Modal Effect Types

WENHAO TANG, The University of Edinburgh, UK
LEO WHITE, Jane Street, UK
STEPHEN DOLAN, Jane Street, UK
DANIEL HILLERSTRÖM, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh, UK
ANTON LORENZEN, The University of Edinburgh, UK

We propose a novel type system for effects and handlers using modal types. Conventional effect systems attach effects to function types, which can lead to verbose effect-polymorphic types, especially for higher-order functions. Our modal effect system provides succinct types for higher-order first-class functions without losing modularity and reusability. The core idea is to decouple effects from function types and instead to track effects through *relative* and *absolute* modalities, which represent transformations on the ambient effects provided by the context.

We formalise the idea of modal effect types in a multimodal System F-style core calculus Met with effects and handlers. Met supports modular effectful programming via modalities without relying on effect variables. We encode a practical fragment of a conventional row-based effect system with effect polymorphism, which captures most common use-cases, into Met in order to formally demonstrate the expressive power of modal effect types. To recover the full power of conventional effect systems beyond this fragment, we seamlessly extend Met to Mete with effect variables. We propose a surface language Metel for Mete with a sound and complete type inference algorithm inspired by FreezeML.

1 Introduction

Effect systems allow a typed programming language to express information about what a function does when running, instead of merely providing information about what sort of results it might produce when finished.

Consider the standard map function:

$$\texttt{map} : \forall \alpha\, \beta.\ (\mathsf{List}\ \alpha,\ \alpha \to \beta) \quad\to\quad \mathsf{List}\ \beta$$

In a typical functional programming language, this type is a statement about the values that map accepts and returns (that it takes a list of α and a function from α to β, and returns a list of β), but is silent about which effects may occur during its evaluation.

The effect systems of, say, Koka [31] or Links [21] give the following more precise type to `map`:

$$\texttt{map} : \forall \alpha\, \beta\, \varepsilon.\ (\mathsf{List}\ \alpha,\ \alpha \xrightarrow{\varepsilon} \beta) \quad\xrightarrow{\varepsilon}\quad \mathsf{List}\ \beta$$

This type uses *effect polymorphism*, quantifying over an *effect variable* ε, in order to express that the effects that may be performed by `map (xs, f)` are precisely those that may be performed by calls to `f`. That is, `map` performs no effects of its own, beyond those of the callback `f`.

While this type precisely expresses what we want to say about `map`, the annotation burden of this style of effect system is larger than it might first appear. While only a small amount of text needs to be added to turn the first type into the second, the problem lies in the quantity and location of places where it is needed. Functions like `map` that use no effectful features themselves still need to be annotated, as does essentially every higher-order function.

Authors' Contact Information: Wenhao Tang, wenhao.tang@ed.ac.uk, The University of Edinburgh, UK; Leo White, lwhite@janestreet.com, Jane Street, UK; Stephen Dolan, sdolan@janestreet.com, Jane Street, UK; Daniel Hillerström, daniel.hillerstrom@ed.ac.uk, The University of Edinburgh, UK; Sam Lindley, sam.lindley@ed.ac.uk, The University of Edinburgh, UK; Anton Lorenzen, anton.lorenzen@ed.ac.uk, The University of Edinburgh, UK.

This is a mild burden to the authors of new code, but a significant obstacle to extending an existing language with effectful features: signatures of much existing library code must be rewritten to support effect polymorphism, even in old libraries that do not use the new features at all. The need to update such libraries makes it difficult to add an effect system to a language in a backwards-compatible way. Instead, our goal is to support precise tracking of effects, without polluting the type of non-effectful functions like `map`.

1.1 Annotating Effect Transitions

Important steps towards this goal were taken by the languages Frank [12, 33] and Effekt [7, 8], both of which use the original type for `map`, allowing use of effectful callbacks without requiring effect polymorphism annotations in the type of `map`.

The key idea enabling use of these simpler types in both languages is the *ambient effect context*. All functions are typed assuming a certain set of possible effects, and annotations are required at *transitions* between different effect contexts. Since the argument to `map` uses the same effects as `map` itself, there is no transition and hence no annotation is required.

Both Frank and Effekt achieve this by special typing support for computations that appear in argument position. In Frank, an *adjustment* is attached to each argument, specifying how the ambient effects of the called function relate to the effects provided to its argument. In Effekt, arrow types appearing in argument position are parsed as *blocks*, second-class function types that inherit ambient effects. While different, both of these mechanisms give elegant typings of handlers, but become more complicated with more advanced uses of arrow types, such as when closures are captured and/or inserted into data structures. The essential reason is that both argument types decorated with adjustments in Frank and block types in Effekt are second-class, and they use different methods to bypass this restriction. In Frank, first-class higher-order functions rely on some syntactic sugar to insert effect variables. In Effekt, first-class use of closures was initially disallowed entirely, and now supported with extra annotations on captured capabilities.

We build on this insight that types should mark transitions between effect contexts, rather than repeating the full effect context. We extend the idea by decoupling it from function arguments, and making effect transitions available as a true type constructor, usable in any context.

We work in the framework of modal types, following multimodal type theory (MTT) [17, 18], where each possible effect context is a *mode*, and each possible transition between effect contexts is a *modality*. We support both *relative* modalities, which describe a local change to an effect context such as entering a new handler (similar to Frank's adjustments), and *absolute* modalities, which describe the full effect context (similar to Frank's abilities).

Unlike Frank and Effekt, our modalities are not tied to function arrows, and can be applied anywhere, even nested inside complex data structures. Our modal effect system also works smoothly with pure first-class higher-order functions; they all type check without requiring hidden effect variables or extra annotations, and can be applied to effectful arguments.

1.2 Contributions

The main contributions of this paper are:

- We give high-level overview of the main ideas through a series of examples that illustrate the verbosity of conventional effect systems, how they can be simplified by the absolute and relative modalities, and how modal effect types enable us to write expressive effectful programs in a sound and succinct way (Section 2).
- We introduce Met, a multimode and multimodal core calculus with effect handlers and modal effect types (Section 3). We prove its type soundness and effect safety.

- We extend Met with data types and richer kinds of handlers. We further extend it to Mete with effect variables to recover the full power of traditional effect systems (Section 4).
- To illustrate the expressiveness of modal effect types, we formally prove that a practical fragment of traditional row-based effect systems is encodable in Met. (Section 5).
- To demonstrate the feasibility of programming with modal effect types, we introduce Metel, a surface language based on Mete with a sound and complete type inference algorithm which can automatically unbox modalities for variables (Section 6).
- We discuss the relationships of modal effect types with the Frank language, capability-based effect systems, and multimodal type theory. (Section 7).

Section 7 also discusses other related work and Section 8 concludes.

2 Programming with Modal Effect Types

In this section we give a series of examples to illustrate the main ideas of modal effect types. We demonstrate how modal effect types allow composition of higher-order functions and effect handlers in a modular manner with succinct types. The key enablers for this programming style are the relative and absolute modalities, which provide the programmer with a novel typing mechanism to manage effect contexts. The examples are written in Metel, whose core calculus we introduce in Sections 3 and 4, and whose design we discuss further in Section 6. Metel is a typed functional language equipped with a modal effect type system for programming with effects and handlers.

2.1 Seamless First-Class Higher-Order Functions

First-class higher-order functions are a staple ingredient of functional programming. As we explained in the introduction, extending an existing language with traditional row-based effect typing requires adding effect variables to the type signatures of pure higher-order functions. Modal effect types offer a backwards-compatible alternative, requiring no extra effect variables for types of higher-order functions that do not themselves use effects. For instance, in Metel we write the standard type for the curried implementation of `map`.

```
map : ∀ a b . (a → b) → List a → List b
map f nil         = nil
map f (cons x xs) = cons (f x) xs
```

This is a genuine first-class higher-order function which can be partially applied, passed around, stored in data types, and so forth. Metel, unlike Frank, does not implicitly insert any effect variables in the type signature of `map`. We may still apply `map` to any function that performs any effects from the effect context in which `map` is invoked.

The effect context for global definitions is empty (though in a practical programming language it could include some built-in effects). Metel captures this fact by implicitly *boxing* the type signature of each global definition in the *empty absolute* modality `[]`. The elaborated type signature for `map` is:

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

Since `map` itself is pure, the default empty effect context suffices. As we shall see shortly, `map` can be invoked under any effect context by way of unboxing and sub-effecting.

In general an *absolute* modality has the form `[E]`, which specifies that the effect context is `E`. In Section 2.2 we further consider absolute modalities. In Section 2.3 we also discuss *relative modalities*.

Metel automatically unboxes variables like `map` when they are used, meaning that programmers may omit empty absolute modalities from the signatures of pure functions. Consequently, modal effect types can be retrofitted onto an existing programming language, while preserving the signatures of pure functions.

2.2 Absolute Modalities Define Full Effect Contexts

In METEL, modalities are used to change the effect context. An absolute modality is absolute in the sense that it specifies an entire new effect context to replace the current one with. As an example consider an implementation of a yield-style generator [24] using an effectful operation `yield : Int ⇒ 1` which takes an integer and returns a unit.

```
gen : [yield](List Int → 1)
gen xs = map (fun x → do yield x) xs; ()
```

The `gen` function implements an integer generator which reflects a given list as a computation by yielding each element of the list. In the function body we apply `map` to an effectful function that invokes the operation `gen` via the keyword **do**. The absolute modality `[yield]` specifies the effect context required to run `gen` (it must be able to perform `yield`). The type signature tells METEL to implicitly box `gen` with the modality `[yield]`.

The use of `map` is implicitly unboxed enabling implicit sub-effecting to coerce the empty effects of its definition to the `yield` effect of its invocation site. In general unboxing and sub-effecting allow functions to be used in a larger effect context than the one in which it was defined, for instance:

```
gen' : [yield, foo, bar, baz](List Int → 1)
gen' xs = gen xs
```

In a traditional row-based effect system, the effect context is changed by way of effect polymorphism, and we would give the following type signature to `gen`.

$$\texttt{gen : } \forall \texttt{ e . List Int} \xrightarrow{\texttt{yield, e}} \texttt{1}$$

2.3 Relative Modalities Define Effect Transformations

Henceforth, we will frequently refer to the effect context in which a given term or variable is used as the *ambient* effect context. Pure higher-order functions like `map` are local in the sense that they do not change the ambient effect context. Modal effect types come into their own when the programming language has facilities that act on effect contexts, such as handlers and masks [4].

For example, we can implement an effect handler for `yield` that reifies a given computation into a list by interpreting each `yield` as consing the element onto the list.

```
asList m = handle m () with
  return ()  ⇒ nil
  yield  x r ⇒ cons x (r ())
```

The body of `asList` applies the function `m` inside a handler. In the handler we have to consider two things: 1) what happens when `m` returns; and 2) what happens when `m` performs `yield`. In the first case, we map the unit value `()` to the empty list `nil`. In the second case, we cons the yielded element `x` onto the list returned by the application of `r`. Here `r` is bound to the continuation of performing `yield` inside `m`. Its argument type is given by the return type of the operation being handled (unit in the case of `yield`) and its return type is given by the return type of the handler, i.e. `r : 1 → List Int`. The continuation `r` reinstalls the handler such that residual invocations of `yield` are handled in the same manner. This style is known as deep handlers in the literature [26].

We can annotate this function with an absolute modality.

```
asList : [yield](1 → 1) → List Int
```

Often this type is not the one we want as it means that the function parameter is *only* allowed to use the `yield` operation. The absolute modality fixes the effect context, preventing the function argument from using other effects. Sometimes it may be desirable to do so, however, more often we want to be able to handle the specific `yield` operation of an arbitrary effectful function that performs multiple different operations. To this end, we can instead use *relative modalities*, which enable us to describe the relative change that the handler makes to the ambient effect context, e.g.

```
asList : <yield>(1 → 1) → List Int
```

The relative modality <yield> is part of the parameter type and indicates that the effect context for the term inside is derived by extending the ambient effect context with `yield`. Thus, when `m` is automatically unboxed and used in `asList`, the effect context required by `m` matches the effect context in the scope of the `yield` handler. The relative modality here captures the fact that `asList` handles the `yield` effect when invoking `m`, but also allows `m` to perform other effects (which will be forwarded to an outer handler). In a traditional row-based effect system, we would give the following type signature to `asList`.

$$\texttt{asList : } \forall\, \texttt{e}\ .\ (\texttt{1} \xrightarrow{\texttt{yield, e}} \texttt{1}) \xrightarrow{\texttt{e}} \texttt{List Int}$$

To run `asList`, we must box its argument with <yield>.

```
> asList <yield>(fun () → gen [3,1,4,1,5,9])
# [3,1,4,1,5,9] : List Int
```

The syntax <yield>(...) boxes the term inside with the relative modality <yield>. It extends the ambient effect context with `yield` for the program inside, allowing the effectful function `gen` to be used. Note that both `asList` and `gen` are automatically unboxed as usual.

In general, relative modalities have the form <L|D>, where `L` is a row of operations that is masked from the ambient effect context, and `D` is a row of operations that extends the ambient effect context. We write <D> as shorthand for <|D>. We expand more on masking in Section 2.8.

2.4 Effect Safety and No Accidental Handling

In `asList`, the parameter `m` is used under the same effect context in which it is introduced. In general, METEL restricts the use of any variable whose value depends on the effect context at the time of its binding occurrence (e.g., a function *not* boxed by an absolute modality). Such a variable may only be used under an effect context compatible with one at the binding occurrence.

This property is important for guaranteeing effect safety, i.e., that all effects are handled. For instance, the following program is ill-typed

```
asListWrong : <yield>(1 → 1) → List Int   # ill-typed
asListWrong m = m (); [37,42]
```

because `m` requires an effect context that permits the `yield` effect and yet the effect context of the definition of `asListWrong` is empty.

This property also forces effect types to reflect where effects are handled, thus preventing the accidental handling problem [54]. For instance, we cannot give the following type to `asList`.

```
asList : (1 → 1) → List Int        # ill-typed
asList m = handle m () with ...     # same as in Section 2.3
```

The problem is that the handler extends the effect context with `yield`, and yet `m` is introduced before this extension. As a result, the value bound to `m` might use a `yield` operation from its effect context provided by a different handler instead of `asList` (we follow Leijen [30] to allow duplicated labels). If this type was allowed, `asList` would handle this `yield` unexpectedly

2.5 Composing Handlers

We can compose handlers modularly in Metel. For example, consider two integer state operations `get : 1 ⇒ Int` and `put : Int ⇒ 1`. We can implement a standard state handler by interpreting a computation over state operations as a state-passing function.

```
state : ∀ [a] . <get, put>(1 → a) → Int → (a, Int)
state m = handle m () with
  return x     ⇒ fun s → (x, s)
  get    () r  ⇒ fun s → r s s
  put    s' r  ⇒ fun s → r () s'
```

The attentive reader may have observed that the type variable a is declared inside a box. We shall discuss the reason for this syntax in Section 2.7.

With state operations, we can write a generator which yields the prefix sum of a list.

```
prefixSum : [yield, get, put](List Int → 1)
prefixSum xs = map (fun x → do put (do get + x); do yield (do get)) xs; ()
```

The absolute modality `[yield, get, put]` aggregates all effects performed in the definition.

We can now handle `prefixSum` by composing two handlers in sequence.

```
> asList <yield>(fun () →
    state <get,put>(fun () → prefixSum [3,1,4,1,5,9]) 0; ())
# [3,4,8,9,14,23] : List Int
```

Following the pattern we saw previously for handlers, we explicitly box the arguments with relative modalities in order to extend the effect context with the handled effects. Observe how we use `state` modularly: its type signature mentions only `get` and `put` even though it is applied to a computation which invokes `prefixSum`, which also uses `yield`.

2.6 Effect Transformations

We give an example similar to the one from Section 2.2 of Brachthäuser et al. [7], in which an effect handler is used to transform a computation by reperforming the handled effect. The following handler transforms all generated integers with a function and then re-generates them.

```
regen : [yield]((Int → Int) → <yield>(1 → 1) → 1)
regen f m = handle m () with
  return ()  ⇒ ()
  yield  s r ⇒ do yield (f s); r ()
```

The intuition behind the type signature for `regen` is as follows: we handle the `yield` operation for the second argument (as indicated by `<yield>`), and the whole function also uses `yield` (as indicated by `[yield]`). This type is similar to those given by Effekt and Frank modulo syntactic differences. In contrast, Koka infers the following more verbose type.

```
∀<e>. (f : (int) → <yield|e> int) → ((g : () → <yield,yield|e> ()) → <yield|e> ())
```

2.7 Escaping Handlers and Absolute Kinds

One of the fundamental ideas of modal effect types is to track transformations on effect contexts, rather than just full effect contexts. As a consequence, when a value leaves the scope of a handler, its ambient effect context changes, and we must keep track of this change. For instance, the most general type for the state handler define in Section 2.5 is as follows.

```
state : ∀ a . <get, put>(1 → a) → Int → (<get, put>a, Int)
```

The return value has type `<get, put>a` instead of just `a` because it comes from an effect context which extends the ambient one with `get` and `put`. However, this handler does not handle operations in return values. We must guarantee that the effect context in which the return value is used provides operations `get` and `put`.

As a special case, values boxed with absolute modalities do not depend on the current effect context, and thus can flexibly leave the scope of handlers. We can also give the following specialised type for the state handler where `a` is always boxed with the empty absolute modality `[]`.

```
state : ∀ a . <get, put>(1 → []a) → Int → (a, Int)
```

Because of automatic unboxing, this is a valid type for `state` without changing its definition.

In practice, it is useful to allow a value of base type or an algebraic data type that contains only base types or a type boxed with absolute modalities to appear anywhere, including escaping escaping the scope of a handler. Such values can never depend on the effect context in which they are used. We introduce a kind system to Metel in which the `Abs` kind classifies only such absolute data types, whereas the `Any` kind classifies all data types. Subkinding allows any `Abs` type to be treated as an `Any` type. By default all type variables have kind `Any`. Recall the type for the state handler in Section 2.5.

```
state : ∀ [a] . <get, put>(1 → a) → Int → (a, Int)
```

The syntax `∀ [a]` ascribes kind `Abs` to `a`, and thus allows values of type `a` to leave the scope of the handler. In practice it is usually desirable for return types of computations inside handler scopes to have absolute kind, so that they can escape, but if a handler is used locally then this need not always be the case.

2.8 Masking

Handlers extend the ambient effect context with those effects that they handle. Dually, masks remove the effects they mask from the ambient effect context [4]. Masking is a useful device to conceal private implementation details [35].

We give an example of implementing `find` with `yield` to show how masks work in Metel.

```
findWrong : (Int → Bool) → List Int → Maybe Int   # ill-typed
findWrong p xs = handle (map (fun x → if p x then do yield x else ()) xs) with
  return _    ⇒ nothing
  yield  x _ ⇒ just x
```

This program is ill-typed as the predicate `p` is bound under the ambient effect context but used in the scope of a handler.

To fix it, we can mask `yield` and rewrite the handled expression as follows.

```
(map (fun x → if maska<yield>(p x) then do yield x else ()) xs)
```

The `maska<yield>(...)` form masks the operation `yield` from the ambient effect context. Now the effect context for `p` is equivalent to the ambient one, since the transformations of extending with `yield` followed by masking with `yield` cancel each other.

We use the keyword **maska** rather than simply **mask** because leaving the scope of masks also changes the effect context. The situation is similar to the one we encountered in Section 2.7 where we were concerned with allowing some values to escape the scope of a handler. The term `mask <yield>(p x)` yields a value of type `<yield|>Bool` instead of `Bool`, where `<yield|>` is a relative modality masking `yield` from the ambient effect context. Even though `p x` returns a boolean value here, Metel cannot automatically unbox the value in order to ensure completeness of type inference. The **maska** form enables the special case of yielding values of absolute kind such as booleans.

2.9 Cooperative Concurrency

We now consider an example of a richer effect handler which implements cooperative concurrency with a UNIX-style fork operation [23, 44]. We simplify the signature of fork ever-so-slightly such that it returns a boolean to indicate whether the parent or child process should be evaluated, i.e. `ufork : 1 ⇒ Bool`. In addition, we require an operation `suspend : 1 ⇒ 1` that suspends the current process such that another process can run.

We model a process as a data type that embeds a continuation function which takes the list of suspended processes as input and returns unit. In addition, we define auxiliary functions `push` for appending a process onto a queue and `next` which pops and runs the next process.

```
data Proc = proc (List Proc → ())

push : ∀ a . a → List a → List a
push x xs = xs ++ cons x nil
```

```
next : List Proc → ()
next q = case q of
  nil                → ()
  cons (proc p) ps → p ps
```

The following handler implements a scheduler by using the state-passing technique to thread the process queue through the handler activations.

```
schedule : <ufork, suspend>(1 → 1) → List Proc → 1
schedule m = handle m () with
  return  ()   ⇒ fun q → next q
  suspend () r ⇒ fun q → next (push (proc (r ())) q)
  ufork   () r ⇒ fun q → r true (push (proc (r false)) q)
```

The **return**-case is triggered when a process finishes, thus we run the next available process. In the `suspend`-case we enqueue the continuation, before we run the next available process. Finally, in the `ufork`-case we implement the process duplication behaviour of UNIX fork by first enqueuing one application of the continuation, and then immediately applying the continuation to resume one of the process copies. Note that in the above code we seamlessly store effectful functions in data types, similar to how one would do it in a functional language without an effect type system.

2.10 Modal Types with Effect Variables

There is no free lunch; modal effect types cannot offer everything that row-based effect types provide without some cost. An important use case that requires explicit effect variables is implementing higher-order operations [49, 50, 52].

In Metel, we restrict argument and result types of operations to be absolute for effect safety. This is because effect handlers provide non-trivial manipulation of control-flow, which allows operation arguments and results to seamlessly move between different effect contexts. For example, suppose we were to allow an operation `leak : (1 → Int) ⇒ 1`, we could write the following unsafe program.

```
handle (handle (do leak (fun _ → do yield 42)) { yield → ... }) { leak p → p }
```

The `yield` operation is used under an effect context containing `yield`, which is added by the `yield` handler. However, the handler of `leak` binds the closure `(fun _ → do yield 42)` to `p` and leaks it. Requiring `leak` to have the signature `[yield](1 → Int) ⇒ 1` fixes the leakage problem as it specifies the full effect context for the argument of `leak`.

Using such an absolute modality in this fashion impedes modularity. As another example, consider a higher-order fork operation which takes a thunk as an argument. We may specify the full effect context for the child process, such as the following signature.

```
effect fork : [fork, suspend](1 → 1) ⇒ 1
```

However, if we want to support processes that use other effects as well then either we have to change the signature or we need to extend our modal type system with effect variables. With an effect variable `e`, we can define the following parameterised signature.

```
effect fork e : [fork e, suspend, e](1 → 1) ⇒ 1
```

Fortunately, as we demonstrate in Section 4.5, modal effect types are compatible with explicit effect variables, and indeed Metel supports them.

2.11 Modalities Anywhere

Unlike adjustments in Frank and block annotations in Effekt mentioned in Section 1.1, modal types are first-class types just like data types and can appear anywhere. For instance, we can put two functions with modal types in a pair and handle them separately.

```
handleTwo : (<yield>(1 → 1), <yield>(1 → 1)) → (List Int, List Int)
handleTwo (x, y) = (asList ~x, asList ~y)
```

The syntax `~x` freezes the variable `x`, and prevents it from being automatically unboxed, following FreezeML [15]. Thus we can directly apply `asList` to it without re-boxing.

The type inference algorithm of Metel also supports instantiation of type variables with modal types, by analogy to impredicativity of first-class polymorphism. As a consequence, Metel enjoys various stability properties. For instance, given the standard identity function `id` and application function `app`, type inference of Metel is stable under replacing any term `t` with `id t` and any application `t1 t2` with `app t1 t2`.

3 A Multimodal Core Calculus with Effect Handlers

In this section, we introduce Met, a System F-style call-by-value core calculus with effect handlers and modal effect types. We present its static and dynamic semantics as well as its meta theory. We defer extensions including data types, alternative forms of handlers, and explicit effect variables to Section 4. Met is closely related to multimodal type theory (MTT) [17, 18], especially its simply-typed fragment [29]. We present Met without assuming any background on MTT, and discuss the relationships in Section 7.3.

3.1 Syntax

The syntax of Met is as follows.

$$
\begin{array}{lrcl}
\text{Types} & A, B & ::= & \alpha \mid \forall \alpha^K.A \\
 & & \mid & A \to B \mid \mu A \\
\text{Masks} & L & ::= & \cdot \mid \ell, L \\
\text{Extensions} & D & ::= & \cdot \mid \ell : P, D \\
\text{Effect Contexts} & E, F & ::= & \cdot \mid \ell : P, E \\
\text{Signatures} & P & ::= & A \twoheadrightarrow B \mid - \\
\text{Modalities} & \mu & ::= & [E] \mid \langle L | D \rangle \\
\text{Kinds} & K & ::= & \mathsf{Abs} \mid \mathsf{Any}
\end{array}
$$

$$
\begin{array}{lrcl}
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, \alpha : K \mid \Gamma, x :_{\mu_F} A \mid \Gamma, \text{🔒}_{\mu_F} \\
\text{Terms} & M, N & ::= & x \mid \lambda x^A.M \mid M\, N \mid \Lambda \alpha^K.V \mid M\, A \\
 & & \mid & \mathbf{mod}_\mu\, V \mid \mathbf{let}_\nu\, \mathbf{mod}_\mu\, x = V\, \mathbf{in}\, M \\
 & & \mid & \mathbf{do}\; \ell\; M \mid \mathbf{mask}_L\, M \\
 & & \mid & \mathbf{handle}\, M\, \mathbf{with}\, H \\
\text{Values} & V, W & ::= & x \mid \lambda x^A.M \mid \Lambda \alpha^K.V \mid V\, A \mid \mathbf{mod}_\mu\, V \\
\text{Handlers} & H & ::= & \{\mathbf{return}\; x \mapsto M\} \mid \{\ell\; p\; r \mapsto M\} \uplus H
\end{array}
$$

$\boxed{D + E}$ $\boxed{E - L}$ $\boxed{L \bowtie D}$

$$
\begin{aligned}
D + E &= D, E \\
\cdot - L &= \cdot \\
(\ell : P, E) - L &= \begin{cases} E - L' & \text{if } L \equiv \ell, L' \\ \ell : P, (E - L) & \text{otherwise} \end{cases}
\end{aligned}
\qquad
\begin{aligned}
\cdot \bowtie D &= (\cdot, D) \\
(\ell, L) \bowtie D &= \begin{cases} (L', D'') & \text{if } D' \equiv \ell : P, D'' \\ ((\ell, L'), D') & \text{otherwise} \end{cases} \\
&\quad \text{where } (L', D') = L \bowtie D
\end{aligned}
$$

Fig. 1. Operations on Effect Contexts for Met.

Met extends a System F-style calculus with standard constructs for effects and handlers as well as the main novelty of this work: modal effect types. We highlight the novel features in grey.

3.2 Effect Contexts as Modes

The modes of Met are effect contexts E, which are scoped rows of effect labels [30]. Each label denotes an effectful operation. An effect may contain the same label multiple times. Each label has a signature. A signature can be an arrow of the form $A \twoheadrightarrow B$, which takes an argument of type A and returns a value of type B, or absent $-$ (similar to presence types [43]), which indicates that the operation of this label cannot be invoked.

Following Rémy [43] and Leijen [30], we identify effects up to reordering of distinct labels, and allow absent labels to be freely added to or removed from the right of effect contexts. For instance, $\ell : P, \ell' : -$ is equivalent to $\ell : P$. We can think of an effect context as denoting a map from labels to infinite sequences of signatures where a cofinite tail of each sequence contains only $-$.

Extensions D and masks L are used respectively to extend effect contexts with more labels or removes some labels from them. Extensions are like effect contexts except that we do not ignore labels with absent signatures in their equivalence relation, so $\ell : P, \ell' : -$ and $\ell : P$ are distinct.

We define a sub-effecting relation on effect contexts: $E \leqslant E'$ if we can replace the absent signatures in E with proper signatures to obtain E'. We also have a subtyping relation on extensions $D \leqslant D'$. Different from sub-effecting, it requires D and D' to contain the same row of labels, but allows absent signatures in D to be replaced by other signatures in D'. We give the full rules for type equivalence and sub-effecting in Appendix A.1.

Masks L are simply multi-sets of labels without signatures, as we do not require signatures when masking labels from effect contexts. The actions of extending $D + E$ and masking $E - L$ are defined in Figure 1. We write $L \bowtie D = (L', D')$ for the difference between L and D. The L' are those labels in L not appearing in the domain of D, and the D' are those labels in D not appearing in L.

3.3 Modalities Manipulating Effect Contexts

In conventional row-based effect systems, such as Koka or Links, an effect annotation on a function type specifies all of the effects that the function may perform when it is invoked. In Met, effect annotations only specify effects relative to the ambient effect context, as functions may also use any operations from the ambient effect context. Effect annotations are given via *modalities*, which construct a new effect context relative to an ambient effect context as follows.

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

The absolute modality $[E]$ replaces the ambient effect context F with E. This is similar to how effect annotations on functions in row-based effect systems work. Intuitively, we may think of the type $[E](A \to B)$ as corresponding roughly to the type $A \to^{E} B$ in traditional effect type systems. The relative modality $\langle L|D \rangle$ is the key feature that makes effectful programming without effect variables viable in MET. It specifies the a transformation on the ambient effect context. It masks the labels L in F before extending the resulting context with D. We call $\langle\!|D\rangle$ an extension modality, $\langle L|\!\rangle$ a mask modality, and $\langle|\rangle$ the identity modality. We write $\mathbb{1}$ for the identity modality.

Modalities are monotone total functions on effect contexts. If $E \leqslant F$, we have $\mu(E) \leqslant \mu(F)$.

We write μ_F for the pair of μ and F where F is the effect context that μ acts on. We refer to such a pair as an indexed modality. We write $\mu_F : E \to F$ if $\mu(F) = E$. (The arrow goes from E to F instead of the other direction to keep closer to MTT [17, 18]. For readers familiar with MTT, indexed modalities μ_F correspond to the notion of modalities in MTT as they are concrete morphisms between modes and our modalities μ actually correspond to indexed families of modalities in MTT.)

Modality Composition. We can compose the actions of modalities in the intuitive way.

$$
\begin{array}{rcl l}
\mu & \circ & [E] & = & [E] \\
[E] & \circ & \langle L|D\rangle & = & [D + (E - L)] \\
\langle L_1|D_1\rangle & \circ & \langle L_2|D_2\rangle & = & \langle L_1 + L|D_2 + D\rangle \quad \text{where } (L, D) = L_2 \bowtie D_1
\end{array}
$$

To keep close to MTT, our composition reads from left to right. First, an absolute modality completely specifies the new effect context, thus shadowing any other modality μ. Second, replacing the effect context with E and then masking L and extending with D is equivalent to just replacing with $D + (E - L)$. Third, sequential masking and extending can be combined into one by using $L_2 \bowtie D_1$ to cancel the overlapping part of L_2 and D_1. For instance, we have $\langle\!|\ell : P\rangle \circ \langle \ell|\!\rangle = \langle|\rangle$.

Composition is well-defined since composing followed by applying is equivalent to sequentially applying $(\mu \circ \nu)(E) = \nu(\mu(E))$. We also have associativity $(\mu \circ \nu) \circ \xi = \mu \circ (\nu \circ \xi)$ and identity $\mathbb{1}$.

The definition of composition naturally generalises to indexed modalities μ_F. We can compose $\mu_F : E \to F$ and $\nu_E : E' \to E$ to get $\mu_F \circ \nu_E : E' \to F$ which is defined as $(\mu \circ \nu)_F$.

Modality Transformations. Just as modalities allow us to manipulate effect contexts, we need transformations that allow us to change modalities[1].

We write $\mu_F \Rightarrow \nu_F$ for a transformation between indexed modalities $\mu_F : E \to F$ and $\nu_F : E' \to F$. Intuitively, such a transformation describes how under ambient effect context F, the action of μ can be replaced by the action of ν. In particular, if we have a variable boxed by μ under the effect context F, we can use it under a new effect context derived by applying ν to F.

What properties do we expect from $\mu_F \Rightarrow \nu_F$? To guarantee effect safety, the new effect context E given by applying ν should be larger than the E' given by applying μ. To avoid accidental handling, when μ is relative (which means the variable depends on the ambient effect context), the new effect context E' should not accidentally capture more effects than those specified by μ and the ambient effect context. Moreover, we want the transformation to be stable under sub-effecting. We formally define $\mu_F \Rightarrow \nu_F$ by the transitive closure of the following three rules.

$$
\frac{\mu_F : E' \to F \qquad E \leqslant E'}{[E]_F \Rightarrow \mu_F}\;\textsc{MT-Abs}
\qquad
\frac{D \leqslant D'}{\langle L|D\rangle_F \Rightarrow \langle L|D'\rangle_F}\;\textsc{MT-Upcast}
\qquad
\frac{(F - L) \equiv \ell : P, E}{\langle \ell, L|D, \ell : P\rangle_F \Leftrightarrow \langle L|D\rangle_F}\;\textsc{MT-Expand}
$$

MT-Abs allows us to transform an absolute modality to any other modality as long as no effect leaks. MT-Upcast allow us to upcast a label with an absent signature in D to an arbitrary signature, since the corresponding operation is unused. Recall that the subtyping relation between extensions only upcasts signatures. MT-Expand is bidirectional. It allows us to simultaneously mask and extend some operations given that these operations exist in the ambient effect context F.

[1] The interested reader may wonder if we would need yet another notion of transforming a modality transformation, but thankfully this is not necessary: there is only one modality transformation between any two modalities

Let us give some examples here. First, $[]_E \Rightarrow \mu_E$ always holds, consistent with the intuition that pure values can be used anywhere. Second, $\langle\!|\ell : -\rangle_E \Rightarrow \langle\!|\ell : P\rangle_E$ always holds. Third, we have $\langle \ell | \ell : P\rangle_{\ell:P,E} \Leftrightarrow \langle | \rangle_{\ell:P,E}$ in both directions. Last, $\langle\!|\ell : P\rangle_E \Rightarrow \langle\!|\ell : P, \ell' : P'\rangle_E$ does not hold for any E, avoiding accidental handling.

The following lemma shows that the syntactic definition of transformation matches the semantics of our intuition. The proof is in Appendix A.4.

Lemma 3.1 (Semantics of modality transformation). *We have $\mu_F \Rightarrow \nu_F$ if and only if $\mu(F') \leqslant \nu(F')$ for all F' with $F \leqslant F'$.*

Attentive readers may have observed that this lemma characterises the essence of effect safety, but does not mention accidental handling explicitly. Actually, since Met allows same labels to have different signatures, effect safety implies that there is no accidental handling. For instance, $\langle | \rangle_F \Rightarrow \langle\!|\ell : P\rangle_F$ violates Lemma 3.1 since $F, \ell : P' \nleqslant \ell : P, F, \ell : P'$ when $P \nleqslant P'$.

3.4 Kinds and Contexts

$$\boxed{\Gamma \vdash A : K} \quad \boxed{\Gamma \vdash P} \quad \boxed{\Gamma \vdash (\mu, A) \Rightarrow \nu \mathbin{@} F}$$

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

$$\frac{\Gamma \vdash A : \mathsf{Any} \qquad \Gamma \vdash B : \mathsf{Any}}{\Gamma \vdash A \to B : \mathsf{Any}} \qquad \frac{\Gamma \vdash A : \mathsf{Abs} \qquad \Gamma \vdash B : \mathsf{Abs}}{\Gamma \vdash A \twoheadrightarrow B} \qquad \frac{\Gamma \vdash A : \mathsf{Abs}}{\Gamma \vdash (\mu, A) \Rightarrow \nu \mathbin{@} F} \qquad \frac{\mu_F \Rightarrow \nu_F}{\Gamma \vdash (\mu, A) \Rightarrow \nu \mathbin{@} F}$$

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

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

Fig. 2. Selected kinding, well-formedness, and auxiliary rules for Met.

As illustrated in Section 2.7, we have two kinds Abs and Any. The Abs kind is a sub-kind of the kind of all types Any, and denotes types of values that are guaranteed not to use operations from the ambient effect context.

We show the kinding and well-formedness rules for types and signatures in Figure 2, relying on the well-formedness of modalities and effect contexts, which is standard and defined in Appendix A.1. Function arrows have kind Any due to the possibility of using operations from the ambient effect context. Boxing a type by the absolute modality yields an absolute type as it cannot depend on the ambient effect context.

A type at kind Abs may still contain an effectful computation, as long as it is contained within an absolute modality. We restrict the kind of the argument and return value of effects to be Abs in order to prevent effect leakage as discussed in Section 2.10.

Contexts are ordered. We define the relation $\Gamma \mathbin{@} E$ that context Γ is well-formed at effect context E in Figure 2. Each term variable binding $x :_{\mu_F} A$ in contexts is tagged with an indexed modality μ_F which arises from unboxing. Intuitively, this annotation means that the term bound to x is defined inside modality μ under the effect context F.

Contexts contain locks carrying indexed modalities which track effect transformations for variable bindings. For instance, the following context is well-formed at effect context E. Reading from left to right, the lock $\text{🔒}_{[E]_F}$ switches the effect context from F to E.

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

Following MTT, we define $\mathsf{locks}(-)$ to compose all the modalities on the locks in a context.

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

Following MTT, we identify contexts up to the following two equations.

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

3.5 Typing

The typing rules of Met are shown in Figure 3. The typing judgement $\Gamma \vdash M : A \ @\ E$ means that the term M has type A under context Γ and effect context E. As usual, we require $\Gamma \ @\ E$, $\Gamma \vdash E$, $\Gamma \vdash A : K$ for some K, and well-formedness for type annotations as well-formedness conditions. We explain the interesting rules, which are highlighted in grey; the other rules are standard.

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

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

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

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

$$\textsc{T-TAbs}\ \frac{\Gamma, \alpha : K \vdash V : A \ @\ E}{\Gamma \vdash \Lambda \alpha^K.V : \forall \alpha^K.A \ @\ E}
\qquad
\textsc{T-Abs}\ \frac{\Gamma, x : A \vdash M : B \ @\ E}{\Gamma \vdash \lambda x^A.M : A \to B \ @\ E}
\qquad
\textsc{T-TApp}\ \frac{\Gamma \vdash M : \forall \alpha^K.B \ @\ E \qquad \Gamma \vdash A : K}{\Gamma \vdash M\,A : B[A/\alpha] \ @\ E}$$

$$\textsc{T-App}\ \frac{\Gamma \vdash M : A \to B \ @\ E \qquad \Gamma \vdash N : A \ @\ E}{\Gamma \vdash M\,N : B \ @\ E}
\qquad
\textsc{T-Do}\ \frac{E = \ell : A \twoheadrightarrow B, F \qquad \Gamma \vdash N : A \ @\ E}{\Gamma \vdash \mathbf{do}\ \ell\ N : B \ @\ E}$$

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

$$\textsc{T-Handler}\ \frac{\begin{array}{c} H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\ \Gamma, \text{🔒}_{\langle D \rangle_F} \vdash M : A \ @\ D + F \qquad \Gamma, x : \langle D \rangle A \vdash N : B \ @\ F \\ D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N_i : B \ @\ F]_i \end{array}}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : B \ @\ F}$$

Fig. 3. Typing rules for core Met.

Modality Introduction and Elimination. Modalities are introduced by T-Mod and eliminated by T-Letmod. The term $\mathbf{mod}_\mu\, V$ introduces modality μ to the type of the conclusion and lock 🔒_{μ_F} into the context of the premise, and requires the value V to be well-typed under the new effect context E manipulated by μ. The lock 🔒_{μ_F} tracks the change to the effect context. We restrict $\mathbf{mod}$ to values as it manipulates effect contexts [2, 34]. Otherwise, a term such as $\mathbf{mod}_{\langle| \ell \rangle}\ (\mathbf{do}\ \ell\ V)$ would type check under the empty effect context but get stuck.

Following MTT, we use let-style modality elimination which takes another modality ν in addition to the modality μ that is eliminated from V. This is crucial for sequential unboxing. For instance, y and z in the following term are bound as $y :_{\nu} \mu A$ and $z :_{\nu \circ \mu} A$, respectively.

$$\lambda x^{\nu \mu A}.\textbf{let mod}_{\nu}\ y = x \textbf{ in let}_{\nu}\ \textbf{mod}_{\mu}\ z = y \textbf{ in } M$$

As with boxing, unboxing is restricted to values. We treat a type application of a value as itself a value as type application does not perform any effects. Consequently, we gain the flexibility to use type applications in boxing and unboxing.

Masking and Handling. Masking and handling provide specialised means to introduce values with relative modalities. A mask $\textbf{mask}_L\, M$ introduces the mask modality $\langle L \rangle$, and a handler $\textbf{handle}\ M\ \textbf{with}\ H$ binds a value boxed with the extension modality $\langle D \rangle$ in its return clause. Unlike **mod**, these constructs apply to computations as they perform masking and handling semantically.

As shown in Section 2.7, entering the scope of a handler for operations D means that D is extended with the ambient effect context. Values escaping a handler must be boxed with $\langle D \rangle$ since they may use these previously extended operations. Similarly, going into the scope of $\textbf{mask}_L$ means that effect labels L are removed from the ambient effects F. For those values leaving masks, they need to be boxed with $\langle L \rangle$ since they cannot use these previously masked operations.

Accessing Variables. The T-Var rule uses the auxiliary judgement $\Gamma \vdash (\mu, A) \Rightarrow \nu \mathbin{@} F$ defined in Figure 2. Variables of absolute types can always be used as they do not depend on the ambient effect context. For a non-absolute term variable binding $x :_{\mu_F} A$ from context $\Gamma, x :_{\mu_F} A, \Gamma'$, we must guarantee that it is safe to use x in the current effect context. The effect context where x is introduced is F. As we track all transformations on effect contexts up to the binding of x as locks in Γ', the current effect context E is obtained by applying all modalities on locks in Γ' to F. Thus, the condition $\mu_F \Rightarrow \mathsf{locks}(\Gamma')_F$ defined in Section 3.3 is needed for effect safety.

Let us look at some examples. Consider the following judgement.

$$🔒_{\langle \ell_2 \rangle},\ y :_{\langle \ell_1 \rangle_{\ell_2}} 1 \to \mathsf{Int} \vdash \textbf{handle}\ y\ ()\ \textbf{with}\ \{\ell_1\} : \_ \mathbin{@} \ell_2$$

The handler introduces a lock $🔒_{\langle \ell_1 \rangle_{\ell_2}}$. This judgement is valid because we have $\langle \ell_1 \rangle_{\ell_2} \Rightarrow \langle \ell_1 \rangle_{\ell_2}$. It would be invalid if we were to extend the handler to handle ℓ_2, as $\langle \ell_1 \rangle_{\ell_2} \Rightarrow \langle \ell_1, \ell_2 \rangle_{\ell_2}$ does not hold. Otherwise, the function y might use ℓ_2 which is accidentally handled here.

$$🔒_{\langle \ell_2 \rangle},\ y :_{\langle \ell_1 \rangle_{\ell_2}} 1 \to \mathsf{Int} \vdash_{\mathsf{wrong}} \textbf{handle}\ y\ ()\ \textbf{with}\ \{\ell_1, \ell_2\} : \_ \mathbin{@} \ell_2$$

We can fix this judgement by masking ℓ_2. The transformation $\langle \ell_1 \rangle_{\ell_2} \Rightarrow (\langle \ell_1, \ell_2 \rangle_{\ell_2} \circ \langle \ell_2 \rangle_{\ell_1, \ell_2, \ell_2})$ is well-defined since $\langle \ell_1, \ell_2 \rangle_{\ell_2} \circ \langle \ell_2 \rangle_{\ell_1, \ell_2, \ell_2} = \langle \ell_1 \rangle_{\ell_2}$.

$$🔒_{\langle \ell_2 \rangle},\ y :_{\langle \ell_1 \rangle_{\ell_2}} 1 \to \mathsf{Int} \vdash \textbf{handle}\ \textbf{mask}_{\ell_2}\ (y\ ())\ \textbf{with}\ \{\ell_1, \ell_2\} : \_ \mathbin{@} \ell_2$$

Subeffecting. Subeffecting is incorporated into the T-Var rule within the transformation relation $\mu_F \Rightarrow \nu_F$. We have seen how subeffecting works in Section 2.2. We give another example here upcasting $[]$ to $[E]$.

$$\lambda x^{[](\mathsf{Int} \to \mathsf{Int})}.\textbf{let mod}_{[]}\ y = x \textbf{ in mod}_{[E]}\ y : [](\mathsf{Int} \to \mathsf{Int}) \to [E](\mathsf{Int} \to \mathsf{Int})$$

Due to subeffecting, given a variable binding $x : 1 \to 1$ under ambient effect context E, we cannot assume E is exactly the effect context required to invoke a function bound to x. For instance, consider the following program.

$$\textbf{let}\ f = \textbf{mod}_{[]}\ (\lambda x^{1 \to 1}.x\ ()) \textbf{ in let mod}_{[]}\ g = f \textbf{ in } g\ (\lambda \_.\textbf{do}\ \ell\ V; ())$$

Though the function $\lambda x^{1 \to 1}.x$ is typed checked with the empty ambient effect context, the term bound to x in the application of g actually invokes ℓ.

3.6 Masking and Handling with Absolute Kinds

Masking attaches a mask modality to the return value of the term being masked, and handling attaches an extension modality to the return value of the term being handled. In practice, these return values often have absolute kind, which means these modalities can be omitted. We provide the following syntactic sugar to treats absolute return values specially for masking and handling. We also introduce syntactic sugar for specialised unboxing.

$$
\begin{array}{rcl}
\mathbf{mask}_L^{\mathsf{Abs}}\ M & \doteq & \mathbf{let}\ \mathbf{mod}_{\langle L|\rangle}\ x = \mathbf{mask}_L\ M\ \mathbf{in}\ x \\
\mathbf{handle}^{\mathsf{Abs}}\ M\ \mathbf{with}\ H & \doteq & \mathbf{handle}\ M\ \mathbf{with}\ H' \\
\text{where} & & H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\
 & & H' = \{\mathbf{return}\ x \mapsto \mathbf{let}\ \mathbf{mod}_{\langle|D\rangle}\ x = x\ \mathbf{in}\ N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\
\mathbf{let}\ \mathbf{mod}_\mu\ = M\ \mathbf{in}\ N & \doteq & (\lambda x.\mathbf{let}\ \mathbf{mod}_\mu\ x = x\ \mathbf{in}\ N)\ M \\
\mathbf{let}\ \mathbf{mod}_{\mu;\nu}\ x = V\ \mathbf{in}\ M & \doteq & \mathbf{let}\ \mathbf{mod}_\mu\ x = V\ \mathbf{in}\ \mathbf{let}_\mu\ \mathbf{mod}_\nu\ x = x\ \mathbf{in}\ M
\end{array}
$$

The following typing rules are derivable for absolute A, which allow us to elide modalities:

$$
\textsc{T-MaskAbs}\quad
\frac{\Gamma \vdash A : \mathsf{Abs} \qquad \Gamma, \text{🔒}_{\langle L|\rangle_F} \vdash M : A \mathbin{@} F - L}{\Gamma \vdash \mathbf{mask}_L^{\mathsf{Abs}}\ M : A \mathbin{@} F}
$$

$$
\textsc{T-HandleAbs}\quad
\frac{\begin{array}{cc}\Gamma \vdash A : \mathsf{Abs} & H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\ \Gamma, \text{🔒}_{\langle|D\rangle_F} \vdash M : A \mathbin{@} D + F & \Gamma, x : A \vdash N : B \mathbin{@} F \\ D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i & [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N_i : B \mathbin{@} F]_i\end{array}}{\Gamma \vdash \mathbf{handle}^{\mathsf{Abs}}\ M\ \mathbf{with}\ H : B \mathbin{@} F}
$$

3.7 Operational Semantics

The operational semantics for Met is quite standard. As type application values can reduce, we first define value normal forms U that cannot reduce, and evaluation contexts $\mathcal{E}$:

$$
\begin{array}{lrl}
\text{Value normal forms} & U ::= & x \mid \lambda x^A.M \mid \Lambda\alpha^K.V \mid \mathbf{mod}_\mu\, U \\
\text{Evaluation contexts} & \mathcal{E} ::= & [\,] \mid \mathcal{E}\ A \mid \mathcal{E}\ N \mid U\ \mathcal{E} \mid \mathbf{mod}_\mu\, \mathcal{E} \mid \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = \mathcal{E}\ \mathbf{in}\ M \\
 & \mid & \mathbf{do}\ \ell\, \mathcal{E} \mid \mathbf{mask}_L\, \mathcal{E} \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H
\end{array}
$$

The reduction rules are as follows.

$$
\begin{array}{lrcll}
\textsc{E-App} & (\lambda x^A.M)\ U & \rightsquigarrow & M[U/x] & \\
\textsc{E-TApp} & (\Lambda\alpha.V)\ A & \rightsquigarrow & V[A/\alpha] & \\
\textsc{E-LetMod} & \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = \mathbf{mod}_\mu\, U\ \mathbf{in}\ M & \rightsquigarrow & M[U/x] & \\
\textsc{E-Mask} & \mathbf{mask}_L\ U & \rightsquigarrow & \mathbf{mod}_{\langle L|\rangle}\ U & \\
\textsc{E-Ret} & \mathbf{handle}\ U\ \mathbf{with}\ H & \rightsquigarrow & N[(\mathbf{mod}_{\langle|D\rangle}\ U)/x], \text{where } (\mathbf{return}\ x \mapsto N) \in H & \\
\textsc{E-Op} & \mathbf{handle}\ \mathcal{E}[\mathbf{do}\ \ell\ U]\ \mathbf{with}\ H & \rightsquigarrow & N[U/p, (\lambda y.\mathbf{handle}\ \mathcal{E}[y]\ \mathbf{with}\ H)/r], & \\
 & & & \text{where } 0\text{--free}(\ell, \mathcal{E}) \text{ and } (\ell\ p\ r \mapsto N) \in H & \\
\textsc{E-Lift} & \mathcal{E}[M] & \rightsquigarrow & \mathcal{E}[N], & \text{if } M \rightsquigarrow N
\end{array}
$$

The only slightly non-standard aspect of the rules is the boxing of values escaping masks and handlers. In E-Ret, we assume handlers are decorated with the operations D that they handle.

Following Biernacki et al. [4], the predicate $n\text{--free}(\ell, \mathcal{E})$ is defined inductively on evaluation contexts as follows. The meta function $\mathsf{count}(\ell; L)$ yields the number of ℓ labels in L. We omit the inductive cases that do not change n. Notice that the cases for introduction and elimination of modalities fall into this category as they require values which cannot be of the form $\mathbf{do}\ \ell\ V$.

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

$$\frac{(n+1)\text{–free}(\ell, \mathcal{E}) \qquad \ell \in \mathsf{dom}(H)}{n\text{–free}(\ell, \mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H)} \qquad \frac{n\text{–free}(\ell, \mathcal{E}) \qquad \ell \notin \mathsf{dom}(H)}{n\text{–free}(\ell, \mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H)}$$

3.8 Type Soundness and Effect Safety

We prove type soundness and effect safety for Met. Our proofs cover the extensions in Section 4.

Met enjoys relatively standard substitution properties along the lines of Kavvos and Gratzer [29]. For example, we have the following rule for substituting values with modalities into terms.

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

We state and prove the relevant properties in Appendix A.5.

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 type E, if it is either in value normal form $M = U$ or of form $M = \mathcal{E}[\mathbf{do}\ \ell\ U]$ for $\ell \in E$ and $n\text{–free}(\ell, \mathcal{E})$.

We have the following theorems which in together give type soundness and effect safety, proved in Appendices A.6 and A.7.

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

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

4 Extensions to the Core Calculus

In this section we demonstrate that Met scales to support data types, richer handlers, and other useful primitives that provide extra expressiveness. We also introduce Mete, an extension of Met with effect variables, recovering the full expressive power of row-based effect systems. We prove type soundness and effect safety for all extensions.

4.1 Data Types and Crisp Induction

We demonstrate the extensibility of Met with data types by extending it with pair and sum types. Figure 4 shows the syntax and typing rules. The T-Pair, T-Inl, and T-Inr are standard introduction rules. The elimination rules T-CrispPair and T-CrispSum are more interesting. In addition to normal pattern matching, they interpret the value V under the effect context transformed by certain modalities ν, which can then be tagged to the variable bindings in case clauses. They follow the crisp induction principles of multimodal type theory [18, 45]. These crisp elimination rules provide extra expressiveness. For example, we can write the following function which transforms a sum of type $\mu(A+B)$ to another sum of type $(\mu A + \mu B)$. This function is not expressible without crisp elimination rules.

$$\lambda x^{\mu(A+B)}.\mathbf{let}\ \mathbf{mod}_\mu\ y = x\ \mathbf{in}\ \mathbf{case}_\mu\ y\ \mathbf{of}\ \{\mathbf{inl}\ x_1 \mapsto \mathbf{inl}\ (\mathbf{mod}_\mu\ x_1), \mathbf{inr}\ x_2 \mapsto \mathbf{inr}\ (\mathbf{mod}_\mu\ x_2)\}$$

The semantics of this extension is standard and shown in Appendix A.2.

$$\textsc{T-Pair}\ \frac{\Gamma \vdash M : A \mathbin{@} E \qquad \Gamma \vdash N : B \mathbin{@} E}{\Gamma \vdash (M, N) : (A, B) \mathbin{@} E} \qquad \textsc{T-Inl}\ \frac{\Gamma \vdash M : A \mathbin{@} E}{\Gamma \vdash \mathbf{inl}\ M : A + B \mathbin{@} E} \qquad \textsc{T-Inr}\ \frac{\Gamma \vdash M : B \mathbin{@} E}{\Gamma \vdash \mathbf{inr}\ M : A + B \mathbin{@} E}$$

$$\textsc{T-CrispPair}\ \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F} \vdash V : (A, B) \mathbin{@} E \qquad \Gamma, x :_{\nu_F} A, y :_{\nu_F} B \vdash M : A' \mathbin{@} F}{\Gamma \vdash \mathbf{case}_\nu\ V\ \mathbf{of}\ (x, y) \mapsto M : A' \mathbin{@} F}$$

$$\textsc{T-CrispSum}\ \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F} \vdash V : A + B \mathbin{@} E \qquad \Gamma, x :_{\nu_F} A \vdash M_1 : A' \mathbin{@} F \qquad \Gamma, y :_{\nu_F} B \vdash M_2 : A' \mathbin{@} F}{\Gamma \vdash \mathbf{case}_\nu\ V\ \mathbf{of}\ \{\mathbf{inl}\ x \mapsto M_1, \mathbf{inr}\ y \mapsto M_2\} : A' \mathbin{@} F}$$

Fig. 4. Typing rules for data types in Met.

4.2 Commuting Modalities and Type Abstraction

Crisp elimination rules in Section 4.1 allow us to commute modalities and data types. Similarly, it is also sound and useful to commute type abstractions and modalities. However, the current modality elimination rule cannot do so, for a similar reason to why it is not possible to transform $\forall \alpha.A + B$ to $(\forall \alpha.A) + (\forall \alpha.B)$ in System F. We extend modality elimination to the form $\mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda\overline{\alpha^K} x = V\ \mathbf{in}\ M$ which allows V to use additional type variables in $\overline{\alpha^K}$ which are abstracted when bound to x. The extended typing and reduction rules are as follows.

$$\textsc{T-LetMod'}\ \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F}, \overline{\alpha : K} \vdash V : \mu A \mathbin{@} E \qquad \Gamma, x :_{\nu_F \circ \mu_E} \forall\overline{\alpha^K}.A \vdash M : B \mathbin{@} F}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda\overline{\alpha^K}.x = V\ \mathbf{in}\ M : B \mathbin{@} F}$$

$$\textsc{E-LetMod'} \quad \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda\overline{\alpha^K}.x = \mathbf{mod}_\mu\ U\ \mathbf{in}\ M \rightsquigarrow M[(\Lambda\overline{\alpha^K}.U)/x]$$

For instance, we can now write a function of type $\forall\alpha^K.\mu A \to \mu(\forall\alpha.A)$ where $\alpha \notin \mathrm{ftv}(\mu)$ as follows.

$$\lambda x^{\forall\alpha^K.\mu A}.\mathbf{let}\ \mathbf{mod}_\mu\ \Lambda\alpha^K.y = x\ \alpha\ \mathbf{in}\ \mathbf{mod}_\mu\ y$$

4.3 Boxing Computations under Empty Effect Contexts

We have restricted boxes to values in order to guarantee effect safety. This restriction is not essential for $[]$. For example, suppose we have $f :_{[]} (A \to B)$ and $x :_{[]} A$, it is sound to treat $\mathbf{mod}_{[]}\ (f\ x)$ as a computation which returns a value of type $[]B$. As $f\ x$ is evaluated under the empty effect context, we can guarantee that it cannot get stuck on unhandled operations.

We extend the introduction rule for the empty absolute modality to allow non-value terms with the following typing rule.

$$\textsc{T-BoxAbs}\ \frac{\Gamma, \text{🔒}_{[]_F} \vdash M : A \mathbin{@} \cdot}{\Gamma \vdash \mathbf{mod}_{[]}\ M : []A \mathbin{@} F}$$

The same generalisation applies to T-Mask and T-Handler. As an example, we can write the following *app* function.

$$\begin{aligned} app &: \forall\alpha.\forall\beta.[](\alpha \to \beta) \to []\alpha \to []\beta \\ app &= \Lambda\alpha.\Lambda\beta.\lambda f.\lambda x.\mathbf{let}\ \mathbf{mod}_{[]}\ f = f\ \mathbf{in}\ \mathbf{let}\ \mathbf{mod}_{[]}\ x = x\ \mathbf{in}\ \mathbf{mod}_{[]}\ (f\ x) \end{aligned}$$

The formula corresponding to the type of this function is commonly referred to as Axiom K in modal logic and is also satisfied by other similar modalities such as the safe modality of Choudhury and Krishnaswami [10].

4.4 Absolute and Shallow Handlers

Up to now we have considered only *deep* handlers of the form **handle** M **with** H where M depends on the ambient effect contexts. Deep handlers automatically wrap the handler around the body of the continuation r captured in a handler clause, and thus r depends on the ambient effect context. Though this usually suffices in practice, in some cases we may want the computation M or the continuation to be absolute, i.e., independent from the ambient effect context. This situation is more prevalent in METE with effect variables.

We extend the handler syntax to $\textbf{handle}^{\mathbb{A}}$ M **with** H with the following typing rule.

$$\textsc{T-Handler}^{\mathbb{A}}\quad \frac{\begin{array}{c} D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{[D+E]_F} \vdash M : A \mathbin{@} D + E \\ \Gamma, x : [D+E]A \vdash N : B \mathbin{@} F \qquad [\Gamma, p_i : A_i, r_i : [F](B_i \to B) \vdash N_i : B \mathbin{@} F]_i \end{array}}{\Gamma \vdash \textbf{handle}^{\mathbb{A}}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}$$

The T-HANDLER$^{\mathbb{A}}$ rule extends the context with an absolute lock $\text{🔒}_{[D+E]_F}$ specifying the effect context for M, and boxes the continuation r with the absolute modality $[F]$, where F exactly gives the effect context after handling. We also extend the handler syntax with shallow handlers $\textbf{handle}^{\dagger}$ M **with** H, in which the handler is not automatically wrapped around the body of continuations, and absolute shallow handlers $\textbf{handle}^{\mathbb{A}\dagger}$ M **with** H [22, 26]. The full syntax, typing rules, and semantics for these handlers are shown in Appendix A.2.

4.5 Effect Variables

Though MET suffices for many common use-cases of effects and handlers in practice, there are situations in which it is useful to refer to one or more effect contexts that differ from the ambient one (such as the higher-order fork operation in Section 2.10).

METE, the extension of MET with effect variables, is quite lightweight and straightforward.

$$\text{Effects} \quad E ::= \cdot \mid \ell : P, E \mid \varepsilon \mid E\backslash L \qquad\qquad \text{Kinds} \quad K ::= \cdots \mid \mathsf{Eff}$$

$\boxed{E \equiv F}$ $\boxed{E \leqslant F}$

$$\frac{}{E\backslash\cdot \equiv E} \qquad \frac{}{\cdot\backslash L \equiv \cdot} \qquad \frac{}{(\ell : P, E)\backslash(\ell, L) \equiv E\backslash L} \qquad \frac{\ell \notin L}{(\ell : P, E)\backslash L \equiv \ell : P, E\backslash L}$$

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

$\boxed{E - L}$

$$\varepsilon\backslash L - L' \quad = \quad \varepsilon\backslash(L, L')$$

We extend the syntax of effect contexts E with effect variables ε. As is typical for row polymorphism, we restrict each effect type to contain at most one effect variable. We also extend the syntax with effect masking $E\backslash L$, which means the effect types given by masking L from E. The latter is needed to keep the syntax of effect contexts closed under the masking operation $E - L$; otherwise we cannot define $\varepsilon - L$. In other words, the syntax of effects is the free algebra generated from extending D, E and masking $E\backslash L$ with base elements $\cdot$ and ε.

The effect equivalence and subeffecting rules are extended in a relatively standard way. We do not allow non-trivial equivalence or subtyping between different effect variables. We always identity effects up to the equivalence relation. That is, we can directly treat syntax of effects as the free algebra quotiented by the equivalence relation $E \equiv F$. Observe that using the equivalence relation, all open effect types with effect variable ε can be simplified to an equivalent normal form $D, \varepsilon \backslash L$. We assume the operation $E - L$ is defined for effects E in normal form and extend it with one case for effect variables.

As the extension of Mete is local and only influences relevant definitions of effects, the meta theory and proofs for Met directly apply to Mete without any non-trivial changes.

5 Encoding Row-based Effect Systems into Met

Even without effect variables, Met is expressive enough to encode programs from conventional row-based effect systems as long as effect variables on function arrows always refer to the lexically closest one. This is an important special case, since most functions in practice use at most one effect variable. For example, as of July 2024, the Koka repository contains 520 effectful functions across 112 files but only 86 functions across 5 files use more than one effect variable, almost all of them internal primitives for handlers not exposed to programmers. Moreover, almost all programs in the Frank repository make no mention of effect variables at all, relying on syntactic sugar to hide the single effect variable.

5.1 Row Effect Types with a Single Effect Variable

We define $\mathrm{F}^1_{\text{eff}}$, a System F-style core calculus with row-based effect types in the style of Koka [31], but where each scope can only refer to a single effect variable. The syntax is defined as follows.

$$
\begin{array}{llrl}
\text{Types} & A, B & ::= & \mathsf{Int} \mid A \to^{\{E|\varepsilon\}} B \mid \forall \varepsilon.A \\
\text{Terms} & M, N & ::= & x \mid \lambda^{\{E|\varepsilon\}} x^A.M \mid M\,N \mid \Lambda\varepsilon.V \mid M\,\{E|\varepsilon\} \\
 & & \mid & \textbf{mask}_L\ M \mid \textbf{do}\ \ell\ M \mid \textbf{handle}\ M\ \textbf{with}\ H \\
\text{Values} & V, W & ::= & x \mid \lambda^{\{E|\varepsilon\}} x^A.M \mid \Lambda\varepsilon.V \\
\text{Effects} & E, F, L, D & ::= & \cdot \mid \ell, E \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x :_\varepsilon A \mid \Gamma, \blacklozenge_E \mid \Gamma, \blacklozenge^\Lambda_E
\end{array}
$$

We include integers, effectful function arrows, and effect abstraction $\forall \varepsilon.A$. As we consider only one effect variable at a time, we need not track effect variables on function types and effect type abstraction. Nonetheless, we include them in grey font for easier comparison with existing calculi. In Γ, we track for each variable the effect variable at which effect context it was introduced. Further, we add markers $\blacklozenge_E$ and $\blacklozenge^\Lambda_E$ to the context, which track the change of effect context due to functions, masks, handlers, and effect abstraction. These markers are not needed by the typing rules but help with the encoding. As with Met, we require contexts to be ordered. To convey the essential idea of the encoding, we omit type polymorphism and data types from $\mathrm{F}^1_{\text{eff}}$; we discuss these extensions in Section 5.3. For simplicity we also assume operation signatures come from a global context $\Sigma = \{\overline{\ell : A \twoheadrightarrow B}\}$, thus unifying extensions, masks, and effects (effect contexts) into one syntactic category. Mirroring our kind restriction for operation signatures in Met, we assume that these A and B are not function arrows, but they can be effect abstractions (which may themselves contain function arrows).

Figure 5 gives the typing rules of $\mathrm{F}^1_{\text{eff}}$. The judgement $\Gamma \vdash M : A \,!\, \{E|\varepsilon\}$ states that in context Γ, the term M has type A under an effect context consisting of concrete effects E extended with effect variable ε. The typing rules are mostly standard for row-based effect type systems.

In the R-Var rule, we ensure that either the current effect variable matches the effect variable at which the variable was introduced or that the value is an effect abstraction. These constraints guarantee programs can only use one effect variable in one scope.

The R-App, R-Do, R-Mask, and R-Handler rules are standard, while the R-Abs rule is standard except for requiring the effect variable to remain unchanged.

The R-EAbs rule introduces a new effect variable ε' and the R-EApp rule instantiates an effect abstraction. While conventional systems allow instantiating with any effect row, this rule only allows instantiation with the ambient effects. The instantiation operator $[\{E|\varepsilon\}/]$ implements standard type substitution for the single effect variable.

$$\begin{aligned}
\mathsf{Int}[\{E|\varepsilon\}/] &= \mathsf{Int}\\
(A \to^{\{F|\varepsilon'\}} B)[\{E|\varepsilon\}/] &= A[\{E|\varepsilon\}/] \to^{\{F,E|\varepsilon\}} B[\{E|\varepsilon\}/]\\
(\forall\varepsilon'.A)[\{E|\varepsilon\}/] &= \forall\varepsilon'.A
\end{aligned}$$

Revisiting the example from Section 2.6, we can write the regen function in F^1_{eff} as follows:

$$\begin{aligned}
&\mathsf{regen} : \forall.(\mathsf{Int} \to^{\mathsf{Yield}} \mathsf{Int}) \to^{\mathsf{Yield}} ((1 \to^{\mathsf{Yield},\mathsf{Yield}} 1) \to^{\mathsf{Yield}} 1)\\
&\mathsf{regen} = \Lambda.\lambda f.\lambda m.\textbf{handle}\ m\ ()\ \textbf{with}\ \{\textbf{return}\ x \mapsto x, \mathsf{Yield}\, s\, r \mapsto \textbf{do}\ \mathsf{Yield}\ (f\ s); r\ ()\}
\end{aligned}$$

5.2 Encoding

We now give translations for types and contexts of F^1_{eff} into Met. We transform F^1_{eff} types at effect context E to modal types in Met by the translation $[\![-]\!]_E$. For integer types, we insert the identity modality. For function arrows, the relative modality $\langle E - F | F - E\rangle$ heralds the transition from effect context E to effect context F as we enter the function. For effect abstraction, the empty absolute modality simulates entering a new effect context with different effect variables. We translate contexts by translating each type and moving top-level modalities to their bindings. For each marker, we insert a corresponding lock to reflect the changes of effect context.

$$\boxed{\Gamma \vdash M : A \,!\, \{E|\varepsilon\}}$$

R-Var
$$\frac{\varepsilon = \varepsilon' \text{ or } A = \forall\varepsilon''.A'}{\Gamma_1, x :_{\varepsilon'} A, \Gamma_2 \vdash x : A \,!\, \{E|\varepsilon\}}$$

R-Abs
$$\frac{\Gamma, \blacklozenge_E, x :_\varepsilon A \vdash M : B \,!\, \{F|\varepsilon\}}{\Gamma \vdash \lambda^{\{F|\varepsilon\}} x^A.M : A \to^{\{F|\varepsilon\}} B \,!\, \{E|\varepsilon\}}$$

R-App
$$\frac{\Gamma \vdash M : A \to^{\{E|\varepsilon\}} B \,!\, \{E|\varepsilon\} \qquad \Gamma \vdash N : A \,!\, \{E|\varepsilon\}}{\Gamma \vdash M\, N : B \,!\, \{E|\varepsilon\}}$$

R-EAbs
$$\frac{\varepsilon' \notin \mathsf{ftv}(\Gamma) \qquad \Gamma, \blacklozenge^{\Lambda}_E \vdash V : A \,!\, \{\cdot|\varepsilon'\}}{\Gamma \vdash \Lambda\varepsilon'.V : \forall\varepsilon'.A \,!\, \{E|\varepsilon\}}$$

R-EApp
$$\frac{\Gamma \vdash M : \forall\varepsilon'.A \,!\, \{E|\varepsilon\}}{\Gamma \vdash M\,\{E|\varepsilon\} : A[\{E|\varepsilon\}/] \,!\, \{E|\varepsilon\}}$$

R-Mask
$$\frac{\Gamma, \blacklozenge_{L+E} \vdash M : A \,!\, \{E|\varepsilon\}}{\Gamma \vdash \textbf{mask}_L\ M : A \,!\, \{L+E|\varepsilon\}}$$

R-Do
$$\frac{(\ell : A \twoheadrightarrow B) \in \Sigma \qquad \Gamma \vdash M : A \,!\, \{\ell, E|\varepsilon\}}{\Gamma \vdash \textbf{do}\ \ell\ M : B \,!\, \{\ell, E|\varepsilon\}}$$

R-Handler
$$\frac{\begin{array}{c}\Gamma, \blacklozenge_E \vdash M : A \,!\, \{\overline{\ell_i}, E|\varepsilon\} \qquad \Gamma, x :_\varepsilon A \vdash N : B \,!\, \{E|\varepsilon\}\\ \{\ell_i : A_i \twoheadrightarrow B_i\} \subseteq \Sigma \qquad [\Gamma, p_i :_\varepsilon A_i, r_i :_\varepsilon B_i \to^E B \vdash N_i : B \,!\, \{E|\varepsilon\}]_i\\ H = \{\textbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i\end{array}}{\Gamma \vdash \textbf{handle}\ M\ \textbf{with}\ H : B \,!\, \{E|\varepsilon\}}$$

Fig. 5. Typing rules of F^1_{eff}

$$
\begin{array}{rcl@{\qquad}rcl}
[\![\mathsf{Int}]\!]_E &=& \langle|\rangle\mathsf{Int} & [\![\cdot]\!]_E &=& \cdot \\
[\![A \to^F B]\!]_E &=& \langle E - F|F - E\rangle([\![A]\!]_F \to [\![B]\!]_F) & [\![\Gamma, x : A]\!]_E &=& [\![\Gamma]\!]_E, x :_{\mu_E} A' \text{ for } \mu A' = [\![A]\!]_E \\
[\![\forall.A]\!]_E &=& [\,][\![A]\!]. & [\![\Gamma, \blacklozenge_F]\!]_E &=& [\![\Gamma]\!]_F, \text{🔒}_{\langle F-E|E-F\rangle} \\
\mathsf{topmod}(\mu A) &=& \mu & [\![\Gamma, \blacklozenge_F^{\Lambda}]\!]_{.} &=& [\![\Gamma]\!]_F, \text{🔒}_{[\,]}
\end{array}
$$

Observe that not every valid typing judgement in $\mathrm{F}^1_{\mathrm{eff}}$ can be transformed to valid typing judgement in Met, because the translation depends on markers in contexts, while the typing of $\mathrm{F}^1_{\mathrm{eff}}$ does not. We define well-scoped typing judgements, which characterise the typing judgements for which our encoding is well-defined, as follows.

Definition 5.1 (Well-scoped). A typing judgement $\Gamma_1, x :_\varepsilon A, \Gamma_2 \vdash M : B \,!\, E$ is *well-scoped for* x if either $x \notin \mathsf{fv}(M)$ or $\blacklozenge_F^{\Lambda} \notin \Gamma_2$ or $A = \forall.A'$. A typing judgement $\Gamma \vdash M : A \,!\, E$ is *well-scoped* if it is well-scoped for all $x \in \Gamma$.

In particular, if the judgement at the bottom of a derivation tree is well-scoped, then every judgement in the derivation tree is well-scoped.

Figure 6 shows the translation from $\mathrm{F}^1_{\mathrm{eff}}$ terms with their types and effect contexts to Met terms. We have the following type preservation theorem. The proof is given in Appendix A.8.

Lemma 5.2 (Type preservation of encoding). *If* $\Gamma \vdash M : A \,!\, \{E|\varepsilon\}$ *is well-scoped, then* $M : A \,!\, E \dashrightarrow M'$ *and* $[\![\Gamma]\!]_E \vdash M' : [\![A]\!]_E \;@\; E$.

In the term translation, all terms are translated to boxed terms with proper modalities consistent with those given by the type translation, such that used term variables are always accessible after translation. We *greedily unbox* top-level modalities of term variables when they are bound, and *lazily box* them when they are used. Throughout, we use the syntax defined in Section 3.6.

Greedy unboxing happens for variable bindings such as λ-abstractions and handlers. In the R-Abs case, we unbox the top-level modality of variable x immediately after x is bound. Additionally, we box the whole function with the relative modality $\langle E - F|F - E\rangle$, reflecting the effect context transition. In the R-Handler case, we similarly unbox the variable bindings for return clauses and operation clauses immediately after they are bound. In the operation clauses, we need only unbox the argument to the handler p_i; the resume function r_i is introduced under the current effect context E. In the return clause, we unbox x with $\langle\!| \overline{\ell_i} \rangle \circ \mu$ and then transform this modality to μ' given by $\mathsf{topmod}([\![A]\!]_E)$ in order to match the current effect context E. We have proved this modality transformation and the ones mentioned below in Appendix A.8.

Similar to the R-Abs case, the R-EAbs case boxes the translated value with the empty absolute modality. Similar to the return clauses of the R-Handler case, the R-Mask case transforms the modality $\langle L |\!\rangle \circ \mu_1$ to μ_2 in order to match the current effect context $L + E$.

Lazy boxing happens when variables are used in the R-Var rule. Note that variables might be used at a different effect context than they were introduced, in which case we must establish the existence of a modality transformation.

As a result of translating all terms to boxed terms, we must insert unboxing for elimination rules such as R-App and R-EApp. Nothing special happens for the R-Do case.

$$\boxed{M : A\,!\,E \dashrightarrow M'}$$

$$\textsc{R-Var}\quad \frac{\mu \coloneqq \mathsf{topmod}(\llbracket A \rrbracket_E)}{x : A\,!\,E \dashrightarrow \mathbf{mod}_\mu\, x}$$

$$\textsc{R-App}\quad \frac{M : A \to^E B\,!\,E \dashrightarrow M' \qquad N : A\,!\,E \dashrightarrow N' \qquad x\ \mathsf{fresh}}{M\,N : B\,!\,E \dashrightarrow \mathbf{let}\ \mathbf{mod}_{\langle|\rangle}\ x = M'\ \mathbf{in}\ x\,N'}$$

$$\textsc{R-Abs}\quad \frac{M : B\,!\,F \dashrightarrow M' \qquad \nu \coloneqq \langle E - F | F - E \rangle \qquad \mu \coloneqq \mathsf{topmod}(\llbracket A \rrbracket_F)}{\lambda^F x^A.M : A \to^F B\,!\,E \dashrightarrow \mathbf{mod}_\nu\, (\lambda x^{\llbracket A \rrbracket_F}.\mathbf{let}\ \mathbf{mod}_\mu\ x = x\ \mathbf{in}\ M')}$$

$$\textsc{R-EAbs}\quad \frac{V : A\,!\,\cdot \dashrightarrow V'}{\Lambda.V : \forall.A\,!\,E \dashrightarrow \mathbf{mod}_{[]}\, V'}$$

$$\textsc{R-EApp}\quad \frac{M : \forall.A\,!\,E \dashrightarrow M' \qquad x\ \mathsf{fresh}}{M\,@ : A[E/]\,!\,E \dashrightarrow \mathbf{let}\ \mathbf{mod}_{[]}\ x = M'\ \mathbf{in}\ x}$$

$$\textsc{R-Do}\quad \frac{M : A\,!\,\ell, E \dashrightarrow M'}{\mathbf{do}\ \ell\ M : B\,!\,\ell, E \dashrightarrow \mathbf{do}\ \ell\ M'}$$

$$\textsc{R-Mask}\quad \frac{M : A\,!\,E \dashrightarrow M' \qquad \mu_1 \coloneqq \mathsf{topmod}(\llbracket A \rrbracket_E) \qquad \mu_2 \coloneqq \mathsf{topmod}(\llbracket A \rrbracket_{L+E})}{\mathbf{mask}_L\ M : A\,!\,L + E \dashrightarrow \mathbf{let}\ \mathbf{mod}_{\langle L|\rangle;\mu_1}\ x = \mathbf{mask}_L\ M'\ \mathbf{in}\ \mathbf{mod}_{\mu_2}\, x}$$

$$\textsc{R-Handler}\quad \frac{\begin{array}{c} M : A\,!\,\overline{\ell_i}, E \dashrightarrow M' \qquad N : B\,!\,E \dashrightarrow N' \qquad [N_i : B\,!\,E \dashrightarrow N_i']_i \\ \mu \coloneqq \mathsf{topmod}(\llbracket A \rrbracket_{\overline{\ell_i},E}) \qquad \mu' \coloneqq \mathsf{topmod}(\llbracket A \rrbracket_E) \\ N'' \coloneqq \mathbf{let}\ \mathbf{mod}_{\langle|\overline{\ell_i}\rangle;\mu}\ x = x\ \mathbf{in}\ \mathbf{let}_{\mu'}\ \mathbf{mod}_{\langle|\rangle}\ x = \mathbf{mod}_{\langle|\rangle}\ x\ \mathbf{in}\ N' \\ [\mu_i \coloneqq \mathsf{topmod}(\llbracket A_i \rrbracket_\cdot) \qquad N_i'' \coloneqq \mathbf{let}\ \mathbf{mod}_{\mu_i}\ p_i = p_i\ \mathbf{in}\ N_i']_i \\ H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \qquad H' \coloneqq \{\mathbf{return}\ x \mapsto N''\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i''\}_i \end{array}}{\mathbf{handle}\ M\ \mathbf{with}\ H : B\,!\,E \dashrightarrow \mathbf{handle}\ M'\ \mathbf{with}\ H'}$$

Fig. 6. Encoding of F^1_{eff} in Met.

Revisiting the regen example from Section 2.6, we can directly translate the F^1_{eff} version above as follows into Met, omitting boxing and unboxing of the identity modality $\langle|\rangle$.

$$\mathsf{regen} : [\mathsf{Yield}]((\langle|\rangle(\mathsf{Int} \to \mathsf{Int}) \to \langle|\rangle(\langle|\mathsf{Yield}\rangle(1 \to 1) \to 1)))$$

$$\begin{aligned} \mathsf{regen} = \mathbf{mod}_{[]}(\mathbf{mod}_{\langle|\mathsf{Yield}\rangle}(\lambda f.(\lambda m.&\mathbf{let}\ \mathbf{mod}_{\langle|\mathsf{Yield}\rangle}\ m = m\ \mathbf{in}\ \mathbf{handle}\ m\,()\ \mathbf{with}\ \{ \\ &\mathbf{return}\ x \mapsto \mathbf{let}\ \mathbf{mod}_{\langle|\mathsf{Yield}\rangle}\ x = x\ \mathbf{in}\ x, \\ &\mathsf{Yield}\ s\ r \mapsto \mathbf{do}\ \mathsf{Yield}\ (f\ s); r\ ()\ \}))) \end{aligned}$$

This is essentially the same program as in Section 2.6, but with significant noise due to the greedy unboxing and (omitted) identity boxes. In practice, identity boxes are not necessary — they are only generated here to keep the encoding uniform. On the other hand, greedy unboxing is useful in practice. In Section 6, we show how Metel can automatically infer unboxing.

5.3 Extensibility of the Encoding

We have omitted value type polymorphism and data types in our encoding in order to focus on conveying the core idea. We now discuss how to extend the encoding to support these features.

Recall that the encoding in Section 5.2 translates each F^1_{eff} type and term to a boxed Met type and term consistently such that variable accessibility is preserved. Generalising the encoding to type polymorphism is relatively easy, as we need only ensure variable accessibility. For a polymorphic value with type $\forall\alpha.A$, the translation on the value of type A would give a value of modal type $\mu A'$ in MET. We can use our extension in Section 4.2 to commute the quantifier and modality to obtain a value of type $\mu(\forall\alpha.A')$.

Generalising the encoding to data types is more involved. For instance, given a pair of type (A, B), the translation on its components might give terms of type $\mu A'$ and $\nu B'$ with unrelated modalities. This makes it impossible to give the pair a modality other than $\langle|\rangle$, which can not be used in all contexts where the pair can be used in F^1_{eff}. To ensure variable accessibility, we need to greedily destruct the pair and unbox its components with modalities μ and ν respectively. The uses of this pair variable in the translated function body are replaced by fresh pairs of these unboxed components. For variable bindings of recursive data types, we need to greedily destruct only to the extent that the data type is unfolded in the function body (where we may treat recursive invocations as opaque). While this requires a somewhat global translation, it does not require destructing and unboxing the recursive data type more than a small number of times.

The essential reason for the translation being global comes from the fact that we use let-style unboxing following MTT. For modalities with certain structure (right adjoints), it is possible to use Fitch-style unboxing [11] which allows terms to be directly unboxed without binding [17, 46]. We are interested in exploring whether we could extend MET to use Fitch-style unboxing and thus give a compositional local encoding for recursive data types. Fortunately, these issues appear not to cause problems in practice. Functional programs typically use pattern-matching in a structured way that plays nicely with automatic unboxing.

6 A Surface Language with Type Inference

In this section we briefly outline the design of METEL, a call-by-value surface language based on METE with Hindley-Milner type inference [13] for ML types and modalities without complicated constraint solving (albeit some annotations are required for modalities).

The problem of inferring modal effect types is closely related to that of inferring first-class polymorphism. Box introduction is analogous to type abstraction (which type inference algorithms realise through generalisation). Box elimination is analogous to type application (which type inference algorithms realise through instantiation). As such, one can adapt any of the myriad techniques for combining first-class polymorphism with Hindley-Milner type inference. METEL is inspired by the approach of FREEZEML [15], a system that supports full impredicative polymorphism with a combination of type annotations and *frozen* term variables which disable instantiation. METEL is a conservative extension of ML, and thus can fully infer types for any ML programs without the need for any annotations. METEL uses the machinery of FREEZEML to support modal effect types, but does not support first-class polymorphism (although incorporating it using FREEZEML's mechanism would be relatively straightforward).

A central feature of METEL that makes it more convenient to program with than METE is that it infers unboxing when variables are used. For instance, the following METEL program

$$\lambda m^{\langle|\mathsf{Ask}\rangle(1\to\mathsf{Int})}.\textbf{handle}\ m\ ()\ \textbf{with}\ \{\mathsf{Ask}\ \_\ r \mapsto r\ 42\}$$

is elaborated to the following METE program:

$$\lambda m^{\langle|\mathsf{Ask}\rangle(1\to\mathsf{Int})}.\textbf{let mod}_{\langle|\mathsf{Ask}\rangle}\ \hat{m} = m\ \textbf{in handle}\ \hat{m}\ ()\ \textbf{with}\ \{\mathsf{Ask}\ \_\ r \mapsto r\ 42\}$$

We now summarise the key ideas behind the design of METEL.

- The underlying philosophy of METEL is to "never guess modalities". This is analogous to the underlying philosophy of FREEZEML to "never guess polymorphism".
- Following FREEZEML (and algorithmic presentations of ML) instantiation is performed by default when a variable is used (x).

- Similarly, METEL also performs unboxing for variables by default via elaboration.
- METEL allows type variables to be instantiated with modal types. This is analogous to allowing type variables to be instantiated with polymorphic types (giving rise to impredicative polymorphism) in FREEZEML.
- Following FREEZEML, variables can be *frozen* in order to suppress such instantiation ($\lceil x \rceil$, written `~x` in ASCII text as shown in Section 2.11).
- Boxing is never inferred. Though it would be possible to infer limited use of boxing in let-bindings (following FREEZEML and algorithmic presentations of ML), this would yield the most general modality which is not typically what we require for handlers.
- Type annotations are required for function argument types that contain modal types.
- Type annotations are only required for those bindings that contain modal types.

We lack space to include full technical details of METEL in the body of the paper, and in any case most of the subtleties and design choices are in essence the same as those one encounters in treating type inference with first-class polymorphism. The full specification for METEL is given in Appendix B. We formalise the type inference algorithm following the approach of type inference in context [19, 20]. Soundness and completeness of type inference is proved in Appendix C.

We have chosen a design inspired by FREEZEML in the full knowledge that other designs may be better suited to other circumstances. But as a means for enabling us to write the examples in Section 2 and for demonstrating the feasibility of implementing sound and complete type inference for modal effect types it has fulfilled its purpose. In the future, we intend to explore and implement an alternative design as an extension to OCAML, building on and complementing recent work on modal types for OCAML [34], and making use of existing means for supporting first-class polymorphism in OCAML.

7 Discussion and Related Work

We first discuss the most relevant systems: FRANK [12, 33], EFFEKT [7, 8], and $CC_{<:\Box}$ [6]. Then we discuss the relationship between MET and MTT [17, 18, 29]. Finally we discuss other related work.

7.1 Do Be Do Be Do

Our absolute and relative modalities are inspired by the *abilities* and *adjustments* in FRANK [12, 33]. Absolute modalities and abilities both specify the whole effect context required to run some computation, while relative modalities and adjustments both specify deltas to the ambient effect context. A key difference is that FRANK restricts adjustments to appear only beside function parameters and essentially treats these parameters as second-class computation variables. To write higher-order programs, FRANK implicitly inserts effect variables to pass ambient effects around. MET generalises abilities and adjustments to modalities which can appear flexibly in types, eliminating effect variables altogether. As demonstrated in Section 5, FRANK with implicit effect variables and no closed abilities is expressible in MET. FRANK's *adaptors* are richer than MET's masking, although we expect relative modalities to extend readily to cover this use.

7.2 Capability-based Effect Systems

Capability-based effect systems [6–8] interpret effects as capabilities and offer a form of implicit effect polymorphism through capability passing.

For example, in EFFEKT the `asList` for `Yield` has the following type:

```
def asList{ f: 1 ⇒ List[Int] / { Yield } }: List[Int] / {}
```

Here the block parameter `f` is allowed to use the capability `Yield` in addition to those from the context. The capability annotation `{Yield}` on its type is similar to our relative modalities.

A key difference between Effekt and Met is that Effekt requires blocks to be second-class, while Met supports first-class functions. Brachthäuser et al. [7] recovers first-class functions by boxing blocks. However, such boxed blocks cannot use capabilities from the context any more, because the boxes on types fully specifies the required capabilities, similar to our absolute modalities. For example, we can obtain a curried version of map in Effekt by boxing the result.

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

The return value has type `List[A] ⇒ List[B] at {f}`. The decoration `{f}` indicates that the return function captures the capability `f`. This sort of annotation is reminiscent of an effect variable. This is telling for why Met is not expressive enough to encode Effekt. To encode captured capability variables, as in `map1`, we need the expressiveness provided by effect variables in Mete.

Another key difference is that Effekt uses named handlers [5, 51, 54] where operations are dispatched to a specific named handler, whereas Met uses Plotkin and Pretnar [41]-style handlers where operations dispatched to the first matching handler in the evaluation context. Named handlers provide a form of effect generativity. In the future it would be interesting to explore variants of modal effect types with capabilities and generative effects [14].

$CC_{<:\square}$ [6], the basis for capture tracking in Scala 3, also provides succinct types for uncurried higher-order functions like `map`. As in Effekt, the curried version requires the result function to be explicitly annotated with its capture set `{f}`.

7.3 Relationship between Met and Multimodal Type Theory

The literature on multimodal type theory organises the structure of modes (objects), modalities (morphisms between objects), and their transformations (2-cells between morphisms) in a *2-category* [17, 18, 29] (or, in the case of a single mode, a semiring [1, 9, 39, 40]). In Met, modes are effect contexts E, modalities are $\mu_F : E \to F$, and transformations are $\mu_F \Rightarrow \nu_F$. However, we have found that 2-categories are not sufficient in a system that also includes submoding. To deal with this extra structure, we extend the 2-category to a *double category* with an additional kind of vertical morphisms between objects (in Met, vertical morphisms are the preorder relation $E \leqslant F$), as also proposed by Katsumata [28]. As a result, the transformations do not strictly require the two modalities to have the same sources and targets, enabling us to have $[]_F \Rightarrow [E]_F$ in Met. The relationship between Met and MTT is explained in detail in Appendix A.3.

7.4 Other Related Work

We discuss other related work on effect systems and modal types.

Row-based Effect Systems. Row polymorphism is one popular approach to implementing effect systems for effect handlers. Links [21] use Rémy-style row polymorphism with presence types [43], while Koka [31] and Frank [33] use scoped rows [30] which allow duplicated labels. Morris and McKinna [36] proposes a general framework for comparing different kinds of row types, and Yoshioka et al. [53] proposes a similar framework focusing on comparing effect rows. Met adopts Leijen-style scoped rows meanwhile allows operation signatures to be absent, similar to presence types. Mete extends Met with effect variables by row polymorphism and extending the algebraic structure of row types to be closed under extensions and masks.

Subtyping-based Effect Systems. Eff [3, 42] is equipped with an effect system with both effect variables and sub-effecting, based on the type inference and elaboration described in Karachalias et al. [27], which supports constraint solving for sub-effecting between effect variables. The effect system of Helium [5] is based on finite sets, offering a natural sub-effecting relation corresponding to set-inclusion. As such, their system aligns closely with Lucassen and Gifford [35]-style effect systems. Tang et al. [47] proposes an effectful calculus with effect polymorphism and sub-effecting via qualified types [25] following Rose [36]. We have both effect variables and sub-effecting in Mete and Metel but do not consider non-trivial constraint solving.

Modal Types and Effects. Nanevski [37] proposes a modal calculus for handling exceptions, using a necessity modality indexed by the set of names of used effects. Zyuzin and Nanevski [55] extends contextual modal types [38] to algebraic effects and handlers, using a contextual necessity modality to track effects and modelling context reachability as effect handling. Both of their necessity modalities are similar to our absolute modalities. They do not have similar constructs to our relative modalities. They both give comonadic semantics to the modalities, while Met adopts the standard CBV semantics and restrict modalities to values. They focus on theoretical work, while we aim to design a practical effect system with succinct types and backward compatibility. Choudhury and Krishnaswami [10] proposes to use the necessity modality to recover purity from an effectful calculus. This is similar to our empty absolute modality, especially when extended as in Section 4.3.

Effects in Call-By-Push-Value. In CBPV [32], effects are usually tracked on typing judgements for computations and captured into types when switching to values [16, 26, 48]. Met tracks effect contexts as modes for all terms in typing judgements to have succinct effect types.

8 Conclusion

We have proposed a novel modal effect type system which manages effect contexts by tracking changes to them via absolute and relative modalities. We formalised modal effect types in a core calculus following multimodal type theory. We illustrated our design through a collection of examples in a surface language with sound and complete type inference. We demonstrated the expressiveness of the calculus by encoding a practical fragment of a traditional effect system.

Future work includes: implementing our system as an extension to OCaml; exploring extensions of modal effect types with Fitch-style unboxing, named handlers, generative effects, and capabilities; combining modal effect types with control-flow linearity; and developing a denotational semantics.

References

[1] Andreas Abel and Jean-Philippe Bernardy. 2020. A unified view of modalities in type systems. *Proc. ACM Program. Lang.* 4, ICFP, Article 90 (aug 2020), 28 pages. https://doi.org/10.1145/3408972

[2] Danel Ahman. 2023. When Programs Have to Watch Paint Dry. In *Foundations of Software Science and Computation Structures*, Orna Kupferman and Pawel Sobocinski (Eds.). Springer Nature Switzerland, Cham, 1–23.

[3] Andrej Bauer and Matija Pretnar. 2013. An Effect System for Algebraic Effects and Handlers. In *Algebra and Coalgebra in Computer Science*, Reiko Heckel and Stefan Milius (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–16.

[4] 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. https://doi.org/10.1145/3158096

[5] 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. https://doi.org/10.1145/3371116

[6] 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. https://doi.org/10.1145/3618003

[7] 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. https://doi.org/10.1145/3527320

[8] 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. https://doi.org/10.1145/3428194

[9] Pritam Choudhury, Harley Eades III, Richard A. Eisenberg, and Stephanie Weirich. 2021. A graded dependent type system with a usage-aware semantics. *Proc. ACM Program. Lang.* 5, POPL, Article 50 (jan 2021), 32 pages. https://doi.org/10.1145/3434331

[10] Vikraman Choudhury and Neel Krishnaswami. 2020. Recovering purity with comonads and capabilities. *Proc. ACM Program. Lang.* 4, ICFP (2020), 111:1–111:28. https://doi.org/10.1145/3408993

[11] Ranald Clouston. 2018. Fitch-Style Modal Lambda Calculi. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10803)*, Christel Baier and Ugo Dal Lago (Eds.). Springer, 258–275. https://doi.org/10.1007/978-3-319-89366-2_14

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

[13] Luis Damas and Robin Milner. 1982. Principal Type-Schemes for Functional Programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Albuquerque, New Mexico) *(POPL '82)*. Association for Computing Machinery, New York, NY, USA, 207–212. https://doi.org/10.1145/582153.582176

[14] Paulo Emílio de Vilhena and François Pottier. 2023. A Type System for Effect Handlers and Dynamic Labels. In *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13990)*, Thomas Wies (Ed.). Springer, 225–252. https://doi.org/10.1007/978-3-031-30044-8_9

[15] Frank Emrich, Sam Lindley, Jan Stolarek, James Cheney, and Jonathan Coates. 2020. FreezeML: complete and easy type inference for first-class polymorphism. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 423–437. https://doi.org/10.1145/3385412.3386003

[16] 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. https://doi.org/10.1017/S0956796819000121

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

[18] Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2020. Multimodal Dependent Type Theory. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 492–506. https://doi.org/10.1145/3373718.3394736

[19] Adam Gundry, Conor McBride, and James McKinna. 2010. Type Inference in Context. In *MSFP@ICFP*. ACM, 43–54.

[20] Adam Michael Gundry. 2013. *Type inference, Haskell and dependent types.* Ph. D. Dissertation. University of Strathclyde, Glasgow, UK. http://oleg.lib.strath.ac.uk/R/?func=dbin-jump-full&object_id=22728

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

[22] Daniel Hillerström and Sam Lindley. 2018. Shallow Effect Handlers. In *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11275)*, Sukyoung Ryu (Ed.). Springer, 415–435. https://doi.org/10.1007/978-3-030-02768-1_22

[23] Daniel Hillerström. 2022. *Foundations for Programming and Implementing Effect Handlers.* Ph. D. Dissertation. The University of Edinburgh, UK. https://doi.org/10.7488/era/2122

[24] Roshan P. James and Amr Sabry. 2011. Yield: Mainstream Delimited Continuations. Informal proceedings of TPDC@RDP'11. https://legacy.cs.indiana.edu/~sabry/papers/yield.pdf.

[25] Mark P. Jones. 1994. A Theory of Qualified Types. *Sci. Comput. Program.* 22, 3 (1994), 231–256. https://doi.org/10.1016/0167-6423(94)00005-0

[26] 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. https://doi.org/10.1145/2500365.2500590

[27] Georgios Karachalias, Matija Pretnar, Amr Hany Saleh, Stien Vanderhallen, and Tom Schrijvers. 2020. Explicit effect subtyping. *J. Funct. Program.* 30 (2020), e15. https://doi.org/10.1017/S0956796820000131

[28] Shin-ya Katsumata. 2018. A Double Category Theoretic Analysis of Graded Linear Exponential Comonads. In *Foundations of Software Science and Computation Structures*, Christel Baier and Ugo Dal Lago (Eds.). Springer International Publishing, Cham, 110–127.

[29] 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. https://doi.org/10.1017/BSL.2023.14

[30] 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.

[31] 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. https://doi.org/10.1145/3009837.3009872

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

[33] 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. https://doi.org/10.1145/3009837.3009897

[34] Anton Lorenzen, Leo White, Stephen Dolan, Richard A. Eisenberg, and Sam Lindley. 2024. Oxidizing OCaml with Modal Memory Management. *Proc. ACM Program. Lang.* 8, ICFP (2024). https://antonlorenzen.de/oxidizing-ocaml-modal-memory-management.pdf

[35] John M. Lucassen and David K. Gifford. 1988. Polymorphic Effect Systems. In *POPL*. ACM Press, 47–57. https://doi.org/10.1145/73560.73564

[36] J. Garrett Morris and James McKinna. 2019. Abstracting Extensible Data Types: Or, Rows by Any Other Name. *Proc. ACM Program. Lang.* 3, POPL, Article 12 (jan 2019), 28 pages. https://doi.org/10.1145/3290325

[37] Aleksandar Nanevski. 2005. A modal calculus for exception handling. In *Intuitionistic Modal Logics and Applications Workshop (IMLA'05), Chicago, IL*.

[38] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. 2008. Contextual modal type theory. *ACM Trans. Comput. Log.* 9, 3 (2008), 23:1–23:49. https://doi.org/10.1145/1352582.1352591

[39] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.* 3, ICFP, Article 110 (jul 2019), 30 pages. https://doi.org/10.1145/3341714

[40] Tomas Petricek, Dominic Orchard, and Alan Mycroft. 2014. Coeffects: a calculus of context-dependent computation. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming* (Gothenburg, Sweden) *(ICFP '14)*. Association for Computing Machinery, New York, NY, USA, 123–135. https://doi.org/10.1145/2628136.2628160

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

[42] Matija Pretnar. 2014. Inferring Algebraic Effects. *Log. Methods Comput. Sci.* 10, 3 (2014). https://doi.org/10.2168/LMCS-10(3:21)2014

[43] 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.

[44] Dennis Ritchie and Ken Thompson. 1974. The UNIX Time-Sharing System. *Commun. ACM* 17, 7 (1974), 365–375.

[45] Michael Shulman. 2018. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Math. Struct. Comput. Sci.* 28, 6 (2018), 856–941. https://doi.org/10.1017/S0960129517000147

[46] Michael Shulman. 2023. Semantics of multimodal adjoint type theory. In *Proceedings of the 39th Conference on the Mathematical Foundations of Programming Semantics, MFPS XXXIX, Indiana University, Bloomington, IN, USA, June 21-23, 2023 (EPTICS, Vol. 3)*, Marie Kerjean and Paul Blain Levy (Eds.). EpiSciences. https://doi.org/10.46298/ENTICS.12300

[47] Wenhao Tang, Daniel Hillerström, Sam Lindley, and J. Garrett Morris. 2024. Soundly Handling Linearity. *Proc. ACM Program. Lang.* 8, POPL, Article 54 (jan 2024), 29 pages. https://doi.org/10.1145/3632896

[48] Cassia Torczon, Emmanuel Suárez Acevedo, Shubh Agrawal, Joey Velez-Ginorio, and Stephanie Weirich. 2023. Effects and Coeffects in Call-By-Push-Value (Extended Version). arXiv:2311.11795 [cs.PL] https://arxiv.org/abs/2311.11795

[49] Birthe van den Berg and Tom Schrijvers. 2023. A Framework for Higher-Order Effects & Handlers. *CoRR* abs/2302.01415 (2023). https://doi.org/10.48550/arXiv.2302.01415 arXiv:2302.01415

[50] Nicolas Wu, Tom Schrijvers, and Ralf Hinze. 2014. Effect Handlers in Scope. *SIGPLAN Not.* 49, 12 (Sept. 2014), 1–12. https://doi.org/10.1145/2775050.2633358

[51] 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. https://doi.org/10.1145/3563289

[52] Zhixuan Yang and Nicolas Wu. 2023. Modular Models of Monoids with Operations. *Proc. ACM Program. Lang.* 7, ICFP (2023), 566–603. https://doi.org/10.1145/3607850

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

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

[55] Nikita Zyuzin and Aleksandar Nanevski. 2021. Contextual modal types for algebraic effects and handlers. *Proc. ACM Program. Lang.* 5, ICFP (2021), 1–29. https://doi.org/10.1145/3473580

A Full Specification, Meta Theory, and Proofs for Met

We provide the specification, meta theory, and proofs for Met omitted in Section 3. Our proofs for meta theory of Met consider all extensions in Section 4 including effect variables (Mete).

A.1 Extra Rules

The full kinding and well-formedness rules for Met are shown in Figure 7. We include the kind Eff and syntax $E\backslash L$ to also cover Mete. The type equivalence and sub-effecting rules are shown in Figure 8. We highlight the special rule that allows us to add or remove absent labels from the right.

$$\boxed{\Gamma \vdash A : K} \boxed{\Gamma \vdash \mu} \boxed{\Gamma \vdash E : K} \boxed{\Gamma \vdash L} \boxed{\Gamma \vdash D} \boxed{\Gamma \vdash P} \boxed{\Gamma \vdash (\mu, A) \Rightarrow \nu \mathbin{@} F}$$

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

$$\frac{\Gamma \vdash A : \mathsf{Any} \qquad \Gamma \vdash B : \mathsf{Any}}{\Gamma \vdash A \to B : \mathsf{Any}} \qquad \frac{\Gamma, \alpha : K \vdash A : K'}{\Gamma \vdash \forall \alpha^K . A : K'} \qquad \frac{\Gamma \vdash L \qquad \Gamma \vdash D}{\Gamma \vdash \langle L|D\rangle} \qquad \frac{\Gamma \vdash E : \mathsf{Eff}}{\Gamma \vdash [E]}$$

$$\frac{}{\Gamma \vdash \cdot : \mathsf{Eff}} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash E : \mathsf{Eff}}{\Gamma \vdash \ell : P, E : \mathsf{Eff}} \qquad \frac{\Gamma \vdash E : \mathsf{Eff} \qquad \Gamma \vdash L}{\Gamma \vdash E\backslash L : \mathsf{Eff}}$$

$$\frac{}{\Gamma \vdash -} \qquad \frac{\Gamma \vdash A : \mathsf{Abs} \qquad \Gamma \vdash B : \mathsf{Abs}}{\Gamma \vdash A \twoheadrightarrow B} \qquad \frac{}{\Gamma \vdash L} \qquad \frac{}{\Gamma \vdash \cdot} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash D}{\Gamma \vdash \ell : P, D}$$

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

Fig. 7. Full kinding and well-formedness rules for Met and Mete.

A.2 Full Specification for Extensions to Met

Figure 9 gives the syntax and typing rules for data types, absolute and shallow handlers. Figure 10 gives the extensions to value normal forms, evaluation contexts, and operational semantics for the extensions with data types, absolute and relative handlers in Section 4.

$\boxed{L \equiv L'}$ $\boxed{D \equiv D'}$ $\boxed{E \equiv F}$ $\boxed{P \equiv P'}$ $\boxed{\mu \equiv \nu}$ $\boxed{A \equiv B}$

$$\frac{}{\cdot \equiv \cdot} \qquad \frac{L_1 \equiv L_2 \qquad L_2 \equiv L_3}{L_1 \equiv L_3} \qquad \frac{L \equiv L'}{\ell, L \equiv \ell, L'} \qquad \frac{\ell \neq \ell' \qquad L \equiv L'}{\ell, \ell', L \equiv \ell', \ell, L}$$

$$\frac{}{\cdot \equiv \cdot} \qquad \frac{D_1 \equiv D_2 \qquad D_2 \equiv D_3}{D_1 \equiv D_3} \qquad \frac{P \equiv P' \qquad D \equiv D'}{\ell : P, D \equiv \ell : P', D'} \qquad \frac{\ell \neq \ell'}{\ell : P, \ell' : P', D \equiv \ell' : P', \ell : P, D}$$

$$\frac{}{\cdot \equiv \cdot} \qquad \frac{E_1 \equiv E_2 \qquad E_2 \equiv E_3}{E_1 \equiv E_3} \qquad \frac{P \equiv P' \qquad E \equiv E'}{\ell : P, E \equiv \ell : P', E'} \qquad \frac{\ell \neq \ell'}{\ell : P, \ell' : P', E \equiv \ell' : P', \ell : P, E}$$

$$\frac{}{E, \ell : - \equiv E} \qquad \frac{A \equiv A' \qquad B \equiv B'}{A \twoheadrightarrow B \equiv A' \twoheadrightarrow B'} \qquad \frac{}{- \equiv -} \qquad \frac{}{\alpha \equiv \alpha} \qquad \frac{\mu \equiv \nu \qquad A \equiv B}{\mu A \equiv \nu B}$$

$$\frac{E \equiv F}{[E] \equiv [F]} \qquad \frac{L \equiv L' \qquad D \equiv D'}{\langle L | D \rangle \equiv \langle L' | D' \rangle} \qquad \frac{A \equiv A' \qquad B \equiv B'}{A \to B \equiv A' \to B'} \qquad \frac{A \equiv B}{\forall \alpha^K . A \equiv \forall \alpha^K . B}$$

$\boxed{P \leqslant P'}$ $\boxed{E \leqslant F}$ $\boxed{D \leqslant D'}$

$$\frac{}{P \leqslant P} \qquad \frac{}{- \leqslant P} \qquad \frac{}{\cdot \leqslant \cdot}$$

$$\frac{\begin{array}{cc} E_1 \equiv \ell : P_1, E_1' & E_2 \equiv \ell : P_2, E_2' \\ P_1 \leqslant P_2 & E_1' \leqslant E_2' \end{array}}{E_1 \leqslant E_2} \qquad \frac{\begin{array}{cc} D_1 \equiv \ell : P_1, D_1' & D_2 \equiv \ell : P_2, D_2' \\ P_1 \leqslant P_2 & D_1' \leqslant D_2' \end{array}}{D_1 \leqslant D_2}$$

Fig. 8. Type equivalence and sub-effecting for Met.

$$
\begin{array}{lrcl}
\text{Types} & A, B & ::= & \cdots \mid (A, B) \mid A + B \\
\text{Terms} & M, N & ::= & \cdots \mid (M, N) \mid \mathbf{inl}\ M \mid \mathbf{inr}\ M \mid \mathbf{case}_\nu\ V\ \mathbf{of}\ \{\overline{Q \mapsto M}\} \\
\text{Patterns} & Q & ::= & (x, y) \mid \mathbf{inl}\ x \mid \mathbf{inr}\ x \\
\text{Values} & V, W & ::= & \cdots \mid (V, W) \mid \mathbf{inl}\ V \mid \mathbf{inr}\ V
\end{array}
$$

T-Pair
$$\frac{\Gamma \vdash M : A \mathbin{@} E \qquad \Gamma \vdash N : B \mathbin{@} E}{\Gamma \vdash (M, N) : (A, B) \mathbin{@} E}$$

T-Inl
$$\frac{\Gamma \vdash M : A \mathbin{@} E}{\Gamma \vdash \mathbf{inl}\ M : A + B \mathbin{@} E}$$

T-Inr
$$\frac{\Gamma \vdash M : B \mathbin{@} E}{\Gamma \vdash \mathbf{inr}\ M : A + B \mathbin{@} E}$$

T-CrispPair
$$\frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F} \vdash V : (A, B) \mathbin{@} E \qquad \Gamma, x :_{\nu_F} A, y :_{\nu_F} B \vdash M : A' \mathbin{@} F}{\Gamma \vdash \mathbf{case}_\nu\ V\ \mathbf{of}\ (x, y) \mapsto M : A' \mathbin{@} F}$$

T-CrispSum
$$\frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F} \vdash V : A + B \mathbin{@} E \qquad \Gamma, x :_{\nu_F} A \vdash M_1 : A' \mathbin{@} F \qquad \Gamma, y :_{\nu_F} B \vdash M_2 : A' \mathbin{@} F}{\Gamma \vdash \mathbf{case}_\nu\ V\ \mathbf{of}\ \{\mathbf{inl}\ x \mapsto M_1, \mathbf{inr}\ y \mapsto M_2\} : A' \mathbin{@} F}$$

$$
\begin{array}{lrcl}
\text{Decorations} & \delta & ::= & \cdot \mid \mathbb{A} \mid \dagger \mid \mathbb{A}\dagger \\
\text{Terms} & M, N & ::= & \mathbf{handle}^\delta\ M\ \mathbf{with}\ H
\end{array}
$$

T-Handler$^{\mathbb{A}}$
$$\frac{D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{[D+E]_F} \vdash M : A \mathbin{@} D + E \qquad \Gamma, x : [D + E]A \vdash N : B \mathbin{@} F \qquad [\Gamma, p_i : A_i, r_i : [F](B_i \to B) \vdash N_i : B \mathbin{@} F]_i}{\Gamma \vdash \mathbf{handle}^{\mathbb{A}}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}$$

T-ShallowHandler
$$\frac{D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{\langle D \rangle} \vdash M : A \mathbin{@} D + F \qquad \Gamma, x : \langle D \rangle A \vdash N : B \mathbin{@} F \qquad [\Gamma, p_i : A_i, r_i : \langle D \rangle (B_i \to A) \vdash N_i : B \mathbin{@} F]_i}{\Gamma \vdash \mathbf{handle}^{\dagger}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}$$

T-ShallowHandler$^{\mathbb{A}}$
$$\frac{D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{[D+E]} \vdash M : A \mathbin{@} D + E \qquad \Gamma, x : [D + E]A \vdash N : B \mathbin{@} F \qquad [\Gamma, p_i : A_i, r_i : [D + E](B_i \to A) \vdash N_i : B \mathbin{@} F]_i}{\Gamma \vdash \mathbf{handle}^{\mathbb{A}\dagger}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}$$

Fig. 9. Syntax and typing rules for data types, absolute and shallow handlers in Met.

$$
\begin{array}{ll}
\text{Value normal forms} & U ::= \cdots \mid (U_1, U_2) \mid \textbf{inl}\ U \mid \textbf{inr}\ U \\
\text{Evaluation contexts} & \mathcal{E} ::= \cdots \mid (\mathcal{E}, N) \mid (U, \mathcal{E}) \mid \textbf{inl}\ \mathcal{E} \mid \textbf{inr}\ \mathcal{E} \mid \textbf{case}_\nu\ \mathcal{E}\ \textbf{of}\ \{\overline{Q \mapsto M}\} \\
& \quad \mid\ \textbf{handle}^\delta\ \mathcal{E}\ \textbf{with}\ H
\end{array}
$$

$$
\begin{array}{ll}
\text{E-CrispPair} & \textbf{case}_\mu\ (U_1, U_2)\ \textbf{of}\ (x, y) \mapsto N \rightsquigarrow N[U_1/x, U_2/y] \\
\text{E-CrispInl} & \textbf{case}_\mu\ \textbf{inl}\ U\ \textbf{of}\ \{\textbf{inl}\ x \mapsto N_1, \cdots\} \rightsquigarrow N_1[U/x] \\
\text{E-CrispInr} & \textbf{case}_\mu\ \textbf{inr}\ U\ \textbf{of}\ \{\textbf{inr}\ y \mapsto N_2, \cdots\} \rightsquigarrow N_2[U/y] \\
\text{E-Ret}^{\mathbb{A}} & \textbf{handle}\ U\ \textbf{with}\ H \rightsquigarrow N[(\textbf{mod}_{[D+E]}\ U)/x] \\
& \quad \text{where } (\textbf{return}\ x \mapsto N) \in H \\
\text{E-Op}^{\mathbb{A}} & \textbf{handle}^{\mathbb{A}}\ \mathcal{E}[\textbf{do}\ \ell\ U]\ \textbf{with}\ H \rightsquigarrow \\
& \quad N[U/p, (\textbf{mod}_{[F]}\ (\lambda y.\textbf{handle}^{\mathbb{A}}\ \mathcal{E}[y]\ \textbf{with}\ H))/r] \\
& \quad \text{where } 0\text{-free}(\ell, \mathcal{E}) \text{ and } (\ell\ p\ r \mapsto N) \in H \\
\text{E-Ret}^{\dagger} & \textbf{handle}^{\dagger}\ U\ \textbf{with}\ H \rightsquigarrow N[(\textbf{mod}_{\langle\!|D\rangle}\ U)/x] \\
& \quad \text{where } (\textbf{return}\ x \mapsto N) \in H \\
\text{E-Op}^{\dagger} & \textbf{handle}^{\dagger}\ \mathcal{E}[\textbf{do}\ \ell\ U]\ \textbf{with}\ H \rightsquigarrow N[U/p, (\lambda y.\mathcal{E}[y])/r] \\
& \quad \text{where } 0\text{-free}(\ell, \mathcal{E}) \text{ and } (\ell\ p\ r \mapsto N) \in H \\
\text{E-Ret}^{\mathbb{A}\dagger} & \textbf{handle}^{\mathbb{A}\dagger}\ U\ \textbf{with}\ H \rightsquigarrow N[(\textbf{mod}_{[D+E]}\ U)/x] \\
& \quad \text{where } (\textbf{return}\ x \mapsto N) \in H \\
\text{E-Op}^{\mathbb{A}\dagger} & \textbf{handle}^{\mathbb{A}\dagger}\ \mathcal{E}[\textbf{do}\ \ell\ U]\ \textbf{with}\ H \rightsquigarrow \\
& \quad N[U/p, (\textbf{mod}_{[D+E]}\ (\lambda y.\mathcal{E}[y]))/r] \\
& \quad \text{where } 0\text{-free}(\ell, \mathcal{E}) \text{ and } (\ell\ p\ r \mapsto N) \in H
\end{array}
$$

Fig. 10. Operational semantics for data types and more handlers in Met.

A.3 The Double Category of Effects

Fig. 11. 2-cells in a 2-category compared to 2-cells in a double category.

A double category extends a 2-category with an additional kind of morphisms. Alongside the regular morphisms, now called *horizontal* morphisms, there are also *vertical* morphisms that connect the objects of the 2-category. This makes it possible to generalise the 2-cells to transform arbitrary morphisms, whose source and target are connected by vertical morphisms. Figure 11 shows the differences between 2-cells in a 2-category and those in a double category using syntax of Met.

In Met, objects/modes are given by effect contexts, the horizontal morphisms by modalities, the vertical morphisms by the sub-effecting relation, and 2-cells by the modality transformations.

Now we show that it indeed has the structure of a double category.

Since the sub-effecting relation is a preorder, effect contexts (objects) E and sub-effecting (vertical morphisms) $E \leqslant F$ obviously form a category given by the poset.

We repeat the definition of modalities and modality composition from Section 3.3 here for easy reference. We directly define them directly in terms of morphisms between modes.

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

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

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 A.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. □

In Section 3, we only define the modality transformations of shape $\mu_F \Rightarrow \nu_F$ where the targets of μ and ν are required to be the same effect context F. This is enough for presenting the calculus, but we can further extend it to allow $\mu_F \Rightarrow \nu_{F'}$ where $F \leqslant F'$. This is used in the meta theory for Met such as the lock weakening lemma (Lemma A.11.3).

The extended modality transformation relation is defined by the transitive closure of the following rules. Compared to the definition in Section 3.3, the only new rule is MT-Mono.

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

MT-Upcast
$$\frac{D \leqslant D'}{\langle L|D\rangle_F \Rightarrow \langle L|D'\rangle_F}$$

MT-Expand
$$\frac{(F - L) \equiv \ell : P, E}{\langle \ell, L|D, \ell : P\rangle_F \Leftrightarrow \langle L|D\rangle_F}$$

MT-Mono
$$\frac{F \leqslant F'}{\mu_F \Rightarrow \mu_{F'}}$$

The following lemmas shows that the transformation $\mu_F \Rightarrow \nu_{F'}$ satisfies the requirement of being 2-cells in the double category of effects with well-defined vertical and horizontal composition.

Lemma A.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. To make proving easier, we give the resulting rules by taking the transitive closure.

$$\frac{\mu_{F'} : E' \to F' \qquad E \leqslant E' \qquad F \leqslant F'}{[E]_F \Rightarrow \mu_{F'}}$$

$$\frac{L = \mathsf{dom}(D) \qquad D_1 \leqslant D'_1 \qquad (F' - L_1) \equiv D, E \qquad F \leqslant F'}{\langle L_1 | D_1 \rangle_F \Rightarrow \langle L, L_1 | D'_1, D \rangle_{F'}}$$

$$\frac{L = \mathsf{dom}(D) \qquad D_1 \leqslant D'_1 \qquad (F' - L_1) \equiv D, E \qquad F \leqslant F'}{\langle L, L_1 | D_1, D \rangle_F \Rightarrow \langle L_1 | D'_1 \rangle_{F'}}$$

It is easy to see that sources and targets of morphisms increase. Vertical composition follows directly from the fact that we take the transitive closure. Horizontal compositions follows from case analysis on shapes of modalities being composed. □

More on Relationships between Met and Multimodal Type Theory. In addition to extending to a double category, Met also differs from MTT in the usage of morphism families. In types and terms we use μ, indexed families of morphisms between modes, instead of concrete morphisms μ_F. This is very useful to allow term variables to be used flexibly in different effect contexts larger than where they are defined. As a result, every type is always well-defined at any modes, which implies that we do not need to define the judgement $A \ @ \ E$ as in MTT. Moreover, one important benefit of having types well-defined at any modes is that type quantifiers do not need to carry the additional information about the modes at which the type variables can be used, greatly simplifying the type system. Otherwise, polymorphic types would have forms $\forall \alpha^{K \ @ \ E}.A$, where E indicates the mode of the type variable α.

In contexts, we still keep concrete morphisms μ_F, which makes the proof trees of terms much more structured than using morphism families.

A.4 Lemmas for Modes and Modalities

Beyond the structure and properties of double categories shown in Appendix A.3, we have some extra properties on modes and modalities in Met.

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 A.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. □

We prove the lemma on the equivalence between syntactic and semantic definition of modality transformation in Section 3.3. This lemma can be generalised to the general form of 2-cells in a double category $\mu_F \Rightarrow \nu_{F'}$ where $F \leqslant F'$.

Lemma 3.1 (Semantics of modality transformation). *We have $\mu_F \Rightarrow \nu_F$ if and only if $\mu(F') \leqslant \nu(F')$ for all F' with $F \leqslant F'$.*

Proof. From left to right, it is obvious that the semantics is preserved after taking the transitive closure. We only need to show the transformation given by each rule satisfies the semantics.

Case MT-Abs. Follow from Lemma A.3.

Case MT-Upcast. Since $D \leqslant D'$, we have $D + (F - L) \leqslant D' + (F - L)$ for any F.

Case MT-Expand. Since $(F - L) \equiv \ell : P, E$, for any $F \leqslant F'$ we have $(F' - L) \equiv \ell : P, E'$ for some E'. Both sides act on F' give $D, \ell : P, E'$.

From left to right, we need to show that for all pairs μ_F and ν_F satisfying the semantic definition, we have $\mu_F \Rightarrow \nu_F$ in the transitive closure of the three syntactic rules. This obviously holds for those transformation starting from absolute modalities. For those transformation starting from relative modalities, observe that they can only be transformed other relative modalities by the semantic definition. By taking the transitive closure of the last two rules, we have

$$\frac{L = \mathrm{dom}(D) \qquad D_1 \leqslant D'_1 \qquad (F - L_1) \equiv D, E}{\langle L_1 | D_1 \rangle_F \Rightarrow \langle L, L_1 | D'_1, D \rangle_F}$$

$$\frac{L = \mathrm{dom}(D) \qquad D_1 \leqslant D'_1 \qquad (F - L_1) \equiv D, E}{\langle L, L_1 | D_1, D \rangle_F \Rightarrow \langle L_1 | D'_1 \rangle_F}$$

Suppose $\langle L_1 | D_1 \rangle_F$ and $\langle L_2 | D_2 \rangle_F$ satisfies that $D_1 + (F' - L_1) \leqslant D_2 + (F' - L_2)$ (1) for all $F \leqslant F'$. Case analysis on the relationship between D_1 and D_2.

Case D_2 is longer than D_1. By (1) we have $D_2 \equiv D'_1, D$ for $D_1 \leqslant D'_1$. Let $L = \mathrm{dom}(D)$. Using proof by contradiction, we can show that $L_2 \equiv L, L_1$ and $(F - L_1) \equiv D, E$ for some E; otherwise, we can always properly set F' to violate (1) meanwhile satisfying $F \leqslant F'$. Thus, this case is covered by the first rule of the transitive closure.

Case D_1 is longer than D_2. We have $D_1 \equiv D'_2, D$ for $D'_2 \leqslant D_2$. Similar to the above case, using proof by contradiction we can show that it is covered by the second rule of the transitive closure.

□

Our proofs for type soundness of Met do not use ad-hoc case analysis on shapes of modalities or reply on any specific properties about the definition of composition and transformation (except for the parts about effect handlers since they specify the required modalities in the typing rules). As a result, it should be able to generalise our calculus and proofs to other mode theories satisfying certain extra properties. We state some properties of the mode theory as the following lemmas for easier reference in proofs. Most of them directly follow from the definition.

Lemma A.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 A.2 □

Lemma A.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 A.2 □

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

Proof. Follow from Lemma 3.1 □

Lemma A.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. □

A.5 Lemmas for Met

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

Lemma A.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 $\vdash U : \forall \alpha.A \; @ \; E$, then U is of shape $\Lambda \alpha.V$.*
4. *If $\vdash U : (A, B) \; @ \; E$, then U is of shape (U_1, U_2).*
5. *If $\vdash U : A + B \; @ \; E$, then U is either of shape $\mathbf{inl}\; U'$ or of shape $\mathbf{inr}\; U'$.*

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 $\llparenthesis \Gamma \rrparenthesis_{F'} \; @ \; F'$.

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

The have the following lemma showing that the index update operation preserves the locks$(-)$ operation except for updating the index.

Lemma A.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 A.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 A.9 and Lemma A.7. □

We have the following structural lemmas.

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

1. *Variable weakening.*

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

2. *Variable swapping.*

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

3. *Lock weakening.*

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

4. *Type variable weakening.*

$$\frac{\Gamma, \Gamma' \vdash M : B \mathrel{@} E}{\Gamma, \alpha : K, \Gamma' \vdash M : B \mathrel{@} E}$$

5. *Type variable swapping.*

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

Proof. 1, 2, 4, and 5 follow from straightforward induction on the typing derivation. For 3, we also proceed by induction on the typing derivation. The most interesting case is T-Var. Other cases mostly follow from IHs.

Case

T-Var
$$\frac{\nu'_{F_1} = \mathsf{locks}(\Gamma_2) : E \to F_1 \qquad \mu'_{F_1} \Rightarrow \nu'_{F_1}\ (1) \text{ or } \Gamma \vdash A : \mathsf{Abs}}{\Gamma_1, x :_{\mu'_{F_1}}, \Gamma_2 \vdash x : A \mathrel{@} E}$$

Trivial when A is pure. Otherwise, case analysis on where the lock weakening happens.

Case Γ. Supposing $\Gamma_1 = \Gamma, 🔒_{\mu_F}, \Gamma_0$ and after lock weakening we have $\Gamma, 🔒_{\nu_F}, \Gamma'_0, x :_{\mu'_{F'_1}}, \Gamma'_2$ where $\Gamma'_2 = (\!|\Gamma_2|\!)_{F'_1} : E' \to F'_1$ and $\Gamma'_0 = (\!|\Gamma_0|\!)_{F'} : F'_1 \to F'$. By Lemma A.9 on Γ_0, $F \leqslant F'$, and Lemma A.3, we have $F_1 \leqslant F'_1$. Then by (1) and Lemma A.6, we have $\mu'_{F'_1} \Rightarrow \nu'_{F'_1}$. Then by Lemma A.9 we have $\nu'_{F'_1} = \mathsf{locks}(\Gamma'_2)$. Finally by T-Var we have

$$\Gamma, 🔒_{\nu_F}, \Gamma'_0, x :_{\mu'_{F'_1}}, \Gamma'_2 \vdash x : A \mathrel{@} E'$$

Case Γ_2. Suppose $\Gamma_2 = \Gamma_0, 🔒_{\mu_F}, \Gamma'$. is weakened to $\Gamma'_2 = \Gamma_0, 🔒_{\nu_F}, (\!|\Gamma'|\!)_{F'}$. By Corollary A.10 we have $\mathsf{locks}(\Gamma') \Rightarrow \mathsf{locks}((\!|\Gamma'|\!)_{F'})$. Then by Lemma A.5 we have we have $\mathsf{locks}(\Gamma_2) \Rightarrow \mathsf{locks}(\Gamma'_2)$. By Lemma A.4 and (1), we have $\mu'_{F_1} \Rightarrow \mathsf{locks}(\Gamma'_2)$. Finally by T-Var we have

$$\Gamma, x :_{\mu'_{F_1}}, \Gamma'_2 \vdash x : A \mathrel{@} E'$$

Case

T-Mod
$$\frac{\mu'_E : F_1 \to E \qquad \Gamma, 🔒_{\mu_F}, \Gamma', 🔒_{\mu'_E} \vdash V : A \mathrel{@} F_1\ (1)}{\Gamma, 🔒_{\mu_F}, \Gamma' \vdash \mathbf{mod}_{\mu'}\ V : \mu' A \mathrel{@} E}$$

We have

$$(\!|\Gamma', 🔒_{\mu'_E}|\!)_{F'} = (\!|\Gamma'|\!)_{F'}, (\!|🔒_{\mu'_E}|\!)_{E'} = (\!|\Gamma'|\!)_{F'}, 🔒_{\mu'_{E'}}.$$

Supposing $\mu'_{E'} : F'_1 \to E'$, by $\mathsf{locks}((\!|\Gamma'|\!)_{F'}, 🔒_{\mu'_{E'}}) : F'_1 \to F'$ and IH on (1), we have

$$\Gamma, 🔒_{\mu_F}, (\!|\Gamma'|\!)_{F'}, 🔒_{\mu'_{E'}} \vdash V : A \mathrel{@} F'_1.$$

Then by T-Mod we have

$$\Gamma, 🔒_{\mu_F}, (\!|\Gamma'|\!)_{F'} \vdash \mathbf{mod}_{\mu'}\ V : \mu' A \mathrel{@} E'.$$

Case

T-LetMod

$$\frac{\Gamma, \text{🔒}_{\mu_F}, \Gamma', \text{🔒}_{\nu'_E} \vdash V : \mu' A \mathbin{@} F_1 \ (1) \qquad \nu'_E : F_1 \to E \qquad \Gamma, \text{🔒}_{\mu_F}, \Gamma', x :_{\nu'_E \circ \mu'_{F_1}} A \vdash M : B \mathbin{@} E \ (2)}{\Gamma, \text{🔒}_{\mu_F}, \Gamma' \vdash \mathbf{let}_{\nu'}\ \mathbf{mod}_{\mu'}\ x = V\ \mathbf{in}\ M : B \mathbin{@} E}$$

By IH on (1), we have

$$\Gamma, \text{🔒}_{\nu_F}, (\!|\Gamma'|\!)_{F'}, \text{🔒}_{\nu'_{E'}} \vdash V : \mu' A \mathbin{@} F'_1$$

where $\nu'_{E'} : F'_1 \to E'$. By IH on (2), we have

$$\Gamma, \text{🔒}_{\nu_F}, (\!|\Gamma'|\!)_{F'}, x :_{\nu'_{E'} \circ \mu'_{F'_1}} A \vdash M : B \mathbin{@} E'.$$

Then by T-LetMod, we have

$$\Gamma, \text{🔒}_{\mu_F}, (\!|\Gamma'|\!)_{F'} \vdash \mathbf{let}_{\nu'}\ \mathbf{mod}_{\mu'}\ x = V\ \mathbf{in}\ M : B \mathbin{@} E'$$

Case

T-LetMod'

$$\frac{\Gamma, \text{🔒}_{\mu_F}, \Gamma', \text{🔒}_{\nu'_E}, \overline{\alpha : K} \vdash V : \mu' A \mathbin{@} F_1 \ (1) \qquad \nu'_E : F_1 \to E \qquad \Gamma, \text{🔒}_{\mu_F}, \Gamma', x :_{\nu'_E \circ \mu'_{F_1}} \forall \overline{\alpha^K}.A \vdash M : B \mathbin{@} E \ (2)}{\Gamma, \text{🔒}_{\mu_F}, \Gamma' \vdash \mathbf{let}_{\nu'}\ \mathbf{mod}_{\mu'}\ \Lambda\overline{\alpha^K}.x = V\ \mathbf{in}\ M : B \mathbin{@} E}$$

Similar to the case for T-LetMod. BY IH on (1), we have

$$\Gamma, \text{🔒}_{\nu_F}, (\!|\Gamma'|\!)_{F'}, \text{🔒}_{\nu'_{E'}}, \overline{\alpha : K} \vdash V : \mu' A \mathbin{@} F'_1$$

where $\nu'_{E'} : F'_1 \to E'$. By IH on (2), we have

$$\Gamma, \text{🔒}_{\nu_F}, (\!|\Gamma'|\!)_{F'}, x :_{\nu'_{E'} \circ \mu'_{F'_1}} \forall \overline{\alpha^K}.A \vdash M : B \mathbin{@} E'.$$

Then by T-LetMod', we have

$$\Gamma, \text{🔒}_{\nu_F}, (\!|\Gamma'|\!)_{F'} \vdash \mathbf{let}_{\nu'}\ \mathbf{mod}_{\mu'}\ \Lambda\overline{\alpha^K}.x = V\ \mathbf{in}\ M : B \mathbin{@} E'$$

Case T-TAbs, T-Abs, T-TApp, T-App, T-Do, T-Mask, T-Handler, and extensions. Follow from IH. Similar to the two cases T-Mod and T-LetMod we have shown.

□

As a corollary of Lemma A.11.3, the following sub-effecting rule is admissible.

Corollary A.12 (Sub-effecting). *The following rule is admissible.*

$$\frac{\Gamma \vdash M : A \mathbin{@} E \qquad \mathsf{locks}(\Gamma) : E \to F \qquad F \leqslant F' \qquad \mathsf{locks}((\!|\Gamma|\!)_{F'}) : E' \to F'}{(\!|\Gamma|\!)_{F'} \vdash M : A \mathbin{@} E'}$$

Proof. Follow from Lemma A.11.3 by adding the lock $\text{🔒}_{[F]}$ to the left of Γ in $\Gamma \vdash M : A \mathbin{@} E$, and weaken it to $\text{🔒}_{[F']}$. Note that typing judgements still hold after adding a lock to or removing a lock from the left of the context, as long as the new contexts are still well-defined. □

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

Lemma A.13 (Pure Promotion). *The following promotion rule is admissible.*

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

Proof. By induction on the typing derivation of V.

Case T-Var. Trivial.

Case

$$\frac{\mu_E : F_1 \to E \qquad \Gamma_1, \Gamma, \text{🔒}_{\mu_E} \vdash V : A \mathbin{@} F_1 \ (1)}{\Gamma_1, \Gamma \vdash \mathbf{mod}_\mu\, V : \mu A \mathbin{@} E} \ \text{T-Mod}$$

Case analysis on the shape of μ.

Case μ is relative. By kinding, A is also pure. By IH on (1), we have

$$\Gamma_1, \Gamma', \text{🔒}_{\mu_{E'}} \vdash V : A \mathbin{@} F_1'$$

where $\mu_{E'} : F_1' \to E'$. Then by T-Mod we have

$$\Gamma_1, \Gamma' \vdash \mathbf{mod}_\mu\, V : \mu A \mathbin{@} E'$$

Case μ is absolute. We have $\mu = [F_1]$ and $\mathsf{locks}(\Gamma', \text{🔒}_{\mu_{E'}}) = [F_1]_F = \mathsf{locks}(\Gamma, \text{🔒}_{\mu_E})$. Thus, replacing the context $(\Gamma, \text{🔒}_{\mu_E})$ with $(\Gamma', \text{🔒}_{\mu_{E'}})$ in (1) does not influence all usages of T-Var in the derivation tree of (1). We have

$$\Gamma_1, \Gamma', \text{🔒}_{\mu_{E'}} \vdash V : A \mathbin{@} F_1$$

Then by T-Mod we have

$$\Gamma_1, \Gamma' \vdash \mathbf{mod}_\mu\, V : \mu A \mathbin{@} E'$$

Case T-TAbs. Follow from IH and Lemma A.11.5.

Case T-Abs. Impossible since function types are impure.

Case Data Types. Follow from IHs.

□

Lemma A.14 (Substitution). *The following substitution rules are admissible.*

1. *Preservation of kinds under type substitution.*

$$\frac{\Gamma \vdash A : K \qquad \Gamma, \alpha : K, \Gamma' \vdash B : K'}{\Gamma, \Gamma' \vdash B[A/\alpha] : K'}$$

2. *Preservation of types under type substitution.*

$$\frac{\Gamma \vdash A : K \qquad \Gamma, \alpha : K, \Gamma' \vdash M : B \mathbin{@} E}{\Gamma, \Gamma' \vdash M[A/\alpha] : B[A/\alpha] \mathbin{@} E}$$

3. *Preservation of types under value substitution.*

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

Proof.

1. By straightforward induction on the kinding derivation.
2. By straightforward induction on the typing derivation of M.
3. By induction on the typing derivation of M. Trivial when variable x is not used. In the following induction we always assume x is used.

Case

$$\frac{\nu_F = \text{locks}(\Gamma') : E \to F \qquad \mu_F \Rightarrow \nu_F \ (1) \text{ or } \Gamma \vdash A : \mathsf{Abs}}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash x : A \ @ \ E}\ \textsc{T-Var}$$

Case analysis on the purity of A

Case Impure. By $\Gamma, \text{🔒}_{\mu_F} \vdash V : A \ @ \ F'$, (1), and Lemma A.11.3, we have

$$\Gamma, \text{🔒}_{\nu_F} \vdash V : A \ @ \ E.$$

Then, by context equivalence, Lemma A.11.1, and Lemma A.11.4, we have

$$\Gamma, \Gamma' \vdash V : A \ @ \ E.$$

Case Pure. By $\Gamma, \text{🔒}_{\mu_F} \vdash V : A \ @ \ F'$ and Lemma A.13, we have

$$\Gamma, \Gamma' \vdash V : A \ @ \ E.$$

Case

$$\frac{\mu'_E : F_1 \to E \qquad \Gamma, x :_{\mu_F} A, \Gamma', \text{🔒}_{\mu'_E} \vdash W : B \ @ \ F_1 \ (1)}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash \mathbf{mod}_{\mu'} \ W : \mu' B \ @ \ E}\ \textsc{T-Mod}$$

By IH on (1) we have

$$\Gamma, \Gamma', \text{🔒}_{\mu'_E} \vdash W[V/x] : B \ @ \ F_1.$$

Then by T-Mod we have

$$\Gamma, \Gamma' \vdash (\mathbf{mod}_{\mu'} \ W)[V/x] : \mu' B \ @ \ E$$

Case

$$\frac{\nu_E : F_1 \to E \qquad \Gamma, x :_{\mu_F} A, \Gamma', \text{🔒}_{\nu_E} \vdash W : \mu' A' \ @ \ F_1 \ (1) \qquad \Gamma, x :_{\mu_F} A, \Gamma', y :_{\nu_E \circ \mu'_{F_1}} A' \vdash M : B \ @ \ E \ (2)}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash \mathbf{let}_\nu \ \mathbf{mod}_{\mu'} \ y = W \ \mathbf{in} \ M : B \ @ \ E}\ \textsc{T-LetMod}$$

By IH on (1), we have

$$\Gamma, \Gamma', \text{🔒}_{\nu_E} \vdash W[V/x] : \mu' A' \ @ \ F_1.$$

By IH on (2), we have

$$\Gamma, \Gamma', y :_{\nu_E \circ \mu'_{F_1}} A' \vdash M[V/x] : B \ @ \ E.$$

Then by T-LetMod, we have

$$\Gamma, \Gamma' \vdash (\mathbf{let}_\nu \ \mathbf{mod}_{\mu'} \ y = W \ \mathbf{in} \ M)[V/x] : B \ @ \ E$$

Case

$$\frac{\nu_E : F_1 \to E \qquad \Gamma, x :_{\mu_F} A, \Gamma', \text{🔒}_{\nu_E}, \overline{\alpha : K} \vdash V : \mu' A' \ @ \ F_1 \ (1) \qquad \Gamma, x :_{\mu_F} A, \Gamma', y :_{\nu_E \circ \mu'_{F_1}} \forall \overline{\alpha^K}.A' \vdash M : B \ @ \ E \ (2)}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash \mathbf{let}_\nu \ \mathbf{mod}_{\mu'} \ \Lambda \overline{\alpha^K}.y = V \ \mathbf{in} \ M : B \ @ \ E}\ \textsc{T-LetMod'}$$

Similar to the case for T-LetMod. Our goal follows from IH on (1), IH on (2), and T-LetMod'.

Case

$$\text{T-Mask}\quad \frac{\Gamma, x :_{\mu_F} A, \Gamma', \text{🔒}_{\langle L|\rangle_E} \vdash M : B \mathbin{@} E - L \ (1)}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash \mathbf{mask}_L\ M : \langle L|\rangle B \mathbin{@} E}$$

By IH on (1) we have

$$\Gamma, \Gamma', \text{🔒}_{\langle L|\rangle_E} \vdash M[V/x] : B \mathbin{@} E - L.$$

Then by T-Mask we have

$$\Gamma, \Gamma' \vdash (\mathbf{mask}_L\ M)[V/x] : \langle L|\rangle B \mathbin{@} E$$

Case

$$\text{T-Handler}\quad \frac{\begin{array}{c} D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, x :_{\mu_F} A, \Gamma', \text{🔒}_{\langle|D\rangle_E} \vdash M : A_0 \mathbin{@} D + E \ (1) \\ \Gamma, x :_{\mu_F} A, \Gamma', y : \langle|D\rangle A_0 \vdash N : B \mathbin{@} E \ (2) \\ [\Gamma, x :_{\mu_F} A, \Gamma', p_i : A_i, r_i : B_i \to B \vdash N_i : B \mathbin{@} E \ (3)]_i \end{array}}{\Gamma, x :_{\mu_F} A, \Gamma' \vdash \mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ y \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} E}$$

Follow from IH on (1),(2),(3), and reapplying T-Handler.

Case T-TAbs, T-TApp, T-Abs, T-App, T-Do. Follow from IH.

Case Extensions. Follow from IH.

□

A.6 Progress

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

Proof. By induction on the typing derivation $\vdash M : A \mathbin{@} E$. The most non-trivial cases are T-Mask and T-Handler. Other cases follow from IHs and reduction rules, using Lemma A.8.

Case M is in a value normal form U. Trivial. Base case.

Case T-Do. Trivial. Base case.

Case T-Mod. $\mathbf{mod}_\mu\ V$. By IH on V.

Case T-Letmod. $\mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = V\ \mathbf{in}\ N$. By IH on V, if V is reducible then M is reducible; otherwise, V is in a value normal form, then by Lemma A.8 we have that M is reducible by E-Letmod.

Case T-Letmod'. Similar to the case for T-Letmod.

Case T-TApp. $M\,A$. Similarly by IH on M, Lemma A.8, and E-TApp.

Case T-App. $M\,N$. Similarly by IH on M and N, Lemma A.8, and E-App.

Case T-Mask. $\mathbf{mask}^E\ M$. By IH on M.

- Case M is reducible. Trivial.
- Case M is in a value normal form. By E-Mask.
- Case $M = \mathcal{E}[\mathbf{do}\ \ell\ U]$ with n–free$(\ell, \mathcal{E})$. The whole term is in a normal form.

Case Handlers. The general form is $\mathbf{handle}^\delta\ M\ \mathbf{with}\ H$. By IH on M.

- Case M is reducible. Trivial.
- Case M is in a value normal form. By E-Ret.
- Case $M = \mathcal{E}[\mathbf{do}\ \ell\ U]$ with n–free$(\ell, \mathcal{E})$. If $n = 0$ and $\ell \in H$, then reducible by E-Op. Otherwise, the whole term is in a normal form.

Case T-BoxAbs. $\mathbf{mod}_{[]}\ M$. If $M \rightsquigarrow N$, follow by IH on M. Otherwise, M must be in a value normal form because the T-BoxAbs requires M to have the empty effect. In this case, $\mathbf{mod}_{[]}\ M$ is also in a value normal form.

Case Data Types. Similar to other cases.

□

A.7 Subject Reduction

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

Proof. By induction on the typing derivation $\Gamma \vdash M : A \mathbin{@} E$.

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

Case

$$\textsc{T-Mod}\quad \frac{\mu_F : E \to F \qquad \Gamma, \text{🔒}_{\mu_F} \vdash V : A \mathbin{@} E\ (1)}{\Gamma \vdash \mathbf{mod}_\mu\ V : \mu A \mathbin{@} F}$$

The only way to reduce is by E-Lift and $V \rightsquigarrow W$. IH on (1) gives

$$\Gamma, \text{🔒}_{\mu_F} \vdash W : A \mathbin{@} E.$$

Then by T-Mod we have

$$\Gamma \vdash \mathbf{mod}_\mu\ W : \mu A \mathbin{@} F.$$

Case

$$\textsc{T-Letmod}\quad \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F} \vdash V : \mu A \mathbin{@} E\ (1) \qquad \Gamma, x :_{\nu_F \circ \mu_E} A \vdash M : B \mathbin{@} F\ (2)}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = V\ \mathbf{in}\ M : B \mathbin{@} F}$$

By case analysis on the reduction.

Case E-Lift with $V \rightsquigarrow W$. By IH on (1) and reapplying T-Letmod.

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

$$\mathbf{let}_\nu\ \mathbf{mod}_\mu\ x = \mathbf{mod}_\mu\ U\ \mathbf{in}\ M \rightsquigarrow M[U/x].$$

Inversion on (1) gives

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

where $\mu_E : E' \to E$. By context equivalence, we have

$$\Gamma, \text{🔒}_{\nu_F \circ \mu_E} \vdash U : A \mathbin{@} E'$$

where $\nu_F \circ \mu_E : E' \to F$. By Lemma A.14.3 and (2), we have

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

Case

$$\textsc{T-Letmod'}\quad \frac{\nu_F : E \to F \qquad \Gamma, \text{🔒}_{\nu_F}, \overline{\alpha : K} \vdash V : \mu A \mathbin{@} E\ (1) \qquad \Gamma, x :_{\nu_F \circ \mu_E} \forall \overline{\alpha^K}.A \vdash M : B \mathbin{@} F\ (2)}{\Gamma \vdash \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda \overline{\alpha^K}.x = V\ \mathbf{in}\ M : B \mathbin{@} F}$$

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

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

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

$$\mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda\overline{\alpha^K}.x = \mathbf{mod}_\mu\, U\ \mathbf{in}\ M \rightsquigarrow M[(\forall\overline{\alpha^K}.U)/x].$$

Inversion on (1) gives

$$\Gamma, \text{🔒}_{\nu_F}, \overline{\alpha : K}, \text{🔒}_{\mu_E} \vdash U : \mu A \ @\ E'.$$

where $\mu_E : E' \to E$. By Lemma A.11.5 we have

$$\Gamma, \text{🔒}_{\nu_F}, \text{🔒}_{\mu_E}, \overline{\alpha : K} \vdash U : A \ @\ E'.$$

By context equivalence, we have

$$\Gamma, \text{🔒}_{\nu_F \circ \mu_E}, \overline{\alpha : K} \vdash U : A \ @\ E'.$$

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

$$\Gamma, \text{🔒}_{\nu_F \circ \mu_E} \vdash \Lambda\overline{\alpha^K}.U : \forall\overline{\alpha^K}.A \ @\ E'.$$

By Lemma A.14.3 and (2), we have

$$\Gamma \vdash M[U/x] : B \ @\ F.$$

Case T-TAbs,T-Abs. Impossible as there is no further reduction.

Case

$$\textsc{T-TApp}\ \dfrac{\Gamma \vdash M : \forall\alpha^K.B \ @\ E\ (1) \qquad \Gamma \vdash A : K\ (2)}{\Gamma \vdash M\,A : B[A/\alpha] \ @\ E}$$

By case analysis on the reduction.

Case E-Lift with $M \rightsquigarrow N$. By IH on (1) and reapplying T-TApp.

Case E-TApp. We have $M = \Lambda\alpha^K.V$ and

$$(\Lambda\alpha^K.V)\,A \rightsquigarrow V[A/\alpha].$$

Inversion on (1) gives

$$\Gamma, \alpha : K \vdash V : B \ @\ E.$$

Then by Lemma A.14.2 on (2), we have

$$\Gamma \vdash V[A/\alpha] : B[A/\alpha] \ @\ E.$$

Case

$$\textsc{T-App}\ \dfrac{\Gamma \vdash M : A \to B \ @\ E\ (1) \qquad \Gamma \vdash N : A \ @\ E\ (2)}{\Gamma \vdash M\,N : B \ @\ E}$$

By case analysis on the reduction.

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

Case E-Lift with $N \rightsquigarrow N'$. By IH on (2) and reapplying T-App.

Case E-App. We have $M = \lambda x^A.M'$, $N = U$, and

$$M\,N \rightsquigarrow M'[U/x].$$

Inversion on (1) gives

$$\Gamma, x : A \vdash M' : B \ @\ E.$$

Then by Lemma A.14.3 we have

$$\Gamma \vdash M'[U/x] : B \ @\ E.$$

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

Case

$$\text{T-Mask}\ \frac{\Gamma, \blacksquare_{\langle L|\rangle_F} \vdash M : A \mathbin{@} F - L\ (1)}{\Gamma \vdash \mathbf{mask}_L\ M : \langle L|\rangle A \mathbin{@} F}$$

By case analysis on the reduction.

Case E-Lift with $M \rightsquigarrow N$. By IH on (1) and reapplying T-Mask.

Case E-Mask. We have $M = U$ and

$$\mathbf{mask}_L\ U \rightsquigarrow \mathbf{mod}_{\langle L|\rangle}\ U.$$

By $\langle L|\rangle_F : F - L \to F$ and T-Mod, we have

$$\Gamma \vdash \mathbf{mod}_{\langle L|\rangle}\ U : \langle L|\rangle A \mathbin{@} F.$$

Case

$$\text{T-Handler}\ \frac{\begin{array}{c} H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\ D = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \blacksquare_{\langle|D\rangle_F} \vdash M : A \mathbin{@} D + F\ (1) \\ \Gamma, x : \langle|D\rangle A \vdash N : B \mathbin{@} F\ (2) \qquad [\Gamma, p_i : A_i, r_i : B_i \to B \vdash N_i : B \mathbin{@} F\ (3)]_i \end{array}}{\Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : B \mathbin{@} F}$$

By case analysis on the reduction.

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

Case E-Ret. We have $M = U$ and

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

By (1), $\langle D|\rangle_F : F \to D + F$, and T-Mod, we have

$$\Gamma \vdash \mathbf{mod}_{\langle|D\rangle}\ U : A \mathbin{@} F.$$

Then by (2) and Lemma A.14.3 we have

$$\Gamma \vdash N[(\mathbf{mod}_{\langle|D\rangle}\ U)/x] : B \mathbin{@} F.$$

Case E-Op. We have $M = \mathcal{E}[\mathbf{do}\ \ell_j\ U]$, $0\text{-free}(\ell_j, \mathcal{E})$, $\ell_j\ p_j\ r_j \mapsto N_j$, and

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

Since D is well-kinded, A_j and B_j are pure. By inversion on $\mathbf{do}\ \ell_j\ U$ we have

$$\Gamma, \blacksquare_{\langle|D\rangle_F} \vdash U : A_j \mathbin{@} D + F.$$

By A_j is pure and Lemma A.13, we have

$$\Gamma, \blacksquare_{\langle|D\rangle_F}, \blacksquare_{\langle L|\rangle_{D+F}} \vdash U : A_j \mathbin{@} F$$

where $L = \text{dom}(D)$. By context equivalence, we have

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

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

$$\Gamma, y : B_j, \blacksquare_{\langle|D\rangle_F} \vdash \mathcal{E}[y] : A \mathbin{@} D + F.$$

Then by T-Handler and T-Abs we have

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

Finally, by (3), (4), (5), and Lemma A.14.3 we have

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

Case

T-Handle$^{\mathbb{A}}$

$$\frac{\begin{array}{c} H = \{\textbf{return } x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\ L = \{\ell_i\}_i \qquad E = \{\ell_i : A_i \twoheadrightarrow B_i\}_i \qquad \Gamma, \text{🔒}_{[D+E]_F} \vdash M : A \,@\, D + E\ (1) \\ \Gamma, x : [D+E]A \vdash N : B \,@\, F\ (2) \qquad [\Gamma, p_i : A_i, r_i : [F](B_i \to B) \vdash N_i : B \,@\, F\ (3)]_i \end{array}}{\Gamma \vdash \textbf{handle}^{\mathbb{A}}\ M \textbf{ with } H : B \,@\, F}$$

By case analysis on the reduction.

Case E-Lift with $M \rightsquigarrow M'$. By IHs and reapplying T-Handle$^{\mathbb{A}}$.

Case E-Ret$^{\mathbb{A}}$. We have $M = U$ and

$$\textbf{handle } M \textbf{ with } H \rightsquigarrow N[(\textbf{mod}_{[D+E]} U)/x].$$

By (1), $[D+F]_F : D + E \to F$, and T-Mod, we have

$$\Gamma \vdash \textbf{mod}_{[D+E]}\ U : [D+E]A \,@\, F.$$

Then by (2) and Lemma A.14.3 we have

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

Case E-Op$^{\mathbb{A}}$. We have $M = \mathcal{E}[\textbf{do } \ell_j\ U]$, $0\text{–free}(\ell_j, \mathcal{E})$, $\ell_j\ p_j\ r_j \mapsto N_j$, and

$$\textbf{handle}^{\mathbb{A}}\ M \textbf{ with } H \rightsquigarrow N_j[U/p, (\textbf{mod}_{[E]}\ (\lambda y.\textbf{handle}^{\mathbb{A}}\ \mathcal{E}[y] \textbf{ with } H))/r].$$

Since D is well-kinded, A_j and B_j are pure. By inversion on $\textbf{do } \ell_j\ U$, we have

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

By A_j is pure and Lemma A.13, we have

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

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

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

By $[F]_F \circ [D+E]_F = [D+E]_F$ and context equivalence, we have

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

Since B_j is pure, we can swap $y : B_j$ with $\text{🔒}_{[F]_F}$ and derive

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

By T-Handler$^{\mathbb{A}}$, we have

$$\Gamma, \text{🔒}_{[F]_F}, y : B_j \vdash \textbf{handle}^{\mathbb{A}}\ \mathcal{E}[y] \textbf{ with } H : B \,@\, E.$$

Then by T-Abs and T-Mod we have

$$\Gamma \vdash \textbf{mod}_{[F]}\ (\lambda y.\textbf{handle}^{\mathbb{A}}\ \mathcal{E}[y] \textbf{ with } H) : [F](B_j \to B) \,@\, F\ (5).$$

Finally, by (3), (4), (5), and Lemma A.14.3 we have

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

Case Shallow handlers. Similar to the cases of deep handlers.

Case Data Types. Nothing more special than the cases we have already shown. Introduction rules follows from IHs and reapplying the same typing rules. Elimination rules require to additionally consider their corresponding reduction rules.

□

A.8 Proof of Encoding

We prove the encoding from F^1_{eff} into Met in Section 5.

Definition 5.1 (Well-scoped). A typing judgement $\Gamma_1, x :_\varepsilon A, \Gamma_2 \vdash M : B \,!\, E$ is *well-scoped for x* if either $x \notin \mathsf{fv}(M)$ or $\blacklozenge^{\Lambda}_F \notin \Gamma_2$ or $A = \forall.A'$. A typing judgement $\Gamma \vdash M : A \,!\, E$ is *well-scoped* if it is well-scoped for all $x \in \Gamma$.

Lemma A.15 (Well-scopedness of Derivation Trees). *If the judgement at the bottom of a derivation tree is well-scoped, then every judgement in the derivation tree is well-scoped.*

Proof. Assume the contrary. Let $\Gamma_1, x :_\varepsilon A, \Gamma_2 \vdash M : B \,!\, E$ be the top-most judgement in the derivation tree with $x \in \mathsf{fv}(M)$ and $\blacklozenge^{\Lambda}_F \in \Gamma_2$ and $A \neq \forall.A'$. By case analysis on whether $\blacklozenge^{\Lambda}_F \in \Gamma_2$ was introduced in the derivation tree.

Case not introduced in the derivation tree: Then the judgement at the bottom of the derivation tree must contain both the marker and x and is not well-scoped for x. Contradiction.

Case introduced in the derivation tree: since we chose the top-most judgement, the judgement must have introduced the marker by an application of the R-EAbs rule. Let ε' be the effect variable introduced at this judgement. Then $\varepsilon \neq \varepsilon'$ by the side-condition of the R-EAbs rule. We have that ε is the ambient effect at the R-Var rule where x is used as a free variable, since we chose the top-most judgement. By the side-condition of the R-Var rule, then $\varepsilon = \varepsilon'$ or $A = \forall.A'$. Contradiction.

□

In the special case we consider there are no absent signatures. This implies that submoding on effects can only add labels to the end. Furthermore, all labels are drawn from a global environment and thus have the same signatures. This allows us to freely permute them in the effect row. In this case, we can strengthen the statement to the following:

Corollary A.16 (Transformation from Index). *If $\langle L_1|D_1\rangle(F) \leqslant \langle L_2|D_2\rangle(F)$ and $L_1 \leqslant F$ and $L_2 \leqslant F$ and $L_1 \bowtie D_1 = L_2 \bowtie D_2$, then $\langle L_1|D_1\rangle_F \Rightarrow \langle L_2|D_2\rangle_F$.*

Proof. We show that for all F' with $F \leqslant F'$, we have $\langle L_1|D_1\rangle(F') \leqslant \langle L_2|D_2\rangle(F')$. Since all signatures are present in F, we have that $F' = F + \bar{l}$ for some collection of labels with signatures $\bar{l}$. Then we use that $L_1 \leqslant F$:

$$\begin{aligned}\langle L_1|D_1\rangle(F') &= \langle L_1|D_1\rangle(F + \bar{l})\\ &= D_1 + ((F + \bar{l}) - L_1)\\ &= D_1 + ((F - L_1) + \bar{l})\\ &= \langle L_1|D_1\rangle(F) + \bar{l}\end{aligned}$$

and the same for $\langle L_2|D_2\rangle(F')$. Since $\langle L_1|D_1\rangle(F) \leqslant \langle L_2|D_2\rangle(F)$ and we can freely permute labels, we have that $(\langle L_1|D_1\rangle(F) + \bar{l}) \leqslant (\langle L_2|D_2\rangle(F) + \bar{l})$. □

The condition that $L_1 \bowtie D_1 = L_2 \bowtie D_2$ can be checked easily, where for the composition of modalities we use the fact that for $\langle L|D\rangle = \langle L_1|D_1\rangle \circ \langle L_2|D_2\rangle$, we have $L \bowtie D = (L_1, L_2) \bowtie (D_1, D_2)$.

Lemma A.17 (First Modality Transformation). *For all* E_1, E_2, E_3*:*

$$(\langle E_1 - E_2 | E_2 - E_1\rangle \circ \langle E_2 - E_3 | E_3 - E_2\rangle)_{E_1} \Leftrightarrow \langle E_1 - E_3 | E_3 - E_1\rangle_{E_1}$$

Proof. We can use Corollary A.16 since $(E_1 - E_3) \leqslant E_1$ and $(E_1 - E_2) + L \leqslant E_1$ where $(L, D) = (E_2 - E_3) \bowtie (E_2 - E_1)$. We have:

$$\begin{aligned}\langle E_1 - E_3 | E_3 - E_1\rangle(E_1) &= (E_3 - E_1) + (E_1 - (E_1 - E_3)) \\ &= (E_3 - E_1) + (E_1 \cap E_3) \\ &= E_3\end{aligned}$$

and using this calculation:

$$\begin{aligned}\langle E_1 - E_2 | E_2 - E_1\rangle \circ \langle E_2 - E_3 | E_3 - E_2\rangle(E_1) &= \langle E_2 - E_3 | E_3 - E_2\rangle(\langle E_1 - E_2 | E_2 - E_1\rangle(E_1)) \\ &= \langle E_2 - E_3 | E_3 - E_2\rangle(E_2) \\ &= E_3\end{aligned}$$

□

Lemma A.18 (Second Modality Transformation). *For all* L, E, F*:*

$$\langle L + (E - F) | F - E\rangle_{L+E} \Rightarrow \langle (L + E) - F | F - (L + E)\rangle_{L+E}$$

Proof. We can use Corollary A.16 since $(L + E) - F \leqslant L + E$ and $L + (E - F) \leqslant L + E$. We have:

$$\begin{aligned}\langle (L + E) - F | F - (L + E)\rangle(L + E) &= (F - (L + E)) + ((L + E) - (L + E - F)) \\ &= (F - (L + E)) + ((L + E) \cap F) \\ &= F\end{aligned}$$

and:

$$\begin{aligned}\langle L + (E - F) | F - E\rangle(L + E) &= (F - E) + ((L + E) - (L + (E - F))) \\ &= (F - E) + (E - (E - F)) \\ &= (F - E) + (E \cap F) \\ &= F\end{aligned}$$

□

Lemma A.19 (Third Modality Transformation). *For all* $\overline{\ell_i}, E, F$*:*

$$(\langle\!|\overline{\ell_i}|\!\rangle \circ \langle \overline{\ell_i}, E - F | F - \overline{\ell_i}, E\rangle)_E \Rightarrow \langle E - F | F - E\rangle_E$$

Proof. We can use Corollary A.16 since $(\langle\!|\overline{\ell_i}|\!\rangle \circ \langle \overline{\ell_i}, E - F | F - \overline{\ell_i}, E\rangle) = \langle \overline{\ell_i}, E - F | F - \overline{\ell_i}, E\rangle(\overline{\ell_i}, E)$ and $\overline{\ell_i}, E - F \leqslant \overline{\ell_i}, E$ and $E - F \leqslant E$. We have $\langle E - F | F - E\rangle(E) = F$ and:

$$\begin{aligned}\langle\!|\overline{\ell_i}|\!\rangle \circ \langle \overline{\ell_i}, E - F | F - \overline{\ell_i}, E\rangle(E) &= \langle \overline{\ell_i}, E - F | F - \overline{\ell_i}, E\rangle(\langle\!|\overline{\ell_i}|\!\rangle(E)) \\ &= \langle \overline{\ell_i}, E - F | F - \overline{\ell_i}, E\rangle(\overline{\ell_i}, E) \\ &= F\end{aligned}$$

□

Lemma A.20 (Translating Instantiated Types). *For all* $\mathrm{F}^1_{\mathrm{eff}}$ *types* A*:* $[\![A]\!]_E = [\![A[E'/]]\!]_{E,E'}$.

Proof. By induction on the type A.

Case $A = \mathsf{Int}$. Trivial.

Case $A = \forall.A'$. Trivial.

Case $A = A' \to^F B'$. Then:

$$\begin{aligned}\llbracket A \rrbracket_E &= \langle E - F | F - E \rangle (\llbracket A' \rrbracket_F \to \llbracket B' \rrbracket_F)\\ \llbracket A[E'/] \rrbracket_{E,E'} &= \langle E, E' - F, E' | F, E' - E, E' \rangle (\llbracket A'[E'/] \rrbracket_{F,E'} \to \llbracket B'[E'/] \rrbracket_{F,E'})\end{aligned}$$

By the induction hypothesis we have:

$$\begin{aligned}\llbracket A' \rrbracket_F &= \llbracket A'[E'/] \rrbracket_{F,E'}\\ \llbracket B' \rrbracket_F &= \llbracket B'[E'/] \rrbracket_{F,E'}\end{aligned}$$

Since we can freely permute labels:

$$\begin{aligned}\langle E, E' - F, E' | F, E' - E, E' \rangle &= \langle E', E - E', F | E', F - E', E \rangle\\ &= \langle E - F | F - E \rangle\end{aligned}$$

□

Lemma 5.2 (Type preservation of encoding). *If* $\Gamma \vdash M : A\,!\,\{E|\varepsilon\}$ *is well-scoped, then* $M : A\,!\,E \dashrightarrow M'$ *and* $\llbracket \Gamma \rrbracket_E \vdash M' : \llbracket A \rrbracket_E \mathbin{@} E$.

Proof. By induction on the typing derivation $\Gamma \vdash M : A\,!\,E$. We prove this for each rule of the translation. As a visual aid, we repeat each rule where we replace the translation premises by the Met judgement implied by the induction hypothesis and the translation in the conclusion by the Met judgement we need to prove.

R-Var

$$\frac{}{\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket_E \vdash \mathsf{rebox}(x; A; E) : \llbracket A \rrbracket_E \mathbin{@} E}$$

We use the $\mathsf{rebox}(x; A; E)$ function defined as follows:

$$\mathsf{rebox}(x; A; E) \quad = \quad \begin{cases} \mathbf{mod}_{\langle | \rangle}\, x, & \text{if } A = \mathsf{Int}\\ \mathbf{mod}_{\langle E - F | F - E \rangle}\, x, & \text{if } A = A' \to^F B'\\ \mathbf{mod}_{[]}\, x, & \text{if } A = \forall.A' \end{cases}$$

This function is exactly equivalent to $\mathbf{mod}_\mu\, x$ where $\mu = \mathsf{topmod}(\llbracket A \rrbracket_E)$ We use the T-Mod rule to introduce the box. By cases on the type A:

Case $A = \mathsf{Int}$. We can use the T-Var rule since $\cdot \vdash \mathsf{Int} : \mathsf{Abs}$.

Case $A = \forall.A'$. Then $\llbracket A \rrbracket_F = [] \llbracket A' \rrbracket$. for all F. By rule MT-Abs, the pure modality transforms into any other modality and so we can use the T-Var rule.

Case $A = A' \to^F B'$. Since the $\mathrm{F}^1_{\mathrm{eff}}$ judgement is well-scoped, we have that $\mathsf{locks}(\Gamma_2)$ is the composition of transition modalities. Furthermore, $\mathsf{locks}(\Gamma') \circ \langle E - F | F - E \rangle : F \to F'$ for the context F' where x as introduced and x is annotated by the modality $\langle F' - F | F - F' \rangle_{F'} : F \to F'$. By Lemma A.17, we can use the T-Var rule.

R-App

$$\frac{\llbracket \Gamma \rrbracket_E \vdash M' : \llbracket A \to^E B \rrbracket_E \mathbin{@} E \qquad \llbracket \Gamma \rrbracket_E \vdash N' : \llbracket A \rrbracket_E \mathbin{@} E \qquad x \text{ fresh}}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{let\ mod}_{\langle | \rangle}\, x = M' \ \mathbf{in}\ x\, N' : \llbracket B \rrbracket_E \mathbin{@} E}$$

We have $\llbracket A \to^E B \rrbracket_E = \langle | \rangle (\llbracket A \rrbracket_E \to \llbracket B \rrbracket_E)$. The claim follows by the T-LetMod and T-App rules.

$$\frac{\text{R-Abs}\quad \llbracket \Gamma, \blacklozenge_{E}, x : A \rrbracket_F \vdash M' : \llbracket B \rrbracket_F \mathbin{@} F \qquad \nu \coloneqq \langle E - F | F - E \rangle \qquad \mu \coloneqq \mathrm{topmod}(\llbracket A \rrbracket_F)}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{mod}_\nu\ (\lambda x^{\llbracket A \rrbracket_F}.\mathbf{let\ mod}_\mu\ x = x\ \mathbf{in}\ M') : \llbracket A \to^F B \rrbracket_E \mathbin{@} E}$$

We have $\llbracket \Gamma, \blacklozenge_{E}, x : A \rrbracket_F = \llbracket \Gamma \rrbracket_E, \text{🔒}_{\langle E - F | F - E \rangle}, x :_{\mu_F} A'$ where $\mu A' = \llbracket A \rrbracket_F$. Further $\llbracket A \to^F B \rrbracket_E = \langle E - F | F - E \rangle(\llbracket A \rrbracket_F \to \llbracket B \rrbracket_F)$. The claim follows from the T-LetMod, T-Abs and T-Mod rules.

$$\frac{\text{R-EAbs}\quad \llbracket \Gamma, \blacklozenge^{\Lambda}_{E} \rrbracket_{\cdot} \vdash V' : \llbracket A \rrbracket_{\cdot} \mathbin{@} \cdot}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{mod}_{[]}\ V' : \llbracket \forall . A \rrbracket_E \mathbin{@} E}$$

We have $\llbracket \Gamma, \blacklozenge^{\Lambda}_{E} \rrbracket_{\cdot} = \llbracket \Gamma \rrbracket_E, \text{🔒}_{[]}$. Further, $\llbracket \forall . A \rrbracket_E = [] \llbracket A \rrbracket_{\cdot}$. The claim follows from the T-Mod rule.

$$\frac{\text{R-EApp}\quad \llbracket \Gamma \rrbracket_E \vdash M' : \llbracket \forall . A \rrbracket_E \mathbin{@} E \qquad x \text{ fresh}}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{let\ mod}_{[]}\ x = M'\ \mathbf{in}\ x : \llbracket A[E/] \rrbracket_E \mathbin{@} E}$$

We have $\llbracket \forall . A \rrbracket_E = [] \llbracket A \rrbracket_{\cdot}$. By Lemma A.20, $\llbracket A \rrbracket_{\cdot} = \llbracket A[E/] \rrbracket_E$. The claim follows by the T-LetMod rule.

$$\frac{\text{R-Do}\quad \ell : A \twoheadrightarrow B \in \Sigma \qquad \llbracket \Gamma \rrbracket_{\ell,E} \vdash M' : \llbracket A \rrbracket_{\ell,E} \mathbin{@} \ell, E}{\llbracket \Gamma \rrbracket_{\ell,E} \vdash \mathbf{do}\ \ell\ M' : \llbracket B \rrbracket_{\ell,E} \mathbin{@} \ell, E}$$

Because we only allow pure values in the effect signatures of $\mathrm{F}^1_{\mathrm{eff}}$, we have that $\llbracket A \rrbracket_{\ell,E} = \llbracket A \rrbracket_{\cdot}$ and $\llbracket B \rrbracket_{\ell,E} = \llbracket B \rrbracket_{\cdot}$, where $\ell : \llbracket A \rrbracket_{\cdot} \twoheadrightarrow \llbracket B \rrbracket_{\cdot}$ in Met. The claim follows directly by the T-Do rule.

$$\frac{\text{R-Mask}\quad \llbracket \Gamma, \blacklozenge_{L+E} \rrbracket_E \vdash M' : \llbracket A \rrbracket_E \mathbin{@} E \qquad \mu_1 \coloneqq \mathrm{topmod}(\llbracket A \rrbracket_E) \qquad \mu_2 \coloneqq \mathrm{topmod}(\llbracket A \rrbracket_{L+E})}{\llbracket \Gamma \rrbracket_{L+E} \vdash \mathbf{let\ mod}_{\langle L | \rangle ; \mu_1}\ x = \mathbf{mask}_L\ M'\ \mathbf{in\ mod}_{\mu_2}\ x : \llbracket A \rrbracket_{L+E} \mathbin{@} L + E}$$

We have $\llbracket \Gamma, \blacklozenge_{L+E} \rrbracket_E = \llbracket \Gamma \rrbracket_{L+E}, \text{🔒}_{\langle (L+E) - E | E - (L+E) \rangle}$. By permuting labels, we have $\langle (L+E) - E | E - (L+E) \rangle = \langle L | \rangle$. The goal follows by the T-LetMod, T-Mask and T-Mod rules if we can show that x can be used under the box. This is clear for integers, since they are pure and otherwise we need to show that $(\langle L | \rangle \circ \mu_1)_{L+E} \Rightarrow (\mu_2)_{L+E}$. For $A = \forall . A'$ this is clear since $\mu_1 = \mu_2 = []$ and $\langle L | \rangle \circ [] = []$. For functions, this follows from Lemma A.18.

$$\frac{\begin{array}{c}\text{R-Handler} \\ \llbracket \Gamma, \blacklozenge_{E} \rrbracket_{\overline{\ell_i}, E} \vdash M' : \llbracket A \rrbracket_{\overline{\ell_i}, E} \mathbin{@} \overline{\ell_i}, E \\ \llbracket \Gamma, x : A \rrbracket_E \vdash N' : \llbracket B \rrbracket_E \mathbin{@} E \qquad [\llbracket \Gamma, p_i : A_i, r_i : B_i \to^E B \rrbracket_E \vdash N'_i : \llbracket B \rrbracket_E \mathbin{@} E]_i \\ \mu \coloneqq \mathrm{topmod}(\llbracket A \rrbracket_{\overline{\ell_i}, E}) \qquad \mu' \coloneqq \mathrm{topmod}(\llbracket A \rrbracket_E) \\ N'' \coloneqq \mathbf{let\ mod}_{\langle | \overline{\ell_i} \rangle ; \mu}\ x = x\ \mathbf{in\ let}_{\mu'}\ \mathbf{mod}_{\langle | \rangle}\ x = \mathbf{mod}_{\langle | \rangle}\ x\ \mathbf{in}\ N' \\ [\mu_i \coloneqq \mathrm{topmod}(\llbracket A_i \rrbracket_{\cdot}) \qquad N''_i \coloneqq \mathbf{let\ mod}_{\mu_i}\ p_i = p_i\ \mathbf{in}\ N'_i]_i \\ H = \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \qquad H' \coloneqq \{\mathbf{return}\ x \mapsto N''\} \uplus \{\ell_i\ p_i\ r_i \mapsto N'_i\}_i\end{array}}{\llbracket \Gamma \rrbracket_E \vdash \mathbf{handle}\ M'\ \mathbf{with}\ H' : \llbracket B \rrbracket_E \mathbin{@} E}$$

We have $[\![\Gamma, \blacklozenge_{E}]\!]_{\overline{\ell_i},E} = [\![\Gamma]\!]_E, \text{🔒}_{\langle E-\overline{\ell_i},E|\overline{\ell_i},E-E\rangle}$. By permuting labels, we have $\langle E - \overline{\ell_i}, E|\overline{\ell_i}, E - E\rangle = \langle|\overline{\ell_i}\rangle$. In the operation clauses, we have that $[\![B_i \to^{E} B]\!]_E = \langle|\rangle([\![B_i]\!]_E \to [\![B]\!]_E)$. Because the argument and return of effects are pure, we have that $[\![B_i]\!]_E = [\![B_i]\!]$. and $[\![A_i]\!]_E = [\![A]\!]$.. We need to unbox the argument p_i though. In the return clause, Met gives us $x : \langle|\overline{\ell_i}\rangle[\![A]\!]_{\overline{\ell_i},E}$, but we need $x : [\![A]\!]_E$. We achieve this by unboxing x fully and then re-boxing it with the modality μ'. This is possible for integers because they are pure, for $\forall$s because of the MT-Abs rule and for functions due to the modality transformation in Lemma A.19.

□

B Full Specification of Metel

In this section, we give a full specification of Metel including the declarative type system, type inference algorithm, meta theory of type inference, and elaboration to the core calculus. The proofs are given in Appendix C.

We focus on formalising the core part of the type inference of Metel. We assume standard language features like algebraic data types and pattern matching when writing examples; they are largely orthogonal to our main contribution of type inference.

B.1 Syntax

The syntax of Metel is shown in Figure 12. The new parts compared to Met are highlighted.

Types	$A, B ::= \alpha \mid A \to B \mid \mu A$
Intuitionistic types	$S, T ::= \alpha \mid S \to T$
Effects	$E ::= \cdot \mid \varepsilon \mid D, E \mid E \backslash L$
Masks and Extensions	$L, D ::= \cdot \mid \ell, L$
Modalities	$\mu ::= [E] \mid \langle L \mid D\rangle$
Type schemes	$\sigma ::= A \mid \forall \alpha^{K}.A$
Kinds	$K ::= \mathsf{Abs} \mid \mathsf{Any} \mid \mathsf{Eff}$
Restrictions	$R ::= \mathsf{i} \mid \mathsf{m}$
Contexts	$\Gamma ::= \cdot \mid \Gamma, \alpha : K \mid \Gamma, x :_{\mu} \sigma \mid \Gamma, \text{🔒}_{\mu}$
Type contexts	$\Delta ::= \cdot \mid \Delta, \alpha : K$
Label contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : A \twoheadrightarrow B$
Modality decorations	$\phi ::= \cdot \mid \mu$
Terms	$M, N ::= x \mid \lceil x \rceil \mid \lambda x.M \mid \lambda x^{A}.M \mid M\,N \mid \mathbf{mod}_{\mu}\,V$
	$\mid \mathbf{let}_{\nu}\,\phi\; x = M \;\mathbf{in}\; N \mid \mathbf{let}\; x^{\sigma} = M \;\mathbf{in}\; N$
	$\mid \mathbf{do}\;\ell\; M \mid \mathbf{mask}_{L}\, M \mid \mathbf{handle}\; M \;\mathbf{with}\; H$
Values	$V, W ::= x \mid \lceil x \rceil \mid \lambda x.M \mid \lambda x^{A}.M \mid \mathbf{mod}_{\mu}\,V$
Handlers	$H ::= \{\mathbf{return}\; x \mapsto M\} \mid \{\ell\; p\; r \mapsto M\} \uplus H$

Fig. 12. Syntax of Metel.

Following FreezeML [15], we always fully unbox variables unless they are explicitly frozen by $\lceil x \rceil$. Restrictions distinguish between intuitionistic types i, which cannot contain any modalities, and modal types, which can contain modalities. Following FreezeML, though rigid type variables α could technically be instantiated to modal types, we allow intuitionistic types to contain them since they are rigid and cannot be unified with other types during type inference. As in ML, we generalise type variables for let-bindings. We combine normal let-binding and modality elimination into one syntax $\mathbf{let}_{\nu}\,\phi\; x = M \;\mathbf{in}\; N$. When $\phi = \cdot$, it is a normal let-binding.

Different from the core calculus, we keep modalities μ in context. We will show in Appendix B.6 that this change does not break soundness and we can always elaborate well-typed closed terms in METEL to well-typed closed terms in METE.

We restrict extensions and effects to only contain present labels whose signatures are given by a global context Σ for simplicity. We do not expect any specific challenges of generalising them with signatures. Notice that we can still reuse all previous definitions of modes and modalities of MET. The only differences are that labels with the same name always have the same signature and absent labels are not allowed to appear explicitly.

For simplicity of type inference, we do not allow negative effects of form $E \backslash L$ to appear in the surface syntax. We write $\vdash M\ \mathbf{pos}$ if all type and modality annotations in M do not contain $E \backslash L$. That is, all effect types should have form either D or D, ε. It still allows annotations in M to contain rigid type and effect variables. This is an acceptable restriction in practice since we rarely need to use effect variables and masking at the same time. And even we do need, we can always just refactor effect types to avoid negative effects to appear in type annotations.

We write $\vdash A\ \mathbf{pos}$ if type A does not contain $E \backslash L$, and $\vdash \Gamma\ \mathbf{pos}$ if all types A of variable bindings satisfy $\vdash A\ \mathbf{pos}$. Note that $\vdash \Gamma\ \mathbf{pos}$ still allows the modalities in $x :_{\mu} \_$ and 🔒$_{\mu}$ to contain effect types of any form including $E \backslash L$.

B.2 Statements in Context and Syntax-Directed Typing Rules

We formalise the syntax-directed type system and type inference algorithm following the approach of type inference in context [20]. We first define statements.

$$\begin{array}{lrl} \text{Statements} & J ::= & J \wedge J' \mid \sigma : (K, R) \mid A \equiv B \mid \sigma \leq_R A \mid M\ \mathbf{ok} \mid \sigma \leq_{\text{gen}} \sigma' \\ & \mid & (\mu, \sigma) \Rightarrow \nu\ @\ E \mid (M; \Delta; A) \Updownarrow^{\dagger} \sigma \mid (M; \nu; \phi; \Delta; A) \Updownarrow (\xi, \sigma) \\ & \mid & (M; \Delta; A) \Downarrow B \mid M : A\ @\ E \end{array}$$

For each statement, we define the judgement $\Gamma \vdash J$ which means the statement J holds in the context Γ. All these judgements require implicit well-formedness conditions for the statements and contexts. That is, all free type and term variables in statements should appear in the context Γ, and all effect labels should appear in the global label context Σ. Contexts are ordered and types can only refer to variables bound on the left of them in contexts.

The kinding $\sigma : (K, R)$, type equivalence $A \equiv B$, instantiation $\sigma \leq_R A$ and term well-formedness $M\ \mathbf{ok}$ are defined in Figure 13. The conjunction of statements is standard and defined as follows.

$$\frac{\Gamma \vdash J \qquad \Gamma \vdash J'}{\Gamma \vdash J \wedge J'} \qquad\qquad \frac{\Gamma \vdash J \wedge J'}{\Gamma \vdash J} \qquad\qquad \frac{\Gamma \vdash J \wedge J'}{\Gamma \vdash J'}$$

Some auxiliary statements and auxiliary functions for typing are defined in Figure 14. The judgement $\Gamma \vdash (\mu, \sigma) \Rightarrow \nu\ @\ E$ checks the accessibility condition for variables. The judgements $\Gamma \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma$ deals with value restriction for T-LETANNO. The judgements $\Gamma \vdash (M; \nu; \phi; \Delta; A)$ deals with value restriction for T-LETMOD, as well as case analyses on the shape of ϕ.

The syntax-directed typing judgement $M : A\ @\ E$ is defined in Figure 15. The typing rules different from Figure 3 are highlighted. The T-FREEZE rule is the relatively standard variable rule. The T-VAR additionally eliminates the modality for x that is retrieved by $\mathsf{split}(\Delta, A)$ defined in Figure 14. It keeps splitting out the top-level modalities of A until reaching a non-modal type or the modality relies on rigid variables in Δ, which are quantified. The T-LETMOD generalise M when M is a value; otherwise, it instantiate the principal type of M with intuitionistic types. The T-HANDLER also instantiate the principal types of M and N with intuitionistic types. This avoids solving global non-trivial constraints on flexible modal or effect variables in type inference.

$\boxed{\Gamma \vdash \sigma : (K, R)}$

$$\frac{\Gamma \ni \alpha : K}{\Gamma \vdash \alpha : (K, \mathsf{res}(K))} \qquad \frac{\Gamma \vdash \sigma : (K, \mathsf{i})}{\Gamma \vdash \sigma : (K, \mathsf{m})} \qquad \frac{\Gamma \vdash \sigma : (\mathsf{Abs}, R)}{\Gamma \vdash \sigma : (\mathsf{Any}, R)} \qquad \frac{\Gamma \vdash A : (K, \mathsf{m})}{\Gamma \vdash \langle L | D \rangle A : (K, \mathsf{m})}$$

$$\frac{\Gamma \vdash E : (\mathsf{Eff}, \mathsf{m}) \qquad \Gamma \vdash A : (\mathsf{Any}, \mathsf{m})}{\Gamma \vdash [E] A : (\mathsf{Abs}, \mathsf{m})} \qquad \frac{\Gamma \vdash A : (\mathsf{Any}, R) \qquad \Gamma \vdash B : (\mathsf{Any}, R)}{\Gamma \vdash A \to B : (\mathsf{Any}, R)}$$

$$\frac{\Gamma, \alpha : K \vdash \sigma : (K', R)}{\Gamma \vdash \forall \alpha^K . \sigma : (K', R)} \qquad \frac{}{\Gamma \vdash \cdot : (\mathsf{Eff}, \mathsf{m})} \qquad \frac{\Gamma \vdash E : (\mathsf{Eff}, \mathsf{m})}{\Gamma \vdash \ell, E : (\mathsf{Eff}, \mathsf{m})}$$

$\boxed{\Gamma \vdash \sigma \leq_R A}$

$$\frac{}{\Gamma \vdash A \leq_R A} \qquad \frac{\Gamma \vdash B : (K, R) \qquad \Gamma \vdash \sigma[B/\alpha] \leq_R A}{\Gamma \vdash \forall \alpha^K . \sigma \leq_R A}$$

$\boxed{\Gamma \vdash \sigma \leq_{\mathrm{gen}} \sigma'}$

$$\frac{\Gamma \vdash \sigma \equiv \sigma'}{\Gamma \vdash \sigma \leq_{\mathrm{gen}} \sigma'} \qquad \frac{\Gamma \vdash B : (K, \mathsf{m}) \qquad \Gamma \vdash \sigma[B/\alpha] \leq_{\mathrm{gen}} \sigma'}{\Gamma \vdash \forall \alpha^K . \sigma \leq_{\mathrm{gen}} \sigma'} \qquad \frac{\Gamma, \alpha : K \vdash \sigma \leq_{\mathrm{gen}} \sigma'}{\Gamma \vdash \sigma \leq_{\mathrm{gen}} \forall \alpha^K . \sigma'}$$

$\boxed{\Gamma \vdash M \ \mathbf{ok}}$

$$\frac{\Gamma \vdash M \ \mathbf{ok} \qquad \Gamma \vdash N \ \mathbf{ok}}{\Gamma \vdash \mathbf{let}_v \ \phi \ x = M \ \mathbf{in} \ N \ \mathbf{ok}} \qquad \frac{\Gamma \vdash \forall \Delta . A : (\mathsf{Any}, \mathsf{m}) \qquad \Gamma, \Delta \vdash M \ \mathbf{ok} \qquad \Gamma \vdash N \ \mathbf{ok}}{\Gamma \vdash \mathbf{let} \ x^{\forall \Delta . A} = M \ \mathbf{in} \ N \ \mathbf{ok}}$$

$$\frac{}{\Gamma \vdash x \ \mathbf{ok}} \qquad \frac{}{\Gamma \vdash \lceil x \rceil \ \mathbf{ok}} \qquad \frac{\Gamma \vdash A : (\mathsf{Any}, \mathsf{m}) \qquad \Gamma \vdash M \ \mathbf{ok}}{\Gamma \vdash \lambda x^A . M \ \mathbf{ok}} \qquad \frac{\Gamma \vdash M \ \mathbf{ok}}{\Gamma \vdash \lambda x . M \ \mathbf{ok}}$$

$$\frac{\Gamma \vdash M \ \mathbf{ok} \qquad \Gamma \vdash N \ \mathbf{ok}}{\Gamma \vdash M \ N \ \mathbf{ok}} \qquad \frac{\Gamma \vdash M \ \mathbf{ok}}{\Gamma \vdash \mathbf{do} \ \ell \ M \ \mathbf{ok}} \qquad \frac{\Gamma \vdash M \ \mathbf{ok}}{\Gamma \vdash \mathbf{mask}_L \ M \ \mathbf{ok}}$$

$$\frac{\begin{array}{c} H = \{\mathbf{return} \ x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i \qquad D = \overline{\ell_i} \\ \Gamma \vdash M \ \mathbf{ok} \qquad \Gamma \vdash N \ \mathbf{ok} \qquad [\Gamma \vdash N_i \ \mathbf{ok}]_i \end{array}}{\Gamma \vdash \mathbf{handle} \ M \ \mathbf{with} \ H \ \mathbf{ok}}$$

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

$$\frac{\Gamma \ni \alpha : K}{\Gamma \vdash \alpha \equiv \alpha} \qquad \frac{\Gamma \vdash \mu \equiv \nu \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash \mu A \equiv \nu B} \qquad \frac{\Gamma \vdash A \equiv A' \qquad \Gamma \vdash B \equiv B'}{\Gamma \vdash A \to B \equiv A' \to B'} \qquad \frac{\Gamma \vdash E \equiv F}{\Gamma \vdash [E] \equiv [F]}$$

$$\frac{L \equiv L' \qquad D \equiv D'}{\Gamma \vdash \langle L | D \rangle \equiv \langle L' | D' \rangle} \qquad \frac{E \equiv F \qquad \Gamma \vdash E}{\Gamma \vdash E \equiv F}$$

Fig. 13. Statements in context for Metel.

$$\boxed{\Gamma \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E} \quad \boxed{\Gamma \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma} \quad \boxed{\Gamma \vdash (M; \nu; \phi; \Delta; A) \Updownarrow (\xi, \sigma)} \quad \boxed{\Gamma \vdash (M; \Delta; A) \Downarrow B}$$

$$\frac{\Gamma \vdash \sigma : \mathsf{Abs}}{\Gamma \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E} \qquad \frac{\mu_F \Rightarrow \nu_F \qquad \nu_F : E \to F}{\Gamma \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E} \qquad \frac{M \in \mathsf{Val}}{\Gamma \vdash (M; \Delta; A) \Updownarrow^{\dagger} \forall \Delta.A}$$

$$\frac{M \notin \mathsf{Val} \qquad \Delta = \cdot}{\Gamma \vdash (M; \Delta; A) \Updownarrow^{\dagger} A} \qquad \frac{\mathsf{principal}(\Gamma; M; \Delta; A) \qquad \Gamma \vdash \forall \Delta.A \preceq_{\mathsf{i}} B}{\Gamma \vdash (M; \Delta; A) \Downarrow B}$$

$$\frac{M \in \mathsf{Val} \qquad \mathsf{principal}(\Gamma, \blacksquare_{\nu}; M; \Delta; \phi A) \qquad \xi = \begin{cases} \nu \circ \mu, & \phi = \mu \\ \nu, & \phi = \cdot \end{cases} \qquad \phi \neq \cdot \text{ or } \nu = \mathbb{1}}{\Gamma \vdash (M; \nu; \phi; \Delta; A) \Updownarrow (\xi, \forall \Delta.A)}$$

$$\frac{M \notin \mathsf{Val} \qquad \nu = \mathbb{1} \qquad \Gamma \vdash (M; \Delta; A) \Downarrow B \qquad \xi = \begin{cases} \mu, & \phi = \mu \\ \mathbb{1}, & \phi = \cdot \end{cases}}{\Gamma \vdash (M; \nu; \phi; \Delta; A) \Updownarrow (\xi, B)}$$

$$\begin{aligned} \mathsf{principal}(\Gamma; M; \Delta; A) \quad &= \quad \Gamma, \Delta \vdash_s M : A \mathbin{@} E \text{ for some } E \text{ such that} \\ &\qquad \text{for any } \Delta', A', E' \text{ with } \Gamma, \Delta' \vdash_s M : A' \mathbin{@} E', \\ &\qquad \text{we have } \Gamma, \Delta' \vdash \forall \Delta.A \leq_{\mathsf{m}} A' \text{ and } E \leqslant E' \\ \mathsf{split}(\Delta; A) \quad &= \quad \begin{cases} \mathsf{let}\ (\nu, B) = \mathsf{split}(\Delta; A')\ \mathsf{in}\ (\mu \circ \nu, B), \\ \qquad\qquad \text{if } A = \mu A' \text{ and } \mathsf{ftv}(\mu) \cap \mathsf{dom}(\Delta) = \varnothing \\ (\mathbb{1}, A), \quad \text{otherwise} \end{cases} \end{aligned}$$

Fig. 14. Auxiliary judgements and meta-functions for Metel.

$$\boxed{\Gamma \vdash_s M : A \mathbin{@} E}$$

T-Freeze
$$\frac{\xi = \mathsf{alocks}(\Gamma') \qquad \Gamma, \Gamma' \vdash (\mu, \forall\Delta.A) \Rightarrow \xi \mathbin{@} E \qquad \Gamma, \Gamma' \vdash \forall\Delta.A \preceq_{\mathrm{m}} B}{\Gamma, x :_{\mu} \forall\Delta.A, \Gamma' \vdash_s \lceil x \rceil : B \mathbin{@} E}$$

T-Var
$$\frac{\xi = \mathsf{alocks}(\Gamma') \qquad (\nu, A') = \mathsf{split}(\Delta; A) \qquad \Gamma, \Gamma' \vdash (\mu \circ \nu, \forall\Delta.A') \Rightarrow \xi \mathbin{@} E \qquad \Gamma, \Gamma' \vdash \forall\Delta.A' \preceq_{\mathrm{m}} B}{\Gamma, x :_{\mu} \forall\Delta.A, \Gamma' \vdash_s x : B \mathbin{@} E}$$

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

T-AbsAnno
$$\frac{\Gamma, x : A \vdash_s M : B \mathbin{@} E}{\Gamma \vdash_s \lambda x^{A}.M : A \to B \mathbin{@} E}$$

T-Abs
$$\frac{\Gamma, x : S \vdash_s M : B \mathbin{@} E}{\Gamma \vdash_s \lambda x.M : S \to B \mathbin{@} E}$$

T-App
$$\frac{\Gamma \vdash_s M : A \to B \mathbin{@} E \qquad \Gamma \vdash_s N : A \mathbin{@} E}{\Gamma \vdash_s M\ N : B \mathbin{@} E}$$

T-LetMod
$$\frac{\Gamma \vdash (M; \nu; \Delta; \phi; A) \Updownarrow (\xi, \sigma) \qquad \Gamma, \text{🔒}_{\nu}, \Delta \vdash_s M : \phi A \mathbin{@} E \qquad \nu_F : E \to F \qquad \Gamma, x :_{\xi} \sigma \vdash_s N : B \mathbin{@} F}{\Gamma \vdash_s \mathbf{let}_{\nu}\ \phi\ x = M\ \mathbf{in}\ N : B \mathbin{@} F}$$

T-Mask
$$\frac{\Gamma, \text{🔒}_{\langle L|\rangle} \vdash_s M : A \mathbin{@} F - L}{\Gamma \vdash_s \mathbf{mask}_L\ M : \langle L|\rangle A \mathbin{@} F}$$

T-LetAnno
$$\frac{\Gamma \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma \qquad \Gamma, \Delta \vdash_s M : A \mathbin{@} E \qquad \Gamma, x : \sigma \vdash_s N : B \mathbin{@} E}{\Gamma \vdash_s \mathbf{let}\ x^{\forall\Delta.A} = M\ \mathbf{in}\ N : B \mathbin{@} E}$$

T-Do
$$\frac{\Sigma \ni \ell : A \twoheadrightarrow B \qquad E = \ell, F \qquad \Gamma \vdash_s M : A \mathbin{@} E}{\Gamma \vdash_s \mathbf{do}\ \ell\ M : B \mathbin{@} E}$$

T-Handler
$$\frac{\begin{array}{c} D = \{\ell_i\}_i \qquad \{\ell_i : A_i \twoheadrightarrow B_i\} \subseteq \Sigma \\ \Gamma \vdash (M; \Delta; A_0) \Downarrow A \qquad \Gamma, \text{🔒}_{\langle|D\rangle}, \Delta \vdash_s M : A_0 \mathbin{@} D + F \\ \Gamma \vdash (N; \Delta'; B_0) \Downarrow B \qquad \Gamma, x : \langle D\rangle A, \Delta' \vdash_s N : B_0 \mathbin{@} F \\ [\Gamma, p_i : A_i, r_i : B_i \to B \vdash_s N_i : B \mathbin{@} F]_i \end{array}}{\Gamma \vdash_s \mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \mathbin{@} F}$$

Fig. 15. Syntax-directed typing rules for Metel.

B.3 Algorithmic Contexts and Metasubstitutions

We distinguish between *rigid* type variables (which come from the object language and can only be unified with intuitionistic types) and *flexible* type variables (which come from algorithms and can be unified with both intuitionistic and modal types). We introduce flexible type variables $\hat{\alpha}$ and extend the syntax of types and contexts as follows.

Types	$A, B ::= \cdots \mid \hat{\alpha}$
Algorithmic contexts	$\Theta ::= \cdot \mid \Theta, \alpha : K \mid \Theta, x : \sigma \mid \Theta, \text{🔒}_{\mu} \mid \Theta, \hat{\alpha} : (K, R) \mid \Theta, \hat{\alpha} = A \mid \Theta \fatsemi$
Suffixes	$\Xi ::= \cdot \mid \Xi, \hat{\alpha} : (K, R) \mid \Xi, \hat{\alpha} = A$

Flexible type variables in algorithmic contexts are either declarations $\hat{\alpha} : (K, R)$ with kinds and restrictions, or definitions $\hat{\alpha} = A$ which indicate that these flexible variables have been solved.

We do not allow type annotations in terms to use flexible type variables. The syntax for type schemes is still $\forall \Delta . A$.

We define metasubstitutions $\theta \fatsemi \Theta \sqsubseteq \Theta'$ from the algorithmic context Θ to Θ' and the equivalence relation between metasubstitutions in Figure 16. They are the same as the definitions in Gundry [20] except for adding more trivial cases for elements including bindings of rigid type variables and locks. Metasubstitutions reflect information increase between contexts.

$\boxed{\theta \fatsemi \Theta \sqsubseteq \Theta'}$

$$\frac{}{\iota \fatsemi \cdot \sqsubseteq \Xi} \qquad \frac{\theta \fatsemi \Theta \sqsubseteq \Theta' \qquad \Theta' \vdash A : (K, R)}{\theta, A/\hat{\alpha} \fatsemi \Theta, \hat{\alpha} : (K, R) \sqsubseteq \Theta'} \qquad \frac{\theta \fatsemi \Theta \sqsubseteq \Theta' \qquad \Theta' \vdash \theta A \equiv B}{\theta, B/\hat{\alpha} \fatsemi \Theta, \hat{\alpha} = A \sqsubseteq \Theta'}$$

$$\frac{\theta \fatsemi \Theta \sqsubseteq \Theta'}{\theta \fatsemi \Theta, \alpha : K \sqsubseteq \Theta', \alpha : K, \Xi} \qquad \frac{\theta \fatsemi \Theta \sqsubseteq \Theta'}{\theta \fatsemi \Theta, x : \sigma \sqsubseteq \Theta', x : \theta\sigma, \Xi}$$

$$\frac{\theta \fatsemi \Theta \sqsubseteq \Theta'}{\theta \fatsemi \Theta, \text{🔒}_{\mu} \sqsubseteq \Theta', \text{🔒}_{\mu}, \Xi} \qquad \frac{\theta \fatsemi \Theta \sqsubseteq \Theta'}{\theta \fatsemi \Theta, \fatsemi \sqsubseteq \Theta', \fatsemi, \Xi}$$

$\boxed{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta'}$

$$\frac{}{\iota \equiv \iota \fatsemi \cdot \sqsubseteq \Xi} \qquad \frac{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta' \qquad \Theta' \vdash A : (K, R) \qquad \Theta' \vdash A \equiv A'}{\theta, A/\hat{\alpha} \equiv \theta', A'/\hat{\alpha} \fatsemi \Theta, \hat{\alpha} : (K, R) \sqsubseteq \Theta'}$$

$$\frac{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta' \qquad \Theta' \vdash \theta A \equiv B \qquad \Theta' \vdash B \equiv B'}{\theta, B/\hat{\alpha} \equiv \theta', B'/\hat{\alpha} \fatsemi \Theta, \hat{\alpha} = A \sqsubseteq \Theta'} \qquad \frac{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta'}{\theta \equiv \theta' \fatsemi \Theta, \alpha : K \sqsubseteq \Theta', \alpha : K, \Xi}$$

$$\frac{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta'}{\theta \equiv \theta' \fatsemi \Theta, x : \sigma \sqsubseteq \Theta', x : \theta\sigma, \Xi} \qquad \frac{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta'}{\theta \equiv \theta' \fatsemi \Theta, \text{🔒}_{\mu} \sqsubseteq \Theta', \text{🔒}_{\mu}, \Xi} \qquad \frac{\theta \equiv \theta' \fatsemi \Theta \sqsubseteq \Theta'}{\theta \equiv \theta' \fatsemi \Theta, \fatsemi \sqsubseteq \Theta', \fatsemi, \Xi}$$

Fig. 16. Metasubstitutions and equivalence of metasubstitutions.

$$\frac{}{\Theta, \hat{\alpha} : (K, R), \Theta' \vdash \hat{\alpha} : (K, R)} \qquad \frac{\Theta \vdash A : (K, R)}{\Theta, \hat{\alpha} = A, \Theta' \vdash \hat{\alpha} : (K, R)}$$

$$\frac{}{\Theta, \hat{\alpha} : (K, R), \Theta' \vdash \hat{\alpha} \equiv \hat{\alpha}} \qquad \frac{\Theta, \Theta' \vdash A \equiv B}{\Theta, \hat{\alpha} = A, \Theta' \vdash \hat{\alpha} \equiv B} \qquad \frac{\Theta, \Theta' \vdash A \equiv B}{\Theta, \hat{\alpha} = A, \Theta' \vdash B \equiv \hat{\alpha}}$$

T-Freeze

$$\frac{\begin{array}{c}\xi = \mathsf{alocks}(\Theta') \qquad \forall \Delta.A = \mathsf{subst}(\Theta; \sigma) \\ \Theta, \Theta' \vdash (\mu, \forall \Delta.A) \Rightarrow \xi \ @\ E \qquad \Theta, \Theta' \vdash \forall \Delta.A \leq_{\mathsf{m}} B\end{array}}{\Theta, x :_{\mu} \sigma, \Theta' \vdash_s \lceil x \rceil : B \ @\ E}$$

T-Var

$$\frac{\begin{array}{c}\xi = \mathsf{alocks}(\Theta') \qquad \forall \Delta.A = \mathsf{subst}(\Theta; \sigma) \\ (\nu, A') = \mathsf{split}(\Delta, A) \qquad \Theta, \Theta' \vdash (\mu \circ \nu, \forall \Delta.A') \Rightarrow \xi \ @\ E \qquad \Theta, \Theta' \vdash \forall \Delta.A' \leq_{\mathsf{m}} B\end{array}}{\Theta, x :_{\mu} \sigma, \Theta' \vdash_s x : B \ @\ E}$$

Fig. 17. Extended rules for statements in algorithmic contexts.

We define $\mathsf{gen}(\Xi; A)$ as substituting solved flexible variables and generalising remaining flexible variables in Ξ. We define $\mathsf{subst}(\Theta; A)$ as substituting solved flexible variables in Θ.

$$\begin{array}{rcl}
\mathsf{gen}(\cdot; A) & = & A \\
\mathsf{gen}(\hat{\alpha} : (K, R), \Xi; A) & = & \forall \alpha : K.\mathsf{gen}(\Xi; A[\alpha/\hat{\alpha}]) \\
\mathsf{gen}(\hat{\alpha} = B, \Xi; A) & = & \mathsf{gen}(\Xi[B/\hat{\alpha}]; A[B/\hat{\alpha}])
\end{array}$$

$$\begin{array}{rcl}
\mathsf{subst}(\cdot; A) & = & A \\
\mathsf{subst}(\hat{\alpha} = B, \Theta; A) & = & \mathsf{subst}(\Theta[B/\hat{\alpha}]; A[B/\hat{\alpha}]) \\
\mathsf{subst}(_, \Theta; A) & = & \mathsf{subst}(\Xi; A)
\end{array}$$

Although the judgements for statements in context are all defined on declarative context Γ, it is easy to extend them to algorithmic contexts Θ. For any $\Gamma \vdash J$, we get $\Theta \vdash J$ almost freely by just replacing letters from Γ to Θ. The only non-trivial modifications are to extend kinding $\Theta \vdash A : (K, R)$, type equivalence $\Theta \vdash A \equiv B$, and typing $\Gamma \vdash M : A \ @\ E$ to cover flexible variables. The extended rules are shown in Figure 17.

The essence of type inference for Metel is that we never guesses flexible modal and effect variables in contexts. This property allows us to avoid collecting and solving non-trivial global constraints on modalities in type inference. We define $\vdash \Theta\ \mathsf{ng}$ if all locks and variable bindings in Θ do not contain unsolved flexible modal or effect variables.

$$\frac{}{\vdash \cdot\ \mathsf{ng}} \qquad \frac{\vdash \Theta\ \mathsf{ng} \qquad \Theta \vdash \mathsf{subst}(\Theta; \sigma)\ \mathsf{ng}}{\vdash \Theta, x :_{\mu} \sigma\ \mathsf{ng}} \qquad \frac{\vdash \Theta\ \mathsf{ng}}{\vdash \Theta, \text{🔒}_{\mu}\ \mathsf{ng}} \qquad \frac{\vdash \Theta\ \mathsf{ng}}{\vdash \Theta \fatsemi\ \mathsf{ng}} \qquad \frac{\vdash \Theta\ \mathsf{ng}}{\vdash \Theta, \alpha : K\ \mathsf{ng}}$$

$$\frac{\vdash \Theta\ \mathsf{ng}}{\vdash \Theta, \hat{\alpha} : (K, R)\ \mathsf{ng}} \qquad \frac{\vdash \Theta\ \mathsf{ng}}{\vdash \Theta, \hat{\alpha} = A\ \mathsf{ng}}$$

The following lemma shows that metasubstitutions preserve this relation.

Lemma B.1 (No guess of modalities and effects in contexts). *If* $\vdash \Theta_0\ \mathbf{ng}$ *and* $\theta \fatsemi \Theta_0 \sqsubseteq \Theta_1$, *then* $\vdash \Theta_1\ \mathbf{ng}$.

B.4 Algorithmic Moving between Contexts

Now we give algorithms to solve statements in contexts except for type inference, which is given individually in the next section. All algorithms have form $\Theta_0 \vdash J \dashv \Theta_1$, which starts from the algorithmic context Θ_0, solves the question of J and ends up with the algorithmic context Θ_1.

We first define the notion of questions, solutions, and minimal solutions for statements that we need algorithms. Same as in the declarative version, we always require the algorithmic contexts Θ for questions and solutions to satisfy $\vdash \Theta\ \mathbf{pos}$.

Statements like kinding $\sigma : (K, R)$, type equivalence $A \equiv B$, and term well-formedness $M\ \mathbf{ok}$ only have inputs; solving them only needs to make sure that the judgements are satisfied. We define solutions and minimal solutions for them.

Definition B.2 (Questions without outputs and their solutions). A question for a statement J which does not have outputs is a tuple $(\Theta_0; J)$ where J is well-scoped in Θ_0. A solution to it is a metasubstitution $\theta \fatsemi \Theta_0 \sqsubseteq \Theta_1$ where $\Theta_1 \vdash \theta J$. The solution is minimal if for any other solution $\theta' \fatsemi \Theta_0 \sqsubseteq \Theta'$, there exists a metasubstitution $\zeta \fatsemi \Theta_1 \sqsubseteq \Theta'$ such that $\theta' \equiv \zeta\theta \fatsemi \Theta_0 \sqsubseteq \Theta'$ (say θ' factors through θ with cofactor ζ). When J is an equivalence statement $A \equiv B$, we additionally require $\vdash A\ \mathbf{pos}$ and $\vdash B\ \mathbf{pos}$.

Other statements separate between inputs and outputs; solving them also requires giving outputs. We define questions, solutions, and minimal solutions for those we need.

Definition B.3 (Questions with outputs and their solutions).

- An instantiation question is a tuple $(\Theta_0; \sigma \preceq_R \bigcirc)$ where σ is well-scoped in Θ_0. A solution to it is a tuple $(\theta \fatsemi \Theta_0 \sqsubseteq \Theta_1; A)$ such that $\Theta_1 \vdash \theta\sigma \preceq_R A$. The solution is minimal if for any other solution $(\theta' \fatsemi \Theta_0 \sqsubseteq \Theta'; A')$, there exists a metasubstitution $\xi \fatsemi \Theta_1 \sqsubseteq \Theta'$ such that $\theta' \equiv \xi\theta \fatsemi \Theta_0 \sqsubseteq \Theta'$ and $\Theta' \vdash \xi A \equiv A'$.
- A transformation question is a tuple $(\Theta_0; (\mu, \sigma) \Rightarrow \nu\ @\ \bigcirc)$ where σ is well-scoped in Θ_0. A solution to it is a tuple $(\theta \fatsemi \Theta_0 \sqsubseteq \Theta_1; E)$ such that $\Theta_1 \vdash (\mu, \theta\sigma) \Rightarrow \nu\ @\ E$. The solution is minimal if for any other solution $(\theta' \fatsemi \Theta_0 \sqsubseteq \Theta'; E')$, there exists a metasubstitution $\xi \fatsemi \Theta_1 \sqsubseteq \Theta'$ such that $\theta' \equiv \xi\theta \fatsemi \Theta_0 \sqsubseteq \Theta'$ and $E \leqslant E'$.
- A type inference question is a tuple $(\Theta_0; M : \bigcirc\ @\ \bigcirc)$ where $\vdash \Theta_0\ \mathbf{ng}$, $\Theta_0 \vdash M\ \mathbf{ok}$, and $\vdash M\ \mathbf{pos}$. A solution to it is a tuple $(\theta \fatsemi \Theta_0 \sqsubseteq \Theta_1; A; E)$ such that $\Theta_1 \vdash M : A\ @\ E$. The solution is minimal if for any other solution $(\theta' \fatsemi \Theta_0 \sqsubseteq \Theta'; A'; E')$, there exists a metasubstitution $\xi \fatsemi \Theta_1 \sqsubseteq \Theta'$ such that $\theta' \equiv \xi\theta \fatsemi \Theta_0 \sqsubseteq \Theta'$ and $\Theta' \vdash \xi A \equiv A'$ and $E \leqslant E'$.

We define the algorithm for solving the questions we need in Figure 18. Note that for some judgements, we only need their declarative forms.

The algorithms for kinding uses the following auxiliary definitions.

$$\begin{aligned}\mathsf{res}(\mathsf{Abs}) &= \mathsf{i}\\ \mathsf{res}(\mathsf{Any}) &= \mathsf{i}\\ \mathsf{res}(\mathsf{Eff}) &= \mathsf{m}\end{aligned}$$

$$K \sqcap K' = \begin{cases}\mathsf{fail}, & \text{if } K = \mathsf{Eff} \text{ or } K' = \mathsf{Eff}\\ \mathsf{Abs}, & \text{if } K = \mathsf{Abs} \text{ or } K' = \mathsf{Abs}\\ \mathsf{Any}, & \text{otherwise}\end{cases}$$

$$K \sqcap K' = \begin{cases}\mathsf{fail}, & \text{if } K = \mathsf{Eff} \text{ or } K' = \mathsf{Eff}\\ \mathsf{Abs}, & \text{if } K = \mathsf{Abs} \text{ or } K' = \mathsf{Abs}\\ \mathsf{Any}, & \text{otherwise}\end{cases}$$

We define the algorithm for unification in Figures 19 and 20. Note that unification $\Theta \vdash A \equiv B \dashv \Theta'$ is only defined for statements $A \equiv B$ satisfying $\vdash A$ **pos** and $\vdash B$ **pos**. We will show later that during type inference no negative effects would appear in types, as long as the input context and terms also satisfy the restriction of no negative effects.

We list some important lemmas here which show the soundness, generality, and completeness of kinding and unification.

Lemma B.4 (Soundness and generality of kind restriction). *If $\Theta_0 \vdash A : (K, R) \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ is a minimal solution of $(\Theta_0; A : (K, R))$*

Lemma B.5 (Completeness of kind restriction). *If $\theta \,⨾\, \Theta_0 \sqsubseteq \Theta$ is a solution to the kinding question $(\Theta_0; A : (K, R))$, then there exists Θ_1 such that $\Theta_0 \vdash A : (K, R) \dashv \Theta_1$.*

Lemma B.6 (Soundness and generality of unification).

1. *If $\Theta_0 \vdash A \equiv B \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ is a minimal solution of $(\Theta_0; A \equiv B)$.*
2. *If $\Theta_0 \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta_1$, A is not a flexible variable, and Ξ only contains declaration of flexible variables appearing in A, then $\Theta_0, \Xi \sqsubseteq \Theta_1$ is a minimal solution of $(\Theta_0; \alpha \equiv A)$.*

Lemma B.7 (Completeness of unification).

1. *If $\theta \,⨾\, \Theta_0 \sqsubseteq \Theta$ is a solution to the unification question $(\Theta_0; A \equiv B)$, then there exists Θ_1 such that $\Theta_0 \vdash A \equiv B \dashv \Theta_1$.*
2. *If $\theta \,⨾\, \Theta_0, \Xi \sqsubseteq \Theta$ is a solution to the unification question $(\Theta_0, \Xi; \hat{\alpha} \equiv A)$, then there exists Θ_1 such that $\Theta_0 \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta_1$.*

B.5 Type Inference

Figure 22 gives type inference algorithm for Metel. It is also in the form of algorithmic moving between contexts $\Theta_0 \vdash M : A \,@\, E \dashv \Theta_1$.

We define $\mathsf{solve}(\mu : E \to F)$ and $\mathsf{solve}(\mu \Rightarrow \nu)$ in Figure 21 which find the minimal index for certain modality and transformation to hold.

The following lemmas show their soundness, generality, and completeness.

Lemma B.8 (Soundness and generality of modality solving).

(1) *If $\mathsf{solve}(\mu : E \to F) = F_1$, then $\mu_{F_1} : E_1 \to F_1$ with $E \leqslant E_1$ and $F \leqslant F_1$. Moreover, for any other $\mu_{F_2} : E_2 \to F_2$ with $F_2 \leqslant F_1$ and $F_2 \not\equiv F_1$, either $E \leqslant E_2$ or $F \leqslant F_2$ does not hold.*

(2) *If $\mathsf{solve}(\mu \Rightarrow \nu) = F$, then $\mu_F \Rightarrow \nu_F$. Moreover, for any other $F' \leqslant F$ with $F' \not\equiv F$, the relation $\mu_{F'} \Rightarrow \nu_{F'}$ does not hold.*

$$\boxed{\Theta \vdash \forall\Delta.A \leq_R B \dashv \Theta'}$$

$$\frac{}{\Theta \vdash A \leq_R A \dashv \Theta} \qquad \frac{\Theta, \hat{\alpha} : (K, R) \vdash \sigma[\hat{\alpha}/\alpha] \leq_R A \dashv \Theta'}{\Theta \vdash \forall \alpha^K.\sigma \leq_R A \dashv \Theta'}$$

$$\boxed{\Theta \vdash \sigma : (K, R) \dashv \Theta'}$$

$$\frac{\Theta \ni \alpha : K' \qquad K' \leqslant K \qquad \mathsf{res}(K) \leqslant R}{\Theta \vdash \alpha : (K, R) \dashv \Theta} \qquad \frac{\Theta \vdash A : (K, R) \dashv \Theta''}{\Theta, \hat{\alpha} = A, \Theta' \vdash \hat{\alpha} : (K, R) \dashv \Theta'', \hat{\alpha} = A, \Theta'}$$

$$\frac{}{\Theta, \hat{\alpha} : (K', R'), \Theta' \vdash \hat{\alpha} : (K, R) \dashv \Theta, \hat{\alpha} : (K' \sqcap K, R' \sqcap R), \Theta'} \qquad \frac{\Theta \vdash A : (K, \mathsf{m}) \dashv \Theta'}{\Theta \vdash \langle L|D \rangle A : (K, \mathsf{m}) \dashv \Theta'}$$

$$\frac{\Theta \vdash E : (\mathsf{Eff}, \mathsf{m}) \dashv \Theta' \qquad \Theta' \vdash A : (\mathsf{Any}, \mathsf{m}) \dashv \Theta''}{\Theta \vdash [E]A : (\mathsf{Abs}, \mathsf{m}) \dashv \Theta''}$$

$$\frac{\Theta \vdash A : (\mathsf{Any}, R) \dashv \Theta_1 \qquad \Theta_1 \vdash B : (\mathsf{Any}, R) \dashv \Theta_2}{\Theta \vdash A \to B : (\mathsf{Any}, R) \dashv \Theta_2} \qquad \frac{\Theta, \alpha : K' \vdash \sigma : (K, R) \dashv \Theta', \alpha : K', \Xi}{\Theta \vdash \forall \alpha^{K'}.\sigma : (K, R) \dashv \Theta', \Xi}$$

$$\frac{}{\Theta \vdash \cdot : (\mathsf{Eff}, \mathsf{m}) \dashv \Theta} \qquad \frac{\Theta \vdash E : (\mathsf{Eff}, \mathsf{m}) \dashv \Theta'}{\Theta \vdash l, E : (\mathsf{Eff}, \mathsf{m}) \dashv \Theta'}$$

$$\boxed{\Theta \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E \dashv \Theta'}$$

$$\frac{F = \mathsf{solve}(\mu \Rightarrow \nu) \qquad \nu_F : E \to F}{\Theta \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E \dashv \Theta} \qquad \frac{\mathsf{solve}(\mu \Rightarrow \nu)\ \mathsf{fails} \qquad \Theta \vdash \sigma : (\mathsf{Abs}, \mathsf{m}) \dashv \Theta'}{\Theta \vdash (\mu, \sigma) \Rightarrow \nu \dashv \Theta'}$$

$$\boxed{\Theta \vdash (M; \nu; \phi; \Delta; A) \Updownarrow (\xi, \sigma) \dashv \Theta'} \ \boxed{\Theta \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma \dashv \Theta'} \ \boxed{\Theta \vdash (M; \Delta; A) \Downarrow \sigma \dashv \Theta'}$$

$$\frac{M \in \mathsf{Val} \qquad \xi = \begin{cases} \nu \circ \mu, & \phi = \mu \\ \nu, & \phi = \cdot \end{cases} \qquad \phi \neq \cdot \text{ or } \nu = \mathbb{1}}{\Theta \vdash (M; \nu; \phi; \Xi; A) \Updownarrow (\xi, \mathsf{gen}(\Xi; A)) \dashv \Theta}$$

$$\frac{M \notin \mathsf{Val} \qquad \nu = \mathbb{1} \qquad \Theta \vdash (M; \Xi; A) \Downarrow B \dashv \Theta' \qquad \xi = \begin{cases} \mu, & \phi = \mu \\ \mathbb{1}, & \phi = \cdot \end{cases}}{\Theta \vdash (M; \nu; \phi; \Xi; A) \Updownarrow (\xi, B) \dashv \Theta'}$$

$$\frac{\Theta \vdash \mathsf{gen}(\Xi; A) \leq_{\mathsf{i}} B \dashv \Theta'}{\Theta \vdash (M; \Xi; A) \Downarrow B \dashv \Theta'} \qquad \frac{M \in \mathsf{Val}}{\Theta \vdash (M; \Delta; A) \Updownarrow^{\dagger} \forall \Delta.A \dashv \Theta} \qquad \frac{M \notin \mathsf{Val} \qquad \Delta = \cdot}{\Theta \vdash (M; \Delta; A) \Updownarrow^{\dagger} \forall \Delta.A \dashv \Theta}$$

Fig. 18. Algorithmic moving between contexts.

$$\boxed{\Theta \vdash A \equiv B \dashv \Theta'}$$

U-RIGID-RIGID
$$\frac{\Theta \ni \alpha : K}{\Theta \vdash \alpha \equiv \alpha \dashv \Theta}$$

U-FLEX-FLEX-ID
$$\frac{\Theta \ni \hat{\alpha} : (K, R)}{\Theta \vdash \hat{\alpha} \equiv \hat{\alpha} \dashv \Theta}$$

U-FLEX-FLEX-L
$$\frac{\hat{\alpha} \neq \hat{\beta} \qquad \Theta \vdash \hat{\beta} : (K, R) \dashv \Theta'}{\Theta, \hat{\alpha} : (K, R) \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', \hat{\alpha} = \hat{\beta}}$$

U-FLEX-FLEX-R
$$\frac{\hat{\alpha} \neq \hat{\beta} \qquad \Theta \vdash \hat{\alpha} : (K, R) \dashv \Theta'}{\Theta, \hat{\beta} : (K, R) \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', \hat{\beta} = \hat{\alpha}}$$

U-FLEX-FLEX-SUBST
$$\frac{\Theta \vdash \hat{\alpha}[A/\hat{\gamma}] \equiv \hat{\beta}[A/\hat{\gamma}] \dashv \Theta'}{\Theta, \hat{\gamma} = A \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', \hat{\gamma} = A}$$

U-FLEX-FLEX-SKIPFLEX
$$\frac{\hat{\gamma} \neq \hat{\alpha} \qquad \hat{\gamma} \neq \hat{\beta} \qquad \Theta \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta'}{\Theta, \hat{\gamma} : (K, R) \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', \hat{\gamma} : (K, R)}$$

U-FLEX-FLEX-SKIPRIGID
$$\frac{\Theta \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta'}{\Theta, \gamma : K \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', \gamma : K}$$

U-FLEX-FLEX-SKIPTERM
$$\frac{\Theta \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta'}{\Theta, x : \sigma \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', x : \sigma}$$

U-FLEX-FLEX-SKIPLOCK
$$\frac{\Theta \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta'}{\Theta, \text{🔒}_{\mu} \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta', \text{🔒}_{\mu}}$$

U-FLEX-FLEX-SKIPMARK
$$\frac{\Theta \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta'}{\Theta \fatsemi \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta' \fatsemi}$$

U-FLEX-RIGID-L
$$\frac{A \text{ non-flex-var} \qquad \Theta \mid \cdot \vdash \hat{\alpha} := A \dashv \Theta'}{\Theta \vdash \hat{\alpha} \equiv A \dashv \Theta'}$$

U-FLEX-RIGID-R
$$\frac{A \text{ non-flex-var} \qquad \Theta \mid \cdot \vdash \hat{\alpha} := A \dashv \Theta'}{\Theta \vdash A \equiv \hat{\alpha} \dashv \Theta'}$$

U-MOD
$$\frac{\Theta \vdash A \equiv A' \dashv \Theta' \qquad \Theta' \vdash \mu \equiv \mu' \dashv \Theta''}{\Theta \vdash \mu A \equiv \mu' A' \dashv \Theta'}$$

U-ARROW
$$\frac{\Theta \vdash A \equiv A' \dashv \Theta' \qquad \Theta' \vdash B \equiv B' \dashv \Theta''}{\Theta \vdash A \to B \equiv A' \to B' \dashv \Theta''}$$

U-RELATIVE
$$\frac{L \equiv L' \qquad D \equiv D'}{\Theta \vdash \langle L|D \rangle \equiv \langle L'|D' \rangle \dashv \Theta}$$

U-ABSOLUTE
$$\frac{\Theta \vdash E \equiv F \dashv \Theta'}{\Theta \vdash [E] \equiv [F] \dashv \Theta'}$$

U-EFFECT-CLOSED
$$\frac{L \equiv L'}{\Theta \vdash L \equiv L' \dashv \Theta}$$

U-EFFECT-L
$$\frac{\begin{array}{c} L' = \mathsf{labels}(E) \qquad \Theta \vdash E : (\mathsf{Eff}, \mathsf{m}) \dashv \Theta \\ L \subseteq L' \qquad \Theta' \mid \cdot \vdash \hat{\varepsilon} := E - L \dashv \Theta' \end{array}}{\Theta \vdash L; \hat{\varepsilon} \equiv E \dashv \Theta'}$$

U-EFFECT-R
$$\frac{\begin{array}{c} L' = \mathsf{labels}(E) \qquad \Theta \vdash E : (\mathsf{Eff}, \mathsf{m}) \dashv \Theta \\ L \subseteq L' \qquad \Theta \mid \cdot \vdash \hat{\varepsilon} := E - L \dashv \Theta' \end{array}}{\Theta \vdash E \equiv L; \hat{\varepsilon} \dashv \Theta'}$$

U-EFFECT-LR
$$\frac{L_1 \nsubseteq L_2 \qquad L_2 \nsubseteq L_1 \qquad \Theta, \hat{\varepsilon} \vdash \hat{\varepsilon}_1 := L_2 - L_1, \hat{\varepsilon} \dashv \Theta_1 \qquad \Theta_1 \vdash \hat{\varepsilon}_2 := L_1 - L_2, \hat{\varepsilon} \dashv \Theta_2}{\Theta \vdash L_1, \hat{\varepsilon}_1 \equiv L_2, \hat{\varepsilon}_2 \dashv \Theta_2}$$

Fig. 19. Unification (Part I).

PROOF. 1. When μ is absolute, trivial. When μ is relative, the delta between the source and target is fixed and we only need to case analysis whether E or F gives the lower bound.

2. When μ is absolute, the minimal index for the transformation is completely determined by ν. Otherwise, relative μ can only be transformed to relative ν. The delta between L_1 and D_1 must be the same as the delta between L_2 and D_2 in order to make the transformation hold. The index is determined by the larger one among L_1 and L_2. □

$\boxed{\Theta \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}$

$$\textsc{U-Flex-Rigid-Solve}\quad \frac{\Theta, \Xi \vdash A : (K, R) \dashv \Theta'}{\Theta, \hat{\alpha} : (K, R) \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta', \hat{\alpha} = A}$$

$$\textsc{U-Flex-Rigid-Subst}\quad \frac{\Theta, \Xi \vdash \hat{\alpha}[B/\hat{\beta}] \equiv A[B/\hat{\beta}] \dashv \Theta'}{\Theta, \hat{\beta} = B \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta', \hat{\beta} = B}$$

$$\textsc{U-Flex-Rigid-Depend}\quad \frac{\hat{\alpha} \neq \hat{\beta} \qquad \hat{\beta} \in \mathsf{ftv}(A) \qquad \Theta \mid \hat{\beta} : (K, R), \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}{\Theta, \hat{\beta} : (K, R) \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}$$

$$\textsc{U-Flex-Rigid-SkipFlex}\quad \frac{\hat{\alpha} \neq \hat{\beta} \qquad \hat{\beta} \notin \mathsf{ftv}(A) \qquad \Theta \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}{\Theta, \hat{\beta} : (K, R) \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta', \hat{\beta} : (K, R)}$$

$$\textsc{U-Flex-Rigid-SkipRigid}\quad \frac{\beta \notin \mathsf{ftv}(A) \qquad \Theta \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}{\Theta, \beta : K \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta', \beta : K}$$

$$\textsc{U-Flex-Rigid-SkipTerm}\quad \frac{\Theta \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}{\Theta, x : \sigma \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta', x : \sigma}$$

$$\textsc{U-Flex-Rigid-SkipLock}\quad \frac{\Theta \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}{\Theta, \text{🔒}_{\mu} \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta', \text{🔒}_{\mu}}$$

$$\textsc{U-Flex-Rigid-SkipMark}\quad \frac{\Theta \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta'}{\Theta \fatsemi \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta' \fatsemi}$$

Fig. 20. Unification (Part II).

$$\begin{aligned}
\mathsf{solve}([E'] : E \to F) &= \begin{cases} F, & E \leqslant E' \\ \mathsf{fail}, & \text{otherwise} \end{cases} \\
\mathsf{solve}(\langle L | D \rangle : E \to F) &= \begin{cases} F, & E \leqslant D + (F - L) \\ L + (E - D), & \text{otherwise} \end{cases} \\
\mathsf{solve}([E] \Rightarrow \nu) &= \mathsf{solve}(\nu : E \to \cdot) \\
\mathsf{solve}(\langle L | D \rangle \Rightarrow [E]) &= \mathsf{fail} \\
\mathsf{solve}(\langle L_1 | D_1 \rangle \Rightarrow \langle L_2 | D_2 \rangle) &= \begin{cases} \mathsf{fail}, & (L, D) \neq (L', D') \\ & \text{where } (L, D) = L_1 \bowtie D_1 \text{ and } (L', D') = L_2 \bowtie D_2 \\ L_2, & L_1 \leqslant L_2 \\ L_1, & \text{otherwise} \end{cases}
\end{aligned}$$

Fig. 21. Solvers for modalities.

Lemma B.9 (Completeness of solving).

- *If* $\mu_{F'} : E' \to F'$ *with* $E \leqslant E'$ *and* $F \leqslant F'$, *then* $\mathsf{solve}(\mu : E \to F) = F''$ *for some* F''.
- *If* $\mu_F \Rightarrow \nu_F$, *then* $\mathsf{solve}(\mu \Rightarrow \nu) = F'$ *for some* F'.

Proof. By definition. □

We prove that type inference does not generate negative effects.

Lemma B.10 (No negative effects). *For the type inference question* $(\Theta_0; M : \bigcirc \,@\, \bigcirc)$ *with the implicit condition* $\vdash \Theta_0$ **pos** *and* $\vdash M$ **pos**, *if* $\Theta_0 \vdash M : A \,@\, E \dashv \Theta_1$, *then* $\vdash \Theta_1$ **pos** *and* $\vdash A$ **pos**.

This lemma guarantees that though the unification algorithm is not defined for negative effects, it would not fail because of negative effects during type inference.

We prove the soundness, generality, and completeness for type inference.

Theorem B.11 (Soundness and generality of type inference). *For the type inference question* $(\Theta_0; M : \bigcirc \mathbin{@} \bigcirc)$, *if* $\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1$, *then* $(\Theta_0 \sqsubseteq \Theta_1, A, E)$ *is a minimal solution.*

Theorem B.12 (Completeness of Type Inference). *If* $\vdash \Theta_0\ \mathbf{ng}$, $\Theta_0 \vdash M\ \mathbf{ok}$, $\theta \fatsemi \Theta_0 \sqsubseteq \Theta$, *and* $\Theta \vdash_s M : A \mathbin{@} F$, *then* $\Theta_0 \vdash M : B \mathbin{@} E \dashv \Theta_1$ *for some* Θ_1, B, *and* E.

All proofs can be found in Appendix C.

$$\boxed{\Theta \vdash M : A \mathbin{@} E \dashv \Theta'}$$

I-Freeze

$$\frac{\xi = \mathsf{alocks}(\Theta_0) \qquad \forall\Delta.A = \mathsf{subst}(\Theta;\sigma) \qquad \Theta,\Theta_0 \vdash (\mu, \forall\Delta.A) \Rightarrow \xi \mathbin{@} E \dashv \Theta_1 \qquad \Theta_1 \vdash \forall\Delta.A \preceq_{\mathrm{m}} B \dashv \Theta_2}{\Theta, x :_\mu \sigma, \Theta_0 \vdash \lceil x \rceil : B \mathbin{@} E \dashv \Theta_2}$$

I-Var

$$\frac{\xi = \mathsf{alocks}(\Theta_0) \qquad \forall\Delta.A = \mathsf{subst}(\Theta;\sigma) \qquad (\nu, A') = \mathsf{split}(\Delta, A) \qquad \Theta,\Theta_0 \vdash (\mu\circ\nu, \forall\Delta.A') \Rightarrow \xi \mathbin{@} E \dashv \Theta_1 \qquad \Theta_1 \vdash \forall\Delta.A \preceq_{\mathrm{m}} B \dashv \Theta_2}{\Theta, x :_\mu \sigma, \Theta_0 \vdash x : B \mathbin{@} E \dashv \Theta_2}$$

I-Mod

$$\frac{\Theta_0, \text{🔒}_\mu \vdash V : A \mathbin{@} E \dashv \Theta_1, \text{🔒}_\mu, \Xi_1 \qquad F = \mathsf{solve}(\mu : E \to \cdot)}{\Theta_0 \vdash \mathbf{mod}_\mu\, V : \mu A \mathbin{@} F \dashv \Theta_1, \Xi_1}$$

I-App

$$\frac{\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1 \qquad \Theta_1 \vdash N : B \mathbin{@} F \dashv \Theta_2 \qquad \Theta_2, \hat{\alpha} : (\mathsf{Any}, \mathrm{m}) \vdash A \equiv B \to \hat{\alpha} \dashv \Theta_3}{\Theta_0 \vdash M\, N : \hat{\alpha} \mathbin{@} E \cup F \dashv \Theta_3}$$

I-AbsAnno

$$\frac{\Theta_0, x : A \vdash M : B \mathbin{@} E \dashv \Theta_1, x : A, \Xi_1}{\Theta_0 \vdash \lambda x^A.M : A \to B \mathbin{@} E \dashv \Theta_1, \Xi_1}$$

I-Abs

$$\frac{\Theta_0, \hat{\alpha} : (\mathsf{Any}, \mathrm{i}), x : \hat{\alpha} \vdash M : B \mathbin{@} E \dashv \Theta_1, x : \hat{\alpha}, \Xi_1}{\Theta_0 \vdash \lambda x.M : \hat{\alpha} \to B \mathbin{@} E \dashv \Theta_1, \Xi_1}$$

I-LetAnno

$$\frac{\Theta_0 \vdash (M;\Delta;A) \Updownarrow^\dagger \sigma \dashv \Theta_1 \qquad \Theta_1 \mathbin{⨾} \Delta \vdash M : A' \mathbin{@} E \dashv \Theta_2 \mathbin{⨾} \Delta, \Xi_2 \qquad \Theta_2 \mathbin{⨾} \Delta, \Xi_2 \vdash A' \equiv A \dashv \Theta_3 \mathbin{⨾} \Delta, \Xi_3 \qquad \Theta_3, x : \sigma \vdash N : B \mathbin{@} F \dashv \Theta_4, x : \sigma, \Xi_4}{\Theta_0 \vdash \mathbf{let}\ x^{\forall\Delta.A} = M\ \mathbf{in}\ N : B \mathbin{@} E \cup F \dashv \Theta_4, \Xi_4}$$

I-LetMod

$$\frac{\Theta_0, \text{🔒}_\nu \vdash M : A \mathbin{@} E \dashv \Theta_1, \text{🔒}_\nu, \Xi_1 \qquad \Theta_1 \mathbin{⨾} \Xi_1, \hat{\alpha} : (\mathsf{Any}, \mathrm{m}) \vdash A \equiv \phi\hat{\alpha} \dashv \Theta_2 \mathbin{⨾} \Xi_2 \qquad \Theta_2 \vdash (M;\nu;\phi;\Xi_2;\hat{\alpha}) \Updownarrow (\xi, \sigma) \dashv \Theta_3 \qquad \Theta_3, x :_\xi \sigma \vdash N : B \mathbin{@} F \dashv \Theta_4 \qquad F' = \mathsf{solve}(\nu : E \to F)}{\Theta_0 \vdash \mathbf{let}_\nu\ \phi\ x = M\ \mathbf{in}\ N : B \mathbin{@} F' \dashv \Theta_3}$$

I-Do

$$\frac{\Sigma \ni \ell : A \twoheadrightarrow B \qquad \Theta_0 \vdash M : A_1 \mathbin{@} E \dashv \Theta_1 \qquad \Theta_1 \vdash A_1 \equiv A \dashv \Theta_2}{\Theta_0 \vdash \mathbf{do}\ \ell\ M : B \mathbin{@} \{\ell\} \cup E \dashv \Theta_2}$$

I-Mask

$$\frac{\Theta_0, \text{🔒}_{\langle L|\rangle} \vdash M : A \mathbin{@} E \dashv \Theta_1 \qquad F = \mathsf{solve}(\langle L|\rangle : E \to \cdot)}{\Theta_0 \vdash \mathbf{mask}_L\ M : \langle L|\rangle A \mathbin{@} F \dashv \Theta_2}$$

I-Handler

$$\frac{\begin{array}{c} D = \{\ell_i\}_i \qquad \{\ell_i : A_i \twoheadrightarrow B_i\} \subseteq \Sigma \\ \Theta, \text{🔒}_{\langle|D\rangle} \vdash M : A_0 \mathbin{@} E' \dashv \Theta', \text{🔒}_{\langle|D\rangle}, \Xi' \qquad \Theta' \vdash (M;\Xi';A_0) \Downarrow A \dashv \Theta_0 \\ \Theta_0, x : \langle|D\rangle A \vdash N : B_0 \mathbin{@} E_r \dashv \Theta'_0, x : \_, \Xi'_0 \qquad \Theta'_0 \vdash (N;\Xi'_0;B_0) \Downarrow B \dashv \Theta_1 \\ [\Theta_i, p_i : A_i, r_i : B_i \to B \vdash N_i : B_i \mathbin{@} E_i \dashv \Theta'_i, p_i : \_, r_i : \_, \Xi'_i \qquad \Theta'_i, \Xi'_i \vdash B_i \equiv B \dashv \Theta_{i+1}]_{i=1}^n \\ E = \mathsf{solve}(\langle|D\rangle : E' \to \cdot) \qquad F = E \cup E_r \cup (\cup_i E_i) \end{array}}{\Theta \vdash \mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_{i=1}^n : B \mathbin{@} F \dashv \Theta_{n+1}}$$

Fig. 22. Type inference for Metel.

B.6 Elaboration to the Core Calculus

The semantics of Metel is given by its elaboration to Mete.

We first fill the gap between Metel and Mete that contexts of Metel do not keep indexes of modalities appearing in locks and variable bindings. Observe that for any typing judgement of closed terms $\vdash_s M : A \mathbin{@} E$, the indexes of modalities for locks and bindings in contexts are completely determined by the derivation tree. We derive a new presentation of the declarative type system for Metel by keeping indexes of locks and bindings in context, and modify auxiliary relations involving modalities to consider indexes as well. The interesting typing rules for the new typing judgement $\Gamma \vdash_{si} M : A \mathbin{@} E$ and auxiliary relations are defined in the non-highlighted parts of Figures 23 to 25. We inline the relation $(M; \nu; \phi; \Delta; A)$ and split T-Unmod into four rules for clarity of elaboration. The following lemma shows the equivalence between the two type systems.

Lemma B.13 (Indexes in contexts can be ignored). *$\vdash_{si} M : A \mathbin{@} E$ if and only if $\vdash_s M : A \mathbin{@} E$.*

The proof follows from straightforward induction on the typing derivations. The only non-trivial case is to show the equivalence of typing rules for variables, since they use the modality transformation relation in different ways. The following lemma shows that the modality transformation relation holds regardless of the targets of modalities.

Lemma B.14 (Source determines transformation). *If $\nu_F : E \to F$ and $\mu_F \Rightarrow \nu_F$, then for any F' such that $\nu_{F'} : E \to F'$, we have $\mu_{F'} \Rightarrow \nu_{F'}$.*

Proof. If $\nu = [E]$, we have $\mu = [E']$ where $E' \leqslant E$. Otherwise, we can show $F = F'$. □

As a corollary, for $\Gamma \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E$, we know that either $\Gamma \vdash \sigma : \mathsf{Abs}$ or $\mu_F \to \nu_F$ for any F with $\nu_F : E \to F$. The reverse direction also holds. This gives the equivalence of the variable rules.

Since the new type system is equivalent to the old one, and it is obvious to derive a derivation tree of the new typing judgement from the old one for closed terms, we restrict elaboration to closed terms and directly define the elaboration on the derivation tree of the new judgement $\Gamma \vdash_{si} M : A \mathbin{@} E$. The elaboration is given as the highlighted parts of Figures 23 to 25. There is nothing really surprising in the elaboration. For all terms that introduce variable bindings x, we immediately unbox it and bind the unboxed result to $\hat{x}$. For variable rules, we use the original x for froze variables, and unboxed $\hat{x}$ for usual variables which are automatically unboxed. Also, in variable, let-binding rules and handler rules, we deal with generalisation and instantiation. The following theorem showing the type preservation. Its proof follows from straightforward induction.

Theorem B.15 (Type preservation). *If $\Gamma \vdash_{si} M : A \mathbin{@} E \dashrightarrow M'$, then $\Gamma \vdash M : A \mathbin{@} E$.*

$$\frac{\mu_F \Rightarrow \nu_F \text{ or } \Gamma \vdash \sigma : \mathsf{Abs}}{\Gamma \vdash (\mu_F, \sigma) \Rightarrow \nu_F \mathbin{@} E} \qquad \frac{\mathsf{principal}(\Gamma; M; \Delta; A) \qquad \Gamma \vdash \forall \Delta.A \leq_{\mathsf{i}} B \dashrightarrow \overline{A'}}{\Gamma \vdash (M; \Delta; A) \Downarrow B \dashrightarrow \overline{A'}}$$

$$\frac{}{\Gamma \vdash A \leq_R A \dashrightarrow \cdot} \qquad \frac{\Gamma \vdash B : (K, R) \qquad \Gamma \vdash \sigma[B/\alpha] \leq_R A \dashrightarrow \overline{A'}}{\Gamma \vdash \forall \alpha^K.\sigma \leq_R A \dashrightarrow B, \overline{A'}}$$

$$\mathsf{unmod}(x; \Delta; A; M) \quad = \quad \textbf{let mod}_\nu\ \Lambda\Delta.\hat{x} = x\ \Delta\ \textbf{in}\ M \quad \text{where } (\nu, \_) = \mathsf{split}(A)$$

Fig. 23. Auxiliary definitions for Metel with indexed contexts and its elaboration.

T-Freeze

$$\frac{\xi_F = \mathsf{alocks}(\Gamma') \qquad \Gamma, \Gamma' \vdash (\mu_F, \forall\Delta.A) \Rightarrow \xi_F \mathbin{@} E \qquad \Gamma, \Gamma' \vdash \forall\Delta.A \leq_{\mathsf{m}} B \dashrightarrow \overline{A'}}{\Gamma, x :_{\mu_F} \forall\Delta.A, \Gamma' \vdash_{si} \lceil x \rceil \dashrightarrow x\ \overline{A'} : B \mathbin{@} E}$$

T-Var

$$\frac{\begin{array}{c} \xi_F = \mathsf{alocks}(\Gamma') \qquad \mu_F : F' \to F \\ (v, A') = \mathsf{split}(\Delta; A) \qquad \Gamma, \Gamma' \vdash (\mu_F \circ \nu_{F'}, \forall\Delta.A') \Rightarrow \xi_F \mathbin{@} E \qquad \Gamma, \Gamma' \vdash \forall\Delta.A' \leq_{\mathsf{m}} B \dashrightarrow \overline{A'} \end{array}}{\Gamma, x :_{\mu_F} \forall\Delta.A, \Gamma' \vdash_{si} x \dashrightarrow \hat{x}\ \overline{A'} : B \mathbin{@} E}$$

T-AbsAnno

$$\frac{\Gamma, x : A \vdash_{si} M : B \mathbin{@} E \dashrightarrow M'}{\begin{array}{c} \Gamma \vdash_{si} \lambda x^A.M \\ \dashrightarrow \lambda x^A.\mathsf{unmod}(x; \cdot; A; M') : A \to B \mathbin{@} E \end{array}}$$

T-Abs

$$\frac{\Gamma, x : S \vdash_{si} M : B \mathbin{@} E \dashrightarrow M'}{\begin{array}{c} \Gamma \vdash_{si} \lambda x.M \\ \dashrightarrow \lambda x^S.\mathsf{unmod}(x; \cdot; S; M') : S \to B \mathbin{@} E \end{array}}$$

T-LetmodVal

$$\frac{\begin{array}{c} \nu_F : E \to F \qquad \xi_F = \nu_F \circ \mu_E \\ \Gamma, \text{🔒}_{\nu_F}, \Delta \vdash_{si} V : \mu A \mathbin{@} E \dashrightarrow V' \\ \Gamma, x :_{\xi_F} \forall\Delta.A \vdash_{si} N : B \mathbin{@} F \dashrightarrow N' \\ N'' = \mathsf{unmod}(x; \Delta; A; N') \end{array}}{\begin{array}{c} \Gamma \vdash_{si} \mathbf{let}_\nu\ \mu\ x = V\ \mathbf{in}\ N \\ \dashrightarrow \mathbf{let}_\nu\ \mathbf{mod}_\mu\ \Lambda\Delta.x = V'\ \mathbf{in}\ N'' : B \mathbin{@} F \end{array}}$$

T-LetmodNonval

$$\frac{\begin{array}{c} M \notin \mathsf{Val} \qquad (M; \Delta; A) \Downarrow A' \dashrightarrow \overline{A_1} \\ \Gamma, \text{🔒}_{\mathbb{1}_F}, \Delta \vdash_{si} M : \mu A \mathbin{@} E \dashrightarrow M' \\ \Gamma, x :_{\mu_F} A' \vdash_{si} N : B \mathbin{@} F \dashrightarrow N' \\ N'' = \mathsf{unmod}(x; \cdot; A'; N') \end{array}}{\begin{array}{c} \Gamma \vdash_{si} \mathbf{let}_{\mathbb{1}}\ \mu\ x = M\ \mathbf{in}\ N \\ \dashrightarrow \mathbf{let}_{\mathbb{1}}\ \mathbf{mod}_\mu\ x = M'[\overline{A_1}/\Delta]\ \mathbf{in}\ N'' : B \mathbin{@} F \end{array}}$$

T-LetVal

$$\frac{\begin{array}{c} \Gamma, \text{🔒}_{\mathbb{1}_F}, \Delta \vdash_{si} V : A \mathbin{@} E \dashrightarrow V' \\ \Gamma, x :_{\mathbb{1}_F} \forall\Delta.A \vdash_{si} N : B \mathbin{@} F \dashrightarrow N' \\ N'' = \mathsf{unmod}(x; \Delta; A; N') \end{array}}{\begin{array}{c} \Gamma \vdash_{si} \mathbf{let}_{\mathbb{1}}\ \ x = V\ \mathbf{in}\ N \\ \dashrightarrow \mathbf{let}\ x = \Lambda\Delta.V'\ \mathbf{in}\ N'' : B \mathbin{@} F \end{array}}$$

T-LetNonval

$$\frac{\begin{array}{c} M \notin \mathsf{Val} \qquad (M; \Delta; A) \Downarrow A' \dashrightarrow \overline{A_1} \\ \Gamma, \text{🔒}_{\mathbb{1}_F}, \Delta \vdash_{si} M : A \mathbin{@} E \dashrightarrow M' \\ \Gamma, x :_{\mathbb{1}_F} A' \vdash_{si} N : B \mathbin{@} F \dashrightarrow N' \\ N'' = \mathsf{unmod}(x; \cdot; A'; N') \end{array}}{\begin{array}{c} \Gamma \vdash_{si} \mathbf{let}_{\mathbb{1}}\ \ x = M\ \mathbf{in}\ N \\ \dashrightarrow \mathbf{let}\ x = M'[\overline{A_1}/\Delta]\ \mathbf{in}\ N'' : B \mathbin{@} F \end{array}}$$

T-LetAnno

$$\frac{\begin{array}{c} \Gamma \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma \qquad \Gamma, \Delta \vdash_{si} M : A \mathbin{@} E \dashrightarrow M' \\ \Gamma, x : \sigma \vdash_{si} N : B \mathbin{@} E \dashrightarrow N' \end{array}}{\Gamma \vdash_{si} \mathbf{let}\ x^{\forall\Delta.A} = M\ \mathbf{in}\ N \dashrightarrow \mathbf{let}\ x = \Lambda\Delta.M'\ \mathbf{in}\ N' : B \mathbin{@} E}$$

T-Handler

$$\frac{\begin{array}{c} D = \{\ell_i\}_i \qquad \{\ell_i : A_i \twoheadrightarrow B_i\} \subseteq \Sigma \\ \Gamma \vdash (M; \Delta; A_0) \Downarrow A \dashrightarrow \overline{A_1} \qquad \Gamma, \text{🔒}_{\langle | D \rangle_F}, \Delta \vdash_{si} M : A_0 \mathbin{@} D + F \dashrightarrow M' \\ \Gamma \vdash (N; \Delta'; B_0) \Downarrow B \dashrightarrow \overline{B_1} \qquad \Gamma, x : \langle\!| D \rangle A, \Delta' \vdash_{si} N : B_0 \mathbin{@} F \dashrightarrow N' \\ [\Gamma, p_i : A_i, r_i : B_i \to B \vdash_{si} N_i : B \mathbin{@} F \dashrightarrow N_i']_i \end{array}}{\begin{array}{c} \Gamma \vdash_{si} \mathbf{handle}\ M\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i \\ \dashrightarrow \mathbf{handle}\ M'[\overline{A_1}/\Delta]\ \mathbf{with}\ \{\mathbf{return}\ x \mapsto N'[\overline{B_1}/\Delta']\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i'\}_i : B \mathbin{@} F \end{array}}$$

Fig. 24. Elaboration from Metel with indexed contexts to Mete (part I).

$$\textsc{T-Mod}\quad \frac{\Gamma, \text{🔒}_{\mu} \vdash_{si} V : A \mathbin{@} E \dashrightarrow V' \qquad \mu_F : E \to F}{\Gamma \vdash_{si} \mathbf{mod}_{\mu}\, V \dashrightarrow \mathbf{mod}_{\mu}\, V' : \mu A \mathbin{@} F}$$

$$\textsc{T-App}\quad \frac{\Gamma \vdash_{si} M : A \to B \mathbin{@} E \dashrightarrow M' \qquad \Gamma \vdash_{si} N : A \mathbin{@} E \dashrightarrow N'}{\Gamma \vdash_{si} M\, N \dashrightarrow M'\, N' : B \mathbin{@} E}$$

$$\textsc{T-Mask}\quad \frac{\Gamma, \text{🔒}_{\langle L |\rangle} \vdash_{si} M : A \mathbin{@} F - L \dashrightarrow M'}{\Gamma \vdash_{si} \mathbf{mask}_L\, M \dashrightarrow \mathbf{mask}_L\, M' : \langle L |\rangle A \mathbin{@} F}$$

$$\textsc{T-Do}\quad \frac{\Sigma \ni \ell : A \twoheadrightarrow B \qquad E = \ell, F \qquad \Gamma \vdash_{si} M : A \mathbin{@} E \dashrightarrow M'}{\Gamma \vdash_{si} \mathbf{do}\ \ell\ M \dashrightarrow \mathbf{do}\ \ell\ M' : B \mathbin{@} E}$$

Fig. 25. Elaboration from Metel with indexed contexts to Mete (part II).

C Proofs for METEL

In this section, we prove the soundness and completeness of the type inference of METEL.

C.1 Definitions and Lemmas

Following Gundry [20], we define the notion of stable statements.

Definition C.1 (Stability). A statement J is stable if it is preserved by metasubstitution. Formally, if $\Theta_0 \vdash J$ and $\theta \mathbin{⨾} \Theta_0 \sqsubseteq \Theta_1$, then $\Theta_1 \vdash \theta J$.

All our statements are stable under metasubstitution. Stability allows us to solve sub-questions step-by-step and compose them to the solution of the whole question.

We have the following lemma showing we can compose minimal solutions of sub-questions to obtain the minimal solution of the whole question.

LEMMA C.2 (THE OPTIMIST'S LEMMA). *If $\theta_0 \mathbin{⨾} \Theta_0 \sqsubseteq \Theta_1$ is a minimal solution of J and $\theta_1 \mathbin{⨾} \Theta_1 \sqsubseteq \Theta_2$ is a minimal solution of J', then $\theta_1\theta_0 \mathbin{⨾} \Theta_0 \sqsubseteq \Theta_2$ is a minimal solution of $J \wedge J'$.*

PROOF. Same as Gundry [20]. Any solution *theta* $\mathbin{⨾} \Theta_0 \sqsubseteq \Theta$ to the question $(\Theta_0, J \wedge J')$ should solve (Θ_0, J), thus factor through θ_0 with cofactor $\zeta_0 \mathbin{⨾} \Theta_1 \sqsubseteq \Theta'$. Then ζ_0 should solve $(\Theta_1, \theta_0 J')$, thus factor through θ_1 with cofactor ζ_1. Our goal follow from θ factors through $\theta_1\theta_0$ with cofactor $\zeta_1 \mathbin{⨾} \Theta_2 \sqsubseteq \Theta$ such that $\theta \equiv \zeta_1\theta_1\theta_0 \mathbin{⨾} \Theta_0 \sqsubseteq \Theta$. □

Although this lemma only applies to questions without outputs defined in Definition B.2, we can use similar ideas in proofs for questions with outputs defined in Definition B.3 .

C.2 Unification

LEMMA B.4 (SOUNDNESS AND GENERALITY OF KIND RESTRICTION). *If $\Theta_0 \vdash A : (K, R) \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ is a minimal solution of $(\Theta_0; A : (K, R))$*

PROOF. We want to show that $\Theta_0 \sqsubseteq \Theta_1$, $\Theta_1 \vdash A : (K, R)$, and for any other solution $\theta \mathbin{⨾} \Theta_0 \sqsubseteq \Theta'$, we have $\theta \mathbin{⨾} \Theta_1 \sqsubseteq \Theta'$. By straightforward induction on the judgement $\Theta \vdash A : (K, R) \dashv \Theta'$. The most non-trivial case is when A is a flexible variable.

$$\frac{}{\Theta, \hat{\alpha} : (K', R'), \Theta' \vdash \hat{\alpha} : (K, R) \dashv \Theta, \hat{\alpha} : (K' \sqcap K, R' \sqcap R), \Theta'}$$

Soundness follows from $K' \sqcap K \leqslant K$ and $R' \sqcap R \leqslant R$. Generality follows from that $\hat{\alpha} : (K', R')$ and $\hat{\alpha} : (K, R)$ must both hold for any solution, and the meet operation $\sqcap$ gives the greatest lower bounds. Other cases follow from IHs and Lemma C.2. □

LEMMA B.5 (COMPLETENESS OF KIND RESTRICTION). *If $\theta \mathbin{⨾} \Theta_0 \sqsubseteq \Theta$ is a solution to the kinding question $(\Theta_0; A : (K, R))$, then there exists Θ_1 such that $\Theta_0 \vdash A : (K, R) \dashv \Theta_1$.*

PROOF. Straightforward induction on the declarative kinding judgements. □

LEMMA B.6 (SOUNDNESS AND GENERALITY OF UNIFICATION).

1. *If $\Theta_0 \vdash A \equiv B \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ is a minimal solution of $(\Theta_0; A \equiv B)$.*
2. *If $\Theta_0 \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta_1$, A is not a flexible variable, and Ξ only contains declaration of flexible variables appearing in A, then $\Theta_0, \Xi \sqsubseteq \Theta_1$ is a minimal solution of $(\Theta_0; \alpha \equiv A)$.*

PROOF. For 1, we want to show that $\Theta_1 \vdash A \equiv B$, and for any other $\theta \mathbin{⨾} \Theta_0 \sqsubseteq \Theta'$ with $\Theta' \vdash \theta A \equiv \theta B$, there exists $\zeta \mathbin{⨾} \Theta_1 \sqsubseteq \Theta'$ such that $\theta \equiv \zeta \mathbin{⨾} \Theta_0 \sqsubseteq \Theta'$. For 2, we want to show that $\Theta_1 \vdash A \equiv B$, and for any other $\theta \mathbin{⨾} \Theta_0, \Xi \sqsubseteq \Theta'$ with $\Theta' \vdash \theta\hat{\alpha} \equiv \theta B$, there exists $\zeta \mathbin{⨾} \Theta_1 \sqsubseteq \Theta'$ such that $\theta \equiv \zeta \mathbin{⨾} \Theta_0 \sqsubseteq \Theta'$. We prove 1 and 2 simultaneously by mutual induction on the unification judgement $\Theta_0 \vdash A \equiv B \dashv \Theta_1$ and $\Theta_0 \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta_1$. Similar to the proof of unification in Gundry [20], the key observation is that most unification rules does not introduce new flexible variables, and for all definitions $\hat{\beta} = B$ in Θ_1, we must have $\Theta' \vdash \theta\hat{\beta} \equiv \theta B$ for the problem to be solved. Most cases follow from similar and routine usages of IHs. We only elaborate interesting and representative cases.

Case U-Rigid-Rigid and U-Flex-Flex-Id. Trivial.

Case U-Flex-Flex-SkipMark.

$$\frac{\Theta_0 \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta_1 \ (1)}{\Theta_0 ; \vdash \hat{\alpha} \equiv \hat{\beta} \dashv \Theta_1 ;}$$

For any other solution $\theta ⨾ \Theta_0 ; \sqsubseteq \Theta' ; \Xi$, we have $\theta ⨾ \Theta_0 \sqsubseteq \Theta'$. IH on (1) gives a cofactor ζ such that $\theta \equiv \zeta ⨾ \Theta_0 \sqsubseteq \Theta'$, which further gives $\theta \equiv \zeta ⨾ \Theta_0 ; \sqsubseteq \Theta' ; \Xi$.

Case U-Flex-Flex-L and U-Flex-Flex-R. Follow from IH.

Case U-Flex-Flex-Subst. Follow from IH.

Case U-Flex-Flex-SkipFlex. Follow from IH.

Case U-Flex-Flex-SkipRigid. Follow from IH.

Case U-Flex-Flex-SkipTerm. Follow from IH.

Case U-Flex-Flex-SkipLock. Follow from IH.

Case U-Flex-Rigid-L and U-Flex-Rigid-R. Follow from IH.

Case U-Mod and U-Arrow. Follow from IH and Lemma C.2.

Case U-Relative and U-Effect-Closed. Trivial.

Case U-Absolute. Follow from IH.

Case U-Effect-L and U-Effect-R. Follow from IH.

Case U-Effect-LR.

$$\frac{\begin{array}{cc} L_1 \not\subseteq L_2 & L_2 \not\subseteq L_1 \\ \Theta_0, \hat{\varepsilon} \vdash \hat{\varepsilon}_1 := L_2 - L_1, \hat{\varepsilon} \dashv \Theta_1 \ (1) & \Theta_1 \vdash \hat{\varepsilon}_2 := L_1 - L_2, \hat{\varepsilon} \dashv \Theta_2 \ (2) \end{array}}{\Theta_0 \vdash L_1, \hat{\varepsilon}_1 \equiv L_2, \hat{\varepsilon}_2 \dashv \Theta_2}$$

For any other solution $\theta ⨾ \Theta_0 \sqsubseteq \Theta'$, suppose $\theta\hat{\varepsilon}_1 = E_1$ and $\theta\hat{\varepsilon}_2 = E_2$. Since $L_1, E_1 = L_2, E_2$, there exists E such that $E_1 = L_2 - L_1, E$ and $E_2 = L_1 - L_2, E$. Then by IHs on (1) and (2), and Lemma C.2, we can show that $\zeta = \theta, E/\hat{\varepsilon}$ is the required cofactor.

Case U-Flex-Rigid-Solve. Any other solutions must solve $A : (K, R)$ and $\alpha \equiv A$. Follow from IH.

Case U-Flex-Rigid-Subst and U-Flex-Rigid-Depend. Follow from IH.

Case U-Flex-Rigid-SkipFlex, U-Flex-Rigid-SkipRigid, U-Flex-Rigid-SkipTerm, U-Flex-Rigid-SkipLock, and U-Flex-Rigid-SkipMark. Follow from IH.

□

Lemma B.7 (Completeness of unification).

1. *If $\theta ⨾ \Theta_0 \sqsubseteq \Theta$ is a solution to the unification question $(\Theta_0; A \equiv B)$, then there exists Θ_1 such that $\Theta_0 \vdash A \equiv B \dashv \Theta_1$.*
2. *If $\theta ⨾ \Theta_0, \Xi \sqsubseteq \Theta$ is a solution to the unification question $(\Theta_0, \Xi; \hat{\alpha} \equiv A)$, then there exists Θ_1 such that $\Theta_0 \mid \Xi \vdash \hat{\alpha} \equiv A \dashv \Theta_1$.*

Proof. We prove 1 and 2 simultaneously. By a straightforward induction on the declarative rules for unification, we can show that if θ is a solution for $(\Theta_0; A \equiv B)$, then it must also solve the questions of all premises for $\Theta_0 \vdash A \equiv B \dashv \_$ in the algorithmic rules. Then by IHs and Lemma B.5 we can show that there exists Θ_1 such that $\Theta_0 \vdash A \equiv B \dashv \Theta_1$ holds. The same applies to $(\Theta_0, \Xi; \hat{\alpha} \equiv A)$. Base cases $\alpha \equiv \alpha$ and $\hat{\alpha} \equiv \hat{\alpha}$ hold trivially. The only case where the algorithmic rules require extra conditions to succeed is U-Flex-Rigid-Solve where in the premise the kinding of A cannot depend on the flexible variable $\hat{\alpha}$. For $\hat{\alpha} \equiv A$ where A is not a flexible variable and $\hat{\alpha}$ is not assigned to a type in Θ_0, we can show that $\Theta \vdash \theta\hat{\alpha} \equiv \theta A$ does not hold for any solutions if A contains $\hat{\alpha}$ by induction on the declarative rules of type equivalence. □

C.3 Type Inference

Lemma C.3 (Soundness and generality of transformation). *For the question* $(\Theta_0, (\mu, \sigma) \Rightarrow \nu \mathbin{@} \bigcirc)$*, if* $\Theta_0 \vdash (\mu, \sigma) \Rightarrow \nu \mathbin{@} E \dashv \Theta_1$*, then* $(\Theta_0 \sqsubseteq \Theta_1, E)$ *is a minimal solution.*

Proof. Follow from Lemma B.4 and Lemma B.8. □

Lemma C.4 (Soundness and generality of instantiation). *For the question* $(\Theta_0, \sigma \preceq_R \bigcirc)$*, if* $\Theta_0 \vdash \sigma \preceq_R A \dashv \Theta_1$*, then* $(\Theta_0 \sqsubseteq \Theta_1, A)$ *is a minimal solution.*

Proof. By definition, $\Theta_1 = \Theta_0, \Xi$ where Ξ contains exactly all flexible type variables introduced by this instantiation. It is obvious that all other solutions can factor through $\Theta_0 \sqsubseteq \Theta_0, \Xi$ by substituting flexible variables in Ξ with proper types. □

Lemma C.5 (Polymorphic weakening). *If* $\Gamma, x :_\mu \sigma, \Gamma' \vdash_s M : A \mathbin{@} E$ *and* $\sigma \preceq_{\text{gen}} \sigma'$*, then* $\Gamma, x :_\mu \sigma', \Gamma' \vdash_s M : A \mathbin{@} E$.

Lemma B.10 (No negative effects). *For the type inference question* $(\Theta_0; M : \bigcirc \mathbin{@} \bigcirc)$ *with the implicit condition* $\vdash \Theta_0$ **pos** *and* $\vdash M$ **pos***, if* $\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1$*, then* $\vdash \Theta_1$ **pos** *and* $\vdash A$ **pos**.

Proof. By straightforward induction on the typing judgement of type inference. □

Theorem B.11 (Soundness and generality of type inference). *For the type inference question* $(\Theta_0; M : \bigcirc \mathbin{@} \bigcirc)$*, if* $\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1$*, then* $(\Theta_0 \sqsubseteq \Theta_1, A, E)$ *is a minimal solution.*

Proof. We want to show that if $\vdash \Theta_0$ **ng**, $\Theta_0 \vdash M$ **ok** and $\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ and $\Theta_1 \vdash M : A \mathbin{@} E$. Moreover, for any $\theta' \mathbin{⨟} \Theta_0 \sqsubseteq \Theta'$ with $\Theta' \vdash M : A' \mathbin{@} E'$, there exists $\zeta \mathbin{⨟} \Theta_1 \sqsubseteq \Theta'$ such that $\theta' \equiv \zeta \mathbin{⨟} \Theta_0 \sqsubseteq \Theta'$, $\Theta' \vdash \zeta A \preceq_{\text{gen}} A'$, and $E \leqslant E'$.

By induction on the derivation of $\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1$. Soundness follows from routine usages of IHs straightforwardly. The only non-trivial cases is for T-Letmod and T-Handler where we probably need to show the principal condition for some terms. They follow from the generality of corresponding sub-judgements, and generality follows from IHs on these sub-judgements.

We focus on proving generality.

Case

I-Freeze

$$\dfrac{\xi = \mathsf{alocks}(\Theta_0) \qquad \forall\Delta.A = \mathsf{subst}(\Theta;\sigma) \qquad \Theta, \Theta_0 \vdash (\mu, \forall\Delta.A) \Rightarrow \xi \mathbin{@} E \dashv \Theta_1\ (1) \qquad \Theta_1 \vdash \forall\Delta.A \preceq_{\mathrm{m}} B \dashv \Theta_2\ (2)}{\Theta, x :_\mu \sigma, \Theta_0 \vdash \lceil x \rceil : B \mathbin{@} E \dashv \Theta_2}$$

For any other solution $(\theta' \mathbin{⨟} \Theta, x :_\mu \sigma, \Theta_0 \sqsubseteq \Theta'_0, x :_\mu \theta'\sigma, \Theta'_1; B'; E')$ where $\Theta' = \Theta'_0, x :_\mu \theta'\sigma, \Theta'_1$, by $\vdash \Theta, x :_\mu \sigma, \Theta_0$ **ng** and Lemma B.1, we have μ and ξ unchanged after metasubstitution of θ'. Moreover, $\mathsf{subst}(\Theta'_0; \theta'\sigma)$ is pure only if $\mathsf{subst}(\Theta;\sigma)$ is pure since substitution preserves purity. Thus, θ' must solve the question of (1). By Lemma C.3 on (1), we have $E \leqslant E'$ (3) and θ' factors through $\Theta, \Theta_0 \sqsubseteq \Theta_1$ (the metasubstitution of (1)) with cofactor $\zeta_1 \mathbin{⨟} \Theta_1 \sqsubseteq \Theta'$. Then (ζ_1, B') must solve (2). By Lemma C.4 on (2), ζ_1 factors through $\Theta_1 \sqsubseteq \Theta_2$ (the metasubstitution of (2)) with cofactor $\zeta_2 \mathbin{⨟} \Theta_2 \sqsubseteq \Theta'$ such that $\Theta' \vdash \zeta_2 B \equiv B'$ (4), Thus, θ' factors through $\Theta, x :_\mu \sigma, \Theta_0 \sqsubseteq \Theta_2$ with cofactor $\zeta_2 \mathbin{⨟} \Theta_2 \sqsubseteq \Theta'$. Our goal follows from cofactor ζ_2, (3), and (4).

Case

$$\textsc{I-Var}\quad \dfrac{\begin{array}{c}\xi = \mathsf{alocks}(\Theta_0) \qquad \forall\Delta.A = \mathsf{subst}(\Theta;\sigma) \qquad (\nu, A') = \mathsf{split}(\Delta, A) \\ \Theta, \Theta_0 \vdash (\mu \circ \nu, \forall\Delta.A') \Rightarrow \xi \mathbin{@} E \dashv \Theta_1\ (1) \qquad \Theta_1 \vdash \forall\Delta.A \preceq_{\mathrm{m}} B \dashv \Theta_2\ (2)\end{array}}{\Theta, x :_\mu \sigma, \Theta_0 \vdash x : B \mathbin{@} E \dashv \Theta_2}$$

For any other solution $(\theta' \fatsemi \Theta, x :_\mu \sigma, \Theta_0 \sqsubseteq \Theta'; B'; E')$, by $\vdash \Theta, x :_\mu \sigma, \Theta_0$ **ng**, Lemma B.1, and the definition of $\mathsf{split}(-)$, we have μ, ν, and ξ unchanged after metasubstitution of θ'. The remaining part is almost the same as the proof for I-Freeze.

Case

$$\textsc{I-LetMod}\quad \dfrac{\begin{array}{c}\Theta_0, 🔒_\nu \vdash M : A \mathbin{@} E \dashv \Theta_1, 🔒_\nu, \Xi_1\ (1) \\ \Theta_1 ⨾ \Xi_1, \hat{\alpha} : (\mathsf{Any}, \mathsf{m}) \vdash A \equiv \phi\hat{\alpha} \dashv \Theta_2 ⨾ \Xi_2\ (2) \qquad \Theta_2 \vdash (M; \nu; \phi; \Xi_2; \hat{\alpha}) \Updownarrow (\xi, \sigma) \dashv \Theta_3\ (3) \\ \Theta_3, x :_\xi \sigma \vdash N : B \mathbin{@} F \dashv \Theta_4\ (4) \qquad F' = \mathsf{solve}(\nu : E \longrightarrow F)\ (5)\end{array}}{\Theta_0 \vdash \mathbf{let}_\nu\ \phi\ x = M\ \mathbf{in}\ N : B \mathbin{@} F' \dashv \Theta_3}$$

For any other solution $(\theta' \fatsemi \Theta_0 \sqsubseteq \Theta'; B'; F_1')$, we have $\Theta' \vdash \mathbf{let}_\nu\ \phi\ x = M\ \mathbf{in}\ N : B' \mathbin{@} F_1'$. Inversion gives

$$\begin{array}{l}\Theta' \vdash (M; \nu; \Delta; \phi; A') \Updownarrow (\xi', \sigma') \\ \Theta', 🔒_\nu, \Delta \vdash_s M : \phi A' \mathbin{@} E' \\ \Theta', x :_{\xi'} \sigma' \vdash_s M : B' \mathbin{@} F_1' \\ \nu_{F_1'} : E' \longrightarrow F_1'\end{array}$$

By definition, we have $\xi' = \xi$. Since ν does not contain flexible variables, by

$$\Theta', 🔒_\nu, \Delta \vdash_s M : \phi A' \mathbin{@} E'$$

we have $(\theta' \fatsemi \Theta_0, 🔒_\nu \sqsubseteq \Theta', 🔒_\nu, \Delta; A'; E')$ solves the question of (1). By IH on (1), we have θ' factors through the metasubstitution of (1) with cofactor $\zeta_1 \fatsemi \Theta_1, 🔒_\nu, \Xi_1 \sqsubseteq \Theta', 🔒_\nu, \Delta$ such that $E \leqslant E'$ and $\Theta' \vdash \zeta_1 A \equiv \phi A'$.

Then $\zeta_1' = \zeta_1, A'/\hat{\alpha}$ must solve the statement of (2). By Lemma B.6 on (2), we have ζ_1' factors through the metasubstitution of (2) with cofactor $\zeta_2 \fatsemi \Theta_2 ⨾ \Xi_2 \sqsubseteq \Theta' ⨾ \Delta$. Case analysis on whether value restriction is satisfied.

Case $M \in \mathsf{Val}$. We have $\sigma = \mathsf{gen}(\Xi_2; \hat{\alpha})$ and $\sigma' = \forall\Delta.A'$. By $\zeta_2 \fatsemi \Theta_2 ⨾ \Xi_2 \sqsubseteq \Theta' ⨾ \Delta$ and $\zeta_2\hat{\alpha} \equiv A'$, we have $\Theta' \vdash \sigma \preceq_{\mathrm{gen}} \sigma'$. Then by Lemma C.5 on $\Theta', x :_{\xi'} \sigma' \vdash_s M : B' \mathbin{@} F'$ and $\xi = \xi'$, we have

$$\Theta', x :_\xi \sigma \vdash_s M : B' \mathbin{@} F_1'$$

Thus, $(\zeta_2 \fatsemi \Theta_3, x :_\xi \sigma \sqsubseteq \Theta', x :_\xi \sigma; B'; F_1')$ solves the question of (4). Observe that by (1) and (2), σ cannot contain flexible modal or effect variables; otherwise it would violate $\vdash \Theta_0$ **ng** since the only way for the type of M to rely on flexible modal or effect variables in Θ_0 is via usage of term variables in Θ_0. Thus we have $\vdash \Theta_3, x :_\xi \sigma$ **ng**. Then by IH on (4), we have ζ_2 factors through the metasubstitution of (4) with cofactor ζ_3 such that $F \leqslant F_1'$ and $\Theta \vdash \zeta_3 B \equiv B'$ (6). By Lemma B.8 on (5) and $\nu_{F_1'} : E' \longrightarrow F_1'$, we have $F' \leqslant F_1'$ (7). Our goal follows from cofactor ζ_3, (6), and (7).

Case $M \notin \mathsf{Val}$. We have

$$\begin{array}{l}\Theta_2 \vdash \mathsf{gen}(\Xi_2; \hat{\alpha}) \preceq_{\mathrm{i}} \sigma \dashv \Theta_3 \\ \Theta' \vdash \forall\Delta.A' \preceq_{\mathrm{i}} \sigma'\end{array}$$

Same as the above sub-case, we have $\Theta' \vdash \mathsf{gen}(\Xi_2; \hat{\alpha}) \preceq_{\mathrm{gen}} \forall\Delta.A'$. By definition of algorithmic $\preceq_{\mathrm{i}}$, we have $\Theta_3 = \Theta_2, \Xi_3$ where Ξ_3 contains the flexible intuitionistic variables appearing in σ. Thus, by $\zeta_2 \mathbin{⨟} \Theta_2 \sqsubseteq \Theta'$, there exists a metasubstitution $\zeta_2' \mathbin{⨟} \Theta_2, \Xi_3 \sqsubseteq \Theta_2$ which substitutes flexible variables in Ξ_3 such that $\Theta' \vdash \zeta_2\zeta_2'\sigma \equiv \sigma'$. Then we have $\zeta_2\zeta_2' \mathbin{⨟} \Theta_3, x :_\xi \sigma \sqsubseteq \Theta', x :_\xi \sigma'$, which gives that $(\zeta_2\zeta_2'; B'; F_1')$ solves (4). Similar to the above sub-case, σ does not contain flexible modal or effect variables since we use $\preceq_\mathsf{i}$ and have $\vdash \Theta_0$ **ng**. Then by IH on (4), we have $\zeta_2\zeta_2'$ factors through the metasubstitution of (4) with cofactor ζ_3 such that $F \leqslant F_1'$ and $\Theta \vdash \zeta_3 B \equiv B'$ (6). By Lemma B.8 on (5) and $\nu_{F_1'} : E' \to F_1'$, we have $F' \leqslant F_1'$ (7). Our goal follows from cofactor ζ_3, (6), and (7).

Case

$$\frac{\Theta_0, \hat{\alpha} : (\mathsf{Any}, \mathsf{i}), x : \hat{\alpha} \vdash M : B \mathbin{@} E \dashv \Theta_1, x : \hat{\alpha}, \Xi_1 \ (1)}{\Theta_0 \vdash \lambda x.M : \hat{\alpha} \to B \mathbin{@} E \dashv \Theta_1, \Xi_1}\ \textsc{I-Abs}$$

For any other solution $(\theta' \mathbin{⨟} \Theta_0 \sqsubseteq \Theta'; A' \to B'; E')$, we have $\Theta' \vdash_s \lambda x.M : A' \to B' \mathbin{@} E'$. Inversion gives

$$\Theta', x : A' \vdash_s M : B' \mathbin{@} E'$$

Letting $\theta_1 = \theta', A'/\hat{\alpha}$, we have that (θ_1, B', E') solves the question of (1). By $\vdash \Theta_0$ **ng** we have $\vdash \Theta_0, \hat{\alpha} : (\mathsf{Any}, \mathsf{i}), x : \hat{\alpha}$ **ng**. Then by IH on (1), we have θ_1 factors through the metasubstitution of (1) with cofactor $\zeta \mathbin{⨟} \Theta_1, x : \hat{\alpha}, \Xi_1 \sqsubseteq \Theta', x : \hat{\alpha}, x : A'$ such that $E \leqslant E'$ (2) and $\Theta' \vdash \zeta B \equiv B'$ (3). Observe that $\theta' \equiv \theta_1 \equiv \zeta \mathbin{⨟} \Theta_0 \sqsubseteq \Theta'$. Our goal follows from cofactor ζ, (2), and (3).

Case

$$\frac{\Theta_0, x : A \vdash M : B \mathbin{@} E \dashv \Theta_1, x : A, \Xi_1 \ (1)}{\Theta_0 \vdash \lambda x^A.M : A \to B \mathbin{@} E \dashv \Theta_1, \Xi_1}\ \textsc{I-AbsAnno}$$

Our goal follows from IH on (1).

Case

$$\frac{\begin{array}{c}\Theta_0 \vdash M : A \mathbin{@} E \dashv \Theta_1 \ (1) \\ \Theta_1 \vdash N : B \mathbin{@} F \dashv \Theta_2 \ (2) \qquad \Theta_2, \hat{\alpha} : (\mathsf{Any}, \mathsf{m}) \vdash A \equiv B \to \hat{\alpha} \dashv \Theta_3 \ (3)\end{array}}{\Theta_0 \vdash M\ N : \hat{\alpha} \mathbin{@} E \cup F \dashv \Theta_3}\ \textsc{I-App}$$

For any other solution $(\theta' \mathbin{⨟} \Theta_0 \sqsubseteq \Theta'; A_1; E_1)$, we have $\Theta' \vdash M\ N : A_1 \mathbin{@} E_1$. Inversion gives

$$\Theta' \vdash_s M : A' \to A_1 \mathbin{@} E_1$$
$$\Theta' \vdash_s N : B' \mathbin{@} E_1$$

Then $(\theta'; A' \to A_1; E_1)$ must solve the question of (1). By IH on (1), we have θ' factors through the metasubstitution of (1) with cofactor $\zeta_1 \mathbin{⨟} \Theta_1 \sqsubseteq \Theta'$ such that $E \leqslant E_1$ and $\Theta' \vdash \zeta_1 A \equiv A' \to A_1$.
Then $(\zeta_1; B'; E_1)$ must solve the question (2). By IH on (2), we have ζ_1 factors through the metasubstitution of (2) with cofactor $\zeta_2 \mathbin{⨟} \Theta_2 \sqsubseteq \Theta'$ such that $F \leqslant E_1$ and $\Theta' \vdash \zeta_2 B \equiv B'$.
Letting $\zeta_2' = \zeta_2, A_1/\hat{\alpha}$, we have ζ_2' solves the statement of (3). By Lemma B.6 on (3), ζ_2' factors through the metasubstitution of (3) with cofactor $\zeta_3 \mathbin{⨟} \Theta_3 \sqsubseteq \Theta'$. By $\zeta_2 \equiv \zeta_3 \mathbin{⨟} \Theta_2, \hat{\alpha} : (\mathsf{Abs}, \mathsf{m}) \sqsubseteq \Theta'$, we have $\Theta' \vdash \zeta_3\hat{\alpha} \equiv A_1$ (4). By $E \leqslant E_1$ and $F \leqslant F_1$ we have $E \cup F \leqslant E_1$ (5). Our goal follows from cofactor ζ_3, (4), and (5).

Case

$$\textsc{I-LetAnno}\quad \frac{\Theta_0 \vdash (M;\Delta;A) \Updownarrow^{\dagger} \sigma \dashv \Theta_1 \;(1) \qquad \Theta_1 ; \Delta \vdash M : A' \mathbin{@} E \dashv \Theta_2 ; \Delta, \Xi_2 \;(2) \qquad \Theta_2 ; \Delta, \Xi_2 \vdash A' \equiv A \dashv \Theta_3 ; \Delta, \Xi_3 \;(3) \qquad \Theta_3, x : \sigma \vdash N : B \mathbin{@} F \dashv \Theta_4, x : \sigma, \Xi_4 \;(4)}{\Theta_0 \vdash \mathbf{let}\; x^{\forall \Delta.A} = M \;\mathbf{in}\; N : B \mathbin{@} E \cup F \dashv \Theta_4, \Xi_4}$$

By definition of $\Updownarrow^{\dagger}$ and (1), $\Theta_0 = \Theta_1$. For any other solution $(\theta' \fatsemi \Theta_0 \sqsubseteq \Theta', B', E_1)$, we have

$$\Theta' \vdash_s \mathbf{let}\; x^{\forall \Delta.A} = M \;\mathbf{in}\; N : B' \mathbin{@} E_1.$$

Inversion gives

$$\begin{array}{l} \Theta' \vdash (M, \Delta, A) \Updownarrow^{\dagger} \sigma \\ \Theta', \Delta \vdash_s M : A \mathbin{@} E_1 \\ \Theta', x : \sigma \vdash_s N : B' \mathbin{@} E_1 \end{array}$$

We have $\theta' \fatsemi \Theta_0 ; \Delta \sqsubseteq \Theta' ; \Delta$ and $\Theta' ; \Delta \vdash_s M : A \mathbin{@} E_1$, which gives that (θ', A, E_1) solves the question of (2). By IH on (2), we have θ' factors through the metasubstitution of (2) with cofactor $\zeta_1 \fatsemi \Theta_2 ; \Delta, \Xi_2 \sqsubseteq \Theta' ; \Delta$ such that $\Theta' ; \Delta \vdash \zeta_1 A' \equiv A$ and $E \leqslant E_1$.
Then ζ_1 solves the statement of (3). By Lemma B.6 on (3), ζ_1 factors through the metasubstitution of (3) with cofactor $\zeta_2 \fatsemi \Theta_3 ; \Delta, \Xi_3 \sqsubseteq \Theta' ; \Delta$.
Since σ does not contain any flexible variable, we have $\zeta_2 \fatsemi \Theta_3, x : \sigma \sqsubseteq \Theta', x : \sigma$. Thus, we have (ζ_2, B', E_1) solves the question of (4). By IH on (4), we have ζ_2 factors through the metasubstitution of (4) with cofactor $\zeta_3 \fatsemi \Theta_4, x : \sigma, \Xi_4 \sqsubseteq \Theta', x : \sigma$ such that $\Theta' \vdash \zeta_3 B \equiv B'$ (5) and $F \leqslant E_1$. By $E \leqslant E_1$ and $F \leqslant E_1$ we have $E \cup F \leqslant E_1$ (6). Our goal follows from cofactor ζ_3, (5), and (6).

Case

$$\textsc{I-Do}\quad \frac{\Sigma \ni \ell : A \twoheadrightarrow B \qquad \Theta_0 \vdash M : A_1 \mathbin{@} E \dashv \Theta_1 \;(1) \qquad \Theta_1 \vdash A_1 \equiv A \dashv \Theta_2 \;(2)}{\Theta_0 \vdash \mathbf{do}\; \ell\; M : B \mathbin{@} \{\ell\} \cup E \dashv \Theta_2}$$

Our goal follows from IH on (1) and Lemma B.6 on (2).

Case

$$\textsc{I-Mask}\quad \frac{\Theta_0, \text{🔒}_{\langle L \rangle} \vdash M : A \mathbin{@} E \dashv \Theta_1 \;(1) \qquad F = \mathsf{solve}(\langle L \rangle : E \longrightarrow \cdot) \;(2)}{\Theta_0 \vdash \mathbf{mask}_L\; M : \langle L \rangle A \mathbin{@} F \dashv \Theta_2}$$

Our goal follows from IH on (1) and Lemma B.8 on (2).

Case

$$\textsc{I-Handler}\quad \frac{\begin{array}{c} D = \{\ell_i\}_i \qquad \{\ell_i : A_i \twoheadrightarrow B_i\} \subseteq \Sigma \\ \Theta, \text{🔒}_{\langle D \rangle} \vdash M : A_0 \mathbin{@} E' \dashv \Theta_{-1}, \text{🔒}_{\langle D \rangle}, \Xi' \;(1) \qquad \Theta_{-1} \vdash (M; \Xi'; A_0) \Downarrow A \dashv \Theta_0 \;(2) \\ \Theta_0, x : \langle D \rangle A \vdash N : B_0 \mathbin{@} E_r \dashv \Theta_0', x : \_, \Xi_0' \;(3) \qquad \Theta_0' \vdash (N; \Xi_0'; B_0) \Downarrow B \dashv \Theta_1 \;(4) \\ [\Theta_i, p_i : A_i, r_i : B_i \longrightarrow B \vdash N_i : B_i \mathbin{@} E_i \dashv \Theta_i', p_i : \_, r_i : \_, \Xi_i' \;(5) \\ \Theta_i', \Xi_i' \vdash B_i \equiv B \dashv \Theta_{i+1} \;(6)]_{i=1}^{n} \\ E = \mathsf{solve}(\langle D \rangle : E' \longrightarrow \cdot) \qquad F = E \cup E_r \cup (\cup_i E_i) \end{array}}{\Theta \vdash \mathbf{handle}\; M \;\mathbf{with}\; \{\mathbf{return}\; x \mapsto N\} \uplus \{\ell_i\; p_i\; r_i \mapsto N_i\}_{i=1}^{n} : B \mathbin{@} F \dashv \Theta_{n+1}}$$

Though this rule looks scary, there is nothing special we need for proving it compared to the cases we have shown. For any other solution $(\theta' \fatsemi \Theta \sqsubseteq \Theta'; B'; F')$, we have

$$\Theta' \vdash_s \mathbf{handle}\; M \;\mathbf{with}\; H : B' \mathbin{@} F'.$$

Inversion gives

$$\begin{array}{l} \Theta' \vdash_s (M; \Delta; A'_0) \Downarrow A' \\ \Theta' \vdash_s (N; \Delta'; B'_0) \Downarrow B' \\ \Theta', \blacksquare_{\langle\!| D \rangle}, \Delta \vdash_s M : \langle\!| D \rangle A'_0 \ @\ D + F' \\ \Theta', x : \langle\!| D \rangle A', \Delta' \vdash_s N : B'_0 \ @\ F' \\ [\Theta', p_i : A_i, r_i : B_i \to B' \vdash_s N_i : B' \ @\ F']_i \end{array}$$

Our goal follow from IHs on (1), (3), (5), and Lemma B.6 on (6). To use IH on (3), we need to show that $\vdash \Theta_0, x : \langle\!| D \rangle A$ **ng** is satisfied and $\langle\!| D \rangle A$ can be transformed to $\langle\!| D \rangle A'$ via a proper metasubstitution. We can show both using (2) similarly to the proof for T-LetMod when value restriction is not satisfied. Similarly, to use IHs on (4), we can again show that $\vdash \Theta_i, p_i : A_i, r_i : B_i \to B$ **ng** is satisfied and $B_i \to B$ can be transformed to $B_i \to B'$ via a proper metasubstitution using (5).

□

Theorem B.12 (Completeness of Type Inference). *If* $\vdash \Theta_0$ **ng**, $\Theta_0 \vdash M$ **ok**, $\theta \,\mathbin{⨾}\, \Theta_0 \sqsubseteq \Theta$, *and* $\Theta \vdash_s M : A \ @\ F$, *then* $\Theta_0 \vdash M : B \ @\ E \dashv \Theta_1$ *for some* Θ_1, B, *and* E.

Proof. By induction on the typing derivation $\Theta \vdash_s M : A \ @\ F$.

Case

$$\frac{\begin{array}{cc} \xi = \mathsf{alocks}(\Theta') & \forall \Delta.A = \mathsf{subst}(\Theta; \sigma) \\ \Theta, \Theta' \vdash (\mu, \forall \Delta.A) \Rightarrow \xi \ @\ E\ (1) & \Theta, \Theta' \vdash \forall \Delta.A \leq_{\mathsf{m}} B \end{array}}{\Theta, x :_\mu \sigma, \Theta' \vdash_s \lceil x \rceil : B \ @\ E} \ \textsc{T-Freeze}$$

Suppose $\Theta_0 = \Theta_{-1}, x :_{\mu'} \sigma', \Theta'_0$. By Lemma B.1 and $\vdash \Theta_0$ **ng**, we have $\vdash \Theta, x :_\mu \sigma, \Theta'$ **ng**, $\mu' = \mu$, and $\mathsf{alocks}(\Theta'_0) = \xi$. Case analysis on (1).

Case There exists F such that $\mu_F \Rightarrow \xi_F$. Then $\Theta_{-1}, \Theta'_0 \vdash (\mu, \sigma') \Rightarrow \xi \ @\ E' \dashv \Theta_1$ also succeeds. Our goal follows from I-Freeze.

Case Otherwise. We have $\Theta, \Theta' \vdash \forall \Delta.A : \mathsf{Abs}$ and $\mathsf{solve}(\mu \Rightarrow \xi)$ fails. Let $\forall \Delta.A' = \mathsf{subst}(\Theta_{-1}; \sigma')$. By $\Theta, \Theta' \vdash \forall \Delta.A : \mathsf{Abs}$ and $\Theta, \Theta' \vdash \sigma = \theta\sigma'$, we have $\Theta, \Theta' \vdash \forall \Delta.A = \theta(\forall \Delta.A')$. Then by $\theta \,\mathbin{⨾}\, \Theta_{-1}, \Theta_0 \sqsubseteq \Theta, \Theta'$, we have $\Theta_{-1}, \Theta'_0 \vdash \forall \Delta.A' : (\mathsf{Abs}, \mathsf{m}) \dashv \Theta_1$ succeeds for some Θ_1. Our goal follows from I-Freeze.

Case

$$\frac{\begin{array}{ccc} & \xi = \mathsf{alocks}(\Theta') & \forall \Delta.A = \mathsf{subst}(\Theta; \sigma) \\ (\nu, A_1) = \mathsf{split}(\Delta, A) & \Theta, \Theta' \vdash (\mu \circ \nu, \forall \Delta.A_1) \Rightarrow \xi \ @\ E & \Theta, \Theta' \vdash \forall \Delta.A_1 \leq_{\mathsf{m}} B \end{array}}{\Theta, x :_\mu \sigma, \Theta' \vdash_s x : B \ @\ E} \ \textsc{T-Var}$$

Suppose $\Theta_0 = \Theta_{-1}, x :_{\mu'} \sigma', \Theta'_0$ and $\forall \Delta.A' = \mathsf{subst}(\Theta_{-1}; \sigma')$. Let $(\nu', A'_1) = \mathsf{split}(\Delta, A')$. By Lemma B.1 and $\vdash \Theta_0$ **ng**, we have $\vdash \Theta, x :_\mu \sigma, \Theta'$ **ng**. Thus, σ and σ' do not contain flexible modal and effect variables, which implies that $\nu = \nu'$. The remaining part is similar to the case of T-Freeze.

Case

$$\frac{\Theta, \blacksquare_\mu \vdash_s V : A \ @\ E\ (1) \qquad \mu_F : E \to F\ (2)}{\Theta \vdash_s \mathbf{mod}_\mu\, V : \mu A \ @\ F} \ \textsc{T-Mod}$$

IH on (1) gives

$$\Theta_0, \blacksquare_\mu \vdash V : A' \ @\ E' \dashv \Theta_1, \blacksquare_\mu, \Xi_1$$

for some A', E', Θ_1 and Ξ_1. By (2) and Lemma B.9, we have $F' = \mathrm{solve}(\mu : E' \to \cdot)$. Our goal follows from I-Mod.

Case

$$\textsc{T-AbsAnno}\ \frac{\Theta, x : A \vdash_s M : B \mathbin{@} E \; (1)}{\Theta \vdash_s \lambda x^A.M : A \to B \mathbin{@} E}$$

Our goal follows from IH on (1) and I-AbsAnno.

Case

$$\textsc{T-Abs}\ \frac{\Theta, x : S \vdash_s M : B \mathbin{@} E \; (1)}{\Theta \vdash_s \lambda x.M : S \to B \mathbin{@} E}$$

Let $\theta' = \theta, S/\hat{\alpha}$. We have $\theta' \mathbin{⨾} \Theta_0, \hat{\alpha} : (\mathsf{Any}, \mathsf{i}), x : \hat{\alpha} \sqsubseteq \Theta, x : S$. Our goal follows from IH on (1) and θ', and I-Abs.

Case

$$\textsc{T-App}\ \frac{\Theta \vdash_s M : A \to B \mathbin{@} E \; (1) \qquad \Theta \vdash_s N : A \mathbin{@} E \; (2)}{\Theta \vdash_s M\ N : B \mathbin{@} E}$$

IH on (1) gives

$$\Theta_0 \vdash M : A' \mathbin{@} E' \dashv \Theta_1 \; (3)$$

for some A', E', and Θ_1. By Theorem B.11 we have θ factors through the metasubstitution of (3) with cofactor $\zeta_1 \mathbin{⨾} \Theta_1 \sqsubseteq \Theta$ such that $\Theta \vdash \zeta_1 A' \equiv A \to B$. Then IH on (2) gives

$$\Theta_1 \vdash N : B' \mathbin{@} F' \dashv \Theta_2 \; (4)$$

for some B', F', and Θ_2. Again by Theorem B.11 we have ζ_1 factors through the metasubstitution of (4) with cofactor $\zeta_2 \mathbin{⨾} \Theta_2 \sqsubseteq \Theta$ such that $\Theta \vdash \zeta_2 B' \equiv A$. Then by Lemma B.6, $\zeta_3 = \zeta_2, B/\hat{\alpha}$ factors through

$$\Theta_2, \hat{\alpha} : (\mathsf{Any}, \mathsf{m}) \vdash A' \equiv B' \to \hat{\alpha} \dashv \Theta_3 \; (5)$$

with some cofactor. Our goal follows from I-App, (3), (4), and (5).

Case

$$\textsc{T-Letmod}\ \frac{\Theta \vdash (M; \nu; \Delta; \phi; A) \Updownarrow (\xi, \sigma) \qquad \Theta, \text{🔒}_\nu, \Delta \vdash_s M : \phi A \mathbin{@} E \; (1) \qquad \nu_F : E \to F \qquad \Theta, x :_\xi \sigma \vdash_s N : B \mathbin{@} F \; (2)}{\Theta \vdash_s \mathbf{let}_\nu\ \phi\ x = M\ \mathbf{in}\ N : B \mathbin{@} F}$$

IH on (1) gives

$$\Theta_0, \text{🔒}_\nu, \Delta \vdash M : A_1 \mathbin{@} E' \dashv \Theta_1, \text{🔒}_\nu, \Delta, \Xi_1 \; (3)$$

for some A_1, E', Θ_1, and Ξ_1. By Theorem B.11, θ factors through the metasubstitution of (3) with cofactor $\zeta_1 \mathbin{⨾} \Theta_1, \text{🔒}_\nu, \Delta, \Xi_1 \sqsubseteq \Theta, \text{🔒}_\nu, \Delta$ such that $\Theta, \Delta \vdash \zeta_1 A_1 \equiv \phi A$. Observe that since M does not mention Δ in type annotations, A_1 cannot contain any rigid variables in Δ, which gives

$$\Theta_0, \text{🔒}_\nu \vdash M : A_1 \mathbin{@} E' \dashv \Theta_1, \text{🔒}_\nu, \Xi_1 \; (4)$$
$$\zeta_1 \mathbin{⨾} \Theta_1, \text{🔒}_\nu, \Xi_1 \sqsubseteq \Theta, \text{🔒}_\nu, \Delta$$

Letting $\zeta_1' = \zeta_1, A/\hat{\alpha}$, by Lemma B.7, we have

$$\Theta_1 \mathbin{⨾} \Xi_1, \hat{\alpha} : (\mathsf{Any}, \mathsf{m}) \vdash A_1 \equiv \phi\hat{\alpha} \dashv \Theta_2 \mathbin{⨾} \Xi_2 \; (5)$$

By Lemma B.6, ζ_1 factors through the metasubstitution of (5) with cofactor $\zeta_2 \between \Theta_2, \text{🔒}_\nu, \Xi_2 \sqsubseteq \Theta, \text{🔒}_\nu, \Delta$. Case analysis on whether value restriction is satisfied.

Case $M \in \mathsf{Val}$. We have $\sigma = \forall\Delta.A$, $\sigma' = \mathsf{gen}(\Xi_2; \hat{\alpha})$, and

$$\Theta_2 \vdash (M; \nu; \phi; \Xi_2; \hat{\alpha}) \Updownarrow (\xi, \sigma') \dashv \Theta_2 \; (6).$$

By $\zeta_2 \between \Theta_2, \text{🔒}_\nu, \Xi_2 \sqsubseteq \Theta, \text{🔒}_\nu, \Delta$, we have $\Theta \vdash \zeta_2\sigma' \leq_{\mathsf{gen}} \sigma$. Then by Lemma C.5 on (2) we have

$$\Theta, x :_\xi \zeta_2\sigma' \vdash_s N : B \mathbin{@} F \; (7).$$

By $\mathsf{principal}(\Theta, \text{🔒}_\nu; M; \Delta; \phi A)$ and $\vdash \Theta \; \mathbf{ng}$, σ does not contain flexible modal or effect variables. Otherwise, σ would not be the principal type since these flexible variables could be further generalised. Thus, $\zeta_2\sigma'$ does neither contain flexible modal or effect variables, which gives $\vdash \Theta, x : \zeta_1\sigma' \; \mathbf{ng}$. Our goal follows from IH on (7), I-Let, (4), (5), (6), and Lemma B.9.

Case $M \notin \mathsf{Val}$. We have

$$\Theta \vdash \forall\Delta.A \leq_{\mathsf{i}} \sigma$$
$$\Theta_2 \vdash \mathsf{gen}(\Xi_2; \hat{\alpha}) \leq_{\mathsf{i}} \sigma' \dashv \Theta_3$$
$$\Theta_2 \vdash (M; \nu; \phi; \Xi_2; \hat{\alpha}) \Updownarrow (\xi, \sigma') \dashv \Theta_3 \; (8)$$

By definition of $\leq_{\mathsf{i}}$, we have $\Theta_3 = \Theta_2, \Xi_3$ where Ξ_3 only contains flexible variables in σ'. By $\zeta_2 \between \Theta_2, \text{🔒}_\nu, \Xi_2 \sqsubseteq \Theta, \text{🔒}_\nu, \Delta$, there exists $\zeta_2' \between \Theta_2, \Xi_3 \sqsubseteq \Theta_2$ such that $\Theta \vdash \zeta_2\zeta_2'\sigma' \equiv \sigma$. Then we have $\zeta_2\zeta_2' \between \Theta_2, \Xi_3, x : \sigma' \sqsubseteq \Theta, x : \sigma$. By $\mathsf{principal}(\Theta, M, \Delta, A)$ and $\vdash \Theta \; \mathbf{ng}$, σ_0 does not contain flexible modal or effect variables, which further gives that σ does not contain flexible modal or effect variables. Thus we have $\vdash \Theta, x : \sigma \; \mathbf{ng}$. Our goal follows from IH on (2), I-Let, (4), (5), and (8).

Case

$$\frac{\Theta \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma \qquad \Theta, \Delta \vdash_s M : A \mathbin{@} E \; (1) \qquad \Theta, x : \sigma \vdash_s N : B \mathbin{@} E \; (2)}{\Theta \vdash_s \mathbf{let}\; x^{\forall\Delta.A} = M \;\mathbf{in}\; N : B \mathbin{@} E}\;\textsc{T-LetAnno}$$

By definition, we have $\Theta_0 \vdash (M; \Delta; A) \Updownarrow^{\dagger} \sigma \dashv \Theta_1$ where $\Theta_0 = \Theta_1$. Our goal follows from IH on (1), Theorem B.11, Lemma B.6, and IH on (2).

Case

$$\frac{\Sigma \ni \ell : A \twoheadrightarrow B \qquad E = \ell, F \qquad \Theta \vdash_s M : A \mathbin{@} E \; (1)}{\Theta \vdash_s \mathbf{do}\; \ell\; M : B \mathbin{@} E}\;\textsc{T-Do}$$

Our goal follows from IH on (1) and Theorem B.11.

Case

$$\frac{\Theta, \text{🔒}_{\langle L |\rangle} \vdash_s M : A \mathbin{@} F - L \; (1)}{\Theta \vdash_s \mathbf{mask}_L\; M : \langle L |\rangle A \mathbin{@} F}\;\textsc{T-Mask}$$

Our goal follows from IH on (1) and Lemma B.9.

Case

$$\textsc{T-Handler}\quad \frac{\begin{array}{cc} D = \{\ell_i\}_i & \{\ell_i : A_i \twoheadrightarrow B_i\} \subseteq \Sigma \\ \Gamma \vdash (M; \Delta; A_0) \Downarrow A\ (1) & \Gamma, 🔒_{\langle\!| D \rangle}, \Delta \vdash_s M : A_0 \ @\ D + F\ (2) \\ \Gamma \vdash (N; \Delta'; B_0) \Downarrow B\ (3) & \Gamma, x : \langle\!| D \rangle A, \Delta' \vdash_s N : B_0 \ @\ F\ (4) \\ \multicolumn{2}{c}{[\Gamma, p_i : A_i, r_i : B_i \to B \vdash_s N_i : B \ @\ F\ (5)]_i} \end{array}}{\Gamma \vdash_s \textbf{handle}\ M\ \textbf{with}\ \{\textbf{return}\ x \mapsto N\} \uplus \{\ell_i\ p_i\ r_i \mapsto N_i\}_i : B \ @\ F}$$

Though this rule looks scary, there is nothing special we need for proving it compared to the cases we have shown. Our goal follows from IHs on (2), (4), and (5), using Theorem B.11 and Lemma B.9. To use IHs on (4) and (5), we need to connect the declarative intuitionistic instantiations of (1) and (3) with their corresponding algorithmic intuitionistic instantiations, as well as main the $\vdash -$ **ng** invariant using the principal condition of (1) and (3), similarly to the proof for T-LetMod when value restriction is not satisfied.

□