A CALCULUS FOR SCOPED EFFECTS & HANDLERS

ROGER BOSMAN [a], BIRTHE VAN DEN BERG [a], WENHAO TANG [b], AND TOM SCHRIJVERS [a]

[a] KU Leuven, Celestijnenlaan 200A
e-mail address: {roger.bosman/birthe.vandenberg/tom.schrijvers}@kuleuven.be

[b] The University of Edinburgh, 10 Crichton Street
e-mail address: wenhao.tang@ed.ac.uk

Abstract. Algebraic effects & handlers have become a standard approach for side-effects in functional programming. Their modular composition with other effects and clean separation of syntax and semantics make them attractive to a wide audience. However, not all effects can be classified as algebraic; some need a more sophisticated handling. In particular, effects that have or create a delimited scope need special care, as their continuation consists of two parts—in and out of the scope—and their modular composition introduces additional complexity. These effects are called *scoped* and have gained attention by their growing applicability and adoption in popular libraries. While calculi have been designed with algebraic effects & handlers built in to facilitate their use, a calculus that supports scoped effects & handlers in a similar manner does not yet exist. This work fills this gap: we present λ_{sc}, a calculus with native support for both algebraic and scoped effects & handlers. It addresses the need for polymorphic handlers and explicit clauses for forwarding unknown scoped operations to other handlers. Our calculus is based on Eff, an existing calculus for algebraic effects, extended with Koka-style row polymorphism, and consists of a formal grammar, operational semantics, a (type-safe) type-and-effect system and type inference. We demonstrate λ_{sc} on a range of examples.

1. Introduction

While monads [Mog89, Mog91, Wad95] have long been the go-to approach for modelling effects, *algebraic effects & handlers* [PP03, PP09] are gaining steadily more traction. They offer a more structured and modular approach to composing effects, based on an algebraic model. The approach consists of two parts: effects denote the syntax of operations, and handlers interpret them by means of structural recursion. By composing handlers that each interpret only a part of the syntax in the desired order, one can modularly build an interpretation for the entire program. Algebraic effects & handlers have been adopted in several libraries such as (e.g., fused-effects [RTWS18], extensible-effects [KSSF19], Eff in OCaml [KS18]) and languages (e.g., Links [HL16], Koka [Lei17], Effekt [BSO20]).

Key words and phrases: algebraic effects, scoped effects, calculus, operational semantics, type- and effect system.

Although the modular approach of algebraic effects & handlers is desirable for every effectful program, it is not always applicable. In particular, those effects that have or introduce a delimited scope (e.g., exceptions, concurrency, local state) are not algebraic. Essentially, these so-called *scoped effects* [WSH14] split the program in two: a scoped computation where the effect is in scope, and a continuation where it is out of scope. This separation breaks algebraicity, which states operations commute with sequencing. Modeling scoped effects as handlers [PP03] has been proposed as a way of encoding scoped effects in an algebraic framework. However, this comes at the cost of modularity [YPW$^+$22]. Instead, a calculus that provides scoped effects & handlers as native features is required. The growing interest in scoped effects & handlers, evidenced by their adoption at GitHub [TRWS22] and in Haskell libraries (e.g., eff [Kin19], polysemy [Mag19], fused-effects [RTWS18]), motivates the need for such a calculus.

This paper aims to fill this gap in the literature: we present λ_{sc}, a calculus that puts scoped effects & handlers on formal footing. Our main source of inspiration is Eff [BP13, BP15, Pre15], a calculus for algebraic effects & handlers, effectively easing programming with those features. Although Eff is an appropriate starting point, the extension to support scoped effects & handlers is non-trivial, for two reasons. First, scoped effects require polymorphic handlers, which we support by adding let-polymorphism and F_ω-style type operators. Second, we need to be able to forward unknown operations in order to keep the desired modularity. Whereas algebraic effects & handlers have a generic (and implicit) forwarding mechanism, scoped effects & handlers need an explicit forwarding clause in order to allow sufficient freedom in their implementation.

In what follows, we formalize λ_{sc}, after introducing the appropriate background (Section 2) and informally motivating the challenges and design choices of our calculus (Section 3). We make the following contributions:

- We design a formal syntax for λ_{sc} terms, types and contexts (Section 4).
- We provide an operational semantics (Section 5).
- We define the type-and-effect system of λ_{sc} (Section 6).
- We formulate and prove λ_{sc}'s metatheoretical properties (Section 6).
- We show the usability of our calculus on a range of examples (Section 7).
- We give a type inference algorithm and show it sound and complete with respect to the declarative type-and-effect system (Appendix F).
- We provide an interpreter of our calculus with type inference in which we implement all our examples (supplementary material).

2. Background & Motivation

This section provides the necessary background and motivates our goal. We review *algebraic effects & handlers* as a modular approach to composing side-effects in effectful programs. Next, we present *scoped effects & handlers*: effects that have or create a delimited scope (such as `once` for nondeterminism [PSWJ18, WSH14]), and motivate the need for a calculus with built-in support for these scoped effects.

2.1. Algebraic effects & handlers.

Algebraic effects & handlers consist of *operations*, denoting their syntax, and *handlers*, denoting their semantics. This separation gives us modular composition, which has intrinsic value *and* allows controlling effect interaction.

2.1.1. *Algebraic Operations.* Effects are denoted by a name (or *label*) and characterized by a *signature* $A \twoheadrightarrow B$, taking a value of type A and producing a value of type B. For example, $\texttt{choose} : () \twoheadrightarrow \mathsf{Bool}$ takes a unit value and produces a boolean (e.g., nondeterministically). *Operations* invoke effects, combining the **op** keyword, an effect to invoke, a *parameter* passed to the effect, and a *continuation*, containing the rest of the program.

$$c_{\mathsf{ND}} = \textbf{op}\ \texttt{choose}\ ()\ (b\,.\,\textbf{if}\ b\ \textbf{then return}\ 1\ \textbf{else return}\ 2)$$

In accordance with its signature, choose is passed (), and in the supplied contination b has type Bool. As a result, c_{ND} is a computation that returns either 1 or 2.

Some operations commute with sequencing. For example:

$$\begin{aligned} &\textbf{do}\ x \leftarrow \textbf{op}\ \texttt{choose}\ ()\ (b\,.\,\textbf{if}\ b\ \textbf{then return}\ 1\ \textbf{else return}\ 2)\,;\textbf{return}\ x^2 \\ \equiv\ &\textbf{op}\ \texttt{choose}\ ()\ (b\,.\,\textbf{do}\ x \leftarrow \textbf{if}\ b\ \textbf{then return}\ 1\ \textbf{else return}\ 2\ \ ;\textbf{return}\ x^2) \end{aligned}$$

This equivalence is an instance of the *algebraicity property*, and operations are *algebraic* if they satisfy this property. Algebraicity states that the sequencing of a computation c_2 after an operation $\textbf{op}\ \ell\ v\ (y\,.\,c_1)$ is equivalent to sequencing the same computation after the *continuation* of this operation:

$$\textbf{do}\ x \leftarrow \textbf{op}\ \ell\ v\ (y\,.\,c_1)\,;c_2\ \equiv\ \textbf{op}\ \ell\ v\ (y\,.\,\textbf{do}\ x \leftarrow c_1\,;c_2)$$

2.1.2. *Handlers. Handlers* give meaning to operations. For example, handler h_{ND} interprets choose nondeterministically:

$$\begin{aligned} h_{\mathsf{ND}} = \textbf{handler}\ \{&\textbf{return}\ x \qquad\quad \mapsto \textbf{return}\ [x] \\ ,&\textbf{op}\ \texttt{choose}\ \_\ k \mapsto \textbf{do}\ xs \leftarrow k\ \mathsf{true}\,;\textbf{do}\ ys \leftarrow k\ \mathsf{false}\,;xs \mathbin{+\!\!+} ys\} \end{aligned}$$

This handler has two clauses. The first clause returns a singleton list in case a value x is returned. The second clause, which interprets choose, executes both branches by applying the continuation k to both true and false, and concatenates their resulting lists with the ($\mathbin{+\!\!+}$)-operator. We apply h_{ND} to c_{ND} with the $\star$-operator to obtain both of its results:

$$h_{\mathsf{ND}} \star c_{\mathsf{ND}} \leadsto^* [1, 2]$$

Algebraic effects & handlers bring several interesting advantages. Most interestingly, their separation of syntax and semantics allows a modular composition of different effects, which in turn allows for altering the meaning of a program by different effect interactions.

2.1.3. *Modular Composition.* Effects can be composed by combining different primitive operations. For example, computation $c_{\mathsf{c,g}}$ below uses $\texttt{get} : () \twoheadrightarrow \mathsf{String}$ in addition to choose.

$$c_{\mathsf{c,g}} = \textbf{op}\ \texttt{choose}\ ()\ (b\,.\,\textbf{if}\ b\ \textbf{then return}\ 1\ \textbf{else op}\ \texttt{get}\ ()\ (x\,.\,\textbf{return}\ x))$$

Instead of having to write a handler for each combinations of effects, algebraic effects & handlers allow us to write a handler specific for the effect get, and to compose it with the existing handler h_{ND}.

$$h_{\mathsf{get}} = \textbf{handler}\ \{\textbf{return}\ x \mapsto \textbf{return}\ x, \textbf{op}\ \texttt{get}\ \_\ k \mapsto k\ 2\}$$

When composing handlers $h_{\mathsf{ND}} \star (h_{\mathsf{get}} \star c_{\mathsf{c,g}})$, h_{get} is applied first, and handles get. Since h_{get} does not contain a clause for choose, it leaves (we say *"forwards"*) the choose operation to be handled by another handler. This forwarding behavior is key to the modular reuse and composition of handlers. Handler h_{ND} then takes care of the remaining effects.

$$h_{\mathsf{ND}} \star (h_{\mathsf{get}} \star c_{\mathsf{c,g}}) \leadsto^* h_{\mathsf{ND}} \star c_{\mathsf{ND}} \leadsto^* [1,2]$$

2.1.4. *Effect Interaction.* One of the valuable features of the modular composition of algebraic effects & handlers is that effects can interact differently by applying their handlers in a different order. Consider the effect $\texttt{inc} : () \twoheadrightarrow \mathsf{Int}$, which produces an (incremented) integer. The handler h_{inc} turns computations into state-passing functions.

$$\begin{aligned} h_{\mathsf{inc}} = \mathbf{handler}\ \{&\mathbf{return}\ x \quad\ \mapsto \mathbf{return}\ (\boldsymbol{\lambda} s\,.\,\mathbf{return}\ (x, s)) \\ ,&\mathbf{op}\ \texttt{inc}\ \_\ k \mapsto \mathbf{return}\ (\boldsymbol{\lambda} s\,.\,\mathbf{do}\ s' \leftarrow s + 1\,;\mathbf{do}\ k' \leftarrow k\ s'\,; k'\ s')\} \end{aligned}$$

The state s represents the current counter value. On every occurrence of `inc`, the incremented value is passed to the continuation twice: (1) for updating the counter value and (2) for returning the result of the operation. The latter is for the continuation and the former for serving the next `inc` operation. We use syntactic sugar to apply the initial counter value to the result of h_{inc}.

$$run_{\mathsf{inc}}\ s\ c \;\equiv\; \mathbf{do}\ c' \leftarrow h_{\mathsf{inc}} \star c\,; c'\ s$$

Computation c_{inc} combines `choose` and `inc`:

$$\begin{aligned} c_{\mathsf{inc}} = \mathbf{op}\ \texttt{choose}\ ()\ (b\,.\,\mathbf{if}\ b\ &\mathbf{then}\ \mathbf{op}\ \texttt{inc}\ ()\ (x\,.\,x + 5) \\ &\mathbf{else}\ \ \mathbf{op}\ \texttt{inc}\ ()\ (y\,.\,y + 2)) \end{aligned}$$

When handling `inc` first, each `choose` branch gets the same initial counter value.

$$\begin{aligned} &\ h_{\mathsf{ND}} \star run_{\mathsf{inc}}\ 0\ c_{\mathsf{inc}} \\ \leadsto^* &\ h_{\mathsf{ND}} \star \mathbf{op}\ \texttt{choose}\ ()\ (b\,.\,\mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star (\mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \texttt{inc}\ ()\ (x\,.\,x + 5) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \ \mathbf{op}\ \texttt{inc}\ ()\ (y\,.\,y + 2))\,; p'\ 0) \\ \leadsto^* &\ \mathbf{return}\ [(6,1),(3,1)] \end{aligned}$$

In contrast, when handling `choose` first, the counter value is threaded through the successive branches.

$$\begin{aligned} &\ run_{\mathsf{inc}}\ 0\ (h_{\mathsf{ND}} \star c_{\mathsf{inc}}) \\ \leadsto^* &\ run_{\mathsf{inc}}\ 0\ (\mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star \mathbf{op}\ \texttt{inc}\ ()\ (x\,.\,x + 5) \\ &\qquad\qquad\quad \mathbf{do}\ ys \leftarrow h_{\mathsf{ND}} \star \mathbf{op}\ \texttt{inc}\ ()\ (y\,.\,y + 2)) \\ &\qquad\qquad\qquad \mathbf{return}\ xs \mathbin{+\!\!+} ys) \\ \leadsto^* &\ ([6,4],2) \end{aligned}$$

2.2. **Scoped effects.** Not all effects are algebraic. For example, some have a delimited scope. Consider the $\texttt{once} : () \twoheadrightarrow ()$ operation, which takes a computation that contains `choose` calls and returns only its first result [PSWJ18]. Subscripting ✗ to indicate an erroneous example, we could attempt to syntactically write this as the algebraic operation $\mathbf{op}\ \texttt{once}_{\text{✗}}\ ()\ (y\,.\,c)$ and try to limit the first of two `choose` operations. We extend h_{ND} with a clause for `once`.

$$\begin{aligned} c_{\mathsf{once}\text{✗}} = &\ \mathbf{do}\ p \leftarrow \mathbf{op}\ \texttt{once}_{\text{✗}}\ \ ()\ (\_\,.\,\mathbf{op}\ \texttt{choose}\ ()\ (b\,.\,\mathbf{return}\ b)) \\ &\ \mathbf{do}\ q \leftarrow \mathbf{op}\ \texttt{choose}\ ()\ (b\,.\,\mathbf{return}\ b) \\ &\qquad \mathbf{return}\ (p, q) \\ h_{\mathsf{once}\text{✗}} = &\ \mathbf{handler}\ \{\ \ldots, \mathbf{op}\ \texttt{once}_{\text{✗}}\ \_\ k \mapsto \mathbf{do}\ ts \leftarrow k\ ()\,; \mathsf{head}\ ts\} \end{aligned}$$

We intend for $h_{\mathsf{once}\text{✗}} \star c_{\mathsf{once}\text{✗}}$ to return $[(\mathsf{true},\mathsf{true}),(\mathsf{true},\mathsf{false})]$ as the first `choose` is limited by `once`$_\text{✗}$ to only return the first alternative. The second `choose` is out of scope of `once`$_\text{✗}$, so should still return both results. However, algebraicity pulls the the second `choose` inside the scope of `once`$_\text{✗}$:

$$
\begin{array}{ll}
 & h_{\mathsf{once}\text{✗}} \star (\mathbf{do}\ p \leftarrow \mathbf{op}\ \mathtt{once}_\text{✗}\ \ ()\ (\_ .\, \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\, \mathbf{return}\ b)) \\
 & \qquad\qquad \mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,\mathbf{return}\ b) \\
 & \qquad\qquad\quad \mathbf{return}\ (p,q)) \\
\rightsquigarrow & h_{\mathsf{once}\text{✗}} \star \mathbf{op}\ \mathtt{once}_\text{✗}\ ()\ (\_ .\, \mathbf{do}\ p \leftarrow \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,\mathbf{return}\ b) \\
 & \qquad\qquad\qquad\qquad\qquad \mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,\mathbf{return}\ b) \\
 & \qquad\qquad\qquad\qquad\qquad\quad \mathbf{return}\ (p,q)) \\
\rightsquigarrow^* & [(\mathsf{true},\mathsf{true})]
\end{array}
$$

There are many more examples of operations that have a scope; we present them in Section 7:

- `catch` for catching exceptions that are raised during program execution;
- `local` for creating local variables (local state);
- `call` for creating a scope in a nondeterministic program, where branches can be cut using the algebraic `cut` operation;
- `depth` for bounding the depth in the depth-bounded search strategy;

Following Wu et al. [WSH14], we call them *scoped operations*. Plotkin and Power [PP03] have already realised that algebraic effects are unable to represent so-called generic effects (e.g., scoped) and propose to model them as handlers. Although used [TRWS22], their solution it is problematic in terms of modularity [WSH14, YPW$^+$22]: it merges syntax and semantics, as they define some effects as handlers. In Section 7 we revisit this issue, showing attempts at encoding scoped effects as handlers, their problems, and how λ_{sc} remedies the situation.

The goal of this work is to implement scoped effects while maintaining a separation between syntax and semantics, and thus preserve modular composition and control over effect interaction. It follows a line of research [PSWJ18, WSH14, YPW$^+$22] that has developed denotational semantic domains, backed by categorical models. What is lacking from the literature is a calculus that allows programming with both algebraic and scoped operations and their handlers.

3. Design Decisions & Challenges

This section informally discusses the design of λ_{sc}, a novel calculus with support for scoped effects & handlers as built-in features. We present our main challenges and design choices.

3.1. Eff **with Koka-based Row Typing.**

Our calculus is based on Eff [Pre15, BP13, BP15], an existing calculus for algebraic effects & handlers. It supports row-based typing in the style of Koka [Lei17]. Computations have types of shape $A\,!\,\langle E\rangle$, where A is the type of the value returned by the computation, and E is a collection effects that *can* occur during its evaluation. For example, $\mathsf{Bool},\mathsf{Bool}\,!\,\langle\mathtt{choose},\mathtt{once}\rangle$ is a type of c_{once}.

Handlers turn one computation into another. Their type reflects that: handlers of type $A\,!\,\langle E\rangle \Rightarrow B\,!\,\langle F\rangle$ take a computation of type $A\,!\,\langle E\rangle$ and return a computation of type $B\,!\,\langle F\rangle$. For example, h_{once} handles `choose` and `once`.

$$h_{\mathsf{once}} : (\mathsf{Bool}, \mathsf{Bool}) \,!\, \langle \texttt{choose}, \texttt{once} \rangle \Rightarrow \mathsf{List}\ (\mathsf{Bool}, \mathsf{Bool}) \,!\, \langle\rangle$$

3.2. **Scoped Effects as Built-in Operations.** As we argued in Section 2.2, modeling scoped effects like `once` as handlers comes at the cost of modularity. To retain this modularity, we add scoped effects as built-in operations with a new notation, signalled by the **sc** keyword.

$$\mathbf{sc}\ \texttt{once}\ ()\ (y \,.\, c_1)\ (z \,.\, c_2)$$

Similar to algebraic operations, scoped operations feature a label `once` to identify the effect, a parameter—in this case ()—, and a continuation $(z \,.\, c_2)$. However, scoped operations differ from algebraic operations by their additional *scoped computation* $(y \,.\, c_1)$. For `once`, the scoped computation entails the computation to be restricted to the first result (i.e. the computation in scope). The dataflow allows for a value y to be passed from the operation to c_1 and a value z from c_1 to c_2.

Adding scoped operations gives rise to a variant of the algebraicity property, which models the desired behavior of sequencing for scoped operations: scoped operations commute with sequencing in the continuation, but leave the scoped computation intact.

$$\mathbf{do}\ x \leftarrow \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2)\,;\, c_3 \ \equiv\ \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, \mathbf{do}\ x \leftarrow c_2\,;\, c_3)$$

Using `once` as a scoped operation correctly restrict only the first `choose`:

$$\begin{aligned}
c_{\mathsf{once}} &= \mathbf{sc}\ \texttt{once}\ ()\ (\_ \,.\, \mathbf{op}\ \texttt{choose}\ (b \,.\, \mathbf{return}\ b)) \\
&\qquad\qquad\qquad\quad (p \,.\, \mathbf{do}\ q \leftarrow \mathbf{op}\ \texttt{choose}\ (b \,.\, \mathbf{return}\ b)\,;\, \mathbf{return}\ (p, q)) \\
h_{\mathsf{once}} &= \mathbf{handler}\ \{\ \ldots, \mathbf{sc}\ \texttt{once}\ \_\ p\ k \mapsto \mathbf{do}\ ts \leftarrow p\ ()\,;\, \mathbf{do}\ t \leftarrow \mathsf{head}\ ts\,;\, k\ t\} \\
h_{\mathsf{once}} &\star c_{\mathsf{once}} \leadsto^{*} [(\mathsf{true}, \mathsf{true}), (\mathsf{true}, \mathsf{false})]
\end{aligned}$$

This novel representation for scoped effects & handlers also brings in additional complexity. Whereas algebraic operations contain a single subcomputation, scoped operations contain *two* of them: the scoped computation and the continuation. The result of the scoped computation is the argument of the continuation: they must agree on a type of this result, which we name the *scoped result type*. For example, consider c_{once}, with overall type $(\mathsf{Bool}, \mathsf{Bool})\,!\,\langle \texttt{once}\,;\texttt{choose} \rangle$. Its scoped result type is (a singular) Bool: it is the type that is *produced* by the scoped computation, and *consumed* by the continuation.

$$\mathbf{sc}\ \texttt{once}\ ()\ \underbrace{(\_ \,.\, \mathbf{op}\ \texttt{choose}\ ()\ (b \,.\, \mathbf{return}\ b))}_{() \to \boxed{\mathsf{Bool}}\,!\,\langle \texttt{once}\,;\texttt{choose} \rangle}\ \underbrace{(p \,.\, \mathbf{do}\ q \leftarrow \mathbf{op} \ldots\ \mathbf{return}\ (p, q))}_{\boxed{\mathsf{Bool}} \to (\mathsf{Bool}, \mathsf{Bool})\,!\,\langle \texttt{once}\,;\texttt{choose} \rangle}$$

Dealing with the presence of this scoped computation type is tricky, and introduces two complications. First, since the type does not occur in the computation's overall type, *polymorphic handlers* are required to handle scoped effects. Second, the scoped result type describes a dependency between the scoped computation and continuation: if one changes its type, the other must match this. This makes generic *forwarding* impossible: it alters the type of the scoped computation, but does not make up for it in the continuation.

3.3. **Polymorphic Handlers.** Applying a handler to a computation involves recursively applying the handler to the computation's subcomputations as well. In the case of algebraic effects, these subcomputations always have the same type as the operation itself, as witnessed by the algebraicity property. This means that calculi that only support algebraic effects & handlers, such as Eff, can (and do) type handlers monomorphically, without severe limitations.

However, typing scoped effect handlers monomorphically *does* limit their implementation freedom: it only allows scoped operations of which the scoped result type matches the operation's overall type. For example, consider the type $(\mathsf{Bool}, \mathsf{Bool})\,!\,\langle \mathtt{choose}\,; \mathtt{once}\rangle \Rightarrow \mathsf{List}\ (\mathsf{Bool}, \mathsf{Bool})\,!\,\langle\rangle$ we previously assigned to h_{once}. This monomorphic type requires the scoped result type to be $(\mathsf{Bool}, \mathsf{Bool})$ as well, as it is the only type of computation monomorphic h_{once} can handle. This is not the case for c_{once}: as established, its scoped result type is Bool (see above). Therefore, scoped computations such as c_{once}, cannot be handled by monomorphic handlers. The solution is to let handlers abstract over the value type of computations, allowing for the handling of scoped operations with *any* scoped result type. This way, h_{once} can be typed as follows:

$$h_{\mathsf{once}} : \forall\, \alpha\,.\,\alpha\,!\,\langle \mathtt{choose}\,; \mathtt{once}\rangle \Rightarrow \mathsf{List}\ \alpha\,!\,\langle\rangle$$

With this polymorphic typing in place, $h_{\mathsf{once}} \star c_{\mathsf{once}}$ can now be evaluated by *polymorphic recursion*. To support this, λ_{sc} features **let**-polymorphism, F_ω-style type operators, and requires all handlers for scoped effects to be polymorphic.

3.4. **Forwarding Unknown Operations.** In order to retain the modularity of composing different effects, as discussed in Section 2.1.3, we write dedicated handlers that interpret only their part of the syntax, and *forward* all remaining operations to other handlers. For algebraic effects forwarding happens generically. For example, consider the forwarding of h_{once} applied to an algebraic operation with `inc`.

$$h_{\mathsf{once}} \star \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,\mathbf{return}\ y) \rightsquigarrow \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,h_{\mathsf{once}} \star \mathbf{return}\ y)$$

One might hope to forward scoped effects in a similar way. For example, consider applying h_{once} to scoped operation $\mathtt{catch} : \mathsf{String} \twoheadrightarrow \mathsf{Bool}$ for catching exceptions.

$$\begin{aligned} c_{\mathsf{catch}} &= \mathbf{sc}\ \mathtt{catch}\ \texttt{"err"}\ (b\,.\,\mathbf{if}\ b\ \mathbf{then}\ \mathbf{return}\ 1\ \mathbf{else}\ \mathbf{return}\ 2)\ (x\,.\,\mathbf{return}\ x) \\ &\quad h_{\mathsf{once}} \star c_{\mathsf{catch}} \\ \rightsquigarrow_{\text{✗}}&\ \mathbf{sc}\ \mathtt{catch}\ \texttt{"err"}\ (b\,.\,h_{\mathsf{once}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{return}\ 1 \ldots)\ (x\,.\,h_{\mathsf{once}} \star \mathbf{return}\ x)\end{aligned}$$

Unfortunately, this does not work. Again, the hurdle is in the scoped result type. In particular, h_{once} introduces a type operator List when handling a computation. The scoped computation now has type $\mathsf{Bool} \to \mathsf{List}\ \mathsf{Int}\,!\,\langle\mathtt{catch}\rangle$, whereas the continuation has type $\mathsf{Int} \to \mathsf{List}\ \mathsf{Int}\,!\,\langle\mathtt{catch}\rangle$.

$$\mathbf{sc}\ \mathtt{catch}\ \texttt{"err"}\ \underbrace{(b\,.\,h_{\mathsf{once}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{return}\ 1\ \mathbf{else} \ldots)}_{\mathsf{Bool}\to \boxed{\mathsf{List}\ \mathsf{Int}}\,!\,\langle\mathtt{catch}\rangle}\ \underbrace{(x\,.\,h_{\mathsf{once}} \star \mathbf{return}\ x)}_{\boxed{\mathsf{Int}}\,\to \mathsf{List}\ \mathsf{Int}\,!\,\langle\mathtt{catch}\rangle}$$

Indeed, applying a handler to a computation changes its type: not only does it remove labels from the effect row, it also may apply a type operator —in this case List—to the type. For scoped operations this is problematic, as the return type of the scoped computation has changed ($\mathsf{List}\ \mathsf{Int}$), whereas the continuation still expects the original type (Int). Thus, scoped effects cannot be forwarded generically. Therefore, we require that every handler is equipped with an explicit forwarding clause for unknown scoped operations. When a handler is defined it is clear what type operator the handler applies. With this information it is possible to define a way of bridging the type discrepancy. For example, h_{once} mitigates the discrepancy between $\mathsf{List\ Int}$ and Int by settling the scoped result type on $\mathsf{List\ Int}$: in the forwarding clause of h_{once}, $\mathsf{concatMap}$ ensures that the transformed continuation now takes a value of type $\mathsf{List\ Int}$ as argument.

$$h_{\mathsf{once}} = \mathbf{handler}\ \{\ \ldots, \mathbf{fwd}\ f\ p\ k \mapsto f\ (p, (\boldsymbol{\lambda} z\,.\,\mathsf{concatMap}\ z\ k))\}$$

We implement forwarding by means of a function f, which is a partial application of $\mathbf{sc}\ \ell\ v$: it takes the pair of a (possibly transformed) scoped computation p' and continuation k' and re-introduces the to-be-forwarded scoped operation with these parameters.

$$f = \boldsymbol{\lambda}(p', k')\,.\,\mathbf{sc}\ \ell\ v\ (y\,.\,p'\ y)\ (z\,.\,k'\ z)$$

The $\mathsf{concatMap}$ is standardly defined as follows.

$$\begin{array}{ll}
\mathsf{concatMap} & : \forall\ \alpha\ \beta\ \mu\,.\,\mathsf{List}\ \beta \to^{\mu} (\beta \to^{\mu} \mathsf{List}\ \alpha) \to^{\mu} \mathsf{List}\ \alpha \\
\mathsf{concatMap}\ [\,] & f = \mathbf{return}\ [\,] \\
\mathsf{concatMap}\ (b : bs)\ f & = \mathbf{do}\ as \leftarrow f\ \ b\,;\, as' \leftarrow \mathsf{concatMap}\ bs\ f\,;\, as \mathbin{+\!\!+} as'
\end{array}$$

In what follows, we put our calculus on formal footing, discussing its syntax, operational semantics and type-and-effect system.

4. Syntax

As stated, λ_{sc} is based on Eff [BP13, BP15]. Before adding support for scoped effects, we have altered Eff from its presentation in [BP13, BP15] in two ways. Firstly, we have made a number of cosmetic changes that arguably improve the readability of the calculus. Secondly, we adopt row-based typing in the style of Koka [Lei17].

Figure 1 displays the syntax of λ_{sc}. The extensions to (our version of) Eff made to support scoped effects are highlighted throughout our presentation of λ_{sc}. The extensions can be summarized by two new handler clauses, a new operation call and the inclusion of let-polymorphism in the terms, and type variables in the types.

4.1. **Terms.**

Like Eff we implement fine-grained call-by-value semantics [LPT03]. Therefore, terms are split into inert values and computations that can be reduced.

Computations. For computations, **return** can be used to return values. Handlers can be applied to values by usage of the $\star$-operator. As seen before, computations may be sequenced by means of do-statements ($\mathbf{do}\ x \leftarrow c_1\,;\, c_2$). Applications reduce, so are computations. As discussed in Section 3.3, to support polymorphic handlers we support let-polymorphism and thus let-bindings. Finally, a computation may be the invocation of an effect by means of an operation.

To be able to differentiate between algebraic and scoped effects, we add the effect keyword **sc** to model scoped effects. Consequently, **op** now ranges over algebraic effects only. Furthermore, we annotate labels with either op or sc to indicate if they are the label of an algebraic or scoped effect, respectively. We implicitly assume any label ℓ occurs either as an algebraic or scoped effect label. Like their algebraic counterparts, scoped effect operations feature a label ℓ^{op} or ℓ^{sc}, argument v and continuation $(y\,.\,c)$ or $(z\,.\,c_2)$. In addition, scoped

$$
\begin{array}{rcll}
\text{values } v & ::= & () \mid (v_1, v_2) \mid x \mid \boldsymbol{\lambda} x \,.\, c \mid h & \\
\text{handlers } h & ::= & \mathbf{handler}\ \{\mathbf{return}\ x \mapsto c_r & \text{return clause} \\
 & & \quad ,\ \mathit{oprs} & \text{effect clauses} \\
 & & \quad ,\ \mathbf{fwd}\, f\ p\ k \mapsto c_f\ \} & \text{forwarding clause} \\
\text{operation clauses } \mathit{oprs} & ::= & \cdot & \\
 & \mid & \mathbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs} & \text{algebraic effect clauses} \\
 & \mid & \mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, \mathit{oprs} & \text{scoped effect clauses} \\
\text{computations } c & ::= & \mathbf{return}\ v & \text{return value} \\
 & \mid & \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, c) & \text{algebraic operation} \\
 & \mid & \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2) & \text{scoped operation} \\
 & \mid & v \star c & \text{handle} \\
 & \mid & \mathbf{do}\ x \leftarrow c_1 \,;\, c_2 & \text{do-statement} \\
 & \mid & v_1\ v_2 & \text{application} \\
 & \mid & \mathbf{let}\ x = v\ \mathbf{in}\ c & \text{let} \\
\text{value types } A,\ B,\ M & ::= & () \mid (A, B) \mid A \to \underline{C} \mid \underline{C} \Rightarrow \underline{D} & \\
 & \mid & \alpha & \text{type variable} \\
 & \mid & \lambda\, \alpha \,.\, A & \text{type operator abstraction} \\
 & \mid & M\ A & \text{type application} \\
\text{type schemes } \sigma & ::= & A \mid \forall\, \mu \,.\, \sigma \mid \forall\, \alpha \,.\, \sigma & \\
\text{computation types } \underline{C},\ \underline{D} & ::= & A\,!\,\langle E \rangle & \\
\text{effect rows } E,\ F & ::= & \cdot \mid \mu \mid \ell \,;\, E & \\
\text{signature contexts } \Sigma & ::= & \cdot \mid \Sigma, \ell^{\mathsf{op}} : A \twoheadrightarrow B \mid \Sigma, \ell^{\mathsf{sc}} : A \twoheadrightarrow B & \\
\text{type contexts } \Gamma & ::= & \cdot \mid \Gamma, x : \sigma \mid \Gamma, \mu \mid \Gamma, \alpha &
\end{array}
$$

Figure 1: Syntax λ_{sc}.

effect operations feature a scoped computation $(y \,.\, c_1)$. This way, the scope of effect ℓ^{sc} is delimited: (scoped) computation $(y \,.\, c_1)$ is in scope, continuation $(z \,.\, c_2)$ is not.

Values. Values consist of the unit value $()$, value pairs (v_1, v_2), variables x, functions $\boldsymbol{\lambda} x \,.\, c$ and handlers h. Handlers h have three kinds of clauses: one return clause, zero or more operation clauses, and a forwarding clause.

The return clause $\mathbf{return}\ x \mapsto c$ denotes that the result x of a computation is processed by computation c.

Algebraic operation clauses $\mathbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c$ specify that handling an effect with label ℓ^{op}, parameter x and continuation k is processed by computation c (e.g., h_{ND}, h_{get}, h_{inc}). In this rule, k is an object-level variable just like x. For scoped effect clauses the extension is analogous to the operation case: we take the algebraic clause and add support for a scoped computation, which in the case for the clause has the form of parameter p.

Finally, as motivated in Section 3.4, we have forwarding clauses of shape $\mathbf{fwd}\, f\ p\ k \mapsto c_f$, that deal with forwarding unknown scoped operation with some label ℓ^{sc}. Computations c_f have access to the scoped computation p and continuation k of the unknown effect they are forwarding. Furthermore, c_f should be able to call ℓ^{sc}. Instead of bringing ℓ^{sc} into scope, we pass it f which in turn invokes ℓ^{sc}. This achieves a simpler type system at no cost to the expressivity of forwarding clauses.

4.2. **Types.** Like terms, types are split: values have value types A, B, computations have computation types $\underline{C}, \underline{D}$. Value types consist of the unit type $()$, pair types (A, B), function types $A \to \underline{C}$, handler types $\underline{C} \Rightarrow \underline{D}$, type variables α and abstraction over them $\boldsymbol{\lambda}\alpha \,.\, A$, and type application $M\ A$. Following convention and to allow for meaningful examples, we may add base value types to the calculus, such as String, Int and Bool. Functions take a value of type A as argument and return a computation of type $\underline{C}$; handlers take a computation of type $\underline{C}$ as argument and return a computation of type $\underline{D}$.

A computation type $A\,!\,\langle E\rangle$ consists of a value type A, representing the type of the value the computation evaluates to, and an effect type E, representing the effects that *may* be called during this evaluation. Different from Eff, we implement effect types as effect rows using row polymorphism [Lei05] in the style of Koka [Lei17]. Therefore, rows E are represented as collections of the previously discussed atomic labels ℓ^{op}, optionally terminated by a row variable μ. Finally, we can abstract over both type and row variables, giving rise to type schemes σ.

5. Operational Semantics

Figure 2 displays the small-step operational semantics of λ_{sc}. Here, relation $c \rightsquigarrow c'$ denotes that computation c steps to computation c', with $\rightsquigarrow^*$ its reflexive, transitive closure. The highlighted rules deal with the extensions that support scoped effects. The following discussion of the semantics is exemplified by snippets of derivations of computations[1]used in Section 2. We refer to Appendix A for the full version of these derivations.

Rules E-AppAbs and E-Let deal with function application and let-binding, respectively, and are standard. The rest of the rules consist of two parts: sequencing and handling.

Sequencing. For sequencing computations $\mathbf{do}\ x \leftarrow c_1\,; c_2$, we distinguish between the situation where c_1 can take a step (E-Do), and where c_1 is in normal form (**return**, **op**, or **sc**). First, if c_1 returns a value v, we substitute v for x in c_2 (E-DoRet). Second, if c_1 is an algebraic operation, we rewrite the computation using the algebraicity property (E-DoOp), bubbling up the algebraic operation to the front of the computation. Third, the new case, where c_1 is a scoped operation, is analogous: the generalization of the algebraicity property (Section 3.2) is used to rewrite the computation (E-DoSc).

Handling. For handling computations with a handler of the form $h \star c$, we distinguish six situations. First, if possible, c takes a step (E-Hand); in the other cases, c is in normal form. If c returns a value v, we use the handler's return clause $\mathbf{return}\ x \mapsto c_r$, switching evaluation to c_r with x replaced by v (E-HandRet).

If computation c is an algebraic operation $\mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c_1)$, its label is looked up in the handler h. If the handler contains an algebraic clause with this label, evaluation switches to the clause's computation c (E-HandOp), with v substituted for parameter x and continuation k replaced by a function that, given the original argument y, contains the already-handled continuation. For example, $h_{\mathsf{ND}} \star c_{\mathsf{ND}}$ (p. 3) reduces as follows.

$$\begin{array}{ll} & h_{\mathsf{ND}} \star c_{\mathsf{ND}} \\ \rightsquigarrow & \mathbf{do}\ xs \leftarrow (\boldsymbol{\lambda} b \,.\, h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{true} \\ & \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b \,.\, h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{false} \end{array}$$

[1]Following convention, these examples may contain elements not present in our calculus, such as integers and if-then-else statements. These may be viewed as syntactic sugar for their Church encodings.

$$\boxed{c \leadsto c'} \quad \text{Computation reduction}$$

$$\frac{}{(\boldsymbol{\lambda} x \,.\, c)\ v \leadsto c\,[v \,/\, x]}\ \textsc{E-AppAbs} \qquad \frac{}{\mathbf{let}\ x = v\ \mathbf{in}\ c \leadsto c\,[v \,/\, x]}\ \textsc{E-Let}$$

$$\frac{c_1 \leadsto c_1'}{\mathbf{do}\ x \leftarrow c_1 \,;\, c_2 \leadsto \mathbf{do}\ x \leftarrow c_1' \,;\, c_2}\ \textsc{E-Do} \qquad \frac{}{\mathbf{do}\ x \leftarrow \mathbf{return}\ v \,;\, c_2 \leadsto c_2\,[v \,/\, x]}\ \textsc{E-DoRet}$$

$$\frac{}{\mathbf{do}\ x \leftarrow \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, c_1) \,;\, c_2 \leadsto \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, \mathbf{do}\ x \leftarrow c_1 \,;\, c_2)}\ \textsc{E-DoOp}$$

$$\frac{}{\mathbf{do}\ x \leftarrow \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2) \,;\, c_3 \leadsto \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, \mathbf{do}\ x \leftarrow c_2 \,;\, c_3)}\ \textsc{E-DoSc}$$

$$\frac{c \leadsto c'}{h \star c \leadsto h \star c'}\ \textsc{E-Hand} \qquad \frac{(\mathbf{return}\ x \mapsto c_r)\ \in\ h}{h \star \mathbf{return}\ v \leadsto c_r\,[v \,/\, x]}\ \textsc{E-HandRet}$$

$$\frac{(\mathbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c)\ \in\ h}{h \star \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, c_1) \leadsto c\,[v \,/\, x, (\boldsymbol{\lambda} y \,.\, h \star c_1) \,/\, k]}\ \textsc{E-HandOp}$$

$$\frac{(\mathbf{op}\ \ell^{\mathsf{op}}\ \_\ \_)\ \notin\ h}{h \star \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, c_1) \leadsto \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, h \star c_1)}\ \textsc{E-FwdOp}$$

$$\frac{(\mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c)\ \in\ h}{h \star \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2) \leadsto c\,[v \,/\, x, (\boldsymbol{\lambda} y \,.\, h \star c_1) \,/\, p, (\boldsymbol{\lambda} z \,.\, h \star c_2) \,/\, k]}\ \textsc{E-HandSc}$$

$$\frac{(\mathbf{sc}\ \ell^{\mathsf{sc}}\ \_\ \_\ \_)\ \notin\ h \qquad (\mathbf{fwd}\ f\ p\ k \mapsto c_f)\ \in\ h \\ g = \boldsymbol{\lambda}(p', k') \,.\, \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, p'\ y)\ (z \,.\, k'\ z)}{h \star \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2) \leadsto c_f\ [(\boldsymbol{\lambda} y \,.\, h \star c_1) \,/\, p, (\boldsymbol{\lambda} z \,.\, h \star c_2) \,/\, k, g \,/\, f]}\ \textsc{E-FwdSc}$$

Figure 2: Operational semantics of λ_{sc}.

$$\begin{aligned} & \quad xs +\!\!+ ys \\ \leadsto^* & \ \mathbf{return}\ [1, 2] \end{aligned}$$

In case h does not contain a clause for label ℓ^{op}, the effect is forwarded (E-FwdOp). Algebraic effects can be forwarded generically: we re-invoke the operation and recursively apply the handler to continuation c_1. For example, during the application of run_{inc} in $h_{\mathsf{ND}} \star run_{\mathsf{inc}}\ 0\ c_{\mathsf{inc}}$ (p. 4), `choose` is forwarded:

$$\begin{aligned} & h_{\mathsf{ND}} \star run_{\mathsf{inc}}\ 0\ c_{\mathsf{inc}} \\ \leadsto\ & h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow \mathbf{op}\ \texttt{choose}\ ()\ (b \,.\, h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \texttt{inc}\ ()\ (x \,.\, x + 5) \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \mathbf{op}\ \texttt{inc}\ ()\ (y \,.\, y + 2)) \,;\, p'\ 0 \\ \leadsto^*\ & \mathbf{return}\ [(6, 1), (3, 1)] \end{aligned}$$

If computation c is a scoped operation $\mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2)$, we again distinguish two situations: the case where h contains a clause for ℓ^{sc}, and where it does not. If h contains a clause for label ℓ^{sc}, evaluation switches to the clause's computation c (E-HandSc), with v substituted for parameter x. Both the scoped computation and the continuation are replaced by a function that contains the already-handled computations c_1 and c_2. For example, this happens for the scoped operation `once` in $h_{\mathsf{once}} \star c_{\mathsf{once}}$ (p. 6).

$$\begin{array}{ll} & h_{\mathsf{once}} \star c_{\mathsf{once}} \\ \leadsto & \textbf{do}\ ts \leftarrow (\boldsymbol{\lambda}\_ \,.\, h_{\mathsf{once}} \star \textbf{op}\ \texttt{choose}\ (b \,.\, \textbf{return}\ b))\ () \\ & \textbf{do}\ t\ \leftarrow \mathsf{head}\ ts \\ & \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\textbf{do}\ q \leftarrow \textbf{op}\ \texttt{choose}\ (b \,.\, \textbf{return}\ b) \,;\textbf{return}\ (p, q)))\ t \\ \leadsto^{*} & \textbf{return}\ [(\mathsf{true}, \mathsf{true}), (\mathsf{true}, \mathsf{false})] \end{array}$$

When h does not contain a clause for label ℓ^{sc}, we must forward the effect. As we argued in Section 3.4, forwarding scoped effects cannot happen generically, but rather proceeds via the handler's forwarding clause $\textbf{fwd}\ f\ p\ k \mapsto c_f$. From there, evaluation switches to computation c_f, in which the usages of scoped computation p and continuation k are replaced by their already-handled equivalents. Computation c_f may reinvoke the unkown scoped operation ℓ^{sc} by means of the parameter f which, when called with a scoped computation p' and continuation k' invokes the unkown scoped operation using the passed computations $\textbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, p'\ y)\ (z \,.\, k'\ z)$.

For a computation, consider again the example described in Section 3.4. Even though we have not given any semantics to `catch` yet (we will do so in Section 7.1), we know h_{once} does not contain a clause for `catch`. As described, `catch` must be forwarded, addressing the type mismatch between the handled scoped computation and handled continuation with $\mathsf{concatMap}$.

$$\begin{array}{ll} & h_{\mathsf{once}} \star c_{\mathsf{catch}} \\ \leadsto & (\boldsymbol{\lambda}(p', k') \,.\, \textbf{sc}\ \texttt{catch}\ \texttt{"err"}\ (b \,.\, p'\ b)\ (x \,.\, k'\ x)) \\ & \quad ((\boldsymbol{\lambda} b \,.\, h_{\mathsf{once}} \star \textbf{if}\ b\ \textbf{then}\ \textbf{return}\ 1\ \textbf{else}\ \textbf{return}\ 2) \\ & \quad ,(\boldsymbol{\lambda} z \,.\, \mathsf{concatMap}\ z\ (\boldsymbol{\lambda} x \,.\, h_{\mathsf{once}} \star \textbf{return}\ x))) \\ \leadsto & \textbf{sc}\ \texttt{catch}\ \texttt{"err"}\ (b \,.\, (\boldsymbol{\lambda} b \,.\, h_{\mathsf{once}} \star \textbf{if}\ b\ \textbf{then}\ \textbf{return}\ 1\ \textbf{else}\ \textbf{return}\ 2)\ b) \\ & \qquad\qquad\qquad (x \,.\, (\boldsymbol{\lambda} z \,.\, \mathsf{concatMap}\ z\ (\boldsymbol{\lambda} x \,.\, h_{\mathsf{once}} \star \textbf{return}\ x))\ x) \end{array}$$

6. Type-and-Effect System

This section presents the type-and-effect system of λ_{sc}. As before, we distinguish between values, computations and handlers.

6.1. **Value typing.**

Figure 3 displays the typing rules for values. Rules T-Var, T-Unit, T-Pair and T-Abs type variables, units, pairs and term abstractions, respectively, and are standard.

Rule T-EqV expresses that typing holds up to equivalence of types. The full type equivalence relation ($A \equiv B$), which also uses the equivalence of rows ($E \equiv_{\langle\rangle} F$), is included in Appendix B. However, these relations can be described as the congruence closure of the following two rules.

$$\frac{}{(\lambda\, \alpha \,.\, A)\ B \;\equiv\; A\,[B \,/\, \alpha]}\ \textsc{Q-AppAbs} \qquad \frac{\ell_1 \neq \ell_2}{\ell_1 \,;\ell_2 \,; E \;\equiv_{\langle\rangle}\; \ell_2 \,; \ell_1 \,; E}\ \textsc{R-Swap}$$

Rule Q-AppAbs captures type application, following F_ω, and R-Swap captures the insignificance of the order in effect rows, following Koka's row typing approach.

$$\boxed{\Gamma \vdash v : \sigma} \quad \text{Value Typing}$$

$$\frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}\ \textsc{T-Var} \qquad \frac{}{\Gamma \vdash () : ()}\ \textsc{T-Unit} \qquad \frac{\Gamma \vdash v_1 : A \qquad \Gamma \vdash v_2 : B}{\Gamma \vdash (v_1, v_2) : (A, B)}\ \textsc{T-Pair}$$

$$\frac{\Gamma, x : A \vdash c : \underline{C}}{\Gamma \vdash \boldsymbol{\lambda} x \,.\, c : A \to \underline{C}}\ \textsc{T-Abs} \qquad \frac{\Gamma \vdash v : A \qquad A \equiv B}{\Gamma \vdash v : B}\ \textsc{T-EqV}$$

$$\frac{\Gamma \vdash v : \forall\, \alpha \,.\, \sigma \qquad \Gamma \vdash A}{\Gamma \vdash v : [A \,/\, \alpha]\, \sigma}\ \textsc{T-Inst} \qquad \frac{\Gamma, \alpha \vdash v : \sigma \qquad \alpha \notin \Gamma}{\Gamma \vdash v : \forall\, \alpha \,.\, \sigma}\ \textsc{T-Gen}$$

$$\frac{\Gamma \vdash v : \forall\, \mu \,.\, \sigma \qquad \Gamma \vdash E}{\Gamma \vdash v : [E \,/\, \mu]\, \sigma}\ \textsc{T-InstEff} \qquad \frac{\Gamma, \mu \vdash v : \sigma \qquad \mu \notin \Gamma}{\Gamma \vdash v : \forall\, \mu \,.\, \sigma}\ \textsc{T-GenEff}$$

Figure 3: Value typing.

$$\boxed{\Gamma \vdash c : \underline{C}} \quad \text{Computation Typing}$$

$$\frac{\Gamma \vdash v_1 : A \to \underline{C} \qquad \Gamma \vdash v_2 : A}{\Gamma \vdash v_1\ v_2 : \underline{C}}\ \textsc{T-App} \qquad \frac{\Gamma \vdash c_1 : A\,!\langle E\rangle \qquad \Gamma, x : A \vdash c_2 : B\,!\langle E\rangle}{\Gamma \vdash \mathbf{do}\ x \leftarrow c_1\,;\, c_2 : B\,!\langle E\rangle}\ \textsc{T-Do}$$

$$\frac{\Gamma \vdash c : \underline{C} \qquad \underline{C} \equiv \underline{D}}{\Gamma \vdash c : \underline{D}}\ \textsc{T-EqC} \qquad \frac{\Gamma \vdash v : \sigma \qquad \Gamma, x : \sigma \vdash c : \underline{C}}{\Gamma \vdash \mathbf{let}\ x = v\ \mathbf{in}\ c : \underline{C}}\ \textsc{T-Let}$$

$$\frac{\Gamma \vdash v : A \qquad \Gamma \vdash E}{\Gamma \vdash \mathbf{return}\ v : A\,!\langle E\rangle}\ \textsc{T-Ret} \qquad \frac{\Gamma \vdash v : \underline{C} \Rightarrow \underline{D} \qquad \Gamma \vdash c : \underline{C}}{\Gamma \vdash v \star c : \underline{D}}\ \textsc{T-Hand}$$

$$\frac{(\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma \qquad \Gamma \vdash v : A_{\mathsf{op}} \qquad \Gamma, y : B_{\mathsf{op}} \vdash c : A\,!\langle E\rangle \qquad \ell^{\mathsf{op}} \in E}{\Gamma \vdash \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y \,.\, c) : A\,!\langle E\rangle}\ \textsc{T-Op}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \in \Sigma \\ \Gamma \vdash v : A_{\mathsf{sc}} \qquad \Gamma, y : B_{\mathsf{sc}} \vdash c_1 : B\,!\langle E\rangle \qquad \Gamma, z : B \vdash c_2 : A\,!\langle E\rangle \qquad \ell^{\mathsf{sc}} \in E\end{array}}{\Gamma \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y \,.\, c_1)\ (z \,.\, c_2) : A\,!\langle E\rangle}\ \textsc{T-Sc}$$

Figure 4: Computation typing.

The final four value typing rules deal with generalization and instantiation of type variables and row variables. Rule T-Inst instantiates the type variables α in a type scheme with a value type A. Rule T-Gen is its dual, abstracting over a type variable. The rules for row variables are similar: T-InstEff instantiates row variable with an effect row E; T-GenEff abstracts over a row variable. The definition of well-scopedness for types $\Gamma \vdash \sigma$ and effect rows $\Gamma \vdash E$ is straightforward (Appendix C).

6.2. **Computation typing.** Figure 4 shows the rules for computation typing. Rules T-App and T-Do capture application and sequencing, and are standard. Like value typing,

typing of computations holds up to equivalence of types (T-EqC). Rule T-Let is part of our extension of Eff, as scoped effects require introducing let-polymorphism.

Rule T-Ret assigns a computation type to a return statement. This type consists of the value v in the return, together with a effect row E. Notice that, as in Koka, this row can be freely chosen.

Rule T-Hand types handler application. The typing rules for handlers and their clauses are discussed in Section 6.3. A handler of type $C \Rightarrow D$ denotes a handler that transforms computations of type C to a computation of type D.

Rule T-Op types algebraic effects. Looking up label ℓ^{op} in Σ yields a signature $A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}$, where A_{op} is the type of the operation's parameter v, and B_{op} is the type of argument y of the continuation. The resulting effect row includes ℓ^{op}. Indeed, $\ell \in E$ means that there is some E' such that $E \equiv_{\langle\rangle} \ell\,; E'$. Finally, the operation's type equals that of continuation c.

Similarly, rule T-Sc types scoped effects. Again, looking up label ℓ^{sc} in Σ yields signature $A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}$ where A_{sc} corresponds to the type of the operation's parameter v. However, where B_{op} in the algebraic case refers to the *continuation's* argument, B_{sc} now describes the *scoped computation's* argument. This leaves the the scoped result type undescribed by the signature, but as discussed in Section 3.3, this freedom is exactly what we want. As for the effect rows, T-Sc requires the rows of the scoped computation to match.

6.3. **Handler typing.** The typing rules for handlers and handler clauses are shown in Figure 5. It consists of four judgments. Judgment $\Gamma \vdash \textbf{return}\ x \mapsto c_r : M\ \ A!\langle E\rangle$ types return clauses, $\Gamma \vdash \mathit{oprs} : M\ \ A!\langle E\rangle$ types operation clauses, and $\Gamma \vdash \textbf{fwd}\ f\ p\ k \mapsto c : M\ \ A!\langle E\rangle$ types forwarding clauses. Finally, $\Gamma \vdash h : \forall\ a\,.\,\alpha!\langle F\rangle \Rightarrow M\ \ \alpha!\langle E\rangle$ types handlers, using the first three judgments.

Return Clauses. Rule T-Return types return clauses of the form $\textbf{return}\ x \mapsto c_r$. It binds variable x to type A, adds it to the context, and returns the type $M\ \ A!\langle E\rangle$ of c_r as the type of the return clause.

Operation Clauses. The judgment $\Gamma \vdash \mathit{oprs} : M\ \ A!\langle E\rangle$ denotes that all operations in the sequence of operations *oprs* have type $M\ \ A!\langle E\rangle$. The base case T-Empty types the empty sequence. The other two cases require the head of the sequence (either **op** or **sc**) to have the same type as the tail.

Rule T-OprOp types algebraic operation clauses $\textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c$. Looking up label ℓ^{op} in Σ yields signature $A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}$, where A_{op} describes the type of parameter x, and B_{op} the type of the argument of continuation k. In order for an operation **op** to have type $M\ \ A!\langle E\rangle$, c should have the same type.

Once again, the case for typing a scoped clause $\textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c$ (T-OprSc) is similar to its algebraic equivalent, extended to include the scoped computation. Notice the type of p and k when typing c. First, as λ_{sc} allows freedom in the scoped result type, the type variable β is used for this type. Second, as shown in the operational semantics (rule E-HandSc), for a clause $\textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c$, computation c uses the *already-handled* subcomputations p and k. Therefore, type operator M occurs in the scoped result type as well as in the continuation's result type:

$$p : B_{\mathsf{sc}} \to M\ \ \beta!\langle E\rangle \qquad\qquad k : \beta \to M\ \ A!\langle E\rangle$$

$$\boxed{\Gamma \vdash \textbf{return}\; x \mapsto c_r : M\ A!\langle E\rangle} \boxed{\Gamma \vdash \mathit{oprs} : M\ A!\langle E\rangle}$$

$$\boxed{\Gamma \vdash \textbf{fwd}\; f\ p\ k \mapsto c_f : M\ A!\langle E\rangle} \quad \text{Return-, operation-, and forwarding-clause typing}$$

$$\frac{\Gamma, x : A \mapsto c_r : M\ A!\langle E\rangle}{\Gamma \vdash \textbf{return}\; x \mapsto c_r : M\ A!\langle E\rangle}\ \textsc{T-Return} \qquad \frac{}{\Gamma \vdash \cdot : M\ A!\langle E\rangle}\ \textsc{T-Empty}$$

$$\frac{\Gamma \vdash \mathit{oprs} : M\ A!\langle E\rangle \qquad (\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma \quad \Gamma, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ A!\langle E\rangle \vdash c : M\ A!\langle E\rangle}{\Gamma \vdash \textbf{op}\; \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs} : M\ A!\langle E\rangle}\ \textsc{T-OprOp}$$

$$\frac{\Gamma \vdash \mathit{oprs} : M\ A!\langle E\rangle \qquad (\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \in \Sigma \quad \Gamma, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta!\langle E\rangle, k : \beta \to M\ A!\langle E\rangle \vdash c : M\ A!\langle E\rangle}{\Gamma \vdash \textbf{sc}\; \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, \mathit{oprs} : M\ A!\langle E\rangle}\ \textsc{T-OprSc}$$

$$\frac{\begin{array}{cc} A_p = \alpha \to M\ \beta!\langle E\rangle & A'_p = \alpha \to \gamma!\langle E\rangle \\ A_k = \beta \to M\ A!\langle E\rangle & A'_k = \gamma \to \delta!\langle E\rangle \end{array} \quad \Gamma, \alpha, \beta, p : A_p, k : A_k, f : \forall\, \gamma\ \delta\,.\, (A'_p, A'_k) \to \delta!\langle E\rangle \vdash c_f : M\ A!\langle E\rangle}{\Gamma \vdash \textbf{fwd}\; f\ p\ k \mapsto c_f : M\ A!\langle E\rangle}\ \textsc{T-Fwd}$$

$$\boxed{\Gamma \vdash h : \forall\, \alpha\,.\, \alpha!\langle E\rangle \Rightarrow M\ \alpha!\langle F\rangle} \quad \text{Handler typing}$$

$$\textsc{T-Handler}\quad \frac{\begin{array}{cc} \langle F\rangle \equiv_{\langle\rangle} \langle \mathit{labels}\,(\mathit{oprs})\,; E\rangle & \Gamma, \alpha \vdash \textbf{return}\; x \mapsto c_r : M\ \alpha!\langle E\rangle \\ \Gamma, \alpha \vdash \mathit{oprs} : M\ \alpha!\langle E\rangle & \Gamma, \alpha \vdash \textbf{fwd}\; f\ p\ k \mapsto c_f : M\ \alpha!\langle E\rangle \end{array}}{\Gamma \vdash \textbf{handler}\; \{\textbf{return}\; x \mapsto c_r, \mathit{oprs}, \textbf{fwd}\; f\ p\ k \mapsto c_f\} : \forall\, \alpha\,.\, \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle}$$

Figure 5: Handler typing.

This means that, even though our focus on mitigating the type mismatch between scoped computation and continuation so far has been on forwarding *unknown* scoped effects, the same applies when handling *known* scoped effects, where computation c accounts for this discrepancy.

Forwarding Clause. Rule T-Fwd types forwarding clauses of the form $\textbf{fwd}\; f\ p\ k \mapsto c_f$. As the forwarding clause needs to be able to forward any scoped effect in E, it cannot make any assumptions about the specific operation $\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}$ to expect. Instead, it abstracts over the operation and treats all possibilities uniformly. This abstraction comes in two parts. Firstly, the function f abstracts over the possible scoped operation calls $\textbf{sc}\; \ell^{\mathsf{sc}}\ v$. Secondly, the type variable α abstracts over the possible argument types B_{sc} of the scoped computation, and type variable β over the scoped result type.

$$p : \alpha \to M\ \beta!\langle E\rangle \qquad\qquad k : \beta \to M\ A!\langle E\rangle$$

Notice the difference between the type of p and k and the arguments of function f: it expects as argument (p', k') a transformed version of the scoped computation and continuation so that they agree on the intermediate type. An intuitive solution would be to transform the scoped computation and continuation as follows to agree on type $M\ \beta$.

$$p' : \alpha \to M\ \beta ! \langle E \rangle \qquad\qquad k' : M\ \beta \to M\ A ! \langle E \rangle$$

However, in some situation we require a more general type γ. For example, when our type constructor M deals with a state, we require uncurrying of the result type to apply k (e.g., h_{inc}, h_{state}, h_{depth}). A similar reasoning goes for the result type of k' and f. Consequently, the type of p' and k' are as follows:

$$p' : \alpha \to \gamma ! \langle E \rangle \qquad\qquad k' : \gamma \to \delta ! \langle E \rangle$$

Transforming the original scoped computation and continuation to agree on this type is the exact purpose of our explicit forwarding clauses.

Handler. Rule T-Handler types handlers with a polymorphic type of the form $\forall\ \alpha \,.\, \alpha\ !\ \langle F \rangle \ \Rightarrow\ M\ \ \alpha\ !\ \langle E \rangle$. A handler consists of a **return**-clause (T-Return), zero or more operation clauses (T-Empty, T-OprOp and T-OprSc), and a forwarding clause (T-Fwd). All clauses should agree on their result type $M\ \alpha\ !\ \langle E \rangle$. Notice that E denotes a collection with at least the labels of the present algebraic and scoped operation clauses in the handler (computed by the *labels*-function).

6.4. **Syntax-directed version of λ_{sc}.** Appendix C contains a syntax-directed version of λ_{sc}, which we prove type safe, and serves as the specification of our type inference algorithm. The syntax-directed version was obtained by the following three transformations.

First we removed rule T-EqV, which re-types expressions to some equivalent type. This rule is used to make types line up exactly at the site of applications, for example by changing the order of the labels in effect rows. As a consequence, in the syntax directe version we essentially inline T-EqV wherever it is needed.

Secondly, we removed the rules dealing with generalisation and instantiation (T-Inst, T-InstEff, T-Gen and T-GenEff). Instead, whenever rules insist on some kind of polymorphism on some subderivation, we extend the environment with fresh type variables, and generlize over them locally, instead of via axillary rules.

Finally, as dealing with higher-kinded polymorphism is orthogonal to our work (and real programming languages like Haskell and OCaml already have their solutions for higher-kinded polymorphism [YW14]), we avoid higher-order unification by annotating handlers with the type operator they apply (e.g. $\mathbf{handler}_M\ \{\ldots\}$ instead of $\mathbf{handler}\ \{\ldots\}$, avoiding higher-order unification, which is undecidable).

6.5. **Metatheory.** The type-and-effect system of λ_{sc} is type safe. In this section we briefly state the theorems to show this; the proofs and used lemmas can be found in Appendix E. We prove type safety by proving Subject Reduction and Progress. As values are inert , these theorems range over computations only. The formulation of Subject Reduction is standard:

Theorem 6.1 (Subject Reduction)**.** *If $\Gamma \vdash c : \underline{C}$ and $c \rightsquigarrow c'$, then there exists a $\underline{C}'$ such that $\underline{C}\ \equiv\ \underline{C}'$ and $\Gamma \vdash c' : \underline{C}'$.*

Apart from an additional normal form $\mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2)$, progress is standard as well:

Theorem 6.2 (Progress)**.** *If $\cdot \vdash c : \underline{C}$, then either:*

- *there exists a computation c' such that $c \rightsquigarrow c'$, or*

- *c is in a* normal form*, which means it is in one of the following forms: (1) $c = \mathbf{return}\ v$, (2) $c = \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c')$, or (3) $c = \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2)$.*

6.6. **Type Inference.** Appendix F contains an inference algorithm for λ_{sc}, based on the approach of Hindley-Milner [Mil78] and Koka [Lei14]. Here, we extend the various typing judgments with a derived substitution θ. For rules with multiple recursive clauses, unifications made during later branches of the inference algorithm are reflected in the result of earlier branches by applying the resulting substitutions to any type derived before. We prove it sound and complete w.r.t. the syntax-directed version as described in Section 6.4.

7. Examples

Now that we have formalized the calculus we can cover some examples. This serves two purposes. First, we will highlight how scoped effects as handlers, the solution proposed by Plotkin and Power [PP03] is problematic, even though it is applied in the real world [TRWS22]. We have postponed doing so, because now that we have formally introduced a calculus, we can immediately show how λ_{sc} addresses these issues. The first two examples in this section (exceptions with catch and reader with local) therefore contain both an attempt at encoding them as an handler, as well as a proper encoding as a **sc** in λ_{sc}. Secondly, the examples exemplify the expressivity of λ_{sc}.

To enhance readability, we write the examples in a higher-level syntax following Eff's conventions: we use top-level definitions, coalesce values and computations, implicitly sequence steps and insert **return** where needed. Furthermore, we drop trivial **return** continuations of operations:

$$\begin{aligned} \mathbf{op}\ \ell^{\mathsf{op}}\ x &\equiv \mathbf{op}\ \ell^{\mathsf{op}}\ x\ (y\,.\,\mathbf{return}\ y) \\ \mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ (y\,.\,c_1) &\equiv \mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ (y\,.\,c_1)\ (z\,.\,\mathbf{return}\ z) \end{aligned}$$

7.1. **Exceptions.** Wu et al. [WSH14] have shown how to catch exceptions with a scoped operation. Raising an exception is an algebraic operation `raise` : $\mathsf{String} \twoheadrightarrow \mathsf{Empty}$, and catching an exception is a scoped operation `catch` : $\mathsf{String} \twoheadrightarrow \mathsf{Bool}$. For example, consider that we are dealing with a counter with a maximum value of 10. The following computation increases the counter by 1 and raises an exception when the counter exceeds 10:

$$\begin{aligned} &\mathit{incr} = \mathbf{do}\ x \leftarrow \mathbf{op}\ \mathtt{inc}\ ()\,; \\ &\qquad\quad \mathbf{if}\ x > 10\ \mathbf{then}\ \mathbf{op}\ \mathtt{raise}\ \mathtt{"Overflow"}\ (y\,.\,\mathsf{absurd}\ y)\ \mathbf{else}\ \mathbf{return}\ x \end{aligned}$$

Clearly, if we start with a state of 8 and call `inc` thrice, we end up with an exception. We want to define a `catch` operation that executes an alternative computation should an exception be thrown.

7.1.1. *Catch as handler.* One might attempt to write catch as a handler. However, as we will see, this method does not have the same modularity and expressivity as our calculus because it cannot achieve the local update semantics [WSH14].

$$
\begin{aligned}
&h_{\mathsf{except}\,\text{✗}} = \mathbf{handler}\ \{\mathbf{return}\ x \mapsto \mathsf{right}\ x, \mathbf{op}\ \mathtt{raise}\ e\ \_ \mapsto \mathsf{left}\ e\}\\
&\mathtt{catch}_{\text{✗}}\ c_1\ c_2 = \mathbf{handler}\ \{\mathbf{return}\ x \qquad \mapsto \mathbf{return}\ x\\
&\qquad\qquad\qquad\qquad\qquad ,\mathbf{op}\ \mathtt{raise}\ \_\ \_ \mapsto c_2 \qquad\qquad \} \star c_1\\
&c_{\mathsf{catch}\,\text{✗}} = \mathbf{do}\ \mathit{incr}\,; \mathtt{catch}_{\text{✗}}\ (\mathbf{do}\ \mathit{incr}\,; \mathbf{do}\ \mathit{incr}\,; \mathbf{return}\ \texttt{"success"})\\
&\qquad\qquad\qquad\qquad\qquad (\mathbf{return}\ \texttt{"fail"})
\end{aligned}
$$

By handling exceptions before state we obtain global update semantics:

$$(\mathit{run}_{\mathsf{inc}}\ 8\ (h_{\mathsf{except}\,\text{✗}} \star c_{\mathsf{catch}\,\text{✗}}) \leadsto^{*} (\mathsf{right}\ (), 11)$$

However, when handling exceptions *after* state, we would expect *local* update semantics, i.e. $\mathsf{right}\ (\texttt{"fail"}, 9)$. However, we again get the global update semantics:

$$h_{\mathsf{except}\,\text{✗}} \star (\mathit{run}_{\mathsf{inc}}\ 8\ c_{\mathsf{catch}\,\text{✗}}) \leadsto^{*} (\mathsf{right}\ (), 11)$$

How can this be? By implementing $\mathtt{catch}_{\text{✗}}$ as a handler, we have lost the separation between syntax and semantics: $\mathtt{catch}_{\text{✗}}$ is supposed to denote syntax, but it contains semantics in the form of a handler. Since we apply $\mathtt{catch}_{\text{✗}}$ to a computation ($c_{\mathsf{catch}\,\text{✗}}$), any containing `raise` will have already been handled by $\mathtt{catch}_{\text{✗}}$ before h_{inc} is applied. In other words, we have lost modular composition, and therefore control over the interactions of effects.

7.1.2. *Catch as scoped effect.* Let us define c_{catch} that defines `catch` as an **sc**:

$$
\begin{aligned}
&c_{\mathsf{catch}} = \mathbf{do}\ \mathit{incr}\,; \mathbf{sc}\ \mathtt{catch}\ \texttt{"Overflow"}\ (b\,.\\
&\qquad\quad \mathbf{if}\ b\ \mathbf{then}\ (\mathbf{do}\ \mathit{incr}\,; \mathbf{do}\ \mathit{incr}\,; \mathbf{return}\ \texttt{"success"})\ \mathbf{else}\ \mathbf{return}\ \texttt{"fail"})
\end{aligned}
$$

The scoped computation's true branch is the program that may *raise* exceptions, while the false branch *deals with* the exception. Our handler interprets exceptions in terms of a sum type $\mathbf{data}\ \alpha + \beta = \mathsf{left}\ \alpha \mid \mathsf{right}\ \beta$, where $\mathsf{left}\ v$ denotes an exception and $\mathsf{right}\ v$ a result.

$$
\begin{aligned}
&h_{\mathsf{except}}\ :\ \forall\, \alpha\ \mu\,.\, \alpha\,!\,\langle \mathtt{raise}\,; \mathtt{catch}\,; \mu\rangle \Rightarrow \mathsf{String} + \alpha\,!\,\langle\mu\rangle\\
&h_{\mathsf{except}} = \mathbf{handler}\\
&\quad \{\mathbf{return}\ x \qquad\quad \mapsto \mathsf{right}\ x\\
&\quad ,\mathbf{op}\ \mathtt{raise}\ e\ \_ \quad \mapsto \mathsf{left}\ e\\
&\quad ,\mathbf{sc}\ \mathtt{catch}\ e\ p\ k \mapsto \mathbf{do}\ x \leftarrow p\ \mathsf{true}\,;\\
&\qquad\qquad\qquad\qquad\quad \mathbf{case}\ x\ \mathbf{of}\ \mathsf{left}\ e' \mid e' = e \rightarrow \mathsf{exceptMap}\ (p\ \mathsf{false})\ k\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \_ \qquad\qquad\quad \rightarrow \mathsf{exceptMap}\ x\ k\\
&\quad ,\mathbf{fwd}\ f\ p\ k \qquad\ \ \mapsto f\ (p, \boldsymbol{\lambda} z\,.\, \mathsf{exceptMap}\ z\ k)\}
\end{aligned}
$$

The return clause and algebraic operation clause for `raise` construct a return value and raise an exception e by calling the right and left constructors, respectively. The scoped operation clause for `catch` catches an exception e. If the scoped computation in $p\ \mathsf{true}$ raises an exception e, it is caught by `catch` and replaced by the scoped computation ($p\ \mathsf{false}$). Otherwise, it continues with $p\ \mathsf{true}$ and its results are passed to the continuation k. For forwarding we essentially return the exception if z fails ($\mathsf{left}\ e$), and we apply the continuation k to z if z succeeds ($\mathsf{right}\ x$).

$$\begin{aligned}&\mathsf{exceptMap} : \forall\, \alpha\, \beta\, \mu\, .\, \mathsf{String} + \beta \to^{\mu} (\beta \to^{\mu} \mathsf{String} + \alpha) \to^{\mu} \mathsf{String} + \alpha \\ &\mathsf{exceptMap}\ z\ k = \mathbf{case}\ z\ \mathbf{of}\ \mathsf{left}\ e \quad \to \mathsf{left}\ e \\ &\qquad\qquad\qquad\qquad\qquad\quad \mathsf{right}\ x \to k\ x\end{aligned}$$

Given an intial counter value 8, we can handle the program c_{catch} with h_{except} and h_{inc}. Different orders of the application of handlers give us different semantics of the interaction of effects [WSH14]. Handling exceptions before increments yields us global updates:

$$run_{\mathsf{inc}}\ 8\ (h_{\mathsf{except}} \star c_{\mathsf{catch}}) \leadsto^{*} (\mathsf{right}\ (), 11)$$

Although an exception is raised and caught, the final value is still updated to 11 by the two `inc` operations and exceeds the maximum value of our counter. When handling exceptions after increments, we obtain the local update semantics:

$$h_{\mathsf{except}} \star (run_{\mathsf{inc}}\ 9\ c_{\mathsf{catch}}) \leadsto^{*} \mathsf{right}\ ((), 9)$$

7.2. **Reader with Local.** Reader entails an `ask` operation that lets one read the (integer) state that is passed around. The scoped effect `local` takes a function f which alters the state, and a computation for which the state should be altered, after which the state should be returned to its original state. For example, in `local` $(\boldsymbol{\lambda} i\, .\, i * 2)$ $(\mathbf{op}\ \mathtt{ask}\ ())$ $(\mathbf{op}\ \mathtt{ask}\ ())$, the first ask receives a state that is doubled, whereas the second ask receives the original state. To exemplify the problems that arise when implementing `local` as a handler, our example uses effect `foo`, which is simply mapped to `ask` by h_{foo}:

$$\begin{aligned}h_{\mathsf{foo}} = \mathbf{handler}\ \{&\mathbf{return}\ x \mapsto \mathbf{return}\ x \\ ,&\mathbf{op}\ \mathtt{foo}\ \_ \ \mapsto \mathtt{ask}\}\end{aligned}$$

7.2.1. *Local as a handler.* Whereas the lack of effect interaction control in example of `catch` as a handler could be described as unfortunate, in the case for ask there is arguably only one correct interaction, which is not the one that arises from scoped effects as handlers. Consider c_{local} below, which includes `foo`, which is mapped to `ask` by h_{foo}.

$$\begin{aligned}&\mathtt{local}_{\text{✗}}\ f\ c = \mathbf{handler}\ \{\mathbf{return}\ x \mapsto \mathbf{return}\ x \\ &\qquad\qquad\qquad\qquad\quad ,\mathbf{op}\ \mathtt{ask}\ \_ \ \mapsto x \leftarrow \mathtt{ask}\, ; \mathbf{return}\ f\ x\} \\ &h_{\mathsf{read}\text{✗}} = \mathbf{handler}\ \{\mathbf{return}\ x \mapsto \boldsymbol{\lambda} m\, .\, (x, m), \mathbf{op}\ \mathtt{ask}\ \_\ k \mapsto \boldsymbol{\lambda} m\, .\, k\ m\ m\} \\ &run_{\mathsf{read}\text{✗}}\ s\ c \ \equiv\ \mathbf{do}\ c' \leftarrow h_{\mathsf{read}\text{✗}} \star c\, ; c'\ s \\ &c_{\mathsf{local}\text{✗}} = x \leftarrow \mathtt{ask}\, ; y \leftarrow \mathtt{foo} \\ &\qquad\quad \mathtt{local}_{\text{✗}}\ (\boldsymbol{\lambda} a \to 2 * a)\ (z \leftarrow \mathtt{ask}\, ; u \leftarrow \mathtt{foo}\, ; \mathbf{return}\ (x, y, z, u))\end{aligned}$$

Since h_{foo} introduces `ask`, we must (re)apply h_{read} after applying h_{foo}. Since `foo` is mapped to `ask`, in $c_{\mathsf{local}\text{✗}}$ we expect x to be equal to y, and z equal to u. Starting with the reader state set to 1, we expect the result $(1, 1, 2, 2)$. Instead, we get:

$$run_{\mathsf{read}}\ 1 \star (h_{\mathsf{foo}} \star c_{\mathsf{local}\text{✗}}) \leadsto^{*} (1, 1, 2, 1)\, .$$

Again, how can this be? The cause is the same as the example with `catch`: since we encode the semantics of $\mathtt{local}_{\text{✗}}$ in its definition, we are forced to perform the handling at the moment of application. Notice that `foo` is not caught by $\mathtt{local}_{\text{✗}}$! Therefore, f is only applied to the `ask`. When `foo` is mapped to `ask` by h_{foo}, $\mathtt{local}_{\text{✗}}$'s effect will already have triggered, which is why f is not applied to it.

7.2.2. *Local as scoped effect.* Using a scoped effect we can properly encode `local`:

$$\begin{array}{l}
h_{\mathsf{read}} \;:\; \forall\, \alpha\, \mu \,.\, \alpha\,!\,\langle \texttt{ask}\,;\texttt{local}\,;\mu\rangle \Rightarrow (\mathsf{Int} \rightarrow^{\mu} (\alpha, \mathsf{Int}))\,!\,\langle\mu\rangle \\
h_{\mathsf{read}} = \mathbf{handler}\;\{\mathbf{return}\; x \quad\;\; \mapsto \boldsymbol{\lambda} m\,.\,(x, m) \\
\qquad\qquad\qquad\;\; ,\mathbf{op}\;\texttt{ask}\;\_\;k \quad\;\, \mapsto \boldsymbol{\lambda} m\,.\,k\;m\;m \\
\qquad\qquad\qquad\;\; ,\mathbf{sc}\;\texttt{local}\;f\;p\;k \mapsto \boldsymbol{\lambda} m\,.\,\mathbf{do}\;(x, \_) \leftarrow p\;()\;(f\;m)\,;k\;x\;m \\
\qquad\qquad\qquad\;\; ,\mathbf{fwd}\;f\;p\;k \quad\;\; \mapsto \boldsymbol{\lambda} m\,.\,f\;(\boldsymbol{\lambda} y\,.\,p\;y\;m, \boldsymbol{\lambda}(z, m')\,.\,k\;z\;m')\} \\
\mathit{run}_{\mathsf{read}}\;s\;c \;\equiv\; \mathbf{do}\;c' \leftarrow h_{\mathsf{read}} \star c\,;c'\;s \\
c_{\mathsf{local}} = x \leftarrow \texttt{ask}\,; y \leftarrow \texttt{foo} \\
\qquad\quad \texttt{local}\;(\boldsymbol{\lambda} a \rightarrow 2 * a)\;(z \leftarrow \texttt{ask}\,; u \leftarrow \texttt{foo}\,;\mathbf{return}\;(x, y, z, u))
\end{array}$$

Note that the forwarding clause of h_{state} is the same as the forwarding clause of h_{inc} in Section 5. Since `local` is now purely syntax, we can apply h_{foo} before h_{read}, and have h_{read} handle the `ask` that h_{foo} outputs:

$$\begin{array}{l}
\quad\; \mathit{run}_{\mathsf{read}}\;1\;(h_{\mathsf{foo}} \star c_{\mathsf{local}}) \\
\rightsquigarrow \mathit{run}_{\mathsf{read}}\;1\;(x \leftarrow \texttt{ask}\,; y \leftarrow \\
\qquad\qquad\quad \texttt{local}\;(\boldsymbol{\lambda} a \rightarrow 2 * a)\;(z \leftarrow \texttt{ask}\,; u \leftarrow \;;\mathbf{return}\;(x, y, z, u))
\end{array}$$

7.3. **Nondeterminism with Cut.** The algebraic operation $\texttt{cut} : () \twoheadrightarrow ()$ provides a different flavor of pruning nondeterminism that has its origin as a Prolog primitive. The idea is that `cut` prunes all remaining branches and only allows the current branch to continue. Typically, we want to keep the effect of `cut` local. This is achieved with the scoped operation $\texttt{call} : () \twoheadrightarrow ()$, as proposed by Wu et al. [WSH14]. To handle `cut` and `call`, we use the $\mathsf{CutList}$ datatype [PS17].

$$\mathbf{data}\;\mathsf{CutList}\;\alpha = \mathsf{opened}\;(\mathsf{List}\;\alpha)\;\;|\;\;\mathsf{closed}\;(\mathsf{List}\;\alpha)$$

We can think of $\mathsf{opened}\;v$ as a list that may be extended and $\mathsf{closed}\;v$ as a list that may not be extended with further elements. This intention is captured in the $\mathsf{append}_{\mathsf{CutList}}$ function, which discards the second list if the constructor of the first list is closed.

$$\begin{array}{l}
\mathsf{append}_{\mathsf{CutList}} : \forall\, \alpha\, \mu\,.\,\mathsf{CutList}\;\alpha \rightarrow^{\mu} \mathsf{CutList}\;\alpha \rightarrow^{\mu} \mathsf{CutList}\;\alpha \\
\mathsf{append}_{\mathsf{CutList}}\;(\mathsf{opened}\;\mathit{xs})\;(\mathsf{opened}\;\mathit{ys}) = \mathsf{opened}\;(\mathit{xs} +\!\!+ \mathit{ys}) \\
\mathsf{append}_{\mathsf{CutList}}\;(\mathsf{opened}\;\mathit{xs})\;(\mathsf{closed}\;\mathit{ys})\;\; = \mathsf{closed}\;\;\,(\mathit{xs} +\!\!+ \mathit{ys}) \\
\mathsf{append}_{\mathsf{CutList}}\;(\mathsf{closed}\;\mathit{xs})\;\;\_ \qquad\qquad\; = \mathsf{closed}\;\;\,\mathit{xs}
\end{array}$$

The handler for nondeterminism with cut is defined as follows:

$$\begin{array}{l}
h_{\mathsf{cut}} \;:\; \forall\, \alpha\, \mu\,.\,\alpha\,!\,\langle \texttt{choose}\,;\texttt{fail}\,;\texttt{cut}\,;\texttt{call}\,;\mu\rangle \Rightarrow \mathsf{CutList}\;\alpha!\langle\mu\rangle \\
h_{\mathsf{cut}} = \mathbf{handler}\;\{\mathbf{return}\;x \qquad\;\; \mapsto \mathsf{opened}\;[x] \\
\qquad\qquad\quad\;\; ,\mathbf{op}\;\texttt{fail}\;\_\;\_ \quad\;\; \mapsto \mathsf{opened}\;[\,] \\
\qquad\qquad\quad\;\; ,\mathbf{op}\;\texttt{choose}\;x\;k \mapsto \mathsf{append}_{\mathsf{CutList}}\;(k\;\mathsf{true})\;(k\;\mathsf{false}) \\
\qquad\qquad\quad\;\; ,\mathbf{op}\;\texttt{cut}\;\_\;k \quad\;\; \mapsto \mathsf{close}\;(k\;()) \\
\qquad\qquad\quad\;\; ,\mathbf{sc}\;\texttt{call}\;\_\;p\;k \;\; \mapsto \mathsf{concatMap}_{\mathsf{CutList}}\;(\mathsf{open}\;(p\;()))\;k \\
\qquad\qquad\quad\;\; ,\mathbf{fwd}\;f\;p\;k \qquad \mapsto f\;(p, \boldsymbol{\lambda} z\,.\,\mathsf{concatMap}_{\mathsf{CutList}}\;z\;k)\}
\end{array}$$

The operation clause for `cut` closes the cutlist and the clause for `call` (re-)opens it when coming out of the scope.

$$\begin{array}{ll}
\mathsf{close} : \forall\ \alpha\ \mu\,.\,\mathsf{CutList}\ \alpha \to^{\mu} \mathsf{CutList}\ \alpha & \mathsf{open} : \forall\ \alpha\ \mu\,.\,\mathsf{CutList}\ \alpha \to^{\mu} \mathsf{CutList}\ \alpha \\
\mathsf{close}\ (\mathsf{closed}\ as) \ = \mathsf{closed}\ as & \mathsf{open}\ (\mathsf{closed}\ as) \ = \mathsf{opened}\ as \\
\mathsf{close}\ (\mathsf{opened}\ as) = \mathsf{closed}\ as & \mathsf{open}\ (\mathsf{opened}\ as) = \mathsf{opened}\ as
\end{array}$$

The function $\mathsf{concatMap}_{\mathsf{CutList}}$ is the cutlist counterpart of $\mathsf{concatMap}$ which takes the extensibility of $\mathsf{CutList}$ (signalled by opened and closed) into account when concatenating.

$$\begin{array}{l}
\mathsf{concatMap}_{\mathsf{CutList}} : \forall\ \alpha\ \beta\ \mu\,.\,\mathsf{CutList}\ \beta \to^{\mu} (\beta \to^{\mu} \mathsf{CutList}\ \alpha) \to^{\mu} \mathsf{CutList}\ \alpha \\
\mathsf{concatMap}_{\mathsf{CutList}}\ (\mathsf{opened}\ [\,])\qquad f = \mathbf{return}\ (\mathsf{opened}\ [\,]) \\
\mathsf{concatMap}_{\mathsf{CutList}}\ (\mathsf{closed}\ [\,])\qquad f = \mathbf{return}\ (\mathsf{closed}\ [\,]) \\
\mathsf{concatMap}_{\mathsf{CutList}}\ (\mathsf{opened}\ (b : bs))\ f = \mathbf{do}\ as \leftarrow f\ b\,; \\
\qquad\qquad\qquad as' \leftarrow \mathsf{concatMap}_{\mathsf{CutList}}\ (\mathsf{opened}\ bs)\ f\,; \\
\qquad\qquad\qquad \mathsf{append}_{\mathsf{CutList}}\ as\ as' \\
\mathsf{concatMap}_{\mathsf{CutList}}\ (\mathsf{closed}\ (b : bs))\ \ f = \mathbf{do}\ as \leftarrow f\ b\,; \\
\qquad\qquad\qquad as' \leftarrow \mathsf{concatMap}_{\mathsf{CutList}}\ (\mathsf{closed}\ bs)\ f\,; \\
\qquad\qquad\qquad \mathsf{append}_{\mathsf{CutList}}\ as\ as'
\end{array}$$

In Section 7.5, we give an example usage of `cut` to improve parsers.

7.4. **Depth-Bounded Search.** The handlers for nondeterminism shown in Section 7 implement the *depth-first search* (DFS) strategy. However, with scoped effects and handlers we can implement other search strategies, such as *depth-bounded search* (DBS) [YPW+22], which uses the scoped operation $\texttt{depth} : \mathsf{Int} \twoheadrightarrow ()$ to bound the depth of the branches in the scoped computation. The handler uses return type $\mathsf{Int} \to^{\mu} \mathsf{List}\ (\alpha, \mathsf{Int})$. Here, the Int parameter is the current depth bound, and the result is a list of (α, Int) pairs, where α denotes the result and Int reflects the remaining global depth bound.[2]

$$\begin{array}{l}
h_{\mathsf{depth}}\ : \forall\ \alpha\ \mu\,.\,\alpha\,!\,\langle \texttt{choose}\,; \texttt{fail}\,; \texttt{depth}\,; \mu\rangle \Rightarrow (\mathsf{Int} \to^{\mu} \mathsf{List}\ (\alpha, \mathsf{Int}))\,!\,\langle\mu\rangle \\
h_{\mathsf{depth}} = \mathbf{handler} \\
\quad \{\mathbf{return}\ x \qquad\ \mapsto \boldsymbol{\lambda} d\,.\,[(x, d)] \\
\quad ,\mathbf{op}\ \texttt{fail}\ \_\ \_ \qquad \mapsto \boldsymbol{\lambda}\_\,.\,[\,] \\
\quad ,\mathbf{op}\ \texttt{choose}\ x\ k\ \ \mapsto \boldsymbol{\lambda} d\,.\,\mathbf{if}\ d \equiv 0\ \mathbf{then}\ [\,]\ \mathbf{else}\ k\ \mathsf{true}\ (d-1) +\!\!+ k\ \mathsf{false}\ (d-1) \\
\quad ,\mathbf{sc}\ \texttt{depth}\ d'\ p\ k \mapsto \boldsymbol{\lambda} d\,.\,\mathsf{concatMap}\ (p\ ()\ d')\ (\boldsymbol{\lambda}(v, \_)\,.\,k\ v\ d) \\
\quad ,\mathbf{fwd}\ f\ p\ k \qquad\ \mapsto \boldsymbol{\lambda} d\,.\,f\ (\boldsymbol{\lambda} y\,.\,p\ y\ d, \boldsymbol{\lambda} vs\,.\,\mathsf{concatMap}\ vs\ (\boldsymbol{\lambda}(v, d)\,.\,k\ v\ d))\}
\end{array}$$

For the `depth` operation, we locally use the given depth bound d' for the scoped computation p and go back to using the global depth bound d for the continuation k. In case of an unknown scoped operation, the forwarding clause just threads the depth bound through, first into the scoped computation and from there into the continuation. For example, the following program (Figure 6) has a local depth bound of 1 and a global depth bound of 2. It discards the results 2 and 3 in the scoped computation as they appear after the second `choose` operation, and similarly, the results 5 and 6 in the continuation are ignored.

$$\begin{array}{l}
c_{\mathsf{depth}} = \mathbf{sc}\ \texttt{depth}\ 1 \\
\qquad\qquad (\_\,.\,\mathbf{do}\ b_1 \leftarrow \mathbf{op}\ \texttt{choose}\ ()\,; \mathbf{if}\ b_1\ \mathbf{then}\ \mathbf{return}\ 1\ \mathbf{else}
\end{array}$$

[2]These pairs (α, Int) differ from Yang et al.'s [YPW+22]'s α in order to make the forwarding clause work.

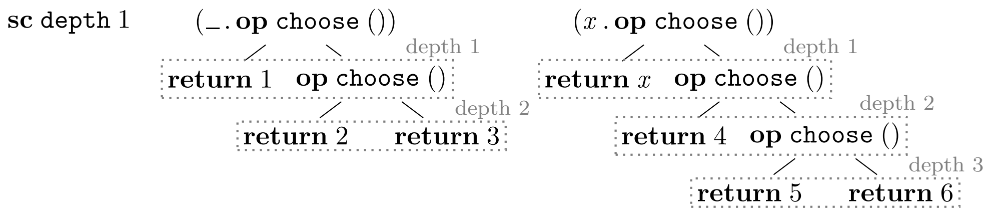

Figure 6: Visual representation of c_{depth}.

$$
\begin{aligned}
&\quad\ \ \mathbf{do}\ b_2 \leftarrow \mathbf{op}\ \texttt{choose}\ ()\,;\mathbf{if}\ b_2\ \mathbf{then\ return}\ 2\ \mathbf{else\ return}\ 3) \\
&(x\,.\,\mathbf{do}\ b_1 \leftarrow \mathbf{op}\ \texttt{choose}\ ()\,;\mathbf{if}\ b_1\ \mathbf{then\ return}\ x\ \mathbf{else} \\
&\quad\ \ \mathbf{do}\ b_2 \leftarrow \mathbf{op}\ \texttt{choose}\ ()\,;\mathbf{if}\ b_2\ \mathbf{then\ return}\ 4\ \mathbf{else} \\
&\quad\ \ \mathbf{do}\ b_3 \leftarrow \mathbf{op}\ \texttt{choose}\ ()\,;\mathbf{if}\ b_3\ \mathbf{then\ return}\ 5\ \mathbf{else\ return}\ 6)
\end{aligned}
$$

$\texttt{>>>}\ (h_{\mathsf{depth}} \star c_{\mathsf{depth}})\ 2$
$[(1,1),(4,0)]$

The result is $[(1,1),(4,0)]$, where the tuple's second parameter represents the global depth bound. Notice that `choose` operations in the scoped computation `depth` do not consume the global depth bound in the handler. For a different implementation, we refer to the Supplementary Material.

7.5. **Parsers.** A parser effect can be achieved by combining the nondeterminism-with-cut effect and a token-consuming effect [WSH14]. The latter features the algebraic operation $\texttt{token} : \mathsf{Char} \twoheadrightarrow \mathsf{Char}$ where $\mathbf{op}\ \texttt{token}\ t$ consumes a single character from the implicit input string; if it is t, it is passed on to the continuation; otherwise the operation fails. The token handler has result type $\mathsf{String} \rightarrow^{\langle \texttt{fail}\,;\mu\rangle} (\alpha, \mathsf{String})$: it threads through the remaining part of the input string. Observe that the function type signals it may `fail`, in case the token does not match.

$$
\begin{aligned}
&h_{\mathsf{token}}\ :\ \forall\,\alpha\,\mu\,.\,\alpha\,!\,\langle \texttt{token}\,;\texttt{fail}\,;\mu\rangle \Rightarrow (\mathsf{String} \rightarrow^{\langle \texttt{fail}\,;\mu\rangle} (\alpha, \mathsf{String}))\,!\,\langle \texttt{fail}\,;\mu\rangle \\
&h_{\mathsf{token}} = \mathbf{handler} \\
&\quad \{\mathbf{return}\ x \qquad\ \ \mapsto \boldsymbol{\lambda} s\,.\,(x, s) \\
&\quad ,\mathbf{op}\ \texttt{token}\ x\ k \mapsto \boldsymbol{\lambda} s\,.\,\mathbf{case}\ s\ \mathbf{of}\ [\,] \qquad\quad \rightarrow \mathsf{failure}\ () \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad (x' : xs) \rightarrow \mathbf{if}\ x \equiv x'\ \mathbf{then}\ k\ x\ xs\ \mathbf{else}\ \mathsf{failure}\ () \\
&\quad ,\mathbf{fwd}\ f\ p\ k \qquad \mapsto \boldsymbol{\lambda} s\,.\,f\ (\boldsymbol{\lambda} y\,.\,p\ y\ s, \boldsymbol{\lambda}(t, s)\,.\,k\ t\ s)\}
\end{aligned}
$$

We give an example parser for a small expression language, in the typical parser combinator style, built on top of the token-consumer and nondeterminism. For convenience, it uses the syntactic sugar $x \diamond y\ \equiv\ \mathbf{op}\ \texttt{choose}\ (b\,.\,\mathbf{if}\ b\ \mathbf{then}\ x\ \mathbf{else}\ y)$.

$$
\begin{aligned}
&\mathsf{digit} \quad\ \ : \forall\,\mu\,.\,() \rightarrow \mathsf{Char}\,!\,\langle \texttt{token}\,;\texttt{choose}\,;\mu\rangle \\
&\mathsf{digit}\ \_ \ \ = \mathbf{op}\ \texttt{token}\ \texttt{'0'} \diamond \mathbf{op}\ \texttt{token}\ \texttt{'1'} \diamond \ldots \diamond \mathbf{op}\ \texttt{token}\ \texttt{'9'} \\
&\mathsf{many}_1 \quad : \forall\,\alpha\,\mu\,.\,(() \rightarrow^{\mu} \alpha) \rightarrow^{\mu} \mathsf{List}\ \alpha \\
&\mathsf{many}_1\ p = \mathbf{do}\ a \leftarrow p\ ()\,;\mathbf{do}\ as \leftarrow \mathsf{many}_1\ p \diamond \mathbf{return}\ [\,]\,;\mathbf{return}\ (a : as) \\
&\mathsf{expr}' \quad\ \ : \forall\,\mu\,.\,() \rightarrow \mathsf{Int}\,!\,\langle \texttt{token}\,;\texttt{choose}\,;\mu\rangle \\
&\mathsf{expr}'\ \_ \ \ = \mathbf{do}\ i \leftarrow \mathsf{term}\ ()\,;\mathbf{do\ op}\ \texttt{token}\ \texttt{'+'}\,;\mathbf{do}\ j \leftarrow \mathsf{expr}'\ ()\,; \\
&\qquad\qquad\ \ \mathbf{return}\ (i + j) \diamond \mathbf{do}\ i \leftarrow \mathsf{term}\ ()\,;\mathbf{return}\ i
\end{aligned}
$$

```
term     : ∀ μ . () → Int ! ⟨token ; choose ; μ⟩
term _   = do i ← factor () ; do op token '*' ; do j ← term () ;
             return (i * j) ◇ do i ← factor () ; return i
factor   : ∀ μ . () → Int ! ⟨token ; choose ; μ⟩
factor _ =   do ds ← many₁ digit ; return (read ds)
           ◇ do op token '(' ; do i ← expr′ () ; do op token ')' ; return i
```

The expr' parser is naive and can be improved by two types of refactoring: (1) factoring out the common prefix in the two branches, and (2) pruning the second branch when the first branch successfully consumes a $+$.

```
expr   : ∀ μ . () → Int ! ⟨token ; choose ; cut ; μ⟩
expr _ = do i ← term () ;
               sc call () (_ . (do op token '+' ; op cut () ;
                                      j ← expr () ; return (i + j)) ◇ i)
```

Here is how we invoke the parser on an example input.

```
>>> h_cut ⋆ (h_token ⋆ expr ()) "(2+5)*8"
opened [(56, ""), (7, "*8")]
```

There are two results in the cutlist. Usually we are only interested in the full parsers, i.e., those that have consumed the entire input string.

8. Related Work

In this section, we discuss related work on algebraic effects, scoped effects, and effect systems.

8.1. **Algebraic Effects & Handlers.**

Many research languages for algebraic effects have been proposed, including Eff [BP13, Pre15], Frank [LMM17], Effekt [BSO20], or have been extended to include them, such as Links [HL16], Koka [Lei17], and Multicore OCaml [SDW$^+$21].

There are also many packages for writing effect handlers in general purpose languages [KSSF19, KS18, RTWS18, Mag19, Kin19]. Yet, as far as we know, λ_{sc} is the first *calculus* that supports scoped effects & handlers.

In contrast with this line of work on algebraic effects, Nanevski [Nan05] provides an alternative view of exceptions based on comonads that characterizes monadic effects as "persistent".

8.2. **Effect Systems.**

Most languages with support for algebraic effects are equipped with an effect system to keep track of the effects that are used in the programs. There is already much work on different approaches to effect systems for algebraic effects.

Eff [BP13, Pre15] uses an effect system based on subtyping relations. Each type of computation is decorated with an effect type Δ to represent the set of operations that might be invoked. The subtyping relations are used to extend the effect type Δ with other effects, which makes it possible to compose programs in a modular way. We did not choose to use the subtyping-based effect types in λ_{sc} as that would require complex subtyping for type operators.

Row polymorphism is another mainstream approach to effect systems. Links [HL16] uses the Rémy style row polymorphism [Rém94], where the row types are able to represent the absence of labels and each label is restricted to appear at most once. Koka [Lei17] uses row polymorphism based on scoped labels [Lei05], which allows duplicated labels and as a result is easier to implement. We can use row polymorphism to write handlers that handle particular effects and forward other effects represented by a row variable. In λ_{sc}, we opted for an effect system similar to Koka's, mainly because of its brevity. We believe that the Links-style effect system should also work well with scoped effects.

8.3. **Scoped Effects & Handlers.** Wu et al. [WSH14] first introduced the idea of scoped effects & handlers to solve the problem of separating syntax from semantics in programming with effects that delimit the scope. They proposed a higher-order syntax, an approach to scoped effects & handlers that has already been implemented in several Haskell packages [RTWS18, Mag19, Kin19]. They use higher-order signatures, which impose less restrictions on the shape of the signatures of scoped operations and allow programmers to delimit the scopes in a freer way than λ_{sc}. The cost of this freedom is the need for programmers to write more functions to distribute handlers for each signature. The higher-order signatures are also not suitable for use in a calculus as the signatures of operations are usually characterised by a pair of types in a calculus.

Pirog et al. [PSWJ18] and Yang et al. [YPW+22] have developed denotational semantic domains of scoped effects, backed by category theorical models. The key idea is to generalize the denotational approach of algebraic effects & handlers that is based on free monads and their unique homomorphisms. Indeed, the underlying category can be seen as a parameter. Then, by shifting from the base category of types and functions to a different (indexed or functor) category, scoped operations and their handlers turn out to be "just" an instance of the generalized notion of algebraic operations and handlers with the same structure and properties. We focus on a calculus for scoped effects instead of the denotational semantics of scoped effects. We make a simplification with respect to Yang et al. [YPW+22] where we avoid duplication of the base algebra and endoalgebra (for the outer and inner scoped respectively), and thus duplication of the scoped effect clauses in our handlers. With respect to Pirog et al. [PSWJ18], we specialize the generic endofunctor Γ with signatures $A_\ell \twoheadrightarrow B_\ell$ of endofunctors of the form $A \times (B \to -)$. Our λ_{sc} calculus uses a similar idea to the 'explicit substitution' monad of Pirog et al. [PSWJ18], a generalization of Ghani and Uustalu's [GU03] monad of explicit substitutions where each operation is associated with two computations representing the computation in scope and out of the scope (continuation) respectively. While the composition of scoped effects has not been considered in their categorical models, we introduced forwarding clauses for the composition, and further restrict handlers to be polymorphic to simplify handling and composing scoped effects.

9. Conclusion and Future Work

In this work, we have presented λ_{sc}, a novel calculus in which scoped effects & handlers are built-in. We have started from Eff, extended with row-typing in the style of Koka, and added scoped effect clauses and operations, polymorphic handlers, and explicit forwarding clauses. Finally, we have demonstrated the usability of λ_{sc} by implementing a range of examples. Although we have given many useful and compelling examples, we acknowledge that, just like algebraic effects, scoped effects are not encompassing (e.g. bracketing). We believe that the features to support scoped effect in λ_{sc} are orthogonal to other language features and can be added to any programming language with algebraic effects, polymorphism and type operators.

Scoped effects require *every* handler in λ_{sc} to be polymorphic and equipped with an explicit forwarding clause. This breaks backwards compatibility: calculi that support only algebraic effects, such as Eff, miss an explicit forwarding clause for scoped operations and allow monomorphic handlers. Actually this problem can be easily solved by kinds and kind polymorphism. The core idea is that we extend λ_{sc} with two kinds op and sc for effect types, such that $\Gamma \vdash E : \mathsf{op}$ means effect type E only contains algebraic operations, and $\Gamma \vdash E : \mathsf{sc}$ means effect type E may contain some scoped operations. Then, for handlers of type $A!\langle E\rangle \Rightarrow M\ A!\langle F\rangle$ which lack forwarding clauses, we can just add the condition $\Gamma \vdash E : \mathsf{op}$ to their typing rules. We have a prototype implementation of this idea, but we leave the full specification and extension of it to future work.

References

[BP13] Andrej Bauer and Matija Pretnar. An effect system for algebraic effects and handlers. In Reiko Heckel and Stefan Milius, editors, *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, volume 8089 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013. `doi:10.1007/978-3-642-40206-7\_1`.

[BP15] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming*, 84(1):108–123, 2015. Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011. URL: `https://www.sciencedirect.com/science/article/pii/S2352220814000194`, `doi:https://doi.org/10.1016/j.jlamp.2014.02.001`.

[BSO20] Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. Effects as capabilities: Effect handlers and lightweight effect polymorphism. *Proc. ACM Program. Lang.*, 4(OOPSLA), November 2020. `doi:10.1145/3428194`.

[GU03] Neil Ghani and Tarmo Uustalu. Explicit substitutions and higher-order syntax. In *Proceedings of the 2003 ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding*, MERLIN '03, page 1–7, New York, NY, USA, 2003. Association for Computing Machinery. `doi:10.1145/976571.976580`.

[HL16] Daniel Hillerström and Sam Lindley. Liberating effects with rows and handlers. In *Proceedings of the 1st International Workshop on Type-Driven Development*, TyDe 2016, page 15–27, New York, NY, USA, 2016. Association for Computing Machinery. `doi:10.1145/2976022.2976033`.

[Kin19] Alexis King. eff – screaming fast extensible effects for less, 2019. `https://github.com/hasura/eff`.

[KS18] Oleg Kiselyov and KC Sivaramakrishnan. Eff directly in ocaml. *Electronic Proceedings in Theoretical Computer Science*, 285:23–58, Dec 2018. URL: `http://dx.doi.org/10.4204/EPTCS.285.2`, `doi:10.4204/eptcs.285.2`.

[KSSF19] Oleg Kiselyov, Amr Sabry, Cameron Swords, and Ben Foppa. extensible-effects: An alternative to monad transformers, 2019. `https://hackage.haskell.org/package/extensible-effects`.

[Lei05] Daan Leijen. Extensible records with scoped labels. In Marko C. J. D. van Eekelen, editor, *Revised Selected Papers from the Sixth Symposium on Trends in Functional Programming, TFP 2005, Tallinn, Estonia, 23-24 September 2005*, volume 6 of *Trends in Functional Programming*, pages 179–194. Intellect, 2005.

[Lei14] Daan Leijen. Koka: Programming with row polymorphic effect types. *Electronic Proceedings in Theoretical Computer Science*, 153, 06 2014. `doi:10.4204/EPTCS.153.8`.

[Lei17] Daan Leijen. Type directed compilation of row-typed algebraic effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL

2017, page 486–499, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009872.

[LMM17] Sam Lindley, Conor McBride, and Craig McLaughlin. Do be do be do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 500–514, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009897.

[LPT03] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003. doi:10.1016/S0890-5401(03)00088-9.

[Mag19] Sandy Maguire. polysemy: Higher-order, low-boilerplate free monads, 2019. https://hackage.haskell.org/package/polysemy.

[Mil78] Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978. doi:10.1016/0022-0000(78)90014-4.

[Mog89] Eugenio Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Edinburgh University, Department of Computer Science, June 1989.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55 – 92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science. doi:https://doi.org/10.1016/0890-5401(91)90052-4.

[Nan05] Aleksandar Nanevski. A modal calculus for exception handling. 01 2005.

[PP03] Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Appl. Categorical Struct.*, 11(1):69–94, 2003. doi:10.1023/A:1023064908962.

[PP09] Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In Giuseppe Castagna, editor, *Programming Languages and Systems*, pages 80–94, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-00590-9_7.

[Pre15] Matija Pretnar. An introduction to algebraic effects and handlers. invited tutorial paper. *Electronic Notes in Theoretical Computer Science*, 319:19–35, 2015. The 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI). URL: https://www.sciencedirect.com/science/article/pii/S1571066115000705, doi:https://doi.org/10.1016/j.entcs.2015.12.003.

[PS17] Maciej Piróg and Sam Staton. Backtracking with cut via a distributive law and left-zero monoids. *J. Funct. Program.*, 27:e17, 2017. doi:10.1017/S0956796817000077.

[PSWJ18] Maciej Piróg, Tom Schrijvers, Nicolas Wu, and Mauro Jaskelioff. Syntax and semantics for operations with scopes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 809–818, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209166.

[Rém94] Didier Rémy. Type inference for records in a natural extension of ml. In *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*. Citeseer, 1994.

[RTWS18] Rob Rix, Patrick Thomson, Nicolas Wu, and Tom Schrijvers. fused-effects: A fast, flexible, fused effect system, 2018. https://hackage.haskell.org/package/fused-effects.

[SDW$^+$21] KC Sivaramakrishnan, Stephen Dolan, Leo White, Tom Kelly, Sadiq Jaffer, and Anil Madhavapeddy. Retrofitting effect handlers onto ocaml. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 206–221, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3453483.3454039.

[TRWS22] Patrick Thomson, Rob Rix, Nicolas Wu, and Tom Schrijvers. Fusing industry and academia at github (experience report). *Proc. ACM Program. Lang.*, 6(ICFP):496–511, 2022. doi:10.1145/3547639.

[Wad95] Philip Wadler. Monads for functional programming. In Johan Jeuring and Erik Meijer, editors, *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, Båstad, Sweden, May 24-30, 1995, Tutorial Text*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer, 1995. doi:10.1007/3-540-59451-5_2.

[WSH14] Nicolas Wu, Tom Schrijvers, and Ralf Hinze. Effect handlers in scope. In *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, Haskell '14, page 1–12, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2633357.2633358.

[YPW$^+$22] Zhixuan Yang, Marco Paviotti, Nicolas Wu, Birthe van den Berg, and Tom Schrijvers. Structured handling of scoped effects. In Ilya Sergey, editor, *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13240 of *Lecture Notes in Computer Science*, pages 462–491. Springer, 2022. `doi:10.1007/978-3-030-99336-8\_17`.

[YW14] Jeremy Yallop and Leo White. Lightweight higher-kinded polymorphism. In Michael Codish and Eijiro Sumii, editors, *Functional and Logic Programming*, pages 119–135, Cham, 2014. Springer International Publishing.

Contents

Appendix A. Semantic Derivations

This Appendix contains semantic derivations of different handler applications that are used in the examples throughout this paper.

A.1. Nondeterminism.

$$\begin{array}{ll}
\multicolumn{2}{l}{h_{\mathsf{ND}} \star c_{\mathsf{ND}} \;\equiv\; h_{\mathsf{ND}} \star \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,\mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)} \\
\rightsquigarrow & \{\text{- E-HandOp -}\} \\
& \quad \mathbf{do}\ xs \leftarrow (\boldsymbol{\lambda} y\,.\,h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{true} \\
& \quad \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} y\,.\,h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{false} \\
& \qquad xs +\!\!+ ys \\
\rightsquigarrow & \{\text{- E-AppAbs -}\} \\
& \quad \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star \mathbf{if}\ \mathsf{true}\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2 \\
& \quad \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} y\,.\,h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{false} \\
& \qquad xs +\!\!+ ys \\
\rightsquigarrow & \{\text{- reducing } \mathbf{if} \text{ -}\} \\
& \quad \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star \mathbf{return}\ 1 \\
& \quad \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} y\,.\,h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{false} \\
& \qquad xs +\!\!+ ys \\
\rightsquigarrow & \{\text{- E-HandRet -}\} \\
& \quad \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} y\,.\,h_{\mathsf{ND}} \star \mathbf{if}\ b\ \mathbf{then\ return}\ 1\ \mathbf{else\ return}\ 2)\ \mathsf{false} \\
& \qquad [1] +\!\!+ ys \\
\rightsquigarrow^{*} & \{\text{- similar to above (the first branch of } \mathbf{if}\text{) -}\} \\
& \qquad [1] +\!\!+ [2] \\
\rightsquigarrow & \{\text{- reducing } +\!\!+ \text{ -}\} \\
& \qquad \mathbf{return}\ [1, 2]
\end{array}$$

A.2. Increment.

$$\begin{array}{ll}
\multicolumn{2}{l}{\quad h_{\mathsf{ND}} \star run_{\mathsf{inc}}\ 0\ c_{\mathsf{inc}} \;\equiv\; h_{\mathsf{ND}} \star (\boldsymbol{\lambda} c\ p\,.\,\mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star p\,;\, p'\ c)\ 0\ c_{\mathsf{inc}}} \\
\rightsquigarrow & \{\text{- E-AppAbs -}\} \\
& h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star c_{\mathsf{inc}}\,;\, p'\ 0 \\
\equiv & \{\text{- definition of } c_{\mathsf{inc}} \text{ -}\} \\
& h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,\mathbf{if}\ b\ \ \mathbf{then\ op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2))\,;\, p'\ 0 \\
\rightsquigarrow & \{\text{- E-FwdOp -}\} \\
& h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then\ op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2))\,;\, p'\ 0 \\
\rightsquigarrow & \{\text{- E-Hand and E-DoOp -}\} \\
& h_{\mathsf{ND}} \star \mathbf{op}\ \mathtt{choose}\ ()\ (b\,.\,\mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then\ op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;\, p'\ 0) \\
\rightsquigarrow & \{\text{- E-HandOp -}\} \\
& \mathbf{do}\ xs \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then\ op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;\, p'\ 0)\ \mathsf{true}\,;
\end{array}$$

$$\begin{array}{ll}
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- E-}\textsc{AppAbs}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ \mathsf{true}\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0 \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- reducing }\mathbf{if}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star (\mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5)\,;p'\ 0) \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- E-}\textsc{HandOp}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star (\mathbf{do}\ p' \leftarrow \mathbf{return}\ (\boldsymbol{\lambda} s\,.\,\mathbf{do}\ s' \leftarrow s+1\,; \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \mathbf{do}\ k' \leftarrow (\boldsymbol{\lambda} x\,.\,h_{\mathsf{inc}} \star (x+5))\ s'\,; k'\ s')\,;p'\ 0) \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- E-}\textsc{DoRet}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star (\boldsymbol{\lambda} s\,.\,\mathbf{do}\ s' \leftarrow s+1\,; k' \leftarrow (\boldsymbol{\lambda} x\,.\,h_{\mathsf{inc}} \star (x+5))\ s'\,; k'\ s')\ 0 \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow^{*} & \{\text{- E-}\textsc{AppAbs}\text{ and reducing + -}\} \\
& \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star (h_{\mathsf{inc}} \star (\mathbf{return}\ 6))\ 1 \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- E-}\textsc{HandRet}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow (\boldsymbol{\lambda} s\,.\,\mathbf{return}\ (6, s))\ 1 \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- E-}\textsc{AppAbs}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow h_{\mathsf{ND}} \star \mathbf{return}\ (6, 1) \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,; \\
& xs \mathbin{+\!\!+} ys \\
\rightsquigarrow & \{\text{- E-}\textsc{HandOp}\text{ -}\} \\
& \mathbf{do}\ xs \leftarrow \mathbf{return}\ [(6, 1)] \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b\,.\,h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow h_{\mathsf{inc}} \star \mathbf{if}\ b\ \mathbf{then}\ \mathbf{op}\ \mathtt{inc}\ ()\ (x\,.\,x+5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ \mathbf{op}\ \mathtt{inc}\ ()\ (y\,.\,y+2)\,;p'\ 0)\ \mathsf{false}\,;
\end{array}$$

$$\begin{array}{ll}
& xs +\!\!+ ys \\
\leadsto & \text{\{- E-DoRet -\}} \\
& \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} b \,.\, h_{\mathsf{ND}} \star \mathbf{do}\ p' \leftarrow \mathbf{if}\ b\ \mathbf{then}\ h_{\mathsf{inc}} \star \mathbf{op}\ \mathtt{inc}\ ()\ (x \,.\, x + 5) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathbf{else}\ \ h_{\mathsf{inc}} \star \mathbf{op}\ \mathtt{inc}\ ()\ (y \,.\, y + 2)\,; p'\ 0)\ \mathsf{false}\,; \\
& [(6,1)] +\!\!+ ys \\
\leadsto^* & \text{\{- similar to above (the first branch of \textbf{if}) -\}} \\
& \mathbf{do}\ ys \leftarrow \mathbf{return}\ [(3,1)] \\
& [(6,1)] +\!\!+ ys \\
\leadsto^* & \text{\{- E-DoRet -\}} \\
& [(6,1)] +\!\!+ [(3,1)] \\
\leadsto^* & \text{\{- reducing } +\!\!+ \text{ -\}} \\
& \mathbf{return}\ [(6,1),(3,1)]
\end{array}$$

A.3. **Once.**

$$\begin{array}{ll}
\multicolumn{2}{l}{h_{\mathsf{once}} \star c_{\mathsf{once}}} \\
\leadsto & \text{\{- E-HandSc -\}} \\
& \mathbf{do}\ ts \leftarrow (\boldsymbol{\lambda} y \,.\, h_{\mathsf{once}} \star \mathbf{op}\ \mathtt{choose}\ ()\ (x \,.\, \mathbf{return}\ x))\ ()\,; \\
& \mathbf{do}\ t\ \ \leftarrow \mathsf{head}\ ts\,; \\
& \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ t \\
\leadsto & \text{\{- E-Do and E-AppAbs -\}} \\
& \mathbf{do}\ ts \leftarrow h_{\mathsf{once}} \star \mathbf{op}\ \mathtt{choose}\ ()\ (x \,.\, \mathbf{return}\ x)\,; \\
& \mathbf{do}\ t\ \ \leftarrow \mathsf{head}\ ts\,; \\
& \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ t \\
\leadsto & \text{\{- E-Do and E-HandOp -\}} \\
& \mathbf{do}\ ts \leftarrow \mathbf{do}\ xs \leftarrow (\boldsymbol{\lambda} x \,.\, h_{\mathsf{once}} \star \mathbf{return}\ x)\ \mathsf{true}\,; \mathbf{do}\ ys \leftarrow (\boldsymbol{\lambda} x \,.\, h_{\mathsf{once}} \star \mathbf{return}\ x)\ \mathsf{false}\,; \\
& \qquad\qquad xs +\!\!+ ys\,; \\
& \mathbf{do}\ t\ \ \leftarrow \mathsf{head}\ ts\,; \\
& \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ t \\
\leadsto & \text{\{- E-Do and E-AppAbs -\}} \\
& \mathbf{do}\ ts \leftarrow \mathbf{do}\ xs \leftarrow h_{\mathsf{once}} \star \mathbf{return}\ \mathsf{true}\,; \mathbf{do}\ ys \leftarrow h_{\mathsf{once}} \star \mathbf{return}\ \mathsf{false}\,; xs +\!\!+ ys\,; \\
& \mathbf{do}\ t\ \ \leftarrow \mathsf{head}\ ts\,; \\
& \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ t \\
\leadsto & \text{\{- E-Do and E-HandRet -\}} \\
& \mathbf{do}\ ts \leftarrow \mathbf{do}\ xs \leftarrow \mathbf{return}\ [\mathsf{true}]\,; \mathbf{do}\ ys \leftarrow \mathbf{return}\ [\mathsf{false}]\,; xs +\!\!+ ys\,; \\
& \mathbf{do}\ t\ \ \leftarrow \mathsf{head}\ ts\,; \\
& \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ t \\
\leadsto^* & \text{\{- E-DoRet -\}} \\
& \mathbf{do}\ ts \leftarrow [\mathsf{true}, \mathsf{false}]\,; \\
& \mathbf{do}\ t\ \ \leftarrow \mathsf{head}\ ts\,; \\
& \quad (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ t \\
\leadsto^* & \text{\{- E-DoRet -\}} \\
& (\boldsymbol{\lambda} p \,.\, h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \mathtt{choose}\ (b \,.\, \mathbf{return}\ b)\,; \mathbf{return}\ (p,q)))\ \mathsf{true} \\
\leadsto & \text{\{- E-AppAbs -\}}
\end{array}$$

$$
\begin{array}{l}
\quad h_{\mathsf{once}} \star (\mathbf{do}\ q \leftarrow \mathbf{op}\ \texttt{choose}\ (b\,.\,\mathbf{return}\ b)\,;\mathbf{return}\ (\mathsf{true}, q)) \\
\rightsquigarrow^{*} \quad \text{\{- similar to A.1 (handling of \texttt{choose}) -\}} \\
\quad \mathbf{return}\ [(\mathsf{true}, \mathsf{true}), (\mathsf{true}, \mathsf{false})]
\end{array}
$$

Appendix B. Type Equivalence Rules

This appendix shows the type equivalence rules of λ_{sc}. Figures 7 and 8 contains the rules. Rules Q-AppAbs and Q-Swap deserve special attention. The other rules are straightforward.

$\boxed{\sigma_1 \;\equiv\; \sigma_2}$ Type equivalence

$$\frac{}{\sigma \;\equiv\; \sigma}\ \textsc{Q-Refl} \qquad \frac{\sigma_1 \;\equiv\; \sigma_2}{\sigma_2 \;\equiv\; \sigma_1}\ \textsc{Q-Symm} \qquad \frac{\sigma_1 \;\equiv\; \sigma_2 \qquad \sigma_2 \;\equiv\; \sigma_3}{\sigma_1 \;\equiv\; \sigma_3}\ \textsc{Q-Trans}$$

$$\frac{A_1 \;\equiv\; A_2 \qquad B_1 \;\equiv\; B_2}{(A_1, B_1) \;\equiv\; (A_2, B_2)}\ \textsc{Q-Pair} \qquad \frac{A \;\equiv\; B \qquad \underline{C} \;\equiv\; \underline{D}}{A \to \underline{C} \;\equiv\; B \to \underline{D}}\ \textsc{Q-Fun}$$

$$\frac{\underline{\underline{C_1}} \;\equiv\; \underline{\underline{D_1}} \qquad \underline{\underline{C_2}} \;\equiv\; \underline{\underline{D_2}}}{\underline{C_1} \Rightarrow \underline{C_2} \;\equiv\; \underline{D_1} \Rightarrow \underline{D_2}}\ \textsc{Q-Hand} \qquad \frac{\sigma_1 \;\equiv\; \sigma_2}{\forall\, \alpha\,.\, \sigma_1 \;\equiv\; \forall\, \alpha\,.\, \sigma_2}\ \textsc{Q-AllTy}$$

$$\frac{\sigma_1 \;\equiv\; \sigma_2}{\forall\, \mu\,.\, \sigma_1 \;\equiv\; \forall\, \mu\,.\, \sigma_2}\ \textsc{Q-AllRow} \qquad \frac{A \;\equiv\; B}{\lambda\, \alpha\,.\, A \;\equiv\; \lambda\, \alpha\,.\, B}\ \textsc{Q-Abs}$$

$$\frac{M_1 \;\equiv\; M_2 \qquad A \;\equiv\; B}{M_1\ A \;\equiv\; M_2\ B}\ \textsc{Q-App} \qquad \frac{}{(\lambda\, \alpha\,.\, A)\ B \;\equiv\; A\,[B \,/\, \alpha]}\ \textsc{Q-AppAbs}$$

$$\frac{A \;\equiv\; B \qquad E \;\equiv_{\langle\rangle}\; F}{A\,!\,\langle E \rangle \equiv\; B\,!\,\langle F \rangle}\ \textsc{Q-Comp}$$

Figure 7: Type equivalence of λ_{sc}.

$\boxed{E \;\equiv_{\langle\rangle}\; F}$ Row equivalence

$$\frac{}{E \;\equiv_{\langle\rangle}\; E}\ \textsc{R-Refl} \qquad \frac{E \;\equiv_{\langle\rangle}\; F}{F \;\equiv_{\langle\rangle}\; E}\ \textsc{R-Symm} \qquad \frac{E_1 \;\equiv_{\langle\rangle}\; E_2 \qquad E_2 \;\equiv_{\langle\rangle}\; E_3}{E_1 \;\equiv_{\langle\rangle}\; E_3}\ \textsc{R-Trans}$$

$$\frac{E \;\equiv_{\langle\rangle}\; F}{\ell\,; E \;\equiv_{\langle\rangle}\; \ell\,; F}\ \textsc{R-Head} \qquad \frac{\ell_1 \;\neq\; \ell_2}{\ell_1\,; \ell_2\,; E \;\equiv_{\langle\rangle}\; \ell_2\,; \ell_1\,; E}\ \textsc{R-Swap}$$

Figure 8: Row equivalence of λ_{sc}.

Appendix C. Well-scopedness Rules

This appendix shows the well-scopedness rules of λ_{sc}. Figure 9 contains the rules.

$\boxed{\Gamma \vdash \sigma}$ $\boxed{\Gamma \vdash M}$ $\boxed{\Gamma \vdash E}$ $\boxed{\Gamma \vdash \underline{C}}$ Type well-scopedness

$$\frac{}{\Gamma \vdash ()}\ \textsc{W-Unit} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash (A, B)}\ \textsc{W-Pair} \qquad \frac{\alpha \in \Gamma}{\Gamma \vdash \alpha}\ \textsc{W-Var}$$

$$\frac{\Gamma, \alpha \vdash A}{\Gamma \vdash \forall\, \alpha \,.\, A}\ \textsc{W-All} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash E}{\Gamma \vdash A \,!\, \langle E \rangle}\ \textsc{W-Comp} \qquad \frac{\Gamma, \alpha \vdash A}{\Gamma \vdash \lambda\, \alpha \,.\, A}\ \textsc{W-Abs}$$

$$\frac{\Gamma \vdash M \qquad \Gamma \vdash A}{\Gamma \vdash M\ A}\ \textsc{W-App} \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \underline{C}}{\Gamma \vdash A \to \underline{C}}\ \textsc{W-Fun} \qquad \frac{\Gamma \vdash \underline{C} \qquad \Gamma \vdash \underline{D}}{\Gamma \vdash \underline{C} \Rightarrow \underline{D}}\ \textsc{W-Hand}$$

$$\frac{\mu \in \Gamma}{\Gamma \vdash \mu}\ \textsc{W-RowVar} \qquad \frac{}{\Gamma \vdash \cdot}\ \textsc{W-EmptyRow} \qquad \frac{\Gamma \vdash E}{\Gamma \vdash \ell \,;\, E}\ \textsc{W-Extension}$$

Figure 9: Well-scopedness rules of λ_{sc}.

This section describes the syntax-direction version of λ_{sc}. It is this version we prove type safe in Appendix E. Furthermore, it serves as the specification of our type inference algorithm as described in Appendix F.

The syntax-directed rules can be found in Figure 10 for value typing, Figure 11 for computation and Figure 12 for handler typing.

$\boxed{\Gamma \vdash v : A}$ Value Typing

$$\frac{(x : \sigma) \in \Gamma \qquad \sigma \leqslant A \qquad \Gamma \vdash A}{\Gamma \vdash x : A}\ \textsc{SD-Var} \qquad \frac{}{\Gamma \vdash () : ()}\ \textsc{SD-Unit}$$

$$\frac{\Gamma \vdash v_1 : A \qquad \Gamma \vdash v_2 : B}{\Gamma \vdash (v_1, v_2) : (A, B)}\ \textsc{SD-Pair} \qquad \frac{\Gamma, x : A \vdash c : \underline{C}}{\Gamma \vdash \boldsymbol{\lambda} x \,.\, c : A \to \underline{C}}\ \textsc{SD-Abs}$$

$$\textsc{SD-Handler}\quad \frac{\begin{array}{c} F \equiv_{\langle\rangle} \mathit{labels}\,(\mathit{oprs})\,; E \qquad \alpha \notin \Gamma \qquad \Gamma, \alpha \vdash_M \mathbf{return}\ x \mapsto c_r : M\ \alpha!\langle E\rangle \\ \Gamma, \alpha \vdash_M \mathit{oprs} : M\ \alpha!\langle E\rangle \qquad \Gamma, \alpha \vdash_M \mathbf{fwd}\ f\ p\ k \mapsto c_f : M\ \alpha!\langle E\rangle \qquad \Gamma \vdash A \end{array}}{\Gamma \vdash \mathbf{handler}_M\ \{\mathbf{return}\ x \mapsto c_r, \mathit{oprs}, \mathbf{fwd}\ f\ p\ k \mapsto c_f\} : A!\langle F\rangle \Rightarrow M\ A!\langle E\rangle}$$

Figure 10: Syntax-directed value typing.

The syntax-directed system is obtained by incorporating the non-syntax-directed rules into the syntax-directed-ones where needed. In particular, we inline the non-syntax-directed rules for equivalence (T-EqV and T-EqC) into the syntax-directed rules that mention the same type or row twice in their assumptions (e.g., SD-App, SD-Do). Similarly, we inline the rules T-Inst, T-InstEff, T-Gen and T-GenEff for instantiating and generalizing type and row variables. The generalization is incorporated into the rule for let-bindings (T-Let). Instantiation is incorporated into the variable rule (T-Var) using $\sigma \leqslant A$ defined in Figure 13.

Instantiation is also incorporated into the handler rule: we implicitly instantiate α with an arbitrary type A, which results in a monomorphically typed handler. However, since SD-Handler insists on sufficiency polymorphic handler clauses, we can still handle scoped effects by polymorphic recursion.

Figure 14 displays declarative and syntax-directed typing derivations for both inline handler application ($h \star c$) as well as let-bound handlers. As can be seen in the first derivation, in the case of inline handler application, the declarative system derives a polymorphically typed handler, which is instantiated. The syntax-directed system essentially combines these steps, as can be seen in the second derivation. In the case of a let-bound handler, the declarative system keeps the polymorphic handler type as-is (third derivation). The syntax-directed system however instantiates and then immediately generalizes handlers, as can be seen in the fourth derivation.

The other rules of the declarative system are syntax-directed and remain unchanged.

$\boxed{\Gamma \vdash c : \underline{C}}$ Computation Typing

$$\frac{\Gamma \vdash v_1 : A_1 \to \underline{C} \qquad \Gamma \vdash v_2 : A_2 \qquad A_1 \ \equiv \ A_2}{\Gamma \vdash v_1 \ v_2 : \underline{C}} \ \textsc{SD-App}$$

$$\frac{\Gamma \vdash c_1 : A\,!\langle E_1\rangle \qquad \Gamma, x : A \vdash c_2 : B\,!\langle E_2\rangle \qquad E_1 \ \equiv \ E_2}{\Gamma \vdash \mathbf{do}\ x \leftarrow c_1\,;c_2 : B\,!\langle E_2\rangle} \ \textsc{SD-Do}$$

$$\frac{\Gamma, \overline{\alpha}, \overline{\mu} \vdash v : A \qquad (\overline{\alpha} \ \notin \ \Gamma) \qquad (\overline{\mu} \ \notin \ \Gamma) \qquad \Gamma, x : \overline{\forall\, \alpha}\,.\,\overline{\forall\, \mu}\,.\,A \vdash c : \underline{C}}{\Gamma \vdash \mathbf{let}\ x = v\ \mathbf{in}\ c : \underline{C}} \ \textsc{SD-Let}$$

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash \mathbf{return}\ v : A\,!\langle E\rangle} \ \textsc{SD-Ret}$$

$$\frac{\Gamma \vdash v : \underline{C}_1 \Rightarrow \underline{D}_1 \qquad \Gamma \vdash c : \underline{C}_2 \qquad \underline{C}_1 \ \equiv \ \underline{C}_2 \qquad \underline{D}_1 \ \equiv \ \underline{D}_2}{\Gamma \vdash v \star c : \underline{D}_2} \ \textsc{SD-Hand}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \ \in \ \Sigma \\ \Gamma \vdash v : A_1 \qquad A_{\mathsf{op}} \ \equiv \ A_1 \qquad \Gamma, y : B_{\mathsf{op}} \vdash c : A\,!\langle E\rangle \qquad \ell^{\mathsf{op}} \ \in \ E\end{array}}{\Gamma \vdash \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c) : A\,!\langle E\rangle} \ \textsc{SD-Op}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \ \in \ \Sigma \qquad \Gamma \vdash v : A_1 \qquad A_{\mathsf{sc}} \ \equiv \ A_1 \\ \Gamma, y : B_{\mathsf{sc}} \vdash c_1 : B\,!\langle E_1\rangle \qquad \Gamma, z : B \vdash c_2 : A\,!\langle E_2\rangle \qquad E_1 \ \equiv \ E_2 \qquad \ell^{\mathsf{sc}} \ \in \ E_2\end{array}}{\Gamma \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : A\,!\langle E_2\rangle} \ \textsc{SD-Sc}$$

Figure 11: Syntax-directed computation typing.

$\boxed{\Gamma \vdash_M \ \textbf{return}\ x \mapsto c_r : M\ A!\langle E\rangle}$ $\boxed{\Gamma \vdash_M \ \mathit{oprs} : M\ A!\langle E\rangle}$

$\boxed{\Gamma \vdash_M \ \textbf{fwd}\ f\ p\ k \mapsto c_f : M\ A!\langle E\rangle}$

Return-, operation-, and forwarding-clause typing

$$\frac{\Gamma, x : A_1 \vdash c_r : M\ A_2!\langle E\rangle \qquad A_1 \equiv A_2}{\Gamma \vdash_M \ \textbf{return}\ x \mapsto c_r : M\ A!\langle E_2\rangle}\ \textsc{SD-Return} \qquad \frac{}{\Gamma \vdash_M \ \cdot : M\ A!\langle E\rangle}\ \textsc{SD-Empty}$$

$$\frac{\begin{array}{c}\Gamma \vdash_M \ \mathit{oprs} : M\ A_1!\langle E_1\rangle \qquad (\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma \\ \Gamma, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ A_1!\langle E_1\rangle \vdash c : M\ A_2!\langle E_2\rangle \\ M\ A_1!\langle E_1\rangle \equiv M\ A_2!\langle E_2\rangle\end{array}}{\Gamma \vdash_M \ \textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs} : M\ A_2!\langle E_2\rangle}\ \textsc{SD-OprOp}$$

$$\frac{\begin{array}{c}\Gamma \vdash_M \ \mathit{oprs} : M\ A_1!\langle E_1\rangle \qquad (\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \in \Sigma \qquad \beta \notin \Gamma \\ \Gamma, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta!\langle E_1\rangle, k : \beta \to M\ A_1!\langle E_1\rangle \vdash c : M\ A_2!\langle E_2\rangle \\ M\ A_1!\langle E_1\rangle \equiv M\ A_2!\langle E_2\rangle\end{array}}{\Gamma \vdash_M \ \textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, \mathit{oprs} : M\ A_2!\langle E_2\rangle}\ \textsc{SD-OprSc}$$

$$\frac{\begin{array}{c}\alpha, \beta, \gamma, \delta \notin \Gamma \qquad A_p = \alpha \to M\ \beta!\langle E_1\rangle \\ A'_p = \alpha \to \gamma!\langle E_1\rangle \qquad A_k = \beta \to M\ A_1!\langle E_1\rangle \qquad A'_k = \gamma \to \delta!\langle E_1\rangle \\ \Gamma, \alpha, \beta, p : A_p, k : A_k, f : \forall\, \gamma\ \delta\,.\,(A'_p, A'_k) \to \delta!\langle E_1\rangle \vdash c_f : M\ A_2!\langle E_2\rangle \\ M\ A_1!\langle E_1\rangle \equiv M\ A_2!\langle E_2\rangle \qquad \Gamma \vdash A_2\end{array}}{\Gamma \vdash_M \ \textbf{fwd}\ f\ p\ k \mapsto c_f : M\ A_2!\langle E_2\rangle}\ \textsc{SD-Fwd}$$

Figure 12: Syntax-directed handler typing.

$\boxed{\sigma \leqslant A}$ σ-instantiation

$$\frac{}{A \leqslant A}\ \sigma\text{-}\textsc{Inst-Base} \qquad \frac{[B\ /\ \alpha]\sigma \leqslant A}{\forall\, \alpha\,.\,\sigma \leqslant A}\ \sigma\text{-}\textsc{Inst-}\alpha \qquad \frac{[E\ /\ \mu]\ \sigma \leqslant A}{\forall\, \mu\,.\,\sigma \leqslant A}\ \sigma\text{-}\textsc{Inst-}\mu$$

Figure 13: σ-instantiation.

$$\dfrac{\dfrac{\dfrac{\Gamma, \alpha \vdash \mathit{oprs} : M\ \alpha!\langle E\rangle}{\Gamma \vdash h : \forall\, \alpha\,.\, \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle}\ \textsc{T-Handler} \qquad \Gamma \vdash A}{\Gamma \vdash h : A!\langle F\rangle \Rightarrow M\ A!\langle E\rangle}\ \textsc{T-Inst} \qquad \Gamma \vdash c : A!\langle F\rangle}{\Gamma \vdash h \star c : M\ A!\langle E\rangle}\ \textsc{T-Hand}$$

$$\dfrac{\dfrac{\Gamma, \alpha \vdash_M \ \mathit{oprs} : M\ \alpha!\langle E\rangle \qquad \Gamma \vdash A}{\Gamma \vdash h_M : A!\langle F\rangle \Rightarrow M\ A!\langle E\rangle}\ \textsc{SD-Handler} \qquad \Gamma \vdash c : A!\langle F\rangle}{\Gamma \vdash h_M \star c : M\ A!\langle E\rangle}\ \textsc{SD-Hand}$$

$$\dfrac{\dfrac{\Gamma, \alpha \vdash \mathit{oprs} : M\ \alpha!\langle E\rangle}{\Gamma \vdash h : \forall\, \alpha\,.\, \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle}\ \textsc{T-Handler} \qquad \Gamma, x : \forall\, \alpha\,.\, \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle \vdash c : \underline{C}}{\Gamma \vdash \mathbf{let}\ x = h\ \mathbf{in}\ c : \underline{C}}\ \textsc{T-Let}$$

$$\dfrac{\dfrac{\begin{array}{c}\Gamma, \alpha, \beta \vdash_M \ \mathit{oprs} : M\ \beta!\langle E\rangle \\ \Gamma, \alpha \vdash \alpha\end{array}}{\Gamma \vdash h_M : \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle}\ \textsc{SD-Handler} \qquad \Gamma, x : \forall\, \alpha\,.\, \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle \vdash c : \underline{C}}{\Gamma \vdash \mathbf{let}\ x = h_M\ \mathbf{in}\ c : \underline{C}}\ \textsc{SD-Let}$$

Figure 14: Handler generalisation and instantiation

E.1. **Lemmas.**

Lemma E.1 (Canonical forms).
- *If* $\cdot \vdash v : A \to \underline{C}$ *then* v *is of shape* $\boldsymbol{\lambda} x \,.\, c$.
- *If* $\cdot \vdash v : \underline{C} \Rightarrow \underline{D}$ *then* v *is of shape* h.

Lemma E.2 (Generalisation-equivalence). *If* $\sigma_1 \leqslant A_1$ *and* $\sigma_1 \equiv \sigma_2$*, then there exists a* A_2 *such that* $A_1 \equiv A_2$ *and* $\sigma_2 \leqslant A_2$.

Lemma E.3 (Generalisation-instantiation). *If* $\Gamma, \overline{\alpha}, \overline{\mu} \vdash v : A$ *and* $\overline{\forall\, \alpha}\ \overline{\forall\, \mu} \,.\, A \leqslant B$*, then* $\Gamma \vdash v : B$.

Lemma E.4 (Preservation of types under term substitution). *Given* $\Gamma_1, \overline{\alpha}, \overline{\mu} \vdash v : A_1$ *and* $A_1 \equiv A_2$ *we have that:*
- *If* $\Gamma_1, x : \overline{\forall\, \alpha}\ \overline{\forall\, \mu} \,.\, A_2, \Gamma_2 \vdash c : \underline{C}_1$*, then there exists a* $\underline{C}_2$ *such that* $\underline{C}_1 \equiv \underline{C}_2$ *and* $\Gamma_1, \Gamma_2 \vdash [v \,/\, x]\ c : \underline{C}_2$.
- *If* $\Gamma_1, x : \overline{\forall\, \alpha}\ \overline{\forall\, \mu} \,.\, A_2, \Gamma_2 \vdash v : B_1$*, then there exists a* B_2 *such that* $B_1 \equiv B_2$ *and* $\Gamma_1, \Gamma_2 \vdash [v \,/\, x]\ v : B_2$.

Proof. By mutual induction on the typing derivations. The only interesting case, SD-VAR, requires us to show that, given $\Gamma_1, x : \overline{\forall\, \alpha}\ \overline{\forall\, \mu} \,.\, A_2, \Gamma_2 \vdash y : B_1$, there exists a B_2 such that $B_1 \equiv B_2$ and $\Gamma_1, \Gamma_2 \vdash [v \,/\, x]\ y : B_2$. If $x \neq y$, it is trivial. If $x = y$, then $\overline{\forall\, \alpha}\ \overline{\forall\, \mu} \,.\, A_2 \leqslant B_1$, which means by Lemma E.2 there exists a B_2 such that $B_1 \equiv B_2$ and $\overline{\forall\, \alpha}\ \overline{\forall\, \mu} \,.\, A_1 \leqslant B_2$, which means the result follows from Lemma E.3. □

Lemma E.5 (Preservation of types under type substitution). *If* $\Gamma_1, \alpha, \Gamma_2 \vdash c : \underline{C}$ *and* $\Gamma_1 \vdash B$*, then* $\Gamma_1, [B \,/\, \alpha]\ \Gamma_2 \vdash c : [B \,/\, \alpha]\ \underline{C}$.

Lemma E.6 (Unused binding insertion). *If* $\Gamma_1, \Gamma_2 \vdash c : \underline{C}$ *and* $x \notin c$ *then* $\Gamma_1, x : A, \Gamma_2 \vdash c : \underline{C}$.

Lemma E.7 (Handlers are polymorphic). *If* $\Gamma \vdash h : A\,!\langle F \rangle \Rightarrow M\ A\,!\langle E \rangle$ *and* $\Gamma \vdash B$*, then* $\Gamma \vdash h : B\,!\langle F \rangle \Rightarrow M\ B\,!\langle E \rangle$.

Lemma E.8 (Op membership). *If* $\Gamma \vdash oprs : \underline{C}$ *and* $\mathbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c \in oprs$*, then there exists* $oprs_1$ *and* $oprs_2$ *such that* $oprs = oprs_1, \mathbf{op}\ \ell^{\mathsf{op}}\ k \vdash c, oprs_2$ *and* $\Gamma \vdash \mathbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, oprs_2 : \underline{C}$.

Lemma E.9 (Sc membership). *If* $\Gamma \vdash oprs : \underline{C}$ *and* $(\mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c) \in h$*, then there exists* $oprs_1$ *and* $oprs_2$ *such that* $oprs = oprs_1, \mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, oprs_2$ *and* $\Gamma \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, oprs_2 : \underline{C}$.

E.2. **Subject reduction.**

Theorem 6.1 (Subject Reduction). *If* $\Gamma \vdash c : \underline{C}$ *and* $c \rightsquigarrow c'$*, then there exists a* $\underline{C}'$ *such that* $\underline{C} \equiv \underline{C}'$ *and* $\Gamma \vdash c' : \underline{C}'$.

Proof. Assume, without loss of generality, that $\underline{C} = B\,!\langle F \rangle$ for some B, F. Proceed by induction on the derivation $c \rightsquigarrow c'$.
- E-APPABS: Inversion on $\Gamma \vdash (\boldsymbol{\lambda} x \,.\, c)\ v : B\,!\langle F \rangle$ (SD-APP) gives $\Gamma \vdash \boldsymbol{\lambda} x \,.\, c : A_1 \to B\,!\langle F \rangle$ (1), $\Gamma \vdash v : A_2$ (2), and $A_1 \equiv A_2$ (3). Inversion on fact 1 (SD-ABS) gives $\Gamma, x : A_1 \vdash c : B\,!\langle F \rangle$ (4), which means the goal follows from facts 2 and 4 and Lemma E.4.

- E-LET: Inversion on $\Gamma \vdash \textbf{let } x = v \textbf{ in } c : B!\langle F \rangle$ (SD-LET) gives $\Gamma \vdash v : A$ (1), $\sigma = \mathit{gen}\ (A, \Gamma)$ (2), and $\Gamma, x : \sigma \vdash c : B!\langle F \rangle$ (3), which means the goal follows from facts 1 and 3 and Lemma E.4.
- E-DO: Follows from the IH.
- E-DORET: Inversion on $\Gamma \vdash \textbf{do } x \leftarrow \textbf{return } v \textbf{ in } c : B!\langle F_2 \rangle$ (SD-DO) gives $\Gamma \vdash \textbf{return } v : A!\langle F_1 \rangle$ (1) and $\Gamma, x : A \vdash c : B!\langle F_2 \rangle$ (2). Inversion on (1) (SD-RET) gives $\Gamma \vdash v : A$ (3). The case follows from facts 2 and 4 and Lemma E.4.
- E-DOOP: Similar to E-DOSC. By inversion on $\Gamma \vdash \textbf{do } x \leftarrow \textbf{op } \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) \textbf{ in } c_2 : B!\langle F_2 \rangle$ (SD-DO) we have that $\Gamma \vdash \textbf{op } \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : A!\langle F_1 \rangle$ (1), $\Gamma, x : A \vdash c_2 : B!\langle F_2 \rangle$ (2), and $F_1 \equiv F_2$ (3). From inversion on fact 1 (SD-OP) it follows that $\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}} \in \Sigma$ (4), $\Gamma \vdash v : A_1$ (5), $A_{\mathsf{op}} \equiv A_1$ (6), $\Gamma, y : B_{\mathsf{op}} \vdash c_1 : A!\langle F_1 \rangle$ (7), and $\ell^{\mathsf{op}} \in F_1$ (8). Lemma E.6 on (2) gives us $\Gamma, y : B_{\mathsf{op}}, x : A \vdash c_2 : B!\langle F_2 \rangle$ (9). Facts 3, 7 and 9 and rule SD-DO give us $\Gamma, y : B_{\mathsf{op}} \vdash \textbf{do } x \leftarrow c_1 \textbf{ in } c_2 : B!\langle F_2 \rangle$ (10). Our goal then follows from facts 4, 5, 6, 8, and 10 and rule SD-OP.
- E-DOSC: Similar to E-DOOP. By inversion on $\Gamma \vdash \textbf{do } x \leftarrow \textbf{sc } \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) \textbf{ in } c_3 : B!\langle F3 \rangle$ (SD-DO) we have that $\Gamma \vdash \textbf{sc } \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : A!\langle F_2 \rangle$ (1), $\Gamma, x : A \vdash c_3 : B!\langle F3 \rangle$ (2), and $F_2 \equiv F3$ (2.1). From inversion on fact 1 (SD-SC) it follows that $\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}} \in \Sigma$ (3), $\Gamma \vdash v : A_1$ (4), $A_{\mathsf{sc}} \equiv A_1$ (5), $\Gamma, y : B_{\mathsf{sc}} \vdash c_1 : B'!\langle F_1 \rangle$ (6), $\Gamma, z : B' \vdash c_2 : A!\langle F_2 \rangle$ (7), $F_1 \equiv F_2$ (8), and $\ell^{\mathsf{sc}} \in F_2$ (9). Lemma E.6 on (2) gives us $\Gamma, z : B', x : A \vdash c_3 : B!\langle F3 \rangle$ (10), which means facts 2.1, 7 and 10 and rule SD-DO give us $\Gamma, z : B' \vdash \textbf{do } x \leftarrow c_2 \textbf{ in } c_3 : B!\langle F3 \rangle$ (11). Our goal then follows from facts 3, 4, 5, 6, 8 9, and 11 and rule SD-SC.
- E-HAND: Follows from the IH.
- E-HANDRET: By inversion on $\Gamma \vdash h \star \textbf{return } v : B!\langle F_2 \rangle$ (SD-HAND) we have that $\Gamma \vdash h : \underline{C}_1 \Rightarrow B!\langle F_2 \rangle$ (1), $\Gamma \vdash \textbf{return } v : \underline{C}_2$ (2), and $\underline{C}_1 \equiv \underline{C}_2$ (3). Inversion on fact 1 (SD-HANDLER) gives $B = M\ A_2$, $\underline{C}_1 = A_2!\langle E \rangle$, and $\Gamma, \alpha \vdash_M \textbf{return } x \mapsto c_r : M\ \alpha!\langle F_2 \rangle$ (4). Based on fact (3) we get that $\underline{C}_2 = A2'!\langle E' \rangle$, $A_2 \equiv A2'$ (4), and $E \equiv E'$ (5). Inversion on fact 4 (SD-RETURN) gives $\Gamma, \alpha, x : A_1 \vdash c_r : M\ \alpha!\langle F_2 \rangle$ (5) and $A_1 \equiv A_2$ (6). Inversion on fact 2 (SD-RET) gives $\Gamma \vdash v : A2'$ (7). From facts 4-8 and Lemma E.4, we get that $\Gamma, \alpha \vdash [v \boldsymbol{\lambda} x]\ c_r : M\ \alpha!\langle F_2 \rangle$ (8). We obtain our goal from fact 8 and Lemma E.5.
- E-HANDOP: By inversion on $\Gamma \vdash h \star \textbf{op } \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : B!\langle F_2 \rangle$ (SD-HAND) we have that $\Gamma \vdash h : \underline{C}_1 \Rightarrow B!\langle F_2 \rangle$ (1), $\Gamma \vdash \textbf{op } \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : \underline{C}_2$ (2), and $\underline{C}_1 \equiv \underline{C}_2$ (3). Inversion on fact 1 (SD-HANDLER) gives $B = M\ A_2$, $\underline{C}_1 = A_2!\langle E \rangle$, and $\Gamma, \alpha \vdash_M \mathit{oprs} : M\ \alpha!\langle F_2 \rangle$ (4). Based on fact (3) we get that $\underline{C}_2 = A2'!\langle E' \rangle$, $A_2 \equiv A2'$ (4), and $E \equiv E'$ (5). Inversion on fact 2 (SD-OP) gives us $\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}} \in \Sigma$ (6), $\Gamma \vdash v : A_1$ (7), $A_{\mathsf{op}} \equiv A_1$ (8), $\Gamma, y : B_{\mathsf{op}} \vdash c_1 : A2'!\langle E' \rangle$ (9), and $\ell^{\mathsf{op}} \in E'$ (10). By Lemma E.8 we get that $\Gamma, \alpha \vdash_M \textbf{op } \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs}_2 : M\ \alpha!\langle F_2 \rangle$ (11). Inversion on fact 11 (SD-OPROP) gives that $\Gamma, \alpha \vdash_M \mathit{oprs} : M\ \alpha!\langle F_1 \rangle$ (12), $(\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma$ (13), $\Gamma, \alpha, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \alpha!\langle F_1 \rangle \vdash c : M\ \alpha!\langle F_2 \rangle$ (14), and $F_1 \equiv F_2$ (15). Facts 1, 4 and 9 in combination with constructors SD-ABS and ST-HAND gives us that $\Gamma \vdash \boldsymbol{\lambda} y\,.\,h \star c_1 : B_{\mathsf{op}} \to M\ A_2!\langle F_2 \rangle$ (16). The goal follows from facts 7, 8, 14 and 16 and lemmas Lemmas E.4 and E.5.
- E-FWDOP By inversion on $\Gamma \vdash h \star \textbf{op } \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : B!\langle F_2 \rangle$ (SD-HAND) we have that $\Gamma \vdash h : \underline{C}_1 \Rightarrow B!\langle F_2 \rangle$ (1), $\Gamma \vdash \textbf{op } \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : \underline{C}_2$ (2), and $\underline{C}_1 \equiv \underline{C}_2$ (3). Inversion on fact 1 (SD-HANDLER) gives $B = M\ A_2$, and $\underline{C}_1 = A_2!\langle E \rangle$. Based on fact (3) we get that $\underline{C}_2 = A2'!\langle E' \rangle$, $A_2 \equiv A2'$ (4), and $E \equiv E'$ (5). Inversion on fact 2 (SD-OP) gives us $\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}} \in \Sigma$ (6), $\Gamma \vdash v : A_1$ (7), $A_{\mathsf{op}} \equiv A_1$ (8), $\Gamma, y : B_{\mathsf{op}} \vdash c_1 : A2'!\langle E' \rangle$ (9), and

$\ell^{\mathsf{op}} \in E'$ (10). The goal follows from facts 1, 4, 5, 6, 7, 8, 10, constructors SD-Hand and SD-Op, and Lemma E.6.

- E-HandSc: By inversion on $\Gamma \vdash h \star \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : B\,!\langle F_2\rangle$ (SD-Hand) we have that $\Gamma \vdash h : \underline{C_1} \Rightarrow B\,!\langle F_2\rangle$ (1), $\Gamma \vdash \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : \underline{C_2}$ (2), and $\underline{C_1} \equiv \underline{C_2}$ (3). Inversion on fact 1 (SD-Handler) gives $B = M\ A_2$, $\underline{C_1} = A_2\,!\langle E_1\rangle$, and $\Gamma, \alpha \vdash_M \mathit{oprs} : M\ \alpha\,!\langle F_2\rangle$ (4). Based on fact (3) we get that $\underline{C_2} = A2'\,!\langle E_2\rangle$, $A_2 \equiv A2'$ (5), and $E_1 \equiv E_2$ (6). Inversion on fact 2 (SD-Sc) gives us $\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}} \in \Sigma$ (7), $\Gamma \vdash v : A_1$ (8), $A_{\mathsf{sc}} \equiv A_1$ (9), $\Gamma, y : B_{\mathsf{sc}} \vdash c_1 : A_3\,!\langle E_3\rangle$ (10), $\Gamma, z : A_3 \vdash c_2 : A2'\,!\langle E_2\rangle$ (11), $E_3 \equiv E_2$ (12), and $\ell^{\mathsf{sc}} \in E_2$ (13). Lemma E.8 we get that $\Gamma, \alpha \vdash_M \mathbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs}_2 : M\ \alpha\,!\langle F_2\rangle$ (13.1). Inversion on fact 13.1 (SD-OprSc gives $\ell^{\mathsf{sc}} \in \Sigma$ (14), β *fresh* (15), $\Gamma, \alpha, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta\,!\langle F3\rangle, k : \beta \to M\ \alpha\,!\langle F3\rangle \vdash c : M\ \alpha\,!\langle F_2\rangle$ (16), and $F_2 \equiv F3$ (17). Facts 1, 5, 6, 10 and 12, constructors SD-Abs and SD-Hand and Lemmas E.6 and E.7 give us that $\Gamma, \beta \vdash \boldsymbol{\lambda} y\,.\,h \star c_1 : p : B_{\mathsf{sc}} \to M\ \beta\,!\langle F_2\rangle$ (18). Facts 1, 5, 6 and 11 and constructors SD-Abs and SD-Hand and Lemma E.6 give us that $\Gamma, \beta \vdash \boldsymbol{\lambda} z\,.\,h \star c_2 : \beta \to M\ A_2\,!\langle F_2\rangle$ (19). The goal now follows from facts 8, 9, 16, 17 and 18 and Lemma E.4.
- E-FwdSc By inversion on $\Gamma \vdash h \star \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c_1) : B\,!\langle F_2\rangle$ (SD-Hand) we have that $\Gamma \vdash h : \underline{C_1} \Rightarrow B\,!\langle F_2\rangle$ (1), $\Gamma \vdash \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : \underline{C_2}$ (2), and $\underline{C_1} \equiv \underline{C_2}$ (3). Inversion on fact 1 (SD-Handler) gives $B = M\ A_2$, $\underline{C_1} = A_2\,!\langle E_1\rangle$, and $\Gamma, \alpha \vdash_M \mathbf{fwd}\ f\ p\ k \mapsto c_f : M\ \alpha\,!\langle F_2\rangle$ (4). Based on fact (3) we get that $\underline{C_2} = A2'\,!\langle E_2\rangle$, $A_2 \equiv A2'$ (5), and $E_1 \equiv E_2$ (6). Inversion on fact 2 (SD-Sc) gives us $\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}} \in \Sigma$ (7), $\Gamma \vdash v : A_1$ (8), $A_{\mathsf{sc}} \equiv A_1$ (9), $\Gamma, y : B_{\mathsf{sc}} \vdash c_1 : A_3\,!\langle E_3\rangle$ (10), $\Gamma, z : A_3 \vdash c_2 : A2'\,!\langle E_2\rangle$ (11), $E_3 \equiv E_2$ (12), and $\ell^{\mathsf{sc}} \in E_2$ (13). Inversion on fact 4 (SD-Fwd) gives $A_p = \alpha' \to M\ \beta\,!\langle F_1\rangle$, $A'_p = \alpha' \to \gamma\,!\langle F_1\rangle$, $A_k = \beta \to M\ A_4\,!\langle F_1\rangle$, $A'_k = \gamma \to \delta\,!\langle F_1\rangle$, $\Gamma, \alpha, \alpha', \beta, p : A_p, k : A_k, f : \forall\,\gamma\ \delta\,.\,(A'_p, A'_k) \to \delta\,!\langle F_1\rangle \vdash c_f : M\ \alpha\,!\langle F_2\rangle$ (14) and $M\ A_1\,!\langle F_1\rangle \equiv M\ \alpha\,!\langle F_2\rangle$ (15). Facts 1, 6, 10 and 12, constructors SD-Abs and SD-Hand and Lemmas E.6 and E.7 give us that $\Gamma, \alpha, y : B_{\mathsf{sc}} \vdash h \star c_1 : M\ A_3\,!\langle F_2\rangle$ (16) Facts 1, 6, 10 and 12, constructors SD-Abs and SD-Hand and Lemma E.6 give us that $\Gamma, \alpha \vdash \boldsymbol{\lambda} z\,.\,h \star c_2 : A_3 \to M\ A_4\,!\langle F_2\rangle$ (17) Facts 7, 8, 9, 13, the fact that $\ell^{\mathsf{sc}} \notin \mathit{labels}\,(\mathit{oprs})$, constructors SD-Abs, SD-App and SD-Var and Lemma E.6 give us that $\Gamma, \alpha, (p', k') : \forall\,\gamma\ \delta\,.\,(B_{\mathsf{sc}} \to \gamma\,!\langle F_1\rangle, \gamma \to \delta\,!\langle F_1\rangle) \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,p'\ y)\ (z\,.\,k'\ z) : \delta\,!\langle F_1\rangle$ (18). Our goal then follows from facts 14, 15, 16, 17, 18 and Lemma E.4.

□

E.3. **Progress.**

Theorem 6.2 (Progress)**.** *If $\cdot \vdash c : \underline{C}$, then either:*

- *there exists a computation c' such that $c \rightsquigarrow c'$, or*
- *c is in a* normal form*, which means it is in one of the following forms: (1) $c = \mathbf{return}\ v$, (2) $c = \mathbf{op}\ \ell^{op}\ v\ (y\,.\,c')$, or (3) $c = \mathbf{sc}\ \ell^{sc}\ v\ (y\,.\,c_1)\ (z\,.\,c_2)$.*

Proof. By induction on the typing derivation $\cdot \vdash c : \underline{C}$.

- SD-App: Here, $\cdot \vdash v_1\ v_2$. Since v_1 has type $A \to B\,!\langle F\rangle$, by Lemma E.1 it must be of shape $\boldsymbol{\lambda} x\,.\,c$, which means we can step by rule E-AppAbs.
- SD-Do: Here, $\cdot \vdash \mathbf{do}\ x \leftarrow c_1\ \mathbf{in}\ c_2 : \underline{C}$. By the induction hypothesis, c_1 can either step (in which case we can step by E-Do), or it is a computation result. Every possible form has a corresponding reduction: if $c_1 = \mathbf{return}\ v$ we can step by E-DoRet, if

$c_1 = \textbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c)$ we can step by E-DoOp, and if $\textbf{sc}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c_1')\ (z\,.\,c_2')$ we can step by E-DoSc.

- SD-Let: Here, $\cdot \vdash \textbf{let}\ x = v\ \textbf{in}\ c : \underline{C}$, which means we can step by E-Let.
- SD-Ret, SD-Op, and SD-Sc: all of these are computation results (forms 1, 2, and 3, resp.).
- SD-Hand: Here $\cdot \vdash v \star c : M\ A!\langle F\rangle$. By Lemma E.1, v is of shape h. By the induction hypothesis, c can either step (in which case we can step by E-Hand), or it is in a normal form. Proceed by case split on the three forms.
  (1) Case $c = \textbf{return}\ v$. Since $\cdot \vdash h : \underline{C} \Rightarrow \underline{D}$, there must be some $(\textbf{return}\ x \mapsto c_r) \in h$ which means we can step by rule E-HandRet.
  (2) Case $c = \textbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c')$. Depending on $(\textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c) \in h$ we can step by E-HandOp or E-FwdOp.
  (3) Case $c = \textbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2)$. If $(\textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c) \in h$, we can step by E-HandSc. If not, since $\cdot \vdash h : \underline{C} \Rightarrow \underline{D}$, there must be some $(\textbf{fwd}\ f\ p\ k \mapsto c_f) \in h$ which means we can step by rule E-FwdSc.

□

$$
\begin{array}{rcl}
\text{value types } \widehat{A}, \widehat{B} & ::= & () \mid (\widehat{A}, \widehat{B}) \mid \widehat{A} \to \underline{\widehat{C}} \mid \underline{\widehat{C}} \Rightarrow \underline{\widehat{D}} \\
& \mid & \alpha \mid \widehat{\alpha} \mid \lambda\, \alpha\,.\, \widehat{A} \mid M\, \widehat{A} \\
\text{type schemes } \widehat{\sigma} & ::= & \widehat{A} \mid \forall\, \mu\,.\, \widehat{\sigma} \mid \forall\, \alpha\,.\, \widehat{\sigma} \mid \forall\, \widehat{\mu}\,.\, \widehat{\sigma} \mid \forall\, \widehat{\alpha}\,.\, \widehat{\sigma} \\
\text{computation types } \underline{\widehat{C}}, \underline{\widehat{D}} & ::= & \widehat{A}\,!\,\langle \widehat{E} \rangle \\
\text{effect rows } \widehat{E} & ::= & \cdot \mid \mu \mid \widehat{\mu} \mid \ell\,; \widehat{E} \\
\\
\text{signature contexts } \Sigma & ::= & \cdot \mid \Sigma, \ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}} \mid \ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}} \\
\text{type contexts } \widehat{\Gamma} & ::= & \cdot \mid \widehat{\Gamma}, x : \widehat{A} \mid \widehat{\Gamma}, \mu \mid \widehat{\Gamma}, \alpha \\
\text{unification worklists } U & ::= & \cdot \mid \widehat{A} \sim \widehat{B}, U \mid \widehat{E} \sim \widehat{F}, U \mid \underline{\widehat{C}} \sim \underline{\widehat{D}}, U
\end{array}
$$

Figure 15: Types for algorithmic system.

Appendix F. Type Inference

F.1. **Algorithmic Syntax.** For type inference, we follow the approach of Koka [Lei14]. The syntax for types in our algorithmic system can be found in Figure 15. We add unification variables $\widehat{\alpha}$ for types and $\widehat{\mu}$ for rows to our syntax. The hat on types $\widehat{A}, \underline{\widehat{C}}$, rows $\widehat{E}$ and contexts $\widehat{\Gamma}$ indicate that they may contain unification variables. Furthermore, we add unification worklists U to represent the collection of types or rows that need to be unified.

The type equivalence for algorithmic types is a trivial extension of type equivalence for the declarative system (Figure 7). The well-scopedness rules are also a trivial extension of Figure 9. Notice that we do not record unification variables in the contexts, so they are not checked in the well-scopedness rules. Finally, notice that we do not allow free type variables in the annotation of type operators M.

F.2. **Algorithmic Rules.** Our type inference algorithm consists of algorithmic typing rules that output a type $\widehat{A}$ and a substitution θ. The judgment $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta$ states that under $\widehat{\Gamma}$, a value v has algorithmic type $\widehat{A}$, giving a substitution θ. This algorithm is based on Hindley-Milner's algorithm W [Mil78], assigning types or unification variables to each (sub)term and generating substitutions by solving unification constraints, originating from algorithmic rules.

The algorithmic type inference rules can be found in Figure 16 for values, Figure 17 for computations, and Figure 18 for handlers. In these rules, we use unification $\widehat{A} \sim \widehat{B} : \theta$ which states that two types $\widehat{A}$ and $\widehat{B}$ can be unified, giving rise to a substitution θ. The same holds for the unification of rows $\widehat{E} \sim \widehat{F} : \theta$ and the unification of computation types $\underline{\widehat{C}} \sim \underline{\widehat{D}} : \theta$. We discuss the unification algorithm in more detail in Appendix F.3.

The most interesting case is the algorithmic handler rule. Here, we derive (unification) types for each of the subterms and require them to be a computation type with a type variable (e.g., $\widehat{A}_1 = \widehat{\alpha}_1$). Furthermore, all derived types of the subterms should be equivalent. We express this by unifying them (e.g., $\theta_2 \widehat{\alpha}_1\,!\,\langle \widehat{E}_1 \rangle \sim \widehat{\alpha}_2\,!\,\langle \widehat{E}_2 \rangle : \theta_3$). Then, we implicitly instantiate the resulting type using some fresh unification variable $\widehat{\alpha}_4$.

In these inference rules, we use $uv\,(\widehat{A})$ to represent the set of unification variables in $\widehat{A}$.

Furthermore, we use $rng\ (\theta)$ to indicate the range of the substitution θ.

$\boxed{\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta}$ Value typing

$$\frac{(x : \widehat{\sigma}) \in \widehat{\Gamma} \qquad \widehat{\sigma} \leqslant \widehat{A} \qquad \widehat{\Gamma} \vdash \widehat{A}}{\widehat{\Gamma} \vdash x : \widehat{A} \dashv \emptyset}\ \textsc{A-Var} \qquad \frac{}{\widehat{\Gamma} \vdash () : () \dashv \emptyset}\ \textsc{A-Unit}$$

$$\frac{\widehat{\Gamma} \vdash v_1 : \widehat{A} \dashv \theta_1 \qquad \theta_1\widehat{\Gamma} \vdash v_2 : \widehat{B} \dashv \theta_2}{\widehat{\Gamma} \vdash (v_1, v_2) : (\theta_2\widehat{A}, \widehat{B}) \dashv \theta_{2..1}}\ \textsc{A-Pair} \qquad \frac{\widehat{\alpha}\ \mathit{fresh} \qquad \widehat{\Gamma}, x : \widehat{\alpha} \vdash c : \underline{\widehat{C}} \dashv \theta}{\widehat{\Gamma} \vdash \boldsymbol{\lambda} x\,.\,c : \theta\widehat{\alpha} \to \underline{\widehat{C}} \dashv \theta}\ \textsc{A-Abs}$$

$$\frac{\begin{array}{c}\widehat{\Gamma} \vdash_M \ \mathbf{return}\ x \mapsto c_r : M\ \widehat{A}_1 ! \langle \widehat{E}_1 \rangle \dashv \theta_1 \\ \widehat{A}_1 = \widehat{\alpha}_1 \qquad \theta_1\widehat{\Gamma} \vdash_M \ \mathit{oprs} : M\ \widehat{A}_2 ! \langle \widehat{E}_2 \rangle \dashv \theta_2 \qquad \widehat{A}_2 = \widehat{\alpha}_2 \\ \theta_2(\widehat{\alpha}_1 ! \langle \widehat{E}_1 \rangle) \sim \widehat{\alpha}_2 ! \langle \widehat{E}_2 \rangle : \theta_3 \qquad \theta_{3..1}\widehat{\Gamma} \vdash_M \ \mathbf{fwd}\ f\ p\ k \mapsto c_f : M\ \widehat{A}_3 ! \langle \widehat{E}_3 \rangle \dashv \theta_4 \\ \widehat{A}_3 = \widehat{\alpha}_3 \qquad \theta_{4..3}(\widehat{\alpha}_2 ! \langle \widehat{E}_2 \rangle) \sim \widehat{\alpha}_3 ! \langle \widehat{E}_3 \rangle : \theta_5 \\ \widehat{\alpha}_4 = \theta_5\widehat{\alpha}_3 \qquad \widehat{\alpha}_4 \notin \mathit{fv}\,(\theta_{5..1}\widehat{\Gamma}) \qquad \langle \widehat{F} \rangle = \langle \mathit{labels}\,(\mathit{oprs})\,; \theta_5\ \widehat{E}_3 \rangle\end{array}}{\begin{array}{c}\widehat{\Gamma} \vdash \mathbf{handler}_M\ \{\mathbf{return}\ x \mapsto c_r, \mathit{oprs}, \mathbf{fwd}\ f\ p\ k \mapsto c_f\}: \\ \widehat{\alpha}_4 ! \langle \widehat{F} \rangle \Rightarrow M\ \widehat{\alpha}_4 ! \langle \theta_5\widehat{E}_3 \rangle \dashv \theta_{5..1}\end{array}}\ \textsc{A-Handler}$$

Figure 16: Type inference rules for values.

$\boxed{\widehat{\Gamma} \vdash c : \underline{\widehat{C}} \dashv \theta}$ Computation typing

$$\frac{\widehat{\Gamma} \vdash v_1 : \widehat{A}_1 \dashv \theta_1 \qquad \theta_1\widehat{\Gamma} \vdash v_2 : \widehat{A}_2 \dashv \theta_2 \qquad \widehat{\alpha}, \widehat{\mu}\ \textit{fresh} \qquad \theta_2\widehat{A}_1 \sim (\widehat{A}_2 \to \widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle) : \theta_3}{\widehat{\Gamma} \vdash v_1\ v_2 : \theta_3(\widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle) \dashv \theta_{3..1}}\ \text{A-App}$$

$$\frac{\widehat{\Gamma} \vdash c_1 : \widehat{A}\,!\,\langle\widehat{E}\rangle \dashv \theta_1 \qquad \theta_1\widehat{\Gamma}, x : \widehat{A} \vdash c_2 : \widehat{B}\,!\,\langle\widehat{F}\rangle \dashv \theta_2 \qquad \theta_2\widehat{E} \sim \widehat{F} : \theta_3}{\widehat{\Gamma} \vdash \mathbf{do}\ x \leftarrow c_1\,;\,c_2 : \theta_3(\widehat{B}\,!\,\langle\widehat{F}\rangle) \dashv \theta_{3..1}}\ \text{A-Do}$$

$$\frac{\begin{array}{c}\widehat{\Gamma}, \overline{\alpha}, \overline{\mu} \vdash v : \widehat{A} \dashv \theta_1 \qquad \overline{\alpha}, \overline{\mu} \notin \widehat{\Gamma} \qquad \overline{\widehat{\alpha}}\,\overline{\widehat{\mu}} = uv\,(\widehat{A}) - uv\,(\widehat{\Gamma}) \\ \theta_1\widehat{\Gamma}, x : \forall\,\overline{\alpha}\,.\,\forall\,\overline{\mu}\,.\,\forall\,\overline{\widehat{\alpha}}\,.\,\forall\,\overline{\widehat{\mu}}\,.\,\widehat{A} \vdash c : \underline{\widehat{C}} \dashv \theta_2\end{array}}{\widehat{\Gamma} \vdash \mathbf{let}\ x = v\ \mathbf{in}\ c : \underline{\widehat{C}} \dashv \theta_{2..1}}\ \text{A-Let}$$

$$\frac{\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta \qquad \widehat{\mu}\ \textit{fresh}}{\widehat{\Gamma} \vdash \mathbf{return}\ v : \widehat{A}\,!\,\langle\widehat{\mu}\rangle \dashv \theta}\ \text{A-Ret}$$

$$\frac{\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_1 \qquad \theta_1\widehat{\Gamma} \vdash c : \underline{\widehat{C}} \dashv \theta_2 \qquad \widehat{\alpha}, \widehat{\mu}\ \textit{fresh} \qquad \theta_2\widehat{A} \sim \underline{\widehat{C}} \Rightarrow \widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle : \theta_3}{\widehat{\Gamma} \vdash v \star c : \theta_3\widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle \dashv \theta_{3..1}}\ \text{A-Hand}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma \qquad \widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_1 \\ A_{\mathsf{op}} \sim \widehat{A} \dashv \theta_2 \qquad \theta_{2..1}\widehat{\Gamma}, y : B_{\mathsf{op}} \vdash c : \widehat{A}'\,!\,\langle\widehat{E}\rangle \dashv \theta_3 \qquad \widehat{\mu}\ \textit{fresh} \qquad \widehat{E} \sim \langle \ell^{\mathsf{op}}\,;\widehat{\mu}\rangle : \theta_4\end{array}}{\widehat{\Gamma} \vdash \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c) : \theta_4(\widehat{A}'\,!\,\langle\widehat{E}\rangle) \dashv \theta_{4..1}}\ \text{A-Op}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \in \Sigma \\ \widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_1 \qquad A_{\mathsf{sc}} \sim \widehat{A} : \theta_2 \qquad \theta_{2..1}\widehat{\Gamma}, y : B_{\mathsf{sc}} \vdash c_1 : \widehat{B}\,!\,\langle\widehat{E}\rangle \dashv \theta_3 \\ \widehat{\mu}\ \textit{fresh} \qquad \widehat{E} \sim \langle \ell^{\mathsf{sc}}\,;\widehat{\mu}\rangle : \theta_4 \qquad \theta_{4..1}\widehat{\Gamma}, z : \theta_4\widehat{B} \vdash c_2 : \widehat{A}'\,!\,\langle\widehat{F}\rangle \dashv \theta_5 \qquad \widehat{F} \sim \theta_{5..4}\widehat{E} : \theta_6\end{array}}{\widehat{\Gamma} \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : \theta_6\widehat{A}'\,!\,\langle\widehat{F}\rangle \dashv \theta_{6..1}}\ \text{A-Sc}$$

Figure 17: Type inference rules for computations.

$\boxed{\widehat{\Gamma} \vdash_M \textbf{return}\, x \mapsto c_r : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta}$ $\boxed{\widehat{\Gamma} \vdash_M oprs : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta}$

$\boxed{\widehat{\Gamma} \vdash_M \textbf{fwd}\, f\ p\ k \mapsto c : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta}$

Return, operation, and forwarding typing

$$\frac{\widehat{\alpha}, \widehat{\mu}\ \mathit{fresh} \qquad \widehat{\Gamma}, x : \widehat{\alpha} \vdash c_r : \underline{\widehat{C}} \dashv \theta_1 \qquad \underline{C} \sim \theta_1(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) : \theta_2}{\widehat{\Gamma} \vdash_M \textbf{return}\, x \mapsto c_r : \theta_{2..1}\ (M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_{2..1}}\ \text{A-Return}$$

$$\frac{\widehat{\alpha}, \widehat{\mu}\ \mathit{fresh}}{\widehat{\Gamma} \vdash_M \cdot : M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle \dashv \emptyset}\ \text{A-Empty}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma \qquad \widehat{\Gamma} \vdash_M oprs : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_1 \\ \theta_1\widehat{\Gamma}, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \widehat{A}!\langle\widehat{E}\rangle \vdash c : \underline{\widehat{C}} \dashv \theta_2 \qquad \theta_2(M\ \widehat{A}!\langle\widehat{E}\rangle) \sim \underline{\widehat{C}} : \theta_3\end{array}}{\widehat{\Gamma} \vdash_M \textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, oprs : \theta_{3..2}(M\ \widehat{A}!\langle\widehat{E}\rangle) \dashv \theta_{3..1}}\ \text{A-OprOp}$$

$$\frac{\begin{array}{c}(\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \in \Sigma \qquad \widehat{\Gamma} \vdash_M oprs : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_1 \\ \beta\ \mathit{fresh} \qquad \theta_1\widehat{\Gamma}, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta!\langle\widehat{E}\rangle, k : \beta \to M\ \widehat{A}!\langle\widehat{E}\rangle \vdash c : \underline{\widehat{C}} \dashv \theta_2 \\ \theta_2(M\ \widehat{A}!\langle\widehat{E}\rangle) \sim \underline{\widehat{C}} : \theta_3 \qquad \beta \notin \mathit{rng}\ (\theta_{3..1})\end{array}}{\widehat{\Gamma} \vdash_M \textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, oprs : \theta_{3..2}(M\ \widehat{A}!\langle\widehat{E}\rangle) \dashv \theta_{3..1}}\ \text{A-OprSc}$$

$$\frac{\begin{array}{c}\alpha, \beta, \gamma, \delta, \widehat{\alpha}, \widehat{\mu}\ \mathit{fresh} \\ \widehat{A}_p = \alpha \to M\ \beta!\langle\widehat{\mu}\rangle \qquad \widehat{A}'_p = \alpha \to \gamma!\langle\widehat{\mu}\rangle \qquad \widehat{A}_k = \beta \to M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle \\ \widehat{A}'_k = \gamma \to \delta!\langle\widehat{\mu}\rangle \qquad \widehat{\Gamma}, \alpha, \beta, p : \widehat{A}_p, k : \widehat{A}_k, f : \forall\, \gamma\, \delta\, .\, (\widehat{A}'_p, \widehat{A}'_k) \to \delta!\langle\widehat{\mu}\rangle \vdash c_f : \underline{\widehat{C}} \dashv \theta_1 \\ \theta_1(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \sim \underline{\widehat{C}} : \theta_2 \qquad \alpha, \beta, \gamma, \delta \notin \mathit{rng}\ (\theta_{2..1})\end{array}}{\widehat{\Gamma} \vdash_M \textbf{fwd}\, f\ p\ k \mapsto c_f : \theta_{2..1}(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_{2..1}}\ \text{A-Fwd}$$

Figure 18: Type inference rules for handlers.

$\boxed{\widehat{\sigma} \leqslant \widehat{A}}$ $\widehat{\sigma}$-instantiation

$$\frac{}{\widehat{A} \leqslant \widehat{A}}\ \widehat{\sigma}\text{-Inst-Base}$$

$$\frac{\widehat{\alpha}\ \mathit{fresh} \qquad [\widehat{\alpha} / \alpha]\, \widehat{\sigma} \leqslant \widehat{A}}{\forall\, \alpha\, .\, \widehat{\sigma} \leqslant \widehat{A}}\ \widehat{\sigma}\text{-Inst-}\widehat{\alpha} \qquad \frac{\widehat{\mu}\ \mathit{fresh} \qquad [\widehat{\mu} / \mu]\, \widehat{\sigma} \leqslant \widehat{A}}{\forall\, \mu\, .\, \widehat{\sigma} \leqslant \widehat{A}}\ \widehat{\sigma}\text{-Inst-}\widehat{\mu}$$

$$\frac{\widehat{\alpha}'\ \mathit{fresh} \qquad [\widehat{\alpha}' / \widehat{\alpha}]\, \widehat{\sigma} \leqslant \widehat{A}}{\forall\, \widehat{\alpha}\, .\, \widehat{\sigma} \leqslant \widehat{A}}\ \widehat{\sigma}\text{-Inst-}\widehat{\alpha}' \qquad \frac{\widehat{\mu}'\ \mathit{fresh} \qquad [\widehat{\mu}' / \widehat{\mu}]\, \widehat{\sigma} \leqslant \widehat{A}}{\forall\, \widehat{\mu}\, .\, \widehat{\sigma} \leqslant \widehat{A}}\ \widehat{\sigma}\text{-Inst-}\widehat{\mu}'$$

Figure 19: $\widehat{\sigma}$-instantiation.

F.3. **Unification Algorithm.** Our unification algorithm works in two steps, as shown in Figure 20.

First, we β-reduce the value types and computation types (Figure 21). Then, we unify types or rows that occur in the unification worklist step by step (Figure 22). The unification algorithm $\twoheadrightarrow$ transforms a unification worklist U_1 and an initial substitution θ_1 into a new unification worklist U_2 and substitution θ_2. It can be split in three parts. The first part deals with computation types. The unification of two computation types boils down to separately unifying their value types and row types. The second part concerns the unification of value types. We first pattern match on unit types, handler types, function types and tuples. Two (unification) type variables with the same name are considered equal. Unifying a unification variable with a type $\widehat{A}$ means substituting this unification variable by $\widehat{A}$ in the given substitution as well as in all occurrences in the remaining unification worklist.

The reasoning for unifying two rows is similar. The function $\mathsf{find}_\ell\ \widehat{E}$ extracts the label ℓ from $\widehat{E}$ and returns the remaining part:

$$\begin{aligned}
\mathsf{find}_\ell\ \langle\rangle &= \text{error} \\
\mathsf{find}_\ell\ \mu &= (\mu', [\langle \ell\,; \mu' \rangle \,/\, \mu]) \ \text{ where } \mu' \text{ is a fresh variable} \\
\mathsf{find}_\ell\ \langle \ell'\,; \widehat{E} \rangle &= (\widehat{E}, \emptyset) \qquad\quad \text{where } \ell = \ell' \\
\mathsf{find}_\ell\ \langle \ell'\,; \widehat{E} \rangle &= (\langle \ell'\,; \widehat{E}' \rangle, \theta) \ \text{ where } \ell \neq \ell' \ \text{ and } (\widehat{E}', \theta) = \mathsf{find}_\ell\ \widehat{E}
\end{aligned}$$

The function $\mathsf{tail}\ \widehat{E}$ returns the last row variable if $\widehat{E}$ is open, and returns $\langle\rangle$ if $\widehat{E}$ is closed.

$\boxed{\widehat{A} \sim \widehat{B} : \theta}$ $\boxed{\widehat{E} \sim \widehat{F} : \theta}$ $\boxed{\widehat{\underline{C}} \sim \widehat{\underline{D}} : \theta}$ Unification

$$\frac{\emptyset \vdash \mathrm{reduce}(\widehat{A}) \sim \mathrm{reduce}(\widehat{B}), \cdot \longrightarrow^* \theta \vdash \cdot}{\widehat{A} \sim \widehat{B} : \theta}\ \textsc{U-Val}$$

$$\frac{\emptyset \vdash \widehat{E} \sim \widehat{F}, \cdot \longrightarrow^* \theta \vdash \cdot}{\widehat{E} \sim \widehat{F} : \theta}\ \textsc{U-Row} \qquad \frac{\emptyset \vdash \mathrm{reduce}(\widehat{\underline{C}}) \sim \mathrm{reduce}(\widehat{\underline{D}}), \cdot \longrightarrow^* \theta \vdash \cdot}{\widehat{\underline{C}} \sim \widehat{\underline{D}} : \theta}\ \textsc{U-Comp}$$

Figure 20: Main unification judgments.

$\boxed{\mathrm{reduce}(\widehat{A}) = \widehat{B}}$ $\boxed{\mathrm{reduce}(\widehat{\underline{C}}) = \widehat{\underline{D}}}$ Type reduction

$$\begin{array}{rcl}
\mathrm{reduce}(()) & = & () \\
\mathrm{reduce}((\widehat{A}, \widehat{B})) & = & (\mathrm{reduce}(\widehat{A}), \mathrm{reduce}(\widehat{B})) \\
\mathrm{reduce}(\widehat{A} \to \widehat{\underline{C}}) & = & \mathrm{reduce}(\widehat{A}) \to \mathrm{reduce}(\widehat{\underline{C}}) \\
\mathrm{reduce}(\widehat{\underline{C}} \Rightarrow \widehat{\underline{D}}) & = & \mathrm{reduce}(\widehat{\underline{C}}) \Rightarrow \mathrm{reduce}(\widehat{\underline{D}}) \\
\mathrm{reduce}(\alpha) & = & \alpha \\
\mathrm{reduce}(\widehat{\alpha}) & = & \widehat{\alpha} \\
\mathrm{reduce}((\lambda\, \alpha \,.\, \widehat{A})\ \widehat{B}) & = & \mathrm{reduce}([\widehat{B} \,/\, \alpha]\ \widehat{A}) \\
\mathrm{reduce}(\widehat{A}!\langle\widehat{E}\rangle) & = & \mathrm{reduce}(\widehat{A})!\langle\widehat{E}\rangle
\end{array}$$

Figure 21: Reduction rules.

$\boxed{\theta_1 \vdash U_1 \longrightarrow \theta_2 \vdash U_2}$ Unification

$$\begin{array}{rcl}
\theta \vdash \widehat{A}_1 ! \langle \widehat{E}_1 \rangle \sim \widehat{A}_2 ! \langle \widehat{E}_2 \rangle, U & \longrightarrow & \theta \vdash \widehat{A}_1 \sim \widehat{A}_2, \widehat{E}_1 \sim \widehat{E}_2, U \\
\\
\theta \vdash () \sim (), U & \longrightarrow & \theta \vdash U \\
\theta \vdash \widehat{C}_1 \Rightarrow \widehat{D}_1 \sim \widehat{C}_2 \Rightarrow \widehat{D}_2, U & \longrightarrow & \theta \vdash \widehat{C}_1 \sim \widehat{C}_2, \widehat{D}_1 \sim \widehat{D}_2, U \\
\theta \vdash (\widehat{A}_1 \to \widehat{C}_1) \sim (\widehat{A}_2 \to \widehat{C}_2), U & \longrightarrow & \theta \vdash \widehat{A}_1 \sim \widehat{A}_2, \widehat{C}_1 \sim \widehat{C}_2, U \\
\theta \vdash (\widehat{A}_1, \widehat{A}_2) \sim (\widehat{A}_3, \widehat{A}_4), U & \longrightarrow & \theta \vdash \widehat{A}_1 \sim \widehat{A}_3, \widehat{A}_2 \sim \widehat{A}_4, U \\
\theta \vdash \alpha \sim \alpha, U & \longrightarrow & \theta \vdash U \\
\theta \vdash \widehat{\alpha} \sim \widehat{\alpha}, U & \longrightarrow & \theta \vdash U \\
\theta \vdash \widehat{\alpha} \sim \widehat{A}, U & \longrightarrow & [\widehat{A} \,/\, \widehat{\alpha}]\theta \vdash [\widehat{A} \,/\, \widehat{\alpha}]\ U \\
 & & \qquad \textsf{with } \widehat{\alpha} \ \notin \ \widehat{A} \\
\theta \vdash \widehat{A} \sim \widehat{\alpha}, U & \longrightarrow & [\widehat{A} \,/\, \widehat{\alpha}]\theta \vdash [\widehat{A} \,/\, \widehat{\alpha}]\ U \\
 & & \qquad \textsf{with } \widehat{\alpha} \ \notin \ \widehat{A} \\
\\
\theta \vdash \langle\rangle \sim \langle\rangle, U & \longrightarrow & \theta \vdash U \\
\theta \vdash \langle \ell \,;\, \widehat{E} \rangle \sim \widehat{F}, U & \longrightarrow & \theta'\theta \vdash \theta' U \\
 & & \qquad \textsf{with } (\_, \theta') = \textsf{find}_\ell\ \widehat{F} \textsf{ and tail } \widehat{E} \ \notin \ \theta' \\
\theta \vdash \widehat{F} \sim \langle \ell \,;\, \widehat{E} \rangle, U & \longrightarrow & \theta'\theta \vdash \theta' U \\
 & & \qquad \textsf{with } (\_, \theta') = \textsf{find}_\ell\ \widehat{F} \textsf{ and tail } \widehat{E} \ \notin \ \theta' \\
\theta \vdash \mu \sim \mu, U & \longrightarrow & \theta \vdash U \\
\theta \vdash \widehat{\mu} \sim \widehat{\mu}, U & \longrightarrow & \theta \vdash U \\
\theta \vdash \widehat{\mu} \sim \widehat{E}, U & \longrightarrow & [\widehat{E} \,/\, \widehat{\mu}]\theta \vdash [\widehat{E} \,/\, \widehat{\mu}]\ U \textsf{ with } \widehat{\mu} \ \notin \ \widehat{E} \\
\theta \vdash \widehat{E} \sim \widehat{\mu}, U & \longrightarrow & [\widehat{E} \,/\, \widehat{\mu}]\theta \vdash [\widehat{E} \,/\, \widehat{\mu}]\ U \textsf{ with } \widehat{\mu} \ \notin \ \widehat{E}
\end{array}$$

Figure 22: Unification rules.

F.4. **Lemmas for Proof.** In order to prove the soundness and completeness of our type inference algorithm with respect to the declarative type system, we require the following helper lemmas. Figure 23 displays instantiation of contexts. The following lemmas hold for this instantiation.

Lemma F.1 (Instantiation instantiates bindings)**.**
If $\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta$ *and* $x : \widehat{\sigma} \;\in\; \widehat{\Gamma}$ *then there exists some* $\sigma = \theta\widehat{\sigma}$ *such that* $x : \sigma \;\in\; \Gamma$*.*

Lemma F.2 (Inverse of Instantiation instantiates bindings)**.** *If* $\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta$ *and* $x : \sigma \;\in\; \Gamma$ *then there exists* $x : \widehat{\sigma} \;\in\; \widehat{\Gamma}$ *such that* $\sigma = \theta\widehat{\sigma}$*.*

The following lemma states that we can move substitutions from being instantiated to be applied to the context, and back.

Lemma F.3 (Moving subtitutions to instantiation)**.**
One can move instantiations θ_1 *from being applied to* $\widehat{\Gamma}$ *to being generated by environment instantiation and back. That is,* $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_2 \iff \widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_2\theta_1$*.*

The following lemma states that inferred substitutions are domain-restricted, i.e. they only substitute the unification variables in the context the and inferred type.

Lemma F.4 (Inferred substitutions are domain-restricted)**.**

- *For all* $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta$*,* $\theta = \theta|_{\widehat{\Gamma}}\theta|_{\widehat{A}}$*.*
- *For all* $\widehat{\Gamma} \vdash c : \underline{\widehat{C}} \dashv \theta$*,* $\theta = \theta|_{\widehat{\Gamma}}\theta|_{\underline{\widehat{C}}}$*.*

The notation $\theta|_{\widehat{\Gamma}}$ represents the substitution generated by removing all substitution of unification variables that are not in $\widehat{\Gamma}$ from θ. Similarly, the notation $\theta|_{\widehat{A}}$ ($\theta|_{\underline{\widehat{C}}}$) represents the substitution generated by removing all substitution of unification variables that are not in $\widehat{A}$ ($\underline{\widehat{C}}$) from θ.

The following lemma states that we can split a substitution with respect to its domain restriction.

Lemma F.5 (Substitution split w.r.t domain restriction)**.** *If* $\theta_1 = \theta_2|_{\widehat{\Gamma}}$*, then there exists* θ *such that* $\theta_2 = \theta\theta_1$*.*

The following lemma states that the unification algorithm is sound.

Lemma F.6 (Unification unifies)**.**
After unification, subtituted terms are equivalent. That is:

- $\widehat{A} \sim \widehat{B} : \theta \implies \theta\widehat{A} \;\equiv\; \theta\widehat{B}$
- $\underline{\widehat{C}} \sim \underline{\widehat{D}} : \theta \implies \theta\underline{\widehat{C}} \;\equiv\; \theta\underline{\widehat{D}}$
- $\underline{\underline{\widehat{E}}} \sim \underline{\underline{\widehat{F}}} : \theta \implies \theta\underline{\underline{\widehat{E}}} \;\equiv\; \theta\underline{\underline{\widehat{F}}}$

Dually, the following lemma states that unification gives a principal unifier.

Lemma F.7 (Unification gives a principal unifier)**.** *The unification algorithm gives a principal unifier. That is:*

- $\theta\widehat{A} \;\equiv\; \theta\widehat{B}$ *implies there exists* θ_1, θ_2 *such that* $\widehat{A} \sim \widehat{B} : \theta_1$*,* $\theta = \theta_2\theta_1$*.*
- $\theta\underline{\underline{\widehat{E}}} \;\equiv\; \theta\underline{\underline{\widehat{F}}}$ *implies there exists* θ_1, θ_2 *such that* $\underline{\underline{\widehat{E}}} \sim \underline{\underline{\widehat{F}}} : \theta_1$*,* $\theta = \theta_2\theta_1$*.*
- $\theta\underline{\widehat{C}} \;\equiv\; \theta\underline{\widehat{D}}$ *implies there exists* θ_1, θ_2 *such that* $\underline{\widehat{C}} \sim \underline{\widehat{D}} : \theta_1$*,* $\theta = \theta_2\theta_1$*.*

$\boxed{\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta}$ $\widehat{\Gamma}$-instantiation

$$\frac{}{\Gamma \rightsquigarrow \Gamma\,;\emptyset}\;\Gamma\text{-INST-BASE}$$

$$\frac{\Gamma, \overline{\alpha} \vdash A : * \qquad \widehat{\alpha} \in fv(\widehat{\Gamma}) \qquad \Gamma, \overline{\alpha}, [A \,/\, \widehat{\alpha}]\, \widehat{\Gamma} \rightsquigarrow \Gamma_{out}\,;\theta}{\Gamma, \widehat{\Gamma} \rightsquigarrow \Gamma_{out}\,; [A \,/\, \widehat{\alpha}]\theta}\;\Gamma\text{-INST-TYVAR}$$

Figure 23: Reduction rules.

The same holds for the instantiation algorithm. The following lemma states that it is sound and uses two helper lemmas to prove it.

Lemma F.8 (Algorithmic to declarative instantiation)**.**
For all $\widehat{\sigma} \leqslant \widehat{A}$ there exists a θ such that $\theta\widehat{\sigma} \leqslant \theta\,\widehat{A}$.

Proof. By induction on $\widehat{\sigma} \leqslant \widehat{A}$, using Lemmas F.9 and F.10. □

Lemma F.9 (Substitution/instantiation inlining)**.**
If $\alpha \notin \theta$ then $\theta[\widehat{\alpha} \,/\, \alpha]\, \widehat{\sigma} = [\theta\widehat{\alpha} \,/\, \alpha]\, \theta\widehat{\sigma}$.

Lemma F.10 (Substitution/instantiation inlining – Effects)**.**
If $\mu \notin \theta$ then $\theta[\widehat{\mu} \,/\, \mu]\, \widehat{\sigma} = [\theta\widehat{\mu} \,/\, \mu]\, \theta\widehat{\sigma}$.

The following lemma is essentially the dual of Lemma F.8.

Theorem F.11 (Declarative to algorithmic instantiation)**.** *For all $\sigma \leqslant A$ and $\theta\widehat{\sigma} = \sigma$, there exists $\widehat{A}$ and θ' such that $\widehat{\sigma} \leqslant \widehat{A}$ and $\theta'\theta\widehat{A} = A$*

Proof. Mutual induction on $\sigma \leqslant A$ and $\widehat{\sigma} \leqslant \widehat{A}$. □

F.5. **Soundness.** In words, soundness states that all algorithmic typing judgments have a declarative counterpart. For proving this property, we ignore the differences in the domain of substitutions, which implies that the notion of equality is implicitly restricted to the substitution domain.

Theorem F.12 (Soundness)**.** *All algorithmic typing judgments have a declarative counterpart:*

- $\forall\, \widehat{\Gamma}, v, \widehat{A}, \theta_{\mathsf{inf}}$: **if** $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_{\mathsf{inf}}$
  then $\forall\, \Gamma, \theta_{\mathsf{inst}}$: **if** $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ and $\forall\,\theta : \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\widehat{A} = A$
  then $\Gamma \vdash v : A$.
- $\forall\, \widehat{\Gamma}, c, \underline{\widehat{C}}, \theta_{\mathsf{inf}}$: **if** $\widehat{\Gamma} \vdash c : \underline{\widehat{C}} \dashv \theta_{\mathsf{inf}}$
  then $\forall\, \Gamma, \theta_{\mathsf{inst}}$: **if** $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ and $\forall\,\theta : \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\underline{\widehat{C}} = \underline{C}$
  then $\Gamma \vdash c : \underline{C}$.
- $\forall\, \widehat{\Gamma}, M, x, c_r, \widehat{A}, \widehat{E}, \theta_{\mathsf{inf}}$: **if** $\widehat{\Gamma} \vdash_M$ **return** $x \mapsto c_r : M\;\widehat{A}!\langle\widehat{E}\rangle \dashv \theta_{\mathsf{inf}}$
  then $\forall\, \Gamma, \theta_{\mathsf{inst}}$: **if** $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$
  and $\forall\,\theta : \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(M\;\widehat{A}!\langle\widehat{E}\rangle) = M\;A!\langle E\rangle$
  then $\Gamma \vdash$ **return** $x \mapsto c_r : M\;A!\langle E\rangle$.

- $\forall\, \widehat{\Gamma}, M, \mathit{oprs}, \widehat{A}, \widehat{E}, \theta_{\mathsf{inf}} :$ **if** $\widehat{\Gamma} \vdash_M\ \mathit{oprs} : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_{\mathsf{inf}}$
  then $\forall\, \Gamma, \theta_{\mathsf{inst}}$: **if** $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$
  and $\forall\, \theta : \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(M\ \widehat{A}!\langle\widehat{E}\rangle) = M\ A!\langle E\rangle$
  then $\Gamma \vdash \mathit{oprs} : M\ A!\langle E\rangle$.
- $\forall\, \widehat{\Gamma}, M, f, p, k, c, \widehat{A}, \widehat{E}, \theta_{\mathsf{inf}} :$ **if** $\widehat{\Gamma} \vdash_M$ **fwd** $f\ p\ k \mapsto c : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_{\mathsf{inf}}$
  then $\forall\, \Gamma, \theta_{\mathsf{inst}}$: **if** $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$
  and $\forall\, \theta : \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(M\ \widehat{A}!\langle\widehat{E}\rangle) = M\ A!\langle E\rangle$
  then $\Gamma \vdash$ **fwd** $f\ p\ k \mapsto c : M\ A!\langle E\rangle$.

Proof. We prove the statement by mutual induction on these judgments. In what follows we ignore the differences in the domain of substitutions, implicitly restricting the notion of equality to the substitution domain.

Value typing.

A-VAR We know that $\widehat{\Gamma} \vdash x : \widehat{A} \dashv \emptyset$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\, \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\widehat{A} = A$ for some A (2).

From Lemma F.1, (1) and $(x : \widehat{\sigma}) \in \widehat{\Gamma}$ (A-VAR), we have that $(x : \theta_{\mathsf{inst}}\widehat{\sigma}) \in \Gamma$.

From Lemma F.8, $\widehat{\sigma} \leqslant \widehat{A}$ (A-VAR) and (2), we have that $\theta\theta_{\mathsf{inst}}\widehat{\sigma} \leqslant \theta\theta_{\mathsf{inst}}\widehat{A}$.

As θ is applied after instantiation with θ_{inst}, it has no influence on $\widehat{\sigma}$. Thus $\theta\theta_{\mathsf{inst}}\widehat{\sigma} \equiv \theta_{\mathsf{inst}}\widehat{\sigma} \equiv \sigma$ (4).

Thus, we have that $(x : \sigma) \in \Gamma$ and $\sigma \leqslant A$ (2) (4) so that $\Gamma \vdash x : A$ (SD-VAR).

A-UNIT Trivial case.

A-PAIR We know that $\widehat{\Gamma} \vdash (v_1, v_2) : (\theta_2\widehat{A}, \widehat{B}) \dashv \theta_{2..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\theta_{2..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\, \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(\theta_2\widehat{A}, \widehat{B}) = (A, B)$ for some A, B (2).

From the *induction hypothesis for* v_1 we have the following:
if $\widehat{\Gamma} \vdash v_1 : \widehat{A} \dashv \theta_1$ (A-PAIR)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_2$ (Lemma F.3, (1)) and $\forall\, \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_2\widehat{A} = A$ (2)
then $\Gamma \vdash v_1 : A$.

From the *induction hypothesis for* v_2 we have the following:
if $\theta_1\widehat{\Gamma} \vdash v_2 : \widehat{B} \dashv \theta_2$ (A-PAIR)
then $\theta_{2..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (Lemma F.3, (1)) and $\forall\, \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\widehat{B} = B$ (2)
then $\Gamma \vdash v_2 : B$.

From (SD-PAIR) we conclude that $\Gamma \vdash (v_1, v_2) : (A, B)$.

A-ABS We know that $\widehat{\Gamma} \vdash \boldsymbol{\lambda} x\,.\,c : \theta_{\mathsf{inf}}\widehat{\alpha} \to \underline{\widehat{C}} \dashv \theta_{\mathsf{inf}}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\, \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(\theta_{\mathsf{inf}}\widehat{\alpha} \to \underline{\widehat{C}}) = A \to \underline{C}$ for some $A, \underline{C}$ (2).

From the *induction hypothesis for* c we have the following:
if $\widehat{\Gamma}, x : \widehat{\alpha} \vdash c : \underline{\widehat{C}} \dashv \theta_{\mathsf{inf}}$ (A-ABS)
then $\theta_{\mathsf{inf}}\widehat{\Gamma}, x : \widehat{\alpha} \rightsquigarrow \Gamma_1\,;\theta_{\mathsf{inst}}$ (Lemma F.3, (1)) and $\forall\, \theta\,.\,\Gamma_1 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\underline{\widehat{C}} = \underline{C}$ (2)

then $\Gamma_1 \vdash c : \underline{C}$.

From (Lemma F.3), (1) and (2), we know that $\Gamma_1 = \Gamma, x : \theta_{\mathsf{inst}}\widehat{\alpha} = \Gamma, x : A$, where $\theta_{\mathsf{inst}}\widehat{\alpha} = \theta\theta_{\mathsf{inst}}\theta_{\mathsf{inf}}\widehat{\alpha} = A$ because only θ_{inst} influences the instantiation of $\widehat{\alpha}$.

From (SD-Abs) we conclude that $\Gamma, x : A \vdash \boldsymbol{\lambda} x\,.\,c : A \to \underline{C}$.

A-Handler We know that $\widehat{\Gamma} \vdash \mathbf{handler}_M\ \{\mathbf{return}\ x \mapsto c_r, \mathit{oprs}, \mathbf{fwd}\ f\ \ p\ \ k \mapsto c_f\} : \widehat{\alpha}_4!\langle\widehat{F}\rangle \Rightarrow M\ \widehat{\alpha}_4!\langle\theta_5\widehat{E}_3\rangle \dashv \theta_{5..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{5..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(\widehat{\alpha}_4!\langle\widehat{F}\rangle \Rightarrow M\ \ \widehat{\alpha}_4!\langle\theta_5\widehat{E}_3\rangle) = \alpha!\langle F\rangle \Rightarrow M\ \alpha!\langle E\rangle$ for some M, α, E, F (2).

From the *induction hypothesis for* **fwd** $f\ \ p\ \ k \mapsto c_f$ we have the following:
if $\theta_{3..1}\widehat{\Gamma} \vdash_M\ \mathbf{fwd}\ f\ \ p\ \ k \mapsto c_f : M\ \widehat{\alpha}_3!\langle\widehat{E}_3\rangle \dashv \theta_4$ (A-Handler)
then $\theta_4\theta_{3..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_5$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_5(M\ \ \widehat{\alpha}_3!\langle\widehat{E}_3\rangle) = M\ \ \alpha!\langle E\rangle$ since (2) and $\widehat{\alpha}_4 = \theta_5\widehat{\alpha}_3$ (A-Hanlder)
then $\Gamma \vdash \mathbf{fwd}\ f\ \ p\ \ k \mapsto c_f : M\ \ \alpha!\langle E\rangle$.

From the *induction hypothesis for oprs* we have the following:
if $\theta_1\widehat{\Gamma} \vdash_M\ \mathit{oprs} : M\ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle \dashv \theta_2$ (A-Handler)
then $\theta_{2..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{5..3}$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{5..3}(M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle) = M\ \ \alpha!\langle E\rangle$ (3)
then $\Gamma \vdash \mathit{oprs} : M\ \ \alpha!\langle E\rangle$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_{5..3}(M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle)$
(Lemma F.6, A-Handler)
$= \theta\theta_{\mathsf{inst}}\theta_5(M\ \ \widehat{\alpha}_3!\langle\widehat{E}_3\rangle)$
(2), (A-Handler)
$= M\ \ \alpha!\langle E\rangle$

From the *induction hypothesis for* **return** $x \mapsto c_r$ we have the following:
if $\widehat{\Gamma} \vdash_M\ \mathbf{return}\ x \mapsto c_r : M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle \dashv \theta_1$ (A-Handler)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{5..2}$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{5..2}(M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle) = M\ \ \alpha!\langle E\rangle$ (4)
then $\Gamma \vdash \mathbf{return}\ x \mapsto c_r : M\ \ \alpha!\langle E\rangle$.

where (4): $\theta\theta_{\mathsf{inst}}\theta_{5..2}(M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle)$
(Lemma F.6, A-Handler)
$= \theta\theta_{\mathsf{inst}}\theta_{5..3}(M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle)$
(3)
$= M\ \ \alpha!\langle E\rangle$

Furthermore, we have that $\langle\widehat{F}\rangle = \langle \mathit{labels}\,(\mathit{oprs})\,;\theta_5\widehat{E}_3\rangle$ and since substitutions preserve equivalence also $F = \theta\theta_{\mathsf{inst}}\widehat{F}\ \equiv_{\langle\rangle}\ \mathit{labels}\,(\mathit{oprs})\,;\theta\theta_{\mathsf{inst}}\theta_5\widehat{E}_3 = \mathit{labels}\,(\mathit{oprs})\,;E$.

From (SD-Handler) we conclude that $\Gamma \vdash \textbf{handler}_M \; \{ \textbf{return}\; x \mapsto c_r, \mathit{oprs}, \textbf{fwd}\; f\; p\; k \mapsto c_f \} : A!\langle F \rangle \Rightarrow M\; A!\langle E \rangle$.

Computation typing.

A-App We know that $\widehat{\Gamma} \vdash v_1\; v_2 : \theta_3(\widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_{3..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\theta_{3..1}\widehat{\Gamma} \rightsquigarrow \Gamma\, ; \theta_{\mathsf{inst}}$ (1) and $\forall\, \theta \,.\, \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3(\widehat{\alpha}!\langle\widehat{\mu}\rangle) = \underline{C}$ for some $\underline{C}$ (2).

From the *induction hypothesis for* v_2 we have the following:
if $\theta_1\widehat{\Gamma} \vdash v_2 : \widehat{A}_2 \dashv \theta_2$ (A-App)
then $\theta_2\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\, ; \theta_{\mathsf{inst}}\theta_3$ (Lemma F.3, (1)) and $\forall\, \theta \,.\, \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3\widehat{A}_2 = A$ for some A (3)
then $\Gamma \vdash v_2 : A$.

From the *induction hypothesis for* v_1 we have the following:
if $\widehat{\Gamma} \vdash v_1 : \widehat{A}_1 \dashv \theta_1$ (A-App)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\, ; \theta_{\mathsf{inst}}\theta_{3..2}$ (Lemma F.3, (1)) and $\forall\, \theta \,.\, \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{A}_1 = A \to \underline{C}$ (4)
then $\Gamma \vdash v_1 : A \to \underline{C}$.

where (4): $\theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{A}_1$
(Lemma F.6, A-App)
$= \theta\theta_{\mathsf{inst}}\theta_3(\widehat{A}_2 \to \widehat{\alpha}!\langle\widehat{\mu}\rangle)$
$= \theta\theta_{\mathsf{inst}}\theta_3(\widehat{A}_2) \to \theta\theta_{\mathsf{inst}}\theta_3\widehat{\alpha}!\langle\widehat{\mu}\rangle$
(2), (3)
$= A \to \underline{C}$

From (SD-App) we conclude that $\Gamma \vdash v_1\; v_2 : \underline{C}$.

A-Let We know that $\widehat{\Gamma} \vdash \textbf{let}\; x = v\; \textbf{in}\; c : \underline{\widehat{C}} \dashv \theta_{2..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\theta_{2..1}\widehat{\Gamma} \rightsquigarrow \Gamma\, ; \theta_{\mathsf{inst}}$ (1) and $\forall\, \theta \,.\, \Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\underline{\widehat{C}} = \underline{C}$ for some $\underline{C}$ (2).

From the *induction hypothesis for* v we have the following:
if $\widehat{\Gamma}, \overline{\alpha}, \overline{\mu} \vdash v : \widehat{A} \dashv \theta_1$ with $\overline{\alpha}, \overline{\mu} \;\notin\; \widehat{\Gamma}$ (A-Let)
then $\theta_1\widehat{\Gamma}, \overline{\alpha}, \overline{\mu} \rightsquigarrow \Gamma_1\, ; \theta_{\mathsf{inst}}$ (Lemma F.3, (1)) where $\Gamma_1 = \Gamma, \theta_{\mathsf{inst}}\theta_{2..1}\overline{\alpha}, \theta_{\mathsf{inst}}\theta_{2..1}\overline{\mu} = \Gamma, \overline{\alpha}, \overline{\mu}$, because $\overline{\alpha}, \overline{\mu} \;\notin\; \theta_{2..1}$ and $\forall\, \theta \,.\, \Gamma_1 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_2\widehat{A} = A$ for some A (3)
then $\Gamma_1 \vdash v : A$.

From the *induction hypothesis for* c we have the following:
if $\theta_1\widehat{\Gamma}, x : \widehat{\sigma} \vdash c : \underline{\widehat{C}} \dashv \theta_2$ with $\widehat{\sigma} = \forall\, \overline{\alpha} \,.\, \forall\, \overline{\mu} \,.\, \forall\, \overline{\widehat{\alpha}} \,.\, \forall\, \overline{\widehat{\mu}} \,.\, \widehat{A}$ (A-Let)
then $\theta_2(\theta_1\widehat{\Gamma}, x : \widehat{\sigma}) \rightsquigarrow \Gamma_2\, ; \theta_{\mathsf{inst}})$ (Lemma F.3, (1)) and $\forall\, \theta \,.\, \Gamma_2 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\underline{\widehat{C}} = \underline{C}$ (2)
then $\Gamma \vdash c : \underline{C}$.

From (Lemma F.3), (1) and (3), we know that $\Gamma_2 = \Gamma, x : \theta_{\mathsf{inst}}\theta_2(\forall\, \overline{\alpha} \,.\, \forall\, \overline{\mu} \,.\, \forall\, \overline{\widehat{\alpha}} \,.\, \forall\, \overline{\widehat{\mu}} \,.\, \widehat{A}) = \Gamma, x : \forall\, \overline{\alpha} \,.\, \forall\, \overline{\mu} \,.\, A$, where $\mathit{dom}\,(\theta_{\mathsf{inst}}\theta_2) \;\cap\; (\mathit{uv}\,(\widehat{A}) \setminus \mathit{uv}\,(\theta_1\widehat{\Gamma})) = \emptyset$.

From (SD-Let) we conclude that $\Gamma \vdash \textbf{let}\; x = v\; \textbf{in}\; c : \underline{C}$.

A-Ret We know that $\widehat{\Gamma} \vdash \textbf{return}\ v : \widehat{A}\,!\,\langle\widehat{\mu}\rangle \dashv \theta_{\mathsf{inf}}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(\widehat{A}\,!\,\langle\widehat{\mu}\rangle) = A\,!\,\langle E\rangle$ for some A, E (2).

From the *induction hypothesis for* v we have the following:
if $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_{\mathsf{inf}}$ (A-Ret)
then $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\widehat{A} = A$ (2)
then $\Gamma \vdash v : A$.

From (SD-Ret) we conclude that $\Gamma \vdash \textbf{return}\ v : A\,!\,\langle E\rangle$.

A-Do We know that $\widehat{\Gamma} \vdash \textbf{do}\ x \leftarrow c_1\,;c_2 : \theta_3(\widehat{B}\,!\,\langle\widehat{F}\rangle) \dashv \theta_{3..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{3..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3(\widehat{B}\,!\,\langle\widehat{F}\rangle) = B\,!\,\langle E\rangle$ for some B, E (2).

From the *induction hypothesis for* c_1 we have the following:
if $\widehat{\Gamma} \vdash c_1 : \widehat{A}\,!\,\langle\widehat{E}\rangle \dashv \theta_1$ (A-Do)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{3..2}$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}(\widehat{A}\,!\,\langle\widehat{E}\rangle) = A\,!\,\langle E\rangle$ (3)
then $\Gamma \vdash c_1 : A\,!\,\langle E\rangle$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_{3..2}(\widehat{A}\,!\,\langle\widehat{E}\rangle)$
$= \theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{A}\,!\,\langle\theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{E}\rangle$
$= A\,!\,\langle\theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{E}\rangle$ for some A (4)
(Lemma F.6, A-Do)
$= A\,!\,\langle\theta\theta_{\mathsf{inst}}\theta_3\widehat{F}\rangle$
(2)
$= A\,!\,\langle E\rangle$

From the *induction hypothesis for* c_2 we have the following:
if $\theta_1\widehat{\Gamma}, x : \widehat{A} \vdash c_2 : \widehat{B}\,!\,\langle\widehat{F}\rangle \dashv \theta_2$ (A-Do)
then $\theta_2(\theta_1\widehat{\Gamma}, x : \widehat{A}) \rightsquigarrow \Gamma_2\,;\theta_{\mathsf{inst}}\theta_3$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma_2 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3(\widehat{B}\,!\,\langle\widehat{F}\rangle) = B\,!\,\langle E\rangle$ (2)
then $\Gamma_2 \vdash c_2 : B\,!\,\langle E\rangle$.

From (Lemma F.3), (1) and (4), we know that $\Gamma_2 = \Gamma, x : \theta_{\mathsf{inst}}\theta_{3..2}\widehat{A} = \Gamma, x : A$.

From (SD-Do) we conclude that $\Gamma \vdash \textbf{do}\ x \leftarrow c_1\,;c_2 : B\,!\,\langle E\rangle$.

A-Op We know that $\widehat{\Gamma} \vdash \textbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c) : \theta_4(\widehat{A}'\,!\,\langle\widehat{E}\rangle) \dashv \theta_{4..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{4..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_4(\widehat{A}'\,!\,\langle\widehat{E}\rangle) = A\,!\,\langle E\rangle$ for some A, E (2).

From the *induction hypothesis for* v we have the following:
if $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_1$ (A-Op)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{4..2}$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{4..2}\widehat{A} = A_{\mathsf{op}}$ (3)
then $\Gamma \vdash v : A_{\mathsf{op}}$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_{4..2}\widehat{A}$
(Lemma F.6, A-Op)
$= \theta\theta_{\mathsf{inst}}\theta_{4..2}(A_{\mathsf{op}})$
$= A_{\mathsf{op}}$

From the *induction hypothesis for* c we have the following:
if $\theta_2\widehat{\Gamma}, y : B_{\mathsf{op}} \vdash c : \widehat{A}'\,!\,\langle\widehat{E}\rangle \dashv \theta_3$ (A-Op)
then $\theta_3(\theta_{2..1}\widehat{\Gamma}, y : B_{\mathsf{op}}) \rightsquigarrow \Gamma, y : B_{\mathsf{op}}\,;\theta_{\mathsf{inst}}\theta_4$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma, y : B_{\mathsf{op}} \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_4(\widehat{A}'\,!\,\langle\widehat{E}\rangle) = A\,!\,\langle E\rangle$ (2)
then $\Gamma, y : B_{\mathsf{op}} \vdash c : A\,!\,\langle E\rangle$.

Furthermore, from (A-Op) follows that $(\ell^{\mathsf{op}} : A_{\mathsf{op}} \twoheadrightarrow B_{\mathsf{op}}) \in \Sigma$. Following (A-Op) and Lemma F.6, we have that $\theta_4\langle\widehat{E}\rangle \equiv \theta_4\langle\ell^{\mathsf{op}}\,;\widehat{\mu}\rangle$. Since substitutions preserve equivalence, we also have that $E = \theta\theta_{\mathsf{inst}}\theta_4\widehat{E} \equiv_{\langle\rangle} \theta\theta_{\mathsf{inst}}\theta_4(\ell^{\mathsf{op}}\,;\widehat{\mu}) = \ell^{\mathsf{op}}, E'$ for some E' so that $\ell^{\mathsf{op}} \in E$.

From (SD-Op) we conclude that $\Gamma \vdash \mathbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c) : A\,!\,\langle E\rangle$.

A-Sc We know that $\widehat{\Gamma} \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : \theta_6(\widehat{A}'\,!\,\langle\widehat{F}\rangle) \dashv \theta_{6..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{6..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_6(\widehat{A}'\,!\,\langle\widehat{F}\rangle) = A\,!\,\langle E\rangle$ for some A, E (2).

From the *induction hypothesis for* v we have the following:
if $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_1$ (A-Sc)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{6..2}$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{6..2}\widehat{A} = A_{\mathsf{sc}}$ (3)
then $\Gamma \vdash v : A_{\mathsf{sc}}$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_{6..2}\widehat{A}$
(Lemma F.6, A-Sc)
$= \theta\theta_{\mathsf{inst}}\theta_{6..2}(A_{\mathsf{sc}})$
$= A_{\mathsf{sc}}$

From the *induction hypothesis for* c_1 we have the following:
if $\theta_{2..1}\widehat{\Gamma}, y : B_{\mathsf{sc}} \vdash c_1 : \widehat{B}\,!\,\langle\widehat{E}\rangle \dashv \theta_3$ (A-Sc)
then $\theta_3(\theta_{2..1}\widehat{\Gamma}, y : B_{\mathsf{sc}}) \rightsquigarrow \Gamma, y : B_{\mathsf{sc}}\,;\theta_{\mathsf{inst}}\theta_{6..4}$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma, y : B_{\mathsf{sc}} \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{6..4}(\widehat{B}\,!\,\langle\widehat{E}\rangle) = B\,!\,\langle E\rangle$ (4)
then $\Gamma, y : B_{\mathsf{sc}} \vdash c_1 : B\,!\,\langle E\rangle$.

where (4): $\theta\theta_{\mathsf{inst}}\theta_{6..4}(\widehat{B}\,!\,\langle\widehat{E}\rangle)$
$= \theta\theta_{\mathsf{inst}}\theta_{6..4}\widehat{B}\,!\,\langle\theta\theta_{\mathsf{inst}}\theta_{6..4}\widehat{E}\rangle$
$= B\,!\,\langle\theta\theta_{\mathsf{inst}}\theta_{6..4}\widehat{E}\rangle$ for some B (5)
(Lemma F.6, A-Sc)
$= B\,!\,\langle\theta\theta_{\mathsf{inst}}\theta_6\widehat{F}\rangle$
(2)
$= B\,!\,\langle E\rangle$

From the *induction hypothesis for* c_2 we have the following:
if $\theta_{4..1}\widehat{\Gamma}, z : \theta_4\widehat{B} \vdash c_2 : \widehat{A}'!\langle\widehat{F}\rangle \dashv \theta_5$ (A-Sc)
then $\theta_5(\theta_{4..1}\widehat{\Gamma}, z{:}\theta_4\widehat{B}) \rightsquigarrow \Gamma_3\,;\theta_{\mathsf{inst}}\theta_6$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma_3 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_6(\widehat{A}'!\langle\widehat{F}\rangle) = A!\langle E\rangle$ (2)
then $\Gamma_3 \vdash c_2 : A!\langle E\rangle$.

From (Lemma F.3), (1) and (5), we know that $\Gamma_3 = \Gamma, z : \theta_{\mathsf{inst}}\theta_{6..4}\widehat{B} = \Gamma, z : B$.

Furthermore, from (A-Sc) follows that $(\ell^{\mathsf{sc}} : A_{\mathsf{sc}} \twoheadrightarrow B_{\mathsf{sc}}) \in \Sigma$. Following (A-Sc) and Lemma F.6, we have that $\theta_4\langle\widehat{E}\rangle \equiv \theta_4\langle\ell^{\mathsf{sc}}\,;\widehat{\mu}\rangle$. Since substitutions preserve equivalence, we also have that $E = \theta\theta_{\mathsf{inst}}\theta_{6..4}\widehat{E} \equiv_{\langle\rangle} \theta\theta_{\mathsf{inst}}\theta_{6..4}(\ell^{\mathsf{sc}}\,;\widehat{\mu}) = \ell^{\mathsf{sc}}, E'$ for some E' so that $\ell^{\mathsf{sc}} \in E$.

From (SD-Sc) we conclude that $\Gamma \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : A!\langle E\rangle$.

A-Hand We know that $\widehat{\Gamma} \vdash v \star c : \theta_3(\widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_{3..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\theta_{3..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3(\widehat{\alpha}!\langle\widehat{\mu}\rangle) = \underline{D}$ for some $\underline{D}$ (2).

From the *induction hypothesis for* c we have the following:
if $\theta_1\widehat{\Gamma} \vdash c : \widehat{\underline{C}} \dashv \theta_2$ (A-Hand)
then $\theta_{2..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_3$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3\widehat{\underline{C}} = \underline{C}$ for some $\underline{C}$ (3)
then $\Gamma \vdash c : \underline{C}$.

From the *induction hypothesis for* v we have the following:
if $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_1$ (A-Hand)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{3..2}$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{A} = \underline{C} \Rightarrow \underline{D}$ (4)
then $\Gamma \vdash v : \underline{C} \Rightarrow \underline{D}$.

where (4): $\theta\theta_{\mathsf{inst}}\theta_{3..2}\widehat{A}$
(Lemma F.6, A-App)
$= \theta\theta_{\mathsf{inst}}\theta_3(\widehat{\underline{C}} \Rightarrow \widehat{\alpha}!\langle\widehat{\mu}\rangle)$
$= \theta\theta_{\mathsf{inst}}\theta_3(\widehat{\underline{C}}) \Rightarrow \theta\theta_{\mathsf{inst}}\theta_3\widehat{\alpha}!\langle\widehat{\mu}\rangle$
(2), (3)
$= \underline{C} \Rightarrow \underline{D}$

From (SD-Hand) we conclude that $\Gamma \vdash v \star c : \underline{D}$.

Return clause.

A-Return We know that $\widehat{\Gamma} \vdash_M \mathbf{return}\ x \mapsto c_r : \theta_{2..1}(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_{2..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{2..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{2..1}(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) = M\ A!\langle E\rangle$ for some M, A, E (2).

From the *induction hypothesis for* c_r we have the following:
if $\widehat{\Gamma}, x : \widehat{\alpha} \vdash c_r : \widehat{\underline{C}} \dashv \theta_1$ (A-Abs)

then $\theta_1(\widehat{\Gamma}, x{:}\widehat{\alpha}) \rightsquigarrow \Gamma_1 \,;\theta_{\mathsf{inst}}\theta_2$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma_1 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_2\underline{\widehat{C}} = M\ \ A!\langle E\rangle$ (3)
then $\Gamma_1 \vdash c_r : M\ \ A!\langle E\rangle$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_2\underline{\widehat{C}}$
(Lemma F.6, A-Return)
$= \theta\theta_{\mathsf{inst}}\theta_{2..1}(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle)$
(2)
$= M\ \ A!\langle E\rangle$

From (Lemma F.3), (1) and (2), we know that $\Gamma_1 = \Gamma, x : \theta_{\mathsf{inst}}\theta_{2..1}\widehat{\alpha} = \Gamma, x : A$, where $\theta_{\mathsf{inst}}\theta_{2..1}\widehat{\alpha} = \theta\theta_{\mathsf{inst}}\theta_{2..1}\widehat{\alpha} = A$ because θ does not influence the instantiation of $\widehat{\alpha}$.

From (SD-Return) we conclude that $\Gamma \vdash \textbf{return}\ x \mapsto c_r : M\ \ A!\langle E\rangle$.

Operation clauses.

A-Empty Trivial case:
We know that $\widehat{\Gamma} \vdash_M \ \cdot : M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle \dashv \emptyset$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ and $\forall\,\theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) = M\ \ A!\langle E\rangle$ for some M, A, E.

From (SD-Empty) we conclude that $\Gamma \vdash \cdot : M\ \ A!\langle E\rangle$.

A-OprOp We know that $\widehat{\Gamma} \vdash_M\ \textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs} : \theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle) \dashv \theta_{3..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{3..1}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle) = M\ \ A!\langle E\rangle$ for some M, A, E (2).

From the *induction hypothesis for oprs* we have the following:
if $\widehat{\Gamma} \vdash_M\ \mathit{oprs} : M\ \ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_1$ (A-OprOp)
then $\theta_1\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}\theta_{3..2}$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}M\ \ \widehat{A}!\langle\widehat{E}\rangle = M\ \ A!\langle E\rangle$ (2)
then $\Gamma \vdash \mathit{oprs} : M\ \ A!\langle E\rangle$.

From the *induction hypothesis for c* we have the following:
if $\theta_1\widehat{\Gamma}, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \ \widehat{A}!\langle\widehat{E}\rangle \vdash c : \underline{C} \dashv \theta_2$ (A-OprOp)
then $\theta_2(\theta_1\widehat{\Gamma}, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \ \widehat{A}!\langle\widehat{E}\rangle) \rightsquigarrow \Gamma_2\,;\theta_{\mathsf{inst}}\theta_3$ (Lemma F.3, (1)) and $\forall\,\theta\,.\,\Gamma_2 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3\underline{C} = M\ \ A!\langle E\rangle$ (3)
then $\Gamma_2 \vdash c : M\ \ A!\langle E\rangle$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_3\underline{\widehat{C}}$
(Lemma F.6, A-OprOp)
$= \theta\theta_{\mathsf{inst}}\theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle)$
(2)
$= M\ \ A!\langle E\rangle$

From (Lemma F.3), (1) and (2), we know that $\Gamma_2 = \Gamma, x{:}A_{\mathsf{op}}, k{:}B_{\mathsf{op}} \to \theta_{\mathsf{inst}}\theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle)$ $= \Gamma, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \ A!\langle E\rangle$.

From (SD-OprOp) we conclude that $\Gamma \vdash \textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs} : M\ \ A!\langle E\rangle$.

A-OprSc We know that $\widehat{\Gamma}\ \vdash_M\ \textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, \mathit{oprs} : \theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle) \dashv \theta_{3..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{3..1}\widehat{\Gamma} \leadsto \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle) = M\ \ A!\langle E\rangle$ for some M, A, E (2).

From the *induction hypothesis for oprs* we have the following:
if $\widehat{\Gamma} \vdash_M\ \mathit{oprs} : M\ \ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_1$ (A-OprOp)
then $\theta_1\widehat{\Gamma} \leadsto \Gamma\,;\theta_{\mathsf{inst}}\theta_{3..2}$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{3..2}M\ \ \widehat{A}!\langle\widehat{E}\rangle = M\ \ A!\langle E\rangle$ (2)
then $\Gamma \vdash \mathit{oprs} : M\ \ A!\langle E\rangle$.

From the *induction hypothesis for c* we have the following:
if $\theta_1\widehat{\Gamma}, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \ \beta!\langle E\rangle, k : \beta \to M\ \ \widehat{A}!\langle\widehat{E}\rangle \vdash c : \underline{C} \dashv \theta_2$ (A-OprOp)
then $\theta_2(\theta_1\widehat{\Gamma}, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \ \beta!\langle E\rangle, k : \beta \to M\ \ \widehat{A}!\langle\widehat{E}\rangle) \leadsto \Gamma_2\,;\theta_{\mathsf{inst}}\theta_3$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma_2 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_3\underline{C} = M\ \ A!\langle E\rangle$ (3)
then $\Gamma_2 \vdash c : M\ \ A!\langle E\rangle$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_3\widehat{\underline{C}}$
(Lemma F.6, A-OprSc)
$= \theta\theta_{\mathsf{inst}}\theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle)$
(2)
$= M\ \ A!\langle E\rangle$

From (Lemma F.3), (1) and (2), we know that $\Gamma_2 = \Gamma, \beta, x{:}A_{\mathsf{sc}}, p{:}B_{\mathsf{sc}} \to M\ \ \beta!\langle\widehat{E}\rangle, k{:}\beta \to \theta_{\mathsf{inst}}\theta_{3..2}(M\ \ \widehat{A}!\langle\widehat{E}\rangle) = \Gamma, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \ \beta!\langle E\rangle, k : \beta \to M\ \ A!\langle E\rangle$ because $\beta\ \notin\ \theta_{3..2}$.

From (SD-OprSc) we conclude that $\Gamma \vdash \textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, \mathit{oprs} : M\ \ A!\langle E\rangle$.

Forwarding clause.

A-Fwd We know that $\widehat{\Gamma}\ \vdash_M\ \textbf{fwd}\ f\ p\ k \mapsto c_f : \theta_{2..1}(M\ \ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_{2..1}$ so we can choose $\Gamma, \theta_{\mathsf{inst}}$ such that
$\theta_{2..1}\widehat{\Gamma} \leadsto \Gamma\,;\theta_{\mathsf{inst}}$ (1) and $\forall\ \theta\,.\,\Gamma \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_{2..1}(M\ \ \widehat{\alpha}!\langle\widehat{\mu}\rangle) = M\ \ A!\langle E\rangle$ for some M, A, E (2).

From the *induction hypothesis for c_f* we have the following:
if $\widehat{\Gamma}, \alpha, \beta, p : \widehat{A}_p, k : \widehat{A}_k, f : \forall\ \gamma\ \delta\,.\,(\widehat{A}'_p, \widehat{A}'_k) \to \delta!\langle\widehat{\mu}\rangle) \vdash c_f : \underline{C} \dashv \theta_1$ (A-OprOp)
then $\theta_1(\widehat{\Gamma}, \alpha, \beta, p : \widehat{A}_p, k : \widehat{A}_k, f : \forall\ \gamma\ \delta\,.\,(\widehat{A}'_p, \widehat{A}'_k) \to \delta!\langle\widehat{\mu}\rangle) \leadsto \Gamma_1\,;\theta_{\mathsf{inst}}\theta_2$ (Lemma F.3, (1)) and $\forall\ \theta\,.\,\Gamma_1 \vdash \theta$ and $\theta\theta_{\mathsf{inst}}\theta_2\underline{C} = M\ \ A!\langle E\rangle$ (3)
then $\Gamma_1 \vdash c_f : M\ \ A!\langle E\rangle$.

where (3): $\theta\theta_{\mathsf{inst}}\theta_2\widehat{\underline{C}}$
(Lemma F.6, A-Fwd)
$= \theta\theta_{\mathsf{inst}}\theta_{2..1}(M\ \ \widehat{\alpha}!\langle\widehat{\mu}\rangle)$
(2)

$= M\ \ A!\langle E\rangle$

From (Lemma F.3), (1) and (2), we know that $\Gamma_1 = \Gamma, \theta_{\mathsf{inst}}\theta_{2..1}\alpha, \theta_{\mathsf{inst}}\theta_{2..1}\beta\ p{:}\theta_{\mathsf{inst}}\theta_{2..1}\widehat{A}_p, k{:}$ $\theta_{\mathsf{inst}}\theta_{2..1}\widehat{A}_k, f : \forall\, \gamma\ \delta\, .\, (\theta_{\mathsf{inst}}\theta_{2..1}\widehat{A}'_p, \theta_{\mathsf{inst}}\theta_{2..1}\widehat{A}'_k) \to \theta_{\mathsf{inst}}\theta_{2..1}(\delta\,!\,\langle\widehat{\mu}\rangle) = \Gamma, \alpha, \beta, p : A_p, k : A_k, f :$ $\forall\, \gamma\ \delta\, .\, (A'_p, A'_k) \to M\ \ A!\langle E\rangle$.

From (SD-Fwd) we conclude that $\Gamma \vdash \mathbf{fwd}\ f\ \ p\ \ k \mapsto c_f : M\ \ A!\langle E\rangle$.

$\square$

F.6. **Completeness.** Completeness states that all declarative typing judgments have a algorithmic counterpart.

Theorem F.13 (Completeness)**.** *All declarative typing judgments have an algorithmic counterpart:*

- *If* $\Gamma \vdash v : A$ *then for all* $\widehat{\Gamma}$ *and* θ_a*, if* $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$ *then there exists* $\widehat{A}, \theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ *such that* $\theta_a = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$*,* $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_{\mathsf{inf}}$*, and* $\theta_d\theta_{\mathsf{inst}}\widehat{A} = A$*.*
- *If* $\Gamma \vdash c : \underline{C}$ *then for all* $\widehat{\Gamma}$ *and* θ_a*, if* $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$ *then there exists* $\widehat{A}, \theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ *such that* $\theta_a = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$*,* $\widehat{\Gamma} \vdash c : \underline{\widehat{C}} \dashv \theta_{\mathsf{inf}}$*, and* $\theta_d\theta_{\mathsf{inst}}\underline{\widehat{C}} = \underline{C}$*.*
- *If* $\Gamma \vdash \mathbf{return}\ x \mapsto c_r : M\ \ A!\langle E\rangle$ *then for all* $\widehat{\Gamma}$ *and* θ_a*, if* $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$ *then there exists* $\widehat{A}, \theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ *such that* $\theta_a = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$*,* $\widehat{\Gamma} \vdash_M \mathbf{return}\ x \mapsto c_r : M\ \ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_{\mathsf{inf}}$*, and* $\theta_d\theta_{\mathsf{inst}}M\ \ \widehat{A}!\langle\widehat{E}\rangle = M\ \ A!\langle E\rangle$*.*
- *If* $\Gamma \vdash oprs : M\ \ A!\langle E\rangle$ *then for all* $\widehat{\Gamma}$ *and* θ_a*, if* $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$ *then there exists* $\widehat{A}, \theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ *such that* $\theta_a = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$*,* $\widehat{\Gamma} \vdash_M oprs : M\ \ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_{\mathsf{inf}}$*, and* $\theta_d\theta_{\mathsf{inst}}M\ \ \widehat{A}!\langle\widehat{E}\rangle = M\ \ A!\langle E\rangle$*.*
- *If* $\Gamma \vdash \mathbf{fwd}\ f\ \ p\ \ k \mapsto c_f : M\ \ A!\langle E\rangle$ *then for all* $\widehat{\Gamma}$ *and* θ_a*, if* $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$ *then there exists* $\widehat{A}, \theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ *such that* $\theta_a = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$*,* $\widehat{\Gamma} \vdash_M \mathbf{fwd}\ f\ \ p\ \ k \mapsto c_f : M\ \ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_{\mathsf{inf}}$*, and* $\theta_d\theta_{\mathsf{inst}}M\ \ \widehat{A}!\langle\widehat{E}\rangle = M\ \ A!\langle E\rangle$*.*

Proof. Note that by Lemma F.3, $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$, and $\theta_a = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$, we can derive that $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$. In the following proof, we treat it as part of the conclusion of the theorem for simplicity. Also, we do not deal with the equivalence relations of types explicitly. All comparisons between types are considered to use equivalence relations implicitly.

Prove by mutual induction on these judgments.

Value typing.

SD-Var For any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$, let $\theta_{\mathsf{inst}} = \theta_a$ and $\theta_{\mathsf{inf}} = \emptyset$. By Lemma F.2 and $(x : \sigma) \in \Gamma$ we get that there exists $(x : \widehat{\sigma}) \in \widehat{\Gamma}$ such that $\theta_a\widehat{\sigma} = \sigma$. By Theorem F.11 and $\Gamma \vdash \theta_a\widehat{\sigma} \leqslant A$, there exists θ_d and $\widehat{A}$ such that $\theta_d\theta_a\widehat{A} = A = \theta_d\theta_{\mathsf{inst}}\widehat{A}$ and $\widehat{\sigma} \leqslant \widehat{A}$. Then, $\Gamma \vdash A$ gives us $\widehat{\Gamma} \vdash \widehat{A}$. Thus, by A-Var, we have $\widehat{\Gamma} \vdash x : \widehat{A} \dashv \theta_{\mathsf{inf}}$. We also have $\theta_a = \theta_{\mathsf{inst}}\theta_{\mathsf{inf}}$ and $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$. It is easy to check the conclusion is satisfied given $\theta_a, \theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ defined as above.

SD-Unit Let $\theta_{\mathsf{inst}} = \theta_a$ and $\theta_d = \theta_{\mathsf{inf}} = \emptyset$. The conclusion follows from the assumptions.

SD-Pair By the IH on v_1, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}(\widehat{\Gamma})$, $\widehat{\Gamma} \vdash v_1 : \widehat{A} \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$ (1), and $\theta_4\theta_3\widehat{A} = A$.

Then we apply the IH on v_2 to (1), there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}}$, $\theta_2\widehat{\Gamma} \vdash v_2 : \widehat{B} \dashv \theta_6$, $\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_7$, $\theta_8\theta_7\widehat{B} = B$.

By A-Pair, we have $\widehat{\Gamma} \vdash (v_1, v_2) : (\theta_6\widehat{A}, \widehat{B}) \dashv \theta_6\theta_2$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_7$, and $\theta_d = \theta_4\theta_8$, the conclusion is satisfied:

– $\theta_1 = \theta_7\theta_6\theta_2|_{\widehat{\Gamma}}$
– $\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_7$
– $\theta_4\theta_8\theta_7(\theta_6\widehat{A}, \widehat{B}) = (A, B)$ (the new unification variables in $\widehat{A}$ and $\widehat{B}$ does not overlap)

SD-Abs By IH on the function body c, for any $\widehat{\Gamma}, x : \widehat{A} \rightsquigarrow \Gamma, x : A\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}\ (\widehat{\Gamma})$, $\widehat{\Gamma}, x : \widehat{A} \vdash c : \underline{\widehat{C}} \dashv \theta_2$, $\theta_2(\widehat{\Gamma}, x : \widehat{A}) \rightsquigarrow \Gamma, x : A\,;\theta_3$, and $\theta_4\theta_3\underline{\widehat{C}} = \underline{C}$.

Let the arbitrary type $\widehat{A}$ be an fresh unification variable $\widehat{\alpha}$, we have $\widehat{\Gamma}, x : \widehat{\alpha} \vdash c : \underline{\widehat{C}} \dashv \theta_2$. By A-Abs, we have $\widehat{\Gamma} \vdash \boldsymbol{\lambda} x\,.\,c : \theta_2\widehat{\alpha} \to \underline{\widehat{C}} \dashv \theta_2$.

Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_2$, $\theta_{\mathsf{inst}} = \theta_3|_{\theta_2\widehat{\Gamma}}$, and $\theta_d = \theta_4$, the conclusion is satisfied.

SD-Handler By the IH on the **return** clause, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_1$, taking a fresh type variable a that is not in $\widehat{\Gamma}$ and Γ. there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = (\theta_3\theta_2)|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash$ **return** $x \mapsto c_r : M\ \ \widehat{A}_1!\langle\widehat{E}_1\rangle \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, $\theta_4\theta_3(M\ \ \widehat{A}_1!\langle\widehat{E}_1\rangle) = M\ \ a!\langle E\rangle$ (1). Because α is a fresh type variable, $\widehat{A}_1$ must be an unification variable $\widehat{\alpha}_1$.

Then we apply the IH on the operation clauses *oprs* to $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_3 = (\theta_7\theta_6)|_{\theta_2\widehat{\Gamma}}$ (2), $\theta_2\widehat{\Gamma} \vdash \mathit{oprs} : M\ \ \widehat{A}_2!\langle\widehat{E}_2\rangle \dashv \theta_6$, $\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_7$ (3), $\theta_8\theta_7(M\ \ \widehat{A}_2!\langle\widehat{E}_2\rangle) = M\ \ a!\langle E\rangle$ (4). Similarly, $\widehat{A}_2$ must be an unification variable $\widehat{\alpha}_2$.

By (1), (2), and (4), we have $\theta_4(\theta_7\theta_6)|_{\theta_2\widehat{\Gamma}}(M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle) = \theta_8\theta_7(M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle)$. Because the new unification variables in $M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle$ and $M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle$ do not overlap, we have $\theta_8\theta_4\theta_7\theta_6(M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle) = \theta_8\theta_4\theta_7(M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle)$. By Lemma F.7, there exists θ_y, θ_x such that $\theta_8\theta_4\theta_7 = \theta_y\theta_x$ and $\theta_6(M\ \ \widehat{\alpha}_1!\langle\widehat{E}_1\rangle) \sim M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle : \theta_x$. By restricting the domain to be $\theta_6\theta_2\widehat{\Gamma}$, we have $\theta_7|_{\theta_6\theta_2\widehat{\Gamma}} = (\theta_y\theta_x)|_{\theta_6\theta_2\widehat{\Gamma}}$ (5).

By (3) and (5), we have $\theta_x\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_9$, where $\theta_9 = \theta_y|_{\theta_x\theta_6\theta_2\widehat{\Gamma}}$. By the IH on the **fwd** clause, there exists $\theta_{10}, \theta_{11}, \theta_{12}$ such that $\theta_9 = (\theta_{11}\theta_{10})|_{\theta_x\theta_6\theta_2\widehat{\Gamma}}$ (6), $\theta_x\theta_6\theta_2\widehat{\Gamma} \vdash$ **fwd** $f\ p\ k \mapsto c_f : M\ \ \widehat{A}_3!\langle\widehat{E}_3\rangle \dashv \theta_{10}$, $\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{11}$, $\theta_{12}\theta_{11}(M\ \ \widehat{A}_3!\langle\widehat{E}_3\rangle) = M\ \ \alpha!\langle E\rangle$ (6). Similar to the above, $\widehat{A}_3$ must be an unification variable $\widehat{\alpha}_3$.

By (4) and (7), we have $\theta_8\theta_7(M\ \ \widehat{A}_2!\langle\widehat{E}_2\rangle) = \theta_{12}\theta_{11}(M\ \ \widehat{A}_3!\langle\widehat{E}_3\rangle)$. By (6) and the fact that new unification variables in $M\ \ \widehat{\alpha}_2!\langle\widehat{E}_2\rangle$ and $M\ \ \widehat{\alpha}_3!\langle\widehat{E}_3\rangle$ do not overlap, we have $\theta_{12}\theta_y\theta_{11}\theta_{10}\theta_x(M\ \ \widehat{A}_2!\langle\widehat{E}_2\rangle) = \theta_{12}\theta_y\theta_{11}(M\ \ \widehat{A}_3!\langle\widehat{E}_3\rangle)$. By Lemma F.7, there exists θ'_y, θ'_x such that $\theta_{12}\theta_y\theta_{11} = \theta'_y\theta'_x$ and $\theta_{10}\theta_x(M\ \ \widehat{A}_2!\langle\widehat{E}_2\rangle) \sim M\ \ \widehat{A}_3!\langle\widehat{E}_3\rangle : \theta'_x$. By restricting the domain to be $\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma}$, we have $\theta_{11} = (\theta'_y\theta'_x)|_{\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma}}$. Because $\theta'_y\theta'_x(M\ \ \widehat{\alpha}_3!\langle\widehat{E}_3\rangle) = \theta_{12}\theta_y\theta_{11}(M\ \ \widehat{\alpha}_3!\langle\widehat{E}_3\rangle) = M\ \ \alpha!\langle E\rangle$ and α is fresh, we have that $\theta'_x\widehat{\alpha}_3$ must be an unification variable $\widehat{\alpha}_4$.

Because $\theta'_y\widehat{\alpha}_4 = \theta_{12}\theta_y\theta_{11}\widehat{\alpha}_3 = \theta_{12}\theta_{11}\widehat{\alpha}_3 = \alpha$, $\theta'_y\theta'_x\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma} = \theta_{11}\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma} = \Gamma$, and $\alpha \notin \Gamma$, we have $\theta'_y\ \widehat{\alpha}_4 \notin \theta'_y\theta'_x\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma}$, which leads to $\widehat{\alpha}_4 \notin \theta'_x\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma}$[3].

Let $\langle\widehat{F}\rangle = \langle \mathit{labels}\ (\mathit{oprs})\,;\theta'_x\ \ \widehat{E}_3\rangle$, by A-Handler, we have $\widehat{\Gamma} \vdash$ **handler** $\{\ldots\} : \widehat{\alpha}_4!\langle\widehat{F}\rangle \Rightarrow M\ \ \widehat{\alpha}_4!\langle\theta'_x\ \ \widehat{E}_3\rangle \dashv \theta'_x\theta_{10}\theta_x\theta_6\theta_2$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta'_x\theta_{10}\theta_x\theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta'_y|_{\theta'_x\theta_{10}\theta_x\theta_6\theta_2\widehat{\Gamma}}$, and $\theta_d = \theta'_y$, the conclusion is satisfied.

[3]here $\widehat{\alpha} \notin \widehat{\Gamma}$ actually means $\widehat{\alpha} \notin \mathit{fv}(\widehat{\Gamma})$

Computation typing.

SD-Ret By the IH on the return value v, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_a = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash \textbf{return}\ v : \widehat{A}\,!\,\widehat{\mu} \dashv \theta_2$, $\theta_2\ \widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, and $\theta_4\theta_3\widehat{A} = A$. Take a fresh unification variable $\widehat{\mu}$, by A-Ret we have $\widehat{\Gamma} \vdash \textbf{return}\ v : \widehat{A}\,!\,\widehat{\mu} \dashv \theta_2$.

Setting $\theta_{\mathsf{inf}} = \theta_2$, $\theta_{\mathsf{inst}} = \theta_3$, and $\theta_d = [E\ /\ \widehat{\mu}]\theta_4$, it is easy to check the conclusion is satisfied.

SD-App By the IH on v_1, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}\ (\widehat{\Gamma})$, $\widehat{\Gamma} \vdash v_1 : \widehat{A}_1 \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, and $\theta_4\theta_3\widehat{A}_1 = A \to \underline{C}$. Then we apply the IH on v_2 to the judgement $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, which gives us that there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}}$, $\theta_2\widehat{\Gamma} \vdash v_2 : \widehat{A}_2 \dashv \theta_6$, $\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma : \theta_7$, and $\theta_8\theta_7\widehat{A}_2 = A$. Let $\underline{C} = B\,!\,\langle E\rangle$, and take fresh unification variables $\widehat{\alpha}, \widehat{\mu}$, we have $[B\ /\ \widehat{\alpha}, E\ /\ \widehat{\mu}]\theta_8\theta_7(\widehat{A}_2 \to \widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle) = A \to \underline{C} = \theta_4\theta_3\widehat{A}_1 = \theta_4\theta_7\theta_6\widehat{A}_1$.

Note that θ_4 only substitutes new unification variables in $\theta_3\widehat{A}_1 = \theta_7\theta_6\widehat{A}_1$, and $[B/\widehat{\alpha}, E/\widehat{\mu}]\theta_8$ only substitutes new unification variables in $\theta_7(\widehat{A}_2 \to \widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle)$, we have the equation $[B/\widehat{\alpha}, E/\widehat{\mu}]\theta_8\theta_4\theta_7(\theta_6\widehat{A}_1) = [B/\widehat{\alpha}, E/\widehat{\mu}]\theta_8\theta_4\theta_7(\widehat{A}_2 \to \widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle)$. By Lemma F.7, there exists θ_{10}, θ_9 such that $\theta_6\widehat{A}_1 \sim (\widehat{A}_2 \to \widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle) : \theta_9$ and $[B\ /\ \widehat{\alpha}, E\ /\ \widehat{\mu}]\theta_8\theta_4\theta_7 = \theta_{10}\theta_9$. Restricting the domain to be $\widehat{\Gamma}$, we have $\theta_7 = (\theta_{10}\theta_9)|_{\widehat{\Gamma}}$. By Lemma F.3, we have $\theta_9\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma : \theta_{10}|_{\widehat{\Gamma}}$. By A-App, we have $\widehat{\Gamma} \vdash v_1\ v_2 : \theta_9\widehat{\alpha}\,!\,\langle\widehat{\mu}\rangle \dashv \theta_9\theta_6\theta_2$.

Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_9\theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_{10}|_{\widehat{\Gamma}}$, and $\theta_d = [B\ /\ \widehat{\alpha}, E\ /\ \widehat{\mu}]$, it is easy to check the conclusion of the theorem is satisfied.

SD-Do By the IH on c_1, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash c_1 : \widehat{A}\,!\,\langle\widehat{E}\rangle \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, and $\theta_4\theta_3(\widehat{A}\,!\,\langle\widehat{E}\rangle) = A\,!\,\langle E\rangle$. By $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$ and the fact that θ_4 only substitutes unification variables in $\widehat{A}$, we have $\theta_2\widehat{\Gamma}, x : \widehat{A} \rightsquigarrow \Gamma, x : A\,;\theta_4\theta_3$ (1).

Then we apply the IH on c_2 to (1), which gives that there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_4\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}, x:\widehat{A}}$, $\theta_2\widehat{\Gamma}, x{:}\widehat{A} \vdash c_2{:}\widehat{B}!\widehat{F} \dashv \theta_6$, $\theta_6(\theta_2\widehat{\Gamma}, x{:}\widehat{A}) \rightsquigarrow \Gamma, x{:}A{:}\theta_7$, and $\theta_8\theta_7(\widehat{B}\,!\,\langle\widehat{F}\rangle) = B\,!\,\langle E\rangle$. We have $\theta_8\theta_7\widehat{F} = E$ and $\theta_4\theta_3\widehat{E} = E = \theta_7\theta_6\widehat{E}$. Note that θ_8 only substitutes new unification variables in $\theta_7\widehat{F}$, we have $\theta_8\theta_7\widehat{F} = \theta_8\theta_7\theta_6\widehat{E}$. Thus, by Lemma F.7, there exists θ_9 and θ_{10} such that $\theta_6\widehat{E} \sim \widehat{F} : \theta_9$, $\theta_8\theta_7 = \theta_{10}\theta_9$. Restricting the domain to be $\widehat{\Gamma}$, we have $\theta_7 = (\theta_{10}\theta_9)|_{\widehat{\Gamma}}$. Thus, by Lemma F.3, we have $\theta_9|_{\widehat{\Gamma}}\theta_6(\theta_2\widehat{\Gamma}, x : \widehat{A}) \rightsquigarrow \Gamma, x : A : \theta_{10}|_{\widehat{\Gamma}}$. By A-Do, we also have $\widehat{\Gamma} \vdash \textbf{do}\ x \leftarrow c_1\,;c_2 : \theta_9(\widehat{B}\,!\,\langle\widehat{F}\rangle) \dashv \theta_9\theta_6\theta_2$.

Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_9\theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_{10}|_{\widehat{\Gamma}}$, and $\theta_d = \theta_8$, it is easy to check the conclusion is satisfied:

- $\theta_a = \theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}} = \theta_4\theta_3\theta_2|_{\widehat{\Gamma}} = \theta_7\theta_6\theta_2|_{\widehat{\Gamma}} = (\theta_{10}\theta_9\theta_6\theta_2)|_{\widehat{\Gamma}} = (\theta_{\mathsf{inst}}\theta_{\mathsf{inf}})|_{\widehat{\Gamma}}$
- $\widehat{\Gamma} \vdash \textbf{do}\ x \leftarrow c_1\,;c_2 : \theta_9(\widehat{B}\,!\,\langle\widehat{F}\rangle) \dashv \theta_{\mathsf{inf}}$
- $\theta_{\mathsf{inf}}\ \widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$
- $\theta_d\theta_{\mathsf{inst}}(\widehat{B}\,!\,\langle\widehat{F}\rangle) = \theta_8\theta_{10}|_{\widehat{\Gamma}}\theta_9(\widehat{B}\,!\,\langle\widehat{F}\rangle) = \theta_8\theta_{10}\theta_9(\widehat{B}\,!\,\langle\widehat{F}\rangle) = \theta_8\theta_7(\widehat{B}\,!\,\langle\widehat{F}\rangle) = B\,!\,\langle E\rangle$

SD-Let By the IH on v, for all $\widehat{\Gamma}, \overline{\alpha}, \overline{\mu} \rightsquigarrow \Gamma, \overline{\alpha}, \overline{\mu}\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma}, \overline{\alpha}, \overline{\mu} \vdash v : \widehat{A} \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$, and $\theta_4\theta_3\widehat{A} = A$.

We define θ_x as the set of all substitutions in $(\theta_4\theta_3)|_{\widehat{A}}$ which do not substitute any unification variable in $\mathit{fv}\ (\theta_2\widehat{\Gamma})$. Let $\sigma = \forall\ \overline{\alpha}\,.\forall\ \overline{\mu}\,.\,A$ and $\widehat{\sigma} = \forall\ \overline{\alpha}\,.\forall\ \overline{\mu}\,.\,\theta_x\widehat{A}$. By $\theta_4\theta_3\widehat{A} = A$, we have $\theta_2\widehat{\Gamma}, x : \widehat{\sigma} \rightsquigarrow \Gamma, x : \sigma\,;\theta_3$ (1).

Then we apply the IH on c to (1), which gives us that there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}}$, $\theta_2\widehat{\Gamma}, x : \widehat{\sigma} \vdash c : \underline{\widehat{C}} \dashv \theta_6$ (2), $\theta_6(\theta_2\widehat{\Gamma}, x : \widehat{\sigma}) \rightsquigarrow \Gamma, x : \sigma\,; \theta_7$, and $\theta_8\theta_7\underline{\widehat{C}} = \underline{C}$.

Let $\widehat{\overline{\alpha}}\ \widehat{\overline{\mu}} = uv\,(\widehat{A}) - uv\,(\theta_2\widehat{\Gamma})$ and $\widehat{\sigma}' = \forall\,\overline{\alpha}\,.\,\forall\,\overline{\mu}\,.\,\forall\,\widehat{\overline{\alpha}}\,.\,\forall\,\widehat{\overline{\mu}}\,.\,\widehat{A}$. If we replace the $\widehat{\sigma}$ in (2) with $\widehat{\sigma}'$, by the fact that the substitution θ_x does not substitutes anything in $\theta_2\widehat{\Gamma}$, the computation c is still well-typed, and there exists θ_y such that $\theta_2\widehat{\Gamma}, x : \widehat{\sigma}' \vdash c : \underline{\widehat{C}}' \dashv \theta_6$ and $\theta_y\underline{\widehat{C}}' = \underline{\widehat{C}}$ and θ_y also does not substitutes anything in $\theta_6\theta_2\widehat{\Gamma}$. Thus, $\theta_2\widehat{\Gamma}, x{:}\widehat{\sigma}' \vdash c{:}\underline{\widehat{C}} \dashv \theta_y\theta_6$. By A-Let, we have $\widehat{\Gamma} \vdash \textbf{let}\ x = v\ \textbf{in}\ c : \underline{\widehat{C}} \dashv \theta_y\theta_6\theta_2$.

Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_y\theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_7$, and $\theta_d = \theta_8$, the conclusion is satisfied.

SD-Op By the IH on v, for all $\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash v : \widehat{A}' \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_3$, and $\theta_4\theta_3\widehat{A}' = A_{\mathsf{op}}$. By Lemma F.7 and $\theta_4\theta_3\widehat{A}' = A_{\mathsf{op}}$, there exists θ_x, θ_y such that $\theta_4\theta_3 = \theta_y\theta_x$ and $\widehat{A}' \sim A_{\mathsf{op}} : \theta_x$. By $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_3$ and the fact that θ_4 only substitutes new unification variables in $\widehat{A}'$, we have $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_4\theta_3$. By restricting the domain to be $\theta_2\widehat{\Gamma}$ and the fact that θ_4 only substitutes new unification variables in $\widehat{A}'$, we have $(\theta_y\theta_x)|_{\theta_2\widehat{\Gamma}} = (\theta_4\theta_3)|_{\theta_2\widehat{\Gamma}} = \theta_3|_{\theta_2\widehat{\Gamma}} = \theta_3$. By $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_3$ and Lemma F.3, we have $\theta_x\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_z$, where $\theta_z = \theta_y|_{\theta_2\widehat{\Gamma}}$.

Then, we apply the IH on c to the juedgement $\theta_x\theta_2\widehat{\Gamma}, y : B_{\mathsf{op}} \rightsquigarrow \Gamma, y : B_{\mathsf{op}}\,; \theta_z$, there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_z = \theta_7\theta_6$, $\theta_x\theta_2\widehat{\Gamma}, y : B_{\mathsf{op}} \vdash c : \widehat{A}\,!\,\langle\widehat{E}\rangle \dashv \theta_6$, $\theta_6(\theta_x\theta_2\widehat{\Gamma}, y : B_{\mathsf{op}}) \rightsquigarrow \Gamma, y : B_{\mathsf{op}}\,; \theta_7$, and $\theta_8\theta_7(\widehat{A}\,!\,\langle\widehat{E}\rangle) = A\,!\,\langle E\rangle$. By Lemma F.7 and $\theta_8\theta_7\widehat{E} = E = \theta_8\theta_7 E$, there exists θ'_x, θ'_y such that $\theta_8\theta_7 = \theta'_y\theta'_x$ and $\widehat{E} \sim E : \theta'_x$ (1). Note that since $\ell^{\mathsf{op}} \in E$ and the row type $\langle \ell^{\mathsf{op}}\,; \widehat{\mu}\rangle$ is the most general type that explicitly contains label ℓ^{op}, the unification of $\widehat{E}$ and $\langle \ell^{\mathsf{op}}\,; \widehat{\mu}\rangle$ must succeed. Suppose $\widehat{E} \sim \langle \ell^{\mathsf{op}}\,; \widehat{\mu}\rangle : \theta_9$ (2) and $E \ \equiv_{\langle\rangle}\ \ell^{\mathsf{op}}\,; E'$, by A-Op we have $\widehat{\Gamma} \vdash \textbf{op}\ \ell^{\mathsf{op}}\ v\ (y\,.\,c) : \theta_9(\widehat{A}\,!\,\langle\widehat{E}\rangle) \dashv \theta_9\theta_6\theta_x\theta_2$. Now we do a case analysis to prove that there exists θ_{10} such that $\theta_7 = (\theta_{10}\theta_9)|_{\theta_6\theta_x\theta_2\widehat{\Gamma}}$:

$\widehat{\mu} \notin \mathit{dom}\,(\theta_9)$: We have $[E'\,/\,\widehat{\mu}]\theta_9\widehat{E} = E$. Thus, $[E'\,/\,\widehat{\mu}]\theta_9 = (\theta_8\theta_7)|_{\widehat{E}}$. By Lemma F.5, there exists θ_{10} such that $\theta_8\theta_7 = \theta_{10}[E'\,/\,\widehat{\mu}]\theta_9$. By restircting the domains of the substitutions on both side of the equation to $\theta_6\theta_x\theta_2\widehat{\Gamma}$ and simplifying the equation, we have $\theta_7 = (\theta_{10}\theta_9)|_{\theta_6\theta_x\theta_2\widehat{\Gamma}}$.

$\widehat{\mu} \in \mathit{dom}\,(\theta_9)$: Let θ'_9 be the substitution that generated from removing the substitution of $\widehat{\mu}$ from θ_9. Suppose $\langle E\rangle = \langle \ell^{\mathsf{op}}\,; E'\rangle$, by (1), we have $\theta'_x[E'\,/\,\widehat{\mu}]\widehat{E} = \theta'_x[E'\,/\,\widehat{\mu}]\langle \ell^{\mathsf{op}}\,; \widehat{\mu}\rangle$. By Lemma F.7 and (2), there exists θ_{11} such that $\theta'_x[E'\,/\,\widehat{\mu}] = \theta_{11}\theta_9$. Removing the substitution of $\widehat{\mu}$ from both sides of the equation, we have $\theta'_x = \theta_{11}\theta'_9$, where θ'_9 is the substitution generated by removing the substitution of $\widehat{\mu}$ from θ_9. Thus, by $\theta_8\theta_7 = \theta'_y\theta'_x$, we have $\theta_8\theta_7 = \theta'_y\theta_{11}\theta'_9$. By restricting the domain to be $\theta_6\theta_x\theta_2\widehat{\Gamma}$, we have $\theta_7 = (\theta'_y\theta_{11}\theta'_9)|_{\theta_6\theta_x\theta_2\widehat{\Gamma}} = (\theta'_y\theta_{11}\theta_9)|_{\theta_6\theta_x\theta_2\widehat{\Gamma}}$

Finally, setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_9\theta_6\theta_x\theta_2$, $\theta_{\mathsf{inst}} = \theta_{10}|_{\theta_9\theta_6\theta_x\theta_2\widehat{\Gamma}}$ and $\theta_d = \theta_8$, it is easy to check the conclusion is satisfied.

SD-Sc By the IH on v, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_1$, there exists $\theta_2, \theta_3, \theta_4$, such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_3$, and $\theta_4\theta_3\widehat{A} = A_{\mathsf{sc}}$. By Lemma F.7, there exists θ_y, θ_x such that $\theta_4\theta_3 = \theta_y\theta_x$ and $\widehat{A} \sim (A_{\mathsf{sc}}) : \theta_x$. By restricting the domain to be $\widehat{\Gamma}$, we have $\theta_3 = (\theta_4\theta_3)|_{\theta_2\widehat{\Gamma}} = (\theta_y\theta_x)|_{\theta_2\widehat{\Gamma}}$. By Lemma F.3, we have $\theta_x|_{\widehat{\Gamma}}\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_5$ (1), where $\theta_5 = \theta_y|_{\theta_x\widehat{\Gamma}}$.

Then we apply the IH on c_1 to (1), which gives us that there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_5 = \theta_7\theta_6$, $\theta_x\theta_2\widehat{\Gamma}, y : B_{\mathsf{sc}} \vdash c_1 : \widehat{B}\,!\langle\widehat{E}\rangle \dashv \theta_6$, $\theta_6(\theta_x\theta_2\widehat{\Gamma}, y : B_{\mathsf{sc}}) \rightsquigarrow \Gamma, y : B_{\mathsf{sc}}\,; \theta_7$, and $\theta_8\theta_7\widehat{B}\,!\langle\widehat{E}\rangle = B\,!\langle E\rangle$. By Lemma F.7, there exists θ'_y, θ'_x such that $\theta_8\theta_7 = \theta'_y\theta'_x$ and $\widehat{E} \sim E : \theta'_x$. Because $\ell^{\mathsf{sc}} \in E$ and $\langle \ell^{\mathsf{sc}}\,; \widehat{\mu}\rangle$ is the most general row type containing label ℓ^{sc}, there exists θ_9 such that $\widehat{E} \sim \langle \ell^{\mathsf{sc}}\,; \widehat{\mu}\rangle : \theta_9$. By a similar case analysis to the SD-Sc case, we get that there exists θ_{10} such that $\theta_7 = \theta_{10}\theta_9|_{\theta_6\theta_x\theta_2\widehat{\Gamma}}$. By Lemma F.3 and $\theta_6\theta_x\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_7$, we have that $\theta_9\theta_6\theta_x\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_{10}$, which implies $\theta_9\theta_6\theta_x\theta_2\widehat{\Gamma}, z : \theta_9\widehat{B} \rightsquigarrow \Gamma, z : B\,; \theta_{11}$ (2) where $\theta_{11} = \theta_8\theta_{10}$.

Then we apply the IH on c_2 to (2), which gives us that there exists $\theta_{12}, \theta_{13}, \theta_{14}$ such that $\theta_{11} = \theta_{13}\theta_{12}$, $\theta_9\theta_6\theta_x\theta_2\widehat{\Gamma}, z : \theta_9\widehat{B} \vdash c_2 : \widehat{A}'\,!\langle\widehat{F}\rangle \dashv \theta_{12}$, $\theta_{12}(\theta_9\theta_6\theta_x\theta_2\widehat{\Gamma}, z : \theta_9\widehat{B}) \rightsquigarrow \Gamma, z : B\,; \theta_{13}$, and $\theta_{14}\theta_{13}(\widehat{A}'\,!\langle\widehat{F}\rangle) = A\,!\langle E\rangle$. By Lemma F.7, there exists θ_z, θ_w such that $\theta_{14}\theta_{13} = \theta_z\theta_w$ and $\widehat{F} \sim E : \theta_w$. By restricting the domain to be $\widehat{\Gamma}' = \theta_{12}\theta_9\theta_6\theta_x\theta_2\widehat{\Gamma}$, we have $\theta_{13} = (\theta_{14}\theta_{13})|_{\widehat{\Gamma}'} = (\theta_z\theta_w)|_{\widehat{\Gamma}'}$.

By A-Sc, we have $\widehat{\Gamma} \vdash \mathbf{sc}\ \ell^{\mathsf{sc}}\ v\ (y\,.\,c_1)\ (z\,.\,c_2) : \theta_w(\widehat{A}'\,!\langle\widehat{F}\rangle) \dashv \theta_w\theta_{12}\theta_9\theta_6\theta_x\theta_2$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_w\theta_{12}\theta_9\theta_6\theta_x\theta_2$, $\theta_{\mathsf{inst}} = \theta_z|_{\theta_w\widehat{\Gamma}'}$, and $\theta_d = \theta_{14}$, the conclusion is satisfied.

SD-Hand By the IH on v, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash v : \widehat{A} \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_3$ (1), and $\theta_4\theta_3\widehat{A} = \underline{C} \Rightarrow \underline{D}$. By $\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}}$, we have $\theta_4\theta_7(\theta_6\widehat{A}) = \underline{C} \Rightarrow \underline{D}$ (1).

Then we apply the IH on c to (1), which gives us that there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}}$, $\theta_2\widehat{\Gamma} \vdash c : \underline{\widehat{C}} \dashv \theta_6$, $\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_7$ (3), and $\theta_8\theta_7\underline{\widehat{C}} = \underline{C}$. Taking two fresh unification variables $\widehat{\alpha}, \widehat{\mu}$ and supposing $\underline{D} = A\,!\langle E\rangle$, we have $[A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_8\theta_7(\underline{\widehat{C}} \Rightarrow \alpha\,!\langle\mu\rangle) = \underline{C} \Rightarrow \underline{D}$ (2).

Notice that θ_4 only substitutes new unification variables in $\widehat{A}$, and $[A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_8$ only substitutes new unification variables in $(\underline{\widehat{C}} \Rightarrow \alpha\,!\langle\mu\rangle)$, we can combine (1) and (2) to get the equation $[A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_8\theta_4\theta_7(\theta_6\widehat{A}) = [A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_8\theta_4\theta_7(\underline{\widehat{C}} \Rightarrow \alpha\,!\langle\mu\rangle)$. By Lemma F.7, there exists θ_x, θ_y such that $[A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_8\theta_4\theta_7 = \theta_y\theta_x$ and $\theta_6\widehat{A} \sim \underline{\widehat{C}} \Rightarrow \widehat{\alpha}\,!\langle\widehat{\mu}\rangle : \theta_x$. By restricting the domain to be $\theta_6\theta_2\widehat{\Gamma}$, we have $\theta_7 = (\theta_y\theta_x)|_{\theta_6\theta_2\widehat{\Gamma}}$.

By A-Hand, we have $\widehat{\Gamma} \vdash v \star c : \theta_x(\widehat{\alpha}\,!\langle\widehat{\mu}\rangle) \dashv \theta_x\theta_6\theta_2$. By Lemma F.3 and (3), we have $\theta_x\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_y|_{\theta_x\theta_6\theta_2\widehat{\Gamma}}$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_x theta\_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_y|_{\theta_x\theta_6\theta_2\widehat{\Gamma}}$, and $\theta_d = \theta_4$, it is easy to check the conclusion is satisfied.

Return clause.

SD-Return Taking a fresh unification variable $\widehat{\alpha}$, for any $\widehat{\Gamma}, x : \widehat{\alpha} \rightsquigarrow \Gamma, x : A\,; \theta_1$, there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma}, x : \widehat{\alpha} \vdash c_r : \underline{\widehat{C}} \dashv \theta_2$, $\theta_2(\widehat{\Gamma}, x : \widehat{\alpha}) \rightsquigarrow \Gamma, x : A\,; \theta_3$ (1), and $\theta_4\theta_3\underline{\widehat{C}} = M\ \ A\,!\langle E\rangle$. By (1), we have $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,; \theta_3|_{\theta_2\widehat{\Gamma}}$ (2).

Because $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$ and $\theta_1\widehat{\alpha} = A$, we have $M\ \ A\,!\,E = [E\,/\,\widehat{\mu}]\theta_3\theta_2(M\ \ \widehat{\alpha}\,!\langle\widehat{\mu}\rangle) = \theta_4\theta_3\underline{\widehat{C}}$. Note that θ_4 only substitutes new unification variables in $\underline{\widehat{C}}$, we have $\theta_4[E\,/\,\widehat{\mu}]\theta_3\theta_2(M\ \ \widehat{\alpha}\,!\langle\widehat{\mu}\rangle) = \theta_4[E\,/\,\widehat{\mu}]\theta_3\underline{\widehat{C}}$. By Lemma F.7, there exists θ_x, θ_y such that $\theta_4[E\,/\,\widehat{\mu}]\theta_3 = \theta_y\theta_x$ and $\underline{\widehat{C}} \sim \widehat{\alpha}\,!\langle\widehat{\mu}\rangle : \theta_x$. By restricting the domain to be $\theta_2\widehat{\Gamma}$, we have $\theta_3|_{\theta_2\widehat{\Gamma}} = (\theta_y\theta_x)|_{\theta_2\widehat{\Gamma}}$.

By A-Return, we have $\widehat{\Gamma} \vdash \textbf{return}\ x \mapsto c_r : \theta_x\theta_2(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_x\theta_2$. By (2) and Lemma F.3, we have $\theta_x\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_x\theta_2$, $\theta_{\mathsf{inst}} = \theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$, and $\theta_d = [E\ /\ \widehat{\mu}]$, it is easy to check the conclusion is satisfied.

Operation clauses.

SD-Empty Our goal is for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_a$ and fresh unification variables $\widehat{\alpha}, \widehat{\mu}$, there exists $\theta_{\mathsf{inf}}, \theta_{\mathsf{inst}}, \theta_d$ such that $\theta_a = \theta_{\mathsf{inst}}\theta_{\mathsf{inf}}$, $\widehat{\Gamma} \vdash \cdot : M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle \dashv \theta_{\mathsf{inf}}$, $\theta_{\mathsf{inf}}\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_{\mathsf{inst}}$, $\theta_d\theta_{\mathsf{inst}}(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) = M\ A!\langle E\rangle$. It is easy to check that setting $\theta_{\mathsf{inf}} = \emptyset$, $\theta_{\mathsf{inst}} = \theta_a$, and $\theta_d = [A\ /\ \widehat{\alpha}, E\ /\ \widehat{\mu}]$ satisfies our goal.

SD-OprOp By the IH on *oprs*, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4, \widehat{A}, \widehat{E}$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash \mathit{oprs} : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$ (1), and $\theta_4\theta_3(M\ \widehat{A}!\langle\widehat{E}\rangle) = M\ A!\langle E\rangle$ (2). Note that here we want θ_4 to be minimal, i.e. $\theta_4 = \theta_4|_{\theta_3(M\ \widehat{A}!\langle\widehat{E}\rangle)}$.

By (1) and (2), we have $\theta_2\widehat{\Gamma}, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \widehat{A}!\langle\widehat{E}\rangle \rightsquigarrow \Gamma, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ A!\langle E\rangle\,;\theta_4\theta_3$ (3). Let $\widehat{\Gamma}' = \theta_2\widehat{\Gamma}, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ \widehat{A}!\langle\widehat{E}\rangle$ and $\Gamma' = \Gamma, x : A_{\mathsf{op}}, k : B_{\mathsf{op}} \to M\ A!\langle E\rangle$. Applying the IH on c to (3), there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_4\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}, x:A_{\mathsf{op}}, k:B_{\mathsf{op}} \to M\ \widehat{A}!\langle\widehat{E}\rangle}$, $\widehat{\Gamma}' \vdash c : \underline{\widehat{C}} \dashv \theta_6$, $\theta_6\widehat{\Gamma}' \rightsquigarrow \Gamma'\,;\theta_7$, and $\theta_8\theta_7\underline{\widehat{C}} = M\ A!\langle E\rangle$ (4).

By (2), (4), $\theta_4\theta_3 = \theta_7\theta_6$, and the fact that θ_8 only substitutes new unification variables in $\underline{\widehat{C}}$, we have $\theta_8\theta_7\theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle) = \theta_8\theta_7\underline{\widehat{C}}$. By Lemma F.7, there exists θ_y, θ_x such that $\theta_8\theta_7 = \theta_y\theta_x$ and $\theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle) \sim \underline{\widehat{C}} : \theta_x$. By restricting the domain to be $\theta_2\widehat{\Gamma}$, we have $\theta_7|_{\theta_2\widehat{\Gamma}} = (\theta_y\theta_x)|_{\theta_2\widehat{\Gamma}}$ (5).

By A-OprOp, we have $\widehat{\Gamma} \vdash_M \textbf{op}\ \ell^{\mathsf{op}}\ x\ k \mapsto c, \mathit{oprs} : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_x\theta_6\theta_2$. By $\theta_6\widehat{\Gamma}' \rightsquigarrow \Gamma'\,;\theta_7$ and (5), we have $\theta_x\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_x\theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$, and $\theta_d = \theta_4$, the conclusion is satisfied.

SD-OprSc Similar to the case of SD-OprOp.

By the IH on *oprs*, for any $\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_1$, there exists $\theta_2, \theta_3, \theta_4, \widehat{A}, \widehat{E}$ such that $\theta_1 = \theta_3\theta_2|_{\widehat{\Gamma}}$, $\widehat{\Gamma} \vdash \mathit{oprs} : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_2$, $\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma\,;\theta_3$ (1), and $\theta_4\theta_3(M\ \widehat{A}!\langle\widehat{E}\rangle) = M\ A!\langle E\rangle$ (2). Note that here we want θ_4 to be minimal, i.e. $\theta_4 = \theta_4|_{\theta_3(M\ \widehat{A}!\langle\widehat{E}\rangle)}$.

Take a fresh type variable β. By (1) and (2), we have $\theta_2\widehat{\Gamma}, \beta, x{:}A_{\mathsf{sc}}, p{:}B_{\mathsf{sc}} \to M\ \beta!\langle\widehat{E}\rangle, k{:}\beta \to M\ \widehat{A}!\langle\widehat{E}\rangle \rightsquigarrow \Gamma, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta!\langle E\rangle, k : \beta \to M\ A!\langle E\rangle\,;\theta_4\theta_3$ (3). Let $\widehat{\Gamma}' = \theta_2\widehat{\Gamma}, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta!\langle\widehat{E}\rangle, k : \beta \to M\ \widehat{A}!\langle\widehat{E}\rangle$ and $\Gamma' = \Gamma, \beta, x : A_{\mathsf{sc}}, p : B_{\mathsf{sc}} \to M\ \beta!\langle E\rangle, k : \beta \to M\ A!\langle E\rangle$. Applying the IH on c to (3), there exists $\theta_6, \theta_7, \theta_8$ such that $\theta_4\theta_3 = \theta_7\theta_6|_{\theta_2\widehat{\Gamma}, x:A_{\mathsf{sc}}, p:B_{\mathsf{sc}} \to M\ \beta!\langle\widehat{E}\rangle, k:\beta \to M\ \widehat{A}!\langle\widehat{E}\rangle}$, $\widehat{\Gamma}' \vdash c : \underline{\widehat{C}} \dashv \theta_6$, $\theta_6\widehat{\Gamma}' \rightsquigarrow \Gamma'\,;\theta_7$, and $\theta_8\theta_7\underline{\widehat{C}} = M\ A!\langle E\rangle$ (4).

By (2), (4), $\theta_4\theta_3 = \theta_7\theta_6$, and the fact that θ_8 only substitutes new unification variables in $\underline{\widehat{C}}$, we have $\theta_8\theta_7\theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle) = \theta_8\theta_7\underline{\widehat{C}}$. By Lemma F.7, there exists θ_y, θ_x such that $\theta_8\theta_7 = \theta_y\theta_x$ and $\theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle) \sim \underline{\widehat{C}} : \theta_x$. By restricting the domain to be $\theta_2\widehat{\Gamma}$, we have $\theta_7|_{\theta_2\widehat{\Gamma}} = (\theta_y\theta_x)|_{\theta_2\widehat{\Gamma}}$ (5).

Since β is fresh, we have $\beta \notin \mathit{rng}\ (\theta_2)$. By $\theta_6\widehat{\Gamma}' \rightsquigarrow \Gamma'\,;\theta_7$, we have $\beta \notin \mathit{rng}\ (\theta_6|_{\widehat{\Gamma}'})$. Because $\theta_7\theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle) = M\ A!\langle E\rangle$, we have $\beta \notin \theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle)$. Thus, $\beta \notin \theta_x$ and $\beta \notin \theta_6|_{\underline{\widehat{C}}}$. Thus, by Lemma F.4, we have $\beta \notin \mathit{rng}\ (\theta_6)$. Finally, we have $\beta \notin \mathit{rng}\ (\theta_x\theta_6\theta_2)$.

By A-OprSc, we have $\widehat{\Gamma} \vdash_M \textbf{sc}\ \ell^{\mathsf{sc}}\ x\ p\ k \mapsto c, oprs : M\ \widehat{A}!\langle\widehat{E}\rangle \dashv \theta_x\theta_6\theta_2$. By $\theta_6\widehat{\Gamma}' \rightsquigarrow \Gamma' ; \theta_7$ and (5), we have $\theta_x\theta_6\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma ; \theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$. Setting $\theta_a = \theta_1$, $\theta_{\mathsf{inf}} = \theta_x\theta_6\theta_2$, $\theta_{\mathsf{inst}} = \theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$, and $\theta_d = \theta_4$, the conclusion is satisfied.

Forwarding clause.

SD-Fwd Take fresh unification variables $\widehat{\alpha}, \widehat{\mu}$, fresh type variables α, β, and fresh type variable names γ, δ. For any $\widehat{\Gamma} \rightsquigarrow \Gamma ; \theta_a$, we have $\widehat{\Gamma}, \alpha, \beta, p : \widehat{A}_p, k : \widehat{A}_k, f : \forall\ \gamma\ \delta\,.\,(\widehat{A}'_p, \widehat{A}'_k) \to \delta!\langle\widehat{\mu}\rangle \rightsquigarrow \Gamma, \alpha, \beta, p : A_p, k : A_k, f : \forall\ \gamma\ \delta\,.\,(A'_p, A'_k) \to \delta!\langle E\rangle ; [A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_a$ (1). All $A_p, A_k, A'_p, A'_k, \widehat{A}_p, \widehat{A}_k, \widehat{A}'_p, \widehat{A}'_k$ are defined the same as what are defined in rules SD-Fwd and A-Fwd. Let $\widehat{\Gamma}' = \widehat{\Gamma}, \alpha, \beta, p : \widehat{A}_p, k : \widehat{A}_k, f : \forall\ \gamma\ \delta\,.\,(\widehat{A}'_p, \widehat{A}'_k) \to \delta!\langle\widehat{\mu}\rangle$ and $\Gamma' = \Gamma, \alpha, \beta, p : A_p, k : A_k, f : \forall\ \gamma\ \delta\,.\,(A'_p, A'_k) \to \delta!\langle E\rangle$.

Applying the IH on c_f to (1), there exists $\theta_2, \theta_3, \theta_4$ such that $\theta_3\theta_2|_{\widehat{\Gamma}, p:\widehat{A}_p, k:\widehat{A}_k, f:\forall\,\gamma\,\delta\,.\,(\widehat{A}'_p, \widehat{A}'_k)\to\delta!\langle\widehat{\mu}\rangle} = [A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_a$, $\widehat{\Gamma}' \vdash c_f : \underline{\widehat{C}} \dashv \theta_2$, $\theta_2\widehat{\Gamma}' \rightsquigarrow \Gamma' ; \theta_3$ (2), and $\theta_4\theta_3\underline{\widehat{C}} = M\ A!\langle E\rangle$.

Because $\widehat{\alpha}$ and $\widehat{\mu}$ are fresh, we have $\theta_4[A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]\theta_a(m\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) = M\ A!\langle E\rangle$, which leads to $\theta_4\theta_3\theta_2(m\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) = M\ A!\langle E\rangle = \theta_4\theta_3\underline{\widehat{C}}$. By Lemma F.7, there exists θ_x, θ_y such that $\theta_4\theta_3 = \theta_y\theta_x$ and $\underline{\widehat{C}} \sim \theta_2(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) : \theta_x$. By restricting the domain to be $\theta_2\widehat{\Gamma}$, we have $\theta_3|_{\theta_2\widehat{\Gamma}} = (\theta_y, \theta_x)|_{\theta_2\widehat{\Gamma}}$.

It is obvious that $\gamma, \delta \notin rng\ (\theta_x\theta_2)$ because they are bounded by universal quantifiers. By $\theta_6\widehat{\Gamma}' \rightsquigarrow \Gamma' ; \theta_7$, we have $\alpha, \beta \notin rng\ (\theta_6|_{\widehat{\Gamma}'})$. Because $\theta_7\theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle) = M\ A!\langle E\rangle$, we have $\alpha, \beta \notin \theta_6(M\ \widehat{A}!\langle\widehat{E}\rangle)$. Thus, $\alpha, \beta \notin \theta_x$ and $\alpha, \beta \notin \theta_6|_{\underline{\widehat{C}}}$. Thus, by Lemma F.4, we have $\alpha, \beta \notin rng\ (\theta_6)$. Finally, we have $\alpha, \beta, \gamma, \delta \notin rng\ (\theta_x\theta_6\theta_2)$.

By A-Fwd, we have $\widehat{\Gamma} \vdash_M \textbf{fwd}\ f\ p\ k \mapsto c_f : \theta_x\theta_2(M\ \widehat{\alpha}!\langle\widehat{\mu}\rangle) \dashv \theta_x\theta_2$. By (1) and Lemma F.3, we have $\theta_x\theta_2\widehat{\Gamma} \rightsquigarrow \Gamma ; \theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$. Setting $\theta_{\mathsf{inf}} = \theta_x\theta_2$, $\theta_{\mathsf{inst}} = \theta_y|_{\theta_x\theta_2\widehat{\Gamma}}$, and $\theta_d = [A\,/\,\widehat{\alpha}, E\,/\,\widehat{\mu}]$, the conclusion is satisfied.

$\square$

F.7. **Discussion.** Our type inference algorithm works as is for most of the examples in Section 7. The type operators M used by a few handlers have additional type variables, which go beyond the basic type inference algorithm we have presented.

For example, the type of the handler of the reader with local handler in Section 7.2.2 is

$$h_{\mathsf{read}} : \forall\ \alpha\ \mu\,.\,\alpha\,!\,\langle\mathtt{ask}\,;\mathtt{local}\,;\mu\rangle \Rightarrow (\mathsf{Int} \to^{\mu} (\alpha, \mathsf{Int}))!\langle\mu\rangle$$

Thus, for type inference, h_{read} has the type annotation $\boldsymbol{\lambda}\alpha\,.\,\mathsf{Int} \to^{\mu} (\alpha, \mathsf{Int})$ which contains type variable μ that is bounded by $\forall\ \mu$ in the type of h_{read}. Our basic algorithm does not allow free type variables in the annotation.

We see several ways to generalize our algorithm to accommodate these examples. The first approach is to allow free type variables to occur in the type annotation and add the condition that the free variables in the annotation are not in the range of the inferred substitution of the handler typing judgment. This condition ensures that the free type variables in the annotation will not escape the lexical scope of the handler type. The new handler typing rule is shown in Figure 24.

Another approach is to extend the type annotation of handlers from $\boldsymbol{\lambda}\alpha\,.\,A$ to $\boldsymbol{\lambda}\alpha\,.\,\boldsymbol{\lambda}\mu\,.\,A$ to bind the extra type variable μ. In this way, we also need to adjust the typing rules of the

$$\dfrac{\begin{array}{c} \widehat{\Gamma} \vdash_M \textbf{return}\, x \mapsto c_r : M\ \widehat{A}_1 ! \langle \widehat{E}_1 \rangle \dashv \theta_1 \\ \widehat{A}_1 = \widehat{\alpha}_1 \qquad \theta_1 \widehat{\Gamma} \vdash_M \mathit{oprs} : M\ \widehat{A}_2 ! \langle \widehat{E}_2 \rangle \dashv \theta_2 \qquad \widehat{A}_2 = \widehat{\alpha}_2 \\ \theta_2(\widehat{\alpha}_1 ! \langle \widehat{E}_1 \rangle) \sim \widehat{\alpha}_2 ! \langle \widehat{E}_2 \rangle : \theta_3 \qquad \theta_{3..1} \widehat{\Gamma} \vdash_M \textbf{fwd}\, f\ p\ k \mapsto c_f : M\ \widehat{A}_3 ! \langle \widehat{E}_3 \rangle \dashv \theta_4 \\ \widehat{A}_3 = \widehat{\alpha}_3 \qquad \theta_{4..3}(\widehat{\alpha}_2 ! \langle \widehat{E}_2 \rangle) \sim \widehat{\alpha}_3 ! \langle \widehat{E}_3 \rangle : \theta_5 \qquad \widehat{\alpha}_4 = \theta_5 \widehat{\alpha}_3 \qquad \widehat{\alpha}_4 \notin \mathit{fv}\,(\theta_{5..1} \widehat{\Gamma}) \\ \langle \widehat{F} \rangle = \langle \mathit{labels}\,(\mathit{oprs})\, ; \theta_5\ \widehat{E}_3 \rangle \qquad \mathit{fv}\,(M) \cap \mathit{fv}\,(\theta_{5..1} \widehat{\Gamma}) = \emptyset \end{array}}{\begin{array}{c} \widehat{\Gamma} \vdash \textbf{handler}_M\ \{\textbf{return}\, x \mapsto c_r, \mathit{oprs}, \textbf{fwd}\, f\ p\ k \mapsto c_f\} : \\ \widehat{\alpha}_4 ! \langle \widehat{F} \rangle \Rightarrow M\ \widehat{\alpha}_4 ! \langle \theta_5 \widehat{E}_3 \rangle \dashv \theta_{5..1} \end{array}} \ \text{A-HANDLER'}$$

Figure 24: The new type inference rule for handlers.

handler and handler clauses to pass extra parameters to the type operator M. This approach is less general than the first one, so we have used the first approach in our implementation.