Rows and Capabilities as Modal Effects with Names

WENHAO TANG, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh, UK

Effect handlers allow programmers to model and compose computational effects modularly. Effect systems statically guarantee that all effects are handled. Several recent practical effect systems are based on either row polymorphism or capabilities. However, there remains a gap in understanding the precise relationship between effect systems with such disparate foundations.

We compare the expressive power of three styles of effect systems for effect handlers: row-based effect systems, as in Koka, capability-based effect systems, as in Effekt, and modal effect types, a recent novel approach based on multimodal type theory. Modal effect types provide modular effect types without relying on effect polymorphism. Since both Koka and Effekt support named handlers, we first propose Metn, an extension to modal effect types with named handlers. We give macro translations from variants of row-based and capability-based effect systems into modal effect types that preserve types and semantics. Our translations not only enable a formal comparison of the expressive power of effect systems with different foundations, but also provide a fresh perspective on modal effect types.

1 Introduction

Effect handlers [24] provide a modular and powerful abstraction to define and compose computational effects including states, concurrency nondeterminism, probability, etc. Effect systems statically ensure that all effects used in a program are handled. The literature includes much work on effect systems for effect handlers based on a range of different theoretical foundations. Two of the most popular and well-studied approaches are row-based effect systems [12, 19, 22] and capability-based effect systems [3–5].

Row-based effect systems, as in the research languages Koka [19, 27], Links [12], and Frank [22], follow the traditional monadic reading of effects: effects are what computations do when they run. They treat effect types as a row of effects and each function arrow is annotated with an effect. For modularity, they implement parametric effect polymorphism via row polymorphism [18, 25]. For example, the `map` function in the standard library of Koka has the following type.

```
forall<a,b,e> (xs : list<a>, f : (a) → e b) → e list<b>
```

It is polymorphic in its effects `e`, which must agree with the effect performed by `f`.

Capability-based effect systems, as in the research language Effekt [4, 5] and an extension to Scala 3 [3], adopt a contextual reading of effects: effects are capabilities provided by the context. Treating effects as capabilities enables a notion of *contextual effect polymorphism* [5] which allows effect-polymorphic reuse of functions without effect variables. For example, the `map` function in the standard library of Effekt has the following type.

```
def map[A, B](l: List[A]){ f: A ⇒ B }: List[B]
```

The argument `f` is not restricted to pure functions; it can use whatever capabilities the context provides. Contextual effect polymorphism relies on functions being second-class in Effekt. To write a curried `map`, which requires first-class functions, we must capture capability variables in types:

```
def map1[A,B] {f: A ⇒ B}: List[A] ⇒ List[B] at {f}
```

The capture set `at {f}` tracks the information that the return function invokes the capability `f`.

Though row-based and capability-based effect systems are both well-studied, their relationship is not. A formal comparison of their expressiveness is challenging as they have quite different

Authors' Contact Information: Wenhao Tang, wenhao.tang@ed.ac.uk, The University of Edinburgh, UK; Sam Lindley, sam.lindley@ed.ac.uk, The University of Edinburgh, UK.

theoretical foundations. An alternative foundation has recently emerged in the form of *modal effect types* (Met) [26], a novel approach to effect systems based on multimodal type theory [10, 11, 15]. Similar to Koka, Met has native support for first-class functions. Similar to Effekt, Met manages effects via *effect contexts*, the collection of effects provided by the context, and allows effect-polymorphic reuse of functions without effect variables for a broad range of programs. For example, the map function in Met has the following type.

```
map : ∀ a b . []((a → b) → List a → List b)
```

The syntax [] is a modality indicating that the `map` function itself does not perform effects. All function arrows here are first-class and the argument can use whatever effects from the context.

1.1 Rows and Capabilities as Modal Effect Types

In this paper, we bridge the gap between row-based effect systems and capability-based effect systems via modal effect types. We exhibit type-preserving and semantics-preserving *macro translations* [8] of row-based and capability-based effect systems into modal effect types. We select Koka and Effekt as representatives of row-based and capability-based effect systems, respectively.

Both row-based effect systems and modal effect types were originally conceived with standard Plotkin and Pretnar [24]-style effect handlers (which we call *unnamed handlers*) in mind, whereas capability-based effect systems are typically combined with *named handlers* [2, 5, 29] (also called lexically-scoped handlers). Xie et al. [27] extends Koka with support for named handlers. To resolve this mismatch, we first propose a backward-compatible variant of Met with named handlers, called Metn. This variant itself is non-trivial and demonstrates the generalisability of modal effect types.

To capture the essence of Koka and Effekt, we work with representative core calculi. System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ [27] is a core calculus formalising the full power of the row-based effect system of Koka with scope-safe named handlers. It supports parametric effect polymorphism and uses rank-2 polymorphism to ensure scope safety of handler names. System C [4] is a core calculus formalising the full power of the capability-based effect system of Effekt. It supports contextual effect polymorphism and provides explicit *boxes* for capturing capability variables in types, and thus constructing first-class functions. We show that Metn *extended with effect variables* is sufficiently expressive to encode both System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and System C.

To study their expressiveness in a tighter more ergonomic setting, we work with subcalculi of our representative core calculi. System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ is a fragment of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ where each scope can only refer to the lexically closest effect variable, following Tang et al. [26]. Consequently, in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ there is actually no need to ever mention the single effect variable in types, corresponding to the syntactic sugar that Frank [22] uses to hide effect variables in types. System Ξ is a fragment of System C with only second-class functions and no boxes. Its type cannot mention any capability variables. We prove that simply-typed Metn *without effect variables* is sufficiently expressive to encode both System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ and System Ξ.

The main contributions of this paper are as follows.

- We give a high-level overview of modal effect types, named handlers, and the ideas of our encodings through a series of examples (Section 2).
- We formally present Metn, a core calculus with named handlers and modal effect types (Section 3). We introduce several extensions to Metn including effect variables and masking handler names (Section 4).
- We present System Ξ and System C, two core calculi for Effekt (Section 5). We encode them into Metn and prove that our encodings preserve types and semantics (Section 6).
- We present System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$, two core calculi for Koka (Section 7). We encode them into Metn and prove that our encodings preserve types and semantics (Section 8).

Section 9 discusses related and future work. The full specifications and proofs can be found in the supplementary material.

2 Overview

In this section, we first recap core ideas of modal effect types and show how named handlers work with modal effect types. Then we provide a high-level overview of row-based and capability-based effect systems and how they are encoded with modal effect types. For simplicity, we restrict attention to effect with a single operation, and omit handler return clauses. Nonetheless, our results extend mutatis mutandis to accommodate both effects with multiple operations and handler return clauses. Also, our translation examples in this section differ slightly from the results derived from strictly applying the translations in Sections 6 and 8, omitting boring details only used to keep the uniformity of translations.

2.1 A Taste of Modal Effect Types

We provide a quick overview on the core ideas of modal effect types in METN, a backward-compatible extension to MET [26]. We formally present METN in Section 3 and provide a full specification of METN with both unnamed and named handlers in Appendix A.

METN subsumes simply-typed λ-calculus, faithfully. That is, any program well-typed in simply-typed λ-calculus is well-typed in METN with exactly the same syntax, type, and semantics. For example, we have the standard application function $app_{\text{METN}} \doteq \lambda f^{\text{Int}\to 1}.\lambda x^{\text{Int}}.f\ x$ in METN. Throughout the overview, we use meta-level macros defined by $\doteq$ in red to refer to code snippets.

Effect Contexts. METN does not annotate effect types on function arrows. Consider an effect $\texttt{Gen} = \{\texttt{yield} : \texttt{Int} \twoheadrightarrow 1\}$ with one operation `yield` that takes an integer and returns a unit. We have the following function which invokes `yield` using the **do** syntax.

$$\vdash gen_{\text{METN}} \doteq \lambda x^{\text{Int}}.\mathbf{do}\ \texttt{yield}\ x : \texttt{Int} \to 1\ @\ \texttt{Gen}$$

This function has type $\texttt{Int} \to 1$ with no effect annotation, which does not imply purity but means that it can use any effects from the *ambient effect context* as specified by $@\ \texttt{Gen}$. This is the key for METN to achieve effect-polymorphic reuse of functions without really using effect polymorphism. For example, we can apply the app_{METN} function to the gen_{METN} function as follows.

$$\vdash (\lambda f^{\text{Int}\to 1}.\lambda x^{\text{Int}}.f\ x)\ (\lambda x^{\text{Int}}.\mathbf{do}\ \texttt{yield}\ x)\ 42 : 1\ @\ \texttt{Gen}$$

Note that even though the type of f does not specify effect `Gen`, it still takes an argument which invokes `yield`, as the effect context provides us with `Gen`. There is also a natural notion of subeffecting on effect contexts. For instance, we can upcast the ambient effect context of gen_{METN} as follows.

$$\vdash \lambda x^{\text{Int}}.\mathbf{do}\ \texttt{yield}\ x : \texttt{Int} \to 1\ @\ \texttt{Gen}, \texttt{IO}, \texttt{Read}$$

Absolute Modalities. It is useful to be able to specify a new effect context in types different from the ambient one. For example, we may want to keep the information that gen only uses effect `Gen` even though the ambient effect context provides more as in the above judgement. We can do so via an *absolute modality* as follows.

$$\frac{🔒_{[\texttt{Gen}]} \vdash \lambda x^{\text{Int}}.\mathbf{do}\ \texttt{yield}\ x : \texttt{Int} \to 1\ @\ \texttt{Gen}}{\vdash \mathbf{mod}_{[\texttt{Gen}]}\ (\lambda x^{\text{Int}}.\mathbf{do}\ \texttt{yield}\ x) : [\texttt{Gen}](\texttt{Int} \to 1)\ @\ \texttt{Gen}, \texttt{IO}, \texttt{Read}}$$

The syntax $\mathbf{mod}_{[\texttt{Gen}]}$ introduces an absolute modality $[\texttt{Gen}]$ to the type which specifies a singleton effect context of `Gen`. As reflected in the typing judgement of the premise, the inside function $\lambda x^{\text{Int}}.\mathbf{do}\ \texttt{yield}\ x$ can only use the effect `Gen` from this new effect context. The lock $🔒_{[\texttt{Gen}]}$ tracks the switch of the effect context and controls the accessibility of variables on the left of it. Only those variables that we know do not use any other effects than Gen can be used. This is important to effect safety. For example, consider the following invalid derivation.

$$\frac{f : \texttt{Int} \to 1, \blacksquare_{[\texttt{Gen}]} \nvdash \lambda x^{\texttt{Int}}.f\ x : \texttt{Int} \to 1 \ @\ \texttt{Gen}}{f : \texttt{Int} \to 1 \nvdash \textbf{mod}_{[\texttt{Gen}]}\ (\lambda x^{\texttt{Int}}.f\ x) : [\texttt{Gen}](\texttt{Int} \to 1) \ @\ \texttt{Gen}, \texttt{IO}, \texttt{Read}}$$

This program is unsafe because we do not know which effects f uses and cannot just claim that it only uses the effect Gen. The $\blacksquare_{[\texttt{Gen}]}$ rejects the usage of the function variable f. In Section 3 we will dive into details of the type system of METN.

In order to eliminate the modality of gen_{METN} and apply it, we use the **let mod** syntax.

$$\frac{\_ \qquad f :_{[\texttt{Gen}]} \texttt{Int} \to 1 \vdash f\ 42 : 1 \ @\ \texttt{Gen}}{\vdash \textbf{let mod}_{[\texttt{Gen}]}\ f = \textbf{mod}_{[\texttt{Gen}]}\ (\lambda x^{\texttt{Int}}.\textbf{do}\ \texttt{yield}\ x)\ \textbf{in}\ f\ 42 : 1 \ @\ \texttt{Gen}}$$

We write _ to denote a judgement that we have omitted. The binding of f is annotated with the modality [Gen] to track that the value it is bound to may use and only use the effect Gen. Consequently, the usage of f in f 42 requires the ambient effect context to contain the effect Gen.

Relative Modalities. As well as being able to specify a fresh effect context from scratch with an absolute modality, it is also useful to be able to specify a new effect context by transforming the ambient one. For example, consider a handler for the effect Gen.

$$\vdash sum1_{\text{METN}} \doteq \lambda f.\textbf{let mod}_{[\texttt{Gen}]}\ f = f\ \textbf{in handle}\ f\ ()\ \textbf{with}\ H_{\texttt{Gen}} : [\texttt{Gen}](1 \to \texttt{Int}) \to \texttt{Int} \ @\ \cdot$$

where $H_{\texttt{Gen}} = \{\texttt{yield}\ p\ r \mapsto p + r\ ()\}$. This function takes an argument f of type $[\texttt{Gen}](1 \to \texttt{Int})$ and eliminates its modality first before applying it. (For concreteness, we will explicitly write such bureaucratic modality elimination, using a grey font, but in practice it can be inferred [26].) In the handler $H_{\texttt{Gen}}$, the variable p of type Int is bound to the parameter of the yield operation, and the variable r of type $1 \to \texttt{Int}$ is bound to its continuation. We add the yielded integer p to the result of the continuation r. As our handlers are *deep* [14], other yield operations in r are also handled and their arguments are added together. For example, the following program reduces to 79.

$$\vdash sum1_{\text{METN}}\ (\textbf{mod}_{[\texttt{Gen}]}\ (\lambda().\textbf{do}\ \texttt{yield}\ 42; \textbf{do}\ \texttt{yield}\ 37; 0)) : \texttt{Int} \ @\ \cdot$$

It is rather restrictive to insist that the argument f can only use the effect Gen. In order to support arbitrary additional effects in f, we can use a *relative modality* $\langle\!|\texttt{Gen}\rangle$, which transforms the ambient context by extending it with Gen.

$$\vdash sum2_{\text{METN}} \doteq \lambda f.\textbf{let mod}_{\langle|\texttt{Gen}\rangle}\ f = f\ \textbf{in handle}\ f\ ()\ \textbf{with}\ H_{\texttt{Gen}} : (\langle|\texttt{Gen}\rangle(1 \to \texttt{Int})) \to \texttt{Int} \ @\ \cdot$$

Consequently, by upcasting the ambient effect context to $\texttt{Read} = \{\texttt{ask} : 1 \twoheadrightarrow \texttt{Int}\}$, we can apply $sum2_{\text{METN}}$ to a function that invokes both yield and ask as follows.

$$\vdash sum2_{\text{METN}}\ (\textbf{mod}_{\langle|\texttt{Gen}\rangle}\ (\lambda().\textbf{do}\ \texttt{yield}\ 42; \textbf{do}\ \texttt{ask}\ ())) : \texttt{Int} \ @\ \texttt{Read}$$

We now briefly describe two more esoteric features of MET that we will make use of later.

Effect Variables. Tang et al. [26] show that, though for a wide range of situations effect polymorphism is entirely unnecessary, effect variables can still be useful in MET for implementing features such as higher-order effects. Moreover, they demonstrate that MET can be naturally extended with effect variables. We can, for instance give an effect-polymorphic version of the sum handler.

$$\begin{aligned} \vdash sume_{\text{METN}} &\doteq \Lambda\varepsilon.\textbf{mod}_{[\varepsilon]}\ (\lambda f^{[\texttt{Gen},\varepsilon](1\to\texttt{Int})}.\textbf{let mod}_{[\texttt{Gen},\varepsilon]}\ f = f\ \textbf{in handle}\ f\ ()\ \textbf{with}\ H_{\texttt{Gen}}) \\ &: \ \forall\varepsilon.[\varepsilon]([\texttt{Gen},\varepsilon](1 \to \texttt{Int}) \to \texttt{Int}) \ @\ \cdot \end{aligned}$$

We can apply $sume_{\textsc{Metn}}$ to functions with different effects by instantiating ε differently. Note that this is merely an artificial example to show how effect variables work in Metn; $sum2_{\textsc{Metn}}$ already achieves this kind of modularity without relying on an effect variable ε.

Absolute Handlers. In $sume_{\textsc{Metn}}$, the continuation r in $H_{\texttt{Gen}}$ is given a function type $1 \to \texttt{Int}$ just as in other versions of the sum handler. However, here we actually know more about r: it can only use effects in ε since r is bound to the continuations of `yield` operations invoked in function f and f has type $[\texttt{Gen}, \varepsilon](1 \to \texttt{Int})$. It would be ideal to have an absolute modality $[\varepsilon]$ wrapping the function type of r. An *absolute handler* allows us to additionally wrap the continuation function with some absolute modality.

$$\vdash \Lambda\varepsilon.\mathbf{mod}_{[\varepsilon]}\ (\lambda f^{[\texttt{Gen},\varepsilon](1\to\texttt{Int})}.\mathbf{let\ mod}_{[\texttt{Gen},\varepsilon]}\ f = f\ \mathbf{in\ handle}^{\spadesuit}\ f\ ()\ \mathbf{with}\ H_{\texttt{Gen}}) : \texttt{Int} \ @\ \cdot$$

The only syntax change is the $\spadesuit$ annotation which indicates an absolute handler. The typing rule for the absolute handler above introduces a lock $\text{🔒}_{[\varepsilon]}$ to guarantee that the continuation cannot use any other effects than ε. We will come back to its typing rule in Section 4.2.

2.2 Met with Named Handlers

Standard effect handlers are unnamed, but some systems use named handlers [2, 5, 27, 29]. The difference is analogous to that between single-prompt delimited control (unnamed effect handlers) and multiprompt delimited control (named effect handlers).

Effect Instances. What should we do if we want to use two generators at the same time? This can be tricky with unnamed handlers. Met allows us to have multiple appearances of `Gen` in an effect context and refer to different appearances of them by inserting masks [1, 7, 19] manually. Named handlers offer a more direct alternative by giving names to handlers; operations must then be invoked with an explicit handler name.

$$\frac{a : \texttt{Gen}, \text{🔒}_{\langle\!|a\rangle}, b : \texttt{Gen}, \text{🔒}_{\langle\!|b\rangle} \vdash \mathbf{do}_a\ \texttt{yield}\ 42; \mathbf{do}_b\ \texttt{yield}\ 37; 0 : \texttt{Int}\ @\ a, b}{\vdash \mathbf{handle}_a\ (\mathbf{handle}_b\ (\mathbf{do}_a\ \texttt{yield}\ 42; \mathbf{do}_b\ \texttt{yield}\ 37; 0)\ \mathbf{with}\ H_{\texttt{Gen}})\ \mathbf{with}\ H_{\texttt{Gen}} : \texttt{Int}\ @\ \cdot}$$

The keyword $\mathbf{handle}_a$ introduces a name binding $a : \texttt{Gen}$ and $\mathbf{handle}_b$ introduces another name binding $b : \texttt{Gen}$. The locks $\text{🔒}_{\langle\!|a\rangle}$ and $\text{🔒}_{\langle\!|b\rangle}$ allow the effect context to be extended with these names. The handled computation yields 42 via the keyword $\mathbf{do}_a$ with the handler name a and 37 via the keyword $\mathbf{do}_b$ with handler name b. These two operation invocations are handled by the corresponding handlers (which in this case are the same, but in general need not be).

As another example, we can define a function $sum_{\textsc{Metn}}$ that takes an argument and handles it with a named handler for the `Gen` effect as follows.

$$\begin{aligned}\vdash sum_{\textsc{Metn}} &\doteq \lambda f^{\forall a^{\texttt{Gen}}.\langle\!|a\rangle(1\to\texttt{Int})}.\mathbf{handle}_b\ (\mathbf{let\ mod}_{\langle\!|b\rangle}\ f = f\ b\ \mathbf{in}\ f\ ())\ \mathbf{with}\ H_{\texttt{Gen}} \\ &: (\forall a^{\texttt{Gen}}.\langle\!|a\rangle(1 \to \texttt{Int})) \to \texttt{Int}\ @\ \cdot\end{aligned}$$

As with $sum2_{\textsc{Metn}}$, the type of f has a relative modality $\langle\!|a\rangle$ which extends the ambient effect context with the handler name a. Moreover, f abstracts over the handler name a for the effect `Gen` via the name abstraction $\forall a^{\texttt{Gen}}$. We use the name b bound by the handler $\mathbf{handle}_a$ to instantiate f through the name application $f\ b$. We can apply $sum_{\textsc{Metn}}$ as follows, where we write $\lambda\!\!\!\lambda a^{\texttt{Gen}}$ for name abstraction on the term level.

$$\vdash sum_{\textsc{Metn}}\ (\lambda\!\!\!\lambda a^{\texttt{Gen}}.\lambda().\mathbf{do}_a\ \texttt{yield}\ 42; \mathbf{do}_a\ \texttt{yield}\ 37; 0)$$

Scope Safety. In the term $\mathbf{handle}_a\ M\ \mathbf{with}\ H$, how can we guarantee that the handler name a cannot escape the scope of M? First, by well-scopedness, we can ensure that the return type of M

cannot mention the handler name a. However, this is not enough in METN, as we can write the following valid derivation where $H_{\mathsf{Read}} = \{\mathsf{ask}\ ()\ r \mapsto r\ 42\}$.

$$\frac{a : \mathsf{Read} \vdash \lambda().\mathbf{do}_a\ \mathsf{ask}\ () : 1 \to \mathsf{Int} \ @\ a \qquad \_}{\vdash \mathbf{handle}_a\ (\lambda().\mathbf{do}_a\ \mathsf{ask}\ ())\ \mathbf{with}\ H_{\mathsf{Read}} : 1 \to \mathsf{Int} \ @\ \cdot}$$

The function $\lambda().\mathbf{do}_a\ \mathsf{ask}\ ()$ is directly returned from the handled computation and can be used to invoke the ask operation with the handler name a out of the scope of the handler of name a. To prevent handler names from escaping, we restrict the return type of the handled computation to be an *absolute type* of kind Abs. A type is absolute if function types in it only appear inside absolute modalities. Values of such types cannot depend on the ambient effect context. Consequently, in order to type the above program, we would have to wrap the handled computation in an absolute modality that mentions a, such as the following.

$$\nvdash \mathbf{handle}_a\ (\mathbf{mod}_{[a]}\ (\lambda().\mathbf{do}_a\ \mathsf{ask}\ ()))\ \mathbf{with}\ H_{\mathsf{Read}} : [a](1 \to \mathsf{Int}) \ @\ \cdot$$

But this is ill-typed as the handler name a does now appear in the return type.

Masking Handler Names. It can be too restrictive sometimes to force the handled computation of named handlers to return values of absolute types. For instance, the following judgement is rejected but actually does not leak any handler names.

$$\nvdash \mathbf{handle}_a\ ((\mathbf{handle}_b\ (\lambda x^{\mathsf{Int}}.\mathbf{do}_a\ \mathsf{yield}\ x)\ \mathbf{with}\ H_{\mathsf{Read}})\ 42; 0)\ \mathbf{with}\ H_{\mathsf{Gen}} : \mathsf{Int} \ @\ \cdot$$

The handled computation returns a function $\lambda x^{\mathsf{Int}}.\mathbf{do}_a\ \mathsf{yield}\ x$ which only uses the handler name a. It is definitely fine to return it from the handler with the name b. To allow such programs, we extend masks to allow handler names to be removed from the effect context.

$$\dfrac{\dfrac{a : \mathsf{Gen}, 🔒_{\langle| a\rangle}, b : \mathsf{Gen}, 🔒_{\langle| b\rangle}, 🔒_{\langle b |\rangle} \vdash \lambda x^{\mathsf{Int}}.\mathbf{do}_a\ \mathsf{yield}\ x : \mathsf{Int} \to 1 \ @\ a}{a : \mathsf{Gen}, 🔒_{\langle| a\rangle}, b : \mathsf{Gen}, 🔒_{\langle| b\rangle} \vdash \mathbf{mask}_b\ (\lambda x^{\mathsf{Int}}.\mathbf{do}_a\ \mathsf{yield}\ x) : \langle b|\rangle(\mathsf{Int} \to 1) \ @\ a, b}}{\vdash \mathbf{handle}_a\ ((\mathbf{handle}_b\ (\mathbf{mask}_b\ (\lambda x^{\mathsf{Int}}.\mathbf{do}_a\ \mathsf{yield}\ x))\ \mathbf{with}\ H_{\mathsf{Read}})\ 42; 0)\ \mathbf{with}\ H_{\mathsf{Gen}} : \mathsf{Int} \ @\ \cdot}$$

The syntax $\mathbf{mask}_b$ removes the handler name b from the effect context. Moreover, it introduces a relative modality $\langle b|\rangle$ to the type and a lock $🔒_{\langle b|\rangle}$ to the context of the premise. The lock $🔒_{\langle b|\rangle}$ removes the name b extended by the lock $🔒_{\langle| b\rangle}$, leaving only name a in the effect context. Operationally, the $\mathbf{handle}_b$ and $\mathbf{mask}_b$ cancel each other, and the program reduces to 42.

2.3 Encoding Effect Rows without using Effect Variables

System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ [27] is a core calculus with an effect system based on Leijen [18]-style row types and formalises scope-safe named handlers in Koka. We first consider a fragment of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ called System $\mathsf{F}^{\epsilon+\mathsf{sn}}_1$ inspired by Tang et al. [26] where in each position of types and terms we can only refer to the lexically closest effect variable. This fragment characterises the syntactic sugar that Frank uses to hide effect variables and covers a wide range of practical programs. For example, we can write the standard application function in System $\mathsf{F}^{\epsilon+\mathsf{sn}}_1$ as follows, where λs are annotated with the effects of functions in the form of λ^E.

$$\vdash app_{\mathsf{F}^{\epsilon+\mathsf{sn}}_1} \doteq \Lambda\varepsilon.\lambda^{\varepsilon} f^{\mathsf{Int}\to^{\varepsilon} 1}.\lambda^{\varepsilon} x^{\mathsf{Int}}.f\ x : \forall\varepsilon.(\mathsf{Int} \to^{\varepsilon} 1) \to^{\varepsilon} \mathsf{Int} \to^{\varepsilon} 1$$

By instantiating the effect variable ε to different effect rows E, we can apply $app_{\mathsf{F}^{\epsilon+\mathsf{sn}}_1}$ to different effectful functions. We can encode $app_{\mathsf{F}^{\epsilon+\mathsf{sn}}_1}$ in METN as app_{METN} defined in Section 2.1 with an empty absolute modality wrapping around it as follows. We write $[\![-]\!]$ for translations.

$$\vdash [\![app_{\mathsf{F}^{\epsilon+\mathsf{sn}}_1}]\!] \doteq \mathbf{mod}_{[]}\ (\lambda f^{\mathsf{Int}\to 1}.\lambda x^{\mathsf{Int}}.f\ x) : []((\mathsf{Int} \to 1) \to \mathsf{Int} \to 1) \ @\ \cdot$$

The empty absolute modality $[]$ here simulates the behaviour of the effect abstraction $\Lambda\varepsilon$ in $app_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$, as by subeffecting it also enables $[\![app_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}]\!]$ to take any effectful functions. For instance, we can translate an instantiation and application of $app_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$ to a modality elimination and application.

$$[\![app_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}\ E\ V]\!] \quad = \quad \textbf{let mod}_{[]}\ f = [\![app_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}]\!]\ \textbf{in}\ f\ [\![V]\!]$$

System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ treats handler names as first-class values and uses rank-2 polymorphism to ensure their scope safety, similar to the technique used by `runST` in Haskell [16]. System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ uses the **nhandler** H syntax to introduce a handler which has a rank-2 polymorphic type.

$$\vdash sum_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}} \doteq \Lambda\varepsilon.\textbf{nhandler}^{\varepsilon}\ \{\mathtt{yield}\ p\ r \mapsto p + r\ ()\}\ : \forall\varepsilon.(\forall a.\mathsf{ev}\,\mathtt{Gen}^a \to^{\mathtt{Gen}^a,\varepsilon} \mathtt{Int}) \to^{\varepsilon} \mathtt{Int} \mid \cdot$$

For now, let us ignore the scope variable a. The term $sum_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$ is polymorphic over other effects ε that it does not handle. It takes an argument of type $\forall a.\mathsf{ev}\,\mathtt{Gen}^a \to^{\mathtt{Gen}^a,\varepsilon} \mathtt{Int}$, where the *evidence* type $\mathsf{ev}\,\mathtt{Gen}^a$ stands for a first-class handler name for a handler that handles the effect `Gen`. For example, we can apply $sum_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$ as follows.

$$\vdash sum_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}\ \cdot\ (\Lambda a.\lambda h^{\mathsf{ev}\,\mathtt{Gen}^a}.\textbf{do}\ h\ 42; \textbf{do}\ h\ 37; 0) : \mathtt{Int} \mid \cdot$$

We instantiate the effect variable ε to the empty effect row $\cdot$. The named handler handles the operation invocations $\textbf{do}\ h\ 42$ and $\textbf{do}\ h\ 37$ with the handler name h. We omit the operation label `yield` as there is only one operation in the effect `Gen`. This program reduces to 79.

The scope variable a guarantees that the handler name h cannot escape the scope of the handler; as a is universally quantified it cannot escape its scope. For instance, if the handled computation returned a function of type $\mathtt{1} \to^{\mathtt{Gen}^a} \mathtt{1}$ rather than an integer, then the type of the named handler

$$\forall\varepsilon.(\forall a.\mathsf{ev}\,\mathtt{Gen}^a \to^{\mathtt{Gen}^a,\varepsilon} (\mathtt{1} \to^{\mathtt{Gen}^a} \mathtt{1})) \to^{\varepsilon} (\mathtt{1} \to^{\mathtt{Gen}^a} \mathtt{1})$$

would be ill-scoped as the rightmost a appears unbound.

In System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$, handler names are first-class values, whereas in Metn they are second-class that cannot be passed and returned as values. In order to encode named handlers of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ in Metn, we translate an evidence type $\mathsf{ev}\,\mathtt{Gen}^a$ to a function type $[a](\mathtt{Int} \to \mathtt{1})$. For an operation invocation $\textbf{do}\ h\ 42$ where $h : \mathsf{ev}\,\mathtt{Gen}^a$, we can simulate it by unboxing the translated function and applying it to 42. The translation of $sum_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$ is as follows.

$$\begin{aligned}\vdash [\![sum_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}]\!] \doteq\ & \textbf{mod}_{[]}\ (\lambda f^{\forall a^{\mathtt{Gen}}.\langle|a\rangle([a](\mathtt{Int}\to\mathtt{1})\to\mathtt{Int})}.\textbf{handle}_a \\ & \quad (\textbf{let mod}_{\langle|a\rangle}\ f = f\ a\ \textbf{in}\ f\ (\textbf{mod}_{[a]}\ (\lambda x^{\mathtt{Int}}.\textbf{do}_a\ \mathtt{yield}\ x)))\ \textbf{with}\ H_{\mathtt{Gen}}) \\ & : []((\forall a^{\mathtt{Gen}}.\langle a\rangle([a](\mathtt{Int} \to \mathtt{1}) \to \mathtt{Int})) \to \mathtt{Int})\ @\ \cdot\end{aligned}$$

The translation result is similar to sum_{Metn} in Section 2.2. The main difference is that we pass a function $\textbf{mod}_{[a]}(\lambda x^{\mathtt{Int}}.\textbf{do}_a\ \mathtt{yield}\ x)$ to replace the first-class handler name in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$. Our encoding demonstrates that there is actually no gap in the expressiveness between first-class handler names and second-class handler names.

We present System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ in Section 7.2 and encode it into Metn without effect variables in Section 8.2. Our encoding exploits the ability to mask handler names.

2.4 Encoding Effect Rows using Effect Variables

Not all programs in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ live in the fragment of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$. For instance, considering the application function $app_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$, the top function arrow does not necessarily need to be annotated with the effect variable ε. We can define the following function not covered in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$.

$$\vdash app_{\mathsf{F}^{\epsilon+\mathsf{sn}}} \doteq \Lambda\varepsilon'.\Lambda\varepsilon.\lambda^{\varepsilon'} f^{\mathtt{Int}\to^{\varepsilon}\mathtt{Int}}.\lambda^{\varepsilon} x^{\mathtt{Int}}.f\ x : \forall\varepsilon'.\forall\varepsilon.(\mathtt{Int} \to^{\varepsilon} \mathtt{Int}) \to^{\varepsilon'} \mathtt{Int} \to^{\varepsilon} \mathtt{Int}$$

Our previous translation fails now as the right two function arrows necessarily must share the same effect context. In order to precisely encode $\mathit{app}_{\mathsf{F}^{\epsilon+\mathsf{sn}}}$, we need to use effect variables in Metn.

$$\begin{aligned}\vdash [\![\mathit{app}_{\mathsf{F}^{\epsilon+\mathsf{sn}}}]\!] &\doteq \Lambda\varepsilon'.\Lambda\varepsilon.\mathbf{mod}_{[\varepsilon']}\ (\lambda f^{[\varepsilon](\mathtt{Int}\to\mathtt{Int})}.\mathbf{let\ mod}_{[\varepsilon]}\ f = f\ \mathbf{in}\ \mathbf{mod}_{[\varepsilon]}\ (\lambda x^{\mathtt{Int}}.f\ x)) \\ &: \forall\varepsilon'.\forall\varepsilon.[\varepsilon']([\varepsilon](\mathtt{Int}\to\mathtt{Int})\to[\varepsilon](\mathtt{Int}\to\mathtt{Int}))\ @\ \cdot\end{aligned}$$

As we can see, the principle of the translation here is quite different from that for System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$. Previously, we translate away all abstractions and usages of effect variables. Now the translation strictly preserves effect abstraction. Moreover, for each function type $A \to^E B$ in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, we translate it to a function type with an absolute modality as $[[\![E]\!]]([\![A]\!] \to [\![B]\!])$ in Metn.

Following the translation principle here, we can give a different translation of $\mathit{sum}_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$. We write $\mathit{sum}_{\mathsf{F}^{\epsilon+\mathsf{sn}}}$ for $\mathit{sum}_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$ interpreted in $\mathsf{F}^{\epsilon+\mathsf{sn}}$.

$$\begin{aligned}\vdash [\![\mathit{sum}_{\mathsf{F}^{\epsilon+\mathsf{sn}}}]\!] &\doteq \Lambda\varepsilon.\mathbf{mod}_{[\varepsilon]}\ (\lambda f^{\forall a^{\mathsf{Gen}}.[a,\varepsilon]([a](\mathtt{Int}\to 1)\to\mathtt{Int})}.\mathbf{handle}^{\spadesuit}_{a} \\ &\qquad (\mathbf{let\ mod}_{[a,\varepsilon]}\ f = f\ a\ \mathbf{in}\ f\ (\mathbf{mod}_{[a]}(\lambda x^{\mathtt{Int}}.\mathbf{do}_a\ \mathtt{yield}\ x)))\ \mathbf{with}\ H_{\mathsf{Gen}}) \\ &: \forall\varepsilon.[\varepsilon]((\forall a^{\mathsf{Gen}}.[a,\varepsilon]([a](\mathtt{Int}\to 1)\to\mathtt{Int}))\to\mathtt{Int})\ @\ \cdot\end{aligned}$$

We abstract over an effect variable ε and change the relative modality $\langle\!|a\rangle$ to the absolute modality $[a, \varepsilon]$. Crucially, we also switch to an absolute handler here. Using an absolute handler gives the continuation r in H_{Gen} the right type $[\varepsilon](1 \to \mathtt{Int})$, maintaining the invariant that every function arrow with an effect annotation is translated to a function arrow with an absolute modality.

We present System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ in Section 7.1 and encode it into Metn with effect variables in Section 8.1.

2.5 Encoding Capabilities without using Effect Variables

Effekt is a research programming language with an effect system based on capabilities which enables contextual effect polymorphism, reducing the burden of effect variables in a similar way to Metn. System Ξ [5] is an earlier version of the core calculus for Effekt in explicit capability-passing style with only second-class functions (called blocks). System Ξ does not track effects explicitly. All effects are capabilities provided by the context and we can only use them when they are in scope. System Ξ does not distinguish between capabilities representing effects and capabilities representing block variables. For clarity, here we capitalise the former. As an example, the $\mathit{gen}_{\textsc{Metn}}$ function in Section 2.1 is defined as follows in System Ξ.

$$\mathit{Yield} : \mathtt{Int} \Rightarrow 1 \vdash \mathit{gen}_{\Xi} \doteq \{x^{\mathtt{Int}} \Rightarrow \mathit{Yield}(x)\} : \mathtt{Int} \Rightarrow 1$$

Following Brachthäuser et al. [5], we write braces to delimit blocks and use parentheses to wrap their arguments when calling them. Double arrows denote second-class functions. This block invokes the capability *Yield* from the context to yield an integer 42. Similar to Met, though block arrows in System Ξ have no effect annotation, they are not pure but can use any capabilities from the context. This is the key to contextual effect polymorphism in Effekt. Let us consider the application function in System Ξ:

$$\vdash \mathit{app}_{\Xi} \doteq \{(x^{\mathtt{Int}}, f^{\mathtt{Int}\Rightarrow 1}) \Rightarrow f(x)\} : (\mathtt{Int}, \mathtt{Int} \Rightarrow 1) \Rightarrow 1$$

The block app_{Ξ} takes an integer x and another block f. It allows the block argument f to use any capabilities from the context. For instance, we may use $\mathit{app}_{\Xi}(42, \mathit{gen}_{\Xi})$ as long as the capability $\mathit{Yield} : \mathtt{Int} \Rightarrow 1$ is in the context. Note that System Ξ requires value parameters such as x to appear before block parameters such as f. Moreover, due to being second-class, blocks must be fully applied and cannot be returned.

A named handler in System Ξ introduces a new capability to its scope. For instance, we can define a handler that introduces a *Yield* capability and use it to handle a computation yielding 42[1].

$$\vdash \mathit{sum42}_{\Xi} \doteq \textbf{handle}\ \{\mathit{Yield}^{\mathtt{Int}\Rightarrow 1} \Rightarrow \mathit{Yield}(42)\}\ \textbf{with}\ \{\mathtt{yield}\ p\ r \mapsto p + r(())\} : \mathtt{Int}$$

Capabilities cannot leave the scope of their corresponding handlers because they are second-class. We can neither directly return a capability representing an effect nor a block that captures other capabilities. For instance, the following judgement is invalid.

$$\nvdash \textbf{handle}\ \{\mathit{Yield}^{\mathtt{Int}\Rightarrow 1} \Rightarrow \mathit{Yield}\}\ \textbf{with}\ \{\mathtt{yield}\ p\ r \mapsto p + r(())\} : \mathtt{Int}$$

The encoding of System Ξ in Metn is quite direct. It mostly just rewrites the syntax of second-class blocks in System Ξ to standard first-class curried functions in Metn. For instance, app_{Ξ} is encoded directly as $\lambda x^A.\lambda f^{A\to B}.f\ x$. The only non-trivial case is for named handlers where we simulate capability introduction by constructing a function, similar to the encoding of named handlers in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ and System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. For instance, to encode $\mathit{sum42}_{\Xi}$, we can bind a function that invokes the $\mathtt{yield}$ operation to f and use it to replace the usage of the *Yield* capability.

$$\vdash [\![\mathit{sum42}_{\Xi}]\!] \doteq \textbf{handle}_a\ (\textbf{let}\ f = \lambda x^{\mathtt{Int}}.\textbf{do}_a\ \mathtt{yield}\ x\ \textbf{in}\ f\ 42)\ \textbf{with}\ H_{\mathsf{Gen}} : \mathtt{Int}\ @\ \cdot$$

Note that we cannot return f from the handled computation in Metn, because the typing rule requires the return type to either have kind Abs or have a relative modality $\langle a|\rangle$ that removes a.

As System Ξ has no effect system, our encoding does not use modalities of Metn. Our encoding of System Ξ in Metn shows that it is unnecessary to completely sacrifice first-class functions to obtain contextual effect polymorphism.

We present System Ξ in Section 5.2 and encode it in Metn without effect variables in Section 6.1.

2.6 Encoding Capabilities using Effect Variables

System C [4], a later core calculus for Effekt, extends System Ξ with boxes which lift second-class blocks to first-class values. The type of a boxed block is annotated with a capability set, specifying all capabilities the block may use. For example, we can write a curried version of app_{Ξ} as follows.

$$\vdash \mathit{app}_C \doteq \{f^{\mathtt{Int}\Rightarrow 1} \Rightarrow \textbf{box}\ \{x^{\mathtt{Int}} \Rightarrow f(x)\}\} : (f : \mathtt{Int} \Rightarrow 1) \Rightarrow (\mathtt{Int} \Rightarrow 1\ \textbf{at}\ \{f\})$$

System C allows us to use term-level capability (block) variables in types, providing a lightweight form of dependent type. The name of the block variable f now also appears in the type as $f : \mathtt{Int} \Rightarrow 1$. This is essentially a binding for a capability variable f in the type. The return type is $\mathtt{Int} \Rightarrow 1\ \textbf{at}\ \{f\}$, a boxed block of type $\mathtt{Int} \Rightarrow 1$ annotated with the capability set $\{f\}$ (as the return block invokes f). The **box** construct allows a block to be boxed as a first-class value (much like **mod** in Metn).

To encode app_C in Metn, we cannot simply ignore the binding and usage of capability variables appearing in types. However, Metn does not allow us to use a term variable f in types. We can solve this problem by introducing an effect variable f^* for it. The encoding of app_C is as follows.

$$\begin{aligned}\vdash [\![\mathit{app}_C]\!] \doteq\ & \Lambda f^*.\textbf{mod}_{\langle| f^*\rangle}\ (\lambda f^{[f^*](\mathtt{Int}\to 1)}.\textbf{let mod}_{[f^*]}\ f = f\ \textbf{in}\ \textbf{mod}_{[f^*]}\ (\lambda x.f\ x)) \\ : \ & \forall f^*.\langle| f^*\rangle([f^*](\mathtt{Int} \to 1) \to [f^*](\mathtt{Int} \to 1))\ @\ \cdot\end{aligned}$$

The type of the argument f is wrapped in an absolute modality $[f^*]$. The effect variable f^* plays the role of representing the term variable f at the level of types. This is illustrated by the invocation $f\ x$ in the return function, which requires f^* to be present in the effect context specified by the absolute modality $[f^*]$. We translate **box** into a modality introduction. The extension modality $\langle| f^*\rangle$ is necessary here as System C always assumes blocks may invoke block parameters directly.

[1]Technically we can omit the operation label yield in the handler clause. We keep it to be consistent with other calculi.

As with the transition from System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ to System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, we can now give a different translation of the named handler $\mathit{sum42}_{\Xi}$ which uses absolute handlers and effect variables in Metn. We write $\mathit{sum42}_C$ for $\mathit{sum42}_{\Xi}$ interpreted in System C.

$$\vdash [\![\mathit{sum42}_C]\!] \doteq \mathbf{handle}^{\spadesuit}_{f^*}\ (\mathbf{let\ mod}_{[f^*]}\ f = \mathbf{mod}_{[f^*]}\ (\lambda x^{\mathtt{Int}}.\mathbf{do}_{f^*}\ x)\ \mathbf{in}\ f\ 42)\ \mathbf{with}\ H_{\mathsf{Gen}} : \mathtt{Int}\ @\ \cdot$$

For the function f used to simulate the capability *Yield*, we just use the handler name f^* in Metn instead of introducing a new effect variable. As with the translation to System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, absolute handlers are necessary in order to give the continuation function a sufficiently precise type.

We present System C in Section 5.3 and encode it into Metn with effect variables in Section 6.2.

3 A Core Calculus with Modal Effect Types and Named Handlers

Metn is a non-trivial variant of Met [26] with named handlers. Metn is completely backward compatible to Met; we provide a full specification of Metn with unnamed handlers and all extensions in Appendix A and prove its type soundness in Appendix B. Our presentation follows that of Met. We refer to Tang et al. [26] for a deeper introduction to modal effect types and Kavvos and Gratzer [15] for more details on simply-typed multimodal type theory, the foundation of Metn.

3.1 Syntax

The syntax of Metn is as follows. We highlight syntax relevant to modal effect types (same as in Met) in orange and syntax relevant to named handlers in grey.

$$
\begin{array}{lrcl}
\text{Types} & A, B & ::= & 1 \mid A \to B \mid \mu A \mid \forall a^{\ell}.A \\
\text{Masks} & L & ::= & \cdot \\
\text{Extensions} & D & ::= & \cdot \mid a, D \\
\text{Effect Contexts} & E, F & ::= & \cdot \mid a, E \\
\text{Modalities} & \mu & ::= & [E] \mid \langle L | D \rangle \\
\text{Kinds} & K & ::= & \mathsf{Abs} \mid \mathsf{Any} \\
\text{Label Contexts} & \Sigma & ::= & \cdot \mid \Sigma, \ell : \overline{\{\mathtt{op} : A \twoheadrightarrow B\}} \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, \blacksquare_{\mu_F} \mid \Gamma, x :_{\mu_F} A \mid \Gamma, a : \ell
\end{array}
$$

$$
\begin{array}{lrcl}
\text{Terms} & M, N & ::= & () \mid x \mid \lambda x^A.M \mid M\,N \mid \mathbf{mod}_{\mu}\,V \\
& & \mid & \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ x = V\ \mathbf{in}\ M \\
& & \mid & \Lambda a^{\ell}.V \mid M\,a \mid \mathbf{do}_a\ \mathtt{op}\ M \\
& & \mid & \mathbf{handle}_a\ M\ \mathbf{with}\ H \\
\text{Values} & V, W & ::= & () \mid x \mid \lambda x^A.M \mid \mathbf{mod}_{\mu}\,V \mid V\ a \\
& & \mid & \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ x = V\ \mathbf{in}\ W \mid \Lambda a^{\ell}.V \\
\text{Handlers} & H & ::= & \{\mathbf{return}\ x \mapsto M\} \\
& & \mid & H \uplus \{\mathtt{op}\ p\ r \mapsto M\}
\end{array}
$$

Different from Met in Tang et al. [26], we group operations op into effects ℓ and assume a global context Σ that defines all effect labels. This makes Metn more consistent with other calculi we present later. We write $\Sigma(\ell)$ for the set of operations that ℓ provides. Each operation op is associated with an arrow of the form $A \twoheadrightarrow B$, indicating that the operation takes an argument of type A and returns a value of type B. Operation names uniquely determine which effects they belong to.

Our syntax of values include name application and let-style unboxing when components are also values, following the notion of *complex values* in call-by-push-value [20].

3.2 Effect Contexts, Extensions, and Masks

Met treats effect contexts as scoped rows [18, 25]. With named handlers, effect contexts E are simply sets of handler names a: neither order nor duplicates matter. We define sub-effecting as set inclusion. Both equivalence $\Gamma \vdash E \equiv F$ and sub-effecting $\Gamma \vdash E \leqslant F$ are indexed by a context Γ for well-scopedness of handler names. Extensions D are sets of handler names (like effect contexts) and masks L are always empty. We will extend the syntax when we consider effect variables and masking handler names in Section 4. We define addition $D + E = D \cup E$ as join and subtraction $E - L = E \backslash L$ as set difference. Similarly, we define $L - D = L \backslash D$ and $D - L = D \backslash L$.

3.3 Modalities

Modalities manipulate effect contexts as follows.

$$[E](F) \;=\; E \qquad\qquad \langle L|D\rangle(F) \;=\; D + (F - L)$$

The absolute modality $[E]$ completely replaces the effect context F with E. The relative modality $\langle L|D\rangle$ changes the effect context F locally by specifying the masking L and extension D to it.

Following Met, we write μ_F for the pair of μ and F where F is the effect context that μ manipulates.

Modality Composition. We define the composition of modalities as follows.

$$\begin{array}{rcrcl} \mu & \circ & [E] & = & [E] \\ [E] & \circ & \langle L|D\rangle & = & [D + (E - L)] \\ \langle L_1|D_1\rangle & \circ & \langle L_2|D_2\rangle & = & \langle L_1 + (L_2 - D_1)|D_2 + (D_1 - L_2)\rangle \end{array}$$

The composition reads from left to right. First, an absolute modality fully determines the new effect context E no matter what μ does before. Second, setting the effect context to E followed by manipulating E with $\langle L|D\rangle$ is equivalent to directly setting the effect context to $D + (E - L)$. Third, consecutive masking and extending can be combined into one by cancelling their overlaps.

Composition is well-defined as we have $(\mu \circ \nu)(E) \;=\; \nu(\mu(E))$. We also have associativity $(\mu \circ \nu) \circ \xi = \mu \circ (\nu \circ \xi)$ and identity $\langle\rangle$.

Modality Transformation. Modality transformation tells us when one modality can be coerced into another. Given a value of modal type μA under some effect context F, we can coerce its modality to ν if the modality transformation relation $\Gamma \vdash \mu \Rightarrow \nu \mathrel{@} F$ holds. We define modality transformation as the transitive closure of following rules, ignoring masks for now.

$$\text{MT-Abs}\ \frac{\mu(F) = E' \qquad \Gamma \vdash E \leqslant E'}{\Gamma \vdash [E] \Rightarrow \mu \mathrel{@} F} \qquad\qquad \text{MT-Extend}\ \frac{\Gamma \vdash D \leqslant D' + F}{\Gamma \vdash \langle|D\rangle \Rightarrow \langle|D'\rangle \mathrel{@} F}$$

Rule MT-Abs allows us to transform an absolute modality to any other modality as long as all handler names in E are still in E'. Rule MT-Extend allows us to transform a relative modality $\langle|D\rangle$ to another relative modality $\langle|D'\rangle$ as long as D' and F cover all handler names in D.

The following lemma shows the soundness of modality transformation in the sense that we do not leak any effects no matter how the ambient effect context F is upcast.

Lemma 3.1 (Soundness of modality transformation). *For modality transformation $\Gamma \vdash \mu \Rightarrow \nu \mathrel{@} F$, we have $\mu(F') \leqslant \nu(F')$ for all F' with $F \leqslant F'$.*

3.4 Kinding and Contexts

We have two kinds for values where Abs is a sub-kind of Any. A type has kind Abs if any function type in it appears inside some absolute modality. For example, $1 \to 1$ does not have kind Abs while $[](1 \to 1)$ does. For any operation $\mathtt{op} : A \twoheadrightarrow B$ in the label context Σ, types A and B should have kind Abs to avoid effect leakage following Tang et al. [26].

Contexts are ordered. We have $\Gamma \mathrel{@} E$ if the context Γ is well-formed at the effect context E. For instance, the following context is well-formed at the effect context E.

$$x :_{\mu_F} A_1, y :_{\nu_F} A_2, \text{🔒}_{[E]_F}, z :_{\xi_E} A_3, w : A_4 \mathrel{@} E$$

Let us read from left to right. Variable w is at effect context E (it is technically tagged with an identity modality which is omitted). Variable z is tagged with modality ξ_E, which means it is not at effect context E but actually at effect context $\xi(E)$. Lock $\text{🔒}_{[E]_F}$ changes the effect context to E. We go back to F. Variables x and y are at effect contexts $\mu(F)$ and $\nu(F)$, respectively.

Each modality in the context carry an index of the effect context it manipulates, making the switching of effect contexts clear. We omit this index when it is obvious.

Formal definitions of kinding and context well-formedness rules are in Appendix A.2. We define $\mathsf{locks}(-)$ to compose all the modalities on the locks in a context.

$$\mathsf{locks}(\cdot) = \mathbb{1} \qquad \mathsf{locks}(\Gamma, \text{🔒}_{\mu_F}) = \mathsf{locks}(\Gamma) \circ \mu \qquad \mathsf{locks}(\Gamma, x :_{\mu_F} A) = \mathsf{locks}(\Gamma)$$

We identify contexts up to the following two equations.

$$\Gamma, \text{🔒}_{\mathbb{1}_E} \ @\ E = \Gamma\ @\ E \qquad\qquad \Gamma, \text{🔒}_{\mu_F}, \text{🔒}_{\nu_{F'}}\ @\ E = \Gamma, \text{🔒}_{(\mu\circ\nu)_F}\ @\ E$$

3.5 Typing

Figure 1 gives the typing rules for Metn. As before, we highlight rules relevant to modal effect types in orange and rules relevant to named handlers in grey. The typing judgement $\Gamma \vdash M : A\ @\ E$ means that the term M has type A under context Γ and effect context E with $\Gamma\ @\ E$.

$$\boxed{\Gamma \vdash (\mu, A) \Rightarrow \nu\ @\ F} \qquad \frac{\Gamma \vdash A : \mathsf{Abs}}{\Gamma \vdash (\mu, A) \Rightarrow \nu\ @\ F} \qquad \frac{\Gamma \vdash \mu \Rightarrow \nu\ @\ F}{\Gamma \vdash (\mu, A) \Rightarrow \nu\ @\ F}$$

$$\boxed{\Gamma \vdash M : A\ @\ E}$$

$$\textsc{T-Var}\ \frac{\Gamma \vdash (\mu, A) \Rightarrow \mathsf{locks}(\Gamma')\ @\ F}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash x : A\ @\ E} \qquad \textsc{T-Mod}\ \frac{\Gamma, \text{🔒}_{\mu_F} \vdash V : A\ @\ \mu(F)}{\Gamma \vdash \mathbf{mod}_\mu\ V : \mu A\ @\ F}$$

$$\textsc{T-LetMod}\ \frac{\Gamma, \text{🔒}_{\nu_F} \vdash V : \mu A\ @\ \nu(F) \qquad \Gamma, x :_{(\nu\circ\mu)_F} A \vdash M : B\ @\ F}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = V\ \mathbf{in}\ M : B\ @\ F}$$

$$\textsc{T-Abs}\ \frac{\Gamma, x : A \vdash M : B\ @\ E}{\Gamma \vdash \lambda x^A.M : A \to B\ @\ E} \qquad \textsc{T-App}\ \frac{\Gamma \vdash M : A \to B\ @\ E \qquad \Gamma \vdash N : A\ @\ E}{\Gamma \vdash M\ N : B\ @\ E}$$

$$\textsc{T-NAbs}\ \frac{\Gamma, a : \ell \vdash V : A\ @\ E}{\Gamma \vdash \Lambda a^\ell.V : \forall a^\ell.A\ @\ E} \qquad \textsc{T-NApp}\ \frac{\Gamma \vdash M : \forall a^\ell.A\ @\ E \qquad \Gamma \ni b : \ell}{\Gamma \vdash M\ b : A[b/a]\ @\ E}$$

$$\textsc{T-DoName}\ \frac{\Gamma \ni a : \ell \qquad \Sigma(\ell) \ni \mathsf{op} : A \twoheadrightarrow B \qquad \Gamma \vdash N : A\ @\ a, E}{\Gamma \vdash \mathbf{do}_a\ \mathsf{op}\ N : B\ @\ a, E}$$

$$\textsc{T-HandleName}\ \frac{\begin{array}{c} H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i \qquad \Gamma \vdash A : \mathsf{Abs} \\ \Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, a : \ell, \text{🔒}_{\langle a \rangle_E} \vdash M : A\ @\ a, E \\ \Gamma, x : A \vdash N : B\ @\ E \qquad [\Gamma, p : A_i, r : B_i \to B \vdash N : B\ @\ E]_i \end{array}}{\Gamma \vdash \mathbf{handle}_a\ M\ \mathbf{with}\ H : B\ @\ E}$$

Fig. 1. Typing rules and auxiliary rules for Metn.

Modality Introduction and Elimination. Rule T-Mod introduces a modality μ to the conclusion and puts a lock into the context of the premise as well as changes the effect context. Rule T-LetMod eliminates a modality μ and moves it to the variable binding. We have seen examples of them in Section 2.1. There is another modality ν in T-LetMod which is needed for technical reason to support sequential unboxing. For instance, given a variable $x : \nu\mu A$ with two modalities, to unbox both ν and μ, we can first unbox it to $y :_\nu \mu A$ and then to $z :_{\nu\circ\mu} A$ as follows.

$$\mathbf{let\ mod}_\nu\ y = x\ \mathbf{in\ let}_\nu\ \mathbf{mod}_\mu\ z = y\ \mathbf{in}\ M$$

We restrict **mod** and $\mathbf{let}_\nu\ \mathbf{mod}_\mu$ to values to avoid effect leakage as in Met [26].

Accessing Variables. Locks control the accessibility of variables. Rule T-Var uses the auxiliary judgement $\Gamma \vdash (\mu, A) \Rightarrow \mathsf{locks}(\Gamma')\ @\ F$ to check whether we can access a variable $x :_{\mu_F} A$ given all locks in Γ'. When A has kind Abs, we can always use x as it does not depend on the effect context. Otherwise, we need to make sure the coercion from μ to $\mathsf{locks}(\Gamma')$ is safe by checking the modality transformation relation $\Gamma \vdash \mu \Rightarrow \mathsf{locks}(\Gamma')\ @\ F$. For instance, $a : \ell, b : \ell, x :_{\langle\!| a \rangle} 1 \to 1, \text{🔒}_{[b]} \vdash x : 1 \to 1\ @\ b$ is ill-typed since we cannot transform $\langle\!| a \rangle$ to $[b]$.

Named Handlers. Rule T-NAbs introduces a name abstraction and rule T-NApp eliminates a name abstraction. They are standard. The T-DoName rule invokes an operation op with a handler name a, requiring the name a to be in the effect context. Rule T-HandleName defines a named handler and uses it to handle a computation M. In the typing judgement of M, we not only put the name binding $a : \ell$ in the context, but also put a lock with a relative modality $\langle\!| a \rangle$ which extends the ambient effect context E with a. For the return value A of M, we require it to have kind Abs to avoid leaking effects as discussed in Section 2.2. We will relax this restriction when we consider masking handler names in Section 4.4. The well-scopedness of A under context Γ makes sure that name a cannot appear in A freely. The other parts of the handler rule are standard.

3.6 Syntactic Sugar

We define some syntactic sugar which we use to simplify our encodings in Sections 6 and 8.

$$
\begin{array}{rcl}
\mathbf{mod}_{\mu;\nu}\ V & \doteq & \mathbf{mod}_{\mu}\ (\mathbf{mod}_{\nu}\ V) \\
\mathbf{let}\ x = M\ \mathbf{in}\ N & \doteq & (\lambda x.N)\ M \\
\mathbf{let}\ \mathbf{mod}_{\mu}\ = M\ \mathbf{in}\ N & \doteq & (\lambda x.\mathbf{let}\ \mathbf{mod}_{\mu}\ x = x\ \mathbf{in}\ N)\ M \\
\mathbf{let}\ \mathbf{mod}_{\mu;\nu}\ x = M\ \mathbf{in}\ N & \doteq & \mathbf{let}\ \mathbf{mod}_{\mu}\ x = M\ \mathbf{in}\ \mathbf{let}_{\mu}\ \mathbf{mod}_{\nu}\ x = x\ \mathbf{in}\ N
\end{array}
$$

3.7 Operational Semantics

Our semantics for named handlers follows the generative semantics of Biernacki et al. [2]. Handler instances h are dynamically generated to replace handler names. We manage them in an instance context defined as $\Omega ::= \cdot \mid \Omega, h : \ell$. We extend syntax to allow using handler instances h. Moreover, since values V can reduce, we define value normal forms U which cannot reduce further. The definitions for all new syntax and the operational semantics are given in Figure 2. The semantics is pretty standard. We write $H \propto \ell$ if handler H handles all operations in effect ℓ, i.e., $H = \{\overline{\mathsf{op}\ p\ r \mapsto N}\}$ and $\Sigma(\ell) = \{\overline{\mathsf{op} : A \twoheadrightarrow B}\}$. The reduction relation has the form $M \mid \Omega \rightsquigarrow N \mid \Omega$. We omit Ω when it is not used and updated. Only E-Gen extends Ω. Typing judgements are also extended to $\Omega \mid \Gamma \vdash M : A\ @\ E$ for runtime terms.

3.8 Type Soundness and Effect Safety

To state syntactic type soundness, we first define normal forms.

Definition 3.2 (Normal Forms). We say a term M is in a normal form with respect to effect context E, if it is either in a value normal form $M = U$ or of form $M = \mathcal{E}[\mathbf{do}_h\ \mathsf{op}\ U]$ for $h \in E$.

We have the following theorems which in together give type soundness and effect safety.

Theorem 3.3 (Progress). *If $\Omega \mid \cdot \vdash M : A\ @\ E$, then either there exists N such that $M \mid \Omega \rightsquigarrow N \mid \Omega'$ or M is in a normal form with respect to E.*

Theorem 3.4 (Subject Reduction). *If $\Omega \mid \Gamma \vdash M : A\ @\ E$ and $M \mid \Omega \rightsquigarrow N \mid \Omega'$, then $\Omega' \mid \Gamma \vdash N : A\ @\ E$.*

Terms	$M ::= \cdots \mid M\ h \mid \mathbf{do}_h\ \mathsf{op}\ M \mid \mathbf{handle}_h\ M\ \mathbf{with}\ H$
Value normal forms	$U ::= x \mid \lambda x^A.M \mid \lambda a^\ell.V \mid \mathbf{mod}_\mu\ U$
Evaluation Contexts	$\mathcal{E} ::= [\ ] \mid \mathcal{E}\ N \mid U\ \mathcal{E} \mid \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = \mathcal{E}\ \mathbf{in}\ M$
	$\mid\ \mathcal{E}\ a \mid \mathcal{E}\ h \mid \mathbf{do}_h\ \mathsf{op}\ \mathcal{E} \mid \mathbf{handle}_h\ \mathcal{E}\ \mathbf{with}\ H$

E-App	$(\lambda x^A.M)\ U \rightsquigarrow M[U/x]$	
E-NApp	$(\lambda a^\ell.U)\ h \rightsquigarrow U[h/a]$	
E-LetMod	$\mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = \mathbf{mod}_\mu\ U\ \mathbf{in}\ M \rightsquigarrow M[U/x]$	
E-Gen	$\mathbf{handle}_a\ M\ \mathbf{with}\ H \mid \Omega \rightsquigarrow \mathbf{handle}_h\ M[h/a]\ \mathbf{with}\ H \mid \Omega, h : \ell$	where h fresh and $H \propto \ell$
E-NRet	$\mathbf{handle}_h\ U\ \mathbf{with}\ H \rightsquigarrow N[U/x]$	where $(\mathbf{return}\ x \mapsto N) \in H$
E-NOp	$\mathbf{handle}_h\ \mathcal{E}[\mathbf{do}_h\ \mathsf{op}\ U]\ \mathbf{with}\ H \rightsquigarrow N[U/p, (\lambda y.\mathbf{handle}_h\ \mathcal{E}[y]\ \mathbf{with}\ H)/r]$	where $(\mathsf{op}\ p\ r \mapsto N) \in H$
E-Lift	$\mathcal{E}[M] \rightsquigarrow \mathcal{E}[N]$	if $M \rightsquigarrow N$

Fig. 2. Operational semantics and runtime constructs for Metn.

4 Extensions to Metn

We introduce four extensions to Metn: moving modalities inside name abstractions, absolute handlers, effect variables, and masking handler names. Our specification and proofs in Appendices A and B cover all these extensions in together.

4.1 Moving Modalities inside Name Abstractions

Similar to commuting data constructors and type abstractions with modalities in Met, it is also sound and useful to commute name abstractions with modalities. However, a function of type $\forall a^\ell.\mu A \to \mu(\forall a^\ell.A)$ where $a \notin \mu$, is not directly expressible in Metn as before using $\mathbf{let\ mod}_\mu$ we must first instantiate the name abstraction. This is analogous to why in System F with sum types we cannot define a function of type $(\forall\alpha.A+B) \to (\forall\alpha.A)+(\forall\alpha.B)$. To support moving name abstraction inside modalities, we extend modality elimination to the following form where new parts compared to T-LetMod are highlighted.

$$\text{T-LetMod'}\ \frac{\Gamma, \text{🔒}_{\nu_F}, \overline{a : \ell} \vdash V : \mu A\ @\ \nu(F) \qquad \Gamma, x :_{(\nu\circ\mu)_F} \forall\overline{a^\ell}.A \vdash M : B\ @\ F}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \lambda\overline{a^\ell}.\,x = V\ \mathbf{in}\ M : B\ @\ F}$$

The value V can use additional names $\overline{a : \ell}$ which are bound in the type of x. Now we can write a function $\lambda x^{\forall a^\ell.\mu A}.\mathbf{let\ mod}_\mu\ \lambda a^\ell.y = x\ a\ \mathbf{in\ mod}_\mu\ y : \forall a^\ell.\mu A \to \mu(\forall a^\ell.A)$ where $a \notin \mu$.

4.2 Absolute Handlers

In rule T-HandleName, the continuation r is only given a function type $B_i \to B$, which means that the effects it may use depends on the ambient effect context E. As shown in Section 2.1, in some cases we can actually wrap the continuation function into an absolute modality. We call such handlers *absolute* handlers. Its typing rule is as follows.

$$\text{T-HandleName}^\spadesuit\ \frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, a : \ell, \text{🔒}_{[a,E]_F} \vdash M : A\ @\ a, E \qquad \Gamma \vdash A : \mathsf{Abs} \qquad \Gamma \vdash E \leqslant F \\ \Gamma, \text{🔒}_{[E]_F}, x : [E]A \vdash N : B\ @\ E \qquad [\Gamma, \text{🔒}_{[E]_F}, p_i : A_i, r_i : [E](B_i \to B) \vdash N_i : B\ @\ E]_i\end{array}}{\Gamma \vdash \mathbf{handle}^\spadesuit_a\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i : B\ @\ F}$$

This rule puts an absolute lock $\blacksquare_{[a,E]_F}$ to the context for M and wraps the continuation r in the absolute modality $[E]$. It also puts the lock $\blacksquare_{[E]_F}$ in the contexts for handler clauses as deep handlers capture themselves into continuations. These locks guarantee that r cannot use any effects not in E.

4.3 Effect Variables

The extension of effect variables to Metn is standard. The syntax and typing rules are as follows.

$$\begin{array}{llll} \text{Types} & A, B ::= \cdots \mid \forall \varepsilon.A & \text{Terms} \; M, N ::= \cdots \mid \Lambda\varepsilon.V \mid M\,E & \text{Contexts} \; \Gamma ::= \cdots \mid \Gamma, \varepsilon \\ \text{Effect Contexts} & E ::= \cdots \mid \varepsilon, E & \text{Values} \; V, W ::= \cdots \mid \Lambda\varepsilon.V \mid V\,E & \end{array}$$

$$\text{T-EAbs}\;\frac{\Gamma, \varepsilon \vdash V : A \;@\; E}{\Gamma \vdash \Lambda\varepsilon.V : \forall\varepsilon.A \;@\; E} \qquad \text{T-EApp}\;\frac{\Gamma \vdash M : \forall\varepsilon.A \;@\; E \qquad \Gamma \vdash F}{\Gamma \vdash M\,F : A[F/\varepsilon] \;@\; E}$$

Effect contexts can contain effect variables. The equivalence and subeffecting relations on effect contexts remain set equivalence and set inclusion. An effect variable is equivalent only to itself.

4.4 Masking Handler Names

Allowing masks to contain handler names unifies the syntax of masks and extensions. We keep effect contexts separate for consistency with Section 4.3. The new syntax is as follows.

$$\begin{array}{llll} \text{Terms} & M, N ::= \cdots \mid \mathbf{mask}_L\; M & \text{Masks and Extensions} & L, D ::= \cdot \mid a, D \mid D \backslash L \\ \text{Name Kinds} & R ::= \cdot \mid \sharp & \text{Effect Contexts} & E ::= \cdot \mid a, E \mid E \backslash L \\ \text{Contexts} & \Gamma ::= \cdots \mid \Gamma, a :_R \ell & & \end{array}$$

The term syntax $\mathbf{mask}_L\; M$ masks handler names in L from the ambient effect context. We extend the syntax of effect contexts with difference $E \backslash L$. This new syntax is crucial for ensuring that effect contexts are closed under the operation $E - L$, as it is not always possible to tell whether handler names introduced by name abstractions are equivalent or not as they may either be instantiated to the same or different names. The syntax of masks and extensions is also extended with $D \backslash L$.

There are some cases where we can tell that two handler names are different. First, handler names must be different if they have different labels. Second, named handlers always introduce unique names. For a handler name introduced by a handler, it must be different from all other names introduced earlier. We use $\sharp$ to annotate those handler names introduced by handlers. We define the relation $\Gamma \vdash a \not\equiv b$ as the symmetric closure of the following two rules.

$$\text{NeqLabel}\;\frac{\ell \neq \ell'}{\Gamma_0, b :_{R'} \ell', \Gamma_1, a :_R \ell, \Gamma_2 \vdash a \not\equiv b} \qquad \text{NeqOrder}\;\frac{}{\Gamma_0, b :_R \ell, \Gamma_1, a :_\sharp \ell, \Gamma_2 \vdash a \not\equiv b}$$

We use boolean algebras to define the equivalence and subeffecting relations for effect contexts. Given a context Γ, we define $\mathbb{B}(\Gamma)$ as the boolean algebra given by the power set of the set of all names in Γ. We have the standard union ($\cup$), intersection ($\cap$), and set complement ($\neg$) operations. We write equivalence over $\mathbb{B}(\Gamma)$ as $\equiv_{\mathbb{B}(\Gamma)}$. We extend $\mathbb{B}(\Gamma)$ with an extra axiom that $a \cap b \equiv_{\mathbb{B}(\Gamma)} \varnothing$ if $\Gamma \vdash a \not\equiv b$. Each effect context E well-scoped in Γ corresponds to an element in $\mathbb{B}(\Gamma)$; as standard we define $E \backslash F = E \cap (\neg F)$. We formally define the equivalence and subeffecting relations as follows.

$$\frac{E \equiv_{\mathbb{B}(\Gamma)} F}{\Gamma \vdash E \equiv F} \qquad \frac{E \cap (\neg F) \equiv_{\mathbb{B}(\Gamma)} \emptyset}{\Gamma \vdash E \leqslant F}$$

Modalities remain unchanged except for the modality transformation relation, where we replace the old MT-Extend rule with the following two rules.

$$\frac{\text{MT-ExtendRemove}\quad \Gamma \vdash L' \leqslant L \qquad \Gamma \vdash D \leqslant D' + (F - L')}{\Gamma \vdash \langle L|D\rangle \Rightarrow \langle L'|D'\rangle \ @\ F} \qquad \frac{\text{MT-ExpandShrink}\quad \Gamma \vdash (F - L) \equiv D', E}{\Gamma \vdash \langle L|D\rangle \Leftrightarrow \langle D', L|D, D'\rangle \ @\ F}$$

Rule MT-ExtendRemove allows us to remove names from masks and add more names to extensions. Rule MT-ExpandShrink is bidirectional and also appears in Met. It allows us to simultaneously expand or shrink extensions and masks with the same set of names.

For typing, we add a new rule for masking and update the handler rule.

$$\frac{\text{T-Mask}\quad \Gamma, \blacksquare_{\langle L \rangle_E} \vdash M : A \ @\ E - L}{\Gamma \vdash \mathbf{mask}_L\ M : \langle L \rangle A \ @\ E} \qquad \frac{\text{T-HandleName'}\quad \Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \quad \Gamma, a :_\sharp \ell, \blacksquare_{\langle a \rangle_E} \vdash M : \langle a \rangle A \ @\ a, E \quad \Gamma, x : A \vdash N : B \ @\ E \quad [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N : B \ @\ E]_i}{\Gamma \vdash \mathbf{handle}_a\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \ @\ E}$$

Rule T-Mask removes names L from the ambient effect context. Rule T-HandleName annotates the name binding a in the context with $\sharp$ to indicate that it is a name introduced by a handler. Consequently, we know that name a must be different from all other names in context Γ. Moreover, instead of requiring the return type A of the handled computation to have kind Abs, rule T-HandleName uses a relative modality $\langle a \rangle$ to remove name a. The new T-HandleName' rule subsumes the previous one in Section 3.5 as we can always mask an absolute value freely.

For semantics, we update the E-NRet rule to the following one which takes a masked value, removes its mask, and substitute it into the return clause.

$$\text{E-NRet'}\quad \mathbf{handle}_h\ (\mathbf{mod}_{\langle h \rangle}\ V)\ \mathbf{with}\ H \rightsquigarrow N[V/x], \text{where } (\mathbf{return}\ x \mapsto N) \in H$$

5 Capability-Based Effect Systems a la Effekt

In this section we briefly present two core calculi for the language Effekt [6] with capability-based effect systems and named handlers: System Ξ [5], which only supports second-class functions, and System C [4], which recovers first-class functions via boxes. We refer to the original papers for more details on System Ξ and System C.

5.1 Simplifications

For simplicity and uniformity, for calculi in this section and in Section 7 as well as encodings in Sections 6 and 8, we assume that each effect label ℓ only contains one operation and omit return clauses of handlers. As in Metn, we also assume a global label context Σ. In summary, all calculi share the definitions of following two syntactic categories.

$$\text{Handlers}\quad H ::= \{\mathsf{op}\ p\ r \mapsto N\} \qquad \text{Label Contexts}\quad \Sigma ::= \cdot \mid \Sigma, \ell : \{\mathsf{op} : A \twoheadrightarrow B\}$$

For an effect $\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\}$, we write A_{op} or A_ℓ for type A, and B_{op} or B_ℓ for type B. As there is only one operation in each effect, we omit the operation name when invoking operations.

When referring to the name of a typing or operational semantics rule, we sometimes also mention the calculus name to disambiguate. For instance, T-Var-Metn refers to the rule T-Var in Metn.

5.2 System Ξ

The syntax and typing rules of System Ξ are given in Figure 3. System Ξ is fine-grain call-by-value [21] and distinguishes between first-class values V, second-class blocks G (i.e., functions), and computations M. We have three forms of judgements for them. Most of the rules are standard. Rule T-Block introduces a block $\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}$ which binds value variables $\overline{x : A}$ and block variables $\overline{f : T}$ in computation M. Rule T-Call fully applies a block P to values $\overline{V_i}$ and blocks $\overline{Q_j}$. Rule

$$\begin{array}{llll}
\text{Value Types} & A, B ::= 1 & \text{Computations} & M, N ::= \textbf{return}\ V \mid \textbf{let}\ x = M\ \textbf{in}\ N \mid \textbf{def}\ f = G\ \textbf{in}\ N \\
\text{Block Types} & T ::= (\overline{A}, \overline{T}) \Rightarrow B & & \qquad \mid P(\overline{V}, \overline{Q}) \mid \textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ H \\
\text{Values} & V, W ::= x \mid () & \text{Block Values} & P, Q ::= f \mid \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\} \\
\text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, f : T & \text{Blocks} & G ::= P
\end{array}$$

$\boxed{\Gamma \vdash V : A}$ $\boxed{\Gamma \vdash G : T}$ $\boxed{\Gamma \vdash M : A}$

$$\textsc{T-Unit}\ \frac{}{\Gamma \vdash () : 1} \qquad \textsc{T-Var}\ \frac{\Gamma \ni x : A}{\Gamma \vdash x : A} \qquad \textsc{T-BlockVar}\ \frac{\Gamma \ni f : T}{\Gamma \vdash f : T} \qquad \textsc{T-Block}\ \frac{\Gamma, \overline{x : A}, \overline{f : T} \vdash M : B}{\Gamma \vdash \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\} : (\overline{A}, \overline{T}) \Rightarrow B}$$

$$\textsc{T-Value}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \textbf{return}\ V : A} \qquad \textsc{T-Let}\ \frac{\Gamma \vdash M : A \qquad \Gamma, x : A \vdash N : B}{\Gamma \vdash \textbf{let}\ x = M\ \textbf{in}\ N : B} \qquad \textsc{T-Def}\ \frac{\Gamma \vdash G : T \qquad \Gamma, f : T \vdash N : B}{\Gamma \vdash \textbf{def}\ f = G\ \textbf{in}\ N : B}$$

$$\textsc{T-Call}\ \frac{\begin{array}{c}\Gamma \vdash P : (\overline{A_i}, \overline{T_j}) \Rightarrow B \\ \Gamma \vdash V_i : A_i \qquad \Gamma \vdash Q_j : T_j\end{array}}{\Gamma \vdash P(\overline{V_i}, \overline{Q_j}) : B} \qquad \textsc{T-Handle}\ \frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \\ \Gamma, f : (A') \Rightarrow B' \vdash M : A \qquad \Gamma, p : A', r : (B') \Rightarrow A \vdash N : A\end{array}}{\Gamma \vdash \textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ \{\mathsf{op}\ p\ r \mapsto N\} : A}$$

Fig. 3. Syntax and typing rules for System Ξ.

T-Let binds the return value of a computation and rule T-Def binds a block. Rule T-Handle defines a named handler which introduces a capability (block variable) f to the scope of M. Invocations of the operation op via calling f in M is handled by this handler. As we discussed in Section 2.5, System Ξ guarantees that the capability f cannot escape by restricting them to be second-class.

Similar to the operational semantics of Metn in Section 3.7, System Ξ also generates handler instances dynamically and has a reduction relation $M \mid \Omega \rightsquigarrow N \mid \Omega$. The most interesting reduction rule is E-Gen which uses a runtime capability value $\mathbf{cap}_h$ to substitute f.

$$\textsc{E-Gen} \quad \textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ H \mid \Omega \rightsquigarrow \textbf{handle}_h\ M[\mathbf{cap}_h/f]\ \textbf{with}\ H \mid \Omega, h : \ell \quad \text{where } h \text{ fresh}$$

The full specification of operational semantics can be found in Appendix C.1.

5.3 System C

Figure 4 gives the syntax and typing rules for System C. We highlight new syntax compared to System Ξ. In contexts, we have auxiliary delimiters $\llcorner \cdots \lrcorner$ and markers $\clubsuit_C$ that are only used for encodings in Section 6.2. Variables cannot commute with markers and delimiters.

We have three judgements for values, blocks, and computations individually. Judgements for blocks $\Gamma \vdash P : T \mid C$ and computations $\Gamma \vdash M : A \mid C$ now explicitly track capability sets C. The capability set C is a subset of block variables in Γ and represents those capabilities may be used. Rules T-BSub and T-Sub allow us to upcast the capability set to a larger one. Rule T-Box boxes a block P into a first-class value whose type T **at** C tracks the capability set C of the block. Rule T-Unbox unboxes a value into a block and moves the capability set from its type to the judgement.

There are two rules for block variables. For a *transparent* block variable binding $f :^C T$, we know this block may use capabilities from C. Rule T-Transparent exactly tracks the capability set C. For a *tracked* block variable binding $f :^* T$, we do not know which capabilities it may use. Rule T-Tracked tracks the block variable f itself as a capability. Rule T-Def binds a block G to a transparent block variable $f :^{C'} T$ where C' is the capability set of G. Rule T-Block binds a list of

$$\begin{array}{llll}
\text{Value Types} & A, B ::= 1 \mid T \textbf{ at } C & \text{Values} & V, W ::= x \mid () \mid \textbf{box } G \\
\text{Block Types} & T ::= (\overline{A}, \overline{f : T}) \Rightarrow B & \text{Blocks} & G ::= P \mid \textbf{unbox } V \\
\text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, f :^C T \mid \Gamma, \llcorner \overline{x : A}, \overline{f :^* T} \lrcorner \mid \Gamma, \clubsuit_C & \text{Capability Sets} & C ::= \{\overline{f}\}
\end{array}$$

$\boxed{\Gamma \vdash V : A}$ $\boxed{\Gamma \vdash G : T \mid C}$

$$\text{T-Box}\ \frac{\Gamma, \clubsuit \vdash G : T \mid C}{\Gamma \vdash \textbf{box } G : T \textbf{ at } C} \qquad \text{T-Transparent}\ \frac{\Gamma \ni f :^C T}{\Gamma \vdash f : T \mid C} \qquad \text{T-Tracked}\ \frac{\Gamma \ni f :^* T}{\Gamma \vdash f : T \mid \{f\}} \qquad \text{T-Unbox}\ \frac{\Gamma \vdash V : T \textbf{ at } C}{\Gamma \vdash \textbf{unbox } V : T \mid C}$$

$$\text{T-Block}\ \frac{\Gamma, \llcorner \overline{x : A}, \overline{f :^* T} \lrcorner \vdash M : B \mid C \cup \{\overline{f}\}}{\Gamma \vdash \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\} : (\overline{A}, \overline{f : T}) \Rightarrow B \mid C} \qquad \text{T-BSub}\ \frac{\Gamma \vdash G : T \mid C' \qquad C' \subseteq C}{\Gamma \vdash G : T \mid C}$$

$\boxed{\Gamma \vdash M : A \mid C}$

$$\text{T-Value}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \textbf{return } V : A \mid \cdot} \qquad \text{T-Let}\ \frac{\Gamma \vdash M : A \mid C \quad \Gamma, x : A \vdash N : B \mid C'}{\Gamma \vdash \textbf{let } x = M \textbf{ in } N : B \mid C \cup C'}$$

$$\text{T-Call}\ \frac{\Gamma \vdash P : (\overline{A_i}, \overline{f_j : T_j}) \Rightarrow B \mid C \qquad \overline{\Gamma \vdash V_i : A_i} \qquad \overline{\Gamma, \clubsuit_{C \cup \overline{C_j}} \vdash Q_j : T_j \mid C_j}}{\Gamma \vdash P(\overline{V}_i, \overline{Q}_j) : B[\overline{C_j / f_j}] \mid C \cup \overline{C_j}}$$

$$\text{T-Sub}\ \frac{\Gamma \vdash M : A \mid C' \quad C' \subseteq C}{\Gamma \vdash M : A \mid C} \qquad \text{T-Def}\ \frac{\Gamma, \clubsuit_C \vdash G : T \mid C' \quad \Gamma, f :^{C'} T \vdash M : A \mid C}{\Gamma \vdash \textbf{def } f = G \textbf{ in } M : A \mid C}$$

$$\text{T-Handle}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \quad \Gamma, \llcorner f :^* (A') \Rightarrow B' \lrcorner \vdash M : A \mid C \cup \{f\} \quad \Gamma, p : A', r :^C (B') \Rightarrow A \vdash N : A \mid C}{\Gamma \vdash \textbf{handle } \{f \Rightarrow M\} \textbf{ with } \{\mathsf{op}\ p\ r \mapsto N\} : A \mid C}$$

Fig. 4. Syntax and typing rules for System C (omitting unchanged parts from System Ξ).

tracked block variables $\overline{f :^* T}$ whose capability sets we do not know. We add all $\overline{f}$ to the capability set of the block body M as they may be used. The block type $(\overline{A}, \overline{f : T}) \Rightarrow B$ now contain names of block variables f which could appear in B, giving a lightweight form of dependent types.

Rule T-Call fully applies a block P. Compared to T-Call-System Ξ, it substitutes each block variable f_j with the capability set C_j in type B. The capability set of the call is the union of P and all its block arguments. Rule T-Handle handles a computation. Compared to T-Handle-System Ξ, it tracks the usage of the capability f by marking it as a tracked block variable and adds it to the capability set of M. The continuation r is transparent as we know it only uses capabilities in C.

The operational semantics of System C is given in Appendix C.1.

6 Encoding Capability-Based Effect Systems into Metn

We show how to encode System Ξ in Metn without effect variables and System C in Metn with effect variables. Full specifications and proofs can be found in Appendices C.2 and D.

6.1 Encoding System Ξ in Metn without Effect Variables

Figure 5 gives the encoding of System Ξ in Metn without effect variables. This encoding is straightforward and does not even use any modalities. We mostly just translate the syntax of second-class blocks in System Ξ to first-class functions in Metn. The only interesting case is named handlers **handle** $\{f \Rightarrow M\}$ **with** H, where we use the function $\lambda x^{[\![A_{\mathsf{op}}]\!]}.\mathbf{do}_a\ x$ to simulate

$$\begin{array}{rcl} [\![-]\!] & : & \text{Type} \to \text{Type} \\ [\![(\overline{A}, \overline{T}) \Rightarrow B]\!] & = & \overline{[\![A]\!]} \to \overline{[\![T]\!]} \to [\![B]\!] \end{array} \qquad \begin{array}{rcl} [\![-]\!] & : & \text{Block} \to \text{Term} \\ [\![\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}]\!] & = & \overline{\lambda x^{[\![A]\!]}}\, \overline{f^{[\![T]\!]}}.[\![M]\!] \end{array}$$

$$\begin{array}{rcl} [\![-]\!] & : & \text{Computation} \to \text{Term} \\ [\![P(\overline{V}, \overline{Q})]\!] & = & [\![P]\!]\ \overline{[\![V]\!]}\ \overline{[\![Q]\!]} \\ \left[\!\!\left[\begin{array}{l}\textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with} \\ \{\mathsf{op}\ p\ r \mapsto N\}\end{array}\right]\!\!\right] & = & \begin{array}{l}\textbf{handle}_a\ (\textbf{let}\ f = \lambda x^{[\![A_{\mathsf{op}}]\!]}.\textbf{do}_a\ x\ \textbf{in}\ [\![M]\!]) \\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \mathsf{op}\ p\ r \mapsto [\![N]\!]\}\end{array} \end{array}$$

Fig. 5. An encoding of System Ξ in METN without effect variables (omitting homomorphic cases).

the capability f. The following theorems state that the encoding preserves types and operational semantics.

THEOREM 6.1 (TYPE PRESERVATION). *If $\Gamma \vdash M : A$ in System Ξ, then $[\![\Gamma]\!] \vdash [\![M]\!] : [\![A]\!] \mathrel{@} \cdot$ in METN. Similarly for values and blocks.*

THEOREM 6.2 (SEMANTICS PRESERVATION). *If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System Ξ, then $[\![M]\!] \mid [\![\Omega]\!] \rightsquigarrow^* [\![N]\!] \mid [\![\Omega']\!]$ in METN.*

6.2 Encoding System C in METN with Effect Variables

In Section 5.3 we incorporate *absolute markers* $\clubsuit_C$ and *relative delimiters* $\llcorner \overline{x : A}, \overline{f :^* T} \lrcorner$ in contexts. The typing of System C does not rely on these markers, but our translation does. Thus we need to ensure that markers in the context are well-formed in order to define the translation.

Definition 6.3 (Well-formed). A context Γ in System C is well-formed if every block variable binding of form $f :^* T$ in Γ appears within a pair of relative delimiters $\llcorner\,\lrcorner$ and there are no nested delimiters. A typing judgement in System C is well-formed if its context is well-formed.

Without loss of expressiveness, we consider only well-formed judgements. Figure 6 encodes System C in METN with effect variables. The term translation is typed-directed and essentially defined on typing judgements in System C. Rather than writing out the full judgement, we annotate components of a term with their types and capability sets as necessary.

We encode boxes in System C as absolute modalities in METN, as they both specify which effects the program inside can use. A boxed block type T **at** C with capability set C is translated to a modal type $[[\![C]\!]][\![T]\!]$. Correspondingly, on the term level, we translate **box** into modality introduction and **unbox** into modality elimination. The translation of blocks of type $(A, \overline{f : T}) \Rightarrow B$ is more involved because System C allows block variables $f : T$ to appear in types. As shown in Section 2.6, we promote a term variable f to the type level in METN by introducing an effect variable f^* for it. The type T for a block argument $f : T$ is translated to the modal type $[f^*][\![T]\!]$, which ensures that whenever we want to invoke the argument f, the effect variable f^* must be in the effect context. Consequently, when a block variable f is captured in a box in System C, its associated effect variable f^* must be captured in an absolute modality in METN. System C always allows a block to use all its block arguments $\overline{f}$. To encode this, we use a relative modality $\langle \overline{f^*} \rangle$ to extend the effect context with the effect variables associated to arguments $\overline{f}$. Contrary to the translation of block construction, the translation of block call $P(\overline{V}, \overline{Q})$ instantiates effect variables, eliminates the relative modality, and applies the block to its arguments. Block arguments $\overline{Q}$ are wrapped into absolute modalities of their capability sets.

For every block variable binding, such as f in $\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}$ or **def** $f = G$ **in** N, we always immediately eliminate its top-level modality to simplify its later use. Each use of a block variable f is translated to $\hat{f}$, whose modality has been eliminated. The translation of contexts also reflects this principle. For a transparent block variable $f :^{C} T$, we know it uses capabilities in C. Thus we translate it into $f : [C]\llbracket T \rrbracket$ and eliminate its modality as $\hat{f} :_{[C]} \llbracket T \rrbracket$. For a tracked block variable $f :^{*} T$, we do not know which capabilities it may use. Thus we introduce a fresh effect variable f^{*} for it, translate it into $f : [f^{*}]\llbracket T \rrbracket$, and eliminate its modality as $\hat{f} :_{[f^{*}]} \llbracket T \rrbracket$. The translation of contexts is indexed by the current capability set and our markers track the changes of the capability set. An absolute marker $\clubsuit_{C}$ indicates a complete change and we translate it into a lock with an absolute modality. A relative delimiter $\llcorner \overline{x : A}, \overline{f :^{*} T} \lrcorner$ indicates that capabilities f are extended to the capability set, and we translate it into a lock with a relative modality.

The translation of handlers follows our example $sum42_{C}$ in Section 2.6, where we omit the return clause and the separate binding of f. In the return clause, we need to eliminate the modality of x because absolute handlers wrap the return value with an absolute modality as in Section 4.2. Similarly, in the op clause, we need to eliminate the modality of the continuation function r.

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Cap Set} \to \text{Effect Context} \\
\llbracket \{\overline{f}\} \rrbracket &= \overline{f^{*}}
\end{aligned}
\qquad
\begin{aligned}
\llbracket - \rrbracket_{-} &: \text{Context} \times \text{Cap Set} \to \text{Context} \\
\llbracket \Gamma, \llcorner \overline{x : A}, \overline{f :^{*} T} \lrcorner \rrbracket_{C} &= \llbracket \Gamma \rrbracket_{C \backslash \{\overline{f}\}}, \overline{f^{*}}, \text{🔒}_{\langle\!| \llbracket \{\overline{f}\} \cap C \rrbracket \rangle}, \\
&\qquad \overline{x : \llbracket A \rrbracket}, \overline{f : [f^{*}]\llbracket T \rrbracket}, \overline{\hat{f} :_{[f^{*}]} \llbracket T \rrbracket}
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Type} \to \text{Type} \\
\llbracket T \ \mathbf{at} \ C \rrbracket &= [\llbracket C \rrbracket]\llbracket T \rrbracket \\
\llbracket (\overline{A}, \overline{f : T}) \Rightarrow B \rrbracket &= \forall \overline{f^{*}}.\langle\!| \overline{f^{*}} \rangle (\overline{\llbracket A \rrbracket} \to \overline{[f^{*}]\llbracket T \rrbracket} \to \llbracket B \rrbracket)
\end{aligned}
\qquad
\begin{aligned}
\llbracket \Gamma, f :^{C} T \rrbracket_{C'} &= \llbracket \Gamma \rrbracket_{C'}, f : [\llbracket C \rrbracket]\llbracket T \rrbracket, \hat{f} :_{[\llbracket C \rrbracket]} \llbracket T \rrbracket \\
\llbracket \Gamma, \clubsuit_{C} \rrbracket_{C'} &= \llbracket \Gamma \rrbracket_{C}, \text{🔒}_{[\llbracket C' \rrbracket]_{\llbracket C \rrbracket}}
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Value / Block} \to \text{Term} \\
\llbracket \mathbf{box} \ G : T \ \mathbf{at} \ C \rrbracket &= \mathbf{mod}_{[\llbracket C \rrbracket]} \ \llbracket G \rrbracket \\
\llbracket f \rrbracket &= \hat{f} \\
\llbracket \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\} \rrbracket &= \Lambda \overline{f^{*}}.\mathbf{mod}_{\langle\!| \overline{f^{*}} \rangle} (\lambda \overline{x^{\llbracket A \rrbracket}} \ \overline{f^{[f^{*}]\llbracket T \rrbracket}}.\overline{\mathbf{let\ mod}_{[f^{*}]} \ \hat{f} = f \ \mathbf{in}} \ \llbracket M \rrbracket) \\
\llbracket \mathbf{unbox} \ V : T \mid C \rrbracket &= \mathbf{let\ mod}_{[\llbracket C \rrbracket]} \ x = \llbracket V \rrbracket \ \mathbf{in} \ x
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Computation} \to \text{Term} \\
\llbracket \mathbf{def} \ f = G^{T \mid C} \ \mathbf{in} \ N \rrbracket &= \mathbf{let} \ f = \mathbf{mod}_{[\llbracket C \rrbracket]} \ \llbracket G \rrbracket \ \mathbf{in\ let\ mod}_{[\llbracket C \rrbracket]} \ \hat{f} = f \ \mathbf{in} \ \llbracket N \rrbracket \\
\llbracket P(\overline{V_i}, \overline{Q_j^{T_j \mid C_j}}) \rrbracket &= \mathbf{let\ mod}_{\langle\!| \overline{\llbracket C_j \rrbracket} \rangle} \ x = \llbracket P \rrbracket \ \overline{\llbracket C_j \rrbracket} \ \mathbf{in} \ x \ \overline{\llbracket V_i \rrbracket} \ \overline{(\mathbf{mod}_{[\llbracket C_j \rrbracket]} \ \llbracket Q_j \rrbracket)} \\
\left\llbracket \begin{array}{l} \mathbf{handle} \ \{f \Rightarrow M\} \ \mathbf{with} \\ \{\mathsf{op} \ p \ r \mapsto N\} : A \mid C \end{array} \right\rrbracket &= \mathbf{handle}^{\spadesuit}_{f^{*}} \ (\mathbf{let} \ f = \mathbf{mod}_{[f^{*}]} \ (\lambda x^{\llbracket A_{\mathsf{op}} \rrbracket}.\mathbf{do}_{f^{*}} \ x) \ \mathbf{in} \\
&\qquad\qquad \mathbf{let\ mod}_{[f^{*}]} \ \hat{f} = f \ \mathbf{in} \ \llbracket M \rrbracket) \\
&\quad \mathbf{with} \ \{\mathbf{return} \ x \mapsto \mathbf{let\ mod}_{[\llbracket C \rrbracket]} \ x' = x \ \mathbf{in} \ x' \\
&\qquad\quad \mathsf{op} \ p \ r \mapsto \mathbf{let\ mod}_{[\llbracket C \rrbracket]} \ \hat{r} = r \ \mathbf{in} \ \llbracket N \rrbracket\}
\end{aligned}
$$

Fig. 6. An encoding of System C in Metn with effect variables (omitting homomorphic cases).

Theorem 6.4 (Type Preservation). *If $\Gamma \vdash M : A \mid C$ is a well-formed typing judgement in System C, then $\llbracket \Gamma \rrbracket_{C} \vdash \llbracket M \rrbracket : \llbracket A \rrbracket \ @ \ \llbracket C \rrbracket$ in Metn. Similarly for typing judgements of values and blocks.*

Theorem 6.5 (Semantics Preservation). *If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System C, then $\llbracket M \rrbracket \mid \llbracket \Omega \rrbracket \rightsquigarrow^{*} \llbracket N \rrbracket \mid \llbracket \Omega' \rrbracket$ in Metn.*

7 Row-Based Effect Systems a la Koka

In this section we briefly present two core calculi with row-based effect systems in the style of Koka [17]: System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ [27], a calculus with named handlers and effect polymorphism, and System $\mathsf{F}^{\epsilon+\mathsf{sn}}_{1}$, a fragment of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ with a single effect variable. Our full specifications of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and System $\mathsf{F}^{\epsilon+\mathsf{sn}}_{1}$ in Appendices C.3 and C.4 also include unnamed handlers.

7.1 System $\mathsf{F}^{\epsilon+\mathsf{sn}}$

The syntax of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ is as follows. We highlight syntax for named handlers.

$$
\begin{array}{llll}
\text{Value Types} & A, B ::= 1 \mid A \to^{E} B \mid \forall \alpha^{K}.A \mid \mathsf{ev}\,\ell^{a} & \text{Kind} & K ::= \mathsf{Effect} \mid \mathsf{Scope} \\
\text{Other Types} & T ::= E \mid a & \text{Values} & V, W ::= () \mid x \mid \lambda^{E} x^{A}.M \mid \Lambda \alpha^{K}.V \mid \mathbf{nhandler}\ H \\
\text{Effect Rows} & E ::= \cdot \mid \varepsilon \mid \ell^{a}, E & \text{Computations} & M, N ::= \mathbf{return}\ V \mid V\ W \mid V\ T \mid \mathbf{do}\ V\ W \\
\text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, \alpha : K \mid \Gamma, \clubsuit & & \quad \mid\ \mathbf{let}\ x = M\ \mathbf{in}\ N
\end{array}
$$

Different from Xie et al. [27], our version of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ is fine-grain call-by-value. Effect rows E are scoped rows [18] with effect variables ε. Each effect label ℓ is annotated with a scope variable a. By convention, we write ε for effect variables of kind Effect and a for scope variables of kind Scope. We use α to range over type variables. We omit kinds when they are obvious from alphabets. Markers $\clubsuit$ in contexts are only used for the encoding in Section 8.1.

The typing rules of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ are standard [12, 19]. We show three representative rules here.

$$
\text{T-Abs}\ \frac{\Gamma, \clubsuit, x : A \vdash M : B \mid F}{\Gamma \vdash \lambda^{F} x^{A}.M : A \to^{F} B}
\qquad
\text{T-DoName}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \quad \Gamma \vdash V : \mathsf{ev}\,\ell^{a} \quad \Gamma \vdash W : A}{\Gamma \vdash \mathbf{do}\ V\ W : B \mid \ell^{a}, E}
$$

$$
\text{T-NamedHandler}\ \frac{H = \{\mathsf{op}\ p\ r \mapsto N\} \quad \Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \quad \Gamma, \clubsuit, p : A', r : B' \to^{F} A \vdash N : A \mid F}{\Gamma \vdash \mathbf{nhandler}\ H : (\forall a^{\mathsf{Scope}}.\mathsf{ev}\,\ell^{a} \to^{\ell^{a}, F} A) \to^{F} A}
$$

Rule T-Abs introduces a λ-abstraction. Rule T-NamedHandler introduces a named handler as a function, whose argument takes a handler name of the evidence type $\mathsf{ev}\,\ell^{a}$. An evidence type specifies an effect ℓ and a scope variable a. The usage of rank-2 polymorphism guarantees that the handler name of type $\mathsf{ev}\,\ell^{a}$ cannot escape the scope of the handler. Rule T-DoName invokes an operation via a handler name M of the evidence type $\mathsf{ev}\,\ell^{a}$. The full typing rules are given in Appendix C.3

Similar to the operational semantics of Metn and System C, reduction rules in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ are also of form $M \mid \Omega \rightsquigarrow N \mid \Omega'$. The most interesting rule is E-Gen which reduces a handler application to a runtime **handle** construct and passes a runtime evidence value $\mathbf{ev}_h$ to the argument.

$$
\text{E-Gen} \quad \mathbf{nhandler}\ H\ V \mid \Omega \rightsquigarrow \mathbf{handle}_h\ V\ a\ \mathbf{ev}_h\ \mathbf{with}\ H \mid \Omega, h : \ell^{a} \quad \text{where } a, h \text{ fresh and } H \propto \ell
$$

The full specification of operational semantics is given in Appendix C.3.

7.2 System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$

Figure 7 gives the syntax and typing rules of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$. They are mostly the same as those in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. As we only refer to one effect variable at a time, we need not track effect variables on types and terms. Nonetheless, we include them in grey font as $\{E|\varepsilon\}$ for easier comparison with System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. As a result of not having effect variables in types, we annotate term variable x in context Γ with the effect variable that it refers to. We need not kinds as the only form of explicit type variables are scope variables a. To avoid accidentally referring to multiple effect variables via invoking and handling operations, we restrict argument and result types of operations to not contain function arrows not appearing under effect abstractions.

Typing judgements $\Gamma \vdash_{\varepsilon} M : A \mid E$ and $\Gamma \vdash_{\varepsilon} V : A \mid E$ are annotated with the lexically closest effect variable ε. We use red for the effect rows E in value judgements; they are not needed for typing but important to our encoding. The typing rules are mostly standard. Rule T-Var is crucial to ensuring that each term can only refer to at most one effect variable. If the current effect variable matches the effect variable at which the variable was introduced, this condition is satisfied. Otherwise, the type of that variable has to be of form $\overline{\forall a}.\forall \varepsilon''.A'$ or $\overline{\forall a}.1$ which do not refer to any effect variable. Rule T-Abs does not change the effect variable, while rule T-EAbs rule introduces a

$$
\begin{array}{llll}
\text{Value Types} & A, B ::= 1 \mid A \to^{\{E|\varepsilon\}} B & \text{Values} & V, W ::= () \mid x \mid \lambda^{\{E|\varepsilon\}} x^A.M \mid \Lambda a.V \mid \Lambda\varepsilon.V \\
& \quad \mid \forall\varepsilon.A \mid \forall a.A \mid \mathsf{ev}\,\ell^a & & \quad \mid \mathbf{nhandler}\ H \\
\text{Effect Rows} & E ::= \cdot \mid \ell^a, E & \text{Computations} & M, N ::= V \mid V\,W \mid V\,A \mid V\,\#\{E|\varepsilon\} \\
\text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x :_\varepsilon A \mid \Gamma, \alpha : K \mid \Gamma, \clubsuit_E \mid \Gamma, \diamondsuit_E & & \quad \mid \mathbf{do}\ V\,W \mid \mathbf{let}\ x = M\ \mathbf{in}\ N
\end{array}
$$

$\boxed{\Gamma \vdash_\varepsilon V : A \mid E}$ $\boxed{\Gamma \vdash_\varepsilon M : A \mid E}$

$$
\text{T-Unit}\ \frac{}{\Gamma \vdash_\varepsilon () : 1 \mid E}
\qquad
\text{T-Var}\ \frac{\varepsilon = \varepsilon' \text{ or } A = \overline{\forall a}.\forall\varepsilon''.A' \text{ or } A = \overline{\forall a}.1}{\Gamma, x :_{\varepsilon'} A, \Gamma' \vdash_\varepsilon x : A \mid E}
\qquad
\text{T-Abs}\ \frac{\Gamma, \diamondsuit_E, x :_\varepsilon A \vdash_\varepsilon M : B \mid F}{\Gamma \vdash_\varepsilon \lambda^{\{F|\varepsilon\}} x^A.M : A \to^{\{F|\varepsilon\}} B \mid E}
$$

$$
\text{T-SAbs}\ \frac{\Gamma, a \vdash_\varepsilon V : A \mid E}{\Gamma \vdash_\varepsilon \Lambda a.V : \forall a.A \mid E}
\qquad
\text{T-EAbs}\ \frac{\Gamma, \clubsuit_E \vdash_{\varepsilon'} V : A \mid \cdot \qquad \varepsilon' \notin \mathrm{ftv}(\Gamma)}{\Gamma \vdash_\varepsilon \Lambda\varepsilon'.V : \forall\varepsilon'.A \mid E}
\qquad
\text{T-Value}\ \frac{\Gamma \vdash_\varepsilon V : A \mid E}{\Gamma \vdash_\varepsilon \mathbf{return}\ V : A \mid E}
\qquad
\text{T-SApp}\ \frac{\Gamma \vdash_\varepsilon V : \forall a.A \mid E \qquad \Gamma \ni b}{\Gamma \vdash_\varepsilon V\,b : A[b/a] \mid E}
$$

$$
\text{T-Let}\ \frac{\Gamma \vdash_\varepsilon M : A \mid E \qquad \Gamma, x : A \vdash_\varepsilon N : B \mid E}{\Gamma \vdash_\varepsilon \mathbf{let}\ x = M\ \mathbf{in}\ N : B \mid E}
\qquad
\text{T-EApp}\ \frac{\Gamma \vdash_\varepsilon V : \forall\varepsilon'.A \mid E}{\Gamma \vdash_\varepsilon V\,\#\{E|\varepsilon\} : A[\{E|\varepsilon\}/\varepsilon'] \mid E}
\qquad
\text{T-App}\ \frac{\Gamma \vdash_\varepsilon V : A \to^{\{E|\varepsilon\}} B \mid E \qquad \Gamma \vdash_\varepsilon W : A \mid E}{\Gamma \vdash_\varepsilon V\,W : B \mid E}
$$

$$
\text{T-DoName}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \quad E = \ell^a, F \quad \Gamma \vdash_\varepsilon V : \mathsf{ev}\,\ell^a \mid E \quad \Gamma \vdash_\varepsilon W : A \mid E}{\Gamma \vdash_\varepsilon \mathbf{do}\ V\,W : B \mid E}
$$

$$
\text{T-NamedHandler}\ \frac{H = \{\mathsf{op}\ p\ r \mapsto N\} \quad \Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \quad \Gamma, \diamondsuit_E, p :_\varepsilon A', r :_\varepsilon B' \to^{\{F|\varepsilon\}} A \vdash_\varepsilon N : A \mid F}{\Gamma \vdash_\varepsilon \mathbf{nhandler}\ H : (\forall a.\mathsf{ev}\,\ell^a \to^{\{\ell^a, F|\varepsilon\}} A) \to^{\{F|\varepsilon\}} A \mid E}
$$

Fig. 7. Syntax and typing rules for System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$.

fresh effect variable ε'. Rule T-EApp instantiates an effect abstraction with the current effect row E. The operation $[\{E|\varepsilon\}/\varepsilon']$ is just standard type substitution.

The operational semantics of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ is given in Appendix C.4.

8 Encoding Row-Based Effect Systems into Metn

We show how to encode System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ into Metn with effect variables and System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ into Metn without effect variables. The latter makes use of masking handler names. The full specifications and proofs with both unnamed and named handlers can be found in Appendices C.5 and D.

8.1 Encoding System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ in Metn with Effect Variables

Not every well-typed term in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ is meaningful in the sense that we can find an appropriate evaluation context to fully apply its abstractions and handle its effects. For example, a function of type $\forall a.\mathsf{ev}\,I^a \times \mathsf{ev}\,J^a \to^{I^a, J^a} 1$ cannot be applied and handled when $I \neq J$, because handlers cannot provide two evidences values of type $\mathsf{ev}\,I^a$ and $\mathsf{ev}\,J^a$ with the same scope variable a but different interfaces. (This type becomes meaningful with umbrella effects in Xie et al. [27].) Each handler introduces a fresh scope variable with some fixed effect. Since in Metn each handler name is associated with its effect when being bound, our translation does not work on terms with this kind of inconsistency. To resolve the mismatch, we define the notion of *consistent typing judgements*, in which each scope variable is associated with a unique effect label.

$$
\begin{array}{rcl@{\qquad}rcl}
[\![-]\!] & : & \text{Type} \longrightarrow \text{Type} & [\![-]\!] & : & \text{Effect Row} \longrightarrow \text{Effect Context} \\
[\![A \to^{E} B]\!] & = & [[\![E]\!]]([\![A]\!] \to [\![B]\!]) & [\![\ell^{a}, E]\!] & = & a, [\![E]\!] \\
[\![\forall a^{\ell}.A]\!] & = & \forall a^{\ell}.[\![A]\!] & [\![-]\!]_{-} & : & \text{Context} \times \text{Effect Row} \longrightarrow \text{Context} \\
[\![\mathsf{ev}\,\ell^{a}]\!] & = & [a]([\![A_\ell]\!] \to [\![B_\ell]\!]) & [\![\Gamma, \clubsuit]\!]_{E} & = & [\![\Gamma]\!]., \text{🔒}_{[[\![E]\!]]}.
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Value / Computation} \longrightarrow \text{Term} \\
[\![\Lambda\varepsilon.V]\!] & = & \Lambda\varepsilon.[\![V]\!] \\
[\![\lambda^{E} x^{A}.M]\!] & = & \mathbf{mod}_{[[\![E]\!]]}\ (\lambda x^{[\![A]\!]}.[\![M]\!]) \\
[\![V^{A \to^{E} B}\ W]\!] & = & \mathbf{let\ mod}_{[[\![E]\!]]}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] \\
[\![\mathbf{do}\ V^{\mathsf{ev}\,\ell^{a}}\ W]\!] & = & \mathbf{let\ mod}_{[a]}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] \\
\left[\!\!\left[\begin{array}{l}\mathbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\} \\ : (\forall a^{\ell}.\mathsf{ev}\,\ell^{a} \to^{\ell^{a},E} A) \to^{E} A\end{array}\right]\!\!\right] & = & \mathbf{mod}_{[[\![E]\!]]}\ (\lambda f.\mathbf{handle}^{\spadesuit}_{a}\ (\mathbf{let}\ f = f\ a\ \mathbf{in\ let\ mod}_{[[\![\mathsf{ev}\,\ell^{a}, E]\!]]}\ f = f\ \mathbf{in} \\
 & & \qquad f\ (\mathbf{mod}_{[a]}\ (\lambda x^{[\![A_{\mathsf{op}}]\!]}.\mathbf{do}_{a}\ x))) \\
 & & \mathbf{with}\ \{\mathbf{return}\ x \mapsto \mathbf{let\ mod}_{[[\![E]\!]]}\ x = x\ \mathbf{in}\ x, \mathsf{op}\ p\ r \mapsto [\![N]\!]\})
\end{array}
$$

Fig. 8. An encoding of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ in Metn with effect variables (omitting homomorphic cases).

Definition 8.1 (Consistency). A typing judgement $\Gamma \vdash M : A \mid E$ or $\Gamma \vdash V : A \mid E$ in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ is consistent if for each scope variable a bound in Γ or in A, it appears and only appears as ℓ^{a} for some effect label ℓ. A well-typed term M or value A is consistent if its judgement is consistent.

For a consistent judgement, we can annotate each scope variable binding as Λa^{ℓ} and translate its occurrence in the context to $a : \ell$. We define our translation on annotated consistent judgements.

Figure 8 encodes System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ in Metn with effect variables. The term translation is type-directed and we annotate components with their types when needed. As illustrated by examples in Section 2.4, the core idea is to translate an effectful function arrow $A \to^{E} B$ into a function arrow with an absolute modality $[[\![E]\!]]([\![A]\!] \to [\![B]\!])$. Consequently, the translation of lambda abstraction $\lambda^{E} x^{A}.M$ introduces a modality $[[\![E]\!]]$, and the translation of application $V\ W$ first eliminates the modality of $[\![V]\!]$ before applying it. Translations of abstraction and application of effect rows are homomorphic and omitted. Abstraction of scope variables is translated to name abstraction in Metn. The evidence type $\mathsf{ev}\,\ell^{a}$ is translated to a function type $[a]([\![A_\ell]\!] \to [\![B_\ell]\!])$ which invokes the operation in the effect ℓ. Correspondingly, an operation invocation **do** $V\ W$ for $V : \mathsf{ev}\,\ell^{a}$ is translated similarly to a function application. A named handler **nhandler** H is translated to a function that takes a function argument f and handles it with a named handler. We pass the term $\mathbf{mod}_{[a]}\ (\lambda x.\mathbf{do}_{a}\ x)$ to the argument f to simulate the handler name of type $\mathsf{ev}\,\ell^{a}$.

We omit homomorphic cases for the translations of contexts and effect contexts. The translation of contexts is indexed by the current effect row. In contexts, we add absolute markers $\clubsuit$ which indicate the switch from an effectful computation into a value as in T-Abs and T-NamedHandler, reading from right to left. An absolute marker is translated to a lock with an absolute modality which changes the effect context from some $[\![E]\!]$ to the empty effect context.

The following theorems show that our encoding preserves types and operational semantics.

Theorem 8.2 (Type Preservation). *If $\Gamma \vdash M : A \mid E$ is a consistent judgement in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, then $[\![\Gamma]\!]_{E} \vdash [\![M]\!] : [\![A]\!]\ @\ [\![E]\!]$ in Metn. Similarly for typing judgements of values.*

Lemma 8.3 (Semantics Preservation). *If M is consistent and well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, then $[\![M]\!] \mid [\![\Omega]\!] \rightsquigarrow^{*} [\![N]\!] \mid [\![\Omega']\!]$ in Metn.*

8.2 Encoding System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ without Effect Variables

In Section 7.2, we incorporate *absolute markers* $♣_E$ and *relative markers* $♦_E$ in contexts. Similar to the translation of System C in Section 6.2, we need certain well-formedness conditions on these markers. Following Tang et al. [26], we define well-formed typing judgements as follows.

Definition 8.4 (Well-formed). A typing judgement $\Gamma_1, x :_\varepsilon A, \Gamma_2 \vdash_\varepsilon M : B \mid E$ is *well-formed for* x if either $x \notin \mathsf{fv}(M)$ or $♣ \notin \Gamma_2$ or $A = \overline{\forall a}.\forall.A'$ or $A = \overline{\forall a}.1$. A typing judgement $\Gamma \vdash_\varepsilon M : A \mid E$ is *well-formed* if it is well-scoped for all $x \in \Gamma$. Similarly for typing judgements of values.

Restricting to well-formed judgements does not lose any expressiveness since a typing judgement with the empty context is well-formed, and every judgement in the derivation tree of a well-formed judgement is well-formed. As System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ is a fragment of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, we also need to rule out inconsistent terms where a scope variable is associated with different effect labels. The definition of consistent terms and judgements in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ is the same as Definition 8.1 of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. As in Section 8.1, our translation is defined on these consistent typing judgements where scope variable bindings are annotated with effect labels.

Figure 9 encodes System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ in Metn without effect variables. The translation of types is indexed by the current effect row which corresponds to the ambient effect context in Metn. This requires us to keep the effect row in the typing judgements for values as in Section 7.2. Different from the translation of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ where we wrap every function type with an absolute modality, here we wrap it with a relative modality so that the function can still use effects from the ambient effect context. When translating a function type $A \to^F B$ at effect row E, we use a relative modality $\langle [\![E]\!] - [\![F]\!] | [\![F]\!] - [\![E]\!] \rangle$ which exactly changes the effect context from $[\![E]\!]$ to $[\![F]\!]$. Note that we need the extension of masking handler names in Section 4.4 since E may contain handler names not in F. An effect quantifier $\forall.A$ introduces a new effect variable shadowing the previous one. We encode it via the empty absolute modality which also completely replaces the previous effect context. Evidence types are translated in the same way as in the translation of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. Note that for operation argument types A_ℓ and result types B_ℓ the effect row does not influence their translations due to the restriction in Section 7.2. For name abstraction $\forall a^\ell.A$, we recursively translate type A, and move the top-level modality of $[\![A]\!]$ to the top of the translation result except for the part of handler name a for well-scopedness.

As we can see, each type in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ is translated to a modal type in Metn with exactly one top-level modality. This property enables us to uniformly eliminate the top-level modalities when translating variable bindings, similar to the technique we use in Section 6.2 to encode System C. For example, in the translation of $\lambda^F x^A.M$ and **let** $x = M$ **in** N, we eliminate the modality of x and bind it to $\hat{x}$. Also, in the translation of contexts, we translate a term variable $x : A$ into x and $\hat{x}$. Immediately eliminating modalities when binding variables works well, especially given that Metn adopts let-style modality elimination. The translation of a variable x inserts an appropriate new modality for $\hat{x}$ whose previous modality has been eliminated.

The translation of contexts is indexed by the current effect row. We translate an absolute marker to a lock with an absolute modality and a relative marker to a lock with a relative modality.

The translation on terms is formally a translation on typing derivations, similar to previous encodings. For convenience, we also index the term translation with the effect row. The value translation follows the translations of their types. The abstraction and application of scope variables require a case analysis on whether the name a appears in the top-level modality or not. We exploit the extension of moving name abstractions inside modalities from Section 4.1. For function application, we eliminate the identity modality of the function. Note that the top-level modality of the function is always identity guaranteed by T-App-System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$. The translation of operation invocation is the same as in Section 8.1. For named handlers, the example $\mathit{sum}_{\mathsf{F}_1^{\epsilon+\mathsf{sn}}}$ in Section 2.3 is simplified since

$$
\begin{array}{rcl}
[\![-]\!]_{-} & : & \text{Type} \times \text{Effect Row} \to \text{Type} \\
[\![1]\!]_E & = & \langle\rangle 1 \\
[\![A \to^F B]\!]_E & = & \langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle ([\![A]\!]_F \to [\![B]\!]_F) \\
[\![\forall.A]\!]_E & = & [\,][\![A]\!]. \\
[\![\forall a^\ell.A]\!]_E & = & \begin{cases} \nu(\forall a^\ell.\langle\!| a\rangle B), & \text{if } \mu \equiv \nu \circ \langle\!| a\rangle \\ \mu(\forall a^\ell.B), & \text{otherwise} \end{cases} \\
 & & \text{where } [\![A]\!]_E = \mu B \\
[\![\mathsf{ev}\,\ell^a]\!]_E & = & [a]([\![A_\ell]\!]. \to [\![B_\ell]\!].)
\end{array}
\qquad
\begin{array}{rcl}
[\![-]\!] & : & \text{Effect Row} \to \text{Effect Context} \\
[\![\cdot]\!] & = & \cdot \\
[\![\ell^a, E]\!] & = & a, [\![E]\!] \\
[\![-]\!]_{-} & : & \text{Context} \times \text{Effect Row} \to \text{Context} \\
[\![\cdot]\!]_E & = & \cdot \\
[\![\Gamma, x : A]\!]_E & = & [\![\Gamma]\!]_E, x : \mu A', \hat{x} :_\mu A' \text{ for } \mu A' = [\![A]\!]_E \\
[\![\Gamma, a : \ell]\!]_E & = & [\![\Gamma]\!]_E, a : \ell \\
[\![\Gamma, \clubsuit_E]\!]. & = & [\![\Gamma]\!]_E, \blacksquare_{[\,]} \\
[\![\Gamma, \blacklozenge_F]\!]_E & = & [\![\Gamma]\!]_F, \blacksquare_{\langle [\![F]\!] - [\![E]\!] \mid [\![E]\!] - [\![F]\!] \rangle}
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!]_{-} & : & \text{Value / Computation} \times \text{Effect Row} \to \text{Term} \\
[\![()]\!]_E & = & \mathbf{mod}_{\langle\rangle}\ () \\
[\![x^A]\!]_E & = & \mathbf{mod}_\mu\ \hat{x} \quad \text{where } [\![A]\!]_E = \mu\_ \\
[\![\Lambda.V]\!]_E & = & \mathbf{mod}_{[\,]}\ [\![V]\!]_E \\
[\![\Lambda a^\ell.V^A]\!]_E & = & \begin{cases} \mathbf{let\ mod}_\nu\ \lambda a^\ell.x = (\mathbf{let\ mod}_\mu\ x = [\![V]\!]_E\ \mathbf{in\ mod}_{\nu;\langle\!| a\rangle}\ x) \\ \quad \mathbf{in\ mod}_\nu\ x & \text{if } \mu \equiv \nu \circ \langle\!| a\rangle \\ \mathbf{let\ mod}_\mu\ \lambda a^\ell.x = [\![V]\!]_E\ \mathbf{in\ mod}_\mu\ x, & \text{otherwise} \end{cases} \\
 & & \text{where } [\![A]\!]_E = \mu\_ \\
[\![\lambda^F x^A.M]\!]_E & = & \mathbf{mod}_{\langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle}\ (\lambda x^{[\![A]\!]_F}.\mathbf{let\ mod}_\mu\ \hat{x} = x\ \mathbf{in}\ [\![M]\!]_F) \text{ where} [\![A]\!]_F = \mu\_ \\
[\![\mathbf{let}\ x = M^A\ \mathbf{in}\ N]\!]_E & = & \mathbf{let}\ x = [\![M]\!]_E\ \mathbf{in\ let\ mod}_\mu\ \hat{x} = x\ \mathbf{in}\ [\![N]\!]_E \quad \text{where} [\![A]\!]_E = \mu\_ \\
[\![V\#]\!]_E & = & \mathbf{let\ mod}_{[\,]}\ x = [\![V]\!]_E\ \mathbf{in}\ x \\
[\![V^{\forall a^\ell.A}\ b]\!]_E & = & \begin{cases} \mathbf{let\ mod}_\nu\ x = [\![V]\!]_E\ \mathbf{in\ let}_\nu\ \mathbf{mod}_{\langle\!| a\rangle}\ y = x\ b\ \mathbf{in\ mod}_\mu\ y & \text{if } \mu \equiv \nu \circ \langle\!| a\rangle \\ \mathbf{let\ mod}_\mu\ x = [\![V]\!]_E\ \mathbf{in\ mod}_\mu\ (x\ b) & \text{otherwise} \end{cases} \\
 & & \text{where } [\![A]\!]_E = \mu\_ \\
[\![V\ W]\!]_E & = & \mathbf{let\ mod}_{\langle\rangle}\ x = [\![V]\!]_E\ \mathbf{in}\ x\ [\![W]\!]_E \\
[\![\mathbf{do}\ V^{\mathsf{ev}\,\ell^a}\ W]\!]_E & = & \mathbf{let\ mod}_{[a]}\ x = [\![V]\!]_E\ \mathbf{in}\ x\ [\![W]\!]_E
\end{array}
$$

$$
\begin{array}{l}
[\![\mathbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (\forall a^\ell.\mathsf{ev}\,\ell^a \to^{\ell^a,F} A) \to^F A]\!]_E \ = \ \mathbf{mod}_{\langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle}\ (\\
\quad \lambda f.\mathbf{handle}_a\ (\mathbf{let\ mod}_{\langle\!| a\rangle}\ f' = f\ a\ \mathbf{in\ let\ mod}_\mu\ y = f'\ (\mathbf{mod}_{[a]}\ (\lambda x^{[\![A_{\mathsf{op}}]\!].}.\mathbf{do}_a\ x))\ \mathbf{in\ mod}_{\langle a |\!\rangle;\nu}\ y) \\
\qquad \mathbf{with}\ \{\mathbf{return}\ x \mapsto x, \mathsf{op}\ p\ r \mapsto \mathbf{let\ mod}_{\mu_p}\ \hat{p} = p\ \mathbf{in\ let\ mod}_{\langle\rangle}\ \hat{r} = r\ \mathbf{in}\ [\![N]\!]_F\}) \\
\quad \text{where } [\![A]\!]_{\ell^a,F} = \mu\_ \text{ and } [\![A]\!]_F = \nu\_ \text{ and } [\![A_{\mathsf{op}}]\!]_F = \mu_p\_
\end{array}
$$

Fig. 9. An encoding of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ in Metn without effect variables.

the return type of the handled computation is `Int` which has kind Abs, satisfying the requirement of T-HandleName in Section 3.5. For a general translation, we switch to the rule T-HandleName' in Section 4.4 and use the relative modality $\langle a |\!\rangle$ to prevent the return value from using the name a.

The following theorems state that our encoding preserves types and operational semantics.

Theorem 8.5 (Type Preservation). *If $\Gamma \vdash_\epsilon M : A \mid E$ is consistent and well-formed in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$, then $[\![\Gamma]\!]_E \vdash [\![M]\!]_E : [\![A]\!]_E \ @\ [\![E]\!]$ in Metn. Similarly for typing judgements of values.*

Theorem 8.6 (Semantics Preservation). *If M is consistent and well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$, then there exists N' in Metn such that $[\![M]\!] \mid [\![\Omega]\!] \rightsquigarrow^* N' \mid [\![\Omega']\!]$ and $[\![N]\!] \mid [\![\Omega']\!] \rightsquigarrow_v^* N' \mid [\![\Omega']\!]$. in Metn, where $\rightsquigarrow_v$ refers to reduction of values in Metn.*

This semantics preservation theorem is not a one-to-many mapping, in contrast to our previous semantics preservation theorems. This is because Metn allows reduction in complex values such as type application, whereas System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ does not.

9 Related and Future Work

Abstracting Effect Systems. Yoshioka et al. [28] study different formalisations of effect collections in a traditional effect system. Instead of giving macro translations, they propose a general framework whose effect type can be instantiated to various kinds of sets and row types. They point out that extending their framework to cover capability-based effect systems is challenging. It is interesting future work to explore if we can design an expressive framework by abstracting over the mode theory of modal effect types such that we can cover all styles of effect systems.

Expressive Power of Effect Handlers. Forster et al. [9] compare the expressive power of effect handlers, monadic reflection, and delimited control in a simply-typed setting and show that delimited control cannot encode effect handlers in a type-preserving way. Piróg et al. [23] extend the comparison between effect handlers and delimited control to a polymorphic setting and show their equivalence. Ikemori et al. [13] further show the typed equivalence between named handlers and multi-prompt delimited control. In contrast to these works, which compare effect handlers with other programming abstractions, we compare different effect systems for effect handlers.

Future Work. We are interested in designing a type inference algorithm for METN.

References

[1] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2018. Handle with care: relational interpretation of algebraic effects and handlers. *Proc. ACM Program. Lang.* 2, POPL (2018), 8:1–8:30. doi:10.1145/3158096

[2] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2020. Binders by day, labels by night: effect instances via lexically scoped handlers. *Proc. ACM Program. Lang.* 4, POPL (2020), 48:1–48:29. doi:10.1145/3371116

[3] Aleksander Boruch-Gruszecki, Martin Odersky, Edward Lee, Ondrej Lhoták, and Jonathan Immanuel Brachthäuser. 2023. Capturing Types. *ACM Trans. Program. Lang. Syst.* 45, 4 (2023), 21:1–21:52. doi:10.1145/3618003

[4] Jonathan Immanuel Brachthäuser, Philipp Schuster, Edward Lee, and Aleksander Boruch-Gruszecki. 2022. Effects, capabilities, and boxes: from scope-based reasoning to type-based reasoning and back. *Proc. ACM Program. Lang.* 6, OOPSLA1 (2022), 1–30. doi:10.1145/3527320

[5] Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. 2020. Effects as capabilities: effect handlers and lightweight effect polymorphism. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 126:1–126:30. doi:10.1145/3428194

[6] Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. [n. d.]. Effekt Language: A language with lexical effect handlers and lightweight effect polymorphism. https://effekt-lang.org. Accessed 2025-02-28.

[7] Lukas Convent, Sam Lindley, Conor McBride, and Craig McLaughlin. 2020. Doo bee doo bee doo. *J. Funct. Program.* 30 (2020), e9. doi:10.1017/S0956796820000039

[8] Matthias Felleisen. 1991. On the Expressive Power of Programming Languages. *Sci. Comput. Program.* 17, 1-3 (1991), 35–75. doi:10.1016/0167-6423(91)90036-W

[9] Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. 2019. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *J. Funct. Program.* 29 (2019), e15. doi:10.1017/S0956796819000121

[10] Daniel Gratzer. 2023. *Syntax and semantics of modal type theory.* Ph. D. Dissertation. Aarhus University.

[11] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2021. Multimodal Dependent Type Theory. *Log. Methods Comput. Sci.* 17, 3 (2021). doi:10.46298/LMCS-17(3:11)2021

[12] Daniel Hillerström and Sam Lindley. 2016. Liberating Effects with Rows and Handlers *(TyDe 2016)*. Association for Computing Machinery, New York, NY, USA, 15–27. doi:10.1145/2976022.2976033

[13] Kazuki Ikemori, Youyou Cong, and Hidehiko Masuhara. 2023. Typed Equivalence of Labeled Effect Handlers and Labeled Delimited Control Operators. In *International Symposium on Principles and Practice of Declarative Programming, PPDP 2023, Lisboa, Portugal, October 22-23, 2023*, Santiago Escobar and Vasco T. Vasconcelos (Eds.). ACM, 4:1–4:13. doi:10.1145/3610612.3610616

[14] Ohad Kammar, Sam Lindley, and Nicolas Oury. 2013. Handlers in action. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, Greg Morrisett and Tarmo Uustalu (Eds.). ACM, 145–158. doi:10.1145/2500365.2500590

[15] G. A. Kavvos and Daniel Gratzer. 2023. Under Lock and Key: a Proof System for a Multimodal Logic. *Bull. Symb. Log.* 29, 2 (2023), 264–293. doi:10.1017/BSL.2023.14

[16] John Launchbury and Simon L. Peyton Jones. 1995. State in Haskell. *LISP Symb. Comput.* 8, 4 (1995), 293–341.

[17] Daan Leijen. [n. d.]. Koka: A strongly typed functional-style language with effect types and handlers. https://koka-lang.github.io. Accessed 2025-02-28.

[18] Daan Leijen. 2005. Extensible records with scoped labels. In *Revised Selected Papers from the Sixth Symposium on Trends in Functional Programming, TFP 2005, Tallinn, Estonia, 23-24 September 2005 (Trends in Functional Programming, Vol. 6)*, Marko C. J. D. van Eekelen (Ed.). Intellect, 179–194.

[19] Daan Leijen. 2017. Type Directed Compilation of Row-Typed Algebraic Effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL '17)*. Association for Computing Machinery, New York, NY, USA, 486–499. doi:10.1145/3009837.3009872

[20] Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantics Structures in Computation, Vol. 2. Springer.

[21] Paul Blain Levy, John Power, and Hayo Thielecke. 2003. Modelling environments in call-by-value programming languages. *Inf. Comput.* 185, 2 (2003), 182–210. doi:10.1016/S0890-5401(03)00088-9

[22] Sam Lindley, Conor McBride, and Craig McLaughlin. 2017. Do Be Do Be Do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL 2017)*. Association for Computing Machinery, New York, NY, USA, 500–514. doi:10.1145/3009837.3009897

[23] Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. 2019. Typed Equivalence of Effect Handlers and Delimited Control. In *FSCD (LIPIcs, Vol. 131)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:16.

[24] Gordon D. Plotkin and Matija Pretnar. 2013. Handling Algebraic Effects. *Log. Methods Comput. Sci.* 9, 4 (2013).

[25] Didier Rémy. 1994. Type Inference for Records in a Natural Extension of ML. In *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*. Citeseer.

[26] Wenhao Tang, Leo White, Stephen Dolan, Daniel Hillerström, Sam Lindley, and Anton Lorenzen. 2024. Modal Effect Types. *CoRR* abs/2407.11816 (2024). doi:10.48550/ARXIV.2407.11816 arXiv:2407.11816 Accepted by OOPSLA 2025.

[27] Ningning Xie, Youyou Cong, Kazuki Ikemori, and Daan Leijen. 2022. First-class names for effect handlers. *Proc. ACM Program. Lang.* 6, OOPSLA2 (2022), 30–59. doi:10.1145/3563289

[28] Takuma Yoshioka, Taro Sekiyama, and Atsushi Igarashi. 2024. Abstracting Effect Systems for Algebraic Effect Handlers. *CoRR* abs/2404.16381 (2024). doi:10.48550/ARXIV.2404.16381 arXiv:2404.16381

[29] Yizhou Zhang and Andrew C. Myers. 2019. Abstraction-safe effect handlers via tunneling. *Proc. ACM Program. Lang.* 3, POPL (2019), 5:1–5:29. doi:10.1145/3290318

A Full Specification of METN with Named Handlers, Unnamed Handlers, and Extensions

We provide the full specification of METN including unnamed handlers in MET [26] and all extensions in Section 4.

A.1 Syntax

Figure 10 gives the syntax of METN with both named and unnamed handlers as well as all extensions. We highlight syntax for named handlers and extensions, i.e., new parts from core MET. We combine masks L and extensions D into one syntax category.

Types	$A, B ::= 1 \mid A \to B \mid \mu A \mid \forall a^{\ell}.A \mid \forall \varepsilon^{Y}.A$
Masks and Extensions	$L, D ::= \cdot \mid \ell, D \mid a, D \mid D\backslash L$
Effect Contexts	$E, F ::= \cdot \mid \ell, E \mid a, E \mid E\backslash L \mid \varepsilon, E \mid \varepsilon\backslash L, E$
Modalities	$\mu ::= [E] \mid \langle L \vert D \rangle$
Interface Contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : \{\overline{\mathsf{op} : A \twoheadrightarrow B}\}$
Kinds	$K ::= \mathsf{Abs} \mid \mathsf{Any}$
Effect Kinds	$Y ::= \mathsf{Names} \mid \mathsf{Effect}$
Name Kinds	$R ::= \cdot \mid \sharp$
Contexts	$\Gamma ::= \cdot \mid \Gamma, 🔒_{\mu_F} \mid \Gamma, a :_R \ell \mid \Gamma, x :_{\mu_F} A \mid \Gamma, \varepsilon : Y$
Terms	$M, N ::= () \mid x \mid \lambda x^{A}.M \mid M\,N \mid \mathbf{mod}_{\mu}\,V$
	$\mid \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ x = V\ \mathbf{in}\ N$
	$\mid \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ \Lambda\!\!\!\!\lambda\overline{a^{\ell}}.\Lambda\overline{\varepsilon^{Y}}.x = V\ \mathbf{in}\ N$
	$\mid \Lambda\!\!\!\!\lambda a^{\ell}.V \mid M\,a \mid \mathbf{do}_{a}\ \mathsf{op}\ M \mid \mathbf{mask}_{L}\ M$
	$\mid \mathbf{handle}^{\delta}_{\eta}\ M\ \mathbf{with}\ H \mid \Lambda\varepsilon^{Y}.V \mid M\ E$
Values	$V, W ::= () \mid x \mid \lambda x^{A}.M \mid \mathbf{mod}_{\mu}\,V$
	$\mid \Lambda\!\!\!\!\lambda a^{\ell}.V \mid \Lambda\varepsilon^{Y}.V \mid V\ a \mid V\ E$
	$\mid \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ x = V\ \mathbf{in}\ W$
	$\mid \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ \Lambda\!\!\!\!\lambda\overline{a^{\ell}}.\Lambda\overline{\varepsilon^{Y}}.x = V\ \mathbf{in}\ W$
Handlers	$H ::= \{\mathbf{return}\ x \mapsto M\} \mid H \uplus \{\mathsf{op}\ p\ r \mapsto M\}$
Handler Superscripts	$\delta ::= \cdot \mid \spadesuit$
Handler Subscripts	$\eta ::= \cdot \mid a$
Handler Names	a, b

Fig. 10. Syntax of METN with extensions.

To combine effect variables, named handlers, and unnamed handlers together, we need to distinguish between effects consist of only handler names and those also contain effect labels. Effect variables for the former can appear multiple times in an effect context, while effect variables for the latter cannot. We introduce effect kinds Y where Effect refers to any effect contexts and Names refers to those with only handler names.

In the syntax of effect contexts we include $\varepsilon\backslash L, E$ as a valid form. In the syntax of handlers, different combinations of superscripts and subscripts give us four forms of handlers: unnamed handlers ($\delta = \cdot, \eta = \cdot$), named handlers ($\delta = \cdot, \eta = a$), absolute unnamed handlers ($\delta = \spadesuit, \eta = \cdot$), and absolute named handlers ($\delta = \spadesuit, \eta = a$).

A.2 Kinding and Well-Formedness

The full kinding and well-formedness rules for METN are defined in Figure 11. For masks and extensions, we also use the kinds Effect and Names to distinguish between those with only handler names and those that also contain effect labels. The kinding for effect contexts guarantees that there can only be at most one effect variable ε not of kind Names. For the global label context Σ we require $\cdot \vdash \ell$ for all effect labels ℓ in it.

A.3 Type Equivalence and Sub-typing

Type equivalence is given in Figure 12. As in Section 4.4, we use the parameterised boolean algebra $\mathbb{B}(\Gamma)$. Note that with the extension of effect variables, every effect context of kind Names under context Γ still corresponds to an element in the boolean algebra $\mathbb{B}(\Gamma)$.

By equivalence relations we can always normalise masks and extensions to forms of $\overline{\ell}, D$ with $\Gamma \vdash D : \text{Names}$. We can always normalise effect contexts to either $\overline{\ell}, E$ or $\overline{\ell}, \varepsilon \backslash L, E$ with $\Gamma \vdash E : \text{Names}$ and $\Gamma \vdash \varepsilon : \text{Effect}$. Subeffecting is given in Figure 13. For brevity, we directly define it on normalised effect contexts.

Note that although the current formalisation cannot actually distinguish between multiple appearances of the same effect label in an effect context, we still strictly follow the principle of scoped rows [18] that we can only swap distinguished labels. This becomes matters when we consider parameterised effects where effect labels take type arguments in effect contexts.

A.4 Meta Operations

We simply define $D + E = D, E$ and $E - L = E \backslash L$. Other plus and minus operations between different syntactic categories are defined in the same way. The operation $L \bowtie D$ used in Tang et al. [26] is replaced by operations $L - D$ and $D - L$.

A.5 Mode Theory

In the terminology of MTT, effect contexts are *modes*. The structure of modes, modalities, and modality transformation constitute the mode theory.

To make the proofs easier, we write modalities in the form μ_F a lot as they are morphisms between modes, while μ is an indexed family of morphisms. We call μ_F *concrete modalities*. We repeat the definitions of modalities and modality composition which are the same as those in Section 3.3 and also the same as in MET. We directly define them in the form of concrete modalities.

$$
\begin{array}{rcrcl}
[E]_F & : & E & \to & F \\
\langle L|D\rangle_F & : & D + (F - L) & \to & F
\end{array}
$$

$$
\begin{array}{rcrcl}
[E']_F & \circ & [E]_{E'} & = & [E]_F \\
\langle L|D\rangle_F & \circ & [E]_{D+(F-L)} & = & [E]_F \\
[E]_F & \circ & \langle L|D\rangle_E & = & [D + (E - L)]_F \\
\langle L_1|D_1\rangle_F & \circ & \langle L_2|D_2\rangle_{D_1+(F-L_1)} & = & \langle L_1 + (L_2 - D_1)|D_2 + (D_1 - l_2)\rangle_F
\end{array}
$$

The modality transformation relations are defined as the transitive closure of the following rules.

$\boxed{\Gamma \vdash A : K}$

$$\frac{\Gamma \vdash A : \mathsf{Abs}}{\Gamma \vdash A : \mathsf{Any}} \qquad \frac{}{\Gamma \vdash 1 : \mathsf{Abs}} \qquad \frac{\Gamma \vdash [E] \qquad \Gamma \vdash A : \mathsf{Any}}{\Gamma \vdash [E]A : \mathsf{Abs}} \qquad \frac{\Gamma \vdash \langle L|D\rangle \qquad \Gamma \vdash A : K}{\Gamma \vdash \langle L|D\rangle A : K}$$

$$\frac{\Gamma \vdash A : \mathsf{Any} \qquad \Gamma \vdash B : \mathsf{Any}}{\Gamma \vdash A \to B : \mathsf{Any}} \qquad \frac{\Gamma, a : \ell \vdash A : K}{\Gamma \vdash \mathbb{V}a^{\ell}.A : K} \qquad \frac{\Gamma, \varepsilon : Y \vdash A : K}{\Gamma \vdash \forall \varepsilon^{Y}.A : K}$$

$\boxed{\Gamma \vdash a : \ell}$ $\boxed{\Gamma \vdash \ell}$ $\boxed{\Gamma \vdash A \twoheadrightarrow B}$

$$\frac{\Gamma \ni a : \ell}{\Gamma \vdash a : \ell} \qquad \frac{\Sigma(\ell) = \{\overline{\mathsf{op} : A \twoheadrightarrow B}\} \qquad \overline{\Gamma \vdash A \twoheadrightarrow B}}{\Gamma \vdash \ell} \qquad \frac{\Gamma \vdash A : \mathsf{Abs} \qquad \Gamma \vdash B : \mathsf{Abs}}{\Gamma \vdash A \twoheadrightarrow B}$$

$\boxed{\Gamma \vdash E : Y}$

$$\frac{\Gamma \vdash E : \mathsf{Names}}{\Gamma \vdash E : \mathsf{Effect}} \qquad \frac{}{\Gamma \vdash \cdot : \mathsf{Names}} \qquad \frac{\Gamma \ni \varepsilon : Y}{\Gamma \vdash \varepsilon : Y} \qquad \frac{\vdash \ell \qquad \Gamma \vdash E : Y}{\Gamma \vdash \ell, E : \mathsf{Effect}}$$

$$\frac{\Gamma \vdash a : \ell \qquad \Gamma \vdash E : Y}{\Gamma \vdash a, E : Y} \qquad \frac{\Gamma \vdash \varepsilon : \mathsf{Names} \qquad \Gamma \vdash E : Y}{\Gamma \vdash \varepsilon, E : Y} \qquad \frac{\Gamma \vdash \varepsilon : \mathsf{Effect} \qquad \Gamma \vdash E : \mathsf{Names}}{\Gamma \vdash \varepsilon, E : \mathsf{Effect}}$$

$$\frac{\Gamma \vdash E : Y \qquad \Gamma \vdash L}{\Gamma \vdash E \backslash L : Y} \qquad \frac{\Gamma \vdash \varepsilon \backslash L : \mathsf{Names} \qquad \Gamma \vdash E : Y}{\Gamma \vdash \varepsilon \backslash L, E : Y} \qquad \frac{\Gamma \vdash \varepsilon \backslash L : \mathsf{Effect} \qquad \Gamma \vdash E : \mathsf{Names}}{\Gamma \vdash \varepsilon \backslash L, E : \mathsf{Effect}}$$

$\boxed{\Gamma \vdash D : Y}$

$$\frac{\Gamma \vdash D : \mathsf{Names}}{\Gamma \vdash D : \mathsf{Effect}} \qquad \frac{}{\Gamma \vdash \cdot : \mathsf{Names}} \qquad \frac{\Gamma \vdash \ell \qquad \Gamma \vdash D : Y}{\Gamma \vdash \ell, D : \mathsf{Effect}}$$

$$\frac{\Gamma \vdash a : \ell \qquad \Gamma \vdash D : Y}{\Gamma \vdash a, D : Y} \qquad \frac{\Gamma \vdash D : Y \qquad \Gamma \vdash L}{\Gamma \vdash D \backslash L : Y}$$

$\boxed{\Gamma \vdash \mu}$

$$\frac{\Gamma \vdash L : \mathsf{Effect} \qquad \Gamma \vdash D : \mathsf{Effect}}{\Gamma \vdash \langle L|D\rangle} \qquad \frac{\Gamma \vdash E : \mathsf{Effect}}{\Gamma \vdash [E]}$$

$\boxed{\Gamma\ @\ E}$

$$\frac{}{\cdot\ @\ E} \qquad \frac{\Gamma\ @\ F \qquad \mu_F : E \to F \qquad \Gamma \vdash A : K}{\Gamma, x :_{\mu_F} A\ @\ F} \qquad \frac{\Gamma\ @\ F \qquad \mu_F : E \to F}{\Gamma, 🔒_{\mu_F}\ @\ E}$$

$$\frac{\Gamma\ @\ E}{\Gamma, \varepsilon : Y\ @\ E} \qquad \frac{\Gamma\ @\ E}{\Gamma, a :_R \ell\ @\ E}$$

Fig. 11. Kinding and well-formedness rules for METN with extensions.

$\boxed{\Gamma \vdash D \equiv D'}$

$$\frac{}{D \equiv D} \qquad \frac{D_1 \equiv D_2 \qquad D_2 \equiv D_3}{D_1 \equiv D_3} \qquad \frac{D \equiv D'}{\ell, D \equiv \ell, D'} \qquad \frac{D \equiv D'}{a, D \equiv a, D'}$$

$$\frac{\ell \neq \ell'}{\ell, \ell', D \equiv \ell', \ell, D} \qquad \frac{}{a, \ell, D \equiv \ell, a, D} \qquad \frac{}{a, b, D \equiv b, a, D} \qquad \frac{}{a, a, D \equiv a, D} \qquad \frac{}{D\backslash\cdot \equiv D}$$

$$\frac{}{\cdot\backslash L \equiv \cdot} \qquad \frac{D \equiv D' \qquad L \equiv L'}{D\backslash L \equiv D'\backslash L'} \qquad \frac{}{(\ell, D)\backslash(\ell, L) \equiv D\backslash L} \qquad \frac{\ell \notin L}{(\ell, D)\backslash L \equiv \ell, D\backslash L}$$

$$\frac{}{(D\backslash L)\backslash L' \equiv D\backslash(L, L')} \qquad \frac{\Gamma \vdash D : \mathsf{Names} \qquad \Gamma \vdash D' : \mathsf{Names} \qquad D \equiv_{\mathbb{B}(\Gamma)} D'}{\Gamma \vdash D \equiv D'}$$

$\boxed{\Gamma \vdash E \equiv F}$

Mostly the same as that of D plus a few new rules for effect variables.

$$\frac{}{(\varepsilon, E)\backslash L \equiv \varepsilon\backslash L, E\backslash L} \qquad \frac{L \equiv L' \qquad E \equiv E'}{\varepsilon\backslash L, E \equiv \varepsilon\backslash L', E'}$$

$\boxed{\Gamma \vdash \mu \equiv \nu}$

$$\frac{E \equiv F}{[E] \equiv [F]} \qquad \frac{L \equiv L' \qquad D \equiv D'}{\langle L|D\rangle \equiv \langle L'|D'\rangle}$$

$\boxed{\Gamma \vdash A \equiv B}$

$$\frac{\mu \equiv \nu \qquad A \equiv B}{\mu A \equiv \nu B} \qquad \frac{A \equiv A' \qquad B \equiv B'}{A \to B \equiv A' \to B'} \qquad \frac{A \equiv B}{\forall \varepsilon^{Y}.A \equiv \forall \varepsilon^{Y}.B} \qquad \frac{A \equiv B}{\forall a^{\ell}.A \equiv \forall a^{\ell}.B}$$

Fig. 12. Type equivalence for Metn with extensions.

$\boxed{\Gamma \vdash E \leqslant F}$

$$\frac{E \equiv E'}{E \leqslant E'} \qquad \frac{E_1 \leqslant E_2 \qquad E_2 \leqslant E_3}{E_1 \leqslant E_3} \qquad \frac{E \leqslant E'}{\ell, E \leqslant \ell, E'}$$

$$\frac{\Gamma \vdash E : \mathsf{Names} \qquad \Gamma \vdash E' : \mathsf{Names} \qquad E \cap (\neg E') \equiv_{\mathbb{B}(\Gamma)} \emptyset}{\Gamma \vdash E \leqslant E'}$$

$$\frac{\Gamma \vdash F : \mathsf{Names} \qquad \Gamma \vdash F' : \mathsf{Names} \qquad \Gamma \vdash F \leqslant F'}{\Gamma \vdash F \leqslant E, F'} \qquad \frac{L' \leqslant L \qquad E \leqslant E'}{\varepsilon\backslash L, E \leqslant \varepsilon\backslash L', E'}$$

Fig. 13. Subeffecting for Metn with extensions.

$\boxed{\Gamma \vdash \mu \Rightarrow \nu \ @\ F}$

$$\textsc{MT-Abs}\quad \frac{\mu_F : E' \to F \qquad \Gamma \vdash E \leqslant E'}{\Gamma \vdash \Gamma \vdash [E] \Rightarrow \mu \ @\ F} \qquad \textsc{MT-Expand}\quad \frac{\Gamma \vdash (F - L) \equiv D', \_}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle D', L | D, D' \rangle \ @\ F} \qquad \textsc{MT-Shrink}\quad \frac{\Gamma \vdash (F - L) \equiv D', \_}{\Gamma \vdash \langle D', L | D, D' \rangle \Rightarrow \langle L | D \rangle \ @\ F}$$

$$\textsc{MT-Extend}\quad \frac{\Gamma \vdash \mathsf{labels}(D) \equiv \mathsf{labels}(D') \qquad \Gamma \vdash \mathsf{names}(D) \leqslant \mathsf{names}(D' + (F - L))}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle L | D' \rangle \ @\ F} \qquad \textsc{MT-Remove}\quad \frac{\Gamma \vdash \mathsf{labels}(L) \equiv \mathsf{labels}(L') \qquad \Gamma \vdash \mathsf{names}(L') \leqslant \mathsf{names}(L)}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle L' | D \rangle \ @\ F}$$

Since now we have both effect labels and handler names, we write $\mathsf{names}(E)$ for the part of handler names in E and $\mathsf{labels}(E)$ for the part of effect labels in E. We define it on normal forms of effect contexts as follows where $\Gamma \vdash E : \mathsf{Names}$ and $\Gamma \nvdash \varepsilon : \mathsf{Names}$.

$$\begin{array}{rcl} \mathsf{names}(\overline{\ell}, E) & = & E \\ \mathsf{names}(\overline{\ell}, \varepsilon \backslash L, E) & = & E \\ \mathsf{labels}(\overline{\ell}, E) & = & \overline{\ell} \\ \mathsf{labels}(\overline{\ell}, \varepsilon \backslash L, E) & = & \overline{\ell}, \varepsilon \backslash L \end{array}$$

We provide the transitive closure of modality transformation rules as follows.

$$\textsc{MT-Abs}\quad \frac{\mu_{F'} : E' \to F \qquad E \leqslant E'}{\Gamma \vdash [E] \Rightarrow \mu \ @\ F}$$

$$\textsc{MT-Rel}\quad \frac{\begin{array}{c} \Gamma \vdash \mathsf{labels}(L) \equiv \mathsf{labels}(L') \qquad \Gamma \vdash \mathsf{labels}(D) \equiv \mathsf{labels}(D') \\ \Gamma \vdash \mathsf{names}(L') \leqslant \mathsf{names}(L) \qquad \Gamma \vdash \mathsf{names}(D) \leqslant \mathsf{names}(D' + (F - L')) \\ \Gamma \vdash F - L' \equiv D_1, \_ \equiv D_2, \_ \end{array}}{\Gamma \vdash \langle D_1, L | D, D_1 \rangle \Rightarrow \langle D_2, L' | D', D_2 \rangle \ @\ F}$$

With concrete modalities, we write $\Gamma \vdash \mu_F \Rightarrow \nu_F$ for $\Gamma \vdash \mu \Rightarrow \nu \ @\ F$.

A.6 Typing Rules

Figure 14 gives the typing rules of Metn with extensions. We highlight rules for named handlers. We only provide the extended versions of named handler rules T-HandleName' and T-HandleName♠' which consider masking handler names. Rule T-Var uses the following auxiliary judgement.

$\boxed{\Gamma \vdash (\mu, A) \Rightarrow \nu \ @\ F}$

$$\frac{\Gamma \vdash A : \mathsf{Abs}}{\Gamma \vdash (\mu, A) \Rightarrow \nu \ @\ F} \qquad \frac{\Gamma \vdash \mu \Rightarrow \nu \ @\ F}{\Gamma \vdash (\mu, A) \Rightarrow \nu \ @\ F}$$

A.7 Operational Semantics

Runtime constructs and evaluation contexts are defined as follows. Note that we update the syntax of η which further influences the syntax of invoking and handling effects.

$$\boxed{\Gamma \vdash M : A \mathrel{@} E}$$

$$\textsc{T-Var}\quad \frac{\nu_F = \mathsf{locks}(\Gamma') : E \to F \qquad \Gamma \vdash (\mu, A) \Rightarrow \nu \mathrel{@} F}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash x : A \mathrel{@} E}$$

$$\textsc{T-Mod}\quad \frac{\mu_F : E \to F \qquad \Gamma, \text{🔒}_{\mu_F} \vdash V : A \mathrel{@} E}{\Gamma \vdash \mathbf{mod}_\mu\, V : \mu A \mathrel{@} F}$$

$$\textsc{T-LetMod}\quad \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F} \vdash V : \mu A \mathrel{@} E \qquad \Gamma, x :_{\nu_F \circ \mu_E} A \vdash N : B \mathrel{@} F}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = V\ \mathbf{in}\ N : B \mathrel{@} F}$$

$$\textsc{T-LetMod'}\quad \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F}, \overline{a : \ell}, \overline{\varepsilon : Y} \vdash V : \mu A \mathrel{@} E \qquad \Gamma, x :_{\nu_F \circ \mu_E} \forall\!\!\!\forall \overline{a^\ell}.\forall \overline{\varepsilon^Y}.A \vdash N : B \mathrel{@} F}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \lambda\!\!\!\lambda \overline{a^\ell}.\Lambda \overline{\varepsilon^Y}.x = V\ \mathbf{in}\ N : B \mathrel{@} F}$$

$$\textsc{T-Unit}\quad \frac{}{\Gamma \vdash () : 1 \mathrel{@} E}$$

$$\textsc{T-Abs}\quad \frac{\Gamma, x : A \vdash M : B \mathrel{@} E}{\Gamma \vdash \lambda x^A.M : A \to B \mathrel{@} E}$$

$$\textsc{T-App}\quad \frac{\Gamma \vdash M : A \to B \mathrel{@} E \qquad \Gamma \vdash N : A \mathrel{@} E}{\Gamma \vdash M\ N : B \mathrel{@} E}$$

$$\textsc{T-NAbs}\quad \frac{\Gamma, a : \ell \vdash V : A \mathrel{@} E}{\Gamma \vdash \lambda\!\!\!\lambda a^\ell.V : \forall\!\!\!\forall a^\ell.A \mathrel{@} E}$$

$$\textsc{T-NApp}\quad \frac{\Gamma \vdash M : \forall\!\!\!\forall a^\ell.A \mathrel{@} E \qquad \Gamma \ni b : \ell}{\Gamma \vdash M\ b : A[b/a] \mathrel{@} E}$$

$$\textsc{T-EAbs}\quad \frac{\Gamma, \varepsilon : Y \vdash V : A \mathrel{@} E}{\Gamma \vdash \Lambda \varepsilon^Y.V : \forall \varepsilon^Y.A \mathrel{@} E}$$

$$\textsc{T-EApp}\quad \frac{\Gamma \vdash M : \forall \varepsilon^Y.A \mathrel{@} E \qquad \Gamma \vdash F : Y}{\Gamma \vdash M\ F : A[F/\varepsilon] \mathrel{@} E}$$

$$\textsc{T-Mask}\quad \frac{\Gamma, \text{🔒}_{\langle L |\rangle_F} \vdash M : A \mathrel{@} F - L}{\Gamma \vdash \mathbf{mask}_L\ M : \langle L |\rangle A \mathrel{@} F}$$

$$\textsc{T-Do}\quad \frac{E = \ell, F \qquad \Sigma(\ell) \ni \mathsf{op} : A \twoheadrightarrow B \qquad \Gamma \vdash N : A \mathrel{@} E}{\Gamma \vdash \mathbf{do}\ \mathsf{op}\ N : B \mathrel{@} E}$$

$$\textsc{T-DoName}\quad \frac{E = a, F \qquad \Gamma \ni a : \ell \qquad \Sigma(\ell) \ni \mathsf{op} : A \twoheadrightarrow B \qquad \Gamma \vdash N : A \mathrel{@} E}{\Gamma \vdash \mathbf{do}_a\ \mathsf{op}\ N : B \mathrel{@} E}$$

$$\textsc{T-Handle}\quad \frac{\Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{\langle| \ell \rangle_F} \vdash M : A \mathrel{@} \ell, F \qquad \Gamma, x : \langle| \ell \rangle A \vdash N : B \mathrel{@} F \qquad [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N : B \mathrel{@} F]_i}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \mathrel{@} F}$$

$$\textsc{T-Handle}^\spadesuit\quad \frac{\Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{[\ell, E]_F} \vdash M : A \mathrel{@} \ell, E \qquad [E]_F \Rightarrow \mathbb{1}_F \qquad \Gamma, \text{🔒}_{[E]_F}, x : [\ell, E] A \vdash N : B \mathrel{@} E \qquad [\Gamma, \text{🔒}_{[E]_F}, p_i : A_i, r_i : [E](B_i \to B) \vdash N_i : B \mathrel{@} E]_i}{\Gamma \vdash \mathbf{handle}^\spadesuit\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \mathrel{@} F}$$

$$\textsc{T-HandleName'}\quad \frac{\Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, a :_\sharp \ell, \text{🔒}_{\langle| a \rangle_F} \vdash M : \langle a |\rangle A \mathrel{@} a, F \qquad \Gamma, x : A \vdash N : B \mathrel{@} F \qquad [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N : B \mathrel{@} F]_i}{\Gamma \vdash \mathbf{handle}_a\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \mathrel{@} F}$$

$$\textsc{T-HandleName}^\spadesuit\textsc{'}\quad \frac{\Sigma(\ell) = \{\mathsf{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, a :_\sharp \ell, \text{🔒}_{[a, E]_F} \vdash M : \langle a |\rangle A \mathrel{@} a, E \qquad [E]_F \Rightarrow \mathbb{1}_F \qquad \Gamma, \text{🔒}_{[E]_F}, x : [E] A \vdash N : B \mathrel{@} E \qquad [\Gamma, \text{🔒}_{[E]_F}, p_i : A_i, r_i : [E](B_i \to B) \vdash N_i : B \mathrel{@} E]_i}{\Gamma \vdash \mathbf{handle}_a^\spadesuit\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathsf{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \mathrel{@} F}$$

Fig. 14. Typing rules for Metn with extensions.

Handler Instances	h		
Instance contexts	Ω	$::=$	$\cdot \mid \Omega, h : \ell$
Decorations	$\eta ::= \cdot \mid h$		
Masks and Extensions	D	$::=$	$\cdots \mid h, D$
Effect Contexts	E	$::=$	$\cdots \mid h, E$
Terms	M	$::=$	$\cdots \mid M\ h$
Value normal forms	U	$::=$	$x \mid \lambda x^A.M \mid \Lambda a^I.V \mid \Lambda \varepsilon^Y.V \mid \mathbf{mod}_\mu\ U$
Evaluation Contexts	$\mathcal{E}$	$::=$	$[\,] \mid \mathcal{E}\ N \mid U\ \mathcal{E} \mid \mathcal{E}\ E \mid U\ \mathcal{E} \mid \mathbf{mod}_\mu\ \mathcal{E}$
		$\mid$	$\mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = \mathcal{E}\ \mathbf{in}\ M \mid \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda\overline{a^\ell}.\Lambda\overline{\varepsilon^Y}.x = \mathcal{E}\ \mathbf{in}\ M$
		$\mid$	$\mathcal{E}\ h \mid \mathbf{do}_\eta\ \mathtt{op}\ \mathcal{E} \mid \mathbf{mask}_L\ \mathcal{E} \mid \mathbf{handle}^\delta_\eta\ \mathcal{E}\ \mathbf{with}\ H$

Since handler instances h appear in types, we need to extend the equivalence and subtyping relations as well. We extend our boolean algebra $\mathbb{B}(\Gamma)$ to the power set of the set of all handler names and instances in Γ. We extend the non-equivalence relation used by the boolean algebra with the following two rules.

$$\textsc{NeqInstName}\quad \frac{}{\Gamma \vdash h \not\equiv a} \qquad\qquad \textsc{NeqInstInst}\quad \frac{h \neq h'}{\Gamma \vdash h \not\equiv h'}$$

Typing rules for runtime constructs are given below. Typing judgements essentially have form $\Omega \mid \Gamma \vdash M : A\ @\ E$. We omit Ω as it is globally provided and invariant.

$$\textsc{T-InstApp}\quad \frac{\Gamma \vdash M : \forall a^\ell.A\ @\ E \qquad \Omega \ni h : \ell}{\Gamma \vdash M\ h : A[h/a]\ @\ E}$$

$$\textsc{T-DoInst}\quad \frac{E = h, F \qquad \Omega \ni h : \ell \qquad \Sigma(\ell) \ni \mathtt{op} : A \twoheadrightarrow B \qquad \Gamma \vdash N : A\ @\ E}{\Gamma \vdash \mathbf{do}_h\ \mathtt{op}\ N : B\ @\ E}$$

$$\textsc{T-HandleInst'}\quad \frac{\Sigma(\ell) = \{\mathtt{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{\langle\!| h |\!\rangle_F} \vdash M : \langle\!| h |\!\rangle A\ @\ h, F \qquad \Gamma, x : A \vdash N : B\ @\ F \qquad [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N : B\ @\ F]_i}{\Gamma \vdash \mathbf{handle}_h\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathtt{op}_i\ p_i\ r_i \mapsto N_i\}_i : B\ @\ F}$$

$$\textsc{T-HandleInst}^{\spadesuit}\textsc{'}\quad \frac{\Sigma(\ell) = \{\mathtt{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{[h,E]_F} \vdash M : \langle\!| h |\!\rangle A\ @\ h, E \qquad [E]_F \Rightarrow \mathbb{1}_F \qquad \Gamma, \text{🔒}_{[E]_F}, x : [E]A \vdash N : B\ @\ E \qquad [\Gamma, \text{🔒}_{[E]_F}, p_i : A_i, r_i : [E](B_i \to B) \vdash N_i : B\ @\ E]_i}{\Gamma \vdash \mathbf{handle}^{\spadesuit}_h\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathtt{op}_i\ p_i\ r_i \mapsto N_i\}_i : B\ @\ F}$$

Figure 15 shows the operational semantics of Met.

As we have both dynamic and named handlers, we need to extend the definition of normal forms.

Definition A.1 (Normal Forms). We say a term M is in a normal form with respect to effect context E, if it is either in a value normal form $M = U$, or of form $M = \mathcal{E}[\mathbf{do}_h\ \mathtt{op}\ U]$ for $h \in E$, or of form $M = \mathcal{E}[\mathbf{do}\ \mathtt{op}\ U]$ and $n\text{–free}(\ell, \mathcal{E})$.

The predicate $n\text{–free}(\ell, \mathcal{E})$ is defined as follows, similar to the definition in Met. We have $n\text{–free}(\ell, \mathcal{E})$ if there are n unmasked handlers that handle the effect ℓ in E.

$$
\begin{array}{lrl}
\text{E-App} & (\lambda x^{A}.M)\,U \rightsquigarrow & M[U/x] \\
\text{E-NApp} & (\text{ƛ} a^{\ell}.V)\,h \rightsquigarrow & V[h/a] \\
\text{E-EApp} & (\Lambda \varepsilon^{Y}.V)\,E \rightsquigarrow & V[E/\varepsilon] \\
\text{E-LetMod} & \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ x = \mathbf{mod}_{\mu}\,U\ \mathbf{in}\ M \rightsquigarrow & M[U/x] \\
\text{E-LetMod'} & \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ \text{ƛ}\overline{a^{\ell}}.\Lambda\overline{\varepsilon^{Y}}.x = \mathbf{mod}_{\mu}\,U\ \mathbf{in}\ M \rightsquigarrow & M[(\text{ƛ}\overline{a^{\ell}}.\Lambda\overline{\varepsilon^{Y}}.U)/x] \\
\text{E-Mask} & \mathbf{mask}_{L}\,U \rightsquigarrow & \mathbf{mod}_{\langle L|\rangle}\,U \\
\text{E-Ret} & \mathbf{handle}\ U\ \mathbf{with}\ H \rightsquigarrow & N[(\mathbf{mod}_{\langle|\ell\rangle}\,U)/x], \\
& & \quad \text{where } (\mathbf{return}\ x \mapsto N) \in H \text{ and } H \propto \ell \\
\text{E-Op} & \mathbf{handle}\ \mathcal{E}[\mathbf{do}\ \mathsf{op}\ U]\ \mathbf{with}\ H \rightsquigarrow & N[U/p, (\lambda y.\mathbf{handle}\ \mathcal{E}[y]\ \mathbf{with}\ H)/r], \\
& & \quad \text{where } (\mathsf{op} : \_ \twoheadrightarrow \_) \in \Sigma(\ell), 0\text{–free}(\ell, \mathcal{E}) \text{ and } H \propto \ell \\
\text{E-Ret}^{\spadesuit} & \mathbf{handle}^{\spadesuit}\ U\ \mathbf{with}\ H \rightsquigarrow & N[(\mathbf{mod}_{[\ell,E]}\,U)/x], \\
& & \quad \text{where } (\mathbf{return}\ x \mapsto N) \in H \text{ and } H \propto \ell \\
\text{E-Op}^{\spadesuit} & \mathbf{handle}^{\spadesuit}\ \mathcal{E}[\mathbf{do}\ \mathsf{op}\ U]\ \mathbf{with}\ H \rightsquigarrow & \\
& & N[U/p, (\mathbf{mod}_{[E]}\ (\lambda y.\mathbf{handle}^{\spadesuit}\ \mathcal{E}[y]\ \mathbf{with}\ H))/r] \\
& & \quad \text{where } (\mathsf{op} : \_ \twoheadrightarrow \_) \in \Sigma(\ell), 0\text{–free}(\ell, \mathcal{E}) \text{ and } H \propto \ell \\
\text{E-Gen} & \mathbf{handle}_{a}\ M\ \mathbf{with}\ H \mid \Omega \rightsquigarrow & \mathbf{handle}_{h}\ M[h/a]\ \mathbf{with}\ H \mid \Omega, h : \ell \\
& & \quad \text{where } h \text{ fresh and } H \propto \ell \\
\text{E-NRet} & \mathbf{handle}_{h}\ (\mathbf{mod}_{\langle h|\rangle}\ U)\ \mathbf{with}\ H \rightsquigarrow & N[U/x], \quad \text{where } (\mathbf{return}\ x \mapsto N) \in H \\
\text{E-NOp} & \mathbf{handle}_{h}\ \mathcal{E}[\mathbf{do}_{h}\ \mathsf{op}\ U]\ \mathbf{with}\ H \rightsquigarrow & N[U/p, (\lambda y.\mathbf{handle}_{h}\ \mathcal{E}[y]\ \mathbf{with}\ H)/r], \\
& & \quad \text{where } (\mathsf{op}\ p\ r \mapsto N) \in H \\
\text{E-NRet}^{\spadesuit} & \mathbf{handle}_{h}\ (\mathbf{mod}_{\langle h|\rangle}\ U)\ \mathbf{with}\ H \rightsquigarrow & N[(\mathbf{mod}_{[E]}\ U)/x] \\
& & \quad \text{where } (\mathbf{return}\ x \mapsto N) \in H \\
\text{E-NOp}^{\spadesuit} & \mathbf{handle}^{\spadesuit}_{h}\ \mathcal{E}[\mathbf{do}_{h}\ \mathsf{op}\ U]\ \mathbf{with}\ H \rightsquigarrow & \\
& & N[U/p, (\mathbf{mod}_{[E]}\ (\lambda y.\mathbf{handle}^{\spadesuit}_{h}\ \mathcal{E}[y]\ \mathbf{with}\ H))/r] \\
& & \quad \text{where } (\mathsf{op}\ p\ r \mapsto N) \in H \\
\text{E-Lift} & \mathcal{E}[M] \rightsquigarrow & \mathcal{E}[N], \quad \text{if } M \rightsquigarrow N
\end{array}
$$

Fig. 15. Operational semantics for Metn with extensions.

$$
\frac{}{0\text{–free}(\ell, [\,])}
\qquad
\frac{n\text{–free}(\ell, \mathcal{E})}{n\text{–free}(\ell, \mathbf{do}_{\eta}\ \mathsf{op}\ \mathcal{E})}
\qquad
\frac{n\text{–free}(\ell, \mathcal{E}) \qquad \mathsf{count}(\ell; L) = m}{(n+m)\text{–free}(\ell, \mathbf{mask}_{L}\ \mathcal{E})}
$$

$$
\frac{(n+1)\text{–free}(\ell, \mathcal{E}) \qquad H \propto \ell}{n\text{–free}(\ell, \mathbf{handle}^{\delta}\ \mathcal{E}\ \mathbf{with}\ H)}
\qquad
\frac{n\text{–free}(\ell, \mathcal{E}) \qquad H \not\propto \ell}{n\text{–free}(\ell, \mathbf{handle}^{\delta}\ \mathcal{E}\ \mathbf{with}\ H)}
$$

$$
\frac{n\text{–free}(\ell, \mathcal{E})}{n\text{–free}(\ell, \mathbf{handle}^{\delta}_{a}\ \mathcal{E}\ \mathbf{with}\ H)}
$$

B Meta Theory and Proofs for Metn with Extensions

We provide meta theory and proofs for Metn introduced in Sections 3 and 4 and fully specified in Appendix A. The proofs are based on the proofs for Met in Tang et al. [26]. Though the syntax for the common parts of Metn and Met are slightly different, it is obvious how to adapt the proofs of Met to Metn. Thus, we focus on proving the new parts of Metn relevant to named handlers.

B.1 Properties of the Mode Theory of Metn

Our proofs for type soundness rely on some properties of the mode theory of Metn (full mode theory of Met is given in Appendix A.5), following the properties of the mode theory of Met.

First, the mode theory of Metn should form a double category. The effect contexts and subeffecting obviously form a double category generated by a poset. The effect contexts (objects) and modalities (horizontal morphisms) also form a category since modality composition possesses associativity and identity. We have the following lemma.

Lemma B.1 (Modes and modalities form a category). *Modes and modalities form a category with the identity morphism* $\mathbb{1}_E = \langle\rangle_E : E \to E$ *and the morphism composition* $\mu_F \circ \nu_{F'}$ *such that*

(1) Identity: $\mathbb{1}_F \circ \mu_F = \mu_F = \mu_F \circ \mathbb{1}_E$ *for* $\mu_F : E \to F$.
(2) Associativity: $(\mu_{E_1} \circ \nu_{E_2}) \circ \xi_{E_3} = \mu_{E_1} \circ (\nu_{E_2} \circ \xi_{E_3})$ *for* $\mu_{E_1} : E_2 \to E_1$, $\nu_{E_2} : E_3 \to E_2$, *and* $\xi_{E_3} : E \to E_3$.

Proof. By inlining the definitions of modalities and checking each case. □

As in Met, we need to extend the modality transformation relation a bit for meta theory and proofs. We write $\Gamma \vdash \mu_F \Rightarrow \nu_F$ if $\Gamma \vdash \mu \Rightarrow \nu \; @ \; F$. We extend this judgement to allow $\Gamma \vdash \mu_F \Rightarrow \nu_{F'}$ where $F \leqslant F'$ and add one new rule MT-Mono.

$$\textsc{MT-Mono}\quad \frac{\Gamma \vdash F \leqslant F'}{\Gamma \vdash \mu_F \Rightarrow \mu_{F'}}$$

We show that modality transformations are 2-cells in the double category.

Lemma B.2 (Modality transformations are 2-cells). *If* $\mu_F \Rightarrow \nu_{F'}$, $\mu_F : E \to F$, *and* $\nu_{F'} : E' \to F'$, *then* $E \leqslant E'$ *and* $F \leqslant F'$. *Moreover, the transformation relation is closed under vertical and horizontal composition as shown by the following admissible rules.*

$$\frac{\mu_{F_1} \Rightarrow \nu_{F_2} \qquad \nu_{F_2} \Rightarrow \xi_{F_3}}{\mu_{F_1} \Rightarrow \xi_{F_3}} \qquad \frac{\mu_F \Rightarrow \mu'_{F'} \qquad \nu_E \Rightarrow \nu'_{E'} \qquad \mu_F : E \to F \qquad \mu'_{F'} : E' \to F'}{\mu_F \circ \nu_E \Rightarrow \mu'_{F'} \circ \nu'_{E'}}$$

Proof. We first take the transitive closures of the modality transformation rules.

$$\textsc{MT-Abs}\quad \frac{\mu_{F'} : E' \to F' \qquad E \leqslant E' \qquad F \leqslant F'}{[E]_F \Rightarrow \mu_{F'}}$$

$$\textsc{MT-Rel}\quad \frac{\begin{array}{c}\Gamma \vdash \mathsf{labels}(L) \equiv \mathsf{labels}(L') \qquad \Gamma \vdash \mathsf{labels}(D) \equiv \mathsf{labels}(D') \\ \Gamma \vdash \mathsf{names}(L') \leqslant \mathsf{names}(L) \qquad \Gamma \vdash \mathsf{names}(D) \leqslant \mathsf{names}(D' + (F' - L')) \\ \Gamma \vdash F' - L' \equiv D_1, \_ \equiv D_2, \_ \qquad \Gamma \vdash F \leqslant F'\end{array}}{\Gamma \vdash \langle D_1, L | D, D_1\rangle_F \Rightarrow \langle D_2, L' | D', D_2\rangle_{F'}}$$

Vertical composition follows directly from the fact that we take the transitive closure. Horizontal compositions follows from a case analysis on shapes of modalities being composed. The most nontrivial case is when all of μ_F, ν_E, $\mu'_{F'}$, and $\nu'_{E'}$ are relative modalities. The new part compared to Met is that we can remove names from L and add names to D' in MT-Rel. It is obvious that horizontal composition preserves this property. □

Beyond being a double category, we show some extra properties. The most important one is that horizontal morphisms (sub-effecting) act functorially on vertical ones (modalities). In other words, the action of μ on effect contexts gives a total monotone function.

Lemma B.3 (Monotone modalities). *If $\mu_F : E \to F$ and $F \leqslant F'$, then $\mu_{F'} : E' \to F'$ with $E \leqslant E'$.*

Proof. By definition. ◻

Lemma 3.1 (Soundness of modality transformation). *For modality transformation $\Gamma \vdash \mu \Rightarrow \nu \ @\ F$, we have $\mu(F') \leqslant \nu(F')$ for all F' with $F \leqslant F'$.*

Proof. We prove it for the full version of rules in Appendix A.5. It is obvious that taking the transitive closure does not break this lemma. We only need to show the transformation given by each rule satisfies the semantics.

Case MT-Abs. Follow from Lemma B.3.
Case MT-Extend and MT-Remove. Obvious.
Case MT-Expand. Since $(F - L) \equiv D', E$, for any $F \leqslant F'$ we have $(F' - L) \equiv D', E'$ for some E'. Both sides act on F' give D, D', E'.
Case MT-Shrink. Similar to the above case.

◻

We state some properties of the mode theory as the following lemmas for easier references in proofs. Most of them directly follow from the definition.

Lemma B.4 (Vertical composition). *If $\mu_{F_1} \Rightarrow \nu_{F_2}$ and $\nu_{F_2} \Rightarrow \xi_{F_3}$, then $\mu_{F_1} \Rightarrow \xi_{F_3}$.*

Proof. Follow from Lemma B.2 ◻

Lemma B.5 (Horizontal composition). *If $\mu_F : E \to F$, $\mu'_{F'} : E' \to F'$, $\mu_F \Rightarrow \mu'_{F'}$, and $\nu_E \Rightarrow \nu'_{E'}$, then $\mu_F \circ \nu_E \Rightarrow \mu'_{F'} \circ \nu'_{E'}$.*

Proof. Follow from Lemma B.2 ◻

Lemma B.6 (Monotone modality transformation). *If $\mu_F \Rightarrow \nu_F$ and $F \leqslant F'$, then $\mu_{F'} \Rightarrow \nu_{F'}$.*

Proof. Obvious by looking at the transitive closure rules of modality transformation MT-Abs and MT-Rel in Appendix A.5. For MT-Abs, we use Lemma B.3. ◻

Lemma B.7 (Asymmetric reflexivity of modality transformation). *If $F \leqslant F'$ and $\mu_F : E \to F$, then $\mu_F \Rightarrow \mu_{F'}$.*

Proof. By definition. ◻

B.2 Lemmas for the Calculus

We prove structural and substitution lemmas for Metn as well as some other auxiliary lemmas for proving type soundness.

Lemma B.8 (Canonical forms).

1. *If $\vdash U : \mu A \ @\ E$, then U is of shape $\mathbf{mod}_\mu\, U'$.*
2. *If $\vdash U : A \to B \ @\ E$, then U is of shape $\lambda x^A.M$.*
3. *If $\vdash U : \forall \varepsilon^Y.A \ @\ E$, then U is of shape $\Lambda \varepsilon^Y.V$.*
4. *If $\vdash U : \mathbb{V} a^\ell.A \ @\ E$, then U is of shape $\lambda\!\!\!\lambda a^\ell.V$.*
5. *If $\vdash U : 1 \ @\ E$, then U is $()$.*

Proof. Directly follows from the typing rules. □

In order to define the lock weakening lemma, we first define a context update operation $\llparenthesis\Gamma\rrparenthesis_{F'}$ which gives a new context derived from updating the indexes of all locks and variable bindings in Γ such that $\mathsf{locks}(\llparenthesis\Gamma\rrparenthesis_{F'}) : \_ \to F'$.

$$
\begin{array}{rcl}
\llparenthesis\cdot\rrparenthesis_F & = & \cdot \\
\llparenthesis \text{🔒}_{[E]_{F'}}, \Gamma' \rrparenthesis_F & = & \text{🔒}_{[E]_F}, \Gamma' \\
\llparenthesis \text{🔒}_{\langle L|D\rangle_{F'}}, \Gamma' \rrparenthesis_F & = & \text{🔒}_{\langle L|D\rangle_F}, \llparenthesis\Gamma'\rrparenthesis_{D+(F-L)} \\
\llparenthesis x :_{\mu_{F'}} A, \Gamma' \rrparenthesis_F & = & x :_{\mu_F} A, \llparenthesis\Gamma'\rrparenthesis_F \\
\llparenthesis \varepsilon : Y, \Gamma' \rrparenthesis_F & = & \varepsilon : Y, \llparenthesis\Gamma'\rrparenthesis_F \\
\llparenthesis a : \ell, \Gamma' \rrparenthesis_F & = & a : \ell, \llparenthesis\Gamma'\rrparenthesis_F
\end{array}
$$

We have the following lemma showing that the index update operation preserves the $\mathsf{locks}(-)$ operation except for updating the index.

Lemma B.9 (Index update preserves composition). *If $\mu_F = \mathsf{locks}(\Gamma) : E \to F$, $F \leqslant F'$, and $\mathsf{locks}(\llparenthesis\Gamma\rrparenthesis_{F'}) : E' \to F'$, then $\mathsf{locks}(\llparenthesis\Gamma\rrparenthesis_{F'}) = \mu_{F'}$.*

Proof. By straightforward induction on the context and using the property that $(\mu\circ\nu)_F = \mu_F \circ \nu_E$ for $\mu_F : E \to F$. □

Corollary B.10 (Index update preserves transformation). *If $\mathsf{locks}(\Gamma) : E \to F$, $F \leqslant F'$, and $\mathsf{locks}(\llparenthesis\Gamma\rrparenthesis_{F'}) : E' \to F'$, then $\mathsf{locks}(\Gamma) \Rightarrow \mathsf{locks}(\llparenthesis\Gamma\rrparenthesis_{F'})$.*

Proof. Immediately follow from Lemma B.9 and Lemma B.7. □

We have the following structural lemmas.

Lemma B.11 (Structural rules). *The following structural rules are admissible.*

1. *Variable weakening.*

$$\frac{\Gamma, \Gamma' \vdash M : B \mathrel{@} E \qquad \Gamma, x :_{\mu_F} A, \Gamma' \mathrel{@} E}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash M : B \mathrel{@} E}$$

2. *Variable swapping.*

$$\frac{\Gamma, x :_{\mu_F} A, y :_{\nu_F} B, \Gamma' \vdash M : A' \mathrel{@} E}{\Gamma, y :_{\nu_F} B, x :_{\mu_F} A, \Gamma' \vdash M : A' \mathrel{@} E}$$

3. *Lock weakening.*

$$\frac{\Gamma, \text{🔒}_{\mu_F}, \Gamma' \vdash M : A \mathrel{@} E \qquad \mu_F \Rightarrow \nu_F \qquad \nu_F : F' \to F \qquad \mathsf{locks}(\llparenthesis\Gamma'\rrparenthesis_{F'}) : E' \to F'}{\Gamma, \text{🔒}_{\nu_F}, \llparenthesis\Gamma'\rrparenthesis_{F'} \vdash M : A \mathrel{@} E'}$$

4. *Type variable weakening.*

$$\frac{\Gamma, \Gamma' \vdash M : B \mathrel{@} E}{\Gamma, \varepsilon : Y, \Gamma' \vdash M : B \mathrel{@} E}$$

5. *Type variable swapping.*

$$\frac{\Gamma_1, \Gamma_2, \varepsilon : Y, \Gamma_3 \vdash M : A \mathrel{@} E}{\Gamma_1, \varepsilon : Y, \Gamma_3 \vdash M : A \mathrel{@} E} \qquad \frac{\alpha \notin \mathsf{ftv}(\Gamma_2) \qquad \Gamma_1, \varepsilon : Y, \Gamma_3 \vdash M : A \mathrel{@} E}{\Gamma_1, \Gamma_2, \varepsilon : Y, \Gamma_3 \vdash M : A \mathrel{@} E}$$

6. *Name weakening.*

$$\frac{\Gamma, \Gamma' \vdash M : B \mathbin{@} E}{\Gamma, a : (\ell, R), \Gamma' \vdash M : B \mathbin{@} E}$$

Proof. Follow from the proofs for structural rules of Met in Tang et al. [26]. The new cases relevant to named handlers follow from similar induction patterns of existing cases in Met. □

The following lemma reflects the intuition that pure values can be used in any effect context.

Lemma B.12 (Pure Promotion). *The following promotion rule is admissible.*

$$\frac{\begin{array}{c}\Gamma_1, \Gamma \vdash V : A \mathbin{@} E \qquad \Gamma_1 \vdash A : \mathsf{Abs} \\ \mathsf{locks}(\Gamma) : E \to F \qquad \mathsf{locks}(\Gamma') : E' \to F \qquad \mathsf{fv}(V) \cap \mathsf{dom}(\Gamma') = \emptyset\end{array}}{\Gamma_1, \Gamma' \vdash V : A \mathbin{@} E'}$$

Proof. Follow from the proof for the same lemma of Met. □

Lemma B.13 (Substitution). *The following substitution rules are admissible.*

1. *Preservation of kinds under effect substitution.*

$$\frac{\Gamma \vdash E : Y \qquad \Gamma, \varepsilon : Y, \Gamma' \vdash B : K}{\Gamma, \Gamma' \vdash B[E/\varepsilon] : K}$$

2. *Preservation of types under effect substitution.*

$$\frac{\Gamma \vdash E : Y \qquad \Gamma, \varepsilon : Y, \Gamma' \vdash M : B \mathbin{@} F}{\Gamma, \Gamma' \vdash M[E/\varepsilon] : B[E/\varepsilon] \mathbin{@} F}$$

3. *Preservation of types under value substitution.*

$$\frac{\Gamma, \blacksquare_{\mu_F} \vdash V : A \mathbin{@} F' \qquad \Gamma, x :_{\mu_F} A, \Gamma' \vdash M : B \mathbin{@} E}{\Gamma, \Gamma' \vdash M[V/x] : B \mathbin{@} E}$$

4. *Preservation of kinds under name substitution.*

$$\frac{\Gamma \ni b : \ell \qquad \Gamma, a : \ell, \Gamma' \vdash B : K}{\Gamma, \Gamma' \vdash B[b/a] : K}$$

5. *Preservation of types under name substitution.*

$$\frac{\Gamma \ni b : \ell \qquad \Gamma, a : \ell, \Gamma' \vdash M : B \mathbin{@} F}{\Gamma, \Gamma' \vdash M[b/a] : B[b/a] \mathbin{@} F}$$

Proof.
1,2,4,5. Follow from straightforward induction.
3. Follow from the proof for the same lemma of Met. The new cases relevant to named handlers follow from similar induction patterns of existing cases in Met. □

B.3 Progress

Theorem 3.3 (Progress). *If $\Omega \mid \cdot \vdash M : A \mathbin{@} E$, then either there exists N such that $M \mid \Omega \rightsquigarrow N \mid \Omega'$ or M is in a normal form with respect to E.*

Proof. By induction on the typing derivation $\Omega \mid \cdot \vdash M : A \mathbin{@} E$. Most cases follow from the proof for progress of Met in Tang et al. [26]. We only show new cases relevant to named handlers.

Case T-NApp. $M\ a$. Follow from IH on M, Lemma B.8, and E-NApp.

Case T-DoName. $\mathbf{do}_{\ h}\ \mathsf{op}\ M$. We have $h \in E$. Either M is reducible or the whole term is in a normal form with respect to E.

Case T-HandleName' and T-HandleName♠'. By E-Gen.

Case T-HandleInst' and T-HandleInst♠'. $\mathbf{handle}^{\spadesuit}_{h}\ M\ \mathbf{with}\ H$.

- Case M is reducible. Trivial.
- Case M is a value. By E-Ret.
- Case $M = \mathcal{E}[\mathbf{do}_{h'}\ \ell\ U]$. Since M is not reducible itself, there is no handler for h' in $\mathcal{E}$. Because of well-typedness, we have either $h = h'$ or $h' \in E$. If $h = h'$, by E-Op. Otherwise, it is in a normal form with respect to E.

□

B.4 Subject Reduction

Theorem 3.4 (Subject Reduction). *If $\Omega \mid \Gamma \vdash M : A \mathbin{@} E$ and $M \mid \Omega \rightsquigarrow N \mid \Omega'$, then $\Omega' \mid \Gamma \vdash N : A \mathbin{@} E$.*

Proof. By induction on the typing derivation $\Omega \mid \Gamma \vdash M : A \mathbin{@} E$. Most cases follow from the proof for subject reduction of Met. We only show new cases relevant to named handlers.

Case

T-Letmod'

$$\dfrac{\nu_F : E \longrightarrow F \qquad \vdash V : \mathsf{Complex} \qquad \Gamma, \text{🔒}_{\nu_F}, \overline{a : \ell}, \overline{\varepsilon : Y} \vdash V : \mu A \mathbin{@} E\ (1) \qquad \Gamma, x :_{\nu_F \circ \mu_E} \forall \overline{a^{\ell}}.\forall \overline{\varepsilon^{Y}}.A \vdash N : B \mathbin{@} F\ (2)}{\Gamma \vdash \mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ \lambda \overline{a^{\ell}}.\Lambda \overline{\varepsilon^{Y}}.x = V\ \mathbf{in}\ N : B \mathbin{@} F}$$

Similar to the case for T-Letmod'. By case analysis on the reduction.

Case E-Lift with $V \rightsquigarrow V'$. By IH on (1) and reapplying T-Letmod'.

Case E-Letmod'. We have $V = \mathbf{mod}_{\mu}\ U$ and

$$\mathbf{let}_{\nu}\ \mathbf{mod}_{\mu}\ \lambda \overline{a^{\ell}}.\Lambda \overline{\varepsilon^{Y}}.x = \mathbf{mod}_{\mu}\ U\ \mathbf{in}\ N \rightsquigarrow N[(\lambda \overline{a^{\ell}}.\Lambda \overline{\varepsilon^{Y}}.U)/x].$$

Inversion on (1) gives

$$\Gamma, \text{🔒}_{\nu_F}, \overline{a : \ell}, \overline{\varepsilon : Y}, \text{🔒}_{\mu_E} \vdash U : \mu A \mathbin{@} E'.$$

where $\mu_E : E' \longrightarrow E$. Swapping variable and name bindings with locks gives

$$\Gamma, \text{🔒}_{\nu_F}, \text{🔒}_{\mu_E}, \overline{a : \ell}, \overline{\varepsilon : Y} \vdash U : A \mathbin{@} E'.$$

By context equivalence, we have

$$\Gamma, \text{🔒}_{\nu_F \circ \mu_E}, \overline{a : \ell}, \overline{\varepsilon : Y} \vdash U : A \mathbin{@} E'.$$

where $\nu_F \circ \mu_E : E' \longrightarrow F$. By T-TAbs we have

$$\Gamma, \text{🔒}_{\nu_F \circ \mu_E} \vdash \lambda \overline{a^{\ell}}.\Lambda \overline{\varepsilon^{Y}}.U : \forall \overline{a^{\ell}}.\forall \overline{\varepsilon^{Y}}.A \mathbin{@} E'.$$

By Lemma B.13.3 and (2), we have

$$\Gamma \vdash N[(\lambda \overline{a^{\ell}}.\Lambda \overline{\varepsilon^{Y}}.U)/x] : B \mathbin{@} F.$$

Case T-NAbs. Impossible as there is no further reduction.

Case

$$\frac{\Gamma \vdash M : \forall a^{\ell}.A \mathbin{@} E\ (1) \qquad \Gamma \ni b : \ell\ (2)}{\Gamma \vdash M\, b : A[b/a] \mathbin{@} E}\ \text{T-NApp}$$

By case analysis on the reduction.

Case E-Lift with $M \mid \Omega \rightsquigarrow N \mid \Omega'$. By IH on (1) and reapplying T-NApp.

Case E-NApp. We have $M = \lambda a^{\ell}.V$ and

$$(\lambda a^{\ell}.V)\, b \rightsquigarrow V[b/a].$$

Inversion on (1) gives

$$\Gamma, a : \ell \vdash V : A \mathbin{@} E.$$

Then by Lemma B.13.5 on (2), we have

$$\Gamma \vdash V[b/a] : A[b/a] \mathbin{@} E.$$

Case T-DoName. Impossible as there is no further reduction.

Case T-DoInst. The only way to reduce is by E-Lift. Follow from IH and reapplying T-DoInst.

Case

$$\frac{\begin{array}{cc}\Sigma(\ell) = \{\mathtt{op}_i : A_i \twoheadrightarrow B_i\}_i & \Gamma, a :_{\sharp} \ell, \blacklozenge_{\langle| a \rangle_F} \vdash M : \langle a |\!\rangle A \mathbin{@} a, F \\ \Gamma, x : A \vdash N : B \mathbin{@} F & [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N : B \mathbin{@} F]_i\end{array}}{\Gamma \vdash \mathbf{handle}_a\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathtt{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}\ \text{T-HandleName'}$$

The only way to reduce is by E-Gen. We have

$$\mathbf{handle}_a\ M\ \mathbf{with}\ H \mid \Omega \quad\rightsquigarrow\quad \mathbf{handle}_h\ M[h/a]\ \mathbf{with}\ H \mid \Omega, h : \ell$$

Follow from T-HandleInst using the new instance context $\Omega, h : \ell$.

Case T-HandleName♠'. Similar to the above.

Case

$$\frac{\begin{array}{cc}\Sigma(\ell) = \{\mathtt{op}_i : A_i \twoheadrightarrow B_i\}_i & \Gamma, \blacklozenge_{\langle| h \rangle_F} \vdash M : \langle h |\!\rangle A \mathbin{@} h, F\ (1) \\ \Gamma, x : A \vdash N : B \mathbin{@} F\ (2) & [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N : B \mathbin{@} F\ (3)]_i\end{array}}{\Gamma \vdash \mathbf{handle}_h\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathtt{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}\ \text{T-HandleInst'}$$

By case analysis on the reduction.

Case E-Lift with $M \rightsquigarrow M'$. By IHs and reapplying T-HandleInst'.

Case E-NRet. We have $M = \mathbf{mod}_{\langle h |\!\rangle}\ U$ and

$$\mathbf{handle}_h\ (\mathbf{mod}_{\langle h |\!\rangle}\ U)\ \mathbf{with}\ H \rightsquigarrow N[U/x].$$

By inversion on (1), $\langle| h \rangle \circ \langle h |\!\rangle = \langle\rangle$, and context equivalence we have

$$\Gamma \vdash U : A \mathbin{@} F.$$

Then by (2) and Lemma B.13.3 we have

$$\Gamma \vdash N[U/x] : B \mathbin{@} F.$$

Case E-NOp. We have $M = \mathcal{E}[\mathbf{do}_h\ \mathtt{op}_j\ U]$, $H \ni \mathtt{op}_j\ p_j\ r_j \mapsto N_j$, and

$$\mathbf{handle}_h\ M\ \mathbf{with}\ H \rightsquigarrow N_j[U/p_j, (\lambda y.\mathbf{handle}_h\ \mathcal{E}[y]\ \mathbf{with}\ H)/r_j].$$

By inversion on $\mathbf{do}_h\ \mathtt{op}_j\ U$ we have

$$\Gamma, \blacklozenge_{\langle| h \rangle_F} \vdash U : A_j \mathbin{@} h, F.$$

By A_j has kind Abs, Lemma B.12, and context equivalence, we have

$$\Gamma \vdash U : A_j \ @\ F\ (4)$$

Observe that B_j being pure allows $y : B_j$ to be accessed in any context. By (1) and a straightforward induction on $\mathcal{E}$ we have

$$\Gamma, y : B_j, \text{🔒}_{\langle\!| h |\!\rangle_F} \vdash \mathcal{E}[y] : A \ @\ h, F.$$

Then by T-HandleInst' and T-Abs we have

$$\Gamma \vdash \lambda y.\mathbf{handle}_h\ \mathcal{E}[y]\ \mathbf{with}\ H : B_j \to B \ @\ F\ (5).$$

Finally, by (3), (4), (5), and Lemma B.13.3 we have

$$\Gamma \vdash N_j[U/p, (\lambda y.\mathbf{handle}_h\ \mathcal{E}[y]\ \mathbf{with}\ H)/r] : B \ @\ F.$$

Case

T-InstHandle$^{\spadesuit}$'

$$\frac{\Sigma(\ell) = \{\mathtt{op}_i : A_i \twoheadrightarrow B_i\}_i \qquad [E]_F \Rightarrow \mathbb{1}_F \qquad \Gamma, \text{🔒}_{[h,E]_F} \vdash M : \langle h |\!\rangle A \ @\ h, E\ (1) \qquad \Gamma, \text{🔒}_{[E]_F}, x : [E]A \vdash N : B \ @\ E\ (2) \qquad [\Gamma, \text{🔒}_{[E]_F}, p_i : A_i, r_i : [E](B_i \to B) \vdash N_i : B \ @\ E\ (3)]_i}{\Gamma \vdash \mathbf{handle}^{\spadesuit}_h\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\mathtt{op}_i\ p_i\ r_i \mapsto N_i\}_i : B \ @\ F}$$

By case analysis on the reduction.

Case E-Lift with $M \leadsto M'$. By IHs and reapplying T-InstHandle$^{\spadesuit}$.

Case E-NRet$^{\spadesuit}$. We have $M = \mathbf{mod}_{\langle h |\!\rangle}\ U$ and

$$\mathbf{handle}\ (\mathbf{mod}_{\langle h |\!\rangle}\ U)\ \mathbf{with}\ H \leadsto N[(\mathbf{mod}_{[E]} U)/x].$$

By (1), $[E]_F \circ [E]_E = [E]_F = [h, E]_F \circ \langle h |\!\rangle_{h,E}$, and context equivalence, we have

$$\Gamma, \text{🔒}_{[E]_F}, \text{🔒}_{[E]_E} \vdash U : A \ @\ E.$$

By T-Mod, we have

$$\Gamma, \text{🔒}_{[E]_F} \vdash \mathbf{mod}_{[E]}\ U : [E]A \ @\ E.$$

Then by (2) and Lemma B.13.3 we have

$$\Gamma, \text{🔒}_{[E]_F} \vdash N[(\mathbf{mod}_{[E]}\ U)/x] : B \ @\ E.$$

By $[E]_F \Rightarrow \mathbb{1}_F$ and Lemma B.11.3 we have

$$\Gamma \vdash N[(\mathbf{mod}_{[E]}\ U)/x] : B \ @\ F.$$

Case E-NOp$^{\spadesuit}$. We have $M = \mathcal{E}[\mathbf{do}_h\ \mathtt{op}_j\ U]$, $\ell_j\ p_j\ r_j \mapsto N_j$, and

$$\mathbf{handle}^{\spadesuit}_h\ M\ \mathbf{with}\ H \leadsto N_j[U/p, (\mathbf{mod}_{[E]}\ (\lambda y.\mathbf{handle}^{\spadesuit}_h\ \mathcal{E}[y]\ \mathbf{with}\ H))/r].$$

By inversion on $\mathbf{do}_h\ \mathtt{op}_j\ U$, we have

$$\Gamma, \text{🔒}_{[h,E]_F} \vdash U : A_j \ @\ h, E.$$

By the fact that A_j is pure, Lemma B.12, and context equivalence, we have

$$\Gamma, \text{🔒}_{[E]_F} \vdash U : A_j \ @\ E\ (4).$$

Observe that B_j being pure allows y to be accessed in any context. By (1) and a straightforward induction on $\mathcal{E}$ we have

$$\Gamma, y : B_j, \text{🔒}_{[h,E]_F} \vdash \mathcal{E}[y] : A \ @\ h, E.$$

By $[E]_F \circ [E]_E \circ [h, E]_E = [h, E]_F$ and context equivalence, we have

$$\Gamma, y : B_j, \text{🔒}_{[E]_F}, \text{🔒}_{[E]_E}, \text{🔒}_{[h,E]_E} \vdash \mathcal{E}[y] : A \mathbin{@} h, E.$$

Since B_j is pure, we can swap $y : B_j$ with locks and derive

$$\Gamma, \text{🔒}_{[E]_F}, \text{🔒}_{[E]_E}, y : B_j, \text{🔒}_{[h,E]_E} \vdash \mathcal{E}[y] : A \mathbin{@} h, E.$$

By T-HandleInst$^{\spadesuit}$', we have

$$\Gamma, \text{🔒}_{[E]_F}, \text{🔒}_{[E]_E}, y : B_j \vdash \mathbf{handle}^{\spadesuit}_h\ \mathcal{E}[y]\ \mathbf{with}\ H : B \mathbin{@} E.$$

Notice that we can put H after absolute locks because all clauses in H have an absolute lock $\text{🔒}_{[E]_E}$ in their contexts. Then by T-Abs and T-Mod we have

$$\Gamma, \text{🔒}_{[E]_F} \vdash \mathbf{mod}_{[E]}\ (\lambda y.\mathbf{handle}^{\spadesuit}_h\ \mathcal{E}[y]\ \mathbf{with}\ H) : [E](B_j \to B) \mathbin{@} E\ (5).$$

By (3), (4), (5), and Lemma B.13.3 we have

$$\Gamma, \text{🔒}_{[E]_F} \vdash N_j[U/p, (\mathbf{mod}_{[F]}\ (\lambda y.\mathbf{handle}^{\spadesuit}_h\ \mathcal{E}[y]\ \mathbf{with}\ H))/r] : B \mathbin{@} E.$$

Finally, by $[E]_F \Rightarrow \mathbb{1}_F$ and Lemma B.11.3 we have

$$\Gamma \vdash N_j[U/p, (\mathbf{mod}_{[F]}\ (\lambda y.\mathbf{handle}^{\spadesuit}_h\ \mathcal{E}[y]\ \mathbf{with}\ H))/r] : B \mathbin{@} F.$$

□

C Full Specifications of Source Calculi and Encodings

In this section, we provide the specifications of the source calculi (System Ξ, System C, System $\mathrm{F}^{\epsilon+\mathrm{sn}}$, System $\mathrm{F}^{\epsilon+\mathrm{sn}}$ 1) and their encodings to Metn that are omitted from Sections 5 to 8.

C.1 Operational Semantics of System Ξ and System C

Reduction in System C is defined not only on terms but also blocks since we have **unbox** V which can reduce. Runtime constructs and evaluation contexts are defined as follows.

Handler Instances	h
Instance Contexts	$\Omega ::= \cdot \mid \Omega, h : \ell$
Contexts	$\Gamma ::= \cdots \mid \Gamma, \llcorner h \lrcorner$
Block Values	$P, Q ::= \cdots \mid \mathbf{cap}_h$
Terms	$M ::= \cdots \mid \mathbf{handle}_h\ M\ \mathbf{with}\ H$
Evaluation Contexts	$\mathcal{E} ::= [\,] \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{def}\ f = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}_h\ \mathcal{E}\ \mathbf{with}\ H$

Typing rules for runtime constructs are as follows.

$$\textsc{T-Cap}\ \frac{\Omega \ni h : \ell \qquad \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\}}{\Gamma \vdash \mathbf{cap}_h : (A) \Rightarrow B \mid \{h\}}$$

$$\textsc{T-Handle}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \llcorner h \lrcorner \vdash M : A \mid C \cup \{h\} \qquad \Gamma, p : A', r :^{C} (B') \Rightarrow A \vdash N : A \mid C}{\Gamma \vdash \mathbf{handle}_h\ M\ \mathbf{with}\ \{\mathsf{op}\ p\ r \mapsto N\} : A \mid C}$$

Since System C uses block variables f as both term-level and type-level variables, we need to substitute them separately. We follow Brachthäuser et al. [4] to overload the notion of substitution. We write C/f for substituting in types and P/f for substituting in terms.

The reduction rules for System C are defined as follows.

$$
\begin{array}{lrcl}
\text{E-Box} & \textbf{unbox}\ (\textbf{box}\ G) & \rightsquigarrow & G \\
\text{E-Let} & \textbf{let}\ x = \textbf{return}\ V\ \textbf{in}\ N & \rightsquigarrow & N[V/x] \\
\text{E-Def} & \textbf{def}\ f = P\ \textbf{in}\ N & \rightsquigarrow & N[P/f] \\
\text{E-Call} & \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}(\overline{V}, \overline{Q}) & \rightsquigarrow & M[\overline{V/x}, \overline{Q/f}, \overline{C/f}] \quad \text{where } \overline{\cdot \vdash Q : T \mid C} \\
\text{E-Gen} & \textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ H \mid \Omega & \rightsquigarrow & \textbf{handle}_h\ M[\textbf{cap}_h/f, \{h\}/f]\ \textbf{with}\ H \mid \Omega, h : \ell \\
& & & \text{where } h \text{ fresh and } H \propto \ell \\
\text{E-NRet} & \textbf{handle}_h\ (\textbf{return}\ V)\ \textbf{with}\ H & \rightsquigarrow & \textbf{return}\ V \\
\text{E-NOp} & \textbf{handle}_h\ \mathcal{E}[\textbf{cap}_h(V)]\ \textbf{with}\ H & \rightsquigarrow & N[V/p, (\lambda y.\textbf{handle}_h\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r], \\
& & & \text{where } H = \{\texttt{op}\ p\ r \mapsto N\} \\
\text{E-Lift} & \mathcal{E}[M] & \rightsquigarrow & \mathcal{E}[N], \qquad \text{if } M \rightsquigarrow N
\end{array}
$$

For the operational semantics of System Ξ, we only need to remove the E-Box rule and substitutions of capability sets C/f.

We provide the translations of runtime constructs and evaluation contexts for System Ξ and System C into Metn. They are needed for showing semantics preservation. Translations of evaluation contexts are analogous to the translations of their corresponding terms.

For System Ξ, we translate runtime constructs as follows.

$$
\begin{array}{rcl}
[\![\textbf{cap}_h]\!] & = & \lambda x^{[\![A]\!]}.\textbf{do}_h\ x \quad \text{where } \Omega \ni h : \ell \text{ and } \Sigma(\ell) = \{\texttt{op} : A \twoheadrightarrow B\} \\
[\![\textbf{handle}_h\ M\ \textbf{with}\ H]\!] & = & \textbf{handle}_h\ [\![M]\!]\ \textbf{with}\ [\![H]\!]
\end{array}
$$

For System C, we translate as follows.

$$
\begin{array}{rcl}
[\![\textbf{cap}_h]\!] & = & \lambda x^{[\![A]\!]}.\textbf{do}_h\ x \quad \text{where } \Omega \ni h : \ell \text{ and } \Sigma(\ell) = \{\texttt{op} : A \twoheadrightarrow B\} \\
[\![\textbf{handle}_h\ M\ \textbf{with}\ H]\!] & = & \textbf{handle}^{\spadesuit}_h\ [\![M]\!]\ \textbf{with}\ [\![H]\!]
\end{array}
$$

C.2 Full Encodings of System Ξ and System C

Our encodings in Section 6 omit homomorphic cases. For completeness, we provide the full encodings with all cases here. Figure 16 gives the full encoding of System Ξ into Metn. Figure 17 gives the full encoding of System C into Metn.

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Types} \to \text{Types} \\
[\![1]\!] & = & 1 \\
[\![(\overline{A}, \overline{T}) \Rightarrow B]\!] & = & \overline{[\![A]\!]} \to \overline{[\![T]\!]} \to [\![B]\!]
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Contexts} \to \text{Contexts} \\
[\![\cdot]\!] & = & \cdot \\
[\![\Gamma, x : A]\!] & = & [\![\Gamma]\!], x : [\![A]\!] \\
[\![\Gamma, f : T]\!] & = & [\![\Gamma]\!], f : [\![T]\!]
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Values} \to \text{Terms} \\
[\![x]\!] & = & x \\
[\![()]\!] & = & ()
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Blocks} \to \text{Terms} \\
[\![f]\!] & = & f \\
[\![\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}]\!] & = & \lambda \overline{x^{[\![A]\!]}}\, \overline{f^{[\![T]\!]}}.[\![M]\!]
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Terms} \to \text{Terms} \\
[\![\mathbf{return}\ V]\!] & = & [\![V]\!] \\
[\![\mathbf{let}\ x = M\ \mathbf{in}\ N]\!] & = & \mathbf{let}\ x = [\![M]\!]\ \mathbf{in}\ [\![N]\!] \\
[\![\mathbf{def}\ f = G\ \mathbf{in}\ N]\!] & = & \mathbf{let}\ f = [\![G]\!]\ \mathbf{in}\ [\![N]\!] \\
[\![P(\overline{V}, \overline{Q})]\!] & = & [\![P]\!]\ \overline{[\![V]\!]}\ \overline{[\![Q]\!]} \\
[\![\mathbf{handle}\ \{f \Rightarrow M\}\ \mathbf{with}\ H]\!] & = & \mathbf{handle}_a\ (\mathbf{let}\ f = \lambda x^{[\![A_{\mathsf{op}}]\!]}.\mathbf{do}_a\ x\ \mathbf{in}\ [\![M]\!])\ \mathbf{with}\ [\![H]\!] \\
 & & \text{where } H = \{\mathsf{op}\ p\ r \mapsto N\} \\
[\![\{\mathsf{op}\ p\ r \mapsto N\}]\!] & = & \{\mathbf{return}\ x \mapsto x, \mathsf{op}\ p\ r \mapsto [\![N]\!]\}
\end{array}
$$

Fig. 16. An encoding of System Ξ in METN without effect variables.

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Value Types} \longrightarrow \text{Types} \\
\llbracket 1 \rrbracket &= 1 \\
\llbracket T\ \mathbf{at}\ C \rrbracket &= [\llbracket C \rrbracket]\llbracket T \rrbracket
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Block Types} \longrightarrow \text{Types} \\
\llbracket (\overline{A}, \overline{f : T}) \Rightarrow B \rrbracket &= \forall \overline{f^*}.\langle\!| \overline{f^*} \rangle (\overline{\llbracket A \rrbracket} \to \overline{[f^*]\llbracket T \rrbracket} \to \llbracket B \rrbracket)
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Capability Sets} \longrightarrow \text{Effect Contexts} \\
\llbracket \{\overline{f}\} \rrbracket &= \overline{f^*}
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket_{-} &: \text{Contexts} \times \text{Capability Sets} \longrightarrow \text{Contexts} \\
\llbracket \cdot \rrbracket_C &= \cdot \\
\llbracket \Gamma, x : A \rrbracket_C &= \llbracket \Gamma \rrbracket_C, x : \llbracket A \rrbracket \\
\llbracket \Gamma, \llcorner \overline{x : A}, \overline{f :^* T} \lrcorner \rrbracket_C &= \llbracket \Gamma \rrbracket_{C \setminus \{\overline{f}\}}, \overline{f^*}, 🔒_{\langle\!| \llbracket \{\overline{f}\} \cap C \rrbracket \rangle}, \overline{x : \llbracket A \rrbracket}, \overline{f : [f^*]\llbracket T \rrbracket}, \overline{\hat{f} :_{[f^*]} \llbracket T \rrbracket} \\
\llbracket \Gamma, f :^C T \rrbracket_{C'} &= \llbracket \Gamma \rrbracket_{C'}, f : [\llbracket C \rrbracket]\llbracket T \rrbracket, \hat{f} :_{[\llbracket C \rrbracket]} \llbracket T \rrbracket \\
\llbracket \Gamma, \clubsuit_C \rrbracket_{C'} &= \llbracket \Gamma \rrbracket_C, 🔒_{[\llbracket C' \rrbracket]_{\llbracket C \rrbracket}}
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Value Judgement} \longrightarrow \text{Term} \\
\llbracket x \rrbracket &= x \\
\llbracket () \rrbracket &= () \\
\llbracket \mathbf{box}\ G : T\ \mathbf{at}\ C \rrbracket &= \mathbf{mod}_{[\llbracket C \rrbracket]}\ \llbracket G \rrbracket
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Block Judgement} \longrightarrow \text{Term} \\
\llbracket f \rrbracket &= \hat{f} \\
\llbracket \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\} \rrbracket &= \Lambda \overline{f^*}.\mathbf{mod}_{\langle\!| \overline{f^*} \rangle} (\lambda \overline{x^{\llbracket A \rrbracket}}\ \overline{f^{[f^*]\llbracket T \rrbracket}}.\overline{\mathbf{let\ mod}_{[f^*]}\ \hat{f} = f\ \mathbf{in}}\ \llbracket M \rrbracket) \\
\llbracket \mathbf{unbox}\ V : T \mid C \rrbracket &= \mathbf{let\ mod}_{[\llbracket C \rrbracket]}\ x = \llbracket V \rrbracket\ \mathbf{in}\ x
\end{aligned}
$$

$$
\begin{aligned}
\llbracket - \rrbracket &: \text{Term Judgement} \longrightarrow \text{Terms} \\
\llbracket \mathbf{let}\ x = M\ \mathbf{in}\ N \rrbracket &= \mathbf{let}\ x = \llbracket M \rrbracket\ \mathbf{in}\ \llbracket N \rrbracket \\
\llbracket \mathbf{def}\ f = G^{T|C}\ \mathbf{in}\ N \rrbracket &= \mathbf{let}\ f = \mathbf{mod}_{[\llbracket C \rrbracket]}\ \llbracket G \rrbracket\ \mathbf{in\ let\ mod}_{[\llbracket C \rrbracket]}\ \hat{f} = f\ \mathbf{in}\ \llbracket N \rrbracket \\
\llbracket P(\overline{V_i}, \overline{Q_j^{T_j|C_j}}) \rrbracket &= \mathbf{let\ mod}_{\langle\!| \overline{\llbracket C_j \rrbracket} \rangle}\ x = \llbracket P \rrbracket\ \overline{\llbracket C_j \rrbracket}\ \mathbf{in}\ x\ \overline{\llbracket V_i \rrbracket}\ \overline{(\mathbf{mod}_{[\llbracket C_j \rrbracket]}\ \llbracket Q_j \rrbracket)} \\
\llbracket \mathbf{return}\ V \rrbracket &= \llbracket V \rrbracket \\
\llbracket \mathbf{handle}\ \{f \Rightarrow M\}\ \mathbf{with}\ H : A \mid C \rrbracket &= \mathbf{handle}^{\spadesuit}_{f^*}\ (\mathbf{let}\ f = \mathbf{mod}_{[f^*]}\ (\lambda x^{\llbracket A_{\mathsf{op}} \rrbracket}.\mathbf{do}_{f^*}\ x)\ \mathbf{in} \\
&\qquad\quad \mathbf{let\ mod}_{[f^*]}\ \hat{f} = f\ \mathbf{in}\ \llbracket M \rrbracket)\ \mathbf{with}\ \llbracket H^C \rrbracket \\
&\quad \text{where } H = \{\mathsf{op}\ p\ r \mapsto N\} \\
\llbracket \{\mathsf{op}\ p\ r \mapsto N\}^C \rrbracket &= \{\mathbf{return}\ x \mapsto \mathbf{let\ mod}_{[\llbracket C \rrbracket]}\ x' = x\ \mathbf{in}\ x' \\
&\quad\ \mathsf{op}\ p\ r \mapsto \mathbf{let\ mod}_{[\llbracket C \rrbracket]}\ \hat{r} = r\ \mathbf{in}\ \llbracket N \rrbracket\}
\end{aligned}
$$

Fig. 17. An encoding of System C in METN with effect variables.

C.3 Full Specification of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ with Both Named and Unnamed Handlers

Figure 18 gives the syntax. Note that we extend the syntax of markers in contexts as well.

Value Types	$A, B ::= 1 \mid A \to^E B \mid \forall \alpha^K.A \mid \mathsf{ev}\ \ell^a$
Types	$T ::= A \mid E \mid a$
Type Variables	α, ε, a
Kind	$K ::= \mathsf{Effect} \mid \mathsf{Scope}$
Effect Rows	$E ::= \cdot \mid \varepsilon \mid \ell, E \mid \ell^a, E$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, \alpha : K \mid \Gamma, \clubsuit_E \mid \Gamma, \blacklozenge_E$
Values	$V, W ::= () \mid x \mid \lambda^E x^A.M \mid \Lambda \alpha^K.V \mid \mathbf{handler}\ H \mid \mathbf{nhandler}\ H$
Computations	$M, N ::= \mathbf{return}\ V \mid V\ W \mid V\ T \mid \mathbf{let}\ x = M\ \mathbf{in}\ N$
	$\mid\ \mathbf{mask}_\ell\ M \mid \mathbf{do}\ \mathsf{op}\ V \mid \mathbf{do}\ V\ W$
Handlers	$H ::= \{\mathsf{op}\ p\ r \mapsto M\}$
Label Contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : \{\mathsf{op} : A \twoheadrightarrow B\}$

Fig. 18. Syntax of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ with both named and unnamed handlers.

Figure 19 gives the typing rules.

For the operational semantics of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, we define runtime constructs and evaluation contexts as follows.

Handler Instances	h
Instance Contexts	$\Omega ::= \cdot \mid \Omega, h : \ell^a$
Computations	$M ::= \cdots \mid \mathbf{handle}_h\ M\ \mathbf{with}\ H$
Values	$V ::= \cdots \mid \mathbf{ev}_h$
Evaluation Contexts	$\mathcal{E} ::= [\ ] \mid \mathbf{let}\ x = \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}_h\ \mathcal{E}\ \mathbf{with}\ H \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H$

Typing rules for runtime constructs are as follows.

$$\textsc{T-Evidence}\quad \frac{\Omega \ni h : \ell^a}{\Gamma \vdash \mathbf{ev}_h : \mathsf{ev}\ \ell^a}$$

$$\textsc{T-Handle}\quad \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \clubsuit_E \vdash M : A \mid \ell, E \qquad \Gamma, p : A', r : B' \to^E A \vdash N : A \mid E}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ \{\mathsf{op}\ p\ r \mapsto N\} : A \mid E}$$

$$\textsc{T-HandleName}\quad \frac{\begin{array}{c}\Sigma(\ell) = \{\ell : A' \twoheadrightarrow B'\} \\ \Omega \ni h : \ell^a \qquad \Gamma, \clubsuit_E \vdash M : A \mid \ell^a, E \qquad \Gamma, p : A', r : B' \to^E A \vdash N : A \mid E\end{array}}{\Gamma \vdash \mathbf{handle}_h\ M\ \mathbf{with}\ \{\mathsf{op}\ p\ r \mapsto N\} : A \mid E}$$

Reduction rules are as follows. We use the predicate $n\mathrm{-free}(\mathsf{op}, \mathcal{E})$ which is defined similarly to that of Met in Appendix A.7.

$$\boxed{\Gamma \vdash V : A}$$

$$\textsc{T-Unit}\ \frac{}{\Gamma \vdash () : 1} \qquad \textsc{T-Var}\ \frac{\Gamma \ni x : A}{\Gamma \vdash x : A} \qquad \textsc{T-Abs}\ \frac{\Gamma, \clubsuit, x : A \vdash M : B \mid F}{\Gamma \vdash \lambda^F x^A.M : A \to^F B} \qquad \textsc{T-TAbs}\ \frac{\Gamma, \alpha : K \vdash V : A}{\Gamma \vdash \Lambda\alpha^K.V : A}$$

$$\textsc{T-Handler}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \clubsuit, p : A', r : B' \to^F A \vdash N : A \mid F}{\Gamma \vdash \textbf{handler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (1 \to^{\ell,F} A) \to^F A}$$

$$\textsc{T-NamedHandler}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \clubsuit, p : A', r : B' \to^F A \vdash N : A \mid F}{\Gamma \vdash \textbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (\forall a^{\mathsf{Scope}}.\mathsf{ev}\ \ell^a \to^{\ell^a,F} A) \to^F A}$$

$$\boxed{\Gamma \vdash M : A \mid E}$$

$$\textsc{T-Value}\ \frac{\Gamma \vdash V : A}{\Gamma \vdash \textbf{return}\ V : A \mid E} \qquad \textsc{T-App}\ \frac{\Gamma \vdash V : A \to^E B \qquad \Gamma \vdash W : A}{\Gamma \vdash V\ W : B \mid E}$$

$$\textsc{T-Let}\ \frac{\Gamma \vdash M : A \mid E \qquad \Gamma, x : A \vdash N : B \mid E}{\Gamma \vdash \textbf{let}\ x = M\ \textbf{in}\ N : B \mid E} \qquad \textsc{T-TApp}\ \frac{\Gamma \vdash V : \forall\alpha^K.B \qquad \Gamma \vdash T : K}{\Gamma \vdash V\ T : B[T/\alpha] \mid E}$$

$$\textsc{T-Do}\ \frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \\ \Gamma \vdash V : A \qquad E = \ell, F\end{array}}{\Gamma \vdash \textbf{do}\ \mathsf{op}\ V : B \mid E} \qquad \textsc{T-DoName}\ \frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \qquad E = \ell^a, F \\ \Gamma \vdash V : \mathsf{ev}\ \ell^a \qquad \Gamma \vdash W : A\end{array}}{\Gamma \vdash \textbf{do}\ V\ W : B \mid E} \qquad \textsc{T-Mask}\ \frac{\Gamma, \blacklozenge_{\ell,E} \vdash M : A \mid E}{\Gamma \vdash \textbf{mask}_\ell\ M : A \mid \ell, E}$$

Fig. 19. Typing rules for System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ with both named and unnamed handlers.

$$\begin{array}{lll}
\textsc{E-TApp} & (\Lambda\alpha^K.V)\ T \rightsquigarrow V[T/\alpha] & \\
\textsc{E-App} & (\lambda x^A.M)\ V \rightsquigarrow M[V/x] & \\
\textsc{E-Mask} & \textbf{mask}_L\ V \rightsquigarrow V & \\
\textsc{E-Ret} & \textbf{handle}\ (\textbf{return}\ V)\ \textbf{with}\ H \rightsquigarrow N[V/x], & \text{where } (\textbf{return}\ x \mapsto N) \in H \\
\textsc{E-Handler} & \textbf{handler}\ H\ V \rightsquigarrow \textbf{handle}\ V\ ()\ \textbf{with}\ H, & \\
\textsc{E-Op} & \textbf{handle}\ \mathcal{E}[\textbf{do}\ \mathsf{op}\ V]\ \textbf{with}\ H \rightsquigarrow N[V/p, (\lambda y.\textbf{handle}\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r], & \\
 & & \text{where } 0\text{–free}(\mathsf{op}, \mathcal{E}) \text{ and } (\mathsf{op}\ p\ r \mapsto N) \in H \\
\textsc{E-Gen} & \textbf{nhandler}\ H\ V \mid \Omega \rightsquigarrow \textbf{handle}_h\ (\textbf{let}\ x = V\ a\ \textbf{in}\ x\ \textbf{ev}_h)\ \textbf{with}\ H \mid \Omega, h : \ell^a & \\
 & & \text{where } a, h \text{ fresh and } H \propto \ell \\
\textsc{E-NRet} & \textbf{handle}_h\ V\ \textbf{with}\ H \rightsquigarrow \textbf{return}\ V & \\
\textsc{E-NOp} & \textbf{handle}_h\ \mathcal{E}[\textbf{do}\ \textbf{ev}_h\ V]\ \textbf{with}\ H \rightsquigarrow N[V/p, (\lambda y.\textbf{handle}_h\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r], & \\
 & & \text{where } (\mathsf{op}\ p\ r \mapsto N) \in H \\
\textsc{E-Lift} & \mathcal{E}[M] \rightsquigarrow \mathcal{E}[N], & \text{if } M \rightsquigarrow N
\end{array}$$

Translations of runtime constructs for System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ are as follows.

$$\begin{aligned} [\![\mathbf{ev}_h]\!] &= \mathbf{mod}_{[a]}\ (\lambda x^{[\![A]\!]}.\mathbf{do}_a\ x) && \text{where } \Omega \ni h : \ell^a \text{ and } \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \\ [\![\mathbf{handle}_h\ M\ \mathbf{with}\ H]\!] &= \mathbf{handle}^{\spadesuit}_a\ [\![M]\!]\ \mathbf{with}\ [\![H]\!] && \text{where } \Omega \ni h : \ell^a \text{ and } \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \end{aligned}$$

They are used in the proof of semantics preservation in Appendix D.1.

C.4 Full Specification of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ with Both Named and Unnamed Handlers

Figure 20 gives the syntax.

Value Types	$A, B ::= 1 \mid A \to^{\{E\mid\varepsilon\}} B \mid \forall\varepsilon.A \mid \forall a.A \mid \mathsf{ev}\ \ell^a$
Type Variables	a, ε
Effect Rows	$E ::= \cdot \mid \ell, E \mid \ell^a, E$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x :_\varepsilon A \mid \Gamma, a \mid \Gamma, \blacklozenge_E \mid \Gamma, \clubsuit_E$
Values	$V, W ::= () \mid x \mid \lambda^{\{E\mid\varepsilon\}} x^A.M \mid \Lambda a.V \mid \Lambda\varepsilon.V \mid \mathbf{handler}\ H \mid \mathbf{nhandler}\ H$
Terms	$M, N ::= \mathbf{return}\ V \mid V\ W \mid V\ A \mid V\ \#\{E\mid\varepsilon\} \mid \mathbf{let}\ x = M\ \mathbf{in}\ N$
	$\mid\ \mathbf{mask}_\ell\ M \mid \mathbf{do}\ \mathsf{op}\ N \mid \mathbf{do}\ M\ N$
Handlers	$H ::= \{\mathsf{op}\ p\ r \mapsto M\}$

Fig. 20. Syntax of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ with both named and unnamed handlers.

Figure 21 gives the typing rules.

The operational semantics of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ follows from that of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ in Appendix C.3. We just need to split E-TApp-System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ into two rules E-EApp and E-NApp as follows.

$$\begin{aligned} &\text{E-EApp} && (\Lambda\varepsilon'.V)\ \{E\mid\varepsilon\} \rightsquigarrow V[\{E\mid\varepsilon\}/\varepsilon'] \\ &\text{E-NApp} && (\Lambda a.V)\ b \rightsquigarrow V[b/a] \end{aligned}$$

Translations of runtime constructs for System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ are as follows.

$$\begin{aligned} [\![\mathbf{ev}_h]\!] &= \mathbf{mod}_{[a]}\ (\lambda x^{[\![A]\!]}.\mathbf{do}_a\ x) \quad \text{where } \Omega \ni h : \ell^a \text{ and } \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \\ [\![\mathbf{handle}_h\ M\ \mathbf{with}\ H : \_ \mid E]\!] &= \mathbf{handle}_a\ (\mathbf{let}_\mu\ \mathbf{mod}_y\ = [\![M]\!]\ \mathbf{in}\ \mathbf{mod}_{\langle a|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!] \\ &\quad \text{where } \Omega \ni h : \ell^a \text{ and } \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \\ &\quad \text{and } \mu = \mathrm{topmod}([\![A]\!]_{\ell^a,E}) \text{ and } \nu = \mathrm{topmod}([\![A]\!]_E) \end{aligned}$$

They are used in the proof of semantics preservation in Appendix D.2.

$$\boxed{\Gamma \vdash_\varepsilon V : A \mid E} \quad \boxed{\Gamma \vdash_\varepsilon M : A \mid E}$$

$$\text{T-Unit}\ \frac{}{\Gamma \vdash_\varepsilon () : 1 \mid E} \qquad \text{T-Var}\ \frac{\varepsilon = \varepsilon' \text{ or } A = \overline{\forall a}.\forall \varepsilon''.A' \text{ or } A = \overline{\forall a}.1}{\Gamma, x :_{\varepsilon'} A, \Gamma' \vdash_\varepsilon x : A \mid E} \qquad \text{T-Abs}\ \frac{\Gamma, \blacklozenge_E, x :_\varepsilon A \vdash_\varepsilon M : B \mid F}{\Gamma \vdash_\varepsilon \lambda^{\{F|\varepsilon\}} x^A.M : A \to^{\{F|\varepsilon\}} B \mid E}$$

$$\text{T-SAbs}\ \frac{\Gamma, a \vdash_\varepsilon V : A \mid E}{\Gamma \vdash_\varepsilon \Lambda a.V : \forall a.A \mid E} \qquad \text{T-EAbs}\ \frac{\Gamma, \clubsuit_E \vdash_{\varepsilon'} V : A \mid \cdot \quad \varepsilon' \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash_\varepsilon \Lambda \varepsilon'.V : \forall \varepsilon'.A \mid E} \qquad \text{T-Value}\ \frac{\Gamma \vdash_\varepsilon V : A \mid E}{\Gamma \vdash_\varepsilon \textbf{return}\ V : A \mid E} \qquad \text{T-SApp}\ \frac{\Gamma \vdash_\varepsilon V : \forall a.A \mid E \quad \Gamma \ni b}{\Gamma \vdash_\varepsilon V\ b : A[b/a] \mid E}$$

$$\text{T-Let}\ \frac{\Gamma \vdash_\varepsilon M : A \mid E \quad \Gamma, x : A \vdash_\varepsilon N : B \mid E}{\Gamma \vdash_\varepsilon \textbf{let}\ x = M\ \textbf{in}\ N : B \mid E} \qquad \text{T-EApp}\ \frac{\Gamma \vdash_\varepsilon V : \forall \varepsilon'.A \mid E}{\Gamma \vdash_\varepsilon V\ \#\{E|\varepsilon\} : A[\{E|\varepsilon\}/\varepsilon'] \mid E} \qquad \text{T-App}\ \frac{\Gamma \vdash_\varepsilon V : A \to^{\{E|\varepsilon\}} B \mid E \quad \Gamma \vdash_\varepsilon W : A \mid E}{\Gamma \vdash_\varepsilon V\ W : B \mid E}$$

$$\text{T-DoName}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \quad E = \ell^a, F \quad \Gamma \vdash_\varepsilon V : \mathsf{ev}\ \ell^a \mid E \quad \Gamma \vdash_\varepsilon W : A \mid E}{\Gamma \vdash_\varepsilon \textbf{do}\ V\ W : B \mid E} \qquad \text{T-Do}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \quad \Gamma \vdash_\varepsilon V : A \mid E \quad E = \ell, F}{\Gamma \vdash_\varepsilon \textbf{do}\ \mathsf{op}\ V : B \mid E} \qquad \text{T-Mask}\ \frac{\Gamma, \blacklozenge_{\ell,E} \vdash_\varepsilon M : A \mid E}{\Gamma \vdash_\varepsilon \textbf{mask}_\ell\ M : A \mid \ell, E}$$

$$\text{T-NamedHandler}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \blacklozenge_E, p :_\varepsilon A', r :_\varepsilon B' \to^{\{F|\varepsilon\}} A \vdash_\varepsilon N : A \mid F}{\Gamma \vdash_\varepsilon \textbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (\forall a.\mathsf{ev}\ \ell^a \to^{\{\ell^a, F|\varepsilon\}} A) \to^{\{F|\varepsilon\}} A \mid E}$$

$$\text{T-Handler}\ \frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \blacklozenge_E, p :_\varepsilon A', r :_\varepsilon B' \to^{\{F|\varepsilon\}} A \vdash_\varepsilon N : A \mid F}{\Gamma \vdash_\varepsilon \textbf{handler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (1 \to^{\{\ell, F|\varepsilon\}} A) \to^{\{F|\varepsilon\}} A \mid E}$$

Fig. 21. Typing rules for System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ with both named and unnamed handlers.

C.5 Full Encodings of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$

Figure 22 gives the encoding of full System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ with both named and unnamed handlers into Metn with effect variables.

Figure 23 gives the encoding of full System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ with both named and unnamed handlers into Metn without effect variables with masking handler names. We define $\mathsf{topmod}(\mu A) = \mu$.

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Type} \longrightarrow \text{Type} \\
[\![1]\!] & = & 1 \\
[\![A \to^{E} B]\!] & = & [[\![E]\!]]([\![A]\!] \to [\![B]\!]) \\
[\![\forall \varepsilon.A]\!] & = & \forall \varepsilon.[\![A]\!] \\
[\![\forall a^{\ell}.A]\!] & = & \mathbb{\forall} a^{\ell}.[\![A]\!] \\
[\![\mathsf{ev}\,\ell^{a}]\!] & = & [a]([\![A_\ell]\!] \to [\![B_\ell]\!])
\end{array}
\qquad
\begin{array}{rcl}
[\![-]\!]_{-} & : & \text{Context} \times \text{Effect Row} \longrightarrow \text{Context} \\
[\![\cdot]\!]_E & = & \cdot \\
[\![\Gamma, x : A]\!]_E & = & [\![\Gamma]\!]_E, x : [\![A]\!] \\
[\![\Gamma, \varepsilon]\!]_E & = & [\![\Gamma]\!]_E, \varepsilon \\
[\![\Gamma, a : \ell]\!]_E & = & [\![\Gamma]\!]_E, a : \ell \\
[\![\Gamma, \clubsuit]\!]_E & = & [\![\Gamma]\!]_{\cdot}, 🔒_{[[\![E]\!]]\cdot}
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Effect Row} \longrightarrow \text{Effect Context} \\
[\![\cdot]\!] & = & \cdot \\
[\![\varepsilon]\!] & = & \varepsilon \\
[\![\ell, E]\!] & = & \ell, [\![E]\!] \\
[\![\ell^{a}, E]\!] & = & a, [\![E]\!]
\end{array}
\qquad
\begin{array}{rcl}
[\![-]\!] & : & \text{Label Context} \longrightarrow \text{Label Context} \\
[\![\cdot]\!] & = & \cdot \\
[\![\Sigma, \ell : \{\mathsf{op} : A \twoheadrightarrow B\}]\!] & = & [\![\Sigma]\!], \ell : \{\mathsf{op} : [\![A]\!] \twoheadrightarrow [\![B]\!]\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Value / Computation} \longrightarrow \text{Term} \\
[\![()]\!] & = & () \\
[\![x]\!] & = & x \\
[\![\Lambda \varepsilon.V]\!] & = & \Lambda \varepsilon.[\![V]\!] \\
[\![\Lambda a^{\ell}.V]\!] & = & \lambda\!\!\!\lambda a^{\ell}.[\![V]\!] \\
[\![\lambda^{E} x^{A}.M]\!] & = & \mathbf{mod}_{[[\![E]\!]]}\ (\lambda x^{[\![A]\!]}.[\![M]\!]) \\
[\![\mathbf{let}\ x = M\ \mathbf{in}\ N]\!] & = & \mathbf{let}\ x = [\![M]\!]\ \mathbf{in}\ [\![N]\!] \\
[\![V\ a]\!] & = & [\![V]\!]\ a \\
[\![V\ E]\!] & = & [\![V]\!]\ [\![E]\!] \\
[\![V^{A \to^{E} B}\ W]\!] & = & \mathbf{let\ mod}_{[[\![E]\!]]}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] \\
[\![\mathbf{mask}_{\ell}\ M]\!] & = & \mathbf{let\ mod}_{\langle \ell | \rangle}\ x = \mathbf{mask}_{\ell}\ [\![M]\!]\ \mathbf{in}\ x \\
[\![\mathbf{do}\ \mathsf{op}\ V]\!] & = & \mathbf{do}\ \mathsf{op}\ [\![V]\!] \\
[\![\mathbf{do}\ V^{\mathsf{ev}\,\ell^{a}}\ W]\!] & = & \mathbf{let\ mod}_{[a]}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] \\
\left[\!\!\left[\begin{array}{l}\mathbf{handler}\ H \\ : (1 \to^{\ell,E} A) \to^{E} A\end{array}\right]\!\!\right] & = & \mathbf{mod}_{[[\![E]\!]]}\ (\lambda f.\mathbf{handle}^{\spadesuit}\ (\mathbf{let\ mod}_{[[\![\ell,E]\!]]}\ f = f\ \mathbf{in}\ f\ ()) \\
 & & \qquad\qquad \mathbf{with}\ [\![H^{\ell;E}]\!]) \\
[\![H^{\ell;E}]\!] & = & \{\mathbf{return}\ x \mapsto \mathbf{let\ mod}_{[[\![\ell,E]\!]]}\ x = x\ \mathbf{in}\ x, \\
 & & \ \mathsf{op}\ p\ r \mapsto [\![N]\!]\}) \\
\left[\!\!\left[\begin{array}{l}\mathbf{nhandler}\ H \\ : (\forall a^{\ell}.\mathsf{ev}\,\ell^{a} \to^{\ell^{a},E} A) \to^{E} A\end{array}\right]\!\!\right] & = & \mathbf{mod}_{[[\![E]\!]]}\ (\lambda f.\mathbf{handle}_{a}^{\spadesuit}\ (\mathbf{let}\ f = f\ a\ \mathbf{in} \\
 & & \qquad \mathbf{let\ mod}_{[[\![\mathsf{ev}\,\ell^{a},E]\!]]}\ f = f\ \mathbf{in} \\
 & & \qquad f\ (\mathbf{mod}_{[a]}\ (\lambda x^{[\![A_{\mathsf{op}}]\!]}.\mathbf{do}_{a}\ x)))\ \mathbf{with}\ [\![H^{E}]\!]) \\
 & & \text{where } H = \{\mathsf{op}\ p\ r \mapsto N\} \\
[\![H^{E}]\!] & = & \{\mathbf{return}\ x \mapsto \mathbf{let\ mod}_{[[\![E]\!]]}\ x = x\ \mathbf{in}\ x, \\
 & & \ \mathsf{op}\ p\ r \mapsto [\![N]\!]\}
\end{array}
$$

Fig. 22. An encoding of full System $F^{\epsilon+sn}$ in Metn with effect variables.

$$
\begin{array}{rcl}
[\![-]\!]_{-} & : & \text{Type} \times \text{Effect Row} \to \text{Type} \\
[\![1]\!]_E & = & \langle\rangle 1 \\
[\![A \to^F B]\!]_E & = & \langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle ([\![A]\!]_F \to [\![B]\!]_F) \\
[\![\forall.A]\!]_E & = & [\,][\![A]\!]_{\cdot} \\
[\![\forall a^\ell.A]\!]_E & = & \begin{cases} \nu(\forall a^\ell.\langle\!| a \rangle B), & \text{if } \mu \equiv \nu \circ \langle\!| a \rangle \\ \mu(\forall a^\ell.B), & \text{otherwise} \end{cases} \\
 & & \text{where } [\![A]\!]_E = \mu B \\
[\![\mathsf{ev}\ \ell^a]\!]_E & = & [a]([\![A]\!]_{\cdot} \to [\![B]\!]_{\cdot}) \\
 & & \text{where } \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\}
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Effect Row} \to \text{Effect Context} \\
[\![\cdot]\!] & = & \cdot \\
[\![\ell, E]\!] & = & \ell, [\![E]\!] \\
[\![\ell^a, E]\!] & = & a, [\![E]\!]
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!]_{-} & : & \text{Context} \times \text{Effect Row} \to \text{Context} \\
[\![\cdot]\!]_E & = & \cdot \\
[\![\Gamma, x : A]\!]_E & = & [\![\Gamma]\!]_E, x : \mu A', \hat{x} :_\mu A' \text{ for } \mu A' = [\![A]\!]_E \\
[\![\Gamma, a : \ell]\!]_E & = & [\![\Gamma]\!]_E, a : \ell \\
[\![\Gamma, \clubsuit_E]\!]_{\cdot} & = & [\![\Gamma]\!]_E, \blacksquare_{[\,]} \\
[\![\Gamma, \blacklozenge_F]\!]_E & = & [\![\Gamma]\!]_F, \blacksquare_{\langle [\![F]\!] - [\![E]\!] \mid [\![E]\!] - [\![F]\!] \rangle}
\end{array}
$$

$$
\begin{array}{rcl}
[\![-]\!] & : & \text{Value / Computation} \to \text{Term} \\
[\![()]\!] & = & \mathbf{mod}_{\langle\rangle}\ () \\
[\![x : A \mid E]\!] & = & \mathbf{mod}_\mu\ \hat{x} \quad \text{where } \mu = \text{topmod}([\![A]\!]_E) \\
[\![\Lambda.V]\!] & = & \mathbf{mod}_{[\,]}\ [\![V]\!] \\
[\![\Lambda a^\ell.V : \forall a^\ell.A \mid E]\!] & = & \begin{cases} \mathbf{let\ mod}_\nu\ \Lambda a^\ell.x = (\mathbf{let\ mod}_\mu\ x = [\![V]\!]\ \mathbf{in\ mod}_{\nu;\langle\!| a \rangle}\ x) & \\ \quad \mathbf{in\ mod}_\nu\ x & \text{if } \mu \equiv \nu \circ \langle\!| a \rangle \\ \mathbf{let\ mod}_\mu\ \Lambda a^\ell.x = [\![V]\!]\ \mathbf{in\ mod}_\mu\ x, & \text{otherwise} \end{cases} \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_E) \\
[\![\lambda^F x^A.M : \_ \mid E]\!] & = & \mathbf{mod}_{\langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle}\ (\lambda x^{[\![A]\!]_F}.\mathbf{let\ mod}_\mu\ \hat{x} = x\ \mathbf{in}\ [\![M]\!]) \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_F) \\
[\![\mathbf{let}\ x = M^A\ \mathbf{in}\ N : \_ \mid E]\!] & = & \mathbf{let}\ x = [\![M]\!]\ \mathbf{in\ let\ mod}_\mu\ \hat{x} = x\ \mathbf{in}\ [\![N]\!] \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_E) \\
[\![V\# : \_ \mid E]\!] & = & \mathbf{let\ mod}_{[\,]}\ x = [\![V]\!]\ \mathbf{in}\ x \\
[\![V^{\forall a^\ell.A}\ b : \_ \mid E]\!] & = & \begin{cases} \mathbf{let\ mod}_\nu\ x = [\![V]\!]\ \mathbf{in} & \\ \quad \mathbf{let}_\nu\ \mathbf{mod}_{\langle\!| a \rangle}\ y = x\ b\ \mathbf{in\ mod}_\mu\ y & \text{if } \mu \equiv \nu \circ \langle\!| a \rangle \\ \mathbf{let\ mod}_\mu\ x = [\![V]\!]\ \mathbf{in\ mod}_\mu\ (x\ b) & \text{otherwise} \end{cases} \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_E) \\
[\![V\ W]\!] & = & \mathbf{let\ mod}_{\langle\rangle}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] \\
[\![\mathbf{mask}_\ell\ M : \_ \mid \ell, E]\!] & = & \mathbf{let\ mod}_{\langle \ell |\rangle;\mu}\ x = \mathbf{mask}_\ell\ [\![M]\!]\ \mathbf{in\ mod}_\nu\ x \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_E) \text{ and } \nu = \text{topmod}([\![A]\!]_{\ell,E}) \\
[\![\mathbf{do}\ \mathsf{op}\ V]\!] & = & \mathbf{do}\ \mathsf{op}\ [\![V]\!] \\
[\![\mathbf{do}\ V\ W]\!] & = & \mathbf{let\ mod}_{[a]}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] \quad \text{where } V : \mathsf{ev}\ \ell^a \\
[\![\mathbf{handler}\ H : A \mid E]\!] & = & \mathbf{mod}_{\langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle}\ (\lambda f.\mathbf{let\ mod}_{\langle\!| \ell \rangle}\ f = f\ \mathbf{in} \\
 & & \mathbf{handle}\ f\ ()\ \mathbf{with}\ [\![H : A \mid E]\!]) \\
\left[\!\!\left[\begin{array}{l}\{\mathsf{op}\ p\ r \mapsto N\} : \\ (1 \to^{\ell,F} A) \to^F A \mid E\end{array}\right]\!\!\right] & = & \{\mathbf{return}\ x \mapsto \mathbf{let\ mod}_{\langle\!| \ell \rangle;\mu}\ x = x\ \mathbf{in\ mod}_\nu\ x, \\
 & & \ \mathsf{op}\ p\ r \mapsto \mathbf{let\ mod}_{\mu_p}\ \hat{p} = p\ \mathbf{in\ let\ mod}_{\langle\rangle}\ \hat{r} = r\ \mathbf{in}\ [\![N]\!]\} \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_{\ell,F}) \text{ and } \nu = \text{topmod}([\![A]\!]_F) \\
 & & \qquad \mu_p = \text{topmod}([\![A']\!]_F) \text{ and } \Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \\
\left[\!\!\left[\begin{array}{l}\mathbf{nhandler}\ H : \\ (\forall a^\ell.\mathsf{ev}\ \ell^a \to^{\ell^a,F} A) \to^F A \mid E\end{array}\right]\!\!\right] & = & \mathbf{mod}_{\langle [\![E]\!] - [\![F]\!] \mid [\![F]\!] - [\![E]\!] \rangle}\ (\lambda f.\mathbf{handle}_a\ (\mathbf{let\ mod}_{\langle\!| a \rangle}\ f' = f\ a\ \mathbf{in} \\
 & & \mathbf{let\ mod}_\mu\ y = f'\ (\mathbf{mod}_{[a]}\ (\lambda x.\mathbf{do}_a\ x))\ \mathbf{in\ mod}_{\langle a |\rangle;\nu}\ y)\ \mathbf{with}\ [\![H^F]\!]) \\
 & & \text{where } \mu = \text{topmod}([\![A]\!]_{\ell^a,F}) \text{ and } \nu = \text{topmod}([\![A]\!]_F) \\
[\![\{\mathsf{op}\ p\ r \mapsto N\}^F]\!] & = & \{\mathbf{return}\ x \mapsto x, \\
 & & \ \mathsf{op}\ p\ r \mapsto \mathbf{let\ mod}_{\mu_p}\ \hat{p} = p\ \mathbf{in\ let\ mod}_{\langle\rangle}\ \hat{r} = r\ \mathbf{in}\ [\![N]\!]\} \\
 & & \text{where } \mu_p = \text{topmod}([\![A_{\mathsf{op}}]\!]_F)
\end{array}
$$

Fig. 23. An encoding of full System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ in Metn without effect variables with masking names.

D Proofs of Encodings

We prove all type preservation and semantics preservation theorems in Sections 6 and 8. For System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$, we prove the full encodings with both named and unnamed handlers provided in Appendix C.5.

D.1 Proofs for the Encoding of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$

Theorem 8.2 (Type Preservation). *If $\Gamma \vdash M : A \mid E$ is a consistent judgement in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, then $[\![\Gamma]\!]_E \vdash [\![M]\!] : [\![A]\!] \mathbin{@} [\![E]\!]$ in Metn. Similarly for typing judgements of values.*

Proof. By induction on consistent typing judgements $\Gamma \vdash M : A \mid E$ in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. Most cases follow from using IH trivially. We elaborate interesting cases.

Case $\boxed{()}$ By T-Unit-System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and T-Unit-Metn.

Case $\boxed{x}$ All translated types have kind Abs in Metn and can be always accessed by T-Var-Metn.

Case $\boxed{\Lambda\varepsilon.V}$ By IH, T-TAbs-System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and T-EAbs-Metn.

Case $\boxed{V\ E}$ By IH, T-TApp-System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and T-EApp-Metn.

Case $\boxed{\Lambda a^\ell.V}$ By IH, T-TAbs-System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and T-NAbs-Metn.

Case $\boxed{V\ a}$ By IH, T-TApp-System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and T-NApp-Metn.

Case $\boxed{\lambda^E x^A.M}$

$$\text{T-Abs}\ \frac{\Gamma, \clubsuit, x : A \vdash M : B \mid E\ (1)}{\Gamma \vdash \lambda^E x^A.M : A \to^E B}$$

By IH on (1), we have

$$[\![\Gamma]\!]._{\text{,}}\text{🔒}_{[[\![E]\!]]}, x : [\![A]\!] \vdash [\![M]\!] : [\![B]\!] \mathbin{@} [\![E]\!]$$

By T-Abs-Metn and T-Mod-Metn, we have

$$[\![\Gamma]\!]. \vdash \mathbf{mod}_{[[\![E]\!]]}\ (\lambda x^{[\![A]\!]}.[\![M]\!]) : [[\![E]\!]]([\![A]\!] \to [\![B]\!]) \mathbin{@} \cdot$$

Case $\boxed{\mathbf{return}\ V}$ By Lemma D.1.

Case $\boxed{\mathbf{let}\ x = M\ \mathbf{in}\ N}$ By IH, syntactic sugar, and T-App-Metn.

Case $\boxed{V\ W}$

$$\text{T-App}\ \frac{\Gamma \vdash V : A \to^E B\ (1) \qquad \Gamma \vdash W : A\ (2)}{\Gamma \vdash V\ W : B \mid E}$$

By IH on (1) and Lemma D.1, we have

$$[\![\Gamma]\!] \vdash [\![V]\!] : [[\![E]\!]]([\![A]\!] \to [\![B]\!]) \mathbin{@} [\![E]\!]$$

By IH on (2) and Lemma D.1, we have

$$[\![\Gamma]\!] \vdash [\![W]\!] : [\![A]\!] \mathbin{@} [\![E]\!]$$

By T-Letmod-Metn and T-App-Metn, we have

$$[\![\Gamma]\!] \vdash \mathbf{let}\ \mathbf{mod}_{[[\![E]\!]]}\ x = [\![V]\!]\ \mathbf{in}\ x\ [\![W]\!] : [\![B]\!] \mathbin{@} [\![E]\!]$$

Case $\boxed{\mathbf{mask}_\ell\ M}$

$$\dfrac{\text{T-MASK} \quad \Gamma, \blacklozenge_{\ell,E} \vdash M : A \mid E\ (1)}{\Gamma \vdash \mathbf{mask}_\ell\ M : A \mid \ell, E}$$

By IH on (1), we have

$$\llbracket\Gamma\rrbracket_{\ell,E}, \text{🔒}_{\langle \ell | \rangle} \vdash \llbracket M\rrbracket : \llbracket A\rrbracket \mathbin{@} \llbracket E\rrbracket$$

By T-MASK-METN, we have

$$\llbracket\Gamma\rrbracket_{\ell,E} \vdash \mathbf{mask}_\ell\ \llbracket M\rrbracket : \langle \ell | \rangle \llbracket A\rrbracket \mathbin{@} \llbracket \ell, E\rrbracket$$

By T-LETMOD-METN, we have

$$\llbracket\Gamma\rrbracket_{\ell,E} \vdash \mathbf{let\ mod}_{\langle \ell | \rangle}\ x = \mathbf{mask}_\ell\ \llbracket M\rrbracket\ \mathbf{in}\ x : \llbracket A\rrbracket \mathbin{@} \llbracket \ell, E\rrbracket$$

Case $\boxed{\mathbf{do}\ \mathsf{op}\ V}$ By IH, Lemma D.1, T-DO-SYSTEM $\mathrm{F}^{\epsilon+\mathrm{sn}}$ and T-DO-METN.

Case $\boxed{\mathbf{do}\ V\ W}$

$$\dfrac{\text{T-DONAME} \quad \Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \qquad E = \ell^a, F \qquad \Gamma \vdash V : \mathsf{ev}\ \ell^a\ (1) \qquad \Gamma \vdash W : A\ (2)}{\Gamma \vdash \mathbf{do}\ V\ W : B \mid E}$$

By IH on (1) and (2) and Lemma D.1, we have

$$\llbracket\Gamma\rrbracket \vdash \llbracket V\rrbracket : [a](\llbracket A\rrbracket \to \llbracket B\rrbracket) \mathbin{@} \llbracket E\rrbracket$$
$$\llbracket\Gamma\rrbracket \vdash \llbracket W\rrbracket : \llbracket A\rrbracket \mathbin{@} \llbracket E\rrbracket$$

By T-LETMOD-METN and T-APP-METN, we have

$$\llbracket\Gamma\rrbracket \vdash \mathbf{let\ mod}_{[a]}\ x = \llbracket V\rrbracket\ \mathbf{in}\ x\ \llbracket W\rrbracket : \llbracket B\rrbracket \mathbin{@} \llbracket E\rrbracket$$

Case $\boxed{\mathbf{handler}\ \{\mathsf{op}\ p\ r \mapsto N\}}$

$$\dfrac{\text{T-HANDLER} \quad \Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \clubsuit_F, p : A', r : B' \to^E A \vdash N : A \mid E\ (1)}{\Gamma \vdash \mathbf{handler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (1 \to^{\ell,E} A) \to^E A \mid F}$$

By IH on (1), we have

$$\llbracket\Gamma\rrbracket_F, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket F\rrbracket}}, p : \llbracket A'\rrbracket, r : [\llbracket E\rrbracket](\llbracket B'\rrbracket \to \llbracket A\rrbracket) \vdash \llbracket N\rrbracket : \llbracket A\rrbracket \mathbin{@} \llbracket E\rrbracket$$

By Lemma B.12 and structural rules, we have

$$\llbracket\Gamma\rrbracket_F, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket F\rrbracket}}, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket E\rrbracket}}, p : \llbracket A'\rrbracket, r : [\llbracket E\rrbracket](\llbracket B'\rrbracket \to \llbracket A\rrbracket) \vdash \llbracket N\rrbracket : \llbracket A\rrbracket \mathbin{@} \llbracket E\rrbracket\ (2)$$

By T-LETMOD-METN, T-VAR-METN, and $\llbracket A\rrbracket$ has kind Abs, we have

$$\llbracket\Gamma\rrbracket_F, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket F\rrbracket}}, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket E\rrbracket}}, x : [\llbracket \ell, E\rrbracket]\llbracket A\rrbracket \vdash \mathbf{let\ mod}_{[\llbracket \ell,E\rrbracket]}\ x = x\ \mathbf{in}\ x : \llbracket A\rrbracket \mathbin{@} \llbracket E\rrbracket\ (3)$$

By T-VAR-METN, T-LETMOD-METN, and T-APP-METN, we have

$$\llbracket\Gamma\rrbracket_F, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket F\rrbracket}}, f : [\llbracket \ell, E\rrbracket](1 \to \llbracket A\rrbracket), \text{🔒}_{[\llbracket \ell,E\rrbracket]} \vdash \mathbf{let\ mod}_{[\llbracket \ell,E\rrbracket]}\ f = f\ \mathbf{in}\ f\ () : \llbracket A\rrbracket \mathbin{@} \llbracket \ell, E\rrbracket\ (4)$$

By T-HANDLE$^\spadesuit$-METN, (2), (3), and (4), we have

$$\llbracket\Gamma\rrbracket_F, \text{🔒}_{[\llbracket E\rrbracket]_{\llbracket F\rrbracket}}, f : [\llbracket \ell, E\rrbracket](1 \to \llbracket A\rrbracket) \vdash \mathbf{handle}^\spadesuit\ (\mathbf{let\ mod}_{[\llbracket \ell,E\rrbracket]}\ f = f\ \mathbf{in}\ f\ ())\ \mathbf{with}\ H : \llbracket A\rrbracket \mathbin{@} \llbracket E\rrbracket$$

where $H = \{\mathbf{return}\ x \mapsto \mathbf{let\ mod}_{[\llbracket \ell,E\rrbracket]}\ x = x\ \mathbf{in}\ x, \mathsf{op}\ p\ r \mapsto \llbracket N\rrbracket\}$.
Our final goal follows from T-ABS-METN, T-MOD-METN, and Lemma B.12.

Case $\boxed{\textbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\}}$

$$\frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad \Gamma, \clubsuit_F, p : A', r : B' \to^E A \vdash N : A \mid E\ (1)}{\Gamma \vdash \textbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\} : (\forall a.\mathsf{ev}\ \ell^a \to^{\ell^a, E} A) \to^E A \mid F}\ \text{T-NamedHandler}$$

By IH on (1), we have

$$[\![\Gamma]\!]_F, \text{🔒}_{[[\![E]\!]]_{[\![F]\!]}}, p : [\![A']\!], r : [[\![E]\!]]([\![B']\!] \to [\![A]\!]) \vdash [\![N]\!] : [\![A]\!] \mathbin{@} [\![E]\!]$$

By Lemma B.12 and structural rules, we have

$$[\![\Gamma]\!]_F, \text{🔒}_{[[\![E]\!]]_{[\![F]\!]}}, \text{🔒}_{[[\![E]\!]]}, p : [\![A']\!], r : [[\![E]\!]]([\![B']\!] \to [\![A]\!]) \vdash [\![N]\!] : [\![A]\!] \mathbin{@} [\![E]\!]\ (2)$$

By T-LetMod-Metn, T-Var-Metn, and $[\![A]\!]$ has kind Abs, we have

$$[\![\Gamma]\!]_F, \text{🔒}_{[[\![E]\!]]_{[\![F]\!]}}, \text{🔒}_{[[\![E]\!]]}, x : [[\![E]\!]][\![A]\!] \vdash \textbf{let mod}_{[[\![E]\!]]}\ x = x\ \textbf{in}\ x : [\![A]\!] \mathbin{@} [\![E]\!]\ (3)$$

By T-Mod-Metn and T-Abs-Metn, we have

$$a : \ell \vdash \textbf{mod}_{[a]}\ (\lambda x.\textbf{do}_a\ x) : [a]([\![A']\!] \to [\![B']\!]) \mathbin{@} a, [\![E]\!]$$

By T-Var-Metn, T-LetMod-Metn, and T-App-Metn, we have

$$\begin{aligned}&[\![\Gamma]\!]_F, \text{🔒}_{[[\![E]\!]]_{[\![F]\!]}}, f : \forall a^\ell.[a, [\![E]\!]]([a]([\![A']\!] \to [\![B']\!]) \to [\![A]\!]), \text{🔒}_{[[\![\ell, E]\!]]} \vdash \\ &\quad \textbf{let}\ f = f\ a\ \textbf{in let mod}_{[a,[\![E]\!]]}\ f = f\ \textbf{in}\ f\ (\textbf{mod}_{[a]}\ (\lambda x.\textbf{do}_a\ x)) : [\![A]\!] \mathbin{@} a, [\![E]\!]\ (4)\end{aligned}$$

By T-HandleName$^\spadesuit$-Metn, (2), (3), and (4), we have

$$[\![\Gamma]\!]_F, \text{🔒}_{[[\![E]\!]]_{[\![F]\!]}}, f : \forall a^\ell.[a, [\![E]\!]]([a]([\![A']\!] \to [\![B']\!]) \to [\![A]\!]) \vdash \textbf{handle}^\spadesuit_a\ M\ \textbf{with}\ H : [\![A]\!] \mathbin{@} [\![E]\!]$$

where

$$\begin{aligned}M &= \textbf{let mod}_{[a,[\![E]\!]]}\ f = f\ a\ \textbf{in}\ f\ (\textbf{mod}_{[a]}\ (\lambda x.\textbf{do}_a\ x))\\ H &= \{\textbf{return}\ x \mapsto \textbf{let mod}_{[[\![\ell,E]\!]]}\ x = x\ \textbf{in}\ x, \mathsf{op}\ p\ r \mapsto [\![N]\!]\}\end{aligned}$$

Our final goal follows from T-Abs-Metn, T-Mod-Metn, and Lemma B.12.

□

The proof relies on the following lemma.

Lemma D.1 (Pure Values). *Given a typing judgement $\Gamma \vdash V : A$ in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, if $[\![\Gamma]\!]_\cdot \vdash [\![V]\!] : [\![A]\!] \mathbin{@} \cdot$ then $[\![\Gamma]\!]_E \vdash [\![V]\!] : [\![A]\!] \mathbin{@} [\![E]\!]$ for any E.*

Proof. By straightforward induction on value typing judgements in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. The most non-trivial case is to show the accessibility of variables. Observe that the change from $[\![\Gamma]\!]_\cdot$ to $[\![\Gamma]\!]_E$ only changes the translations of locks. After translation, all variables in the context either have types of kind Abs or are annotated by an absolute modality. For variables with types of kind Abs, their accessibility is not influenced. For variables annotated with an absolute modality, by MT-Abs, upcasting the effect context can neither influence their accessibility. □

Lemma 8.3 (Semantics Preservation). *If M is consistent and well-typed and $M \mid \Omega \leadsto N \mid \Omega'$ in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$, then $[\![M]\!] \mid [\![\Omega]\!] \leadsto^* [\![N]\!] \mid [\![\Omega']\!]$ in Metn.*

Proof. By induction on M and case analysis on the next reduction rule. Note that values in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ are translated to value normal forms in Metn. We use the same letters for scope variables in System $\mathsf{F}^{\epsilon+\mathsf{sn}}$ and handler instances in Metn to make their correspondence clear.

Case E-App We have

$$[\![(\lambda^E x^A.M)\ V]\!] = \mathbf{let\ mod}_{[[\![E]\!]]}\ x = \mathbf{mod}_{[[\![E]\!]]}\ (\lambda x^{[\![A]\!]}.[\![M]\!])\ \mathbf{in}\ x\ [\![V]\!]$$

Our goal follows from E-LetMod and E-App in Metn. It is obvious that translation preserves value substitution.

Case E-TApp By E-EApp and E-NApp in Metn. It is obvious that translation preserves substitution of effect types and scope variables.

Case E-Let By syntactic sugar and E-App in Metn.

Case E-Mask We have

$$[\![\mathbf{mask}_\ell\ V]\!] = \mathbf{let\ mod}_{\langle \ell |\rangle}\ x = \mathbf{mask}_\ell\ [\![V]\!]\ \mathbf{in}\ x$$

Our goal follows from E-Mask and E-LetMod in Metn.

Case E-Gen

$$\mathbf{nhandler}\ H\ V \mid \Omega \quad \rightsquigarrow \quad \mathbf{handle}_h\ (\mathbf{let}\ x = V\ b\ \mathbf{in}\ x\ \mathbf{ev}_h)\ \mathbf{with}\ H \mid \Omega, h : \ell^b$$

We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!] & = & \mathbf{let\ mod}_{[[\![E]\!]]}\ f = \mathbf{mod}_{[[\![E]\!]]}\ M\ \mathbf{in}\ f\ [\![V]\!] \\ M & = & \lambda f.\mathbf{handle}^\spadesuit_a\ (\mathbf{let}\ g = f\ a\ \mathbf{in\ let\ mod}_{[[\![\mathrm{ev}\ \ell^a, E]\!]]}\ g = g\ \mathbf{in} \\ & & \quad g\ (\mathbf{mod}_{[a]}\ (\lambda x^{[\![A_{\mathrm{op}}]\!]}.\mathbf{do}_a\ x)))\ \mathbf{with}\ [\![H]\!] \end{array}$$

By E-LetMod, E-App, and E-Gen (set the generated instance to b) in Metn, it reduces to

$$\begin{array}{l} \mathbf{handle}^\spadesuit_b\ (\mathbf{let}\ g = [\![V]\!]\ b\ \mathbf{in\ let\ mod}_{[[\![\mathrm{ev}\ \ell^b, E]\!]]}\ g = g\ \mathbf{in} \\ g\ (\mathbf{mod}_{[b]}\ (\lambda x^{[\![A_{\mathrm{op}}]\!]}.\mathbf{do}_b\ x)))\ \mathbf{with}\ [\![H]\!] \end{array}$$

which is equal to $[\![\mathrm{RHS}]\!]$ of the above reduction step.

Case E-NRet By E-NRet$^\spadesuit$ and E-LetMod in Metn.

Case E-NOp

$$\mathbf{handle}_h\ \mathcal{E}[\mathbf{do\ ev}_h\ V]\ \mathbf{with}\ H \quad \rightsquigarrow \quad N[V/p, (\lambda y.\mathbf{handle}_h\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H)/r]$$

where $\Omega \ni h : \ell^b$. We have

$$[\![\mathrm{LHS}]\!] \quad = \quad \mathbf{handle}^\spadesuit_b\ [\![\mathcal{E}[\mathbf{do\ ev}_h\ V]]\!]\ \mathbf{with}\ [\![H]\!]$$

By Lemma D.2, we have

$$\begin{array}{rcl} [\![\mathcal{E}[\mathbf{do\ ev}_h\ V]]\!] & = & [\![\mathcal{E}]\!][\mathbf{let\ mod}_{[b]}\ f = [\![\mathbf{ev}_h]\!]\ \mathbf{in}\ f\ [\![V]\!]] \\ & = & [\![\mathcal{E}]\!][\mathbf{let\ mod}_{[b]}\ f = \mathbf{mod}_{[b]}\ (\lambda x.\mathbf{do}_b\ x)\ \mathbf{in}\ f\ [\![V]\!]] \end{array}$$

Thus, by E-LetMod and E-App in Metn, $[\![\mathrm{LHS}]\!]$ reduces to

$$\mathbf{handle}^\spadesuit_b\ [\![\mathcal{E}]\!][\mathbf{do}_b\ [\![V]\!]]\ \mathbf{with}\ [\![H]\!]$$

Our goal follows from E-NOp$^\spadesuit$ in Metn.

Case E-Ret By E-Ret$^\spadesuit$ and E-LetMod in Metn.

Case E-Handler Similar to the E-Gen case.

Case E-Op Similar to the E-NOp case.

Case E-Lift By IH and Lemma D.2.

□

The proof of semantics preservation relies on the following lemma.

Lemma D.2 (Translation of Effect Contexts). *For the translation* $[\![-]\!]$ *from System* $\mathsf{F}^{\epsilon+\mathsf{sn}}$ *to* Metn*, we have* $[\![\mathcal{E}[M]]\!] = [\![\mathcal{E}]\!][[\![M]\!]]$ *for any evaluation context* $\mathcal{E}$ *and term* M.

Proof. By straightforward case analysis on evaluation contexts of System $\mathsf{F}^{\epsilon+\mathsf{sn}}$. □

D.2 Proof of Encoding System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$

We first prove a few lemmas.

Lemma D.3 (First Modality Transformation). *For all* E_1, E_2, E_3 *with no effect variables:*

$$\Gamma \vdash \langle E_1 - E_2 | E_2 - E_1 \rangle \circ \langle E_2 - E_3 | E_3 - E_2 \rangle \Leftrightarrow \langle E_1 - E_3 | E_3 - E_1 \rangle \ @\ E_1$$

Proof. Observe that when there are no effect variables, syntax of masks L, extensions D, and effect contexts E are the same, and the meta operations $E + F$ and $E - F$ defined on them are basically union and difference on multisets (since effect labels ℓ could appear multiple times). We do a set-theoretical analysis here. The composition on LHS gives

$$\langle (E_1 - E_2) + L | (E_3 - E_2) + D \rangle.$$

where $L = (E_2 - E_3) - (E_2 - E_1) \equiv (E_1 - E_3) \cap E_2$ and $D = (E_2 - E_1) - (E_2 - E_3) \equiv (E_3 - E_1) \cap E_2$. We also have

$$\begin{aligned} E_1 - E_3 &\equiv ((E_1 - E_3) - E_2) + L \\ E_3 - E_1 &\equiv ((E_3 - E_1) - E_2) + D \end{aligned}$$

Thus,

$$\begin{array}{lclcl} (E_1 - E_2) + L & \equiv & ((E_1 - E_3) - E_2) + (E_1 \cap E_3 - E_2) + L & \equiv & (E_1 - E_3) + (E_1 \cap E_3 - E_2) \\ (E_3 - E_2) + D & \equiv & ((E_3 - E_1) - E_2) + (E_3 \cap E_1 - E_2) + D & \equiv & (E_3 - E_1) + (E_3 \cap E_1 - E_2) \end{array}$$

By MT-Shrink, the transformation from left to right holds; by MT-Expand, the transformation from right to left holds. □

Lemma D.4 (Second Modality Transformation). *For all* ℓ, E, F*:*

$$\Gamma \vdash \langle \ell \rrparenthesis \circ \langle E - F | F - E \rangle \Rightarrow \langle (\ell, E) - F | F - (\ell, E) \rangle \ @\ \ell, E$$

Proof. Case analysis on whether F has more ℓ than E.

Case More ℓ in F. By MT-Shrink.

Case More ℓ in E or equal. Both sides are equivalent.

□

Lemma D.5 (Third Modality Transformation). *For all* ℓ, E, F*:*

$$\Gamma \vdash \llparenthesis \ell \rangle \circ \langle (\ell, E) - F | F - (\ell, E) \rangle \Rightarrow \langle E - F | F - E \rangle \ @\ E$$

Proof. Case analysis on whether F has more ℓ than E.

Case More ℓ in F. Both sides are equivalent.

Case More ℓ in E or equal. By MT-Shrink.

□

Lemma D.6 (Fourth Modality Transformation). *For all* a, E, F*:*

$$\Gamma, a :_\sharp \ell \vdash \langle (a, E) - F | F - (a, E) \rangle \Rightarrow \langle a \rrparenthesis \circ \langle E - F | F - E \rangle \ @\ a, E$$

Proof. Since a is the rightmost handler name in the context with index $\sharp$, we know that it is different from all other names, i.e., $a \backslash b \equiv a$ for all b. This allows us to discuss the number of a in E and F. Case analysis on whether F has more a than E.

Case More a in F. By MT-Expand.
Case More a in E or equal. Both sides are equivalent.

□

Lemma D.7 (Translating Instantiated Types). *For all System* $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ *types* A*, we have* $[\![A]\!]_E = [\![A[E'/]]\!]_{E,E'}$.

Proof. By induction on the type A.

Case $A = 1$. Trivial.
Case $A = \forall.A'$. Trivial.
Case $A = A' \to^F B'$. We have

$$[\![A]\!]_E = \langle E - F | F - E \rangle([\![A']\!]_F \to [\![B']\!]_F)$$
$$[\![A[E'/]]\!]_{E,E'} = \langle E, E' - F, E' | F, E' - E, E' \rangle([\![A'[E'/]]\!]_{F,E'} \to [\![B'[E'/]]\!]_{F,E'})$$

By the induction hypothesis we have:

$$[\![A']\!]_F = [\![A'[E'/]]\!]_{F,E'}$$
$$[\![B']\!]_F = [\![B'[E'/]]\!]_{F,E'}$$

Finally we have

$$\langle E, E' - F, E' | F, E' - E, E' \rangle = \langle E', E - E', F | E', F - E', E \rangle = \langle E - F | F - E \rangle$$

Case $A = \forall a^\ell.A'$. Follow from a case analysis on A' similar to above cases.

□

With these lemmas, we can now prove type preservation.

Theorem 8.5 (Type Preservation). *If* $\Gamma \vdash_\varepsilon M : A \mid E$ *is consistent and well-formed in System* $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$*, then* $[\![\Gamma]\!]_E \vdash [\![M]\!]_E : [\![A]\!]_E \mathbin{@} [\![E]\!]$ *in* Metn*. Similarly for typing judgements of values.*

Proof. By induction on the typing judgement $\Gamma \vdash_\varepsilon M : A \mid E$. As a visual aid, we repeat each rule where we replace the translation premises by the Metn judgement implied by the induction hypothesis and the translation in the conclusion by the Metn judgement we need to prove. For brevity, we use E to refer to both an effect type in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ and its translation in Metn since the translation of effect types E is trivial.

Case $\boxed{()}$ By T-Unit-System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$, T-Unit-Metn, T-Mod-Metn, and $\vdash 1 : \mathsf{Abs}$.
Case $\boxed{x}$

$$\frac{\mu \coloneqq \mathsf{topmod}([\![A]\!]_E)}{[\![\Gamma_1, x : A, \Gamma_2]\!]_E \vdash \mathbf{mod}_\mu\ \hat{x} : [\![A]\!]_E \mathbin{@} E}$$

By case analysis on the type A:

Case $A = 1$. We can use T-Var-Metn since $\cdot \vdash 1 : \mathsf{Abs}$.
Case $A = \forall.A'$. Then $\mathsf{topmod}([\![A]\!]_F) = []$ for all F. We can use T-Var-Metn by MT-Abs.
Case $A = \mathsf{ev}\ \ell^a$. Then $\mathsf{topmod}([\![A]\!]_F) = [a]$ for all F. We can use T-Var-Metn by MT-Abs.
Case $A = A' \to^F B'$. By well-formedness of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ judgement, we have that $\mathsf{locks}(\Gamma_2)$ is a relative modality. Suppose $\mathsf{locks}(\Gamma_2) : E \to F'$. Let $\nu_{F'} = \langle F' - E | E - F' \rangle_{F'}$ which is the canonical relative modality from E to F'. It is easy to see that $\nu_{F'} \Rightarrow \mathsf{locks}(\Gamma_2)$. We also have that $\mu = \langle E - F | F - E \rangle$ and x is annotated by the modality $\langle F' - F | F - F' \rangle_{F'} : F \to F'$. Lemma D.3 gives $\langle F' - F | F - F' \rangle_{F'} \Rightarrow \nu_{F'} \circ \mu_E$, which further gives $\langle F' - F | F - F' \rangle_{F'} \Rightarrow \mathsf{locks}(\Gamma_2) \circ \mu_E$. Thus we can use T-Var-Metn.

Case $A = \forall \overline{a^\ell}.A'$ where A' does not have top-level bindings of scope variables. Observe that the translation of scope abstraction basically moves the inner modality to the top level. Follow from case analysis on the shape of A' similar to the four cases above.

Case $\boxed{\Lambda.V}$

$$\frac{\llbracket \Gamma, \clubsuit_E \rrbracket_{\cdot} \vdash \llbracket V \rrbracket : \llbracket A \rrbracket_{\cdot} \ @\ \cdot}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{mod}_{[]}\ \llbracket V \rrbracket : \llbracket \forall.A \rrbracket_E \ @\ E}$$

We have $\llbracket \Gamma, \clubsuit_E \rrbracket_{\cdot} = \llbracket \Gamma \rrbracket_E, \text{🔒}_{[]}$ and $\llbracket \forall.A \rrbracket_E = [] \llbracket A \rrbracket_{\cdot}$. Our goal follows from T-Mod-Metn.

Case $\boxed{V\,\#}$

$$\frac{\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket \forall.A \rrbracket_E \ @\ E}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{let\ mod}_{[]}\ x = \llbracket V \rrbracket\ \mathbf{in}\ x : \llbracket A[E/] \rrbracket_E \ @\ E}$$

We have $\llbracket \forall.A \rrbracket_E = [] \llbracket A \rrbracket_{\cdot}$. By Lemma D.7, we have $\llbracket A \rrbracket_{\cdot} = \llbracket A[E/] \rrbracket_E$. Our goal follows from T-Letmod-Metn.

Case $\boxed{\Lambda a^\ell.V}$

$$\frac{\llbracket \Gamma, a : \ell \rrbracket_E \vdash \llbracket V \rrbracket_E : \llbracket A \rrbracket_E \ @\ E}{\llbracket \Gamma \rrbracket_E \vdash M : \llbracket \forall a^\ell.A \rrbracket_E \ @\ E}$$

where $\mathrm{topmod}(\llbracket A \rrbracket_E) = \mu B$ and M depends on μ. Case analysis on μ.

Case $\boxed{\mu \equiv \nu \circ \langle\!| a \rangle}$ We have $a \notin \nu$ and

$$M = \mathbf{let\ mod}_\nu\ \lambda a^\ell.x = (\mathbf{let\ mod}_\mu\ x = \llbracket V \rrbracket\ \mathbf{in\ mod}_{\nu;\langle\!| a \rangle}\ x)\ \mathbf{in\ mod}_\nu\ x$$

and $\llbracket \forall a^\ell.A \rrbracket_E = \nu(\forall a^\ell.\langle\!| a \rangle B)$. Our goal follows from T-Letmod'-Metn and T-Mod-Metn. The most non-trivial step is that the top-level let-binding of M introduces a variable binding $x :_\nu \forall a^\ell.\langle\!| a \rangle B$ to the context, which can be used in $\mathbf{mod}_\nu\ x$. The inner let-binding splits the top-level modality of $\llbracket V \rrbracket$ from μ to $\nu \circ \langle\!| a \rangle$.

Case $\boxed{\text{otherwise}}$ We have $a \notin \mu$ and $M = \mathbf{let\ mod}_\nu\ \lambda a^\ell.x = \llbracket V \rrbracket\ \mathbf{in\ mod}_\mu\ x$ and $\llbracket \forall a^\ell.A \rrbracket_E = \mu(\forall a^\ell.B)$. Our goal follows from T-Letmod'-Metn and T-Mod-Metn.

Case $\boxed{\mathbf{return}\ V}$ By IH.

Case $\boxed{\mathbf{let}\ x = M\ \mathbf{in}\ N}$ By IHs, syntactic sugar, T-App-Metn, and T-Letmod-Metn.

Case $\boxed{V\ b}$

$$\frac{\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket \forall a^\ell.A \rrbracket_E \ @\ E \qquad \llbracket \Gamma \rrbracket_E \ni b : \ell \qquad \mu B := \llbracket A \rrbracket_E}{\llbracket \Gamma \rrbracket_E \vdash V' : \llbracket A[b/a] \rrbracket_E \ @\ E}$$

where V' depends on μ. By case analysis on μ.

Case $\boxed{\mu \equiv \nu \circ \langle\!| a \rangle}$ We have

$$V' = \mathbf{let\ mod}_\nu\ x = \llbracket V \rrbracket\ \mathbf{in\ let}_\nu\ \mathbf{mod}_{\langle\!| a \rangle}\ y = x\ b\ \mathbf{in\ mod}_\mu\ y$$

and $\llbracket \forall a^\ell.A \rrbracket_E = \nu(\forall a^\ell.\langle\!| a \rangle B)$. We have $a \notin \nu$. It is obvious that type translation preserves name substitution. Thus we have $\llbracket A \rrbracket_E[b/a] \equiv \llbracket A[b/a] \rrbracket_E$. Our goal follows from T-NApp-Metn, T-Letmod-Metn and T-Mod-Metn.

Case $\boxed{\text{otherwise}}$ We have $V' = \mathbf{let\ mod}_\mu\ x = \llbracket V \rrbracket\ \mathbf{in\ mod}_\mu\ (x\ b)$ and $\llbracket \forall a^\ell.A \rrbracket_E = \mu(\forall a^\ell.B)$. We have $a \notin \mu$. Our goal follows from T-Letmod-Metn, T-Mod-Metn, and $\llbracket A \rrbracket_E[b/a] \equiv \llbracket A[b/a] \rrbracket_E$.

Case $\boxed{\lambda^F x^A.M}$

$$\frac{[\![\Gamma, \blacklozenge_E, x : A]\!]_F \vdash [\![M]\!] : [\![B]\!]_F \;@\; F \qquad \mu \coloneqq \mathrm{topmod}([\![A]\!]_F)}{[\![\Gamma]\!]_E \vdash \mathbf{mod}_{\langle E-F|F-E\rangle}\; (\lambda x^{[\![A]\!]_F}.\mathbf{let\ mod}_\mu\; \hat{x} = x \;\mathbf{in}\; [\![M]\!]) : [\![A \to^F B]\!]_E \;@\; E}$$

We have $[\![\Gamma, \blacklozenge_E, x : A]\!]_F = [\![\Gamma]\!]_E, \text{🔒}_{\langle E-F|F-E\rangle}, x : \mu A, \hat{x} :_{\mu_F} A'$ where $\mu A' = [\![A]\!]_F$. Further $[\![A \to^F B]\!]_E = \langle E-F|F-E\rangle([\![A]\!]_F \to [\![B]\!]_F)$. Our goal follows from T-LetMod-Metn, T-Abs-Metn and T-Mod-Metn.

Case $\boxed{V\ W}$

$$\frac{\begin{array}{c}[\![\Gamma]\!]_E \vdash [\![V]\!] : [\![A \to^E B]\!]_E \;@\; E \\ [\![\Gamma]\!]_E \vdash [\![W]\!] : [\![A]\!]_E \;@\; E\end{array}}{[\![\Gamma]\!]_E \vdash \mathbf{let\ mod}_{\langle\rangle}\; x = [\![V]\!] \;\mathbf{in}\; x\; [\![W]\!] : [\![B]\!]_E \;@\; E}$$

We have $[\![A \to^E B]\!]_E = \langle\rangle([\![A]\!]_E \to [\![B]\!]_E)$. Our goal follows from T-LetMod-Metn and T-App-Metn.

Case $\boxed{\mathbf{do}\ \mathsf{op}\ V}$

$$\frac{\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \qquad [\![\Gamma]\!]_{\ell,E} \vdash [\![V]\!] : [\![A]\!]_{\ell,E} \;@\; \ell, E}{[\![\Gamma]\!]_{\ell,E} \vdash \mathbf{do}\ \mathsf{op}\ [\![V]\!] : [\![B]\!]_{\ell,E} \;@\; \ell, E}$$

Since A and B have kind Abs, we have $[\![A]\!]_{\ell,E} = [\![A]\!]_\cdot$ and $[\![B]\!]_{\ell,E} = [\![B]\!]_\cdot$. Our goal follows directly from T-Do-Metn.

Case $\boxed{\mathbf{do}\ V\ W}$

$$\frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op} : A \twoheadrightarrow B\} \\ [\![\Gamma]\!]_{\ell^a,E} \vdash [\![V]\!] : [\![\mathsf{ev}\ \ell^a]\!]_{\ell^a,E} \;@\; a, E \qquad [\![\Gamma]\!]_{\ell^a,E} \vdash [\![W]\!] : [\![A]\!]_{\ell^a,E} \;@\; a, E\end{array}}{[\![\Gamma]\!]_{\ell^a,E} \vdash \mathbf{let\ mod}_{[a]}\; x = [\![V]\!] \;\mathbf{in}\; x\; [\![W]\!] : [\![B]\!]_{\ell^a,E} \;@\; a, E}$$

Since A and B have kind Abs, we have $[\![A]\!]_{\ell,E} = [\![A]\!]_\cdot$ and $[\![B]\!]_{\ell,E} = [\![B]\!]_\cdot$. We also have $[\![\mathsf{ev}\ \ell^a]\!]_{\ell,E} = [a]([\![A]\!]_\cdot \to [\![B]\!]_\cdot)$. Our goal follows from T-LetMod-Metn and T-App-Metn.

Case $\boxed{\mathbf{mask}_\ell\ M}$

$$\frac{\begin{array}{c}[\![\Gamma, \blacklozenge_{\ell,E}]\!]_E \vdash [\![M]\!] : [\![A]\!]_E \;@\; E \\ \mu \coloneqq \mathrm{topmod}([\![A]\!]_E) \qquad \nu \coloneqq \mathrm{topmod}([\![A]\!]_{\ell,E})\end{array}}{[\![\Gamma]\!]_{\ell,E} \vdash \mathbf{let\ mod}_{\langle \ell | \rangle;\mu}\; x = \mathbf{mask}_\ell\; [\![M]\!] \;\mathbf{in\ mod}_\nu\; x : [\![A]\!]_{\ell,E} \;@\; \ell, E}$$

We have $[\![\Gamma, \blacklozenge_{\ell,E}]\!]_E = [\![\Gamma]\!]_{\ell,E}, \text{🔒}_{\langle \ell | \rangle}$. Our goal follows from T-LetMod-Metn, T-Mask-Metn and T-Mod-Metn if we can show that x can be accessed under the box. For units and effect abstraction, this is trivial. For functions, we need to show the transformation $\langle \ell | \rangle \circ \mu \Rightarrow \nu \;@\; \ell, E$ holds, which follows from Lemma D.4. For name abstraction, similar to other cases as its translation basically moves the inner modality to the top level.

Case $\boxed{\mathbf{handler}\ \{\mathsf{op}\ p\ r \mapsto N\}}$

$$\frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad [\![\Gamma, \blacklozenge_E, p : A', r : B' \to^F A]\!]_F \vdash [\![N]\!] : [\![A]\!]_F \;@\; F\ (1) \\ N_1 \coloneqq \mathbf{let\ mod}_{\langle | \ell \rangle;\mu}\; x = x \;\mathbf{in\ mod}_\nu\; x \\ N_2 \coloneqq \mathbf{let\ mod}_{\mu_p}\; \hat{p} = p \;\mathbf{in\ let\ mod}_{\langle\rangle}\; \hat{r} = r \;\mathbf{in}\; [\![N]\!] \\ \mu_p \coloneqq \mathrm{topmod}([\![A']\!]_F) \qquad \mu \coloneqq \mathrm{topmod}([\![A]\!]_{\ell,F}) \qquad \nu \coloneqq \mathrm{topmod}([\![A]\!]_F)\end{array}}{\begin{array}{c}[\![\Gamma]\!]_E \vdash \mathbf{mod}_{\langle E-F|F-E\rangle}\; (\lambda f.\mathbf{let\ mod}_{\langle | \ell \rangle}\; f = f \;\mathbf{in} \\ \mathbf{handle}\; f\; ()\; \mathbf{with}\; \{\mathbf{return}\; x \mapsto N_1, \mathsf{op}\; p\; r \mapsto N_2\}) : [\![(1 \to^{\ell,F} A) \to^F A]\!]_E \;@\; E\end{array}}$$

We have $[\![(1 \to^{\ell,F} A) \to^F A]\!]_E = \langle E - F|F - E\rangle(\langle\!|\ell\rangle(1 \to [\![A]\!]_{\ell,F}) \to [\![A]\!]_F)$. Since A' and B' have kind Abs, we have $[\![A']\!]_{F'} = [\![A']\!]_.$ and $[\![B']\!]_{F'} = [\![B']\!]_.$ for any F'. For the operation clause, by (1), $[\![B' \to^F A]\!]_F = \langle\rangle([\![B']\!]_F \to [\![A]\!]_F)$, and T-LetMod-Metn, we have

$$[\![\Gamma]\!]_E, \blacksquare_{\langle E-F|F-E\rangle}, p : [\![A']\!]_F, r : \langle\rangle([\![B']\!]_F \to [\![A]\!]_F) \vdash N_2 : [\![A]\!]_F \ @\ F\ (2)$$

For the return clause, by T-LetMod-Metn and T-Mod-Metn, we have

$$[\![\Gamma]\!]_E, \blacksquare_{\langle E-F|F-E\rangle}, x : \langle\!|\ell\rangle[\![A]\!]_{\ell,F} \vdash N_1 : [\![A]\!]_F \ @\ F\ (3)$$

If we can show the accessibility of x in N_1. When A is unit or effect abstraction, this is trivial. When A is a function type, we need to show the transformation $\langle\!|\ell\rangle \circ \mu \Rightarrow \nu \ @\ F$ holds, which follows from Lemma D.5. When A is a name abstraction, follow by a case analysis similar to other cases above. Our final goal then follows from (2), (3), T-LetMod-Metn, T-App-Metn, T-Abs-Metn, T-Mod-Metn, and T-Handle-Metn.

Case $\boxed{\mathbf{nhandler}\ \{\mathsf{op}\ p\ r \mapsto N\}}$

$$\frac{\begin{array}{c}\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad [\![\Gamma, \blacklozenge_E, p : A', r : B' \to^F A]\!]_F \vdash [\![N]\!] : [\![A]\!]_F \ @\ F\ (1) \\ M := \mathbf{let\ mod}_{\langle\!|a\rangle}\ f' = f\ a\ \mathbf{in\ let\ mod}_\mu\ y = f'\ (\mathbf{mod}_{[a]}\ (\lambda x.\mathbf{do}_a\ x))\ \mathbf{in\ mod}_{\langle a|\!\rangle;\nu}\ y \\ N_2 := \mathbf{let\ mod}_{\mu_p}\ \hat{p} = p\ \mathbf{in\ let\ mod}_{\langle\rangle}\ \hat{r} = r\ \mathbf{in}\ [\![N]\!] \\ \mu_p := \mathrm{topmod}([\![A']\!]_F) \qquad \mu := \mathrm{topmod}([\![A]\!]_{\ell^a,F}) \qquad \nu := \mathrm{topmod}([\![A]\!]_F)\end{array}}{\begin{array}{c}[\![\Gamma]\!]_E \vdash \mathbf{mod}_{\langle E-F|F-E\rangle}\ (\lambda f.\mathbf{handle}_a\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto x, \mathsf{op}\ p\ r \mapsto N_2\}) \\ : [\![(\forall a^\ell.\mathsf{ev}\ \ell^a \to^{a,F} A) \to^F A]\!]_E \ @\ E\end{array}}$$

We have

$$\begin{array}{ll} & [\![(\forall a^\ell.\mathsf{ev}\ \ell^a \to^{\ell^a,F} A) \to^F A]\!]_E \\ = & \langle E - F|F - F\rangle((\forall a^\ell.\langle\!|a\rangle([a]([\![A']\!]_a \to [\![B']\!]_a) \to [\![A]\!]_{a,F})) \to [\![A]\!]_F)\end{array}$$

Since A' and B' have kind Abs, we have $[\![A']\!]_{F'} = [\![A']\!]_.$ and $[\![B']\!]_{F'} = [\![B']\!]_.$ for any F'. For the operation clause, (1), $[\![B' \to^F A]\!]_F = \langle\rangle([\![B']\!]_F \to [\![A]\!]_F)$, and T-LetMod-Metn, gives

$$[\![\Gamma]\!]_E, \blacksquare_{\langle E-F|F-E\rangle}, p : [\![A']\!]_F, r : \langle\rangle([\![B']\!]_F \to [\![A]\!]_F) \vdash N_2 : [\![A]\!]_F \ @\ F\ (2)$$

For the return clause, we have

$$[\![\Gamma]\!]_E, \blacksquare_{\langle E-F|F-E\rangle}, x : [\![A]\!]_F \vdash x : [\![A]\!]_F \ @\ F\ (3)$$

For the computation M to be handled, we need to show

$$[\![\Gamma]\!]_E, \blacksquare_{\langle E-F|F-E\rangle}, f : A_f, a :_\sharp \ell, \blacksquare_{\langle\!|a\rangle_F} \vdash M : \langle a|\!\rangle[\![A]\!]_F \ @\ F\ (4)$$

where $A_f = \forall a^\ell.\langle\!|a\rangle([a]([\![A']\!]_a \to [\![B']\!]_a) \to [\![A]\!]_{a,F})$. This mostly follows from straightforward application of various typing rules in Metn. The only non-trivial part is to show the accessibility of y under $\mathbf{mod}_{\langle a|\!\rangle;\nu}$ in M. When A is unit or effect abstraction, this is trivial. When A is a function type, we need to show the transformation $[\![\Gamma]\!]_F, a :_\sharp \ell \vdash \mu \Rightarrow \langle a|\!\rangle \circ \nu \ @\ a, F$ holds, which follows from Lemma D.6. When A is a name abstraction, similar to other cases before. Our final goal then follows from (2), (3), (4), T-Abs-Metn, T-Mod-Metn, and T-HandleName-Metn.

□

Theorem 8.6 (Semantics Preservation). *If M is consistent and well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System $\mathsf{F}_1^{\epsilon+\mathrm{sn}}$, then there exists N' in Metn such that $[\![M]\!] \mid [\![\Omega]\!] \rightsquigarrow^* N' \mid [\![\Omega']\!]$ and $[\![N]\!] \mid [\![\Omega']\!] \rightsquigarrow_v^* N' \mid [\![\Omega']\!]_.$ in Metn, where $\rightsquigarrow_v$ refers to reduction of values in Metn.*

Proof. By induction on M and case analysis on the next reduction rule. Values in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ are translated to values in Metn, and we can always further reduce the translated values to value normal forms. For brevity, we do not distinguish between effect rows E and their translations. We also use the same letters for scope variables in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ and handler instances in Metn to make their correspondence clear. We ignore most of instance contexts Ω on the reduction relations as they are global and only increase. The translation of terms depends on types and effect contexts. We sometimes write $[\![M]\!]_E$ to emphasis that term M is at effect context E.

Case E-App We have $\langle E - E | E - E \rangle = \langle\rangle$ and

$$[\![(\lambda^E x^A.M)\ V : \_ \mid E]\!] = \mathbf{let\ mod}_{\langle\rangle}\ f = \mathbf{mod}_{\langle\rangle}\ (\lambda x^{[\![A]\!]_E}.\mathbf{let\ mod}_\mu\ x = x\ \mathbf{in}\ [\![M]\!])\ \mathbf{in}\ f\ [\![V]\!]$$

where $\mu = \mathrm{topmod}([\![A]\!]_E)$. Our goal follows from E-Letmod and E-App in Metn. It is obvious that translation preserves value substitution.

Case E-EApp

$$(\Lambda.V)\# \quad \rightsquigarrow \quad V[E/]$$

where the term is at effect context E. We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!]_E & = & \mathbf{let\ mod}_{[]}\ x = \mathbf{mod}_{[]}\ [\![V]\!]_\cdot\ \mathbf{in}\ x \\ [\![\mathrm{RHS}]\!]_E & = & [\![V[E/]]\!]_E \end{array}$$

We have $[\![\mathrm{LHS}]\!]_E \rightsquigarrow^* [\![V]\!]_\cdot$. By a straightforward case analysis on V, we can show that $[\![V]\!]_\cdot = [\![V[E/]]\!]_E$. The most interesting case is when V is a lambda abstraction $\lambda^F x.M$, where we have $\langle \cdot - F | F - \cdot \rangle = \langle E - (F,E) | (F,E) - E \rangle$.

Case E-NApp

$$(\Lambda\alpha^\ell.V)\ b \quad \rightsquigarrow \quad V[b/a]$$

where the term is at effect context E and V has type A. Let $\mu = \mathrm{topmod}([\![A]\!]_E)$. Suppose $[\![V]\!] \rightsquigarrow^* U$. By type soundness of Metn and Lemma B.8, we have $U = \mathbf{mod}_\mu\ U'$. Case analysis on μ.

Case $\mu \equiv \nu \circ \langle\!| a \rangle$ We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!] & = & \mathbf{let\ mod}_\nu\ x = W\ \mathbf{in\ let}_\nu\ \mathbf{mod}_{\langle\!| a\rangle}\ y = x\ b\ \mathbf{in\ mod}_\mu\ y \\ W & = & \mathbf{let\ mod}_\nu\ \lambda a^\ell.x = (\mathbf{let\ mod}_\mu\ x = [\![V]\!]\ \mathbf{in\ mod}_{\nu\circ\langle\!| a\rangle}\ x)\ \mathbf{in\ mod}_\nu\ x \end{array}$$

It is easy to show that $[\![\mathrm{LHS}]\!] \rightsquigarrow^* (\mathbf{mod}_\mu\ U')[b/a]$. Our goal follows from $[\![\mathrm{RHS}]\!] \rightsquigarrow_n^* (\mathbf{mod}_\mu\ U')[b/a]$.

Case otherwise We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!] & = & \mathbf{let\ mod}_\mu\ x = W\ \mathbf{in\ mod}_\mu\ (x\ b) \\ W & = & \mathbf{let\ mod}_\mu\ \lambda a^\ell.x = [\![V]\!]\ \mathbf{in\ mod}_\mu\ x \end{array}$$

It is easy to show that $[\![\mathrm{LHS}]\!] \rightsquigarrow^* \mathbf{mod}_\mu\ U'[b/a]$. Our goal follows from $[\![\mathrm{RHS}]\!] \rightsquigarrow_n^* \mathbf{mod}_\mu\ U'[b/a]$.

Case E-Let By syntactic sugar, E-Letmod, and E-App in Metn.

Case E-Mask We have

$$[\![\mathbf{mask}_\ell\ V]\!]_{\ell,E} = \mathbf{let\ mod}_{\langle \ell |\!\rangle;\mu}\ x = \mathbf{mask}_\ell\ [\![V]\!]_E\ \mathbf{in\ mod}_\nu\ x$$

where $\mu = \mathrm{topmod}([\![A]\!]_E)$ and $\nu = \mathrm{topmod}([\![A]\!]_{\ell,E})$. By syntactic sugar, E-Mask, and E-Letmod in Metn, it reduces to

$$\mathbf{let}_{\langle \ell |\!\rangle}\ \mathbf{mod}_\mu\ x = [\![V]\!]_E\ \mathbf{in\ mod}_\nu\ x$$

Suppose $[\![V]\!]_E \rightsquigarrow^* U$, by type soundness of METN and Lemma B.8, we have $U = \mathbf{mod}_\mu\ U'$. Thus, by E-LETMOD-METN, the above term further reduces to

$$\mathbf{mod}_\nu\ U'$$

By Lemma D.9, $[\![V]\!]_{\ell,E} \rightsquigarrow^*_n \mathbf{mod}_\nu\ U'$.

Case E-GEN

$$\mathbf{nhandler}\ H\ V \quad {}_\Omega\!\rightsquigarrow_{\Omega,h:\ell^b} \quad \mathbf{handle}_h\ (\mathbf{let}\ x = V\ b\ \mathbf{in}\ x\ \mathbf{ev}_h)\ \mathbf{with}\ H$$

Suppose the term is at effect context E. We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!] & = & \mathbf{let\ mod}_{\langle\rangle}\ g = \mathbf{mod}_{\langle\rangle}\ W\ \mathbf{in}\ g\ [\![V]\!]_E \\ W & = & \lambda f.\mathbf{handle}_a\ (\mathbf{let\ mod}_{\langle| a\rangle}\ f' = f\ a\ \mathbf{in} \\ & & \mathbf{let\ mod}_\mu\ y = f'\ (\mathbf{mod}_{[a]}\ (\lambda x.\mathbf{do}_a\ x))\ \mathbf{in\ mod}_{\langle a|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!] \\ [\![\mathrm{RHS}]\!] & = & \mathbf{handle}_h\ (\mathbf{let}_{\mu'}\ \mathbf{mod}_y\ = M\ \mathbf{in\ mod}_{\langle b|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!] \\ M & = & \mathbf{let}\ x = [\![V]\!]_{h,E}\ b\ \mathbf{in}\ [\![x\ \mathbf{ev}_h]\!] \end{array}$$

where $\mu = \mathrm{topmod}([\![A]\!]_{\ell^a,F})$ and $\nu = \mathrm{topmod}([\![A]\!]_F)$ and $\mu' = \mathrm{topmod}([\![A]\!]_{h,F})$ By E-LETMOD, E-APP, and E-GEN (set the generated instance to b) in METN, $[\![LHS]\!]$ reduces to

$$\begin{array}{l} \mathbf{handle}_b\ (\mathbf{let\ mod}_{\langle| b\rangle}\ f' = [\![V]\!]_E\ b\ \mathbf{in} \\ \mathbf{let\ mod}_{\mu'}\ y = f'\ (\mathbf{mod}_{[b]}\ (\lambda x.\mathbf{do}_b\ x))\ \mathbf{in\ mod}_{\langle b|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!] \end{array}$$

Suppose $[\![V]\!]\ b \rightsquigarrow^* U$. By type-soundness of METN and Lemma B.8, we have $U = \mathbf{mod}_{\langle| b\rangle}\ U'$. Thus, by E-LETMOD-METN, the above term further reduces to

$$N = \mathbf{handle}_b\ (\mathbf{let\ mod}_{\mu'}\ y = U'\ (\mathbf{mod}_{[b]}\ (\lambda x.\mathbf{do}_b\ x))\ \mathbf{in\ mod}_{\langle b|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!]$$

By Lemma D.9, we have $[\![\mathrm{RHS}]\!] \rightsquigarrow^*_n N$.

Case E-NRET

$$\mathbf{handle}_h\ V\ \mathbf{with}\ H \quad \rightsquigarrow \quad \mathbf{return}\ V$$

Suppose the term is at effect context E and $\Omega \ni h : \ell^b$. We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!]_E & = & \mathbf{handle}_b\ (\mathbf{let\ mod}_\mu\ y = [\![V]\!]_{b,E}\ \mathbf{in\ mod}_{\langle b|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!] \\ & & \text{where } \mu = \mathrm{topmod}([\![A]\!]_{b,E}) \text{ and } \nu = \mathrm{topmod}([\![A]\!]_E) \\ [\![\mathrm{RHS}]\!]_E & = & [\![V]\!]_E \end{array}$$

Our goal follows from Lemma D.9, E-LETMOD and E-NRET♠ in METN.

Case E-NOP

$$\mathbf{handle}_h\ \mathcal{E}[\mathbf{do\ ev}_h\ V]\ \mathbf{with}\ H \quad \rightsquigarrow \quad N[V/p, (\lambda y.\mathbf{handle}_h\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H)/r]$$

Suppose the term is at effect context E and $\Omega \ni h : \ell^b$. We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!]_E & = & \mathbf{handle}_b\ (\mathbf{let\ mod}_\mu\ y = [\![\mathcal{E}[\mathbf{do\ ev}_b\ V]]\!]\ \mathbf{in\ mod}_{\langle b|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!] \\ [\![\mathrm{RHS}]\!]_E & = & [\![N]\!]_E[[\![V]\!]/p, [\![(\lambda y.\mathbf{handle}_h\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H)]\!]/r] \end{array}$$

By Lemma D.2, we have

$$\begin{array}{rcl} [\![\mathcal{E}[\mathbf{do\ ev}_h\ V]]\!] & = & [\![\mathcal{E}]\!][\mathbf{let\ mod}_{[b]}\ f = [\![\mathbf{ev}_h]\!]\ \mathbf{in}\ f\ [\![V]\!]] \\ & = & [\![\mathcal{E}]\!][\mathbf{let\ mod}_{[b]}\ f = \mathbf{mod}_{[b]}\ (\lambda x.\mathbf{do}_b\ x)\ \mathbf{in}\ f\ [\![V]\!]] \end{array}$$

Thus, by E-LETMOD and E-APP in METN, $[\![\mathrm{LHS}]\!]$ reduces to

$$\mathbf{handle}_b\ (\mathbf{let\ mod}_\mu\ y = [\![\mathcal{E}]\!][\mathbf{do}_b\ [\![V]\!]]\ \mathbf{in\ mod}_{\langle b|\rangle;\nu}\ y)\ \mathbf{with}\ [\![H]\!]$$

Note that V is absolute and thus its translation does not depend on the effect context. Our goal follows from E-NOP in METN.

Case E-Handler

$$\textbf{handler}\ H\ V \quad\rightsquigarrow\quad \textbf{handle}\ V\ ()\ \textbf{with}\ H$$

Suppose the term is at effect context E. We have

$$\begin{array}{rcl} [\![\text{LHS}]\!] &=& \textbf{let mod}_{\langle\rangle}\ g = \textbf{mod}_{\langle\rangle}\ W\ \textbf{in}\ g\ [\![V]\!]_E \\ W &=& \lambda f.\textbf{let mod}_{\langle|\ell\rangle}\ f = f\ \textbf{in handle}\ f\ ()\ \textbf{with}\ [\![H]\!] \\ [\![\text{RHS}]\!] &=& \textbf{handle}\ [\![V\ ()]\!]_{\ell,E}\ \textbf{with}\ [\![H]\!] \end{array}$$

By E-LetMod and E-App in Metn, LHS reduces to

$$\textbf{let mod}_{\langle|\ell\rangle}\ f = [\![V]\!]_E\ \textbf{in handle}\ f\ ()\ \textbf{with}\ [\![H]\!]$$

Suppose $[\![V]\!]_E \rightsquigarrow^* U$. Since V is a function with effect row ℓ, E, by type soundness of Metn and Lemma B.8, we have $U = \textbf{mod}_{\langle|\ell\rangle}\ U'$. Thus, $[\![\text{LHS}]\!]$ further reduces to

$$N = \textbf{handle}\ U'\ ()\ \textbf{with}\ [\![H]\!]$$

By Lemma D.9, we can show that $[\![\text{RHS}]\!] \rightsquigarrow^*_n N$.

Case E-Ret

$$\textbf{handle}\ V\ \textbf{with}\ H \quad\rightsquigarrow\quad V$$

Suppose the term is at effect context E. We have

$$\begin{array}{rcl} [\![\text{LHS}]\!]_E &=& \textbf{handle}_h\ [\![V]\!]_{\ell,E}\ \textbf{with}\ \{\textbf{return}\ x \mapsto \textbf{let mod}_{\langle|\ell\rangle;\mu}\ x = x\ \textbf{in mod}_\nu\ x, \_\} \\ && \text{where}\ \mu = \text{topmod}([\![A]\!]_{\ell,E})\ \text{and}\ \nu = \text{topmod}([\![A]\!]_E) \\ [\![\text{RHS}]\!]_E &=& [\![V]\!]_E \end{array}$$

Our goal follows from Lemma D.9, E-Ret and E-LetMod in Metn. Similar to cases above.

Case E-Op Similar to the E-NOp case.

Case E-Lift By IH and Lemma D.2.

□

The proof of semantics preservation relies on the following lemma.

Lemma D.8 (Translation of Effect Contexts). *For the translation $[\![-]\!]$ from System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ to Metn, we have $[\![\mathcal{E}[M]]\!] = [\![\mathcal{E}]\!][[\![M]\!]]$ for any evaluation context $\mathcal{E}$ and term M.*

Proof. By straightforward case analysis on evaluation contexts of System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$. □

Lemma D.9 (Translation of Values). *For a well-typed value V in System $\mathsf{F}_1^{\epsilon+\mathsf{sn}}$ with $\Gamma \vdash_\varepsilon V : A \mid E$ and $\Gamma \vdash_\varepsilon V : A \mid F$, we have $[\![V]\!]_E = \textbf{mod}_\mu\ W$ and $[\![V]\!]_F = \textbf{mod}_\nu\ W$ for some value W in Metn, where $\mu = \text{topmod}([\![A]\!]_E)$ and $\nu = \text{topmod}([\![A]\!]_F)$.*

Proof. By case analysis on shape of V and check definition of translation. □

D.3 Proofs for Encoding System Ξ

Theorem 6.1 (Type Preservation). *If $\Gamma \vdash M : A$ in System Ξ, then $[\![\Gamma]\!] \vdash [\![M]\!] : [\![A]\!]\ @\ \cdot$ in Metn. Similarly for values and blocks.*

Proof. By induction on typing judgements in System Ξ. We prove a stronger version where the conclusion says that $[\![\Gamma]\!] \vdash [\![M]\!] : [\![A]\!]\ @\ E$ for any well-scoped E in Metn. As a visual aid, for each non-trivial case we repeat its typing rule where we replace the translation premises by the Metn judgement implied by the induction hypothesis and the translation in the conclusion by the Metn judgement we need to prove.

Case () By T-Unit-System Ξ and T-Unit-Metn.

Case $\boxed{x}$ By T-Var-System Ξ and T-Var-Metn. Variables are always accessible after translation as there is no lock in translated contexts at all.

Case $\boxed{f}$ By T-BlockVar-System Ξ and T-Var-Metn. Block variables are always accessible after translation as there is no lock in translated contexts at all.

Case $\boxed{\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}}$

$$\frac{[\![\Gamma]\!], \overline{x : [\![A]\!]}, \overline{f : [\![T]\!]} \vdash [\![M]\!] : [\![B]\!] \mathbin{@} E}{[\![\Gamma]\!] \vdash \lambda \overline{x^{[\![A]\!]}}\ \overline{f^{[\![T]\!]}}.[\![M]\!] : [\![(\overline{A}, \overline{T}) \Rightarrow B]\!] \mathbin{@} E}$$

We have $[\![(\overline{A}, \overline{T}) \Rightarrow B]\!] = \overline{[\![A]\!]} \to \overline{[\![T]\!]} \to [\![B]\!]$. Our goal follows from T-Abs-Metn.

Case $\boxed{\textbf{return}\ V}$ By IH.

Case $\boxed{\textbf{let}\ x = M\ \textbf{in}\ N}$ By IH, T-Let-System Ξ, and T-Let-Metn.

Case $\boxed{\textbf{def}\ f = G\ \textbf{in}\ N}$ By IH, T-Def-System Ξ, and T-Let-Metn.

Case $\boxed{P(\overline{V}, \overline{Q})}$

$$\frac{[\![\Gamma]\!] \vdash [\![P]\!] : [\![(\overline{A}_i, \overline{T}_j) \Rightarrow B]\!] \mathbin{@} E \qquad \overline{[\![\Gamma]\!] \vdash [\![V]\!] : [\![A]\!] \mathbin{@} E} \qquad \overline{[\![\Gamma]\!] \vdash [\![Q]\!] : [\![T]\!] \mathbin{@} E}}{[\![\Gamma]\!] \vdash [\![P]\!]\ \overline{[\![V]\!]}\ \overline{[\![Q]\!]} : [\![B]\!] \mathbin{@} E}$$

We have $[\![(\overline{A}, \overline{T}) \Rightarrow B]\!] = \overline{[\![A]\!]} \to \overline{[\![T]\!]} \to [\![B]\!]$. Our goal follows from T-App-Metn.

Case $\boxed{\textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ \{\mathsf{op}\ p\ r \mapsto N\}}$

$$\frac{\Sigma(\ell) = \{\mathsf{op} : A' \twoheadrightarrow B'\} \qquad [\![\Gamma]\!], f : [\![(A') \Rightarrow B']\!] \vdash [\![M]\!] : [\![A]\!] \mathbin{@} E_1\ (1)\ \text{for any } E_1 \qquad [\![\Gamma]\!], p : [\![A']\!], r : [\![(B') \Rightarrow A]\!] \vdash [\![N]\!] : [\![A]\!] \mathbin{@} E\ (2)}{[\![\Gamma]\!] \vdash \textbf{handle}_a\ (\textbf{let}\ f = \lambda x^{[\![A']\!]}.\textbf{do}_a\ x\ \textbf{in}\ [\![M]\!])\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \mathsf{op}\ p\ r \mapsto [\![N]\!]\} : [\![A]\!] \mathbin{@} E}$$

By (1), $[\![(A') \Rightarrow B']\!] = [\![A']\!] \to [\![B']\!]$, and MT-Extend, we have

$$[\![\Gamma]\!], a :_\sharp \ell, \blacksquare_{\langle a \rangle}, f : [\![A']\!] \to [\![B']\!] \vdash [\![M]\!] : [\![A]\!] \mathbin{@} a, E$$

which further gives

$$[\![\Gamma]\!], a :_\sharp \ell, \blacksquare_{\langle a \rangle} \vdash \textbf{let}\ f = \lambda x^{[\![A']\!]}.\textbf{do}_a\ x\ \textbf{in}\ [\![M]\!] : [\![A]\!] \mathbin{@} a, E\ (3)$$

Observe that $[\![A]\!]$ always has kind Abs. Our goal then follows from (2), $[\![(B') \Rightarrow A]\!] = [\![B']\!] \to [\![A]\!]$, (3), and T-HandleName-Metn.

□

Theorem 6.2 (Semantics Preservation). *If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System Ξ, then $[\![M]\!] \mid [\![\Omega]\!] \rightsquigarrow^* [\![N]\!] \mid [\![\Omega']\!]$ in Metn.*

Proof. By induction on M and case analysis on the next reduction rule. Note that values in System Ξ are translated to value normal forms in Metn.

Case $\boxed{\text{E-Let}}$ Let-binding in Metn is syntactic sugar of lambda application. By E-App-Metn.

Case $\boxed{\text{E-Def}}$ Similar to the above case.

Case $\boxed{\text{E-Call}}$

$$\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}(\overline{V}, \overline{G}) \quad \rightsquigarrow \quad M[\overline{V/x}, \overline{G/f}]$$

We have

$$[\![\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}(\overline{V}, \overline{G})]\!] \quad = \quad (\lambda \overline{x^{[\![A]\!]}}\ \overline{f^{[\![T]\!]}}.[\![M]\!])\ \overline{[\![V]\!]}\ \overline{[\![G]\!]}$$

By multiple usages of E-App-Metn.

Case E-Gen

$$\textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ H \quad {}_{\Omega}{\rightsquigarrow}_{\Omega,h:\ell} \quad \textbf{handle}_h\ M[\textbf{cap}_h/f]\ \textbf{with}\ H$$

where $H = \{\mathtt{op}\ p\ r \mapsto N\}$. We have

$$\begin{array}{rcl} [\![\mathrm{LHS}]\!] & = & \textbf{handle}_a\ (\textbf{let}\ f = \lambda x.\textbf{do}_a\ x\ \textbf{in}\ [\![M]\!])\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \mathtt{op}\ p\ r \mapsto [\![N]\!]\} \\ [\![\mathrm{RHS}]\!] & = & \textbf{handle}_a\ [\![M]\!][[\![\textbf{cap}_h]\!]/f]\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \mathtt{op}\ p\ r \mapsto [\![N]\!]\} \\ [\![\textbf{cap}_h]\!] & = & \lambda x.\textbf{do}_h\ x \end{array}$$

Our goal follows from syntactic sugar and E-App-Metn.

Case E-NRet By E-NRet-Metn.

Case E-NOp

$$\textbf{handle}_h\ \mathcal{E}[\textbf{cap}_h(V)]\ \textbf{with}\ H \quad \rightsquigarrow \quad N[V/p, (\lambda y.\textbf{handle}_h\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r]$$

where $H = \{\mathtt{op}\ p\ r \mapsto N\}$. We have

$$[\![\mathrm{LHS}]\!] \quad = \quad \textbf{handle}_h\ [\![\mathcal{E}[\textbf{cap}_h(V)]]\!]\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \mathtt{op}\ p\ r \mapsto [\![N]\!]\}$$

By Lemma D.10, we have

$$[\![\mathcal{E}[\textbf{cap}_h(V)]]\!] = [\![\mathcal{E}]\!][[\![\textbf{cap}_h(V)]\!]] = [\![\mathcal{E}]\!][(\lambda x.\textbf{do}_h\ x)\ V]$$

Our goal follows from E-App-Metn and E-NOp-Metn

Case E-Lift Follow from IH and Lemma D.10

□

The proof of semantics preservation relies on the following lemma.

Lemma D.10 (Translation of Effect Contexts). *For the translation $[\![-]\!]$ from System Ξ to Metn, we have $[\![\mathcal{E}[M]]\!] = [\![\mathcal{E}]\!][[\![M]\!]]$ for any evaluation context $\mathcal{E}$ and term M.*

Proof. By straightforward case analysis on evaluation contexts of System Ξ. □

D.4 Proof of Encoding System C

Theorem 6.4 (Type Preservation). *If $\Gamma \vdash M : A \mid C$ is a well-formed typing judgement in System C, then $[\![\Gamma]\!]_C \vdash [\![M]\!] : [\![A]\!] @ [\![C]\!]$ in Metn. Similarly for typing judgements of values and blocks.*

Proof. By induction on typing judgements in System C. As a visual aid, for each non-trivial case we repeat its typing rule where we replace the translation premises by the Metn judgement implied by the induction hypothesis and the translation in the conclusion by the Metn judgement we need to prove.

Case () By T-Unit-System C and T-Unit-Metn.

Case x By T-Var-System C and T-Var-Metn. Variables are always accessible after translation as translations of value types always have kind Abs.

Case $\textbf{box}\ G$

$$\frac{[\![\Gamma, \clubsuit]\!]_C \vdash [\![G]\!] : [\![T]\!] @ [\![C]\!]}{[\![\Gamma]\!]. \vdash \textbf{mod}_{[\![C]\!]}\ [\![G]\!] : [\![T\ \textbf{at}\ C]\!] @ \cdot}$$

We have $[\![\Gamma, \clubsuit]\!] = [\![\Gamma]\!].,🔒_{[[\![C]\!]]}$ and $[\![T\ \textbf{at}\ C]\!] = [[\![C]\!]][\![T]\!]$. Our goal follows from T-Mod-Metn.

Case $\boxed{f \text{ transparent}}$

$$\frac{\Gamma \ni f :^{C} T}{[\![\Gamma]\!]_{C} \vdash \hat{f} : [\![T]\!] \ @ \ [\![C]\!]}$$

Suppose $\Gamma = \Gamma_1, f :^{C} T, \Gamma_2$. We have

$$[\![\Gamma_1, f :^{C} T, \Gamma_2]\!]_{C} = [\![\Gamma_1]\!]_{C'}, f : [[\![C]\!]][\![T]\!], \hat{f} :_{[[\![C]\!]]} [\![T]\!], [\![\Gamma_2]\!]_{C}$$

for some C'. Our goal follows from T-Var-Metn and MT-Abs.

Case $\boxed{f \text{ tracked}}$

$$\frac{\Gamma \ni f :^{*} T}{[\![\Gamma]\!]_{f^*} \vdash \hat{f} : [\![T]\!] \ @ \ f^*}$$

By well-formedness of System C judgements, $f :^{*} T$ is inside one and only one pair of delimiter. Suppose $\Gamma = \Gamma_1, \llcorner\Gamma_2, f :^{*} T, \Gamma_3\lrcorner, \Gamma_4$. We have

$$[\![\Gamma_1, \llcorner\Gamma_2, f :^{*} T, \Gamma_3\lrcorner, \Gamma_4]\!]_{f^*} = [\![\Gamma_1]\!]_{\cdot}, \overline{g^*}, \text{🔒}_{\langle| f^* \rangle}, \Gamma_2', f : [f^*][\![T]\!], \hat{f} :_{[f^*]} [\![T]\!], \Gamma_3', [\![\Gamma_4]\!]_{f^*}$$

up to equivalence of contexts where Γ_2' and Γ_3' depends on Γ_2 and Γ_3 and $f^* \in \overline{g^*}$. Our goal follows from T-Var-Metn and MT-Abs.

Case $\boxed{\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}}$

$$\frac{[\![\Gamma, \llcorner\overline{x : A}, \overline{f :^{*} T}\lrcorner]\!]_{[\![C \cup \{\overline{f}\}]\!]} \vdash [\![M]\!] : [\![B]\!] \ @ \ [\![C \cup \{\overline{f}\}]\!]}{[\![\Gamma]\!]_{C} \vdash \Lambda\overline{f^*}.\mathbf{mod}_{\langle|\overline{f^*}\rangle}(\lambda\overline{x^{[\![A]\!]}}\ \overline{f^{[f^*][\![T]\!]}}.\overline{\mathbf{let\ mod}_{[f^*]}\ \hat{f} = f\ \mathbf{in}}\ [\![M]\!]) : [\![(\overline{A}, \overline{f : T}) \Rightarrow B]\!] \ @ \ [\![C]\!]}$$

We have

$$\begin{array}{rcl} [\![\Gamma, \llcorner\overline{x : A}, \overline{f :^{*} T}\lrcorner]\!]_{C \cup \{\overline{f}\}} & = & [\![\Gamma]\!]_{C}, \overline{f^*}, \text{🔒}_{\langle|\overline{f^*}\rangle}, \overline{x : [\![A]\!]}, \overline{f : [f^*][\![T]\!]}, \overline{\hat{f} :_{[f^*]} [\![T]\!]} \\ [\![C \cup \{\overline{f}\}]\!] & = & [\![C]\!], \overline{f^*} \end{array}$$

Our goal follows from T-Abs-Metn, T-Letmod-Metn, and T-Mod-Metn.

Case $\boxed{\mathbf{unbox}\ V}$

$$\frac{[\![\Gamma]\!]_{\cdot} \vdash [\![V]\!] : [\![T\ \mathbf{at}\ C]\!] \ @ \cdot (1)}{[\![\Gamma]\!]_{C} \vdash \mathbf{let\ mod}_{[[\![C]\!]]}\ x = [\![V]\!]\ \mathbf{in}\ x : [\![T]\!] \ @ \ [\![C]\!]}$$

We have $[\![T\ \mathbf{at}\ C]\!] = [[\![C]\!]][\![T]\!]$ By (1) and Lemma D.11, we have

$$[\![\Gamma]\!]_{C} \vdash [\![V]\!] : [[\![C]\!]][\![T]\!] \ @ \ [\![C]\!]$$

Our goal follows from T-Letmod-Metn and MT-Abs ($[[\![C]\!]] \Rightarrow \langle\rangle \ @ \ [\![C]\!]$).

Case $\boxed{\mathbf{let}\ x = M\ \mathbf{in}\ N}$ By IH, Lemma D.11, T-Let-System C, and T-Let-Metn.

Case $\boxed{\mathbf{def}\ f = G\ \mathbf{in}\ M}$

$$\frac{[\![\Gamma, \clubsuit_{C}]\!]_{C'} \vdash [\![G]\!] : [\![T]\!] \ @ \ [\![C']\!]\ (1) \qquad [\![\Gamma, f :^{C'} T]\!]_{C} \vdash [\![M]\!] : [\![A]\!] \ @ \ [\![C]\!]\ (2)}{[\![\Gamma]\!]_{C} \vdash \mathbf{let}\ f = \mathbf{mod}_{[[\![C']\!]]}\ [\![G]\!]\ \mathbf{in\ let\ mod}_{[[\![C']\!]]}\ \hat{f} = f\ \mathbf{in}\ [\![M]\!] : [\![A]\!] \ @ \ [\![C]\!]}$$

We have $[\![\Gamma, \clubsuit_{C}]\!]_{C'} = [\![\Gamma]\!]_{C}, \text{🔒}_{[[\![C']\!]]}$ and $[\![\Gamma, f :^{C'} T]\!]_{C} = [\![\Gamma]\!]_{C}, f : [[\![C']\!]][\![T]\!], \hat{f} :_{[[\![C']\!]]} [\![T]\!]$. Our goal follows from T-Let-Metn, T-Letmod-Metn, and T-Mod-Metn.

Case $P(\overline{V}, \overline{Q})$

$$\frac{\overline{[\![\Gamma]\!]. \vdash [\![V_i]\!] : [\![A_i]\!] \mathbin{@} \cdot \ (1)} \qquad \overline{[\![\Gamma, \clubsuit_{C'}]\!]_{C_j} \vdash [\![Q_j]\!] : [\![T_j]\!] \mathbin{@} [\![C_j]\!] \ (2)} \qquad [\![\Gamma]\!]_C \vdash [\![P]\!] : [\![(\overline{A_i}, \overline{f_j : T_j}) \Rightarrow B]\!] \mathbin{@} [\![C]\!] \qquad C' \coloneqq C \cup \overline{C_j}}{[\![\Gamma]\!]_{C'} \vdash \textbf{let mod}_{\langle \overline{[\![C_j]\!]} \rangle}\ x = [\![P]\!]\ \overline{[\![C_j]\!]}\ \textbf{in}\ x\ \overline{[\![V_i]\!]}\ \overline{(\textbf{mod}_{[[\![C_j]\!]]}\ [\![Q_j]\!])} : [\![B]\!][\overline{[\![C_j]\!]/f_j^*}] \mathbin{@} [\![C']\!]}$$

We have $[\![(\overline{A_i}, \overline{f_j : T_j}) \Rightarrow B]\!] = \forall \overline{f_j^*}.\langle \overline{f_j^*} \rangle (\overline{[\![A_i]\!]} \to \overline{[f_j^*][\![T_j]\!]} \to [\![B]\!])$ and $[\![\Gamma, \clubsuit_{C'}]\!]_{C_j} = [\![\Gamma]\!]_{C'}, \blacksquare_{[[\![C_j]\!]]}$. By (1) and Lemma D.11 we have

$$\overline{[\![\Gamma]\!]_{C'} \vdash [\![V_i]\!] : [\![A_i]\!] \mathbin{@} C'}\ (3)$$

By (2) and T-Mod-Metn we have

$$\overline{[\![\Gamma]\!]_{C'} \vdash \textbf{mod}_{[[\![C_j]\!]]}\ [\![Q_j]\!] : [[\![C_j]\!]][\![T_j]\!] \mathbin{@} [\![C']\!]}\ (4)$$

Also note that translation preserves substitution of capability variables, which gives

$$[\![B]\!][\overline{[\![C_j]\!]/f_j^*}] = [\![B[\overline{C_j/f_j}]]\!]\ (5).$$

Finally our goal follows from (3), (4), (5), T-Letmod-Metn, T-App-Metn, and T-EApp-Metn.

Case $\textbf{return}\ V$ By IH.

Case subtyping of blocks and terms By IH and Lemma D.11.

Case $\textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ \{\texttt{op}\ p\ r \mapsto N\}$

$$\frac{\Sigma(\ell) = \{\texttt{op} : A' \twoheadrightarrow B'\} \qquad [\![\Gamma, \llcorner f :^* (A') \Rightarrow B' \lrcorner]\!]_{C \cup \{f\}} \vdash [\![M]\!] : [\![A]\!] \mathbin{@} [\![C \cup \{f\}]\!]\ (1) \qquad [\![\Gamma, p : A', r :^C (B') \Rightarrow A]\!]_C \vdash [\![N]\!] : [\![A]\!] \mathbin{@} [\![C]\!]\ (2) \qquad M' \coloneqq (\textbf{let}\ f = \textbf{mod}_{[f^*]}\ (\lambda x.\textbf{do}_{f^*}\ x)\ \textbf{in let mod}_{[f^*]}\ \hat{f} = f\ \textbf{in}\ [\![M]\!]) \qquad N_1 \coloneqq \textbf{let mod}_{[[\![C]\!]]}\ x' = x\ \textbf{in}\ x' N_2 \coloneqq \textbf{let mod}_{[[\![C]\!]]}\ \hat{r} = r\ \textbf{in}\ [\![N]\!]}{[\![\Gamma]\!]C \vdash \textbf{handle}^{\spadesuit}_{f^*}\ M'\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \texttt{op}\ p\ r \mapsto N_2\} : [\![A]\!] \mathbin{@} [\![C]\!]}$$

We have

$$\begin{aligned} [\![\Gamma, \llcorner f :^* (A') \Rightarrow B' \lrcorner]\!]_{C \cup \{f\}} &= [\![\Gamma]\!]_C, f^*, \blacksquare_{\langle f^* \rangle}, f : [f^*]([\![A']\!] \to [\![B']\!]), \hat{f} :_{[f^*]} [\![A']\!] \to [\![B']\!] \\ [\![\Gamma, p : A', r :^C (B') \Rightarrow A]\!]_C &= [\![\Gamma]\!]_C, p : [\![A']\!], r : [[\![C]\!]]([\![B']\!] \to [\![A]\!]), \hat{r} :_{[[\![C]\!]]} [\![B']\!] \to [\![A]\!] \end{aligned}$$

By (1) and several typing rules in Metn, we have

$$[\![\Gamma]\!]_C, f^*, \blacksquare_{\langle f^* \rangle} \vdash M' : [\![A]\!] \mid f^*, [\![C]\!]$$

Since $[\![A]\!]$ has kind Abs, by Lemma B.12 and context equivalence, we have

$$[\![\Gamma]\!]_C, f^*, \blacksquare_{[f^*, [\![C]\!]]} \vdash M' : [\![A]\!] \mid f^*, [\![C]\!]\ (3)$$

By (2) and T-Letmod-Metn, we have

$$[\![\Gamma]\!]_C, p : [\![A']\!], r : [[\![C]\!]]([\![B']\!] \to [\![A]\!]) \vdash N_2 : [\![A]\!] \mathbin{@} [\![C]\!]$$

Again by Lemma B.12 and context equivalence, we have

$$[\![\Gamma]\!]_C, \blacksquare_{[[\![C]\!]]}, p : [\![A']\!], r : [[\![C]\!]]([\![B']\!] \to [\![A]\!]) \vdash N_2 : [\![A]\!] \mathbin{@} [\![C]\!]\ (4)$$

We also have

$$[\![\Gamma]\!]_C, \blacksquare_{[[\![C]\!]]}, x : [[\![C]\!]][\![A]\!] \vdash N_1 : [\![A]\!] \mathbin{@} [\![C]\!]\ (4)$$

Our goal follows from (3), (4), (5), T-HandleName$^{\spadesuit}$-Metn. ◻

The proof relies on the following lemma.

Lemma D.11 (Subeffecting). *Given a typing judgement* $\Gamma \vdash M : A \mid C$ *in System C, if* $[\![\Gamma]\!]_C \vdash [\![M]\!] : [\![A]\!] \ @ \ [\![C]\!]$ *and* $C \subseteq C'$ *then* $[\![\Gamma]\!]_{C'} \vdash [\![M]\!] : [\![A]\!] \ @ \ [\![C']\!]$*. Similarly for blocks.*

Proof. By straightforward induction on typing judgements in System C. The most non-trivial case is to show the accessibility of variables. Observe that the change from C to C' only changes locks in the translated context. After translation, all variables in the context either have types of kind Abs or are annotated by an absolute modality. For variables with types of kind Abs, their accessibility is not influenced. For variables annotated with an absolute modality, by MT-Abs, upcasting the effect context can neither influence their accessibility. □

Theorem 6.5 (Semantics Preservation). *If* M *is well-typed and* $M \mid \Omega \leadsto N \mid \Omega'$ *in System C, then* $[\![M]\!] \mid [\![\Omega]\!] \leadsto^* [\![N]\!] \mid [\![\Omega']\!]$ *in* Metn.

Proof. By induction on M and case analysis on the next reduction rule. Values in System C are translated to values in Metn. Not all values in System C are translated to value normal forms in Metn, but we can always further reduce them to value normal forms in Metn. The lemma allows us to have more steps of reduction in Metn.

Case E-Box We have

$$[\![\textbf{unbox}\ (\textbf{box}\ G)]\!] \quad = \quad \textbf{let mod}_{[[\![C]\!]]}\ x = \textbf{mod}_{[[\![C]\!]]}\ [\![G]\!]\ \textbf{in}\ x$$

By E-Letmod-Metn.

Case E-Let We have

$$[\![\textbf{let}\ x = \textbf{return}\ V\ \textbf{in}\ N]\!] \quad = \quad \textbf{let}\ x = [\![V]\!]\ \textbf{in}\ [\![N]\!]$$

LHS reduces to $N[V/x]$ and RHS reduces to $[\![N]\!][[\![V]\!]/x]$. It is easy to show that translation preserves value substitution.

Case E-Call

$$\{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}(\overline{V}, \overline{Q}) \leadsto M[\overline{V/x}, \overline{Q/f}, \overline{C/f}]$$

Let $P = \{(\overline{x : A}, \overline{f : T}) \Rightarrow M\}$, we have

$$\begin{array}{rcl} [\![P]\!] & = & \Lambda \overline{f^*}.\textbf{mod}_{\langle|\overline{f^*}\rangle}(\overline{\lambda x^{[\![A]\!]}}\ \overline{f^{[f^*][\![T]\!]}}.\overline{\textbf{let mod}_{[f^*]}\ \hat{f} = f\ \textbf{in}}\ [\![M]\!]) \\ [\![P(\overline{V_i}, \overline{Q_j})]\!] & = & \textbf{let mod}_{\langle|\overline{[\![C_j]\!]}\rangle}\ x = [\![P]\!]\ \overline{[\![C_j]\!]}\ \textbf{in}\ x\ \overline{[\![V_i]\!]}\ \overline{(\textbf{mod}_{[[\![C_j]\!]]}\ [\![Q_j]\!])} \end{array}$$

Our goal follows from E-Letmod, E-App and E-EApp in Metn, as well as the fact that translation preserves value substitution and type substitution.

Case E-Gen

$$\textbf{handle}\ \{f \Rightarrow M\}\ \textbf{with}\ H \quad {}_{\Omega}\!\!\leadsto_{\Omega,h:\ell} \quad \textbf{handle}_h\ M[\textbf{cap}_h/f, \{h\}/f]\ \textbf{with}\ H$$

where $H = \{\texttt{op\_}\}$. We have

$$\begin{array}{rcl} [\![\textbf{handle}_f\ M\ \textbf{with}\ H]\!] & = & \textbf{handle}^{\spadesuit}_{f^*}\ (\textbf{let}\ f = \textbf{mod}_{[f^*]}\ (\lambda x^{[\![A_{\texttt{op}}]\!]}.\textbf{do}_{f^*}\ x)\ \textbf{in} \\ & & \qquad \textbf{let mod}_{[f^*]}\ \hat{f} = f\ \textbf{in}\ [\![M]\!])\ \textbf{with}\ [\![H]\!] \end{array}$$

Taking the same h as in the rule above, it reduces to

$$\textbf{handle}^{\spadesuit}_h\ [\![M]\!][(\lambda x^{[\![A_{\texttt{op}}]\!]}.\textbf{do}_{f^*}\ x)/\hat{f}, h/f^*]\ \textbf{with}\ [\![H]\!]$$

which is equal to $[\![\textbf{handle}_h\ M[\textbf{cap}_h/f, \{h\}/f]\ \textbf{with}\ H]\!]$.

Case E-NRet By E-NRet$^{\spadesuit}$ and E-Letmod in Metn.

Case E-NOp

$$\textbf{handle}_h\ \mathcal{E}[\textbf{cap}_h(V)]\ \textbf{with}\ H \quad \leadsto \quad N[V/p, (\lambda y.\textbf{handle}_h\ \mathcal{E}[\textbf{return}\ y]\ \textbf{with}\ H)/r]$$

where $H = \{\texttt{op\_}\}$. We have

$$[\![\textbf{handle}_h\ \mathcal{E}[\textbf{cap}_h(V)]\ \textbf{with}\ H]\!] \quad = \quad \textbf{handle}^{\spadesuit}_h\ [\![\mathcal{E}[\textbf{cap}_h(V)]]\!]\ \textbf{with}\ [\![H]\!]$$

By Lemma D.12, we have $[\![\mathcal{E}[\textbf{cap}_h(V)]]\!] = [\![\mathcal{E}]\!][(\lambda x^{[\![A_{\mathsf{op}}]\!]}.\textbf{do}_h\ x)\ [\![V]\!]]$. We can reduce $[\![V]\!]$ to a value normal form in Metn. Our goal follows from E-App and E-NOp$^{\spadesuit}$ in Metn.

Case E-Lift By IH and Lemma D.12.

□

The proof of semantics preservation relies on the following lemma.

Lemma D.12 (Translation of Effect Contexts). *For the translation $[\![-]\!]$ from System C to Metn, we have $[\![\mathcal{E}[M]]\!] = [\![\mathcal{E}]\!][[\![M]\!]]$ for any evaluation context $\mathcal{E}$ and term M.*

Proof. By straightforward case analysis on evaluation contexts of System C. For the case of $\textbf{def}\ f = \mathcal{E}\ \textbf{in}\ N$, note that $\textbf{mod}_\mu\ \mathcal{E}$ is a valid evaluation context in Metn where values can be reduced. □