Beyond Polarity: Towards A Multi-Discipline Intermediate Language with Sharing

Paul Downen
University of Oregon [Eugene, OR, USA]
pdownen@cs.uoregon.edu

Zena M. Ariola
University of Oregon [Eugene, OR, USA]
ariola@cs.uoregon.edu

Abstract

The study of *polarity* in computation has revealed that an "ideal" programming language combines both call-by-value and call-by-name evaluation; the two calling conventions are each ideal for half the types in a programming language. But this binary choice leaves out call-by-need which is used in practice to implement lazy-by-default languages like Haskell. We show how the notion of polarity can be extended beyond the value/name dichotomy to include call-by-need by only adding a mechanism for sharing and the extra *polarity shifts* to connect them, which is enough to compile a Haskell-like functional language with user-defined types.

2012 ACM Subject Classification Theory of computation → Type structures

Keywords and phrases call-by-need, polarity, call-by-push-value, control

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.20

Funding This work is supported by the National Science Foundation under grants CCF-1719158 and CCF-1423617.

1 Introduction

Finding a universal intermediate language suitable for compiling and optimizing both strict and lazy functional programs has been a long-sought holy grail for compiler writers. First there was continuation-passing style (CPS) [21, 2], which hard-codes the evaluation strategy into the program itself. In CPS, all the specifics of evaluation strategy can be understood just by looking at the syntax of the program. Second there were monadic languages [14, 18], that abstract away from the concrete continuation-passing into a general monadic sequencing operation. Besides moving away from continuations, making them an optional rather than mandatory part of sequencing, they make it easier to incorporate other computational effects by picking the appropriate monad for those effects. Third there were adjunctive languages [11, 26, 15], as seen in polarized logic and call-by-push-value λ-calculus, that mix both call-by-name and -value evaluation inside a single program. Like the monadic approach, adjunctive languages make evaluation order explicit within the terms and types of a program, and can easily accommodate effects. However, adjunctive languages also enable more reasoning principles, by keeping the advantages of inductive call-by-value data types, as seen in their denotational semantics. For example, the denotation of a list is just a list of values, not a list of values interspersed with computations that might diverge or cause side effects.

Each of these developments have focused only on call-by-value and -name evaluation, but there are other evaluation strategies out there. For example, to efficiently implement laziness, the Glasgow Haskell Compiler (GHC) uses a core intermediate language which is call-by-need [4] instead of call-by-name: the computation of named expressions is shared throughout the lifetime of their result, so that they need not be re-evaluated again. This may be seen as merely an optimization of call-by-name, but it is one that has a profound impact on the other optimizations the compiler can do. For example, full extensionality of functions (*i.e.,* the η law) does not apply in general, due to issues involving divergence and evaluation order. Furthermore, call-by-need is not just a mere optimization but a full-fledged language choice when effects are introduced [3]: call-by-need and -name are observationally different. This difference may not matter for pure functional programs, but even there, effects *become* important during compilation. For example, it is beneficial to use join points [13], which is a limited form of jump or *goto* statement, to optimize pure functional programs.

So it seems like the quest for a universal intermediate language is still ongoing. To handle all the issues involving evaluation order in modern functional compilers, the following questions, which have been unanswered so far, should also be addressed:

- (Section 3) How do you extend polarity with sharing (*i.e.,* call-by-need)? For example, how do you model the Glasgow Haskell Compiler (GHC) which mixes both call-by-need for ordinary Haskell programs and call-by-value for *unboxed* [19] machine primitives?
- (Section 4) What does a core language need to serve as a compile target for a general functional programming language with user-defined types? What are the *shifts* you need to convert between all three calling conventions? While encoding data types is routine, what do you need to fully encode *co-data types* [9]?
- (Section 5) How do you compile that general functional language to the core intermediate sub-language? And how do you know that it is robust when effects are added?

This paper answers each of these questions. To test the robustness of this idea, we extend it in several directions in the appendix. We generalize to a dual sequent calculus framework that incorporates more calling conventions (specifically, the dual to call-by-need) and connectives not found in functional languages (Appendices C and D). We formally relate our intermediate language with polarity and call-by-push-value (Appendices B and F). Finally, full proofs of correctness and standard meta-theoretic properties are provided (Appendices E and G to I).

2 Polarity, data, and co-data

To begin, let's start with a basic language which is the λ-calculus extended with sums, as expressed by the following types and terms:

$$A, B, C ::= X \mid A \to B \mid A \oplus B$$
$$M, N, P ::= x \mid \lambda x.M \mid M\ N \mid \iota_1 M \mid \iota_2 M \mid \textbf{case}\ M\ \textbf{of}\{\iota_1 x.N \mid \iota_2 y.P\}$$

As usual, an abstraction $\lambda x.M$ is a term of a function type $A \to B$ and an injection $\iota_i M$ is a term of a sum type $A \oplus B$. Terms of function and sum types are used via application $(M\ N)$ and **case** analysis, respectively. Variables x can be of any type, even an atomic type X.

To make this a programming language, we would need to explain how to run programs (say, closed terms of a sum type) to get results. But what should the calling convention be? We could choose to use call-by-value evaluation, wherein a function application $(\lambda x.M)\ N$ is reduced by first evaluating N and then plugging its value in for x, or call-by-name evaluation, wherein the same application is reduced by immediately substituting N for x without further evaluation. We might think that this choice just impacts efficiency, trading off the cost of evaluating an unneeded argument in call-by-value for the potential cost of re-evaluating the same argument many times in call-by-name. However, the choice of calling convention also impacts the properties of the language, and can affect our ability to reason about programs.

Functions are a co-data type [7], so the extensionality law for functions, known as η, expands function terms into trivial λ-abstractions as follows:

$$(\eta_{\to}) \qquad M : A \to B = \lambda x.M\ x \qquad (x \notin FV(M))$$

But once we allow for any computational effects in the language, this law only makes sense with respect to call-by-name evaluation. For example, suppose that we have a non-terminating term Ω (perhaps caused by general recursion) which never returns a value. Then the $\eta_{\to}$ law stipulates that $\Omega = \lambda x.\Omega\ x$. This equality is fine—it does not change the observable behavior of any program—in call-by-name, but in call-by-value, $(\lambda z.5)\ \Omega$ loops forever and $(\lambda z.5)\ (\lambda x.\Omega\ x)$ returns 5. So the full $\eta_{\to}$ breaks in call-by-value.

In contrast, sums are a data type, so one sensible extensionality law for sums, which corresponds to reasoning by induction on the possible cases of a free variable, is expressed by the following law stating that if x has type $A \oplus B$ then it does no harm to **case** on x first:

$$(\eta_{\oplus}) \qquad M = \mathbf{case}\, x\, \mathbf{of}\{\iota_1 y.M[\iota_1 y/x] \mid \iota_2 z.M[\iota_2 z/x]\} \qquad (x : A \oplus B)$$

Unfortunately, this law only makes sense with respect to call-by-value evaluation once we have effects. For example, consider the instance where M is $\iota_1 x$. In call-by-value, variables stand for *values* which are already evaluated because that is all that they might be substituted for. So in either case, when we plug in something like $\iota_i 5$ for x, we get the result $\iota_1(\iota_i 5)$ after evaluating the right-hand side. But in call-by-name, variables range over all terms which might induce arbitrary computation. If we substitute Ω for x, then the left-hand side results in $\iota_1\Omega$ but the right-hand side forces evaluation of Ω with a **case**, and loops forever.

How can we resolve this conflict, where one language feature "wants" call-by-name evaluation and the other "wants" call-by-value? We just could pick one or the other as the default of the language, to the detriment of either functions or sums. Or instead we could integrate the two to get the best of both worlds, and *polarize* the language so that functions are evaluated according to call-by-name, and sums according to call-by-value. That way, both of them have their best properties in the same language, even when effects come into play. Since functions and sums are already distinguished by types, we can leverage the type system to make the call-by-value and -name distinction for us. That is to say, a type A might classify either a call-by-value term, denoted by A_+, or a call-by-name term, denoted by A_-. Put it all together, we get the following polarized typing rules for our basic λ-calculus:

$$A, B, C ::= A_+ \mid A_- \qquad A_-, B_- ::= X^- \mid A_+ \to B_- \qquad A_+, B_+ ::= X^+ \mid A_+ \oplus B_+$$

$$\frac{}{\Gamma, x : A \vdash x : A}\,Var \qquad \frac{\Gamma, x : A_+ \vdash M : B_-}{\Gamma \vdash \lambda x.M : A_+ \to B_-}\,{\to}I \qquad \frac{\Gamma \vdash M : A_+ \to B_- \quad \Gamma \vdash N : A_+}{\Gamma \vdash M\ N : B_-}\,{\to}E$$

$$\frac{\Gamma \vdash M : A_+}{\Gamma \vdash \iota_1 M : A_+ \oplus B_+}\,\oplus I_1 \qquad \frac{\Gamma \vdash M : B_+}{\Gamma \vdash \iota_2 M : A_+ \oplus B_+}\,\oplus I_2$$

$$\frac{\Gamma \vdash M : A_+ \oplus B_+ \quad \Gamma, x : A_+ \vdash N : C \quad \Gamma, y : B_+ \vdash P : C}{\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\{\iota_1 x.N \mid \iota_2 y.P\} : C}\,\oplus E$$

Note that, with this polarization, injections are treated as call-by-value, in $\iota_i M$ the term M is evaluated before the tagged value is returned. More interestingly, the function call $M\ N$ has two parts: the argument N is evaluated before the function is called as in call-by-value, but this only happens once the result is demanded as in call-by-name.

But there's a problem, just dividing up the language into two has severely restricted the ways we can compose types and terms. We can no longer inject a function into a sum, because a function is negative but a sum can only contain positive parts. Even more extreme, the identity function $\lambda x.x : A \to A$ no longer makes sense: the input must be a positive type and the output a negative type, and A cannot be both positive and negative at once. To get around this restriction, we need the ability to *shift* polarity between positive and negative. That way, we can still compose types and terms any way we want, just like before, and have the freedom of making the choice between call-by-name or -value instead of having the language impose one everywhere.

If we continue the data and co-data distinction that we had between sums and functions above, there are different ways of arranging the two shifts in the literature, depending on the viewpoint. In Levy's call-by-push-value [11] the shift from positive to negative $\Uparrow$ (therein called F) can be interpreted as a data type, where the sequencing operation is subsumed by the usual notion of a **case** on values of that data type, and the reverse shift $\Downarrow$ (therein called U) can be interpreted as co-data type:[1]

$$A_-, B_- ::= \dots \mid {\Uparrow}A_+ \qquad \frac{\Gamma \vdash M : A_+}{\Gamma \vdash \mathsf{val}\, M : {\Uparrow}A_+}\,{\Uparrow}I \qquad \frac{\Gamma \vdash M : {\Uparrow}A_+ \quad \Gamma, x : A_+ \vdash N : C}{\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\{\mathsf{val}\, x.N\} : C}\,{\Uparrow}E$$

$$A_+, B_+ ::= \dots \mid {\Downarrow}A_- \qquad \frac{\Gamma \vdash M : A_-}{\Gamma \vdash \lambda\mathsf{enter}.M : {\Downarrow}A_-}\,{\Downarrow}I \qquad \frac{\Gamma \vdash M : {\Downarrow}A_-}{\Gamma \vdash M.\mathsf{enter} : A_-}\,{\Downarrow}E$$

$M.\mathsf{enter}$ can be seen as sending the request enter to M, and $\lambda\mathsf{enter}.M$ as waiting for that request. In contrast, Zeilberger's calculus of unity [25] takes the opposite view, where the shift $\uparrow$ from positive to negative is co-data and the opposite shift $\downarrow$ is data:

$$A_-, B_- ::= \dots \mid {\uparrow}A_+ \qquad \frac{\Gamma \vdash M : A_+}{\Gamma \vdash \lambda\mathsf{eval}.M : {\uparrow}A_+}\,{\uparrow}I \qquad \frac{\Gamma \vdash M : {\uparrow}A_+}{\Gamma \vdash M.\mathsf{eval} : A_+}\,{\uparrow}E$$

$$A_+, B_+ ::= \dots \mid {\downarrow}A_- \qquad \frac{\Gamma \vdash M : A_-}{\Gamma \vdash \mathsf{box}\, M : {\downarrow}A_-}\,{\downarrow}I \qquad \frac{\Gamma \vdash M : {\downarrow}A_- \quad \Gamma, x : A_- \vdash N : C}{\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\{\mathsf{box}\, x.N\} : C}\,{\downarrow}E$$

Here, we do not favor one form over the other and allow both forms to coexist. In turns out that with only call-by-value and -name evaluation, the two pairs of shifts amount to the same thing (more formally, we will see in Section 5 that they are *isomorphic*). But we will see next in Section 3 how extending this basic language calls both styles of shifts into play.

With the polarity shifts between positive and negative types, we can express every program that we could have in the original unpolarized language. The difference is that now since *both* call-by-value and -name evaluation is denoted by different types, the types themselves signify the calling convention. For call-by-name, this encoding is:

$$[\![X]\!]^- = X^- \qquad [\![A \to B]\!]^- = ({\downarrow}[\![A]\!]^-) \to [\![B]\!]^- \qquad [\![A \oplus B]\!]^- = {\Uparrow}(({\downarrow}[\![A]\!]^-) \oplus ({\downarrow}[\![B]\!]^-))$$

$$[\![x]\!]^- = x$$

$$[\![M\ N]\!]^- = [\![M]\!]^-(\mathsf{box}\,[\![N]\!]^-) \qquad [\![\lambda x.M]\!]^- = \lambda y.\,\mathbf{case}\, y\, \mathbf{of}\{\mathsf{box}\, x.[\![M]\!]^-\}$$

$$[\![\iota_i M]\!]^- = \mathsf{val}(\iota_i(\mathsf{box}\,[\![M]\!]^-)) \qquad [\![\mathbf{case}\, M\, \mathbf{of}\{\iota_i x_i.N_i\}]\!]^- = \mathbf{case}\,[\![M]\!]^-\, \mathbf{of}\{\mathsf{val}(\iota_i(\mathsf{box}\, x_i)).[\![N_i]\!]^-\}$$

where the nested pattern $\mathsf{val}(\iota_i(\mathsf{box}\, x_i))$ is expanded in the obvious way. It converts every type into a negative one, and amounts to boxing up the arguments of injections and function calls. The call-by-value encoding is:

$$[\![X]\!]^+ = X^+ \qquad [\![A \to B]\!]^+ = {\Downarrow}([\![A]\!]^+ \to ({\uparrow}[\![B]\!]^+)) \qquad [\![A \oplus B]\!]^+ = [\![A]\!]^+ \oplus [\![B]\!]^+$$

$$[\![x]\!]^+ = x$$

$$[\![M\ N]\!]^+ = (([\![M]\!]^+.\mathsf{enter})\ [\![N]\!]^+).\mathsf{eval} \qquad [\![\lambda x.M]\!]^+ = \lambda\mathsf{enter}.\lambda x.\lambda\mathsf{eval}.[\![M]\!]^+$$

$$[\![\iota_i M]\!]^+ = \iota_i[\![M]\!]^+ \qquad [\![\mathbf{case}\, M\, \mathbf{of}\{\iota_i x_i.N_i\}]\!]^+ = \mathbf{case}\,[\![M]\!]^+\, \mathbf{of}\{\iota_i x_i.[\![N_i]\!]^+\}$$

[1] Note that this ${\Uparrow}E$ rule is an *extension* of the elimination rule for F in call-by-push-value [11], which restricts C to be only a negative type. The impact is that, unlike call-by-push-value, this language allows for *non-value terms* of positive types, similar to SML. The extension is *conservative*, because the interpretation of A_+ values is identical to call-by-push-value, whereas the interpretation of a non-value term of type A_+ would be shifted in call-by-push-value as the computation type ${\Uparrow}A_+$. This interpretation also illustrates how to compile the extended calculus to the lower-level call-by-push-value by ${\Uparrow}$-shifting following the standard encoding of call-by-value, where positive non-value terms have an explicit val wherever they may return a value. More details can be found in Appendices B and F.

It converts every type into a positive one. As such, sum types do not have to change (because, like SML, we have not restricted positive types to only classifying values as in [15]). Instead, the shifts appear in function types: to call a function, we must first enter the abstraction, perform the call, then evaluate the result.

At a basic level, these two encodings make sense from the perspective of typability (corresponding to provability in logic)—by inspection, all of the types line up with their newly-assigned polarities. But programs are meant to be run, so we care about more than just typability. At a deeper level, the encodings are *sound* with respect to equality of terms: if two terms are equal, then their encodings are also equal. We have not yet formally defined equality, so we will return to this question later in Section 5.1.

3 Polarity and sharing

So far we have considered only call-by-value and -name calculi. What about call-by-need, which models sharing and memoization for lazy computation; what would it take to add that, too? The shifts we have are no longer enough: to complete the picture we also require shifts between call-by-need and the other polarities. We need to be able to shift into and out of the positive polarity in order for call-by-need to access data like the sum type. And we also need to be able to shift into and out of the negative polarity for call-by-need to be able to access co-data like the function type. That is a total of four more shifts to connect the ordinary polarized language to the call-by-need world. The question is, how do we align the four different shifts that we saw previously? Since call-by-need only needs access to the positive world for representing data types, we use the data forms of shifts between those two. Dually, since call-by-need only needs access to the negative world for representing co-data types, we use the co-data forms of shifts between those two. We will also need a mechanism for representing sharing. The traditional representation [4] is with **let**-bindings, and so we will do the same. In all, we have:

$$A, B, C ::= A_+ \mid A_- \mid A_\star \qquad A_-, B_- ::= X^- \mid A_+ \to B_- \mid \Uparrow A_+ \mid \uparrow A_+ \mid \uparrow_\star A_\star$$

$$A_\star, B_\star ::= X^\star \mid {}_\star{\Uparrow} A_+ \mid {}_\star{\Downarrow} A_- \qquad A_+, B_+ ::= X^+ \mid A_+ \oplus B_+ \mid \Downarrow A_- \mid \downarrow A_- \mid \downarrow_\star A_\star$$

$$\frac{\Gamma \vdash M : A_\star}{\Gamma \vdash \lambda \mathsf{eval}_\star . M : \uparrow_\star A_\star}\ {\uparrow}I \qquad \frac{\Gamma \vdash M : \uparrow_\star A_\star}{\Gamma \vdash M.\mathsf{eval}_\star : A_\star}\ {\uparrow}E$$

$$\frac{\Gamma \vdash M : A_\star}{\Gamma \vdash \mathsf{box}_\star\, M : \downarrow_\star A_\star}\ {\downarrow}I \qquad \frac{\Gamma \vdash M : \downarrow_\star A_\star \quad \Gamma, x : A_\star \vdash N : C}{\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\{\mathsf{box}_\star\, x.N\} : C}\ {\downarrow}E$$

$$\frac{\Gamma \vdash M : A_+}{\Gamma \vdash \mathsf{val}_\star\, M : {}_\star{\Uparrow} A_+}\ {\Uparrow}I \qquad \frac{\Gamma \vdash M : {}_\star{\Uparrow} A \quad \Gamma, x : A_+ \vdash N : C}{\Gamma \vdash \mathbf{case}\, M\, \mathbf{of}\{\mathsf{val}_\star\, x.N\} : C}\ {\Uparrow}E$$

$$\frac{\Gamma \vdash M : A_-}{\Gamma \vdash \lambda \mathsf{enter}_\star . M : {}_\star{\Downarrow} A_-}\ {\Downarrow}I \qquad \frac{\Gamma \vdash M : {}_\star{\Downarrow} A_-}{\Gamma \vdash M.\mathsf{enter}_\star : A_-}\ {\Downarrow}E$$

$$\frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : C}{\Gamma \vdash \mathbf{let}\, x = M\, \mathbf{in}\, N : C}\ \mathit{Let}$$

Now, how can a call-by-need λ-calculus with functions and sums be encoded into this polarized setting? We effectively combine both the call-by-name and -value encodings, where a shift is used for call-by-need whenever one is used for either of the other two.

$$[\![X]\!]^\star = X^\star \qquad [\![A \to B]\!]^\star = {}_\star{\Downarrow}((\downarrow_\star [\![A]\!]^\star) \to (\uparrow_\star [\![B]\!]^\star)) \qquad [\![A \oplus B]\!]^\star = {}_\star{\Uparrow}((\downarrow_\star [\![A]\!]^\star) \oplus (\downarrow_\star [\![B]\!]^\star))$$

$$\begin{aligned}
[\![x]\!]^\star &= x \\
[\![M\ N]\!]^\star &= (([\![M]\!]^\star.\,\mathsf{enter}_\star)\ (\mathsf{box}_\star\, [\![N]\!]^\star)).\,\mathsf{eval}_\star \\
[\![\lambda x.M]\!]^\star &= \lambda \mathsf{enter}_\star.\lambda y.\, \mathbf{case}\, y\, \mathbf{of}\{\mathsf{box}_\star\, x.\lambda \mathsf{eval}_\star.[\![M]\!]^\star\} \\
[\![\iota_i M]\!]^\star &= \mathsf{val}_\star(\iota_i(\mathsf{box}_\star\, [\![M]\!]^\star)) \\
[\![\mathbf{case}\, M\, \mathbf{of}\{\iota_i x_i.N_i\}]\!]^\star &= \mathbf{case}\, [\![M]\!]^\star\, \mathbf{of}\{\mathsf{val}_\star(\iota_i(\mathsf{box}_\star\, x_i)).[\![N_i]\!]^\star\}
\end{aligned}$$

The key thing to notice here is what is shared and what is not, to ensure that the encoding correctly aligns with call-by-need evaluation. Both the shifts *into* $\star$, the data type ${}_\star{\Uparrow}A_+$ and co-data type ${}_\star{\Downarrow}A_-$, result in terms that can be shared by a **let**. But the shifts *out of* $\star$ are different: the content M of $\mathsf{box}_\star\, M : {\downarrow_\star} A_\star$ is still shared, like a data structure, but the content M of $\lambda\mathsf{eval}_\star.M : {\uparrow_\star} A_\star$ is not, like a λ-abstraction. Therefore, the encoding of an injection $[\![\iota_i M]\!]^\star$ shares the computation of $[\![M]\!]^\star$ throughout the lifetime of the returned value, as for the argument of a function call:

$$[\![\mathbf{case}\, \iota_i M\, \mathbf{of}\, \{\iota_i x_i.N_i\}]\!]^\star = \mathbf{let}\, x_i = [\![M]\!]^\star\, \mathbf{in}\, [\![N_i]\!]^\star \qquad [\![(\lambda x.M)N]\!]^\star = \mathbf{let}\, x = [\![N]\!]^\star\, \mathbf{in}\, [\![M]\!]^\star$$

Whereas, the encoding of a function $[\![\lambda x.M]\!]^\star$, being a value, re-computes $[\![M]\!]^\star$ every time the function is used, which is formalized by the equational theory in Section 4.4.

4 A multi-discipline intermediate language

So far, we have only considered how sharing interacts with polarity in a small language with functions and sums, but programming languages generally have more than just those two types. For example, both SML and Haskell have pairs so we should include those, too, but when do we have enough of a "representative" basis of types that serves as the core kernel language for the general source language? To define our core intermediate language, we will follow the standard practice (as in CPS) of first defining a more general source language, and then identifying the core sub-language that the entire source can be translated into.

The biggest issue is that faithfully encoding types of various disciplines into a core set of primitives is more subtle than it may at first seem. For example, using Haskell's algebraic data type declaration mechanism, we can define both a binary and ternary sum:

$$\begin{array}{l} \mathbf{data}\ \mathsf{Either}\ a\ b\ \mathbf{where} \\ \qquad \mathsf{Left} : a \to \mathsf{Either}\ a\ b \\ \qquad \mathsf{Right} : b \to \mathsf{Either}\ a\ b \end{array} \qquad \begin{array}{l} \mathbf{data}\ \mathsf{Either3}\ a\ b\ c\ \mathbf{where} \\ \qquad \mathsf{Choice1} : a \to \mathsf{Either3}\ a\ b\ c \\ \qquad \mathsf{Choice2} : b \to \mathsf{Either3}\ a\ b\ c \\ \qquad \mathsf{Choice3} : c \to \mathsf{Either3}\ a\ b\ c \end{array}$$

But $\mathsf{Either}\ a\ (\mathsf{Either}\ b\ c)$ does not faithfully represent $\mathsf{Either3}\ a\ b\ c$ in Haskell, even though it does in SML. The two types are convertible:

$$\begin{aligned} \mathit{nest}(\mathsf{Choice1}\, x) &= \mathsf{Left}\, x \\ \mathit{nest}(\mathsf{Choice2}\, y) &= \mathsf{Right}(\mathsf{Left}\, y) \\ \mathit{nest}(\mathsf{Choice3}\, z) &= \mathsf{Right}(\mathsf{Right}\, z) \end{aligned} \qquad \begin{aligned} \mathit{unnest}(\mathsf{Left}\, x) &= \mathsf{Choice1}\, x \\ \mathit{unnest}(\mathsf{Right}(\mathsf{Left}\, y)) &= \mathsf{Choice2}\, y \\ \mathit{unnest}(\mathsf{Right}(\mathsf{Right}\, z)) &= \mathsf{Choice3}\, z \end{aligned}$$

but they do not describe the same values. $\mathsf{Either}\ a\ (\mathsf{Either}\ b\ c)$ types both the observably distinct terms Ω and $\mathsf{Right}\,\Omega$—which can be distinguished by pattern matching—but conversion to $\mathsf{Either3}\ a\ b\ c$ collapses them both to Ω. This is not just an issue of needing $n$ary tuples and sums, the same issue arises when pairs and sums are nested with each other.

To ensure that we model a general enough source language, we will consider one that is *extensible* (*i.e.,* allows for user-defined types encompassing many types found in functional languages) and *multi-discipline* (*i.e.,* allows for programs that mix call-by-value, -name, and -need evaluation). These two features interact with one another: user-defined types can combine parts with different calling conventions. But even though users can define many different types, there is still a fixed core set of types, $\mathcal{F}$, capable of representing them all. For example, an extensible and multi-discipline calculus encompasses both the source and target of the three encodings showed previously in Sections 2 and 3. We now look at the full core intermediate language $\mathcal{F}$, and how to translate general source programs into the core $\mathcal{F}$.

Simple (co-)data types

$$\begin{array}{lll}
\mathbf{data}\,(X{:}+) \oplus (Y{:}+) : + \,\mathbf{where} & \mathbf{data}\,(X{:}+) \otimes (Y{:}+) : + \,\mathbf{where} & \mathbf{data}\, 0 : + \,\mathbf{where} \\
\quad \iota_1 : (X{:}+ \vdash X \oplus Y) & \quad (\_,\_) : (X{:}+, Y{:}+ \vdash X \otimes Y) & \mathbf{data}\, 1 : + \,\mathbf{where}\, () : (\vdash 1) \\
\quad \iota_2 : (Y{:}+ \vdash X \oplus Y) & & \\
\mathbf{codata}\,(X{:}-) \,\&\, (Y{:}-) : - \,\mathbf{where} & \mathbf{codata}\,\top : - \,\mathbf{where} & \mathbf{codata}\,(X{:}+) \to (Y{:}-) : - \,\mathbf{where} \\
\quad \pi_1 : (\mid X \,\&\, Y \vdash X{:}-) & & \quad \mathsf{call} : (X{:}+ \mid X \to Y \vdash Y{:}-) \\
\quad \pi_2 : (\mid X \,\&\, Y \vdash Y{:}-) & &
\end{array}$$

Quantifier (co-)data types

$$\begin{array}{l}
\mathbf{data}\,\exists_k(X{:}k{\to}+) : + \,\mathbf{where} \\
\quad \mathsf{pack} : (X\ Y{:}+ \vdash^{Y:k} \exists_k X) \\
\mathbf{codata}\,\forall_k(X{:}k{\to}-) : - \,\mathbf{where} \\
\quad \mathsf{spec} : (\mid \forall_k X \vdash^{Y:k} X\ Y{:}-)
\end{array}$$

Polarity shift (co-)data types

$$\begin{array}{ll}
\mathbf{data}\,{\downarrow_{\mathcal{S}}}(X{:}\mathcal{S}) : + \,\mathbf{where} & \mathbf{data}\,{}_{\mathcal{S}}{\Uparrow}(X{:}+) : \mathcal{S} \,\mathbf{where} \\
\quad \mathsf{box}_{\mathcal{S}} : (X{:}\mathcal{S} \vdash {\downarrow_{\mathcal{S}}} X) & \quad \mathsf{val}_{\mathcal{S}} : (X{:}+ \vdash {}_{\mathcal{S}}{\Uparrow} X) \\
\mathbf{codata}\,{\uparrow_{\mathcal{S}}}(X{:}\mathcal{S}) : - \,\mathbf{where} & \mathbf{codata}\,{}_{\mathcal{S}}{\Downarrow}(X{:}-) : \mathcal{S} \,\mathbf{where} \\
\quad \mathsf{eval}_{\mathcal{S}} : (\mid {\uparrow_{\mathcal{S}}} X \vdash X{:}\mathcal{S}) & \quad \mathsf{enter}_{\mathcal{S}} : (\mid {}_{\mathcal{S}}{\Downarrow} X \vdash X{:}-)
\end{array}$$

Figure 1 The $\mathcal{F}$ functional core set of (co-)data declarations.

4.1 The functional core intermediate language: $\mathcal{F}$

Our language allows for user-defined data and co-data types. A data type introduces a number of constructors for building values of the type, a co-data type introduces a number of *observers* for observing or interacting with values of the type. Figure 1 presents some important examples that define a *core* set of types, $\mathcal{F}$. The calculus instantiated with just the $\mathcal{F}$ types serves as our core intermediate language, as it contains all the needed functionality.

The data and codata declarations for $\oplus$ and $\to$ correspond to the polarized sum and function types from Section 2, with a slight change of notation: we write $X : +$ instead of X^+. The data declaration of $\oplus$ defines its two *constructors* ι_1 and ι_2, and dually the co-data declaration for $\to$ defines its one *observer* call. The terms of the resulting sum type are exactly as they were presented in Section 2. The function type uses a slightly more verbose notation than the λ-calculus for the sake of regularity: instead of $\lambda x.M$ we have $\lambda\{\mathsf{call}\,\boldsymbol{x}.M\}$ and instead of $M\ N$ we have $M.\mathsf{call}\,N$. That is, dual to a **case** matching on the pattern of a data structure, a λ-abstraction matches on the co-pattern of a co-data observation like $\mathsf{call}\,x$. Besides changing notation, the meaning is the same [7].

There are some points to notice about these two declarations. First, disciplines can be mixed within a single declaration, which is used to define the polarized $\to$ function space that accepts a call-by-value $(+)$ input and returns a call-by-name $(-)$ result, but other combinations are also possible. Second, instead of the function type arrow notation to assign a type to the constructors and observers, we use the turnstyle $(\vdash)$ of a typing judgement. This avoids the issue that a function type arrow already dictates the disciplines for the argument and result, limiting our freedom of choice.

The rest of the core $\mathcal{F}$ types exercise all the functionality of our declaration mechanism. The nullary version of sums (0) has no constructors and an empty $\mathbf{case}\,M\,\mathbf{of}\,\{\}$. We have binary and nullary tuples ($\otimes$, 1), which have terms of the form (M, N) and $()$ and are used by $\mathbf{case}\,M\,\mathbf{of}\,\{(x, y).M\}$ and $\mathbf{case}\,M\,\mathbf{of}\,\{().M\}$, respectively. We also have binary and nullary products ($\&$, $\top$), with two and zero observers, respectively. The terms of binary products have the form $\lambda\{\pi_1.M | \pi_2.N\}$ and can be observed as $M.\pi_i$, and the nullary product has the term $\lambda\{\}$ which cannot be observed in any way. The shifts are also generalized to operate generically over the choice of call-by-name $(-)$, call-by-value $(+)$, and call-by-need $(\star)$, which

$$
\begin{array}{l}
A, B, C ::= X \mid \mathsf{F} \mid \lambda \boldsymbol{X}.A \mid A\ B \qquad \boldsymbol{X} ::= X{:}k \qquad k, l ::= \mathcal{S} \mid k \to l \qquad \mathcal{R}, \mathcal{S}, \mathcal{T} ::= + \mid - \mid \star \\
\mathit{decl} ::= \mathbf{data}\, \mathsf{F}(X{:}k)..: \mathcal{S}\, \mathbf{where}\, \mathsf{K} : (A{:}\mathcal{T}.. \vdash^{\boldsymbol{X}..} \mathsf{F}\, X..).. \\
\qquad\quad \mid \mathbf{codata}\, \mathsf{G}(X{:}k)..: \mathcal{S}\, \mathbf{where}\, \mathsf{O} : (A{:}\mathcal{T}.. \mid \mathsf{G}\, X.. \vdash^{\boldsymbol{X}..} B{:}\mathcal{R}).. \\
\quad p ::= \mathsf{K}\, \boldsymbol{X}..\boldsymbol{y}.. \qquad q ::= \mathsf{O}\, \boldsymbol{X}..\boldsymbol{y}.. \qquad \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} ::= x{:}A \\
M, N ::= x \mid \mathbf{let}\, \boldsymbol{x} = M\, \mathbf{in}\, N \mid M.\mathsf{O}\, B..N.. \mid \mathsf{K}\, B..M.. \mid \lambda\{q_i.M_i \overset{i}{..}\} \mid \mathbf{case}\, M\, \mathbf{of} \{p_i.M_i \overset{i}{..}\}
\end{array}
$$

Figure 2 Syntax of a total, pure functional calculus with (co-)data.

we denote by $\mathcal{S}$. The pair of shifts between $+$ ($\downarrow_{\mathcal{S}}$, ${}_{\mathcal{S}}\Uparrow$) and $-$ ($\uparrow_{\mathcal{S}}$, ${}_{\mathcal{S}}\Downarrow$) for each $\mathcal{S}$ has the same form as in Section 3, where we omit the annotation $\mathcal{S}$ when it is clear from the context.

The last piece of functionality is the ability to introduce *locally quantified* types in a constructor or observer. These quantified type variables are listed as a superscript to the turnstyle, and allow user-defined types to perform type abstraction and polymorphism. Two important examples of type abstraction shown in Figure 1 are the universal ($\forall_k$) and existential ($\exists_k$) quantifiers, which apply to a type function $\lambda X{:}k.A$. We will use the shorthand $\forall X{:}k.A$ for $\forall_k(\lambda X{:}k.A)$ and $\exists X{:}k.A$ for $\exists_k(\lambda X{:}k.A)$. The treatment of quantified types is analogous to System F_ω, where types appear in terms as parameters. For example, the term $\lambda\{\mathsf{spec}\, Y{:}k.M\} : \forall Y{:}k.A$ abstracts over the type variable Y in M, and a polymorphic $M : \forall Y{:}k.A$ can be observed via specialization as $M.\mathsf{spec}\, B : A[B/Y]$. Dually, the term $\mathsf{pack}\, B\ M : \exists Y{:}k.A$ hides the type B in the term $M : A[B/Y]$, and an existential $M : \exists Y{:}k.A$ can be unpacked by pattern matching as $\mathbf{case}\, M\, \mathbf{of}\{\mathsf{pack}\ (Y{:}k)\ (x{:}A).N\}$.

4.2 Syntax

The syntax of our extensible and multi-discipline λ-calculus is given in Figure 2. We refer to each of the three kinds of types ($+$, $-$ and $\star$) as a *discipline* which is denoted by the meta-variables $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$. A data declaration has the general form

$$
\begin{array}{l}
\mathbf{data}\, \mathsf{F}(X_1{:}k_1)..(X_n{:}k_n) : \mathcal{S}\, \mathbf{where}\, \mathsf{K}_1 : (A_{11} : \mathcal{T}_{11}..A_{1n} : \mathcal{T}_{1n} \vdash \mathsf{F}\, X_1..X_n) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_m : (A_{m1} : \mathcal{T}_{m1}..A_{mn} : \mathcal{T}_{mn} \vdash \mathsf{F}\, X_1..X_n)
\end{array}
$$

which declares a new type constructor F and value constructors $\mathsf{K}_1 \dots \mathsf{K}_m$. The dual co-data declaration combines the concepts of functions and products, having the general form

$$
\begin{array}{l}
\mathbf{codata}\, \mathsf{G}(X_1{:}k_1)..(X_n{:}k_n) : \mathcal{S}\, \mathbf{where}\, \mathsf{O}_1 : (A_{11} : \mathcal{T}_{11}..A_{1n} : \mathcal{T}_{1n} \mid \mathsf{G}\, X_1..X_n \vdash B_1 : \mathcal{R}_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_m : (A_{m1} : \mathcal{T}_{m1}..A_{mn} : \mathcal{T}_{mn} \mid \mathsf{G}\, X_1..X_n \vdash B_m : \mathcal{R}_m)
\end{array}
$$

Since an observer is dual to a constructor, the signature is flipped around: the signature for O_1 above can be read as "given parameters of types A_{11} to A_{1n}, O_1 can observe a value of type $\mathsf{G}\, X_1..X_n$ to obtain a result of type B_1."[2]

Notice that we can *also* declare types corresponding to purely call-by-value, -name, and

[2] Both of these notions of data and co-data correspond to *finitary* types, since declarations allow for a finite number of constructors or observers for all data and co-data types, respectively. We could just as well generalize declarations with an infinite number of constructors or observers to also capture *infinitary* types at the usual cost of having infinite branching in **case**s and λs. Since this generalization is entirely mechanical and does not enhance the main argument, we leave it out of the presentation.

$$\frac{\Theta, X : k \vdash_{\mathcal{G}} A : l}{\Theta \vdash_{\mathcal{G}} \lambda X{:}k.A : k \to l} \qquad \frac{\Theta \vdash_{\mathcal{G}} A : k \to l \quad \Theta \vdash_{\mathcal{G}} B : k}{\Theta \vdash_{\mathcal{G}} A\ B : l} \qquad \frac{}{\Theta, X : k \vdash_{\mathcal{G}} X : k} \qquad \frac{(\Theta \vdash_{\mathcal{G}} A : \mathcal{T})..}{(x : A : \mathcal{T}.. \vdash_{\mathcal{G}}^{\Theta})\,\mathbf{ctx}}$$

$$\frac{(\Gamma \vdash_{\mathcal{G}}^{\Theta})\,\mathbf{ctx} \quad \Theta \vdash_{\mathcal{G}} A : \mathcal{S}}{\Gamma, x : A : \mathcal{S} \vdash_{\mathcal{G}}^{\Theta} x : A : \mathcal{S}} \qquad \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \quad \Gamma, x : A : \mathcal{S} \vdash_{\mathcal{G}}^{\Theta} N : C : \mathcal{R}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mathbf{let}\, x{:}A = M\, \mathbf{in}\, N : C : \mathcal{R}} \qquad \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \quad A =_{\beta\eta} B}{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : B : \mathcal{S}}$$

Given $\mathbf{data}\, \mathsf{F}(X{:}k).. : \mathcal{S}\, \mathbf{where}\, \mathsf{K}_i : (A_{ij} : \mathcal{T}_{ij} \overset{j}{..} \vdash^{Y_{ij}:l_{ij} \overset{j}{..}} \mathsf{F}(X..)) \overset{i}{..} \in \mathcal{G}$, we have the rules:

$$\frac{}{\Theta \vdash_{\mathcal{G}} \mathsf{F} : k \to ..\mathcal{S}}$$

$$\frac{(\Gamma \vdash_{\mathcal{G}}^{\Theta})\,\mathbf{ctx} \quad \Theta \vdash_{\mathcal{G}} \mathsf{F}\, C.. : \mathcal{S} \quad (\Theta \vdash_{\mathcal{G}} B_j : l_{ij}) \overset{j}{..} \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} M_j : A_{ij}[C/X.., B_j/Y_{ij} \overset{j}{..}] : \mathcal{T}_{ij}) \overset{j}{..}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mathsf{K}_i\, B_j \overset{j}{..}\ M_j \overset{j}{..} : \mathsf{F}\, C.. : \mathcal{S}}\ \mathsf{F}I_i$$

$$\frac{\Theta \vdash_{\mathcal{G}} C : \mathcal{R} \quad \Gamma \vdash_{\mathcal{G}}^{\Theta} M : \mathsf{F}\, B.. : \mathcal{S} \quad (\Gamma, x_{ij} : A_{ij}[B/X..] : \mathcal{T}_{ij} \overset{j}{..} \vdash_{\mathcal{G}}^{\Theta, Y_{ij}:l_{ij} \overset{j}{..}} N_i : C : \mathcal{R}) \overset{i}{..}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mathbf{case}\, M\, \mathbf{of} \{(\mathsf{K}_i\, Y_{ij}{:}l_{ij} \overset{j}{..}\ x_{ij}{:}A_{ij} \overset{j}{..}).N_i \overset{i}{..}\} : C : \mathcal{R}}\ \mathsf{F}E$$

Given $\mathbf{codata}\, \mathsf{G}(X{:}k).. : \mathcal{S}\, \mathbf{where}\, \mathsf{O}_i : (A_{ij} : \mathcal{T}_{ij} \overset{j}{..} \mid \mathsf{G}(X..) \vdash^{Y_{ij}:l_{ij} \overset{j}{..}} B_i : \mathcal{R}_i) \overset{i}{..} \in \mathcal{G}$, we have the rules:

$$\frac{}{\Theta \vdash_{\mathcal{G}} \mathsf{G} : k \to ..\mathcal{S}}$$

$$\frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : \mathsf{G}\, C'.. : \mathcal{S} \quad (\Theta \vdash_{\mathcal{G}} C_j : l_{ij}) \overset{j}{..} \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} N_j : A_{ij}[C'/X.., C_j/Y_{ij} \overset{j}{..}] : \mathcal{T}_{ij}) \overset{j}{..}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} M.\mathsf{O}_i\, C_j \overset{j}{..}\ N_j \overset{j}{..} : B_i : \mathcal{R}_i}\ \mathsf{G}E_i$$

$$\frac{(\Gamma \vdash_{\mathcal{G}}^{\Theta})\,\mathbf{ctx} \quad \Theta \vdash_{\mathcal{G}} \mathsf{G}\, C.. : \mathcal{S} \quad (\Gamma, x_{ij} : A_{ij}[C/X..] : \mathcal{T}_{ij} \overset{j}{..} \vdash_{\mathcal{G}}^{\Theta, Y_{ij}:l_{ij} \overset{j}{..}} N_i : B_i : \mathcal{R}_i) \overset{i}{..}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \lambda\big\{(\mathsf{O}_i\, Y_{ij}{:}l_{ij} \overset{j}{..}\ x_{ij}{:}A_{ij} \overset{j}{..}).N_i \overset{i}{..}\big\} : \mathsf{G}\, C.. : \mathcal{S}}\ \mathsf{G}I$$

Figure 3 Type system for the pure functional calculus.

-need versions of sums and functions by instantiating $\mathcal{S}$ with $+$, $-$, and $\star$, respectively:

$$\begin{array}{ll}
\mathbf{data}\, (X{:}\mathcal{S}) \oplus^{\mathcal{S}} (Y{:}\mathcal{S}) : \mathcal{S}\, \mathbf{where} & \mathbf{codata}\, (X{:}\mathcal{S}) \xrightarrow{\mathcal{S}} (Y{:}\mathcal{S}) : \mathcal{S}\, \mathbf{where} \\
\quad \iota_1^{\mathcal{S}} : (X{:}\mathcal{S} \vdash X \oplus Y) & \quad \mathsf{call}^{\mathcal{S}} : (X{:}\mathcal{S} \mid X \xrightarrow{\mathcal{S}} Y \vdash Y{:}\mathcal{S}) \\
\quad \iota_2^{\mathcal{S}} : (Y{:}\mathcal{S} \vdash X \oplus Y) &
\end{array}$$

So the extensible language subsumes *all* the languages shown in Sections 2 and 3.

4.3 Type System

The kind and type system is given in Figure 3. In the style of system F_ω, the kind system is just the simply-typed λ-calculus at the level of types—so type variables, functions, and applications—where each connective is a constant of the kind declared in the global environment $\mathcal{G}$. It also includes the judgement $(\Gamma \vdash_{\mathcal{F}}^{\Theta})\,\mathbf{ctx}$ for checking that a typing context is well-formed, meaning that each variable in Γ is assigned a well-kinded type with respect to the type variables in Θ and global environment $\mathcal{G}$.

The typing judgement for terms is $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S}$, where $\mathcal{G}$ is a list of declarations, $\Theta = X : k..$ assigns kinds to type variables, and $\Gamma = x : A : \mathcal{S}..$ assigns explicitly-kinded types to value variables. The interesting feature of the type system is the use of the two-level judgement $M : A : \mathcal{S}$, which has the intended interpretation that "M is of type A *and* A is of kind $\mathcal{S}$." The purpose of this compound statement is to ensure that the introduction rules do not create ill-kinded types by mistake. This maintains the invariant that if $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S}$ is derivable then so is $(\Gamma \vdash_{\mathcal{G}}^{\Theta})\,\mathbf{ctx}$ and $\Theta \vdash_{\mathcal{G}} A : \mathcal{S}$.

For example, in the $\mathcal{F}$ environment from Figure 1, a type like $A \otimes B$ requires that both A and B are of kind $+$, so the $\otimes$ introduction rule for closed pairs of closed types is:

$$\frac{\vdash_{\mathcal{F}} M : A : + \quad \vdash_{\mathcal{F}} N : A : +}{\vdash_{\mathcal{F}} (M, N) : A \otimes B : +}\ \otimes I$$

$$V ::= V_{\mathcal{S}} : A : \mathcal{S} \qquad V_{+} ::= x \mid \mathsf{K}\, B..V.. \mid \lambda\{q_i.M_i \mid \overset{i}{..}\} \qquad V_{-} ::= M \qquad V_{\star} ::= V_{+}$$
$$F ::= \square.\mathsf{O}\, B..V.. \mid \mathbf{case}\,\square\,\mathbf{of}\{p_i.M_i \overset{i}{..}\} \mid \mathbf{let}\, x{:}A{:}{+} = \square\,\mathbf{in}\, M \mid \mathbf{let}\, x{:}A{:}\star = \square\,\mathbf{in}\, H[E[x]]$$
$$E ::= \square \mid F[E] \qquad U ::= \mathbf{let}\, x{:}A{:}\star = M\,\mathbf{in}\,\square \qquad H ::= \square \mid U[H]$$
$$T ::= \mathbf{let}\,\boldsymbol{x} = M\,\mathbf{in}\,\square \mid \mathbf{case}\, M\,\mathbf{of}\{p_i.\square \mid \overset{i}{..}\}$$

$$(\beta_{let}) \qquad \mathbf{let}\,\boldsymbol{x} = V\,\mathbf{in}\, M \sim M[V/\boldsymbol{x}]$$
$$(\beta_{\mathsf{O}}) \qquad \lambda\{..|(\mathsf{O}\,\boldsymbol{Y}..\boldsymbol{x}..).M|..\}.\mathsf{O}\, B..\ N.. \sim \mathbf{let}\,\boldsymbol{x} = N..\,\mathbf{in}\, M[B/\boldsymbol{Y}..]$$
$$(\beta_{\mathsf{K}}) \qquad \mathbf{case}\,\mathsf{K}\, B..N..\,\mathbf{of}\{..|(\mathsf{K}\,\boldsymbol{Y}..\boldsymbol{x}..).M|..\} \sim \mathbf{let}\,\boldsymbol{x} = N..\,\mathbf{in}\, M[B/\boldsymbol{Y}..]$$
$$(\eta_{let}) \qquad \mathbf{let}\, x{:}A = M\,\mathbf{in}\, x \sim M$$
$$(\eta_{\mathsf{G}}) \qquad \lambda\{q_i.(x.q_i) \mid \overset{i}{..}\} \sim x$$
$$(\eta_{\mathsf{F}}) \qquad \mathbf{case}\, M\,\mathbf{of}\{p_i.p_i \mid \overset{i}{..}\} \sim M$$
$$(\kappa_F) \qquad F[T[M_i \overset{i}{..}]] \sim T[F[M_i] \overset{i}{..}]$$
$$(\chi^{\mathcal{S}}) \qquad \mathbf{let}\, y{:}B{:}\mathcal{S} = \mathbf{let}\, x{:}A{:}\mathcal{S} = M_1\,\mathbf{in}\, M_2\,\mathbf{in}\, N \sim \mathbf{let}\, x{:}A{:}\mathcal{S} = M_1\,\mathbf{in}\,\mathbf{let}\, y{:}B{:}\mathcal{S} = M_2\,\mathbf{in}\, N$$
$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{G}} M : A : \mathcal{S} \quad M \sim M' \quad \Gamma \vdash^{\Theta}_{\mathcal{G}} M' : A : \mathcal{S}}{\Gamma \vdash^{\Theta}_{\mathcal{G}} M = M' : A : \mathcal{S}}$$

plus compatibility, reflexivity, symmetry, transitivity

Figure 4 Equational theory for the pure functional calculus.

The constraint that $A : +$ and $B : +$ in the premises to $\otimes I$ ensures that $A \otimes B$ is indeed a type of $+$. This idea is also extended to variables introduced by pattern matching at a specific type by placing a two-level constraint on the variables. For example, the $\rightarrow$ introduction rule for closed function abstractions is:

$$\frac{x : A : + \vdash_{\mathcal{F}} M : B : -}{\vdash_{\mathcal{F}} \lambda\{\mathsf{call}(x{:}A).M\} : A \rightarrow B : -}\ {\rightarrow}I$$

Notice how when the variable x is added to the environment, it has the type assignment $x : A : +$ because the declared argument type of $\rightarrow$ must be some call-by-value type. If the premise of ${\rightarrow}I$ holds, then $A : +$ and $B : -$, so $A \rightarrow B$ is a well-formed type of $-$.

Finally, we also need to check that a global environment $\mathcal{G}$ is well-formed, written $\vdash \mathcal{G}$, which amounts to checking that each declaration is in turn like so:

$$\frac{(X : k..,Y : l.. \vdash_{\mathcal{G}} A : \mathcal{T})..}{\mathcal{G} \vdash \mathbf{data}\,\mathsf{F}(X{:}k)..:\mathcal{S}\,\mathbf{where}\,\mathsf{K} : (A : \mathcal{T}.. \vdash^{Y:l..} \mathsf{F}\, X..)..}$$
$$\frac{(X : k..,Y : l.. \vdash_{\mathcal{G}} A : \mathcal{T}).. \quad (X : k..,Y : l.. \vdash_{\mathcal{G}} B : \mathcal{R})..}{\mathcal{G} \vdash \mathbf{codata}\,\mathsf{G}(X{:}k)..:\mathcal{S}\,\mathbf{where}\,\mathsf{O} : (A : \mathcal{T}.. \mid \mathsf{G}\, X.. \vdash^{Y:l..} B : \mathcal{R})..}$$

And we say that $\mathcal{G}'$ *extends* $\mathcal{G}$ if it contains all declarations in $\mathcal{G}$.

4.4 Equational Theory

The equational theory, given in Figure 4, equates two terms of the same type that behave the same in any well-typed context.[3] The axioms of equality are given by the relation $\sim$, and the typed equality judgement is $\Gamma \vdash^{\Theta}_{\mathcal{G}} M = N : A : \mathcal{S}$. Because of the multi-discipline nature of terms, the main challenge is deciding when terms are substitutable, which controls when the

[3] See Appendices E and I for the operational semantics and its relationship to equality.

β_{let} axiom can fire. For example, $\mathbf{let}\,\boldsymbol{x} = M\,\mathbf{in}\,N$ should immediately substitute M without further evaluation if it is a call-by-name binding, but should evaluate M to a value first before substitution if it is call-by-value. And we need the ability to reason about program fragments (*i.e.,* open terms of any type) wherein a variable x acts like a value in call-by-value *only* if it stands for a value, *i.e.,* we can only substitute values and not arbitrary terms for a call-by-value variable. Thus, we link up the static and dynamic semantics of disciplines: each base kind $\mathcal{S}$ is associated with a different set of substitutable terms $V_{\mathcal{S}}$ called *values*. The set of values for $+$ is the most strict (including only variables, λ-abstractions, and constructions $p[\rho]$ built by plugging in values for the holes in a pattern), $-$ is the most relaxed (admitting every term as substitutable), and $\star$ shares the same notion of value as $+$. A true value, then, is a term $V_{\mathcal{S}}$ belonging to a type of kind $\mathcal{S}$, *i.e.,* $V_{\mathcal{S}} : A : \mathcal{S}$. This way, the calling convention is aligned in both the static realm of types are and dynamic realm of evaluation.

The generic β_{let} axiom relies on the fact that the left-hand side of the axiom is well-typed and every type belongs to (at most) one kind; given $\mathbf{let}\,x{:}A = V\,\mathbf{in}\,M$, then it must be that $A : \mathcal{S}$ and V is of the form $V_{\mathcal{S}} : A : \mathcal{S}$ (both in the current environment). So if $x : A \,\&\, B : -$, then every well-typed binding is subject to substitution via β_{let}, but if $x : A \otimes B : +$ then only a value V_+ in the sense of call-by-value can be substituted. The corresponding extensionality axiom η_{let} eliminates a trivial **let** binding.

The β_{K} and β_{O} axioms match against a constructor K or observer O, respectively, by selecting the matching response within a **case** or λ-abstraction and binding the parameters via a **let**. Special cases of these axioms for a sum injection and function call are:

$$\mathbf{case}\,\iota_i M\,\mathbf{of}\{\iota_1\boldsymbol{x}_1.N_1 \mid \iota_2\boldsymbol{x}_2.N_2\} \sim_{\beta_{\iota_i}} \mathbf{let}\,\boldsymbol{x}_i = M\,\mathbf{in}\,N_i \qquad \lambda\{\mathsf{call}\,\boldsymbol{x}.N\}.\mathsf{call}\,M \sim_{\beta_{\mathsf{call}}} \mathbf{let}\,\boldsymbol{x} = M\,\mathbf{in}\,N$$

The corresponding extensionality axioms η_{G} and η_{F} apply to each co-data type G and data type F to eliminate a trivial λ and **case**, respectively, and again rely on the fact that the left-hand side of the axiom is well-typed to be sensible. The special cases of these axioms for the sum ($\oplus$) and function ($\to$) connectives of $\mathcal{F}$ are:

$$\mathbf{case}\,M\,\mathbf{of}\{\iota_1 x{:}A.\iota_1 x \mid \iota_2 y{:}B.\iota_2 y\} \sim_{\eta_\oplus} M \qquad \lambda\{\mathsf{call}\,y{:}A.(x.\mathsf{call}\,y)\} \sim_{\eta_\to} x$$

The κ_F axiom implements *commutative conversions* which permute a *frame* F of an evaluation context (E) with a *tail* context T, which brings together the frame with the return result of a block-style expression like a **let** or **case**. Frames represent the building blocks of contexts that demand a result from their hole $\square$. The cases for frames are an observation parameterized by values ($\square.\mathsf{O}\,B..V..$), case analysis ($\mathbf{case}\,\square\,\mathbf{of}\{\dots\}$), a call-by-value binding ($\mathbf{let}\,x{:}A{:}{+} = \square\,\mathbf{in}\,M$), or a call-by-need binding which is needed in its body ($\mathbf{let}\,x{:}A{:}{\star} = \square\,\mathbf{in}\,H[E[x]]$). As per call-by-need evaluation, variable x is *needed* when it appears in the eye of an evaluation context E, in the context of a *heap* H of other call-by-need bindings for different variables. Tail contexts point out where results are returned from block-style expressions, so the body of any **let** ($\mathbf{let}\,\boldsymbol{x} = M\,\mathbf{in}\,\square$) or the branches of any **case** ($\mathbf{case}\,M\,\mathbf{of}\{p.\square..\}$). Since a **case** can have zero or more branches, a tail context can have zero or more holes.

Finally, the $\chi^{\mathcal{S}}$ axiom re-associates nested **let** bindings, so long as the discipline of their bindings match. The restriction to matching disciplines is because not all combinations are actually associative [15]; namely the following two ways of nesting call-by-value and -name **let**s are *not* necessarily the same when M_1 causes an effect:

$$(\mathbf{let}\,y{:}B{:}{-} = (\mathbf{let}\,x{:}A{:}{+} = M_1\,\mathbf{in}\,M_2)\,\mathbf{in}\,N) \neq (\mathbf{let}\,x{:}A{:}{+} = M_1\,\mathbf{in}\,\mathbf{let}\,y{:}B{:}{-} = M_2\,\mathbf{in}\,N)$$

In the above, the right-hand side evaluates M_1 first, but the left-hand side first substitutes $\mathbf{let}\,x{:}A{:}{+} = M_1\,\mathbf{in}\,M_2$ for y, potentially erasing or duplicating the effect of M_1. For example, when M_1 is the infinite loop Ω and N is a constant result z which does not depend on y, then the right-hand side loops forever, but the left-hand side just returns z. But when the disciplines match, re-association is sound. In particular, notice that the χ^- instance of the axiom is derivable from β_{let}, and the χ^+ instance of the axiom is derivable from κ_F. The only truly novel instance of re-association is for call-by-need, which generalizes the special case of κ_F when the outer variable y happens to be needed.

Some of the axioms of this theory may appear to be weak, but nonetheless they let us derive some useful equalities. For example, the λ-calculus' full η law for functions

$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A \to B : - \qquad x \notin \Gamma}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\{\mathsf{call}\, x{:}A.(M.\mathsf{call}\, x)\} = M : A \to B : -}$$

is derivable from $\eta_{\to}$ and β_{let}. Furthermore, the sum extensionality law from Section 2, and nullary version for the void type 0

$$\Gamma, x : A_1 \oplus A_2 : + \vdash^{\Theta}_{\mathcal{F}} M = \mathbf{case}\, x\, \mathbf{of}\{\iota_i(y_i{:}A_i).M[\iota_i y_i/x] \overset{i}{\vdots}\} : C : \mathcal{R}$$
$$\Gamma, x : 0 : + \vdash^{\Theta}_{\mathcal{F}} M = \mathbf{case}\, x\, \mathbf{of}\{\} : C : \mathcal{R}$$

are derived from the $\eta_{\oplus}$, η_0, κ_F, and β_{let} axioms. So typed equality of this strongly-normalizing calculus captures "strong sums" (à la [16]). Additionally, the laws of monadic binding [14] (bind-and-return and bind reassociation) and the F functor of call-by-push-value [11] are instances of the generic $\beta\eta\kappa$ laws for the shift data type ${}_{\mathcal{S}}{\Uparrow}A$:

$$\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, \mathsf{box}_{\mathcal{S}}\, V\, \mathbf{of}\{\mathsf{box}_{\mathcal{S}}\, \boldsymbol{x}.M\} =_{\beta_{{}_{\mathcal{S}}\Uparrow}\beta_{let}} M[V/\boldsymbol{x}] : C : \mathcal{R}$$
$$\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{\mathsf{box}_{\mathcal{S}}(x{:}A).\mathsf{box}_{\mathcal{S}}\, x\} =_{\eta_p} M : {}_{\mathcal{S}}{\Uparrow}A : \mathcal{S}$$
$$\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\,(\mathbf{case}\, M\, \mathbf{of}\{\mathsf{box}_{\mathcal{S}}\, \boldsymbol{x}.N\})\, \mathbf{of}\{\mathsf{box}_{\mathcal{T}}\, \boldsymbol{y}.N'\} \qquad : C : \mathcal{R}$$
$$=_{\kappa_F} \mathbf{case}\, M\, \mathbf{of}\{\mathsf{box}_{\mathcal{S}}\, \boldsymbol{x}.\mathbf{case}\, N\, \mathbf{of}\{\mathsf{box}_{\mathcal{T}}\, \boldsymbol{y}.N'\}\}$$

Note that in the third equality, commuting conversions can reassociate ${}_{\mathcal{S}}{\Uparrow}A$ and ${}_{\mathcal{T}}{\Uparrow}B$ bindings for *any* combination of $\mathcal{S}$ and $\mathcal{T}$, including $-$ and $\star$, because a **case** is *always* strict.

Note that, as usual, the equational theory collapses under certain environments and types due to the nullary versions of some connectives: we saw above that with a free variable $x : 0 : +$ all terms are equal, and so too are any two terms of type $\top$ via $\eta_{\top}$ (the nullary form of product in $\mathcal{F}$). Even still, there are many important cases where the equational theory is coherent. One particular sanity check is that, in the absence of free variables, the two sum injections $\iota_1()$ and $\iota_2()$ are not equal, inherited from contextual equivalence in Appendix I.

▶ **Theorem 1** (Closed coherence). *For any global environment $\vdash \mathcal{G}$ extending $\mathcal{F}$, the equality $\vdash_{\mathcal{G}} \iota_1() = \iota_2() : 1 \oplus 1 : +$ is not derivable.*

4.5 Adding effects

So far, we have considered only a pure functional calculus. However, one of the features of polarity is its robustness in the face of computational effects, so let's add some. Two particular effects we can add are *general recursion*, in the form of fixed points, and *control* in the form of μ-abstractions from Parigot's $\lambda\mu$-calculus [17]. To do so, we extend the calculus with the following syntax:

$$M, N ::= \ldots \mid \nu\boldsymbol{x}.M \mid \mu\boldsymbol{\alpha}.J \qquad J ::= \langle M \| \alpha \rangle \qquad \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} ::= \alpha{:}A$$

Fixed-point terms $\nu x{:}A.M$ bind x to the result of M inside M itself. Because fixed points must be unrolled before evaluating their underlying term, their type is restricted to $A : -$. Control extends the calculus with *co-variables* $\alpha, \beta, \ldots$ that bind to *evaluation contexts* instead of values, letting programs abstract over and manipulate their control flow. The evaluation context bound to a co-variable α of any type A can be invoked (any number of times) with a term $M : A$ via a jump $\langle M \| \alpha \rangle$ that never returns a result, and the co-variable α of type A can be bound with a μ-abstraction $\mu\alpha{:}A.J$.

To go along with the new syntax, we have some additional type checking rules:

$$\frac{\Gamma, x : A : - \vdash_{\mathcal{G}}^{\Theta} M : A : - \mid \Delta}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \nu x{:}A.M : A : - \mid \Delta} \qquad \frac{J : (\Gamma \vdash_{\mathcal{G}}^{\Theta} \alpha : A : \mathcal{S}, \Delta)}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mu\alpha{:}A.J : A : \mathcal{S} \mid \Delta} \qquad \frac{\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \mid \alpha : A : \mathcal{S}, \Delta}{\langle M \| \alpha \rangle : (\Gamma \vdash_{\mathcal{G}}^{\Theta} \alpha : A : \mathcal{S}, \Delta)}$$

The judgements in other typing rules from Figure 3 are all generalized to $\Gamma \vdash_{\mathcal{G}}^{\Theta} M : A : \mathcal{S} \mid \Delta$. There is also a typing judgement for jumps of the form $J : (\Gamma \vdash_{\mathcal{F}}^{\Theta} \Delta)$, where Θ, Γ, and Δ play the same roles; the only difference is that J is not given a type for its result. Unlike terms, jumps never return. As in the $\lambda\mu$-calculus, the environment Δ is placed on the right because co-variables represent alternative return paths. For example, a term $x : X : -, y : Y : + \vdash_{\mathcal{F}}^{X:-,Y:+} M : Y : - \mid \beta : Y : +$ could return an X via the main path, as in $M = x$, or a Y via β by aborting the main path, as in $M = \mu\alpha{:}X.\langle y \| \beta \rangle$.

And finally, the equational theory is also extended with the following equality axioms:

$$\begin{array}{llll}
(\nu) & \nu \boldsymbol{x}.M \sim M[\nu \boldsymbol{x}.M/\boldsymbol{x}] & & \\
(\beta_\mu^\alpha) & \langle \mu\boldsymbol{\alpha}.J \| \beta \rangle \sim J[\beta/\boldsymbol{\alpha}] & (\beta_\mu^F) & F[\mu\boldsymbol{\alpha}.J] : B \sim \mu\beta{:}B.J[\langle F \| \beta \rangle / \langle \Box \| \boldsymbol{\alpha} \rangle] \\
(\eta_\mu) & \mu\alpha{:}A.\langle M \| \alpha \rangle \sim M & (\kappa_\mu) & T[\mu\boldsymbol{\alpha}.\langle M_i \| \beta \rangle \overset{i}{\ldots}] \sim \mu\boldsymbol{\alpha}.\langle T[M_i \overset{i}{\ldots}] \| \beta \rangle
\end{array}$$

The ν axiom unrolls a fixed point by one step. The two β_μ axioms are standard generalizations of the $\lambda\mu$-calculus: β_μ^α substitutes one co-variable for another, and β_μ^F captures a single frame of a μ-abstraction's evaluation context via a *structural substitution* that replaces one context with another. The κ_μ is the commuting conversion that permutes a μ-abstraction with a tail context T.

5 Encoding user-defined (co-)data types into $\mathcal{F}$

Equipped with both the extensible source language and the fixed $\mathcal{F}$ target language, we are now able to give an encoding of user-defined (co-)data types in terms of just the core $\mathcal{F}$ connectives from Figure 1. Intuitively, each data type is converted to an existential $\oplus$-sum-of-$\otimes$-products and each co-data type is converted to a universal &-product-of-functions, both annotated by the necessary shifts in and out of $+$ and $-$, respectively. The encoding is parameterized by a global environment $\mathcal{G}$ so that we know the overall shape of each declared connective. Given that $\mathcal{G}$ contains the following data declaration of F, the encoding of F is:

$$\begin{array}{l}
\text{Given } \mathbf{data}\, \mathsf{F}(X{:}k).. : \mathcal{S}\, \mathbf{where}\, \mathsf{K}_i : (A_{ij} : \mathcal{T}_{ij} \overset{j}{\ldots} \vdash^{Y_{ij}:l_{ij} \overset{j}{\ldots}} \mathsf{F}(X..)) \overset{i}{\ldots} \in \mathcal{G} \\
[\![\mathsf{F}]\!]_{\mathcal{G}}^{\mathcal{F}} \triangleq \lambda X{:}k... {}_{\mathcal{S}}{\Uparrow}((\exists Y_{ij}{:}l_{ij}.\overset{j}{\ldots}.((\downarrow_{\mathcal{T}_{ij}} A_{ij}) \otimes \overset{j}{\ldots} 1)) \oplus \overset{i}{\ldots} 0)
\end{array}$$

Dually, given that $\mathcal{G}$ contains the following co-data declaration of G, the encoding of G is:

$$\begin{array}{l}
\text{Given } \mathbf{codata}\, \mathsf{G}(X{:}k).. : \mathcal{S}\, \mathbf{where}\, \mathsf{O}_i : (A_{ij} : \mathcal{T}_{ij} \overset{j}{\ldots} \mid \mathsf{G}(X..) \vdash^{Y_{ij}:l_{ij} \overset{j}{\ldots}} B_i : \mathcal{R}_i) \overset{i}{\ldots} \in \mathcal{G} \\
[\![\mathsf{G}]\!]_{\mathcal{G}}^{\mathcal{F}} \triangleq \lambda X{:}k... {}_{\mathcal{S}}{\Downarrow}((\forall Y_{ij}{:}l_{ij}.\overset{j}{\ldots}.((\downarrow_{\mathcal{T}_{ij}} A_{ij}) \to \overset{j}{\ldots} (\uparrow_{\mathcal{R}_i} B_i))) \,\&\, \overset{i}{\ldots} \top)
\end{array}$$

However, the previous encodings for call-by-name, -value, and -need functions and sums from Sections 2 and 3 are not exactly the same when we take the corresponding declarations of functions and sums from Section 4; the call-by-name and -value encodings are missing some of the shifts used by the generic encoding, and they all elide the terminators (0, 1, and $\top$). Does the difference matter? No, because the encoded types are still *isomorphic*.

▶ **Definition 2** (Type Isomorphism)**.** An isomorphism between two open types of kind k, written $\Theta \vDash_{\mathcal{G}} A \approx B : k$, is defined by induction on k:

- $\Theta \vDash_{\mathcal{G}} A \approx B : k \to l$ when $\Theta, X : k \vDash_{\mathcal{G}} A\ X \approx B\ X : l$, and
- $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$ when, for any x and y, there are terms $x : A : \mathcal{S} \vdash^{\Theta}_{\mathcal{G}} N : B : \mathcal{S}$ and $y : B : \mathcal{S} \vdash^{\Theta}_{\mathcal{G}} M : A : \mathcal{S}$ such that $x{:}A{:}\mathcal{S} \vdash^{\Theta}_{\mathcal{G}} (\mathbf{let}\, y{:}B = N\, \mathbf{in}\, M = x) : A : \mathcal{S}$ and $y{:}B{:}\mathcal{S} \vdash^{\Theta}_{\mathcal{G}} (\mathbf{let}\, x{:}A = M\, \mathbf{in}\, N = y) : B : \mathcal{S}$.

Notice that this is an *open* form of isomorphism: in the base case, an isomorphism between types with free variables is witnessed *uniformly* by a single pair of terms. This uniformity in the face of polymorphism is used to make type isomorphism compatible with the $\forall$ and $\exists$ quantifiers. With this notion of type isomorphism, we can formally state how some of the specific shift connectives are redundant. In particular, within the positive ($+$) and negative ($-$) subset, there are only two shifts of interest since the two different shifts between $-$ and $+$ are isomorphic, and the identity shifts on $+$ and $-$ are isomorphic to an identity on types.

▶ **Theorem 3.** *The following isomorphisms hold (under $\vDash_{\mathcal{F}}$) for all $\vdash A : +$ and $\vdash B : -$*

$$\uparrow_+ A \approx {}_-{\Uparrow} A \qquad \downarrow_- B \approx {}_+{\Downarrow} B \qquad \downarrow_+ A \approx A \approx {}_+{\Uparrow} A \qquad \uparrow_- B \approx B \approx {}_-{\Downarrow} B$$

But clearly the shifts involving $\star$ are not isomorphic, since none of them even share the same kind. Recognizing that sometimes the generic encoding uses unnecessary identity shifts, and given the algebraic properties of polarized types [6], the hand-crafted encodings $[\![A]\!]^{+}$, $[\![A]\!]^{-}$, and $[\![A]\!]^{\star}$ are isomorphic to $[\![A]\!]^{\mathcal{F}}$.

5.1 Correctness of encoding

Type isomorphisms give us a helpful assurance that the encoding of user-defined (co-)data types into $\mathcal{F}$ is actually a faithful one. In every extension of $\mathcal{F}$ with user-defined (co-)data types, all types are isomorphic to their encoding.

▶ **Theorem 4.** *For all $\vdash \mathcal{G}$ extending $\mathcal{F}$ and $\Theta \vdash_{\mathcal{G}} A : k$, $\Theta \vDash_{\mathcal{G}} A \approx [\![A]\!]^{\mathcal{F}}_{\mathcal{G}} : k$.*

Note that this isomorphism is witnessed by terms in the totally pure calculus (without fixed points or μ-abstractions); the encoding works *in spite of* recursion and control, not because of it. Because of the type isomorphism, we can extract a two-way embedding between terms of type A and terms of the encoded type $[\![A]\!]^{\mathcal{F}}_{\mathcal{G}}$ from the witnesses of the type isomorphism. By the properties of isomorphisms, this embedding respects equalities between terms; specifically it is a certain kind of adjunction called an *equational correspondence* [22].

▶ **Theorem 5.** *For all isomorphic types $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$, the terms of type A (*i.e., $\Gamma \vdash^{\Theta}_{\mathcal{G}} M : A : \mathcal{S} \mid \Delta$*) are in equational correspondence with terms of type B (*i.e., $\Gamma \vdash^{\Theta}_{\mathcal{G}} N : B : \mathcal{S} \mid \Delta$*).*

This means is that, in the context of a larger program, a single sub-term can be encoded into the core $\mathcal{F}$ connectives without the rest of the program being able to tell the difference. This is useful in optimizing compilers for functional languages which change the interface of particular functions to improve performance, without hampering further optimizations.

The possible application of this encoding in a compiler is as an intermediate language: rather than encoding just one sub-term, exhaustively encoding the whole term translates from a source language with user-defined (co-)data types into the core $\mathcal{F}$ connectives. The essence of this translation is seen in the way patterns and co-patterns are transformed; given the same generic (co-)data declarations listed in Figure 3, the encodings of (co-)patterns are:

$$\begin{aligned}
[\![\mathsf{K}_i\, \boldsymbol{Y}..\ \boldsymbol{x}..]\!]^{\mathcal{F}}_{\mathcal{G}} &\triangleq \mathsf{val}_{\mathcal{S}}\left(\iota_2^i\,(\iota_1\,(\mathsf{pack}\,\boldsymbol{Y}..\,(\mathsf{box}_{\mathcal{T}}\,\boldsymbol{x}, ..()))))\right)\\
[\![\mathsf{O}_i\, \boldsymbol{Y}..\ \boldsymbol{x}..]\!]^{\mathcal{F}}_{\mathcal{G}} &\triangleq \mathsf{enter}_{\mathcal{S}}.\pi_2^i.\pi_1.\mathsf{spec}\,\boldsymbol{Y}...\mathsf{call}\,\boldsymbol{x}...\mathsf{eval}_{\mathcal{R}_i}
\end{aligned}$$

where ι_2^i denotes i applications of the ι_2 constructor, and π_2^i denotes i projections of the π_2 observer. Using this encoding of (co-)patterns, we can encode (co-)pattern-matching as:

$$[\![\mathbf{case}\,M\,\mathbf{of}\{p_i.N_i\overset{i}{..}\}]\!]^{\mathcal{F}}_{\mathcal{G}} \triangleq \mathbf{case}\,[\![M]\!]_{\mathcal{G}}\,\mathbf{of}\{[\![p_i]\!]_{\mathcal{G}}.[\![N_i]\!]_{\mathcal{G}}\overset{i}{..}\} \qquad [\![\lambda\{q_i.M_i\overset{i}{..}\}]\!]^{\mathcal{F}}_{\mathcal{G}} \triangleq \lambda\left\{[\![q_i]\!]_{\mathcal{G}}.[\![M_i]\!]_{\mathcal{G}}\overset{i}{..}\right\}$$

as well as data structures and co-data observations:

$$\begin{aligned}
[\![p[B/\boldsymbol{Y}..,M/\boldsymbol{x}..]]\!]^{\mathcal{F}}_{\mathcal{G}} &\triangleq [\![p]\!]^{\mathcal{F}}_{\mathcal{G}}[[\![B]\!]^{\mathcal{F}}_{\mathcal{G}}/\boldsymbol{Y}.., [\![M]\!]^{\mathcal{F}}_{\mathcal{G}}/\boldsymbol{x}..]\\
[\![M.(q[B/\boldsymbol{Y}..,N/\boldsymbol{x}..])]\!]^{\mathcal{F}}_{\mathcal{G}} &\triangleq [\![M]\!]^{\mathcal{F}}_{\mathcal{G}}.([\![q]\!]^{\mathcal{F}}_{\mathcal{G}}[[\![B]\!]^{\mathcal{F}}_{\mathcal{G}}/\boldsymbol{Y}.., [\![N]\!]^{\mathcal{F}}_{\mathcal{G}}/\boldsymbol{x}..])
\end{aligned}$$

Note that in the above translation, arbitrary *terms* are substituted instead of just *values* as usual. This encoding of terms with user-defined (co-)data types $\mathcal{G}$ into the core $\mathcal{F}$ types is sound with respect to the equational theory (where Γ and Δ are encoded pointwise).

▶ **Theorem 6.** *If the global environment* $\vdash \mathcal{G}$ *extends* $\mathcal{F}$ *and* $\Gamma \vdash^{\ominus}_{\mathcal{G}} M = N : A \mid \Delta$ *then* $[\![\Gamma]\!]^{\mathcal{F}}_{\mathcal{G}} \vdash^{\ominus}_{\mathcal{F}} [\![M]\!]^{\mathcal{F}}_{\mathcal{G}} = [\![N]\!]^{\mathcal{F}}_{\mathcal{G}} : [\![A]\!]^{\mathcal{F}}_{\mathcal{G}} \mid [\![\Delta]\!]^{\mathcal{F}}_{\mathcal{G}}$.

Since the extensible, multi-discipline language is general enough to capture call-by-value, -name, and -need functional languages—or any combination thereof—this encoding establishes a uniform translation from both ML-like and Haskell-like languages into a common core intermediate language: the polarized $\mathcal{F}$.

6 Conclusion

We have showed here how the idea of polarity can be extended with other calling conventions like call-by-need, which opens up its applicability to the implementation of practical functional languages. In particular, we would like to extend GHC's already multi-discipline intermediate language with the core types in $\mathcal{F}$. Since it already has unboxed types [19] corresponding to positive types, what remains are the fully extensional negative types. Crucially, we believe that negative function types would lift the idea of *call arity*—the number of arguments a function takes before "work" is done—from the level of terms to the level of types. Call arity is used to optimize curried function calls, since passing multiple arguments at once is more efficient that computing intermediate closures as each argument is passed one at a time. No work is done in a negative type until receiving an eval request or unpacking a val, so polarized types compositionally specify multi-argument calling conventions.

For example, a binary function on integers would have the type $\mathsf{Int} \to \mathsf{Int} \to {\uparrow}\mathsf{Int}$, which only computes when both arguments are given, versus the type $\mathsf{Int} \to {\uparrow_{\star}}{}_{\star}{\Downarrow}(\mathsf{Int} \to {\uparrow}\mathsf{Int})$ which specifies work is done after the first argument, breaking the call into two steps since a closure must be evaluated and followed. This generalizes the existing treatment of function closures in call-by-push-value to call-by-need closures. The advantage of lifting this information into types is so that call arity can be taken advantage of in higher order functions. For example, the *zipWith* function takes a binary function to combine two lists, pointwise, and has the type $\forall X{:}\star.\forall Y{:}\star.\forall Z{:}\star.(X \to Y \to Z) \to [X] \to [Y] \to [Z]$ The body of *zipWith* does not know the call arity of the function it's given, but in the polarized type built with negative functions: $\forall X{:}\star.\forall Y{:}\star.\forall Z{:}\star.{\Downarrow}({\downarrow}X \to {\downarrow}Y \to {\uparrow}Z) \to {\downarrow}[X] \to {\downarrow}[Y] \to {\uparrow}[Z]$ the interface in the type spells out that the higher-order function uses the faster two-argument calling convention.

References

1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992. URL: `http://dx.doi.org/10.1093/logcom/2.3.297`, `doi:10.1093/logcom/2.3.297`.

2 Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, New York, NY, USA, 1992.

3 Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In *Typed Lambda Calculi and Applications: 10th International Conference*, TLCA'11, pages 27–44, Berlin, Heidelberg, June 2011. Springer Berlin Heidelberg. URL: `http://dx.doi.org/10.1007/978-3-642-21691-6_6`, `doi:10.1007/978-3-642-21691-6_6`.

4 Zena M. Ariola, John Maraist, Martin Odersky, Matthias Felleisen, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 233–246, New York, NY, USA, 1995. ACM. URL: `http://doi.acm.org/10.1145/199448.199507`, `doi:10.1145/199448.199507`.

5 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ICFP '00, pages 233–243, New York, NY, USA, 2000. ACM. URL: `http://doi.acm.org/10.1145/351240.351262`, `doi:10.1145/351240.351262`.

6 Paul Downen. *Sequent Calculus: A Logic and a Language for Computation and Duality*. PhD thesis, University of Oregon, 2017.

7 Paul Downen and Zena M. Ariola. The duality of construction. In Zhong Shao, editor, *Programming Languages and Systems: 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014*, volume 8410 of *Lecture Notes in Computer Science*, pages 249–269. Springer Berlin Heidelberg, Berlin, Heidelberg, April 2014. URL: `http://dx.doi.org/10.1007/978-3-642-54833-8_14`, `doi:10.1007/978-3-642-54833-8_14`.

8 Paul Downen and Zena M. Ariola. A tutorial on computational classical logic and the sequent calculus. *Journal of Functional Programming*, 28:e3, 2018. `doi:10.1017/S0956796818000023`.

9 Tatsuya Hagino. A typed lambda calculus with categorical type constructors. In David H. Pitt, Axel Poigné, and David E. Rydeheard, editors, *Category Theory and Computer Science*, pages 140–157, Berlin, Heidelberg, September 1987. Springer Berlin Heidelberg. URL: `http://dx.doi.org/10.1007/3-540-18508-9_24`, `doi:10.1007/3-540-18508-9_24`.

10 Philip Johnson-Freyd, Paul Downen, and Zena M. Ariola. Call-by-name extensionality and confluence. *Journal of Functional Programming*, 27:e12, 2017. URL: `http://dx.doi.org/10.1017/S095679681700003X`, `doi:10.1017/S095679681700003X`.

11 Paul Blain Levy. *Call-By-Push-Value*. PhD thesis, Queen Mary and Westfield College, University of London, 2001.

12 Paul Blain Levy. *Jumbo λ-Calculus*, pages 444–455. Springer Berlin Heidelberg, Berlin, Heidelberg, July 2006. URL: `http://dx.doi.org/10.1007/11787006_38`, `doi:10.1007/11787006_38`.

13 Luke Maurer, Paul Downen, Zena M. Ariola, and Simon Peyton Jones. Compiling without continuations. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '17, pages 482–494, New York, NY, USA, June 2017. ACM. URL: `http://dx.doi.org/10.1145/3062341.3062380`, `doi:10.1145/3062341.3062380`.

14 Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 14–23, Piscataway, NJ, USA, 1989. IEEE Press. URL: `http://dl.acm.org/citation.cfm?id=77350.77353`.

15 Guillaume Munch-Maccagnoni. *Syntax and Models of a non-Associative Composition of Programs and Proofs*. PhD thesis, Université Paris Diderot, 2013.

16 Guillaume Munch-Maccagnoni and Gabriel Scherer. Polarised intermediate representation of lambda calculus with sums. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS 2015, pages 127–140. IEEE, July 2015. URL: `http://dx.doi.org/10.1109/LICS.2015.22`, `doi:10.1109/LICS.2015.22`.

17 Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning: International Conference*, LPAR '92, pages 190–201, Berlin, Heidelberg, July 1992. Springer Berlin Heidelberg. URL: `http://dx.doi.org/10.1007/BFb0013061`, `doi:10.1007/BFb0013061`.

18 Simon Peyton Jones and Erik Meijer. Henk: a typed intermediate language. In *Proceedings of the First International Workshop on Types in Compilation*, 1997.

19 Simon L. Peyton Jones and John Launchbury. Unboxed values as first class citizens in a non-strict functional language. In John Hughes, editor, *Functional Programming Languages and Computer Architecture: 5th ACM Conference*, pages 636–666, Berlin, Heidelberg, August 1991. Springer Berlin Heidelberg. URL: `http://dx.doi.org/10.1007/3540543961_30`, `doi:10.1007/3540543961_30`.

20 Andrew M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10(3):321–359, June 2000. URL: `http://dx.doi.org/10.1017/S0960129500003066`, `doi:10.1017/S0960129500003066`.

21 Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975. URL: `http://dx.doi.org/10.1016/0304-3975(75)90017-1`, `doi:10.1016/0304-3975(75)90017-1`.

22 Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4):289–360, November 1993. URL: `http://dx.doi.org/10.1007/BF01019462`, `doi:10.1007/BF01019462`.

23 Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(6):916–941, November 1997. URL: `http://doi.acm.org/10.1145/267959.269968`, `doi:10.1145/267959.269968`.

24 Philip Wadler. Call-by-value is dual to call-by-name. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 189–201, New York, NY, USA, 2003. ACM. URL: `http://doi.acm.org/10.1145/944705.944723`, `doi:10.1145/944705.944723`.

25 Noam Zeilberger. On the unity of duality. *Annals of Pure and Applied Logic*, 153(1):660–96, 2008. URL: `http://dx.doi.org/10.1016/j.apal.2008.01.001`, `doi:10.1016/j.apal.2008.01.001`.

26 Noam Zeilberger. *The Logical Basis of Evaluation Order and Pattern-Matching*. PhD thesis, Carnegie Mellon University, 2009.

A Appendix outline

In this appendix, we give supporting material in the form of an alternative language based on the classical sequent calculus, additional details on the operational semantics and CPS-like kernels of the two calculi, and proofs of the stated theorems. More specifically, the appendix contains the following parts:

- Appendix B gives a summary of the related work, and a high-level overview of how the presentation given here corresponds to other multi-discipline calculi (specifically, *call-by-push-value* and *polarized* calculi) in the literature.
- Appendices C and D give a fully dual generalization of Sections 4 and 5 based on the classical sequent calculus. This generalization includes a dual core $\mathcal{D}$ basis of (co-)data types that serves as the target of encoding as well as the dual of call-by-need evaluation. The encoding admits the same correctness criteria as for the encoding into $\mathcal{F}$.
- Appendix E gives an untyped operational semantics for both the functional and sequent calculi. These operational semantics enjoy standard properties like determinism and type safety.
- Appendix F illustrates the core $\mathcal{F}$ and $\mathcal{D}$ intermediate languages in isolation. These two core languages can be viewed independently of the data and co-data declaration mechanism, which are expressive enough to represent all the other declarable types of their respective systems. We also illustrate the *focused* sub-syntax of the $\mathcal{F}$ and $\mathcal{D}$ calculi, which is calculated as the normal forms with respect to the ς reductions of the operational semantics, and show in detail how the focused sub-syntax corresponds to call-by-push-value and polarized calculi in the literature.
- Appendix G shows the equational correspondence between both the extensible functional calculus and the a subset of the sequent calculus. As a consequence, the correctness criteria of encodings in those two calculi are the same.
- Appendix H shows that types are isomorphic to their encodings, and that the encoding of terms is sound with respect to the equational theory of the extensible source language and the target $\mathcal{D}$ or $\mathcal{F}$.
- Appendix I shows the soundness of the equational theory with respect to a contextual equivalence based on the above operational semantics. The untyped reductions of the equational theory are shown to have the standard rewriting theory properties of confluence (*i.e.,* Church-Rosser) and standardization, which gives soundness of the untyped portion of the equational theory. The typed extensionality axioms are then justified by a logical relation based on an orthogonality model of types.

B Related Work

There have been several polarized languages [11, 26, 15], each with subtly different and incompatible restrictions on which programs are allowed to be written. The most common such restriction corresponds to *focusing* in logic [1]; in the terms used here, focusing means that the parameters to constructors and observers *must* be values. Rather than impose a static focusing restriction on the syntax of programs, we instead imply a dynamic focusing behavior—which evaluates the parameters of constructors and observers before (co-)pattern matching—during execution. Both static and dynamic notions of focusing are two sides of the same coin, and amount to the same end [8].

The other restrictions vary between different frameworks, however. First, we might ask where computation can happen. In Levy's call-by-push-value [11], value types (corresponding to positive types) can only be ascribed to values and computation can only occur at computation types (corresponding to negative types), but in Munch-Maccagnoni's system L [15] computation can occur at *any* type. Zeilberger's calculus of unity [25], which is based on the classical sequent calculus, isolates computation in a separate syntactic category of *statements* which do not have a return type, but is essentially the same as call-by-push-value in this regard as both frameworks only deal with *substitutable* entities, to the exclusion of named computations which may not be duplicated or deleted. Second, we might ask what are the allowable types for variables and, when applicable, co-variables. In call-by-push-value, variables always have positive types, but in the calculus of unity variables have negative types or positive *atomic* types (and dually co-variables have positive types or negative atomic types). These restrictions explain why the two frameworks chose their favored shifts: $\Uparrow$ introduces a positive variable and $\downarrow$ introduces a negative one, and in the setting of the sequent calculus $\Downarrow$ introduces a negative co-variable and $\uparrow$ introduces a positive one. They also explain the calculus of unity's pattern matching: if there cannot be positive variables, then pattern matching *must* continue until it reaches something non-decomposable like a λ-abstraction. In contrast, system L has no restrictions on the types of (co-)variables.

In both of these ways, the language presented here is spiritually closest to system L. A motivation for this choice is that call-by-need forces more generality into the system: if there is no computation and no variables of call-by-need types, then the entire point of sharing work is missed. However, the call-by-value and -name sub-language can still be reduced down to the more restrictive style of call-by-push-value and the calculus of unity. We showed here that the two styles of positive and negative shifts are isomorphic, so the brunt of the work is to reduce to the appropriate normal form. In particular, normalization of the dynamic focusing reductions—originally named ς [24]—along with commuting conversions (κ) and let substitution (β_{let}) can be applied to exhaustion to a term of negative type (and a shift can be applied for positive terms) as a transformation into the more restricted form (for more details, see Appendix F). Additionally, negative variables $x : A : -$ can be eliminated by substituting $y.\mathsf{enter}$ for x where $y : \Downarrow A : +$. Alternatively, the (co-)variables by the calculus of unity can be eliminated by type-directed η-expansion into nested (co-)patterns.

The data and co-data mechanism used here extends the "jumbo" connectives of Levy's jumbo λ-calculus [12] to include a treatment of call-by-need as well the move from mono-discipline to multi-discipline. Our notion of (co-)data is also similar to Zeilberger's [26] definition of types via (co-)patterns, which is fully dual, extended with sharing. The corresponding fully dual notion of data and co-data is shown in Appendix C.

C A dual multi-discipline sequent calculus

So far, we have seen how the extensible functional calculus enables multi-discipline programming and can represent many user-defined types with mixed disciplines via encodings. The advantage of this calculus is that it's close to an ordinary core calculus for functional programs, but the disadvantage is its incomplete *symmetries*. Most $\mathcal{F}$ types have a dual counterpart (& and $\oplus$, $\forall$ and $\exists$, *etc.,*) but types like $\otimes$ and $\to$ do not. The disciplines $+$ and $-$ represent opposite calling conventions, but the opposite of call-by-need ($\star$) is missing. To complete the picture, we now consider a fully *dual* calculus, which is based on the symmetric setting of the classical sequent calculus.

Simple (co-)data types

$$\begin{array}{ll}
\textbf{data}\,(X{:}+) \oplus (Y{:}+) : + \,\textbf{where} & \textbf{data}\,0 : + \,\textbf{where} \\
\quad \iota_1 : (X{:}+ \vdash X \oplus Y) & \\
\quad \iota_2 : (Y{:}+ \vdash X \oplus Y) & \\
\textbf{data}\,(X{:}+) \otimes (Y{:}+) : + \,\textbf{where} & \textbf{data}\,1 : + \,\textbf{where} \\
\quad (\_,\_) : (X{:}+, Y{:}+ \vdash X \otimes Y) & \quad () : (\vdash 1) \\
\textbf{codata}\,(X{:}-) \,\&\, (Y{:}-) : - \,\textbf{where} & \textbf{codata}\,\top : - \,\textbf{where} \\
\quad \pi_1 : (\mid X \,\&\, Y \vdash X{:}-) & \\
\quad \pi_2 : (\mid X \,\&\, Y \vdash Y{:}-) & \\
\textbf{codata}\,(X{:}-) \parr (Y{:}-) : - \,\textbf{where} & \textbf{codata}\,\bot : - \,\textbf{where} \\
\quad [\_,\_] : (\mid X \parr Y \vdash X : -, Y : -) & \quad [] : (\mid \bot \vdash\ ) \\
\textbf{data}\,\ominus(X{:}-) : + \,\textbf{where} & \textbf{codata}\,\neg(X{:}+) : - \,\textbf{where} \\
\quad \mathsf{cont} : (\vdash \ominus X \mid X : -) & \quad \mathsf{throw} : (X : + \mid \neg X \vdash\ )
\end{array}$$

Quantifier (co-)data types

$$\begin{array}{ll}
\textbf{data}\,\exists_k(X{:}k{\to}+) : + \,\textbf{where} & \textbf{codata}\,\forall_k(X{:}k{\to}-) : - \,\textbf{where} \\
\quad \mathsf{pack} : (X\ Y{:}+ \vdash^{Y:k} \exists_k X) & \quad \mathsf{spec} : (\mid \forall_k X \vdash^{Y:k} X\ Y{:}-)
\end{array}$$

Polarity shift (co-)data types

$$\begin{array}{ll}
\textbf{data}\,{\downarrow_{\mathcal{S}}}(X{:}\mathcal{S}) : + \,\textbf{where} & \textbf{data}\,{}_{\mathcal{S}}{\Uparrow}(X{:}+) : \mathcal{S}\,\textbf{where} \\
\quad \mathsf{box}_{\mathcal{S}} : (X{:}\mathcal{S} \vdash {\downarrow_{\mathcal{S}}} X) & \quad \mathsf{val}_{\mathcal{S}} : (X{:}+ \vdash {}_{\mathcal{S}}{\Uparrow} X) \\
\textbf{codata}\,{\Uparrow_{\mathcal{S}}}(X{:}\mathcal{S}) : - \,\textbf{where} & \textbf{codata}\,{}_{\mathcal{S}}{\Downarrow}(X{:}-) : \mathcal{S}\,\textbf{where} \\
\quad \mathsf{eval}_{\mathcal{S}} : (\mid {\Uparrow_{\mathcal{S}}} X \vdash X{:}\mathcal{S}) & \quad \mathsf{enter}_{\mathcal{S}} : (\mid {}_{\mathcal{S}}{\Downarrow} X \vdash X{:}-)
\end{array}$$

Figure 5 The $\mathcal{D}$ dual core set of (co-)data declarations.

C.1 The dual core intermediate language: $\mathcal{D}$

In contrast with functional (co-)data declarations, dual calculus allows for symmetric data and co-data type declarations that are properly dual to one another: they can have multiple inputs to the left (of $\vdash$) and multiple outputs to the right (of $\vdash$). This dual notion of (co-)data is strictly more expressive, and lets us declare the new connectives like so:

$$\begin{array}{ll}
\textbf{codata}\,(X{:}-) \parr (Y{:}-) : - \,\textbf{where} & \textbf{codata}\,\bot : - \,\textbf{where} \\
\quad [\_,\_] : (\mid X \parr Y \vdash X : -, Y : -) & \quad [] : (\mid \bot \vdash\ ) \\
\textbf{data}\,\ominus(X{:}-) : + \,\textbf{where} & \textbf{codata}\,\neg(X{:}+) : - \,\textbf{where} \\
\quad \mathsf{cont} : (\vdash \ominus X \mid X : -) & \quad \mathsf{throw} : (X : + \mid \neg X \vdash\ )
\end{array}$$

Note how these types rely on the newfound flexibility of having zero outputs (for $\bot$ and $\neg$) and more than one output (for $\parr$ and $\ominus$). These four types generalize $\mathcal{F}$, and decompose function types into the more primitive negative disjunction and negation types, analogous to the encoding of functions in classical logic: $A \to B \approx (\neg A) \parr B$. The full set of dual core $\mathcal{D}$ connectives is given in Figure 5.

$$
\begin{array}{l}
A, B, C ::= X \mid \mathsf{F} \mid \lambda \boldsymbol{X}.A \mid A\ B \quad \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} ::= X{:}k \quad k, l ::= \mathcal{S} \mid k \to l \quad \mathcal{R}, \mathcal{S}, \mathcal{T} ::= + \mid - \mid \star \mid \bigstar \\
\mathit{decl} ::= \mathbf{data}\,\mathsf{F}\,X{:}k.. : \mathcal{S}\,\mathbf{where}\,\mathsf{K} : (A : \mathcal{T}.. \vdash^{\boldsymbol{Y}..} \mathsf{F}\,X.. \mid B : \mathcal{R}..) \\
\qquad\quad \mid \mathbf{codata}\,\mathsf{G}\,X{:}k.. : \mathcal{S}\,\mathbf{where}\,\mathsf{O} : (A : \mathcal{T}.. \mid \mathsf{G}\,X.. \vdash^{\boldsymbol{Y}..} B : \mathcal{R}..) \\
c ::= \langle v \| e \rangle \\
v ::= x \mid \mu\boldsymbol{\alpha}.c \mid \nu\boldsymbol{x}.v \mid \lambda\{q_i.c_i \mid \overset{i}{..}\} \mid \mathsf{K}\,A..e..v.. \qquad p ::= \mathsf{K}\,\boldsymbol{Y}..\boldsymbol{\alpha}..\boldsymbol{x}.. \qquad \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} ::= x{:}A \\
e ::= \alpha \mid \tilde{\mu}\boldsymbol{x}.c \mid \tilde{\nu}\boldsymbol{\alpha}.e \mid \tilde{\lambda}\{p_i.c_i \mid \overset{i}{..}\} \mid \mathsf{O}\,A..v..e.. \qquad q ::= \mathsf{O}\,\boldsymbol{Y}..\boldsymbol{x}..\boldsymbol{\alpha}.. \qquad \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta} ::= \alpha{:}A
\end{array}
$$

Figure 6 Syntax of the dual calculus.

C.2 Syntax

The syntax of the dual calculus is given in Figure 6. At the level of programs, the syntax is split in two: dual to *terms* (v) which give an answer are *co-terms* (e) which ask a question. Therefore, each of the features from the functional language are divided into one of two camps. Variables x, μ-abstractions $\mu\boldsymbol{\alpha}.c$, fixed points $\nu\boldsymbol{x}.v$, objects of co-data types $\lambda\{\dots\}$, and data structures like $\iota_i v$ are all terms. In contrast, co-variables α, $\tilde{\mu}$-abstractions $\tilde{\mu}\boldsymbol{x}.c$ (analogous to **let** and dual to μ), co-fixed points $\tilde{\nu}\boldsymbol{\alpha}.e$, case analysis of data structures $\tilde{\lambda}\{\dots\}$ (dual to co-data objects) and co-data observations like $\pi_i e$ (dual to data structures) are all co-terms. A command c is analogous to a $\mathcal{F}$ jump, and puts together an answer (a term v) with a question (a co-term e). In this way, the dual calculus can be seen as inverting the elimination functional terms to the other side of a jump $\langle M \| \alpha \rangle$, expanding the role reserved for α. By giving a body to observations themselves, co-patterns q introduce names for *all* sub-components of observations dual to patterns p: for example, the co-pattern of a projection $\pi_i[\alpha{:}A_i] : A_1 \,\&\, A_2$ is perfectly symmetric to the pattern of an injection $\iota_i(x{:}A_i) : A_1 \oplus A_2$.

At the level of types, there is a dual set of connectives and disciplines. The base kind $\star$ signifies the dual to call-by-need ($\bigstar$), which shares delayed control effects the same way call-by-need shares delayed results. For example, the negative co-data type constructors $⅋$ and $\bot$ of $\mathcal{D}$ are dual to the positive connectives $\otimes$ and 1, respectively: they introduce co-pairs $[e, e'] : A ⅋ B$, which is a pair of co-terms $e : A$ and $e' : B$ accepting inputs of type A and B, and the co-unit $[] : \bot$. Objects of co-data types respond to observations by inverting their *entire* structure and then running a command. For $\&$ this looks like $\lambda\{\pi_1[\alpha{:}A].c_1 \mid \pi_2[\beta{:}B].c_2\} : A \,\&\, B$ and for $⅋$ this looks like $\lambda\{[x{:}A, \beta{:}B].c\} : A ⅋ B$. In lieu of the non-symmetric function type, we instead have two dual negation connectives: the data type constructor $\ominus : - \to +$ and the co-data type constructor $\neg : + \to -$ which introduce the (co-)patterns $\mathsf{cont}(\alpha{:}A) : \ominus A$ and $\mathsf{throw}[x{:}A] : \neg A$. These particular forms of negation are chosen because they are *involutive* up to isomorphism (as defined next in Appendix D); their two compositions are identities on types:

$$\lambda X{:}+.\ominus(\neg X) \approx \lambda X{:}+.X \qquad\qquad \lambda X{:}-.\neg(\ominus X) \approx \lambda X{:}-.X$$

Function types can instead be represented as $A \to B \approx (\neg A) ⅋ B$.

C.3 Type system

The type system of $\mathcal{D}$ is given in Figure 7. One major change from the functional calculus' type system is the use of the traditional single-level typing judgement $v : A$ instead of the two-level $M : A : \mathcal{S}$. This is possible because of the sequent calculus' *sub-formula property*—*Cut* is the only inference rule that introduces arbitrary new types in the premises. By just

$$\Gamma ::= x : A.. \qquad \Delta ::= \alpha : A.. \qquad \Theta ::= A : \mathcal{S}.. \qquad \mathcal{G} ::= \mathit{decl}..$$

$$\frac{\Theta, X : k \vdash_{\mathcal{G}} A : l}{\Theta \vdash_{\mathcal{G}} \lambda X{:}k.A : k \to l} \qquad \frac{\Theta \vdash_{\mathcal{G}} A : k \to l \quad \Theta \vdash_{\mathcal{G}} B : k}{\Theta \vdash_{\mathcal{G}} A\ B : l} \qquad \frac{}{\Theta, X : k \vdash_{\mathcal{G}} X : k}$$

$$\frac{(\Theta \vdash_{\mathcal{G}} A : \mathcal{T}).. \quad (\Theta \vdash_{\mathcal{G}} B : \mathcal{R})..}{(x : A.. \vdash_{\mathcal{G}}^{\Theta} \beta : B..)\ \mathbf{ctx}}$$

$$\frac{\Gamma \vdash_{\mathcal{D}}^{\Theta} v : A \mid \Delta \quad \Theta \vdash A : \mathcal{S} \quad \Gamma \mid e : A \vdash_{\mathcal{D}}^{\Theta} \Delta}{\langle v \| e \rangle : (\Gamma \vdash_{\mathcal{D}}^{\Theta} \Delta)}\mathit{Cut}$$

$$\frac{c : (\Gamma \vdash_{\mathcal{D}}^{\Theta} \alpha : A, \Delta)}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \mu\alpha{:}A.c : A \mid \Delta}\mathit{AR} \qquad \frac{c : (\Gamma, x : A \vdash_{\mathcal{D}}^{\Theta} \Delta)}{\Gamma \mid \tilde{\mu}x{:}A.c : A \vdash_{\mathcal{D}}^{\Theta} \Delta}\mathit{AL}$$

$$\frac{}{\Gamma, x : A \vdash_{\mathcal{D}}^{\Theta} x : A \mid \Delta}\mathit{VR} \qquad \frac{}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \alpha : A \mid \alpha : A, \Delta}\mathit{VL}$$

$$\frac{\Gamma, x : A \vdash_{\mathcal{D}}^{\Theta} v : A \mid \Delta \quad \Theta \vdash_{\mathcal{D}} A : -}{\Gamma \vdash_{\mathcal{D}}^{\Theta} \nu x{:}A.v : A \mid \Delta}\mathit{RR} \qquad \frac{\Gamma \mid e : A \vdash_{\mathcal{D}}^{\Theta} \alpha : A, \Delta \quad \Theta \vdash_{\mathcal{D}} A : +}{\Gamma \mid \tilde{\nu}\alpha{:}A.e : A \vdash_{\mathcal{D}}^{\Theta} \Delta}\mathit{RL}$$

$$\frac{\Gamma \mid e : A \vdash_{\mathcal{D}}^{\Theta} \Delta \quad \Theta \vdash_{\mathcal{D}} A {=_{\beta\eta}} B : \mathcal{S}}{\Gamma \mid e : B \vdash_{\mathcal{D}}^{\Theta} \Delta}\mathit{TCR} \qquad \frac{\Gamma \vdash_{\mathcal{D}}^{\Theta} v : A \mid \Delta \quad \Theta \vdash_{\mathcal{D}} A {=_{\beta\eta}} B : \mathcal{S}}{\Gamma \vdash_{\mathcal{D}}^{\Theta} v : B \mid \Delta}\mathit{TCL}$$

Given **data** $\mathsf{F}(X{:}k)..: \mathcal{S}$ **where** $\mathsf{K}_i : (A_{ij}{:}\mathcal{T}_{ij}\overset{j}{..} \vdash^{Y_{ij}:l_{ij}\overset{j}{..}} \mathsf{F}(X..) \mid B_{ij}{:}\mathcal{R}_{ij}\overset{j}{..})\overset{i}{..} \in \mathcal{G}$, we have the rules:

$$\frac{}{\Theta \vdash_{\mathcal{G}} \mathsf{F} : k.. \to \mathcal{S}}$$

$$\frac{(\Theta \vdash_{\mathcal{G}} C_j : l_{ij})\overset{j}{..} \quad (\Gamma \mid e_j : B_{ij}[C'/X.., C_j/Y_{ij}\overset{j}{..}] \vdash_{\mathcal{G}}^{\Theta} \Delta)\overset{j}{..} \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} v_j : A_{ij}[C'/X.., C_j/Y_{ij}\overset{j}{..}] \mid \Delta)\overset{j}{..}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \mathsf{K}_i\, C_j\overset{j}{..}\ e_j\overset{j}{..}\ v_j\overset{j}{..} : \mathsf{F}\, C'.. \mid \Delta}\ \mathsf{F}R_i$$

$$\frac{c_i : (\Gamma, x_{ij} : A_{ij}[C/X..]\overset{j}{..} \vdash_{\mathcal{G}}^{\Theta, Y_{ij}:l_{ij}\overset{j}{..}} \alpha_{ij} : B_{ij}[C/X..]\overset{j}{..}, \Delta)\overset{i}{..}}{\Gamma \mid \tilde{\lambda}\big\{(\mathsf{K}_i\, Y_{ij}{:}l_{ij}\overset{j}{..}\ x_{ij}{:}A_{ij}\overset{j}{..}\ x_{ij}{:}A_{ij}\overset{j}{..}).c_i\overset{i}{..}\big\} : \mathsf{F}\, C.. \vdash_{\mathcal{G}}^{\Theta} \Delta}\ \mathsf{F}L$$

Given **codata** $\mathsf{G}(X{:}k)..: \mathcal{S}$ **where** $\mathsf{O}_i : (A_{ij} : \mathcal{T}_{ij}\overset{j}{..} \mid \mathsf{G}(X..) \vdash^{Y_{ij}:l_{ij}\overset{j}{..}} B_{ij} : \mathcal{R}_{ij}\overset{j}{..})\overset{i}{..} \in \mathcal{G}$, we have the rules:

$$\frac{}{\Theta \vdash_{\mathcal{G}} \mathsf{G} : k.. \to \mathcal{S}}$$

$$\frac{(\Theta \vdash_{\mathcal{G}} C_j : l_{ij})\overset{j}{..} \quad (\Gamma \vdash_{\mathcal{G}}^{\Theta} v_j : A_{ij}[C'/X.., C_j/Y_{ij}\overset{j}{..}] \mid \Delta)\overset{j}{..} \quad (\Gamma \mid e_j : B_{ij}[C'/X.., C_j/Y_{ij}\overset{j}{..}] \vdash_{\mathcal{G}}^{\Theta} \Delta)\overset{j}{..}}{\Gamma \mid \mathsf{O}_i\, C_j\overset{j}{..}\ v_j\overset{j}{..}\ e_j\overset{j}{..} : \mathsf{F}\, C'.. \vdash_{\mathcal{G}}^{\Theta} \Delta}\ \mathsf{G}L_i$$

$$\frac{c_i : (\Gamma, x_{ij} : A_{ij}[C/X..]\overset{j}{..} \vdash_{\mathcal{G}}^{\Theta, Y_{ij}:l_{ij}\overset{j}{..}} \alpha_{ij} : B_{ij}[C/X..]\overset{j}{..}, \Delta)\overset{i}{..}}{\Gamma \vdash_{\mathcal{G}}^{\Theta} \lambda\big\{[\mathsf{O}_i\, Y_{ij}{:}l_{ij}\overset{j}{..}\ x_{ij}{:}A_{ij}\overset{j}{..}\ \alpha_{ij}{:}B_{ij}\overset{j}{..}].c_i\overset{i}{..}\big\} : \mathsf{G}\, C.. \mid \Delta}\ \mathsf{G}R$$

Figure 7 Type system for the dual calculus.

checking that the type of a *Cut* makes sense in the current environment, well-formedness can be separated from typing: if the conclusion of a derivation is well-formed (*i.e.*, $(\Gamma \vdash_{\mathcal{D}}^{\Theta} \Delta)\ \mathbf{ctx}$), then every judgement in the derivation is too. The other major change is that there is a typing judgement for the new syntactic category of co-terms; $\Gamma \mid e : A \vdash_{\mathcal{D}}^{\Theta} \Delta$ means that e works with a term of type A in the environments Θ, Γ, Δ.

C.4 Equational theory

Lastly, we have the equational theory in Figure 8. The dualities of evaluation—between variable and co-variable bindings, data and co-data, values (answers) and evaluation contexts

$$
\begin{array}{ll}
V_+ ::= x \mid \mathsf{K}\,B..E..V.. \mid \lambda\{q.c..\} & E_- ::= \alpha \mid \mathsf{O}\,B..V..E.. \mid \tilde{\lambda}\{p.c..\} \\
V_{\medstar} ::= V_+ \mid \mu\boldsymbol{\alpha}.H[\langle V_{\medstar} \| \alpha\rangle] & E_{\star} ::= E_- \mid \tilde{\mu}\boldsymbol{x}.H[\langle x \| E_{\star}\rangle] \\
V_- ::= v \qquad V_{\star} ::= V_+ & E_+ ::= e \qquad E_{\medstar} ::= E_- \\
H ::= \square \mid \langle v \| \tilde{\mu}x{:}A{:}\star.H\rangle \mid \langle \mu\alpha{:}A{:}\medstar.H \| e\rangle &
\end{array}
$$

$$
\begin{array}{llll}
(\beta_\mu) & \langle \mu\boldsymbol{\alpha}.c \| E\rangle \sim c[E/\boldsymbol{\alpha}] & (\beta_{\tilde{\mu}}) & \langle V \| \tilde{\mu}\boldsymbol{x}.c\rangle \sim c[V/\boldsymbol{x}] \\
(\eta_\mu) & \mu\alpha{:}A.\langle v \| \alpha\rangle \sim v & (\eta_{\tilde{\mu}}) & \tilde{\mu}x{:}A.\langle x \| e\rangle \sim e \\
(\eta_{\mathsf{G}}) & \lambda\{q_i.\langle x \| q_i\rangle \overset{i}{..}\} \sim x & (\eta_{\mathsf{F}}) & \tilde{\lambda}\{p_i.\langle p_i \| \alpha\rangle \overset{i}{..}\} \sim \alpha \\
(\beta_{\mathsf{O}}) & \multicolumn{3}{l}{\langle \lambda\{.. \mid [\mathsf{O}\,\boldsymbol{Y}..\boldsymbol{x}..\boldsymbol{\alpha}..].c \mid ..\} \,\|\, \mathsf{O}\,B..v..e..\rangle \sim \langle v.. \| \tilde{\mu}\boldsymbol{x}...\langle \mu\boldsymbol{\alpha}...c[B/Y..] \| e..\rangle\rangle} \\
(\beta_{\mathsf{K}}) & \multicolumn{3}{l}{\langle \mathsf{O}\,B..e..v.. \| \tilde{\lambda}\{.. \mid (\mathsf{O}\,\boldsymbol{Y}..\boldsymbol{\alpha}..\boldsymbol{x}..).c \mid ..\}\rangle \sim \langle \mu\boldsymbol{\alpha}...\langle v.. \| \tilde{\mu}\boldsymbol{x}...c[B/Y..]\rangle \| e..\rangle} \\
(\chi^{\star}) & \multicolumn{3}{l}{\langle \mu\alpha{:}A{:}\star.\langle v \| \tilde{\mu}y{:}B{:}\star.c\rangle \| e\rangle \sim \langle v \| \tilde{\mu}y{:}B{:}\star.\langle \mu\alpha{:}A{:}\star.c \| e\rangle\rangle} \\
(\chi^{\medstar}) & \multicolumn{3}{l}{\langle v \| \tilde{\mu}y{:}B{:}\medstar.\langle \mu\alpha{:}A{:}\medstar.c \| e\rangle\rangle \sim \langle \mu\alpha{:}A{:}\medstar.\langle v \| \tilde{\mu}y{:}B{:}\medstar.c\rangle \| e\rangle} \\
(\nu) & \nu\boldsymbol{x}.v \sim v[\nu\boldsymbol{x}.v/\boldsymbol{x}] & (\tilde{\nu}) & \tilde{\nu}\boldsymbol{\alpha}.e \sim e[\nu\boldsymbol{\alpha}.e/\boldsymbol{\alpha}]
\end{array}
$$

$$
\frac{c : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \Delta) \quad c \sim c' \quad c' : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \Delta)}{c = c' : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \Delta)}
$$

$$
\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} v : A \mid \Delta \quad v \sim v' \quad \Gamma \vdash^{\Theta}_{\mathcal{D}} v' : A \mid \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} v = v' : A \mid \Delta}
\qquad
\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} e : A \mid \Delta \quad e \sim e' \quad \Gamma \vdash^{\Theta}_{\mathcal{D}} e' : A \mid \Delta}{\Gamma \mid e = e' : A \vdash^{\Theta}_{\mathcal{D}} \Delta}
$$

plus compatibility, reflexivity, symmetry, transitivity

Figure 8 Equational theory for the dual calculus.

(questions)—are more readily apparent than $\mathcal{F}$. In particular, the notion of substitution discipline for $\mathcal{S}$ is now fully dual as in [7]: a subset of terms (*values* $V_{\mathcal{S}}$) and a subset of co-terms (*co-values* $E_{\mathcal{S}}$) which are substitutable. Furthermore, the known dualities between call-by-value ($+$) and -name ($-$) [5], as well as between $\star$ and $\medstar$ [3], are syntactic dualities between values and co-values. The appearance of the χ axioms now reassociate variable and co-variable bindings, and the important cases are for both $\star$ (corresponding to $\chi^{\star}$ of **let**s in the functional calculus) and $\medstar$. Also note the lack of commuting conversions κ; these follow from the μ axioms.

D Encoding fully dual (co-)data types into $\mathcal{D}$

Now let's looks at the fully dual version of the functional encoding from Section 5. Thanks to the generic notion of shifts, the encoding of dual (co-)data into the core $\mathcal{D}$ connectives is similar to the functional encoding, except that in place of the function type $A \to B$ we use the classical representation $(\ominus A) \parr B$. For the generic (co-)data declarations in Figure 7, we have the following definition:

$$
\begin{aligned}
[\![\mathsf{F}]\!]^{\mathcal{D}}_{\mathcal{G}} &\triangleq \lambda\boldsymbol{X}...{}_{\mathcal{S}}\Uparrow((\exists \boldsymbol{Y}_{ij}.\overset{j}{..}((\ominus(\uparrow_{\mathcal{R}_{ij}} B_{ij})) \otimes \overset{j}{..}((\downarrow_{\mathcal{T}_{ij}} A_{ij}) \otimes \overset{j}{..}1))) \oplus \overset{i}{..}0) \\
[\![\mathsf{G}]\!]^{\mathcal{D}}_{\mathcal{G}} &\triangleq \lambda\boldsymbol{X}...{}_{\mathcal{S}}\Downarrow((\forall \boldsymbol{Y}_{ij}.\overset{j}{..}((\neg(\downarrow_{\mathcal{T}_{ij}} A_{ij})) \parr \overset{j}{..}((\uparrow_{\mathcal{R}_{ij}} B_{ij}) \parr \overset{j}{..}\bot))) \,\&\, \overset{i}{..}\top)
\end{aligned}
$$

The encoding of multi-output data types places a $\ominus$-negates every additional output of a constructor, and the encoding of multi-output co-data is now exactly dual to the data encoding. The encodings of (co-)patterns, (co-)pattern-matching objects, and (co-)data structures follow the above type encoding like so:

$$\begin{aligned}
[\![\mathsf{K}_i\, \boldsymbol{Y}..\ \boldsymbol{\alpha}..\ \boldsymbol{x}..]\!]^{\mathcal{D}}_{\mathcal{G}} &\triangleq \mathsf{val}_{\mathcal{S}}\left(\iota^i_2\,(\iota_1\,(\mathsf{pack}\,\boldsymbol{Y}..\,(\mathsf{cont}[\mathsf{eval}_{\mathcal{R}}\,\boldsymbol{\alpha}],..\,(\mathsf{box}_{\mathcal{T}}\,\boldsymbol{x},..()))))\right)\\
[\![\mathsf{O}_i\, \boldsymbol{Y}..\ \boldsymbol{x}..\ \boldsymbol{\alpha}..]\!]^{\mathcal{D}}_{\mathcal{G}} &\triangleq \mathsf{enter}_{\mathcal{S}}\left[\pi^i_2\,[\pi_1\,[\mathsf{spec}\,\boldsymbol{Y}..\,[\mathsf{throw}[\mathsf{box}_{\mathcal{T}}\,\boldsymbol{x}],..\,[\mathsf{eval}_{\mathcal{R}}\,\boldsymbol{\alpha},..[]]]]]\right]\\
[\![\lambda\{q_i.c_i\overset{i}{\dots}\}]\!]^{\mathcal{D}}_{\mathcal{G}} &\triangleq \lambda\{[\![q_i]\!]^{\mathcal{D}}_{\mathcal{G}}.[\![c_i]\!]^{\mathcal{D}}_{\mathcal{G}}\overset{i}{\dots}\}\\
[\![\tilde{\lambda}\{p_i.c_i\overset{i}{\dots}\}]\!]^{\mathcal{D}}_{\mathcal{G}} &\triangleq \tilde{\lambda}\{[\![p_i]\!]^{\mathcal{D}}_{\mathcal{G}}.[\![c_i]\!]^{\mathcal{D}}_{\mathcal{G}}\overset{i}{\dots}\}\\
[\![p[C/Y..,e/\alpha..,v/x..]]\!]^{\mathcal{D}}_{\mathcal{G}} &= [\![p]\!]^{\mathcal{D}}_{\mathcal{G}}[[\![C]\!]^{\mathcal{D}}_{\mathcal{G}}/Y..,[\![e]\!]^{\mathcal{D}}_{\mathcal{G}}/\alpha..,[\![v]\!]^{\mathcal{D}}_{\mathcal{G}}/x..]\\
[\![q[C/Y..,v/x..],e/\alpha..]\!]^{\mathcal{D}}_{\mathcal{G}} &= [\![q]\!]^{\mathcal{D}}_{\mathcal{G}}[[\![C]\!]^{\mathcal{D}}_{\mathcal{G}}/Y..,[\![v]\!]^{\mathcal{D}}_{\mathcal{G}}/x..,[\![e]\!]^{\mathcal{D}}_{\mathcal{G}}/\alpha..]
\end{aligned}$$

We also have an analogous notion of type isomorphism. The case for higher kinds is the same, and base isomorphism $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$ is witnessed by a pair of inverse commands

$$c : (x : A \vdash^{\Theta}_{\mathcal{G}} \beta : B) \qquad\qquad c' : (y : B \vdash^{\Theta}_{\mathcal{G}} \alpha : A)$$

such that both compositions are identities:

$$\begin{aligned}
\langle \mu\beta{:}B.c \| \tilde{\mu}y{:}B.c' \rangle &= \langle x \| \alpha \rangle : (x : A \vdash^{\Theta}_{\mathcal{G}} \alpha : A)\\
\langle \mu\alpha{:}A.c' \| \tilde{\mu}x{:}A.c \rangle &= \langle y \| \beta \rangle : (y : B \vdash^{\Theta}_{\mathcal{G}} \beta : B)
\end{aligned}$$

With this generalized notion of type isomorphism in $\mathcal{D}$, the analogous local and global encodings are sound for fully dual data and co-data types utilizing any combination of $+$, $-$, $\star$, and $\ast$ evaluation.

▶ **Theorem 7.** *For all $\vdash \mathcal{G}$ extending $\mathcal{D}$ and $\Theta \vdash_{\mathcal{G}} A : k$, $\Theta \vDash_{\mathcal{G}} A \approx [\![A]\!]^{\mathcal{D}}_{\mathcal{G}} : k$.*

▶ **Theorem 8.** *For all $\vdash \mathcal{G}$ extending $\mathcal{D}$, (co-)terms of type A are in equational correspondence with (co-)terms of type $[\![A]\!]^{\mathcal{D}}_{\mathcal{G}}$, respectively.*

▶ **Theorem 9.** *If $\vdash \mathcal{G}$ extends $\mathcal{D}$ and $c = c' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ then $[\![c]\!]^{\mathcal{D}}_{\mathcal{G}} = [\![c']\!]^{\mathcal{D}}_{\mathcal{G}} : ([\![\Gamma]\!]^{\mathcal{D}}_{\mathcal{G}} \vdash^{\Theta}_{\mathcal{F}} [\![\Delta]\!]^{\mathcal{D}}_{\mathcal{G}})$.*

E Operational semantics

E.1 The disciplined sub-syntax

We do not need full type checking to run commands of the sequent calculus, but we still need to be able to decide the *disciplines* of terms in order to determine whether or not they are substitutable. Although disciplines are similar to types (*e.g.,* each term and co-term belongs to a discipline, and there is a discipline for each statically bound (co-)variable that matches with its use, *etc.,*) they are not as strict. Zeilberger [25] generalized the notion of "unityped"—that untyped terms can be seen as terms in a language with only one type—to "bityped" in the polarized setting where call-by-name and -value terms must still be distinguished from each other even when types aren't check. Here we further generalize "bityped" to "disciplined" since there can be more than two calling conventions that need distinguishing. For us, a discipline is one of $+$, $-$, $\star$, and $\ast$, so in our case it corresponds to a language with only four different types of (co-)terms.

To define the well-disciplined (or just "disciplined" for short) terms, co-terms, and commands, we can leverage the existing type system by weakening it with an additional

$$c ::= \langle v_{\mathcal{S}} \| e_{\mathcal{S}} \rangle \qquad v ::= v_{\mathcal{S}} \qquad e ::= e_{\mathcal{S}} \qquad m ::= c \mid v \mid e$$

$$v_{\mathcal{S}} ::= x{:}\mathcal{S} \mid \mu\alpha{:}\mathcal{S}.c \mid v^{\mathcal{G}}_{\mathcal{S}} \qquad e_{\mathcal{S}} ::= \alpha{:}\mathcal{S} \mid \tilde{\mu}x{:}\mathcal{S}.c \mid e^{\mathcal{G}}_{\mathcal{S}}$$

$$v^{\mathcal{G}}_{\mathcal{S}} = \{v^{\mathsf{F}} \mid (\mathsf{F} : k..{\to}\mathcal{S}) \in \mathcal{G}\} \qquad e^{\mathcal{G}}_{\mathcal{S}} = \{e^{\mathsf{F}} \mid (\mathsf{F} : k..{\to}\mathcal{S}) \in \mathcal{G}\}$$

If $\mathbf{data}\,\mathsf{F}(X : k).. : \mathcal{S}\,\mathbf{where}\,\mathsf{K}_i : (A_i : \mathcal{T}_i.. \vdash^{Y:l..} \mathsf{F}\,X.. \mid B_i : \mathcal{R}_i..)\overset{i}{\therefore} \in \mathcal{G}$ then:

$$v^{\mathsf{F}} ::= \mathsf{K}_i(e_{\mathcal{R}_i}.., v_{\mathcal{T}_i}..) \mid \overset{i}{\therefore} \qquad e^{\mathsf{F}} ::= \tilde{\lambda}\{\overrightarrow{\mathsf{K}_i(\alpha{:}\mathcal{R}_i.., x{:}\mathcal{T}_i..).c_i}^{i}\}$$

If $\mathbf{codata}\,\mathsf{G}(X : k).. : \mathcal{S}\,\mathbf{where}\,\mathsf{O}_i : (A_i : \mathcal{T}_i.. \mid \mathsf{G}\,X.. \vdash^{Y:l..} B_i : \mathcal{R}_i..)\overset{i}{\therefore} \in \mathcal{G}$ then:

$$v^{\mathsf{G}} ::= \lambda\{\overrightarrow{\mathsf{O}_i[x{:}\mathcal{T}_i.., \alpha{:}\mathcal{R}_i..].c_i}^{i}\} \qquad e^{\mathsf{G}} ::= \mathsf{O}_i[\overrightarrow{v_{\mathcal{T}_i}}, \overrightarrow{e_{\mathcal{R}_i}}] \mid \overset{i}{\therefore}$$

Figure 9 Disciplined sub-syntax of the untyped sequent calculus under a global environment $\mathcal{G}$.

$$V_+ ::= x{:}+ \mid V^{\mathcal{G}}_{+} \qquad E_+ ::= e_+ \qquad V_- ::= v_- \qquad E_- ::= \alpha{:}- \mid E^{\mathcal{G}}_{-}$$

$$V_\star ::= x{:}\star \mid V^{\mathcal{G}}_{\star} \qquad E_\star ::= \alpha{:}\star \mid \tilde{\mu}x{:}\star.H[\langle x{:}\star \| E_\star \rangle] \mid E^{\mathcal{G}}_{\star}$$

$$E_{\medwhitestar} ::= \alpha{:}\medwhitestar \mid E^{\mathcal{G}}_{\medwhitestar} \qquad V_{\medwhitestar} ::= x{:}\medwhitestar \mid \mu\alpha{:}\medwhitestar.H[\langle V_{\medwhitestar} \| \alpha{:}\medwhitestar \rangle] \mid V^{\mathcal{G}}_{\medwhitestar}$$

$$H ::= \square \mid \langle v_\star \| \tilde{\mu}x{:}\star.H \rangle \mid \langle \mu\alpha{:}\medwhitestar.H \| e_{\medwhitestar} \rangle$$

$$V^{\mathcal{G}}_{\mathcal{S}} = \{V^{\mathsf{F}} \mid (\mathsf{F} : k..{\to}\mathcal{S}) \in \mathcal{G}\} \qquad E^{\mathcal{G}}_{\mathcal{S}} = \{E^{\mathsf{F}} \mid (\mathsf{F} : k..{\to}\mathcal{S}) \in \mathcal{G}\}$$

If $\mathbf{data}\,\mathsf{F}(X : k).. : \mathcal{S}\,\mathbf{where}\,\mathsf{K}_i : (A_i : \mathcal{T}_i.. \vdash^{Y:l..} \mathsf{F}\,X.. \mid B_i : \mathcal{R}_i..)\overset{i}{\therefore} \in \mathcal{G}$ then:

$$V^{\mathsf{F}} ::= \mathsf{K}_i(E_{\mathcal{R}_i}.., V_{\mathcal{T}_i}..) \mid \overset{i}{\therefore} \qquad E^{\mathsf{F}} ::= e^{\mathsf{F}}$$

If $\mathbf{codata}\,\mathsf{G}(X : k).. : \mathcal{S}\,\mathbf{where}\,\mathsf{O}_i : (A_i : \mathcal{T}_i.. \mid \mathsf{G}\,X.. \vdash^{Y:l..} B_i : \mathcal{R}_i..)\overset{i}{\therefore} \in \mathcal{G}$ then:

$$V^{\mathsf{G}} ::= v^{\mathsf{G}} \qquad E^{\mathsf{G}} ::= \mathsf{O}_i[\overrightarrow{V_{\mathcal{T}_i}}, \overrightarrow{E_{\mathcal{R}_i}}] \mid \overset{i}{\therefore}$$

Figure 10 (Co-)Values in the disciplined sub-syntax.

axiom $\Theta \vdash_{\mathcal{G}} A = B : \mathcal{S}$ that collapses all types at a base kind $\mathcal{S}$. We refer to this weakened type system as the *discipline system*. The collapse of types means that, for example, positive pairs can be used when positive sums are expected, or that self-application is now possible, allowing for fixed-point combinators that are the signature of untyped terms. Because there are a finite number of disciplines, the well-disciplined sub-syntax of the sequent calculus can also be defined for a given global environment $\mathcal{G}$ by a grammar as given in Figure 9. Since there is effectively only one type of every kind, we use the kind $\mathcal{S}$ itself as the annotation on bound (co-)variables. Note that the use of variables and co-variables are now also annotated, which means that the environments Γ and Δ are also no longer necessary, since the discipline of a free (co-)variable is now self-evident. This grammar is equivalent to discipline-checking: for every derivation $c : (x : A : \mathcal{T}.. \vdash^{\Theta}_{\mathcal{G}} \beta : B : \mathcal{R}..)$ there is a corresponding command in the disciplined sub-syntax with $x{:}\mathcal{T}$ substituted for each x and $\beta{:}\mathcal{R}$ substituted for each β, and similarly for (co-)terms. When we need to explicitly refer to this type erasure conversion, we will denoted it by *Erase*(_).

We can also give the untyped, but disciplined, definitions of values and co-values as a

$$\begin{aligned}
D &::= \square \mid \langle \square \| e_+ \rangle \mid \langle V_+ \| \square \rangle \mid \langle \square \| e_\medstar \rangle \mid \langle V_\medstar \| \square \rangle \mid \langle v_- \| \square \rangle \mid \langle \square \| E_- \rangle \mid \langle v_\star \| \square \rangle \mid \langle \square \| E_\star \rangle \\
&\qquad \mid \mu\alpha{:}\medstar.D \mid \tilde{\mu}x{:}\star.D \\
\rho &::= A{:}k/X{:}k.., \; V_\mathcal{T}/x{:}\mathcal{T}.., \; E_\mathcal{R}/\alpha{:}\mathcal{R}.. \\
W_\mathcal{S} &= \{V_\mathcal{S}\} - \{x{:}\mathcal{S}\} \quad F_\mathcal{S} = \{E_\mathcal{S}\} - \{\alpha{:}\mathcal{S}\} \\
w_\mathcal{S} &= \{v_\mathcal{S}\} - \{V_\mathcal{S}\} \qquad f_\mathcal{S} = \{e_\mathcal{S}\} - \{E_\mathcal{S}\} \\
P &::= \mathsf{K}\, B..E..\ \square\ e..v.. \mid \mathsf{K}\, B..E..V..\ \square\ v.. \\
Q &::= \mathsf{O}\, B..V..\ \square\ v..e.. \mid \mathsf{O}\, B..V..E..\ \square\ e..
\end{aligned}$$

$$\frac{m \mapsto m'}{D[m] \mapsto D[m']} \qquad \frac{m \mapsto m'}{m \mapsto\!\!\!\!\rightarrow m'} \qquad \frac{m \mapsto\!\!\!\!\rightarrow m' \quad m' \mapsto\!\!\!\!\rightarrow m''}{m \mapsto\!\!\!\!\rightarrow m''} \qquad \frac{}{m \mapsto\!\!\!\!\rightarrow m}$$

$$\begin{array}{rrl}
(\beta_\mu^{\pm\star}) & \langle \mu\alpha{:}\mathcal{S}.c \| E_\mathcal{S} \rangle \mapsto c[E_\mathcal{S}/\alpha{:}\mathcal{S}] & (\mathcal{S} \in \{+,-,\star\}) \\
(\beta_{\tilde\mu}^{\pm\medstar}) & \langle V_\mathcal{S} \| \tilde\mu x{:}\mathcal{S}.c \rangle \mapsto c[V_\mathcal{S}/x{:}\mathcal{S}] & (\mathcal{S} \in \{+,-,\medstar\}) \\
(\phi_\mu^\medstar) & \langle \mu\alpha{:}\medstar.H[\langle V_\medstar \| \alpha{:}\medstar \rangle] \| F_\medstar \rangle \mapsto H[\langle V_\medstar \| \alpha{:}\medstar \rangle][F_\medstar/\alpha{:}\medstar] & \\
(\phi_{\tilde\mu}^\star) & \langle W_\star \| \tilde\mu x{:}\star.H[\langle x{:}\star \| E_\star \rangle] \rangle \mapsto H[\langle x{:}\star \| E_\star \rangle][W_\star/x{:}\star] & \\
(\beta_p) & \langle p[\rho] \| \tilde\lambda\{p_i.c_i \overset{i}{..}\} \rangle \mapsto c_i[\rho] \quad (p = p_i) & \\
(\beta_q) & \langle \lambda\{q_i.c_i \overset{i}{..}\} \| q[\rho] \rangle \mapsto c_i[\rho] \quad (q = q_i) & \\
(\varsigma_p) & P[w_\mathcal{T}] : \mathcal{S} \mapsto \mu\alpha{:}\mathcal{S}.\langle w_\mathcal{T} \| \tilde\mu y{:}\mathcal{T}.\langle P[y{:}\mathcal{T}] \| \alpha{:}\mathcal{S} \rangle \rangle & \\
(\varsigma_p) & P[f_\mathcal{R}] : \mathcal{S} \mapsto \mu\alpha{:}\mathcal{S}.\langle \mu\beta{:}\mathcal{R}.\langle P[\beta{:}\mathcal{R}] \| \alpha{:}\mathcal{S} \rangle \| f_\mathcal{R} \rangle & \\
(\varsigma_q) & Q[f_\mathcal{R}] : \mathcal{S} \mapsto \tilde\mu x{:}\mathcal{S}.\langle \mu\beta{:}\mathcal{R}.\langle x{:}\mathcal{S} \| Q[\beta{:}\mathcal{R}] \rangle \| f_\mathcal{R} \rangle & \\
(\varsigma_q) & Q[w_\mathcal{T}] : \mathcal{S} \mapsto \tilde\mu x{:}\mathcal{S}.\langle w_\mathcal{T} \| \tilde\mu y{:}\mathcal{T}.\langle x{:}\mathcal{S} \| Q[y{:}\mathcal{T}] \rangle \rangle & \\
(\nu) & \nu x{:}-.v_- \mapsto v_-[\nu x{:}-.v_-/x{:}-] & \\
(\tilde\nu) & \tilde\nu\alpha{:}+.e_+ \mapsto e_+[\tilde\nu\alpha{:}+.e_+/\alpha{:}+] &
\end{array}$$

Figure 11 An untyped operational semantics for the extensible dual calculus under a global environment $\mathcal{G}$.

grammar as shown in Figure 10. Note that there is a (co-)variable capture caveat in the definition of call-by-need co-values $E_\star$ and call-by-co-need values $V_\medstar$, where $\tilde\mu x{:}\star.H[\langle x{:}\star \| E_\star \rangle]$ is a co-value only if H does not bind the variable $x{:}\star$ and $\mu\alpha{:}\medstar.H[\langle V_\medstar \| \alpha{:}\medstar \rangle]$ is a value only if H does not bind the co-variable $\alpha{:}\medstar$.

E.2 Untyped rewriting and operational semantics

The untyped, multi-disciplined semantics are given in three forms: an operational syntax (in Figure 11), rewriting theory (in Figure 12), and equational theory (in Figure 13). Each form of semantics builds on the previous one by relating more program fragments with additional rules and closure properties (*i.e.*, full compatibility and symmetry). Note that the reflexive-transitive closure of a single operational step ($\mapsto$), a single reduction ($\to$), and a single conversion ($\leftrightarrow$) is written as $\mapsto\!\!\!\!\rightarrow$, $\twoheadrightarrow$, and $=$, respectively.

$$\frac{m \mapsto m'}{m \to m'} \qquad \frac{m \to m'}{C[m] \to C[m']} \qquad \frac{m \to m'}{m \twoheadrightarrow m'} \qquad \frac{}{m \twoheadrightarrow m} \qquad \frac{m \twoheadrightarrow m' \quad m' \twoheadrightarrow m''}{m \twoheadrightarrow m''}$$

$$\begin{array}{lll}
(\beta_\mu^{\ast}) & \langle \mu\alpha{:}\ast.c \| E_\ast\rangle \to c[E_\ast/\alpha{:}\ast] & \\
(\beta_{\tilde\mu}^{\star}) & \langle V_\star \| \tilde\mu x{:}\star.c\rangle \to c[V_\star/x{:}\star] & \\
(\eta_\mu^{\mathcal{S}}) & \mu\alpha{:}\mathcal{S}.\langle v_{\mathcal{S}} \| \alpha{:}\mathcal{S}\rangle \to v_{\mathcal{S}} & (\alpha{:}\mathcal{S} \notin FV(v_{\mathcal{S}})) \\
(\eta_{\tilde\mu}^{\mathcal{S}}) & \tilde\mu x{:}\mathcal{S}.\langle x{:}\mathcal{S} \| e_{\mathcal{S}}\rangle \to e_{\mathcal{S}} & (x{:}\mathcal{S} \notin FV(e_{\mathcal{S}})) \\
(\delta^{\ast}) & \langle \mu\alpha{:}\ast.c \| e_\ast\rangle \to c & (\alpha{:}\ast \notin FV(c)) \\
(\delta^{\star}) & \langle v_\star \| \tilde\mu x{:}\star.c\rangle \to c & (x{:}\star \notin FV(c))
\end{array}$$

Figure 12 An untyped rewriting theory for multi-discipline sequent calculus under a global environment $\mathcal{G}$.

$$\frac{m \to m'}{m \leftrightarrow m'} \qquad \frac{m \leftrightarrow m'}{C[m] \leftrightarrow C[m']} \qquad \frac{m \leftrightarrow m'}{m' \leftrightarrow m} \qquad \frac{m \leftrightarrow m'}{m = m'} \qquad \frac{}{m = m} \qquad \frac{m = m' \quad m = m''}{m = m''}$$

$$\begin{array}{ll}
(\chi^{\star}) & \langle \mu\alpha{:}A{:}\star.\langle v \| \tilde\mu y{:}B{:}\star.c\rangle \| e\rangle \leftrightarrow \langle v \| \tilde\mu y{:}B{:}\star.\langle \mu\alpha{:}A{:}\star.c \| e\rangle\rangle \\
(\chi^{\ast}) & \langle v \| \tilde\mu y{:}B{:}\ast.\langle \mu\alpha{:}A{:}\ast.c \| e\rangle\rangle \leftrightarrow \langle \mu\alpha{:}A{:}\ast.\langle v \| \tilde\mu y{:}B{:}\ast.c\rangle \| e\rangle
\end{array}$$

Figure 13 An untyped equational theory for multi-discipline sequent calculus for a global environment $\mathcal{G}$.

E.3 Properties of the operational semantics

The design of the above operational semantics is done with an eye certain properties, like *determinism* (each command has at most one step), as well as compatibility with evaluation contexts D. This leads to a notion of type safety, in the usual form of progress and preservation.

▶ **Property E.1.** For all $\mathcal{G}$, $\mathcal{S}$, values $V_{\mathcal{S}}$, and co-values $E_{\mathcal{S}}$,

a) for all substitutions ρ, $V_{\mathcal{S}}[\rho]$ is a value and $E_{\mathcal{S}}[\rho]$ is a co-value,
b) if $V_{\mathcal{S}} \to v_{\mathcal{S}}$ then $v_{\mathcal{S}}$ is a value and if $E_{\mathcal{S}} \to e_{\mathcal{S}}$ then $e_{\mathcal{S}}$ is a co-value, and
c) $V_{\mathcal{S}} \not\mapsto$ and $E_{\mathcal{S}} \not\mapsto$.

Proof. By induction on the definition of values and co-values for every $\mathcal{S}$. ◀

▶ **Property E.2.** For all $\mathcal{G}$ and $\mathcal{S}$,

a) for all substitutions ρ, if $c \mapsto c'$ then $c[\rho] \mapsto c'[\rho]$,
b) for all D, $c \mapsto c'$ if and only if $D[c] \mapsto D[c']$, and
c) if $c \mapsto c'$ and $c \mapsto c''$ then $c' =_\alpha c''$.

and similarly for terms and co-terms.

Proof. By Property E.1 and induction on the definition of $\mapsto$. ◀

▶ **Definition 10.** For any $\mathcal{G}$, a command c is *inside* of c' when there is a context H such that $H[c] = c'$, is the *center* of c' when c is inside any other command inside c, and is *atomic* when the only command inside c is c itself.

▶ **Lemma 11** (Unique decomposition). *Every command c has a* unique *center, denoted by* $\text{center}(c)$*, that is atomic. Furthermore,* $\text{center}(H[c]) = \text{center}(c)$ *and* $\text{center}(\text{center}(c)) = \text{center}(c)$.

Proof. The center of c can be found by decomposing c as $H_1[c_1]$, where H_1 is the longest prefix of $\star$ $\tilde{\mu}$-bindings and $\star$ μ-bindings starting from the top of c.

To show this center is unique, suppose there is another center c_2 such that $H_2[c_2] = c$. Because both c_1 and c_2 must be inside each other, there must be delayed contexts H_1' and H_2' such that $H_1'[c_1] = c_2$ and $H_2'[c_2] = c_1$. The only possibilities are that H_1' and H_2' are both the empty context since $H_1'[H_2'[c_2]] = c_2$ and $H_2'[H_1'[c_1]] = c_1$, meaning that $H_1'[c_1] = c_1 = c_2 = H_2'[c_2]$.

Now for any c, $\text{center}(H[c]) = \text{center}(c)$ by uniqueness and composition of contexts, and $\text{center}(\text{center}(c)) = \text{center}(c)$ follows from the fact that $H[\text{center}(c)] = c$ for some H and the previous fact. Atomicity of $\text{center}(c)$ means the same thing as $\text{center}(\text{center}(c)) = \text{center}(c)$. ◀

▶ **Definition 12** (Need). The set of *needed* (co-)variables of an expression is defined as:

$$x{:}\mathcal{S} \in \text{NV}(m) \iff \exists D. x{:}\mathcal{S} \notin BV(D) \wedge D[x{:}\mathcal{S}] =_\alpha m \not\mapsto$$
$$\alpha{:}\mathcal{S} \in \text{NV}(m) \iff \exists D. \alpha{:}\mathcal{S} \notin BV(D) \wedge D[\alpha{:}\mathcal{S}] =_\alpha m \not\mapsto$$

▶ **Property E.3.** a) $\text{NV}(c) \subseteq FV(c)$, $\text{NV}(v_\mathcal{S}) \subseteq FV(v_\mathcal{S})$, and $\text{NV}(e_\mathcal{S}) \subseteq FV(e_\mathcal{S})$.

b) $\text{NV}(v_\mathcal{S})$ and $\text{NV}(e_\mathcal{S})$ contain at most one variable and co-variable (of discipline $\mathcal{S}$), respectively, and $\text{NV}(c)$ contains at most one variable and at most one co-variable (of the same discipline when there is one of each).

c) $\mu\alpha{:}\star.c$ is a value if and only if $\alpha{:}\star \in \text{NV}(c)$ and $\tilde{\mu}x{:}\star.c$ is a co-value if and only if $x{:}\star \in \text{NV}(c)$.

d) If $c =_\alpha c'$, $v_\mathcal{S} =_\alpha v'_\mathcal{S}$, and $e_\mathcal{S} =_\alpha e'_\mathcal{S}$ then $\text{NV}(c) = \text{NV}(c')$, $\text{NV}(v_\mathcal{S}) = \text{NV}(v'_\mathcal{S})$, and $\text{NV}(e_\mathcal{S}) = \text{NV}(e'_\mathcal{S})$, respectively.

e) If $\text{NV}(c)$, $\text{NV}(v_\mathcal{S})$, and $\text{NV}(e_\mathcal{S})$ are non-empty then $c \not\mapsto$, $v_\mathcal{S} \not\mapsto$, and $e_\mathcal{S} \not\mapsto$, respectively.

▶ **Definition 13.** A command c is *in progress* when $c \mapsto c'$ for some c', *finished* when $\text{NV}(c)$ is non-empty, and *stuck* when c is neither finished nor in progress.

Note that if c is finished then $c \not\mapsto$, so that every command is exactly one of (1) finished, (2) stuck, or (3) in progress. We write $c \Downarrow c'$ to mean that $c \mapsto\!\!\!\!\rightarrow c'$ and c' is finished, and $c \Uparrow$ to mean that there is an infinite standard reduction sequence $c \mapsto c_1 \mapsto c_2 \mapsto \ldots$.

▶ **Lemma 14.** *For all $\mathcal{G}$, Θ, and substitutions ρ such that $BV(\Theta) \cap BV(\rho) = \emptyset$,*

a) if $c : (\Gamma \vdash^\Theta_\mathcal{G} \Delta)$ then $c[\rho] : (\Gamma[\rho] \vdash^\Theta_\mathcal{G} \Delta[\rho])$,
b) if $\Gamma \vdash^\Theta_\mathcal{G} v : A \mid \Delta$ then $\Gamma[\rho] \vdash^\Theta_\mathcal{G} v[\rho] : A[\rho] \mid \Delta[\rho]$, and
c) if $\Gamma \mid e : A \vdash^\Theta_\mathcal{G} \Delta$ then $\Gamma[\rho] \mid e[\rho] : A[\rho] \vdash^\Theta_\mathcal{G} \Delta[\rho]$.

Proof. By induction on the given typing derivation. ◀

▶ **Lemma 15** (Subject reduction).

a) If $c : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ *and* $\mathit{Erase}(c : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)) \to c'$ *then there is some* $c'' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ *such that* $\mathit{Erase}(c'' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)) = c'$*,*
b) if $\Gamma \vdash^{\Theta}_{\mathcal{G}} v : A \mid \Delta$ *and* $\mathit{Erase}(\Gamma \vdash^{\Theta}_{\mathcal{G}} v : A \mid \Delta) \to v'$ *then there is some* $\Gamma \vdash^{\Theta}_{\mathcal{G}} v'' : A \mid \Delta$ *such that* $\mathit{Erase}(\Gamma \vdash^{\Theta}_{\mathcal{G}} v'' : A \mid \Delta) = v'$*, and*
c) if $\Gamma \mid e : A \vdash^{\Theta}_{\mathcal{G}} \Delta$ *and* $\mathit{Erase}(\Gamma \mid e : A \vdash^{\Theta}_{\mathcal{G}} \Delta) \to e'$ *then there is some* $\Gamma \mid e' : A \vdash^{\Theta}_{\mathcal{G}} \Delta$ *such that* $\mathit{Erase}(\Gamma \mid e' : A \vdash^{\Theta}_{\mathcal{G}} \Delta) = e'$*.*

Proof. By Theorem 14 and induction on the given typing derivation, and then cases on the reduction when a rule is applied. ◀

▶ **Property E.4.** For all $\mathcal{G}$,

a) if $c \mapsto c'$ then $c \to c'$, and
b) if $c : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ and $\mathit{Erase}(c : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)) \to c'$ then there is some $\mathit{Erase}(c'' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)) = c'$ such that $c = c'' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$,

and similarly for terms and co-terms.

Proof. Part (a) is immediate by definition. Part (b) follows from Theorem 15 and the fact that the untyped, and discipline-specific $\beta\varsigma$ reduction rules are derivable from the typed $\beta\eta$ equality axioms with the help of $\beta_\mu\beta_{\tilde{\mu}}\eta_\mu\eta_{\tilde{\mu}}$ [7]. ◀

▶ **Theorem 16** (Type safety). *a)* Progress*: Every typed command is either finished or in progress, after type erasure.*
b) Preservation*: If* $c \mapsto c'$ *and* c *is the erasure of some typed command then so is* c'*.*
It follows that $c \Downarrow c'$ *if and only if* $c \mapsto\!\!\!\!\!\to c' \not\mapsto$ *for every* c *that is the erasure of some typed command.*

Proof. Part (a) follows by induction on the typing derivation of the command. Part (b) is a corollary of Theorem 15 and Property E.4. The final statement follows from parts (a) and (b) and determinism of standard reduction by induction on the standard reduction sequence $c \mapsto\!\!\!\!\!\to c'$. ◀

E.4 The functional λ-calculus

The untyped operational semantics the multi-discipline λ-calculus is given in Figure 14. It operates over the multi-discipline sub-syntax of the λ-calculus; as before, this is defined by the extension of the type system with the axiom $\Theta \vdash_{\mathcal{G}} A = B : \mathcal{S}$ which collapses all types at a base kind $\mathcal{S}$. We again use the convention that $M_{\mathcal{S}}$ denotes a term $M : A : \mathcal{S}$.

The generic $\beta^{\pm}_{let}$ rule, where $\pm$ denotes one of $+$ or $-$, implements call-by-value and -name substitution, whereas the more restricted $\phi^{\star}_{let}$ rule implements call-by-need. The β_p and β_q rules match against a pattern p or co-pattern q, respectively. Special cases of this rule are the β_p rule for a sum pattern and β_q for a function co-pattern as follows:

$$\mathbf{case}\, \iota_i V_+ \,\mathbf{of}\, \{\iota_1(x_1{:}+).N_1 \mid \iota_2(x_2{:}+).N_2\} \mapsto_{\beta_p} N_i[V_+/x_i{:}+]$$
$$\lambda\{\mathsf{call}(x{:}+).M\}.\mathsf{call}\, V_+ \mapsto_{\beta_q} M[V_+/x{:}+]$$

The ς rules work toward enabling the β pattern-matching rules by giving names to non-value components (denoted by $R_{\mathcal{S}}$ for the discipline $\mathcal{S}$) of data structures (*i.e.,* pairs and tagged terms) and arguments of observations (*i.e.,* function calls). Naming has two advantages compared with just evaluating non-values in-place: it makes evaluation contexts more regular by converting them into generic **let**-expressions, and it correctly implements sharing with $\star$

$$
\begin{array}{llll}
V ::= V_{\mathcal{S}} : A : \mathcal{S} & V_{+} ::= x \mid \mathsf{K}\, B..V.. \mid \lambda\{q_i.M_i \mid \overset{i}{..}\} & V_{-} ::= M & V_{\star} ::= V_{+} \\
W_{\mathcal{S}} = \{V_{\mathcal{S}}\} - \{x{:}\mathcal{S}\} & R_{\mathcal{S}} = \{M : A : \mathcal{S}\} - \{V_{\mathcal{S}}\} & &
\end{array}
$$

$$
\begin{array}{l}
F ::= \Box.\mathsf{O}\, B..V.. \mid \mathbf{case}\,\Box\,\mathbf{of}\,\{p_i.M_i \overset{i}{..}\} \mid \mathbf{let}\, x{:}{+} = \Box\,\mathbf{in}\, M \mid \mathbf{let}\, x{:}{\star} = \Box\,\mathbf{in}\, H[E[x]] \\
E ::= \Box \mid F[E] \qquad U ::= \mathbf{let}\, x{:}{\star} = M\,\mathbf{in}\,\Box \qquad H ::= \Box \mid U[H] \\
P ::= \mathsf{K}\, B..V..\ \Box\ M.. \qquad Q ::= \mathsf{O}\, B..V..\ \Box\ M.. \\
\rho ::= A{:}k/X{:}k..\ V{:}B/x{:}B..
\end{array}
$$

$$
\frac{M \mapsto N}{E[M] \mapsto E[N]} \qquad \frac{M \mapsto N}{H[M] \mapsto H[N]} \qquad \frac{M \mapsto N}{\langle M \| \alpha \rangle \mapsto \langle N \| \alpha \rangle}
$$

Operational rules for totally pure functional programs:

$$
\begin{array}{lrl}
(\beta^{\pm}_{let}) & \mathbf{let}\, x{:}\mathcal{S} = V_{\mathcal{S}}\,\mathbf{in}\, M \mapsto & M[V_{\mathcal{S}}/x{:}\mathcal{S}] \quad (\mathcal{S} \in \{+,-\}) \\
(\phi^{\star}_{let}) & \mathbf{let}\, x{:}{\star} = W_{\star}\,\mathbf{in}\, H[E[x{:}{\star}]] \mapsto & (H[E[x{:}{\star}]])[W_{\star}/x{:}{\star}] \\
(\beta_p) & \mathbf{case}\, p[\rho]\,\mathbf{of}\,\{p_i.M_i \mid \overset{i}{..}\} \mapsto & M_i[\rho] \quad (p = p_i) \\
(\beta_q) & \lambda\{q_i.M_i \mid \overset{i}{..}\}.(q[\rho]) \mapsto & M_i[\rho] \quad (q = q_i) \\
(\varsigma_p) & P[R_{\mathcal{T}}] \mapsto & \mathbf{let}\, y{:}\mathcal{T} = R_{\mathcal{T}}\,\mathbf{in}\, P[y{:}\mathcal{T}] \\
(\varsigma_q) & M_{\mathcal{S}}.(Q[R_{\mathcal{T}}]) \mapsto & \mathbf{let} x{:}\mathcal{S} = M_{\mathcal{S}}\,\mathbf{in} \\
 & & \mathbf{let} y{:}\mathcal{T} = R_{\mathcal{T}}\,\mathbf{in}\, x{:}\mathcal{S}.Q[y{:}\mathcal{T}] \\
(\kappa^{\star}_F) & F[T[H[V]]] \mapsto & T[F[H[V]]]
\end{array}
$$

Operational rules for recursion and control:

$$
\begin{array}{lrl}
(\nu) & \nu x{:}{-}.M \mapsto & M[\nu x{:}{-}.M/x{:}{-}] \\
(\beta^{\alpha}_{\mu}) & \langle \mu\alpha{:}\mathcal{S}.J \| \beta{:}\mathcal{S} \rangle \mapsto & J[\beta{:}\mathcal{S}/\alpha{:}\mathcal{S}] \\
(\beta^{F}_{\mu}) & F[\mu\alpha{:}\mathcal{S}.J] : \mathcal{R} \mapsto & \mu\beta{:}\mathcal{R}.J[\langle F \| \beta{:}\mathcal{R} \rangle / \langle \Box \| \alpha{:}\mathcal{S} \rangle] \\
(\kappa^{\star}_{\mu}) & T[\mu\alpha{:}\mathcal{S}.\langle M \| \beta{:}\mathcal{T} \rangle] \mapsto & \mu\alpha{:}\mathcal{S}.\langle T[M] \| \beta{:}\mathcal{T} \rangle
\end{array}
$$

Figure 14 An untyped operational semantics for multi-discipline λ-calculus under a global environment $\mathcal{G}$.

components. Specifically, $\mathsf{box}_{\star}\, R_{\star}$ is not a value because $R_{\star}$ is not a value, but $R_{\star}$ should not be evaluated yet until it is needed. Therefore, the correct move is to give some name x to $R_{\star}$ and proceed with the (open) value $\mathsf{box}_{\star}(x{:}{\star})$.

Call-by-need computation uses the $\phi^{\star}_{let}$ and $\kappa^{\star}_{F}$ rules. For the sake of determinism, $\phi^{\star}_{let}$ is a call-by-need version of the $\beta^{\pm}_{let}$ rule that is restricted in two ways: the substitution of x only occurs when the x is *needed* (as in the context $H[E[x]]$), and when the right-hand side is not another variable. The first restriction prevents non-determinism in terms like $\mathbf{let}\, x{:}{\star} = V_{\star}\,\mathbf{in}\,\mathbf{let}\, y{:}{\star} = V'_{\star}\,\mathbf{in}\, N$ (only one of x or y could be needed) and the second in terms like $\mathbf{let}\, x{:}{\star} = M_{\star}\,\mathbf{in}\,\mathbf{let}\, y{:}{\star} = x{:}{\star}\,\mathbf{in}\, y{:}{\star}$ (only $M_{\star}$ can step). The $\kappa^{\star}_{F}$ rule keeps computation moving forward when a value (V) is returned in the context of a heap (H) by commuting a frame (F) of evaluation with a thunk (T) allocation. For example, we have the following

$$
\begin{aligned}
&A,B,C ::= X \mid \mathsf{F} \mid \lambda \boldsymbol{X}.A \mid A\ B \quad \boldsymbol{X},\boldsymbol{Y},\boldsymbol{X} ::= X{:}k \quad k,l ::= \mathcal{S} \mid k \to l \quad \mathcal{R},\mathcal{S},\mathcal{T} ::= + \mid - \mid \star \\
&\quad \mathsf{F},\mathsf{G} ::= \& \mid \to \mid \top \mid \oplus \mid \otimes \mid 0 \mid 1 \mid \forall_k \mid \exists_k \mid {\downarrow_{\mathcal{S}}} \mid {\uparrow_{\mathcal{S}}} \mid {}_{\mathcal{S}}{\Uparrow} \mid {}_{\mathcal{S}}{\Downarrow} \\
&\quad p ::= \iota_1 \boldsymbol{x} \mid \iota_2 \boldsymbol{x} \mid (\boldsymbol{x},\boldsymbol{y}) \mid () \mid \mathsf{pack}\ \boldsymbol{X}\ \boldsymbol{y} \mid \mathsf{box}_{\mathcal{S}}\ \boldsymbol{x} \mid \mathsf{val}_{\mathcal{S}}\ \boldsymbol{x} \qquad \boldsymbol{x},\boldsymbol{y},\boldsymbol{z} ::= x{:}A \\
&\quad q ::= \pi_1 \mid \pi_2 \mid \mathsf{call}\ \boldsymbol{x} \mid \mathsf{spec}\ \boldsymbol{X} \mid \mathsf{eval}_{\mathcal{S}} \mid \mathsf{enter}_{\mathcal{S}} \\
&M,N ::= x \mid \mathbf{let}\ \boldsymbol{x} = M\ \mathbf{in}\ N \mid \lambda\{q_i.M_i \mid \overset{i}{..}\} \mid \mathbf{case}\ M\ \mathbf{of}\{p_i.M_i \mid \overset{i}{..}\} \\
&\qquad \mid M.\pi_1 \mid M.\pi_2 \mid M.\mathsf{call}\ N \mid M.\mathsf{spec}\ A \mid \iota_1 M \mid \iota_2 M \mid (M,N) \mid () \mid \mathsf{pack}\ A\ M \\
&\qquad \mid \mathsf{box}_{\mathcal{S}}\ M \mid \mathsf{val}_{\mathcal{S}}\ M \mid M.\mathsf{eval}_{\mathcal{S}} \mid M.\mathsf{enter}_{\mathcal{S}}
\end{aligned}
$$

Figure 15 Syntax of $\mathcal{F}$: a core, multi-discipline λ-calculus.

$$
\begin{aligned}
M_{\mathcal{S}}, N_{\mathcal{S}} &::= V_{\mathcal{S}} \mid \mathbf{let}\ \boldsymbol{x} = M_{\mathcal{T}}\ \mathbf{in}\ N_{\mathcal{S}} \mid M_{-}.\mathsf{eval}_{\mathcal{S}} \\
&\quad \mid \mathbf{case}\ M_{+}\ \mathbf{of}\{\iota_1 \boldsymbol{x}.N_{\mathcal{S}} \mid \iota_2 \boldsymbol{x}.N'_{\mathcal{S}}\} \mid \mathbf{case}\ M_{+}\ \mathbf{of}\{\} \\
&\quad \mid \mathbf{case}\ M_{+}\ \mathbf{of}\{(\boldsymbol{x},\boldsymbol{y}).N_{\mathcal{S}}\} \mid \mathbf{case}\ M_{+}\ \mathbf{of}\{().N_{\mathcal{S}}\} \\
&\quad \mid \mathbf{case}\ M_{+}\ \mathbf{of}\{(\mathsf{pack}\ \boldsymbol{Y}\,\boldsymbol{x}).N_{\mathcal{S}}\} \\
&\quad \mid \mathbf{case}\ M_{+}\ \mathbf{of}\{\mathsf{box}_{\mathcal{T}}\ \boldsymbol{x}.N_{\mathcal{S}}\} \mid \mathbf{case}\ M_{\mathcal{T}}\ \mathbf{of}\{\mathsf{val}_{\mathcal{T}}\ \boldsymbol{x}.N_{\mathcal{S}}\} \\
V_{-} &::= x \mid M_{-} \mid M_{-}.\pi_1 \mid M_{-}.\mathsf{call}\ V_{+} \mid M_{-}.\pi_2 \mid M_{-}.\mathsf{spec}\ A \\
&\quad \mid \lambda\{\pi_1.M_{-} \mid \pi_2.N_{-}\} \mid \lambda\{\} \mid \lambda(\mathsf{call}\ \boldsymbol{x}).M_{-} \mid \lambda(\mathsf{spec}\ \boldsymbol{Y}).M_{-} \\
&\quad \mid \mathsf{val}_{-}\ V_{+} \mid \lambda\mathsf{enter}_{-}.M_{-} \mid M_{\mathcal{S}}.\mathsf{enter}_{\mathcal{S}} \mid \lambda\mathsf{eval}_{\mathcal{S}}.M_{\mathcal{S}} \\
V_{+} &::= x \mid \iota_1 V_{+} \mid \iota_2 V_{+} \mid (V_{+}, V'_{+}) \mid () \mid \mathsf{pack}\ A\ V_{+} \mid \mathsf{box}_{\mathcal{S}}\ V_{\mathcal{S}} \mid \mathsf{val}_{+}\ V_{+} \mid \lambda\mathsf{enter}_{+}.M_{-} \\
V_{\star} &::= x \mid \mathsf{val}_{\star}\ V_{+} \mid \lambda\mathsf{enter}_{\star}.M_{-}
\end{aligned}
$$

Figure 16 The focused sub-syntax of $\mathcal{F}$.

computation using $\kappa^{\star}_{F}$:

$$
\begin{aligned}
&\mathbf{case}\ \mathsf{box}_{\star}\ R_{\star}\ \mathbf{of}\{\mathsf{box}_{\star}(x{:}\star).M\} \\
&\mapsto_{\varsigma_P} \mathbf{case}\ (\mathbf{let}\ y{:}\star = R_{\star}\ \mathbf{in}\ \mathsf{box}_{\star}(y{:}\star))\ \mathbf{of}\{\mathsf{box}_{\star}(x{:}\star).M\} \\
&\mapsto_{\kappa^{\star}_{F}} \mathbf{let}\ y{:}\star = R_{\star}\ \mathbf{in}\ \mathbf{case}\ \mathsf{box}_{\star}(y{:}\star)\ \mathbf{of}\{\mathsf{box}_{\star}(x{:}\star).M\} \\
&\mapsto_{\beta_p} \mathbf{let}\ y{:}\star = R_{\star}\ \mathbf{in}\ M[y{:}\star/x{:}\star]
\end{aligned}
$$

which will proceed from here by evaluating $M[y{:}\star/x{:}A]$ in the context of the shared binding $y{:}\star = R_{\star}$.

F Core $\mathcal{F}$ and $\mathcal{D}$ calculi

Here we illustrate the $\mathcal{F}$ instance of the extensible functional λ-calculus and the $\mathcal{D}$ instance of the extensible dual sequent calculus, by inlining the respective (co-)data declarations into the syntax and typing rules. These two special instances serve as distinguished *core calculi* that can represent all the other extensions via the encodings shown in Section 5 and Appendix D.

F.1 The core $\mathcal{F}$ functional λ-calculus

The syntax of the core $\mathcal{F}$ calculus is given in Figure 15, the *focused* sub-syntax of the $\mathcal{F}$ calculus is given in Figure 16, and the specific typing rules for the $\mathcal{F}$ connectives are given in Figure 17.

$$
\begin{array}{llll}
\& : - \to - \to - & \to : + \to - \to - & \top : - & \\
\oplus : + \to + \to + & \otimes : + \to + \to + & 0 : + & 1 : + \\
\forall_k : (k \to -) \to - & \downarrow_{\mathcal{S}} : \mathcal{S} \to + & {}_{\mathcal{S}}\Uparrow : + \to \mathcal{S} & \\
\exists_k : (k \to +) \to + & \uparrow_{\mathcal{S}} : \mathcal{S} \to - & {}_{\mathcal{S}}\Downarrow : - \to \mathcal{S} &
\end{array}
$$

$$
\dfrac{\Gamma, x : A : + \vdash^{\Theta}_{\mathcal{F}} M : B : -}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\mathsf{call}(x{:}A).M : A \to B : -} \;{\to}I
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A \to B : - \quad \Gamma \vdash^{\Theta}_{\mathcal{F}} N : A : +}{\Gamma \vdash^{\Theta}_{\mathcal{F}} M.\mathsf{call}\, N : B : -} \;{\to}E
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A : - \quad \Gamma \vdash^{\Theta}_{\mathcal{F}} N : B : -}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\{\pi_1.M \mid \pi_2.N\} : A \,\&\, B : -} \;\&I
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A_1 \,\&\, A_2 : -}{\Gamma \vdash^{\Theta}_{\mathcal{F}} M.\pi_i : A_i : -} \;\&E_i
\qquad
\dfrac{(\Gamma \vdash^{\Theta}_{\mathcal{F}})\,\mathbf{ctx}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\{\} : \top : -} \;\top I
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A_i : +}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \iota_i M : A_1 \oplus A_2 : +} \;\oplus I_i
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A \oplus B : + \quad \Gamma, x : A : + \vdash^{\Theta}_{\mathcal{F}} N_1 : C : \mathcal{R} \quad \Gamma, y : B : + \vdash^{\Theta}_{\mathcal{F}} N_2 : C : \mathcal{R}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{\iota_1(x{:}A).N_1 \mid \iota_2(y{:}A).N_2\} : C : \mathcal{R}} \;\oplus E
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A : + \quad \Gamma \vdash^{\Theta}_{\mathcal{F}} N : B : +}{\Gamma \vdash^{\Theta}_{\mathcal{F}} (M, N) : A \otimes B : +} \;\otimes I
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A \otimes B : + \quad \Gamma, x : A : +, y : B : + \vdash^{\Theta}_{\mathcal{F}} N : C : \mathcal{R}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{(x{:}A, y{:}B).N\} : C : \mathcal{R}} \;\otimes E
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : 0 : + \quad \Theta \vdash C : \mathcal{R} \quad (\Gamma \vdash^{\Theta}_{\mathcal{F}})\,\mathbf{ctx}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{\} : C : \mathcal{R}} \;0E
$$

$$
\dfrac{(\Gamma \vdash^{\Theta}_{\mathcal{F}})\,\mathbf{ctx}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} () : 1 : +} \;1I
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : 1 : + \quad \Gamma \vdash^{\Theta}_{\mathcal{F}} N : C : \mathcal{R}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M.\{().N\} : C : \mathcal{R}} \;1E
$$

$$
\dfrac{\Gamma \vdash^{\Theta, X:k}_{\mathcal{F}} M : A\ X : -}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\mathsf{spec}(X{:}k).M : \forall_k A : -} \;\forall I_k
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : \forall_k A : - \quad \Theta \vdash B : k}{\Gamma \vdash^{\Theta}_{\mathcal{F}} M.\mathsf{spec}\, B : A\ B : -} \;\forall E_k
$$

$$
\dfrac{\Theta \vdash B : k \quad \Gamma \vdash^{\Theta}_{\mathcal{F}} M : A\ B : +}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathsf{pack}_k\ B\ M : \exists_k A : +} \;\exists I_k
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : \exists_k A : + \quad \Gamma, y : A\ X : + \vdash^{\Theta, X:k}_{\mathcal{F}} N : C : \mathcal{R} \quad \Theta \vdash C : \mathcal{R}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{\mathsf{pack}(X{:}k)(y{:}A).N\} : C : \mathcal{R}} \;\exists E_k
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A : \mathcal{S}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathsf{box}_{\mathcal{S}}\, M : \downarrow_{\mathcal{S}} A : +}
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : \downarrow_{\mathcal{S}} A : + \quad \Gamma, x : A : \mathcal{S} \vdash^{\Theta}_{\mathcal{F}} N : C : \mathcal{R}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{\mathsf{box}_{\mathcal{S}}(x{:}A).N\} : C : \mathcal{R}}
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A : -}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\mathsf{enter}_{\mathcal{S}}.M : {}_{\mathcal{S}}\Downarrow A : \mathcal{S}}
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : {}_{\mathcal{S}}\Downarrow A : \mathcal{S}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} M.\mathsf{enter}_{\mathcal{S}} : A : -}
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A : \mathcal{S}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \lambda\mathsf{eval}_{\mathcal{S}}.M : \uparrow_{\mathcal{S}} A : -}
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : \uparrow_{\mathcal{S}} A : -}{\Gamma \vdash^{\Theta}_{\mathcal{F}} M.\mathsf{eval}_{\mathcal{S}} : A : \mathcal{S}}
$$

$$
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : A : +}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathsf{val}_{\mathcal{S}}\, M : {}_{\mathcal{S}}\Uparrow A : \mathcal{S}}
\qquad
\dfrac{\Gamma \vdash^{\Theta}_{\mathcal{F}} M : {}_{\mathcal{S}}\Uparrow A : \mathcal{S} \quad \Gamma, x : A : + \vdash^{\Theta}_{\mathcal{F}} N : C : \mathcal{R}}{\Gamma \vdash^{\Theta}_{\mathcal{F}} \mathbf{case}\, M\, \mathbf{of}\{\mathsf{val}_{\mathcal{S}}(x{:}A).N\} : C : \mathcal{R}}
$$

Figure 17 Typing rules of $\mathcal{F}$ connectives.

The focused sub-syntax of $\mathcal{F}$ defined by the normal forms of $\mathcal{F}$ with respect to the ς rules from the operational semantics. Since these normal forms are closed under further reduction, the focused sub-syntax is a calculus in its own right. Focusing has the effect of restricting the place where *non-values* may occur: the parameters to constructors and observers *must* always be values. For this reason, the focused sub-syntax is split according to the three different disciplines $+$, $-$, $\star$, since the definition of syntactically valid terms is intertwined with the discipline-dependent definition of values. The first thing to note is that all the potential non-values $M_{\mathcal{S}}$ are defined generically for each $\mathcal{S}$: these consist of **case** analysis, **let**-bindings, as well as the observation $M_{\star}.\mathsf{eval}_{\mathcal{S}}$ which forces the evaluation of the negative term $M_\star : \Uparrow_{\mathcal{S}} A$ to get a result of type $A : \mathcal{S}$. For example, when $M_\star.\mathsf{eval}_+$ is a $+$ non-value; in fact, the only non-value which is not a **let** or **case**.

The definition of values varies widely depending on types available at each discipline. For example, there are few $\star$ values because there are only two $\star$ types, ${}_\star\!\Uparrow\! A$ and ${}_\star\!\Downarrow\! A$. The $+$ values includes the usual pairs and sums, as well as a $\mathsf{box}_{\mathcal{S}}\, M_{\mathcal{S}} : \downarrow_{\mathcal{S}} A$ containing a value of another discipline $\mathcal{S}$ and a thunk $\lambda\mathsf{enter}_+.M_- : {}_+\!\Downarrow\! A$ containing a $-$ computation. The negative values also contains functions, products, their observations, and the terms for shifts into $-$. The shifts consist of the ones common to every discipline ($\mathsf{val}_{\mathcal{S}}\, V_{\mathcal{S}}$ and $\lambda\mathsf{enter}_{\mathcal{S}}.M_-$) as well as the observation $M_{\mathcal{S}}.\mathsf{eval}_{\mathcal{S}}$ which allows a (potential non-value) term $M_{\mathcal{S}}$ to appear as a $-$ value.

Note that since everything is substitutable in call-by-name evaluation, the $-$ set of values also contains within it all the $-$ terms; M_- and V_- is synonymous. This is one reason why in call-by-push-value [11], which is a calculus consisting of only substitutable terms, all computation must happen in negative types; it is the only place where computation is substitutable. In this way, $\mathcal{F}$ can be seen as a conservative extension of call-by-push-value. If we eliminate the call-by-need part, and encode the $\downarrow$ shifts and $\uparrow$ shifts in terms of $\Downarrow$ and $\Uparrow$ as per the isomorphism, then we can further simplify the sub-calculus via $\beta_{let}\kappa$ normalization, which entirely eliminates **let**-bindings form the syntax and restricts the discriminant of a **case** to be a value. Further, by converting a positive computation M_+ into a negative value $\mathsf{val}_-\, M_+$, this same normalization also eliminates postive non-values, yielding a smaller sub-calculus in the style of call-by-push-value.

▶ **Definition 17.** The *call-by-push-value sub-calculus* of $\mathcal{F}$ is the restriction of the focused sub-syntax of $\mathcal{F}$ to only values, only the kinds $+$, and $-$, all $\mathcal{F}$ connectives except for the shifts $\downarrow$ and $\uparrow$, and only variables of positive types.

▶ **Theorem 18.** *There exists a typed translation from the $+/-$ restriction of the $\mathcal{F}$ calculus to the call-by-push-value sub-calculus of $\mathcal{F}$ with the following types:*

- *The interesting cases of $[\![A]\!]$ are defined as $[\![\uparrow]\!] \triangleq \Uparrow$ and $[\![\downarrow]\!] \triangleq \Downarrow$, and $[\![A]\!]$ is defined homomorphically otherwise,*
- *For all $x{:}A{:}+ \in \Gamma$ and $y{:}B{:}- \in \Gamma$, there is $x{:}[\![A]\!]{:}+ \in [\![\Gamma]\!]$ and $y{:}\Downarrow[\![B]\!]{:}+ \in [\![\Gamma]\!]$,*
- *If $\Gamma \vdash V_{\mathcal{S}} : A : \mathcal{S}$ then $[\![\Gamma]\!] \vdash [\![V_{\mathcal{S}}]\!] : [\![A]\!] : \mathcal{S}$,*
- *If $\Gamma \vdash M_- : A : -$ then $[\![\Gamma]\!] \vdash [\![M_-]\!] : [\![A]\!] : -$, and*
- *If $\Gamma \vdash M_+ : A : +$ then $[\![\Gamma]\!] \vdash [\![M_+]\!] : \Uparrow[\![A]\!] : -$.*

This translation forms an equational correspondence between $\mathcal{F}$ and its call-by-push-value sub-calculus.

Proof. The translation follows the same procedure described above.

1. Convert all uses of the $\downarrow$ and $\uparrow$ shifts to $\Downarrow$ and $\Uparrow$ shifts, according to the isomorphism between those connectives shown in Theorem 3.
2. Replace all negative free and bound variables $y{:}B{:}- \in \Gamma$ with positive ones $y{:}\Downarrow B{:}+ \in \Gamma$ by substituting $y.\mathsf{enter}$ for y.
3. Reduce the term to the focused sub-syntax of $\mathcal{F}$ by normalizing with respect to the ς reductions (ς reduction is strongly normalizing).
4. Eliminate all **let**s and all **case**s with a non-value discriminant by normalizing with respect to the κ and β_{let} reductions ($\kappa\beta_{let}$ reduction is strongly normalizing).
5. Shift each positive non-value $M : A : +$ as $\mathsf{val}\, M : \Uparrow A : -$, and propagate the val constructor into M to the possible return values via the κ reductions.

$$
\begin{array}{l}
A, B, C ::= X \mid \mathsf{F} \mid \lambda \boldsymbol{X}.A \mid A\ B \quad \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} ::= X{:}k \quad k, l ::= \mathcal{S} \mid k \to l \quad \mathcal{R}, \mathcal{S}, \mathcal{T} ::= + \mid - \mid \star \mid \bigstar \\
\qquad \mathsf{F}, \mathsf{G} ::= \& \mid \parr \mid \top \mid \bot \mid \oplus \mid \otimes \mid 0 \mid 1 \mid \neg \mid \ominus \mid \forall_k \mid \exists_k \mid {\downarrow_{\mathcal{S}}} \mid {\uparrow_{\mathcal{S}}} \mid {}_{\mathcal{S}}{\Uparrow} \mid {}_{\mathcal{S}}{\Downarrow} \\
p ::= \iota_1 \boldsymbol{x} \mid \iota_2 \boldsymbol{x} \mid (\boldsymbol{x}, \boldsymbol{y}) \mid () \mid \mathsf{cont}\, \boldsymbol{\alpha} \mid \mathsf{pack}\, \boldsymbol{X}\ \boldsymbol{y} \mid \mathsf{box}_{\mathcal{S}}\, \boldsymbol{x} \mid \mathsf{val}_{\mathcal{S}}\, \boldsymbol{x} \qquad \qquad \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} ::= x{:}A \\
q ::= \pi_1 \boldsymbol{\alpha} \mid \pi_2 \boldsymbol{\alpha} \mid [\boldsymbol{\alpha}, \boldsymbol{\beta}] \mid [] \mid \mathsf{throw}\, \boldsymbol{x} \mid \mathsf{spec} \boldsymbol{X} \boldsymbol{\alpha} \mid \mathsf{eval}_{\mathcal{S}}\, \boldsymbol{\alpha} \mid \mathsf{enter}_{\mathcal{S}}\, \boldsymbol{\alpha} \qquad \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta} ::= \alpha{:}A \\
c ::= \langle v \| e \rangle \\
v ::= x \mid \mu \boldsymbol{\alpha}.c \mid \nu \boldsymbol{x}.v \mid \lambda\{q_i.c_i \mid \overset{i}{\ldots}\} \mid \iota_1 v \mid \iota_2 v \mid (v, v') \mid () \mid \mathsf{cont}\, e \mid \mathsf{pack}\, A\ v \mid \mathsf{box}_{\mathcal{S}}\, v \mid \mathsf{val}_{\mathcal{S}}\, v \\
e ::= \alpha \mid \tilde{\mu} \boldsymbol{x}.c \mid \tilde{\nu} \boldsymbol{\alpha}.e \mid \tilde{\lambda}\{p_i.c_i \mid \overset{i}{\ldots}\} \mid \pi_1 e \mid \pi_2 e \mid [e, e'] \mid [] \mid \mathsf{throw}\, v \mid \mathsf{spec}\, A\ e \mid \mathsf{eval}_{\mathcal{S}}\, e \mid \mathsf{enter}_{\mathcal{S}}\, e
\end{array}
$$

Figure 18 Syntax of $\mathcal{D}$: a multi-discipline sequent calculus.

$$
c ::= \langle v \| e \rangle \qquad v_{\mathcal{S}} ::= V_{\mathcal{S}} \mid \mu \alpha{:}A{:}\mathcal{S}.c \qquad e_{\mathcal{S}} ::= E_{\mathcal{S}} \mid \tilde{\mu} x{:}A{:}\mathcal{S}.c
$$

$$
\begin{array}{rll rll}
V_+ & ::= & x \mid \iota_1 V_+ \mid \iota_2 V_+ & E_+ & ::= & x \mid \tilde{\mu} x{:}A{:}{+}.c \mid \tilde{\nu} \boldsymbol{\alpha}.E_+ \\
& \mid & (V_+, V'_+) \mid () & & \mid & \tilde{\lambda}\{\iota_1 \boldsymbol{x}.c_1 \mid \iota_2 \boldsymbol{y}.c_2\} \mid \tilde{\lambda}\{\} \\
& \mid & \mathsf{cont}\, E_- \mid \mathsf{pack}\, A\ V_+ & & \mid & \tilde{\lambda}\{(\boldsymbol{x}, \boldsymbol{y}).c\} \mid \tilde{\lambda}\{().c\} \\
& \mid & \mathsf{box}_{\mathcal{S}}\, V_{\mathcal{S}} \mid \mathsf{val}_+\, V_+ & & \mid & \tilde{\lambda}\{\mathsf{cont}\, \boldsymbol{\alpha}.c\} \mid \tilde{\lambda}\{\mathsf{pack}\, \boldsymbol{Y}\ \boldsymbol{x}.c\} \\
& \mid & \lambda\{\mathsf{eval}_+\, \boldsymbol{\alpha}.c\} & & \mid & \tilde{\lambda}\{\mathsf{box}_{\mathcal{S}}\, \boldsymbol{x}.c\} \mid \tilde{\lambda}\{\mathsf{val}_+\, \boldsymbol{x}.c\} \\
& & & & \mid & \mathsf{eval}_+\, E_+ \\
E_- & ::= & \alpha \mid \pi_1 E_- \mid \pi_2 E_- & V_- & ::= & x \mid \mu \alpha{:}A{:}{-}.c \mid \nu \boldsymbol{x}.V_- \\
& \mid & [E_-, E'_-] \mid [] & & \mid & \lambda\{\pi_1 \boldsymbol{\alpha}.c_1 \mid \pi_2 \boldsymbol{\beta}.c_2\} \mid \lambda\{\} \\
& \mid & \mathsf{throw}\, V_- \mid \mathsf{spec}\, A\ E_- & & \mid & \lambda\{[\boldsymbol{\alpha}, \boldsymbol{\beta}].c\} \mid \lambda\{[].c\} \\
& \mid & \mathsf{eval}_{\mathcal{S}}\, E_{\mathcal{S}} \mid \mathsf{enter}_-\, E_- & & \mid & \lambda\{\mathsf{throw}\, \boldsymbol{x}.c\} \mid \lambda\{\mathsf{spec}\, \boldsymbol{Y}\ \boldsymbol{\alpha}.c\} \\
& \mid & \tilde{\lambda}\{\mathsf{box}_-\, \boldsymbol{x}.c\} & & \mid & \lambda\{\mathsf{enter}_-\, \boldsymbol{\alpha}.c\} \mid \lambda\{\mathsf{eval}_{\mathcal{S}}\, \boldsymbol{\alpha}.c\} \\
& & & & \mid & \mathsf{val}_-\, V_+ \\
V_{\bigstar} & ::= & x \mid \mathsf{val}_{\bigstar}\, V_+ \mid \lambda\{\mathsf{enter}_{\bigstar}\, \boldsymbol{\alpha}.c\} & E_{\bigstar} & ::= & \alpha \mid \tilde{\mu} x{:}A{:}{\bigstar}.H[\langle x \| E_{\bigstar} \rangle] \mid \tilde{\lambda}\{\mathsf{val}_{\bigstar}\, \boldsymbol{x}.c\} \mid \mathsf{enter}_{\bigstar}\, E_- \\
E_{\star} & ::= & \alpha \mid \mathsf{enter}_{\star}\, E_- \mid \tilde{\lambda}\{\mathsf{val}_{\star}\, \boldsymbol{x}.c\} & V_{\star} & ::= & x \mid \mu \alpha{:}A{:}{\star}.H[\langle V_{\star} \| \alpha \rangle] \mid \lambda\{\mathsf{enter}_{\star}\, \boldsymbol{\alpha}.c\} \mid \mathsf{val}_{\star}\, V_+
\end{array}
$$

Figure 19 The focused sub-syntax of $\mathcal{D}$.

Each of these steps follows by induction on the typing derivation. Since each of these transformation steps is itself an equational correspondence (where the reverse transformation is just syntactic inclusion into the larger calculus), the composition of entire procedure is an equational correspondence. ◀

F.2 The core $\mathcal{D}$ dual sequent calculus

The syntax of the core dual calculus is given in Figure 18, the *focused* sub-syntax of the $\mathcal{D}$ calculus is given in Figure 19, and the specific typing rules for the $\mathcal{D}$ connectives are given in Figure 20.

The definition of focusing is the same in $\mathcal{D}$ as it was in $\mathcal{F}$: the normalization with respect to ς reduction. Because $\mathcal{D}$ is a symmetric language, this has the effect of carving out the values *and* co-values of every discipline. Thanks to the symmetry between data and co-data abstractions, the definition of (potential) non-value terms and non-co-value co-terms is much more regular: for every discipline $\mathcal{S}$, a general term is either a $\mathcal{S}$ value or a $\mathcal{S}$ μ-abstraction, and dually a general co-term is either a $\mathcal{S}$ co-value or a $\mathcal{S}$ $\tilde{\mu}$-abstraction.

$$
\begin{array}{llll}
\& : - \to - \to - & ⅋ : - \to - \to - & \top : - \quad \bot : - & \neg : + \to - \\
\oplus : + \to + \to + & \otimes : + \to + \to + & 0 : + \quad 1 : + & \ominus : - \to + \\
\forall_k : (k \to -) \to - & \upharpoonleft_{\mathcal{S}} : \mathcal{S} \to - & {}_{\mathcal{S}}\Downarrow : - \to \mathcal{S} & \\
\exists_k : (k \to +) \to + & \downharpoonleft_{\mathcal{S}} : \mathcal{S} \to + & {}_{\mathcal{S}}\Uparrow : + \to \mathcal{S} &
\end{array}
$$

$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} v : A_i \mid \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \iota_i v : A_1 \oplus A_2 \mid \Delta}\,\oplus R_i \qquad \frac{c_1 : (\Gamma, x : A \vdash^{\Theta}_{\mathcal{D}} \Delta) \quad c_2 : (\Gamma, y : B \vdash^{\Theta}_{\mathcal{D}} \Delta)}{\Gamma \mid \tilde{\lambda}\{\iota_1(x{:}A).c_1 \mid \pi_2(y{:}B).c_2\} : A \oplus B \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\oplus L$$

$$\frac{\Gamma \mid e : A_i \vdash^{\Theta}_{\mathcal{D}} \Delta}{\Gamma \mid \pi_i e : A_1 \mathbin{\&} A_2 \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\& L_i \qquad \frac{c_1 : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \alpha : A, \Delta) \quad c_2 : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \beta : B, \Delta)}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{\pi_1[\alpha{:}A].c_1 \mid \pi_2[\beta{:}B].c_2\} : A \mathbin{\&} B \mid \Delta}\,\& R$$

$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} v_1 : A \mid \Delta \quad \Gamma \vdash^{\Theta}_{\mathcal{D}} v_2 : B \mid \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} (v_1, v_2) : A \otimes B \mid \Delta}\,\otimes R \qquad \frac{c : (\Gamma, x : A, y : B \vdash^{\Theta}_{\mathcal{D}} \Delta)}{\Gamma \mid \tilde{\lambda}\{(x{:}A, y{:}B).c\} : A \otimes B \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\otimes L$$

$$\frac{\Gamma \mid e_1 : A \vdash^{\Theta}_{\mathcal{D}} \Delta \quad \Gamma \mid e_2 : B \vdash^{\Theta}_{\mathcal{D}} \Delta}{\Gamma \mid [e_1, e_2] : A ⅋ B \vdash^{\Theta}_{\mathcal{D}} \Delta}\,⅋ L \qquad \frac{c : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \alpha : A, \beta : B, \Delta)}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{[\alpha{:}A, \beta{:}B].c\} : A ⅋ B \mid \Delta}\,⅋ R$$

$$\frac{}{\Gamma \mid \tilde{\lambda}\{\} : 0 \vdash^{\Theta}_{\mathcal{D}} \Delta}\,0L \qquad \frac{}{\Gamma \vdash^{\Theta}_{\mathcal{D}} () : 1 \mid \Delta}\,1R \qquad \frac{}{\Gamma \mid [] : \bot \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\bot L \qquad \frac{}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{\} : \top \mid \Delta}\,\top R$$

$$\frac{\Gamma \mid e : A \vdash^{\Theta}_{\mathcal{D}} \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \mathsf{cont}\, e : \ominus A \mid \Delta}\,\ominus R \qquad \frac{c : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \alpha : A, \Delta)}{\Gamma \mid \tilde{\lambda}\{\mathsf{cont}[\alpha{:}A].c\} : \ominus A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\ominus L$$

$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} v : A \mid \Delta}{\Gamma \mid \mathsf{throw}\, v : \neg A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\neg L \qquad \frac{c : (\Gamma, x : A \vdash^{\Theta}_{\mathcal{D}} \Delta)}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{\mathsf{throw}(x{:}A).c\} : \neg A \mid \Delta}\,\neg R$$

$$\frac{\Theta \vdash_{\mathcal{D}} B : k \quad \Gamma \vdash^{\Theta}_{\mathcal{D}} v : A\ B \mid \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \mathsf{pack}\, B\ v : \exists_k A \mid \Delta}\,\exists R \qquad \frac{c : (\Gamma, y : A\ X \vdash^{\Theta, X : k}_{\mathcal{D}} \Delta)}{\Gamma \mid \tilde{\lambda}\{\mathsf{pack}(X{:}k)(y{:}A).c\} : \exists_k A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\exists L$$

$$\frac{\Theta \vdash_{\mathcal{D}} B : k \quad \Gamma \mid e : A\ B \vdash^{\Theta}_{\mathcal{D}} \Delta}{\Gamma \mid \mathsf{spec}\, B\ e : \forall_k A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\forall L \qquad \frac{c : (\Gamma \vdash^{\Theta, X : k}_{\mathcal{D}} \alpha : A\ X, \Delta)}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{\mathsf{spec}[X{:}k][\alpha{:}A].c\} : \forall_k A \mid \Delta}\,\forall R$$

$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} v : A \mid \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \mathsf{box}_{\mathcal{S}}\, v : \downharpoonleft_{\mathcal{S}} A \mid \Delta}\,\downharpoonleft R \qquad \frac{c : (\Gamma, x : A \vdash^{\Theta}_{\mathcal{D}} \Delta)}{\Gamma \mid \tilde{\lambda}\{\mathsf{box}_{\mathcal{S}}(x{:}A).c\} : \downharpoonleft_{\mathcal{S}} A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\downharpoonleft L$$

$$\frac{\Gamma \mid e : A \vdash^{\Theta}_{\mathcal{D}} \Delta}{\Gamma \mid \mathsf{eval}_{\mathcal{S}}\, e : \upharpoonleft_{\mathcal{S}} A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\upharpoonleft L \qquad \frac{c : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \alpha : A, \Delta)}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{\mathsf{eval}_{\mathcal{S}}(\alpha{:}A).c\} : \upharpoonleft_{\mathcal{S}} A \mid \Delta}\,\upharpoonleft R$$

$$\frac{\Gamma \vdash^{\Theta}_{\mathcal{D}} v : A \mid \Delta}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \mathsf{val}_{\mathcal{S}}\, v : {}_{\mathcal{S}}\Uparrow A \mid \Delta}\,\Uparrow R \qquad \frac{c : (\Gamma, x : A \vdash^{\Theta}_{\mathcal{D}} \Delta)}{\Gamma \mid \tilde{\lambda}\{\mathsf{val}_{\mathcal{S}}(x{:}A).c\} : \downharpoonleft_{\mathcal{S}} A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\Uparrow L$$

$$\frac{\Gamma \mid e : A \vdash^{\Theta}_{\mathcal{D}} \Delta}{\Gamma \mid \mathsf{enter}_{\mathcal{S}}\, e : {}_{\mathcal{S}}\Downarrow A \vdash^{\Theta}_{\mathcal{D}} \Delta}\,\Downarrow L \qquad \frac{c : (\Gamma \vdash^{\Theta}_{\mathcal{D}} \alpha : A, \Delta)}{\Gamma \vdash^{\Theta}_{\mathcal{D}} \lambda\{\mathsf{enter}_{\mathcal{S}}(\alpha{:}A).c\} : {}_{\mathcal{S}}\Downarrow A \mid \Delta}\,\Downarrow R$$

Figure 20 Typing rules of $\mathcal{D}$ connectives.

Thanks to the symmetry of the $\mathcal{D}$ connectives, the definitions of values and co-values also enjoy a similar symmetry. Positive values V_+ are dual to negative co-values E_-, and positive co-values E_- (which are synonymous with positive co-terms e_-) are dual to negative values V_- (which are synonymous with negative terms v_-). Likewise, $\star$ values and co-values are dual to $\star$ co-values and values, respectively.

Also note that, like before, ignoring call-by-need and its dual lets us eliminate the need to consider terms and co-terms which are not substitutable value and co-values. In particular,

$\beta_\mu\beta_{\tilde{\mu}}$ reduction is strongly normalizing, and μ- and $\tilde{\mu}$-abstraction can only appear within a cut. Since every $+$ and $-$ cut with at least one μ- or $\tilde{\mu}$-abstraction can reduce by β_μ either or $\beta_{\tilde{\mu}}$, they can all be normalized away at the cost of duplication from substitution. By eliminating μ and $\tilde{\mu}$, we end up with a calculus in the style of the calculus of unity [25] where everything is either a substitutable value or co-value, or a command which represents all computation. Note that the only types which step outside the coupling of data types with positive types, and co-data types with negative types, are the $\Downarrow$ and $\Uparrow$ shifts that introduce a λ-abstraction in V_+ and E_-. But since these two shifts are isomorphic to $\downarrow$ and $\uparrow$, they can be encoded away, leaving a calculus that corresponds to the calculus of unity. The final translation into the full calculus of unity requires the use of nested patterns and co-patterns to eliminate positive variables and negative co-variables of non-atomic types. With nested (co-)patterns available, η-expansion can be performed until only the allowed variables and co-variables remain.

▶ **Definition 19.** The *flat unity sub-calculus* of $\mathcal{D}$ is the restriction of the focused sub-syntax of $\mathcal{D}$ to only values and co-values, no μ- or $\tilde{\mu}$-abstractions, only the kinds $+$ and $-$, all $\mathcal{D}$ connectives except for the shifts $\Uparrow$ and $\Downarrow$. Note that this sub-calculus does not account for nested (co-)pattern matching, as in the calculus of unity [25]. By extending $\mathcal{D}$ with nested (co-)patterns, we then have the *full unity sub-calculus* of $\mathcal{D}$ with nested (co-)patterns, which meets each of the *flat unity sub-calculus* restrictions as well as the additional restriction that all variables have a negative type or an atomic positive type ($X : +$) and all co-variables have a positive type or an atomic negative type ($X : -$).

▶ **Theorem 20.** *There exists a typed translation from the $+/-$ restriction of the $\mathcal{D}$ calculus to the flat unity sub-calculus of $\mathcal{D}$ with the following types:*

- *If $c : (\Gamma \vdash \Delta)$ then $[\![c]\!] : ([\![\Gamma]\!] \vdash [\![\Delta]\!])$,*
- *If $\Gamma \vdash V_\mathcal{S} : A : \mathcal{S} \mid \Delta$ then $[\![\Gamma]\!] \vdash [\![V_\mathcal{S}]\!] : [\![A]\!] : \mathcal{S} \mid [\![\Delta]\!]$,*
- *If $\Gamma \mid E_\mathcal{S} : A : \mathcal{S} \vdash \Delta$ then $[\![\Gamma]\!] \mid [\![E_\mathcal{S}]\!] : [\![A]\!] : \mathcal{S} \vdash [\![\Delta]\!]$,*
- *If $\Gamma \vdash v_+ : A : + \mid \Delta$ then $[\![\Gamma]\!] \vdash [\![v_+]\!] : \uparrow[\![A]\!] : - \mid [\![\Delta]\!]$, and*
- *If $\Gamma \mid e_- : A : - \vdash \Delta$ then $[\![\Gamma]\!] \mid [\![e_-]\!] : \downarrow[\![A]\!] : + \vdash [\![\Delta]\!]$.*

Where the interesting cases of $[\![A]\!]$ are defined as $[\![\Uparrow]\!] \triangleq [\![\uparrow]\!]$ and $[\![\Downarrow]\!] \triangleq [\![\downarrow]\!]$ and all remaining cases are defined homomorphically, and $[\![\Gamma]\!]$ and $[\![\Delta]\!]$ are defined pointwise on the types of bound (co-)variables. Furthermore, there exists a typed translation using nested (co-)patterns from the flat unity sub-calculus of $\mathcal{D}$ to the full unity sub-calculus of $\mathcal{D}$ with the following types, where Γ has no variables of non-atomic positive types and Δ has no co-variables of non-atomic negative types:

- *If $c : (\Gamma \vdash \Delta)$ then $[\![c]\!] : (\Gamma \vdash \Delta)$,*
- *If $\Gamma \vdash V : A : \mathcal{S} \mid \Delta$ then $\Gamma \vdash [\![V]\!] : A : \mathcal{S} \mid \Delta$, and*
- *If $\Gamma \mid E : A : \mathcal{S} \vdash \Delta$ then $\Gamma \mid [\![E]\!] : A : \mathcal{S} \vdash \Delta$.*

Both of these translations form an equational correspondence between each of dual*, its flat unity sub-calculus, and its full unity sub-calculus.*

Proof. The translations follow the procedure outlined above. For the translation of $\mathcal{D}$ into its flat unity sub-calculus:

1. Convert all uses of the $\Uparrow$ and $\Downarrow$ shifts to $\uparrow$ and $\downarrow$ shifts, according to the isomorphism between those connectives shown in Theorem 3.

2. Reduce the expression (command, term, or co-term) to the focused sub-syntax of $\mathcal{D}$ by normalizing with respect to the ς reductions (ς reduction is strongly normalizing).
3. Eliminate any internal positive μ-abstractions and negative $\tilde{\mu}$-abstractions by normalizing with respect to β_μ and $\beta_{\tilde{\mu}}$ reductions ($\beta_\mu\beta_{\tilde{\mu}}$ reduction is strongly normalizing). The final top-level μ- or $\tilde{\mu}$-abstraction can be eliminated by replacing it with the (co-)pattern match for a shift type. That is, the positive non-value term $\Gamma \vdash \mu\alpha.c : A : + \mid \Delta$ becomes $\Gamma \vdash \lambda\{\mathsf{eval}\,\alpha.c\} : \uparrow A : - \mid \Delta$ and the negative non-co-value co-term $\Gamma \mid \tilde{\mu}x.c : A : - \vdash \Delta$ becomes $\Gamma \mid \tilde{\lambda}\{\mathsf{box}\,x.c\} : \downarrow A : + \mid \Delta$.
4. Eliminate any negative μ-abstractions and positive $\tilde{\mu}$-abstractions by the η-expansion appropriate for the type of the abstraction.

The translation from the flat unity sub-calculus to the full unity sub-calculus uses nested (co-)patterns to repeat the final step of η-expansion above for every bound (co-)variables of a disallowed type based (this process terminates because each step reduces the type of a bound (co-)variables, since the types of $\mathcal{D}$ are non-recursive). Each of these steps follows by induction on the typing derivation. Since each of these steps are individually an equational correspondence (where the reverse translation is just syntactic inclusion into the larger calculus), the composition of the whole procedure also forms an equational correspondence. ◀

G Functional-sequent correspondence

▶ **Definition 21.** The extensible λ-calculus has the following sub-calculi: the *pure sub-calculus* excludes μ-abstractions and jumps, and the *totally pure sub-calculus* also excludes fixed point terms $\nu\boldsymbol{x}.M$.

▶ **Definition 22.** A co-data declaration is *functional* if and only if every observer in the declaration has exactly one output (to the right of $\vdash$), and a data declaration is *functional* if and only if every constructor in the declaration has exactly one output (which must be the declared data type itself). We say that a global environment is *functional* when it contains only functional (co-)data declarations.

The *functional sub-calculus* of the extensible dual calculus excludes the $\star$ discipline, co-fixed point co-terms $\tilde{\nu}\boldsymbol{x}.e$, and non-functional (co-)data type declarations. In other words, patterns in the functional sub-calculus have the form $\mathsf{K}\,\boldsymbol{Y}..\boldsymbol{x}..$ with no co-variables and co-patterns have the form $\mathsf{O}\,\boldsymbol{Y}..\boldsymbol{x}..\boldsymbol{\alpha}$ with exactly one co-variable.

Furthermore, the *purely functional sub-calculus* is the functional sub-calculus where terms are restricted to have zero free co-variables and commands and co-terms both restricted have exactly one free co-variable. The *totally pure functional sub-calculus* additionally excludes fixed point terms $\nu\boldsymbol{x}.v$.

▶ **Lemma 23** (Type preservation). *For all functional global environments $\mathcal{G}$, $\mathcal{F}$ terms M and jumps J, and functional dual terms v, commands c, and co-terms e:*

a) *if* $\Gamma \vdash^{\ominus}_{\mathcal{G}} M : A : \mathcal{S} \mid \Delta$ *then* $\Gamma \vdash^{\ominus}_{\mathcal{G}} \mathcal{L}[\![M]\!] : A : \mathcal{S} \mid \Delta$,
b) *if* $J : (\Gamma \vdash^{\ominus}_{\mathcal{G}} \Delta)$ *then* $\mathcal{L}[\![J]\!] : (\Gamma \vdash^{\ominus}_{\mathcal{G}} \Delta)$,
c) *if* $\Gamma \vdash^{\ominus}_{\mathcal{G}} v : A : \mathcal{S} \mid \Delta$ *then* $\Gamma \vdash^{\ominus}_{\mathcal{G}} \mathcal{N}[\![v]\!] : A : \mathcal{S} \mid \Delta$,
d) *if* $c : (\Gamma \vdash^{\ominus}_{\mathcal{G}} \alpha : A : \mathcal{S}, \Delta)$ *then* $\Gamma \vdash^{\ominus}_{\mathcal{G}} \mathcal{N}[\![c]\!]_{\alpha:A} : A : \mathcal{S} \mid \alpha : A : \mathcal{S}, \Delta$, *and*
e) *if* $\Gamma \mid e : A : \mathcal{S} \vdash^{\ominus}_{\mathcal{G}} \beta : B : \mathcal{R}, \Delta$ *then for all* $\Gamma \vdash^{\ominus}_{\mathcal{G}} M : A : \mathcal{S} \mid \beta : B : \mathcal{R}, \Delta$, $\Gamma \vdash^{\ominus}_{\mathcal{G}} \mathcal{N}[\![e]\!]_{\beta:B}[M] : B : \mathcal{R} \mid \beta : B : \mathcal{R}, \Delta$.

λ-calculus to dual calculus:

$$
\begin{aligned}
\mathcal{L}\llbracket\langle M\|\alpha\rangle\rrbracket &\triangleq \langle\mathcal{L}\llbracket M\rrbracket\|\alpha\rangle\\[1ex]
\mathcal{L}\llbracket x\rrbracket &\triangleq x\\
\mathcal{L}\llbracket\mathbf{let}\,\boldsymbol{x} = M\,\mathbf{in}\,N : C\rrbracket &\triangleq \mu\alpha{:}C.\langle\mathcal{L}\llbracket M\rrbracket\|\tilde{\mu}x.\langle\mathcal{L}\llbracket N\rrbracket\|\alpha\rangle\rangle\\
\mathcal{L}\llbracket\nu\boldsymbol{x}.M\rrbracket &\triangleq \nu\boldsymbol{x}.\mathcal{L}\llbracket M\rrbracket\\
\mathcal{L}\llbracket\mu\boldsymbol{\alpha}.J\rrbracket &\triangleq \mu\boldsymbol{\alpha}.\mathcal{L}\llbracket J\rrbracket\\[1ex]
\mathcal{L}\llbracket\mathsf{K}\,B..\ M..\rrbracket &\triangleq \mathsf{K}\,B..\ \mathcal{L}\llbracket M\rrbracket..\\
\mathcal{L}\llbracket\mathbf{case}\,M\,\mathbf{of}\{p_i.N_i\overset{i}{..}\} : C\rrbracket &\triangleq \mu\alpha{:}C.\langle\mathcal{L}\llbracket M\rrbracket\|\tilde{\lambda}\{p_i.\langle\mathcal{L}\llbracket v\rrbracket\|\alpha\rangle\overset{i}{..}\}\rangle\\[1ex]
\mathcal{L}\llbracket\lambda\{q_i.(M : A_i)\overset{i}{..}\}\rrbracket &\triangleq \lambda\{[q_i\ \alpha_i{:}A_i].\langle\mathcal{L}\llbracket M_i\rrbracket\|\alpha_i\rangle\overset{i}{..}\}\\
\mathcal{L}\llbracket M.\mathsf{O}\,B..\ N.. : C\rrbracket &\triangleq \mu\alpha{:}C.\langle\mathcal{L}\llbracket M\rrbracket\|\,\mathsf{O}\,B..\ \mathcal{L}\llbracket N\rrbracket..\ \alpha\rangle
\end{aligned}
$$

Dual calculus to λ-calculus:

$$
\begin{aligned}
\mathcal{N}\llbracket\langle v\|e\rangle\rrbracket_{\boldsymbol{\alpha}} &\triangleq \mathcal{N}\llbracket e\rrbracket_{\boldsymbol{\alpha}}[\mathcal{N}\llbracket v\rrbracket]\\[1ex]
\mathcal{N}\llbracket x\rrbracket &\triangleq x\\
\mathcal{N}\llbracket\mu\alpha{:}A.c\rrbracket &\triangleq \mu\alpha{:}A.\langle\mathcal{N}\llbracket c\rrbracket_{\alpha:A}\|\alpha\rangle\\
\mathcal{N}\llbracket\nu\boldsymbol{x}.v\rrbracket &\triangleq \nu\boldsymbol{x}.\mathcal{N}\llbracket v\rrbracket\\
\mathcal{N}\llbracket\mathsf{K}\,B..\ v..\rrbracket_{\boldsymbol{\alpha}} &\triangleq \mathsf{K}\,B..\mathcal{N}\llbracket v\rrbracket_{\boldsymbol{\alpha}}..\\
\mathcal{N}\llbracket\lambda\{q_i\ (\boldsymbol{\alpha}_i).c_i\overset{i}{..}\}\rrbracket &\triangleq \lambda\{q_i.\mathcal{N}\llbracket c_i\rrbracket_{\boldsymbol{\alpha}_i}\overset{i}{..}\}\\[1ex]
\mathcal{N}\llbracket\alpha\rrbracket_{\alpha:A}[M] &\triangleq M\\
\mathcal{N}\llbracket\beta\rrbracket_{\alpha:A}[M] &\triangleq \mu\delta{:}A.\langle M\|\beta\rangle \quad (\beta \neq \alpha, \delta \notin FV(M))\\
\mathcal{N}\llbracket\tilde{\mu}\boldsymbol{x}.c\rrbracket_{\boldsymbol{\alpha}}[M] &\triangleq \mathbf{let}\,\boldsymbol{x} = M\,\mathbf{in}\,\mathcal{N}\llbracket c\rrbracket_{\boldsymbol{\alpha}}\\
\mathcal{N}\llbracket\mathsf{O}\,B..\ v..\ e\rrbracket_{\boldsymbol{\alpha}}[M] &\triangleq \mathcal{N}\llbracket e\rrbracket_{\boldsymbol{\alpha}}[M.\mathsf{O}\,B..\ \mathcal{N}\llbracket v\rrbracket..]\\
\mathcal{N}\left\llbracket\tilde{\lambda}\{p_i.c_i\overset{i}{..}\}\right\rrbracket_{\boldsymbol{\alpha}}[M] &\triangleq \mathbf{case}\,M\,\mathbf{of}\{p_i.\mathcal{N}\llbracket c_i\rrbracket_{\boldsymbol{\alpha}}\overset{i}{..}\}
\end{aligned}
$$

Figure 21 Translations between the λ-calculus and the functional sub-calculus of the dual calculus.

Proof. Parts (a) and (b) follow by mutual induction on the syntax of λ-calculus terms and jumps, and parts (c), (d), and (e) follow by mutual induction on the syntax of dual terms, commands, and co-terms. ◀

▶ **Lemma 24** (Equational correspondence). *For all functional global environments $\mathcal{G}$, there is an untyped equational correspondence between dual and the functional sub-calculus of the dual calculus, specifically:*

a) for all λ-calculus terms M, $\mathcal{N}\llbracket\mathcal{L}\llbracket M\rrbracket\rrbracket = M$,
b) for all functional dual terms v, $\mathcal{L}\llbracket\mathcal{N}\llbracket v\rrbracket\rrbracket = v$,
c) if $M = M'$ in $\mathcal{F}$ then $\mathcal{L}\llbracket M\rrbracket = \mathcal{L}\llbracket M'\rrbracket$ in the functional sub-calculus, and
d) if $v = v'$ in the functional sub-calculus then $\mathcal{N}\llbracket v\rrbracket = \mathcal{L}\llbracket v'\rrbracket$ in the λ-calculus.

Likewise, there is a typed equational correspondence between the extensible λ-calculus and functional dual terms of the same type in the same environment.

Proof. Since the translations $\mathcal{N}[\![\_]\!]$ here and $NK[\![\_]\!]$ from [6] are the same (up to the untyped equational theory), the equational correspondence follows as an instance of Theorem 9.6 of [6] extended with (co-)fixed points and the χ axiom. The recursive terms $\nu\boldsymbol{x}.M$ are the same via translation, and are equipped with the same rewriting rule ν, so they correspond one-for-one.

Additionally, the $\chi^\star$ axioms in the two calculi are sound with respect to each other (with the help of some other axioms in the equational theory). For soundness of the dual $\chi^\star$ axiom with respect to the **let**-based one, note that $\chi^\star$

$$\langle \mu\beta{:}\star.\langle v \| \tilde{\mu}x{:}\star.c\rangle \| e\rangle =_{\chi^\star} \langle v \| \tilde{\mu}x{:}\star.\langle \mu\beta{:}\star.c \| e\rangle\rangle$$

is subsumed by $\beta_\mu^\star$ when e is a co-value, and every non-co-value is ς-equivalent to some $\tilde{\mu}$-abstraction, so it suffices to only consider the case when e *is* a $\tilde{\mu}$-abstraction (since ς is already known to be sound):

$$\begin{aligned}
&\mathcal{N}[\![\langle \mu\beta{:}\star.\langle v\|\tilde{\mu}x{:}\star.c\rangle \| \tilde{\mu}y{:}\star.c'\rangle]\!]_{\boldsymbol{\delta}}\\
&=_\alpha \ \mathbf{let}\, y{:}\star = \mu\beta{:}\star.\langle \mathbf{let}\, x{:}\star = \mathcal{N}[\![v]\!]\ \mathbf{in}\, \mathcal{N}[\![c]\!]_{\beta:\star} \| \beta\rangle\\
&\qquad \mathbf{in}\, \mathcal{N}[\![c']\!]_{\boldsymbol{\delta}}\\
&=_{\kappa_\mu} \ \mathbf{let}\, y{:}\star = \ \mathbf{let}\, x{:}\star = \mathcal{N}[\![v]\!]\\
&\qquad\qquad\qquad\quad \mathbf{in}\, \mu\beta{:}\star.\langle \mathcal{N}[\![c]\!]\beta{:}\star \| \beta\rangle\\
&\qquad \mathbf{in}\, \mathcal{N}[\![c']\!]_{\boldsymbol{\delta}}\\
&=_{\chi^\star} \ \mathbf{let}\, x{:}\star = \mathcal{N}[\![v]\!]\\
&\qquad \mathbf{in}\ \mathbf{let}\, y{:}\star = \mu\beta{:}\star.\langle \mathcal{N}[\![c]\!]_{\beta:\star} \| \beta\rangle\\
&\qquad\quad \mathbf{in}\, \mathcal{N}[\![c']\!]_{\boldsymbol{\delta}}\\
&=_\alpha \mathcal{N}[\![\langle v \| \tilde{\mu}x{:}\star.\langle \mu\beta{:}\star.c \| \tilde{\mu}y{:}\star.c'\rangle\rangle]\!]_{\boldsymbol{\delta}}
\end{aligned}$$

Going the other way, soundness of the $\chi^\star$ axiom for reassociating $\star$ **let** bindings is sound in the dual calculus as follows:

$$\begin{aligned}
&\mathcal{L}\left[\!\!\left[\begin{array}{l}\mathbf{let}\, y{:}\star = \ \mathbf{let}\, x{:}\star = M_1\ \mathbf{in}\, M_2\\ \mathbf{in}\, N\end{array}\right]\!\!\right]\\
&=_\alpha \mu\boldsymbol{\delta}.\langle \mu\beta{:}\star.\langle \mathcal{L}[\![M_1]\!] \| \tilde{\mu}x{:}\star.\langle \mathcal{L}[\![M_2]\!] \| \beta{:}\star\rangle\rangle \| \tilde{\mu}y{:}\star.\langle \mathcal{L}[\![N]\!] \| \boldsymbol{\delta}\rangle\rangle\\
&=_{\chi^\star} \mu\boldsymbol{\delta}.\langle \mathcal{L}[\![M_1]\!] \| \tilde{\mu}x{:}\star.\langle \mu\beta{:}\star.\langle \mathcal{L}[\![M_2]\!] \| \beta{:}\star\rangle \| \tilde{\mu}y{:}\star.\langle \mathcal{L}[\![N]\!] \| \boldsymbol{\delta}\rangle\rangle\rangle\\
&=_{\eta_\mu} \mu\boldsymbol{\delta}.\langle \mathcal{L}[\![M_1]\!] \| \tilde{\mu}x{:}\star.\langle \mathcal{L}[\![M_2]\!] \| \tilde{\mu}y{:}\star.\langle \mathcal{L}[\![N]\!] \| \boldsymbol{\delta}\rangle\rangle\rangle\\
&=_{\beta_\mu} \mathcal{L}\left[\!\!\left[\begin{array}{l}\mathbf{let}\, x{:}\star = M_1\\ \mathbf{in}\ \mathbf{let}\, y{:}\star = M_2\ \mathbf{in}\, N\end{array}\right]\!\!\right]
\end{aligned}$$

◀

▶ **Corollary 25** (Isomorphism correspondence). *For all functional global environments $\mathcal{G}$, $\Theta \vdash_{\mathcal{G}} A \approx B : k$ in the extensible λ-calculus if and only if $\Theta \vdash_{\mathcal{G}} A \approx B : k$ in the functional sub-calculus of the extensible dual calculus.*

H Faithfulness of the encodings

First, we show the isomorphism between the call-by-name and -value shifts.

▶ **Theorem 3.** *The following isomorphisms hold (under $\vDash_{\mathcal{F}}$) for all $\vdash A : +$ and $\vdash B : -$*

$$\updownarrow_{+} A \approx {}_{-}\Uparrow A \qquad \downarrow_{-} B \approx {}_{+}\Downarrow B \qquad \downarrow_{+} A \approx A \approx {}_{+}\Uparrow A \qquad \uparrow_{-} B \approx B \approx {}_{-}\Downarrow B$$

Proof. The isomorphism $\vDash_{\mathcal{F}} \downarrow_{-} \approx {}_{+}\Downarrow : - \to +$ is witnessed by the two terms

$$x : \downarrow_{-} Z : + \vdash^{Z:-}_{\mathcal{F}} \mathbf{case}\, x\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\lambda\{\mathsf{enter}_{+}.z\}\} : {}_{+}\Downarrow Z : +$$

$$y : {}_{+}\Downarrow Z : + \vdash^{Z:-}_{\mathcal{F}} \mathsf{box}_{-}(y.\mathsf{enter}_{+}) : \downarrow_{-} Z : +$$

Both compositions of these two terms are equal to their free variable as follows:

$$\begin{aligned}
x : \downarrow_{-} Z : + \vdash^{Z:-}_{\mathcal{F}} \ & \mathbf{let}\, y{:}{}_{+}\Downarrow Z = \mathbf{case}\, x\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\lambda\{\mathsf{enter}_{+}.z\}\} \\
& \mathbf{in}\, \mathsf{box}_{-}(y.\mathsf{enter}_{+}) \\
=_{\kappa_{\mathcal{F}}} \ & \mathbf{case}\, x\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\, \mathbf{let}\, y{:}{}_{+}\Downarrow Z = \lambda\{\mathsf{enter}_{+}.z\} \\
& \qquad \mathbf{in}\, \mathsf{box}_{-}(y.\mathsf{enter}_{+})\} \\
=_{\beta_{let}} \ & \mathbf{case}\, x\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\mathsf{box}_{-}(\lambda\{\mathsf{enter}_{+}.z\}.\mathsf{enter}_{+})\} \\
=_{\beta_{+\Downarrow}} \ & \mathbf{case}\, x\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\mathsf{box}_{-}(z)\} \\
=_{\eta_{\downarrow_{-}}} \ & x : \downarrow_{-} Z : +
\end{aligned}$$

$$\begin{aligned}
y : {}_{+}\Downarrow Z : + \vdash^{Z:-}_{\mathcal{F}} \ & \mathbf{let}\, x{:}\downarrow_{-} Z = \mathsf{box}_{-}(y.\mathsf{enter}_{+}) \\
& \mathbf{in}\, \mathbf{case}\, x\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\lambda\{\mathsf{enter}_{+}.z\}\} \\
=_{\beta_{let}} \ & \mathbf{case}\, \mathsf{box}_{-}(y.\mathsf{enter}_{+})\, \mathbf{of} \{\mathsf{box}_{-}(z{:}Z).\lambda\{\mathsf{enter}_{+}.z\}\} \\
=_{\beta_{\downarrow_{-}}} \ & \lambda\{\mathsf{enter}_{+}.(y.\mathsf{enter}_{+})\} \\
=_{\eta_{+\Downarrow}} \ & y : {}_{+}\Downarrow Z : +
\end{aligned}$$

The isomorphism $\vDash_{\mathcal{F}} \updownarrow_{+} \approx {}_{-}\Uparrow : + \to -$ follows by duality in the sequent calculus, since there is a one-for-one correspondence between type isomorphisms of the two calculi (Theorem 25). Furthermore, the details of the remaining identity shift isomorphisms are shown in [6]. ◀

For the remainder, we show that the type isomorphisms developed in [6] can be extended to higher-kinded types, quantifiers, and call-by-need evaluation. The main impact of the extension is to show the compatibility of type isomorphism with the new type constructors, and to generalize the nested (co-)patterns used in the soundness proof to include existential and universal type abstractions.

H.1 Local encoding

First, we prove that a type isomorphism gives rise to a local form of equational correspondence.

▶ **Theorem 5.** *For all isomorphic types $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$, the terms of type A (i.e., $\Gamma \vdash^{\Theta}_{\mathcal{G}} M : A : \mathcal{S} \mid \Delta$) are in equational correspondence with terms of type B (i.e., $\Gamma \vdash^{\Theta}_{\mathcal{G}} N : B : \mathcal{S} \mid \Delta$).*

Proof. To establish an equational correspondence, we need to find maps between terms of the isomorphic types A and B. Since the same language serves as both the source and the target, we will use a pair of appropriate contexts based on the types A and B that are derived from the given witnesses to the isomorphism.

Suppose that $x : A : \mathcal{S} \vdash^{\Theta}_{\mathcal{G}} P' : B : \mathcal{S}$ and $y : B : \mathcal{S} \vdash^{\Theta}_{\mathcal{G}} P : A : \mathcal{S}$ witnesses the isomorphism $\Theta \vdash_{\mathcal{G}} A \approx B : \mathcal{S}$, where $x, y \notin \Gamma$. The desired contexts are then $C = \mathbf{let}\, x{:}A = \square\, \mathbf{in}\, P'$ for converting from A to B and $C' = \mathbf{let}\, y{:}B = \square\, \mathbf{in}\, P$ for converting back. To show that these context mappings give an equational correspondence, we must show that (1) the mappings preserve equality, and (2) both compositions of the mappings are an identity. The first fact follows immediately from compatibility; if two terms are equal, then they are equal in any context. The second fact is derived from the definition of type isomorphisms at base kinds, and the $\chi^{\mathcal{S}}$ and η_{let} axioms. The composition for $M : A$ is:

$$\begin{aligned}
\Gamma \vdash^{\Theta}_{\mathcal{G}} C'[C[M]] &= (\mathbf{let}\, y{:}B = (\mathbf{let}\, x{:}A = M\, \mathbf{in}\, P')\, \mathbf{in}\, P) \\
&=_{\chi^{\mathcal{S}}} (\mathbf{let}\, x{:}A = M\, \mathbf{in}\, (\mathbf{let}\, y{:}B = P'\, \mathbf{in}\, P)) \\
&=_{iso} (\mathbf{let}\, x{:}A = M\, \mathbf{in}\, x) =_{\eta_{let}} M : A : \mathcal{S} \mid \Delta
\end{aligned}$$

The reverse composition is also equal to the identity: $\Gamma \vdash^{\Theta}_{\mathcal{G}} C[C'[N]] = N : B : \mathcal{S} \mid \Delta$. ◀

▶ **Lemma 26.** *Type isomorphisms are*

a) reflexive: *if* $\Theta \vdash_{\mathcal{G}} A =_{\beta\eta} B : k$ *then* $\Theta \vDash_{\mathcal{G}} A \approx B : k$*,*
b) symmetric: *if* $\Theta \vDash_{\mathcal{G}} A \approx B : k$ *then* $\Theta \vDash_{\mathcal{G}} B \approx A : k$*, and*
c) transitive: *if* $\Theta \vDash_{\mathcal{G}} A \approx B : k$ *and* $\Theta \vDash_{\mathcal{G}} B \approx C : k$ *then* $\Theta \vDash_{\mathcal{G}} A \approx C : k$*.*

Proof. By induction on the kind k. Symmetry and transitivity of higher kinds $k \to l$ follows immediately from the inductive hypothesis. For the base case $\mathcal{S}$, note that symmetry follows directly from the definition of type isomorphism and reflexivity is witnessed by the identities $\langle x \| \alpha \rangle : (x : A \vdash^{\Theta}_{\mathcal{G}} \alpha : B)$ and $\langle x \| \alpha \rangle : (x : B \vdash^{\Theta}_{\mathcal{G}} \alpha : B)$ typable by the *TCR* and *TCL* rules, which is equal to its own self-composition by the η_μ and $\eta_{\tilde{\mu}}$ axioms. The only non-trivial property is transitivity for the base case $\mathcal{S}$, so suppose that we have the isomorphisms $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$ and $\Theta \vDash_{\mathcal{G}} B \approx C : \mathcal{S}$ as witnessed by

$$\begin{aligned}
&c_1 : (x_1 : A \vdash^{\Theta}_{\mathcal{G}} \beta_1 : B) && c'_1 : (y_1 : B \vdash^{\Theta}_{\mathcal{G}} \alpha_1 : A) \\
&c_2 : (y_2 : B \vdash^{\Theta}_{\mathcal{G}} \gamma_2 : C) && c'_2 : (z_2 : C \vdash^{\Theta}_{\mathcal{G}} \beta_2 : B)
\end{aligned}$$

The isomorphism $\Theta \vDash_{\mathcal{G}} A \approx C : \mathcal{S}$ is then witnessed by

$$\begin{aligned}
&\langle \mu\beta_1{:}B.c_1 \| \tilde{\mu}y_2{:}B.c_2 \rangle : (x_1 : A \vdash^{\Theta}_{\mathcal{G}} \gamma_2 : C) \\
&\langle \mu\beta_2{:}B.c'_2 \| \tilde{\mu}y_1{:}B.c'_1 \rangle : (z_2 : C \vdash^{\Theta}_{\mathcal{G}} \alpha_1 : A)
\end{aligned}$$

Note that by the definition of type isomorphism, it must be that $\Theta \vdash_{\mathcal{G}} A : \mathcal{S}$, $\Theta \vdash_{\mathcal{G}} B : \mathcal{S}$, and $\Theta \vdash_{\mathcal{G}} C : \mathcal{S}$. Both compositions of these two commands are equal to identities via the $\chi^{\mathcal{S}}$ law, which is an axiom for $\mathcal{S} = \star$ and $\mathcal{S} = \star$ and derivable from β^+_μ and $\beta^-_{\tilde{\mu}}$ for $\mathcal{S} = +$ and $\mathcal{S} = -$, respectively. In the one direction, we have:

$$\begin{aligned}
&\langle \mu\gamma_2{:}C.\langle \mu\beta_1{:}B.c_1 \| \tilde{\mu}y_2{:}B.c_2 \rangle \| \tilde{\mu}z_2{:}C.\langle \mu\beta_2{:}B.c'_2 \| \tilde{\mu}y_1{:}B.c'_1 \rangle \rangle \\
&=_{\chi^{\mathcal{S}}} \langle \mu\beta_1{:}B.c_1 \| \tilde{\mu}y_2{:}B.\langle \mu\gamma_2{:}C.c_2 \| \tilde{\mu}z_2{:}C.\langle \mu\beta_2{:}B.c'_2 \| \tilde{\mu}y_1{:}B.c'_1 \rangle \rangle \rangle \\
&=_{\chi^{\mathcal{S}}} \langle \mu\beta_1{:}B.c_1 \| \tilde{\mu}y_2{:}B.\langle \mu\beta_2{:}B.\langle \mu\gamma_2{:}C.c_2 \| \tilde{\mu}z_2{:}C.c'_2 \rangle \| \tilde{\mu}y_1{:}B.c'_1 \rangle \rangle \\
&=_{iso} \langle \mu\beta_1{:}B.c_1 \| \tilde{\mu}y_2{:}B.\langle \mu\beta_2{:}B.\langle y_2 \| \beta_2 \rangle \| \tilde{\mu}y_1{:}B.c'_1 \rangle \rangle \\
&=_{\eta_\mu \eta_{\tilde{\mu}}} \langle \mu\beta_1{:}B.c_1 \| \tilde{\mu}y_1{:}B.c'_1 \rangle \\
&=_{iso} \langle x_1 \| \alpha_1 \rangle \\
&: x_1 : A \vdash^{\Theta}_{\mathcal{G}} \alpha_1 : A
\end{aligned}$$

The other direction of composition is equal to the identity $\langle z_2 \| \gamma_2 \rangle$ analogously. ◀

▶ **Lemma 27** (Connective isomorphism compatibility). *For each (co-)data type constructor* $\mathsf{F} : k_i \to \overset{i}{\ldots} l \in \mathcal{D}$, $\Theta \vDash_{\mathcal{G}} \mathsf{F}\, A_i \overset{i}{\ldots} \approx \mathsf{F}\, B_i \overset{i}{\ldots} : l$ *for all* $\mathcal{G}$ *extending* $\mathcal{D}$ *and* $\Theta \vDash_{\mathcal{G}} A_i \approx B_i : k_i$. *Likewise for* $\mathcal{F}$ *in place of* $\mathcal{D}$.

Proof. The most interesting cases of compatibility are for the shifts (since they are the only constructors to involve non-call-by-value and -name evaluation) and quantifiers (since they are higher-order), and the remaining cases for $\oplus$, $\otimes$, 0, 1, &, ⅋, $\top$, $\bot$, and $\to$ are analogous to these (the details of which are shown in [6]).

For the shift data type ${}_{\mathcal{S}}\!\Uparrow : + \to \mathcal{S}$, assume that an isomorphism $\Theta \vDash_{\mathcal{G}} A \approx B : +$ as witnessed by

$$c : (x : A \vdash^{\Theta}_{\mathcal{G}} \beta : B) \qquad\qquad c' : (y : B \vdash^{\Theta}_{\mathcal{G}} \alpha : A)$$

The isomorphism $\Theta \vDash_{\mathcal{G}} {}_{\mathcal{S}}\!\Uparrow A \approx {}_{\mathcal{S}}\!\Uparrow B : \mathcal{S}$ is then witnessed by

$$\langle x' \| \tilde{\lambda}\{\mathsf{val}_{\mathcal{S}}(x{:}A).\langle \mathsf{val}_{\mathcal{S}}(\mu\beta{:}B.c) \| \beta' \rangle\}\rangle : (x'{:}\,{}_{\mathcal{S}}\!\Uparrow A \vdash^{\Theta}_{\mathcal{G}} \beta'{:}\,{}_{\mathcal{S}}\!\Uparrow B)$$
$$\langle y' \| \tilde{\lambda}\{\mathsf{val}_{\mathcal{S}}(y{:}B).\langle \mathsf{val}_{\mathcal{S}}(\mu\alpha{:}A.c') \| \alpha' \rangle\}\rangle : (y'{:}\,{}_{\mathcal{S}}\!\Uparrow B \vdash_{\mathcal{G}} \alpha{:}\,{}_{\mathcal{S}}\!\Uparrow A)$$

The shift data type $\downarrow_{\mathcal{S}} : \mathcal{S} \to +$ is analogous to the above; assume an isomorphism $\Theta \vDash_{\mathcal{G}} A \approx B : \mathcal{S}$ as witnessed by

$$c : (x : A \vdash^{\Theta}_{\mathcal{G}} \beta : B) \qquad\qquad c' : (y : B \vdash^{\Theta}_{\mathcal{G}} \alpha : A)$$

The isomorphism $\Theta \vDash_{\mathcal{G}} {}_{\mathcal{S}}\!\Uparrow A \approx {}_{\mathcal{S}}\!\Uparrow B : +$ is then witnessed by

$$\langle x' \| \tilde{\lambda}\{\mathsf{box}_{\mathcal{S}}(x{:}A).\langle \mathsf{box}_{\mathcal{S}}(\mu\beta{:}B.c) \| \beta' \rangle\}\rangle : (x'{:}\downarrow_{\mathcal{S}} A \vdash^{\Theta}_{\mathcal{G}} \beta'{:}\downarrow_{\mathcal{S}} B)$$
$$\langle y' \| \tilde{\lambda}\{\mathsf{box}_{\mathcal{S}}(y{:}B).\langle \mathsf{box}_{\mathcal{S}}(\mu\alpha{:}A.c') \| \alpha' \rangle\}\rangle : (y'{:}\downarrow_{\mathcal{S}} B \vdash^{\Theta}_{\mathcal{G}} \alpha{:}\downarrow_{\mathcal{S}} A)$$

Where the calculations showing that both compositions are equal to an identity are analogous to the ones for ${}_{\mathcal{S}}\!\Uparrow$.

For the existential quantifier data type $\exists_k : (k \to +) \to CBV$, assume that we have an isomorphism $\Theta \vDash_{\mathcal{G}} A \approx B : k \to +$, which is definitionally the same as an isomorphism $\Theta, Z : k \vDash_{\mathcal{G}} A\ Z \approx B\ Z : +$, is witnessed by

$$c : (x : A\ Z \vdash^{\Theta, Z:k}_{\mathcal{G}} \beta : B\ Z) \qquad\qquad c' : (y : B\ Z \vdash^{\Theta, Z:k}_{\mathcal{G}} \alpha : A\ Z)$$

The isomorphism $\Theta \vDash_{\mathcal{G}} \exists_k A \approx \exists_k B : +$ is then witnessed by

$$\langle x' \| \tilde{\lambda}\{\mathsf{pack}(Z{:}k)(x{:}A\ Z).\langle \mathsf{pack}\, Z\ (\mu\beta{:}B\ Z.c) \| \beta' \rangle\}\rangle : (x' : \exists_k A \vdash^{\Theta}_{\mathcal{G}} \beta' : \exists_k B)$$
$$\langle y' \| \tilde{\lambda}\{\mathsf{pack}(Z{:}k)(y{:}B\ Z).\langle \mathsf{pack}\, Z\ (\mu\alpha{:}A\ Z.c') \| \alpha' \rangle\}\rangle : (y' : \exists_k B \vdash^{\Theta}_{\mathcal{G}} \alpha' : \exists_k A)$$

The compatibility of the type constructors ${}_{\mathcal{S}}\!\Downarrow : - \to \mathcal{S}$, $\uparrow_{\mathcal{S}} : \mathcal{S} \to -$, and $\forall_k : (k \to -) \to -$ follow from the above three cases by duality. ◀

▶ **Lemma 28** (Isomorphism Substitution). *If* $\Theta, X : k \vdash_{\mathcal{D}} A : l$ *and* $\Theta \vDash_{\mathcal{D}} B \approx C : k$ *then* $\Theta \vdash_{\mathcal{D}} A[B/X] \approx A[C/X] : l$. *Likewise for* $\mathcal{F}$ *in place of* $\mathcal{D}$.

Proof. Follows from the compatibility of the $\mathcal{D}$ and $\mathcal{F}$ connectives (Theorem 27) by (first) induction on the kind k, (second) induction on the kind l and (third) induction on the derivation of $\Theta, X : k \vdash_{\mathcal{D}} A : l$. ◀

▶ **Definition 29** (Declaration isomorphism). Two data type declarations are *isomorphic* with respect to a global environment $\mathcal{G}$ containing them, written

$$\begin{aligned}&\vDash_{\mathcal{G}} \mathbf{data}\,\mathsf{F}(X{:}k..) : \mathcal{S}\,\mathbf{where}\,\mathsf{K} : (A : \mathcal{T}.. \vdash^{\Theta} \mathsf{F}\,X.. \mid B : \mathcal{R}..)..\\ &\approx \mathbf{data}\,\mathsf{F}'(X{:}k..) : \mathcal{S}\,\mathbf{where}\,\mathsf{K}' : (A : \mathcal{T}.. \vdash^{\Theta} \mathsf{F}'\,X.. \mid B : \mathcal{R}..)..\end{aligned}$$

when $\vDash_{\mathcal{G}} \mathsf{F} \approx \mathsf{F}' : k \to ..\mathcal{S}$. Dually, two co-data type declarations are *isomorphic* with respect to a global environment $\mathcal{G}$ containing them, written

$$\begin{aligned}&\vDash_{\mathcal{G}} \mathbf{codata}\,\mathsf{G}(X{:}k..) : \mathcal{S}\,\mathbf{where}\,\mathsf{O} : (A : \mathcal{T}.. \mid \mathsf{G}\,X.. \vdash^{\Theta} B : \mathcal{R}..)..\\ &\approx \mathbf{codata}\,\mathsf{G}'(X{:}k..) : \mathcal{S}\,\mathbf{where}\,\mathsf{O}' : (A : \mathcal{T}.. \mid \mathsf{G}'\,X.. \vdash^{\Theta} B : \mathcal{R}..)..\end{aligned}$$

when $\vDash_{\mathcal{G}} \mathsf{G} \approx \mathsf{G}' : k \to ..\mathcal{S}$.

▶ **Lemma 30.** *For any $\vdash \mathcal{G}$ and a declared connective $\mathsf{F} : k \in \mathcal{G}$, then $\vDash_{\mathcal{G}} \mathsf{F} \approx [\![\mathsf{F}]\!]^{\mathcal{D}}_{\mathcal{G}}$.*

Proof. Analogous to Theorem 8.9 of [6], by extending the structural laws of Figures 8.4 and 8.5 with quantified type variables and with the following laws for the $\forall_k$ and $\exists_k$ quantifiers:

$$\vDash_{\mathcal{G}} \begin{array}{l}\mathbf{data}\,\mathsf{F}(\Theta) : +\,\mathbf{where}\\ \quad \mathsf{K} : (A : + \vdash^{Y:l} \mathsf{F}(\Theta) \mid\,)\end{array} \approx_{\exists L} \begin{array}{l}\mathbf{data}\,\mathsf{F}(\Theta) : +\,\mathbf{where}\\ \quad \mathsf{K} : (\exists Y{:}l.A : + \vdash \mathsf{F}(\Theta) \mid\,)\end{array}$$

$$\vDash_{\mathcal{G}} \begin{array}{l}\mathbf{codata}\,\mathsf{G}(\Theta) : +\,\mathbf{where}\\ \quad \mathsf{O} : (\,\mid \mathsf{G}(\Theta) \vdash^{Y:l} A : -)\end{array} \approx_{\exists L} \begin{array}{l}\mathbf{codata}\,\mathsf{G}(\Theta) : +\,\mathbf{where}\\ \quad \mathsf{O} : (\,\mid \mathsf{G}(\Theta) \vdash \forall Y{:}l.A : -)\end{array}$$

◀

▶ **Theorem 31.** *For all $\vdash \mathcal{G}$ extending $\mathcal{D}$ and $\Theta \vdash_{\mathcal{G}} A : k$,*
$\Theta \vDash_{\mathcal{G}} A = [\![A]\!]^{\mathcal{D}}_{\mathcal{G}} : k$

Proof. By induction on the derivation of $\Theta \vdash_{\mathcal{G}} A : k$, using reflexivity (Theorem 26) in the case of type variables and type function abstraction, substitution (Theorem 28) in the case of type application, and the encoding (Theorem 30) in the case of connectives. ◀

▶ **Theorem 4.** *For all $\vdash \mathcal{G}$ extending $\mathcal{F}$ and $\Theta \vdash_{\mathcal{G}} A : k$, $\Theta \vDash_{\mathcal{G}} A \approx [\![A]\!]^{\mathcal{F}}_{\mathcal{G}} : k$.*

Proof. Analogous to Theorem 31, noting that the witnesses to isomorphism lie in the functional sub-calculus of $\mathcal{D}$. ◀

H.2 Global encoding

The particular set of nested patterns and co-patterns used for the dual encodings is encompassed by the following inductive definition:

$$\begin{aligned}
p^{\mathcal{D}} &::= \mathsf{val}_{\mathcal{S}}\, p^{\mathcal{D}}_{\oplus} & q^{\mathcal{D}} &::= \mathsf{enter}_{\mathcal{S}}\, q^{\mathcal{D}}_{\&}\\
p^{\mathcal{D}}_{\oplus} &::= \iota_i p^{\mathcal{D}}_{\oplus} \mid p^{\mathcal{D}}_{\exists} & q^{\mathcal{D}}_{\&} &::= \pi_i q^{\mathcal{D}}_{\&} \mid q^{\mathcal{D}}_{\forall}\\
p^{\mathcal{D}}_{\exists} &::= \mathsf{pack}\,\boldsymbol{Y}\; p^{\mathcal{D}}_{\exists} \mid p^{\mathcal{D}}_{\otimes} & q^{\mathcal{D}}_{\forall} &::= \mathsf{spec}\,\boldsymbol{Y}\; q^{\mathcal{D}}_{\forall} \mid q^{\mathcal{D}}_{⅋}\\
p^{\mathcal{D}}_{\otimes} &::= (\mathsf{cont}\, q^{\mathcal{D}}_{\uparrow}, p^{\mathcal{D}}_{\otimes}) \mid (p^{\mathcal{D}}_{\downarrow}, p^{\mathcal{D}}_{\otimes}) \mid () & q^{\mathcal{D}}_{⅋} &::= [\mathsf{throw}\, p^{\mathcal{D}}_{\downarrow}, q^{\mathcal{D}}_{⅋}] \mid [q^{\mathcal{D}}_{\uparrow}, q^{\mathcal{D}}_{⅋}] \mid []\\
p^{\mathcal{D}}_{\downarrow} &::= \mathsf{box}_{\mathcal{S}}\, \boldsymbol{x} & q^{\mathcal{D}}_{\uparrow} &::= \mathsf{eval}_{\mathcal{S}}\, \boldsymbol{\alpha}
\end{aligned}$$

The particular set of nested patterns and co-patterns used for the functional encodings is encompassed by the following inductive definition:

$$p^{\mathcal{F}} ::= \mathsf{val}_{\mathcal{S}}\, p^{\mathcal{F}}_{\oplus} \qquad\qquad q^{\mathcal{F}} ::= \mathsf{enter}_{\mathcal{S}}\, .q^{\mathcal{F}}_{\&}$$

$$
\begin{array}{llll}
p^{\mathcal{F}}_{\oplus} ::= \iota_i p^{\mathcal{F}}_{\oplus} \mid p^{\mathcal{F}}_{\exists} & & q^{\mathcal{F}}_{\&} ::= \pi_i . q^{\mathcal{F}}_{\&} \mid q^{\mathcal{F}}_{\forall} \\
p^{\mathcal{F}}_{\exists} ::= \mathsf{pack}\,\boldsymbol{Y}\ p^{\mathcal{F}}_{\exists} \mid p^{\mathcal{F}}_{\otimes} & & q^{\mathcal{F}}_{\forall} ::= \mathsf{spec}\,\boldsymbol{Y} . q^{\mathcal{F}}_{\forall} \mid q^{\mathcal{F}}_{\to} \\
p^{\mathcal{F}}_{\otimes} ::= (p^{\mathcal{F}}_{\downarrow}, p^{\mathcal{F}}_{\otimes}) \mid () & & q^{\mathcal{F}}_{\to} ::= \mathsf{call}(p^{\mathcal{F}}_{\downarrow}) . q^{\mathcal{F}}_{\to} \mid q^{\mathcal{F}}_{\uparrow} \\
p^{\mathcal{F}}_{\downarrow} ::= \mathsf{box}_{\mathcal{S}}\, \boldsymbol{x} & & q^{\mathcal{F}}_{\uparrow} ::= \mathsf{eval}_{\mathcal{S}}
\end{array}
$$

▶ **Lemma 32.** *For any $\mathcal{G}$ extending $\mathcal{D}$, the following two standard reductions*

$$
\begin{array}{lll}
(\beta_p) & \langle p^{\mathcal{D}}[\rho] \| \tilde{\lambda}\{p_i . c_i \overset{i}{\ldots}\} \rangle \mapsto c_i[\rho] & (p^{\mathcal{D}} = p_i) \\
(\beta_q) & \langle \lambda\{q_i . c_i \overset{i}{\ldots}\} \| q^{\mathcal{D}}[\rho] \rangle \mapsto c_i[\rho] & (q^{\mathcal{D}} = q_i)
\end{array}
$$

and the following two equations

$$
\frac{\Gamma \mid \tilde{\lambda}\{p^{\mathcal{D}}_i . \langle p^{\mathcal{D}}_i \| \alpha \rangle \overset{i}{\ldots}\} : A \vdash^{\ominus}_{\mathcal{G}} \alpha : A, \Delta}{\Gamma \mid \tilde{\lambda}\{p^{\mathcal{D}}_i . \langle p^{\mathcal{D}}_i \| \alpha \rangle \overset{i}{\ldots}\} = \alpha : A \vdash^{\ominus}_{\mathcal{G}} \alpha : A, \Delta}\ \eta_p
\qquad
\frac{\Gamma, x : A \vdash^{\ominus}_{\mathcal{G}} \lambda\{q^{\mathcal{D}}_i . \langle x \| q^{\mathcal{D}}_i \rangle \overset{i}{\ldots}\} : A \mid \Delta}{\Gamma, x : A \vdash^{\ominus}_{\mathcal{G}} \lambda\{q^{\mathcal{D}}_i . \langle x \| q^{\mathcal{D}}_i \rangle \overset{i}{\ldots}\} = x : A \mid \Delta}\ \eta_q
$$

are derivable, and analogously for $p^{\mathcal{F}}$ and $q^{\mathcal{F}}$ in place of $p^{\mathcal{D}}$ and $q^{\mathcal{D}}$, respectively.

Proof. Analogous to the proof of the derived $\beta\eta$ axioms for nested (co-)patterns in Theorem 8.1 of [6]. ◀

▶ **Theorem 33.** *If $\vdash \mathcal{G}$ extends $\mathcal{D}$ and $c = c' : (\Gamma \vdash^{\ominus}_{\mathcal{G}} \Delta)$ then $[\![c]\!]^{\mathcal{D}}_{\mathcal{G}} = [\![c']\!]^{\mathcal{D}}_{\mathcal{G}} : ([\![\Gamma]\!]^{\mathcal{D}}_{\mathcal{G}} \vdash^{\ominus}_{\mathcal{D}} [\![\Delta]\!]^{\mathcal{D}}_{\mathcal{G}})$.*

Proof. Observe that patterns are encoded into $p^{\mathcal{D}}$ and co-patterns are encoded into $q^{\mathcal{D}}$. Therefore, the β and η axioms of (co-)data types are translated into the nested forms in Theorem 32, which are derivable. Since these are the only part of the syntax that is changed by the encoding, and equality is compatible, every equation is preserved by the encoding. ◀

▶ **Theorem 6.** *If the global environment $\vdash \mathcal{G}$ extends $\mathcal{F}$ and $\Gamma \vdash^{\ominus}_{\mathcal{G}} M = N : A \mid \Delta$ then $[\![\Gamma]\!]^{\mathcal{F}}_{\mathcal{G}} \vdash^{\ominus}_{\mathcal{F}} [\![M]\!]^{\mathcal{F}}_{\mathcal{G}} = [\![N]\!]^{\mathcal{F}}_{\mathcal{G}} : [\![A]\!]^{\mathcal{F}}_{\mathcal{G}} \mid [\![\Delta]\!]^{\mathcal{F}}_{\mathcal{G}}$.*

Proof. Analogous to Theorem 33 and the fact that the type isomorphisms of $\mathcal{F}$ and the functional sub-calculus of $\mathcal{D}$ are in one-to-one correspondence (Theorem 25). ◀

I Coherence and rewriting theory

Here, we show the coherence of equality via some standard properties of the untyped rewriting theory. Let $\succ$ stand for the non-compatible reduction, that is, one of the rewriting rules used to define the untyped $\mapsto$ and $\to$ relations applied exactly *as-is* without any additional closure properties. We write $\succcurlyeq$ for the reflexive closure of $\succ$, and $\gg$ for the reflexive-transitive closure of $\succ$.

I.1 Confluence

Recall that confluence of a rewriting relation $\to_R$ (whose reflexive-transitive closure is denoted by $\twoheadrightarrow_R$) between expressions (here denoted by M) means that for every divergent pair of reductions $M_1 \twoheadleftarrow_R M \twoheadrightarrow_R M_2$ there is a common point of convergence $M_1 \twoheadrightarrow_R M' \twoheadleftarrow_R M_2$. Additionally, strong normalization of $\to_R$ means that there are no infinite reduction sequences $M_1 \to_R M_2 \to_R M_2 \to_R \dots$. For our purposes here, we can disregard the $\phi^{\star}_{\mu}$ and $\phi^{\star}_{\tilde{\mu}}$ rules since they are subsumed by the strictly more general $\beta^{\star}_{\mu}$ and $\beta^{\star}_{\tilde{\mu}}$ rules. Confluence for the multi-discipline sequent calculus then follows by standard methods for a sub-calculus—everything besides ς reduction—which we then extend to the full calculus using the adjunction-based technique from [10].

▶ **Lemma 34.** *For any global environment $\mathcal{G}$, $\beta_\mu\beta_{\tilde{\mu}}\eta_\mu\eta_{\tilde{\mu}}\beta\delta$ reduction is confluent.*

Proof. Note that because values and co-values are closed under reduction (Property E.1), the only critical pairs (between $\beta_\mu^{\mathcal{S}}$ and $\eta_\mu^{\mathcal{S}}$, $\beta_{\tilde{\mu}}^{\mathcal{S}}$ and $\eta_{\tilde{\mu}}^{\mathcal{S}}$, $\beta_\mu^{\star}$ and $\delta^{\star}$, and $\beta_{\tilde{\mu}}^{\star}$ and $\delta^{\star}$) converge in 0 steps (that is to say, the reduct of each side of the critical pair is identical up to α-equivalence). This means that $\beta_\mu\beta_{\tilde{\mu}}\eta_\mu\eta_{\tilde{\mu}}\beta\delta$ reduction forms a combinatorial orthogonal rewriting system, and is therefore confluent. ◀

Unlike the other operational rules, the ς rules can be exhaustively completed ahead of time as a compilation "pre-processor" into a smaller language without ς. The *focused sub-syntax* is exactly the ς-normal forms; that is to say, the applications of constructors and observers are restricted to (co-)values, as in $\mathsf{K}(E..,V..)$ and $\mathsf{O}[V..,E..]$. Furthermore, note that this sub-syntax is closed under reduction, which means that is also outlines the *focused sub-calculus* wherein the ς rules are no longer necessary.

We now give a transformation on commands and (co-)terms into the focused sub-syntax, denoted by $(\_)^\varsigma$, that performs a particular ς-normalization. The $(\_)^\varsigma$ transformation is defined homomorphically on all syntactic forms except for data structures and co-data observations which are defined by induction as follows (where we omit writing the discipline annotations that can be inferred from constructor and observation names based on the global environment):

$$\begin{aligned}
(\mathsf{K}\,B..E..V..)^\varsigma &\triangleq \mathsf{K}\,B..E^\varsigma..V^\varsigma..\\
(\mathsf{K}\,B..E..V..\ v'\ v..)^\varsigma &\triangleq \mu\alpha.\langle v'^\varsigma \| \tilde{\mu}y.\langle(\mathsf{K}\,B..E..V..\ y\ v'..)^\varsigma \| \alpha\rangle\rangle\\
(\mathsf{K}\,B..E..\ e'\ e..v..)^\varsigma &\triangleq \mu\alpha.\langle\mu\beta.\langle(\mathsf{K}\,B..E..\ \beta\ e..v..)^\varsigma \| \alpha\rangle \| e'^\varsigma\rangle\\
[\mathsf{O}\,B..V..E..]^\varsigma &\triangleq \mathsf{O}\,B..V^\varsigma..E^\varsigma..\\
[\mathsf{O}\,B..V..E..\ e'\ e..]^\varsigma &\triangleq \tilde{\mu}x.\langle\mu\beta.\langle x \| [\mathsf{O}\,B..V..E..\ \beta\ e..]^\varsigma\rangle \| e'^\varsigma\rangle\\
[\mathsf{O}\,B..V..\ v'\ v..e..]^\varsigma &\triangleq \tilde{\mu}x.\langle v'^\varsigma \| \tilde{\mu}y.\langle x \| [\mathsf{O}\,V..\ y\ v..e..]^\varsigma\rangle\rangle
\end{aligned}$$

This transformation, when coupled with plain syntactic inclusion, induces an adjunction (more specifically, a reflection) in the style of [23] between the full calculus and the sub-calculus of ς-normal forms.

▶ **Lemma 35.** *ς transformation and inclusion forms a reflection of the full multi-discipline sequent calculus inside its sub-calculus of ς-normal forms. That is to say, the following four properties hold:*

a) For every command c, $c \twoheadrightarrow_{R\varsigma} c^{\mathcal{S}}$.
b) For every ς-normal form c, $c^{\mathcal{S}} =_\alpha c$.
c) If $c \twoheadrightarrow_R c'$ and c, c' are ς-normal forms, then $c \twoheadrightarrow_{R\varsigma} c'$.
d) If $c \twoheadrightarrow_{R\varsigma} c'$ then $c^{\mathcal{S}} \twoheadrightarrow_R c'^\varsigma$.

and similarly for terms and co-terms, where R stands for the $\beta_\mu\beta_{\tilde{\mu}}\eta_\mu\eta_{\tilde{\mu}}\beta\delta$ rules.

Proof. a) This property says that the $(\_)^\varsigma$ transformation performs a ς-normalization (actually, any $R\varsigma$ reduction would be fine, but it happens that only ς-normalization is needed), which follows by mutual induction on the syntax of commands and (co-)terms.
b) This property says that $(\_)^\varsigma$ is an identity function (up to α-equivalence) on ς-normal forms, which again follows by induction on syntax.
c) This property says that reductions on ς-normal forms are included in the full calculus, which is immediate.

d) This is the main property which allows us to transport all reductions from the full calculus into the sub-calculus of ς-normal forms. The cases for applications of a single specific rule follows from the fact that (co-)values are closed under reduction (Property E.1) which implies that V^ς is a value and E^ς is a co-value, and that $(\_)^\varsigma$ distributes over substitution (because it is compositional):

- $(\beta_\mu^{\mathcal{S}})$ $\langle \mu\alpha{:}\mathcal{S}.c \| E_{\mathcal{S}}\rangle \to c[E_{\mathcal{S}}/\alpha{:}\mathcal{S}]$ becomes:

$$\langle \mu\alpha{:}\mathcal{S}.c \| E_{\mathcal{S}}\rangle^\varsigma =_\alpha \langle \mu\alpha{:}\mathcal{S}.(c^\varsigma) \| E_{\mathcal{S}}^\varsigma\rangle \to_{\beta_\mu^{\mathcal{S}}} c^\varsigma[E_{\mathcal{S}}^\varsigma/\alpha{:}\mathcal{S}] =_\alpha (c[E_{\mathcal{S}}/\alpha{:}\mathcal{S}])^\varsigma$$

- $(\eta_\mu^{\mathcal{S}})$ $\mu\alpha{:}\mathcal{S}.\langle v_{\mathcal{S}} \| \alpha{:}\mathcal{S}\rangle \to v_{\mathcal{S}}$ becomes:

$$(\mu\alpha{:}\mathcal{S}.\langle v_{\mathcal{S}} \| \alpha{:}\mathcal{S}\rangle)^\varsigma =_\alpha \mu\alpha{:}\mathcal{S}.\langle v_{\mathcal{S}}^\varsigma \| \alpha{:}\mathcal{S}\rangle \to_{\eta_\mu^{\mathcal{S}}} v_{\mathcal{S}}^\varsigma$$

- (β_{F}) for a data type F in $\mathcal{G}$

$$\begin{aligned}&\langle \mathsf{K}_i\, B..E_{\mathcal{R}_i}..V_{\mathcal{T}_i}..) \| \tilde\lambda\{\overrightarrow{(\mathsf{K}_i\, \boldsymbol{Y}..\alpha{:}\mathcal{R}_i..x{:}\mathcal{T}_i..).c_i}^{i}\}\rangle \\ &\to c_i[B/\boldsymbol{Y}.., E_{\mathcal{R}_i}/\alpha{:}\mathcal{R}_i.., V_{\mathcal{T}_i}/x{:}\mathcal{T}_i..]\end{aligned}$$

becomes:

$$\begin{aligned}&\langle \mathsf{K}_i\, B..E_{\mathcal{R}_i}..V_{\mathcal{T}_i}.. \| \tilde\lambda\{\overrightarrow{(\mathsf{K}_i\, \boldsymbol{Y}..\alpha{:}\mathcal{R}_i..x{:}\mathcal{T}_i..).c_i}^{i}\}\rangle^\varsigma \\ &=_\alpha \langle \mathsf{K}_i\, B..E_{\mathcal{R}_i}^\varsigma..V_{\mathcal{T}_i}^\varsigma..) \| \tilde\lambda\{\overrightarrow{(\mathsf{K}_i\, \boldsymbol{Y}..\alpha{:}\mathcal{R}_i..x{:}\mathcal{T}_i..).c_i^\varsigma}^{i}\}\rangle \\ &\to_{\beta_{\mathsf{F}}} c_i^\varsigma[B/\boldsymbol{Y}.., E_{\mathcal{R}_i}^\varsigma/\alpha{:}\mathcal{R}_i.., V_{\mathcal{T}_i}^\varsigma/x{:}\mathcal{T}_i..] \\ &=_\alpha (c_i[B/\boldsymbol{Y}.., E_{\mathcal{R}_i}/\alpha{:}\mathcal{R}_i.., V_{\mathcal{T}_i}/x{:}\mathcal{T}_i..])^\varsigma\end{aligned}$$

- (ς_{F}) for a data type F in $\mathcal{G}$

$$P[e_{\mathcal{R}}] \to \mu\alpha{:}\mathcal{S}.\langle \mu\beta{:}\mathcal{R}.\langle P[\beta{:}\mathcal{R}] \| \alpha{:}\mathcal{S}\rangle \| e_{\mathcal{R}}\rangle$$

becomes:

$$P[e_{\mathcal{R}}]^\varsigma =_\alpha \mu\alpha{:}\mathcal{S}.\langle \mu\beta{:}\mathcal{R}.\langle P[\beta{:}\mathcal{R}]^\varsigma \| \alpha{:}\mathcal{S}\rangle \| e_{\mathcal{R}}^\varsigma\rangle =_\alpha \mu\alpha{:}\mathcal{S}.\langle \mu\beta{:}\mathcal{R}.\langle P[\beta{:}\mathcal{R}] \| \alpha{:}\mathcal{S}\rangle \| e_{\mathcal{R}}\rangle^\varsigma$$

- $(\delta^\star)$ $\langle \mu\alpha{:}\star.c \| v_\star\rangle \to c$ because $\alpha{:}\star \notin FV(c)$ becomes:

$$\langle \mu\alpha{:}\star.c \| v_\star\rangle^\varsigma =_\alpha \langle \mu\alpha{:}\star.c^\varsigma \| v_\star^\varsigma\rangle \to c^\varsigma$$

- The cases for $\beta_{\tilde\mu}^{\mathcal{S}}$, $\eta_{\tilde\mu}$, β_{G} for a co-data type G in $\mathcal{G}$, ς_{G} for a co-data type G in $\mathcal{G}$, and $\delta^\star$ are dual to the above cases.

The case for reflexivity and transitivity follows from the reflexivity and transitivity of the sub-calculus. The case for compatibility is a little more subtle since the $(\_)^\varsigma$ can change when a non-(co-)value is turned into a (co-)value by an internal reduction. In this case, the translation of the left-hand side simplifies down to the translation of the right-hand side with the help of $\beta_\mu\beta_{\tilde\mu}\eta_\mu\eta_{\tilde\mu}$ reductions. For example, we could have a reduction of the pair $(v, v') \to (V, v')$ of type $A \otimes B$ that converts the first component to a value when both v and v' are non-values. Under translation, this instance reduces as:

$$\begin{aligned}(v, v')^\varsigma &=_\alpha \mu\alpha.\langle v^\varsigma \| \tilde\mu x.\langle (x, v')^\varsigma \| \alpha\rangle\rangle \\ &\to \mu\alpha.\langle V^\varsigma \| \tilde\mu x.\langle (x, v')^\varsigma \| \alpha\rangle\rangle \\ &\to_{\beta_{\tilde\mu}} \mu\alpha.\langle (x, v')^\varsigma \| \alpha\rangle[V^\varsigma/x]\end{aligned}$$

$$\begin{aligned}&\to_{\eta_\mu} (x, v')^\varsigma[V^\varsigma/x]\\ &=_\alpha ((x, v')^\varsigma[V/x])^\varsigma =_\alpha (V, v')^\varsigma\end{aligned}$$

With the help of this fact as necessary, the case for compatibility follows since $(\_)^\varsigma$ is compositional and the sub-calculus is also compatible. ◀

▶ **Theorem 36.** *For any global environment $\mathcal{G}$, all reduction in the multi-discipline sequent calculus is confluent.*

Proof. Follows from Theorem 34 and Theorem 35, as shown in [10]. In particular, let R stand for all non-ς reduction rules and suppose that $m_1 \twoheadleftarrow m \twoheadrightarrow_{R\varsigma} m_2$. It follows that $m_1^\varsigma \twoheadleftarrow_R m^\varsigma \twoheadrightarrow_R m_2$, so by confluence of R there is an m' such that $m_1 \twoheadrightarrow_{R\varsigma} m_1^\varsigma \twoheadrightarrow_R m' \twoheadleftarrow_R m_2^\varsigma \twoheadleftarrow_{R\varsigma} m_2$. ◀

I.2 Standardization

To show standardization, we will first show how non-standard reductions can be postponed after standard ones. The non-standard reduction relation, written as $\rightarrowtail$, and its reflexive-transitive closure, $\twoheadrightarrowtail$, are defined as

$$\frac{m \to m' \quad m \not\mapsto m'}{m \rightarrowtail m'} \qquad \frac{}{m \twoheadrightarrowtail m} \qquad \frac{m \twoheadrightarrowtail m' \quad m' \twoheadrightarrowtail m''}{m \twoheadrightarrowtail m''}$$

▶ **Lemma 37.** *If $m \rightarrowtail m'$ then the status of m (*i.e.*, finished, stuck, in progress) is the same as m'.*

Proof. First, we just show the forward direction: if $m \rightarrowtail m'$ and m' is finished, stuck, or in progress, then m is final, stuck, or in progress, respectively. Note that finished and stuck commands are closed under reduction in general. Also observe the fact that if $D[m] \rightarrowtail m''$ then either

- $m'' =_\alpha D[m']$ for some m' such that $m \rightarrowtail m'$, or
- $m'' =_\alpha D'[m[\rho]]$ for some D' and substitution ρ,

which follows by induction on D. Now, supposing that $m_2' \leftarrowtail D[m] \mapsto D[m_1]$ because $m \succ m_1$, then we can proceed by cases on the previous fact:

- If $m_2' =_\alpha D[m_2]$ then we have $m_2 \leftarrowtail m \succ m_1$, and it can be checked that $m_2 \mapsto m'$ for some m' by cases on the standard reduction rule used for $m \succ m_1$.
- If $m_2' =_\alpha D'[m[\rho]]$ then $m_2' \mapsto D'[m_1[\rho]]$ since standard reduction is closed under substitution.

The backward direction follows from the previous fact because every expression is *exactly one* of finished, stuck, or in progress. That is to say, if we know the status of m' (say, finished), then it would be contradictory for m to be anything else, since that would imply that m' would have to have a different status. ◀

The main crux of the standardization proof is to commute standard and non-standard reductions to perform the standard ones first, so that given any $m \twoheadrightarrowtail\twoheadmapsto m'$ it is also the case that $m \twoheadmapsto\twoheadrightarrowtail m'$. The complication is that commuting a pair of single steps can lead to duplication—standard reductions can duplicate non-standard reductions in sub-expressions, whereas non-standard reductions might be the *next* standard one leading to many standard steps when order is exchanged—so that in general if $m \rightarrowtail\mapsto m'$ then we could wind up with

$m \mapstochar\twoheadrightarrow\rightarrowtail\!\!\!\!\twoheadrightarrow m'$. So the simple single-step case does not directly generalize to multiple steps by induction on the reflexive-transitive closure of reduction sequences.

To work around this problem, we employ a technique that is sometimes used for mitigating similar complications in confluence proofs: work with a *grand* reduction relation that allows for certain multi-step reduction sequences, but still enables induction on the syntactic structure of expressions. Our grand reductions will come in three different flavors: internal non-standard reductions that preserve the top node of the syntax tree ($\Rightarrow^i$), non-standard reductions that might rewrite the top of the syntax tree ($\Rightarrow$), and a prefix of standard reductions followed by a grand non-standard reduction ($\Mapsto$). These are defined by mutual induction on the syntax of expressions, where $\Mapsto$ and $\Rightarrow$ enjoy a generic definition independent on the syntactic form of the left-hand side, and only $\Rightarrow^i$ follows by cases on syntax (where $\pm\text{☆}$ denotes any $\mathcal{S}$ besides $\star$ and $\pm\star$ denotes any $\mathcal{S}$ besides ☆):

$$\frac{m \mapstochar\twoheadrightarrow m' \quad m' \Rightarrow m''}{m \Mapsto m''} \qquad \frac{m \Rightarrow^i m' \quad m' \succcurlyeq m'' \quad m' \not\mapsto m''}{m \Rightarrow m''}$$

$$\frac{\mathcal{S}{\in}\{+,\text{☆}\} \;\; w_{\mathcal{S}}{\Rightarrow^i}v_{\mathcal{S}} \succcurlyeq v'_{\mathcal{S}} \;\; e_{\mathcal{S}}{\Mapsto}e'_{\mathcal{S}} \;\; \langle v_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle{\not\mapsto}\langle v'_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}{\langle w_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \Rightarrow^i \langle v'_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}$$

$$\frac{\mathcal{S}{\in}\{+,\text{☆}\} \;\; V_{\mathcal{S}}{\Rightarrow^i}V'_{\mathcal{S}}{\succcurlyeq}V''_{\mathcal{S}} \;\; e_{\mathcal{S}}{\Rightarrow^i}e'_{\mathcal{S}} \;\; \langle V'_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle{\not\mapsto}\langle V''_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}{\langle V_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \Rightarrow^i \langle V''_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}$$

$$\frac{\mathcal{S}{\in}\{-,\star\} \;\; v_{\mathcal{S}}{\Mapsto}v'_{\mathcal{S}} \;\; f_{\mathcal{S}}{\Rightarrow^i}e_{\mathcal{S}}{\succcurlyeq}e'_{\mathcal{S}} \;\; \langle v'_{\mathcal{S}} \| e_{\mathcal{S}}\rangle{\not\mapsto}\langle v'_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}{\langle v_{\mathcal{S}} \| f_{\mathcal{S}}\rangle \Rightarrow^i \langle v'_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}$$

$$\frac{\mathcal{S}{\in}\{-,\star\} \;\; v_{\mathcal{S}}{\Rightarrow^i}v'_{\mathcal{S}} \;\; E_{\mathcal{S}}{\Rightarrow^i}E'_{\mathcal{S}}{\succcurlyeq}E'_{\mathcal{S}} \;\; \langle v'_{\mathcal{S}} \| E'_{\mathcal{S}}\rangle{\not\mapsto}\langle v'_{\mathcal{S}} \| E''_{\mathcal{S}}\rangle}{\langle v_{\mathcal{S}} \| E_{\mathcal{S}}\rangle \Rightarrow^i \langle v'_{\mathcal{S}} \| E''_{\mathcal{S}}\rangle}$$

$$\frac{}{x \Rightarrow^i x} \qquad \frac{}{\alpha \Rightarrow^i \alpha}$$

$$\frac{c \Mapsto c'}{\mu\alpha{:}\pm\star.c \Rightarrow^i \mu\alpha{:}\pm\star.c'} \qquad \frac{c \Mapsto c'}{\tilde{\mu}x{:}\pm\text{☆}.c \Rightarrow^i \tilde{\mu}x{:}\pm\text{☆}.c'}$$

$$\frac{c \Rightarrow c'}{\mu\alpha{:}\text{☆}.c \Rightarrow^i \mu\alpha{:}\text{☆}.c'} \qquad \frac{c \Rightarrow c'}{\tilde{\mu}x{:}\star.c \Rightarrow^i \tilde{\mu}x{:}\star.c'}$$

$$\frac{e \Mapsto e'.. \quad v \Mapsto v'..}{\mathsf{K}\,B..e..v.. \Rightarrow^i \mathsf{K}\,B..e'..v'..} \qquad \frac{v \Mapsto v'.. \quad e \Mapsto e'..}{\mathsf{O}\,B..v..e.. \Rightarrow^i \mathsf{O}\,B..v'..e'..}$$

$$\frac{c \Mapsto c'..}{\lambda\{q.c..\} \Rightarrow^i \lambda\{q.c'..\}} \qquad \frac{c \Mapsto c'..}{\tilde{\lambda}\{p.c..\} \Rightarrow^i \tilde{\lambda}\{p.c'..\}}$$

▶ **Lemma 38.** *a)* $m \Rightarrow^i m$*,* $m \Rightarrow m$*, and* $m \Mapsto m$*.*

b) $m \Rightarrow^i m'$ *implies* $m \Rightarrow m'$ *implies* $m \Mapsto m'$*.*

c) $m \rightarrowtail m'$ *implies* $m \Rightarrow m'$*,* $m \mapsto m'$ *implies* $m \Mapsto m'$*, and* $m \to m'$ *implies* $m \Mapsto m'$*.*

d) $m \Rightarrow^i m'$ *implies* $D[m] \rightarrowtail\!\!\!\!\twoheadrightarrow D[m']$ *for any* D*,* $m \Rightarrow m'$ *implies* $m \rightarrowtail\!\!\!\!\twoheadrightarrow m'$*, and* $m \Mapsto m'$ *implies* $m \mapstochar\twoheadrightarrow\rightarrowtail\!\!\!\!\twoheadrightarrow m'$*.*

Proof. a) By induction on the syntax of m.

b) By the definition of $\Rightarrow^i$, $\Rightarrow$, and $\Mapsto$.

c) By induction on the syntax of the left-hand side m, using part (a) as necessary for unchanged sub-expressions.

d) By mutual induction on the derivation of $\Rightarrow^i$, $\Rightarrow$, and $\Mapsto$. ◀

▶ **Lemma 39.** *a) If $m \Mapsto m'$, $x[\rho] \Mapsto x[\rho']$ for all x, and $\alpha[\rho] \Mapsto \alpha[\rho']$ for all α, then $m[\rho] \Mapsto m'[\rho']$.*
b) If $v_{\mathcal{S}} \Mapsto v'_{\mathcal{S}}$ and $e_{\mathcal{S}} \Mapsto e'_{\mathcal{S}}$ then $\langle v_{\mathcal{S}} \| e_{\mathcal{S}} \rangle \Mapsto \langle v'_{\mathcal{S}} \| e'_{\mathcal{S}} \rangle$.

Proof. The conjunction of parts (a) and (b) follow simultaneously by induction on the given derivations of $m \Mapsto m'$, $v_{\mathcal{S}} \Mapsto v'_{\mathcal{S}}$, and $e_{\mathcal{S}} \Mapsto e'_{\mathcal{S}}$ by generalizing (b) to: If $v_{\mathcal{S}} \Mapsto v'_{\mathcal{S}}$, $e_{\mathcal{S}} \Mapsto e'_{\mathcal{S}}$, $x[\rho] \Mapsto x[\rho']$ for all x, and $\alpha[\rho] \Mapsto \alpha[\rho']$ for all α then $\langle v_{\mathcal{S}} \| e_{\mathcal{S}} \rangle[\rho] \Mapsto \langle v'_{\mathcal{S}} \| e'_{\mathcal{S}} \rangle[\rho']$.

For part (a), note that any $V_{\mathcal{S}} \Mapsto V'_{\mathcal{S}}$ and $E_{\mathcal{S}} \Mapsto E'_{\mathcal{S}}$ implies $V_{\mathcal{S}} \Rightarrow V'_{\mathcal{S}}$ and $E_{\mathcal{S}} \Rightarrow E'_{\mathcal{S}}$ because (co-)values do not standard reduce. Further, $x \Rightarrow v$ and $\alpha \Rightarrow e$ implies $x \Rightarrow^i v$ and $\alpha \Rightarrow^i e$ since $x \not\mapsto$ and $\alpha \not\mapsto$. It follows that if $m \Rightarrow m'$ and $\rho \Mapsto \rho'$ (pointwise) then $m[\rho] \Mapsto m'[\rho']$ since the cases where substitution replaces a variable or co-variable cannot cause two top-level reductions. The other interesting cases are for $\Rightarrow^i$ reduction of a command, which relies on part (b) of the inductive hypothesis, and $\Rightarrow^i$ of $\star$ μ-abstractions and $\star$ $\tilde{\mu}$-abstractions, which propagate the initial standard reductions of the underlying command (generated by the inductive hypothesis) down to the abstraction itself.

For part (b), the general idea is to perform the standard reductions of $v_{\mathcal{S}} \Mapsto v'_{\mathcal{S}}$ and $e_{\mathcal{S}} \Mapsto e'_{\mathcal{S}}$ first as appropriate (according to the compatibility of $\mapsto$ depending on $\mathcal{S}$), and then commute the possible trailing non-standard η_μ and $\eta_{\tilde{\mu}}$ reductions which then become standard β_μ or $\beta_{\tilde{\mu}}$ reductions. For example, if we have the derivations $e_+ \Mapsto e'_+$ and

$$\dfrac{\dfrac{\dfrac{c \Mapsto \langle v'_+ \| \alpha{:}+ \rangle}{\mu\alpha{:}+.c \Rightarrow^i \mu\alpha{:}+.\langle v'_+ \| \alpha{:}+ \rangle \succ_{\eta^+_\mu} v'_+}}{v_+ \mathrel{\mapstochar\twoheadrightarrow} \mu\alpha{:}+.c \Rightarrow v'_+}}{v_+ \Mapsto v'_+}$$

then because e_+ is a co-value (as every $+$ term is), for any $\rho \Mapsto \rho'$, by the inductive hypothesis (a) on the sub-derivations $c \Mapsto \langle v'_+ \| \alpha{:}+ \rangle$ and $e_+ \Mapsto e'_+$ we have the derivation

$$\dfrac{\begin{array}{l} \qquad\qquad\qquad\qquad\qquad\qquad e_+ \Mapsto e'_+ \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots\, IH \\ \qquad\qquad c \Mapsto \langle v'_+ \| \alpha{:}+ \rangle \quad e_+[\rho] \Mapsto e'[\rho'] \\ \qquad\qquad\qquad\qquad\quad \vdots\, IH \\ \langle v_+ \| e_+ \rangle[\rho] \mathrel{\mapstochar\twoheadrightarrow} \langle \mu\alpha{:}+.c[\rho] \| e_+[\rho] \rangle \\ \qquad\qquad \mapsto_{\beta^+_\mu} c[e_+[\rho]/\alpha{:}+, \rho] \Mapsto \langle v'_+[\rho'] \| e'_+[\rho'] \rangle \end{array}}{\langle v_+ \| e_+ \rangle[\rho] \Mapsto \langle v'_+ \| e'_+ \rangle[\rho]}$$

The other cases are similar to this one. ◀

▶ **Lemma 40.** *If either $m \Mapsto\mapsto m'$, $m \Rightarrow\mapsto m'$, or $m \Rightarrow^i\mapsto m'$ then $m \Mapsto m'$.*

Proof. By mutual induction on the derivation of $\Mapsto$, $\Rightarrow$, and $\Rightarrow^i$. The first case is for $\Mapsto$ followed by $\mapsto$: if

$$\dfrac{m_1 \mathrel{\mapstochar\twoheadrightarrow} m_2 \quad m_2 \Rightarrow m_3}{m_1 \Mapsto m_3 \mapsto m_4}$$

then the inductive hypothesis on $m_2 \Rightarrow m_3 \mapsto m_4$ gives $m_2 \Mapsto m_4$, which is the same as $m_2 \mathrel{\mapstochar\twoheadrightarrow} m'_3 \Rightarrow m_4$ for some m'_3, so by transitivity of $\mapsto$ we have

$$\dfrac{m_1 \mathrel{\mapstochar\twoheadrightarrow} m_2 \mathrel{\mapstochar\twoheadrightarrow} m'_3 \quad m'_3 \Rightarrow m_4}{m_1 \Mapsto m_4}$$

The second group of cases is for $\Rightarrow$ followed by $\mapsto$, which proceeds by cases on the potential trailing non-standard rewriting rule, which dictates more of the $\Rightarrow$ derivation by inversion when present:

- (*reflexive*): If

$$\frac{m_1 \Rightarrow^i m_2}{m_1 \Rightarrow m_2 \mapsto m_3}$$

then the inductive hypothesis on $m_1 \Rightarrow^i m_2 \mapsto m_3$ gives $m_1 \Mapsto m_3$.

- $(\eta_\mu^{\pm\star})$: If

$$\frac{\dfrac{c \Mapsto \langle v_{\pm\star} \| \alpha{:}\pm\star \rangle}{\mu\alpha{:}\pm\star.c \Rightarrow^i \mu\alpha{:}\pm\star.\langle v_{\pm\star} \| \alpha{:}\pm\star \rangle} \quad \mu\alpha{:}\pm\star.\langle v_{\pm\star} \| \alpha{:}\pm\star \rangle \rightarrowtail_{\eta_\mu^{\pm\star}} v_{\pm\star}}{\mu\alpha{:}\pm\star.c \Rightarrow v_{\pm\star} \mapsto v'_{\pm\star}}$$

then $\langle v_{\pm\star} \| \alpha{:}\pm\star \rangle \mapsto \langle v'_{\pm\star} \| \alpha{:}\pm\star \rangle$ and the inductive hypothesis on $c \Mapsto \langle v_{\pm\star} \| \alpha{:}\pm\star \rangle \mapsto \langle v'_{\pm\star} \| \alpha{:}\pm\star \rangle$ gives $c \Mapsto \langle v'_{\pm\star} \| \alpha{:}\pm\star \rangle$ so we have

$$\frac{\dfrac{\dfrac{c \Mapsto \langle v'_{\pm\star} \| \alpha{:}\pm\star \rangle}{\mu\alpha{:}\pm\star.c \Rightarrow^i \mu\alpha{:}\pm\star.\langle v'_{\pm\star} \| \alpha{:}\pm\star \rangle} \quad \mu\alpha{:}\pm\star.\langle v'_{\pm\star} \| \alpha{:}\pm\star \rangle \rightarrowtail_{\eta_\mu^{\pm\star}} v'_{\pm\star}}{\mu\alpha{:}\pm\star.c \Rightarrow v'_{\pm\star}}}{\mu\alpha{:}\pm\star.c \Mapsto v'_{\pm\star}}$$

- $(\eta_\mu^{\medstar})$: If

$$\frac{\dfrac{c \Rightarrow \langle v_{\medstar} \| \alpha{:}\medstar \rangle}{\mu\alpha{:}\medstar.c \Rightarrow^i \mu\alpha{:}\medstar.\langle v_{\medstar} \| \alpha{:}\medstar \rangle} \quad \mu\alpha{:}\medstar.\langle v_{\medstar} \| \alpha{:}\medstar \rangle \rightarrowtail_{\eta_\mu^{\medstar}} v_{\medstar}}{\mu\alpha{:}\medstar.c \Rightarrow v_{\medstar} \mapsto v'_{\medstar}}$$

then $\langle v_{\medstar} \| \alpha{:}\medstar \rangle \mapsto \langle v'_{\medstar} \| \alpha{:}\medstar \rangle$ and the inductive hypothesis on $c \Mapsto \langle v_{\medstar} \| \alpha{:}\medstar \rangle \mapsto \langle v'_{\medstar} \| \alpha{:}\medstar \rangle$ gives $c \mapsto\!\!\!\twoheadrightarrow c' \Rightarrow \langle v'_{\medstar} \| \alpha{:}\medstar \rangle$ for some c', so we have

$$\frac{\dfrac{\dfrac{c' \Rightarrow \langle v'_{\medstar} \| \alpha{:}\medstar \rangle}{\mu\alpha{:}\medstar.c' \Rightarrow^i \mu\alpha{:}\medstar.\langle v'_{\medstar} \| \alpha{:}\medstar \rangle} \quad \mu\alpha{:}\medstar.\langle v'_{\medstar} \| \alpha{:}\medstar \rangle \rightarrowtail v'_{\medstar}}{\mu\alpha{:}\medstar.c \mapsto\!\!\!\twoheadrightarrow \mu\alpha{:}\medstar.c' \Rightarrow v'_{\medstar}}}{\mu\alpha{:}\medstar.c \Mapsto v'_{\medstar}}$$

- $(\beta_\mu^{\medstar})$: If

$$\frac{\dfrac{\dfrac{c_1 \Rightarrow c_2}{\mu\alpha{:}\medstar.c_1 \Rightarrow^i \mu\alpha{:}\medstar.c_2} \quad e_{\medstar} \Mapsto E_{\medstar}}{\langle \mu\alpha{:}\medstar.c_1 \| e_{\medstar} \rangle \Rightarrow^i \langle \mu\alpha{:}\medstar.c_2 \| E_{\medstar} \rangle \rightarrowtail_{\beta_\mu^{\medstar}} c_2[E_{\medstar}/\alpha{:}\medstar]}}{\langle \mu\alpha{:}\medstar.c_1 \| e_{\medstar} \rangle \Rightarrow c_2[E_{\medstar}/\alpha{:}\medstar] \mapsto c_3}$$

then for the $\beta_\mu^{\medstar}$ reduction to be non-standard and followed by a standard reduction, it must be that α is not needed in c_2, *i.e.,* c_2 is not some $H[\langle V_{\medstar} \| \alpha{:}\medstar \rangle]$. Thus, $c_2[E_{\medstar}/\alpha{:}\medstar]$ is also not some $H[\langle V_{\medstar} \| E_{\medstar} \rangle]$, so the standard reduction does not require the substitution of $E_{\medstar}$, meaning that $c_3 =_\alpha c'_3[E_{\medstar}/\alpha{:}\medstar]$ and $c_2 \mapsto c'_3$ for some c'_3. The inductive hypothesis on $c_1 \Rightarrow c_2 \mapsto c'_3$ gives $c_1 \mapsto\!\!\!\twoheadrightarrow c'_1 \Rightarrow c'_3$, so we can proceed by cases on whether or not $\langle \mu\alpha{:}\medstar.c'_1 \| e_{\medstar} \rangle$ is a value:

- (*value*): From $c_1' \Rightarrow c_3'$ and $E_\star' \Mapsto E_\star$, we have a derivation of $c_1'[E_\star'/\alpha{:}\star] \Mapsto c_3'[E_\star/\alpha{:}\star]$, and

$$\begin{aligned}\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle &\mathrel{\mapstochar\twoheadrightarrow} \langle \mu\alpha{:}\star.c_1' \| e_\star\rangle\\ &\mathrel{\mapstochar\twoheadrightarrow} \langle \mu\alpha{:}\star.c_1' \| E_\star'\rangle\\ &\mapsto_{\phi^\star_\mu} c_1'[E_\star'/\alpha{:}\star]\\ &\Mapsto c_3'[E_\star/\alpha{:}\star] =_\alpha c_3\end{aligned}$$

- (*non-value*): We have the derivation

$$\dfrac{\dfrac{\dfrac{c_1' \Rightarrow c_3'}{\mu\alpha{:}\star.c_1' \Rightarrow^i \mu\alpha{:}\star.c_3'} \quad e_\star \Mapsto E_\star}{\langle \mu\alpha{:}\star.c_1' \| e_\star\rangle \Rightarrow^i \langle \mu\alpha{:}\star.c_3' \| E_\star\rangle \rightarrowtail_{\beta^\star_\mu} c_3'[E_\star/\alpha{:}\star]}}{\dfrac{\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle \mathrel{\mapstochar\twoheadrightarrow} \langle \mu\alpha{:}\star.c_1' \| e_\star\rangle \Rightarrow c_3}{\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle \Mapsto c_3}}$$

- ($\delta^\star$): If

$$\dfrac{\dfrac{\dfrac{c_1 \Rightarrow c_2}{\mu\alpha{:}\star.c_1 \Rightarrow^i \mu\alpha{:}\star.c_2} \quad e_\star \Mapsto e_\star'}{\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle \Rightarrow^i \langle \mu\alpha{:}\star.c_2 \| e_\star'\rangle \rightarrowtail_{\delta^\star} c_2}}{\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle \Rightarrow c_2 \mapsto c_3}$$

then the inductive hypothesis on $c_1 \Rightarrow c_2 \mapsto c_3$ gives $c_1 \mathrel{\mapstochar\twoheadrightarrow} c_2' \Rightarrow c_3$ for some c_2' so we have

$$\dfrac{\dfrac{\dfrac{\dfrac{c_2' \Rightarrow c_3}{\mu\alpha{:}\star.c_2' \Rightarrow^i \mu\alpha{:}\star.c_3} \quad e_\star \Mapsto e_\star'}{\langle \mu\alpha{:}\star.c_2' \| e_\star\rangle \Rightarrow^i \langle \mu\alpha{:}\star.c_3 \| e_\star'\rangle \rightarrowtail_{\delta^\star} c_3}}{\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle \mathrel{\mapstochar\twoheadrightarrow} \langle \mu\alpha{:}\star.c_2' \| e_\star\rangle \Rightarrow c_3}}{\langle \mu\alpha{:}\star.c_1 \| e_\star\rangle \Mapsto c_3}$$

- The cases for $\eta^{\pm\bigstar}_\mu$, $\eta^\bigstar_\mu$, $\beta^\bigstar_\mu$, and $\delta^\bigstar$ are dual to the above.

The third group of cases are for $\Rightarrow^i$ followed by a standard rewriting rule, which proceeds by cases on the rule:

- (β^+_μ): If

$$\dfrac{\dfrac{c \Mapsto c'}{\mu\alpha{:}{+}.c \Rightarrow^i \mu\alpha{:}{+}.c'} \quad E_+ \Mapsto E_+'}{\langle \mu\alpha{:}{+}.c \| E_+\rangle \Rightarrow^i \langle \mu\alpha{:}{+}.c' \| E_+'\rangle \mapsto_{\beta^+_\mu} c'[E_+'/\alpha{:}{+}]}$$

then we have

$$\dfrac{\langle \mu\alpha{:}{+}.c \| E_+\rangle \mapsto_{\beta^+_\mu} c[E_+/\alpha{:}{+}] \overset{\overset{c \Mapsto c' \quad E_+ \Mapsto E_+'}{\vdots}}{\Mapsto} c'[E_+'/\alpha{:}{+}]}{\langle \mu\alpha{:}{+}.c \| E_+\rangle \Mapsto c'[E_+'/\alpha{:}{+}]}$$

- (β_μ^-): If

$$\dfrac{\dfrac{c \Mapsto c'}{\mu\alpha{:}-.c \Rightarrow^i \mu\alpha{:}-.c'} \quad E_- \Rightarrow^i E'_-}{\langle \mu\alpha{:}-.c \| E_- \rangle \Rightarrow^i \langle \mu\alpha{:}-.c' \| E'_- \rangle \mapsto_{\beta_\mu^-} c'[E'_-/\alpha{:}-]}$$

then we have $\langle \mu\alpha{:}-.c \| E_- \rangle \Mapsto c'[E'_-/\alpha{:}-]$ analogously to the previous case for β_μ^+ by weakening $E_- \Rightarrow^i E'_-$ to $E_- \Mapsto E'_-$.

- $(\beta_\mu^\star)$: If

$$\dfrac{\dfrac{c \Mapsto c'}{\mu\alpha{:}\star.c \Rightarrow^i \mu\alpha{:}\star.c'} \quad E_\star \Rightarrow^i E'_\star \succcurlyeq_{\eta_\mu^\star} E''_\star}{\langle \mu\alpha{:}\star.c \| E_\star \rangle \Rightarrow^i \langle \mu\alpha{:}\star.c' \| E''_\star \rangle \mapsto_{\beta_\mu^\star} c'[E''_\star/\alpha{:}\star]}$$

then we have $\langle \mu\alpha{:}\star.c \| E_\star \rangle \Mapsto c'[E''_\star/\alpha{:}\star]$ analogously to the previous case for β_μ^+ by weakening $E_\star \Rightarrow^i E'_\star \succcurlyeq_{\eta_\mu^\star} E''_\star$ to $E_\star \Mapsto E''_\star$.

- $(\phi_\mu^\varstar)$: If

$$\dfrac{\dfrac{c \Rightarrow c'}{\mu\alpha{:}\varstar.c \Rightarrow^i \mu\alpha{:}\varstar.c'} \quad E_\varstar \Rightarrow^i E'_\varstar}{\langle \mu\alpha{:}\varstar.c \| E_\varstar \rangle \Rightarrow^i \langle \mu\alpha{:}\varstar.c' \| E''_\varstar \rangle \mapsto_{\phi_\mu^\varstar} c'[E''_\varstar/\alpha{:}\varstar]}$$

where $c' =_\alpha H'[\langle V'_\varstar \| \alpha{:}\varstar \rangle]$ with α not bound by H' because of the standard $\phi_\mu^\varstar$ reduction, and $c =_\alpha H[\langle V_\varstar \| \alpha{:}\varstar \rangle]$ with α not bound by H because $c \Rightarrow c'$ (following by induction on H'), then we have the weakened $E_\varstar \Mapsto E'_\varstar$ from $E_\varstar \Rightarrow^i E'_\varstar$ and the derivation

$$\dfrac{\begin{array}{c} c \Mapsto c' \quad E_\varstar \Mapsto E'_\varstar \\ \vdots \\ \langle \mu\alpha{:}\varstar.c \| E_\varstar \rangle \mapsto_{\phi_\mu^\varstar} c[E_\varstar/\alpha{:}\varstar] \Mapsto c'[E'_\varstar/\alpha{:}\varstar] \end{array}}{\langle \mu\alpha{:}\varstar.c \| E_\varstar \rangle \Mapsto c'[E'_+/\alpha{:}\varstar]}$$

- (β_{G}) for a co-data declaration G in $\mathcal{G}$: For simplicity, we will illustrate $\mathsf{G} = \&$; the general case for an arbitrary co-data declaration is analogous to this one by induction on the components of the possible observations of that co-data type. If

$$\dfrac{\dfrac{c_1 \Mapsto c'_1 \quad c_2 \Mapsto c'_2}{\begin{array}{l} \lambda\{\pi_1\boldsymbol{\alpha}_1.c_1 \mid \pi_2\boldsymbol{\alpha}_2.c_2\} \\ \Rightarrow^i \lambda\{\pi_1\boldsymbol{\alpha}_1.c'_1 \mid \pi_2\boldsymbol{\alpha}_2.c'_2\} \end{array}} \quad E \Rightarrow^i E'}{\begin{array}{l} \langle \lambda\{\pi_1\boldsymbol{\alpha}_1.c_1 \mid \pi_2\boldsymbol{\alpha}_2.c_2\} \| \pi_i E \rangle \\ \Rightarrow^i \langle \lambda\{\pi_1\boldsymbol{\alpha}_1.c'_1 \mid \pi_2\boldsymbol{\alpha}_2.c'_2\} \| \pi_i E' \rangle \\ \mapsto_{\beta_\&} c'_i[E'/\boldsymbol{\alpha}_i] \end{array}}$$

then $E \Rightarrow^i E'$ can be weakened to $E \Mapsto E'$ so we have

$$\dfrac{\begin{array}{c} c_i \Mapsto c'_i \quad E \Mapsto E' \\ \vdots \\ \langle \lambda\{\pi_1\boldsymbol{\alpha}_1.c_1 | \pi_2\boldsymbol{\alpha}_2.c_2\} \| \pi_i E \rangle \mapsto_{\beta_\&} c_i[E/\boldsymbol{\alpha}_i] \Mapsto c'_i[E'/\boldsymbol{\alpha}_i] \end{array}}{\langle \lambda\{\pi_1\boldsymbol{\alpha}_1.c_1 | \pi_2\boldsymbol{\alpha}_2.c_2\} \| \pi_i E \rangle \Mapsto c'_i[E'/\boldsymbol{\alpha}_i]}$$

- (ς_{G}) for a co-data declaration G in $\mathcal{G}$: As before, we will illustrate for the special case where $\mathsf{G} = \&$, which can be generalized to arbitrary co-data declarations by induction on the components of the possible observations. If

$$\frac{f \Mapsto f'}{\pi_i f \Rightarrow^i \pi_i f' \mapsto_{\varsigma_\&} \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| \pi_i \boldsymbol{\beta}\rangle \| f'\rangle}$$

then we have

$$\frac{\begin{array}{c} f \Mapsto f' \\ \vdots \\ \pi_i f \mapsto_{\varsigma_\&} \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| \pi_i \boldsymbol{\beta}\rangle \| f\rangle \Mapsto \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| \pi_i \boldsymbol{\beta}\rangle \| f'\rangle \end{array}}{\pi_i f \Mapsto \tilde{\mu}\boldsymbol{x}.\langle \mu\boldsymbol{\beta}.\langle \boldsymbol{x} \| \pi_i \boldsymbol{\beta}\rangle \| f'\rangle}$$

- The cases of each of the rules $\beta^{\pm\star}_{\tilde{\mu}}$, $\phi^{\star}_{\tilde{\mu}}$, and β_{F} and ς_{F} for a data declaration F in $\mathcal{G}$ being applied after a $\Rightarrow^i$ reduction on a command are dual to the above.

The last group of cases are for $\Rightarrow^i$ followed by $\mapsto$ on a strict sub-expression due to compatibility:

- If $\mathcal{S} \in \{+, \star\}$, $v'_{\mathcal{S}} \mapsto v''_{\mathcal{S}}$, and

$$\frac{w_{\mathcal{S}} \Rightarrow^i v_{\mathcal{S}} \succcurlyeq v'_{\mathcal{S}} \quad e_{\mathcal{S}} \Mapsto e'_{\mathcal{S}}}{\langle w_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \Rightarrow^i \langle v'_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle \mapsto \langle v''_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle}$$

then the inductive hypothesis on $w_{\mathcal{S}} \Rightarrow v'_{\mathcal{S}} \mapsto v''_{\mathcal{S}}$ gives $w_{\mathcal{S}} \Mapsto v''_{\mathcal{S}}$, so that $\langle w_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \Mapsto \langle v''_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle$.
- If $\mathcal{S} \in \{+, \star\}$, $e'_{\mathcal{S}} \mapsto e''_{\mathcal{S}}$, and

$$\frac{V_{\mathcal{S}} \Rightarrow^i V'_{\mathcal{S}} \succcurlyeq V''_{\mathcal{S}} \quad e_{\mathcal{S}} \Rightarrow^i e'_{\mathcal{S}}}{\langle V_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \Rightarrow^i \langle V''_{\mathcal{S}} \| e'_{\mathcal{S}}\rangle \mapsto \langle V''_{\mathcal{S}} \| e''_{\mathcal{S}}\rangle}$$

then the inductive hypothesis on $e_{\mathcal{S}} \Rightarrow^i e'_{\mathcal{S}} \mapsto e''_{\mathcal{S}}$ gives $e_{\mathcal{S}} \Mapsto e''_{\mathcal{S}}$, so that $\langle V_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \Mapsto \langle V''_{\mathcal{S}} \| e''_{\mathcal{S}}\rangle$.
- If $c_2 \mapsto c_3$ and

$$\frac{c_1 \Rightarrow c_2}{\mu\alpha{:}\star.c_1 \Rightarrow^i \mu\alpha{:}\star.c_2 \mapsto \mu\alpha{:}\star.c_3}$$

then the inductive hypothesis on $c_1 \Rightarrow c_2 \mapsto c_3$ gives $c_1 \twoheadmapsto c'_2 \Rightarrow c_3$ for some c'_2, so that

$$\frac{\dfrac{\dfrac{c'_2 \Rightarrow c_3}{\mu\alpha{:}\star.c'_2 \Rightarrow^i \mu\alpha{:}\star.c_3}}{\mu\alpha{:}\star.c_1 \twoheadmapsto \mu\alpha{:}\star.c'_2 \Rightarrow \mu\alpha{:}\star.c_3}}{\mu\alpha{:}\star.c_1 \Mapsto \mu\alpha{:}\star.c_3}$$

- The cases for $-$ and $\star$ commands and $\star$ $\tilde{\mu}$-abstractions are dual to the above.

Note that there is no possible standard reduction on variables and co-variables, $+$ and $-$ μ- and $\tilde{\mu}$-abstractions, pattern-matching terms and co-terms, and data and co-data structures built from (co-)values, so the $\Rightarrow^i$ cases for (co-)terms of these syntactic forms are impossible. ◀

▶ **Lemma 41.** *If $m \twoheadrightarrowtail\twoheadmapsto m'$ then $m \twoheadmapsto\twoheadrightarrowtail m'$. It follows that if $m \twoheadrightarrow m'$ then $m \twoheadmapsto\twoheadrightarrowtail m'$.*

Proof. First note the following facts:

a) The reflexive-transitive closure of $\Rrightarrow$ (written as $\Rrightarrow^*$) is the same as $\mathrel{\rightarrowtail\!\!\!\twoheadrightarrow}$, which follows from the fact that $m \rightarrowtail m'$ implies $m \Rrightarrow m'$ implies $m \mathrel{\rightarrowtail\!\!\!\twoheadrightarrow} m'$.
b) If $m \Rrightarrow\mathrel{\mapsto\!\!\!\twoheadrightarrow} m'$ then $m \mathrel{\mapsto\!\!\!\twoheadrightarrow}\Rrightarrow m'$, which follows by induction on the reflexive-transitive structure of $\mathrel{\mapsto\!\!\!\twoheadrightarrow}$.
c) If $m \Rrightarrow^* \ \mathrel{\mapsto\!\!\!\twoheadrightarrow} m'$ (where $\Rrightarrow^*$ is the reflexive-transitive closure of $\Rrightarrow$) then $m \mathrel{\mapsto\!\!\!\twoheadrightarrow}\Rrightarrow^* m'$, which follows from the previous fact by induction on the reflexive transitive structure of $\Rrightarrow^*$.

Therefore, $m \mathrel{\rightarrowtail\!\!\!\twoheadrightarrow}\mathrel{\mapsto\!\!\!\twoheadrightarrow} m'$ implies $m \mathrel{\mapsto\!\!\!\twoheadrightarrow}\mathrel{\rightarrowtail\!\!\!\twoheadrightarrow} m'$. Finally, since every $\to$ is either $\mapsto$ or $\rightarrowtail$, $m \twoheadrightarrow m'$ implies $m \mathrel{\mapsto\!\!\!\twoheadrightarrow}\mathrel{\rightarrowtail\!\!\!\twoheadrightarrow} m'$ because of the previous fact by induction on the reflexive-transitive structure of $\twoheadrightarrow$, commuting standard reductions before non-standard ones as necessary. ◀

▶ **Theorem 42** (Standardization). *If $m \twoheadrightarrow m'$ and m' is finished then $m \Downarrow m'' \twoheadrightarrow m'$.*

Proof. We know that $m \mathrel{\mapsto\!\!\!\twoheadrightarrow} m'' \mathrel{\rightarrowtail\!\!\!\twoheadrightarrow} m'$ for some m', and since m' is final then m'' must be as well so $m \Downarrow m''$. ◀

I.3 Untyped contextual equivalence

First, we define the following "weak equivalence" relation $\overset{\omega}{\sim}$ on commands that only checks compatibility of their needed variables:

$$c_1 \overset{\omega}{\sim} c_2 \iff \mathrm{NV}(c_1) = \mathrm{NV}(c_2)$$

▶ **Property I.1.** $c_1 \overset{\omega}{\sim} c_2$ if and only if $H_1[c_1] \overset{\omega}{\sim} H_2[c_2]$.

For any relation $\diamond$ on finished commands, we define the following generalization of $\diamond$ up to evaluation, written $\diamond\Downarrow$:

$$c_1 \diamond\Downarrow c_2 \iff (c_1 \Downarrow c_1' \implies \exists c_2 \Downarrow c_2'.c_1' \diamond c_2') \land (c_2 \Downarrow c_2' \implies \exists c_1 \Downarrow c_1'.c_1' \diamond c_2')$$

▶ **Definition 43** (Untyped contextual equivalence). Two well-disciplined (but untyped) expressions m_1 and m_2 of the same syntactic sort (command, $\mathcal{S}$ term, or $\mathcal{S}$ co-term) are *contextually equivalent*, denoted by $m_1 \cong m_2$, exactly when $C[m_1] \overset{\omega}{\sim}\Downarrow C[m_2]$ for all contexts C such that $C[m_1]$ and $C[m_2]$ are well-disciplined commands.

Note that if $\diamond$ is reflexive, transitive, and symmetric on finished commands then so too is $\diamond\Downarrow$. So since that $\overset{\omega}{\sim}$ is reflexive, transitive, and symmetric, so is contextual equivalence.

▶ **Lemma 44.** *Contextual equivalence is the largest compatible relation included in $\overset{\omega}{\sim}\Downarrow$. That is, given any compatible relation $\diamond$ between expressions such that $c_1 \diamond c_2$ implies $c_1 \overset{\omega}{\sim}\Downarrow c_2$, then $m_1 \diamond m_2$ implies $m_1 \cong m_2$.*

Proof. Suppose $m_1 \diamond m_2$ and C is an appropriate context such that $C[m_1]$ and $C[m_2]$ are well-disciplined commands. We know $C[m_1] \diamond C[m_2]$ by compatibility and thus $C[m_1] \overset{\omega}{\sim}\Downarrow C[m_2]$ by assumption. Therefore, $m_1 \cong m_2$. ◀

▶ **Lemma 45.** *If $\chi \notin R$ and $m_1 =_R m_2$ then $m_1 \cong m_2$.*

Proof. Follows from confluence and standardization of R and Theorem 44.

First, we show that R equality between finished commands is included in weak equivalence. Suppose that $c_1 =_R c_2$ and both are finished. By confluence, we know that $c_1 \twoheadrightarrow_R c' \twoheadleftarrow_R c_2$ such that c' is finished as well. Next, observe that $\rightarrow_R$ between finished commands is included in $\overset{\omega}{\approx}$, by cases on the applied reduction rule. It follows that $\twoheadrightarrow_R$ is also included in $\overset{\omega}{\approx}$ by induction on the reflexive-transitive structure of $\twoheadrightarrow_R$, and since $\overset{\omega}{\approx}$ is symmetric and transitive we have that $c_1 \overset{\omega}{\approx} c' \overset{\omega}{\approx} c_2$.

Second, note that equality is compatible by definition, so supposing that $c_1 =_R c_2$, it suffices to show that $c_1 \overset{\omega}{\approx}\Downarrow c_2$. By confluence, we know that $c_1 \twoheadrightarrow c_3 \twoheadleftarrow c_2$ for some c_3, and without loss of generality suppose that $c_1 \Downarrow c_1'$. By confluence of $c_3 \twoheadleftarrow c_1 \mapsto\!\!\!\!\twoheadrightarrow c_1'$, we have $c_3 \twoheadrightarrow c_3' \twoheadleftarrow c_1'$ meaning that $c_2 \twoheadrightarrow c_3 \twoheadrightarrow c_3'$ where c_3' is finished because $c_1' \twoheadrightarrow c_3'$. By standardization, we have that $c_2 \Downarrow c_2' \twoheadrightarrow c_3'$. So since $c_1' =_R c_2'$ and both are finished we have $c_1 \mapsto\!\!\!\!\twoheadrightarrow c_1' \overset{\omega}{\approx} c_2' \twoheadleftarrow\!\!\!\!\leftmapsto c_2$. Therefore, $c_1 \overset{\omega}{\approx}\Downarrow c_2$. ◀

The last step is to prove that the χ axiom on its own is included in contextual equivalence, which we can show by a bisimulation-style of argument.

▶ **Lemma 46.** *If $D_1[m_1] =_\chi m_2'$ then $m_2' =_\alpha D_2[m_2]$ for some D_2 and m_2 (of the same syntactic sort as m_1) such that $m_1 =_\chi m_2$, $D_1[m] =_\chi D_2[m]$ for all m (of the same syntactic sort as m_1 and m_2), and $\mathrm{NV}(D_1[m_1]) = \mathrm{NV}(D_2[m_2])$ when both are commands.*

Proof. The special case for $D_1[m_1] \leftrightarrow_\chi m_2'$ follows by induction on D_1, and the general case for $D_1[m_1] =_\chi m_2'$ follows from the previous fact by induction on the reflexive-transitive-symmetric structure of $=_\chi$ since $\leftrightarrow_\chi$ is also symmetric. ◀

▶ **Lemma 47.** *If $c_1 =_\chi c_2$ then c_1 is finished if and only if c_2 is, and if both are finished then $c_1 \overset{\omega}{\approx} c_2$.*

Proof. Given a finished command c_1, c_2 must also be finished because $\mathrm{NV}(c_1) = \mathrm{NV}(c_2)$, which also implies that $c_1 \overset{\omega}{\approx} c_2$. ◀

▶ **Lemma 48.** *If $m =_\chi\mapsto m'$ then $m \mapsto=_\chi m'$.*

Proof. First, observe that if $m_1 =_\chi m_2 \mapsto m_2'$ because $m_2 \succ m_2'$, then $m_1 \mapsto m_1' =_\chi m_2'$ for some $m_1 \succ m_1'$, which follows by cases on the standard rewriting rule used in $m_2 \succ m_2'$.

Second, suppose that $m =_\chi D_2[m_2] \mapsto D_2'[m_2']$ because $m_2 \succ m_2'$. It follows that $m =_\alpha D_1[m_1]$ for some D_1 and m_1 such that $m_1 =_\chi m_2$ and $D_1[m'] =_\chi D_2[m']$ for all syntactically appropriate m'. As shown previously, $m_1 =_\chi m_2 \succ m_2'$ implies that $m_1 \succ m_1' =_\chi m_2'$, so we have $D_1[m_1] \mapsto D_1[m_1'] =_\chi D_2[m_2']$. ◀

▶ **Lemma 49.** *If $m_1 =_\chi m_2$ then $m_1 \cong m_2$.*

Proof. χ-equality is compatible by definition, so it suffices to show that $c_1 =_\chi c_2$ implies $c_1 \overset{\omega}{\approx}\Downarrow c_2$. Without loss of generality (because both $=_\chi$ and $\overset{\omega}{\approx}\Downarrow$ are symmetric), suppose that $c_1 \Downarrow c_1'$ meaning that $c_1 \mapsto\!\!\!\!\twoheadrightarrow c_1'$ and c_1' is finished. By induction on the reflexive-transitive structure of $\mapsto\!\!\!\!\twoheadrightarrow$, the fact that $c_2 =_\chi c_1 \mapsto\!\!\!\!\twoheadrightarrow c_1'$ and c_1' is finished implies that $c_2 \mapsto\!\!\!\!\twoheadrightarrow c_2' =_\chi c_1'$ for some finished c_2' such that $c_1' \overset{\omega}{\approx} c_2'$. ◀

▶ **Theorem 50** (Untyped soundness)**.** *If $c_1 = c_2$ then $c_1 \cong c_2$.*

Proof. Follows by induction on $c_1 = c_2$ since every equality is a finite alternation of χ and non-χ equalities, and contextual equivalence is reflexive and transitive. ◀

▶ **Corollary 51** (Untyped coherence)**.** *There are two $\overset{\omega}{\approx}$-related and two $\overset{\omega}{\approx}$-unrelated expressions of every syntactic sort. It follows that there are two $=$-related and two $=$-unrelated expressions of every syntactic sort.*

Proof. Since both contextual equivalence and untyped equality is reflexive by definition, any expression of a syntactic sort is related to itself by both $\overset{\omega}{\approx}$ and $=$. Now, pick any two distinct variables x, y and α, β, and note that $\langle x{:}\mathcal{S} \| \alpha{:}\mathcal{S}\rangle \overset{\omega}{\not\approx} \langle y{:}\mathcal{S} \| \alpha{:}\mathcal{S}\rangle$ and $\langle x{:}\mathcal{S} \| \alpha{:}\mathcal{S}\rangle \overset{\omega}{\not\approx} \langle x{:}\mathcal{S} \| \beta{:}\mathcal{S}\rangle$. As special cases of contextual equivalence (with the contexts $\Box$, $\langle \Box \| \alpha{:}\mathcal{S}\rangle$, and $\langle x{:}\mathcal{S} \| \Box\rangle$, respectively), it follows that:

$$\langle x{:}\mathcal{S} \| \alpha{:}\mathcal{S}\rangle \ncong \langle y{:}\mathcal{S} \| \alpha{:}\mathcal{S}\rangle \qquad \langle x{:}\mathcal{S} \| \alpha{:}\mathcal{S}\rangle \ncong \langle x{:}\mathcal{S} \| \beta{:}\mathcal{S}\rangle \qquad x{:}\mathcal{S} \ncong y{:}\mathcal{S} \qquad \alpha{:}\mathcal{S} \ncong \beta{:}\mathcal{S}$$

And since the $=$ relation is included in $\overset{\omega}{\approx}$, these expressions must not be $=$-related. ◀

I.4 Typed contextual equivalence

We now defined a typed version of contextual equivalence that restricts compatibility to only well-typed contexts in observable environments.

▶ **Definition 52** (Observable environments)**.** An input environment Γ is *observable* with respect to $\mathcal{G}$ if $\Gamma = x : \mathsf{G}(A..)..$ where each G is a co-data type constructor $\mathsf{G} : k.. \to \mathcal{S}$ in $\mathcal{G}$ where $\mathcal{S} \neq \star$. Dually, an output environment Δ is *observable* with respect to $\mathcal{G}$ if $\Delta = \alpha : \mathsf{F}(A..)..$ where each F is a data type constructor $\mathsf{F} : k.. \to \mathcal{S}$ in $\mathcal{G}$ where $\mathcal{S} \neq \star$.

▶ **Definition 53** (Typed contextual equivalence)**.**

- Two typed commands are *contextually equivalent*, written as $c_1 \cong c_2 : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$, exactly when $C[c_1] \overset{\omega}{\approx}\!\Downarrow C[c_2]$ for all contexts C such that $C[c_i] : (\Gamma' \vdash^{\Theta}_{\mathcal{G}} \Delta')$ for some observable Γ' and Δ'.
- Two typed terms are *contextually equivalent*, written as $\Gamma \vdash^{\Theta}_{\mathcal{G}} v_1 \cong v_2 : A \mid \Delta$, exactly when $C[v_1] \overset{\omega}{\approx}\!\Downarrow C[v_2]$ for all contexts C such that $C[v_i] : (\Gamma' \vdash^{\Theta}_{\mathcal{G}} \Delta')$ for some observable Γ' and Δ'.
- Two typed co-terms are *contextually equivalent*, written as $\Gamma \mid e_1 \cong e_2 : A \vdash^{\Theta}_{\mathcal{G}} \Delta$, exactly when $C[e_1] \overset{\omega}{\approx}\!\Downarrow C[e_2]$ for all contexts C such that $C[e_i] : (\Gamma' \vdash^{\Theta}_{\mathcal{G}} \Delta')$ for some observable Γ' and Δ'.

Note that, like untyped contextual equivalence (Theorem 44), typed contextual equivalence is the largest typed relation that is compatible (in the typed sense) and included in $\overset{\omega}{\approx}\!\Downarrow$ for commands typed in an observable environment. That is, given any $\diamond$ between typed expressions that is compatible with well-typed contexts and where $c_1 \diamond c_2 : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ for an observable Γ and Δ implies that $c_1 \overset{\omega}{\approx}\!\Downarrow c_2$, then $c_1 \diamond c_2 : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ implies that $c_1 \cong c_2 : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ (and similarly for (co-)terms).

To prove that typed equality is sound with respect to typed contextual equivalence, we instantiate the orthogonality model of types from [6] in a way that accommodates call-by-need and its dual.

Top reduction on a single command, $c \rightsquigarrow c'$, is defined as:

$$\frac{c \succ_{\beta^{\mathcal{S}}_{\mu}\beta^{\mathcal{S}}_{\tilde{\mu}}\beta_{\mathcal{G}}} c'}{c \rightsquigarrow c'} \qquad \frac{v_{\mathcal{S}} \succ_{\varsigma_{\mathcal{G}}} v'_{\mathcal{S}}}{\langle v_{\mathcal{S}} \| E_{\mathcal{S}}\rangle \rightsquigarrow \langle v'_{\mathcal{S}} \| E_{\mathcal{S}}\rangle} \qquad \frac{e_{\mathcal{S}} \succ_{\varsigma_{\mathcal{G}}} e'_{\mathcal{S}}}{\langle V_{\mathcal{S}} \| e_{\mathcal{S}}\rangle \rightsquigarrow \langle V_{\mathcal{S}} \| e_{\mathcal{S}}\rangle}$$

Top reduction is then generalized to a pair of commands, $(c_1, c_2) \rightsquigarrow (c'_1, c'_2)$ as follows:

$$\frac{c_1 \rightsquigarrow c'_1}{(c_1, c_2) \rightsquigarrow (c'_1, c_2)} \qquad \frac{c_1 \rightsquigarrow c'_1 \quad c_2 \rightsquigarrow c'_2}{(c_1, c_2) \rightsquigarrow (c'_1, c'_2)} \qquad \frac{c_2 \rightsquigarrow c'_2}{(c_1, c_2) \rightsquigarrow (c_1, c'_2)}$$

▶ **Definition 54** (Poles and interaction spaces)**.** A *pole* $\mathbb{P}$ is a binary relation on commands. For a given pole $\mathbb{P}$, an $\mathbb{P}$-*interaction space* $\mathbb{A}$ (or just $\mathbb{P}$-*space* for short) is a binary relation between terms and a binary relation between co-terms, both of a common discipline $\mathcal{S}$, such that $v\ \mathbb{A} v'$ and $e\ \mathbb{A}\ e'$ implies that $\langle v \| e\rangle\ \mathbb{P}\ \langle v' \| e'\rangle$. Given two $\mathbb{P}$-spaces $\mathbb{A}$ and $\mathbb{B}$, we write $\mathbb{A} \sqsubseteq \mathbb{B}$ to mean that

$$v\ \mathbb{A}\ v' \implies v\ \mathbb{B}\ v' \qquad\qquad e\ \mathbb{A}\ e' \implies e\ \mathbb{B}\ e'$$

For any poles $\top\!\!\!\top$ and $\perp\!\!\!\perp$ and any $\top\!\!\!\top$-space $\mathbb{W}$, the $\perp\!\!\!\perp_{\mathbb{W}}$-*orthogonal* operation on $\top\!\!\!\top$-spaces, written $\_^{\perp\!\!\!\perp_{\mathbb{W}}}$, is defined as

$$v\ \mathbb{A}^{\perp\!\!\!\perp_{\mathbb{W}}}\ v' \iff \forall e\ \mathbb{A}\ e'.\langle v \| e\rangle \perp\!\!\!\perp \langle v' \| e'\rangle \qquad e\ \mathbb{A}^{\perp\!\!\!\perp_{\mathbb{W}}}\ e' \iff \forall v\ \mathbb{A}\ v'.\langle v \| e\rangle \perp\!\!\!\perp \langle v' \| e'\rangle$$

For any pole $\top\!\!\!\top$ and $\top\!\!\!\top$-space $\mathbb{V}$, the $\mathbb{V}$-*restriction* operation on $\top\!\!\!\top$-spaces, written $\_|_{\mathbb{V}}$, is defined as

$$v\ \ \mathbb{A}|_{\mathbb{V}}\ \ v' \iff v\ \mathbb{V}\ v' \wedge v\ \mathbb{A}\ v' \qquad\qquad e\ \ \mathbb{A}|_{\mathbb{V}}\ \ e' \iff e\ \mathbb{V}\ e' \wedge e\ \mathbb{A}\ e'$$

▶ **Definition 55** (Safety condition)**.** A *safety condition* $\mathbb{S}$ is a pair of poles $(\top\!\!\!\top, \perp\!\!\!\perp)$ such that $\perp\!\!\!\perp \subseteq \top\!\!\!\top$ and the following condition holds:

- *Closure under expansion:* For all $c_1 \top\!\!\!\top c_2$, if $(c_1, c_2) \rightsquigarrow (c_1', c_2')$ and $c_1' \perp\!\!\!\perp c_2'$ then $c_1 \perp\!\!\!\perp c_2$.

▶ **Definition 56** (Worlds)**.** For a safety condition $\mathbb{S} = (\top\!\!\!\top, \perp\!\!\!\perp)$, an $\mathbb{S}$-*world* $\mathbb{T}$ is a triple $(\mathbb{U}, \mathbb{V}, \mathbb{W})$ of $\top\!\!\!\top$-spaces such that $\mathbb{V} \sqsubseteq \mathbb{U}$, $\mathbb{W} \sqsubseteq \mathbb{U}$, and the following conditions hold:

- *Saturation:* For all $v_1\ \mathbb{U}\ v_2$, if $(\langle v_1 \| E_1\rangle, \langle v_2 \| E_2\rangle) \rightsquigarrow (c_1, c_2)$ for all $E_1\ \ \mathbb{W}|_{\mathbb{V}}^{\perp\!\!\!\perp_{\mathbb{W}}}\Big|_{\mathbb{V}}\ \ E_2$ and some $c_1 \perp\!\!\!\perp c_2$, then $v_1\ \mathbb{W}\ v_2$. Dually, for all $e_1\ \mathbb{U}\ e_2$, if $(\langle V_1 \| e_1\rangle, \langle V_2 \| e_2\rangle) \rightsquigarrow (c_1, c_2)$ for all $V_1\ \ \mathbb{W}|_{\mathbb{V}}^{\perp\!\!\!\perp_{\mathbb{W}}}\Big|_{\mathbb{V}}\ \ V_2$ and some $c_1 \perp\!\!\!\perp c_2$, then $e_1\ \mathbb{W}\ e_2$.
- *Generation:* For all $\top\!\!\!\top$-spaces $\mathbb{A} \sqsubseteq \mathbb{W}$, if $\mathbb{A} = \mathbb{A}|_{\mathbb{V}}^{\perp\!\!\!\perp_{\mathbb{W}}}$ then $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp_{\mathbb{W}}}$.

In addition to saturation and generation, there are two more important properties of a collection of worlds (one for each $\mathcal{S}$) that depend on the global environment $\mathcal{G}$. Letting

$$\begin{array}{ll}
\mathbf{data}\,\mathsf{F}(X : k).. : \mathcal{S}\,\mathbf{where} & \mathbf{codata}\,\mathsf{G}(X : k).. : \mathcal{S}\,\mathbf{where} \\
\quad \mathsf{K}_i : (A_{ij} : \mathcal{T}_{ij}\overset{j}{\ldots} \vdash^{Y:l..} \mathsf{F}\,X.. \mid B_{ij} : \mathcal{R}_{ij}\overset{j}{\ldots})\overset{i}{\ldots} & \quad \mathsf{O}_i : (A_{ij} : \mathcal{T}_{ij}\overset{j}{\ldots} \mid \mathsf{G}\,X.. \vdash^{Y:l..} B_{ij} : \mathcal{R}_{ij}\overset{j}{\ldots})\overset{i}{\ldots}
\end{array}$$

stand for a generic pair of data and co-data declarations in $\mathcal{G}$, then the collection of worlds $\mathbb{T}_{\mathcal{S}} = (\mathbb{U}_{\mathcal{S}}, \mathbb{V}_{\mathcal{S}}, \mathbb{W}_{\mathcal{S}})$ should have the two properties of:

- *Focalization:* If $V_j\ \ \mathbb{W}_{\mathcal{T}_{ij}}\big|_{\mathbb{V}_{\mathcal{T}_{ij}}}\ \ V_j'\overset{j}{\ldots}$ and $E_j\ \ \mathbb{W}_{\mathcal{R}_{ij}}\big|_{\mathbb{V}_{\mathcal{R}_{ij}}}\ \ E_j'\overset{j}{\ldots}$ then

  $$\mathsf{K}_i\,B..E_j\overset{j}{\ldots}V_j\overset{j}{\ldots}\ \ \mathbb{W}_{\mathcal{S}}|_{\mathbb{V}_{\mathcal{S}}}\ \ \mathsf{K}_i\,B'..E_j'\overset{j}{\ldots}V_j'\overset{j}{\ldots} \qquad \mathsf{O}_i\,B..V_j\overset{j}{\ldots}E_j\overset{j}{\ldots}\ \ \mathbb{W}_{\mathcal{S}}|_{\mathbb{V}_{\mathcal{S}}}\ \ \mathsf{O}_i\,B'..V_j'\overset{j}{\ldots}E_j'\overset{j}{\ldots}$$

  for each i. Additionally,

  $$\begin{array}{l}
  \lambda\{[\mathsf{O}_i\,\boldsymbol{Y}..x_{ij}{:}\mathcal{T}_{ij}\overset{j}{\ldots}\alpha_{ij}{:}\mathcal{R}_{ij}\overset{j}{\ldots}].c_i\overset{i}{\ldots}\}\ \ \mathbb{W}_{\mathcal{S}}\ \lambda\{[\mathsf{O}_i\,\boldsymbol{Y}..x_{ij}{:}\mathcal{T}_{ij}\overset{j}{\ldots}\alpha_{ij}{:}\mathcal{R}_{ij}\overset{j}{\ldots}].c_i\overset{i}{\ldots}\} \\
  \tilde{\lambda}\{(\mathsf{K}_i\,\boldsymbol{Y}..\alpha_{ij}{:}\mathcal{R}_{ij}\overset{j}{\ldots}, x_{ij}{:}\mathcal{T}_{ij}..).c_i\overset{i}{\ldots}\}\ \ \mathbb{W}_{\mathcal{S}}\ \tilde{\lambda}\{(\mathsf{K}_i\,\boldsymbol{Y}..\alpha_{ij}{:}\mathcal{R}_{ij}\overset{j}{\ldots}, x_{ij}{:}\mathcal{T}_{ij}..).c_i'\overset{i}{\ldots}\}
  \end{array}$$

  if

  $$c_i[B/\boldsymbol{Y}.., V_{ij}/x_{ij}{:}\mathcal{T}_{ij}\overset{j}{\ldots}, E_{ij}/\alpha_{ij}{:}\mathcal{R}_{ij}\overset{j}{\ldots}] \perp\!\!\!\perp c_i'[B'/\boldsymbol{Y}.., V_{ij}'/x_{ij}{:}\mathcal{T}_{ij}\overset{j}{\ldots}, E_{ij}'/\alpha_{ij}{:}\mathcal{R}_{ij}\overset{j}{\ldots}]\overset{i}{\ldots}$$

  for all $V_{ij}\ \ \mathbb{W}_{\mathcal{T}_{ij}}\big|_{\mathbb{V}_{\mathcal{T}_{ij}}}\ \ V_{ij}'\overset{ij}{\ldots}$ and $E_{ij}\ \ \mathbb{W}_{\mathcal{R}_{ij}}\big|_{\mathbb{V}_{\mathcal{R}_{ij}}}\ \ E_{ij}'\overset{ij}{\ldots}$.

- *Extensionality:* If $V \; \mathbb{V}_{\mathcal{S}} \; V'$ and $(\bigcup_i BV(q_i)) \cap (FV(V) \cup FV(V')) = \emptyset$ then

  $$V \; \mathbb{V}_{\mathcal{S}} \; \lambda\{q_i.\langle V' \| q_i\rangle \overset{i}{..}\} \qquad\qquad \lambda\{q_i.\langle V \| q_i\rangle \overset{i}{..}\} \; \mathbb{V}_{\mathcal{S}} \; V'$$

  Dually, if $E \; \mathbb{V}_{\mathcal{S}} \; E'$ and $(\bigcup_i BV(p_i)) \cap (FV(E) \cup FV(E'))$ then

  $$E \; \mathbb{V}_{\mathcal{S}} \; \tilde{\lambda}\{p_i.\langle p_i \| E'\rangle \overset{i}{..}\} \qquad\qquad \tilde{\lambda}\{p_i.\langle p_i \| E\rangle \overset{i}{..}\} \; \mathbb{V}_{\mathcal{S}} \; E'$$

Let the pole $\top\!\!\!\top$ be the total relation (*i.e.*, cross product) of commands with no free $\varstar$ variables and $\star$ co-variables. Likewise, let the interaction spaces $\mathbb{U}_{\mathcal{S}}$ and $\mathbb{V}_{\mathcal{S}}$ be the total relations between $\mathcal{S}$ terms and $\mathcal{S}$ co-terms (for $\mathbb{U}_{\mathcal{S}}$) and $\mathcal{S}$ values and $\mathcal{S}$ co-values (for $\mathbb{V}_{\mathcal{S}}$) which have no free $\varstar$ variables and $\star$ co-variables. The model is then instantiated by the following poles $\perp\!\!\!\perp$, $\mathbb{I}$ and interaction spaces $\mathbb{W}_{\mathcal{S}}$:

$$\begin{aligned}
c \perp\!\!\!\perp c' &\iff c \top\!\!\!\top c' \wedge c \overset{\omega}{\sim}\!\Downarrow c' & \qquad c \,\mathbb{I}\, c' &\iff c \top\!\!\!\top c' \\
v_{\varstar} \; \mathbb{W}_{\varstar} \; v'_{\varstar} &\iff v_{\varstar} \; \mathbb{X}_{\varstar}^{\perp\!\!\!\perp_{\mathbb{U}_{\varstar}}} v'_{\varstar} & \qquad v_{\pm\star} \; \mathbb{W}_{\pm\star} \; v'_{\pm\star} &\iff v_{\pm\star} \; \mathbb{U}_{\pm\star} \; v'_{\pm\star} \\
e_{\star} \; \mathbb{W}_{\star} \; e'_{\star} &\iff e_{\star} \; \mathbb{X}_{\star}^{\perp\!\!\!\perp_{\mathbb{U}_{\star}}} v'_{\star} & \qquad e_{\pm\varstar} \; \mathbb{W}_{\pm\varstar} \; e'_{\pm\varstar} &\iff e_{\pm\varstar} \; \mathbb{U}_{\pm\varstar} \; e'_{\pm\varstar}
\end{aligned}$$

where

$$\mathbb{X}_{\mathcal{S}} = (\{(x{:}\mathcal{S}, x{:}\mathcal{S}) \mid x \in \mathit{Var}\}, \{(\alpha{:}\mathcal{S}, \alpha{:}\mathcal{S}) \mid \alpha \in \mathit{CoVar}\})$$

Now some immediate facts that follow by the above definitions:

- $\perp\!\!\!\perp$ is closed under untyped observational equivalence of $\top\!\!\!\top$ commands: if $c_1 \top\!\!\!\top c_2$ and $c_1 \cong c_2$ then $c_1 \perp\!\!\!\perp c_2$. It follows that $\perp\!\!\!\perp$ is also closed under
  - untyped equality of $\top\!\!\!\top$ commands: if $c_1 \top\!\!\!\top c_2$ and $c_1 = c_2$ then $c_1 \perp\!\!\!\perp c_2$, and
  - $\top\!\!\!\top$-expansion: if $c_1 \top\!\!\!\top c_2$ and $c_1 \twoheadrightarrow c'_1 \perp\!\!\!\perp c'_2 \twoheadleftarrow c_2$ then $c_1 \perp\!\!\!\perp c_2$.

  This means that the triple $\mathbb{S} = (\top\!\!\!\top, \mathbb{I}, \perp\!\!\!\perp)$ is a *safety condition.*
- The $\mathbb{S}$-worlds $\mathbb{T}_{\mathcal{S}} = (\mathbb{U}_{\mathcal{S}}, \mathbb{V}_{\mathcal{S}}, \mathbb{W}_{\mathcal{S}})$ for $\mathcal{S} \in \{+, -\}$ are both (trivially) saturated, generative, focalizing, and extensional.

The $\mathbb{S}$-worlds $\mathbb{T}_{\mathcal{S}} = (\mathbb{U}_{\mathcal{S}}, \mathbb{V}_{\mathcal{S}}, \mathbb{W}_{\mathcal{S}})$ for $\mathcal{S} \in \{\varstar, \star\}$ are also both saturated, generative, and focalizing though these facts are more involved.

▶ **Lemma 57** (Saturation). *For $\mathcal{S} \in \{\varstar, \star\}$,*

a) if $v_1 \; \mathbb{U}_{\mathcal{S}} \; v_2$ and $\langle v_1 \| E_1\rangle \rightsquigarrow c_1 \perp\!\!\!\perp c_2 \leftsquigarrow \langle v_2 \| E_2\rangle$ for all $E_1 \;\; \mathbb{W}_{\mathcal{S}}|_{\mathbb{V}_{\mathcal{S}}}^{\perp\!\!\!\perp_{\mathbb{W}_{\mathcal{S}}}} \;\; E_2$ then $v_1 \; \mathbb{W}_{\mathcal{S}} \; v_2$, and
b) if $e_1 \; \mathbb{U}_{\mathcal{S}} \; e_2$ and $\langle V_1 \| e_1\rangle \rightsquigarrow c_1 \perp\!\!\!\perp c_2 \leftsquigarrow \langle V_2 \| e_2\rangle$ for all $V_1 \;\; \mathbb{W}_{\mathcal{S}}|_{\mathbb{V}_{\mathcal{S}}}^{\perp\!\!\!\perp_{\mathbb{W}_{\mathcal{S}}}} \;\; V_2$ then $e_1 \; \mathbb{W}_{\mathcal{S}} \; e_2$.

Proof. a) For $\mathcal{S} = \star$, this saturation property is trivial since $v_1 \; \mathbb{U}_{\star} \; v_2$ if and only if $v_1 \; \mathbb{W}_{\star} \; v_2$ by definition. For $\mathcal{S} = \varstar$, note that $\alpha_{\varstar} \;\; \mathbb{W}_{\varstar}|_{\mathbb{V}_{\varstar}}^{\perp\!\!\!\perp_{\mathbb{W}_{\varstar}}} \;\; \alpha_{\varstar}$ by definition of $\mathbb{W}_{\varstar}$, and thus

$$\langle v_1 \| \alpha_{\varstar}\rangle \rightsquigarrow c_1 \perp\!\!\!\perp c_2 \leftsquigarrow \langle v_2 \| \alpha_{\varstar}\rangle$$

so $\langle v_1 \| \alpha_{\varstar}\rangle \perp\!\!\!\perp \langle v_2 \| \alpha_{\varstar}\rangle$ by $\mathbb{I}$-expansion of $\perp\!\!\!\perp$, and therefore $v_1 \; \mathbb{W}_{\varstar} \; v_2$.

b) Follows dually to part (a).

◀

▶ **Lemma 58.**

a) If $e_1 \;\mathbb{W}_\star\; e_2$ then either $\langle v_1 \| e_1 \rangle \perp\!\!\!\perp \langle v_2 \| e_2 \rangle$ for all $v_1 \;\mathbb{W}_\star\; v_2$, or there are two co-values $E_1 \;\mathbb{W}_\star\; E_2$ such that $e_i \mapsto\!\!\!\!\twoheadrightarrow E_i$.
b) If $v_1 \;\mathbb{W}_\varstar\; v_2$ then either $\langle v_1 \| e_1 \rangle \perp\!\!\!\perp \langle v_2 \| e_2 \rangle$ for all $e_1 \;\mathbb{W}_\varstar\; e_2$, or there are two values $V_1 \;\mathbb{W}_\star\; V_2$ such that $v_i \mapsto\!\!\!\!\twoheadrightarrow V_i$.

Proof. a) Choose a variable x not free in e_1 and e_2, so that $\langle x{:}\star \| e_1 \rangle \overset{\omega}{\sim}\Downarrow \langle x{:}\star \| e_2 \rangle$, and note that $\langle x{:}\star \| e_i \rangle \mapsto c'$ if and only if $e_i \mapsto e'_i$ and $c' =_\alpha \langle x{:}\star \| e'_i \rangle$ due to the caveat of the $\phi^\star_{\tilde{\mu}}$ rule. Now suppose that $\langle x{:}\star \| e_1 \rangle \Downarrow \langle x{:}\star \| e'_1 \rangle$ and $\langle x{:}\star \| e_2 \rangle \Downarrow \langle x{:}\star \| e'_2 \rangle$, so that the result follows by cases on whether or not e'_i are co-values:

- If both e'_i co-values, the result is immediate by reflexivity of $\mapsto\!\!\!\!\twoheadrightarrow$.
- If both e'_i are not co-values, then let v_1 and v_2 be arbitrary $\star$ terms. e_1 and e_2 must be $\tilde{\mu}$-abstractions, $\tilde{\mu}x_1{:}\star.c_1$ and $\tilde{\mu}x_2{:}\star.c_2$ respectively, such that $c_1 \overset{\omega}{\sim}\Downarrow c_2$ because of $e_1 \;\mathbb{W}_\star\; e_2$ and Property I.1. From $c_1 \overset{\omega}{\sim}\Downarrow c_2$ and Property I.1, it follows that $\langle v_1 \| e_1 \rangle \overset{\omega}{\sim}\Downarrow \langle v_2 \| e_2 \rangle$.
- It is impossible for one of e'_i to be a co-value and the other not. Without loss of generality, suppose e'_1 is any co-value, e'_2 is a non-co-value. For $\langle x{:}\star \| e'_1 \rangle \overset{\omega}{\sim}\Downarrow \langle x{:}\star \| e'_2 \rangle$ to hold, it must be the case that $\langle x{:}\star \| e'_2 \rangle$ is finished and (because $\langle x{:}\star \| e'_1 \rangle$ is finished) and $\langle x{:}\star \| e'_1 \rangle \overset{\omega}{\sim} \langle x{:}\star \| e'_2 \rangle$. Note that $x{:}\star \in \mathrm{NV}\langle x{:}\star \| e'_1 \rangle$ because e'_1 is a co-value and $x{:}\star \notin \mathrm{NV}\langle x{:}\star \| e'_2 \rangle$ because e'_2 is not, which contradicts the requirements of $\overset{\omega}{\sim}$ that demand the set of needed (co-)variables to be the same.

b) Analogous to the proof of part (a) by duality.
Otherwise, both $\langle x{:}\star \| e_i \rangle \not\Downarrow$ so that if $\langle x{:}\star \| e_i \rangle \mapsto\!\!\!\!\twoheadrightarrow \langle x{:}\star \| e'_i \rangle \not\mapsto$ then $\langle x{:}\star \| e'_i \rangle$ is stuck and thus e'_i is not a co-value. It follows that for and any $\star$ terms v_1 and v_2, $\langle v_i \| e_i \rangle \not\Downarrow$, and so $\langle v_2 \| e_1 \rangle \overset{\omega}{\sim}\Downarrow \langle v_2 \| e_2 \rangle$. ◀

▶ **Lemma 59** (Generation). *For $\mathcal{S} \in \{\varstar, \star\}$ and $\mathbb{A} \sqsubseteq \mathbb{W}_\mathcal{S}$, then $\mathbb{A} = \mathbb{A}|^{\perp\!\!\!\perp_{\mathbb{W}_\mathcal{S}}}_{\mathbb{V}_\mathcal{S}}$ implies $\mathbb{A} = \mathbb{A}^{\perp\!\!\!\perp_{\mathbb{W}_\mathcal{S}}}$.*

Proof. Suppose that $\mathcal{S} = \star$, $v_1 \;\mathbb{A}\; v_2$ and $e_1 \;\mathbb{A}\; e_2$, so that we must show that $\langle v_1 \| e_1 \rangle \perp\!\!\!\perp \langle v_2 \| e_2 \rangle$. By Theorem 58, we know that either $\langle v_1 \| e_1 \rangle \perp\!\!\!\perp \langle v_2 \| e_2 \rangle$ immediately (because $\langle v'_1 \| e_1 \rangle \perp\!\!\!\perp \langle v'_2 \| e_2 \rangle$ for any $v'_1 \;\mathbb{W}_\star\; v'_2$) or $e_i \mapsto\!\!\!\!\twoheadrightarrow E_i$ for some E_1 and E_2. In the later case, since $\mathbb{A} = \mathbb{A}|^{\perp\!\!\!\perp_{\mathbb{W}_\mathcal{S}}}_{\mathbb{V}_\mathcal{S}}$ we have that

$$\langle v_1 \| e_1 \rangle \mapsto\!\!\!\!\twoheadrightarrow \langle v_1 \| E_1 \rangle \perp\!\!\!\perp \langle v_2 \| E_2 \rangle \twoheadleftarrow\!\!\!\!\leftarrow\!\!\!\!\!\dashv \langle v_2 \| e_2 \rangle$$

and so $\langle v_1 \| e_1 \rangle \perp\!\!\!\perp \langle v_2 \| e_2 \rangle$ by $\mathbb{I}$-expansion. The case where $\mathcal{S} = \varstar$ is dual. ◀

And finally, observe that the collection of worlds $\mathbb{T}_\mathcal{S}$ for all $\mathcal{S}$ satisfies the focalization and extensionality criteria, where the only interesting case is the $\eta_\mathcal{G}$ conversion of $\star$ data types and $\varstar$ co-data types: in both of these cases, we rely on the fact that free $\star$ co-variables and $\varstar$ variables are forbidden, so that the set of needed (co-)variables cannot be changed by $\eta_\mathcal{G}$ conversion.

Therefore, we have an instance of the parametric model of the sequent calculus, which gives us the adequacy of typed equality in terms.

▶ **Lemma 60** (Adequacy). *a) $c = c' : (\Gamma \vdash^\Theta_\mathcal{G} \Delta)$ implies $c = c' : (\Gamma \vDash^\Theta_\mathcal{G} \Delta)$.*
b) $\Gamma \vdash^\Theta_\mathcal{G} v = v' : A \mid \Delta$ implies $\Gamma \vDash^\Theta_\mathcal{G} v = v' : A \mid \Delta$.
c) $\Gamma \mid e = e' : A \vdash^\Theta_\mathcal{G} \Delta$ implies $\Gamma \mid e = e' : A \vDash^\Theta_\mathcal{G} \Delta$.

Proof. An instance of Lemma 7.12 from [6], where the case for recursive (co-)terms follows the methodology of [20]. ◀

And from adequacy, we get the soundness of typed equality with respect to contextual equivalence.

▶ **Theorem 61** (Typed equality soundness). *a)* $c = c' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$ *implies* $c \cong c' : (\Gamma \vdash^{\Theta}_{\mathcal{G}} \Delta)$.
b) $\Gamma \vdash^{\Theta}_{\mathcal{G}} v = v' : A \mid \Delta$ *implies* $\Gamma \vdash^{\Theta}_{\mathcal{G}} v \cong v' : A \mid \Delta$.
c) $\Gamma \mid e = e' : A \vdash^{\Theta}_{\mathcal{G}} \Delta$ *implies* $\Gamma \mid e \cong e' : A \vdash^{\Theta}_{\mathcal{G}} \Delta$.

Proof. First, note that the equality relation is compatible, so we only need to show that commands that are equal in an observable environment are related by $\overset{\omega}{\sim}\Downarrow$. But this follows from adequacy (Theorem 60), since the interpretation of typing judgements of commands can be instantiated by the identity substitution for every observable Γ and Δ, implying that the commands are related by $\overset{\omega}{\sim}\Downarrow$ by the definition of $\perp\!\!\!\perp$. ◀

▶ **Theorem 1** (Closed coherence). *For any global environment* $\vdash \mathcal{G}$ *extending* $\mathcal{F}$, *the equality* $\vdash_{\mathcal{G}} \iota_1() = \iota_2() : 1 \oplus 1 : +$ *is not derivable.*

Proof. Assuming that $\vdash_{\mathcal{G}} \iota_1() = \iota_2() : 1 \oplus 1 \mid$ is derivable, then $\vdash_{\mathcal{G}} \iota_1() \cong \iota_2() : 1 \oplus 1 \mid$ holds by Theorem 61. It follows that for $i \in \{1, 2\}$, since both

$$\langle \iota_i() \| \tilde{\lambda}\{\iota_1 \boldsymbol{z}.\langle x \| \mathsf{eval}_+\, \alpha\rangle \mid \iota_2 \boldsymbol{z}.\langle y \| \mathsf{eval}_+\, \alpha\rangle\}\rangle : (x{:}{\uparrow}1, y{:}{\uparrow}1 \vdash_{\mathcal{G}} \alpha{:}1)$$

are well-typed in an observable environment, they must be related by $\overset{\omega}{\sim}\Downarrow$. But $i = 1$ evaluates to $\langle x \| \mathsf{eval}_+\, \alpha\rangle$ and $i = 2$ evaluates to $\langle y \| \mathsf{eval}_+\, \alpha\rangle$, so they are not related by $\overset{\omega}{\sim}\Downarrow$, which is a contradiction. Therefore $\vdash_{\mathcal{G}} \iota_1() = \iota_2() : 1 \oplus 1 \mid$ cannot be derivable. Note that since the command lies in the functional sub-calculus of $\mathcal{D}$, they cannot be equal in extensible $\mathcal{F}$ either because of the equational correspondence between the two calculi (Theorem 24). ◀