf{E}\ [\sigma] : \sigma_1 \rightarrow \sigma_2\ [\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow (\lambda x.\ pure\ (x[\lfloor\sigma\rfloor])) \bullet g} \text{[MON-CTAPP]}
$$

$$
\frac{\Gamma \Vdash_{val} v : \sigma_2 \Rightarrow \epsilon\ \sigma_3 \rightsquigarrow v' \qquad \Gamma; w; w' \Vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow g}{\Gamma; w; w' \Vdash_{ec} v\ w\ \mathsf{E} : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow (v'\ w') \bullet g} \text{[MON-CAPP2]}
$$

$$
\frac{\Gamma \Vdash_{ops} h : \sigma \mid l \mid \epsilon \rightsquigarrow h' \qquad \Gamma; \langle\!\langle l{:}(m,h) \mid w \rangle\!\rangle; \langle\!\langle l{:}(m,h') \mid w' \rangle\!\rangle \Vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow g}{\Gamma; w; w' \Vdash_{ec} \mathsf{handle}^{w}_{m}\ h\ \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \epsilon \rightsquigarrow prompt[\epsilon, \sigma]\ m\ w' \circ g} \text{[MON-CHANDLE]}
$$

Fig. 13. Evidence context typing and monadic translation (($\triangleright$) is monadic bind, ($\bullet$) Kleisli composition, and ($\circ$) is regular function composition).

case $e = e_1\ [\sigma]$.

$((e_1\ [\sigma])\ [x{:=}v])^*$
$= (e_1[x{:=}v]\ [\sigma])^*$ by substitution
$= (e_1[x{:=}v])^*$ by erasure
$= e_1^*[x{:=}v^*]$ I.H.
$= (e_1\ [\sigma])^*[x{:=}v^*]$ by erasure

case $e = \mathsf{handle}^{\epsilon}\ h\ e_0$.

$((\mathsf{handle}^{\epsilon}\ h\ e_0)[x{:=}v])^*$
$= (\mathsf{handle}^{\epsilon}\ h[x{:=}v]\ e_0[x{:=}v])^*$ by substitution
$= \mathsf{handle}\ (h[x{:=}v])^*\ (e_0[x{:=}v])^*$ by erasure
$= \mathsf{handle}\ (h^*[x{:=}v^*])\ (e_0^*[x{:=}v^*])$ Part 3 and I.H.
$= (\mathsf{handle}\ h^*\ e_0^*)[x{:=}v^*]$ by substitution
$= (\mathsf{handle}^{\epsilon}\ h\ e_0)^*[x{:=}v^*]$ by erasure

Part 2 By induction on v_0.

case $v_0 = x$.

$(x[x{:=}v])^*$
$= v^*$ by substitution
$= x[x{:=}v^*]$ by substitution
$= x^*[x{:=}v^*]$ by erasure

case $v_0 = y$ and $y \neq x$.

$$\begin{array}{lll}
(y[x{:=}v])^* & & \\
= & y^* & \text{by substitution} \\
= & y & \text{by erasure} \\
= & y[x{:=}v^*] & \text{by substitution} \\
= & y^*[x{:=}v^*] & \text{by erasure}
\end{array}$$

case $v_0 = \lambda^\epsilon\ y : \sigma.\ e.$

$$\begin{array}{lll}
((\lambda^\epsilon\ y : \sigma.\ e)[x{:=}v])^* & & \\
= & (\lambda^\epsilon\ y : \sigma.\ e[x{:=}v]\ )^* & \text{by substitution} \\
= & \lambda y.\ (e[x{:=}v])^* & \text{by erasure} \\
= & \lambda y.\ e^*[x{:=}v^*] & \text{Part 1} \\
= & (\lambda y.\ e^*)[x{:=}v^*] & \text{by substitution} \\
= & (\lambda^\epsilon\ y :\ \sigma.\ e)^*[x{:=}v^*] & \text{by erasure}
\end{array}$$

case $v_0 = \Lambda\alpha^k.\ v_1.$

$$\begin{array}{lll}
((\Lambda\alpha^k.\ v_1)[x{:=}v])^* & & \\
= & (\Lambda\alpha^k \cdot v_1[x{:=}v])^* & \text{by substitution} \\
= & (v_1[x{:=}v])^* & \text{by erasure} \\
= & v_1^*[x{:=}v^*] & \text{I.H.} \\
= & (\ \Lambda\alpha^k.\ v_1)^*[x{:=}v^*] & \text{by erasure}
\end{array}$$

case $v_0 = \mathsf{handler}^\epsilon\ h.$

$$\begin{array}{lll}
(\mathsf{handler}^\epsilon\ h[x{:=}v])^* & & \\
= & (\mathsf{handler}^\epsilon\ h[x{:=}v])^* & \text{by substitution} \\
= & \mathsf{handler}\ (h[x{:=}v])^* & \text{by erasure} \\
= & \mathsf{handler}\ h^*[x{:=}v^*] & \text{Part 3} \\
= & (\mathsf{handler}\ h^*)[x{:=}v^*] & \text{by substitution} \\
= & (\mathsf{handler}^\epsilon\ h)^*[x{:=}v^*] & \text{by erasure}
\end{array}$$

case $v_0 = \mathsf{perform}^\epsilon\ op\ \overline{\sigma}.$

$$\begin{array}{lll}
((\mathsf{perform}^\epsilon\ op\ \overline{\sigma})[x{:=}v])^* & & \\
= & (\mathsf{perform}^\epsilon\ op\ \overline{\sigma})^* & \text{by substitution} \\
= & \mathsf{perform}\ op & \text{by erasure} \\
= & (\mathsf{perform}\ op)[x{:=}v^*] & \text{by substitution} \\
= & (\mathsf{perform}^\epsilon\ op\ \overline{\sigma})^*[x{:=}v^*] & \text{by erasure}
\end{array}$$

Part 3 Follows directly from Part 1.

□

Lemma 10. (*Type Variable Substitution of Type Erasure*)
1. $(e[\alpha{:=}\sigma])^* = e^*$.
2. $(v_0[\alpha{:=}\sigma])^* = v_0^*$.
3. $(h[\alpha{:=}\sigma])^* = h^*$.

Proof. (*Of Lemma 10*) By straightforward induction. Note all types are erased. □

Proof. (*Of Theorem 1*) **case** $(\lambda^\epsilon\ x : \sigma.\ e)\ v \longrightarrow e\ [x{:=}v]$.

$$\begin{array}{ll}
((\lambda^\epsilon\ x : \sigma.\ e)\ v)^* = (\lambda x.\ e^*)\ v^* & \text{by erasure} \\
v^* \text{ is a value} & \text{Lemma 1} \\
(\lambda x.\ e^*)\ v^* \longrightarrow e^*[x{:=}v^*] & (\mathit{app}) \\
(e[x{:=}v])^* = e^*[x{:=}v^*] & \text{Lemma 9}
\end{array}$$

case $(\Lambda\alpha^k.v)\ [\sigma] \longrightarrow v[\alpha{:=}\sigma]$.

$((\Lambda \alpha^k . v)\ [\sigma])^* \ = \ v^*$ by erasure
$(v[\alpha{:=}\sigma])^* \ = \ v^*$ Lemma 10

case $(\mathsf{handler}^\epsilon\ h)\ v \longrightarrow \mathsf{handle}^\epsilon\ h\,(v\,())$.

$((\mathsf{handler}^\epsilon\ h)\ v)^* \ = \ \mathsf{handler}\ h^*\ v^*$ by erasure
v^* is a value Lemma 1
$\mathsf{handler}\ h^*\ v^* \longrightarrow \mathsf{handle}\ h^*\ (v^*\ ())$ (*handler*)

case $\mathsf{handle}^\epsilon\ h \cdot v \longrightarrow v$.

$(\mathsf{handle}^\epsilon\ h \cdot v)^* \ = \ \mathsf{handle}\ h^* \cdot v^*$ by erasure
v^* is a value Lemma 1
$\mathsf{handle}\ h^* \cdot v^* \longrightarrow v^*$ (*return*)

case $\mathsf{handle}^\epsilon\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v \longrightarrow f\ [\overline{\sigma}]\ v\ k_1$.

$k_1 \ = \ \lambda^\epsilon\ x\!:\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^\epsilon\ h \cdot \mathsf{E} \cdot x$ (*perform*)
$(\mathsf{handle}^\epsilon\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v)^* \ = \ \mathsf{handle}\ h^* \cdot \mathsf{E}^* \cdot \mathsf{perform}\ op\ v^*$ by erasure
v^* is a value Lemma 1
$\mathsf{handle}\ h^* \cdot \mathsf{E}^* \cdot \mathsf{perform}\ op\ v^* \longrightarrow f^*\ v^*\ k_2$ (*perform*)
$k_2 \ = \ \lambda x.\ \mathsf{handle}\ h^* \cdot \mathsf{E}^* \cdot x$ above
$k_2 \ = \ k_1^*$ by erasure
$(f\ [\overline{\sigma}]\ v\ k_1)^* \ = \ f^*\ v^*\ k_2$ by erasure
□

B.1.2 Evaluation Context Typing.

Proof. (*Of Lemma 2*) Apply Lemma 16, ignoring all evidence and translations. □

Proof. (*Of Lemma 3*) Apply Lemma 17, ignoring all evidence and translations. □

Proof. (*Of Lemma 4*)

$\varnothing\ \vdash\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v]\ :\ \sigma \mid \langle\rangle$ given
$\varnothing\ \vdash\ \mathsf{perform}\ op\ \overline{\sigma}\ v\ :\ \sigma_1 \mid \lceil \mathsf{E} \rceil^l$ Lemma 3
$\varnothing\ \vdash\ \mathsf{perform}\ op\ \overline{\sigma}\ :\ \sigma_2 \rightarrow \lceil \mathsf{E} \rceil^l\ \sigma_1 \mid \lceil \mathsf{E} \rceil^l$ APP
$\varnothing\ \vdash_{\mathsf{val}}\ \mathsf{perform}\ op\ \overline{\sigma}\ :\ \sigma_2 \rightarrow \lceil \mathsf{E} \rceil^l\ \sigma_1$ VAL
$l\ \in \lceil \mathsf{E} \rceil^l$ OP
$\mathsf{E}\ = \ \mathsf{E}_1 \cdot \mathsf{handle}^\epsilon\ h \cdot \mathsf{E}_2$ By definition of $\lceil \mathsf{E} \rceil^l$
$op \rightarrow f\ \ \in h$ above
$op\ \notin \mathsf{bop}(\mathsf{E}_2)$ Let $\mathsf{handle}^\epsilon\ h$ be the innermost one
□

Proof. (*Of Lemma 5*)

$\varnothing\ \vdash\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v]\ :\ \sigma \mid \epsilon$ given
$\varnothing\ \vdash\ \mathsf{perform}\ op\ \overline{\sigma}\ v\ :\ \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle$ Lemma 3
$\varnothing\ \vdash\ \mathsf{perform}\ op\ \overline{\sigma}\ :\ \sigma_2 \rightarrow \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle\ \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle$ APP
$\varnothing\ \vdash_{\mathsf{val}}\ \mathsf{perform}\ op\ \overline{\sigma}\ :\ \sigma_2 \rightarrow \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle\ \sigma_1$ VAL
$l\ \in \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle$ OP
$op\ \notin \mathsf{bop}(\mathsf{E})$ given
$l\ \in \epsilon$ Follows
□

B.1.3 Substitution.

Lemma 11. (*Variable Substitution*)
If $\Gamma_1, x : \sigma_1, \Gamma_2 \vdash\ e\ :\ \sigma \mid \epsilon$ and $\Gamma_1,\ \Gamma_2 \vdash_{\mathsf{val}}\ v\ :\ \sigma_1$, then $\Gamma_1, \Gamma_2\ \vdash\ e[x{:=}v]\ :\ \sigma \mid \epsilon$.

Proof. (*Of Lemma 11*) Applying Lemma 18, ignoring all evidences and translations. □

Lemma 12. (*Type Variable Substitution*)
If $\Gamma \vdash\ e\ :\ \sigma \mid \epsilon$ and $\vdash_{\mathsf{wf}}\ \sigma_1\ :\ k$, then $\Gamma[\alpha^k{:=}\sigma_1]\ \vdash\ e[\alpha^k{:=}\sigma_1]\ :\ \sigma[\alpha^k{:=}\sigma_1] \mid \epsilon$.

Proof. (*Of Lemma 12*) Applying Lemma 20, ignoring all evidences and translations. □

B.1.4 Progress.

Lemma 13. (*Progress with effects*)
If $\varnothing\ \vdash\ e_1\ :\ \sigma \mid \epsilon$ then either e_1 is a value, or $e_1 \longmapsto e_2$, or $e_1\ =\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v]$, where $op\ :\ \forall \overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2\ \in \Sigma(l)$, and $l \notin bop(\mathsf{E})$.

Proof. (*of Lemma 13*) By induction on typing. **case** $e_1\ =\ v$. The goal holds trivially.
case $e_1\ =\ e_3\ e_4$.

$\varnothing\ \vdash\ e_3\ e_4\ :\ \sigma \mid \epsilon$ given
$\varnothing\ \vdash\ e_3\ :\ \sigma_1 \rightarrow \epsilon\ \sigma \mid \epsilon$ APP
$\varnothing\ \vdash\ e_4\ :\ \sigma_1 \mid \epsilon$ above

By I.H., we know that either e_3 is a value, or $e_3 \longmapsto e_5$, or $e_3\ =\ \mathsf{E}_0[\mathsf{perform}\ op\ \overline{\sigma}\ v]$.

- $e_3 \longmapsto e_5$. Then we know $e_3\ e_4 \longmapsto e_5\ e_4$ by STEP and the goal holds.
- $e_3\ =\ \mathsf{E}_0[\mathsf{perform}\ op\ \overline{\sigma}\ v]$. Let $\mathsf{E}\ =\ \mathsf{E}_0\ e_4$, then we have $e_1\ =\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v]$.
- e_3 is a value. By I.H., we know either e_4 is a value, or $e_4 \longmapsto e_6$, or $e_4\ =\ \mathsf{E}_0[\mathsf{perform}\ op\ \overline{\sigma}\ v]$
  - $e_4 \longmapsto e_6$, then we know $e_3\ e_4 \longmapsto e_3\ e_6$ by STEP and the goal holds.
  - $e_4\ =\ \mathsf{E}_0[\mathsf{perform}\ op\ \overline{\sigma}\ v]$. Let $\mathsf{E}\ =\ e_3\ \mathsf{E}_0$, then we have $e_1\ =\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v]$.
  - e_4 is a value, then we do case analysis on the form of e_3.

subcase $e_3\ =\ x$. This is impossible because x is not well-typed under an empty context.
subcase $e_3\ =\ \lambda x{:}\ \sigma.\ e$. Then by (*app*) and (*step*) we have $(\lambda x\ :\ \sigma \cdot e)\ e_4 \longmapsto e[x{:=}e_4]$.
subcase $e_3\ =\ \Lambda\alpha.\ e$. This is impossible because it does not have a function type.
subcase $e_3\ =\ \mathsf{perform}\ op\ \overline{\sigma}$. Let $\mathsf{E} = \square$, then we have $e_1\ =\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ e_4]$.
subcase $e_3\ =\ \mathsf{handler}^{\epsilon}\ h$. Then by (*handler*) and (*step*) we have
$\mathsf{handler}^{\epsilon}\ h\ e_4 \longrightarrow \mathsf{handle}^{\epsilon}\ h\ (e_4\ ())$.

case $e_1\ =\ e_3\ [\sigma_1]$.

$\varnothing\ \vdash\ e_3\ [\sigma_1]\ :\ \sigma_2[\alpha{:=}\sigma_1]\ \mid \epsilon$ given
$\varnothing\ \vdash\ e_3\ :\ \forall \alpha.\ \sigma_2 \mid \epsilon$ APP

By I.H., we know that either e_3 is a value, or $e_3 \longmapsto e_5$, or $e_3\ =\ \mathsf{E}_0[\mathsf{perform}\ op\ \overline{\sigma}\ v]$.

- $e_3 \longmapsto e_5$. Then we know $e_3\ [\sigma_1] \longmapsto e_5\ [\sigma_1]$ by STEP and the goal holds.
- $e_3\ =\ \mathsf{E}_0[\mathsf{perform}\ op\ \overline{\sigma}\ v]$. Let $\mathsf{E}\ =\ \mathsf{E}_0\ [\sigma_1]$, then we have $e_1\ =\ \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v]$.
- e_3 is a value. Then we do case analysis on the form of e_3.

subcase $e_3\ =\ x$. This is impossible because x is not well-typed under an empty context.
subcase $e_3\ =\ \lambda x{:}\ \sigma.\ e$. This is impossible because it does not have a polymorphic type.
subcase $e_3\ =\ \Lambda\alpha.\ e$. Then by (*tapp*) and (*step*) we have $(\Lambda\alpha.\ e)\ [\sigma_1] \longmapsto e[\alpha{:=}\sigma_1]$.
subcase $e_3\ =\ \mathsf{perform}\ op\ \overline{\sigma}'$. This is impossible because it does not have a polymorphic type.
subcase $e_3\ =\ \mathsf{handler}^{\epsilon}\ h$. This is impossible because it does not have a polymorphic type.

case $e_1\ =\ \mathsf{handle}^{\epsilon}\ h\ e$.

$\varnothing \vdash \text{handle}^{\epsilon}\ h\ e : \sigma \mid \epsilon$ given
$\varnothing \vdash e : \sigma \mid \langle l \mid \epsilon \rangle$ HANDLE

By I.H., we know that either e is a value, or $e \longmapsto e_3$, or $e = \mathsf{E}_0[\text{perform}\ op\ \overline{\sigma}\ v]$.

- $e \longmapsto e_3$. Then we know $\text{handle}^{\epsilon}\ h\ e \longmapsto \text{handle}^{\epsilon}\ h\ e_3$ by STEP and the goal holds.
- $e = \mathsf{E}_0[\text{perform}\ op\ \overline{\sigma}\ v]$, and $op \notin bop(\mathsf{E}_0)$. We discuss whether op is bound in h.
  - $op \to f \in h$. Then by (*perform*) and (*step*) we have $\text{handle}^{\epsilon}\ h \cdot \mathsf{E}_0 \cdot \text{perform}\ op\ \overline{\sigma}\ v \longmapsto f\ \overline{\sigma}\ v\ k$.
  - $op \notin h$. Let $\mathsf{E} = \text{handle}^{\epsilon}\ h\ \mathsf{E}_0$, then we have $e_1 = \mathsf{E}[\text{perform}\ op\ \overline{\sigma}\ v]$.
- e is a value. Then by (*return*) and (*step*) we have $\text{handle}^{\epsilon}\ h\ e \longmapsto e$.

□

Proof. (*Of Theorem 2*) Apply Lemma 13, then we know that either e_1 is a value, or $e_1 \longmapsto e_2$, or $e_1 = \mathsf{E}[\text{perform}\ op\ \overline{\sigma}\ v]$, where $op : \forall\overline{\alpha}.\ \sigma_1 \to \sigma_2 \in \Sigma(l)$, and $l \notin \text{bop}(\mathsf{E})$. For the first two cases, we have proved the goal. For the last case, we prove it by contradiction.

$\varnothing \vdash \mathsf{E}[\text{perform}\ op\ \overline{\sigma}\ v] : \sigma \mid \langle\rangle$ given
$l \notin bop(\mathsf{E})$ given
$l \in \langle\rangle$ Lemma 5
Contradiction

□

B.1.5 Preservation.

Lemma 14. (*Small Step Preservation*)
If $\varnothing \vdash e_1 : \sigma \mid \epsilon$ and $e_1 \longrightarrow e_2$, then $\varnothing \vdash e_2 : \sigma \mid \epsilon$.

Proof. (*of Lemma 14*) By induction on reduction.

case $(\lambda^{\epsilon}\ x : \sigma_1.\ e)\ v \longrightarrow e[x:=v]$.

$\varnothing \vdash (\lambda^{\epsilon}\ x : \sigma_1.\ e)\ v : \sigma_2 \mid \epsilon$ given
$\varnothing \vdash \lambda^{\epsilon}\ x : \sigma_1.\ e : \sigma_1 \to \epsilon\ \sigma_2 \mid \epsilon$ APP
$\varnothing \vdash v : \sigma_1 \mid \epsilon$ above
$x : \sigma_1 \vdash e : \sigma_2 \mid \epsilon$ ABS
$\varnothing \vdash e[x:=v] : \sigma_2 \mid \epsilon$ Lemma 11

case $(\Lambda\alpha.\ v)\ [\sigma] \longrightarrow v[\alpha:=\sigma]$.

$\varnothing \vdash (\Lambda\alpha.\ v)\ [\sigma] : \sigma_1\ [\alpha:=\sigma] \mid \epsilon$ given
$\varnothing \vdash \Lambda\alpha.\ v : \forall\alpha.\ \sigma_1 \mid \epsilon$ TAPP
$\varnothing \vdash_{\text{val}} \Lambda\alpha.\ v : \forall\alpha.\ \sigma_1$ VAL
$\varnothing \vdash_{\text{val}} v : \sigma_1$ TABS
$\varnothing \vdash_{\text{val}} v : \sigma_1 \mid \epsilon$ VAL
$\varnothing \vdash_{\text{val}} v[\alpha:=\sigma] : \sigma_1[\alpha:=\sigma] \mid \epsilon$ Lemma 12

case $(\text{handler}^{\epsilon}\ h)\ v \longrightarrow \text{handle}^{\epsilon}\ h\ (v\ ())$.

$\varnothing \vdash (\mathsf{handler}^{\epsilon}\ h)\ v : \sigma \mid \epsilon$	given
$\varnothing \vdash \mathsf{handler}^{\epsilon}\ h : (() \to \langle l \mid \epsilon\rangle\ \sigma) \to \epsilon\ \sigma \mid \epsilon$	APP
$\varnothing \vdash v : () \to \langle l \mid \epsilon\rangle\ \sigma \mid \epsilon$	above
$\varnothing \vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h : (() \to \langle l \mid \epsilon\rangle\ \sigma) \to \epsilon\ \sigma$	VAL
$\varnothing \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon$	HANDLER
$\varnothing \vdash v : () \to \langle l \mid \epsilon\rangle\ \sigma \mid \langle l \mid \epsilon\rangle$	Lemma 25
$\varnothing \vdash v\,() : \sigma \mid \langle l \mid \epsilon\rangle$	APP
$\varnothing \vdash \mathsf{handle}^{\epsilon}\ h\,(v\,()) : \sigma \mid \langle \epsilon\rangle$	HANDLE

case $\mathsf{handle}^{\epsilon}\ h \cdot v \longrightarrow v$.

$\varnothing \vdash \mathsf{handle}^{\epsilon}\ h \cdot v : \sigma \mid \epsilon$	given
$\varnothing \vdash v : \sigma \mid \langle l \mid \epsilon\rangle$	HANDLE
$\varnothing \vdash v : \sigma \mid \langle \epsilon\rangle$	Lemma 25

case $\mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v \longrightarrow f\ [\overline{\sigma}]\ v\ k$.

$op \notin \mathsf{bop}(\mathsf{E})$ and $op \to f\ \in h$	given
$op : \forall \overline{\alpha}.\ \sigma_1 \to \sigma_2\ \in \Sigma(l)$	given
$k\ =\ \lambda^{\epsilon}\ x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x$	given
$\varnothing \vdash \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v : \sigma \mid \epsilon$	given
$\varnothing \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon$	HANDLE
$\varnothing \vdash_{\mathsf{val}} f : \forall \overline{\alpha}.\ \sigma_1 \to \epsilon\ (\sigma_2 \to \epsilon\ \sigma) \to \epsilon\ \sigma$	OPS
$\varnothing \vdash f : \forall \overline{\alpha}.\ \sigma_1 \to \epsilon\ (\sigma_2 \to \epsilon\ \sigma) \to \epsilon\ \sigma \mid \epsilon$	VAL
$\varnothing \vdash f\ [\overline{\sigma}] : \sigma_1[\overline{\alpha}{:=}\overline{\sigma}] \to \epsilon\ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \to \epsilon\ \sigma) \to \epsilon\ \sigma \mid \epsilon$	TAPP
$\varnothing \vdash \mathsf{perform}\ op\ \overline{\sigma}\ v : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \mid \langle \lceil \mathsf{handle}^{\epsilon}\ h\ \mathsf{E}\rceil^{l} \mid \epsilon\rangle$	Lemma 3
$\varnothing \vdash_{\mathsf{ec}} \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \to \sigma \mid \epsilon$	above
$\varnothing \vdash v : \sigma_1[\overline{\alpha} := \overline{\sigma}] \mid \langle \lceil \mathsf{handle}^{\epsilon}\ h\ \mathsf{E}\rceil^{l} \mid \epsilon\rangle$	APP and TAPP
$\varnothing \vdash v : \sigma_1[\overline{\alpha} := \overline{\sigma}] \mid \epsilon$	Lemma 25
$\varnothing \vdash f\ [\overline{\sigma}]\ v : (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \to \epsilon\ \sigma) \to \epsilon\ \sigma \mid \epsilon$	APP
$x{:}\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \vdash_{\mathsf{val}} x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]$	VAR
$x{:}\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \vdash x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \mid \epsilon$	VAL
$x{:}\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \vdash_{\mathsf{ec}} \mathsf{handle}^{\epsilon} \cdot \mathsf{E} : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \to \sigma \mid \epsilon$	weakening
$x{:}\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \vdash \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x : \sigma \mid \epsilon$	Lemma 2
$\varnothing \vdash_{\mathsf{val}} \lambda^{\epsilon}\ x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \to \epsilon\ \sigma$	ABS
$\varnothing \vdash \lambda^{\epsilon}\ x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \to \epsilon\ \sigma \mid \epsilon$	VAL
$\varnothing \vdash f\ [\overline{\sigma}]\ v\ k : \sigma \mid \epsilon$	APP

□

Proof. (*Of Theorem 3*)

$e_1\ =\ \mathsf{E}[e_1']$	(*step*)
$e_1' \longrightarrow e_2'$	above
$e_2\ =\ \mathsf{E}[e_2']$	above
$\varnothing \vdash \mathsf{E}[e_1'] : \sigma \mid \langle\rangle$	given
$\varnothing \vdash e_1 : \sigma_1 \mid \lceil \mathsf{E}\rceil^{l}$	Lemma 3
$\varnothing \vdash \mathsf{E} : \sigma_1 \to \sigma \mid \langle\rangle$	above
$\varnothing \vdash e_2 : \sigma_1 \mid \lceil \mathsf{E}\rceil^{l}$	Lemma 14
$\varnothing \vdash \mathsf{E}[e_2] : \sigma \mid \langle\rangle$	Lemma 2

□

B.2 Translation from System F^{ϵ} to System F^{ev}

B.2.1 Type Translation.

Lemma 15. (*Stable under substitution*)
Translation is stable under substitution, $\lceil\sigma\rceil[\alpha := \lceil\sigma'\rceil] = \lceil\sigma[\alpha{:=}\sigma']\rceil$.

Proof. (*Of Lemma 15*) By induction on σ.

case $\sigma = \alpha$.

$\lceil\alpha\rceil[\alpha{:=}\lceil\sigma'\rceil]$
$= \alpha[\alpha{:=}\lceil\sigma'\rceil]$ by translation
$= \lceil\sigma'\rceil$ by substitution
$\lceil\alpha[\alpha{:=}\sigma']\rceil$
$= \lceil\sigma'\rceil$ by substitution

case $\sigma = \beta$ and $\beta \neq \alpha$.

$\lceil\beta\rceil[\alpha{:=}\lceil\sigma'\rceil]$
$= \beta[\alpha{:=}\lceil\sigma'\rceil]$ by translation
$= \beta$ by substitution
$\lceil\beta[\alpha{:=}\sigma']\rceil$
$= \lceil\beta\rceil$ by substitution
$= \beta$ by translation

case $\sigma = \sigma_1 \rightarrow \epsilon\ \sigma_2$.

$\lceil\sigma_1 \rightarrow \epsilon\ \sigma_2\rceil[\alpha{:=}\lceil\sigma'\rceil]$
$= (\lceil\sigma_1\rceil \Rightarrow\ \epsilon\ \lceil\sigma_2\rceil)[\alpha{:=}\lceil\sigma'\rceil]$ by translation
$= (\lceil\sigma_1\rceil[\alpha{:=}\lceil\sigma'\rceil]) \Rightarrow\ \epsilon\ (\lceil\sigma_2\rceil[\alpha{:=}\lceil\sigma'\rceil])$ by substitution
$= (\lceil\sigma_1[\alpha{:=}\sigma']\rceil) \Rightarrow\ \epsilon\ (\lceil\sigma_2[\alpha{:=}\sigma']\rceil)$ I.H.
$\lceil(\sigma_1 \rightarrow \epsilon\ \sigma_2)[\alpha{:=}\sigma']\rceil$
$= \lceil\sigma_1[\alpha{:=}\sigma']{\rightarrow}\epsilon\ \sigma_2[\alpha{:=}\sigma']\rceil$ by substitution
$= (\lceil\sigma_1[\alpha{:=}\sigma']\rceil) \Rightarrow\ \epsilon\ (\lceil\sigma_2[\alpha{:=}\sigma']\rceil)$ by translation

case $\sigma = \forall\beta.\ \sigma_1$.

$\lceil\forall\beta.\ \sigma_1\rceil[\alpha{:=}\lceil\sigma'\rceil]$
$= (\forall\beta.\ \lceil\sigma_1\rceil)[\alpha{:=}\lceil\sigma'\rceil]$ by translation
$= \forall\beta.\ \lceil\sigma_1\rceil[\alpha{:=}\lceil\sigma'\rceil]$ by substitution
$= \forall\beta.\ \lceil\sigma_1[\alpha{:=}\sigma']\rceil$ I.H.
$\lceil(\forall\beta.\ \sigma_1)[\alpha{:=}\sigma']\rceil$
$= \lceil\forall\beta.\ \sigma_1[\alpha{:=}\sigma']\rceil$ by substitution
$= \forall\beta.\ \lceil\sigma_1[\alpha{:=}\sigma']\rceil$ by translation

case $\sigma = c\ \tau_1\ \ldots\ \tau_n$.

$\lceil c\ \tau_1\ \ldots\ \tau_n\rceil[\alpha{:=}\lceil\sigma'\rceil]$
$= (c\ \lceil\tau_1\rceil\ \ldots\ \lceil\tau_n\rceil)[\alpha{:=}\lceil\sigma'\rceil]$ by translation
$= c\ (\lceil\tau_1\rceil[\alpha{:=}\lceil\sigma'\rceil]) \ldots (\lceil\tau_n\rceil[\alpha{:=}\lceil\sigma'\rceil])$ by substitution
$= c\ (\lceil\tau_1[\alpha{:=}\sigma']\rceil) \ldots (\lceil\tau_n[\alpha{:=}\sigma']\rceil)$ by I.H.
$\lceil(c\ \tau_1\ \ldots\ \tau_n)[\alpha{:=}\sigma']\rceil$
$= \lceil c\ \tau_1[\alpha{:=}\sigma']\ \ldots\ \tau_n[\alpha{:=}\sigma']\rceil$ by substitution
$= c\ (\lceil\tau_1[\alpha{:=}\sigma']\rceil) \ldots (\lceil\tau_n[\alpha{:=}\sigma']\rceil)$ by translation

□

B.2.2 Evaluation Context Typing.

Lemma 16. (*Evaluation context typing with evidence translation*)
If $\Gamma;\ w \vdash_{ec} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'$ and $\Gamma;\ \langle\!\langle \lceil \mathsf{E}' \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}' \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$,
then $\Gamma;\ w \vdash \mathsf{E}[e] : \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'[e']$.

Proof. (*of Lemma 16*) By induction on the evaluation context typing.

case $\mathsf{E} = \square$. The goal follows trivially.

case $\mathsf{E} = \mathsf{E}_0\ e_0$.

$\Gamma; w \vdash_{ec} \mathsf{E}_0\ e_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}_0'\ w\ e_0'$	given
$\Gamma; w \vdash_{ec} \mathsf{E}_0 : \sigma_1 \rightarrow (\sigma_3 \rightarrow \epsilon\ \sigma_2) \mid \epsilon \rightsquigarrow \mathsf{E}_0'$	CAPP1
$\Gamma; w \vdash e_0 : \sigma_3 \mid \epsilon \rightsquigarrow e_0'$	above
$\lceil \mathsf{E}_0'\ w\ e_0' \rceil = \lceil \mathsf{E}_0' \rceil$	by definition
$\lceil \mathsf{E}_0'\ w\ e_0' \rceil^{l} = \lceil \mathsf{E}_0' \rceil^{l}$	by definition
$\Gamma;\ \langle\!\langle \lceil \mathsf{E}_0'\ w\ e_0 \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0'\ w\ e' \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	given
$\Gamma;\ \langle\!\langle \lceil \mathsf{E}_0' \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	by substitution
$\Gamma; w \vdash \mathsf{E}_0[e] : \sigma_3 \rightarrow \epsilon\ \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}_0'[e']$	I.H.
$\Gamma; w \vdash \mathsf{E}_0[e]\ e_0 : \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}_0'[e]\ w\ e_0'$	APP

case $\mathsf{E} = v\ \mathsf{E}_0$.

$\Gamma; w \vdash_{ec} v\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow v'\ w\ \mathsf{E}_0'$	given
$\Gamma \vdash_{val} v : \sigma_3 \rightarrow \epsilon\ \sigma_2 \rightsquigarrow v'$	CAPP2
$\Gamma; w \vdash_{ec} \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow \mathsf{E}_0'$	above
$\lceil v'\ w\ \mathsf{E}_0' \rceil = \lceil \mathsf{E}_0' \rceil$	by definition
$\lceil v\ w\ \mathsf{E}_0' \rceil^{l} = \lceil \mathsf{E}_0' \rceil^{l}$	by definition
$\Gamma;\ \langle\!\langle \lceil v\ w\ \mathsf{E}_0' \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil v\ w\ \mathsf{E}_0' \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	given
$\Gamma;\ \langle\!\langle \lceil \mathsf{E}_0' \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	by substitution
$\Gamma; w \vdash \mathsf{E}_0[e] : \sigma_3 \mid \epsilon \rightsquigarrow \mathsf{E}'[e']$	I.H.
$\Gamma; w \vdash v\ \mathsf{E}_0[e] : \sigma_2 \mid \epsilon \rightsquigarrow v'\ w'\ \mathsf{E}_0'[e']$	APP

case $\mathsf{E} = \mathsf{E}_0\ [\sigma]$.

$\Gamma; w \vdash_{ec} \mathsf{E}_0\ [\sigma] : \sigma_1 \rightarrow \sigma_3[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow \mathsf{E}_0'\ [\lceil \sigma \rceil]$	given
$\Gamma; w \vdash_{ec} \mathsf{E}_0 : \sigma_1 \rightarrow \forall \alpha.\ \sigma_3 \mid \epsilon \rightsquigarrow \mathsf{E}_0'$	CTAPP
$\lceil \mathsf{E}_0'\ [\lceil \sigma \rceil] \rceil = \lceil \mathsf{E}_0' \rceil$	by definition
$\lceil \mathsf{E}_0'\ [\lceil \sigma \rceil] \rceil^{l} = \lceil \mathsf{E}_0' \rceil^{l}$	by definition
$\Gamma;\ \langle\!\langle \lceil \mathsf{E}_0'\ [\lceil \sigma \rceil] \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0'\ [\lceil \sigma \rceil]\ \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	given
$\Gamma;\ \langle\!\langle \lceil \mathsf{E}_0' \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	by substitution
$\Gamma; w \vdash \mathsf{E}_0[e] : \forall \alpha.\ \sigma_3 \mid \epsilon \rightsquigarrow \mathsf{E}_0'[e']$	I.H.
$\Gamma; w \vdash \mathsf{E}_0[e]\ [\sigma] : \sigma_3[\alpha{:=}\sigma] \rightsquigarrow \mathsf{E}_0'[e']\ [\lceil \sigma \rceil] \mid \epsilon$	TAPP

case $\mathsf{E} = \mathsf{handle}_w\ h\ \mathsf{E}_0$.

$\Gamma; w \vdash_{ec} \mathsf{handle}_w\ h\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{handle}_m^w\ h'\ \mathsf{E}_0'$	given
$\Gamma;\ \langle\!\langle l{:}(m,h) \mid w \rangle\!\rangle \vdash_{ec} \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \langle l \mid \epsilon \rangle$	CHANDLE
$\Gamma \vdash_{ops} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$	above
$\Gamma;\ \langle\!\langle \lceil \mathsf{handle}_m^w\ h\ \mathsf{E}_0' \rceil \mid w \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{handle}_m^w\ h\ \mathsf{E}_0'\ \rceil^{l} \mid \epsilon \rangle \rightsquigarrow e'$	given
$\Gamma;\ \langle\!\langle \lceil \mathsf{E}_0' \rceil \mid \langle\!\langle l{:}(m,h) \mid w \rangle\!\rangle \rangle\!\rangle \vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^{l} \mid l \mid \epsilon \rangle \rightsquigarrow e'$	by definition
$\Gamma;\ \langle\!\langle l{:}(m,h) \mid w \rangle\!\rangle \vdash_{ec} \mathsf{E}_0[e] : \sigma_2 \mid \langle l \mid \epsilon \rangle \rightsquigarrow \mathsf{E}_0'[e']$	I.H.
$\Gamma; w \vdash \mathsf{handle}_w\ h\ \mathsf{E}_0[e] : \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{handle}_m^w\ h'\ \mathsf{E}_0'[e']$	HANDLE

□

Lemma 17. (*Translation evidence corresponds to the evalution context*)
If $\varnothing;\ w\ \vdash\ \mathsf{E}[e] : \sigma \mid \epsilon\ \rightsquigarrow e_1$ then there exists σ_1, E', e' such that $\varnothing;\ w \vdash_{ec}\ \mathsf{E} : \sigma_1 \to \sigma \mid \epsilon\ \rightsquigarrow \mathsf{E}'$, and $\varnothing;\ \langle\!\langle \lceil \mathsf{E}' \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E}' \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$, and $e_1\ =\ \mathsf{E}'[e']$.

Proof. (*Of Lemma 17*) Induction on E.

case $\mathsf{E}\ =\ \square$. Let $\sigma_1\ =\ \sigma$, $\mathsf{E}' = \square$, $e' = e_1$ and the goal holds trivially.

case $\mathsf{E}\ =\ \mathsf{E}_0\ e_0$.

$\varnothing; w\ \vdash\ \mathsf{E}_0[e]\ e_0 : \sigma \mid \epsilon\ \rightsquigarrow e_2\ w\ e_3$	given
$\varnothing; w\ \vdash\ \mathsf{E}_0[e] : \sigma_2 \to\epsilon\ \sigma \mid \epsilon\ \rightsquigarrow e_2$	APP
$\varnothing; w\ \vdash\ e_0 : \sigma_2 \mid \epsilon\ \rightsquigarrow e_3$	above
$\varnothing; w \vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \to (\sigma_2 \to\epsilon\ \sigma) \mid \epsilon\ \rightsquigarrow \mathsf{E}_0'$	I.H.
$\varnothing; w \vdash_{ec}\ \mathsf{E}_0\ e_0 : \sigma_1 \to \sigma \mid \epsilon\ \rightsquigarrow \mathsf{E}_0'\ w\ e_3$	CAPP1
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}_0' \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$	I.H.
$e_2\ =\ \mathsf{E}_0'[e']$	I.H.
$\lceil \mathsf{E}_0'\ w\ e_3 \rceil\ =\ \lceil \mathsf{E}_0' \rceil$	by definition
$\lceil \mathsf{E}_0'\ w\ e_3 \rceil^l\ =\ \lceil \mathsf{E}_0' \rceil^l$	by definition
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}_0'\ w\ e_3 \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E}_0'\ w\ e_3 \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$	by substitution
$\mathsf{E}'\ =\ \mathsf{E}_0'\ w\ e_3$	Let

case $\mathsf{E}\ =\ v\ \mathsf{E}_0$.

$\varnothing; w\ \vdash\ v\ \mathsf{E}_0[e] : \sigma \mid \epsilon\ \rightsquigarrow e_2\ w\ e_3$	given
$\varnothing; w\ \vdash\ v : \sigma_2 \to\epsilon\ \sigma \mid \epsilon\ \rightsquigarrow e_2$	APP
$\varnothing; w\ \vdash\ \mathsf{E}_0[e] : \sigma_2 \mid \epsilon\ \rightsquigarrow e_3$	above
$\varnothing; w \vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \to \sigma_2 \mid \epsilon\ \rightsquigarrow \mathsf{E}_0'$	I.H.
$\varnothing; w \vdash_{ec}\ v\ \mathsf{E}_0 : \sigma_1 \to \sigma \mid \epsilon\ \rightsquigarrow e_2\ w\ \mathsf{E}_0'$	CAPP2
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$	I.H.
$e_3\ =\ \mathsf{E}_0'[e']$	I.H.
$\lceil e_2\ w\ \mathsf{E}_0 \rceil\ =\ \lceil \mathsf{E}_0 \rceil$	by definition
$\lceil e_2\ w\ \mathsf{E}_0 \rceil^l\ =\ \lceil \mathsf{E}_0 \rceil^l$	by definition
$\varnothing;\ \langle\!\langle \lceil e_2\ w\ \mathsf{E}_0 \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$	by substitution
$\mathsf{E}'\ =\ e_2\ w\ \mathsf{E}_0'$	Let

case $\mathsf{E}\ =\ \mathsf{E}_0\ [\sigma_0]$.

$\varnothing; w\ \vdash\ \mathsf{E}_0[e]\ [\sigma_0] : \sigma_2[\alpha{:=}\sigma_0] \mid \epsilon\ \rightsquigarrow e_2\ [\lceil \sigma_0 \rceil]$	given
$\varnothing; w\ \vdash\ \mathsf{E}_0[e] : \forall\alpha \cdot \sigma_2 \mid \epsilon\ \rightsquigarrow e_2$	TAPP
$\varnothing; w \vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \to (\forall\alpha.\ \sigma_2) \mid \epsilon\ \rightsquigarrow \mathsf{E}_0'$	I.H.
$\varnothing; w \vdash_{ec}\ \mathsf{E}_0\ [\sigma_0] : \sigma_1 \to \sigma_2[\alpha{:=}\sigma_0] \mid \epsilon\ \rightsquigarrow \mathsf{E}_0'\ [\lceil \sigma_0 \rceil]$	CTAPP
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}_0' \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E}_0' \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$	I.H.
$e_2\ =\ \mathsf{E}_0'[e']$	I.H.
$\lceil \mathsf{E}_0'\ [\sigma_0] \rceil\ =\ \lceil \mathsf{E}_0' \rceil$	by definition
$\lceil \mathsf{E}_0'\ [\sigma_0] \rceil^l\ =\ \lceil \mathsf{E}_0' \rceil^l$	by definition
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}_0'\ [\sigma_0] \rceil \mid w \rangle\!\rangle\ \vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle\ \rightsquigarrow e'$	by substitution
$\mathsf{E}'\ =\ \mathsf{E}_0'\ [\lceil \sigma_0 \rceil]$	Let

case $\mathsf{E}\ =\ \mathsf{handle}\ h\ \mathsf{E}_0$.

$\varnothing; w \ \vdash\ \mathsf{handle}\ h\ \mathsf{E}_0[e]\ :\ \sigma \mid \epsilon\ \rightsquigarrow \mathsf{handle}^w_m\ h'\ e_2$	given
$\varnothing; \langle\!\langle l\!:\!(m, h') \mid w\rangle\!\rangle\ \vdash\ \mathsf{E}_0[e]\ :\ \sigma \mid \langle l \mid \epsilon\rangle\ \rightsquigarrow e_2$	HANDLE
$\varnothing; \langle\!\langle l\!:\!(m, h') \mid w\rangle\!\rangle \vdash_{\mathsf{ec}}\ \mathsf{E}_0\ :\ \sigma_1 \to \sigma \mid \langle l \mid \epsilon\rangle\ \rightsquigarrow \mathsf{E}'_0$	I.H.
$\varnothing; w \vdash_{\mathsf{ec}}\ \mathsf{handle}\ h\ \mathsf{E}_0\ :\ \sigma_1 \to \sigma \mid \epsilon\ \rightsquigarrow \mathsf{handle}^w_m\ h'\ \mathsf{E}'_0$	CHANDLE
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}'_0 \rceil \mid \langle\!\langle l\!:\!(m, h') \mid w\rangle\!\rangle\rangle\!\rangle\ \vdash\ e\ :\ \sigma_1 \mid \langle \lceil \mathsf{E}'_0 \rceil^l \mid l \mid \epsilon\rangle\ \rightsquigarrow e'$	I.H.
$e_2\ =\ \mathsf{E}'_0[e']$	I.H.
$\langle\!\langle \lceil \mathsf{E}'_0 \rceil \mid \langle\!\langle l\!:\!(m, h') \mid w\rangle\!\rangle\rangle\!\rangle\ =\ \langle\!\langle \lceil \mathsf{E}'_0 \rceil \mid \langle\!\langle\langle\!\langle l\!:\!(m, h')\rangle\!\rangle \mid w\rangle\!\rangle\rangle\!\rangle$	by definition
$\langle\!\langle \lceil \mathsf{E}'_0 \rceil \mid \langle\!\langle\langle\!\langle l\!:\!(m, h')\rangle\!\rangle \mid w\rangle\!\rangle\rangle\!\rangle\ =\ \langle\!\langle \lceil \mathsf{handle}^w_m\ h' \cdot \mathsf{E}'_0 \rceil \mid w\rangle\!\rangle$	by definition
$\lceil \mathsf{handle}^w_m\ h'\ \mathsf{E}'_0 \rceil^l\ =\ \langle \lceil \mathsf{E}'_0 \rceil^l \mid l\rangle$	by definition
$\varnothing;\ \langle\!\langle \lceil \mathsf{handle}^w_m\ h' \cdot \mathsf{E}'_0 \rceil \mid w\rangle\!\rangle\ \vdash\ e\ :\ \sigma_1 \mid \langle \lceil \mathsf{E}' \rceil^l \mid \epsilon\rangle\ \rightsquigarrow e'$	by substitution
$\mathsf{E}'\ =\ \mathsf{handle}^w_m\ h'\ \mathsf{E}'_0$	Let

□

B.2.3 Substitution.

Lemma 18. (*Translation Variable Substitution*)

1. If $\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w \vdash\ e\ :\ \sigma \mid \epsilon\ \rightsquigarrow\ e'$, and $\Gamma_1,\ \Gamma_2 \vdash_{\mathsf{val}}\ v\ :\ \sigma_1\ \rightsquigarrow v'$, then $\Gamma_1, \Gamma_2\ ; w[x\!:=\!v']\ \vdash\ e[x\!:=\!v]\ :\ \sigma \mid \epsilon\ \rightsquigarrow\ e'[x\!:=\!v']$.
2. If $\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w \vdash_{\mathsf{val}}\ v_0\ :\ \sigma\ \rightsquigarrow\ v'_0$, and $\Gamma_1,\ \Gamma_2 \vdash_{\mathsf{val}}\ v\ :\ \sigma_1\ \rightsquigarrow v'$, then $\Gamma_1, \Gamma_2 \vdash_{\mathsf{val}}\ v[x\!:=\!v]\ :\ \sigma\ \rightsquigarrow\ v'_0[x\!:=\!v']$.
3. If $\Gamma_1, x\!:\!\sigma_1, \Gamma_2 \vdash_{\mathsf{ops}}\ h\ :\ \sigma \mid l\ \rightsquigarrow\ h'$, and $\Gamma_1,\ \Gamma_2 \vdash_{\mathsf{val}}\ v\ :\ \sigma_1\ \rightsquigarrow v'$, then $\Gamma_1, \Gamma_2 \vdash_{\mathsf{ops}}\ h[x\!:=\!v]\ :\ \sigma \mid l\ \rightsquigarrow\ h'[x\!:=\!v']$.

Proof. (*Of Lemma 18*)

Part 1 By induction on translation.

case $e\ =\ v_0$.

$\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w\ \vdash\ v_0\ :\ \sigma \mid \epsilon\ \rightsquigarrow v'_0$	given
$\Gamma_1, x\!:\!\sigma_1, \Gamma_2 \vdash_{\mathsf{val}}\ v_0\ :\ \sigma\ \rightsquigarrow v'_0$	VAR
$\Gamma_1, \Gamma_2 \vdash_{\mathsf{val}}\ v_0[x\!:=\!v]\ :\ \sigma\ \rightsquigarrow v'_0[x\!:=\!v']$	Part 2
$\Gamma_1, \Gamma_2; w[x\!:=\!v]\ \vdash\ v_0[x\!:=\!v]\ :\ \sigma \mid \epsilon\ \rightsquigarrow v'_0[x\!:=\!v']$	VAR

case $e\ =\ e_1\ e_2$.

$\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w\ \vdash\ e_1\ e_2\ :\ \sigma \mid \epsilon\ \rightsquigarrow e'_1\ w\ e'_2$	given
$\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w\ \vdash\ e_1\ :\ \sigma_1 \to \epsilon\ \sigma \mid \epsilon\ \rightsquigarrow e'_1$	APP
$\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w\ \vdash\ e_2\ :\ \sigma_1\ \mid \epsilon\ \rightsquigarrow e'_1$	APP
$\Gamma_1, \Gamma_2; w[x\!:=\!v']\ \vdash\ e_1[x\!:=\!v]\ :\ \sigma_1 \to \epsilon\ \sigma \mid \epsilon\ \rightsquigarrow e'_1[x\!:=\!v']$	I.H.
$\Gamma_1, \Gamma_2; w[x\!:=\!v']\ \vdash\ e_2[x\!:=\!v]\ :\ \sigma_1 \mid \epsilon\ \rightsquigarrow e'_2[x\!:=\!v']$	I.H.
$\Gamma_1, \Gamma_2; w[x\!:=\!v']\ \vdash\ e_1[x\!:=\!v]\ e_2[x\!:=\!v]\ :\ \sigma \mid \epsilon\ \rightsquigarrow e'_1[x\!:=\!v']\ w[x\!:=\!v']\ e'_2[x\!:=\!v']$	APP

case $e\ =\ e_1\ [\sigma]$.

$\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w\ \vdash\ e_1\ [\sigma]\ :\ \sigma_1[\alpha\!:=\!\sigma]\ \mid \epsilon\ \rightsquigarrow e'_1\ [\lceil\sigma\rceil]$	given
$\Gamma_1, x\!:\!\sigma_1, \Gamma_2; w\ \vdash\ e_1\ :\ \forall\alpha.\ \sigma_1 \mid \epsilon\ \rightsquigarrow e'_1$	TAPP
$\Gamma_1, \Gamma_2; w[x\!:=\!v']\ \vdash\ e_1[x\!:=\!v]\ :\ \forall\alpha.\ \sigma_1 \mid \epsilon\ \rightsquigarrow e'_1[x\!:=\!v']$	I.H.
$\Gamma_1, \Gamma_2; w[x\!:=\!v']\ \vdash\ e_1[x\!:=\!v]\ [\sigma]\ :\ \sigma_1[\alpha\!:=\!\sigma] \mid \epsilon\ \rightsquigarrow e'_1[x\!:=\!v']\ [\lceil\sigma\rceil]$	TAPP

case $e\ =\ \mathsf{handle}^\epsilon\ h\ e$.

$\Gamma_1, x : \sigma_1, \Gamma_2; w \vdash \text{handle}^{\epsilon}\ h\ e : \sigma \mid \epsilon \rightsquigarrow \text{handle}^{w}_{m}\ h'\ e'$ — given
$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ — HANDLE
$\Gamma_1, x : \sigma_1, \Gamma_2; \langle\!\langle l : (m, h') \mid w \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'$ — above
$\Gamma_1, \Gamma_2 \vdash_{\text{ops}} h[x{:=}v] : \sigma \mid l \mid \epsilon \rightsquigarrow h'[x{:=}v']$ — Part 3
$\Gamma_1, \Gamma_2; \langle\!\langle l : (m, h'[x{:=}v']) \mid w[x{:=}v'] \rangle\!\rangle \vdash e[x{:=}v] : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'[x{:=}v']$ — I.H.
$\Gamma_1, \Gamma_2; w[x{:=}v'] \vdash \text{handle}^{\epsilon}\ h[x{:=}v]\ e[x{:=}v] : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow \text{handle}^{w}_{m}\ h'[x{:=}v']\ e'[x{:=}v']$ — HANDLE

Part 2 By induction on translation.

case $v_0 = x$.

$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} x : \sigma_1 \rightsquigarrow x$ — given
$x[x{:=}v] = v$ — by substitution
$x[x{:=}v'] = v'$ — by substitution
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} v : \sigma_1 \rightsquigarrow v'$ — given
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} x[x{:=}v] : \sigma_1 \rightsquigarrow x[x{:=}v']$ — follows

case $v_0 = y$ and $y \neq x$.

$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} y : \sigma \rightsquigarrow y$ — given
$y : \sigma \in \Gamma_1,\ x : \sigma_1,\ \Gamma_2$ — VAR
$y \neq x$ — given
$y : \sigma \in \Gamma_1,\ \Gamma_2$ — follows
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} y : \sigma \rightsquigarrow y$ — VAR
$y[x{:=}v] = y$ — by substitution
$y[x{:=}v'] = y$ — by substitution
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} y[x{:=}v] : \sigma \rightsquigarrow y[x{:=}v']$ — VAR

case $v_0 = \lambda^{\epsilon}\ y^{\sigma_2}.\ e$.

$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} \lambda^{\epsilon}\ y^{\sigma_2}.\ e : \sigma_2 \rightarrow \sigma_3 \rightsquigarrow \lambda^{\epsilon}\ z : \text{evv}\ \epsilon,\ y : \lceil \sigma_2 \rceil.\ e'$ — given
$\Gamma_1, x : \sigma_1, \Gamma_2, y : \sigma_2; z \vdash e : \sigma_3 \mid \epsilon \rightsquigarrow e'$ — ABS
$\Gamma_1, \Gamma_2, y : \sigma_2; z \vdash e[x{:=}v] : \sigma_3 \mid \epsilon \rightsquigarrow e'[x{:=}v']$ — Part 1
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} \lambda^{\epsilon}\ y^{\sigma_2}.\ e[x{:=}v] : \sigma_2 \rightarrow \sigma_3 \rightsquigarrow \lambda^{\epsilon}\ z : \text{evv}\ \epsilon,\ y : \lceil \sigma_2 \rceil.\ e'[x{:=}v']$ — ABS

case $v_0 = \Lambda \alpha^k.\ v_1$.

$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} \Lambda \alpha^k.\ v_1 : \forall \alpha^k.\ \sigma_2 \rightsquigarrow \Lambda \alpha^k.\ v_1'$ — given
$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} v_1 : \sigma_2 \rightsquigarrow v_1'$ — TABS
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} v_1[x{:=}v] : \sigma_2 \rightsquigarrow v_1'[x{:=}v']$ — I.H.
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} \Lambda \alpha^k.\ v_1[x{:=}v] : \forall \alpha^k.\ \sigma_2 \rightsquigarrow \Lambda \alpha^k.\ v_1'[x{:=}v']$ — TABS

case $v_0 = \text{perform}\ op\ \overline{\sigma}$.

$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} \text{perform}\ op\ \overline{\sigma} : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma_3[\overline{\alpha}{:=}\overline{\sigma}] \rightsquigarrow \text{perform}\ op\ \lceil \overline{\sigma} \rceil$ — given
$op : \forall \overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2 \in \Sigma(l)$ — PERFORM
$(\text{perform}\ op\ \overline{\sigma})[x{:=}v] = \text{perform}\ op\ \overline{\sigma}$ — by substitution
$(\text{perform}\ op\ \lceil \overline{\sigma} \rceil)[x{:=}v'] = \text{perform}\ op\ \lceil \overline{\sigma} \rceil$ — by substitution
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} (\text{perform}\ op\ \overline{\sigma})[x{:=}v] : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma_3[\overline{\alpha}{:=}\overline{\sigma}]$ — PERFORM
$\quad \rightsquigarrow (\text{perform}\ op\ \lceil \overline{\sigma} \rceil)[x{:=}v']$

case $v_0 = \text{handler}^{\epsilon}\ h$.

$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{val}} \text{handler}^{\epsilon}\ h : \sigma \rightsquigarrow \text{handler}^{w}_{m}\ h'$ — given
$\Gamma_1, x : \sigma_1, \Gamma_2 \vdash_{\text{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ — HANDLER
$\Gamma_1, \Gamma_2 \vdash_{\text{ops}} h[x{:=}v] : \sigma \mid l \mid \epsilon \rightsquigarrow h'[x{:=}v']$ — Part 3
$\Gamma_1, \Gamma_2 \vdash_{\text{val}} \text{handler}^{\epsilon}\ h[x{:=}v] : \sigma \rightsquigarrow \text{handler}^{w}_{m}\ h'[x{:=}v']$ — HANDLER

Part 3 Follows directly from Part 2.

□

Lemma 19. (*Translation Evidence Variable Substitution*)
If $\Gamma; w \vdash e : \sigma \mid \epsilon \rightsquigarrow e'$ and $z \notin \Gamma$, then $\Gamma; w[z{:=}w_1] \vdash e : \sigma \mid \epsilon \rightsquigarrow e'[z{:=}w_1]$.

Proof. (*Of Lemma 19*) By induction on the typing.

case $e = v_0$.

$\Gamma; w \vdash v : \sigma \mid \epsilon \rightsquigarrow v'$	given
$\Gamma \vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'$	VAR
$z \notin \Gamma$	given
$v'[z{:=}w_1] = v'$	z out of scope of v'
$\Gamma; w[z{:=}w_1] \vdash v : \sigma \mid \epsilon \rightsquigarrow v'$	Lemma 25

case $e = e_1\ e_2$.

$\Gamma; w \vdash e_1\ e_2 : \sigma \mid \epsilon \rightsquigarrow e_1'\ w\ e_2'$	given
$\Gamma; w \vdash e_1 : \sigma_1 \rightarrow \epsilon\ \sigma \mid \epsilon \rightsquigarrow e_1'$	APP
$\Gamma; w \vdash e_2 : \sigma_1 \mid \epsilon \rightsquigarrow e_1'$	APP
$\Gamma; w[z{:=}w_1] \vdash e_1 : \sigma_1 \rightarrow \epsilon\ \sigma \mid \epsilon \rightsquigarrow e_1'[z{:=}w_1]$	I.H.
$\Gamma; w[z{:=}w_1] \vdash e_2 : \sigma_1 \mid \epsilon \rightsquigarrow e_2'[z{:=}w_1]$	I.H.
$\Gamma; w[z{:=}w_1] \vdash e_1\ e_2 : \sigma \mid \epsilon \rightsquigarrow e_1'[z{:=}w_1]\ w[z{:=}w_1]\ e_2'[z{:=}w_1]$	APP

case $e = e_1\ [\sigma]$.

$\Gamma; w \vdash e_1\ [\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow e_1'\ [\lceil\sigma\rceil]$	given
$\Gamma; w \vdash e_1 : \forall\alpha.\ \sigma_1 \mid \epsilon \rightsquigarrow e_1'$	TAPP
$\Gamma; w[z{:=}w_1] \vdash e_1 : \forall\alpha.\ \sigma_1 \mid \epsilon \rightsquigarrow e_1'[z{:=}w_1]$	I.H.
$\Gamma; w[z{:=}w_1] \vdash e_1\ [\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow e_1'[z{:=}w_1]\ [\lceil\sigma\rceil]$	TAPP

case $e = \mathsf{handle}^{\epsilon}\ h\ e$.

$\Gamma; w \vdash \mathsf{handle}^{\epsilon}\ h\ e : \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}^{w}_{m}\ h'\ e'$	given
$\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$	HANDLE
$\Gamma;\ \langle\!\langle l : (m, h') \mid w \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'$	above
$v \notin \Gamma$	given
$h'[z{:=}w] = h'$	z out of scope of z'
$\Gamma;\ \langle\!\langle l : (m, h'[z{:=}w]) \mid w[x{:=}v'] \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'[z{:=}w]$	I.H.
$\Gamma;\ \langle\!\langle l : (m, h') \mid w[x{:=}v'] \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'[z{:=}w]$	namely
$\Gamma;\ w[x{:=}v'] \vdash \mathsf{handle}^{\epsilon}\ h\ e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow \mathsf{handle}^{w}_{m}\ h'\ e'[x{:=}v']$	HANDLE

□

Lemma 20. (*Translation Type Variable Substitution*)
1. If $\Gamma; w \vdash e : \sigma \mid \epsilon \rightsquigarrow e'$ and $\vdash_{\mathsf{wf}} \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid \epsilon \rightsquigarrow e'[\alpha^k{:=}\lceil\sigma_1\rceil]$.
2. If $\Gamma \vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'$ and $\vdash_{\mathsf{wf}} \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} v[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \rightsquigarrow v'[\alpha^k{:=}\lceil\sigma_1\rceil]$.
3. If $\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ and $\vdash_{\mathsf{wf}} \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{ops}} h[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid l \mid \epsilon \rightsquigarrow v'[\alpha^k{:=}\lceil\sigma_1\rceil]$.

Proof. (*Of Lemma 20*) **Part 1** By induction on translation.

case $e = v_0$.

$\Gamma; w \vdash v_0 : \sigma \mid \epsilon \rightsquigarrow v_0'$	given
$\Gamma \vdash_{\mathsf{val}} v_0 : \sigma \rightsquigarrow v_0'$	VAR
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} v_0[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \rightsquigarrow v_0'[\alpha^k{:=}\lceil\sigma_1\rceil]$	Part 2
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash v_0[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid \epsilon \rightsquigarrow v_0'[\alpha^k{:=}\lceil\sigma_1\rceil]$	VAR

case $e = e_1\ e_2$.

$\Gamma; w \vdash e_1\ e_2 : \sigma \mid \epsilon \rightsquigarrow e_1'\ w\ e_2'$ given
$\Gamma; w \vdash e_1 : \sigma_1 \rightarrow \epsilon\ \sigma \mid \epsilon \rightsquigarrow e_1'$ APP
$\Gamma; w \vdash e_2 : \sigma_1 \mid \epsilon \rightsquigarrow e_1'$ APP
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e_1[\alpha^k{:=}\sigma_1] : \sigma_1[\alpha^k{:=}\sigma_1] \rightarrow \epsilon\ \sigma[\alpha^k{:=}\sigma_1] \mid \epsilon \rightsquigarrow e_1'[\alpha^k{:=}\lceil\sigma_1\rceil]$ I.H.
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e_2[\alpha^k{:=}\sigma_1] : \sigma_1[\alpha^k{:=}\sigma_1] \mid \epsilon \rightsquigarrow e_2'[\alpha^k{:=}\lceil\sigma_1\rceil]$ I.H.
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e_1[\alpha^k{:=}\sigma_1]\ e_2[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\lceil\sigma_1\rceil] \mid \epsilon$ APP
$\quad \rightsquigarrow e_1'[\alpha^k{:=}\lceil\sigma_1\rceil]\ w[\alpha^k{:=}\lceil\sigma_1\rceil]\ e_2'[\alpha^k{:=}\lceil\sigma_1\rceil]$

case $e = e_1\ [\sigma]$.

$\Gamma; w \vdash e_1\ [\sigma] : \sigma_2[\beta{:=}\sigma] \mid \epsilon \rightsquigarrow e_1'\ [\lceil\sigma\rceil]$ given
$\Gamma; w \vdash e_1 : \forall\beta.\ \sigma_2 \mid \epsilon \rightsquigarrow e_1'$ TAPP
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e_1[\alpha^k{:=}\sigma_1] : \forall\beta.\ \sigma_2[\alpha^k{:=}\sigma_1] \mid \epsilon \rightsquigarrow e_1'[\alpha^k{:=}\lceil\sigma_1\rceil]$ I.H.
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e_1[\alpha^k{:=}\sigma_1]\ [\sigma[\alpha^k{:=}\sigma_1]\ ] : (\sigma_2[\alpha^k{:=}\sigma_1])[\beta{:=}\sigma] \mid \epsilon$ TAPP
$\quad \rightsquigarrow e_1'[\alpha^k{:=}\lceil\sigma_1\rceil]\ [\lceil\sigma[\alpha^k{:=}\sigma_1]\rceil]$
$(\sigma_2[\alpha^k{:=}\sigma_1])[\beta{:=}\sigma]$
$= (\sigma_2[\beta{:=}\sigma])[\alpha^k{:=}(\sigma_1[\beta{:=}\sigma])]$ by substitution
$= (\sigma_2[\beta{:=}\sigma])[\alpha^k{:=}\sigma_1]$ β fresh to σ_1
$\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\lceil\sigma_1\rceil] \vdash e_1[\alpha^k{:=}\sigma_1]\ [\sigma[\alpha^k{:=}\sigma_1]\ ] : (\sigma_2[\beta{:=}\sigma])[\alpha^k{:=}\sigma_1] \mid \epsilon$ therefore
$\quad \rightsquigarrow e_1'[\alpha^k{:=}\lceil\sigma_1\rceil]\ [\lceil\sigma[\alpha^k{:=}\sigma_1]\rceil]$

case $e = \mathsf{handle}^\epsilon\ h\ e$.

$\Gamma; w \vdash \mathsf{handle}^\epsilon\ h\ e : \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}_m^w\ h'\ e'$ given
$\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ HANDLE
$\Gamma; \langle\!\langle l : (m, h') \mid w \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon\rangle \rightsquigarrow e'$ above
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{ops}} h[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid l \mid \epsilon \rightsquigarrow h'[\alpha^k{:=}\sigma_1]$ Part 3
$\Gamma[\alpha^k{:=}\sigma_1]; \langle\!\langle l : (m, h'[\alpha^k{:=}\sigma_1]) \mid w[\alpha^k{:=}\sigma_1] \rangle\!\rangle \vdash e[\alpha^k{:=}\sigma_1] : \sigma \mid \langle l \mid \epsilon\rangle \rightsquigarrow e'[\alpha^k{:=}\sigma_1]$ I.H.
$\Gamma[\alpha^k{:=}\sigma_1];\ w[\alpha^k{:=}\sigma_1] \vdash \mathsf{handle}^\epsilon\ h[\alpha^k{:=}\sigma_1]\ e[\alpha^k{:=}\sigma_1] : \sigma \mid \langle l \mid \epsilon\rangle$ HANDLE
$\quad \rightsquigarrow \mathsf{handle}_m^w\ h'[\alpha^k{:=}\sigma_1]\ e'[\alpha^k{:=}\sigma_1]$

Part 2 By induction on translation.

case $v = x$.

$\Gamma \vdash_{\mathsf{val}} x : \sigma \rightsquigarrow x$ given
$x : \sigma \in \Gamma$ VAR
$x : \sigma[\alpha^k{:=}\sigma_1] \in \Gamma[\alpha^k{:=}\sigma_1]$ therefore
$x[\alpha^k{:=}\sigma_1] = x$ by substitution
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} x : \sigma[\alpha^k{:=}\sigma_1] \rightsquigarrow x$ VAR
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} x[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \rightsquigarrow x[\alpha^k{:=}\sigma_1]$ follows

case $v = \lambda^\epsilon\ y^{\sigma_2}.\ e$.

$\Gamma \vdash_{\mathsf{val}} \lambda^\epsilon\ y^{\sigma_2}.\ e : \sigma_2 \rightarrow \sigma_3 \rightsquigarrow \lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ y{:}\lceil\sigma_2\rceil.\ e'$ given
$\Gamma, y{:}\sigma_2;\ z \vdash e : \sigma_3 \mid \epsilon \rightsquigarrow e'$ ABS
$\Gamma[\alpha^k{:=}\sigma_1], y{:}\sigma_2[\alpha^k{:=}\sigma_1];\ z[\alpha^k{:=}\sigma_1] \vdash e[\alpha^k{:=}\sigma_1] : \sigma_3[\alpha^k{:=}\sigma_1] \mid \epsilon \rightsquigarrow e'[\alpha^k{:=}\lceil\sigma_1\rceil]$ Part 1
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} \lambda^\epsilon\ y^{\sigma_2[\alpha^k{:=}\sigma_1]}.\ e[\alpha^k{:=}\sigma_1] : \sigma_2[\alpha^k{:=}\sigma_1] \rightarrow \sigma_3[\alpha^k{:=}\sigma_1]$ ABS
$\quad \rightsquigarrow \lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ y{:}\lceil\sigma_2[\alpha^k{:=}\sigma_1]\rceil.\ e'[\alpha^k{:=}\lceil\sigma_1\rceil]$
$\lceil\sigma_2[\alpha^k{:=}\sigma_1]\rceil = \lceil\sigma_2\rceil[\alpha^k := \lceil\sigma_1\rceil]$ Lemma 15
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} \lambda^\epsilon\ y^{\sigma_2[\alpha^k{:=}\sigma_1]}.\ e[\alpha^k{:=}\sigma_1] : \sigma_2[\alpha^k{:=}\sigma_1] \rightarrow \sigma_3[\alpha^k{:=}\sigma_1]$ ABS
$\quad \rightsquigarrow \lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ y{:}\lceil\sigma_2\rceil[\alpha^k := \lceil\sigma_1\rceil].\ e'[\alpha^k{:=}\lceil\sigma_1\rceil]$

case $v = \Lambda\beta^k.\ v_1$.

$\Gamma \vdash_{\mathsf{val}} \Lambda \beta^k.\ v_1 : \forall \beta^k.\ \sigma_2 \rightsquigarrow \Lambda \beta^k.\ v_1'$ given
$\Gamma \vdash_{\mathsf{val}} v_1 : \sigma_2 \rightsquigarrow v_1'$ TABS
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} v_1[\alpha^k{:=}\sigma_1] : \sigma_2[\alpha^k{:=}\sigma_1] \rightsquigarrow v_1'[\alpha^k{:=}\lceil\sigma_1\rceil]$ I.H.
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} \Lambda \beta^k.\ v_1[\alpha^k{:=}\sigma_1] : \forall \beta^k.\ \sigma_2[\alpha^k{:=}\sigma_1] \rightsquigarrow \Lambda \beta^k.\ v_1'[\alpha^k{:=}\lceil\sigma_1\rceil]$ TABS

case $v = \mathsf{perform}\ op\ \overline{\sigma}$.

$\Gamma \vdash_{\mathsf{val}} \mathsf{perform}\ op\ \overline{\sigma} : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma_3[\overline{\alpha}{:=}\overline{\sigma}] \rightsquigarrow \mathsf{perform}\ op\ \lceil\overline{\sigma}\rceil$ given
$op : \forall \overline{\alpha}.\ \sigma_2 \rightarrow \sigma_3 \in \Sigma(l)$ PERFORM
$(\mathsf{perform}\ op\ \overline{\sigma})[\alpha^k{:=}\sigma_1] = \mathsf{perform}\ op\ \overline{\sigma}[\alpha^k{:=}\sigma_1]$ by substitution
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} \mathsf{perform}\ op\ \overline{\sigma}[\alpha^k{:=}\sigma_1]$ PERFORM
$\quad : \sigma_2[\overline{\alpha}{:=}(\overline{\sigma}[\alpha^k{:=}\sigma_1])] \rightarrow \sigma_3[\overline{\alpha}{:=}(\overline{\sigma}[\alpha^k{:=}\sigma_1])]$
$\quad \rightsquigarrow \mathsf{perform}\ op\ \lceil\overline{\sigma}[\alpha^k{:=}\sigma_1]\rceil$
$\sigma_2[\overline{\alpha}{:=}(\overline{\sigma}[\alpha^k{:=}\sigma_1])]$
$= (\sigma_2[\alpha^k{:=}\sigma_1])[\overline{\alpha}{:=}(\overline{\sigma}[\alpha^k{:=}\sigma_1])]$ α fresh to σ_2
$= (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])[\alpha^k{:=}\sigma_1]$ by substitution
$\sigma_3[\overline{\alpha}{:=}(\overline{\sigma}[\alpha^k{:=}\sigma_1])] = (\sigma_3[\overline{\alpha}{:=}\overline{\sigma}])[\alpha^k{:=}\sigma_1]$ similarly
$\lceil\overline{\sigma}[\alpha^k{:=}\sigma_1]\rceil = \lceil\overline{\sigma}\rceil[\alpha^k{:=}\lceil\sigma_1\rceil]$ Lemma 15
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} \mathsf{perform}\ op\ \overline{\sigma}[\alpha^k{:=}\sigma_1]$ therefore
$\quad : (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])[\alpha^k{:=}\sigma_1] \rightarrow (\sigma_3[\overline{\alpha}{:=}\overline{\sigma}])[\alpha^k{:=}\sigma_1]$
$\quad \rightsquigarrow \mathsf{perform}\ op\ \lceil\overline{\sigma}\rceil[\alpha^k{:=}\lceil\sigma_1\rceil]$

case $v = \mathsf{handler}^{\epsilon}\ h$.

$\Gamma \vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h : \sigma \rightsquigarrow \mathsf{handler}^{w}_{m}\ h'$ given
$\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ HANDLER
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{ops}} h[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid l \mid \epsilon \rightsquigarrow h'[\alpha^k{:=}\lceil\sigma_1\rceil]$ Part 3
$\Gamma[\alpha^k{:=}\sigma_1] \vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \rightsquigarrow \mathsf{handler}^{w}_{m}\ h'[\alpha^k{:=}\lceil\sigma_1\rceil]$ HANDLER

Part 3 Follows directly from Part 2.

□

B.2.4 Translation Soundness.

Proof. (*Of Theorem 4*) Apply Lemma 21 with $\Gamma = \varnothing$ and $w = \langle\!\langle\rangle\!\rangle$. □

Lemma 21. (*Evidence translation respects evidence typing with contexts*)
We use $\lceil w \rceil^{\epsilon}$ to mean that we extract from w all evidence variables, who get their types by inspecting ϵ. So we have:

1. If $\Gamma \vdash_{\mathsf{val}} v : \sigma \mid \epsilon \rightsquigarrow v'$ then $\lceil\Gamma\rceil \Vdash_{\mathsf{val}} v' : \lceil\sigma\rceil$.
2. If $\Gamma; w \vdash e : \sigma \mid \epsilon \rightsquigarrow e'$ then $\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon} ; w \Vdash e' : \lceil\sigma\rceil \mid \epsilon$.
3. If $\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon$, then $\lceil\Gamma\rceil \Vdash_{\mathsf{ops}} \lceil h\rceil : \lceil\sigma\rceil \mid l \mid \epsilon$.
4. If $\Gamma; w \vdash \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'$ then $\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon} ; w \vdash \mathsf{E}' : \lceil\sigma_1\rceil \rightarrow \lceil\sigma_2\rceil$.

Proof. (*Of Lemma 21*) **Part 1** By induction on translation.

case $v = x$.

$\Gamma \vdash_{\mathsf{val}} x : \sigma \mid \epsilon \rightsquigarrow x$ given
$x : \sigma \in \Gamma$ VAR
$x : \lceil\sigma\rceil \in \lceil\Gamma\rceil$
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} x : \lceil\sigma\rceil$ MVAR

case $v = \lambda^{\epsilon}\ x^{\sigma_1}.\ e$.

$\Gamma \vdash_{\mathsf{val}} \lambda^{\epsilon}\, x^{\sigma_1}.\, e : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \;\rightsquigarrow \lambda^{\epsilon}\, z^{\,\mathsf{evv}\;\epsilon},\, x^{\,\sigma_1} \cdot e'$ given
$\Gamma, x : \sigma_1 \,;\, z \vdash e : \sigma_2 \mid \epsilon \;\rightsquigarrow e'$ ABS
$\lceil\Gamma\rceil,\ x : \lceil\sigma_1\rceil,\ z : \mathsf{evv}\;\epsilon \,;\, z \Vdash e' : \lceil\sigma_2\rceil \mid \epsilon$ Part 2
$\lceil\Gamma\rceil \Vdash \lambda^{\epsilon}\, z^{\mathsf{evv}\;\epsilon},\ x : \lceil\sigma_1\rceil.\ e' : \sigma_1 \Rightarrow \epsilon\ \sigma_2$ MABS

case $v = \Lambda\alpha.\ v_0$.

$\Gamma \vdash_{\mathsf{val}} \Lambda\alpha.v_0 : \forall\alpha.\ \sigma_0 \;\rightsquigarrow \Lambda\alpha.v_0'$ given
$\Gamma \vdash_{\mathsf{val}} v_0 : \sigma_0 \;\rightsquigarrow v_0'$ TABS
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} v_0' : \lceil\sigma_0\rceil$ I.H.
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} \Lambda\alpha.\ v_0' : \forall\alpha.\ \lceil\sigma_0\rceil$ MTABS

case $v = \mathsf{perform}\ op$.

$\Gamma \vdash_{\mathsf{val}} \mathsf{perform}\ op : \forall\mu\,\overline{\alpha}.\ \sigma_1 \rightarrow \langle l \mid \mu\rangle\ \sigma_2 \;\rightsquigarrow \mathsf{perform}\ op$ given
$op : \forall\overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2\ \in \Sigma(l)$ PERFORM
$op : \forall\overline{\alpha}.\ \lceil\sigma_1\rceil \rightarrow \lceil\sigma_2\rceil\ \in \lceil\Sigma\rceil(l)$
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} \mathsf{perform}\ op : \forall\mu\,\overline{\alpha}.\ \lceil\sigma_1\rceil \Rightarrow \langle l \mid \mu\rangle\ \lceil\sigma_2\rceil$ MPERFORM

case $v = \mathsf{handler}^{\epsilon}\ h$.

$\Gamma \vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h : (() \rightarrow \langle l \mid \epsilon\rangle\sigma) \rightarrow \epsilon\ \sigma \;\rightsquigarrow \mathsf{handler}\ h'$ given
$\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \;\rightsquigarrow h'$ HANDLER
$\lceil\Gamma\rceil \Vdash_{\mathsf{ops}} h' : \lceil\sigma\rceil \mid l \mid \epsilon$ Part 3
$\Gamma \Vdash_{\mathsf{val}} \mathsf{handler}^{\epsilon}\ h' : (() \Rightarrow \langle l \mid \epsilon\rangle\ \lceil\sigma\rceil) \Rightarrow \epsilon\ \lceil\sigma\rceil$ MHANDLER

Part 2 By induction on translation.

case $e = v$.

$\Gamma; w \vdash v : \sigma \mid \epsilon \;\rightsquigarrow v'$ given
$\Gamma \vdash_{\mathsf{val}} v : \sigma \;\rightsquigarrow v'$ VAR
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} v' : \lceil\sigma\rceil$ Part 1
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon} \Vdash_{\mathsf{val}} v' : \lceil\sigma\rceil$ weakening
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon} \,;\, w \Vdash v' : \lceil\sigma\rceil \mid \epsilon$ MVAR

case $e = e_1\ e_2$.

$\Gamma; w \vdash e_1\ e_2 : \sigma \mid \epsilon \;\rightsquigarrow e_1'\ w\ e_2'$ given
$\Gamma; w \vdash e_1 : \sigma_1 \rightarrow \epsilon\ \sigma \mid \epsilon \;\rightsquigarrow e_1'$ APP
$\Gamma; w \vdash e_2 : \sigma_1 \mid \epsilon \;\rightsquigarrow e_1'$ APP
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon}; w \vdash e_1' : \lceil\sigma_1\rceil \Rightarrow \epsilon\ \lceil\sigma\rceil \mid \epsilon$ I.H.
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon}; w \vdash e_2' : \lceil\sigma_1\rceil \mid \epsilon \;\rightsquigarrow e_2'$ I.H.
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon}; w \Vdash e_1'\ w\ e_2' : \lceil\sigma\rceil \mid \epsilon$ MAPP

case $e = e_1\ [\sigma]$.

$\Gamma; w \vdash e_1\ [\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \;\rightsquigarrow e_1'\ [\lceil\sigma\rceil]$ given
$\Gamma; w \vdash e_1 : \forall\alpha.\ \sigma_1 \mid \epsilon \;\rightsquigarrow e_1'$ TAPP
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon}; w \Vdash e_1' : \forall\alpha.\ \lceil\sigma_1\rceil \mid \epsilon$ I.H.
$\lceil\Gamma\rceil,\ \lceil w\rceil^{\epsilon}; w \Vdash e_1'\ [\lceil\sigma\rceil] : \lceil\sigma_1\rceil[\alpha{:=}\lceil\sigma\rceil]$ MTAPP
$\lceil\sigma_1\rceil[\alpha{:=}\lceil\sigma\rceil] = \lceil\sigma_1[\alpha{:=}\sigma]\rceil$ Lemma 15

case $e = \mathsf{handle}^{\epsilon}\ h\ e$.

$\Gamma; w \vdash \mathsf{handle}^{\epsilon}\ h\ e : \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}^{w}_{m}\ h'\ e'$	given
$\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$	HANDLE
$\Gamma; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'$	above
$\lceil\Gamma\rceil \Vdash_{\mathsf{ops}} h' : \lceil\sigma\rceil \mid l \mid \epsilon \rightsquigarrow h'$	Part 3
$\lceil\Gamma\rceil, \lceil\langle\!\langle (l : (m, h) \mid w) \rangle\!\rangle\rceil^{\langle l \mid \epsilon\rangle} \Vdash e' : \lceil\sigma\rceil \mid \langle l \mid \epsilon \rangle$	I.H.
$\lceil\langle\!\langle (l : (m, h) \mid w) \rangle\!\rangle\rceil^{\langle l \mid \epsilon\rangle} = \lceil\langle\!\langle w \rangle\!\rangle\rceil^{\epsilon}$	by definition
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash \mathsf{handle}^{w}_{m}\ \lceil h\rceil\ \lceil e\rceil : \lceil\sigma\rceil \mid \epsilon$	MHANDLE

Part 3

$\Gamma \vdash_{\mathsf{ops}} \{\, op_1 \to f_1, \ldots, op_n \to f_n \,\} : \sigma \mid l \mid \epsilon \rightsquigarrow \{\, op_i \to f_i' \,\}$	given
$op_i : \forall\overline{\alpha}.\ \sigma_1 \to \sigma_2 \in \Sigma(l) \quad \overline{\alpha} \not\in \mathrm{ftv}(\epsilon\ \sigma)$	OPS
$\Gamma \vdash_{\mathsf{val}} f_i : \forall\overline{\alpha}.\ \sigma_1 \to \epsilon\ (\sigma_2 \to \epsilon\ \sigma) \to \epsilon\ \sigma \rightsquigarrow f_i'$	above
$op_i : \forall\overline{\alpha}.\ \lceil\sigma_1\rceil \to \lceil\sigma_2\rceil \in \lceil\Sigma\rceil(l)$	
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} \lceil f_i'\rceil : \forall\overline{\alpha}.\ \lceil\sigma_1\rceil \Rightarrow \epsilon\ (\lceil\sigma_2\rceil \Rightarrow \epsilon\ \lceil\sigma\rceil) \Rightarrow \epsilon\ \lceil\sigma\rceil$	Part 1
$\lceil\Gamma\rceil \Vdash_{\mathsf{ops}} \{\, op_1 \to f_1', \ldots, op_n \to f_n' \,\} : \lceil\sigma\rceil \mid l \mid \epsilon$	MOPS

Part 4 By induction on translation.

case $\mathsf{E} = \square$. The goal follows trivially by MON-CEMPTY.

case $\mathsf{E} = \mathsf{E}_0\ e$.

$\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E}_0\ e : \sigma_1 \to \sigma_3 \mid \epsilon \rightsquigarrow \mathsf{E}'\ w\ e'$	given
$\Gamma; w \vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e'$	CAPP1
$\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E}_0 : \sigma_1 \to (\sigma_2 \to \epsilon\ \sigma_3) \mid \epsilon \rightsquigarrow \mathsf{E}'$	above
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash e' : \lceil\sigma_2\rceil \mid \epsilon$	Part 2
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} \mathsf{E}' : \lceil\sigma_1\rceil \to (\lceil\sigma_2\rceil \Rightarrow \epsilon\ \lceil\sigma_3\rceil) \mid \epsilon$	I.H.
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} \mathsf{E}'\ w\ e' : \lceil\sigma_1\rceil \to \lceil\sigma_3\rceil \mid \epsilon$	MON-CAPP1

case $\mathsf{E} = v\ \mathsf{E}_0$.

$\Gamma; w \vdash_{\mathsf{ec}} v\ \mathsf{E}_0 : \sigma_1 \to \sigma_3 \mid \epsilon \rightsquigarrow v'\ w\ \mathsf{E}'$	given
$\Gamma \vdash_{\mathsf{val}} v : \sigma_2 \to \epsilon\ \sigma_3 \rightsquigarrow v'$	CAPP2
$\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'$	above
$\lceil\Gamma\rceil \Vdash_{\mathsf{val}} v' : \lceil\sigma_2\rceil \Rightarrow \epsilon\ \lceil\sigma_3\rceil$	Part 1
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} \mathsf{E}' : \lceil\sigma_1\rceil \to \lceil\sigma_2\rceil \mid \epsilon$	I.H.
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} v'\ w\ \mathsf{E}' : \lceil\sigma_1\rceil \to \lceil\sigma_3\rceil \mid \epsilon$	MON-CAPP2

case $\mathsf{E} = \mathsf{E}_0\ [\sigma]$.

$\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E}_0\ [\sigma] : \sigma_1 \to \sigma_2\ [\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow \mathsf{E}'\ [\lceil\sigma\rceil]$	given
$\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E}_0 : \sigma_1 \to \forall\alpha.\ \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}'$	CTAPP
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} \mathsf{E}' : \lceil\sigma_1\rceil \to \forall\alpha.\ \lceil\sigma_2\rceil \mid \epsilon$	I.H.
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} \mathsf{E}'\ [\lceil\sigma\rceil] : \lceil\sigma_1\rceil \to \lceil\sigma_2\rceil[\alpha{:=}\lceil\sigma\rceil] \mid \epsilon$	MON-CTAPP
$\lceil\sigma_2\rceil[\alpha{:=}\lceil\sigma\rceil] = \lceil\sigma_2[\alpha{:=}\sigma]\rceil$	Lemma 15

case $\mathsf{E} = \mathsf{handle}^{\epsilon}\ h\ \mathsf{E}_0$.

$\Gamma; w \vdash_{\mathsf{ec}} \mathsf{handle}^{\epsilon}\ h\ \mathsf{E}_0 : \sigma_1 \to \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}^{w}_{m}\ h'\ \mathsf{E}'$	given
$\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$	CHANDLE
$\Gamma; \langle\!\langle l : (m, h') \mid w \rangle\!\rangle \vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \to \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow \mathsf{E}'$	above
$\lceil\Gamma\rceil \Vdash_{\mathsf{ops}} h' : \lceil\sigma\rceil \mid l \mid \epsilon$	Part 3
$\lceil\Gamma\rceil, \lceil\langle\!\langle l : (m, h') \mid w \rangle\!\rangle\rceil^{\langle l \mid \epsilon\rangle}; \langle\!\langle (m, h')^{l} \mid w \rangle\!\rangle \Vdash_{\mathsf{ec}} \mathsf{E}' : \lceil\sigma_1\rceil \to \lceil\sigma\rceil \mid \langle l \mid \epsilon \rangle$	I.H.
$\lceil\Gamma\rceil, \lceil\langle\!\langle l : (m, h') \mid w \rangle\!\rangle\rceil^{\langle l \mid \epsilon\rangle}; w \Vdash_{\mathsf{ec}} \mathsf{handle}^{w}_{m}\ h'\ \mathsf{E}' : \lceil\sigma_1\rceil \to \lceil\sigma\rceil \mid \epsilon$	MON-CHANDLE
$\lceil\langle\!\langle (m, h')^{l} \mid w \rangle\!\rangle\rceil^{\langle l \mid \epsilon\rangle} = \lceil w\rceil^{\epsilon}$	by definition
$\lceil\Gamma\rceil, \lceil w\rceil^{\epsilon}; w \Vdash_{\mathsf{ec}} \mathsf{handle}^{w}_{m}\ h'\ \mathsf{E}' : \lceil\sigma_1\rceil \to \lceil\sigma\rceil \mid \epsilon$	follows

□

B.3 System F^{ev}

B.3.1 Evaluation Context Typing.

Lemma 22. (*Evaluation context typing*)
If $\Gamma;\ w \Vdash_{ec}\ \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ and $\Gamma;\ \lceil \mathsf{E} \rceil, w\ \Vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle$, then $\Gamma;\ w\ \Vdash\ \mathsf{E}[e] : \sigma_2 \mid \epsilon$.

Proof. (*of Lemma 22*) By induction on the evaluation context typing.
case $\mathsf{E} = \square$. The goal follows trivially.
case $\mathsf{E} = \mathsf{E}_0\ w\ e_0$.

$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0\ w\ e_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ given
$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow (\sigma_3 \Rightarrow \epsilon\ \sigma_2) \mid \epsilon$ MON-CAPP1
$\Gamma; w\ \Vdash\ e_0 : \sigma_3 \mid \epsilon$ above
$\Gamma; w\ \Vdash\ \mathsf{E}_0[e] : \sigma_3 \Rightarrow \epsilon\ \sigma_2 \mid \epsilon$ I.H.
$\Gamma; w\ \Vdash\ \mathsf{E}_0[e]\ w\ e_0 : \sigma_2 \mid \epsilon$ MAPP

case $\mathsf{E} = v\ w\ \mathsf{E}_0$.

$\Gamma; w \Vdash_{ec}\ v\ w\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ given
$\Gamma; w\ \Vdash\ v : \sigma_3 \Rightarrow \epsilon\ \sigma_2 \mid \epsilon$ MON-CAPP2
$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_3 \mid \epsilon$ above
$\Gamma; w\ \Vdash\ \mathsf{E}_0[e] : \sigma_3 \mid \epsilon$ I.H.
$\Gamma; w\ \Vdash\ v\ w\ \mathsf{E}_0[e] : \sigma_2 \mid \epsilon$ MAPP

case $\mathsf{E} = \mathsf{E}_0\ [\sigma]$.

$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0\ [\sigma] : \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ given
$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow \forall \alpha.\ \sigma_3 \mid \epsilon$ MON-CTAPP
$\sigma_3[\alpha{:=}\sigma] = \sigma_1 \rightarrow \sigma_2$ above
$\Gamma; w\ \Vdash\ \mathsf{E}_0[e] : \forall \alpha.\ \sigma_3 \mid \epsilon$ I.H.
$\Gamma; w\ \Vdash\ \mathsf{E}_0[e]\ [\sigma] : \sigma_3[\alpha{:=}\sigma] \mid \epsilon$ MTAPP

case $\mathsf{E} = \mathsf{handle}_w\ h\ \mathsf{E}_0$.

$\Gamma; w \Vdash_{ec}\ \mathsf{handle}_w\ h\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ given
$\Gamma;\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \langle l \mid \epsilon \rangle$ MON-CHANDLE
$\Gamma \Vdash_{ops}\ h : \sigma \mid l \mid \epsilon$ above
$\Gamma;\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash_{ec}\ \mathsf{E}_0[e] : \sigma_2 \mid \langle l \mid \epsilon \rangle$ I.H.
$\Gamma; w\ \Vdash\ \mathsf{handle}_w\ h\ \mathsf{E}_0[e] : \sigma_2 \mid \epsilon$ MHANDLE

□

Proof. (*Of Lemma 6*) Induction on E.
case $\mathsf{E} = \square$. Let $\sigma_1 = \sigma$ and the goal holds trivially.
case $\mathsf{E} = \mathsf{E}_0\ w\ e_0$.

$\varnothing; w\ \Vdash\ \mathsf{E}_0[e]\ w\ e_0 : \sigma \mid \epsilon$ given
$\varnothing; w\ \Vdash\ \mathsf{E}_0[e] : \sigma_2 \Rightarrow \epsilon\ \sigma \mid \epsilon$ MAPP
$\varnothing;\ \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid w \rangle\!\rangle\ \Vdash\ e : \sigma_1 \mid \langle \lceil \mathsf{E}_0 \rceil^l \mid \epsilon \rangle$ I.H.
$\lceil \mathsf{E} \rceil = \lceil \mathsf{E}_0\ w\ e_0 \rceil = \lceil \mathsf{E}_0 \rceil$ by definition
$\lceil \mathsf{E} \rceil^l = \lceil \mathsf{E}_0\ w\ e_0 \rceil^l = \lceil \mathsf{E}_0 \rceil^l$ by definition

case $\mathsf{E} = v\ w\ \mathsf{E}_0$.

$\varnothing; w \Vdash v\ w\ \mathsf{E}_0[e] : \sigma \mid \epsilon$	given
$\varnothing; w \Vdash \mathsf{E}_0[e] : \sigma_2 \mid \epsilon$	MAPP
$\varnothing; \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid w \rangle\!\rangle \Vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0 \rceil^l \mid \epsilon \rangle$	I.H.
$\lceil \mathsf{E} \rceil = \lceil v\ w\ \mathsf{E}_0 \rceil = \lceil \mathsf{E}_0 \rceil$	by definition
$\lceil \mathsf{E} \rceil^l = \lceil v\ w\ \mathsf{E}_0 \rceil^l = \lceil \mathsf{E}_0 \rceil^l$	by definition

case $\mathsf{E} = \mathsf{E}_0\ [\sigma]$.

$\varnothing; w \Vdash \mathsf{E}_0[e]\ [\sigma] : \sigma \mid \epsilon$	given
$\varnothing; w \Vdash \mathsf{E}_0[e] : \forall \alpha.\ \sigma_2 \mid \epsilon$	MTAPP
$\varnothing; \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid w \rangle\!\rangle \Vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0 \rceil^l \mid \epsilon \rangle$	I.H.
$\lceil \mathsf{E} \rceil = \lceil \mathsf{E}_0\ [\sigma] \rceil = \lceil \mathsf{E}_0 \rceil$	by definition
$\lceil \mathsf{E} \rceil^l = \lceil \mathsf{E}_0\ [\sigma] \rceil^l = \lceil \mathsf{E}_0 \rceil^l$	by definition

case $\mathsf{E} = \mathsf{handle}_m\ h\ \mathsf{E}_0$.

$\varnothing; w \Vdash \mathsf{handle}_m\ h\ \mathsf{E}_0[e] : \sigma \mid \epsilon$	given
$\varnothing; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash \mathsf{E}_0[e] : \sigma \mid \langle l \mid \epsilon \rangle$	MHANDLE
$\varnothing \Vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon$	above
$\varnothing; \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid l : (m, h) \mid w \rangle\!\rangle \Vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0 \rceil^l \mid l \mid \epsilon \rangle$	I.H.
$\langle\!\langle \lceil \mathsf{E}_0 \rceil \mid l : (m, h) \rangle\!\rangle = \langle\!\langle \lceil \mathsf{handle}_m\ h \cdot \mathsf{E}_0 \rceil \rangle\!\rangle$	by definition
$\langle \lceil \mathsf{E} \rceil^l \rangle = \lceil \mathsf{handle}_m\ h\ \mathsf{E}_0 \rceil^l = \langle \lceil \mathsf{E}_0 \rceil \mid l \rangle$	by definition

□

Proof. (*Of Lemma 7*)

$\varnothing; \langle\!\langle\rangle\!\rangle \vdash \mathsf{E}[\mathsf{perform}\ op\ \overline{\sigma}\ v] : \sigma \mid \langle\rangle$	given
$\varnothing; \lceil \mathsf{E} \rceil \vdash \mathsf{perform}\ op\ \overline{\sigma}\ v : \sigma_1 \mid \lceil \mathsf{E} \rceil^l$	Lemma 6
$\varnothing; \lceil \mathsf{E} \rceil \vdash \mathsf{perform}\ op\ \overline{\sigma} : \sigma_2 \rightarrow \lceil \mathsf{E} \rceil^l\ \sigma_1 \mid \lceil \mathsf{E} \rceil^l$	APP
$\varnothing \vdash_{\mathsf{val}} \mathsf{perform}\ op\ \overline{\sigma} : \sigma_2 \rightarrow \lceil \mathsf{E} \rceil^l\ \sigma_1$	VAL
$l \in \lceil \mathsf{E} \rceil^l$	OP
$\mathsf{E} = \mathsf{E}_1 \cdot \mathsf{handle}_m^w\ h \cdot \mathsf{E}_2$	By definition of $\lceil \mathsf{E} \rceil^l$
$op \rightarrow f \in h$	above
$op \notin \mathsf{bop}(\mathsf{E}_2)$	Let $\mathsf{handle}^\epsilon\ h$ be the innermost one

□

B.3.2 Correspondence.

Proof. (*Of Theorem 5*)

$\varnothing; \langle\!\langle\rangle\!\rangle \Vdash_{\mathsf{ev}} \mathsf{E}[\mathit{perform}\ op\ \overline{\sigma}\ w\ v] : \sigma \mid \langle\rangle$	given
$\varnothing; \lceil \mathsf{E} \rceil \Vdash_{\mathsf{ev}} \mathsf{perform}\ op\ \overline{\sigma}\ w\ v : \sigma_1 \mid \lceil \mathsf{E} \rceil^l$	Lemma 6
$w = \lceil \mathsf{E} \rceil$	MAPP
$\mathsf{E} = \mathsf{E}_1 \cdot \mathsf{handle}_m^w\ h \cdot \mathsf{E}_2$	Lemma 7
$op \notin \mathsf{bop}(\mathsf{E}_2),\ (op \rightarrow f) \in h$	above
$l \notin \lceil \mathsf{E}_2 \rceil^l$	or otherwise $op \in \mathsf{bop}(\mathsf{E}_2)$
$\lceil \mathsf{E}_1.\ \mathsf{handle}_m^w\ h \cdot \mathsf{E}_2 \rceil = \langle\!\langle \lceil \mathsf{E}_2 \rceil \mid \langle\!\langle l : (m, h) \mid \lceil \mathsf{E}_1 \rceil \rangle\!\rangle \rangle\!\rangle$	by definition
$\langle\!\langle \lceil \mathsf{E}_2 \rceil \mid \langle\!\langle l : (m, h) \mid \lceil \mathsf{E}_1 \rceil \rangle\!\rangle \rangle\!\rangle.l = \langle\!\langle l : (m, h) \mid \lceil \mathsf{E}_1 \rceil \rangle\!\rangle.l$	Follows
$w.l = \lceil \mathsf{E} \rceil.l = \langle\!\langle l : (m, h) \mid \lceil \mathsf{E}_1 \rceil \rangle\!\rangle.l = (m, h)$	

□

B.3.3 Substitution.

Lemma 23. (*Substitution*)
1. If $\Gamma_1,\ x:\ \sigma,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v_1 : \sigma_1$, and Γ_1 , $\Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma$, then $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v_1\,[x{:=}\ v] : \sigma_1$.
2. If $\Gamma_1,\ x:\ \sigma,\ \Gamma_2\ ;\ w \Vdash\ e_1 : \sigma_1 \mid \epsilon$ and Γ_1 , $\Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma$,
then $\Gamma_1,\ \Gamma_2\ ;\ w\,[x := v] \Vdash\ e_1\,[x{:=}\ v] : \sigma_1 \mid \epsilon$.
3. If $\Gamma_1, x:\sigma, \Gamma_2 \Vdash_{\mathsf{ops}}\ \{\ op_1 \to f_1,\ \ldots,\ op_n \to f_n\ \} : \sigma_1 \mid l \mid \epsilon$ and $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma$,
then $\Gamma_1, \Gamma_2 \Vdash_{\mathsf{ops}}\ (\{\ op_1 \to f_1,\ \ldots,\ op_n \to f_n\ \})[x{:=}v] : \sigma_1 \mid l \mid \epsilon$.
4. If $\Gamma_1, x:\sigma, \Gamma_2\ ;\ w \Vdash_{\mathsf{ec}}\ \mathsf{E} : \sigma_1 \to \sigma_2 \mid \epsilon$ and Γ_1 , $\Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma$,
then $\Gamma_1, \Gamma_2\ ;\ w[x{:=}v] \Vdash_{\mathsf{ec}}\ \mathsf{E}\,[x{:=}v] : \sigma_1 \to \sigma_2 \mid \epsilon$.

Proof. (*Of Lemma 23*) Apply Lemma 34, ignoring all translations. □

Lemma 24. (*Type Variable Substitution*)
1. If $\Gamma \Vdash_{\mathsf{val}}\ v : \sigma$ and $\vdash_{\mathsf{wf}}\ \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1] \Vdash_{\mathsf{val}}\ v[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1]$.
2. If $\Gamma; w \Vdash\ e : \sigma \mid \epsilon$ and $\Vdash_{\mathsf{wf}}\ \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\sigma_1] \Vdash\ e[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid \epsilon$.
3. If $\Gamma \Vdash_{\mathsf{ops}}\ h : \sigma \mid l \mid \epsilon$ and $\vdash_{\mathsf{wf}}\ \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1] \Vdash_{\mathsf{ops}}\ h[\alpha^k{:=}\sigma_1] : \sigma[\alpha^k{:=}\sigma_1] \mid l \mid \epsilon$.
4. If $\Gamma; w \Vdash\ \mathsf{E} : \sigma_1 \to \sigma_2 \mid \epsilon$ and $\vdash_{\mathsf{wf}}\ \sigma_1 : k$,
then $\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\sigma_1] \Vdash\ \mathsf{E}[\alpha^k{:=}\sigma_1] : \sigma_1[\alpha^k{:=}\sigma_1] \to \sigma_2[\alpha^k{:=}\sigma_1]$.

Proof. (*Of Lemma 24*) Apply 35, ignoring all translations. □.

Lemma 25. (*Values can have any effect*)
1. If $\Gamma\ ;\ w_1 \Vdash\ v : \sigma \mid \epsilon_1$, then $\Gamma\ ;\ w_2 \Vdash\ v : \sigma \mid \epsilon_2$.
2. If $\Gamma\ ;\ w_1 \vdash\ v : \sigma \mid \epsilon_1 \rightsquigarrow v'$, then $\Gamma\ ;\ w_2 \vdash\ v : \sigma \mid \epsilon_2 \rightsquigarrow v'$.

Proof. (*Of Lemma 25*)
Part 1 By MVAL, we have $\Gamma \Vdash_{\mathsf{val}}\ v : \sigma$. By MVAL, we have $\Gamma\ ;\ w_2 \Vdash\ v : \sigma \mid \epsilon_2$.
Part 2 By VAL, we have $\Gamma \vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$. By VAL, we have $\Gamma\ ;\ w_2 \vdash\ v : \sigma \mid \epsilon_2 \rightsquigarrow v'$. □

B.3.4 Preservation.

Proof. (*Of Theorem 7*)

Let $e_1 = \mathsf{E}[e_1']$, and $e_2 = \mathsf{E}[e_2']$.

$\varnothing; \langle\!\langle\rangle\!\rangle \Vdash\ \mathsf{E}[e_1'] : \sigma \mid \langle\rangle$	given
$\varnothing;\ \lceil\mathsf{E}\rceil \Vdash\ e_1' : \sigma_1 \mid \lceil\mathsf{E}\rceil^l$	Lemma 6
$\varnothing; \langle\!\langle\rangle\!\rangle \Vdash\ \mathsf{E} : \sigma_1 \to \sigma \mid \langle\rangle$	above
$e_1' \longrightarrow e_2'$	given
$\varnothing;\ \lceil\mathsf{E}\rceil \Vdash\ e_2' : \sigma_1 \mid \lceil\mathsf{E}\rceil^l$	Lemma 26
$\varnothing; \langle\!\langle\rangle\!\rangle \Vdash\ \mathsf{E}[e_2'] : \sigma \mid \langle\rangle$	Lemma 22

□

Lemma 26. (*Small step preservation of evidence typing*)
If $\varnothing;\ w \Vdash\ e_1 : \sigma \mid \epsilon$ and $e_1 \longrightarrow e_2$, then $\varnothing;\ w \Vdash\ e_2 : \sigma \mid \epsilon$.

Proof. (*Of Lemma 26*) By induction on reduction.
case $(\lambda^\epsilon z : \mathsf{evv}\ \epsilon,\ x:\sigma_1.\ e)\ w\ v \longrightarrow e[z{:=}w, x{:=}v]$.

$\varnothing; w \Vdash (\lambda^{\epsilon} z : \text{evv}\ \epsilon,\ x : \sigma_1.\ e)\ w\ v : \sigma_2 \mid \epsilon$	given
$\varnothing; w \Vdash \lambda^{\epsilon} z : \text{evv}\ \epsilon, x : \sigma_1.\ e : \sigma_1 \Rightarrow\ \epsilon\ \sigma_2 \mid \epsilon$	MAPP
$\varnothing; w \Vdash\ v : \sigma_1 \mid \epsilon$	above
$z : \text{evv}\ \epsilon, x : \sigma_1;\ z \Vdash\ e : \sigma_2 \mid \epsilon$	MABS
$x : \sigma_1; w \Vdash\ e[z{:=}w] : \sigma_2 \mid \epsilon$	Lemma 23
$\varnothing; w \Vdash\ e[z{:=}w, x{:=}v] : \sigma_2 \mid \epsilon$	Lemma 23

case $(\Lambda\alpha.\ v)\ [\sigma] \longrightarrow v[\alpha{:=}\sigma]$.

$\varnothing; w \Vdash (\Lambda\alpha.\ v)\ [\sigma] : \sigma_1\ [\alpha{:=}\sigma] \mid \epsilon$	given
$\varnothing; w \Vdash\ \Lambda\alpha.\ v : \forall\alpha.\ \sigma_1 \mid \epsilon$	MTAPP
$\varnothing \Vdash_{\text{val}}\ \Lambda\alpha.\ v : \forall\alpha \cdot \sigma_1$	MVAL
$\varnothing \Vdash_{\text{val}}\ v : \sigma_1$	MTABS
$\varnothing; w \Vdash_{\text{val}}\ v : \sigma_1 \mid \epsilon$	MVAL
$\varnothing; w \Vdash_{\text{val}}\ v[\alpha{:=}\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon$	Lemma 24

case $(\text{handler}^{\epsilon}\ h)\ w\ v \longrightarrow \text{handle}^{w}_{m}\ h\ (v\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle\ ())$ with a unique m.

$\varnothing; w \Vdash (\text{handler}^{\epsilon}\ h)\ w\ v : \sigma \mid \epsilon$	given
$\varnothing; w \Vdash\ \text{handler}^{\epsilon}\ h : (() \Rightarrow\ \langle l \mid \epsilon \rangle\ \sigma) \Rightarrow\ \epsilon\ \sigma \mid \epsilon$	MAPP
$\varnothing; w \Vdash\ v : () \Rightarrow\ \langle l \mid \epsilon \rangle\ \sigma \mid \epsilon$	above
$\varnothing \Vdash_{\text{val}}\ \text{handler}^{\epsilon}\ h : (() \Rightarrow\ \langle l \mid \epsilon \rangle\ \sigma) \Rightarrow\ \epsilon\ \sigma$	MVAL
$\varnothing \Vdash_{\text{ops}}\ h : \sigma \mid l \mid \epsilon$	MHANDLER
$\varnothing; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash\ v : () \Rightarrow\ \langle l \mid \epsilon \rangle\ \sigma \mid \langle l \mid \epsilon \rangle$	Lemma 25
$\varnothing; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash\ v\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle\ () : \sigma \mid \langle l \mid \epsilon \rangle$	MAPP
$\varnothing; w \Vdash\ \mathit{handle}^{w}_{m}\ h\ (v\ \langle\!\langle l : (m, h) \mid w \rangle\!\rangle\ ()) : \sigma \mid \langle \epsilon \rangle$	MHANDLE

case $\text{handle}^{w}_{m}\ h \cdot v \longrightarrow v$.

$\varnothing; w \Vdash\ \mathit{handle}^{w}_{m}\ h \cdot v : \sigma \mid \epsilon$	given
$\varnothing; \langle\!\langle l : (m,\ h) \mid w \rangle\!\rangle \Vdash\ v : \sigma \mid \langle l \mid \epsilon \rangle$	MHANDLE
$\varnothing; w \Vdash\ v : \sigma \mid \langle \epsilon \rangle$	Lemma 25

case $\text{handle}^{w}_{m}\ h \cdot \mathsf{E} \cdot \text{perform}\ op\ \overline{\sigma}\ w'\ v \longrightarrow f\ \overline{\sigma}\ w\ v\ w\ k$.

$op \notin \text{bop}(\mathsf{E})$ and $op \rightarrow f\ \in h$	given
$op : \forall\overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2\ \in \Sigma(l)$	given
$k\ =\ \text{guard}^{w}\ (\text{handle}^{w}_{m}\ h \cdot \mathsf{E})\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]$	given
$\varnothing;\ w \Vdash\ \text{handle}^{w}_{m}\ h \cdot \mathsf{E} \cdot \text{perform}\ op\ \overline{\sigma}\ w'\ v : \sigma \mid \epsilon$	given
$\varnothing \Vdash_{\text{ops}}\ h : \sigma \mid l \mid \epsilon$	MHANDLE
$\varnothing \Vdash_{\text{val}}\ f : \forall\overline{\alpha}.\ \sigma_1 \Rightarrow\ \epsilon\ (\sigma_2 \Rightarrow\ \epsilon\ \sigma) \Rightarrow\ \epsilon\ \sigma$	MOPS
$\varnothing;\ w \Vdash\ f : \forall\overline{\alpha}.\ \sigma_1 \Rightarrow\ \epsilon\ (\sigma_2 \Rightarrow\ \epsilon\ \sigma) \Rightarrow\ \epsilon\ \sigma \mid \epsilon$	MVAL
$\varnothing;\ w \Vdash\ f\ \overline{\sigma} : \sigma_1[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow\ \epsilon\ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow\ \epsilon\ \sigma) \Rightarrow\ \epsilon\ \sigma \mid \epsilon$	MTAPP
$\varnothing;\ \langle\!\langle \lceil \mathsf{E} \rceil\ \mid l : (m, h)\ \mid w \rangle\!\rangle \Vdash\ \text{perform}\ op\ \overline{\sigma}\ w'\ v : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \mid \langle \lceil \mathsf{E} \rceil^{l} \mid l \mid \epsilon \rangle$	Lemma 6
$\varnothing;\ w \Vdash_{\text{ec}}\ \mathit{handle}^{w}_{m} \cdot \mathsf{E} : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma \mid \epsilon$	above
$\varnothing; \langle\!\langle \lceil \mathsf{E} \rceil\ \mid l : (m, h)\ \mid w \rangle\!\rangle \Vdash\ v : \sigma_1[\overline{\alpha} := \overline{\sigma}] \mid \langle \lceil \mathsf{E} \rceil^{l} \mid l \mid \epsilon \rangle$	MAPP and MTAPP
$\varnothing;\ w \Vdash\ v : \sigma_1[\overline{\alpha} := \overline{\sigma}] \mid \epsilon$	Lemma 25
$\varnothing;\ w \Vdash\ f\ \overline{\sigma}\ w\ v : (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \epsilon\ \sigma) \Rightarrow\ \epsilon\ \sigma \mid \epsilon$	MAPP
$\varnothing \Vdash_{\text{val}}\ \text{guard}^{w}\ (\text{handle}^{w}_{m}\ h \cdot \mathsf{E})\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow\ \epsilon\ \sigma$	MGUARD
$\varnothing; w \Vdash\ \text{guard}^{w}\ (\text{handle}^{w}_{m}\ h \cdot \mathsf{E})\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow\ \epsilon\ \sigma \mid \epsilon$	MVAL
$\varnothing;\ w \Vdash\ f\ \overline{\sigma}\ w\ v\ w\ k : \sigma \mid \epsilon$	MAPP

case $(\text{guard}^{w_1}\ \mathsf{E}\ \sigma_1)\ w\ v \longrightarrow \mathsf{E}[v]$.

$\varnothing; w \Vdash \mathsf{guard}^w\ \mathsf{E}\ \sigma_1\ w\ v : \sigma \mid \epsilon$	given
$\varnothing; w \Vdash \mathsf{guard}^w\ \mathsf{E}\ \sigma_1 : \sigma_1 \Rightarrow \epsilon\ \sigma \mid \epsilon$	MAPP
$\varnothing; w \Vdash v : \sigma_1 \mid \epsilon$	above
$\varnothing \Vdash_{\mathsf{val}} \mathsf{guard}^w\ \mathsf{E}\ \sigma_1 : \sigma_1 \Rightarrow \epsilon\ \sigma$	MVAL
$\varnothing; w \Vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \epsilon$	MGUARD
$\varnothing; \langle\!\langle \lceil \mathsf{E} \rceil \mid w \rangle\!\rangle \Vdash v : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle$	Lemma 25
$\varnothing; w \Vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma \mid \epsilon$	MGUARD
$\varnothing; w \Vdash \mathsf{E}[v] : \sigma \mid \epsilon$	Lemma 22

□

B.3.5 *Translation Coherence.* We define the equivalence relation inductively as follows.

$$\frac{e[z{:=}w] \ \cong\ \mathsf{E}[x]}{\lambda^\epsilon z, x{:}\sigma.\ e \ \cong\ \mathsf{guard}^w\ \mathsf{E}\ \sigma}\ [\textsc{eq-guard}]$$

$$\frac{\mathsf{E}[x] \ \cong\ e[z{:=}w]}{\mathsf{guard}^w\ \mathsf{E}\ \sigma \ \cong\ \lambda^\epsilon z, x{:}\sigma.\ e}\ [\textsc{eq-guard-symm}]$$

$$\frac{}{m_1 \ \cong\ m_2}\ [\textsc{eq-marker}]$$

$$\frac{}{x \ \cong\ x}\ [\textsc{eq-var}]$$

$$\frac{e_1 \ \cong\ e_2}{\lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ x{:}\ \sigma \cdot e_1 \ \cong\ \lambda^\epsilon\ z{:}\ \mathsf{evv}\ \epsilon,\ x{:}\ \sigma.\ e_2}\ [\textsc{eq-abs}]$$

$$\frac{w_1 \ \cong\ w_2 \qquad \mathsf{E}_1 \ \cong\ \mathsf{E}_2}{\mathsf{guard}^{w_1}\ \mathsf{E}_1\ \sigma \ \cong\ \mathsf{guard}^{w_2}\ \mathsf{E}_2\ \sigma}\ [\textsc{eq-guard}]$$

$$\frac{e_1 \ \cong\ e_3 \qquad w_1 \ \cong\ w_2 \qquad e_2 \ \cong\ e_4}{e_1\ w_1\ e_3 \ \cong\ e_2\ w_2\ e_4}\ [\textsc{eq-app}]$$

$$\frac{v_1 \ \cong\ v_2}{\Lambda\alpha.\ v_1 \ \cong\ \Lambda\alpha.\ v_2}\ [\textsc{eq-tabs}]$$

$$\frac{e_1 \ \cong\ e_2}{e_1[\sigma] \ \cong\ e_2[\sigma]}\ [\textsc{eq-tapp}]$$

$$\frac{}{\mathsf{perform}\ op \ \cong\ \mathsf{perform}\ op}\ [\textsc{eq-perform}]$$

$$\frac{h_1 \ \cong\ h_2}{\mathsf{handler}^\epsilon\ h_1 \ \cong\ \mathsf{handler}^\epsilon\ h_2}\ [\textsc{eq-handler}]$$

$$\frac{m_1 \;\cong\; m_2 \qquad w_1 \;\cong\; w_2 \qquad e_1 \;\cong\; e_2 \qquad h_1 \;\cong\; h_2}{\mathsf{handle}^{w_1}_{m_1}\; h_1\; e_1 \;\cong\; \mathsf{handle}^{w_2}_{m_2}\; h_2\; e_2}\;[\textsc{eq-handle}]$$

Lemma 27. (*Translation is deterministic*)
1. If $\Gamma; w \vdash e : \sigma \mid \epsilon \rightsquigarrow e_1$, and $\Gamma; w \vdash e : \sigma \mid \epsilon \rightsquigarrow e_2$, then e_1 and e_2 are equivalent up to EQ-MARKER. By definition, we also have $e_1 \cong e_2$.
2. If $\Gamma \vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v_1$, and $\Gamma \vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v_2$, then v_1 and v_2 are equivalent up to EQ-MARKER. By definition, we also have $v_1 \cong v_2$.
3. If $\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid \epsilon \rightsquigarrow h_1$, and $\Gamma \vdash_{\mathsf{ops}} h : \sigma \mid \epsilon \rightsquigarrow h_2$, then h_1 and h_2 are equivalent up to EQ-MARKER. By definition, we also have $h_1 \cong h_2$.
4. If $\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}_1$, and $\Gamma; w \vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow \mathsf{E}_2$, then E_1 and E_2 are equivalent up to EQ-MARKER. By definition, we also have $\mathsf{E}_1 \cong \mathsf{E}_2$.

Proof. (*Of Lemma 27*) By a straightforward induction on the translation. Note the only difference is introduced in HANDLE and CHANDLE, where we may have chosen different m's. □

Lemma 28. (*Evaluation context equivalence*)
If $\mathsf{E}_1 \cong \mathsf{E}_2$, and $e_1 \cong e_2$, then $\mathsf{E}_1[e_1] \cong \mathsf{E}_2[e_2]$.

Proof. (*Of Lemma 28*) By a straightforward induction on the context equivalence. □

Lemma 29. (*Equivalence substitution*)
1. If $e_1 \cong e_2$, and $v_1 \cong v_2$, then $e_1[x{:=}v_1] \cong e_2[x{:=}v_2]$.
2. If $e_1 \cong e_2$, then $e_1[\alpha{:=}\sigma] \cong e_2[\alpha{:=}\sigma]$.

Proof. (*Of Lemma 29*) By a straightforward induction on the equivalence relation. □

Proof. (*Of Lemma 8*) By induction on the reduction.

case $e_1 \;=\; (\lambda^{\epsilon}\; z{:}\mathsf{evv}\;\epsilon,\; x : \sigma.\; e_1')\; w_1\; v_1$ and $e_1 \longrightarrow e_1'[z{:=}w_1,\; x{:=}v_1]$.

By case analysis on the equivalence relation.

subcase $e_2 \;=\; (\lambda^{\epsilon}\; z{:}\mathsf{evv}\;\epsilon,\; x : \sigma.\; e_2')\; w_2\; v_2$ with $e_1' \cong e_2'$, $w_1 \cong w_2$ and $v_1 \cong v_2$.

$(\lambda^{\epsilon}\; z{:}\mathsf{evv}\;\epsilon,\; x : \sigma.\; e_2')\; w_2\; v_2 \longrightarrow e_2'[z{:=}w_2, x{:=}v_2]$	(*app*)
$e_1'[z{:=}w_1,\; x{:=}v_1] \cong e_2'[z{:=}w_2, x{:=}v_2]$	Lemma 29

subcase $e_2 \;=\; (\mathsf{guard}^{w}\; \mathsf{E}\; \sigma)\; w_2\; v_2$ with $e_1'[z{:=}w] \cong \mathsf{E}[x]$, $w_1 \cong w_2$ and $v_1 \cong v_2$. We discuss whether w_2 is equivalent to w.

- $w_2 \;=\; w$.

$\mathsf{guard}^{w_2}\; \mathsf{E}\; \sigma\; w_2\; v_2 \longrightarrow \mathsf{E}[v_2]$	(*guard*)
$e_1'[z{:=}w_2] \cong \mathsf{E}[x]$	given
$e_1'[z{:=}w_1] \cong \mathsf{E}[x]$	$w_1 \cong w_2$
$(e_1'[z{:=}w_1])[x{:=}v_1] \cong (\mathsf{E}[x])[x{:=}v_2]$	Lemma 29

- $w_2 \neq w$. Then e_2 get stuck as no rule applies.

case $e_1 \;=\; (\Lambda\alpha.\; e_1')\; [\sigma]$ and $e_1 \longrightarrow e_1'[\alpha{:=}\sigma]$.

$e_2 \;=\; (\Lambda\alpha.\; e_2')\; [\sigma]$	by equivalence
$e_1' \cong e_2'$	above
$e_2 \longrightarrow e_2'[\alpha{:=}\sigma]$	(*tapp*)
$e_1'[\alpha{:=}\sigma] \cong e_2'[\alpha{:=}\sigma]$	Lemma 29

case $e_1 \;=\; (\mathsf{handler}^{\epsilon}\; h_1)\; w_1\; v_1$ and $e_1 \longrightarrow \mathsf{handle}^{w_1}_{m}\; h_1\; (v_1\; \langle\!\langle l{:}(m,h) \mid w_1 \rangle\!\rangle\; ())$.

$e_2 \;=\; (\mathsf{handler}^{\epsilon}\; h_2)\; w_2\; v_2$ by equivalence
$v_1 \;\cong\; v_2$ above
$w_1 \;\cong\; w_2$ above
$h_1 \;\cong\; h_2$ above
$e_2 \longrightarrow \mathsf{handle}_m^{w_2}\; hh\; (v_2\; \langle\!\langle l : (m, h) \mid w_2 \rangle\!\rangle\; ())$ (*handler*)
$\mathsf{handle}_m^{w_1}\; h_1\; (v_1\; \langle\!\langle l : (m, h) \mid w_1 \rangle\!\rangle\; ()) \;\cong\; \mathsf{handle}_m^{w_2}\; h_2\; (v_2\; \langle\!\langle l : (m, h) \mid w_2 \rangle\!\rangle\; ())$ congruence

case $e_1 \;=\; \mathsf{handle}_m^{w_1}\; h_1 \cdot v_1$ and $e_1 \longrightarrow v_1$.

$e_2 \;=\; \mathsf{handle}_m^{w_2}\; h_2 \cdot v_2$ by equivalence
$v_1 \;\cong\; v_2$ above
$w_1 \;\cong\; w_2$ above
$h_1 \;\cong\; h_2$ above
$e_2 \longrightarrow v_2$ (*return*)

case $e_1 \;=\; \mathsf{handle}_m^{w_1}\; h_1 \cdot \mathsf{E}_1 \cdot \mathsf{perform}^{\epsilon'}\; op\; [\overline{\sigma}]\; w'\; v_1$ and $e_1 \longrightarrow f_1\; [\overline{\sigma}]\; w\; v_1\; w\; k_1$,
where $k_1 \;=\; \mathsf{guard}^{w_1}\; (\mathsf{handle}_m^{w_1}\; h \cdot \mathsf{E}_1)\; \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]$.

$e_2 \;=\; \mathsf{handle}_m^{w_2}\; h \cdot \mathsf{E}_2 \cdot \mathsf{perform}^{\epsilon'}\; op\; [\overline{\sigma}]\; w'\; v_2$ by equivalence
$v_1 \;\cong\; v_2$ above
$w_1 \;\cong\; w_2$ above
$\mathsf{E}_1 \;\cong\; \mathsf{E}_2$ above
$h_1 \;\cong\; h_2$ above
$f_1 \;\cong\; f_2$ therefore
$e_2 \longrightarrow f_2\; [\overline{\sigma}]\; w_2\; v_2\; w_2\; k_2$ (*perform*)
$k_2 \;=\; \mathsf{guard}^{w_2}\; (\mathsf{handle}_m^{w_2}\; h \cdot \mathsf{E}_2)\; \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]$ above
$k_1 \;\cong\; k_2$ congruence
$f_1\; [\overline{\sigma}]\; w_1\; v_1\; w_1\; k_1 \;\cong\; f_2\; [\overline{\sigma}]\; w_2\; v_2\; w_2\; k_2$ congruence

case $e_1 \;=\; (\mathsf{guard}^{w_1}\; \mathsf{E}_1\; \sigma)\; w_1\; v_1$ and $e_1 \longrightarrow \mathsf{E}_1[v_1]$.
By case analysis on the equivalence relation.
subcase

$e_2 \;=\; (\mathsf{guard}^{w_2}\; \mathsf{E}_2\; \sigma)\; w_3\; v_2$ by equivalence
$\mathsf{E}_1 \;\cong\; \mathsf{E}_2$ above
$v_1 \;\cong\; v_2$ above
$w_1 \;\cong\; w_2$ above
$w_1 \;\cong\; w_2$ above

If $w_2 \;=\; w_3$, then e_2 gets stuck as no rule applies.
If $w_2 \;=\; w_3$, then

$e_2 \longrightarrow \mathsf{E}_2[v_2]$ (*guard*)
$\mathsf{E}_1[v_1] \;\cong\; \mathsf{E}_2[v_2]$ Lemma 28

subcase

$e_2 \;=\; (\lambda z,\; x.\; e_2)\; w_2\; v_2$ by equivalence
$w_1 \;\cong\; w_2$ above
$v_1 \;\cong\; v_2$ above
$\mathsf{E}_1[x] \cong e_2[z{:=}w_1]$ above
$\mathsf{E}_1[x] \cong e_2[z{:=}w_2]$ $w_1 \;\cong\; w_2$
$e_2 \longrightarrow e_2[z{:=}w_2, x{:=}v_2]$ (*app*)
$(\mathsf{E}_1[x])[x{:=}v_1] \;\cong\; (e_2[z{:=}w_2])[x{:=}v_2]$ Lemma 29

□

Lemma 30. (*Small step evidence translation is coherent*)
If $\varnothing\,;\; w \vdash e_1 : \sigma \mid \epsilon \rightsquigarrow e_1'$ and $e_1 \longrightarrow e_2$, and $\varnothing;\; w \vdash e_2 : \sigma \mid \epsilon \rightsquigarrow e_2'$, then exists a e_2'', such that $e_1' \longrightarrow e_2''$ and $e_2'' \cong e_2'$.

Proof. (*Of Lemma 30*) By case analysis on the induction.

case $(\lambda^\epsilon x\!:\!\sigma_1.\, e)\; v \longrightarrow e[x{:=}v]$.

$\varnothing;\; w \vdash (\lambda^\epsilon x\!:\!\sigma_1.\, e)\; v : \sigma \mid \epsilon \rightsquigarrow (\lambda^\epsilon\; z : \mathsf{evv}\; \epsilon,\; x : \lceil\sigma_1\rceil.\, e')\; w\; v'$	given
$(\lambda^\epsilon\; z : \mathsf{evv}\; \epsilon,\; x : \lceil\sigma_1\rceil.\, e')\; w\; v' \longrightarrow e'[z{:=}w][x : v]$	(*app*)
$\varnothing;\; w \vdash \lambda^\epsilon\; x\!:\, \sigma_1.\, e : \sigma_1 \rightarrow\epsilon\; \sigma \mid \epsilon \rightsquigarrow (\lambda^\epsilon\; z : \mathsf{evv}\; \epsilon,\; x : \lceil\sigma_1\rceil \cdot e')$	APP
$\varnothing;\; w \vdash v : \sigma_1 \mid \epsilon \rightsquigarrow v'$	above
$\varnothing \vdash_{\mathsf{val}} \lambda^\epsilon\; x\!:\, \sigma_1.\, e : \sigma_1 \rightarrow\epsilon\; \sigma \rightsquigarrow (\lambda^\epsilon\; z : \mathsf{evv}\; \epsilon,\; x : \lceil\sigma_1\rceil \cdot e')$	VAL
$x\!:\!\sigma_1;\; z \vdash e : \sigma \mid \epsilon \rightsquigarrow e'$	ABS
$x\!:\!\sigma_1;\; z[z{:=}w] \vdash e : \sigma \mid \epsilon \rightsquigarrow e'[z{:=}w]$	Lemma 19
$x\!:\!\sigma_1;\; w \vdash e : \sigma \mid \epsilon \rightsquigarrow e'[z{:=}w]$	by substitution
$\varnothing \vdash_{\mathsf{val}} v : \sigma_1 \rightsquigarrow v'$	VAL
$\varnothing;\; w \vdash e[x{:=}v] : \sigma \mid \epsilon \rightsquigarrow e'[z{:=}w][x{:=}v']$	Lemma 18

case $(\Lambda\alpha.\, v)\; [\sigma] \longrightarrow v[\alpha{:=}\sigma]$.

$\varnothing; w \vdash (\Lambda\alpha.\, v)\; [\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow (\Lambda\alpha.\, v')\; [\lceil\sigma\rceil]$	given
$\varnothing; w \vdash \Lambda\alpha.\, v : \forall\alpha.\, \sigma_1 \mid \epsilon \rightsquigarrow \Lambda\alpha.\, v'$	TAPP
$(\Lambda\alpha.\, v')\; [\lceil\sigma\rceil] \longrightarrow v'[\alpha{:=}\lceil\sigma\rceil]$	(*tapp*)
$\varnothing; w \vdash v : \sigma_1 \mid \epsilon \rightsquigarrow v'$	TABS
$\varnothing; w \vdash v[\alpha{:=}\sigma] \rightsquigarrow v'[\alpha{:=}\lceil\sigma\rceil]$	Lemma 20

case $(\mathsf{handler}^\epsilon\; h)\; v \longrightarrow \mathsf{handle}^\epsilon\; h\,(v\,())$.

$\varnothing;\; w \vdash (\mathsf{handler}^\epsilon\; h)\; v : \sigma \mid \epsilon \rightsquigarrow (\mathsf{handler}^\epsilon\; h')\; w\; v'$	given
$\varnothing;\; w \vdash v : \sigma \mid \epsilon \rightsquigarrow v'$	APP
$\varnothing;\; \langle\!\langle (l : m, h) \mid w \rangle\!\rangle \vdash v : \sigma \mid \langle l \mid \epsilon\rangle \rightsquigarrow v'$	Lemma 25
$\varnothing; w \vdash \mathsf{handle}^\epsilon\; h\,(v\,()) : \sigma \mid \epsilon \rightsquigarrow (\mathsf{handle}^w_m\; h'\; (v'\; \langle\!\langle l\!:\!(m, h) \mid w \rangle\!\rangle\; ()\;)\;)$	given
$(\mathsf{handler}^\epsilon\; h')\; w\; v \longrightarrow \mathsf{handle}^w_m\; h'\; (v'\; \langle\!\langle l\!:\!(m, h) \mid w \rangle\!\rangle\; ())$	(*handler*)

case $\mathsf{handle}^\epsilon\; h \cdot v \longrightarrow v$

$\varnothing; w \vdash \mathsf{handle}^\epsilon\; h \cdot v : \sigma \mid \epsilon \rightsquigarrow \mathsf{handle}^w_m\; w\; v'$	given
$\varnothing; \langle\!\langle l\!:\!(m, h) \mid w \rangle\!\rangle \vdash v : \sigma \mid \langle l \mid \epsilon\rangle\; v'$	HANDLE
$\varnothing; w \vdash v : \sigma \mid \epsilon \rightsquigarrow v'$	Lemma 25
$\mathsf{handle}^w_m\; w\; v\,' \longrightarrow v'$	(*return*)

case $\mathsf{handle}^\epsilon\; h \cdot \mathsf{E} \cdot \mathsf{perform}\; op\; \overline{\sigma}\; v \longrightarrow f\; \overline{\sigma}\; v\; k$.

$op : \forall\overline{\alpha}.\ \sigma_1 \rightarrow \sigma_2\ \in \Sigma(l)$	given
$k\ =\ \lambda^{\epsilon}\ x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot x$	given
$\varnothing;\ w\ \vdash\ \mathsf{handle}^{\epsilon}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ v : \sigma \mid \epsilon$	given
$\rightsquigarrow \mathsf{handle}^{w}_{m_1}\ h_1 \cdot \mathsf{E}_1 \cdot \mathsf{perform}\ op\ \lceil\overline{\sigma}\rceil\ w'\ v$	
$\varnothing;\ w \vdash_{ec}\ \mathsf{handle}^{\epsilon}\ h\cdot \mathsf{E} : \sigma_2 \rightarrow \sigma \mid \epsilon\ \rightsquigarrow \mathsf{handle}^w_{m_1}\ h_1 \cdot \mathsf{E}_1$	Lemma 17
$x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}];\ w \vdash_{ec}\ \mathsf{handle}^\epsilon\ h\cdot\mathsf{E} : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma \mid \epsilon\ \rightsquigarrow \mathsf{handle}^w_{m_1}\ h_1\cdot \mathsf{E}_1$	Weakening
$x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}];\ w \vdash x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\ \rightsquigarrow x \mid \epsilon$	VAR and VAL
$x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}];\ w \vdash_{ec}\ \mathsf{handle}^\epsilon\ h\cdot\mathsf{E}\cdot x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma \mid \epsilon$	Lemma 16
$\rightsquigarrow \mathsf{handle}^w_{m_1}\ h_1\cdot \mathsf{E}_1 \cdot x$	
$\varnothing\ ;\ w \vdash \lambda^\epsilon\ x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^\epsilon\ h\cdot\mathsf{E}\cdot x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma \mid \epsilon$	given
$\rightsquigarrow \lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \lceil\sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\rceil.\ \mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x$	
$k_1\ =\ \lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \lceil\sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\rceil.\ \mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x$	let
$\varnothing;\ w \vdash f\ \overline{\sigma}\ v\ k : \sigma \mid \epsilon\ \rightsquigarrow f'\ \lceil\overline{\sigma}\rceil\ w\ v'\ w\ k_1$	APP
$k_2\ =\ \mathsf{guard}^w\ (\mathsf{handle}^w_{m_1}\ h_1\cdot\mathsf{E}_1)\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]$	let
$\mathsf{handle}^w_{m_1}\ h_1\cdot\mathsf{E}_1\cdot\mathsf{perform}\ op\ \lceil\overline{\sigma}\rceil\ w'\ v \longrightarrow f'\ \lceil\overline{\sigma}\rceil\ w\ v'\ w\ k_2$	(*perform*)
$\varnothing \vdash_{val}\ \lambda^\epsilon\ x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}].\ \mathsf{handle}^\epsilon\ h\cdot\mathsf{E}\cdot x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightarrow \sigma$	VAL
$\rightsquigarrow \lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \lceil\sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\rceil.\ \mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x$	
$x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\ ;\ z \vdash \mathsf{handle}^\epsilon\ h\cdot\mathsf{E}\cdot x : \sigma \mid \epsilon\ \rightsquigarrow \mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x$	ABS
$x : \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\ ;\ z[z{:=}w] \vdash \mathsf{handle}^\epsilon\ h\cdot\mathsf{E}\cdot x : \sigma \mid \epsilon$	Lemma 19
$\rightsquigarrow (\mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x)\ [z{:=}w]$	
$(\mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x)[z{:=}w]\ \cong\ \mathsf{handle}^w_{m_1}\ h_1\cdot\mathsf{E}_1\cdot x$	Lemma 27
$\lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \lceil\sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\rceil.\ \mathsf{handle}^z_{m_2}\ h_2\cdot\mathsf{E}_2\cdot x$	EQ-GUARD
$\cong\ \mathsf{guard}^w\ (\mathsf{handle}^w_{m_1}\ h_1\cdot\mathsf{E}_1)\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}]$	
$k_1\ \cong\ k_2$	namely
$f'\ \lceil\overline{\sigma}\rceil\ w\ v'\ w\ k_1\ \cong\ f'\ \lceil\overline{\sigma}\rceil\ w\ v'\ w\ k_2$	congruence

□

Proof. (*Of Theorem 8*)

$e_1 \longmapsto e_2$	given
$e_1\ =\ \mathsf{E}_1[e_3]$	STEP
$e_2\ =\ \mathsf{E}_1[e_4]$	above
$e_3 \longrightarrow e_4$	above
$\varnothing; \langle\rangle\ \vdash\ \mathsf{E}_1[e_3] : \sigma \mid \langle\rangle\ \rightsquigarrow e_1'$	given
$e_1'\ =\ \mathsf{E}_1'[e_3']$	Lemma 17
$\varnothing; \langle\rangle\ \vdash\ \mathsf{E}_1 : \sigma_1 \rightarrow \sigma \mid \langle\rangle\ \rightsquigarrow \mathsf{E}_1'$	above
$\varnothing;\ \lceil\mathsf{E}_1'\rceil\ \vdash\ e_3 : \sigma_1 \mid \lceil\mathsf{E}_1'\rceil^l\ \rightsquigarrow e_3'$	above
$\varnothing; \langle\rangle\ \vdash\ \mathsf{E}_1[e_4] : \sigma \mid \langle\rangle\ \rightsquigarrow e_2'$	given
$e_2'\ =\ \mathsf{E}_1''[e_4']$	Lemma 17
$\varnothing; \langle\rangle\ \vdash\ \mathsf{E}_1 : \sigma_1 \rightarrow \sigma \mid \langle\rangle\ \rightsquigarrow \mathsf{E}_1''$	above
$\varnothing;\ \lceil\mathsf{E}_1'\rceil\ \vdash\ e_4 : \sigma_1 \mid \lceil\mathsf{E}_1'\rceil^l\ \rightsquigarrow e_4'$	above
$e_3\ \cong\ e_4$	Lemma 30
$\mathsf{E}_1'\ \cong\ \mathsf{E}_1''$	Lemma 27
$\mathsf{E}_1'[e_3']\ \cong\ \mathsf{E}_1''[e_4']$	Lemma 28

□

B.3.6 Uniqueness of handlers. Handle-safe expressions have the following induction principle: (1) (base case) If e contains no handle^w_m terms, then e has the property; (2) (induction step) If e_1 has the property, and $e_1 \longmapsto e_2$, then e_2 has the property.

Lemma 31. (*Handle-evidence in handle-safe* F^{ev} *expressions is closed*)
If a handle-safe expression contains $\mathsf{handle}^w_m\ h\ e$, then w has no free variables.

Proof. (*Of Lemma 31*) **Base case**: Since there is no $\mathsf{handle}^w_m\ h\ e$, the lemma holds trivially.

Induction step: We want to prove that if e_1 has the property, and $e_1 \longmapsto e_2$,then e_2 has the property. We do case analysis of the operational semantics.

case $\mathsf{E} \cdot (\lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \sigma.\ e)\ w\ v \longmapsto \mathsf{E} \cdot e[z{:=}w, x{:=}v]$.

We know that in e, we have w_1 in $\mathsf{handle}^{w_0}_m\ h\ e_0$ is closed, therefore $(\mathsf{handle}^{w_0}_m\ h\ e_0)[z{:=}w, x{:=}v]\ =\ \mathsf{handle}^{w_0}_m\ h[z{:=}w, x{:=}v]\ e_0[z{:=}w, x{:=}v]$ and w_0 is still closed. And other handle evidences in E are already closed.

case $\mathsf{E} \cdot (\Lambda \alpha^k.\ v)\ [\sigma] \longmapsto \mathsf{E} \cdot v[\alpha{:=}\sigma]$.

We know that in e, we have w_1 in $\mathsf{handle}^{w_0}_m\ h\ e_0$ is closed, therefore $(\mathsf{handle}^{w_0}_m\ h\ e_0)[\alpha{:=}\sigma]\ =\ \mathsf{handle}^{w_0}_m\ h[\alpha{:=}\sigma]\ e_0[\alpha{:=}\sigma]$ and w_0 is still closed. And other handle evidences in E are already closed.

case $\mathsf{E} \cdot (\mathsf{handler}^\epsilon\ h)\ w\ v \longmapsto \mathsf{E}.\ \mathsf{handle}^w_{m_1}\ h\,(v\ \langle\!\langle l : (m_1, h) \mid w \rangle\!\rangle\ ())$ with m_1 unique. We know that w is closed. And other handle evidences in E are already closed.

case $\mathsf{E} \cdot \mathsf{handle}^w_m\ h \cdot v \longmapsto \mathsf{E} \cdot v$.

We already know handle evidences in E and v are closed.

case $\mathsf{E}_1 \cdot \mathsf{handle}^w_m\ h \cdot \mathsf{E}_2 \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w'\ v \longmapsto \mathsf{E}_1 \cdot f[\overline{\sigma}]\ w\ v\ w\ k$, where $k = \mathsf{guard}^w\ (\mathsf{handle}^w_m\ h \cdot \mathsf{E}_2)\ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])$ and $(op \to f)\ \in h$.

We know that w is closed. And other handle evidences in E_1,E_2,f,v are already closed.

case $\mathsf{E}_1 \cdot (\mathsf{guard}^w\ \mathsf{E}\ \sigma)\ w\ v \longmapsto \mathsf{E}_1 \cdot \mathsf{E}[v]$.

We already know handle evidences in E, E_1 and v are closed. □

Definition 2. (*m-mapping*)
We say an expression e is m-mapping, if every m in e can uniquely determine its w and h. Namely, if e contains $\mathsf{handle}^{w_1}_m\ h_1\ e_1$ and $\mathsf{handle}^{w_2}_m\ h_2\ e_2$, then $w_1\ =\ w_2$ and $h_1\ =\ h_2$.

Lemma 32. (*Handle-free* F^{ev} *expression is m-mapping*)
Any handle-free F^{ev} expression e is m-mapping.

Proof.
Base case: Since there is no handle^w_m, there is no m. So e is m-mapping trivially.

Induction step: We want to prove that if e_1 is m-mapping, and $e_1 \longmapsto e_2$,then e_2 is m-mapping. By case analysis on $e_1 \longmapsto e_2$.

case $\mathsf{E} \cdot (\lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \sigma \cdot e)\ w\ v \longmapsto \mathsf{E} \cdot e[z{:=}w, x{:=}v]$.

Due to Lemma 31, we know all handle-evidences are closed. Therefore, the substitution does not change those handle-evidences, and for all original pair of $\mathsf{handle}^{w_1}_m\ h_1\ e_1$ and $\mathsf{handle}^{w_2}_m\ h_2\ e_2$ for each m, we know $w_1\ =\ w_2$ still holds true.

Note v may be duplicated in e, which can introduce new pairs. Consider $(\lambda x.\ (x,\ x))\ (\lambda z.\ \mathsf{handle}^z_m\ e) \longrightarrow ((\lambda z.\ \mathsf{handle}^z_m\ e),\ (\lambda z.\ \mathsf{handle}^z_m\ e))$. Here the argument is duplicated, and now we have a new pair $(\lambda z.\ \mathsf{handle}^z_m\ e)$ and $(\lambda z.\ \mathsf{handle}^z_m\ e)$, where m maps to two z's. Unfortunately, those z's are actually different, as under α-renaming, the expression is equivalent to $((\lambda z_1.\ \mathsf{handle}^{z_1}_m\ e),\ (\lambda z_2.\ \mathsf{handle}^{z_2}_m\ e))$. And we have $z_1\ \neq z_2$!

Luckily, this situation cannot happen for handle-safe expressions. As due to Lemma 31, handle_m^w has no free variables in w. Therefore, for one handle handle_m^w, even if it is duplicated, for the new pair handle_m^w and handle_m^w, we still have $w = w$.

case $\mathsf{E} \cdot (\Lambda \alpha^k.\ v)\ [\sigma] \longmapsto \mathsf{E} \cdot v[\alpha{:=}\sigma]$.

Due to Lemma 31, we know all handle-evidences are closed. Therefore, the substitution does not change those handle-evidences, and for all original pair of $\mathsf{handle}_m^{w_1}\ h_1\ e_1$ and $\mathsf{handle}_m^{w_2}\ h_2\ e_2$ for each m, we know $w_1\ =\ w_2$ still holds true.

case $\mathsf{E} \cdot (\mathsf{handler}^{\epsilon}\ h)\ w\ v \longmapsto \mathsf{E}.\ \mathsf{handle}_{m_1}^{w}\ h\ (v\ \langle\!\langle l{:}(m_1, h) \mid w \rangle\!\rangle\ ())$ with m_1 unique.

Every pair in E, h and v is a pair in $\mathsf{E} \cdot (\mathsf{handler}^{\epsilon}\ h)\ w\ v$. So it is still m-mapping. Given m_1 unique, we know there is no other $\mathsf{handle}_{m_1}^{w_2}\ h_2\ e_2$. So $\mathsf{E} \cdot \mathsf{handle}_{m_1}^{w}\ h\ (v\ \langle\!\langle l{:}(m_1, h) \mid w \rangle\!\rangle\ ())$ is m-mapping.

case $\mathsf{E} \cdot \mathsf{handle}_m^w\ h \cdot v \longmapsto \mathsf{E} \cdot v$.

Every pair in $\mathsf{E} \cdot v$ is a pair in $\mathsf{E} \cdot \mathsf{handle}_m^w\ h \cdot v$. So we know it is m-mapping.

case $\mathsf{E} \cdot \mathsf{handle}_m^w\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w'\ v \longmapsto \mathsf{E} \cdot f[\overline{\sigma}]\ w\ v\ w\ k$, where $k = \mathsf{guard}^w\ (\mathsf{handle}_m^w\ h \cdot \mathsf{E})\ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])$ and $(op \to f)\ \in h$.

Every pair in $\mathsf{E} \cdot f[\overline{\sigma}]\ w\ v\ w\ k$ is a pair in $\mathsf{E} \cdot \mathsf{hanndle}_m^w\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w'\ v$. So we know it is m-mapping.

case $\mathsf{E}_1 \cdot (\mathsf{guard}^w\ \mathsf{E}\ \sigma)\ w\ v \longmapsto \mathsf{E}_1 \cdot \mathsf{E}[v]$.

Every pair in $\mathsf{E}_1 \cdot \mathsf{E}[v]$ is a pair in $\mathsf{E}_1 \cdot (\mathsf{guard}^w\ \mathsf{E}\ \sigma)\ w\ v$. So we know it is m-mapping. □

Proof. (*Of Theorem 6*) We prove it by contradiction.

$m_1\ =\ m_2$	suppose
$w_1\ =\ w_2$	Lemma 32
$\Gamma\ ;\ w\ \Vdash\ \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2 \cdot \mathsf{handle}_{m_2}^{w_2}\ h \cdot e_0\ :\ \sigma \mid \epsilon$	given
$\Gamma\ ;\ w\ \Vdash\ \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2 \cdot \mathsf{handle}_{m_2}^{w_1}\ h \cdot e_0\ :\ \sigma \mid \epsilon$	$w_1\ =\ w_2$
$\Gamma\ ;\ \langle\!\langle\ \lceil \mathsf{E}_1 \rceil \mid w\ \rangle\!\rangle\ \Vdash\ \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2 \cdot \mathsf{handle}_{m_2}^{w_1}\ h \cdot e_0\ :\ \sigma_1 \mid \langle \lceil \mathsf{E}_1 \rceil^l \mid \epsilon \rangle$	Lemma 6
$w_1\ =\ \langle\!\langle\ \lceil \mathsf{E}_1 \rceil \mid w\ \rangle\!\rangle$	MHANDLE
$\Gamma\ ;\ \langle\!\langle\ \lceil \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2\ \rceil \mid w\ \rangle\!\rangle\ \Vdash\ \mathsf{handle}_{m_2}^{w_1}\ h \cdot e_0\ :\ \sigma_2 \mid \langle \lceil \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2\ \rceil^l \mid \epsilon \rangle$	Lemma 6
$w_1\ =\ \langle\!\langle\ \lceil \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2\ \rceil \mid w\ \rangle\!\rangle$	MHANDLE
$\langle\!\langle\ \lceil \mathsf{E}_1 \rceil \mid w\ \rangle\!\rangle\ =\ \langle\!\langle\ \lceil \mathsf{E}_1 \cdot \mathsf{handle}_{m_1}^{w_1}\ h \cdot \mathsf{E}_2\ \rceil \mid w\ \rangle\!\rangle$	follows
contradiction	

□

B.4 Monadic Translation

During the proof, we also use the inverse monadic bind, defined as

$$g \lhd \mathit{pure}\ x\ =\ g\ x$$
$$g \lhd (\mathit{yield}\ m\ f\ \mathit{cont})\ =\ \mathit{yield}\ m\ f\ (g \bullet \mathit{cont})$$

B.4.1 Multi-Prompt Delimited Continuations.

Proof. (*of Theorem 9*) By induction over the evaluation rules. In particular, $\mathsf{handle}_m^w\ h \cdot \mathsf{E} \cdot \mathsf{perform}^l\ op\ w'\ v \longrightarrow f\ w\ v\ w\ k$ where $op \to f\ \in h$ (**1**), $k\ =\ \mathsf{guard}^w\ (\mathit{handle}_m^w\ h \cdot \mathsf{E})$ (**2**), and $op \notin \mathsf{bop}(\mathsf{E})$ (**3**).

In that case, by Theorem 5, we have $w'.l\ =\ (m, h)$ (**4**), and can thus derive:

$$
\begin{array}{ll}
\lceil \mathsf{handle}_m^w\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ w'\ v \rceil & (1),(4),\text{translation} \\
=\ \mathsf{prompt}_m^w \cdot \lceil \mathsf{E} \rceil \cdot \mathsf{yield}_m\ (\lambda w\ k.\ \lceil f \rceil\ w\ \lceil v \rceil\ w\ k) & \\
m \notin \lceil \mathsf{E} \rceil^m & (3),\ \text{Theorem 6} \\
\longrightarrow (\lambda w\ k.\ \lceil f \rceil\ w\ \lceil v \rceil\ w\ k)\ w\ (\mathsf{guard}^w\ (\mathsf{prompt}_m^w \cdot \lceil \mathsf{E} \rceil)) & (2),\ (yield) \\
\longrightarrow \lceil f \rceil\ w\ \lceil v \rceil\ w\ (\mathsf{guard}^w\ (\mathsf{prompt}_m^w \cdot \lceil \mathsf{E} \rceil) & \\
=\ \lceil f \rceil\ w\ \lceil v \rceil\ w\ (\mathsf{guard}^w\ \lceil \mathsf{handle}_m^w\ h \cdot \mathsf{E} \rceil) & \\
=\ \lceil f\ w\ v\ w\ (\mathsf{guard}^w\ (\mathsf{handle}_m^w\ h \cdot \mathsf{E})) \rceil &
\end{array}
$$

□

B.4.2 Monadic Type Translation.

Lemma 33. (*Monadic Translation Stable under substitution*)
$\lfloor \sigma \rfloor [\alpha := \lfloor \sigma' \rfloor] \ = \ \lfloor \sigma[\alpha := \sigma'] \rfloor$.

Proof. (*Of Lemma 33*) By induction on σ.

case $\sigma = \alpha$.

$$
\begin{array}{ll}
\lfloor \alpha \rfloor [\alpha := \lfloor \sigma' \rfloor] & \\
=\ \alpha[\alpha := \lfloor \sigma' \rfloor] & \text{by translation} \\
=\ \lfloor \sigma' \rfloor & \text{by substitution} \\
\lfloor \alpha[\alpha := \sigma'] \rfloor & \\
=\ \lfloor \sigma' \rfloor & \text{by substitution}
\end{array}
$$

case $\sigma = \beta$ and $\beta \neq \alpha$.

$$
\begin{array}{ll}
\lfloor \beta \rfloor [\alpha := \lfloor \sigma' \rfloor] & \\
=\ \beta[\alpha := \lfloor \sigma' \rfloor] & \text{by translation} \\
=\ \beta & \text{by substitution} \\
\lfloor \beta[\alpha := \sigma'] \rfloor & \\
=\ \lfloor \beta \rfloor & \text{by substitution} \\
=\ \beta & \text{by translation}
\end{array}
$$

case $\sigma = \sigma_1 \Rightarrow\ \epsilon\ \sigma_2$.

$$
\begin{array}{ll}
\lfloor \sigma_1 \Rightarrow\ \epsilon\ \sigma_2 \rfloor [\alpha := \lfloor \sigma' \rfloor] & \\
=\ (\mathsf{evv}\ \epsilon \rightarrow \lfloor \sigma_1 \rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor \sigma_2 \rfloor)[\alpha := \lfloor \sigma' \rfloor] & \text{by translation} \\
=\ \mathsf{evv}\ \epsilon \rightarrow \lfloor \sigma_1 \rfloor [\alpha := \lfloor \sigma' \rfloor] \rightarrow \mathsf{mon}\ \epsilon\ (\lfloor \sigma_2 \rfloor [\alpha := \lfloor \sigma' \rfloor]) & \text{by substitution} \\
=\ \mathsf{evv}\ \epsilon \rightarrow (\lfloor \sigma_1[\alpha := \sigma'] \rfloor) \rightarrow \mathsf{mon}\ \epsilon\ (\lceil \sigma_2[\alpha := \sigma'] \rceil) & \text{I.H.} \\
\lfloor (\sigma_1 \Rightarrow\ \epsilon\ \sigma_2)[\alpha := \sigma'] \rfloor & \\
=\ \lceil \sigma_1[\alpha := \sigma'] \Rightarrow \epsilon\ \sigma_2[\alpha := \sigma'] \rceil & \text{by substitution} \\
=\ \mathsf{evv}\ \epsilon \rightarrow (\lfloor \sigma_1[\alpha := \sigma'] \rfloor) \rightarrow \mathsf{mon}\ \epsilon\ (\lfloor \sigma_2[\alpha := \sigma'] \rfloor) & \text{by translation}
\end{array}
$$

case $\sigma = \forall \beta.\ \sigma_1$.

$$
\begin{array}{ll}
\lfloor \forall \beta.\ \sigma_1 \rfloor [\alpha := \lfloor \sigma' \rfloor] & \\
=\ (\forall \beta.\ \lfloor \sigma_1 \rfloor)[\alpha := \lfloor \sigma' \rfloor] & \text{by translation} \\
=\ \forall \beta.\ \lfloor \sigma_1 \rfloor [\alpha := \lfloor \sigma' \rfloor] & \text{by substitution} \\
=\ \forall \beta.\ \lfloor \sigma_1[\alpha := \sigma'] \rfloor & \text{I.H.} \\
\lfloor (\forall \beta.\ \sigma_1)[\alpha := \sigma'] \rfloor & \\
=\ \lfloor \forall \beta.\ \sigma_1[\alpha := \sigma'] \rfloor & \text{by substitution} \\
=\ \forall \beta.\ \lfloor \sigma_1[\alpha := \sigma'] \rfloor & \text{by translation}
\end{array}
$$

case $\sigma = c\ \tau_1\ \ldots\ \tau_n$.

$\lfloor c\ \tau_1 \ldots \tau_n \rfloor[\alpha{:=}\lfloor\sigma'\rfloor]$
$= (c\ \lfloor\tau_1\rfloor \ldots \lfloor\tau_n\rfloor)[\alpha{:=}\lfloor\sigma'\rfloor]$ by translation
$= c\ (\lfloor\tau_1\rfloor[\alpha{:=}\lfloor\sigma'\rfloor]) \ldots (\lfloor\tau_n\rfloor[\alpha{:=}\lfloor\sigma'\rfloor])$ by substitution
$= c\ (\lfloor\tau_1[\alpha{:=}\sigma']\rfloor) \ldots (\lfloor\tau_n[\alpha{:=}\sigma']\rfloor)$ by I.H.
$\lfloor(c\ \tau_1 \ldots \tau_n)[\alpha{:=}\sigma']\rfloor$
$= \lfloor c\ \tau_1[\alpha{:=}\sigma'] \ldots \tau_n[\alpha{:=}\sigma']\rfloor$ by substitution
$= c\ (\lfloor\tau_1[\alpha{:=}\sigma']\rfloor) \ldots (\lfloor\tau_n[\alpha{:=}\sigma']\rfloor)$ by translation
□

B.4.3 Substitution.

Lemma 34. (*Monadic Translation Variable Substitution*)
1. If $\Gamma_1,\ x{:}\ \sigma,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v_1 : \sigma_1 \rightsquigarrow v_1'$, and $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$,
then $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v_1\,[x{:=}\ v] : \sigma_1 \rightsquigarrow v_1'[x{:=}v']$.
2. If $\Gamma_1,\ x{:}\ \sigma,\ \Gamma_2\ ;\ w;\ w' \Vdash\ e_1 : \sigma_1 \mid \epsilon \rightsquigarrow e_1'$ and $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$,
then $\Gamma_1,\ \Gamma_2\ ;\ w\,[x := v];\ w'[x{:=}v'] \Vdash\ e_1\,[x{:=}\ v] : \sigma_1 \mid \epsilon \rightsquigarrow e_1'[x{:=}v']$.
3. If $\Gamma_1, x{:}\sigma, \Gamma_2 \Vdash_{\mathsf{ops}}\ \{\, op_1 \to f_1,\ \ldots,\ op_n \to f_n \,\} : \mathsf{hnd}^l\ \epsilon\ \sigma_1 \mid \epsilon \rightsquigarrow e$ and $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$,
then $\Gamma_1, \Gamma_2 \Vdash_{\mathsf{ops}}\ (\{\, op_1 \to f_1,\ \ldots,\ op_n \to f_n \,\})[x{:=}v] : \mathsf{hnd}^l\ \epsilon\ \sigma_1 \mid \epsilon \rightsquigarrow e[x{:=}v']$.
4. If $\Gamma_1, x{:}\sigma, \Gamma_2\ ;\ w\ ;\ w' \Vdash_{\mathsf{ec}}\ \mathsf{E} : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow g$ and $\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$,
then $\Gamma_1, \Gamma_2\ ;\ w[x{:=}v]\ ;\ w'[x{:=}v'] \Vdash_{\mathsf{ec}}\ \mathsf{E}\,[x{:=}v] : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow g[x{:=}v']$.

Proof. (*Of Lemma 34*) **Part 1** By induction on typing.
case $v_1 = x$.
$\sigma = \sigma_1$ MVAL
$\Gamma_1,\ x{:}\ \sigma,\ \Gamma_2 \Vdash_{\mathsf{val}}\ x : \sigma \rightsquigarrow x$ given
$\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma_1 \rightsquigarrow v'$ given
case $v_1 = y$ where $y \neq x$.
$v_1[x{:=}v] = y$ by substitution
$v_1'[x{:=}v'] = y$ by substitution
$y : \sigma_1 \in \Gamma_1,\ \Gamma_2$ MVAR
$\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ y : \sigma_1 \rightsquigarrow y$ MVAR
case $v_1 = \lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ y : \sigma_2.\ e$.
$\Gamma_1, x{:}\sigma, \Gamma_2 \Vdash_{\mathsf{val}}\ \lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ y : \sigma_2.\ e : \sigma_1 \rightsquigarrow \lambda z\ x.\ e'$ given
$\sigma_1 = \sigma_2 \Rightarrow \epsilon\ \sigma_3$ MABS
$(\Gamma_1, x{:}\sigma,\ \Gamma_2,\ z{:}\ \mathsf{evv}\ \epsilon,\ y{:}\ \sigma_2)\ ;\ z;\ z \Vdash\ e : \sigma_3 \mid \epsilon \rightsquigarrow e'$ above
$\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$ given
$\Gamma_1,\ \Gamma_2,\ z{:}\ \mathsf{evv}\ \epsilon,\ y{:}\ \sigma_2 \Vdash_{\mathsf{val}}\ v : \sigma \rightsquigarrow v'$ weakening
$(\Gamma_1,\ \Gamma_2,\ z{:}\ \mathsf{evv}\ \epsilon,\ y{:}\ \sigma_2)\ ;\ z;\ z \Vdash\ e[x{:=}v] : \sigma_3 \mid \epsilon \rightsquigarrow e'[x{:=}v']$ Part 2
$\Gamma_1,\ \Gamma_2 \Vdash_{\mathsf{val}}\ \lambda^\epsilon\ z{:}\mathsf{evv}\ \epsilon,\ y : \sigma_2.\ e[x{:=}v] : \sigma_1 \rightsquigarrow \lambda z\ x.\ e'[x{:=}v']$ MABS
case $v_1 = \mathsf{guard}^w\ \mathsf{E}\ \sigma_1$.
$\Gamma_1, x{:}\sigma, \Gamma_2 \Vdash_{\mathsf{val}}\ \mathsf{guard}^w\ \mathsf{E}\ \sigma_2 : \sigma_1 \rightsquigarrow \mathit{guard}\ w'\ e'$ given
$\sigma_1 = \sigma_2 \Rightarrow \epsilon\ \sigma_3$ MGUARD
$\Gamma_1, x{:}\sigma, \Gamma_2\ ;\ w\ ;\ w' \Vdash_{\mathsf{ec}}\ \mathsf{E} : \sigma_2 \to \sigma_3 \mid \epsilon \rightsquigarrow e'$ above
$\Gamma_1, x{:}\sigma, \Gamma_2 \Vdash_{\mathsf{val}}\ w : \mathsf{evv}\ \epsilon \rightsquigarrow w'$ above
$\Gamma_1, \Gamma_2\ ;\ w[x{:=}v];\ w'[x{:=}v'] \Vdash\ \mathsf{E}\,[x{:=}v] : \sigma_2 \to \sigma_3 \mid \epsilon \rightsquigarrow e'[x{:=}v']$ Part 4
$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\ w[x{:=}v] : \mathsf{evv}\ \epsilon \rightsquigarrow w'[x{:=}v']$ I.H.
$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\ \mathsf{guard}^{w[x{:=}v]}\ \mathsf{E}[x{:=}v]\ \sigma_2 : \sigma_2 \Rightarrow \epsilon\ \sigma_3 \rightsquigarrow \mathit{guard}\ w'[x{:=}v']\ e'[x{:=}v']$ MGUARD

case $v_1 \;=\; \Lambda\alpha.\; v_2.$

$\Gamma_1, x : \sigma, \Gamma_2 \Vdash_{\mathsf{val}}\; \Lambda\alpha.\; v_2 \;:\; \sigma_1 \;\rightsquigarrow \Lambda\alpha.\; v_2'$ given

$\sigma_1 \;=\; \forall\alpha.\; \sigma_2$ MTABS

$\Gamma_1, x : \sigma, \Gamma_2 \Vdash_{\mathsf{val}}\; v_2 \;:\; \sigma_2$ above

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\; v_2[x{:=}v] \;:\; \sigma_2 \;\rightsquigarrow v_2'[x{:=}v']$ I.H.

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\; \Lambda\alpha.\; v_2[x{:=}v] \;:\; \forall\alpha \cdot \sigma_2 \;\rightsquigarrow \Lambda\alpha.\; v_2'[x{:=}v']$ MTABS

case $v_1 \;=\; \mathsf{perform}\; op\; \overline{\sigma}.$

$\Gamma_1, x : \sigma, \Gamma_2 \Vdash_{\mathsf{val}}\; \mathsf{perform}\; op\; \overline{\sigma} \;:\; \sigma_1 \;\rightsquigarrow perform^{op}\; [\langle l \mid \mu\rangle,\; \lfloor\overline{\sigma}\rfloor]$ given

$\sigma_1 \;=\; \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow\; \langle l \mid \mu\rangle\; \sigma_3[\overline{\alpha}{:=}\overline{\sigma}]$ MPERFORM

$op \;:\; \forall\overline{\alpha}.\; \sigma_2 \rightarrow \sigma_3$ ß$\Sigma(l)$ above

$(\mathsf{perform}\; op\; \overline{\sigma})[x{:=}v] \;=\; \mathsf{perform}\; op\; \overline{\sigma}$ by substitution

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\; \mathsf{perform}\; op\; \overline{\sigma} \;:\; \sigma_1 \;\rightsquigarrow perform^{op}\; [\langle l \mid \mu\rangle,\; \lfloor\overline{\sigma}\rfloor]$ MPERFORM

case $v_1 \;=\; \mathsf{handler}^{\epsilon}\; h.$

$\Gamma_1, x : \sigma, \Gamma_2 \Vdash_{\mathsf{val}}\; handler^{\epsilon}\; h \;:\; \sigma_1 \;\rightsquigarrow handler^{l}\; [\epsilon,\; \lfloor\sigma\rfloor]\; h'$ given

$\sigma_1 \;=\; (() \Rightarrow\; \langle l \mid \epsilon\rangle\; \sigma) \Rightarrow \epsilon\; \sigma$ MHANDLER

$\Gamma_1, x : \sigma, \Gamma_2 \Vdash_{\mathsf{ops}}\; h \;:\; \mathsf{hnd}^{l}\; \epsilon\; \sigma \mid \epsilon \;\rightsquigarrow h'$ above

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{ops}}\; h[x{:=}v] \;:\; \mathsf{hnd}^{l}\; \epsilon\; \sigma \mid \epsilon \;\rightsquigarrow h'[x{:=}v']$ Part 3

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\; handler^{\epsilon}\; h[x{:=}v] \;:\; \sigma_1 \;\rightsquigarrow handler^{l}\; [\epsilon,\; \lfloor\sigma\rfloor]\; h'[x{:=}v']$ MHANDLER

Part 2 By induction on typing.

case $e_1 \;=\; v_1.$

$\Gamma_1, x : \sigma, \Gamma_2\; ;\; w\; ;\; w' \;\Vdash\; v_1 \;:\; \sigma_1 \;\mid \epsilon \;\rightsquigarrow v_1'$ given

$\Gamma_1, x : \sigma, \Gamma_2 \Vdash_{\mathsf{val}}\; v_1 \;:\; \sigma_1 \;\rightsquigarrow v_1'$ MVAL

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}}\; v_1[x{:=}v] \;:\; \sigma_1 \;\rightsquigarrow v_1'[x{:=}v']$ Part 1

$\Gamma_1, \Gamma_2;\; w\,[x{:=}v];\; w'[x{:=}v'] \Vdash_{\mathsf{val}}\; v_1[x{:=}v] \;:\; \sigma_1 \mid \epsilon \;\rightsquigarrow v_1'[x{:=}v']$ MVAL

case $e_1 \;=\; e_2\; w\; e_3.$

$\Gamma_1, x : \sigma, \Gamma_2; w; w' \Vdash\; e_2\; w\; e_3 \;:\; \sigma_1 \mid \epsilon \;\rightsquigarrow e_2' \triangleright (\lambda f.\; e_3' \triangleright f\; w')$ given

$\Gamma_1, x : \sigma, \Gamma_2; w; w' \Vdash\; e_2 : \sigma_2 \Rightarrow \epsilon\; \sigma_1 \mid \epsilon \;\rightsquigarrow e_2'$ MAPP

$\Gamma_1, x : \sigma, \Gamma_2; w; w' \Vdash\; e_3 : \sigma_2 \mid \epsilon \;\rightsquigarrow e_3'$ above

$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash\; e_2[x{:=}v] : \sigma_2 \Rightarrow \epsilon\; \sigma_1 \mid \epsilon \;\rightsquigarrow e_2'[x{:=}v']$ I.H.

$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash\; e_3[x{:=}v] : \sigma_2 \mid \epsilon \;\rightsquigarrow e_2'[x{:=}v']$ I.H.

$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash\; e_2[x{:=}v]\; w[x{:=}v]\; e_3[x{:=}v] \;:\; \sigma_1 \mid \epsilon$ MAPP

$\rightsquigarrow e_2'[x{:=}v'] \triangleright (\lambda f.\; e_3'[x{:=}v'] \triangleright f\; w'[x{:=}v'])$

case $e_1 \;=\; e_2\; [\sigma_2].$

$\Gamma_1, x : \sigma, \Gamma_2; w; w' \Vdash\; e_2\; [\sigma_2] : \sigma_1 \mid \epsilon \;\rightsquigarrow e_2' \triangleright (\lambda x.\; pure\; (x\; [\lfloor\sigma_2\rfloor]))$ given

$\sigma_1 \;=\; \sigma_3\; [\alpha{:=}\sigma_2]$ MTAPP

$\Gamma_1, x : \sigma, \Gamma_2; w; w' \Vdash\; e_2 \;:\; \forall\alpha.\; \sigma_3 \;\mid \epsilon \;\rightsquigarrow e_2'$ above

$\Gamma_1, \Gamma_2; w[x{:=}v]; w[x{:=}v'] \Vdash\; e_2[x{:=}v] \;:\; \forall\alpha.\; \sigma_3 \mid \epsilon \;\rightsquigarrow e_2'[x{:=}v']$ I.H.

$\Gamma_1, \Gamma_2; w[x{:=}v]; w[x{:=}v'] \Vdash\; e_2[x{:=}v]\; [\sigma_2] \;:\; \sigma_3[\alpha{:=}\sigma_2] \mid \epsilon$ MTAPP

$\rightsquigarrow e_2'[x{:=}v'] \triangleright (\lambda x.\; pure\; (x\; [\lfloor\sigma_2\rfloor]))$

case $e_1 \;=\; \mathsf{handle}^{w}_{m}\; h\; e_2.$

$\Gamma_1, x:\sigma, \Gamma_2; w; w' \Vdash \mathsf{handle}^w_m\ h\ e_2 : \sigma_1 \mid \epsilon \rightsquigarrow \mathit{prompt}\ m\ w'\ e'_2$ given
$\Gamma_1, x:\sigma, \Gamma_2 \Vdash_{\mathsf{ops}} h : \mathsf{hnd}^\epsilon\ \sigma_1 \mid \epsilon \rightsquigarrow h'$ MHANDLE
$\Gamma_1, x:\sigma, \Gamma_2; \langle\!\langle l:(m,h) \mid w\rangle\!\rangle; \langle\!\langle l:(m,h') \mid w'\rangle\!\rangle \Vdash e_2 : \sigma_1 \mid \langle l \mid \epsilon\rangle \rightsquigarrow e'_2$ above
$h \in \Sigma(l)$ above
$\Gamma_1, \Gamma_2; (\langle\!\langle l:(m,h) \mid w\rangle\!\rangle)[x{:=}v]; (\langle\!\langle l:(m,h') \mid w'\rangle\!\rangle)[x{:=}v']$ I.H.
$\quad \Vdash e_2[x{:=}v] : \sigma_1 \mid \langle l \mid \epsilon\rangle \rightsquigarrow e'_2[x{:=}v']$
$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{ops}} h[x{:=}v] : \mathsf{hnd}^\epsilon\ \sigma_1 \mid \epsilon \rightsquigarrow h'[x{:=}v']$ Part 3
$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash \mathsf{handle}^{w[x:=v]}_m\ h[x{:=}v]\ e_2[x{:=}v] : \sigma_1 \mid \langle\epsilon\rangle$ MHANDLE
$\quad \rightsquigarrow \mathit{prompt}\ m\ w'[x{:=}v']\ e'_2[x{:=}v']$

Part 3

$\Gamma_1, x:\sigma, \Gamma_2 \Vdash_{\mathsf{ops}} \{\ op_1 \to f_1, \ldots, op_n \to f_n\ \} : \mathsf{hnd}^l\ \epsilon\ \sigma_1 \mid \epsilon \rightsquigarrow \{op_1 \to f'_1, \ldots, op_n \to f'_n\}$ given
$\Gamma_1, x:\sigma, \Gamma_2 \Vdash_{\mathsf{val}} f_i : \forall\overline{\alpha}.\ \sigma_1 \Rightarrow \epsilon\ (\sigma_2 \Rightarrow \epsilon\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow f'_i$ MOPS
$op_i : \forall\overline{\alpha}.\ \sigma_1 \to \sigma_2 \in \Sigma(l) \quad \overline{\alpha} \not\cap \mathrm{ftv}(\epsilon\ \sigma)$ above
$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}} f_i[x{:=}v] : \forall\overline{\alpha}.\ \sigma_1 \Rightarrow \epsilon\ (\sigma_2 \Rightarrow \epsilon\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow f'_i[x{:=}v']$ Part 1
$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{ops}} \{\ op_1 \to f_1[x{:=}v], \ldots, op_n \to f_n[x{:=}v]\ \} : \mathsf{hnd}^l\ \epsilon\ \sigma_1 \mid \epsilon$ MOPS
$\quad \rightsquigarrow \{op_1 \to f'_1[x{:=}v'], \ldots, op_n \to f'_n[x{:=}v']\}$

Part 4 By induction on typing.

case $\mathsf{E} = \Box$.

$\Gamma_1, x:\sigma, \Gamma_2 ; w ; w' \Vdash_{\mathsf{ec}} \Box : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow \mathit{id}$ given
$\sigma_1 = \sigma_2$ MON-CEMPTY
$\Box[x{:=}v] = \Box$ by substitution
$\Gamma_1, \Gamma_2 ; w[x{:=}v] ; w'[x{:=}v'] \Vdash_{\mathsf{ec}} \Box : \sigma_1 \to \sigma_1 \mid \epsilon \rightsquigarrow \mathit{id}$ MON-CEMPTY

case $\mathsf{E} = \mathsf{E}_1\ w\ e$.

$\Gamma_1, x:\sigma, \Gamma_2 ; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_1\ w\ e : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow (\lambda f.\ e' \rhd f\ w) \bullet g$ given
$\Gamma_1, x:\sigma, \Gamma_2 ; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_1 : \sigma_1 \to (\sigma_3 \Rightarrow \epsilon\ \sigma_2) \mid \epsilon \rightsquigarrow g$ MON-CAPP1
$\Gamma_1, x:\sigma, \Gamma_2 ; w; w' \Vdash e : \sigma_3 \mid \epsilon \rightsquigarrow e'$ above
$\Gamma_1, \Gamma_2 ; w[x{:=}v]; w'[x{:=}v'] \Vdash_{\mathsf{ec}} \mathsf{E}_1[x{:=}v] : \sigma_1 \to (\sigma_3 \Rightarrow \epsilon\ \sigma_2) \mid \epsilon \rightsquigarrow g[x{:=}v']$ I.H.
$\Gamma_1, \Gamma_2 ; w[x{:=}v]; w'[x{:=}v'] \Vdash e[x{:=}v] : \sigma_3 \mid \epsilon \rightsquigarrow e'[x{:=}v']$ Part 2
$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash_{\mathsf{ec}} \mathsf{E}_1[x{:=}v]\ w[x{:=}v]\ e[x{:=}v] : \sigma_1 \to \sigma_2 \mid \epsilon$ MON-CAPP1
$\quad \rightsquigarrow (\lambda f.\ e'[x{:=}v'] \rhd f\ w[x{:=}v']) \bullet g[x{:=}v']$

case $\mathsf{E} = v_1\ w\ \mathsf{E}_1$.

$\Gamma_1, x:\sigma, \Gamma_2 ; w; w' \Vdash_{\mathsf{ec}} v_1\ w\ \mathsf{E}_1 : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow v'_1\ w' \bullet g$ given
$\Gamma_1, x:\sigma, \Gamma_2 ; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_1 : \sigma_1 \to \sigma_3 \mid \epsilon \rightsquigarrow g$ MON-CAPP2
$\Gamma_1, x:\sigma, \Gamma_2 \Vdash_{\mathsf{val}} v_1 : \sigma_3 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow v'_1$ above
$\Gamma_1, \Gamma_2 ; w[x{:=}v]; w'[x{:=}v'] \Vdash_{\mathsf{ec}} \mathsf{E}_1[x{:=}v] : \sigma_1 \to \sigma_3 \mid \epsilon \rightsquigarrow g[x{:=}v']$ I.H.
$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{val}} v_1[x{:=}v] : \sigma_3 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow v'_1[x{:=}v']$ Part 2
$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash_{\mathsf{ec}} v_1[x{:=}v]\ w[x{:=}v]\ \mathsf{E}_1[x{:=}v] : \sigma_1 \to \sigma_2 \mid \epsilon$ MON-CAPP2
$\quad \rightsquigarrow v'_1[x{:=}v']\ w'[x{:=}v']$

case $\mathsf{E} = \mathsf{E}_1\ [\sigma]$.

$\Gamma_1, x:\sigma, \Gamma_2 ; w ; w' \Vdash_{\mathsf{ec}} \mathsf{E}_1\ [\sigma] : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow (\lambda x.\ \mathit{pure}\ x) \bullet g$ given
$\Gamma_1, x:\sigma, \Gamma_2 ; w ; w' \Vdash_{\mathsf{ec}} \mathsf{E}_1 : \sigma_1 \to \forall\alpha.\ \sigma_3 \mid \epsilon \rightsquigarrow g$ MON-CTAPP
$\sigma_2 = \sigma_3[\alpha{:=}\sigma]$ above
$\Gamma_1, \Gamma_2 ; w[x{:=}v] \Vdash_{\mathsf{ec}} \mathsf{E}_1[x{:=}v] : \sigma_1 \to \forall\alpha.\ \sigma_3 \mid \epsilon \rightsquigarrow g[x{:=}v']$ I.H.
$\Gamma_1, x:\sigma, \Gamma_2; w[x{:=}v] \Vdash_{\mathsf{ec}} \mathsf{E}_1[x{:=}v]\ [\sigma] : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow (\lambda x.\ \mathit{pure}\ x) \bullet g[x{:=}v']$ MON-CTAPP

case $\mathsf{E} = \mathsf{handle}^w_m\ h\ \mathsf{E}_1$.

$\Gamma_1, x:\sigma, \Gamma_2; w; w' \Vdash_{\mathsf{ec}}\ \mathsf{handle}^w_m\ h\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma_2 \mid \epsilon\ \rightsquigarrow\ \mathit{prompt}\ m\ w' \circ g$ given

$\Gamma_1, x:\sigma, \Gamma_2; w; w' \Vdash_{\mathsf{ops}}\ h\ :\ \mathsf{hnd}^l\ \epsilon\ \sigma_2 \mid \epsilon\ \rightsquigarrow\ h'$ above

$\Gamma_1, x:\sigma, \Gamma_2; \langle\!\langle l:(m, h) \mid w \rangle\!\rangle; \langle\!\langle l:(m, h') \mid w' \rangle\!\rangle \Vdash_{\mathsf{ec}}\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma_2 \mid \langle l \mid \epsilon \rangle\ \rightsquigarrow\ g$ above

$\Gamma_1, \Gamma_2; \langle\!\langle l:(m, h[x{:=}v]) \mid w[x{:=}v] \rangle\!\rangle; \langle\!\langle l:(m, h'[x{:=}v']) \mid w'[x{:=}v'] \rangle\!\rangle$ I.H.

$\quad \Vdash_{\mathsf{ec}}\ \mathsf{E}[x{:=}v]\ :\ \sigma_1 \rightarrow \sigma_2 \mid \langle l \mid \epsilon \rangle\ \rightsquigarrow\ g[x{:=}v']$

$\Gamma_1, \Gamma_2 \Vdash_{\mathsf{ops}}\ h[x{:=}v]\ :\ \mathsf{hnd}^l\ \epsilon\ \sigma_2 \mid \epsilon\ \rightsquigarrow\ h'[x{:=}v']\ ]$ Part 3

$\Gamma_1, \Gamma_2; w[x{:=}v]; w'[x{:=}v'] \Vdash_{\mathsf{ec}}\ \mathsf{handle}^{w[x{:=}v]}_m\ h[x{:=}v]\ \mathsf{E}[x{:=}v]\ :\ \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ MON-CHANDLE

$\quad \rightsquigarrow\ \mathit{prompt}\ m\ w'[x{:=}v'] \circ g[x{:=}v']$

□

Lemma 35. (*Monadic Translation Type Variable Substitution*)

1. If $\Gamma \Vdash_{\mathsf{val}}\ v\ :\ \sigma\ \rightsquigarrow\ v'$ and $\vdash_{\mathsf{wf}}\ \sigma_1\ :\ k$,
then $\Gamma[\alpha^k{:=}\sigma_1] \Vdash_{\mathsf{val}}\ v[\alpha^k{:=}\sigma_1]\ :\ \sigma[\alpha^k{:=}\sigma_1]\ \rightsquigarrow\ v'[\alpha^k{:=}\lfloor\sigma_1\rfloor]$.
2. If $\Gamma; w; w' \Vdash\ e\ :\ \sigma \mid \epsilon\ \rightsquigarrow\ e'$ and $\Vdash_{\mathsf{wf}}\ \sigma_1\ :\ k$,
then $\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\sigma_1]; w'[\alpha^{\lfloor k \rfloor}{:=}\lfloor\sigma_1\rfloor] \Vdash\ e[\alpha^k{:=}\sigma_1]\ :\ \sigma[\alpha^k{:=}\sigma_1] \mid \epsilon\ \rightsquigarrow\ e'[\alpha^k{:=}\lfloor\sigma_1\rfloor]$.
3. If $\Gamma \Vdash_{\mathsf{ops}}\ h\ :\ \sigma \mid l \mid \epsilon\ \rightsquigarrow\ h'$ and $\vdash_{\mathsf{wf}}\ \sigma_1\ :\ k$,
then $\Gamma[\alpha^k{:=}\sigma_1] \Vdash_{\mathsf{ops}}\ h[\alpha^k{:=}\sigma_1]\ :\ \sigma[\alpha^k{:=}\sigma_1] \mid l \mid \epsilon\ \rightsquigarrow\ v'[\alpha^k{:=}\lfloor\sigma_1\rfloor]$.
4. If $\Gamma; w; w' \Vdash\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma_2 \mid \epsilon\ \rightsquigarrow\ g$ and $\vdash_{\mathsf{wf}}\ \sigma_1\ :\ k$,
then $\Gamma[\alpha^k{:=}\sigma_1]; w[\alpha^k{:=}\sigma_1]; w'[\alpha^k = \lfloor\sigma_1\rfloor] \Vdash\ \mathsf{E}[\alpha^k{:=}\sigma_1]\ :\ \sigma_1[\alpha^k{:=}\sigma_1] \rightarrow \sigma_2[\alpha^k{:=}\sigma_1]\ \rightsquigarrow\ v'[\alpha^k{:=}\lfloor\sigma_1\rfloor]$.

Proof. (*Of Lemma 35*) **Part 1** By induction on typing.

case $v\ =\ x$.

$\Gamma \Vdash_{\mathsf{val}}\ x\ :\ \sigma\ \rightsquigarrow x$ given

$x:\sigma\ \in \Gamma$ MVAR

$x:\sigma[\alpha{:=}\sigma_1]\ \in \Gamma[\alpha{:=}\sigma_1]$ follows

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}}\ x\ :\ \sigma[\alpha{:=}\sigma_1]\ \rightsquigarrow x$ MVAR

case $v\ =\ \lambda^\epsilon\ z:\mathsf{evv}\ \epsilon,\ y\ :\ \sigma_2.\ e$.

$\Gamma \Vdash_{\mathsf{val}}\ \lambda^\epsilon\ z:\mathsf{evv}\ \epsilon,\ y\ :\ \sigma_2.\ e\ \ :\ \sigma_2 \Rightarrow \epsilon\ \sigma_3\ \rightsquigarrow \lambda z\ x.\ e'$ given

$(\Gamma,\ z:\ \mathsf{evv}\ \epsilon,\ y:\ \sigma_2)\ ;\ z;\ z\ \Vdash\ e\ :\ \sigma_3 \mid \epsilon\ \rightsquigarrow e'$ MABS

$(\Gamma[\alpha{:=}\sigma_1]\ z:\ \mathsf{evv}\ \epsilon,\ y:\ \sigma_2[\alpha{:=}\sigma_1])\ ;\ z;\ z\ \Vdash\ e[\alpha{:=}\sigma_1]\ :\ \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon\ \rightsquigarrow e'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 2

$\Gamma \Vdash_{\mathsf{val}}\ \lambda^\epsilon\ z:\mathsf{evv}\ \epsilon,\ y\ :\ \sigma_2[\alpha{:=}\sigma_1].\ e[\alpha{:=}\sigma_1]\ :\ \sigma_2[\alpha{:=}\sigma_1] \Rightarrow \epsilon\ \sigma_3[\alpha{:=}\sigma_1]\ \rightsquigarrow \lambda z\ x.\ e'[\alpha{:=}\lfloor\sigma_1\rfloor]$ MABS

case $v\ =\ \mathsf{guard}^w\ \mathsf{E}\ \sigma_2$.

$\Gamma \Vdash_{\mathsf{val}}\ \mathsf{guard}^w\ \mathsf{E}\ \sigma_2\ :\ \sigma_2 \Rightarrow \epsilon\ \sigma_3\ \rightsquigarrow \mathit{guard}\ w'\ e'$ given

$\Gamma\ ;\ w\ ; w' \Vdash_{\mathsf{ec}}\ \mathsf{E}\ :\ \sigma_2 \rightarrow \sigma_3 \mid \epsilon\ \rightsquigarrow e'$ MGUARD

$\Gamma \Vdash_{\mathsf{val}}\ w\ :\ \mathsf{evv}\ \epsilon\ \rightsquigarrow w'$ above

$\Gamma[\alpha{:=}\sigma_1]\ ;\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\sigma_1]\ \Vdash\ \mathsf{E}\ [\alpha{:=}\sigma_1]\ :\ \sigma_2[\alpha{:=}\sigma_1] \rightarrow \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon\ \rightsquigarrow e'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 4

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}}\ w[\alpha{:=}\sigma_1]\ :\ \mathsf{evv}\ \epsilon\ \rightsquigarrow w'[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}}\ \mathsf{guard}^{w[\alpha{:=}\sigma_1]}\ \mathsf{E}[\alpha{:=}\sigma_1]\ \sigma_2\ :\ \sigma_2[\alpha{:=}\sigma_1] \Rightarrow \epsilon\ \sigma_3[\alpha{:=}\sigma_1]$ MGUARD

$\quad \rightsquigarrow \mathit{guard}\ w'[\alpha{:=}\lfloor\sigma_1\rfloor]\ e'[\alpha{:=}\lfloor\sigma_1\rfloor]$

case $v\ =\ \Lambda\alpha.\ v_2$.

$\Gamma \Vdash_{\mathsf{val}}\ \Lambda\alpha.\ v_2\ :\ \forall\alpha.\ \sigma_2\ \rightsquigarrow \Lambda\alpha.\ v_2'$ given

$\Gamma \Vdash_{\mathsf{val}}\ v_2\ :\ \sigma_2$ MTABS

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}}\ v_2[\alpha{:=}\sigma_1]\ :\ \sigma_2[\alpha{:=}\sigma_1]\ \rightsquigarrow v_2'[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}}\ \Lambda\alpha.\ v_2[\alpha{:=}\sigma_1]\ :\ \forall\alpha \cdot \sigma_2\ \rightsquigarrow \Lambda\alpha.\ v_2'[\alpha{:=}\lfloor\sigma_1\rfloor]$ MTABS

case $v\ =\ \mathsf{perform}\ \mathit{op}\ \overline{\sigma}$.

$\Gamma \Vdash_{\mathsf{val}} \ \mathsf{perform}\ op\ \overline{\sigma} \ : \ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow \ \langle l \mid \mu\rangle\ \sigma_3[\overline{\alpha}{:=}\overline{\sigma}] \ \rightsquigarrow \ perform^{op}\ [\langle l \ \mid \mu\rangle,\ \lfloor\overline{\sigma}\rfloor]$ given

$op \ : \ \forall\overline{\alpha}.\ \sigma_2 \rightarrow \sigma_3 \ \in \Sigma(l)$ MPERFORM

$(\mathsf{perform}\ op)[\alpha{:=}\sigma_1] \ = \ \mathsf{perform}\ op$ by substitution

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}} \ \mathsf{perform}\ op\ \overline{\sigma}[\alpha{:=}\sigma_1]\ ] \ : \ \sigma_2[\overline{\alpha}{:=}(\overline{\sigma}[\overline{\alpha}{:=}\overline{\sigma}])] \Rightarrow \ \langle l \mid \mu\rangle\ \sigma_3[\overline{\alpha}{:=}(\overline{\sigma}[\alpha{:=}\sigma_1])]$ MPERFORM

$\quad \rightsquigarrow perform^{op}\ [\langle l \mid \mu\rangle,\ \lfloor\overline{\sigma}\rfloor]$

$\sigma_2[\overline{\alpha}{:=}(\overline{\sigma}[\alpha{:=}\sigma_1])]$

$= \ (\sigma_2[\alpha{:=}\sigma_1])[\overline{\alpha}{:=}(\overline{\sigma}[\alpha{:=}\sigma_1])]$ α fresh to σ_2

$= \ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])[\alpha{:=}\sigma_1]$ by substitution

$\sigma_3[\overline{\alpha}{:=}(\overline{\sigma}[\alpha{:=}\sigma_1])] \ = \ (\sigma_3[\overline{\alpha}{:=}\overline{\sigma}])[\alpha^k{:=}\sigma_1]$ similarly

$\lfloor\overline{\sigma}[\alpha{:=}\sigma_1]\rfloor \ = \ \lfloor\overline{\sigma}\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]$ Lemma 15

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}} \ \mathsf{perform}\ op\ \overline{\sigma}[\alpha{:=}\sigma_1]$ therefore

$\quad : \ (\sigma_2[\overline{\alpha}{:=}\overline{\sigma}])[\alpha{:=}\sigma_1] \Rightarrow \langle l \mid \mu\rangle\ (\sigma_3[\overline{\alpha}{:=}\overline{\sigma}])[\alpha{:=}\sigma_1]$

$\quad \rightsquigarrow \mathsf{perform}^{op}\ [\langle l \mid \mu\rangle,\ \lfloor\overline{\sigma}\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]\ ]$

case $v \ = \ \mathsf{handler}^{\epsilon}\ h.$

$\Gamma \Vdash_{\mathsf{val}} \ handler^{\epsilon}\ h \ : \ \sigma_2 \ \rightsquigarrow handler^{l}\ [\epsilon,\ \lfloor\sigma\rfloor]\ h'$ given

$\sigma_2 \ = \ (() \Rightarrow \ \langle l \mid \epsilon\rangle\ \sigma) \Rightarrow \epsilon\ \sigma$ MHANDLER

$\Gamma \Vdash_{\mathsf{ops}} \ h \ : \ \mathsf{hnd}^{l}\ \epsilon\ \sigma \mid \epsilon \ \rightsquigarrow h'$ above

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{ops}} \ h[\alpha{:=}\sigma_1] \ : \ \mathsf{hnd}^{l}\ \epsilon\ \sigma[\alpha{:=}\sigma_1] \mid \epsilon \ \rightsquigarrow h'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 3

$\lfloor\sigma[\alpha{:=}\sigma_1]\rfloor \ = \ \lfloor\sigma\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]$ Lemma 15

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}} \ handler^{\epsilon}\ h[\alpha{:=}\sigma_1] \ : \ \sigma_2[\alpha{:=}\sigma_1] \ \rightsquigarrow handler^{l}\ [\epsilon,\ \lfloor\sigma\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]\ ]\ h'[\alpha{:=}\lfloor\sigma_1\rfloor]$ MHANDLER

Part 2 By induction on typing.

case $e \ = \ v.$

$\Gamma\ ;\ w\ ;\ w' \ \Vdash \ v \ : \ \sigma \ \mid \epsilon \ \rightsquigarrow v'$ given

$\Gamma \Vdash_{\mathsf{val}} \ v \ : \ \sigma \ \rightsquigarrow v'$ MVAL

$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\mathsf{val}} \ v[\alpha{:=}\sigma_1] \ : \ \sigma[\alpha{:=}\sigma_1] \ \rightsquigarrow v'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 1

$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\sigma_1]\ ;\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{\mathsf{val}} \ v_1[\alpha{:=}\sigma_1] \ : \ \sigma[\alpha{:=}\sigma_1] \mid \epsilon \ \rightsquigarrow v'[\alpha{:=}\lfloor\sigma_1\rfloor]$ MVAL

case $e \ = \ e_2\ w\ e_3.$

$\Gamma;\ w;\ w' \Vdash \ e_2\ w\ e_3 \ : \ \sigma_3 \mid \epsilon \ \rightsquigarrow e_2' \rhd (\lambda f.\ e_3' \rhd f\ w')$ given

$\Gamma;\ w;\ w' \Vdash \ e_2 : \sigma_2 \Rightarrow \epsilon\ \sigma_3 \mid \epsilon \ \rightsquigarrow e_2'$ MAPP

$\Gamma;\ w;\ w' \Vdash \ e_3 : \sigma_2 \mid \epsilon \ \rightsquigarrow e_3'$ above

$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash \ e_2[\alpha{:=}\sigma_1] : \sigma_2[\alpha{:=}\sigma_1] \Rightarrow \epsilon\ \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon \ \rightsquigarrow e_2'[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.

$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash \ e_3[\alpha{:=}\sigma_1] : \sigma_2[\alpha{:=}\sigma_1] \mid \epsilon \ \rightsquigarrow e_3'[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.

$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash \ e_2[\alpha{:=}\sigma_1]\ w[\alpha{:=}\sigma_1]\ e_3[\alpha{:=}\sigma_1] \ : \ \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon$ MAPP

$\quad \rightsquigarrow e_2'[\alpha{:=}\lfloor\sigma_1\rfloor] \rhd (\lambda f.\ e_3'[\alpha{:=}\lfloor\sigma_1\rfloor] \rhd f\ w'[\alpha{:=}\lfloor\sigma_1\rfloor])$

case $e \ = \ e_2\ [\sigma_2].$

$\Gamma;\ w;\ w' \Vdash \ e_2\ [\sigma_2] : \ \sigma_1 \mid \epsilon \ \rightsquigarrow e_2' \rhd (\lambda x.\ pure\ (x\ [\lfloor\sigma_2\rfloor]))$ given

$\sigma_1 \ = \ \sigma_3\ [\alpha{:=}\sigma_2]$ MTAPP

$\Gamma;\ w;\ w' \Vdash \ e_2 \ : \ \forall\beta.\ \sigma_3 \mid \epsilon \ \rightsquigarrow e_2'$ above

$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\sigma_1];\ w[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash \ e_2[\alpha{:=}\sigma_1] \ : \ \forall\beta.\ \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon \ \rightsquigarrow e_2'[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.

$(\sigma_3[\alpha{:=}\sigma_1])[\beta{:=}(\sigma_2[\alpha{:=}\sigma_1])\ ] \ = \ (\sigma_3[\beta{:=}\sigma_2])[\alpha{:=}\sigma_1]$ by substitution

$\lfloor\sigma_2[\alpha{:=}\sigma_1]\rfloor \ = \ \lfloor\sigma_2\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]$ Lemma 33

$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\sigma_1];\ w[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash \ e_2[\alpha{:=}\sigma_1]\ [\sigma_2[\alpha{:=}\sigma_1]\ ] \ : \ (\sigma_3[\beta{:=}\sigma_2])[\alpha{:=}\sigma_1] \ \mid \epsilon$ MTAPP

$\quad \rightsquigarrow e_2'[\alpha{:=}\lfloor\sigma_1\rfloor] \rhd (\lambda x.\ pure\ (x\ [\lfloor\sigma_2\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]]))$

case $e \ = \ \mathsf{handle}_m^w\ h\ e_2.$

$\Gamma; w; w' \Vdash \text{handle}_m^w\ h\ e_2 : \sigma \mid \epsilon \rightsquigarrow \textit{prompt}\ m\ w'\ e_2'$ given
$\Gamma \Vdash_{\text{ops}} h : \text{hnd}^{\epsilon}\ \sigma \mid \epsilon \rightsquigarrow h'$ MHANDLE
$\Gamma; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle; \langle\!\langle l : (m, h') \mid w' \rangle\!\rangle \Vdash e_2 : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e_2'$ above
$h \in \Sigma(l)$ above
$\Gamma[\alpha{:=}\sigma_1]; (\langle\!\langle l : (m, h) \mid w \rangle\!\rangle)[\alpha{:=}\sigma_1]; (\langle\!\langle l : (m, h') \mid w' \rangle\!\rangle)[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.
$\qquad \qquad
$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\text{ops}} h[\alpha{:=}\sigma_1] : \sigma[\alpha{:=}\sigma_1] \mid l \mid \epsilon \mid \epsilon \rightsquigarrow h'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 3
$\Gamma[\alpha{:=}\sigma_1]; w[\alpha{:=}\sigma_1]; w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash \text{handle}_m^{w[\alpha:=\sigma_1]}\ h[\alpha{:=}\sigma_1]\ e_2[\alpha{:=}\sigma_1] : \sigma \mid \langle\epsilon\rangle$ MHANDLE
$\quad \rightsquigarrow \textit{prompt}\ m\ w'[\alpha{:=}\lfloor\sigma_1\rfloor]\ e_2'[\alpha{:=}\lfloor\sigma_1\rfloor]$

Part 3

$\Gamma \Vdash_{\text{ops}} \{\ op_1 \to f_1, \ldots, op_n \to f_n\ \} : \sigma \mid l \mid \epsilon \mid \epsilon \rightsquigarrow \{op_1 \to f_1', \ldots, op_n \to f_n'\}$ given
$\Gamma \Vdash_{\text{val}} f_i : \forall \overline{\alpha}.\ \sigma_3 \Rightarrow \epsilon\ (\sigma_2 \Rightarrow \epsilon\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow f_i'$ MOPS
$op_i : \forall \overline{\alpha}.\ \sigma_3 \to \sigma_2 \in \Sigma(l) \quad \overline{\alpha} \not\cap \text{ftv}(\epsilon\ \sigma)$ above
$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\text{val}} f_i[\alpha{:=}\sigma_1] : \forall \overline{\alpha}.\ \sigma_3 \Rightarrow \epsilon\ (\sigma_2 \Rightarrow \epsilon\ \sigma[\alpha{:=}\lfloor\sigma_1\rfloor]) \Rightarrow \epsilon\ \sigma[\alpha{:=}\lfloor\sigma_1\rfloor] \rightsquigarrow f_i'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 1
$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\text{ops}} \{\ op_1 \to f_1[\alpha{:=}\sigma_1], \ldots, op_n \to f_n[\alpha{:=}\sigma_1]\ \} : \sigma[\alpha{:=}\lfloor\sigma_1\rfloor] \mid l \mid \epsilon \mid \epsilon$ MOPS
$\quad \rightsquigarrow \{op_1 \to f_1'[\alpha{:=}\lfloor\sigma_1\rfloor], \ldots, op_n \to f_n'[\alpha{:=}\lfloor\sigma_1\rfloor]\}$

Part 4 By induction on typing.

case $\mathsf{E} = \square$.

$\Gamma[\alpha{:=}\sigma_1]\ ;\ w\ ;\ w' \Vdash_{\text{ec}} \square : \sigma_1 \to \sigma_2 \mid \epsilon \rightsquigarrow id$ given
$\sigma_1 = \sigma_2$ MON-CEMPTY
$\square[\alpha{:=}\sigma_1] = \square$ by substitution
$\Gamma_1, \Gamma_2\ ;\ w[\alpha{:=}\sigma_1]\ ;\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{\text{ec}} \square : \sigma_1[\alpha{:=}\sigma_1] \to \sigma_1[\alpha{:=}\sigma_1] \mid \epsilon \rightsquigarrow id$ MON-CEMPTY

case $\mathsf{E} = \mathsf{E}_1\ w\ e$.

$\Gamma\ ;\ w;\ w' \Vdash_{\text{ec}} \mathsf{E}_1\ w\ e : \sigma \to \sigma_2 \mid \epsilon \rightsquigarrow (\lambda f.\ e' \triangleright f\ w) \bullet g$ given
$\Gamma\ ;\ w;\ w' \Vdash_{\text{ec}} \mathsf{E}_1 : \sigma \to (\sigma_3 \Rightarrow \epsilon\ \sigma_2) \mid \epsilon \rightsquigarrow g$ MON-CAPP1
$\Gamma\ ;\ w;\ w' \Vdash e : \sigma_3 \mid \epsilon \rightsquigarrow e'$ above
$\Gamma[\alpha{:=}\sigma_1]\ ;\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{\text{ec}} \mathsf{E}_1[\alpha{:=}\sigma_1]$ I.H.
$\quad : \sigma[\alpha{:=}\sigma_1] \to (\sigma_3[\alpha{:=}\sigma_1] \Rightarrow \epsilon\ \sigma_2[\alpha{:=}\sigma_1]) \mid \epsilon \rightsquigarrow g[\alpha{:=}\lfloor\sigma_1\rfloor]$
$\Gamma[\alpha{:=}\sigma_1]\ ;\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash e[\alpha{:=}\sigma_1] : \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon \rightsquigarrow e'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 2
$\Gamma[\alpha{:=}\sigma_1]\ ;\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{\text{ec}} \mathsf{E}_1[\alpha{:=}\sigma_1]\ w[\alpha{:=}\sigma_1]\ e[\alpha{:=}\sigma_1]$ MON-CAPP1
$\quad : \sigma[\alpha{:=}\sigma_1] \to \sigma_2[\alpha{:=}\sigma_1] \mid \epsilon \rightsquigarrow (\lambda f.\ e'[\alpha{:=}\lfloor\sigma_1\rfloor] \triangleright f\ w[\alpha{:=}\lfloor\sigma_1\rfloor]) \bullet g[\alpha{:=}\lfloor\sigma_1\rfloor]$

case $\mathsf{E} = v_1\ w\ \mathsf{E}_1$.

$\Gamma;\ w;\ w' \Vdash_{\text{ec}} v_1\ w\ \mathsf{E}_1 : \sigma \to \sigma_2 \mid \epsilon \rightsquigarrow v_1'\ w' \bullet g$ given
$\Gamma;\ w;\ w' \Vdash_{\text{ec}} \mathsf{E}_1 : \sigma \to \sigma_3 \mid \epsilon \rightsquigarrow g$ MON-CAPP2
$\Gamma \Vdash_{\text{val}} v_1 : \sigma_3 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow v_1'$ above
$\Gamma[\alpha{:=}\sigma_1]\ ;\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{\text{ec}} \mathsf{E}_1[\alpha{:=}\sigma_1] : \sigma[\alpha{:=}\sigma_1] \to \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon \rightsquigarrow g[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.
$\Gamma[\alpha{:=}\sigma_1] \Vdash_{\text{val}} v_1[\alpha{:=}\sigma_1] : \sigma_3 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow v_1'[\alpha{:=}\lfloor\sigma_1\rfloor]$ Part 2
$\Gamma[\alpha{:=}\sigma_1]\ ;\ w[\alpha{:=}\sigma_1];\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{\text{ec}} v_1[\alpha{:=}\sigma_1]\ w[\alpha{:=}\sigma_1]\ \mathsf{E}_1[\alpha{:=}\sigma_1]$ MON-CAPP2
$\quad : \sigma[\alpha{:=}\sigma_1] \to \sigma_2[\alpha{:=}\sigma_1] \mid \epsilon \rightsquigarrow v_1'[\alpha{:=}\lfloor\sigma_1\rfloor]\ w'[\alpha{:=}\lfloor\sigma_1\rfloor]$

case $\mathsf{E} = \mathsf{E}_1\ [\sigma]$.

$\Gamma\,;\ w\,;w' \Vdash_{ec}\ \mathsf{E}_1\ [\sigma]\ :\ \sigma \rightarrow \sigma_2 \mid \epsilon\ \rightsquigarrow (\lambda x.\ \mathit{pure}\ (x\ [\lfloor\sigma\rfloor])) \bullet g$ given
$\Gamma\,;\ w\,;w' \Vdash_{ec}\ \mathsf{E}_1\ :\ \sigma \rightarrow \forall\alpha.\ \sigma_3 \mid \epsilon\ \rightsquigarrow g$ MON-CTAPP
$\sigma_2\ =\ \sigma_3[\alpha{:=}\sigma]$ above
$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{ec}\ \mathsf{E}_1[\alpha{:=}\sigma_1]\ :\ \sigma[\alpha{:=}\sigma_1] \rightarrow \forall\alpha.\ \sigma_3[\alpha{:=}\sigma_1] \mid \epsilon\ \rightsquigarrow g[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.
$\lfloor\sigma\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor]\ =\ \lfloor\sigma[\alpha{:=}\sigma_1]\rfloor$ Lemma 33
$\Gamma[\alpha{:=}\sigma_1];\ w[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{ec}\ \mathsf{E}_1[\alpha{:=}\sigma_1]\ [\sigma[\alpha{:=}\sigma_1]\ ]$ MON-CTAPP
$\quad :\ \sigma[\alpha{:=}\sigma_1] \rightarrow \sigma_2[\alpha{:=}\sigma_1] \mid \epsilon\ \rightsquigarrow (\lambda x.\ \mathit{pure}\ (x\ \lfloor\sigma\rfloor[\alpha{:=}\lfloor\sigma_1\rfloor])) \bullet g[\alpha{:=}\lfloor\sigma_1\rfloor]$

case $\mathsf{E}\ =\ \mathsf{handle}^w_m\ h\ \mathsf{E}_1$.

$\Gamma; w; w' \Vdash_{ec}\ \mathsf{handle}^w_m\ h\ \mathsf{E}\ :\ \sigma \rightarrow \sigma_2 \mid \epsilon\ \rightsquigarrow \mathit{prompt}\ m\ w' \circ g$ given
$\Gamma; w; w' \Vdash_{ops}\ \ h\ :\ \sigma_2 \mid l \mid \epsilon\ \rightsquigarrow h'$ above
$\Gamma; \langle\!\langle l : (m, h) \mid w\rangle\!\rangle; \langle\!\langle l : (m, h') \mid w'\rangle\!\rangle \Vdash_{ec}\ \mathsf{E}\ :\ \sigma \rightarrow \sigma_2 \mid \langle l \mid \epsilon\rangle\ \rightsquigarrow g$ above
$\Gamma[\alpha{:=}\sigma_1];\ \langle\!\langle l : (m, h[\alpha{:=}\sigma_1]) \mid w[]\rangle\!\rangle; \langle\!\langle l : (m, h'[\alpha{:=}\lfloor\sigma_1\rfloor]) \mid w'[\alpha{:=}\lfloor\sigma_1\rfloor]\rangle\!\rangle \Vdash_{ec}\ \mathsf{E}[\alpha{:=}\lfloor\sigma_1\rfloor]$ I.H.
$\quad :\ \sigma[\alpha{:=}\sigma_1] \rightarrow \sigma_2[\alpha{:=}\sigma_1] \mid \langle l \mid \epsilon\rangle\ \rightsquigarrow g[\alpha{:=}\lfloor\sigma_1\rfloor]$
$\Gamma[\alpha{:=}\sigma_1] \Vdash_{ops}\ \ h[\alpha{:=}\sigma_1]\ :\ \sigma_2[\alpha{:=}\sigma_1] \mid l \mid \epsilon\ \rightsquigarrow h'[\alpha{:=}\lfloor\sigma_1\rfloor]\ ]$ Part 3
$\Gamma[\alpha{:=}\sigma_1]; w[\alpha{:=}\lfloor\sigma_1\rfloor]; w'[\alpha{:=}\lfloor\sigma_1\rfloor] \Vdash_{ec}\ \mathsf{handle}^{w[\alpha{:=}\sigma_1]}_m\ h[\alpha{:=}\sigma_1]\ \mathsf{E}[\alpha{:=}\sigma_1]\ :\ \sigma_1 \rightarrow \sigma_2 \mid \epsilon$ MON-CHANDLE
$\quad \rightsquigarrow \mathit{prompt}\ m\ w'[\alpha{:=}\lfloor\sigma_1\rfloor] \circ g[\alpha{:=}\lfloor\sigma_1\rfloor]$

□

B.4.4 Evaluation Context Typing.

Lemma 36. (*Monadic contexts*)
If $\Gamma; w \Vdash_{ec}\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma_2 \mid \epsilon\ \rightsquigarrow g$ and $\Gamma; \langle\!\langle \lceil \mathsf{E}\rceil \mid w\rangle\!\rangle \Vdash\ e\ :\ \sigma_1 \mid \langle \lceil \mathsf{E}\rceil^l \mid \epsilon\rangle\ \rightsquigarrow e'$ then $\Gamma; w \Vdash\ \mathsf{E}[e]\ :\ \sigma_2 \mid \epsilon$ (due to Lemma 22) and $\Gamma; w \Vdash\ \mathsf{E}[e]\ :\ \sigma_2 \mid \epsilon\ \rightsquigarrow g\ e'$.

Proof. (*Of Lemma 36*) By induction on the evaluation context typing.

case $\mathsf{E}\ =\ \square$.

$\Gamma; w \Vdash_{ec}\ \square\ :\ \sigma_1 \rightarrow \sigma_1\ \rightsquigarrow \mathit{id}$ given
$\Gamma; w \Vdash\ e\ :\ \sigma_1 \mid \epsilon\ \rightsquigarrow e'$ given
e'
$=\ \mathit{id}\ e'$ *id*

case $\mathsf{E}\ =\ \mathsf{E}_0\ w\ e_0$.

$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0\ w\ e_0\ :\ \sigma_1 \rightarrow \sigma_2 \mid \epsilon\ \rightsquigarrow (\lambda f.\ f\ w \lhd e_0') \bullet g$ given
$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0\ :\ \sigma_1 \rightarrow (\sigma_3 \Rightarrow\ \epsilon\ \sigma_2)\ \mid \epsilon\ \rightsquigarrow g$ above
$\lceil \mathsf{E}_0\ w\ e_0\rceil\ =\ \lceil \mathsf{E}_0\rceil$ by definition
$\lceil \mathsf{E}_0\ w\ e_0\rceil^l\ =\ \lceil \mathsf{E}_0\rceil^l$ by definition
$\Gamma; w \Vdash\ \mathsf{E}_0[e]\ :\ \sigma_3 \Rightarrow\ \epsilon\ \sigma_2\ \mid \epsilon\ \rightsquigarrow g\ e'$ I.H.
$\Gamma; w \Vdash\ \mathsf{E}_0[e]\ w\ e_0\ :\ \sigma_2 \mid \epsilon\ \rightsquigarrow (\lambda f.\ f\ w \lhd e_0') \lhd g\ e'$ MAPP
$(\lambda f.\ f\ w \lhd e_0') \lhd g\ e'$
$=\ ((\lambda f.\ f\ w \lhd e_0') \bullet g)\ e'$ by ($\bullet$)

case $\mathsf{E}\ =\ v\ w\ \mathsf{E}_0$.

$\Gamma; w \Vdash_{ec}\ v\ w\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow (v'\ w) \bullet g$	given
$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow g$	above
$\lceil v\ w\ \mathsf{E}_0 \rceil = \lceil \mathsf{E}_0 \rceil$	by definition
$\lceil v\ w\ \mathsf{E}_0 \rceil^l = \lceil \mathsf{E}_0 \rceil^l$	by definition
$\Gamma; w \Vdash \mathsf{E}_0[e] : \sigma_3 \mid \epsilon \rightsquigarrow g\ e'$	I.H.
$\Gamma; w \Vdash v\ w\ \mathsf{E}_0[e] : \sigma_2 \mid \epsilon \rightsquigarrow (\lambda f.\ f\ w \lhd (g\ e')) \lhd (pure[\lfloor \sigma_3 \Rightarrow \epsilon\ \sigma_2 \rfloor]\ v')$	MAPP
$(\lambda f.\ f\ w \lhd (g\ e')) \lhd (pure[\lfloor \sigma_3 \Rightarrow \epsilon\ \sigma_2 \rfloor]\ v')$	
$= (\lambda f.\ f\ w \lhd (g\ e'))\ v'$	$(\lhd)$
$= v'\ w \lhd g\ e'$	reduce
$= (v'\ w \bullet g)\ e'$	$(\bullet)$

case $\mathsf{E} = \mathsf{E}_0\ [\sigma]$.

$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0\ [\sigma] : \sigma_1 \rightarrow \sigma_2[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow (\lambda x.\ pure\ (x[\lfloor \sigma \rfloor])) \bullet g$	given
$\Gamma; w \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow \forall \alpha.\ \sigma_2 \mid \epsilon \rightsquigarrow g$	above
$\lceil \mathsf{E}_0\ [\sigma] \rceil = \lceil \mathsf{E}_0 \rceil$	by definition
$\lceil \mathsf{E}_0\ [\sigma] \rceil^l = \lceil \mathsf{E}_0 \rceil^l$	by definition
$\Gamma; w \Vdash \mathsf{E}_0[e] : \forall \alpha.\ \sigma_2 \mid \epsilon \rightsquigarrow g\ e'$	I.H.
$\Gamma; w \Vdash \mathsf{E}_0[e]\ [\sigma] : \sigma_2[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow (\lambda x.\ pure\ (x[\lfloor \sigma \rfloor])) \lhd (g\ e')$	MTAPP
$(\lambda x.\ pure\ (x[\lfloor \sigma \rfloor])) \lhd g\ e'$	
$= (\lambda x.\ pure\ (x[\lfloor \sigma \rfloor]) \bullet g)\ e'$	of $(\bullet)$

case $\mathsf{E} = \mathsf{handle}^w_m\ h\ \mathsf{E}_0$.

$\Gamma; w \Vdash_{ec}\ \mathsf{handle}^w_m\ h\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow prompt[\epsilon, \lfloor \sigma \rfloor]\ m\ w \circ g$	given
$\Gamma; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash_{ec}\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \langle l \mid \epsilon \rangle \rightsquigarrow g$	above
$\langle\!\langle \lceil \mathsf{handle}^w_m\ h\ \mathsf{E}_0 \rceil \mid w \rangle\!\rangle = \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \rangle\!\rangle$	by definition
$\langle \lceil \mathsf{handle}^w_m\ h\ \mathsf{E}_0 \rceil^l \mid \epsilon \rangle = \langle \lceil \mathsf{E}_0 \rceil^l \mid \langle l \mid \epsilon \rangle \rangle$	by definition
$\Gamma; \langle\!\langle \lceil \mathsf{E} \rceil \mid w \rangle\!\rangle \Vdash e : \sigma_1 \mid \langle \lceil \mathsf{E} \rceil^l \mid \epsilon \rangle \rightsquigarrow e'$	given
$\Gamma; \langle\!\langle \lceil \mathsf{E}_0 \rceil \mid \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \rangle\!\rangle \Vdash e : \sigma_1 \mid \langle \lceil \mathsf{E}_0 \rceil^l \mid \langle l \mid \epsilon \rangle \rangle \rightsquigarrow e'$	by substitution
$\Gamma; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle \Vdash \mathsf{E}_0[e] : \sigma_2 \mid \langle l \mid \epsilon \rangle \rightsquigarrow g\ e'$	I.H.
$\Gamma; w \Vdash \mathsf{handle}^w_m\ h\ (\mathsf{E}_0[e]) : \sigma_2 \mid \epsilon \rightsquigarrow prompt[\epsilon, \lfloor \sigma \rfloor]\ m\ w\ (g\ e')$	MHANDLE
$prompt[\epsilon, \lfloor \sigma \rfloor]\ m\ w\ (g\ e')$	
$= (prompt[\epsilon, \lfloor \sigma \rfloor]\ m\ w \circ g)\ e'$	$(\circ)$

□

Definition 3.
Define a certain form of expression r, as $r ::= id \mid e \bullet r \mid prompt\ m\ w \circ r$.

$$\begin{aligned} \mathrm{bm}(id) &= \varnothing \\ \mathrm{bm}(e \bullet r) &= \mathrm{bm}(r) \\ \mathrm{bm}(prompt\ m\ w \circ r) &= \mathrm{bm}(r) \cup \{ m \} \end{aligned}$$

Lemma 37.
If $\Gamma; w \Vdash_{ec}\ \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow r$.

Proof. (*Of Lemma 37*) By straightforward induction on the evaluation context typing. □

Lemma 38. (($\bullet$) *associates with* ($\circ$))
1. $e_1 \bullet (e_2 \circ e_3) = (e_1 \bullet e_2) \circ e_3$.

Proof. (*Of Lemma 38*)

$(e_1 \bullet (e_2 \circ e_3))\ x$
$= \ e_1 \lhd ((e_2 \circ e_3)\ x)$ definition of $(\bullet)$
$= \ e_1 \lhd (e_2\ (e_3\ x))$ definition of $(\circ)$
$((e_1 \bullet e_2) \circ e_3)\ x$
$= \ (e_1 \bullet e_2)\ (e_3\ x)$ definition of $(\circ)$
$= \ e_1 \lhd (e_2\ (e_3\ x))$ definition of $(\bullet)$
□

Lemma 39. (($\circ$) *properties*)
1. $e \circ id \ = \ e.$
2. $id \circ e \ = \ e.$

Lemma 40. (*Yield hoisting*)
If $m \notin \mathrm{bm}(r)$, then $r\ (\mathsf{yield}\ m\ f\ cont) \ = \ \mathsf{yield}\ m\ f\ (r \circ cont)$.

Proof. (*Of Lemma 40*) By induction on r.
case $r \ = \ id.$
$id\ (\mathsf{yield}\ m\ f\ cont)$
$= \ \mathsf{yield}\ m\ f\ cont$ by id
$= \ \mathsf{yield}\ m\ f\ (id \circ cont)$ Lemma 39.2
case $r \ = \ e \bullet r_0.$
$(e \bullet r_0)\ (\mathsf{yield}\ m\ f\ cont)$
$= \ e \lhd (r_0\ (\mathsf{yield}\ m\ f\ cont))$ definition of $(\bullet)$
$= \ e \lhd (\mathsf{yield}\ m\ f\ (r_0 \circ cont))$ I.H.
$= \ \mathsf{yield}\ m\ f\ (e \bullet (r_0 \circ cont))$ definition of $(\lhd)$
$= \ \mathsf{yield}\ m\ f\ ((e \bullet r_0) \circ cont)$ Lemma 38
case $r \ = \ prompt\ m_1\ w \circ r_0.$
$(prompt\ m_1\ w \circ r_0)\ (\mathsf{yield}\ m\ f\ cont)$
$= \ prompt\ m_1\ w\ (r_0\ (\mathsf{yield}\ m\ f\ cont))$ definition of $(\circ)$
$= \ prompt\ m_1\ w\ (\mathsf{yield}\ m\ f\ (r_0 \circ cont))$ I.H.
$(m \notin \mathrm{bop}(\ prompt\ m_1\ w \circ r_0))$ given
$(m \ \neq \ m_1\ ))$ follows
$= \ \mathsf{yield}\ m\ f\ (prompt\ m_1\ w \circ (r_0 \circ cont))$ definition of *prompt*
$= \ \mathsf{yield}\ m\ f\ ((prompt\ m_1\ w \circ r_0) \circ cont)$ $(\circ)$ is associative
□

Lemma 41.
If $m \notin \lceil \mathsf{E} \rceil^m$, and $\Gamma;\ w\ \Vdash\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma_2\ \rightsquigarrow r$, then $m \notin \mathrm{bm}(r)$.

Proof. (*Of Lemma 41*) By a straightforward induction on the evaluation context translation. The only interesting case is MON-CHANDLE,

$\Gamma\ ;\ w\ \Vdash\ \mathsf{hangle}^w_{m_1}\ h\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma \mid \epsilon\ \rightsquigarrow prompt[\epsilon, \sigma]\ m\ w \circ r$ given
$\Gamma\ ;\ \langle\!\langle l : (m, h') \mid w \rangle\!\rangle\ \Vdash\ \mathsf{E}\ :\ \sigma_1 \rightarrow \sigma \mid \langle l \mid \epsilon \rangle\ \rightsquigarrow r$ MON-CHANDLE
$m \notin \mathrm{bm}(r)$ I.H.
$m \notin \lceil \mathsf{handle}^w_{m_1}\ h\ \mathsf{E} \rceil^m$ given
$m \ \neq \ m_1$ Follows
$m \notin \mathrm{bm}(prompt[\epsilon, \sigma]\ m\ w \circ r)$ Follows
□

B.4.5 Translation Coherence.

Proof. (*Of Theorem 11*) Apply Lemma 42, with $w = w' = \langle\!\langle\rangle\!\rangle$. $\square$

Lemma 42. (*Coherence of the Monadic Translation*)
If $\varnothing; w; w' \Vdash e_1 : \sigma \mid \langle\rangle \rightsquigarrow e_1'$ and $e_1 \longrightarrow e_2$, then also $\varnothing; w; w' \Vdash e_2 : \sigma \mid \langle\rangle \rightsquigarrow e_2'$ where $e_1' \longrightarrow^* e_2'$.

Proof. (*Of Theorem 42*) Induction on the operational rules.

case $(\lambda^{\epsilon}\, z{:}\mathsf{evv}\ \epsilon,\ x{:}\sigma.\ e)\ w\ v \longrightarrow e[z{:=}w,\ x{:=}v]$.

$\varnothing;\ w;\ w' \Vdash (\lambda^{\epsilon}\, z{:}\mathsf{evv}\ \epsilon,\ x{:}\sigma.\ e)\ w\ v : \sigma \mid \epsilon \rightsquigarrow (\lambda f.\ f\ w' \lhd \mathit{pure}\ v') \lhd (\mathit{pure}\ (\lambda z\ x.\ e'))$	given
$(\lambda f.\ f\ w' \lhd \mathit{pure}\ v') \lhd (\mathit{pure}\ (\lambda z\ x.\ e'))$	
$\longrightarrow (\lambda f.\ f\ w' \lhd \mathit{pure}\ v')\ (\lambda z\ x.\ e')$	$(\lhd)$
$\longrightarrow (\lambda z\ x.\ e')\ w' \lhd \mathit{pure}\ v'$	reduces
$\longrightarrow (\lambda z\ x.\ e')\ w'\ v'$	$(\lhd)$
$\longrightarrow e'[z{:=}w',\ x{:=}v']$	$(\lhd)$
$z{:}\mathsf{evv}\ \epsilon,\ x{:}\sigma;\ w;\ w' \Vdash e : \sigma_1 \Rightarrow \epsilon\ \sigma \mid \epsilon \rightsquigarrow e'$	MAPP,MABS
$\varnothing; w; w' \Vdash e[z{:=}w,\ x{:=}v] : \sigma \mid \epsilon \rightsquigarrow e'[z{:=}w', x{:=}v']$	Lemma 34

case $(\Lambda\alpha^{k}.\ v)\ [\sigma] \longrightarrow v[\alpha{:=}\sigma]$.

$\varnothing;\ w;\ w' \Vdash (\Lambda\alpha^{k}.\ v)\ [\sigma] : \sigma_2[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow (\lambda x.\ \mathit{pure}\ (x\ [\lfloor\sigma\rfloor])) \lhd \mathit{pure}\ (\Lambda\alpha.\ v')$	given
$(\lambda x.\ \mathit{pure}\ (x\ [\lfloor\sigma\rfloor])) \lhd \mathit{pure}\ (\Lambda\alpha.\ v')$	
$\longmapsto (\lambda x.\ \mathit{pure}\ (x\ [\lfloor\sigma\rfloor]))\ (\Lambda\alpha.\ v')$	$(\lhd)$
$\longrightarrow (\mathit{pure}\ ((\Lambda\alpha.\ v')\ [\lfloor\sigma\rfloor]))$	(app)
$\longmapsto \mathit{pure}\ (v'[\alpha{:=}\sigma])$	
$\varnothing; w; w' \Vdash v[\alpha{:=}\sigma] : \sigma_2[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow \mathit{pure}\ (v'[\alpha{:=}\sigma])$	Lemma 35

case $(\mathsf{handler}^{\epsilon}\ h)\ w\ v \longrightarrow \mathsf{handle}_m^{w}\ h\ (v\ \langle\!\langle l{:}(m,h) \mid w\rangle\!\rangle\ ())$ with m unique.

$\varnothing; w; w' \Vdash (\mathsf{handler}^{\epsilon}\ h)\ w\ v : \sigma \mid \epsilon$	given
$\rightsquigarrow (\lambda f.\ f\ w' \lhd \mathit{pure}\ v) \lhd \mathit{pure}\ (\mathit{handler}^{l}\ [\epsilon,\sigma]\ h')$	
$(\lambda f.\ f\ w' \lhd \mathit{pure}\ v') \lhd \mathit{pure}\ (\mathit{handler}^{l}\ [\epsilon,\sigma]\ h')$	
$\longrightarrow (\lambda f.\ f\ w' \lhd \mathit{pure}\ v')\ (\mathit{handler}^{l}\ [\epsilon,\sigma]\ h')$	$(\lhd)$
$\longrightarrow (\mathit{handler}^{l}\ [\epsilon,\sigma]\ h')\ w' \lhd \mathit{pure}\ v'$	reduces
$\longrightarrow (\mathit{handler}^{l}\ [\epsilon,\sigma]\ h')\ w'\ v'$	$(\lhd)$
$\longrightarrow \mathit{freshm}\ (\lambda m.\ \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (v'\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle\ ()))$	*handler*
$\longrightarrow \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (v'\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle\ ()$	given m unique
$\varnothing; w; w' \Vdash \mathsf{handle}_m^{w}\ h\ (v\ \langle\!\langle l{:}(m,h) \mid w\rangle\!\rangle\ ()) : \sigma \mid \epsilon$	given
$\rightsquigarrow \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (\ (\lambda f.\ f\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle \lhd \mathit{pure}\ ()\ ) \lhd \mathit{pure}\ v')$	
$\mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (\ (\lambda f.\ f\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle \lhd \mathit{pure}\ ()\ ) \lhd \mathit{pure}\ v')$	
$\longmapsto \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (\ (\lambda f.\ f\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle \lhd \mathit{pure}\ ()\ )\ v')$	$(\lhd)$
$\longmapsto \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (v'\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle \lhd \mathit{pure}\ ()\ )$	reduces
$\longmapsto \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (v'\ \langle\!\langle l{:}(m,h) \mid w'\rangle\!\rangle\ ()\ )$	$(\lhd)$

case $\mathsf{handle}_m^{w}\ h \cdot v \longrightarrow v$.

$\varnothing;\ w;\ w' \Vdash \mathsf{handle}_m^{w}\ h \cdot v : \sigma \mid \epsilon \rightsquigarrow \mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (\mathit{pure}\ v')$	given
$\mathit{prompt}[\epsilon,\sigma]\ m\ w'\ (\mathit{pure}\ v')$	
$\longrightarrow \mathit{pure}\ v'$	*prompt*

case $\mathsf{handle}_m^{w}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w_1\ v \longrightarrow f\ w\ v\ w\ k$ with $(op \to f) \in h$, $op \notin \mathrm{bop}(\mathsf{E})$, and $k = \mathsf{guard}^{w}\ (\mathsf{handle}_m^{w}\ h \cdot \mathsf{E})$.
From the assumption:
$\varnothing; w; w' \Vdash \mathsf{handle}_m^{w}\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w_1\ v \rightsquigarrow e_1'$ and

$\varnothing; w; w' \Vdash\ f\ w\ v\ w\ k \ \rightsquigarrow (\lambda f_0.\ f_0\ w' \lhd (pure\ k')) \lhd ((\lambda f_1.\ f_1\ w' \lhd (pure\ v')) \lhd (pure\ f'))$
with $k' = guard\ w'\ (prompt\ m\ w' \circ g)$ where $\varnothing; w; w' \Vdash_{ec}\ \mathsf{E} \ \rightsquigarrow g$.

We can simplify the translation of $f\ w\ v\ w\ k$ as:
$(\lambda f_0.\ f_0\ w' \lhd (pure\ k')) \lhd ((\lambda f_1.\ f_1\ w' \lhd (pure\ v')) \lhd (pure\ f'))$
$\longmapsto (\lambda f_0.\ f_0\ w' \lhd (pure\ k')) \lhd ((\lambda f_1.\ f_1\ w' \lhd (pure\ v')) \lhd \mathsf{pure}\ f')$
$\longmapsto (\lambda f_0.\ f_0\ w' \lhd (pure\ k')) \lhd (f'\ w' \lhd (pure\ v'))$
$\longmapsto (\lambda f_0.\ f_0\ w' \lhd (pure\ k')) \lhd (f'\ w'\ v')$

$\varnothing; w; w' \Vdash\ \mathsf{handle}^w_m\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w_1\ v \ \rightsquigarrow e'_1$	given
e'_1	
$=\ \mathsf{prompt}\ m\ w'\ (g\ ((\lambda f.\ f\ w'_1 \lhd pure\ v') \lhd pure\ (perform^{op}[\epsilon, \overline{\sigma}])))$	Lemma 36
$\longrightarrow \mathsf{prompt}\ m\ w'\ (g\ ((\lambda f.\ f\ w'_1 \lhd pure\ v')\ (perform^{op}[\epsilon, \overline{\sigma}])))$	$(\lhd)$
$\longrightarrow \mathsf{prompt}\ m\ w'\ (g\ (perform^{op}[\epsilon, \overline{\sigma}]\ w'_1 \lhd pure\ v'\ ))$	reduces
$\longrightarrow \mathsf{prompt}\ m\ w'\ (g\ (perform^{op}[\epsilon, \overline{\sigma}]\ w'_1\ v'\ ))$	$(\lhd)$
$\longrightarrow \mathsf{prompt}\ m\ w'\ (g\ (let\ (m,\ h)\ =\ w'_1.l\ in$	*perform*
$\quad yield\ m\ (\lambda w\ k.\ (\lambda f_0.\ f_0\ w\ k) \lhd (h.op)\ w\ v')\ ))$	
$(w'_1.l\ =\ (m,\ h))$	Theorem 5
$\longrightarrow \mathsf{prompt}\ m\ w'\ (g\ (yield\ m\ (\lambda w\ k.\ (\lambda f_0.\ f_0\ w\ k) \lhd (h.op)\ w\ v')\ ))$	
$\longrightarrow \mathsf{prompt}\ m\ w'\ (g\ (yield\ m\ (\lambda w\ k.\ (\lambda f_0.\ f_0\ w\ k) \lhd f'\ w\ v')\ ))$	
let $f'' = (\lambda w\ k.\ (\lambda f_0.\ f_0\ w\ k) \lhd f'\ w\ v')$	
$\longrightarrow prompt\ m\ w'\ (g\ (\mathsf{yield}\ m\ f''\ id))$	*yield*
(g is of form r)	Lemma 37
$(op \notin \mathsf{bop}(\mathsf{E}))$	given
$(op \rightarrow f)\ \in h$	given
$(h \notin \mathsf{bh}(\mathsf{E}))$	otherwise $op\ \in \mathsf{bop}(\mathsf{E})$
($\mathsf{handle}^w_m\ h \cdot \mathsf{E} \cdot \mathsf{perform}\ op\ \overline{\sigma}\ w_1\ v$ is m-mapping)	Lemma 32
$(m \notin \lceil \mathsf{E} \rceil^m)$	otherwise $h\ \in \mathsf{bh}(\mathsf{E})$
$(m \notin \mathsf{bm}(g))$	Lemma 41
$\longmapsto^* prompt\ m\ w'\ (\mathsf{yield}\ m\ f''\ (g \circ id))$	Lemma 40
$=\ prompt\ m\ w'\ (\mathsf{yield}\ m\ f''\ g)$	Lemma 39.1
$\longrightarrow f''\ w'\ (guard\ w'\ (prompt\ m\ w' \circ g))$	*prompt*
$=\ f''\ w'\ k'$	
$=\ (\lambda w\ k.\ (\lambda f_0.\ f_0\ w\ k) \lhd f'\ w\ v')\ w'\ k'$	
$\longrightarrow (\lambda f_0.\ f_0\ w'\ k') \lhd f'\ w'\ v'$	(app)
$=_\beta\ (\lambda f_0.\ f_0\ w' \lhd pure\ k') \lhd f'\ w'\ v'$	
$=_\beta\ (\lambda f_0.\ f_0\ w' \lhd pure\ k') \lhd (f'\ w' \lhd (pure\ v'))$	
$=_\beta\ (\lambda f_0.\ f_0\ w' \lhd pure\ k') \lhd ((\lambda f_1.\ f_1\ w' \lhd (pure\ v')) \lhd pure\ f')$	

case $(\mathsf{guard}^w\ \mathsf{E}\ \sigma)\ w\ v \longrightarrow \mathsf{E}[v]$.

$\varnothing; w; w' \Vdash (\mathsf{guard}^w\ \mathsf{E}\ \sigma_1)\ w\ v : \sigma_2 \mid \epsilon \rightsquigarrow (\lambda f.\ f\ w' \lhd (pure\ v')) \lhd (pure\ (guard\ w'\ g))$	given
$\varnothing; w; w' \Vdash (\mathsf{guard}^w\ \mathsf{E}\ \sigma_1) : \sigma_1 \Rightarrow \epsilon\ \sigma_2 \mid \epsilon \rightsquigarrow (pure\ (guard\ w'\ g))$	(*mapp*)
$\varnothing; w; w' \Vdash v : \sigma_1 \mid \epsilon \rightsquigarrow (pure\ v')$	above
$\varnothing; w; w' \Vdash \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow g'$	MGUARD
$(\lambda f.\ f\ w' \lhd (pure\ v')) \lhd (pure\ (guard\ w'\ g))$	
$\longmapsto (\lambda f.\ f\ w' \lhd (pure\ v'))\ (guard\ w'\ g)$	$(\lhd)$
$\longmapsto guard\ w'\ g\ w' \lhd (pure\ v')$	(*app*)
$\longmapsto guard\ w'\ g\ w'\ v'$	$(\lhd)$
$\longmapsto if\ (w' == w')\ then\ g\ (pure\ v')\ else\ wrong$	*guard*
$\longmapsto g\ (pure\ v')$	$w' == w'$
$\varnothing; w; w' \Vdash \mathsf{E}[v] : \sigma_2 \mid \epsilon \rightsquigarrow g'\ (pure\ v')$	Lemma 36

□

B.4.6 Translation Soundness.

Proof. (*Of Theorem 11*) Applying Lemma 43 with $w = \langle\rangle$ and $w' = \langle\!\langle\rangle\!\rangle$. □

Lemma 43. (*Monadic Translation is Sound*)
1. If $\Gamma; w; w' \Vdash e : \sigma \mid \epsilon \rightsquigarrow e'$, then $\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} e' : \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$.
2. If $\Gamma \Vdash_{\mathsf{val}} v : \sigma \rightsquigarrow v'$, then $\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} v' : \lfloor\sigma\rfloor$.
3. If $\Gamma \Vdash_{\mathsf{ops}} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$, then $h' : \mathsf{hnd}^l\ \epsilon\ \lfloor\sigma\rfloor$.
4. If $\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow e$, then $\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} e : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$.

Proof. (*Of Theorem 43*) **Part 1** By induction on the translation.

case $e = v$.

$\Gamma; w; w' \Vdash v : \sigma \mid \epsilon \rightsquigarrow pure\ [\lfloor\sigma\rfloor]\ v'$	given
$\Gamma \Vdash_{\mathsf{val}} v : \sigma \mid \epsilon \rightsquigarrow v'$	MVAL
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} v' : \lfloor\sigma\rfloor$	Part 2
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} pure\ [\lfloor\sigma\rfloor]\ v' : \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$	*pure*, FTAPP and FAPP

case $e = e\ [\sigma]$.

$\Gamma; w; w' \Vdash e[\sigma] : \sigma_1[\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow e' \rhd (\lambda x.\ pure\ (x[\lfloor\sigma\rfloor]))$	given
$\Gamma; w; w' \Vdash e : \forall\alpha.\ \sigma_1 \mid \epsilon \rightsquigarrow e'$	MTAPP
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} e' : \mathsf{mon}\ \epsilon\ (\forall\alpha.\ \lfloor\sigma_1\rfloor)$	I.H.
$\lfloor\Gamma\rfloor,\ x : \forall\alpha.\ \lfloor\sigma_1\rfloor \vdash pure\ (x[\lfloor\sigma\rfloor]) : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor[\alpha{:=}\lfloor\sigma\rfloor]$	*pure*, FTAPP and FAPP
$\lfloor\Gamma\rfloor,\ x : \forall\alpha.\ \lfloor\sigma_1\rfloor \vdash pure\ (x[\lfloor\sigma\rfloor]) : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1[\alpha{:=}\sigma]\rfloor$	Lemma 33
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} \lambda x.\ pure\ (x[\lfloor\sigma\rfloor]) : (\forall\alpha.\ \lfloor\sigma_1\rfloor) \rightarrow \mathsf{mon}\ \lfloor\sigma_1[\alpha{:=}\sigma]\rfloor$	FABS
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} e' \rhd (\lambda x.\ pure\ (x[\lfloor\sigma\rfloor])) : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1[\alpha{:=}\sigma]\rfloor$	$\rhd$

case $e = e_1\ e_2$.

$\Gamma; w; w' \Vdash e_1\ w\ e_2 : \sigma \mid \epsilon \rightsquigarrow e_1' \triangleright (\lambda f.\ e_2' \triangleright f\ w')$ given
$\Gamma; w; w' \Vdash e_1 : \sigma_2 \Rightarrow \epsilon\ \sigma \mid \epsilon \rightsquigarrow e_1'$ MAPP
$\Gamma; w; w' \Vdash e_2 : \sigma_2 \mid \epsilon \rightsquigarrow e_2'$ above
$\lfloor\Gamma\rfloor \vdash_F e_1' : \mathsf{mon}\ \epsilon\ (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor)$ I.H
$\lfloor\Gamma\rfloor \vdash_F e_2' : \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$ I.H
$\lfloor\Gamma\rfloor, f : (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor) \vdash_F f : \mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$ FVAR
$\lfloor\Gamma\rfloor \vdash_F w' : \mathsf{evv}\ \epsilon$ given
$\lfloor\Gamma\rfloor, f : (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor) \vdash_F w' : \mathsf{evv}\ \epsilon$ weakening
$\lfloor\Gamma\rfloor, f : (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor) \vdash_F f\ w' : \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$ FAPP
$\lfloor\Gamma\rfloor, f : (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor) \vdash_F e_2' : \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$ weakening
$\lfloor\Gamma\rfloor, f : (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor) \vdash_F e_2' \triangleright f\ w' : \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$ $\triangleright$
$\lfloor\Gamma\rfloor \vdash_F (\lambda f.\ e_2' \triangleright f\ w') : (\mathsf{evv}\ \epsilon \to \lfloor\sigma_2\rfloor \to \mathsf{mon}\ \lfloor\sigma\rfloor) \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$ FABS
$\lfloor\Gamma\rfloor \vdash_F e_1' \triangleright (\lambda f.\ e_2' \triangleright f\ w') : \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor$ $\triangleright$

case $e = \mathsf{handle}_m^w\ h\ e_0$.

$\Gamma; w; w' \Vdash \mathsf{handle}_m^w\ h\ e_0 : \sigma \mid \epsilon \rightsquigarrow \mathit{prompt}\ [\epsilon, \lfloor\sigma\rfloor]\ m\ w'\ e'$ given
$\Gamma \Vdash_{ops} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ MHANDLE
$\Gamma; \langle\!\langle l : (m, h) \mid w \rangle\!\rangle; \langle\!\langle l : (m, h') \mid w' \rangle\!\rangle \Vdash e : \sigma \mid \langle l \mid \epsilon \rangle \rightsquigarrow e'$ above
$\lfloor\Gamma\rfloor \vdash_F h' : \mathsf{hnd}^l\ \epsilon\ \lfloor\sigma\rfloor$ Part 3
$\lfloor\Gamma\rfloor \vdash_F' e' : \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \sigma$ I.H.
$\lfloor\Gamma\rfloor \vdash_F \mathit{prompt}\ [\epsilon, \lfloor\sigma\rfloor]\ m\ w'\ e' : \mathsf{mon}\ \epsilon\ \sigma$ *prompt*, FTAPP and FAPP

Part 2

By induction on the translation. **case** $v = x$.

$\Gamma \Vdash_{val} x : \sigma \rightsquigarrow x$ given
$x : \sigma \in \Gamma$ MVAR
$x : \lfloor\sigma\rfloor \in \lfloor\Gamma\rfloor$ follows
$\lfloor\Gamma\rfloor \vdash_F x : \lfloor\sigma\rfloor$ FVAR

case $v = \lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \sigma.\ e$.

$\Gamma \Vdash_{val} \lambda^\epsilon\ z : \mathsf{evv}\ \epsilon,\ x : \sigma_1.e : \sigma_1 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow \lambda z\ x.\ e'$ given
$\Gamma, z : \mathsf{evv}\ \epsilon, x : \sigma_1; z; z \Vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e'$ MABS
$\lfloor\Gamma\rfloor, z : \mathsf{evv}\ \epsilon, x : \lfloor\sigma_1\rfloor \vdash_F e' : \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$ Part 1
$\lfloor\Gamma\rfloor \vdash_F \lambda z\ x.\ e' : \mathsf{evv}\ \epsilon \to \lfloor\sigma_1\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$ FABS

case $v = \Lambda\alpha^k.\ v_0$.

$\Gamma \Vdash_{val} \Lambda\alpha.\ v : \forall\alpha.\ \sigma \rightsquigarrow \Lambda\alpha.\ v'$ given
$\Gamma \Vdash_{val} v : \sigma \rightsquigarrow v'$ MTABS
$\lfloor\Gamma\rfloor \vdash_F v' : \lfloor\sigma\rfloor$ I.H.
$\lfloor\Gamma\rfloor \vdash_F \Lambda\alpha.\ v' : \forall\alpha.\ \lfloor\sigma\rfloor$ FTABS

case $v = \mathsf{handler}^\epsilon\ h$.

$\Gamma \Vdash_{val} \mathsf{handler}^\epsilon\ h : (() \Rightarrow \langle l \mid \epsilon \rangle\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow \mathit{handler}^l\ [\epsilon, \lfloor\sigma\rfloor]\ h'$ given
$\Gamma \Vdash_{ops} h : \sigma \mid l \mid \epsilon \rightsquigarrow h'$ MHANDLE
$\lfloor\Gamma\rfloor \vdash_F h' : \mathsf{hnd}^l\ \epsilon\ \lfloor\sigma\rfloor$ Part 3
$\lfloor\Gamma\rfloor \vdash_F \mathit{handler}^l\ [\epsilon, \lfloor\sigma\rfloor]\ h' : \mathsf{evv}\ \epsilon \to (\mathsf{evv}\ \langle l \mid \epsilon \rangle \to () \to \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \sigma) \to \mathsf{mon}\ \epsilon\ \sigma$ *handler*, FTAPP. FAPP

case $v = \mathsf{perform}^\epsilon\ op\ \overline{\sigma}$.

$\Gamma \Vdash_{val} \mathsf{perform}^\epsilon\ op\ \overline{\sigma} : \sigma_1[\overline{\alpha}{:=}\overline{\sigma}] \Rightarrow \langle l \mid \epsilon \rangle\ \sigma_2[\overline{\alpha}{:=}\overline{\sigma}] \rightsquigarrow \mathit{perform}^{op}\ [\langle l \mid \epsilon \rangle, \lfloor\overline{\sigma}\rfloor]$ given
$op : \forall\overline{\alpha}.\ \sigma_1 \to \sigma_2 \in \Sigma(l)$ MPERFORM
$\lfloor\Gamma\rfloor \vdash_F \mathit{perform}^{op}\ [\langle l \mid \epsilon \rangle, \lfloor\overline{\sigma}\rfloor] : \mathsf{evv}\ \langle l \mid \epsilon \rangle \to \lfloor\sigma_1\rfloor[\overline{\alpha}{:=}\lfloor\overline{\sigma}\rfloor] \to \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \lfloor\sigma_2\rfloor[\overline{\alpha}{:=}\lfloor\overline{\sigma}\rfloor]$ *perform*, FTAPP
$\lfloor\Gamma\rfloor \vdash_F \mathit{perform}^{op}\ [\langle l \mid \epsilon \rangle, \lfloor\overline{\sigma}\rfloor] : \mathsf{evv}\ \langle l \mid \epsilon \rangle \to \lfloor\sigma_1[\overline{\alpha}{:=}\overline{\sigma}]\rfloor \to \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \lfloor\sigma_2[\overline{\alpha}{:=}\overline{\sigma}]\rfloor$ Lemma 33

case $v = \mathsf{guard}^w\ \mathsf{E}\ \sigma$.

$\Gamma \Vdash_{\mathsf{val}} \mathsf{guard}^w\ \mathsf{E}\ \sigma_1 : \sigma_1 \Rightarrow \epsilon\ \sigma_2 \rightsquigarrow \mathit{guard}\ w'\ e'$	given
$\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E} : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow e'$	MGUARD
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} e' : \mathsf{mon}\ \epsilon\ \sigma_1 \rightarrow \mathsf{mon}\ \epsilon\ \sigma_2$	Part 4
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} \mathit{guard}\ w'\ e' : \mathsf{evv}\ \epsilon \rightarrow \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$	*guard*, FAPP

Part 3

$\Gamma \Vdash_{\mathsf{ops}} \{op_1 \rightarrow f_1, \ldots, op_n \rightarrow f_n\} : \sigma \mid l \mid \epsilon \rightsquigarrow op_1 \rightarrow f_1', \ldots, op_n \rightarrow f_n'$	given
$\Gamma \Vdash_{\mathsf{val}} f_i : \forall\overline{\alpha}.\ \sigma_1 \Rightarrow \epsilon\ (\sigma_2 \Rightarrow \epsilon\ \sigma) \Rightarrow \epsilon\ \sigma \rightsquigarrow f_i'$	MOPS
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} f_i' : \forall\overline{\alpha}.\ \mathsf{evv}\ \epsilon \rightarrow \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ (\mathsf{evv}\ \epsilon \rightarrow (\mathsf{evv}\ \epsilon \rightarrow \lfloor\sigma_2\rfloor \rightarrow \mathsf{mon}\ \lfloor\sigma\rfloor) \rightarrow \mathsf{mon}\ \lfloor\sigma\rfloor)$	Part 2
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} f_i' : \forall\overline{\alpha}.\ \mathsf{op}\ \lfloor\sigma_1\rfloor\ \lfloor\sigma_2\rfloor\ \epsilon\ \lfloor\sigma\rfloor$	*op*
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} op_1 \rightarrow f_1', \ldots, op_n \rightarrow f_n' : \{op_1 : \forall\overline{\alpha}.\ \mathsf{op}\ \lfloor\sigma_1\rfloor\ \lfloor\sigma_2\rfloor\ \epsilon\ \lfloor\sigma\rfloor, \ldots, op_n : \forall\overline{\alpha}.\ \mathsf{op}\ \lfloor\sigma_1\rfloor\ \lfloor\sigma_2\rfloor\ \epsilon\ \lfloor\sigma\rfloor\}$	follows
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} op_1 \rightarrow f_1', \ldots, op_n \rightarrow f_n' : \mathsf{hnd}^l\ \epsilon\ \lfloor\sigma\rfloor$	follows

Part 4

By induction on the translation.

case $\mathsf{E} = \square$.

$\Gamma; w; w' \Vdash_{\mathsf{ec}} \square : \sigma \rightarrow \sigma \mid \epsilon \rightsquigarrow \mathit{id}$	given
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} \mathit{id} : \lfloor\sigma\rfloor \rightarrow \lfloor\sigma\rfloor$	*id*

case $\mathsf{E} = \mathsf{E}_0\ w\ e$.

$\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_0\ w\ e : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow (\lambda f.\ e' \triangleright f\ w) \bullet g$	given
$\Gamma; w; w' \Vdash e : \sigma_2 \mid \epsilon \rightsquigarrow e'$	MON-CAPP1
$\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_0 : \sigma_1 \rightarrow (\sigma_2 \Rightarrow \epsilon\ \sigma_3) \mid \epsilon \rightsquigarrow g$	above
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} e' : \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$	Part 1
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} g : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ (\mathsf{evv}\ \epsilon \rightarrow \lfloor\sigma_2\rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_3\rfloor)$	I.H.
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} \lambda f.\ e' \triangleright f\ w : (\mathsf{evv}\ \epsilon \rightarrow \lfloor\sigma_2\rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_3\rfloor) \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_3\rfloor$	FABS, $\triangleright$
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} (\lambda f.\ e' \triangleright f\ w) \bullet g : \mathsf{mon}\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_3\rfloor$	$\bullet$

case $\mathsf{E} = \mathsf{E}_0\ [\sigma]$.

$\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_0\ [\sigma] : \sigma_1 \rightarrow \sigma_2\ [\alpha{:=}\sigma] \mid \epsilon \rightsquigarrow (\lambda x.\ \mathit{pure}\ (x[\lfloor\sigma\rfloor])) \bullet g$	given
$\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_0 : \sigma_1 \rightarrow \forall\alpha.\ \sigma_2 \mid \epsilon \rightsquigarrow g$	MON-CTAPP
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} g : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ (\forall\alpha.\ \lfloor\sigma_2\rfloor)$	I.H.
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} (\lambda x.\ \mathit{pure}\ (x[\lfloor\sigma\rfloor])) : (\forall\alpha.\ \lfloor\sigma_2\rfloor) \rightarrow \mathsf{mon}\ \lfloor\sigma_2\rfloor[\alpha{:=}\lfloor\sigma\rfloor]$	FABS, *pure*
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} (\lambda x.\ \mathit{pure}\ (x[\lfloor\sigma\rfloor])) : (\forall\alpha.\ \lfloor\sigma_2\rfloor) \rightarrow \mathsf{mon}\ \lfloor\sigma_2[\alpha{:=}\sigma]\rfloor$	Lemma 33
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} (\lambda x.\ \mathit{pure}\ (x[\lfloor\sigma\rfloor])) \bullet g : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \lfloor\sigma_2[\alpha{:=}\sigma]\rfloor$	$\bullet$

case $\mathsf{E} = v\ w\ \mathsf{E}_0$.

$\Gamma; w; w' \Vdash_{\mathsf{ec}} v\ w\ \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_3 \mid \epsilon \rightsquigarrow (v'\ w) \bullet g$	given
$\Gamma; w; w' \Vdash_{\mathsf{ec}} \mathsf{E}_0 : \sigma_1 \rightarrow \sigma_2 \mid \epsilon \rightsquigarrow g$	MON-CAPP2
$\Gamma \Vdash_{\mathsf{val}} v : \sigma_2 \Rightarrow \epsilon\ \sigma_3 \rightsquigarrow v'$	above
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} g : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \epsilon\ \lfloor\sigma_2\rfloor$	I.H.
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} v' : \mathsf{evv}\ \epsilon \rightarrow \lfloor\sigma_2\rfloor \rightarrow \mathsf{mon}\ \lfloor\sigma_3\rfloor$	Part 1
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} v'\ w : \lfloor\sigma_2\rfloor \rightarrow \mathsf{mon}\ \lfloor\sigma_3\rfloor$	FAPP
$\lfloor\Gamma\rfloor \vdash_{\mathsf{F}} (v'\ w) \bullet g : \mathsf{mon}\ \epsilon\ \lfloor\sigma_1\rfloor \rightarrow \mathsf{mon}\ \lfloor\sigma_3\rfloor$	$\bullet$

case $\mathsf{E} = \mathsf{handle}_m^w\ h\ \mathsf{E}_0$.

$$
\begin{array}{ll}
\Gamma; w; w' \Vdash_{ec} \ \mathsf{handle}^w_m\ h\ \mathsf{E}_0 \ :\ \sigma_1 \to \sigma \mid \epsilon \ \rightsquigarrow \mathit{prompt}[\epsilon,\sigma]\ m\ w \circ g & \text{given} \\
\Gamma \Vdash_{ops}\ h \ :\ \sigma \mid l \mid \epsilon \ \rightsquigarrow h' & \textsc{mon-chandle} \\
\Gamma;\ \langle\!\langle l\!:\!(m,h) \mid w \rangle\!\rangle; \langle\!\langle l\!:\!(m,h') \mid w' \rangle\!\rangle \Vdash_{ec}\ \mathsf{E}_0 \ :\ \sigma_1 \to \sigma \mid \langle l \mid \epsilon \rangle \ \rightsquigarrow g & \text{above} \\
\lfloor\Gamma\rfloor \vdash_{\mathsf{F}}\ h' \ :\ \mathsf{hnd}^l\ \epsilon\ \lfloor\sigma\rfloor & \text{Part 3} \\
\lfloor\Gamma\rfloor \vdash_{\mathsf{F}}\ g \ :\ \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \lfloor\sigma_1\rfloor \to \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \lfloor\sigma\rfloor & \text{above} \\
\lfloor\Gamma\rfloor \vdash_{\mathsf{F}}\ \mathit{prompt}[\epsilon,\sigma]\ m\ w \ :\ \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \lfloor\sigma\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor & \text{above} \\
\lfloor\Gamma\rfloor \vdash_{\mathsf{F}}\ \mathit{prompt}[\epsilon,\sigma]\ m\ w \circ g \ :\ \mathsf{mon}\ \langle l \mid \epsilon \rangle\ \lfloor\sigma_1\rfloor \to \mathsf{mon}\ \epsilon\ \lfloor\sigma\rfloor & \circ
\end{array}
$$

□

C FURTHER EXTENSIONS

This section elaborates some further results and extensions to System F^ϵ.

C.1 Divergence

It is well-known that System F is strongly normalizing and evaluation does not diverge [Girard 1986; Girard et al. 1989] It would be nice to extend that property to System F^ϵ. Unfortunately, the extension with algebraic effect handlers is subtle and we cannot claim strong normalization directly. In particular, the following seemingly well-typed program by Bauer and Pretnar [2015] diverges but has no direct recursion. Assume $\mathit{cow} \ :\ \{\ \mathit{moo} \ :\ () \to (() \to \langle \mathit{cow} \rangle\ ())\ \} \ \in \Sigma$, and let $h \ =\ \{\ \mathit{moo} \to \lambda x. \lambda^{\langle\rangle} k.\ k\ (\lambda^{\langle \mathit{cow} \rangle} y.\ \mathsf{perform}\ \mathit{moo}\ ()\ ())\ \}$, then:

$$
\begin{array}{lll}
 & \mathsf{handler}^{\langle\rangle}\ h\ (\lambda^{\langle \mathit{cow} \rangle}\_.\ \mathsf{perform}\ \mathit{moo}\ ()\ ()) & \\
\longmapsto^* & \mathsf{handle}\ h \cdot \mathsf{perform}\ \mathit{moo}\ ()\ () & (*) \\
= & \mathsf{handle}\ h \cdot \square\ () \cdot \mathsf{perform}\ \mathit{moo}\ () & \\
\longmapsto & \{\ k \ =\ \lambda^{\langle\rangle} x\!:\!() \to \langle \mathit{cow} \rangle\ ().\ \mathsf{handle}\ h \cdot \square\ () \cdot x\ \} & \\
\longmapsto & f\ ()\ k & \\
\longmapsto & k\ (\lambda^{\langle \mathit{cow} \rangle} y.\ (\mathsf{perform}\ \mathit{moo}\ ())\ ()) & \\
\longmapsto & \mathsf{handle}\ h \cdot \square\ () \cdot (\lambda^{\langle \mathit{cow} \rangle} y.\ (\mathsf{perform}\ \mathit{moo}\ ())\ ()) & \\
\longmapsto & \mathsf{handle}\ h \cdot \square\ () \cdot \mathsf{perform}\ \mathit{moo}\ () & \\
= & \mathsf{handle}\ h \cdot \mathsf{perform}\ \mathit{moo}\ ()\ () & (*) \\
\Uparrow & &
\end{array}
$$

The reason for the divergence is that we have accidentally introduced a fancy data type with handlers of the form $\{\ \mathit{op}_i \to f_i\ \}$. As discussed in Section 5.2, we translate the operation signatures to handler data types, where a signature:

$$l \ :\ \{\ \mathit{op} \ :\ \forall \overline{\alpha}.\ \sigma_1 \to \sigma_2\ \}$$

gets translated into a data-type:

$$\mathsf{data}\ \mathsf{hnd}^l\ \mu\ r \ =\ \mathsf{hnd}^l\ \{\ \mathit{op} \ :\ \forall \overline{\alpha}.\ \mathsf{op}\ \sigma_1\ \sigma_2\ \mu\ r\ \}$$

where operations op are a type alias defined as:

$$\mathsf{alias}\ \mathsf{op}\ \alpha\ \beta\ \mu\ r \ \doteq\ \mathsf{evv}\ \mu \to \alpha \to \mathsf{mon}\ (\mathsf{evv}\ \mu \to (\mathsf{evv}\ \mu \to \beta \to \mathsf{mon}\ \mu\ r) \to \mathsf{mon}\ \mu\ r)$$

For the encoding of this data type in F^v we can use the standard technique in terms of universal quantification – as remarked by Wadler [1990]: "*Thus, it is safe to extend the polymorphic lambda calculus by adding least fixpoint types with type variables in positive position. Indeed, no extension is required: such types already exist in the language! If* F X *represents a type containing* X *in positive position only, then least fixpoints may be defined in terms of universal quantification*", e.g. as:

$$\mathsf{lfix}\ \alpha.\ \mathsf{F}\ \alpha \ =\ \forall \alpha.\ (\mathsf{F}\ \alpha \to \alpha) \to \alpha$$

Now we can see where the divergence comes from in our example: the resulting data type $\mathsf{hnd}^{\mathit{cow}}$ cannot be encoded in System F (and F^v) as it occurs in a negative position itself!

Expressions	e	$::=$	$\ldots$	
		$\mid$	$\mathsf{sub}^{\epsilon}\ e$	effect subsumption
Evaluation Context	F	$::=$	$\ldots \mid \mathsf{sub}^{\epsilon}\ \mathsf{F}$	
	E	$::=$	$\ldots \mid \mathsf{sub}^{\epsilon}\ \mathsf{E}$	
Operational Rules	$\ldots$			
	$\mathsf{sub}^{\epsilon}\ v \longrightarrow v$			

$$\frac{\Gamma; w' \vdash e : \sigma \mid \epsilon' \rightsquigarrow e' \quad \epsilon' \sqsubseteq \epsilon \mid w \rightsquigarrow w'}{\Gamma; w \vdash \mathsf{sub}^{\epsilon'}\ e : \sigma \mid \epsilon \rightsquigarrow \mathsf{sub}^{\epsilon'}\ e'}\ [\textsc{sub}]$$

$$\frac{\Gamma; w'_0; w'_1 \Vdash e : \sigma \mid \epsilon' \rightsquigarrow e' \quad \epsilon' \sqsubseteq \epsilon \mid w_1 \rightsquigarrow w'_1}{\Gamma; w_0; w_1 \Vdash \mathsf{sub}^{\epsilon'}\ e : \sigma \mid \epsilon \rightsquigarrow \mathit{cast}[\epsilon, \epsilon', \lfloor \sigma \rfloor]\ e'}\ [\textsc{esub}]$$

$$\frac{}{\epsilon \sqsubseteq \epsilon \mid w \rightsquigarrow w}\ [\textsc{sub-refl}] \qquad \frac{\epsilon' \sqsubseteq \epsilon \mid \mathsf{del}^{l}\ w \rightsquigarrow w'}{\langle l \mid \epsilon' \rangle \sqsubseteq \langle l \mid \epsilon \rangle \mid w \rightsquigarrow \langle\!\langle l : w.l \mid w' \rangle\!\rangle}\ [\textsc{sub-head}]$$

$$\frac{}{\langle\rangle \sqsubseteq \epsilon \mid w \rightsquigarrow \langle\!\langle\rangle\!\rangle}\ [\textsc{sub-total}] \qquad \frac{\langle l' \mid \epsilon' \rangle \sqsubseteq \epsilon \mid \mathsf{del}^{l}\ w \rightsquigarrow w' \quad l \neq l'}{\langle l' \mid \epsilon' \rangle \sqsubseteq \langle l \mid \epsilon \rangle \mid w \rightsquigarrow w'}\ [\textsc{sub-forget}]$$

$$\begin{aligned} &\mathit{cast} : \forall \mu\ \mu'\ \alpha.\ \mathsf{mon}\ \mu'\ \alpha \longrightarrow \mathsf{mon}\ \mu\ \alpha \\ &\mathit{cast}\ (\mathsf{pure}\ x) \qquad\quad\ = \mathsf{pure}\ x \\ &\mathit{cast}\ (\mathsf{yield}\ m\ f\ \mathit{cont}) = \mathsf{yield}\ m\ f\ (\mathit{cast} \bullet \mathit{cont}) \end{aligned}$$

Fig. 14. Effect Subsumption

The operation result parameter β in the op alias occurs in a negative position, and if it is instantiated with a function itself, like $() \rightarrow \langle \mathit{cow} \rangle\ ()$, the monadic translation has type $\mathsf{evv}\ \langle \mathit{cow} \rangle \rightarrow () \rightarrow \mathsf{mon}\ ()$ where the evidence is now a single element vector with one element of type $\exists \mu\ r.\ (\mathsf{marker}\ \mu\ r\ \times \mathsf{hnd}^{\mathit{cow}}\ \mu\ r)$, i.e. the evidence contains the the handler type itself, $\mathsf{hnd}^{\mathit{cow}}$, recursively in a negative position. As a consequence, it cannot be encoded using the standard techniques to System F [Wadler 1990] without breaking strong normalization. In practice, compilers can easily verify if an effect type l occurs negatively in any operation signature to check if effects can be used to encode non-termination. We can use this too to guarantee termination on well-typed System F^{ϵ} terms as long as we require that there are no negative occurrences of l in any signature $l : \{\ \mathit{op}_i : \sigma_i \rightarrow \sigma'_i\ \}$.

C.2 Effect Subsumption

Figure 14 defines effect subsumption in System F^{ϵ} together with typing and translation rules. Note that subsumption is quite different from subtyping as it is syntactical over terms and does not change the equality relation between types. The subsumption relation $\epsilon' \sqsubseteq \epsilon \mid w \rightsquigarrow w'$ states that

ϵ' is a sub-effect of ϵ, and that evidence w, of type evv ϵ, can be run-time translated into evidence w' of type evv ϵ'. The del^l operation is defined as:

$$
\begin{array}{lcl}
\text{del}^l & : & \forall\mu.\ \text{evv}\ \langle l \mid \mu\rangle \rightarrow \text{evv}\ \mu \\
\text{del}^l\ \langle\!\langle\ \rangle\!\rangle & = & \langle\!\langle\ \rangle\!\rangle \\
\text{del}^l\ \langle\!\langle l : ev,\ w\rangle\!\rangle & = & w \\
\text{del}^l\ \langle\!\langle l' : ev,\ w\rangle\!\rangle & = & \langle\!\langle l' : ev,\ \text{del}^l\ w\rangle\!\rangle \quad \text{iff } l \neq l'
\end{array}
$$

At runtime sub is translated to the *cast* function as it has no runtime effect except for changing the effect type of the monad. All evidence is already in the right form due to the sub-effect relation.

Using subsumption we can derive the OPEN and CLOSE type rules introduced by Leijen [2017c]:

$$
\frac{\Gamma \vdash e : \sigma_1 \rightarrow \langle l_1, \ldots, l_n\rangle\ \sigma_2 \mid \epsilon}{\Gamma \vdash \text{open}^{\epsilon'}\ e : \sigma_1 \rightarrow \langle l_1, \ldots, l_n \mid \epsilon'\rangle\ \sigma_2 \mid \epsilon}\ [\text{OPEN}]
$$

$$
\frac{\Gamma \vdash e : \forall\mu.\ \sigma_1 \rightarrow \langle l_1, \ldots, l_n \mid \mu\rangle\ \sigma_2 \mid \epsilon}{\Gamma \vdash \text{close}\ e : \sigma_1 \rightarrow \langle l_1, \ldots, l_n\rangle\ \sigma_2 \mid \epsilon}\ [\text{CLOSE}]
$$

where we can derive each conclusion as:

$$
\begin{array}{lcl}
\text{open}^{\epsilon}\ e & \doteq & \lambda^{\langle l_1, \ldots, l_n \mid \epsilon\rangle} x : \sigma_1.\ \text{sub}^{\langle l_1, \ldots, l_n\rangle}\ (e\ x) \\
\text{close}\ e & \doteq & \lambda^{\langle l_1, \ldots, l_n\rangle} x : \sigma_1.\ e[\langle\rangle]\ x
\end{array}
$$

The subsumption rule can be used in practice to give many functions a closed effect type (using the OPEN rule at instantiation) which in turn allows more operations to use a constant offset to index the handler in the evidence.

C.3 Effect Masking

Figure 15 defines the rules for masking [Convent et al. 2020]. This is also called *inject* [Leijen 2016], or *lift* [Biernacki et al. 2017] in the literature. It is an essential operation for orthogonal composition of effect handlers as it allows one to skip the innermost handler. For example, we may execute an action f together with another internal action g where we only want to handle exceptions for g but not the ones raised in f. With mask we can write this as:

$$\text{handler}\ h_{exn}\ (\lambda\_.\ g\ ();\ \text{mask}^{exn}\ (f\ ()))$$

Even though the operational rule has no effect, we redefine the bound operations to reflect that mask causes the innermost handler to be skipped. There are various ways to do this, Leijen [2016] uses a special context definition while Biernacki et al. [2017] use an n-free definition. Here we simply redefine bop in terms of mbop which uses a multi-map where every operation initially maps to zero. The handle frame increments the count for bound operations while mask decrements them, effectively skipping the next handler in the HANDLE operational rule.

As we can see in the MASK rule, masking simply removes the top evidence for the effect l from the evidence vector. If f itself raises an exception, the outer evidence will now be used. Therefore, once we do a monadic translation, the evidence is already transformed and, just like subsumption, the mask operation itself has no further runtime effect anymore (and can use the *cast* function as well).

Expressions	e	::=	$\ldots$	
		$\mid$	$\mathsf{mask}^l\ e$	effect masking
Evaluation Context	F	::=	$\ldots \mid \mathsf{mask}^l\ \mathsf{F}$	
	E	::=	$\ldots \mid \mathsf{mask}^l\ \mathsf{E}$	
Operational Rules	$\ldots$			
	$\mathsf{mask}^l\ v \longrightarrow v$			

$$\frac{\Gamma; \mathsf{del}^l\ w\ \vdash\ e : \sigma \mid \epsilon\ \rightsquigarrow e'}{\Gamma; w\ \vdash\ \mathsf{mask}^l\ e : \sigma \mid \langle l \mid \epsilon\rangle\ \rightsquigarrow \mathsf{mask}^l\ e'}\ [\textsc{mask}]$$

$$\frac{\Gamma; \mathsf{del}^l\ w; \mathsf{del}^l\ w'\ \Vdash\ e : \sigma \mid \epsilon\ \rightsquigarrow e'}{\Gamma; w; w'\ \Vdash\ \mathsf{mask}^l\ e : \sigma \mid \langle l \mid \epsilon\rangle\ \rightsquigarrow \mathit{cast}[\langle l \mid \epsilon\rangle, \epsilon, \lfloor \sigma \rfloor]\ e'}\ [\textsc{emask}]$$

$$\mathsf{bop}(\mathsf{E}) = \{\ op \mid (op : i)\ \in \mathsf{mbop}(\mathsf{E}),\ i\ \geqslant 1\ \}$$

$$\begin{aligned}
\mathsf{mbop}(\square) &= \{\!\{\ op : 0 \mid op\ \in \Sigma \}\!\} \\
\mathsf{mbop}(\mathsf{E}\ e) &= \mathsf{mbop}(\mathsf{E}) \\
\mathsf{mbop}(v\ \mathsf{E}) &= \mathsf{mbop}(\mathsf{E}) \\
\mathsf{mbop}(\mathsf{handle}\ h\ \mathsf{E}) &= \mathsf{mbop}(\mathsf{E})\ +\ \{\!\{\ op : 1 \mid (op \to f)\ \in h\ \}\!\} \\
\mathsf{mbop}(\mathsf{mask}^l\ \mathsf{E}) &= \mathsf{mbop}(\mathsf{E})\ +\ \{\!\{\ op : -1 \mid (op : \sigma_1 \to \sigma_2)\ \in \Sigma(l)\ \}\!\}
\end{aligned}$$

Fig. 15. Effect Masking