A Polarized Basis for Simple Types

PAUL DOWNEN, University of Oregon
ZENA M. ARIOLA, University of Oregon

We employ encodings all the time as programming language designers, implementers, and theorists, but those encodings are not always accurate representations in practical languages where program features, like exceptions or even recursion, can sometimes turn obvious encodings into leaky abstractions. Here, we show how polarized types let us rely on the common encodings we know and love for supporting user-defined types from both eager and lazy languages like ML and Haskell. We use type isomorphisms as a technique for showing that the proposed encodings are faithful, so that we can encode and decode without any loss of information, and that they exhibit the mathematical and logical properties that we should expect. In the end, the polarized basis of types gives us a unified core language for both eager and lazy functional languages alike.

CCS Concepts: • **Theory of computation → Type structures**; • **Software and its engineering → Data types and structures**;

Additional Key Words and Phrases: simple types, polarity, control effects, sequent calculus, type isomorphism

ACM Reference format:
Paul Downen and Zena M. Ariola. 2018. A Polarized Basis for Simple Types. *Proc. ACM Program. Lang.* 1, 1, Article 1 (January 2018), 54 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

How many types do programming languages really need? Mainstream statically typed programming languages all have some mechanisms for programmers to declare their own custom types to make writing software easier, so in practice there seems to be a limitless supply of different types in languages. However, when we are not working *in* a language but *with* a language—for example, to study its theoretical properties or to develop practical implementations—the fewer constructs and types the language has the easier it is to work with. This is where functional programming languages can shine by using their connection with logic to simplify the language. For example, logic tells us that we only need a binary conjunction connective since larger conjunctions can be encoded by nesting applications of the binary one, and nothing is gained or lost because either nesting (to the left or to the right) is equivalently provable. This lets us encode complex propositions and connectives in terms of a smaller number of more basic connectives. These encodings correspond to common techniques to simplify models of functional programming languages down to just a handful of primitive types because the rest can be encoded by converting to and fro.

Unfortunately, in real functional languages these seemingly obvious encodings are not accurate because the two types do not actually describe the *same* set of program behaviors. If we want to say that a type is unnecessary because it can be encoded away, we should expect a one-for-one correspondence between the programs of both types, but this is often not the case because of the *computational effects* in the language. For example, encoding triples as nested pairs does not cause issue in an eager language like SML, but in a lazy language like Haskell we can observe a

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

difference because of divergent (*i.e.,* infinitely looping) or erroneous values like $1/0$ or *undefined*. The conversion between nested pairs of type $(a, (b, c))$ and triples of type (a, b, c) goes as follows:

$$\mathit{fromTriple}\ (x, y, z) = (x, (y, z)) \qquad \mathit{toTriple}\ p = (\mathit{fst}\ p, \mathit{fst}\ (\mathit{snd}\ p), \mathit{snd}\ (\mathit{snd}\ p))$$

However, the nested pair type $(\mathit{Int}, (\mathit{Bool}, \mathit{String}))$ in Haskell contains both $(1, \bot)$ and $(1, (\bot, \bot))$, where $\bot = \mathit{undefined}$, which can be distinguished by pattern-matching: $\mathbf{case}\ x\ \mathbf{of}\ (\_, (\_, \_)) \to 9$ yields 9 when $x = (1, (\bot, \bot))$ but gives *undefined* when $x = (1, \bot)$. Yet the two different pairs are collapsed in the triple type (Int, b, c) which can only express $(1, \bot, \bot)$, so a round trip to and from triples doesn't give back what we put in.[1]

The issues with unfaithful encodings are not just isolated to lazy languages; eager languages like SML have similar problems with different types. For example, the common technique known as *currying* in functional languages, which converts between functions of type $(a, b) \to c$ and $a \to (b \to c)$, lets us represent binary functions as unary functions nested in the right way:

$$\mathit{curry}\ f\ x\ y = f\ (x, y) \qquad \mathit{uncurry}\ f\ (x, y) = f\ x\ y$$

However, this encoding too is not accurate. The type $a \to (b \to c)$ contains both functions $\lambda x.\, \mathbf{raise}\ \mathit{Div}$ and $\lambda x.\lambda y.\, \mathbf{raise}\ \mathit{Div}$ (where $\mathbf{raise}\ \mathit{Div}$ raises a divide-by-zero exception in SML), which are observably different because the partial application $f\ 1$ raises an exception when $f = \lambda x.\, \mathbf{raise}\ \mathit{Div}$ and returns a function when $f = \lambda x.\lambda y.\, \mathbf{raise}\ \mathit{Div}$. Yet, *uncurry* collapses these distinct values into $\lambda(x, y).\, \mathbf{raise}\ \mathit{Div}$, so a round-trip of *uncurry*ing and *curry*ing does not give back the same function.[2] Both of these counter-examples to simple, well-accepted encodings still cause trouble with general recursion or loops—found pervasively among mainstream programming languages—in place of exceptions: just replace *undefined* and $\mathbf{raise}\ \mathit{Div}$ with an infinite loop.

The impact of round-trip inaccuracy means that encodings have limited use in practice, for example, to simplify intermediate languages in an optimizing compiler. Clearly in source programming languages, we would prefer to have n-ary tuples instead of encoding them by hand. Likewise in the target language, it is better to represent n-ary tuples directly when compiling lazy languages since they improve efficiency by reducing excessive indirection and thunking caused by nested lazy pairs. But in the middle of the compiler, it can be helpful to simplify the intermediate language by reducing the complexity and number of different programming constructs to some minimal core. That means that to utilize the above encodings in the middle of the compilation process, we need to go back and forth between encoding and decoding, and inaccurate encodings means that properties of the source and target language are lost: for example, the fact that $\lambda(x, y).f\ (x, y) = f$ in SML is lost by currying since $\lambda x.\lambda y.f\ x\ y \neq f$. The impact of such inaccuracies is that using the encoding can prevent optimizations that would be sound in the source language, or even worse have the potential to introduce unsound transformations with respect to the target implementation.

So does this mean all hope is lost when our functional languages have effects, or even just general recursion? Must we choose between living with unfaithful encodings or giving up entirely on the game of encoding complex types into simpler primitives altogether? Thankfully, we do not have to choose! The root cause of the unfaithfulness comes from a mismatch in the opportunities for eagerness or laziness of programs that is implicit in types. This inherent connection between types and evaluation strategy (*i.e.,* eagerness and laziness) has been studied under the guise of *polarity* [Munch-Maccagnoni 2013; Zeilberger 2009], which was originally developed in the setting of proof

[1] There is also the stricter alternative definition $\mathit{toTriple}\ (x, (y, z)) = (x, y, z)$, but this instead collapses $(5, \bot)$ and $\bot$.

[2] Haskell also exhibits problems with currying, but instead due to differences in strictness on pairs. Specifically, $\lambda(\_, \_).9$ and $\lambda\_.9$ are different functions of type $(a, b) \to \mathit{Int}$ because they differ on the input $\bot :: (a, b)$, but these two functions are collapsed by a round-trip through currying and uncurrying (one way or the other, depending on the strictness of *uncurry*). See appendix E for more details on lazy currying.

search rather than evaluation, as well as the *call-by-push-value* strategy [Levy 2003]. And it turns out that this fine-grained approach where programs can intermingle *both* eager and lazy evaluation at will gives us the tools we need to build strong encodings of types in languages with effects that accurately represent the full gamut of user-defined types.

For our setting, we will work with a language based on the classical sequent calculus (section 2). This calculus has a built-in control effect which makes the above issues of eagerness and laziness relevant for encodings (since an abort causes similar issues as an exception or an infinite loop), allows for both mixing both eager and lazy evaluation (in terms of call-by-value, call-by-name, call-by-need, or its dual) within a single program, and lets us express a set of basic connectives with pleasant symmetries and algebraic properties. As contributions, we give:[3]

- A primitive basis of polarized connectives suitable for encoding all simple (*i.e.,* monomorphic and non-recursive) user-defined (co-)data types in languages with effects (section 3).
- A polarized definition of isomorphism between types and (co-)data declarations (section 4).
- A syntactic theory of isomorphisms for (co-)data types and their declarations (section 5).
- A demonstration that the commonly expected algebraic and logical laws are sound (with respect to type isomorphism) for the primitive basis of polarized connectives (section 6).
- An encoding of all user-defined (co-)data types in terms of the primitive basis, such that the encoded type is indeed isomorphic to the user-defined one (section 7).

Since we intend for this work to be applicable to compilers which optimize programs by rewriting code, our focus is on *syntactic* theories which represent isomorphisms and program equivalence as applications of purely syntactic program transformations. For that reason, we use a syntactic equational theory to decide a canonical, finite set of (co-)data types (*i.e.,* connectives) used to represent all the others. This basis of connectives should be able to represent any types that mix evaluation strategies as in practical functional languages (like Haskell and OCaml) and any types that are expressible in the *classical* sequent calculus, which is a superset of the (co-)data types found in functional programming languages [Downen et al. 2015]. As a side-effect of establishing this general-purpose basis for types, we learn the following new insights:

- Usually only call-by-value and call-by-name are considered in these kinds of (polarized) encodings, but based on the analysis of Downen and Ariola [2014] we addresses how to integrate other evaluation strategies like call-by-need which is necessary for practical lazy languages. This is achieved by relying on the properties of *linearity* and *thunkability* that Munch-Maccagnoni [2013] shows arise in polarized languages.
- Polarized [Zeilberger 2008] and call-by-push-value [Levy 2001] languages usually only have the two connectives for shifting between eager (values) and lazy (computations). Here, we instead use *four* for each basic evaluation order: two as data and two as co-data. The existence of two different adjoint shift pairs is not well understood, and here we put it on firmer ground. Furthermore, while some shifts are redundant for call-by-value and call-by-name (amounting to trivial identity types), *all* four shifts seem important for embedding call-by-need.

2 A SOURCE SEQUENT CALCULUS WITH USER-DEFINED TYPES

Our source language for representing user-defined data and co-data types is based on the classical sequent calculus. In contrast to the λ-calculus, this setting lets us simultaneously model both ML- and Haskell-like languages within one framework, and to express additional types (like $A \parr B$ and $-A$ in section 3) with pleasant symmetric properties (like the algebraic and logical laws in section 6) that are generally not found in λ-based languages.

[3] All the proofs for theorems stated in these sections are given in appendices A to D.

$$x \in \mathit{Variable} ::= \ldots \qquad \alpha \in \mathit{CoVariable} ::= \ldots \qquad \mathsf{K} \in \mathit{Constructor} ::= \ldots \qquad \mathsf{O} \in \mathit{Observer} ::= \ldots$$

$$c \in \mathit{Command} ::= \langle v \| e \rangle$$

$$v \in \mathit{Term} ::= x \mid \mu\alpha.c \mid \mathsf{K}(\vec{e}, \vec{v}) \mid \mu\left(\overrightarrow{\mathsf{O}[\vec{x}, \vec{\alpha}].c}\right) \qquad e \in \mathit{CoTerm} ::= \alpha \mid \tilde{\mu}x.c \mid \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha}, \vec{x}).c}\right] \mid \mathsf{O}[\vec{v}, \vec{e}]$$

Fig. 1. An untyped language of (co-)data in the sequent calculus.

The untyped syntax of our sequent calculus language is given in fig. 1, and is based on the calculus by Downen and Ariola [2014]. This syntax is coarsely divided into three categories which correspond to three different roles in a program: *terms* v which produce results, *co-terms* e which consume results, and *commands* c which run. Terms and co-terms are symmetric reflections of one another, and commands are formed by linking a term v and co-term e, written $\langle v \| e \rangle$, so that the output of v is fed as input to e, or in other words, so that e can observe the result of v. There are two generic (co-)terms—μ- and $\tilde{\mu}$-abstractions—which name their partner before running another command: the term $\mu\alpha.c$ names its output α while running c, and the co-term $\tilde{\mu}x.c$ names its input x while running c. We also have more specific (co-)terms in the form of data and co-data that orient the constructive and destructive forces of computation, corresponding to algebraic data types (ADTs) from functional languages and a form of objects or functional abstractions, respectively. Data represents constructive production and destructive consumption: a data structure is built by collecting several other (co-)terms with a constructor, $\mathsf{K}(e_1, \ldots, e_m, v_n, \ldots, v_1)$, which is deconstructed by a pattern-matching co-term, $\tilde{\mu}[\mathsf{K}(\alpha_1, \ldots, \alpha_m, x_n, \ldots, x_1).c \ldots]$, that responds based on the shape of its input, listing cases for the possible constructions it might receive while giving a name to their constituent parts. Co-data represents destructive production and constructive consumption: a co-data observation is built by collecting several other (co-)terms with an observer, $\mathsf{O}[v_n, \ldots, v_1, e_1, \ldots, e_m]$, which is deconstructed by a pattern-matching term, $\mu(\mathsf{O}[x_n, \ldots, x_1, \alpha_1, \ldots, \alpha_m].c \ldots)$, that responds based on the shape of its output, listing cases for the possible observations that might be made of it while giving a name to their constituent parts.

The type system shown in fig. 2 for carving out well-behaved programs takes the form of an annotated sequent calculus.[5] In particular, there are three different forms of judgements for assigning types to programs: a term producing an A-output is typed by the sequent $\Gamma \vdash_{\mathcal{G}} v : A \mid \Delta$, a co-term consuming an A-input is typed by the sequent $\Gamma \mid e : A \vdash_{\mathcal{G}} \Delta$, command (which neither produces nor consumes) is typed by the sequent $c : (\Gamma \vdash_{\mathcal{G}} \Delta)$. In each case, A is the *active type* of the sequent, denoting the primary input or output of an expression as appropriate, Γ is an *input environment* assigning types to free variables, Δ is an *output environment* assigning types to free co-variables, and $\mathcal{G}$ holds the *global environment* declaring the meaning of type and (co-)term constructors. We sometimes omit explicitly naming $\mathcal{G}$ when it is implicit from context. The core typing rules [Curien and Herbelin 2000] correspond to the core rules of Gentzen's [1935] sequent calculus: the *VR* and *VL* rules for typing free (co-)variables correspond to the initial sequent, and the *Cut* rule for forming commands is the ordinary cut. In addition, we have the activation rules *AR* and *AL* for typing μ- and $\tilde{\mu}$-abstractions, which turns a passive type assigned to a free (co-)variable of a command into the active type of the abstraction.

The main interest in the type system, however, is the way that it (1) tracks different evaluation strategies within a program, and (2) models arbitrary user-defined data and co-data types. For the

[4]This is just shorthand for asserting that a (co-)data declaration of $\mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'$ is in $\mathcal{G}$.

[5]These rules differ from the LK sequent calculus because they treat the structural rules of weakening, contraction, and exchange implicitly rather than explicitly. We use this presentation only for the simplicity of reducing the number of rules, since the matter of explicit or implicit structural rules, while potentially of interest, is orthogonal to the main topic here.

$$X \in \mathit{TypeVar} ::= \ldots \qquad \mathsf{F}, \mathsf{G} \in \mathit{TypeCon} ::= \ldots$$

$$\mathcal{S}, \mathcal{T}, \mathcal{U} \in \mathit{BaseKind} ::= \mathcal{V} \mid \mathcal{N} \mid \mathcal{LV} \mid \mathcal{LN} \qquad A, B, C \in \mathit{Type} ::= X \mid \mathsf{F}(\overrightarrow{A})$$

$$\mathit{decl} \in \mathit{Declaration} ::= \mathbf{data}\ \mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{K} : \left(\overrightarrow{A : \mathcal{T}} \vdash \mathsf{F}(\overrightarrow{X}) \mid \overrightarrow{B : \mathcal{U}}\right)}$$

$$\mid \mathbf{codata}\ \mathsf{G}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{O} : \left(\overrightarrow{A : \mathcal{T}} \mid \mathsf{G}(\overrightarrow{X}) \vdash \overrightarrow{B : \mathcal{U}}\right)}$$

$$\mathcal{G} \in \mathit{GlobalEnv} ::= \overrightarrow{\mathit{decl}} \qquad \Theta \in \mathit{TypeEnv} ::= \overrightarrow{X : \mathcal{S}} \qquad \Gamma \in \mathit{InputEnv} ::= \overrightarrow{x : A} \qquad \Delta \in \mathit{OutputEnv} ::= \overrightarrow{\alpha : A}$$

$$\mathit{Judgement} ::= \Theta \vdash_{\mathcal{G}} A : \mathcal{S} \mid c : (\Gamma \vdash_{\mathcal{G}} \Delta) \mid (\Gamma \vdash_{\mathcal{G}} v : A \mid \Delta) \mid (\Gamma \mid e : A \vdash_{\mathcal{G}} \Delta)$$

Type kinding rules

$$\frac{}{\Theta, X : \mathcal{S} \vdash_{\mathcal{G}} X : \mathcal{S}}\ VT \qquad \frac{(\mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}')^{4} \in \mathcal{G} \quad \overrightarrow{\Theta \vdash_{\mathcal{G}} A : \mathcal{S}}}{\Theta \vdash_{\mathcal{G}} \mathsf{F}(\overrightarrow{A}) : \mathcal{S}'}\ \mathsf{F}T$$

Core typing rules

$$\frac{\Gamma \vdash_{\mathcal{G}} v : A \mid \Delta \quad \vdash_{\mathcal{G}} A : \mathcal{S} \quad \Gamma \mid e : A \vdash_{\mathcal{G}} \Delta}{\langle v \| e \rangle : (\Gamma \vdash_{\mathcal{G}} \Delta)}\ Cut$$

$$\frac{}{\Gamma, x : A \vdash_{\mathcal{G}} x : A \mid \Delta}\ VR \qquad \frac{c : (\Gamma \vdash_{\mathcal{G}} \alpha : A, \Delta)}{\Gamma \vdash_{\mathcal{G}} \mu\alpha.c : A \mid \Delta}\ AR \qquad \frac{c : (\Gamma, x : A \vdash_{\mathcal{G}} \Delta)}{\Gamma \mid \tilde{\mu}x.c : A \vdash_{\mathcal{G}} \Delta}\ AL \qquad \frac{}{\Gamma \mid \alpha : A \vdash_{\mathcal{G}} \alpha : A, \Delta}\ VL$$

(Co-)Data typing rules

Given $\mathbf{data}\ \mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{K}_i : \left(\overrightarrow{A_{ij}}^{j} \vdash \mathsf{F}(\overrightarrow{X}) \mid \overrightarrow{B_{ij}}^{j}\right)}^{i} \in \mathcal{G}$, then we have the following rules:

$$\frac{\overrightarrow{\Gamma \mid e : B_{ij}\{\overrightarrow{C/X}\} \vdash_{\mathcal{G}} \Delta}^{j} \quad \overrightarrow{\Gamma \vdash_{\mathcal{G}} v : A_{ij}\{\overrightarrow{C/X}\} \mid \Delta}^{j}}{\Gamma \vdash_{\mathcal{G}} \mathsf{K}_i(\overrightarrow{e}, \overrightarrow{v}) : \mathsf{F}(\overrightarrow{C}) \mid \Delta}\ \mathsf{F}R_{\mathsf{K}_i} \qquad \frac{\overrightarrow{c_i : \left(\Gamma, \overrightarrow{x_i : A_i\{\overrightarrow{C/X}\}} \vdash_{\mathcal{G}} \overrightarrow{\alpha_i : B_i\{\overrightarrow{C/X}\}}, \Delta\right)}^{i}}{\Gamma \mid \tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\overrightarrow{\alpha_i}, \overrightarrow{x_i}).c_i}^{i}\right] : \mathsf{F}(\overrightarrow{C}) \vdash_{\mathcal{G}} \Delta}\ \mathsf{F}L$$

Given $\mathbf{codata}\ \mathsf{G}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{O}_i : \left(\overrightarrow{A_{ij}}^{j} \mid \mathsf{G}(\overrightarrow{X}) \vdash \overrightarrow{B_{ij}}^{j}\right)}^{i} \in \mathcal{G}$, then we have the following rules:

$$\frac{\overrightarrow{c_i : \left(\Gamma, \overrightarrow{x_i : A_i\{\overrightarrow{C/X}\}} \vdash_{\mathcal{G}} \overrightarrow{\alpha_i : B_i\{\overrightarrow{C/X}\}}, \Delta\right)}^{i}}{\Gamma \vdash_{\mathcal{G}} \mu\left(\overrightarrow{\mathsf{O}_i[\overrightarrow{x_i}, \overrightarrow{\alpha_i}].c_i}^{i}\right) : \mathsf{G}(\overrightarrow{C}) \mid \Delta}\ GR \qquad \frac{\overrightarrow{\Gamma \vdash_{\mathcal{G}} v : A_{ij}\{\overrightarrow{C/X}\} \mid \Delta}^{j} \quad \overrightarrow{\Gamma \mid e : B_{ij}\{\overrightarrow{C/X}\} \vdash_{\mathcal{G}} \Delta}^{j}}{\Gamma \mid \mathsf{O}_i[\overrightarrow{v}, \overrightarrow{e}] : \mathsf{G}(\overrightarrow{C}) \vdash_{\mathcal{G}} \Delta}\ \mathsf{GL}_{\mathsf{O}_i}$$

Fig. 2. A multi-kinded type system for (co-)data in the sequent calculus.

first purpose, we divide types into *four* different basic kinds: $\mathcal{V}$ for call-by-value constructs, $\mathcal{N}$ for call-by-name constructs, $\mathcal{LV}$ for "lazy call-by-value" (*a.k.a* call-by-need) constructs [Ariola et al. 1995], and $\mathcal{LN}$ for "lazy call-by-name" (*i.e.,* the dual of call-by-need) [Ariola et al. 2011]. In other words, we use kinds for denoting the evaluation strategy of typed programs, so that (co-)terms $v : A : \mathcal{V}$ and $e : A : \mathcal{V}$ are evaluated under a call-by-value strategy, $v : A : \mathcal{N}$ and $e : A : \mathcal{N}$ are evaluated according to call-by-name, and likewise for $\mathcal{LV}$ and $\mathcal{LN}$. If we want to refer to programs involving call-by-value and call-by-name evaluation, then we can restrict the base kinds down to only $\mathcal{V}$ and $\mathcal{N}$, which we call the $\mathcal{VN}$ sub-calculus. For the second purpose, we include a mechanism for declaring new (co-)data types which enriches the type system with new rules for (co-)terms of the declared type, corresponding to the logical left and right rules of the sequent calculus. For example, to model pairs from functional languages, we can declare the following

pair type for types of kind $\mathcal{S}$—instantiating $\mathcal{S}$ with $\mathcal{V}$ for ML-like pairs and with $\mathcal{N}$ or $\mathcal{LV}$ for Haskell-like pairs—as follows to get the associated left and right rules for pairs:

$$\mathbf{data}\,(X : \mathcal{S}) \times_{\mathcal{S}} (Y : \mathcal{S}) : \mathcal{S}\ \mathbf{where}\ \mathsf{Pair}_{\mathcal{S}} : (X : \mathcal{S}, Y : \mathcal{S} \vdash X \times_{\mathcal{S}} Y \mid)$$

$$\frac{\Gamma \vdash_{\mathcal{G}} v_1 : A \mid \Delta \quad \Gamma \vdash_{\mathcal{G}} v_2 : B \mid \Delta}{\Gamma \vdash_{\mathcal{G}} \mathsf{Pair}_{\mathcal{S}}(v_1, v_2) : A \times_{\mathcal{S}} B \mid \Delta} \times_{\mathcal{S}} R_{\mathsf{Pair}_{\mathcal{S}}} \qquad \frac{c : (\Gamma, x_1 : A, x_2 : B \vdash_{\mathcal{G}} \Delta)}{\Gamma \mid \tilde{\mu}[\mathsf{Pair}_{\mathcal{S}}(x_1, x_2).c] : A \times_{\mathcal{S}} B \vdash_{\mathcal{G}} \Delta} \times_{\mathcal{S}} L$$

For this data type, the structure $\mathsf{Pair}_{\mathcal{S}}(v_1, v_2)$ is just like a pair from the respective functional language, and the usual case-analysis expression can be written as: $\mathbf{case}\ v\ \mathbf{of}\ \mathsf{Pair}_{\mathcal{S}}(x_1, x_2) \Rightarrow v' \triangleq \mu\alpha.\langle v \| \tilde{\mu}[\mathsf{Pair}_{\mathcal{S}}(x_1, x_2).\langle v \| \alpha\rangle]\rangle$. As another example, function types do not need to be primitives in this language, since they can be declared as co-data types. In particular, we have both eager and lazy functions by again instantiating the right kind for $\mathcal{S}$—picking $\mathcal{V}$ for eager functions and $\mathcal{N}$ or $\mathcal{LV}$ for lazy ones—to get the associated left and right rules for functions:

$$\mathbf{codata}\,(X : \mathcal{S}) \to_{\mathcal{S}} (Y : \mathcal{S}) : \mathcal{S}\ \mathbf{where}\ \mathsf{Call}_{\mathcal{S}} : (X : \mathcal{S} \mid X \to_{\mathcal{S}} Y \vdash Y : \mathcal{S})$$

$$\frac{c : (\Gamma, x : A \vdash_{\mathcal{G}} \beta : B, \Delta)}{\Gamma \vdash_{\mathcal{G}} \mu(\mathsf{Call}_{\mathcal{S}}[x, \beta].c) : A \to_{\mathcal{S}} B \mid \Delta} \to_{\mathcal{S}} R \qquad \frac{\Gamma \vdash_{\mathcal{G}} v : A \mid \Delta \quad \Gamma \mid e : B \vdash_{\mathcal{G}} \Delta}{\Gamma \mid \mathsf{Call}_{\mathcal{S}}[v, e] : A \to_{\mathcal{S}} B \vdash_{\mathcal{G}} \Delta} \to_{\mathcal{S}} L$$

The familiar λ-calculus notation for function abstraction and application is written as: $\lambda x.v \triangleq \mu(\mathsf{Call}_{\mathcal{S}}[x, \beta].\langle v \| \beta\rangle)$ and $v\ v' \triangleq \mu\beta.\langle v \| \mathsf{Call}_{\mathcal{S}}[v', \beta]\rangle$. In general, we say that a (co-)data declaration is *well-formed* with respect to some other declarations $\mathcal{G}$ if all the types are of the claimed kind, as stated formally by the following rules:

$$\frac{\overrightarrow{\overrightarrow{\Theta \vdash_{\mathcal{G}} A_{ij} : \mathcal{T}_{ij}}^{j}}^{i} \quad \overrightarrow{\overrightarrow{\Theta \vdash_{\mathcal{G}} B_{ij} : \mathcal{U}_{ij}}^{j}}^{i}}{\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{S}\ \mathbf{where}} \mathsf{F}F \qquad \frac{\overrightarrow{\overrightarrow{\Theta \vdash_{\mathcal{G}} A_{ij} : \mathcal{T}_{ij}}^{j}}^{i} \quad \overrightarrow{\overrightarrow{\Theta \vdash_{\mathcal{G}} B_{ij} : \mathcal{U}_{ij}}^{j}}^{i}}{\mathbf{codata}\ \mathsf{G}(\Theta) : \mathcal{S}\ \mathbf{where}} \mathsf{G}F$$

$$\mathcal{G} \vdash \overrightarrow{\mathsf{K}_i : \left(\overrightarrow{A_{ij} : \mathcal{T}_{ij}}^{j} \vdash \mathsf{F}(\Theta) \mid \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^{j}\right)}^{i} \qquad \mathcal{G} \vdash \overrightarrow{\mathsf{O}_i : \left(\overrightarrow{A_{ij} : \mathcal{T}_{ij}}^{j} \mid \mathsf{G}(\Theta) \vdash \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^{j}\right)}^{i}$$

Downen et al. [2015, 2016] goes into more formal detail into how the data and co-data features in the sequent calculus correspond to analogous features from functional languages, including translation to and from the λ-calculus.

Finally, we give an equational theory in fig. 3 for reasoning about the operational behavior of programs based on the theory by Downen and Ariola [2014]. The essence of evaluation strategy is captured by the restrictions on the $\tilde{\mu}$ and μ rules, which may only substitute *values* for variables and *co-values* for co-variables. This avoids the known unfortunate dilemma of computation that all commands are equal if substitution is unrestricted, since we would have $c =_{\mu} \langle \mu\_.c \| \tilde{\mu}\_.c' \rangle =_{\tilde{\mu}} c'$. As described by Curien and Herbelin [2000], there are (at least) two different disciplines for substitution: the *call-by-value* discipline where the substitutable values exclude μ-abstractions (as in $\mathit{Value}_{\mathcal{V}}$) but everything is a co-value (as in $\mathit{CoValue}_{\mathcal{V}}$), and the *call-by-name* discipline where the substitutable co-values exclude $\tilde{\mu}$-abstractions (as in $\mathit{CoValue}_{\mathcal{N}}$) but everything is a value (as in $\mathit{Value}_{\mathcal{N}}$).

We extend beyond this binary choice to also include two further restricted notions of (co-)values that add memoization to the basic $\mathcal{V}$ and $\mathcal{N}$ as formalized previously by Downen and Ariola [2014]. The *lazy call-by-value*, *a.k.a call-by-need*, $\mathcal{LV}$ discipline only substitutes call-by-value values but also restricts co-values to only those co-terms that "need" to their input to continue (as shown in $\mathit{CoValue}_{\mathcal{LV}}$): each call-by-name co-value needs their input, but so do $\tilde{\mu}$-abstractions that observe their input variable with another $\mathcal{LV}$-co-value. Dually, the *lazy call-by-name* $\mathcal{LN}$ discipline flips the priorities around to only substitute call-by-name co-values while also restricting values to only those terms that are "eager" to produce their output (as shown in $\mathit{Value}_{\mathcal{LN}}$): each call-by-value value immediately produces a result, but so do $\tilde{\mu}$-abstractions that pass another $\mathcal{LN}$ value to their output co-variable. The demand for input or output in both of the $\mathcal{LV}$ and $\mathcal{LN}$ may occur within

$$V \in \mathit{Value} ::= V_{\mathcal{V}} : A : \mathcal{V} \mid V_{\mathcal{N}} : A : \mathcal{N} \mid V_{\mathcal{LV}} : A : \mathcal{LV} \mid V_{\mathcal{LN}} : A : \mathcal{LN}$$
$$E \in \mathit{CoValue} ::= E_{\mathcal{V}} : A : \mathcal{V} \mid E_{\mathcal{N}} : A : \mathcal{N} \mid E_{\mathcal{LV}} : A : \mathcal{LV} \mid E_{\mathcal{LN}} : A : \mathcal{LN}$$
$$V_{\mathcal{V}} \in \mathit{Value}_{\mathcal{V}} ::= x \mid \mathsf{K}(\vec{E}, \vec{V}) \mid \mu\left(\overrightarrow{\mathsf{O}[\vec{x}, \vec{\alpha}].c}\right) \qquad E_{\mathcal{V}} \in \mathit{CoValue}_{\mathcal{V}} ::= e$$
$$V_{\mathcal{N}} \in \mathit{Value}_{\mathcal{N}} ::= v \qquad E_{\mathcal{N}} \in \mathit{CoValue}_{\mathcal{N}} ::= \alpha \mid \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha}, \vec{x}).c}\right] \mid \mathsf{O}[\vec{V}, \vec{E}]$$
$$V_{\mathcal{LV}} \in \mathit{Value}_{\mathcal{LV}} ::= V_{\mathcal{V}} \qquad E_{\mathcal{LV}} \in \mathit{CoValue}_{\mathcal{LV}} ::= E_{\mathcal{N}} \mid \tilde{\mu}x.D\left[\langle x \| E_{\mathcal{LV}} \rangle\right]$$
$$V_{\mathcal{LN}} ::= V_{\mathcal{V}} \mid \mu\alpha.D\left[\langle V_{\mathcal{LN}} \| \alpha \rangle\right] \qquad E_{\mathcal{LN}} \in \mathit{CoValue}_{\mathcal{LN}} ::= E_{\mathcal{N}}$$
$$D \in \mathit{DelayedCxt} ::= \Box \mid \langle v : A : \mathcal{LV} \| \tilde{\mu}x{:}A{:}\mathcal{LV}.D \rangle \mid \langle \mu\alpha{:}A{:}\mathcal{LN}.D \| e : A : \mathcal{LN} \rangle$$

Core substitution axioms

$$(\mu) \quad \langle \mu\alpha.c \| E \rangle = c\,\{E/\alpha\} \qquad (\eta_\mu) \quad \mu\alpha.\langle v \| \alpha \rangle = v \qquad (\alpha \notin FV(v))$$
$$(\tilde{\mu}) \quad \langle V \| \tilde{\mu}x.c \rangle = c\,\{V/x\} \qquad (\eta_{\tilde{\mu}}) \quad \tilde{\mu}x.\langle x \| e \rangle = e \qquad (x \notin FV(e))$$

(Co-)Data $\beta\eta$ axioms

Given **data** $\mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'$ **where** $\overrightarrow{\mathsf{K}_i : \left(\vec{A}_i \vdash \mathsf{F}(\vec{X}) \mid \vec{B}_i\right)^i} \in \mathcal{G}$, then we have the following rules:

$$(\beta^{\mathsf{F}}) \quad \left\langle \mathsf{K}_i(\vec{e}, \vec{v}) \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\vec{\alpha_i}, \vec{x_i}).c_i}^i\right] \right\rangle = \langle \mu\vec{\alpha_i}.\langle \vec{v} \| \tilde{\mu}\vec{x_i}.c_i \rangle \| \vec{e} \rangle$$
$$(\eta^{\mathsf{F}}) \quad \beta : \mathsf{F}(\vec{C}) = \tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\vec{\alpha_i}, \vec{x_i}).\langle \mathsf{K}_i(\vec{\alpha_i}, \vec{x_i}) \| \beta \rangle}^i\right]$$

Given **codata** $\mathsf{G}(\overrightarrow{X : \mathcal{S}}) : \mathcal{S}'$ **where** $\overrightarrow{\mathsf{O}_i : \left(\vec{A}_i \mid \mathsf{G}(\vec{X}) \vdash \vec{B}_i\right)^i} \in \mathcal{G}$, then we have the following rules:

$$(\beta^{\mathsf{G}}) \quad \left\langle \mu\left(\overrightarrow{\mathsf{O}_i[\vec{x_i}, \vec{\alpha_i}].c_i}^i\right) \middle\| \mathsf{O}_i[\vec{v}, \vec{e}] \right\rangle = \langle \vec{v} \| \tilde{\mu}\vec{x_i}.\langle \mu\vec{\alpha_i}.c_i \| \vec{e} \rangle \rangle$$
$$(\eta^{\mathsf{G}}) \quad y : \mathsf{G}(\vec{C}) = \mu\left(\overrightarrow{\mathsf{O}_i[\vec{x_i}, \vec{\alpha_i}].\langle y \| \mathsf{O}_i[\vec{x_i}, \vec{\alpha_i}] \rangle}^i\right)$$

Fig. 3. A multi-discipline equational theory for the sequent calculus.

the context of a series of other delayed $\mathcal{LV}$ and $\mathcal{LN}$ bindings represented by a context D. We can then safely merge together these four notions of (co-)values, since we can always use the kind of a type a (co-)term inhabits to determine whether or not is a (co-)value of the appropriate discipline,[6] and we sometimes explicitly write $\mu_{\mathcal{V}}$ and $\tilde{\mu}_{\mathcal{V}}$ when substituting a $\mathcal{V}$ (co-)value and $\mu_{\mathcal{N}}$ and $\tilde{\mu}_{\mathcal{N}}$ when substituting a $\mathcal{N}$ (co-)value, and so on, for clarity.

The four different disciplines of substitution represent four different possible evaluation orders for running programs, where the priority for determining which side of a command is in charge is reflected in the possible μ and $\tilde{\mu}$ substitutions. For example, consider a generic typed command of the form $\langle \mu\alpha.c_1 \| \tilde{\mu}x.c_2 \rangle$, so that the next step of the command is determined by the kind of A in the interaction between $\mu\alpha.c_1 : A$ and $\tilde{\mu}x.c_2 : A$. When $A : \mathcal{V}$, the μ rule is able to substitute $\tilde{\mu}x.c_2$ for α in c_1 since the $\tilde{\mu}$-abstraction is a $\mathcal{V}$-co-value, and thus the producer $\mu\alpha.c_1$ has priority to take control of the program. Dually when $A : \mathcal{N}$, the $\tilde{\mu}$ rule is able to substitute $\mu\alpha.c_1$ for x in c_2 since the μ-abstraction is a $\mathcal{N}$-value, and thus the consumer $\tilde{\mu}x.c_2$ has priority. When $A : \mathcal{LV}$, the priorities can shift back and forth as neither side of the command is substitutable, which represents call-by-need by being demand-driven (by prioritizing the consumer first) and memoizing (by only duplicating simple values and not complex computations). Instead, we must delay the

[6] As discussed previously by Downen and Ariola [2014], full static typing is not necessary for deciding which (co-)terms are (co-)values, since a more coarse-grained distinction can be made with either a bi-typing system [Zeilberger 2009] or multiple syntactic categories [Munch-Maccagnoni and Scherer 2015]. However, here we use the kinds of types to decide the (co-)value discipline to reduce the overhead of additional rules or syntax, since static typing gives the necessary distinctions.

binding and work within c_2 giving it first priority; if it happens that c_2 evaluates to a command of the form $D[\langle x \| E_{\mathcal{LV}} \rangle]$ that needs to observe x, then $\tilde{\mu}x.c_2 = \tilde{\mu}x.D[\langle x \| E_{\mathcal{LV}} \rangle]$ has become a $\mathcal{LV}$-co-value which is subject to substitution, thereby switching priority to c_1. When $A : \mathcal{LN}$, the priorities are the other way around and are production-driven (by prioritizing the producer first) and co-memoizing (by only duplicating simple observations and not complex continuations). We must delay the binding and work within c_1 giving it first priority; if it happens that c_1 evaluates to a command of the form $D[\langle V_{\mathcal{LN}} \| \alpha \rangle]$ that needs return to α, then $\mu\alpha.c_1 = \mu\alpha.D[\langle V_{\mathcal{LN}} \| \alpha \rangle]$ has become a $\mathcal{LN}$-value which is subject to substitution, thereby switching priority to c_2.

The essence of (co-)data types is captured by the β and η rules, which are analogues of the rules of the same name from the λ-calculus. Notice that these rules do not depend on the choice of substitution discipline because they are not affected by the (co-)value restrictions. The β rules for both data and co-data types use pattern matching to break apart structures and observations, selecting the appropriate response and binding the constituent parts with μ- and $\tilde{\mu}$-abstractions, where we make use of the following shorthand notation for a sequence of bindings:

$$\langle v_1, \ldots, v_n \| \tilde{\mu}x_1, \ldots, x_n.c \rangle \triangleq \langle v_1 \| \tilde{\mu}x_1. \ldots \langle v_n \| \tilde{\mu}x_n.c \rangle \rangle$$
$$\langle \mu\alpha_1, \ldots, \alpha_n.c \| e_1, \ldots, e_n \rangle \triangleq \langle \mu\alpha_1. \ldots \langle \mu\alpha_n.c \| e_n \rangle \| e_1 \rangle$$

The η rules for both data and co-data types expand a (co-)variable of the type into the pattern-matching construct which breaks down its given structure or observation into all possible cases and then reconstitutes a fresh copy to forward to that (co-)variable.

3 A TARGET SEQUENT CALCULUS WITH THE POLAR BASIS

Our target language—which we will use for encoding all the constructs from the source—is not really a different language at all. Rather, it is a limited subset of the source language, consisting of only a fixed, finite number of different constructs. The idea is to declare just a handful of (co-)data types up front, collectively named $\mathcal{P}$, and then forget the declaration mechanism entirely to prevent the language from being extended with any new types, so that we can think of the target language as a calculus inductively defined with the types built from $\mathcal{P}$. The key, then, is to ensure that all the programs from the source language can be *faithfully* encoded into the limited constructs included in the target, without running into the same troubles of unfaithful encodings from section 1.

The brunt of our pre-defined, primitive data and co-data types is given in fig. 4. Each of these (co-)data types are chosen for their symmetry—for each one, there is a dual mirror image on the other side—and because they all reflect one, and only one, aspect of the functionality allowed by the (co-)data declaration mechanism. The additive (co-)data types reflect the use of multiple different constructors or observers for a type by giving a choice between two ($\oplus$ and $\&$) or a choice of no (0 and $\top$) alternatives. The multiplicative (co-)data types reflect the use of multiple components within structures or observations, by giving a combination of two ($\otimes$ and $\parr$) or no (1 and $\bot$) parts. And finally, we have the negation (co-)data types, which reflect the ability for data structures to contain co-terms and co-data observations to contain terms. The typing rules for (co-)data types are shown in fig. 5, which are derived from the general form from fig. 2.[7]

[7]Readers familiar with Girard's [1987] linear logic will no doubt notice that our additive and multiplicative (co-)data types in fig. 4 are named after the linear logic connectives. And yet the derived typing rules in fig. 5 are not the rules of linear logic because the $\otimes R$ and $\parr R$ rules should join different environments from both premises rather than share them, and the $1R$ and $\bot R$ rules should force the environments to be empty. This discrepancy can be easily fixed by changing the general rules for (co-)data types so that rules for data structures and co-data observations join separate environments used to type each of their parts, as done by Munch-Maccagnoni [2009]. The cost, however, is that this presentation needs explicit structural rules, which we had sought to avoid in section 2 for the purpose of simplicity.

Additive (co-)data types

$$\textbf{data}\,(X:\mathcal{V})\oplus(Y:\mathcal{V}):\mathcal{V}\,\textbf{where} \qquad \textbf{codata}\,(X:\mathcal{N})\,\&\,(Y:\mathcal{N}):\mathcal{N}\,\textbf{where}$$
$$\iota_1:(X:\mathcal{V}\vdash X\oplus Y\mid) \qquad \pi_1:(\mid X\,\&\,Y\vdash X:\mathcal{N})$$
$$\iota_2:(Y:\mathcal{V}\vdash X\oplus Y\mid) \qquad \pi_2:(\mid X\,\&\,Y\vdash Y:\mathcal{N})$$

$$\textbf{data}\,0:\mathcal{V}\,\textbf{where} \qquad \textbf{codata}\,\top:\mathcal{N}\,\textbf{where}$$

Multiplicative (co-)data types

$$\textbf{data}\,(X:\mathcal{V})\otimes(Y:\mathcal{V}):\mathcal{V}\,\textbf{where} \qquad \textbf{codata}\,(X:\mathcal{N})\,\parr\,(Y:\mathcal{N}):\mathcal{N}\,\textbf{where}$$
$$(\_,\_):(X:\mathcal{V},Y:\mathcal{V}\vdash X\otimes Y\mid) \qquad [\_,\_]:(\mid X\parr Y\vdash X:\mathcal{N},Y:\mathcal{N})$$

$$\textbf{data}\,1:\mathcal{V}\,\textbf{where}\;():(\vdash 1\mid) \qquad \textbf{codata}\,\bot:\mathcal{N}\,\textbf{where}\;[]:(\mid\bot\vdash)$$

Involutive negation (co-)data types

$$\textbf{data}\,-(X:\mathcal{N}):\mathcal{V}\,\textbf{where} \qquad \textbf{codata}\,\neg(X:\mathcal{V}):\mathcal{N}\,\textbf{where}$$
$$-:(\vdash -X\mid X:\mathcal{N}) \qquad \neg:(X:\mathcal{V}\mid\neg X\vdash)$$

Fig. 4. Declarations of the primitive polarized data and co-data types.

Additive typing rules

$$\frac{\Gamma\vdash_{\mathcal{P}} v:A_i\mid\Delta}{\Gamma\vdash_{\mathcal{P}}\iota_i\,(v):A_1\oplus A_2\mid\Delta}\oplus R_i\quad i=1,2 \qquad \frac{\Gamma\mid e:A_i\vdash_{\mathcal{P}}\Delta}{\Gamma\mid\pi_2\,[e]:A_1\,\&\,A_2\vdash_{\mathcal{P}}\Delta}\,\&L_i\quad i=1,2$$

$$\frac{c_1:(\Gamma,x:A\vdash_{\mathcal{P}}\Delta)\quad c_2:(\Gamma,y:B\vdash_{\mathcal{P}}\Delta)}{\Gamma\mid\tilde{\mu}[\iota_1\,(x).c_1\mid\iota_2\,(y)\,c_2]:A\oplus B\vdash_{\mathcal{P}}\Delta}\oplus L \qquad \frac{c_1:(\Gamma\vdash_{\mathcal{P}}\alpha:A,\Delta)\quad c_2:(\Gamma\vdash_{\mathcal{P}}\beta:B,\Delta)}{\Gamma\vdash_{\mathcal{P}}\mu(\pi_1\,[\alpha].c_1\mid\pi_2\,[\beta].c_2):A\,\&\,B\mid\Delta}\,\&R$$

no $0R$ rule $\qquad \dfrac{}{\Gamma\mid\tilde{\mu}[]:0\vdash_{\mathcal{P}}\Delta}\,0L \qquad \dfrac{}{\Gamma\vdash_{\mathcal{P}}\mu():\top\mid\Delta}\,\top R$ $\qquad$ no $\top L$ rule

Multiplicative typing rules

$$\frac{\Gamma\vdash_{\mathcal{P}} v_1:A\mid\Delta\quad\Gamma\vdash_{\mathcal{P}} v_2:B\mid\Delta}{\Gamma\vdash_{\mathcal{P}}(v_1,v_2):A\otimes B\mid\Delta}\otimes R \qquad \frac{\Gamma\mid e_1:A\vdash_{\mathcal{P}}\Delta\quad\Gamma\mid e_2:B\vdash_{\mathcal{P}}\Delta}{\Gamma\mid[e_1,e_2]:A\parr B\vdash_{\mathcal{P}}\Delta}\,\parr L$$

$$\frac{c:(\Gamma,x:A,y:B\vdash_{\mathcal{P}}\Delta)}{\Gamma\mid\tilde{\mu}[(x,y).c]:A\otimes B\vdash_{\mathcal{P}}\Delta}\otimes L \qquad \frac{c:(\Gamma\vdash_{\mathcal{P}}\alpha:A,\beta:B,\Delta)}{\Gamma\vdash_{\mathcal{P}}\mu([\alpha,\beta].c):A\parr B\mid\Delta}\,\parr R$$

$$\frac{}{\Gamma\vdash_{\mathcal{P}}():1\mid\Delta}\,1R \qquad \frac{c:(\Gamma\vdash_{\mathcal{P}}\Delta)}{\Gamma\mid\tilde{\mu}[().c]:1\vdash_{\mathcal{P}}\Delta}\,1L \qquad \frac{c:(\Gamma\vdash_{\mathcal{P}}\Delta)}{\Gamma\vdash_{\mathcal{P}}\mu([].c):\bot\mid\Delta}\,\bot R \qquad \frac{}{\Gamma\mid[]:\bot\vdash_{\mathcal{P}}\Delta}\,\bot L$$

Involutive negation typing rules

$$\frac{\Gamma\mid e:A\vdash_{\mathcal{P}}\Delta}{\Gamma\vdash_{\mathcal{P}}-(e):-A\mid\Delta}-R \qquad \frac{c:(\Gamma\vdash_{\mathcal{P}}\alpha:A,\Delta)}{\Gamma\mid\tilde{\mu}[-(\alpha).c]:-A\vdash_{\mathcal{P}}\Delta}-L \qquad \frac{c:(\Gamma,x:A\vdash_{\mathcal{P}}\Delta)}{\Gamma\vdash_{\mathcal{P}}\mu(\neg[x].c):\neg A\mid\Delta}\,\neg R \qquad \frac{\Gamma\vdash_{\mathcal{P}} v:A\mid\Delta}{\Gamma\mid\neg[v]:\neg A\vdash_{\mathcal{P}}\Delta}\,\neg L$$

Fig. 5. Derived typing rules for the primitive polarized data and co-data types.

We might think that we have some flexibility in choosing the kinds of types—denoting the substitution discipline—involved in the declarations in fig. 4. But as it turns out, since we want to use these (co-)data types as the backbone of faithful encodings, our hand is forced. Intuitively, each of these declarations follows a simple rule of thumb for choosing the kinds for types: every type to the left (of $\vdash$) is $\mathcal{V}$ and every type to the right is $\mathcal{N}$, except for the active type whose kind is the reverse. This rule of thumb has a few consequences. The first is that every data type is call-by-value and every co-data type is call-by-name, which follows the general wisdom of

$$\textbf{data}\downarrow_{\mathcal{S}}(X:\mathcal{S}):\mathcal{V}\ \textbf{where}\downarrow_{\mathcal{S}}:(X:\mathcal{S}\vdash\downarrow_{\mathcal{S}}X\mid)\qquad \textbf{codata}\uparrow_{\mathcal{S}}(X:\mathcal{S}):\mathcal{N}\ \textbf{where}\uparrow_{\mathcal{S}}:(\mid\uparrow_{\mathcal{S}}X\vdash X:\mathcal{S})$$

$$\textbf{data}\ {}_{\mathcal{S}}\Uparrow(X:\mathcal{V}):\mathcal{S}\ \textbf{where}\ {}_{\mathcal{S}}\Uparrow:(X:\mathcal{V}\vdash{}_{\mathcal{S}}\Uparrow X\mid)\qquad \textbf{codata}\ {}_{\mathcal{S}}\Downarrow(X:\mathcal{N}):\mathcal{S}\ \textbf{where}\ {}_{\mathcal{S}}\Downarrow:(\mid{}_{\mathcal{S}}\Downarrow X\vdash X:\mathcal{N})$$

Fig. 6. Declarations of the shifts between disciplines (*i.e.,* base kinds) as data and co-data types.

polarization in computation [Munch-Maccagnoni 2013; Zeilberger 2009]. The second consequence is that every data type constructor builds on $\mathcal{V}$ types and every co-data type constructor builds on $\mathcal{N}$ types, except for the negation constructors which are reversed because their underlying (co-)terms are reversed. The last consequence is that the notion of data type values and co-data type co-values are hereditarily as restrictive as possible, where a structure or observation is only a (co-)value if it contains components that are (co-)values in the most restrictive sense.

The basic (co-)data types from fig. 4 are still incomplete, though, for our purpose of encoding *all* (co-)data types expressible in the source language. In particular, how could we possibly represent a type like the call-by-name pair $A \times_{\mathcal{N}} B$ from section 2? The $\otimes$ data type constructor won't do since it operates over the wrong kind of types. Even worse, how can we represent types that make use of the two memoizing kinds $\mathcal{LV}$ and $\mathcal{LN}$? To address these issues, we need a mechanism for plainly "shifting" between the different base kinds of types, and to do that we must break our rule of thumb. For the moment, let's consider only conversions between $\mathcal{V}$ and $\mathcal{N}$. One way to do the conversion is with singleton (co-)data types, declared as follows, that *wraps* a component of the another kind:

$$\textbf{data}\downarrow(X:\mathcal{N}):\mathcal{V}\ \textbf{where}\downarrow:(X:\mathcal{N}\vdash\downarrow X\mid)\qquad \textbf{codata}\uparrow(X:\mathcal{V}):\mathcal{N}\ \textbf{where}\uparrow:(\mid\uparrow X\vdash X:\mathcal{V})$$

The other possibility is a singleton (co-)data type that *is* of another kind, declared as follows:

$$\textbf{codata}\Downarrow(X:\mathcal{N}):\mathcal{V}\ \textbf{where}\Downarrow:(\mid\Downarrow X\vdash X:\mathcal{N})\qquad \textbf{data}\Uparrow(X:\mathcal{V}):\mathcal{N}\ \textbf{where}\Uparrow:(X:\mathcal{V}\vdash\Uparrow X\mid)$$

As it turns out, we will use both styles of shifts because they are each useful in different situations for encoding complex (co-)data types. And in the more general case where we have all four different base kinds, we will rely on both the ability to shift *into* the canonical $\mathcal{V}$ and $\mathcal{N}$ kinds and then *out* again. As a technical device, we will use a family of shifts parameterized by a base kind as defined in fig. 6, with the above as defaults when a kind is unspecified. The idea is that $\downarrow_{\mathcal{S}}$ and $\uparrow_{\mathcal{S}}$ shift *to* $\mathcal{V}$ and $\mathcal{N}$ (respectively) from $\mathcal{S}$, whereas ${}_{\mathcal{S}}\Uparrow$ and ${}_{\mathcal{S}}\Downarrow$ shift *from* $\mathcal{V}$ and $\mathcal{N}$ (respectively) to $\mathcal{S}$. More explicitly, this means defining a quadruple of shift connectives for each of the four base kinds in the language. These parameterized shifts include some redundancy (as we will see in section 6.3), but they are useful notationally for generically manipulating types, and also accomodate the addition of more evaluation modes like call-by-need that go beyond basic call-by-value and call-by-name.

By combining the polarized types from fig. 4 with the shifts from fig. 6, we get the polarized basis $\mathcal{P}$ for all user-defined (co-)data types. In particular, the polarized basis is expressive enough to translate programs using any collection $\mathcal{G}$ of user-defined (co-)data types as shown in fig. 7, so that if $c : (\Gamma \vdash_{\mathcal{G}} \Delta)$ then $[\![c]\!]_{\mathcal{G}} : ([\![\Gamma]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![\Delta]\!]_{\mathcal{G}})$ (where $[\![\Gamma]\!]_{\mathcal{G}}$ and $[\![\Delta]\!]_{\mathcal{G}}$ are defined pointwise). We informally use deep pattern matching to aid writing the translation, with the understanding that it is desugared into several shallow patterns in the obvious way, and to express the repeated composition of the binary connectives, we define the ("big") versions of the additive and multiplicative polarized connectives over n-ary vectors of types as follows:

$$\bigoplus \epsilon \triangleq 0 \qquad \bigoplus(A,\vec{B}) \triangleq A \oplus \left(\bigoplus \vec{B}\right) \qquad \bigotimes \epsilon \triangleq 1 \qquad \bigotimes(A,\vec{B}) \triangleq A \otimes \left(\bigotimes \vec{B}\right)$$

$$\& \epsilon \triangleq \top \qquad \&(A,\vec{B}) \triangleq A \,\&\, \left(\& \vec{B}\right) \qquad \parr \epsilon \triangleq \bot \qquad \parr(A,\vec{B}) \triangleq A \parr \left(\parr \vec{B}\right)$$

$$[\![X]\!]_{\mathcal{G}} \triangleq X \quad [\![x]\!]_{\mathcal{G}} \triangleq x \quad [\![\mu\alpha.c]\!]_{\mathcal{G}} \triangleq \mu\alpha.[\![c]\!]_{\mathcal{G}} \quad [\![\langle v \| e\rangle]\!]_{\mathcal{G}} \triangleq \langle [\![v]\!]_{\mathcal{G}} \| [\![e]\!]_{\mathcal{G}}\rangle \quad [\![\tilde{\mu}x.c]\!]_{\mathcal{G}} \triangleq \tilde{\mu}x.[\![c]\!]_{\mathcal{G}} \quad [\![\alpha]\!]_{\mathcal{G}} \triangleq \alpha$$

Given **data** $\mathsf{F}(\Theta) : \mathcal{S}$ **where** $\overrightarrow{\mathsf{K}_i : \left(\overrightarrow{A_{i1} : \mathcal{T}_{ij}}^{j} \vdash \mathsf{F}(\Theta) \mid \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^{j}\right)}^{i} \in \mathcal{G}$:

$$[\![\mathsf{F}(\vec{C})]\!]_{\mathcal{G}} \triangleq \mathcal{S}\Uparrow\left(\bigoplus\left(\overrightarrow{\left(\bigotimes\left(\overrightarrow{-(\uparrow_{\mathcal{U}_{ij}}[\![B_{ij}]\!]_{\mathcal{G}}\theta)}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}[\![A_{ij}]\!]_{\mathcal{G}}\theta}^{j}\right)\right)}^{i}\right)\right) \qquad \left(\theta = \overrightarrow{\left\{[\![C]\!]_{\mathcal{G}}/X\right\}}\right)$$

$$[\![\mathsf{K}_i(\overrightarrow{e_{ij}}^{j}, \overrightarrow{v_{ij}}^{j})]\!]_{\mathcal{G}} \triangleq \mathcal{S}\Uparrow(\iota_i(\overrightarrow{-(\uparrow_{\mathcal{U}_{ij}}[[\![e_{ij}]\!]_{\mathcal{G}}])}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}([\![v_{ij}]\!]_{\mathcal{G}})}^{j}))$$

$$\left[\!\left[\tilde{\mu}[\mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}).c_i]\right]\!\right]_{\mathcal{G}} \triangleq \overrightarrow{\tilde{\mu}[\mathcal{S}\Uparrow(\iota_i(\overrightarrow{-(\uparrow_{\mathcal{U}_{ij}}[\alpha_{ij}])}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j})).[\![c_i]\!]_{\mathcal{G}}]}^{i}$$

Given **codata** $\mathsf{G}(\Theta) : \mathcal{S}$ **where** $\overrightarrow{\mathsf{O}_i : \left(\overrightarrow{A_{ij} : \mathcal{T}_{ij}}^{j} \mid \mathsf{G}(\Theta) \vdash \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^{j}\right)}^{i} \in \mathcal{G}$:

$$[\![\mathsf{G}(\vec{C})]\!]_{\mathcal{G}} \triangleq \mathcal{S}\Downarrow\left(\&\left(\overrightarrow{\left(\overrightarrow{\mathcal{P}\left(\overrightarrow{\neg(\downarrow_{\mathcal{T}_{ij}}[\![A_{ij}]\!]_{\mathcal{G}}\theta)}^{j}, \overrightarrow{\uparrow_{\mathcal{U}_{ij}}[\![B_{ij}]\!]_{\mathcal{G}}\theta}^{j}\right)}^{i}\right)\right) \qquad \left(\theta = \overrightarrow{\left\{[\![C]\!]_{\mathcal{G}}/X\right\}}\right)$$

$$[\![\mathsf{O}_i[\overrightarrow{v_{ij}}^{j}, \overrightarrow{e_{ij}}^{j}]]\!]_{\mathcal{G}} \triangleq \mathcal{S}\Downarrow[\pi_i[\overrightarrow{\neg(\downarrow_{\mathcal{T}_{ij}}([\![v_{ij}]\!]_{\mathcal{G}}))}^{j}, \overrightarrow{\uparrow_{\mathcal{U}_{i1}}[[\![e_{ij}]\!]_{\mathcal{G}}]}^{j}]]$$

$$\left[\!\left[\mu(\overrightarrow{\mathsf{O}_i[\overrightarrow{x_{ij}}^{j}, \overrightarrow{\alpha_{ij}}^{j}].c_i})^{i}\right]\!\right]_{\mathcal{G}} \triangleq \mu(\overrightarrow{\mathcal{S}\Downarrow[\pi_i[\neg[\downarrow_{\mathcal{T}_{ij}}(x_{ij})]^{j}, \uparrow_{\mathcal{T}_{ij}}[\alpha_{ij}]^{j}]].[\![c_i]\!]_{\mathcal{G}}}^{i})$$

Fig. 7. A polarizing translation from the source language into the target $\mathcal{P}$.

$$\iota_i(v) \triangleq \iota_2\left(.^{i}.\iota_1(v)\right) \qquad (v_n, \ldots, v_1) \triangleq (v_n, (\ldots, (v_1, ())))$$

$$\pi_i[e] \triangleq \pi_2\left[.^{i}.\pi_1[e]\right] \qquad [e_1, \ldots, e_n] \triangleq [e_1, [\ldots, [e_n, []]]]$$

This encoding is *sound* in that equations in the source, including η, are preserved in the target.

Theorem 3.1 (Polarization soundness). *For $i = 1, 2$,*

a) if $c_i : (\Gamma \vdash_{\mathcal{G}} \Delta)$ and $c_1 = c_2$ then $[\![c_i]\!]_{\mathcal{G}} : ([\![\Gamma]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![\Delta]\!]_{\mathcal{G}})$ and $[\![c_1]\!]_{\mathcal{G}} = [\![c_2]\!]_{\mathcal{G}}$,
b) if $\Gamma \vdash_{\mathcal{G}} v_i : A \mid \Delta$ and $v_1 = v_2$ then $[\![\Gamma]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![v_i]\!]_{\mathcal{G}} : [\![A]\!]_{\mathcal{G}} \mid [\![\Delta]\!]_{\mathcal{G}}$ and $[\![v_1]\!]_{\mathcal{G}} = [\![v_2]\!]_{\mathcal{G}}$, and
c) if $\Gamma \mid e_i : A \vdash_{\mathcal{G}} \Delta$ and $e_1 = e_2$ then $[\![\Gamma]\!]_{\mathcal{G}} \mid [\![e_i]\!]_{\mathcal{G}} : [\![A]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![\Delta]\!]_{\mathcal{G}}$ and $[\![e_1]\!]_{\mathcal{G}} = [\![e_2]\!]_{\mathcal{G}}$, and

But is the converse statement of completeness—that if the encodings of two commands or (co-)terms are equal then they are equal to begin with—also true? Unfortunately not so directly; the polarizing encoding has the effect of "anonymizing" types by moving away from a nominal style, where the different declarations lead to distinct types, to a more structural style, where differently declared types can be collapsed if they share a common underlying pattern. This collapse of types doesn't mean that all hope is lost, however, because the nominally distinct (co-)terms are only collapsed *between* types not *within* types; there is still a one-for-one correspondence between typed (co-)terms of the same type in the source with the encoded (co-)terms in the target. To argue this case, we turn to applying the idea of *isomorphisms* between types [Di Cosmo 1995].

4 WHAT IS AN ISOMORPHISM BETWEEN TYPES?

Usually, we can say that two types are isomorphic when there are mappings to and from both of them whose composition is an identity mapping. In the setting of the sequent calculus, we interpret "mappings" as open commands with a free variable and co-variable, and the "identity" mapping is the simple command $\langle x \| \alpha\rangle$ connecting its free (co-)variables.

Definition 4.1 (Type isomorphism). Two closed types A and B are *isomorphic*, written $A \approx B$, if and only if there exist commands $c : (x : A \vdash \beta : B)$ and $c' : (y : B \vdash \alpha : A)$ for any x, y, α, β such that the following equalities hold:

$$\langle \mu\beta.c \| \tilde{\mu}y.c' \rangle = \langle x \| \alpha \rangle : (x : A \vdash \alpha : A) \qquad \langle \mu\alpha.c' \| \tilde{\mu}x.c \rangle = \langle y \| \beta \rangle : (y : B \vdash \beta : B)$$

Moreover, two open types A and B with free type variables $\overrightarrow{X : \mathcal{S}}$ are *isomorphic*, written as $\overrightarrow{X : \mathcal{S}} \vDash A \approx B$, if and only if for all types $\overrightarrow{C : \mathcal{S}}$, it follows that $A\{\overrightarrow{C/X}\} \approx B\{\overrightarrow{C/X}\}$.

Note that this definition of isomorphism between types is equivalent to a more traditional presentation in terms of inverse functions within the language. In particular, two types $A : \mathcal{S}$ and $B : \mathcal{S}$ are isomorphic in the sense of definition 4.1 if and only if there are two closed function values $V : A \to_{\mathcal{S}} B$ and $V' : B \to_{\mathcal{S}} A$ such that $V' \circ V = id : A \to_{\mathcal{S}} A$ and $V \circ V' = id : B \to_{\mathcal{S}} B$, because we can always abstract over the open commands to get closed functions, or call the functions to retrieve open commands, and one form is inverse whenever the other is. However, definition 4.1 has the advantage of not assuming that our language has a primitive function type (since they are just user-defined co-data types like any other), and of avoiding the awkwardness of mapping between the different kinds of types that might be isomorphic to one another.

But do type isomorphisms give us the right sense of a one-for-one correspondence between (co-)values of those types? As it turns out, an isomorphism $A \approx B$ provides just enough structure to convert all equalities between A-typed (co-)values to B-typed (co-)values. Note that this conversion is a compositional mapping *within* the language, which makes it a potential syntactic transformation for marshaling between two different but isomorphic interfaces in the context of a compiler.

Theorem 4.1. *For any isomorphism $A \approx B$ and environments Γ and Δ, there are contexts C and C' such that if $\Gamma \vdash_{\mathcal{G}} V_i : A \mid \Delta$ and $\Gamma \vdash_{\mathcal{G}} E_i : A \mid \Delta$ (for $i = 1, 2$), then $\Gamma \vdash_{\mathcal{G}} C[V_i] : B \mid \Delta$ and $\Gamma \vdash_{\mathcal{G}} C'[E_i] : B \mid \Delta$, $v_1 = v_2$ if and only if $C[V_1] = C[V_2]$, and $E_1 = E_2$ if and only if $C'[E_1] = C'[E_2]$.*

Proof. Let $c : (x : A \vdash \beta : B)$ and $c' : (y : B \vdash \alpha : A)$ witnesses the isomorphism $A \approx B$, where $x, y \notin \Gamma$ and $\alpha, \beta \notin \Delta$. The desired contexts are then $C \triangleq \mu\beta.\langle \square \| \tilde{\mu}x.c \rangle$ and $C' \triangleq \tilde{\mu}y.\langle \mu\alpha.c' \| \square \rangle$. $C[V_1] = C[V_2]$ follows from $V_1 = V_2$ and $C'[E_1] = C'[E_2]$ follows from $E_1 = E_2$ by just applying the assumed equalities within the context C and C'. More interestingly, we can derive $V_1 = V_2$ from $C[V_1] = C[V_2]$ from the definition of the isomorphism by placing them in an even larger context where we have the equality:

$$\begin{aligned} \mu\alpha.\langle C[V_i] \| \tilde{\mu}y.c' \rangle &\triangleq \mu\alpha.\langle \mu\beta.\langle V_i \| \tilde{\mu}x.c \rangle \| \tilde{\mu}y.c' \rangle =_{\tilde{\mu}} \mu\alpha.\langle V_i \| \tilde{\mu}x.\langle \mu\beta.c \| \tilde{\mu}y.c \rangle \rangle \\ &=_{Iso} \mu\alpha.\langle V_i \| \tilde{\mu}x.\langle x \| \alpha \rangle \rangle =_{\eta_\mu \eta_{\tilde{\mu}}} V_i \end{aligned}$$

And since $C[V_1] = C[V_2]$, we have $V_1 = \mu\alpha.\langle C[V_1] \| \tilde{\mu}y.c' \rangle = \mu\alpha.\langle C[v_2] \| \tilde{\mu}y.c' \rangle = V_2$. $E_1 = E_2$ follows from $C'[E_1] = C'[E_2]$ similarly because of the fact that $\tilde{\mu}x.\langle \mu\beta.c \| C[E_i] \rangle = E_i$. □

Having defined isomorphisms between types, we should ask if they actually form an equivalence relation as expected; are type isomorphisms closed under reflexivity, symmetry, and transitivity? The reflexivity and symmetry of the isomorphism relation between types is rather straightforward.

Theorem 4.2 (Reflexivity and Symmetry). *(a) $A \approx A$, and (b) if $A \approx B$ then $B \approx A$.*

Proof. The symmetry of type isomorphism follows immediately from its symmetric definition. More interestingly, we can establish the reflexive isomorphism of any type with the extensionality laws of μ- and $\tilde{\mu}$-abstractions. In particular, for a given A, we have the command $\langle x \| \alpha \rangle : (x : A \vdash \alpha : A)$ which serves as both open commands of the isomorphism $A \approx A$. The fact that the self-composition of this command is equal to itself comes from the η_μ and $\eta_{\tilde{\mu}}$ axioms: $\langle \mu\alpha.\langle x \| \alpha \rangle \| \tilde{\mu}x.\langle x \| \alpha \rangle \rangle =_{\eta_\mu} \langle x \| \tilde{\mu}x.\langle x \| \alpha \rangle \rangle =_{\eta_{\tilde{\mu}}} \langle x \| \alpha \rangle$. □

In contrast, transitivity of type isomorphisms is tricker, and in fact it is not guaranteed to hold in every possible situation. In particular, the transitivity of isomorphism relies on the exchange of μ- and $\tilde{\mu}$-bindings, which reassociates the composition of commands, but this not always valid in the multi-discipline scenario. Specifically, given two any two kinds of typed (co-)terms, $v : A : \mathcal{S}$ and $e : B : \mathcal{T}$, the kind-sensitive *exchange* law $\chi_{\mathcal{S}\vdash\mathcal{T}}$ is:

$$(\chi_{\mathcal{S}\vdash\mathcal{T}}) \quad \langle v \| \tilde{\mu}x{:}A{:}\mathcal{S}.\, \langle \mu\alpha{:}B{:}\mathcal{T}.\, c \| e\rangle\rangle = \langle \mu\alpha{:}B{:}\mathcal{T}.\, \langle v \| \tilde{\mu}x{:}A{:}\mathcal{S}.\, c\rangle \| e\rangle \quad (x \notin FV(e), \alpha \notin FV(v))$$

And when exchanging bindings of the same kind $\mathcal{S}$, we just write $\chi_{\mathcal{S}}$ for $\chi_{\mathcal{S}\vdash\mathcal{S}}$. This exchange law can be used to re-associate the binding structure of a program and in turn justify the transitive composition of type isomorphisms. However, exchange is not valid for some kind combinations. For any $\mathcal{S}$, $\chi_{\mathcal{N}\vdash\mathcal{S}}$ is derivable from the universal strength of the $\tilde{\mu}_{\mathcal{N}}$ axiom and likewise $\chi_{\mathcal{S}\vdash\mathcal{V}}$ is derivable from the strong $\mu_{\mathcal{V}}$ axiom. So for all combinations of $\mathcal{N}$ and $\mathcal{V}$, each of $\chi_{\mathcal{N}\vdash\mathcal{N}}$, $\chi_{\mathcal{V}\vdash\mathcal{V}}$, and $\chi_{\mathcal{N}\vdash\mathcal{V}}$ hold, but $\chi_{\mathcal{V}\vdash\mathcal{N}}$ is invalidated by the following counter example:[8]

$$\langle \mu\_{:}A{:}\mathcal{V}.\, c_1 \| \tilde{\mu}x{:}A{:}\mathcal{V}.\, \langle \mu\alpha{:}B{:}\mathcal{N}.\, c \| \tilde{\mu}\_{:}B{:}\mathcal{N}.\, c_2\rangle\rangle =_{\mu_{\mathcal{V}}} c_1 \neq c_2 =_{\tilde{\mu}_{\mathcal{N}}} \langle \mu\alpha{:}B{:}\mathcal{N}.\, \langle \mu\_{:}A{:}\mathcal{V}.\, c_1 \| \tilde{\mu}x{:}A{:}\mathcal{V}.\, c\rangle \| \tilde{\mu}\_{:}B{:}\mathcal{N}.\, c_2\rangle$$

And when we encounter lazy bindings for types of kind $\mathcal{LV}$ or $\mathcal{LN}$, which perform memoization with delayed computations, both μ and $\tilde{\mu}$ substitution laws have been restricted so we can no longer lean on their universal strength for justifying $\chi_{\mathcal{LV}}$ and $\chi_{\mathcal{LN}}$.

Transitivity is not only important for the purpose of saying that type isomorphism is an equivalence relation; just like composition in programming, it is a key source of compositionality that lets us assemble little isomorphisms together incrementally to get a big result. So if we can't rely on always having $\chi_{\mathcal{S}\vdash\mathcal{T}}$ for any combination of $\mathcal{S}$ and $\mathcal{T}$, what can we do instead to ensure transitivity? As it turns out, the specific witnesses maps of the isomorphisms we're interested in here have special properties themselves that ensure that we can exchange bindings as in the χ law. In particular, some terms behave similarly to values (but are not necessarily values) and are *thunkable* in the production of their output whereas some co-terms behave similarly to co-values and are *linear* on the use of their input, which allows for the same sorts of binding re-associations.[9]

Definition 4.2 (Thunkability and Linearity). A term v is *thunkable* if and only if for all e, c, $x \notin FV(e)$, and $\alpha \notin FV(v)$, $\langle v \| \tilde{\mu}x.\, \langle \mu\alpha.c \| e\rangle\rangle = \langle \mu\alpha.\, \langle v \| \tilde{\mu}x.c\rangle \| e\rangle$. Dually, a co-term e is *linear* if and only if for all v, c, $x \notin FV(e)$, and $\alpha \notin FV(v)$, $\langle \mu\alpha.\, \langle v \| \tilde{\mu}x.c\rangle \| e\rangle = \langle v \| \tilde{\mu}x.\, \langle \mu\alpha.c \| e\rangle\rangle$.

We can then use the idea of thunkability and linearity—a semantic generalization of the syntactic idea of values and co-values—to restrict the sorts of maps that are allowed as witnesses of type isomorphisms thereby strengthening their compositionality. This gives us a polarized view of type isomorphisms, where *positive* isomorphisms are built on linear maps and *negative* isomorphisms are built on thunkable maps.

Definition 4.3 (Polarized type isomorphism). Two closed types A and B are *positively isomorphic*, written $A \approx^{+} B$, if and only if there is an isomorphism $A \approx B$ witnessed by the commands $c : (x : A \vdash \beta : B)$ and $c' : (y : B \vdash \alpha : A)$ such that $\tilde{\mu}x.c$ and $\tilde{\mu}y.c'$ are linear. Dually, A and B are *negatively isomorphic*, written $A \approx^{-} B$, if and only if there is an isomorphism $A \approx B$ witnessed by the commands $c : (x : A \vdash \beta : B)$ and $c' : (y : B \vdash \alpha : A)$ such that $\mu\alpha.c'$ and $\mu\beta.c$ are thunkable. Moreover, we write $\overrightarrow{X : \mathcal{S}} \vDash A \approx^{+} B$ and $\overrightarrow{X : \mathcal{S}} \vDash A \approx^{-} B$ to mean for all $\overrightarrow{C : \mathcal{S}}$, $A\{\overrightarrow{C/X}\} \approx^{+} B\{\overrightarrow{C/X}\}$ and $A\{\overrightarrow{C/X}\} \approx^{-} B\{\overrightarrow{C/X}\}$, respectively.

Note that, while all isomorphisms between two $\mathcal{V}$-kinded types are positive and all isomorphisms between two $\mathcal{N}$-kinded types are negative, the definition of $A \approx^{+} B$ and $A \approx^{-} B$ does not refer to a

[8] Invalidity of $\chi_{\mathcal{V}\vdash\mathcal{N}}$ corresponds to the loss of associativity in categorical models of polarity [Munch-Maccagnoni 2013].
[9] This is a syntactic rephrasing and generalization (beyond just combination call-by-value and call-by-name) of thunkable and linear morphisms in duploids defined in terms of their associativity properties [Munch-Maccagnoni 2013].

prescribed $\mathcal{V}$ or $\mathcal{N}$ discipline, but rather about the behavior of particular programs independent of their discipline. This polar view of type isomorphism *does* give us a computation-based equivalence relation on types regardless of the particular discipline involved.

Theorem 4.3 (Polarized Isomorphism Equivalence). *a)* $A \approx^+ A$ *and* $A \approx^- A$*,*
b) if $A \approx^+ B$ *then* $B \approx^+ A$ *and if* $A \approx^- B$ *then* $B \approx^- A$*, and*
c) if $A \approx^+ B$ *and* $B \approx^+ C$ *then* $A \approx^+ C$ *and if* $A \approx^- B$ *and* $B \approx^- C$ *then* $A \approx^- C$*.*

Finally, we extend the idea of type isomorphisms to (co-)data declaration isomorphisms.

Definition 4.4 (Declaration isomorphism). Two data declarations are *isomorphic*, written[10]

$$\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{S}\ \mathbf{where}\ \overrightarrow{\mathsf{K} : (\Gamma \vdash \mathsf{F}(\Theta) \mid \Delta)} \approx \mathbf{data}\ \mathsf{F}'(\Theta) : \mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{K}' : (\Gamma \vdash \mathsf{F}'(\Theta) \mid \Delta)}$$

if and only if $\Theta \vDash \mathsf{F}(\Theta) \approx \mathsf{F}'(\Theta)$. Dually, we say that two co-data declarations are *isomorphic*, written

$$\mathbf{codata}\ \mathsf{G}(\Theta) : \mathcal{S}\ \mathbf{where}\ \overrightarrow{\mathsf{O} : (\Gamma \mid \mathsf{G}(\Theta) \vdash \Delta)} \approx \mathbf{codata}\ \mathsf{G}'(\Theta) : \mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{O}' : (\Gamma' \mid \mathsf{G}'(\Theta) \vdash \Delta')}$$

if and only if $\Theta \vDash \mathsf{G}(\Theta) \approx \mathsf{G}'(\Theta)$. *Positive* and *negative* declaration isomorphisms are defined similarly in terms of positive and negative type isomorphisms, respectively.

Theorem 4.4 (Polarized declaration isomorphism equivalence). *The positive and negative (co-)data declaration isomorphism relations are both (a) reflexive, (b) symmetric, and (c) transitive.*

Proof. The closure properties follow from the type isomorphism equivalence relation (theorem 4.3) underlying definition 4.4. □

This more specific notion of type-based isomorphism is the backbone of the syntactic theory that we will develop for the purpose of reasoning more easily about (co-)data types in general, the polarized basis $\mathcal{P}$ of (co-)data types, and eventually the faithfulness of the polarization translation.

5 A SYNTACTIC THEORY OF (CO-)DATA TYPE ISOMORPHISMS

Before turning to our main result—that every user-defined (co-)data type can be represented by an isomorphic type composed solely from the polarized basic connectives—we first explore a theory for type isomorphisms based on data and co-data declarations. The advantage of focusing on (co-)data type declarations for studying type isomorphisms is that the declarations themselves provide a larger context for localized manipulations surrounded by extra alternatives (of other constructors and observers) and extra components (within the same constructor or observer). The end result is that we only need to manually verify a few fundamental (co-)data type isomorphisms by hand, while the particular isomorphisms of interest can be easily composed out of basic building blocks.

5.1 Structural laws of declarations

We present an theory for the structural laws of data and co-data type isomorphisms in figs. 8 and 9, which are exactly dual to one another and capture several facts about isomorphic ways to declare (co-)data types.

- *Commute*: The first group of laws state that the parts of any declaration may be reordered, including (1) the order of components within the signature of a constructor or observer, and (2) the order of constructor or observer alternatives within the declaration. These axioms are useful to show that the listed orders of the various parts of a declaration don't matter.

[10] We reuse Γ and Δ as shorthand for the list of types $\overrightarrow{A : \mathcal{T}}$ and $\overrightarrow{B : \mathcal{U}}$ within the signatures of constructors and observers.

- *Mix*: The second group of laws states how two isomorphisms between (co-)data type declarations may be combined together. In particular, there are two ways to mix declaration isomorphisms: (1) an isomorphic pair of single-alternative declarations can have the components of their single constructor or observer mixed into the signature of all the alternatives of another declaration isomorphism, and (2) a pair of declaration isomorphism can have their respective alternatives mixed together. These inference rules let us use localized reasoning within a small (co-)data type declaration, and then compose the results together into a large declaration isomorphism that does everything all at once.
- *Shift*: The third group of laws state that every (call-by-value) $\mathcal{V}$ data declaration isomorphism and every (call-by-name) $\mathcal{N}$ co-data declaration isomorphism may be generalized to (co-)data types of any kind $\mathcal{S}$.
- *Interchange*: The fourth group of laws show how isomorphisms between data type declarations and co-data type declarations can be interchanged one-for-one with one another, so long as the data type is call-by-value ($\mathcal{V}$) and the co-data type is call-by-name ($\mathcal{N}$).
- *Compatibility*: The final group of laws state that certain isomorphisms between types can be lifted into an isomorphism between data and co-data type declarations with constructors and observations containing a component of that type as either an input or an output.

These laws let us derive other facts about isomorphisms between (co-)data types. As an example, applying the shift laws to the trivial cases of the commute laws for data declarations lets us rename constructor and type names, telling us there is only one empty and unit type for any kind $\mathcal{S}$:

$$\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{S}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}(\Theta) \mid) \approx \mathbf{data}\ \mathsf{F}'(\Theta) : \mathcal{S}\ \mathbf{where}\ \mathsf{K}' : (\vdash \mathsf{F}'(\Theta) \mid)$$

$$\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{S}\ \mathbf{where}\ \approx \mathbf{data}\ \mathsf{F}'(\Theta) : \mathcal{S}\ \mathbf{where}$$

Additionally, the mix laws let us extend an existing isomorphism by combining it with a reflexive isomorphism of any declaration, letting us add on arbitrary other alternatives or components to two isomorphic data declarations:

$$\frac{\mathbf{data}\ \mathsf{F}_1(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_1 : (\Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1)} \quad \approx \quad \mathbf{data}\ \mathsf{F}'_1(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'_1 : (\Gamma'_1 \vdash \mathsf{F}'(\Theta) \mid \Delta'_1)} \qquad \overline{\mathbf{data}\ \mathsf{F}_2(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_2 : (\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)} \quad \approx \quad \mathbf{data}\ \mathsf{F}_2(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_2 : (\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)}}}{\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_1 : (\Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1)}\ \overrightarrow{\mathsf{K}_2 : (\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)} \approx \mathbf{data}\ \mathsf{F}'(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'_1 : (\Gamma'_1 \vdash \mathsf{F}'(\Theta) \mid \Delta'_1)}\ \overrightarrow{\mathsf{K}_2 : (\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)}}$$

$$\frac{\mathbf{data}\ \mathsf{F}_1(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_1 : (\Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1)} \quad \approx \quad \mathbf{data}\ \mathsf{F}'_1(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'_1 : (\Gamma'_1 \vdash \mathsf{F}'(\Theta) \mid \Delta'_1)} \qquad \overline{\mathbf{data}\ \mathsf{F}_2(\Theta) : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_2 : (\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2) \quad \approx \quad \mathbf{data}\ \mathsf{F}_2(\Theta) : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_2 : (\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)}}{\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_1 : (\Gamma_2, \Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1, \Delta_1)} \approx \mathbf{data}\ \mathsf{F}'(\Theta) : \mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'_1 : (\Gamma_2, \Gamma'_1 \vdash \mathsf{F}'(\Theta) \mid \Delta'_1, \Delta_2)}}$$

We can justify the laws in figs. 8 and 9 in terms of the definitions of type and (co-)data declaration isomorphisms. In particular, we can calculate when specific instances of two (co-)data types happen to be isomorphic, so that the laws for declaration isomorphisms are sound when the specific instances hold in general for any matching choice of types.[11] Each specific isomorphism instance justifies the soundness of the proposed structural laws for (co-)data declarations.

Theorem 5.1 (Structural laws). *The declaration isomorphism laws in figs. 8 and 9 are all sound.*

There is one more property about (co-)data declarations that will be useful in the following sections: certain singleton (co-)data types are just trivial wrappers around another type. In the right

[11] The statement and proofs of these facts can be found in appendix C.

Data Commute

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}{:}(\Gamma_2, \Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1, \Delta_2) \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}'{:}(\Gamma_1, \Gamma_2 \vdash \mathsf{F}'(\Theta) \mid \Delta_2, \Delta_1)$$

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}_1{:}(\Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1)}\ \overrightarrow{\mathsf{K}_2{:}(\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)} \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}'_2{:}(\Gamma_2 \vdash \mathsf{F}'(\Theta) \mid \Delta_2)}\ \overrightarrow{\mathsf{K}'_1{:}(\Gamma_1 \vdash \mathsf{F}'(\Theta) \mid \Delta_1)}$$

Data Mix

$$\frac{\mathbf{data}\ \mathsf{F}_1(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}_1{:}(\Gamma_1 \vdash \mathsf{F}_1(\Theta) \mid \Delta_1)} \approx \mathbf{data}\ \mathsf{F}'_1(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}'_1{:}(\Gamma'_1 \vdash \mathsf{F}'_1(\Theta) \mid \Delta'_1)} \qquad \mathbf{data}\ \mathsf{F}_2(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}_2{:}(\Gamma_2 \vdash \mathsf{F}_2(\Theta) \mid \Delta_2) \approx \mathbf{data}\ \mathsf{F}'_2(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}'_2{:}(\Gamma'_2 \vdash \mathsf{F}'_2(\Theta) \mid \Delta'_2)}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}{:}(\Gamma_2, \Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1, \Delta_2)} \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'{:}(\Gamma'_2, \Gamma'_1 \vdash \mathsf{F}'(\Theta) \mid \Delta'_1, \Delta'_2)}}$$

$$\frac{\mathbf{data}\ \mathsf{F}_1(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}_1{:}(\Gamma_1 \vdash \mathsf{F}_1(\Theta) \mid \Delta_1)} \approx \mathbf{data}\ \mathsf{F}'_1(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}'_1{:}(\Gamma'_1 \vdash \mathsf{F}'_1(\Theta) \mid \Delta'_1)} \qquad \mathbf{data}\ \mathsf{F}_2(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}_2{:}(\Gamma_2 \vdash \mathsf{F}_2(\Theta) \mid \Delta_2)} \approx \mathbf{data}\ \mathsf{F}'_2(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \overrightarrow{\mathsf{K}'_2{:}(\Gamma'_2 \vdash \mathsf{F}'_2(\Theta) \mid \Delta'_2)}}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}_1{:}(\Gamma_1 \vdash \mathsf{F}(\Theta) \mid \Delta_1)}\ \overrightarrow{\mathsf{K}_2{:}(\Gamma_2 \vdash \mathsf{F}(\Theta) \mid \Delta_2)} \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'_1{:}(\Gamma'_1 \vdash \mathsf{F}'(\Theta) \mid \Delta'_1)}\ \overrightarrow{\mathsf{K}'_2{:}(\Gamma'_2 \vdash \mathsf{F}'(\Theta) \mid \Delta'_2)}}$$

Data Shift

$$\frac{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}{:}(\Gamma \vdash \mathsf{F}(\Theta) \mid \Delta)} \approx \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'{:}(\Gamma' \vdash \mathsf{F}'(\Theta) \mid \Delta')}}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{S}\ \mathbf{where}\ \overrightarrow{\mathsf{K}{:}(\Gamma \vdash \mathsf{F}(\Theta) \mid \Delta)} \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{S}'\ \mathbf{where}\ \overrightarrow{\mathsf{K}'{:}(\Gamma' \vdash \mathsf{F}'(\Theta) \mid \Delta')}}$$

Co-data-Data Interchange

$$\frac{\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{N}\ \mathbf{where}\ \overrightarrow{\mathsf{O}{:}(\Gamma \mid \mathsf{G}(\vec{X}) \vdash \Delta)} \approx \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \overrightarrow{\mathsf{O}'{:}(\Gamma' \mid \mathsf{G}'(\vec{X'}) \vdash \Delta')}}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}{:}(\Gamma \vdash \mathsf{F}(\Theta) \mid \Delta)} \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \overrightarrow{\mathsf{K}'{:}(\Gamma' \vdash \mathsf{F}'(\vec{X'}) \mid \Delta')}}$$

Data Compatibility

$$\frac{\Theta \vdash A{:}\mathcal{V} \quad \Theta \vdash B{:}\mathcal{V} \quad \Theta \vDash A \approx B}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}{:}(A{:}\mathcal{V} \vdash \mathsf{F}(\Theta) \mid) \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}'{:}(B{:}\mathcal{V} \vdash \mathsf{F}'(\Theta) \mid)} \qquad \frac{\Theta \vDash A \approx^- B}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}{:}(A{:}\mathcal{S} \vdash \mathsf{F}(\Theta) \mid) \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}'{:}(B{:}\mathcal{S} \vdash \mathsf{F}'(\Theta) \mid)}$$

$$\frac{\Theta \vdash A{:}\mathcal{N} \quad \Theta \vdash B{:}\mathcal{N} \quad \Theta \vDash A \approx B}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}{:}(\ \vdash \mathsf{F}(\Theta) \mid A{:}\mathcal{N}) \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}'{:}(\ \vdash \mathsf{F}'(\Theta) \mid B{:}\mathcal{N})} \qquad \frac{\Theta \vDash A \approx^+ B}{\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}{:}(\ \vdash \mathsf{F}(\Theta) \mid A{:}\mathcal{S}) \approx^+ \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\quad \mathsf{K}'{:}(\ \vdash \mathsf{F}'(\Theta) \mid B{:}\mathcal{S})}$$

Fig. 8. A theory for structural laws of data type declaration isomorphisms.

circumstances, these wrappers can be identified with their underlying types, up to isomorphism, which lets us connect the world of (co-)data declarations with the world of actual types.

Lemma 5.1 ((Co-)Data identity). *a) For any* $\mathbf{data}\ \mathsf{F}(\Theta) : \mathcal{U}\ \mathbf{where}\ \mathsf{K} : (A : \mathcal{T} \vdash \mathsf{F}(\Theta) \mid)$, *if* $\mathcal{T} = \mathcal{V}$ *then* $\Theta \vDash \mathsf{F}(\Theta) \approx^+ A$ *and if* $\mathcal{U} = \mathcal{V}$ *then* $\Theta \vDash \mathsf{F}(\Theta) \approx A$.

b) For any $\mathbf{codata}\ \mathsf{G}(\Theta) : \mathcal{U}\ \mathbf{where}\ \mathsf{O} : (\ \mid \mathsf{G}(\Theta) \vdash A : \mathcal{T})$, *if* $\mathcal{T} = \mathcal{N}$ *then* $\Theta \vDash \mathsf{G}(\Theta) \approx^- A$ *and if* $\mathcal{U} = \mathcal{N}$ *then* $\Theta \vDash \mathsf{G}(\Theta) \approx A$.

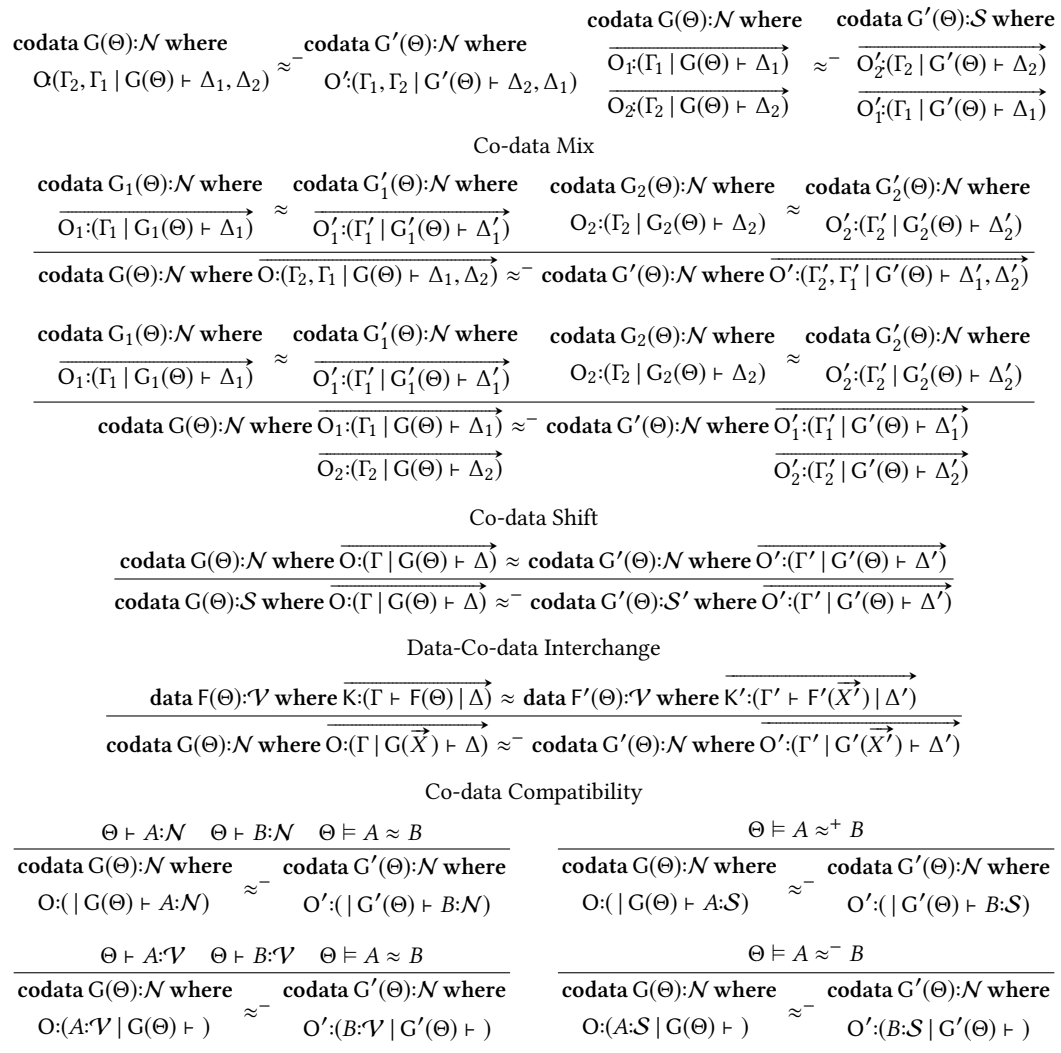

Fig. 9. A theory for structural laws of co-data type declaration isomorphisms.

5.2 Internal polarized laws of declarations

Now that we have established some basic structural laws about isomorphisms between general user-defined (co-)data types, we can focus on some more specific laws about the polarized primitives from fig. 4. In particular, we can show that the polar basis play a part in a family of isomorphisms that closely resemble some of the logical rules of the sequent calculus. Namely, each of the left rules for the positive data types and the right rules for the negative co-data types from fig. 5 correspond to an isomorphism between (co-)data declarations with signatures matching the premises and conclusion of the rules, as shown in fig. 10. The role of using declarations for this purpose is to give enough structural substrate for stating the rules: sequents with multiple inputs and multiple

Additive laws

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}_1{:}\,(A{:}\mathcal{V} \vdash \mathsf{F}(\Theta)\mid)\ \mathsf{K}_2{:}\,(B{:}\mathcal{V} \vdash \mathsf{F}(\Theta)\mid) \approx^{+}_{\oplus L} \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}'{:}\,(A \oplus B{:}\mathcal{V} \vdash \mathsf{F}'(\Theta)\mid)$$

$$\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{O}_1{:}\,(\mid \mathsf{G}(\Theta) \vdash A{:}\mathcal{N})\ \mathsf{O}_2{:}\,(\mid \mathsf{G}(\Theta) \vdash B{:}\mathcal{N}) \approx^{-}_{\&R} \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}'{:}\,(\mid \mathsf{G}'(\Theta) \vdash A\ \&\ B{:}\mathcal{N})$$

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where} \approx^{+}_{0L} \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}'{:}\,(0{:}\mathcal{V} \vdash \mathsf{F}'(\Theta)\mid)$$

$$\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{N}\ \mathbf{where} \approx^{-}_{\top R} \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}'{:}\,(\mid \mathsf{G}'(\Theta) \vdash \top{:}\mathcal{N})$$

Multiplicative laws

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}{:}\,(A{:}\mathcal{V}, B{:}\mathcal{V} \vdash \mathsf{F}(\Theta)\mid) \approx^{+}_{\otimes L} \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}'{:}\,(A \otimes B{:}\mathcal{V} \vdash \mathsf{F}'(\Theta)\mid)$$

$$\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}{:}\,(\vdash \mathsf{G}(\Theta)\mid A{:}\mathcal{N}, B{:}\mathcal{N}) \approx^{-}_{\parr R} \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}'{:}\,(\vdash \mathsf{G}'(\Theta)\mid A \parr B{:}\mathcal{N})$$

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}{:}\,(\vdash \mathsf{F}(\Theta)\mid) \approx^{+}_{1L} \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}'{:}\,(1{:}\mathcal{V} \vdash \mathsf{F}'(\Theta)\mid)$$

$$\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}{:}\,(\mid \mathsf{G}(\Theta) \vdash) \approx^{-}_{\bot R} \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}'{:}\,(\mid \mathsf{G}'(\Theta) \vdash \bot{:}\mathcal{N})$$

Negation Laws

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}{:}\,(\vdash \mathsf{F}(\Theta)\mid A{:}\mathcal{N}) \approx^{+}_{-L} \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}'{:}\,(-A{:}\mathcal{V} \vdash \mathsf{F}(\Theta)\mid)$$

$$\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}{:}\,(A{:}\mathcal{V} \mid \mathsf{G}(\Theta) \vdash) \approx^{-}_{\neg R} \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}'{:}\,(\mid \mathsf{G}(\Theta) \vdash \neg A{:}\mathcal{N})$$

Shift Laws

$$\mathbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}{:}\,(A{:}\mathcal{S} \vdash \mathsf{F}(\Theta)\mid) \approx^{+}_{\downarrow_{\mathcal{S}} L} \mathbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \mathbf{where}\ \mathsf{K}'{:}\,(\downarrow_{\mathcal{S}} A{:}\mathcal{V} \vdash \mathsf{F}'(\Theta)\mid)$$

$$\mathbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{K}{:}\,(\mid \mathsf{G}(\Theta) \vdash A{:}\mathcal{S}) \approx^{-}_{\uparrow_{\mathcal{S}} R} \mathbf{codata}\ \mathsf{G}'(\Theta){:}\mathcal{N}\ \mathbf{where}\ \mathsf{O}'{:}\,(\mid \mathsf{G}'(\Theta) \vdash \uparrow_{\mathcal{S}} A{:}\mathcal{N})$$

Fig. 10. Isomorphism laws of internal polarized sub-structures

outputs can be expressed by the types of constructors or observers, and multiple premises can be expressed by multiple alternatives for constructors or observers. As a result, we can reason about the polarized primitives appearing as part of the structure of larger (co-)data types.

Theorem 5.2 (Polarized laws). *The declaration isomorphism laws in fig. 10 are all sound.*

In addition to the specific laws of fig. 10, each of the polarized connectives is compatible with isomorphism. For example, if we have $A \approx A'$, then we also have $A \oplus B \approx A' \oplus B$ and $B \oplus A \approx B \oplus A'$. The only limitation is that we limit the types and programs to just the call-by-value ($\mathcal{V}$) and call-by-name ($\mathcal{N}$) base kinds in order to establish the compatibility of the $\mathcal{V}$-$\mathcal{N}$ shift pairs. This fact lets us apply type isomorphisms within the context of certain larger types: if two types are isomorphic, then we can build on them with polarized connectives however we want and still have an isomorphism. Said another way, for any type A made from polarized connectives and the $\mathcal{V}$ and $\mathcal{N}$ shifts, and any other isomorphic types $B \approx C$, we can substitute both B and C for X in A and still have the isomorphism $A\{B/X\} \approx A\{C/X\}$.

Theorem 5.3 (Polarized isomorphism substitution). *In the $\mathcal{VN}$ sub-calculus, for any types $\Theta, X : \mathcal{S} \vdash_{\mathcal{P}} A : \mathcal{T}$, $\Theta \vdash_{\mathcal{G}} B : \mathcal{S}$, and $\Theta \vdash_{\mathcal{G}} C : \mathcal{S}$, if $\Theta \vDash B \approx C$ then $\Theta \vDash A\{B/X\} \approx A\{C/X\}$.*

6 A SYNTACTIC THEORY OF POLARIZED TYPE ISOMORPHISMS

We have just seen in the previous section that there is *an* encoding of user-defined (co-)data types solely in terms of the basic polarized connectives. However, how do we know that this encoding is canonical, or that there are not many different and unrelated encodings for the same purpose? Does it matter what order in which the components of (co-)data types are put together, or in which way they are nested? Or maybe we could instead encode (co-)data types in terms of the positive $\oplus$ and $\otimes$ connectives instead of the negative & and $\parr$?

$$A \oplus B \approx^+ B \oplus A \qquad A \otimes B \approx^+ B \otimes A \qquad A \otimes (B \oplus C) \approx^+ (A \otimes B) \oplus (A \otimes C)$$
$$(A \oplus B) \oplus C \approx^+ A \oplus (B \oplus C) \qquad (A \otimes B) \otimes C \approx^+ A \otimes (B \otimes C) \qquad (A \oplus B) \otimes C \approx^+ (A \otimes C) \oplus (B \otimes C)$$
$$0 \oplus A \approx^+ A \approx^+ A \oplus 0 \qquad 1 \otimes A \approx^+ A \approx^+ A \otimes 1 \qquad A \otimes 0 \approx^+ 0 \approx^+ 0 \otimes A$$

$$A \,\&\, B \approx^- B \,\&\, A \qquad A \parr B \approx^- B \parr A \qquad A \parr (B \,\&\, C) \approx^- (A \parr B) \,\&\, (A \parr C)$$
$$(A \,\&\, B) \,\&\, C \approx^- A \,\&\, (B \,\&\, C) \qquad (A \parr B) \parr C \approx^- A \parr (B \parr C) \qquad (A \,\&\, B) \parr C \approx^- (A \parr C) \,\&\, (B \parr C)$$
$$\top \,\&\, A \approx^- A \approx^- A \,\&\, \top \qquad \bot \parr A \approx^- A \approx^- A \parr \bot \qquad A \parr \top \approx^- \top \approx^- \top \parr A$$

Fig. 11. Algebraic laws of the polarized basis of types.

As it turns out, none of these differences matter. The advantage of using the polarized connectives, as declared in fig. 4, as the basis for encodings is that they exhibit many pleasant—if none too surprising—properties, some of which have been explored previously by Zeilberger [2009] and Munch-Maccagnoni [2013]. That is, in contrast with types like call-by-name tuples or call-by-value functions, the relationships between types that we should expect—corresponding to common and well-known relationships from algebra and logic–*are* full-fledged isomorphisms between polarized types even in the face of effects that allow for terms to diverge without a result.

6.1 Algebraic laws

Let's begin by first exploring the algebraic properties of the polarized connectives, in particular, the isomorphic relationships between the additive and multiplicative connectives from fig. 4

- On the positive side, the $\oplus$ and 0 connectives form a *commutative monoid* of types up to isomorphism—meaning they satisfy commutative, associative, and unit laws as positive isomorphisms between types—and so do the $\otimes$ and 1 connectives. Furthermore, all four together form a *commutative semiring* up to positive isomorphism—meaning that the "multiplication" $\otimes$ distributes over $\oplus$ and is annihilated by 0.
- On the negative side, the $\&$ and $\top$ connectives form a commutative monoid up to negative isomorphism and $\parr$ and $\bot$ do as well. All four together form a commutative semiring.

The algebraic laws of the additive and multiplicative connectives are summarized in fig. 11.

We can verify that each of these isomorphisms are, in fact, isomorphisms using the previously-established laws of (co-)data declarations in general and internal polarized-substructures in particular from figs. 8 to 10. The technique follows the observation that, because of lemma 5.1, if we have either a singleton data or co-data declaration isomorphism of the form:

$$\textbf{data}\ \mathsf{F}() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A : \mathcal{V} \vdash \mathsf{F}() \mid) \approx^+ \textbf{data}\ \mathsf{F}'() : \mathcal{V}\ \textbf{where}\ \mathsf{K}' : (A' : \mathcal{V} \vdash \mathsf{F}'() \mid)$$

$$\textbf{codata}\ \mathsf{G}() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\mid \mathsf{G}() \vdash A : \mathcal{N}) \approx^- \textbf{codata}\ \mathsf{G}'() : \mathcal{N}\ \textbf{where}\ \mathsf{O}' : (\mid \mathsf{G}'() \vdash A' : \mathcal{N})$$

then we have $A \approx A'$ by composing $A \approx^+ \mathsf{F}() \approx^+ \mathsf{F}'() \approx^+ A'$ or $A \approx^- \mathsf{G}() \approx^- \mathsf{G}'() \approx^- A'$. Therefore, we can prove isomorphism laws about the polarized (co-)data types by (1) placing both sides of the proposed isomorphism within a singleton data or co-data type, as appropriate, (2) "unpacking" the two sides within the structure of the containing (co-)data type declaration, and (3) use the laws of declaration isomorphisms to show the two sides are indeed isomorphic. For example, combining the binary connectives with their corresponding units is an identity operation that leaves types unchanged, up to isomorphism. These unit laws rely on the fact that the right and left laws for the nullary connectives "cancel out," in an appropriate way, any occurrence of the nullary connective within a (co-)data declaration as described by the $1L$, $0L$, $\bot R$, and $\top R$ laws. For the multiplicative 1

$$-(A \mathbin{\&} B) \approx^{+} (-A) \oplus (-B) \quad -\top \approx^{+} 0 \quad -(A \parr B) \approx^{+} (-A) \otimes (-B) \quad -\bot \approx^{+} 1 \quad -(\neg A) \approx^{+} A$$
$$\neg(A \oplus B) \approx^{-} (\neg A) \mathbin{\&} (\neg B) \quad \neg 0 \approx^{-} \top \quad \neg(A \otimes B) \approx^{-} (\neg A) \parr (\neg B) \quad \neg 1 \approx^{-} \bot \quad \neg(-A) \approx^{-} A$$

Fig. 12. De Morgan duality laws of the polarized basis of types.

$$\downarrow_{\mathcal{V}} A \approx^{+} A \approx {}_{\mathcal{V}}\Uparrow A \qquad\qquad \uparrow_{\mathcal{N}} A \approx^{-} A \approx {}_{\mathcal{N}}\Downarrow A$$

Fig. 13. Identity laws of the redundant self-shift connectives.

and $\bot$ connectives, we use the fact that 1 vanishes from the left-hand side of a constructor and $\bot$ vanishes from the right-hand side of an observer:

data $\mathsf{F}_1() : \mathcal{V}$ **where** $\mathsf{K} : (1 \otimes A : \mathcal{V} \vdash \mathsf{F}_1() \mid)$	**codata** $\mathsf{G}_1() : \mathcal{N}$ **where** $\mathsf{O} : (\mid \mathsf{G}_1() \vdash \bot \parr A : \mathcal{N})$
$\approx_{\otimes L}$ **data** $\mathsf{F}_2() : \mathcal{V}$ **where** $\mathsf{K} : (1 : \mathcal{V}, A : \mathcal{V} \vdash \mathsf{F}_2() \mid)$	$\approx_{\parr R}$ **codata** $\mathsf{G}_2() : \mathcal{N}$ **where** $\mathsf{O} : (\mid \mathsf{G}_2() \vdash \bot : \mathcal{N}, A : \mathcal{N})$
$\approx_{1L}$ **data** $\mathsf{F}_3() : \mathcal{V}$ **where** $\mathsf{K} : (A : \mathcal{V} \vdash \mathsf{F}_3() \mid)$	$\approx_{\bot R}$ **codata** $\mathsf{G}_3() : \mathcal{N}$ **where** $\mathsf{O} : (\mid \mathsf{G}_3() \vdash A : \mathcal{N})$
$\approx_{1L}$ **data** $\mathsf{F}_4() : \mathcal{V}$ **where** $\mathsf{K} : (A : \mathcal{V}, 1 : \mathcal{V} \vdash \mathsf{F}_4() \mid)$	$\approx_{\bot R}$ **codata** $\mathsf{G}_4() : \mathcal{N}$ **where** $\mathsf{O} : (\mid \mathsf{G}_4() \vdash A : \mathcal{N}, \bot : \mathcal{N})$
$\approx_{\otimes L}$ **data** $\mathsf{F}_5() : \mathcal{V}$ **where** $\mathsf{K} : (A \otimes 1 : \mathcal{V} \vdash \mathsf{F}_5() \mid)$	$\approx_{\parr R}$ **codata** $\mathsf{G}_5() : \mathcal{N}$ **where** $\mathsf{O} : (\mid \mathsf{G}_5() \vdash A \parr \bot : \mathcal{N})$

Note the use of the mix law to extend $1L$ and $\bot R$ to allow for an extra component along side the unit connective. The rest of the algebraic laws in fig. 11 can be derived from the laws in fig. 10.

6.2 Duality laws

Isomorphism of types also gives us common logical properties of the polarized connectives based on duality, established with the same technique used in section 6.1. In particular, we get two parallel copies of the De Morgan laws—one for $-$ negation and the other for $\neg$ negation—that relates the positive data types with the negative co-data types as shown in fig. 12. The positive "or" ($\oplus$) is dualized into the negative "and" ($\&$) and the positive "and" ($\otimes$) is dualized into the negative "or" ($\parr$). Additionally, the two negation connectives cancel each other out, up to isomorphism. That is to say, they are a characterization of *involutive negation* as data and co-data types.[12]

6.3 Shift laws

The last group of polarized connectives, the shifts, have not appeared in any of the algebraic or duality laws here. That is partially because their role is not to represent the structural aspects of (co-)data types—like the ability to contain several components or offer multiple alternatives—but instead serve to explicitly signal the mechanisms, like the ability to delay a computation and force it later, that integrate different evaluation strategies. In fact, the presence of shifts have the effect of *prohibiting* the usual algebraic and dual laws of polarized types as we previously saw in the counter-examples from section 1 that appear in practice in functional programming languages.

Returning to the examples of unfaithful encodings from section 1, consider again the problem of encoding triples in terms of pairs in a Haskell-like lazy language, where lazy pairs are described by the $\times_{\mathcal{N}}$ data type declared previously in section 2, and lazy triples are represented as:

$$\textbf{data}\ \mathsf{LazyTriple}(X{:}\mathcal{N}, Y{:}\mathcal{N}, Z{:}\mathcal{N}) : \mathcal{N}\ \textbf{where}\ \mathsf{L3} : (X{:}\mathcal{N}, Y{:}\mathcal{N}, Z{:}\mathcal{N} \vdash \mathsf{LazyTriple}(X, Y, Z) \mid)$$

[12] This fact was noticed in another guise by Zeilberger [2009] and further brought to the forefront by Munch-Maccagnoni [2014]. The key is to have two dual negations, where one can be encoded with implication ($\neg A \approx A \to \bot$) and its dual can be encoded with the logical dual of implication called subtraction ($-A \approx 1 - A$).

By applying the polarization encoding from fig. 7 to a $\mathcal{G}$ containing both $\times_{\mathcal{N}}$ and LazyTriple, we get that $X{:}\mathcal{N}, Y{:}\mathcal{N}, Z{:}\mathcal{N} \vDash [\![\mathsf{LazyTriple}(X, Y, Z)]\!]_{\mathcal{G}} \approx \Uparrow(\downarrow X \otimes (\downarrow Y \otimes \downarrow Z))$ and $X{:}\mathcal{N}, Y{:}\mathcal{N}, Z{:}\mathcal{N} \vDash [\![X \times_{\mathcal{N}} (Y \times_{\mathcal{N}} Z)]\!]_{\mathcal{G}} \approx \Uparrow(\downarrow X \otimes \downarrow\Uparrow(\downarrow Y \otimes \downarrow Z))$,[13] but these two types represent very different spaces of possible program behaviors because of the extra shifts in the encoding of $X \times_{\mathcal{N}} (Y \times_{\mathcal{N}} Z)$. In other words, the difference between the two is that the type $X \times_{\mathcal{N}} (Y \times_{\mathcal{N}} Z)$ allows for extra values like $\mathsf{Pair}_{\mathcal{N}}(x, \mu\_.\, \langle y \| \beta \rangle)$, where $\mu\_.\, \langle y \| \beta \rangle$ is a term that does not return any result, but $\mathsf{LazyTriple}(X, Y, Z)$ does not, which is explicitly expressed by the presence or absence of shifts in their encoding. Furthermore, whereas we can apply properties like associativity of $\otimes$ within the encoding of $\mathsf{LazyTriple}(X, Y, Z)$, where $X{:}\mathcal{N}, Y{:}\mathcal{N}, Z{:}\mathcal{N} \vDash \Uparrow(\downarrow X \otimes (\downarrow Y \otimes \downarrow Z)) \approx \Uparrow((\downarrow X \otimes \downarrow Y) \otimes \downarrow Z)$, this is blocked by the extra shifts in $\Uparrow(\downarrow X \otimes \downarrow\Uparrow(\downarrow Y \otimes \downarrow Z))$, which prevent the law from applying.

We can also view the troubles with currying in an ML-like eager language in terms of the extra shifts that appear in the representation of call-by-value functions described by the $\to_{\mathcal{V}}$ co-data type from section 2, whose encoding simplifies to $X{:}\mathcal{V}, Y{:}\mathcal{V} \vDash [\![X \to_{\mathcal{V}} Y]\!]_{\mathcal{G}} \approx \Downarrow(\neg X \parr \uparrow Y)$. Again, the shifts get in the way when we try to apply the algebraic or logical laws of the polarized connectives. The type of uncurried call-by-value functions is $X{:}\mathcal{V}, Y{:}\mathcal{V}, Z{:}\mathcal{V} \vDash [\![(X \otimes Y) \to_{\mathcal{V}} Z]\!] \approx \Downarrow(\neg(X \otimes Y) \parr \uparrow Z) \approx \Downarrow(\neg X \parr \neg Y \parr \uparrow Z)$, whereas the type of curried call-by-value functions is $X{:}\mathcal{V}, Y{:}\mathcal{V}, Z{:}\mathcal{V} \vDash [\![X \to_{\mathcal{V}} (Y \to_{\mathcal{V}} Z)]\!] \approx \Downarrow(\neg X \parr \uparrow\Downarrow(\neg Y \parr \uparrow Z))$, which is not the same because of the extra shifts that appear in the curried call-by-value function.

This does not mean that the shifts are completely lawless, however. Since we began with a large family of shifts—singleton data and co-data type constructors mapping between any kind $\mathcal{S}$ and $\mathcal{V}$ or $\mathcal{N}$—some of them turn out to be redundant as shown in fig. 13. The data shifts $\downarrow_{\mathcal{V}}$ and ${}_{\mathcal{V}}\Uparrow$ for wrapping a $\mathcal{V}$ type as another $\mathcal{V}$ type and the co-data shifts $\uparrow_{\mathcal{N}}$ and ${}_{\mathcal{N}}\Downarrow$ for doing the same to $\mathcal{N}$ types are all identity operations on types, up to isomorphism. In particular, the data declarations for $\downarrow_{\mathcal{V}}$ and ${}_{\mathcal{V}}\Uparrow$ are the simplest instance of lemma 5.1 (a) which means that $\downarrow_{\mathcal{V}} A \approx^{+} A \approx^{+} {}_{\mathcal{V}}\Uparrow A$, and likewise $\uparrow_{\mathcal{N}} A \approx^{-} A \approx^{-} {}_{\mathcal{N}}\Downarrow A$ because of lemma 5.1 (b). This fact tells us that the polarizing translation on already-polarized types is actually an identity up to isomorphism, *i.e.,* for any $\Theta \vdash_{\mathcal{P}} A : \mathcal{S}$, it follows that $\Theta \vDash [\![A]\!]_{\mathcal{P}} \approx A$. For example, we have $X{:}\mathcal{V}, Y{:}\mathcal{V} \vDash [\![X \oplus Y]\!]_{\mathcal{P}} \triangleq {}_{\mathcal{V}}\Uparrow(((\downarrow_{\mathcal{V}} X \otimes 1) \oplus (\downarrow_{\mathcal{V}} Y \otimes 1)) \oplus 0) \approx X \oplus Y$ for the additive data type, and $X{:}\mathcal{V} \vDash [\![\neg X]\!]_{\mathcal{P}} \triangleq {}_{\mathcal{N}}\Downarrow(\top \,\&\, (\bot \parr (\neg\downarrow_{\mathcal{V}} X))) \approx \neg X$ for the negation co-data type, justifying our rule of thumb for deciding the appropriate disciplines for the polarized basis $\mathcal{P}$ of (co-)data types.

6.4 Functional laws

So far, our attention has been largely focused on properties of the polarized basis of (co-)data types from fig. 4, some of which, like $\parr$, are unfamiliar as programming constructs. But what about a more familiar construct like functions? We have seen that call-by-value functions don't behave as nicely as we'd like, which can be understood as inconvenient extra shifts between kinds denoting an unfortunate choice of discipline. So is there a type of function that avoids these problems? As it turns out, there is a multi-discipline, "primordial" [Zeilberger 2009] function type that captures the best of both the call-by-value and -name worlds, represented by the co-data declaration:

$$\textbf{codata}\,(X : \mathcal{V} \to Y : \mathcal{N}) : \mathcal{N}\ \textbf{where}\ \_ \cdot \_ : (X : \mathcal{V} \mid X \to Y \vdash Y : \mathcal{N})$$

which corresponds to the call-by-push-value function type [Levy 2001]. The particular placement of $\mathcal{V}$ and $\mathcal{N}$ again follows the rule of thumb from section 3, so as a consequence the polarized

[13] More specifically, the immediate output of translation is $[\![\mathsf{LazyTriple}(X, Y, Z)]\!]_{\mathcal{G}} \triangleq \Uparrow((\downarrow X \otimes (\downarrow Y \otimes (\downarrow Z \otimes 1))) \oplus 0)$ and $[\![X \times_{\mathcal{N}} Y]\!]_{\mathcal{G}} \triangleq \Uparrow((\downarrow X \otimes (\downarrow Y \otimes 1)) \oplus 0)$, which is cleaned up as shown by the laws in section 6.1.

$$A \to B \approx^- (-B) \to (\neg A) \qquad A \to (B \,\&\, C) \approx^- (A \to B) \,\&\, (A \to C)$$
$$(A \otimes B) \to C \approx^- A \to (B \to C) \qquad (A \oplus B) \to C \approx^- (A \to C) \,\&\, (B \to C) \qquad -(A \to B) \approx^- A \otimes (-B)$$
$$1 \to A \approx^- A \qquad A \to \top \approx^- \top \qquad A \to (\neg B) \approx^- \neg(A \otimes B)$$
$$A \to \bot \approx^- \neg A \qquad 0 \to A \approx^- \top$$

Fig. 14. Derived laws of polarized functions.

encoding for $A \to B$ avoids any non-trivial shifts. Because of the identity laws for shifts from fig. 13, the polarizing encoding for the above declaration $\mathcal{G}$ simplifies down to just $\neg$ and $\parr$:

$$X : \mathcal{V}, Y : \mathcal{N} \vDash [\![X \to Y]\!]_{\mathcal{G}} \approx^- {}_{\mathcal{N}}\Downarrow(\neg(\downarrow_{\mathcal{V}} X) \parr (\uparrow_{\mathcal{N}} Y)) \approx^- \neg X \parr Y$$

This gives us the most primitive expression of functions in our multi-discipline language; the rest can be encoded in terms of the above polarized function type by adding back the extra shifts.

Alternatively, we could have chosen to replace the unfamiliar $\parr$ with this function type. Because of the involutive nature of the dual $\neg$ and $-$ negations, we have the following encoding of $\parr$ disjunction in terms of $\to$ implication and $-$ negation:

$$A \parr B \approx^- \neg(-A) \parr B \approx^- (-A) \to B$$

Certainly functions are more familiar than $\parr$ as a programming construct, but the cost of leaning on this familiarity is the loss of symmetry because functions are a "half-negated disjunction." In particular, we can recast all of the algebraic and logical laws about $\parr$ in terms of $\to$ as shown in fig. 14—some of which are familiar properties of implication—that are all derived from the encoding $A \to B \approx^- \neg A \parr B$. The commutativity, associativity, and unit laws of the underlying $\parr$ give us contrapositive, currying, thunking, and negating laws:

$$A \to B \approx^- (\neg A) \parr B \approx^- B \parr (\neg A) \approx^- (\neg(-B)) \parr (\neg A) \approx^- (-B) \to (\neg A)$$
$$(A \otimes B) \to C \approx^- (\neg(A \otimes B)) \parr C \approx^- ((\neg A) \parr (\neg B)) \parr C \approx^- (\neg A) \parr ((\neg B) \parr C) \approx^- A \to (B \to C)$$
$$1 \to A \approx^- (\neg 1) \parr A \approx^- \bot \parr A \approx^- A$$
$$A \to \bot \approx^- (\neg A) \parr \bot \approx^- \neg A$$

Likewise, distributing $\parr$ over $\&$ and annihilating it with $\top$ recognizes certain functions types as products or trivial unit types:

$$A \to (B \,\&\, C) \approx^- (\neg A) \parr (B \,\&\, C) \approx^- ((\neg A) \parr B) \,\&\, ((\neg A) \parr C) \approx^- (A \to B) \,\&\, (A \to C)$$
$$(A \oplus B) \to C \approx^- (\neg(A \oplus B)) \parr C \approx^- ((\neg A) \,\&\, (\neg B)) \parr C \approx^- ((\neg A) \parr C) \,\&\, ((\neg B) \parr C) \approx^- (A \to C) \,\&\, (B \to C)$$
$$A \to \top \approx^- (\neg A) \parr \top \approx^- \top$$
$$0 \to A \approx^- (\neg 0) \parr A \approx^- \top \parr A \approx^- \top$$

And finally, the De Morgan duality between $\parr$ and $\otimes$ tells us that the continuation of a $\to$ function is a $\otimes$ pair, and dually that a continuation for a $\otimes$ pair is a $\to$ function:

$$-(A \to B) \approx^- -((\neg A) \parr B) \approx^- (-(\neg A)) \otimes (-B) \approx^- A \otimes (-B)$$
$$A \to (\neg B) \approx^- (\neg A) \parr (\neg B) \approx^- \neg(A \otimes B)$$

7 THE FAITHFULNESS OF POLARIZATION

Now that we have laid down some laws for declaration isomorphisms, we can put them to use for encoding user-defined (co-)data types in terms of the polarized basis from fig. 4. In particular, we can extend the laws from fig. 10 for polarized sub-structures appearing within a simple singleton

declaration to apply to any general (co-)data type using the mix laws from figs. 8 and 9. For example, given a declaration of the form

$$\begin{array}{l}\textbf{data}\ \mathsf{F}(\Theta) : \mathcal{V}\ \textbf{where}\\ \quad \mathsf{K}_0 : (\Gamma_0, A : \mathcal{V}, B : \mathcal{V} \vdash \mathsf{F}(\Theta) \mid \Delta_0)\\ \quad \overrightarrow{\mathsf{K} : (\Gamma \vdash \mathsf{F}(\Theta) \mid \Delta)}\end{array}$$

we can combine the A and B components of the K_0 constructor with the $\otimes$ connective by starting with the $\otimes L$ law, and then building up to the full declaration of F by applying the mix law to the appropriate reflexive isomorphisms as discussed in section 5 as follows:

$$\dfrac{\dfrac{\dfrac{\textbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \textbf{where}\ \mathsf{K}{:}\ (A{:}\mathcal{V}, B{:}\mathcal{V} \vdash \mathsf{F}(\Theta) \mid) \approx^+ \textbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \textbf{where}\ \mathsf{K}'{:}\ (A \otimes B{:}\mathcal{V} \vdash \mathsf{F}'(\Theta) \mid)}{\textbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \textbf{where}\ \mathsf{K}_0{:}\ (A{:}\mathcal{V}, B{:}\mathcal{V}, \Gamma_0 \vdash \mathsf{F}(\Theta) \mid \Delta_0) \approx^+ \textbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \textbf{where}\ \mathsf{K}'_0{:}\ (A \otimes B{:}\mathcal{V}, \Gamma_0 \vdash \mathsf{F}'(\Theta) \mid \Delta_0)}}{\textbf{data}\ \mathsf{F}(\Theta){:}\mathcal{V}\ \textbf{where}\ \mathsf{K}_0{:}\ (A{:}\mathcal{V}, B{:}\mathcal{V}, \Gamma_0 \vdash \mathsf{F}(\Theta) \mid \Delta_0) \approx^+ \textbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{V}\ \textbf{where}\ \mathsf{K}'_0{:}\ (A \otimes B{:}\mathcal{V}, \Gamma_0 \vdash \mathsf{F}'(\Theta) \mid \Delta_0)}}{\overrightarrow{\mathsf{K}{:}\ (\Gamma \vdash \mathsf{F}(\Theta) \mid \Delta)} \qquad \overrightarrow{\mathsf{K}{:}\ (\Gamma \vdash \mathsf{F}'(\Theta) \mid \Delta)}}$$

Similarly, other combinations of components at different positions in constructors of F can be isolated and targeted with the commute laws for data declarations. This idea is the central technique for proving the faithfulness of the polarizing encoding, which just repeats the above procedure until we are left with only a singleton (co-)data type that "wraps" its encoding. First we consider how to encode a just one (co-)data type declaration in terms of the polarized basis.

Theorem 7.1 (Polarizing (co-)data declarations).

a) For all $\textbf{data}\ \mathsf{F}(\Theta){:}\mathcal{S}\ \textbf{where}\ \overrightarrow{\mathsf{K}_i{:}(\overrightarrow{A_{ij}{:}\mathcal{T}_{ij}}^j \vdash \mathsf{F}(\Theta) \mid \overrightarrow{B_{ij}{:}\mathcal{U}_{ij}}^j)}^i \in \mathcal{G}, \Theta \vDash \mathsf{F}(\Theta) \approx^+ [\![\mathsf{F}(\Theta)]\!]_{\mathcal{G}}$

b) For all $\textbf{codata}\ \mathsf{G}(\Theta){:}\mathcal{S}\ \textbf{where}\ \overrightarrow{\mathsf{O}_i{:}(\overrightarrow{A_{ij}{:}\mathcal{T}_{ij}}^j \mid \mathsf{G}(\Theta) \vdash \overrightarrow{B_{ij}{:}\mathcal{U}_{ij}}^j)}^i \in \mathcal{G}, \Theta \vDash \mathsf{G}(\Theta) \approx^- [\![\mathsf{G}(\Theta)]\!]_{\mathcal{G}}$

Proof. a) Observe that we have the following data isomorphism by extending the polarized laws from fig. 10 with the mix and commute laws from fig. 8:

$$\begin{array}{l}\textbf{data}\ \mathsf{F}_1(\Theta) : \mathcal{V}\ \textbf{where}\ \overrightarrow{\mathsf{K}_i : \left(\overrightarrow{A_{ij} : \mathcal{T}_{ij}}^j \vdash \mathsf{F}_1(\Theta) \mid \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^j\right)}^i\\
\approx^+_{\downarrow L}\ \textbf{data}\ \mathsf{F}_2(\Theta) : \mathcal{V}\ \textbf{where}\ \overrightarrow{\mathsf{K}_i : \left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij} : \mathcal{V}}^j \vdash \mathsf{F}_2(\Theta) \mid \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^j\right)}^i\\
\approx^+_{\uparrow R}\ \textbf{data}\ \mathsf{F}_3(\Theta) : \mathcal{V}\ \textbf{where}\ \overrightarrow{\mathsf{K}_i : \left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij} : \mathcal{V}}^j \vdash \mathsf{F}_3(\Theta) \mid \overrightarrow{\uparrow_{\mathcal{U}_{ij}} B_{ij} : \mathcal{N}}^j\right)}^i\\
\approx^+_{-L}\ \textbf{data}\ \mathsf{F}_4(\Theta) : \mathcal{V}\ \textbf{where}\ \overrightarrow{\mathsf{K}_i : \left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij} : \mathcal{V}}^j, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij}) : \mathcal{V}}^j \vdash \mathsf{F}_4(\Theta) \mid \right)}^i\\
\approx^+_{1L, \otimes L}\ \textbf{data}\ \mathsf{F}_5(\Theta) : \mathcal{V}\ \textbf{where}\ \overrightarrow{\mathsf{K}_i : \left(\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}}^j, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij})}^j\right) : \mathcal{V} \vdash \mathsf{F}_5(\Theta) \mid \right)}^i\\
\approx^+_{0L, \oplus L}\ \textbf{data}\ \mathsf{F}_6(\Theta) : \mathcal{V}\ \textbf{where}\ \mathsf{K} : \left(\bigoplus\left(\overrightarrow{\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}}^j, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij})}^j\right)}^i\right) : \mathcal{V} \vdash \mathsf{F}_6(\Theta) \mid \right)\end{array}$$

With the above isomorphism between F_1 and F_6, it follows from the data shift law that:

$$\begin{array}{ll}\textbf{data}\ \mathsf{F}(\Theta){:}\mathcal{S}\ \textbf{where} & \textbf{data}\ \mathsf{F}'(\Theta){:}\mathcal{S}\ \textbf{where}\\
\overrightarrow{\mathsf{K}_i{:}\left(\overrightarrow{A_{ij}{:}\mathcal{T}_{ij}}^j \vdash \mathsf{F}(\Theta) \mid \overrightarrow{B_{ij}{:}\mathcal{U}_{ij}}^j\right)}^i \approx^+ & \mathsf{K}{:}\left(\bigoplus\left(\overrightarrow{\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}}^j, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij})}^j\right)}^i\right) : \mathcal{V} \vdash \mathsf{F}_6(\Theta) \mid \right)\end{array}$$

Therefore, we get $\Theta \vDash \mathsf{F}'(\Theta) \approx^+ \mathcal{S}\Uparrow\left(\bigoplus\left(\overrightarrow{\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}}^{j}, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij})}^{j}\right)}^{i}\right)\right) \triangleq [\![\mathsf{F}(\Theta)]\!]_{\mathcal{G}}$ by applying data compatibility[14] to the reflexive isomorphism of $\left(\bigoplus\left(\overrightarrow{\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}}^{j}, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij})}^{j}\right)}^{i}\right)\right)$, so by positive transitivity $\Theta \vDash \mathsf{F}(\Theta) \approx^+ [\![\mathsf{F}(\Theta)]\!]_{\mathcal{G}}$

b) Analogous to the proof of theorem 7.1 (a) by duality. □

Now that we know how to encode individual (co-)data types in isolation, we look to a global encoding of types in the $\mathcal{VN}$ sub-calculus made out of a collection $\mathcal{G}$ of (co-)data declarations. The only limitation on the group of declarations $\mathcal{G}$ is that they be well-formed and non-cyclic, written $\vdash \mathcal{G}$ defined by the following inference rules, where ϵ is the empty list of declarations:

$$\frac{}{\vdash \epsilon} \qquad \frac{\vdash \mathcal{G} \quad \mathcal{G} \vdash decl}{\vdash \mathcal{G}, decl}$$

The non-cyclic requirement ensures that the dependency chains between declarations is well-founded, so the process of inlining the encodings of (co-)data types will eventually terminate and give a final, fully-expanded encoding.

Theorem 7.2 ((Co-)Data Polarization). *In the $\mathcal{VN}$ sub-calculus, for any well-formed $\vdash \mathcal{G}$ and type $\Theta \vdash_{\mathcal{G}} A : \mathcal{S}$, we have $\Theta \vDash A \approx [\![A]\!]_{\mathcal{G}}$.*

Proof. By lexicographic induction on (1) the derivation of $\vdash \mathcal{G}$, and (2) the derivation of $\Theta \vdash_{\mathcal{G}} A : \mathcal{S}$. Note that in the $\mathcal{VN}$ sub-calculus, every isomorphism $A \approx B$ between types of the same kind is either positive or negative, so the so same-kinded type isomorphisms are always transitive. The case when A is a variable is immediate. The case where $A = \mathsf{F}(\overrightarrow{C})$ for some

$$\mathbf{data}\ \mathsf{F}(\overrightarrow{X : \mathcal{S}'}) : \mathcal{S}\ \mathbf{where}\ \overrightarrow{\mathsf{K} : \left(\overrightarrow{A : \mathcal{T}} \vdash \mathsf{F}(\overrightarrow{X}) \mid \overrightarrow{B : \mathcal{U}}\right)} \in \mathcal{G}$$

follows from theorems 5.3 and 7.1. In particular, we have $\overrightarrow{\Theta \vDash C \approx [\![C]\!]_{\mathcal{G}}}$, $\overrightarrow{X : \mathcal{S}'} \vDash A_{ij} \approx [\![A_{ij}]\!]_{\mathcal{G}'}$, and $\overrightarrow{X : \mathcal{S}'} \vDash B_{ij} \approx [\![B_{ij}]\!]_{\mathcal{G}'}$ from the inductive hypothesis for some $\mathcal{G}'$ strictly smaller than $\mathcal{G}$. From theorem 7.1, we have $\mathsf{F}(\overrightarrow{C}) \approx \mathcal{S}\Uparrow(\overrightarrow{\bigoplus(\overrightarrow{\bigotimes(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}\sigma}^{j}, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij}\sigma}^{j}))}^{i})})$ where $\sigma = \overrightarrow{\{C/X\}}$, and from theorem 5.3 we know that

$$\Theta \vDash \mathcal{S}\Uparrow\left(\bigoplus\left(\overrightarrow{\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} A_{ij}\sigma}^{j}, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} B_{ij}\sigma}^{j}\right)}^{i}\right)\right)$$

$$\approx \mathcal{S}\Uparrow\left(\bigoplus\left(\overrightarrow{\bigotimes\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}} [\![A_{ij}]\!]_{\mathcal{G}}[\![\sigma]\!]_{\mathcal{G}}}^{j}, \overrightarrow{-(\uparrow_{\mathcal{U}_{ij}} [\![B_{ij}]\!]_{\mathcal{G}}[\![\sigma]\!]_{\mathcal{G}}}^{j}\right)}^{i}\right)\right)$$

where $[\![\sigma]\!]_{\mathcal{G}} = \overrightarrow{\{[\![C]\!]_{\mathcal{G}}/X\}}$. Therefore, we have $\Theta \vDash \mathsf{F}(\overrightarrow{C}) \approx [\![\mathsf{F}(\overrightarrow{C})]\!]_{\mathcal{G}}$ by distributing the substitution σ over translation. The case for a co-data declaration in $\mathcal{G}$ follows similarly. □

Note that as an immediate consequence of the full (co-)data polarization encoding (theorem 7.2), we can generalize the fact that isomorphism distributes over substitution into a type made from polarized connectives (theorem 5.3) to conclude that isomorphism distributes over substitution into *any* type built from (non-cyclic) (co-)data type constructors. In the $\mathcal{VN}$ sub-calculus, for any $\vdash \mathcal{G}$; $\Theta, X : \mathcal{S} \vdash_{\mathcal{G}} A : \mathcal{T}$; $\Theta \vdash_{\mathcal{G}} B : \mathcal{S}$; and $\Theta \vdash_{\mathcal{G}} C : \mathcal{S}$, if $\Theta \vDash B \approx C$ then $\Theta \vDash A\{B/X\} \approx A\{C/X\}$. This fact means that we can apply any isomorphism within the context of any $\mathcal{VN}$ (co-)data type.

[14] The particular instance of data compatibility used here is specified and proved in the appendix lemma C.3 (a).

8 RELATED WORK

Multi-discipline languages. We use a multi-discipline target language based on polarity in logic [Andreoli 1992; Girard 1993; Laurent 2002] to faithfully encode user-defined types from both strict and lazy functional languages. Similar polarized languages have been used as a framework for modeling other functional features like the extensionality of sum types [Munch-Maccagnoni and Scherer 2015], sub-typing [Zeilberger 2009], dependent types [Licata and Harper 2008], and delimited control [Munch-Maccagnoni 2014; Zeilberger 2010]. As a related approach [Curien et al. 2016], Levy's [2003] call-by-push-value paradigm mixes both functional and imperative features.

In contrast, monadic languages, like Moggi's [1989] computational λ-calculus, is a more established technique in functional programming for combining call-by-value and -name evaluation within programs where the evaluation strategy can be seen as an effect, which can be used as an intermediate language for compiling both strict and lazy functional languages [Peyton Jones et al. 1998]. However, the two styles are not so distant; polarity and call-by-push-value revolve around a more fine-grained adjunction model of computation [Curien et al. 2016] where the shifts between call-by-value and call-by-name types (à la fig. 6) form an adjunction, so that a round-trip of shifting gives a monad contained within just one kind of type [Levy 2003; Zeilberger 2008].

Polarized type isomorphisms. Our interest in type isomorphisms [Di Cosmo 1995] are as a technical device for ensuring that encodings programming constructs are faithful. Isomorphisms between polarized types are especially interesting because of the competition between different evaluation strategies, and several definitions arose with various levels of generality. On the more specific end, Zeilberger [2009] separately defines isomorphisms between positive data types and between negative co-data types. Munch-Maccagnoni [2013] gives a more general definition of type isomorphisms with inverse mappings between any two types such that either (1) the types have the same polarity, or (2) the mappings are both *thunkable* and *linear*. As we saw here, the thunkable and linear restrictions work to reconcile the impact of competing evaluation strategies on the transitive relationship between types. In comparison, the polarized notion of syntactic type isomorphisms considered here further generalizes Munch-Maccagnoni's [2013] by only requiring thunkability (for a positive isomorphism) or linearity (for a negative isomorphism). Furthermore, the first option in which both types are $\mathcal{V}$-kinded or $\mathcal{N}$-kinded is subsumed since a $\mathcal{V}$ isomorphism is always positive and a $\mathcal{N}$ isomorphism is always negative by definition. The benefit of this further generalization is call-by-need and other evaluation strategies can be integrated into the language while still preserving the same sorts of type isomorphisms from the $\mathcal{VN}$ setting. Levy [2017] formulates a single definition of type isomorphism that applies uniformly to many languages, including multi-discipline ones, by analogy to contextual equivalence of programs: two types are *contextually isomorphic* if they give the same type (up to ordinary isomorphism) when substituted into any context. This makes a property like theorem 5.3 into the "gold standard" definition.

Complex connectives. The idea of synthesizing complex connectives in terms of basic polarized building blocks appeared before in Girard's [1993] *chimeric* connectives. This idea appears again in Levy's [2006b] jumbo λ-calculus, where the *jumbo* connectives serve a similar purpose as user-defined (co-)data types, except limited to only intuitionistic types that would be found in a functional language, and not classical ones with multiple conclusions like $A \parr B$ or $-A$. Levy [2006a,b] shows that encoding globally call-by-value and call-by-name languages with jumbo connectives into call-by-push-value primitives gives equivalent types using denotational methods, but only considers type isomorphisms based on syntactic program transformations for effect-free languages. Here, we bring out the symmetries underlying encodings of complex connectives by using the basic connectives from classical linear logic [Girard 1987] while retaining the entirely syntactic form

of type isomorphisms that could be used as program transformations in compilers. We also show how multiple evaluation strategies can be mixed inside complex connectives, including additional evaluation strategies like call-by-need that goes beyond call-by-value and -name.

Multi-language semantics. The multi-discipline language used here could also be seen as a combination of many languages—one for each kind—into one. The idea of multi-language semantics [Matthews and Findler 2007] has been used for ensuring *full abstraction*—a one-for-one soundness and completeness between source and target languages—but for different applications in compilation including continuation-passing style translation [Ahmed and Blume 2011], closure conversion [New et al. 2016], and modular compiler verification [Ahmed 2015; Perconti and Ahmed 2014]. The connection between shifts (fig. 6) and language barriers deserves more investigation.

9 CONCLUSION

We employ encodings all the time as programming language designers, implementers, and theorists, but those encodings are not always accurate representations in practical languages where program features, like exceptions or even just recursion, can sometimes turn "obvious" encodings into leaky abstractions. Here, we have seen how a polarized basis of types let us rely on the common encodings we know and love for supporting both user-defined data and co-data types in eager, lazy, and mixed languages. We used the idea of type isomorphisms as a technique for making sure that the proposed encodings are faithful, so that we can encode and decode without any loss of information, and that they exhibit the mathematical and logical properties that we should expect. In order to support evaluation strategies beyond just call-by-value and call-by-name, we saw how two pairs of shifts were needed to enter and exit the call-by-value and -name worlds where (co-)data behaves best.

We limited the source language to simple types for simplicity, but the technique presented here straightforwardly extends from monomorphic types to polymorphic ones. In particular, we can model polymorphism in system F_ω style with explicit type abstractions and instantiations by extending the language with type functions and letting constructors of data types and observers of co-data type include hidden quantified types (generalizing the $\exists$ and $\forall$ quantifiers) as shown by Downen et al. [2015]. The generalized type quantifications in user-defined types be encoded using an existential $\exists$ data type and universal $\forall$ co-data type, and since the addition of parametrically quantified types does not change the dynamic behavior of programs, all the same sorts of type isomorphisms hold. Extending the polar basis to represent recursive (co-)data types, however, would require some more work to treat formally in a similar approach as the one taken here.

In this paper, we looked at a multi-discipline language that lets programs combine four different kinds of evaluation strategies: call-by-value, call-by-name, lazy call-by-value (call-by-need), and lazy call-by-name (the dual of call-by-need). However, the sequent calculus that our source language was based on [Downen and Ariola 2014] is actually much more general, and accommodates any number of different base kinds for controlling the substitution disciplines. We can add any other base kind $\mathcal{D}$ to the source language and the same encoding technique only requires that $\mathcal{D}$ satisfies some basic sanity conditions [Downen et al. 2015] of *stability* (*i.e.,* (co-)values are closed under substitution) and *focalization* (*i.e.,* that there are enough (co-)values). Then in the target polarized language, we get another quadruple of shifts from fig. 6. This way, the encodings presented here bridge the gap the languages we want to use and their polarized foundation.

The style of accepting multiple kinds of types can be used for a practical model of intermediate languages for implementing functional languages, and arguably *already has* been put to use in a preliminary form. The core intermediate language of the Glasgow Haskell Compiler (GHC) includes (roughly) two distinct kinds of types: types with a lazy semantics that correspond to ordinary source Haskell types and *unboxed* types [Peyton Jones and Launchbury 1991] for representing things

like machine integers which correspond to positive types that (must) have an eager semantics. In this light, GHC's core intermediate language can be understood as a limited form of polarized intermediate language. We intend to build on previous work by Maurer et al. [2017] and Downen et al. [2016] to investigate how GHC can be extended with additional constructs from polarized logic; there is already current work on implementing unboxed (*i.e.,* positive) sum types, but what's still missing are *unlifted* (*i.e.,* negative) functions and other negative co-data types. In the end, the polarized basis of types gives us a unified core language that fuses purity with impurity for both lazy and eager functional languages alike.

REFERENCES

Amal Ahmed. 2015. Verified Compilers for a Multi-Language World. In *1st Summit on Advances in Programming Languages (SNAPL 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 32. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 15–31. DOI: http://dx.doi.org/10.4230/LIPIcs.SNAPL.2015.15

Amal Ahmed and Matthias Blume. 2011. An Equivalence-preserving CPS Translation via Multi-language Semantics. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11)*. ACM, New York, NY, USA, 431–444. DOI: http://dx.doi.org/10.1145/2034773.2034830

Jean-Marc Andreoli. 1992. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation* 2, 3 (1992), 297–347. DOI: http://dx.doi.org/10.1093/logcom/2.3.297

Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. 2011. Classical Call-By-Need and Duality. In *Typed Lambda Calculi and Applications: 10th International Conference (TLCA'11)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 27–44. DOI: http://dx.doi.org/10.1007/978-3-642-21691-6_6

Zena M. Ariola, John Maraist, Martin Odersky, Matthias Felleisen, and Philip Wadler. 1995. A Call-By-Need Lambda Calculus. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '95)*. ACM, New York, NY, USA, 233–246. DOI: http://dx.doi.org/10.1145/199448.199507

Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. 2016. A Theory of Effects and Resources: Adjunction Models and Polarised Calculi. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '16)*. ACM, New York, NY, USA, 44–56. DOI: http://dx.doi.org/10.1145/2837614.2837652

Pierre-Louis Curien and Hugo Herbelin. 2000. The Duality of Computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*. ACM, New York, NY, USA, 233–243. DOI: http://dx.doi.org/10.1145/351240.351262

Roberto Di Cosmo. 1995. *Isomorphisms of Types: From λ-calculus to Information Retrieval and Language Design*. Birkhauser Verlag, Basel, Switzerland.

Paul Downen and Zena M. Ariola. 2014. The Duality of Construction. In *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*, Zhong Shao (Ed.). Lecture Notes in Computer Science, Vol. 8410. Springer Berlin Heidelberg, Berlin, Heidelberg, 249–269. DOI: http://dx.doi.org/10.1007/978-3-642-54833-8_14

Paul Downen, Philip Johnson-Freyd, and Zena M. Ariola. 2015. Structures for Structural Recursion. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming (ICFP '15)*. ACM, New York, NY, USA, 127–139. DOI: http://dx.doi.org/10.1145/2784731.2784762

Paul Downen, Luke Maurer, Zena M. Ariola, and Simon Peyton Jones. 2016. Sequent Calculus As a Compiler Intermediate Language. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming (ICFP '16)*. ACM, New York, NY, USA, 74–88. DOI: http://dx.doi.org/10.1145/2951913.2951931

Gerhard Gentzen. 1935. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift* 39, 1 (1935), 176–210. DOI: http://dx.doi.org/10.1007/BF01201353

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science* 50, 1 (1987), 1–101. DOI: http://dx.doi.org/10.1016/0304-3975(87)90045-4

Jean-Yves Girard. 1993. On the Unity of Logic. *Annals of Pure and Applied Logic* 59, 3 (1993), 201–217. DOI: http://dx.doi.org/10.1016/0168-0072(93)90093-S

Olivier Laurent. 2002. *Étude de la polarisation en logique*. Ph.D. Dissertation. Université de la Méditerranée - Aix-Marseille II.

Paul Blain Levy. 2001. *Call-By-Push-Value*. Ph.D. Dissertation. Queen Mary and Westfield College, University of London.

Paul Blain Levy. 2003. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantics Structures in Computation, Vol. 2. Springer Netherlands. DOI: http://dx.doi.org/10.1007/978-94-007-0954-6

Paul Blain Levy. 2006a. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation* 19, 4 (01 Dec. 2006), 377–414. DOI: http://dx.doi.org/10.1007/s10990-006-0480-6

Paul Blain Levy. 2006b. *Jumbo λ-Calculus.* Springer Berlin Heidelberg, Berlin, Heidelberg, 444–455. DOI: http://dx.doi.org/10.1007/11787006_38

Paul Blain Levy. 2017. Contextual Isomorphisms. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017).* ACM, New York, NY, USA, 400–414. DOI: http://dx.doi.org/10.1145/3009837.3009898

Daniel R. Licata and Robert Harper. 2008. Positively Dependent Types. In *Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification (PLPV '09).* ACM, New York, NY, USA, 3–14. DOI: http://dx.doi.org/10.1145/1481848.1481851

Jacob Matthews and Robert Bruce Findler. 2007. Operational Semantics for Multi-language Programs. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '07).* ACM, New York, NY, USA, 3–10. DOI: http://dx.doi.org/10.1145/1190216.1190220

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

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

Guillaume Munch-Maccagnoni. 2009. Focalisation and Classical Realisability. In *Computer Science Logic: 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL (CSL 2009)*, Erich Grädel and Reinhard Kahle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 409–423. DOI: http://dx.doi.org/10.1007/978-3-642-04027-6_30

Guillaume Munch-Maccagnoni. 2013. *Syntax and Models of a non-Associative Composition of Programs and Proofs.* Ph.D. Dissertation. Université Paris Diderot.

Guillaume Munch-Maccagnoni. 2014. Formulae-as-Types for an Involutive Negation. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (CSL-LICS '14).* ACM, New York, NY, USA, Article 70, 10 pages. DOI: http://dx.doi.org/10.1145/2603088.2603156

Guillaume Munch-Maccagnoni and Gabriel Scherer. 2015. Polarised Intermediate Representation of Lambda Calculus with Sums. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015).* IEEE, 127–140. DOI: http://dx.doi.org/10.1109/LICS.2015.22

Max S. New, William J. Bowman, and Amal Ahmed. 2016. Fully Abstract Compilation via Universal Embedding. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming (ICFP 2016).* ACM, New York, NY, USA, 103–116. DOI: http://dx.doi.org/10.1145/2951913.2951941

James T. Perconti and Amal Ahmed. 2014. Verifying an Open Compiler Using Multi-language Semantics. In *Proceedings of the 23rd European Symposium on Programming Languages and Systems - Volume 8410.* Springer-Verlag New York, Inc., New York, NY, USA, 128–148. DOI: http://dx.doi.org/10.1007/978-3-642-54833-8_8

Simon Peyton Jones, Mark Shields, John Launchbury, and Andrew Tolmach. 1998. Bridging the Gulf: A Common Intermediate Language for ML and Haskell. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '98).* ACM, New York, NY, USA, 49–61. DOI: http://dx.doi.org/10.1145/268946.268951

Simon L. Peyton Jones and John Launchbury. 1991. Unboxed Values as First Class Citizens in a Non-Strict Functional Language. In *Functional Programming Languages and Computer Architecture: 5th ACM Conference*, John Hughes (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 636–666. DOI: http://dx.doi.org/10.1007/3540543961_30

Noam Zeilberger. 2008. On the Unity of Duality. *Annals of Pure and Applied Logic* 153, 1 (2008), 660–96. DOI: http://dx.doi.org/10.1016/j.apal.2008.01.001

Noam Zeilberger. 2009. *The Logical Basis of Evaluation Order and Pattern-Matching.* Ph.D. Dissertation. Carnegie Mellon University.

Noam Zeilberger. 2010. Polarity and the Logic of Delimited Continuations. In *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science (LICS '10).* IEEE Computer Society, Washington, DC, USA, 219–227. DOI: http://dx.doi.org/10.1109/LICS.2010.23

A PROOF OF SOUNDNESS OF THE POLARIZATION ENCODING

The polarizing encoding of (co-)data types as shown in fig. 7 is stated in terms of deep pattern matching on data structures and co-data observations, which avoids the terrifying bureaucracy of the many levels of shallow patterns needed to implement the translation. Thankfully, these deep patterns fit a certain form which makes them much easier to desugar compared to fully general patterns. In particular, every pattern used in the encoding begins with a match on a ${}_{\mathcal{S}}\Uparrow$ or ${}_{\mathcal{S}}\Downarrow$ shift, then several nested matches on the additive structure of type $A \oplus B$ or $A \,\&\, B$, and then concludes with a match on the multiplicative structure of the following form:

$$p \in \mathit{Pattern} ::= {}_{\mathcal{S}}\Uparrow(p^+) \qquad q \in \mathit{CoPattern} ::= {}_{\mathcal{S}}\Downarrow[q^+]$$
$$p^+ \in \mathit{AddPattern} ::= p^\times \mid \iota_1\,(p^+) \mid \iota_2\,(p^+) \qquad q^+ \in \mathit{AddCoPattern} ::= q^\times \mid \pi_1\,[q^+] \mid \pi_2\,[q^+]$$
$$p^\times \in \mathit{MultPattern} ::= x \mid () \mid (x, p^\times) \mid -(q^\times) \mid \downarrow_{\mathcal{S}}(x) \qquad q^\times \in \mathit{MultCoPattern} ::= \alpha \mid [] \mid [\alpha, q^\times] \mid \neg p^\times \mid \uparrow_{\mathcal{S}}[\alpha]$$

We can then easily desugar (co-)patterns of this form by just un-nesting the pattern one level at a time within the alternatives of every pattern matching (co-)term as follows:

$$\tilde{\mu}\left[\overrightarrow{{}_{\mathcal{S}}\Uparrow(p_i^+).c_i}^{\,i}\right] \triangleq \tilde{\mu}\left[{}_{\mathcal{S}}\Uparrow x.\left\langle x \middle\| \tilde{\mu}\left[\overrightarrow{p_i^+.c_i}^{\,i}\right]\right\rangle\right] \qquad \mu\left(\overrightarrow{{}_{\mathcal{S}}\Downarrow[q_i^+].c_i}^{\,i}\right) \triangleq \mu\left({}_{\mathcal{S}}\Downarrow[\alpha].\left\langle \mu\left(\overrightarrow{q_i^+.c_i}^{\,i}\right) \middle\| \alpha\right\rangle\right)$$

$$\tilde{\mu}\begin{bmatrix}\overrightarrow{\iota_1\,(p_i^+).c_i}^{\,i}\\ \overrightarrow{\iota_2\,(p_i'^+).c_i'}^{\,i}\end{bmatrix} \triangleq \tilde{\mu}\begin{bmatrix}\iota_1\,(x).\left\langle x \middle\| \tilde{\mu}\left[\overrightarrow{p_i^+.c_i}^{\,i}\right]\right\rangle\\ \iota_2\,(x).\left\langle x \middle\| \tilde{\mu}\left[\overrightarrow{p_i'^+.c_i'}^{\,i}\right]\right\rangle\end{bmatrix} \qquad \mu\begin{pmatrix}\overrightarrow{\pi_1\,[q_i^+].c_i}^{\,i}\\ \overrightarrow{\pi_2\,[q_i'^+].c_i'}^{\,i}\end{pmatrix} \triangleq \mu\begin{pmatrix}\pi_1\,[\alpha].\left\langle \mu\left(\overrightarrow{q_i^+.c_i}^{\,i}\right) \middle\| \alpha\right\rangle\\ \pi_2\,[\alpha].\left\langle \mu\left(\overrightarrow{q_i'^+.c_i'}^{\,i}\right) \middle\| \alpha\right\rangle\end{pmatrix}$$

$$\tilde{\mu}[x.c] \triangleq \tilde{\mu}x.c \qquad \mu(\alpha.c) \triangleq \mu\alpha.c$$
$$\tilde{\mu}\left[(y, p^\times).c\right] \triangleq \tilde{\mu}\left[(y, x).\left\langle x \middle\| \tilde{\mu}\left[p^\times.c\right]\right\rangle\right] \qquad \mu([\beta, q^\times].c) \triangleq \mu([\beta, \alpha].\left\langle \mu(q^\times.c) \middle\| \alpha\right\rangle)$$
$$\tilde{\mu}\left[-(q^\times).c\right] \triangleq \tilde{\mu}\left[-(\alpha).\left\langle \mu(q^\times.c) \middle\| \alpha\right\rangle\right] \qquad \mu(\neg[p^\times].c) \triangleq \mu(\neg[x].\left\langle x \middle\| \tilde{\mu}\left[p^\times.c\right]\right\rangle)$$

Additionally, in order to prove the soundness of the η law for (co-)data types with respect to the encoding, we use a couple helpful tricks with η. First, note that the seemingly stronger version of the η law for co-data types which applies to values (or the stronger η law for data types that applies to co-values)

$$(\eta^{\mathsf{F}}_{\mathcal{S}}) \quad E : \mathsf{F}(\vec{C}) = \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha}, \vec{x}).\langle \mathsf{K}(\vec{\alpha}, \vec{x}) \| E\rangle}\right] (\eta^{\mathsf{G}}_{\mathcal{S}}) \quad V : \mathsf{G}(\vec{C}) = \mu\left(\overrightarrow{\mathsf{O}[\vec{x}, \vec{\alpha}].\langle V \| \mathsf{O}[\vec{x}, \vec{\alpha}]\rangle}\right)$$

can be derived from the η law on (co-)variables by combining with the η_μ and $\eta_{\tilde{\mu}}$ rules for μ- and $\tilde{\mu}$-abstractions as follows:

$$\begin{aligned}
E : \mathsf{F}(\vec{C}) &=_{\eta_\mu \eta_{\tilde{\mu}}} \tilde{\mu} y{:}\,\mathsf{F}(\vec{C}).\, \left\langle \mu\beta{:}\,\mathsf{F}(\vec{C}).\, \langle y \| \beta\rangle \middle\| E\right\rangle \\
&=_{\eta^{\mathsf{F}}} \tilde{\mu} y{:}\,\mathsf{F}(\vec{C}).\, \left\langle \mu\beta{:}\,\mathsf{F}(\vec{C}).\, \left\langle y \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha}, \vec{x}).\langle \mathsf{K}(\vec{\alpha}, \vec{x}) \| \beta\rangle}\right]\right\rangle \middle\| E\right\rangle \\
&=_{\mu} \tilde{\mu} y{:}\,\mathsf{F}(\vec{C}).\, \left\langle y \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha}, \vec{x}).\langle \mathsf{K}(\vec{\alpha}, \vec{x}) \| E\rangle}\right]\right\rangle \\
&=_{\eta_{\tilde{\mu}}} \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha}, \vec{x}).\langle \mathsf{K}(\vec{\alpha}, \vec{x}) \| E\rangle}\right]
\end{aligned}$$

$$\begin{aligned}
V : \mathsf{G}(\vec{C}) &=_{\eta_\mu \eta_{\tilde{\mu}}} \mu\beta{:}\,\mathsf{G}(\vec{C}).\, \left\langle V \middle\| \tilde{\mu} y{:}\,\mathsf{G}(\vec{C}).\, \langle y \| \beta\rangle\right\rangle \\
&=_{\eta^{\mathsf{G}}} \mu\beta{:}\,\mathsf{G}(\vec{C}).\, \left\langle V \middle\| \tilde{\mu} y{:}\,\mathsf{G}(\vec{C}).\, \left\langle \mu\left(\overrightarrow{\mathsf{O}[\vec{x}, \vec{\alpha}].\langle y \| \mathsf{O}[\vec{x}, \vec{\alpha}]\rangle}\right) \middle\| \beta\right\rangle\right\rangle \\
&=_{\tilde{\mu}} \mu\beta{:}\,\mathsf{G}(\vec{C}).\, \left\langle \mu\left(\overrightarrow{\mathsf{O}[\vec{x}, \vec{\alpha}].\langle V \| \mathsf{O}[\vec{x}, \vec{\alpha}]\rangle}\right) \middle\| \beta\right\rangle \\
&=_{\eta^{\mathsf{G}}} \mu\left(\overrightarrow{\mathsf{O}[\vec{x}, \vec{\alpha}].\langle V \| \mathsf{O}[\vec{x}, \vec{\alpha}]\rangle}\right)
\end{aligned}$$

Second, note that we have the following equalities

$$(\tilde{\mu}\eta^{\mathsf{F}}_{\mathcal{S}}) \qquad c = \left\langle z \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha},\vec{x}).c\,\{\mathsf{K}(\vec{\alpha},\vec{x})/z\}}\right]\right\rangle \qquad (\tilde{\mu}z.c : \mathsf{F}(\vec{C}) \in CoValue_{\mathcal{S}})$$

$$(\mu\eta^{\mathsf{G}}_{\mathcal{S}}) \qquad c = \left\langle \mu\left(\overrightarrow{\mathsf{H}[\vec{x},\vec{\alpha}].c\,\{\mathsf{H}[\vec{x},\vec{\alpha}]/\gamma\}}\right) \middle\| \gamma\right\rangle \qquad (\mu\gamma.c : \mathsf{G}(\vec{C}) \in Value_{\mathcal{S}})$$

the first of which is derived from the η law of the F as follows:

$$\begin{aligned} c &=_{\tilde{\mu}_{\mathcal{S}}} \langle z \| \tilde{\mu}z.c \rangle \\ &=_{\eta^{\mathsf{F}}_{\mathcal{S}}} \left\langle z \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha},\vec{x}).\langle \mathsf{K}(\vec{\alpha},\vec{x}) \| \tilde{\mu}z.c\rangle}\right]\right\rangle \qquad (\tilde{\mu}z.c : \mathsf{F}(\vec{C}) \in CoValue_{\mathcal{S}}) \\ &=_{\tilde{\mu}_{\mathcal{S}}} \left\langle z \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}(\vec{\alpha},\vec{x}).c\,\{\mathsf{K}(\vec{\alpha},\vec{x})/z\}}\right]\right\rangle \end{aligned}$$

and the second of which is likewise derived from the η law of G as follows:

$$\begin{aligned} c &=_{\mu_{\mathcal{S}}} \langle \mu\gamma.c \| \gamma \rangle \\ &=_{\eta^{\mathsf{G}}_{\mathcal{S}}} \left\langle \mu\left(\overrightarrow{\mathsf{H}[\vec{x},\vec{\alpha}].\langle \mu\gamma.c \| \mathsf{H}[\vec{x},\vec{\alpha}]\rangle}\right) \middle\| \gamma\right\rangle \qquad (\mu\gamma.c : \mathsf{G}(\vec{C}) \in Value_{\mathcal{S}}) \\ &=_{\mu_{\mathcal{S}}} \left\langle \mu\left(\overrightarrow{\mathsf{H}[\vec{x},\vec{\alpha}].c\,\{\mathsf{H}[\vec{x},\vec{\alpha}]/\gamma\}}\right) \middle\| \gamma\right\rangle \end{aligned}$$

As examples, the particular instances of this rule for the polarized data types are:

$$(\tilde{\mu}\eta^{\otimes}_{\mathcal{V}}) \qquad c = \langle z \| \tilde{\mu}[(x,y).c\,\{(x,y)/z\}]\rangle \qquad (z : A \otimes B)$$

$$(\tilde{\mu}\eta^{\oplus}_{\mathcal{V}}) \qquad c = \langle z \| \tilde{\mu}[\iota_1(x).c\,\{\iota_1(x)/z\} | \iota_2(y).c\,\{\iota_2(y)/z\}]\rangle \qquad (z : A \oplus B)$$

$$(\tilde{\mu}\eta^{1}_{\mathcal{V}}) \qquad c = \langle z \| \tilde{\mu}[().c\,\{()/z\}]\rangle \qquad (z : 1)$$

$$(\tilde{\mu}\eta^{0}_{\mathcal{V}}) \qquad c = \langle z \| \tilde{\mu}[]\rangle \qquad (z : 0)$$

$$(\tilde{\mu}\eta^{-}_{\mathcal{V}}) \qquad c = \langle z \| \tilde{\mu}[-(\alpha).c\,\{-(\alpha)/z\}]\rangle \qquad (z : -A)$$

and the instances for the polarized co-data types are:

$$(\mu\eta^{⅋}_{\mathcal{N}}) \qquad c = \langle \mu([\alpha,\beta].c\,\{[\alpha,\beta]/\gamma\}) \| \gamma\rangle \qquad (\gamma : A ⅋ B)$$

$$(\mu\eta^{\&}_{\mathcal{N}}) \qquad c = \langle \mu(\pi_1[\alpha].c\,\{\pi_1[\alpha]/\gamma\} | \pi_2[\beta].c\,\{\pi_2[\beta]/\gamma\}) \| \gamma\rangle \qquad (\gamma : A \,\&\, B)$$

$$(\mu\eta^{\bot}_{\mathcal{N}}) \qquad c = \langle \mu([].c\,\{[]/\gamma\}) \| \gamma\rangle \qquad (\gamma : \bot)$$

$$(\mu\eta^{\top}_{\mathcal{N}}) \qquad c = \langle \mu() \| \gamma\rangle \qquad (\gamma : \top)$$

$$(\mu\eta^{\neg}_{\mathcal{N}}) \qquad c = \langle \mu(\neg[x].c\,\{\neg[x]/\gamma\}) \| \gamma\rangle \qquad (\gamma : \neg A)$$

With the above observations about pattern matching and extensionality, we are now ready to prove that the translation is sound.

Theorem 3.1 (Polarization soundness). *For $i = 1, 2$,*

a) if $c_i : (\Gamma \vdash_{\mathcal{G}} \Delta)$ and $c_1 = c_2$ then $[\![c_i]\!]_{\mathcal{G}} : ([\![\Gamma]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![\Delta]\!]_{\mathcal{G}})$ and $[\![c_1]\!]_{\mathcal{G}} = [\![c_2]\!]_{\mathcal{G}}$,
b) if $\Gamma \vdash_{\mathcal{G}} v_i : A \mid \Delta$ and $v_1 = v_2$ then $[\![\Gamma]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![v_i]\!]_{\mathcal{G}} : [\![A]\!]_{\mathcal{G}} \mid [\![\Delta]\!]_{\mathcal{G}}$ and $[\![v_1]\!]_{\mathcal{G}} = [\![v_2]\!]_{\mathcal{G}}$, and
c) if $\Gamma \mid e_i : A \vdash_{\mathcal{G}} \Delta$ and $e_1 = e_2$ then $[\![\Gamma]\!]_{\mathcal{G}} \mid [\![e_i]\!]_{\mathcal{G}} : [\![A]\!]_{\mathcal{G}} \vdash_{\mathcal{P}} [\![\Delta]\!]_{\mathcal{G}}$ and $[\![e_1]\!]_{\mathcal{G}} = [\![e_2]\!]_{\mathcal{G}}$, and

Proof. The fact that well-typed commands and (co-)terms have the associated translated type follows straightforwardly by (mutual) induction on their typing derivations. More interesting is the translation of equalities across the encoding. Note that since the translation is compositional and hygienic, the reflexive, symmetric, transitive, and (importantly) congruent closure of the equational theory is guaranteed. Therefore, we only need to check that each axiom is preserved by the translation. In that regard, it is important to note the fact that (1) that (co-)values translate to

(co-)values and (2) substitution distributes over translation (that is, $[\![c]\!]_{\mathcal{G}}\left\{[\![V]\!]_{\mathcal{G}}/x\right\} =_{\alpha} [\![c\left\{V/x\right\}]\!]_{\mathcal{G}}$, *etc.*), both of which can be confirmed by induction on the syntax of (co-)terms.

The substitution axioms translate directly without change because of the above mentioned two facts about (co-)values and substitution, like so:

(η_{μ}) $\mu\alpha.\langle v\|\alpha\rangle = v$ translates to $[\![\mu\alpha.\langle v\|\alpha\rangle]\!]_{\mathcal{G}} \triangleq \mu\alpha.\langle [\![v]\!]_{\mathcal{G}}\|\alpha\rangle =_{\eta_{\mu}} [\![v]\!]_{\mathcal{G}}$

($\eta_{\tilde{\mu}}$) $\tilde{\mu}x.\langle x\|e\rangle = e$ translates to $[\![\tilde{\mu}x.\langle x\|e\rangle]\!]_{\mathcal{G}} \triangleq \tilde{\mu}x.\langle x\|[\![e]\!]_{\mathcal{G}}\rangle =_{\eta_{\tilde{\mu}}} [\![e]\!]_{\mathcal{G}}$

(μ) $\langle\mu\alpha.c\|E\rangle = c\left\{E/\alpha\right\}$ translates to $[\![\langle\mu\alpha.c\|E\rangle]\!] \triangleq \left\langle\mu\alpha.[\![c]\!]_{\mathcal{G}}\middle\|[\![E]\!]_{\mathcal{G}}\right\rangle =_{\mu} [\![c]\!]_{\mathcal{G}}\left\{[\![E]\!]_{\mathcal{G}}/\alpha\right\} =_{\alpha} [\![c\left\{E/\alpha\right\}]\!]_{\mathcal{G}}$

($\tilde{\mu}$) $\langle V\|\tilde{\mu}x.c\rangle = c\left\{V/x\right\}$ translates to $[\![\langle V\|\tilde{\mu}x.c\rangle]\!] \triangleq \left\langle [\![V]\!]_{\mathcal{G}}\middle\|\tilde{\mu}x.[\![c]\!]_{\mathcal{G}}\right\rangle =_{\tilde{\mu}} [\![c]\!]_{\mathcal{G}}\left\{[\![V]\!]_{\mathcal{G}}/x\right\} =_{\alpha} [\![c\left\{V/x\right\}]\!]_{\mathcal{G}}$

Given **data** $\mathsf{F}(\Theta) : \mathcal{S}$ **where** $\overrightarrow{\mathsf{K}_i : \left(\overrightarrow{A_{i1} : \mathcal{T}_{ij}}^{j} \vdash \mathsf{F}(\Theta) \mid \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^{j}\right)}^{i} \in \mathcal{G}$ we have:

(β^{F}) $\left\langle \mathsf{K}_i(\overrightarrow{e_{ij}}^{j}, \overrightarrow{v_{ij}}^{j})\middle\|\tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}).c_i}^{i}\right]\right\rangle = \left\langle\mu\overrightarrow{\alpha_{ij}}^{j}.\left\langle\overrightarrow{v_{ij}}\middle\|\tilde{\mu}\overrightarrow{x_{ij}}^{j}.c_i\right\rangle\middle|\overrightarrow{e_{ij}}^{j}\right\rangle$ translates by induction on the pattern $\iota_i\left(\overrightarrow{\neg\left[\uparrow\mathcal{U}_{ij}\left[\alpha_{ij}\right]\right]}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right)$ to:

$$
\begin{aligned}
&\left[\!\!\left[\left\langle \mathsf{K}_i(\overrightarrow{e_{ij}}^{j}, \overrightarrow{v_{ij}}^{j})\middle\|\tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}).c_i}^{i}\right]\right\rangle\right]\!\!\right]_{\mathcal{G}}\\
&\triangleq \left\langle \iota_i\left(\overrightarrow{\neg\left[\uparrow\mathcal{U}_{ij}\left[[\![e_{ij}]\!]_{\mathcal{G}}\right]\right]}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle\|\tilde{\mu}\left[\overrightarrow{\iota_i\left(\overrightarrow{\neg\left[\uparrow\mathcal{U}_{ij}\left[\alpha_{ij}\right]\right]}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right).[\![c_i]\!]_{\mathcal{G}}}^{i}\right]\right\rangle\\
&=_{\beta^{\oplus}\eta_{\tilde{\mu}}} \left\langle\left(\overrightarrow{\neg\left[\uparrow\mathcal{U}_{ij}\left[[\![e_{ij}]\!]_{\mathcal{G}}\right]\right]}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle\|\tilde{\mu}\left[\left(\overrightarrow{\neg\left[\uparrow\mathcal{U}_{ij}\left[\alpha_{ij}\right]\right]}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right).[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\\
&\triangleq \left\langle\left(\neg\left[\uparrow\mathcal{U}_{i1}\left[[\![e_{i1}]\!]_{\mathcal{G}}\right]\right], \ldots, \neg\left[\uparrow\mathcal{U}_{im}\left[[\![e_{im}]\!]_{\mathcal{G}}\right]\right], \overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle|\right.\\
&\quad\left.\tilde{\mu}\left[\left(\neg[\uparrow\mathcal{U}_{i1}[\alpha_{i1}]], \ldots, \neg[\uparrow\mathcal{U}_{im}[\alpha_{im}]], \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right).[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\\
&=_{\beta^{\otimes}\eta_{\tilde{\mu}}} \left\langle\neg\left[\uparrow\mathcal{U}_{i1}\left[[\![e_{i1}]\!]_{\mathcal{G}}\right]\right]\middle|\right.\\
&\quad\left.\tilde{\mu}\left[\neg[\uparrow\mathcal{U}_{i1}[\alpha_{i1}]].\ldots.\left\langle\neg\left[\uparrow\mathcal{U}_{im}\left[[\![e_{im}]\!]_{\mathcal{G}}\right]\right]\middle|\right.\right.\right.\\
&\quad\left.\left.\left.\tilde{\mu}\left[\neg[\uparrow\mathcal{U}_{im}[\alpha_{im}]].\left\langle\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle\|\tilde{\mu}\left[\overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}.[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\right]\right\rangle\right]\right\rangle\\
&=_{\beta^{\neg}\eta_{\mu}} \left\langle\tilde{\mu}\left[\uparrow\mathcal{U}_{i1}[\alpha_{i1}].\ldots.\left\langle\tilde{\mu}\left[\uparrow\mathcal{U}_{im}[\alpha_{im}].\left\langle\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle\|\tilde{\mu}\left[\overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}.[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\right]\middle\|\uparrow\mathcal{U}_{im}\left[[\![e_{im}]\!]_{\mathcal{G}}\right]\right\rangle\right]\right|\\
&\quad\left|\uparrow\mathcal{U}_{i1}\left[[\![e_{i1}]\!]_{\mathcal{G}}\right]\right\rangle\\
&=_{\beta^{\uparrow}\eta_{\mu}} \left\langle\mu\alpha_{i1}.\ldots.\left\langle\mu\alpha_{1m}.\left\langle\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle\|\tilde{\mu}\left[\overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}.[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\middle\|[\![e_{im}]\!]_{\mathcal{G}}\right\rangle\middle\|[\![e_{i1}]\!]_{\mathcal{G}}\right\rangle\\
&\triangleq \left\langle\mu\overrightarrow{\alpha_{ij}}^{j}.\left\langle\left(\overrightarrow{\downarrow_{\mathcal{T}_{ij}}\left([\![v_{ij}]\!]_{\mathcal{G}}\right)}^{j}\right)\middle\|\tilde{\mu}\left[\overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}.[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\middle\|\overrightarrow{[\![e_{ij}]\!]_{\mathcal{G}}}^{j}\right\rangle\\
&=_{\beta^{\otimes}\beta^{1}\eta_{\tilde{\mu}}} \left\langle\mu\overrightarrow{\alpha_{ij}}^{j}.\left\langle\downarrow_{\mathcal{T}_{i1}}\left([\![v_{i1}]\!]_{\mathcal{G}}\right)\middle\|\tilde{\mu}\left[\downarrow_{\mathcal{T}_{i1}}(x_{i1}).\ldots.\left\langle\downarrow_{\mathcal{T}_{in}}\left([\![v_{in}]\!]_{\mathcal{G}}\right)\middle\|\tilde{\mu}\left[\downarrow_{\mathcal{T}_{in}}(x_{in}).[\![c_i]\!]_{\mathcal{G}}\right]\right\rangle\right]\right\rangle\middle\|\overrightarrow{[\![e_{ij}]\!]_{\mathcal{G}}}^{j}\right\rangle
\end{aligned}
$$

$$
\begin{aligned}
&=_{\beta^{\downarrow}\eta_{\tilde{\mu}}} \left\langle \mu\overrightarrow{\alpha_{ij}}^{j}.\left\langle [\![v_{i1}]\!]_{\mathcal{G}} \middle\| \tilde{\mu}x_{i1}.\ldots.\left\langle [\![v_{in}]\!]_{\mathcal{G}} \middle\| \tilde{\mu}x_{in}.[\![c_i]\!]_{\mathcal{G}} \right\rangle \right\rangle \middle\| \overrightarrow{[\![e_{ij}]\!]_{\mathcal{G}}}^{j} \right\rangle \\
&\triangleq \left\langle \mu\overrightarrow{\alpha_{ij}}^{j}.\left\langle \overrightarrow{[\![v_{ij}]\!]_{\mathcal{G}}}^{j} \middle\| \tilde{\mu}\overrightarrow{x_{ij}}^{j}.[\![c_i]\!]_{\mathcal{G}} \right\rangle \middle\| \overrightarrow{[\![e_{ij}]\!]_{\mathcal{G}}}^{j} \right\rangle \\
&\triangleq [\![\langle \mu\overrightarrow{\alpha_{ij}}^{j}.\langle \overrightarrow{v_{ij}} \| \tilde{\mu}\overrightarrow{x_{ij}}^{j}.c_i \rangle \| \overrightarrow{e_{ij}}^{j} \rangle]\!]_{\mathcal{G}}
\end{aligned}
$$

(η^{F}) $\beta : \mathsf{F}(\overrightarrow{C}) = \tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}).\left\langle \mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}) \middle\| \beta \right\rangle}^{i}\right]$ translates by induction on the pattern $\iota_i\left(\overrightarrow{\neg\left[\uparrow_{\mathcal{U}_{ij}}[\alpha_{ij}]\right]}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right)$ to:

$$
\begin{aligned}
&\left[\!\left[\tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}).\left\langle \mathsf{K}_i(\overrightarrow{\alpha_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}) \middle\| \beta \right\rangle}^{i}\right]\right]\!\right]_{\mathcal{G}} \\
&\triangleq \tilde{\mu}\left[\overrightarrow{\mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{-\left(\uparrow_{\mathcal{U}_{ij}}[\alpha_{ij}]\right)}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right)\right).\left\langle \mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{-\left(\uparrow_{\mathcal{U}_{ij}}[\alpha_{ij}]\right)}^{j}, \overrightarrow{\downarrow_{\mathcal{T}_{ij}}(x_{ij})}^{j}\right)\right) \middle\| \beta \right\rangle}^{i}\right] \\
&=_{\tilde{\mu}\eta_{\mathcal{V}}^{\downarrow}} \tilde{\mu}\left[\overrightarrow{\mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{-\left(\uparrow_{\mathcal{U}_{ij}}[\alpha_{ij}]\right)}^{j}, \overrightarrow{x_{ij}}^{j}\right)\right).\left\langle \mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{-\left(\uparrow_{\mathcal{U}_{ij}}[\alpha_{ij}]\right)}^{j}, \overrightarrow{x_{ij}}^{j}\right)\right) \middle\| \beta \right\rangle}^{i}\right] \\
&=_{\tilde{\mu}\eta_{\mathcal{V}}^{\uparrow}} \tilde{\mu}\left[\overrightarrow{\mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{-(\alpha_{ij})}^{j}, \overrightarrow{x_{ij}}^{j}\right)\right).\left\langle \mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{-(\alpha_{ij})}^{j}, \overrightarrow{x_{ij}}^{j}\right)\right) \middle\| \beta \right\rangle}^{i}\right] \\
&=_{\tilde{\mu}\eta_{\mathcal{V}}^{-}} \tilde{\mu}\left[\overrightarrow{\mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{y_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}\right)\right).\left\langle \mathcal{S}\Uparrow\left(\iota_i\left(\overrightarrow{y_{ij}}^{j}, \overrightarrow{x_{ij}}^{j}\right)\right) \middle\| \beta \right\rangle}^{i}\right] \\
&=_{\tilde{\mu}\eta_{\mathcal{V}}^{\otimes}\eta_{\mathcal{V}}^{1}} \tilde{\mu}\left[\overrightarrow{\mathcal{S}\Uparrow(\iota_i(x)).\langle \mathcal{S}\Uparrow(\iota_i(x)) \| \beta \rangle}^{i}\right] \\
&=_{\tilde{\mu}\eta_{\mathcal{V}}^{\oplus}} \tilde{\mu}[\mathcal{S}\Uparrow(x).\langle \mathcal{S}\Uparrow(x) \| \beta \rangle] \\
&=_{\eta^{\Uparrow}} \beta
\end{aligned}
$$

Given **codata** $\mathrm{G}(\Theta) : \mathcal{S}$ **where** $\overrightarrow{\mathrm{O}_i : \left(\overrightarrow{A_{ij} : \mathcal{T}_{ij}}^{j} \mid \mathrm{G}(\Theta) \vdash \overrightarrow{B_{ij} : \mathcal{U}_{ij}}^{j}\right)}^{i} \in \mathcal{G}$ we have:

(β^{G}) is analogous to the translation of β^{F} by duality.
(η^{G}) is analogous to the translation of η^{F} by duality. □

B PROOFS OF TYPE ISOMORPHISM PROPERTIES

Theorem B.1. *Two types A and B are isomorphic if and only if there are two closed values $V_1 : A \longrightarrow B$ and $V_2 : B \longrightarrow A$ such that the following equalities hold:*

$$V_2 \circ V_1 = \lambda x.x : A \longrightarrow A \qquad\qquad V_1 \circ V_2 = \lambda y.y : B \longrightarrow B$$

Proof. Recall that the λ-abstraction $\lambda x.v$ is syntactic sugar for the object $\mu(x \cdot \alpha.\langle v \| \alpha \rangle)$, so that the identity function $\lambda x.x$ expands to $\mu(x \cdot \alpha.\langle x \| \alpha \rangle)$. Additionally, the composition of functions, $f \circ g$, is defined in the sequent calculus as:

$$f \circ g \triangleq \mu(x \cdot \alpha.\langle f \| \mu\beta.\, \langle g \| x \cdot \beta \rangle \cdot \alpha \rangle)$$

First, we show that an isomorphism between types A and B gives us two inverse functions between types A and B. Let $c_1 : (x : A \vdash \beta : B)$ and $c_2 : (y : B \vdash \alpha : A)$ be the open commands given by the isomorphism. These commands can be closed as the two functional objects:

$$V_1 : A \to B \triangleq \mu(x \cdot \beta.c_1) \qquad\qquad V_2 : B \to A \triangleq \mu(y \cdot \alpha.c_2)$$

To show that these are inverses, we have $V_2 \circ V_1$:

$$\begin{aligned} V_2 \circ V_1 &\triangleq \mu(x \cdot \alpha.\langle V_2 \| \mu\beta.\, \langle V_1 \| x \cdot \beta\rangle \cdot \alpha\rangle) \\ &=_{\beta^{\to}\mu\tilde{\mu}} \mu(x \cdot \alpha.\langle V_2 \| \mu\beta.c_1 \cdot \alpha\rangle) \\ &=_{\beta^{\to}\mu} \mu(x \cdot \alpha.\langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle) \\ &=_{Iso} \mu(x \cdot \alpha.\langle x \| \alpha\rangle) \\ &\triangleq \lambda x.x \end{aligned}$$

The fact that $V_1 \circ V_2 = \lambda y.y$ follows analogously.

Second, we show that two inverse functions between types A and B give us an isomorphism between types A and B. Let $V_1 : A \to B$ and $V_2 : B \to A$ be the closed inverse functions. These functions can be applied as the two open commands:

$$c_1 \triangleq \langle V_1 \| x \cdot \beta\rangle : (x : A \vdash \beta : B) \qquad\qquad c_2 \triangleq \langle V_2 \| y \cdot \alpha\rangle : (y : B \vdash \alpha : A)$$

To show that these commands form an isomorphism, we have:

$$\begin{aligned} \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle &\triangleq \langle \mu\beta.\, \langle V_1 \| x \cdot \beta\rangle \| \tilde{\mu}y.\, \langle V_2 \| y \cdot \alpha\rangle\rangle \\ &=_{\varsigma^{\to}} \langle V_2 \| \mu\beta.\, \langle V_1 \| x \cdot \beta\rangle \cdot \alpha\rangle \\ &=_{\beta^{\to}\mu\tilde{\mu}} \langle \mu(\langle V_2 \| \mu\beta.\, \langle V_1 \| x \cdot \beta\rangle \cdot \alpha\rangle) \| x \cdot \alpha\rangle \\ &\triangleq \langle V_2 \circ V_1 \| x \cdot \alpha\rangle \\ &=_{Inv} \langle \mu(x \cdot \alpha.\langle x \| \alpha\rangle) \| x \cdot \alpha\rangle \\ &=_{\beta^{\to}\mu\tilde{\mu}} \langle x \| \alpha\rangle \end{aligned}$$

The fact that $\langle \mu\alpha.c_2 \| \tilde{\mu}x.c_1\rangle = \langle y \| \beta\rangle$ follows analogously. □

Lemma B.1 (Linear and Thunkable Composition). *For any commands $c_1 : (x : A \vdash \beta : B)$ and $c_2 : (y : B \vdash \alpha : A)$,*

a) if $\tilde{\mu}x.c_1$ and $\tilde{\mu}y.c_2$ are linear then $\tilde{\mu}x.\, \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle$ is also linear, and
b) if $\mu\alpha.c_2$ and $\mu\beta.c_1$ are thunkable then $\mu\alpha.\, \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle$ is also thunkable.

Proof. a) Let v be any term and c be any command such that $\beta, \gamma \notin FV(v)$. The linearity of the co-term $\tilde{\mu}x.\, \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle$ follows from the linearity of both $\tilde{\mu}x.c_1$ and $\tilde{\mu}y.c_2$:

$$\begin{aligned} \langle \mu\gamma.\, \langle v \| \tilde{\mu}z.c\rangle \| \tilde{\mu}x.\langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle\rangle &=_{Lin} \langle \mu\beta.\, \langle \mu\gamma.\, \langle v \| \tilde{\mu}z.c\rangle \| \tilde{\mu}x.c_1\rangle \| \tilde{\mu}y.c_2\rangle \\ &=_{Lin} \langle \mu\beta.\, \langle v \| \tilde{\mu}z.\, \langle \mu\gamma.c \| \tilde{\mu}x.c_1\rangle\rangle \| \tilde{\mu}y.c_2\rangle \\ &=_{Lin} \langle v \| \tilde{\mu}z.\, \langle \mu\beta.\, \langle \mu\gamma.c \| \tilde{\mu}x.c_1\rangle \| \tilde{\mu}y.c_2\rangle\rangle \\ &=_{Lin} \langle v \| \tilde{\mu}z.\, \langle \mu\gamma.c \| \tilde{\mu}x.\, \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle\rangle\rangle \end{aligned}$$

b) Let e be any co-term and c be any command such that $y, z \notin FV(e)$. The thunkability of the term $\mu\alpha.\langle\mu\beta.c_1 \| \tilde{\mu}y.c_2\rangle$ follows from the thunkability of both $\mu\alpha.c_2$ and $\mu\beta.c_1$:

$$\begin{aligned}\langle\mu\alpha.\langle\mu\beta.c_1\|\tilde{\mu}y.c_2\rangle\|\tilde{\mu}z.\langle\mu\gamma.c\|e\rangle\rangle &=_{Thk} \langle\mu\beta.c_1\|\tilde{\mu}y.\langle\mu\alpha.c_2\|\tilde{\mu}z.\langle\mu\gamma.c\|e\rangle\rangle\rangle \\ &=_{Thk} \langle\mu\beta.c_1\|\tilde{\mu}y.\langle\mu\gamma.\langle\mu\alpha.c_2\|\tilde{\mu}z.c\rangle\|e\rangle\rangle \\ &=_{Thk} \langle\mu\gamma.\langle\mu\beta.c_1\|\tilde{\mu}y.\langle\mu\alpha.c_2\|\tilde{\mu}z.c\rangle\rangle\|e\rangle \\ &=_{Thk} \langle\mu\gamma.\langle\mu\alpha.\langle\mu\beta.c_1\|\tilde{\mu}y.c_2\rangle\|\tilde{\mu}z.c\rangle\|e\rangle \qquad \square\end{aligned}$$

Lemma B.2. *a) v is linear if $v = V$ for some value V.*
b) e is thunkable if $e = E$ for some co-value E.

Proof. a) Suppose we have any command c, co-term e, variable $x \notin FV(e)$ and co-variable $\alpha \notin FV(v)$. Because we assumed that $v = V$ for some value V, we have the following equality:

$$\begin{aligned}\langle v\|\tilde{\mu}x.\langle\mu\alpha.c\|e\rangle\rangle &= \langle V\|\tilde{\mu}x.\langle\mu\alpha.c\|e\rangle\rangle \\ &=_{\tilde{\mu}} \langle\mu\alpha.c\,\{V/x\}\|e\rangle \\ &=_{\tilde{\mu}} \langle\mu\alpha.\langle V\|\tilde{\mu}x.c\rangle\|e\rangle \\ &= \langle\mu\alpha.\langle v\|\tilde{\mu}x.c\rangle\|e\rangle\end{aligned}$$

b) Analogous to the proof of lemma B.2 (a) by duality. $\square$

Lemma B.3. *a) For any types $A : \mathcal{V}$ and $B : \mathcal{V}$, $A \approx B$ if and only if $A \approx^+ B$.*
b) For any types $A : \mathcal{N}$ and $B : \mathcal{N}$, $A \approx B$ if and only if $A \approx^- B$.

Proof. Note that both $A \approx^+ B$ and $A \approx^- B$ imply $A \approx B$ by definition, so we only need to show that $A \approx B$ implies $A \approx^+ B$ and $A \approx^- B$ in the appropriate case. Suppose that $A \approx B$ is witnessed by the commands $c : (x : A \vdash \beta : B)$ and $c' : (y : B \vdash \alpha : A)$. Note that *every* $\mathcal{V}$-co-term is a $\mathcal{V}$-co-value (*i.e.,* $e \in CoValue_{\mathcal{V}}$ for all $e : A : \mathcal{V}$), so all co-terms $e : A : \mathcal{V}$ are trivially linear by lemma B.2 (b). Thus, when $A : \mathcal{V}$ and $B : \mathcal{V}$, $\tilde{\mu}x.c : A : \mathcal{V}$ and $\tilde{\mu}y.c' : B : \mathcal{V}$ are linear as required by $A \approx^+ B$. Dually, note that *every* $\mathcal{N}$-term is a $\mathcal{N}$-value (*i.e.,* $v \in Value_{\mathcal{N}}$ for all $v : A : \mathcal{N}$), so all terms $v : A : \mathcal{N}$ are trivially thunkable by lemma B.2 (a). Thus, when $A : \mathcal{N}$ and $B : \mathcal{N}$, $\mu\alpha.c' : A : \mathcal{N}$ and $\mu\beta.c : B : \mathcal{N}$ are thunkable as required by $A \approx^- B$. $\square$

Theorem 4.3 (Polarized Isomorphism Equivalence). *a) $A \approx^+ A$ and $A \approx^- A$,*
b) if $A \approx^+ B$ then $B \approx^+ A$ and if $A \approx^- B$ then $B \approx^- A$, and
c) if $A \approx^+ B$ and $B \approx^+ C$ then $A \approx^+ C$ and if $A \approx^- B$ and $B \approx^- C$ then $A \approx^- C$.

Proof. a) Note that in the witness of isomorphism reflexivity in the proof of theorem 4.2 (a), $\langle x\|\alpha\rangle : (x : A \vdash \alpha : A)$, we have that $\tilde{\mu}x.\langle x\|\alpha\rangle =_{\eta_{\tilde{\mu}}} \alpha$ is linear and $\mu\alpha.\langle x\|\alpha\rangle$ is thunkable via lemma B.2, so the reflexive isomorphism is both positive and negative.

b) The symmetry of positive and negative isomorphisms follows immediately from their symmetric definition.

c) Suppose that $A \approx^+ B$ is witnessed by the commands $c_1 : (x : A \vdash \beta : B)$ and $c_2 : (y : B \vdash \alpha : A)$ and $B \approx^+ C$ is witnessed by the commands $B \approx^+ C$ is witnessed by the commands $c_3 : (y' : B \vdash \gamma : C)$ and $c_4 : (z : C \vdash \beta' : B)$. The isomorphism $A \approx^+ C$ is then established by composing c_1 with c_3 and c_2 with c_4 as follows:

$$c_5 \triangleq \langle\mu\beta.c_1\|\tilde{\mu}y'.c_3\rangle : (x : A \vdash \gamma : C) \qquad c_6 \triangleq \langle\mu\beta'.c_4\|\tilde{\mu}y.c_2\rangle : (z : C \vdash \alpha : A)$$

From the linearity of $\tilde{\mu}y.c_2$ and $\tilde{\mu}z.c_4$ implied by $A \approx^+ B$ and $B \approx^+ C$, we get that the composition of c_5 and c_6 is the identity command $\langle x \| \alpha \rangle : (x : A \vdash \alpha : A)$ as follows:

$$\begin{aligned}
\langle \mu\gamma.c_5 \| \tilde{\mu}z.c_6 \rangle &\triangleq \langle \mu\gamma.\langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle \| \tilde{\mu}z.\langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle \rangle \\
&=_{Lin} \langle \mu\beta'.\langle \mu\gamma.\langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle \| \tilde{\mu}z.c_4 \rangle \| \tilde{\mu}y.c_2 \rangle \\
&=_{Lin} \langle \mu\beta'.\langle \mu\beta.c_1 \| \tilde{\mu}y'.\langle \mu\gamma.c_3 \| \tilde{\mu}z.c_4 \rangle \rangle \| \tilde{\mu}y.c_2 \rangle \\
&=_{Iso} \langle \mu\beta'.\langle \mu\beta.c_1 \| \tilde{\mu}y'.\langle y' \| \beta' \rangle \rangle \| \tilde{\mu}y.c_2 \rangle \\
&=_{\eta_{\tilde{\mu}}} \langle \mu\beta'.\langle \mu\beta.c_1 \| \beta' \rangle \| \tilde{\mu}y.c_2 \rangle \\
&=_{\eta_{\mu}} \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2 \rangle \\
&=_{Iso} \langle x \| \alpha \rangle
\end{aligned}$$

And from the linearity of $\tilde{\mu}y'.c_3$ and $\tilde{\mu}x.c_1$ implied by $A \approx^+ B$ and $B \approx^+ C$, we get that the composition of c_6 and c_5 is the identity command $\langle z \| \gamma \rangle : (z : C \vdash \gamma : C)$ as follows:

$$\begin{aligned}
\langle \mu\alpha.c_6 \| \tilde{\mu}x.c_5 \rangle &\triangleq \langle \mu\alpha.\langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle \| \tilde{\mu}x.\langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle \rangle \\
&=_{Lin} \langle \mu\beta.\langle \mu\alpha.\langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle \| \tilde{\mu}x.c_1 \rangle \| \tilde{\mu}y'.c_3 \rangle \\
&=_{Lin} \langle \mu\beta.\langle \mu\beta'.c_4 \| \tilde{\mu}y.\langle \mu\alpha.c_2 \| \tilde{\mu}x.c_1 \rangle \rangle \| \tilde{\mu}y'.c_3 \rangle \\
&=_{Iso} \langle \mu\beta.\langle \mu\beta'.c_4 \| \tilde{\mu}y.\langle y \| \beta \rangle \rangle \| \tilde{\mu}y'.c_3 \rangle \\
&=_{\eta_{\tilde{\mu}}} \langle \mu\beta.\langle \mu\beta'.c_4 \| \beta \rangle \| \tilde{\mu}y'.c_3 \rangle =_{\eta_{\mu}} \langle \mu\beta'.c_4 \| \tilde{\mu}y'.c_3 \rangle =_{Iso} \langle z \| \gamma \rangle
\end{aligned}$$

Finally, we get that the co-terms $\tilde{\mu}x.c_5$ and $\tilde{\mu}z.c_6$ are linear from lemma B.1 (a).
The transitivity of negative type isomorphisms are similar to the positive case. Suppose that $A \approx^- B$ is witnessed by the commands $c_1 : (x : A \vdash \beta : B)$ and $c_2 : (y : B \vdash \alpha : A)$ and $B \approx^- C$ is witnessed by the commands $B \approx^+ C$ is witnessed by the commands $c_3 : (y' : B \vdash \gamma : C)$ and $c_4 : (z : C \vdash \beta' : B)$. The isomorphism $A \approx^- C$ is then established by composing c_1 with c_3 and c_2 with c_4 as follows:

$$c_5 \triangleq \langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle : (x : A \vdash \gamma : C) \qquad c_6 \triangleq \langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle : (z : C \vdash \alpha : A)$$

From the thunkability of $\mu\beta.c_1$ and $\mu\gamma.c_3$ implied by $A \approx^- B$ and $B \approx^- C$, we get that the composition of c_5 and c_6 is the identity command $\langle x \| \alpha \rangle : (x : A \vdash \alpha : A)$ as follows:

$$\begin{aligned}
\langle \mu\gamma.c_5 \| \tilde{\mu}z.c_6 \rangle &\triangleq \langle \mu\gamma.\langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle \| \tilde{\mu}z.\langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle \rangle \\
&=_{Thk} \langle \mu\beta.c_1 \| \tilde{\mu}y'.\langle \mu\gamma.c_3 \| \tilde{\mu}z.\langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle \rangle \rangle \\
&=_{Thk} \langle \mu\beta.c_1 \| \tilde{\mu}y'.\langle \mu\beta'.\langle \mu\gamma.c_3 \| \tilde{\mu}z.c_4 \rangle \| \tilde{\mu}y.c_2 \rangle \rangle \\
&=_{Iso} \langle \mu\beta.c_1 \| \tilde{\mu}y'.\langle \mu\beta'.\langle y' \| \beta' \rangle \| \tilde{\mu}y.c_2 \rangle \rangle \\
&=_{\eta_{\mu}} \langle \mu\beta.c_1 \| \tilde{\mu}y'.\langle y' \| \tilde{\mu}y.c_2 \rangle \rangle \\
&=_{\eta_{\tilde{\mu}}} \langle \mu\beta.c_1 \| \tilde{\mu}y.c_2 \rangle \\
&=_{Iso} \langle x \| \alpha \rangle
\end{aligned}$$

And from the thunkability of $\mu\beta'.c_2$ and $\mu\alpha.c_2$ implied by $A \approx^- B$ and $B \approx^- C$, we get that the composition of c_6 and c_5 is the identity command $\langle z \| \gamma \rangle : (z : C \vdash \gamma : C)$ as follows:

$$\begin{aligned}
\langle \mu\alpha.c_6 \| \tilde{\mu}x.c_5 \rangle &\triangleq \langle \mu\alpha.\langle \mu\beta'.c_4 \| \tilde{\mu}y.c_2 \rangle \| \tilde{\mu}x.\langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle \rangle \\
&=_{Thk} \langle \mu\beta'.c_4 \| \tilde{\mu}y.\, \langle \mu\alpha.c_2 \| \tilde{\mu}x.\langle \mu\beta.c_1 \| \tilde{\mu}y'.c_3 \rangle \rangle \rangle \\
&=_{Thk} \langle \mu\beta'.c_4 \| \tilde{\mu}y.\, \langle \mu\beta.\, \langle \mu\alpha.c_2 \| \tilde{\mu}x.c_1 \rangle \| \tilde{\mu}y'.c_3 \rangle \rangle \\
&=_{Iso} \langle \mu\beta'.c_4 \| \tilde{\mu}y.\, \langle \mu\beta.\, \langle y \| \beta \rangle \| \tilde{\mu}y'.c_3 \rangle \rangle \\
&=_{\eta_\mu} \langle \mu\beta'.c_4 \| \tilde{\mu}y.\, \langle y \| \tilde{\mu}y'.c_3 \rangle \rangle \\
&=_{\eta_{\tilde{\mu}}} \langle \mu\beta'.c_4 \| \tilde{\mu}y'.c_3 \rangle \\
&=_{Iso} \langle z \| \gamma \rangle
\end{aligned}$$

Finally, we get that the terms $\mu\gamma.c_5$ and $\mu\alpha.c_6$ are thunkable from lemma B.1 (b). □

C PROOFS OF THE (CO-)DATA STRUCTURAL LAWS

Lemma C.1 (Data commute instance). $\mathsf{F}(\overrightarrow{C}) \approx^+ \mathsf{F}'(\overrightarrow{C'})$ *for any* $\overrightarrow{C : \mathcal{S}}$, $\overrightarrow{C' : \mathcal{S}'}$ *and declarations*

a) **data** $\mathsf{F}(\overrightarrow{X{:}\mathcal{S}}){:}\mathcal{V}$ **where** $\mathsf{K}{:}(\Gamma_2, \Gamma_1 \vdash \mathsf{F}(\overrightarrow{X}) \mid \Delta_1, \Delta_2)$ *and* **data** $\mathsf{F}'(\overrightarrow{X'{:}\mathcal{S}'}){:}\mathcal{V}$ **where** $\mathsf{K}'{:}(\Gamma_1', \Gamma_2' \vdash \mathsf{F}'(\overrightarrow{X'}) \mid \Delta_2', \Delta_1')$

such that $\Gamma_1\theta = \Gamma_1'\theta'$, $\Gamma_2\theta = \Gamma_2'\theta'$, $\Delta_1\theta = \Delta_1'\theta'$, $\Delta_1\theta = \Delta_1'\theta'$, $\theta = \overrightarrow{\{C/X\}}$, *and* $\theta' = \overrightarrow{\{C'/X'\}}$, *or*

b) **data** $\mathsf{F}(\overrightarrow{X{:}\mathcal{S}}){:}\mathcal{V}$ **where** $\overrightarrow{\mathsf{K}_1{:}(\Gamma_1 \vdash \mathsf{F}(\overrightarrow{X}) \mid \Delta_1)}$ $\overrightarrow{\mathsf{K}_2{:}(\Gamma_2 \vdash \mathsf{F}(\overrightarrow{X}) \mid \Delta_2)}$ *and* **data** $\mathsf{F}'(\overrightarrow{X'{:}\mathcal{S}'}){:}\mathcal{V}$ **where** $\overrightarrow{\mathsf{K}_2'{:}(\Gamma_2' \vdash \mathsf{F}'(\overrightarrow{X'}) \mid \Delta_2')}$ $\overrightarrow{\mathsf{K}_1'{:}(\Gamma_1' \vdash \mathsf{F}'(\overrightarrow{X'}) \mid \Delta_1')}$

such that $\overrightarrow{\Gamma_1\theta = \Gamma_1'\theta'}$, $\overrightarrow{\Gamma_2\theta = \Gamma_2'\theta'}$, $\overrightarrow{\Delta_1\theta = \Delta_1'\theta'}$, $\overrightarrow{\Delta_2\theta = \Delta_2'\theta'}$, $\theta = \overrightarrow{\{C/X\}}$, *and* $\theta' = \overrightarrow{\{C'/X'\}}$.

Proof. The isomorphisms between $\mathsf{F}(\overrightarrow{C})$ and $\mathsf{F}'(\overrightarrow{C'})$ are established by $c : (x : \mathsf{F}(\overrightarrow{C}) \vdash \alpha' : \mathsf{F}'(\overrightarrow{C'}))$ and $c' : (x' : \mathsf{F}'(\overrightarrow{C'}) \vdash \alpha : \mathsf{F}(\overrightarrow{C}))$ as follows:

a)
$$\begin{aligned}
c &\triangleq \langle x \| \tilde{\mu}[\mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}).\langle \mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}) \| \alpha' \rangle] \rangle \\
c' &\triangleq \langle x' \| \tilde{\mu}[\mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}).\langle \mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}) \| \alpha \rangle] \rangle
\end{aligned}$$

b)
$$c \triangleq \left\langle x \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}_1(\overrightarrow{\beta_1}, \overrightarrow{y_1}).\langle \mathsf{K}_1'(\overrightarrow{\beta_1}, \overrightarrow{y_1}) \| \alpha' \rangle \\ \mathsf{K}_2(\overrightarrow{\beta_2}, \overrightarrow{y_2}).\langle \mathsf{K}_2'(\overrightarrow{\beta_2}, \overrightarrow{y_2}) \| \alpha' \rangle \end{bmatrix} \right\rangle \qquad c' \triangleq \left\langle x' \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}_2'(\overrightarrow{\beta_2}, \overrightarrow{y_2}).\langle \mathsf{K}_2(\overrightarrow{\beta_2}, \overrightarrow{y_2}) \| \alpha \rangle \\ \mathsf{K}_1'(\overrightarrow{\beta_1}, \overrightarrow{y_1}).\langle \mathsf{K}_1(\overrightarrow{\beta_1}, \overrightarrow{y_1}) \| \alpha \rangle \end{bmatrix} \right\rangle$$

And note that since both $\mathsf{F}(\overrightarrow{C}) : \mathcal{V}$ and $\mathsf{F}'(\overrightarrow{C'}) : \mathcal{V}$, the isomorphism must be positive (lemma B.3 (a)).

For part (a), the composition of c' and c along α and x of type $\mathsf{F}(\overrightarrow{C})$ is equal to the identity command $\langle x' \| \alpha' \rangle$ via the β^{F} and $\eta^{\mathsf{F}'}$ axioms as follows:

$$\begin{aligned}
&\langle \mu\alpha.c' \| \tilde{\mu}x.c \rangle \\
&\triangleq \left\langle \mu\alpha.\langle x' \| \tilde{\mu}[\mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}).\langle \mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}) \| \alpha \rangle] \rangle \middle| \mu x.\langle x \| \tilde{\mu}[\mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}).\langle \mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}) \| \alpha' \rangle] \rangle \right\rangle \\
&=_{\eta_{\tilde{\mu}}} \left\langle \mu\alpha.\langle x' \| \tilde{\mu}[\mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}).\langle \mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}) \| \alpha \rangle] \rangle \middle| \tilde{\mu}[\mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}).\langle \mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}) \| \alpha' \rangle] \right\rangle \\
&=_{\mu} \langle x' \| \tilde{\mu}[\mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}).\langle \mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}) \| \tilde{\mu}[\mathsf{K}(\overrightarrow{\beta_1}, \overrightarrow{\beta_2}, \overrightarrow{y_2}, \overrightarrow{y_1}).\langle \mathsf{K}'(\overrightarrow{\beta_2}, \overrightarrow{\beta_1}, \overrightarrow{y_1}, \overrightarrow{y_2}) \| \alpha' \rangle] \rangle] \rangle
\end{aligned}$$

$$=_{\beta^{\mathsf{F}}\mu\tilde{\mu}} \langle x' \| \tilde{\mu}[\mathsf{K}'(\vec{\beta_2}, \vec{\beta_1}, \vec{y_1}, \vec{y_2}).\langle \mathsf{K}'(\vec{\beta_2}, \vec{\beta_1}, \vec{y_1}, \vec{y_2}) \| \alpha' \rangle] \rangle$$
$$=_{\eta^{\mathsf{F}'}} \langle x' \| \alpha' \rangle$$

And the reverse composition of c and c' along α' and x' of type $\mathsf{F}'(\vec{C'})$ is equal to the identity command $\langle x \| \alpha \rangle$ via the $\beta^{\mathsf{F}'}$ and η^{F}.

For part (b), the composition of c' and c along α and x of type $\mathsf{F}(\vec{C})$ is equal to the identity command $\langle x' \| \alpha' \rangle$ via the β^{F} and $\eta^{\mathsf{F}'}$ axioms as follows:

$$\langle \mu\alpha.c' \| \tilde{\mu}x.c \rangle$$
$$\triangleq \left\langle \mu\alpha. \left\langle x' \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}) \| \alpha \rangle \\ \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}_1(\vec{\beta_1}, \vec{y_1}) \| \alpha \rangle \end{bmatrix} \right\rangle \middle\| \tilde{\mu}x. \left\langle x \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}) \| \alpha' \rangle \\ \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}) \| \alpha' \rangle \end{bmatrix} \right\rangle \right\rangle$$
$$=_{\eta_{\tilde{\mu}}} \left\langle \mu\alpha. \left\langle x' \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}) \| \alpha \rangle \\ \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}_1(\vec{\beta_1}, \vec{y_1}) \| \alpha \rangle \end{bmatrix} \right\rangle \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}) \| \alpha' \rangle \\ \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}) \| \alpha' \rangle \end{bmatrix} \right\rangle$$
$$=_{\mu} \left\langle x' \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}).\left\langle \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}) \middle\| \tilde{\mu} \left[\mathsf{K}_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}) \| \alpha' \rangle \mid \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}) \| \alpha' \rangle \right] \right\rangle \\ \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}).\left\langle \mathsf{K}_1(\vec{\beta_1}, \vec{y_1}) \middle\| \tilde{\mu} \left[\mathsf{K}_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}) \| \alpha' \rangle \mid \mathsf{K}_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}) \| \alpha' \rangle \right] \right\rangle \end{bmatrix} \right\rangle$$
$$=_{\beta^{\mathsf{F}}\mu\tilde{\mu}} \left\langle x' \middle\| \tilde{\mu} \begin{bmatrix} \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}).\langle \mathsf{K}'_2(\vec{\beta_2}, \vec{y_2}) \| \alpha' \rangle \\ \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}).\langle \mathsf{K}'_1(\vec{\beta_1}, \vec{y_1}) \| \alpha' \rangle \end{bmatrix} \right\rangle$$
$$=_{\eta^{\mathsf{F}'}} \langle x' \| \alpha' \rangle$$

And the reverse composition of c and c' along α' and x' of type $\mathsf{F}'(\vec{C'})$ is equal to the identity command $\langle x \| \alpha \rangle$ via the $\beta^{\mathsf{F}'}$ and η^{F}. □

Lemma C.2 (Data mix instance). *For any types and* $\overrightarrow{C : \mathcal{S}}$, $\overrightarrow{C' : \mathcal{S}'}$ *data declarations*

$$\textbf{data } \mathsf{F}_1(\overrightarrow{X : \mathcal{S}}) : \mathcal{V} \textbf{ where} \qquad \overrightarrow{\mathsf{K}_1 : \left(\Gamma_1 \vdash \mathsf{F}_1(\vec{X}) \mid \Delta_1\right)}$$
$$\textbf{data } \mathsf{F}'_1(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{V} \textbf{ where} \qquad \overrightarrow{\mathsf{K}'_1 : \left(\Gamma'_1 \vdash \mathsf{F}'_1(\vec{X'}) \mid \Delta'_1\right)}$$
$$\textbf{data } \mathsf{F}_2(\overrightarrow{X : \mathcal{S}}) : \mathcal{V} \textbf{ where} \qquad \overrightarrow{\mathsf{K}_2 : \left(\Gamma_2 \vdash \mathsf{F}_2(\vec{X}) \mid \Delta_2\right)}$$
$$\textbf{data } \mathsf{F}'_2(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{V} \textbf{ where} \qquad \overrightarrow{\mathsf{K}'_2 : \left(\Gamma'_2 \vdash \mathsf{F}'_2(\vec{X'}) \mid \Delta'_2\right)}$$
$$\textbf{data } \mathsf{F}_3(\overrightarrow{X : \mathcal{S}}) : \mathcal{V} \textbf{ where} \qquad \mathsf{K}_3 : \left(\Gamma_3 \vdash \mathsf{F}_3(\vec{X}) \mid \Delta_3\right)$$
$$\textbf{data } \mathsf{F}'_3(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{V} \textbf{ where} \qquad \mathsf{K}'_3 : \left(\Gamma'_3 \vdash \mathsf{F}'_3(\vec{X'}) \mid \Delta'_3\right)$$
$$\textbf{data } \mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{V} \textbf{ where} \qquad \overrightarrow{\mathsf{K}_4 : \left(\Gamma_3, \Gamma_1 \vdash \mathsf{F}(\vec{X}) \mid \Delta_1, \Delta_3\right)} \quad \overrightarrow{\mathsf{K}_5 : \left(\Gamma_3, \Gamma_2 \vdash \mathsf{F}(\vec{X}) \mid \Delta_2, \Delta_3\right)}$$
$$\textbf{data } \mathsf{F}'(\overrightarrow{X' : \mathcal{S}}) : \mathcal{V} \textbf{ where} \qquad \overrightarrow{\mathsf{K}'_4 : \left(\Gamma'_3, \Gamma'_1 \vdash \mathsf{F}'(\vec{X'}) \mid \Delta'_1, \Delta'_3\right)} \quad \overrightarrow{\mathsf{K}'_5 : \left(\Gamma'_3, \Gamma'_2 \vdash \mathsf{F}'(\vec{X'}) \mid \Delta'_2, \Delta'_3\right)}$$

if $\mathsf{F}_1(\vec{C}) \approx \mathsf{F}'_1(\vec{C'})$, $\mathsf{F}_2(\vec{C}) \approx \mathsf{F}'_2(\vec{C'})$, *and* $\mathsf{F}_3(\vec{C}) \approx \mathsf{F}'_3(\vec{C'})$, *then* $\mathsf{F}(\vec{C}) \approx^{+} \mathsf{F}'(\vec{C'})$.

Proof. Suppose that the isomorphisms $\mathsf{F}_1(\vec{C}) \approx \mathsf{F}'_1(\vec{C'})$, $\mathsf{F}_2(\vec{C}) \approx \mathsf{F}'_2(\vec{C'})$, and $\mathsf{F}_3(\vec{C}) \approx \mathsf{F}'_3(\vec{C'})$ are witnessed by the commands

$$c_1 : (x_1 : \mathsf{F}_1(\vec{C}) \vdash \alpha'_1 : \mathsf{F}'_1(\vec{C'})) \quad c_2 : (x_2 : \mathsf{F}_2(\vec{C}) \vdash \alpha'_2 : \mathsf{F}'_2(\vec{C'})) \quad c_3 : (x_3 : \mathsf{F}_3(\vec{C}) \vdash \alpha'_3 : \mathsf{F}'_3(\vec{C'}))$$
$$c'_1 : (x'_1 : \mathsf{F}'_1(\vec{C'}) \vdash \alpha_1 : \mathsf{F}_1(\vec{C})) \quad c'_2 : (x'_2 : \mathsf{F}'_2(\vec{C'}) \vdash \alpha_2 : \mathsf{F}_2(\vec{C})) \quad c'_3 : (x'_3 : \mathsf{F}'_3(\vec{C'}) \vdash \alpha_3 : \mathsf{F}_3(\vec{C}))$$

respectively. Then isomorphisms between $\mathsf{F}(\vec{C})$ and $\mathsf{F}'(\vec{C'})$ are established by $c : (x : \mathsf{F}(\vec{C}) \vdash \alpha' : \mathsf{F}'(\vec{C'}))$ and $c' : (x' : \mathsf{F}'(\vec{C'}) \vdash \alpha : \mathsf{F}(\vec{C}))$ as follows:

$$c \triangleq \left\langle x \middle\| \tilde{\mu} \left[\begin{array}{l} \mathsf{K}_{4i}(\vec{\beta_{1i}}, \vec{\beta_3}, \vec{y_3}, \vec{y_{1i}}).\left\langle v'_{1i} \middle\| \tilde{\mu}\left[\mathsf{K}'_{1i}(\vec{\beta'_{1j}}, \vec{y'_{1j}}).\left\langle v'_3 \middle\| \tilde{\mu}\left[\mathsf{K}'_3(\vec{\beta'_3}, \vec{y'_3}).\left\langle \mathsf{K}'_{4j}(\vec{\beta'_{1j}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{1j}}) \middle\| \alpha' \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \\ \mathsf{K}_{5i}(\vec{\beta_{2i}}, \vec{\beta_3}, \vec{y_3}, \vec{y_{2i}}).\left\langle v'_{2i} \middle\| \tilde{\mu}\left[\mathsf{K}'_{2i}(\vec{\beta'_{2j}}, \vec{y'_{2j}}).\left\langle v'_3 \middle\| \tilde{\mu}\left[\mathsf{K}'_3(\vec{\beta'_3}, \vec{y'_3}).\left\langle \mathsf{K}'_{5j}(\vec{\beta'_{2j}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{2j}}) \middle\| \alpha' \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \end{array} \right] \right\rangle$$

$$c' \triangleq \left\langle x' \middle\| \tilde{\mu} \left[\begin{array}{l} \mathsf{K}'_{4i}(\vec{\beta'_{1i}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{1i}}).\left\langle v_3 \middle\| \tilde{\mu}\left[\mathsf{K}_3(\vec{\beta_3}, \vec{y_3}).\left\langle v_{1i} \middle\| \tilde{\mu}\left[\mathsf{K}_{1j}(\vec{\beta_{1j}}, \vec{y_{1j}}).\left\langle \mathsf{K}_{4j}(\vec{\beta_{1j}}, \vec{\beta_3}, \vec{y_3}, \vec{\beta_{1j}}) \middle\| \alpha \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \\ \mathsf{K}'_{5i}(\vec{\beta'_{2i}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{2i}}).\left\langle v_3 \middle\| \tilde{\mu}\left[\mathsf{K}_3(\vec{\beta_3}, \vec{y_3}).\left\langle v_{2i} \middle\| \tilde{\mu}\left[\mathsf{K}_{2j}(\vec{\beta_{2j}}, \vec{y_{2j}}).\left\langle \mathsf{K}_{5j}(\vec{\beta_{1j}}, \vec{\beta_3}, \vec{y_3}, \vec{\beta_{1j}}) \middle\| \alpha \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \end{array} \right] \right\rangle$$

where we make use of the following shorthand:

$$v_{1i} \triangleq \mu\alpha_1.\langle \mathsf{K}'_{1i}(\vec{\beta'_{1i}}, \vec{y'_{1i}}) \| \tilde{\mu} x'_1.c'_1 \rangle \quad v_{2i} \triangleq \mu\alpha_2.\langle \mathsf{K}'_{2i}(\vec{\beta'_{2i}}, \vec{y'_{2i}}) \| \tilde{\mu} x'_2.c'_2 \rangle \quad v_3 \triangleq \mu\alpha_3.\langle \mathsf{K}'_3(\vec{\beta'_3}, \vec{y'_3}) \| \tilde{\mu} x'_3.c'_3 \rangle$$
$$v'_{1i} \triangleq \mu\alpha'_1.\langle \mathsf{K}_{1i}(\vec{\beta_{1i}}, \vec{y_{1i}}) \| \tilde{\mu} x_1.c_1 \rangle \quad v'_{2i} \triangleq \mu\alpha'_2.\langle \mathsf{K}_{2i}(\vec{\beta_{2i}}, \vec{y_{2i}}) \| \tilde{\mu} x_2.c_2 \rangle \quad v'_3 \triangleq \mu\alpha'_3.\langle \mathsf{K}_3(\vec{\beta_3}, \vec{y_3}) \| \tilde{\mu} x_3.c_3 \rangle$$

And note that since both $\mathsf{F}(\vec{C}) : \mathcal{V}$ and $\mathsf{F}'(\vec{C'}) : \mathcal{V}$, the isomorphism must be positive (lemma B.3 (a)).

The composition of c and c' along α' and x' of type $\mathsf{F}'(\vec{C'})$ is equal to the identity command $\langle x \| \alpha \rangle$ via the combined strength of the $\tilde{\mu}$ and η axioms for the call-by-value data types F'_1, F'_2, and F'_3, as previously discussed in appendix A, as well as the call-by-value χ axiom to reassociate the bindings to bring the isomorphisms for those data types together, as follows:

$$\langle \mu\alpha'.c \| \tilde{\mu} x'.c' \rangle$$
$$=_{\eta_{\tilde{\mu}}}$$
$$\left\langle \mu\alpha'.\left\langle x \middle\| \tilde{\mu} \left[\begin{array}{l} \mathsf{K}_{4i}(\vec{\beta_{1i}}, \vec{\beta_3}, \vec{y_3}, \vec{y_{1i}}).\left\langle v'_{1i} \middle\| \tilde{\mu}\left[\mathsf{K}'_{1i}(\vec{\beta'_{1j}}, \vec{y'_{1j}}).\left\langle v'_3 \middle\| \tilde{\mu}\left[\mathsf{K}'_3(\vec{\beta'_3}, \vec{y'_3}).\left\langle \mathsf{K}'_{4j}(\vec{\beta'_{1j}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{1j}}) \middle\| \alpha' \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \\ \mathsf{K}_{5i}(\vec{\beta_{2i}}, \vec{\beta_3}, \vec{y_3}, \vec{y_{2i}}).\left\langle v'_{2i} \middle\| \tilde{\mu}\left[\mathsf{K}'_{2i}(\vec{\beta'_{2j}}, \vec{y'_{2j}}).\left\langle v'_3 \middle\| \tilde{\mu}\left[\mathsf{K}'_3(\vec{\beta'_3}, \vec{y'_3}).\left\langle \mathsf{K}'_{5j}(\vec{\beta'_{2j}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{2j}}) \middle\| \alpha' \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \end{array} \right] \right\rangle \middle\| \right.$$
$$\left. \tilde{\mu} \left[\begin{array}{l} \mathsf{K}'_{4i}(\vec{\beta'_{1i}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{1i}}).\left\langle v_3 \middle\| \tilde{\mu}\left[\mathsf{K}_3(\vec{\beta_3}, \vec{y_3}).\left\langle v_{1i} \middle\| \tilde{\mu}\left[\mathsf{K}_{1j}(\vec{\beta_{1j}}, \vec{y_{1j}}).\left\langle \mathsf{K}_{4j}(\vec{\beta_{1j}}, \vec{\beta_3}, \vec{y_3}, \vec{\beta_{1j}}) \middle\| \alpha \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \\ \mathsf{K}'_{5i}(\vec{\beta'_{2i}}, \vec{\beta'_3}, \vec{y'_3}, \vec{y'_{2i}}).\left\langle v_3 \middle\| \tilde{\mu}\left[\mathsf{K}_3(\vec{\beta_3}, \vec{y_3}).\left\langle v_{2i} \middle\| \tilde{\mu}\left[\mathsf{K}_{2j}(\vec{\beta_{2j}}, \vec{y_{2j}}).\left\langle \mathsf{K}_{5j}(\vec{\beta_{1j}}, \vec{\beta_3}, \vec{y_3}, \vec{\beta_{1j}}) \middle\| \alpha \right\rangle \right]^{j} \right\rangle \right]^{i} \right\rangle \end{array} \right] \right\rangle$$
$$=_{\mu\tilde{\mu}\beta^{\mathsf{F}'}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle v'_{1i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{1i}(\overrightarrow{\beta'_{1j}}, \overrightarrow{y'_{1j}}).\ \langle v'_3 \| \tilde{\mu}[\mathsf{K}'_3(\overrightarrow{\beta'_3}, \overrightarrow{y'_3}).\ \langle v_3 \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{1j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle v'_{2i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{2i}(\overrightarrow{\beta'_{2j}}, \overrightarrow{y'_{2j}}).\ \langle v'_3 \| \tilde{\mu}[\mathsf{K}'_3(\overrightarrow{\beta'_3}, \overrightarrow{y'_3}).\ \langle v_3 \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{2j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \end{array} \right\rangle$$

$$= {}_{\tilde{\mu}\eta^{\mathsf{F}'_3}_{\mathcal{V}}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle \mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}) \| \tilde{\mu} x_3.\ \langle v'_{1i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{1i}(\overrightarrow{\beta'_{1j}}, \overrightarrow{y'_{1j}}).\ \langle \mu\alpha'_3.c_3 \| \tilde{\mu} x'_3.\ \langle \mu\alpha_3.c'_3 \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{1j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle \mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}) \| \tilde{\mu} x_3.\ \langle v'_{2i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{2i}(\overrightarrow{\beta'_{2j}}, \overrightarrow{y'_{2j}}).\ \langle \mu\alpha'_3.c_3 \| \tilde{\mu} x'_3.\ \langle \mu\alpha_3.c'_3 \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{2j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \end{array} \right\rangle$$

$$= {}_{\chi_{\mathcal{V}}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle \mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}) \| \tilde{\mu} x_3.\ \langle v'_{1i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{1i}(\overrightarrow{\beta'_{1j}}, \overrightarrow{y'_{1j}}).\ \langle \mu\alpha_3.\ \langle \mu\alpha'_3.c_3 | \tilde{\mu} x'_3.c_3 \rangle \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{1j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle \mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}) \| \tilde{\mu} x_3.\ \langle v'_{2i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{2i}(\overrightarrow{\beta'_{2j}}, \overrightarrow{y'_{2j}}).\ \langle \mu\alpha_3.\ \langle \mu\alpha'_3.c_3 | \tilde{\mu} x'_3.c'_3 \rangle \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{2j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \end{array} \right\rangle$$

$$= {}_{Iso\eta_{\mu}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle \mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}) \| \tilde{\mu} x_3.\ \langle v'_{1i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{1i}(\overrightarrow{\beta'_{1j}}, \overrightarrow{y'_{1j}}).\ \langle x_3 \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{1j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle \mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}) \| \tilde{\mu} x_3.\ \langle v'_{1i} | \\ \quad |\tilde{\mu}[\mathsf{K}'_{2i}(\overrightarrow{\beta'_{2j}}, \overrightarrow{y'_{2j}}).\ \langle x_3 \| \tilde{\mu}[\mathsf{K}_3(\overrightarrow{\beta_3}, \overrightarrow{y_3}).\ \langle v_{2j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle]\rangle]\rangle \end{array} \right\rangle$$

$$= {}_{\tilde{\mu}\mu\beta^{\mathsf{F}_3}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle v'_{1i} \| \tilde{\mu}[\overrightarrow{\mathsf{K}'_{1i}(\overrightarrow{\beta'_{1j}}, \overrightarrow{y'_{1j}}).\ \langle v_{1j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle}^{j}]\rangle}^{i} \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle v'_{2i} \| \tilde{\mu}[\overrightarrow{\mathsf{K}'_{2i}(\overrightarrow{\beta'_{2j}}, \overrightarrow{y'_{2j}}).\ \langle v_{2j} \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle}^{j}]\rangle}^{i} \end{array} \right\rangle$$

$$= {}_{\tilde{\mu}\eta^{\mathsf{F}'_1}_{\mathcal{V}}\eta^{\mathsf{F}'_2}_{\mathcal{V}}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle \mathsf{K}_{1i}(\overrightarrow{\beta_{1i}}, \overrightarrow{y_{1i}}) \| \tilde{\mu} x_1.\ \langle \mu\alpha'_1.c_1 \| \tilde{\mu} x'_1.\ \langle \mu\alpha_1.c'_1 \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle\rangle\rangle \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle \mathsf{K}_{2i}(\overrightarrow{\beta_{2i}}, \overrightarrow{y_{2i}}) \| \tilde{\mu} x_2.\ \langle \mu\alpha'_2.c_2 \| \tilde{\mu} x'_2.\ \langle \mu\alpha_2.c'_2 \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle\rangle\rangle \end{array} \right\rangle$$

$$= {}_{\chi_{\mathcal{V}}}$$

$$\left\langle x \middle| \tilde{\mu} \middle| \begin{array}{l} \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\ \langle \mathsf{K}_{1i}(\overrightarrow{\beta_{1i}}, \overrightarrow{y_{1i}}) \| \tilde{\mu} x_1.\ \langle \mu\alpha_1.\ \langle \mu\alpha'_1.c_1 | \tilde{\mu} x'_1.c'_1 \rangle \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\ \langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle\rangle\rangle \\ \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\ \langle \mathsf{K}_{2i}(\overrightarrow{\beta_{2i}}, \overrightarrow{y_{2i}}) \| \tilde{\mu} x_2.\ \langle \mu\alpha_2.\ \langle \mu\alpha'_2.c_2 | \tilde{\mu} x'_2.c'_2 \rangle \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\ \langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle\rangle\rangle \end{array} \right\rangle$$

$$= {}_{Iso}$$

$$\left\langle x \middle\| \tilde{\mu} \left[\begin{array}{l} \overrightarrow{\mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\langle \mathsf{K}_{1i}(\overrightarrow{\beta_{1i}}, \overrightarrow{y_{1i}}) \| \tilde{\mu} x_1.\langle \mu\alpha_1.\ \langle x_1 \| \alpha_1 \rangle \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle\rangle}^{i} \\ \overrightarrow{\mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\langle \mathsf{K}_{2i}(\overrightarrow{\beta_{2i}}, \overrightarrow{y_{2i}}) \| \tilde{\mu} x_2.\langle \mu\alpha_2.\ \langle x_2 \| \alpha_2 \rangle \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle\rangle}^{i} \end{array} \right] \right\rangle$$

$$=_{\eta_\mu \eta_{\tilde{\mu}}}$$

$$\left\langle x \middle\| \tilde{\mu} \left[\begin{array}{l} \overrightarrow{\mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\langle \mathsf{K}_{1i}(\overrightarrow{\beta_{1i}}, \overrightarrow{y_{1i}}) \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{1j}(\overrightarrow{\beta_{1k}}, \overrightarrow{y_{1k}}).\langle \mathsf{K}_{4k}(\overrightarrow{\beta_{1k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1k}}) \| \alpha \rangle}^{k}]\rangle}^{i} \\ \overrightarrow{\mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\langle \mathsf{K}_{2i}(\overrightarrow{\beta_{2i}}, \overrightarrow{y_{2i}}) \| \tilde{\mu}[\overrightarrow{\mathsf{K}_{2j}(\overrightarrow{\beta_{2k}}, \overrightarrow{y_{2k}}).\langle \mathsf{K}_{5k}(\overrightarrow{\beta_{2k}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2k}}) \| \alpha \rangle}^{k}]\rangle}^{i} \end{array} \right] \right\rangle$$

$$=_{\mu\tilde{\mu}\beta^{\mathsf{F}_1}\beta^{\mathsf{F}_2}}$$

$$\left\langle x \middle\| \tilde{\mu} \left[\begin{array}{l} \overrightarrow{\mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{1i}}).\langle \mathsf{K}_{4i}(\overrightarrow{\beta_{1i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{1i}}) \| \alpha \rangle}^{i} \\ \overrightarrow{\mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{y_{2i}}).\langle \mathsf{K}_{5i}(\overrightarrow{\beta_{2i}}, \overrightarrow{\beta_3}, \overrightarrow{y_3}, \overrightarrow{\beta_{2i}}) \| \alpha \rangle}^{i} \end{array} \right] \right\rangle$$

$$=_{\eta^{\mathsf{F}}}$$

$$\langle x \| \alpha \rangle$$

And the reverse composition of c' and c along α and x of type $\mathsf{F}(\overrightarrow{C})$ is equal to the identity command $\langle x' \| \alpha' \rangle$ similarly. □

Lemma C.3 (Data compatibility instance). *For any types $A : \mathcal{T}$, $A' : \mathcal{T}'$, $\overrightarrow{C : \mathcal{S}}$, $\overrightarrow{C' : \mathcal{S}'}$,*

a) given $\begin{array}{l} \mathbf{data}\ \mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{V}\ \mathbf{where} \\ \quad \mathsf{K} : (A : \mathcal{T} \vdash \mathsf{F}(\overrightarrow{X}) \mid) \end{array}$ *and* $\begin{array}{l} \mathbf{data}\ \mathsf{F}'(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{V}\ \mathbf{where} \\ \quad \mathsf{K}' : (A' : \mathcal{T}' \vdash \mathsf{F}'(\overrightarrow{X'}) \mid) \end{array}$ *then* $\mathsf{F}(\overrightarrow{C}) \approx^{+} \mathsf{F}'(\overrightarrow{C'})$

if either $A\overrightarrow{\{C/X\}} \approx^{-} A'\overrightarrow{\{C'/X'\}}$ *or* $A\overrightarrow{\{C/X\}} \approx A'\overrightarrow{\{C'/X'\}}$ *and* $\mathcal{T} = \mathcal{T}' = \mathcal{V}$, *and*

b) given $\begin{array}{l} \mathbf{data}\ \mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{V}\ \mathbf{where} \\ \quad \mathsf{K} : (\vdash \mathsf{F}(\overrightarrow{X}) \mid A : \mathcal{T}) \end{array}$ *and* $\begin{array}{l} \mathbf{data}\ \mathsf{F}'(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{V}\ \mathbf{where} \\ \quad \mathsf{K}' : (\vdash \mathsf{F}'(\overrightarrow{X'}) \mid A' : \mathcal{T}') \end{array}$ *then* $\mathsf{F}(\overrightarrow{C}) \approx^{+} \mathsf{F}'(\overrightarrow{C'})$

if either $A\overrightarrow{\{C/X\}} \approx^{+} A'\overrightarrow{\{C'/X'\}}$ *or* $A\overrightarrow{\{C/X\}} \approx A'\overrightarrow{\{C'/X'\}}$ *and* $\mathcal{T} = \mathcal{T}' = \mathcal{N}$.

Proof. Let $B = A\overrightarrow{\{C/X\}}$, $B' = A'\overrightarrow{\{C'/X'\}}$, and suppose that the commands $c_1 : (y : B \vdash \beta' : B')$ and $c_1' : (y' : B' \vdash \beta : B)$ witness the isomorphism $B \approx B'$. The isomorphisms between $\mathsf{F}(\overrightarrow{C})$ and $\mathsf{F}'(\overrightarrow{C'})$ are established by the commands $c : (x : \mathsf{F}(\overrightarrow{C}) \vdash \alpha' : \mathsf{F}'(\overrightarrow{C'}))$ and $c' : (x' : \mathsf{F}'(\overrightarrow{C'}) \vdash \alpha : \mathsf{F}(\overrightarrow{C}))$ as follows:

a) $c \triangleq \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mu\beta'.c_1 \| \tilde{\mu}y'.\langle \mathsf{K}'(y') \| \alpha' \rangle\rangle] \rangle \qquad c' \triangleq \langle x' \| \tilde{\mu}[\mathsf{K}'(y').\big\langle \mu\beta.c_1' \big| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \big\rangle] \rangle$

b) $c \triangleq \langle x \| \tilde{\mu}[\mathsf{K}(\beta).\big\langle \mu\beta'.\langle \mathsf{K}'(\beta') \| \alpha' \rangle \big| \tilde{\mu}y'.c_1' \big\rangle] \rangle \qquad c' \triangleq \langle x' \| \tilde{\mu}[\mathsf{K}'(\beta').\langle \mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle \| \tilde{\mu}y.c_1 \rangle] \rangle$

For part (a), the composition of c and c' along α' and x' of type $\mathsf{F}'(\overrightarrow{C'})$ is equal to the identity command $\langle x' \| \alpha' \rangle$ via $\beta^{\mathsf{F}'}$ and η^{F} along with either thunkability or the call-by-value $\chi_{\mathcal{V}}$ to reveal the isomorphism, as follows:

$$\begin{aligned}
\langle \mu\alpha'.c \| \tilde{\mu}x'.c' \rangle &=_{\eta_{\tilde{\mu}}} \big\langle \mu\alpha'.\langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mu\beta'.c_1 \| \tilde{\mu}y'.\langle \mathsf{K}'(y') \| \alpha' \rangle\rangle] \big| \tilde{\mu}[\mathsf{K}'(y').\big\langle \mu\beta.c_1' \big| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \big\rangle] \big\rangle \\
&=_{\mu} \big\langle x \big| \tilde{\mu}\big[\mathsf{K}(y).\big\langle \mu\beta'.c_1 \big| \tilde{\mu}y'.\big\langle \mathsf{K}'(y') \big| \tilde{\mu}[\mathsf{K}'(y').\big\langle \mu\beta.c_1' \big| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \big\rangle] \big\rangle \big\rangle \big] \big\rangle \\
&=_{\tilde{\mu}\beta^{\mathsf{F}'}} \big\langle x \big| \tilde{\mu}\big[\mathsf{K}(y).\big\langle \mu\beta'.c_1 \big| \tilde{\mu}y'.\big\langle \mu\beta.c_1' \big| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \big\rangle \big\rangle \big] \big\rangle \\
&=_{Thk/\chi_{\mathcal{V}}} \big\langle x \big| \tilde{\mu}\big[\mathsf{K}(y).\big\langle \mu\beta.\big\langle \mu\beta'.c_1 \big| \tilde{\mu}y'.c_1' \big\rangle \big| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \big\rangle \big] \big\rangle \\
&=_{Iso} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mu\beta.\langle y \| \beta \rangle \| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle\rangle] \rangle \\
&=_{\eta_\mu \tilde{\mu}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mathsf{K}(y) \| \alpha \rangle] \rangle \\
&=_{\eta^{\mathsf{F}}} \langle x \| \alpha \rangle
\end{aligned}$$

Note that the equation marked $\mathit{Thk}/\chi_{\mathcal{V}}$ follows in the case that *either* $\mu\beta'.c_1'$ is thunkable (by the definition of thunkability) *or* $\mathcal{T} = \mathcal{V}$ (in which case $\tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle$ is co-value and the $\mu_{\mathcal{V}}$ axiom applies). Likewise, the composition of c_2' and c_1' along α' and x' of type $\mathsf{F}_1(\overrightarrow{C})$ is equal to the identity command $\langle y' \| \beta' \rangle$ analogously to the derivation above.

For part (b), the composition of c and c' along α' and x' of type $\mathsf{F}'(\overrightarrow{C'})$ is equal to the identity command $\langle x' \| \alpha' \rangle$ also via $\beta^{\mathsf{F}'}$ and η^{F} along with linearity or the call-by-name $\chi_{\mathcal{N}}$ to reveal the isomorphism, as follows:

$$
\begin{aligned}
\langle \mu\alpha'.c \| \tilde{\mu}x'.c' \rangle &=_{\eta_{\tilde{\mu}}} \left\langle \mu\alpha'.c\langle x \| \tilde{\mu}[\mathsf{K}(\beta).\left\langle \mu\beta'.\langle \mathsf{K}'(\beta') \| \alpha' \rangle \middle| \tilde{\mu}y'.c_1' \right\rangle] \right\rangle \middle| \tilde{\mu}[\mathsf{K}'(\beta').\langle \mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle \| \tilde{\mu}y.c_1 \rangle] \right\rangle \\
&=_{\mu} \langle x \| \tilde{\mu}[\mathsf{K}(\beta).\left\langle \mu\beta'.\langle \mathsf{K}'(\beta') \| \tilde{\mu}[\mathsf{K}'(\beta').\langle \mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle \| \tilde{\mu}y.c_1 \rangle] \right\rangle \middle| \tilde{\mu}y'.c_1' \right\rangle] \rangle \\
&=_{\mu\beta^{\mathsf{F}'}} \langle x \| \tilde{\mu}[\mathsf{K}(\beta).\left\langle \mu\beta'.\langle \mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle \| \tilde{\mu}y.c_1 \right\rangle \middle| \tilde{\mu}y'.c_1' \rangle] \rangle \\
&=_{\mathit{Lin}/\chi_{\mathcal{N}}} \langle x \| \tilde{\mu}[\mathsf{K}(\beta).\left\langle \mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle \middle| \tilde{\mu}y.\left\langle \mu\beta'.c_1 \middle| \tilde{\mu}y'.c_1' \right\rangle \right\rangle] \rangle \\
&=_{\mathit{Iso}} \langle x \| \tilde{\mu}[\mathsf{K}(\beta).\langle \mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle \| \tilde{\mu}y.\langle y \| \beta \rangle \rangle] \rangle \\
&=_{\eta_{\tilde{\mu}}\mu} \langle x \| \tilde{\mu}[\mathsf{K}(\beta).\langle \mathsf{K}(\beta) \| \alpha \rangle] \rangle \\
&=_{\eta^{\mathsf{F}}} \langle x \| \alpha \rangle
\end{aligned}
$$

Note that the equation marked $\mathit{Lin}/\chi_{\mathcal{N}}$ follows in the case that *either* $\tilde{\mu}y'.c_1'$ is linear *or* $\mathcal{T} = \mathcal{N}$ (in which case $\mu\beta.\langle \mathsf{K}(\beta) \| \alpha \rangle$ is a value and the $\tilde{\mu}_{\mathcal{N}}$ axiom applies). Likewise, the composition of c_2' and c_1' along α' and x' of type $\mathsf{F}_1(\overrightarrow{C})$ is equal to the identity command $\langle y' \| \beta' \rangle$ analogously to the derivation above.

Finally, note that the isomorphism $\mathsf{F}(\overrightarrow{C}) \approx \mathsf{F}'(\overrightarrow{C'})$ must be positive since $\mathsf{F}(\overrightarrow{C}) : \mathcal{V}$ and $\mathsf{F}'(\overrightarrow{C'}) : \mathcal{V}$ (lemma B.3 (a)). □

Lemma C.4 ((Co-)Data interchange shift instance). *For any types* $\overrightarrow{C : \mathcal{S}}, \overrightarrow{C' : \mathcal{S}'}$ *and (co-)data declarations*

$$
\begin{array}{ll}
\textbf{data}\ \mathsf{F}(\overrightarrow{X : \mathcal{S}}) : \mathcal{T}\ \textbf{where} & \textbf{data}\ \mathsf{F}'(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{T}\ \textbf{where} \\
\quad \overrightarrow{\mathsf{K} : \left(\Gamma \vdash \mathsf{F}(\overrightarrow{X : \mathcal{S}}) \mid \Delta\right)} & \quad \overrightarrow{\mathsf{K}' : \left(\Gamma' \vdash \mathsf{F}'(\overrightarrow{X' : \mathcal{S}'}) \mid \Delta'\right)} \\
\textbf{codata}\ \mathsf{G}(\overrightarrow{X : \mathcal{S}}) : \mathcal{U}\ \textbf{where} & \textbf{codata}\ \mathsf{G}'(\overrightarrow{X' : \mathcal{S}'}) : \mathcal{U}\ \textbf{where} \\
\quad \overrightarrow{\mathsf{O} : (\Gamma \mid \mathsf{F}(\overrightarrow{X : \mathcal{S}}) \vdash \Delta)} & \quad \overrightarrow{\mathsf{O}' : (\Gamma' \mid \mathsf{G}'(\overrightarrow{X' : \mathcal{S}'}) \vdash \Delta')}
\end{array}
$$

$\mathsf{F}(\overrightarrow{C}) \approx \mathsf{F}'(\overrightarrow{C'})$ *implies* $\mathsf{G}(\overrightarrow{C}) \approx^{-} \mathsf{G}'(\overrightarrow{C'})$ *when* $\mathcal{T} = \mathcal{V}$ *and* $\mathsf{G}(\overrightarrow{C}) \approx \mathsf{G}'(\overrightarrow{C'})$ *implies* $\mathsf{F}(\overrightarrow{C}) \approx^{+} \mathsf{F}'(\overrightarrow{C'})$ *when* $\mathcal{U} = \mathcal{N}$.

Proof. First, suppose that the commands $c_1 : (x_1 : \mathsf{F}(\overrightarrow{C}) \vdash \alpha_1' : \mathsf{F}'(\overrightarrow{C'}))$ and $c_1' : (x_1' : \mathsf{F}'(\overrightarrow{C'}) \vdash \alpha_1 : \mathsf{F}(\overrightarrow{C}))$ witness the isomorphism $\mathsf{F}(\overrightarrow{C}) \approx^{+} \mathsf{F}'(\overrightarrow{C'})$. Then the negative isomorphism between $\mathsf{G}(\overrightarrow{C})$ and $\mathsf{G}'(\overrightarrow{C'})$ is established by:

$$
\begin{aligned}
c_2 &\triangleq \left\langle \mu\left(\overrightarrow{\mathsf{O}_i'[\overrightarrow{y_i'}, \overrightarrow{\beta_i'}].\left\langle \mu\alpha_1.\left\langle \mathsf{K}_i'(\overrightarrow{\beta_i'}, \overrightarrow{y_i'}) \middle| \tilde{\mu}x_1'.c_1' \right\rangle \middle| \tilde{\mu}\left[\overrightarrow{\mathsf{K}_j(\overrightarrow{\beta_j}, \overrightarrow{y_j}).\langle x_2 \| \mathsf{O}_j[\overrightarrow{y_j}, \overrightarrow{\beta_j}] \rangle} \right]^{j} \right\rangle}^{i} \right) \middle| \alpha_2' \right\rangle \\
&\quad : (x_2 : \mathsf{G}(\overrightarrow{C}) \vdash \alpha_2' : \mathsf{G}'(\overrightarrow{C'})) \\
c_2' &\triangleq \left\langle \mu\left(\overrightarrow{\mathsf{O}_i[\overrightarrow{y_i}, \overrightarrow{\beta_i}].\left\langle \mu\alpha_1'.\left\langle \mathsf{K}_i(\overrightarrow{\beta_i}, \overrightarrow{y_i}) \middle| \tilde{\mu}x_1.c_1 \right\rangle \middle| \tilde{\mu}\left[\overrightarrow{\mathsf{K}_j'(\overrightarrow{\beta_j'}, \overrightarrow{y_j'}).\langle x_2' \| \mathsf{O}_j'[\overrightarrow{y_j'}, \overrightarrow{\beta_j'}] \rangle} \right]^{j} \right\rangle}^{i} \right) \middle| \alpha_2 \right\rangle \\
&\quad : (x_2' : \mathsf{G}'(\overrightarrow{C'}) \vdash \alpha_2 : \mathsf{G}(\overrightarrow{C}))
\end{aligned}
$$

Note that both $\mu\alpha_2.c_2'$ and $\mu\alpha_2'.c_2$ are thunkable (by lemma B.2 (a)) because they are both η_μ-equivalent to a value for any choice of $\mathcal{U}$. So we only need to demonstrate that both compositions of c_2 and c_2' are equal to the appropriate identity commands.

The composition of c_2' and c_2 along α_2 and x_2 of type $\mathrm{G}(\vec{C})$ is equal to the identity command $\langle x_2' \| \alpha_2' \rangle$ via the β^{G}, $\eta^{\mathrm{F}}_{\mathcal{V}}$, $\beta^{\mathrm{F}'}$, and $\eta^{\mathrm{G}'}$ axioms as follows:

$$
\begin{aligned}
&\langle \mu\alpha_2.c_2' \| \tilde{\mu}x_2.c_2 \rangle \\
&\triangleq \left\langle \mu\alpha_2.\left\langle \mu\left(\overrightarrow{\mathrm{O}_i[\vec{y_i},\vec{\beta_i}].\left\langle \mu\alpha_1'.\left\langle \mathrm{K}_i(\vec{\beta_i},\vec{y_i}) \middle\| \tilde{\mu}x_1.c_1 \right\rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_j'(\vec{\beta_j'},\vec{y_j'}).\langle x_2' \| \mathrm{O}_j'[\vec{y_j'},\vec{\beta_j'}]\rangle}^{j}\right]\right\rangle}^{i}\right) \middle\| \alpha_2 \right\rangle \right| \\
&\quad \left| \tilde{\mu}x_2.\left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left\langle \mu\alpha_1.\left\langle \mathrm{K}_i'(\vec{\beta_i'},\vec{y_i'}) \middle\| \tilde{\mu}x_1'.c_1' \right\rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_j(\vec{\beta_j},\vec{y_j}).\langle x_2 \| \mathrm{O}_j[\vec{y_j},\vec{\beta_j}]\rangle}^{j}\right]\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \right\rangle \\
&=_{\eta_\mu} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i[\vec{y_i},\vec{\beta_i}].\left\langle \mu\alpha_1'.\left\langle \mathrm{K}_i(\vec{\beta_i},\vec{y_i}) \middle\| \tilde{\mu}x_1.c_1 \right\rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_j'(\vec{\beta_j'},\vec{y_j'}).\langle x_2' \| \mathrm{O}_j'[\vec{y_j'},\vec{\beta_j'}]\rangle}^{j}\right]\right\rangle}^{i}\right) \middle\| \right. \\
&\quad \left. \tilde{\mu}x_2.\left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left\langle \mu\alpha_1.\left\langle \mathrm{K}_i'(\vec{\beta_i'},\vec{y_i'}) \middle\| \tilde{\mu}x_1'.c_1' \right\rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_j(\vec{\beta_j},\vec{y_j}).\langle x_2 \| \mathrm{O}_j[\vec{y_j},\vec{\beta_j}]\rangle}^{j}\right]\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \right\rangle \\
&=_{\tilde{\mu}} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_j(\vec{\beta_j},\vec{y_j}).\left\langle \mu\left(\overrightarrow{\mathrm{O}_k[\vec{y_k},\vec{\beta_k}].\left| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_l'(\vec{\beta_l'},\vec{y_l'}).\langle x_2' \| \mathrm{O}_l'[\vec{y_l'},\vec{\beta_l'}]\rangle}^{l}\right]\right\rangle}^{k}\right) \middle\| \mathrm{O}_j[\vec{y_j},\vec{\beta_j}]\right\rangle}^{j}\right]\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{\beta^{\mathrm{G}}\mu\tilde{\mu}} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_j(\vec{\beta_j},\vec{y_j}).\left\langle \mu\alpha_1'.\left\langle \mathrm{K}_j(\vec{\beta_j},\vec{y_j}) \middle\| \tilde{\mu}x_1.c_1 \right\rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_l'(\vec{\beta_l'},\vec{y_l'}).\langle x_2' \| \mathrm{O}_l'[\vec{y_l'},\vec{\beta_l'}]\rangle}^{l}\right]\right\rangle}^{j}\right]\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{\tilde{\mu}\eta^{\mathrm{F}}_{\mathcal{V}}} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left\langle \mu\alpha_1.\left\langle \mathrm{K}_i'(\vec{\beta_i'},\vec{y_i'}) \middle\| \tilde{\mu}x_1'.c_1' \right\rangle \middle\| \tilde{\mu}x_1.\left\langle \mu\alpha_1'.c_1 \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_l'(\vec{\beta_l'},\vec{y_l'}).\langle x_2' \| \mathrm{O}_l'[\vec{y_l'},\vec{\beta_l'}]\rangle}^{l}\right]\right\rangle\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{\chi_{\mathcal{V}}} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left\langle \mathrm{K}_i'(\vec{\beta_i'},\vec{y_i'}) \middle\| \tilde{\mu}x_1'.\left\langle \mu\alpha_1'.\langle \mu\alpha_1.c_1' \| \tilde{\mu}x_1.c_1 \rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_l'(\vec{\beta_l'},\vec{y_l'}).\langle x_2' \| \mathrm{O}_l'[\vec{y_l'},\vec{\beta_l'}]\rangle}^{l}\right]\right\rangle\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{Iso} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left\langle \mathrm{K}_i'(\vec{\beta_i'},\vec{y_i'}) \middle\| \tilde{\mu}x_1'.\left\langle \mu\alpha_1'.\langle x_1' \| \alpha_1' \rangle \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_l'(\vec{\beta_l'},\vec{y_l'}).\langle x_2' \| \mathrm{O}_l'[\vec{y_l'},\vec{\beta_l'}]\rangle}^{l}\right]\right\rangle\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{\eta_\mu\eta_{\tilde{\mu}}} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\left\langle \mathrm{K}_i'(\vec{\beta_i'},\vec{y_i'}) \middle\| \tilde{\mu}\left[\overrightarrow{\mathrm{K}_l'(\vec{\beta_l'},\vec{y_l'}).\langle x_2' \| \mathrm{O}_l'[\vec{y_l'},\vec{\beta_l'}]\rangle}^{l}\right]\right\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{\beta^{\mathrm{F}'}\mu\tilde{\mu}} \left\langle \mu\left(\overrightarrow{\mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}].\langle x_2' \| \mathrm{O}_i'[\vec{y_i'},\vec{\beta_i'}]\rangle}^{i}\right) \middle\| \alpha_2' \right\rangle \\
&=_{\eta^{\mathrm{G}'}} \langle x_2' \| \alpha_2' \rangle
\end{aligned}
$$

The composition of c_2 and c_2' along α_2' and x_2' of type $\mathrm{G}'(\vec{C})$ is equal to the identity command $\langle x_2 \| \alpha_2 \rangle$ via the $\beta^{\mathrm{G}'}$, $\eta^{\mathrm{F}'}_{\mathcal{V}}$, β^{F}, and η^{G} similarly.

Second, suppose that the commands $c_2 : (x_2 : \mathsf{G}(\vec{C}) \vdash \alpha_2' : \mathsf{G}'(\vec{C'}))$ and $c_2' : (x_2' : \mathsf{G}'(\vec{C'}) \vdash \alpha_2 : \mathsf{G}(\vec{C}))$ witness the isomorphism $\mathsf{G}(\vec{C}) \approx \mathsf{G}'(\vec{C'})$. Then the isomorphism between $\mathsf{F}(\vec{C})$ and $\mathsf{F}'(\vec{C'})$ is established by:

$$c_1 \triangleq \left\langle x_1 \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\vec{\beta_i}, \vec{y_i}).\left\langle \mu\left(\overrightarrow{\mathsf{O}_j'[\vec{y_j'}\vec{\beta_j'}].\langle \mathsf{K}_j'(\vec{y_j'}, \vec{\beta_j'}) \| \alpha_1' \rangle}^{j}\right) \middle\| \tilde{\mu} x_2'.\left\langle \tilde{\mu}\alpha_2.c_2' \middle\| \mathsf{O}_i[\vec{y_i}, \vec{\beta_i}]\right\rangle\right\rangle}^{i}\right]\right\rangle$$
$$: (x_1 : \mathsf{F}(\vec{C}) \vdash \alpha_1' : \mathsf{F}'(\vec{C'}))$$
$$c_1' \triangleq \left\langle x_1' \middle\| \tilde{\mu}\left[\overrightarrow{\mathsf{K}_i(\vec{\beta_i}, \vec{y_i}).\left\langle \mu\left(\overrightarrow{\mathsf{O}_j[\vec{y_j}\vec{\beta_j}].\langle \mathsf{K}_j(\vec{y_j}, \vec{\beta_j}) \| \alpha_1 \rangle}^{j}\right) \middle\| \tilde{\mu} x_2.\left\langle \tilde{\mu}\alpha_2'.c_2 \middle\| \mathsf{O}_i'[\vec{y_i'}, \vec{\beta_i'}]\right\rangle\right\rangle}^{i}\right]\right\rangle$$
$$: (x_1' : \mathsf{F}'(\vec{C'}) \vdash \alpha_1 : \mathsf{F}(\vec{C}))$$

Again, note that both $\tilde{\mu}x_1.c_1$ and $\tilde{\mu}x_1'.c_1'$ are linear (by lemma B.2 (b)) because they are both $\eta_{\tilde{\mu}}$-equivalent to a co-value for any choice of $\mathcal{T}$. Furthermore, both compositions of c and c' are equal to the identity command analogously to the previous part by duality. □

Lemma 5.1 ((Co-)Data identity). *a) For any* **data** $\mathsf{F}(\Theta) : \mathcal{U}$ **where** $\mathsf{K} : (A : \mathcal{T} \vdash \mathsf{F}(\Theta) \mid)$, *if* $\mathcal{T} = \mathcal{V}$ *then* $\Theta \vDash \mathsf{F}(\Theta) \approx^{+} A$ *and if* $\mathcal{U} = \mathcal{V}$ *then* $\Theta \vDash \mathsf{F}(\Theta) \approx A$.
b) For any **codata** $\mathsf{G}(\Theta) : \mathcal{U}$ **where** $\mathsf{O} : (\mid \mathsf{G}(\Theta) \vdash A : \mathcal{T})$, *if* $\mathcal{T} = \mathcal{N}$ *then* $\Theta \vDash \mathsf{G}(\Theta) \approx^{-} A$ *and if* $\mathcal{U} = \mathcal{N}$ *then* $\Theta \vDash \mathsf{G}(\Theta) \approx A$.

Proof. a) Let $\Theta = \overrightarrow{X : \mathcal{S}}$, suppose $\overrightarrow{C : \mathcal{S}}$, and let $B = A\overrightarrow{\{C/X\}}$. $\mathsf{F}(\vec{C}) \approx B$ is established by the commands:

$$c_1 \triangleq \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle : (x : \mathsf{F}(\vec{C}) \vdash \beta : B) \qquad c_2 \triangleq \langle \mathsf{K}(y) \| \alpha \rangle : (y : B \vdash \alpha : \mathsf{F}(\vec{C}))$$

First, the composition of c_2 and c_1 along α and x of type $\mathsf{F}(\vec{C}) : \mathcal{U}$ is equal to the identity command $\langle y \| \beta \rangle$ by using the η_μ and $\eta_{\tilde{\mu}}$ axioms to reveal the β^{F} redex as follows:

$$\begin{aligned}\langle \mu\alpha.c_2 \| \tilde{\mu}x.c_1 \rangle &\triangleq \langle \mu\alpha.\langle \mathsf{K}(y) \| \alpha \rangle \| \tilde{\mu}x.\langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \\ &=_{\eta_\mu \eta_{\tilde{\mu}}} \langle \mathsf{K}(y) \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \\ &=_{\beta^{\mathsf{F}}} \langle y \| \tilde{\mu}y.\langle y \| \beta \rangle \rangle \\ &=_{\tilde{\mu}} \langle y \| \beta \rangle\end{aligned}$$

Next, suppose that $\mathcal{T} = \mathcal{V}$. The composition of c_1 and c_2 along β and y of type $B : \mathcal{V}$ is equal to the identity command $\langle x \| \alpha \rangle$ by using the strength of the $\mu_{\mathcal{V}}$ axiom to reveal the η^{F} redex as follows:

$$\begin{aligned}\langle \mu\beta.c_1 \| \tilde{\mu}y.c_2 \rangle &\triangleq \langle \mu\beta.\langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \rangle \\ &=_{\mu_{\mathcal{V}}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \tilde{\mu}y.\langle \mathsf{K}(y) \| \alpha \rangle \rangle] \rangle \\ &=_{\tilde{\mu}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mathsf{K}(y) \| \alpha \rangle] \rangle \\ &=_{\eta^{\mathsf{F}}} \langle x \| \alpha \rangle\end{aligned}$$

Otherwise, suppose that $\mathcal{U} = \mathcal{V}$. The composition of c_1 and c_2 along β and y of type $B : \mathcal{T}$ is equal to the identity command $\langle x \| \alpha \rangle$ by using the combined strength of the $\mu_{\mathcal{V}}$ and η^{F} axioms to percolate out the case analysis on x and create an inner β^{F} redex:

$$\begin{aligned}
\langle \mu\beta.c_1 \| \tilde{\mu}y.c_2 \rangle &\triangleq \langle \mu\beta.\, \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \| \tilde{\mu}y.\, \langle \mathsf{K}(y) \| \alpha \rangle \rangle \\
&=_{\mu_{\mathcal{V}}} \langle x \| \tilde{\mu}x.\, \langle \mu\beta.\, \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \| \tilde{\mu}y.\, \langle \mathsf{K}(y) \| \alpha \rangle \rangle \rangle \\
&=_{\mu_{\mathcal{V}}\eta^{\mathsf{F}}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mathsf{K}(y) \| \tilde{\mu}x.\, \langle \mu\beta.\, \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \| \tilde{\mu}y.\, \langle \mathsf{K}(y) \| \alpha \rangle \rangle \rangle] \rangle \\
&=_{\tilde{\mu}_{\mathcal{V}}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mu\beta.\, \langle \mathsf{K}(y) \| \tilde{\mu}[\mathsf{K}(y).\langle y \| \beta \rangle] \rangle \| \tilde{\mu}y.\, \langle \mathsf{K}(y) \| \alpha \rangle \rangle] \rangle \\
&=_{\beta^{\mathsf{F}}\tilde{\mu}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mu\beta.\, \langle y \| \beta \rangle \| \tilde{\mu}y.\, \langle \mathsf{K}(y) \| \alpha \rangle \rangle] \rangle \\
&=_{\eta_{\mu}\tilde{\mu}} \langle x \| \tilde{\mu}[\mathsf{K}(y).\langle \mathsf{K}(y) \| \alpha \rangle] \rangle =_{\eta^{\mathsf{F}}} \langle x \| \alpha \rangle
\end{aligned}$$

Finally, we need to demonstrate the required linearity constraints for a positive type isomorphism. First, note that by lemma B.2 $\tilde{\mu}x.c_1$ is linear and $\mu\alpha.c_2$ is thunkable for any choice of $\mathcal{T}$ and $\mathcal{U}$. In the case that $\mathcal{T} = \mathcal{V}$, then $\tilde{\mu}y.c_2 : \mathcal{V}$ which is trivially linear since all $\mathcal{V}$-co-terms are co-values, and thus $\Theta \vDash \mathsf{F}(\Theta) \approx^{+} A$.

b) Analogous to the proof of lemma 5.1 (a) by duality. □

Theorem 5.1 (Structural laws). *The declaration isomorphism laws in figs. 8 and 9 are all sound.*

Proof. The data laws all follow by generalizing the particular instances where the data types are isomorphic: the commute, interchange, and compatibility laws are all immediate consequence of lemmas C.1, C.3 and C.4, both mix laws follow from lemma C.2 by taking either F_1 and F'_1 to be the empty data declaration of no alternatives or taking F_3 and F'_3 to be the unit data declaration of one alternative with no components (both of which are isomorphic by reflexivity), and the shift law follows by applying lemma C.4 twice. The co-data laws follow from the data laws by lemma C.4. □

Theorem 5.2 (Polarized laws). *The declaration isomorphism laws in fig. 10 are all sound.*

Proof. Due to the (co-)data interchange laws from figs. 8 and 9 we only need to demonstrate half of the isomorphisms in fig. 10 since each side implies the other. So let us focus only on the more familiar data type declarations, because all the laws for polarized co-data sub-structures are derived from those. In each case, the main technique for establishing these laws is that, for any substitution θ matching the environment Θ, the data type $\mathsf{F}'(\Theta)\theta$ on each right-hand side is positively isomorphic to the single component of the single alternative under the substitution θ according to lemma 5.1 because that single component is call-by-value (*e.g.*, $1 : \mathcal{V}$). What remains is to then demonstrate that in each case, the data type $\mathsf{F}(\Theta)\theta$ is *also* positively isomorphic to that same single component type.

The sub-structure laws for the nullary data types $(0, 1)$ are the easiest to show. Note how for the $0L$ law we directly have that $\mathsf{F}(\Theta) \approx^{+} 0$ as a trivial case of lemma C.1 (b), and $\mathsf{F}'(\Theta) \approx^{+} 0$ by lemma 5.1 (a), so together we know $\mathsf{F}(\Theta) \approx^{+} 0 \approx^{+} \mathsf{F}'(\Theta)$ by positive transitivity (theorem 4.3 (c)). Similarly for the $1L$ law, we have $\mathsf{F}(\Theta) \approx^{+} 1$ as a trivial case of lemma C.1 (a), and so we get $\mathsf{F}(\Theta) \approx^{+} 1 \approx^{+} \mathsf{F}'(\Theta)$ from lemma 5.1 (a) and theorem 4.3 (c) as well.

The sub-structural laws for the unary data types $(-, \downarrow_{\mathcal{S}})$ follow a different line of reasoning, but are not much more difficult to demonstrate. For instance, consider the negating $-L$ law, where we know that $\mathsf{F}(\Theta)\theta \approx^{+} -A\theta$ by lemma C.3 (b) because $A\theta \approx^{+} A\theta$ by reflexivity, and $\mathsf{F}'(\Theta)\theta \approx^{+} -A\theta$ by lemma 5.1 (a). Additionally, the shifting $\downarrow_{\mathcal{S}}L$ law is sound because we know that $\mathsf{F}(\Theta)\theta \approx^{+} \downarrow_{\mathcal{S}}A\theta$ by lemma C.3 (a) because of the reflexive isomorphism $A\theta \approx^{-} A\theta$, and $\mathsf{F}'(\Theta)\theta \approx^{+} \downarrow_{\mathcal{S}}A\theta$ by lemma 5.1 (a).

And finally, the sub-structural laws for the binary (co-)data types $(\oplus, \otimes)$ require the most effort. This is because each of these types have two parts, and so we must relate one part at a time and then

mix the result together. In particular, we know that $\mathsf{F}_1(\Theta)\theta \approx^+ \mathsf{F}'_1(A,B)\theta$ and $\mathsf{F}_2(\Theta)\theta \approx^+ \mathsf{F}'_2(A,B)\theta$ for the declarations

$$\textbf{data}\ \mathsf{F}_1(\Theta) : \mathcal{V}\ \textbf{where}\ \mathsf{K}_1 : (A : \mathcal{V} \vdash \mathsf{F}_1(\Theta) \mid) \qquad \textbf{data}\ \mathsf{F}'_1(X:\mathcal{V}, Y:\mathcal{V}) : \mathcal{V}\ \textbf{where}\ \mathsf{K}'_1 : (X : \mathcal{V} \vdash \mathsf{F}'_1(\Theta) \mid)$$
$$\textbf{data}\ \mathsf{F}_2(\Theta) : \mathcal{V}\ \textbf{where}\ \mathsf{K}_2 : (B : \mathcal{V} \vdash \mathsf{F}_2(\Theta) \mid) \qquad \textbf{data}\ \mathsf{F}'_2(X:\mathcal{V}, Y:\mathcal{V}) : \mathcal{V}\ \textbf{where}\ \mathsf{K}'_2 : (Y : \mathcal{V} \vdash \mathsf{F}'_2(\Theta) \mid)$$

by applying lemma C.3 (a) to the reflexive isomorphisms $A\theta \approx^- X\{A\theta/X\}$ and $B\theta \approx^- Y\{B\theta/Y\}$. Now note the two different ways to mix these isomorphisms together with lemma C.2. First, we could mix the above $\mathsf{F}_1(\Theta)\theta \approx^+ \mathsf{F}'_1(A,B)\theta$ and $\mathsf{F}_2(\Theta)\theta \approx^+ \mathsf{F}'_2(A,B)\theta$ as the first two isomorphisms while the third is $\mathsf{F}_3(\Theta)\theta \approx^+ \mathsf{F}'_3(A,B)\theta$ given by lemma C.1 (a) of the trivial data declarations

$$\textbf{data}\ \mathsf{F}_3(\Theta) : \mathcal{V}\ \textbf{where}\ \mathsf{K}_3 : (\vdash \mathsf{F}_3(\Theta) \mid) \qquad \textbf{data}\ \mathsf{F}'_3(X:\mathcal{V}, Y:\mathcal{V}) : \mathcal{V}\ \textbf{where}\ \mathsf{K}'_3 : (\vdash \mathsf{F}'_3(X,Y) \mid)$$

which tells us that $\mathsf{F}(\Theta)\theta \approx^+ A \oplus B$ as required by the $\oplus L$ law. Second, we could mix together $\mathsf{F}_1(\Theta)\theta \approx^+ \mathsf{F}'_1(A,B)\theta$ and $\mathsf{F}_2(\Theta)\theta \approx^+ \mathsf{F}'_2(A,B)\theta$ as the second two isomorphisms while the first is $\mathsf{F}_0(\Theta)\theta \approx^+ \mathsf{F}'_0(A,B)\theta$ by lemma C.1 (b) of the trivial data declarations

$$\textbf{data}\ \mathsf{F}_3(\Theta) : \mathcal{V}\ \textbf{where} \qquad\qquad \textbf{data}\ \mathsf{F}'_3(X:\mathcal{V}, Y:\mathcal{V}) : \mathcal{V}\ \textbf{where}$$

which tells us that $\mathsf{F}(\Theta)\theta \approx^+ A \otimes B$ as required by the $\otimes L$ law. □

Theorem 5.3 (Polarized isomorphism substitution). *In the $\mathcal{VN}$ sub-calculus, for any types $\Theta, X : \mathcal{S} \vdash_{\mathcal{P}} A : \mathcal{T}$, $\Theta \vdash_{\mathcal{G}} B : \mathcal{S}$, and $\Theta \vdash_{\mathcal{G}} C : \mathcal{S}$, if $\Theta \vDash B \approx C$ then $\Theta \vDash A\{B/X\} \approx A\{C/X\}$.*

Proof. By induction on the typing derivation of $\Theta, X : \mathcal{S} \vdash_{\mathcal{P}} A : \mathcal{T}$ and the fact that each polarized connective is compatible with isomorphism. For example, in the case of $\otimes$, given $A_1 : \mathcal{V}$ and $A_2 : \mathcal{V}$ and that $A_1 \approx A'_1$ and $A_2 \approx A'_2$ from the inductive hypothesis, we have

$$\begin{aligned}
&\textbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A_1 \otimes A_2 : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^+_{\otimes L} \textbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A_1 : \mathcal{V}, A_2 : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
&\approx^+_{IH} \textbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A'_1 : \mathcal{V}, A'_2 : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
&\approx^+_{\otimes L} \textbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A'_1 \otimes A'_2 : \mathcal{V} \vdash \mathsf{F}_4() \mid)
\end{aligned}$$

by the $\otimes L$, data compatibility, and data mix laws, and so $A_1 \otimes A_2 \approx^+ \mathsf{F}_1() \approx^+ \mathsf{F}_4() \approx^+ A'_1 \otimes A'_2$ by lemma 5.1. The compatibility of the other polarized connectives follows similarly. The trickiest cases are for the shifts. ${}_{\mathcal{S}}\Uparrow$ follows immediately from lemma C.3 using a similar argument as $\otimes$ since the single component is call-by-value, and ${}_{\mathcal{S}}\Downarrow$ is dual since its component is call-by-name. $\downarrow_{\mathcal{S}}$ follows for the same reason as ${}_{\mathcal{S}}\Uparrow$ when $\mathcal{S} = \mathcal{V}$. Otherwise $\mathcal{S} = \mathcal{N}$ and we have $A : \mathcal{N}$, $A' : \mathcal{N}$, and so $A \approx A'$ implies $A \approx^- A'$ (lemma B.3 (b)), so $\downarrow_{\mathcal{N}} A \approx^+ \downarrow_{\mathcal{N}} A'$ by lemma C.3. Likewise, the compatibility of $\uparrow_{\mathcal{S}}$ follows dually to the case for $\downarrow_{\mathcal{S}}$. □

D PROOFS OF THE ALGEBRAIC AND DUALITY LAWS

As described in section 6, all of the follow laws are derived from (co-)data declarations of the form

$$\textbf{data}\ \mathsf{F}() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A : \mathcal{V} \vdash \mathsf{F}() \mid) \approx^+ \textbf{data}\ \mathsf{F}'() : \mathcal{V}\ \textbf{where}\ \mathsf{K}' : (A' : \mathcal{V} \vdash \mathsf{F}'() \mid)$$

$$\textbf{codata}\ \mathsf{G}() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\mid \mathsf{G}() \vdash A : \mathcal{N}) \approx^- \textbf{codata}\ \mathsf{G}'() : \mathcal{N}\ \textbf{where}\ \mathsf{O}' : (\mid \mathsf{G}'() \vdash A' : \mathcal{N})$$

via lemma 5.1 to get $A \approx A'$ by composing $A \approx^+ \mathsf{F}() \approx^+ \mathsf{F}'() \approx^+ A'$ or $A \approx^- \mathsf{G}() \approx^- \mathsf{G}'() \approx^- A'$. The specific (co-)data declaration isomorphisms needed for each law are derived from the laws for polarized connectives appearing in (co-)data declaration in fig. 10 extended with the general laws in figs. 8 and 9.

D.1 Proofs of the algebraic laws

D.1.1 Commutativity. The commutativity laws for reordering the binary connectives, unsurprisingly, follows from the commutativity laws for reordering the parts of declarations. For the multiplicative $\otimes$ and $\parr$, we use the first commute law to reorder the components within a single constructor or observer, as follows:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A \otimes B : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A : \mathcal{V}, B : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
&\approx^{+}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (B : \mathcal{V}, A : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (B \otimes A : \mathcal{V} \vdash \mathsf{F}_4() \mid)
\end{aligned}$$

$$\begin{aligned}
&\mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash A \parr B : \mathcal{N}) \\
&\approx^{-}_{\parr R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_2() \vdash A : \mathcal{N}, B : \mathcal{N}) \\
&\approx^{-}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_3() \vdash B : \mathcal{N}, A : \mathcal{N}) \\
&\approx^{-}_{\parr L}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_4() \vdash B \parr A : \mathcal{N})
\end{aligned}$$

Whereas for the additive $\oplus$ and $\&$, we use the second commute law to reorder the alternatives within a declaration as shown in the following isomorphism:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A \oplus B : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^{+}_{\oplus L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_1 : (A : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (B : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
&\approx^{+}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_2 : (B : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_1 : (A : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
&\approx^{+}_{\oplus L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (B \oplus A : \mathcal{V} \vdash \mathsf{F}_4() \mid)
\end{aligned}$$

$$\begin{aligned}
&\mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash A \,\&\, B : \mathcal{N}) \\
&\approx^{-}_{\& R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where}\ \mathsf{O}_1 : (\mid \mathsf{G}_2() \vdash A : \mathcal{N}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\mid \mathsf{G}_2() \vdash B : \mathcal{N}) \\
&\approx^{-}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where}\ \mathsf{O}_2 : (\mid \mathsf{G}_3() \vdash B : \mathcal{N}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_1 : (\mid \mathsf{G}_3() \vdash A : \mathcal{N}) \\
&\approx^{-}_{\& R}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_4() \vdash B \,\&\, A : \mathcal{N})
\end{aligned}$$

D.1.2 Unit. Combining the binary connectives with their corresponding units is an identity operation that leaves types unchanged, up to isomorphism. These unit laws rely on the fact that the right and left laws for the nullary connectives "cancel out," in an appropriate way, any occurence of the nullary connective within a (co-)data declaration as described by the $1L$, $0L$, $\bot R$, and $\top R$ laws. For the multiplicative 1 and $\bot$ connectives, we use the fact that 1 vanishes from the left-hand side of a constructor and $\bot$ vanishes from the right-hand side of an observer:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (1 \otimes A : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (1 : \mathcal{V}, A : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
&\approx^{+}_{1L}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
&\approx^{+}_{1L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A : \mathcal{V}, 1 : \mathcal{V} \vdash \mathsf{F}_4() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A \otimes 1 : \mathcal{V} \vdash \mathsf{F}_5() \mid)
\end{aligned}$$

$$\begin{aligned}
&\textbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_1() \vdash \bot \parr A : \mathcal{N}) \\
\approx^{-}_{\parr R}\ &\textbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_2() \vdash \bot : \mathcal{N}, A : \mathcal{N}) \\
\approx^{-}_{\bot R}\ &\textbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_3() \vdash A : \mathcal{N}) \\
\approx^{-}_{\bot R}\ &\textbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_4() \vdash A : \mathcal{N}, \bot : \mathcal{N}) \\
\approx^{-}_{\parr R}\ &\textbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_5() \vdash A \parr \bot : \mathcal{N})
\end{aligned}$$

Note the use of the mix law to extend $1L$ and $\bot R$ to allow for an extra component along side the unit connective. Alternatively, for the additive 0 and $\top$ connectives, we use the fact that any constructor containing a 0 on its left-hand side completely vanishes itself, whereas an observer containing a $\top$ on its right-hand side vanishes:

$$\begin{aligned}
&\textbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (0 \oplus A : \mathcal{V} \vdash \mathsf{F}_1()\ |\ ) \\
\approx^{+}_{\oplus L}\ &\textbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \textbf{where}\ \mathsf{K}_1 : (0 : \mathcal{V} \vdash \mathsf{F}_2()\ |\ ) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{K}_2 : (A : \mathcal{V} \vdash \mathsf{F}_2()\ |\ ) \\
\approx^{+}_{0L}\ &\textbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A : \mathcal{V} \vdash \mathsf{F}_3()\ |\ ) \\
\approx^{+}_{0L}\ &\textbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \textbf{where}\ \mathsf{K}_1 : (A : \mathcal{V} \vdash \mathsf{F}_4()\ |\ ) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{K}_2 : (0 : \mathcal{V} \vdash \mathsf{F}_4()\ |\ ) \\
\approx^{+}_{\oplus L}\ &\textbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A \oplus 0 : \mathcal{V} \vdash \mathsf{F}_5()\ |\ )
\end{aligned}$$

$$\begin{aligned}
&\textbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_1() \vdash \top \mathbin{\&} A : \mathcal{N}) \\
\approx^{-}_{\& R}\ &\textbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_2() \vdash \top : \mathcal{N}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{O}_2 : (\ |\ \mathsf{G}_2() \vdash A : \mathcal{N}) \\
\approx^{-}_{\top R}\ &\textbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_3() \vdash A : \mathcal{N}) \\
\approx^{-}_{\top R}\ &\textbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_4() \vdash A : \mathcal{N}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{O}_2 : (\ |\ \mathsf{G}_4() \vdash \top : \mathcal{N}) \\
\approx^{-}_{\& R}\ &\textbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_5() \vdash A \mathbin{\&} \top : \mathcal{N})
\end{aligned}$$

Again, the mix law is used to extend $0L$ and $\top R$ for (co-)data declarations with another alternative.

D.1.3 Associativity. Nested applications of the same binary connective can be reassociated, up to isomorphism. This is because (co-)data declarations are "flat:" there is a single, flat list of alternative, with each one containing a single, flat list of components on either side of the turnstyle. Therefore, after we fully unpack a nested application of a connective, it flattens out, so that we may repack the

same parts back together in the other order. For the multiplicative $\otimes$ and $\parr$, we have the following isomorphism:

$$\begin{array}{ll}
& \mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : ((A \otimes B) \otimes C : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
\approx^{+}_{\otimes L} & \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where} \\
& \qquad \mathsf{K} : (A \otimes B : \mathcal{V}, C : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
\approx^{+}_{\otimes L} & \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where} \\
& \qquad \mathsf{K} : (A : \mathcal{V}, B : \mathcal{V}, C : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
\approx^{+}_{\otimes L} & \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where} \\
& \qquad \mathsf{K} : (A : \mathcal{V}, B \otimes C : \mathcal{V} \vdash \mathsf{F}_4() \mid) \\
\approx^{+}_{\otimes L} & \mathbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \mathbf{where} \\
& \qquad \mathsf{K} : (A \otimes (B \otimes C) : \mathcal{V} \vdash \mathsf{F}_5() \mid)
\end{array}$$

$$\begin{array}{ll}
& \mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash (A \parr B) \parr C : \mathcal{N}) \\
\approx^{-}_{\parr R} & \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where} \\
& \qquad \mathsf{O} : (\mid \mathsf{G}_2() \vdash A \parr B : \mathcal{N}, C : \mathcal{N}) \\
\approx^{-}_{\parr R} & \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where} \\
& \qquad \mathsf{O} : (\mid \mathsf{G}_3() \vdash A : \mathcal{N}, B : \mathcal{N}, C : \mathcal{N}) \\
\approx^{-}_{\parr R} & \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where} \\
& \qquad \mathsf{O} : (\mid \mathsf{G}_4() \vdash A : \mathcal{N}, B \parr C : \mathcal{N}) \\
\approx^{-}_{\parr R} & \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where} \\
& \qquad \mathsf{O} : (\mid \mathsf{G}_4() \vdash A \parr (B \parr C) : \mathcal{N})
\end{array}$$

Note that the mix law is used to extend $\otimes L$ and $\parr R$ to allow for an extra component on either side of the main pair. For the additive $\oplus$ and $\&$, we have the following isomorphisms, again using mix to extend $\oplus L$ and $\& R$ to allow for an extra alternative before or after the main pair:

$$\begin{array}{ll}
& \mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : ((A \oplus B) \oplus C : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
\approx^{+}_{\oplus L} & \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_1 : (A \oplus B : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (C : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
\approx^{+}_{\oplus L} & \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_1 : (A : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (B : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_3 : (C : \mathcal{V} \vdash \mathsf{F}_3() \mid) \\
\approx^{+}_{\oplus L} & \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K}_1 : (A : \mathcal{V} \vdash \mathsf{F}_4() \mid) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (B \oplus C : \mathcal{V} \vdash \mathsf{F}_4() \mid) \\
\approx^{+}_{\oplus L} & \mathbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A \oplus (B \oplus C) : \mathcal{V} \vdash \mathsf{F}_5() \mid)
\end{array}$$

$$\begin{aligned}
&\textbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_1() \vdash (A \mathbin{\&} B) \mathbin{\&} C : \mathcal{N})\\
\approx^{-}_{\&R}\ &\textbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_2() \vdash A \mathbin{\&} B : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\ |\ \mathsf{G}_2() \vdash C : \mathcal{N})\\
\approx^{-}_{\&R}\ &\textbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_3() \vdash A : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\ |\ \mathsf{G}_3() \vdash B : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_3 : (\ |\ \mathsf{G}_3() \vdash C : \mathcal{N})\\
\approx^{-}_{\&R}\ &\textbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_4() \vdash A : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\ |\ \mathsf{G}_4() \vdash B \mathbin{\&} C : \mathcal{N})\\
\approx^{-}_{\&R}\ &\textbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_5() \vdash A \mathbin{\&} (B \mathbin{\&} C) : \mathcal{N})
\end{aligned}$$

D.1.4 Distributivity. Distributing a multiplication over an addition also arises from the flat nature of (co-)data declarations much like reassociating a binary connective. The difference is that when the addition is flattened out into the structure of the declaration, the multiplied type is carried along for the ride (via the mix law) and copied across both alternatives, as shown in the following isomorphisms:

$$\begin{aligned}
&\textbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A \otimes (B \oplus C) : \mathcal{V} \vdash \mathsf{F}_1() \mid)\\
\approx^{+}_{\otimes L}\ &\textbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \textbf{where}\ \mathsf{K} : (A : \mathcal{V}, B \oplus C : \mathcal{V} \vdash \mathsf{F}_2() \mid)\\
\approx^{+}_{\oplus L}\ &\textbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \textbf{where}\ \mathsf{K}_1 : (A : \mathcal{V}, B : \mathcal{V} \vdash \mathsf{F}_3() \mid)\\
&\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (A : \mathcal{V}, C : \mathcal{V} \vdash \mathsf{F}_3() \mid)\\
\approx^{+}_{\otimes L}\ &\textbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \textbf{where}\ \mathsf{K}_1 : (A \otimes B : \mathcal{V} \vdash \mathsf{F}_4() \mid)\\
&\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (A : \mathcal{V}, C : \mathcal{V} \vdash \mathsf{F}_4() \mid)\\
\approx^{+}_{\otimes L}\ &\textbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \textbf{where}\ \mathsf{K}_1 : (A \otimes B : \mathcal{V} \vdash \mathsf{F}_5() \mid)\\
&\qquad\qquad\qquad\qquad\quad \mathsf{K}_2 : (A \otimes C : \mathcal{V} \vdash \mathsf{F}_5() \mid)\\
\approx^{+}_{\oplus L}\ &\textbf{data}\ \mathsf{F}_6() : \mathcal{V}\ \textbf{where}\\
&\qquad \mathsf{K} : ((A \otimes B) \oplus (A \otimes C) : \mathcal{V} \vdash \mathsf{F}_6() \mid)
\end{aligned}$$

$$\begin{aligned}
&\textbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_1() \vdash A \parr (B \mathbin{\&} C) : \mathcal{N})\\
\approx^{-}_{\parr R}\ &\textbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \textbf{where}\ \mathsf{O} : (\ |\ \mathsf{G}_2() \vdash A : \mathcal{N}, B \mathbin{\&} C : \mathcal{N})\\
\approx^{-}_{\&R}\ &\textbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_3() \vdash A : \mathcal{N}, B : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\ |\ \mathsf{G}_3() \vdash A : \mathcal{N}, C : \mathcal{N})\\
\approx^{-}_{\parr R}\ &\textbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_4() \vdash A \parr B : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\ |\ \mathsf{G}_4() \vdash A : \mathcal{N}, C : \mathcal{N})\\
\approx^{-}_{\parr R}\ &\textbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \textbf{where}\ \mathsf{O}_1 : (\ |\ \mathsf{G}_5() \vdash A \parr B : \mathcal{N})\\
&\qquad\qquad\qquad\qquad\quad \mathsf{O}_2 : (\ |\ \mathsf{G}_5() \vdash A \parr C : \mathcal{N})\\
\approx^{-}_{\&R}\ &\textbf{codata}\ \mathsf{G}_6() : \mathcal{N}\ \textbf{where}\\
&\qquad \mathsf{O} : (\ |\ \mathsf{G}_6() \vdash (A \parr B) \mathbin{\&} (A \parr Cx) : \mathcal{N})
\end{aligned}$$

D.1.5 Annihilation. When a type is multiplied by the additive unit, it is cancelled out. This occurs because, unlike an addition, the multiplication places the type next to the unit where it is in harms

way. Thus, when $0L$ and $\top R$ are extended (by the mix law) to allow for an extra component alongside the units, it is swept aside as the entire alternative is deleted, as in the following isomorphisms:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A \otimes 0 : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A : \mathcal{V}, 0 : \mathcal{V} \vdash \mathsf{F}_2() \mid) \\
&\approx^{+}_{0L}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where} \\
&\approx^{+}_{0L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (0 : \mathcal{V}, A : \mathcal{V} \vdash \mathsf{F}_4() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (0 \otimes A : \mathcal{V} \vdash \mathsf{F}_5() \mid)
\end{aligned}$$

$$\begin{aligned}
&\mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash A \parr \top : \mathcal{N}) \\
&\approx^{-}_{\parr R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_2() \vdash A : \mathcal{N}, \top : \mathcal{N}) \\
&\approx^{-}_{\top R}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where} \\
&\approx^{-}_{\top R}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_4() \vdash \top : \mathcal{N}, A : \mathcal{N}) \\
&\approx^{-}_{\parr R}\ \mathbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_5() \vdash \top \parr A : \mathcal{N})
\end{aligned}$$

D.2 Proofs of the duality laws

D.2.1 Involutive negation. Double negation elimination is, perhaps, deceptively simple: the $-L$ and $\neg R$ laws just flip the double-negated type back and forth across the turnstyle until both negations disappear:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-(\neg A) : \mathcal{V} \vdash \mathsf{F}_1 \mid) && \mathbf{codata}\ \mathsf{G}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1 \vdash \neg(-A) : \mathcal{N}) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_2 \mid \neg A : \mathcal{N}) && \approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{O} : (-A : \mathcal{V} \mid \mathsf{G}_2 \vdash) \\
&\approx^{+}_{\neg R}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (A : \mathcal{V} \vdash \mathsf{F}_3 \mid) && \approx^{-}_{-L}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_3 \vdash A : \mathcal{N})
\end{aligned}$$

Note that in the case of $-(\neg A)$, $\neg R$ must be used in a data declaration instead of a co-data declaration, and likewise $-L$ must be used in a co-data declaration for $\neg(-A)$. This can be accomplished with the (co-)data interchange laws, that let us convert each data isomorphism from fig. 10 into a co-data isomorphism and vice versa.

D.2.2 Constant negation. Involutive negation converts "true" into "false" and "false" into "true," but it also swaps between the data and co-data formulations of each. For the multiplicative units, the data type 1 for true is the negation of the co-data type $\bot$ for false because both represent a (co-)data type with one alternative containing nothing:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-\bot : \mathcal{V} \vdash \mathsf{F}_1 \mid) && \mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1 \vdash \neg 1 : \mathcal{N}) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_2 \mid \bot : \mathcal{N}) && \approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (1 : \mathcal{V} \mid \mathsf{G}_2 \vdash) \\
&\approx^{+}_{\bot R}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_3 \mid) && \approx^{-}_{1L}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_3 \vdash) \\
&\approx^{+}_{1L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (1 : \mathcal{V} \vdash \mathsf{F}_4 \mid) && \approx^{-}_{\bot R}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_4 \vdash \bot : \mathcal{N})
\end{aligned}$$

For the additive units, the data type 0 for false is the negation of the co-data type $\top$ for true because both represent a (co-)data type with no alternatives:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-\top : \mathcal{V} \vdash \mathsf{F}_1() \mid) && \mathbf{codata}\ \mathsf{G}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash \neg 0 : \mathcal{N}) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_2() \mid \top : \mathcal{N}) && \approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{O} : (0 : \mathcal{V} \mid \mathsf{G}_2() \vdash) \\
&\approx^{+}_{\top R}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where} && \approx^{-}_{0L}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{V}\ \mathbf{where} \\
&\approx^{+}_{0L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (0 : \mathcal{V} \vdash \mathsf{F}_4() \mid) && \approx^{-}_{\top R}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_4() \vdash \top : \mathcal{N})
\end{aligned}$$

D.2.3 De Morgan laws. Involutive negation also converts "and" into "or" and "or" into "and" while interchanging data with co-data. For the multiplicatives, the connective $\otimes$ is an "and" pair that amalgamates two pieces of data into a single structure, and this is the negation of $\parr$ which is an "or" pair that conjoins two observers together:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-(A \parr B) : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_2() \mid A \parr B : \mathcal{N}) \\
&\approx^{+}_{\parr R}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_3() \mid A : \mathcal{N}, B : \mathcal{N}) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-A : \mathcal{V} \vdash \mathsf{F}_4() \mid B : \mathcal{N}) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-A : \mathcal{V}, -B : \mathcal{V} \vdash \mathsf{F}_5() \mid) \\
&\approx^{+}_{\otimes L}\ \mathbf{data}\ \mathsf{F}_6() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : ((-A) \otimes (-B) : \mathcal{V} \vdash \mathsf{F}_6() \mid)
\end{aligned}$$

$$\begin{aligned}
&\mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash \neg(A \otimes B) : \mathcal{N}) \\
&\approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (A \otimes B : \mathcal{V} \mid \mathsf{G}_2() \vdash) \\
&\approx^{-}_{\otimes L}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (A : \mathcal{V}, B : \mathcal{V} \mid \mathsf{G}_3() \vdash) \\
&\approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (A : \mathcal{V} \mid \mathsf{G}_4() \vdash \neg B : \mathcal{N}) \\
&\approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_5() \vdash \neg A : \mathcal{N}, \neg B : \mathcal{N}) \\
&\approx^{-}_{\parr R}\ \mathbf{codata}\ \mathsf{G}_6() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_6() \vdash (\neg A) \parr (\neg B) : \mathcal{N})
\end{aligned}$$

For the additives, the connective $\oplus$ is an "or" that yields one of two possible alternative types of answers, and this is the negation of & which gives observers the option of one of two possible types of questions:

$$\begin{aligned}
&\mathbf{data}\ \mathsf{F}_1() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (-(A \,\&\, B) : \mathcal{V} \vdash \mathsf{F}_1() \mid) \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_2() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : (\vdash \mathsf{F}_2() \mid A \,\&\, B : \mathcal{N}) \\
&\approx^{+}_{\&R}\ \mathbf{data}\ \mathsf{F}_3() : \mathcal{V}\ \mathbf{where}\ \begin{aligned}\mathsf{K}_1 &: (\vdash \mathsf{F}_3() \mid A : \mathcal{N}) \\ \mathsf{K}_2 &: (\vdash \mathsf{F}_3() \mid B : \mathcal{N})\end{aligned} \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_4() : \mathcal{V}\ \mathbf{where}\ \begin{aligned}\mathsf{K}_1 &: (-A : \mathcal{V} \vdash \mathsf{F}_4() \mid) \\ \mathsf{K}_2 &: (\vdash \mathsf{F}_4() \mid B : \mathcal{N})\end{aligned} \\
&\approx^{+}_{-L}\ \mathbf{data}\ \mathsf{F}_5() : \mathcal{V}\ \mathbf{where}\ \begin{aligned}\mathsf{K}_1 &: (-A : \mathcal{V} \vdash \mathsf{F}_5() \mid) \\ \mathsf{K}_2 &: (-B : \mathcal{V} \vdash \mathsf{F}_5() \mid)\end{aligned} \\
&\approx^{+}_{\oplus L}\ \mathbf{data}\ \mathsf{F}_6() : \mathcal{V}\ \mathbf{where}\ \mathsf{K} : ((-A) \oplus (-B) : \mathcal{V} \vdash \mathsf{F}_6() \mid)
\end{aligned}$$

$$\begin{aligned}
&\mathbf{codata}\ \mathsf{G}_1() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_1() \vdash \neg(A \oplus B) : \mathcal{N}) \\
&\approx^{-}_{\neg R}\ \mathbf{codata}\ \mathsf{G}_2() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (A \oplus B : \mathcal{V} \mid \mathsf{G}_2() \vdash) \\
&\approx^{-}_{\oplus L}\ \mathbf{codata}\ \mathsf{G}_3() : \mathcal{N}\ \mathbf{where}\ \begin{aligned}\mathsf{O}_1 &: (A : \mathcal{V} \mid \mathsf{G}_3() \vdash) \\ \mathsf{O}_2 &: (B : \mathcal{V} \mid \mathsf{G}_3() \vdash)\end{aligned} \\
&\approx^{-}_{\oplus L}\ \mathbf{codata}\ \mathsf{G}_4() : \mathcal{N}\ \mathbf{where}\ \begin{aligned}\mathsf{O}_1 &: (\mid \mathsf{G}_4() \vdash \neg A : \mathcal{N}) \\ \mathsf{O}_2 &: (B : \mathcal{V} \mid \mathsf{G}_4() \vdash)\end{aligned} \\
&\approx^{-}_{\oplus L}\ \mathbf{codata}\ \mathsf{G}_5() : \mathcal{N}\ \mathbf{where}\ \begin{aligned}\mathsf{O}_1 &: (\mid \mathsf{G}_5() \vdash \neg A : \mathcal{N}) \\ \mathsf{O}_2 &: (\mid \mathsf{G}_5() \vdash \neg B : \mathcal{N})\end{aligned} \\
&\approx^{-}_{\&R}\ \mathbf{codata}\ \mathsf{G}_6() : \mathcal{N}\ \mathbf{where}\ \mathsf{O} : (\mid \mathsf{G}_6() \vdash (\neg A) \,\&\, (\neg B) : \mathcal{N})
\end{aligned}$$

E NONLINEAR SHIFT LAWS

In section 1, we mentioned that the well-known isomorphism $(A \times B) \to C \approx A \to (B \to C)$ doesn't hold under the naïve reading in either ML or Haskell because currying and uncurrying are not truly inverses of each other. This is in stark contrast to the pure, call-by-name λ-calculus with products where the currying isomorphism *does* indeed hold. This is because in the call-by-name λ-calculus with products, we get to use the full η law for functions ($\lambda x.f\ x =_\eta f$) as well as the surjectivity law for products ($(\pi_1\ x, \pi_2\ x) =_{surj} x$). Then using the ordinary definition of *curry* and *uncurry* in terms of projection,

$$curry \triangleq \lambda f.\lambda x.\lambda y.f\ (x, y) \qquad\qquad uncurry \triangleq \lambda f.\lambda z.f\ (\pi_1\ z)\ (\pi_2\ z)$$

we can demonstrate that the composition $curry \circ uncurry$ is the identity function because of the call-by-name η law for functions

$$\begin{aligned} curry \circ uncurry &=_\beta \lambda f.curry\ (uncurry\ f) =_\beta \lambda f.\lambda x.\lambda y.uncurry\ f\ (x, y) \\ &=_\beta \lambda f.\lambda x.\lambda y.f\ (\pi_1(x, y))\ (\pi_2(x, y))) =_\pi \lambda f.\lambda x.\lambda y.f\ x\ y =_\eta \lambda f.f \end{aligned}$$

and we can also demonstrate that the reverse composition $uncurry \circ curry$ is the identity function because of the surjectivity law for products

$$\begin{aligned} uncurry \circ curry &=_\beta \lambda f.uncurry\ (curry\ f) =_\beta \lambda f.\lambda z.curryf\ (\pi_1\ z)\ (\pi_2\ z) \\ &=_\beta \lambda f.\lambda z.f\ (\pi_1\ z, \pi_2\ z) =_{surj} \lambda f.\lambda z.f\ z =_\eta \lambda f.f \end{aligned}$$

So what goes wrong in real programming languages like ML and Haskell? In ML, we can only rely on a restriction version of the η law, so that $\lambda y.(f\ x)\ y \neq f\ x$ because the function call $f\ x$ is a computation which might case effects—like nontermination, exceptions, or mutation—instead of a value like $\lambda y.(f\ x)\ y$ which causes no effects. In Haskell, we cannot rely on the surjectivity law for pair types, because the context $\mathbf{case}\ \Box\ \mathbf{of}\ (x, y) \to ()$ distinguishes between a constructed pair and a computation of a pair, so that $\mathbf{case}\ z\ \mathbf{of}\ (x, y) \to ()$ will not return any result if z never returns a pair (like any expression that causes an error or loops forever), but $\mathbf{case}\ (fst\ z, snd\ z)\ \mathbf{of}\ (x, y) \to ()$ always returns () regardless of what z is.

But then where does that leave our polarized analysis of such encodings? We should expect that if the call-by-name λ-calculus is capable of proving that currying and uncurrying form an isomorphism, then the analogous isomorphism should hold up in the polarized calculus as well. The issue is that since the polarized function space takes a *positive* argument rather than a negative one, we have that the purely call-by-name function type is $A \to_N B \approx (\downarrow A) \to B$ That means that the corresponding uncurried function type, where surjective products are represented as the negative & product type, is $\downarrow(A \,\&\, B) \to C$, which does not fit the mould of our currying isomorphism $(A \otimes B) \to C \approx A \to (B \to C)$ from fig. 14.

As it turns out, there are some additional laws about the shift connectives besides the simple identity laws we used in section 6.3. In particular, [Zeilberger 2009] mentions laws for distributing shifts over other connectives:

$$\downarrow(A \,\&\, B) \approx (\downarrow A) \otimes (\downarrow B) \qquad\qquad \downarrow\top \approx 1$$
$$\uparrow(A \oplus B) \approx (\uparrow A) \parr (\uparrow B) \qquad\qquad \uparrow 0 \approx \bot$$

The ability to distribute shifts over negative products lets us derive the law exactly corresponding to currying in the call-by-name λ-calculus as follows:

$$(A \,\&\, B) \to_N C \approx \downarrow(A \,\&\, B) \to C \approx (\downarrow A) \otimes (\downarrow B) \to C \approx (\downarrow A) \to (\downarrow B) \to C \approx A \to_N B \to_N C$$

The proofs of these distributive shift laws are actually interesting, because they differ from all the previous ones we used to encode (co-)data types. In particular, distributing shifts over the additive

connectives is explicitly *nonlinear*, and requires copying or erasing objects, which is in contrast to all the previous isomorphisms which are linear in the sense of linear logic (not to be confused with the similarly named "linear co-terms" from definition 4.2 which generalize co-values). For example, the isomorphism $\downarrow\top \approx 1$ is established by the following two commands:

$$c \triangleq \langle ()\|\alpha'\rangle : (x : \downarrow\top \vdash \alpha' : 1) \qquad\qquad c' \triangleq \langle \downarrow(\mu())\|\alpha\rangle : (x' : 1 \vdash \alpha : \downarrow\top)$$

Notice how we completely drop the input x and x' from these commands. The proof that the composition of these commands is the identity relies on the incredible strength of the $\eta^{\top}_{\mathcal{N}}$ and $\eta^{1}_{\mathcal{V}}$ laws, which let us prove that *every* value of the 1 and $\top$ types are interchangeable.

$$\begin{aligned}\langle \mu\alpha'.c\|\tilde{\mu}x'.c'\rangle &=_{\eta_\mu} \langle ()\|\tilde{\mu}x'.\langle\downarrow(\mu())\|\alpha\rangle\rangle =_{\tilde{\mu}} \langle\downarrow(\mu())\|\alpha\rangle \\ &=_{\tilde{\mu}\eta^{\downarrow}} \langle x\|\tilde{\mu}[\downarrow(y).\langle\downarrow(\mu())\|\alpha\rangle]\rangle =_{\eta^{\top}} \langle x\|\tilde{\mu}[\downarrow(y).\langle\downarrow(y)\|\alpha\rangle]\rangle =_{\eta^{\downarrow}} \langle x\|\alpha\rangle\end{aligned}$$

$$\langle \mu\alpha.c'\|\tilde{\mu}x.c\rangle =_{\eta_\mu} \langle\downarrow(\mu())\|\tilde{\mu}x.\langle()\|\alpha'\rangle\rangle =_{\tilde{\mu}} \langle()\|\alpha\rangle =_{\tilde{\mu}\eta^{1}_{\mathcal{V}}} \langle x\|\tilde{\mu}[().\langle()\|\alpha\rangle]\rangle =_{\eta^{1}} \langle x\|\alpha\rangle$$

Scaling up, the isomorphism $\downarrow(A \,\&\, B) \approx (\downarrow A) \otimes (\downarrow B)$ is established by the following two commands (where we make use of a little deep pattern matching for notational convenience):

$$\begin{aligned} c &\triangleq \langle x\|\tilde{\mu}[\downarrow(y).\langle(\downarrow(\mu\beta_1.\langle y\|\pi_1\,[\beta_1]\rangle), \downarrow(\mu\beta_2.\langle y\|\pi_2\,[\beta_2]\rangle))\|\alpha'\rangle]\rangle \\ &\quad : (x : \downarrow(A \,\&\, B) \vdash \alpha' : (\downarrow A) \otimes (\downarrow B)) \\ c' &\triangleq \langle x'\|\tilde{\mu}[(\downarrow(y_1), \downarrow(y_2)).\langle\downarrow(\mu(\pi_1\,[\beta_1].\langle y_1\|\beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2\|\beta_2\rangle))\|\alpha\rangle]\rangle \\ &\quad : (x' : (\downarrow A) \otimes (\downarrow B) \vdash \alpha : \downarrow(A \,\&\, B)) \end{aligned}$$

Notice that in c, the inner value of type $A \,\&\, B$ is necessarily duplicated and observed twice to extract both the A and B components to build the pair. Contrarily in c', the inner values $\downarrow A$ and $\downarrow B$ are each only referenced in one branch of the product case analysis, effectively dropping one or the other depending on which projection is requested which violates the normal discipline from linear logic. However, these commands are also inverses thanks to the correctly chosen disciplines for the respective types and the strength of the call-by-value $\eta^{\downarrow}_{\mathcal{V}}$ law.

$$\begin{aligned}&\left\langle \mu\alpha'.c \middle\| \tilde{\mu}x'.c' \right\rangle \\ &=_{\eta_{\tilde{\mu}}} \begin{array}{l}\langle \mu\alpha'.\langle x\|\tilde{\mu}[\downarrow(y).\langle(\downarrow(\mu\beta_1.\langle y\|\pi_1\,[\beta_1]\rangle), \downarrow(\mu\beta_2.\langle y\|\pi_2\,[\beta_2]\rangle))\|\alpha'\rangle]\rangle| \\ |\tilde{\mu}[(\downarrow(y_1), \downarrow(y_2)).\langle\downarrow(\mu(\pi_1\,[\beta_1].\langle y_1\|\beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2\|\beta_2\rangle))\|\alpha\rangle]\rangle\end{array} \\ &=_{\mu} \left\langle x \middle\| \tilde{\mu}\left[\downarrow(y).\ \begin{array}{l}\langle(\downarrow(\mu\beta_1.\langle y\|\pi_1\,[\beta_1]\rangle), \downarrow(\mu\beta_2.\langle y\|\pi_2\,[\beta_2]\rangle))| \\ |\tilde{\mu}[(\downarrow(y_1), \downarrow(y_2)).\langle\downarrow(\mu(\pi_1\,[\beta_1].\langle y_1\|\beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2\|\beta_2\rangle))\|\alpha\rangle]\rangle\end{array}\ \right] \right\rangle \\ &=_{\beta^{\otimes}} \left\langle x \middle\| \tilde{\mu}\left[\downarrow(y).\ \begin{array}{l}\langle\downarrow(\mu\beta_1.\langle y\|\pi_1\,[\beta_1]\rangle)| \\ |\tilde{\mu}[\downarrow(y_1).\langle\downarrow(\mu\beta_2.\langle y\|\pi_2\,[\beta_2]\rangle)\|\tilde{\mu}[\downarrow(y_2).\langle\downarrow(\mu(\pi_1\,[\beta_1].\langle y_1\|\beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2\|\beta_2\rangle))\|\alpha\rangle]\rangle]\rangle\end{array}\ \right] \right\rangle \\ &=_{\beta^{\downarrow}} \langle x\|\tilde{\mu}[\downarrow(y).\langle\mu\beta_1.\langle y\|\pi_1\,[\beta_1]\rangle\|\tilde{\mu}y_1.\langle\mu\beta_2.\langle y\|\pi_2\,[\beta_2]\rangle\|\tilde{\mu}y_2.\langle\downarrow(\mu(\pi_1\,[\beta_1].\langle y_1\|\beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2\|\beta_2\rangle))\|\alpha\rangle\rangle\rangle]\rangle \\ &=_{\tilde{\mu}_{\mathcal{N}}\mu} \langle x\|\tilde{\mu}[\downarrow(y).\langle\downarrow(\mu(\pi_1\,[\beta_1].\langle y_1\|\pi_1\,[\beta_1]\rangle \mid \pi_2\,[\beta_2].\langle y_2\|\pi_2\,[\beta_2]\rangle))\|\alpha\rangle]\rangle \\ &=_{\eta^{\&}} \langle x\|\tilde{\mu}[\downarrow(y).\langle\downarrow(y)\|\alpha\rangle]\rangle \\ &=_{\eta^{\downarrow}} \langle x\|\alpha\rangle\end{aligned}$$

$$
\begin{aligned}
\left\langle \mu\alpha.c' \middle\| \tilde{\mu}x.c \right\rangle &=_{\eta_{\tilde{\mu}}} \begin{array}{l} \langle \mu\alpha.\, \langle x' \| \tilde{\mu}[(\downarrow(y_1), \downarrow(y_2)).\langle \downarrow(\mu(\pi_1\,[\beta_1].\langle y_1 \| \beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2 \| \beta_2\rangle)) \| \alpha\rangle]\rangle \mid \\ \mid \tilde{\mu}[\downarrow(y).\langle(\downarrow(\mu\beta_1.\, \langle y \| \pi_1\,[\beta_1]\rangle), \downarrow(\mu\beta_2.\, \langle y \| \pi_2\,[\beta_2]\rangle)) \| \alpha'\rangle]\rangle \end{array} \\
&=_{\eta_{\tilde{\mu}}} \left\langle x' \middle\| \tilde{\mu}\left[(\downarrow(y_1), \downarrow(y_2)). \begin{array}{l} \langle \downarrow(\mu(\pi_1\,[\beta_1].\langle y_1 \| \beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2 \| \beta_2\rangle)) \mid \\ \mid \tilde{\mu}[\downarrow(y).\langle(\downarrow(\mu\beta_1.\, \langle y \| \pi_1\,[\beta_1]\rangle), \downarrow(\mu\beta_2.\, \langle y \| \pi_2\,[\beta_2]\rangle)) \| \alpha'\rangle] \end{array} \right] \right\rangle \\
&=_{\beta\downarrow\tilde{\mu}} \left\langle x' \middle\| \tilde{\mu}\left[(\downarrow(y_1), \downarrow(y_2)). \left\langle \begin{array}{l} (\downarrow(\mu\beta_1.\, \langle \mu(\pi_1\,[\beta_1].\langle y_1 \| \beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2 \| \beta_2\rangle) \| \pi_1\,[\beta_1]\rangle), \\ \downarrow(\mu\beta_2.\, \langle \mu(\pi_1\,[\beta_1].\langle y_1 \| \beta_1\rangle \mid \pi_2\,[\beta_2].\langle y_2 \| \beta_2\rangle) \| \pi_2\,[\beta_2]\rangle)) \end{array} \middle\| \alpha' \right\rangle \right] \right\rangle \\
&=_{\beta\downarrow\tilde{\mu}} \left\langle x' \middle\| \tilde{\mu}\left[(\downarrow(y_1), \downarrow(y_2)). \left\langle (\downarrow(\mu\beta_1.\, \langle y_1 \| \beta_1\rangle), \downarrow(\mu\beta_2.\, \langle y_2 \| \beta_2\rangle)) \middle\| \alpha' \right\rangle \right] \right\rangle \\
&=_{\beta\downarrow\tilde{\mu}} \left\langle x' \middle\| \tilde{\mu}\left[(\downarrow(y_1), \downarrow(y_2)). \left\langle (\downarrow(y_1), \downarrow(y_2)) \middle\| \alpha' \right\rangle \right] \right\rangle \\
&=_{\tilde{\mu}\eta^{\downarrow}_{\mathcal{V}}} \left\langle x' \middle\| \tilde{\mu}\left[(y_1, y_2). \left\langle (y_1, y_2) \middle\| \alpha' \right\rangle \right] \right\rangle \\
&=_{\eta^{\otimes}} \left\langle x' \middle\| \alpha' \right\rangle
\end{aligned}
$$

The other isomorphisms for distributing shifts, $\uparrow(A \oplus B) \approx (\uparrow A) \parr (\uparrow B)$ and $\uparrow 0 \approx \bot$ follow a dual construction as above.