Department of Computer Science

Research Report No. RR-03-05 ISSN 1470-5559 December 2003

Categories and Types for Axiomatic Domain Theory

Adam Eppendahl

Categories and Types for Axiomatic Domain Theory

Adam Eppendahl

Submitted for the degree of Doctor of Philosophy

University of London

2003

Categories and Types for Axiomatic Domain Theory

Adam Eppendahl

Abstract

Domain Theory provides a denotational semantics for programming languages and calculi containing fixed point combinators and other so-called paradoxical combinators. This dissertation presents results in the category theory and type theory of Axiomatic Domain Theory.

Prompted by the adjunctions of Domain Theory, we extend Benton's linear/nonlinear dual-sequent calculus to include recursive linear types and define a class of models by adding Freyd's notion of algebraic compactness to the monoidal adjunctions that model Benton's calculus.

We observe that algebraic compactness is better behaved in the context of categories with structural actions than in the usual context of enriched categories. We establish a theory of structural algebraic compactness that allows us to describe our models without reference to enrichment. We develop a 2-categorical perspective on structural actions, including a presentation of monoidal categories that leads directly to Kelly's reduced coherence conditions.

We observe that Benton's adjoint type constructors can be treated individually, semantically as well as syntactically, using free representations of distributors.

We type various of fixed point combinators using recursive types and function types, which we consider the core types of such calculi, together with the adjoint types. We use the idioms of these typings, which include oblique function spaces, to give a translation of the core of Levy's Call-By-Push-Value. The translation induces call-by-value and call-by-name translations of the core of Plotkin's Fixed Point Calculus.

Following Freyd, we construct a canonical fixed point operation from the algebras provided by the algebraic compactness of our models. Our analysis of Freyd's construction exposes a remarkable property of morphisms from coalgebras to algebras: morphisms from Gp to s correspond one-for-one to morphisms from p to Hs, where p is a coalgebra for HG and s an algebra for GH. We give an application of this property to the transposition of recursive coalgebras in Taylor's categorical theory of recursion where G is not left adjoint to H.

We develop a theory of parametric transformations corresponding to the uniformity property characterizing canonical fixed points and use this to derive abstract conditions on categories of domains which ensure that the interpretation of fixed point combinators coincides with the canonical fixed point operation.

Submitted for the degree of Doctor of Philosophy

University of London

2003

Contents

List of Figures

Chapter 1

Introduction

Domain theory provides denotational semantics for programming languages whose intended semantics is inconsistent with the properties of ordinary sets and functions. The theory grew out of the unexpected discovery of a denotational semantics for the untyped lambda calculus and has been the subject of constant reformulation. Axiomatic Domain Theory is an abstract formulation that accounts for certain aspects of concrete domains. We present various results in the category theory and type theory of Axiomatic Domain Theory. Although motivated by the structures of Axiomatic Domain Theory we hope our results will be useful to Categorical Semantics in general.

1.1 Background

The untyped lambda calculus is based on an abstraction operation whose intended semantics requires an invertible mapping from functions $S \to S$ to elements of S. Now it is impossible for a *set* S containing more than one element to cover the *set* of functions $S \to S$. So, even without additional features to make the calculus a practical programming language, the only possible semantics given by sets and functions is degenerate. Practical programming languages may be based on simpler operations with perfectly good set semantics, but eventually, with the addition of more sophisticated program constructs, the intended semantics 'goes recursive'. For example, a series of programming languages of increasing sophistication is described in Chapters 9 through 11 of [40]. Each language is given a denotational semantics in semantic domains specified by a set of 'domain equations'. In Chapters 9, 10 and 11 the equations are non-circular and have set solutions. The equation for D in Chapter 11 (page 300), however, is recursive and has no set solution.

Although an equation such as $D \cong D \to D$ has no useful solution if the operations are interpreted naïvely as constructions on sets, the existence of structures—'domains'—with closure properties stronger than sets allows us to write such equations. We must keep in mind, however, that we are using a metalanguage requiring careful interpretation. In other words, the form of denotational metalanguages follows the known closure properties of domains. In the language of Categorical Semantics: the type theory follows the categorical structure.

1.1.1 Axiomatic Domain Theory

Scott's semantics for the untyped lambda calculus is based on the properties of certain partially ordered sets. It is possible for a partially ordered set D containing more than one element to cover the (partially ordered) set of functions $D \to D$ *that preserve colimits of increasing chains in* D. The notions of colimit and increasing chain both use the order on D. Technically, ordered

sets provide a denotational semantics for the untyped lambda calculus and for programming languages such as the one in Chapter 11 of [40], but scientifically the order leads to new questions. A semantics may say that a given term or fragment of code denotes a particular function on a particular set with a particular order, but what is the significance of the order? On the one hand, we would like to ignore it and reason naïvely about our semantics. On the other, perhaps it reflects something important and useful.

So, although the collection of results about partially ordered sets known as Classical Domain Theory provides a new world for the semantics of programming languages, the existence of this new world raises difficult philosophical and technical questions. What is the significance of the order? What are domains *really*? What is their intrinsic logic? What is their intrinsic mathematics? These questions have lead mathematicians and computer scientists to reformulate the classical theory in various ways. Among these are several abstract formulations, including a collection of results known as Axiomatic Domain Theory.

These days, the standard conception of mathematics is entirely axiomatic and so Axiomatic Domain Theory is distinguished from Classical Domain Theory, Geometric Domain Theory and Synthetic Domain Theory not by having axioms but by the role and language of its axioms. In the classical theory, the language of ordered sets is used to axiomatize domains. In the geometric theory, the language of geometric logic is used. In the synthetic theory, the language of topos theory is used, first externally and then internally, to axiomatize first a mathematical universe and then an internal category of domains. In the 'axiomatic' theory, the language of category theory is used to axiomatize categories of domains and categories used to construct them.

While Geometric and Synthetic Domain Theory are guided by philosophical views of program semantics, Axiomatic Domain Theory is guided by the observation of mathematical structure. The concrete partial orders of the classical theory carry all sorts of structure, both at the level of individual partial orders and at the level of categories of partial orders. There are so many technical results about this structure that it becomes difficult to determine the status of any specific construction or property. For example, what is the status of CPO enrichment? The axiomatic theory selects some structure as primitive, derives other structure and presents the constructions and properties of these structures abstractly. For example, CPO enrichment has been shown to follow from certain abstract constructions on structure with certain abstract properties [11]. By way of comparison, Topos Theory does much the same thing for Set Theory: imagine that Topos Theory were known as Axiomatic Set Theory.

The epicentre of Axiomatic Domain Theory would have to be the notion of algebraic compactness. A category is algebraically compact if it has an invariant for every endofunctor (in some class) and these invariants are initial as algebras *and final as coalgebras*. We call such an invariant a 'Freyd algebra' (which is so much more satisfying, homonymically, than 'bifree algebra'). In [14], Freyd observes that certain concrete categories of domains are algebraically compact and shows how this can be used to model parameterized and mixed variance recursive types.

Following Freyd's results, Plotkin and Fiore took algebraic compactness as a target property for abstract constructions yielding categories of domains [10]. Given a categorical framework for partiality and induction, a category of partial maps is constructed and seen to be algebraically compact. This would appear to be done without any mention of ordered sets, but the existence of invariants follows from a derived order enrichment. However, because the order enrichment is derived from the purely categorical axioms for partiality and induction, it is fair to say that this provides an order-free account of categories of domains. On the other hand, the heavy use of enriched category theory presents a significant technical barrier to potential uses of this theory.

1.1.2 Fixed Points

While the solution of seemingly paradoxical domain equations is more than enough to motivate the use of partial orders, the order structure is useful just for its fixed points. The same colimits that are used to solve domain equations give fixed points that can be used to model iterative program constructs. For example, the semantics of the 'while' construct in Chapter 9 of [40] is given by a fixed point operation.

In some languages, such as the lambda calculus, a fixed point operator can actually be programmed. Given a partial order semantics for such a language, it is natural to ask how the fixed points given by the interpretation of such a program compare with the fixed points given by colimits in the partial order. For Scott's semantics of the untyped lambda calculus Park showed that the two coincide, but also constructed a non-standard semantics in which they don't [29].

In what might be seen as the early days of the Axiomatic Domain Theory, the notion of Natural Numbers Object (NNO) was weakened to a notion of Fixed Point Object (FPO) [28]. In Axiomatic Set Theory a NNO generates a collection of maps exemplified by the primitive recursive functions on the set natural numbers. In Axiomatic Domain Theory, an FPO generates maps exemplified by the recursive functions on the natural numbers ordered vertically, but unlike the successor on the naturals the successor on the vertical naturals has a fixed point and so a FPO also induces a collection of fixed points exemplified by least fixed points. A corresponding metalanguage with a fixed point operation is proposed in [8].

Because fixed point objects can be derived from a Freyd algebras, as observed in [14] and [28], attention has now shifted from the axiomatics of recursive maps and least fixed points to the axiomatics of recursive domains. Similarly, attention has shifted to metalanguages with recursive types such as Plotkin's FPC [25].

1.1.3 Categorical Semantics of Linear and Computation Types

In the early days of Categorical Semantics Lawvere showed that the categorical product abstracts essential structure from concrete presentations of algebraic theories. Similarly, cartesian closed categories were shown to abstract from lambda theories. In both cases, there is a slight tension between the concrete presentations and the categorical structure. Algebraic theories, in deference to Universal Algebra, are not simply categories with products. Lambda theories, in deference now to Category Theory, are not simply (typed) lambda calculi. But it is a good sort of tension and many theoreticians take it for granted that type theories are closely related to categorical structure.

With linear lambda calculi things get a bit hairy. It is immediately clear, for example, how tensor should be handled, but bang is more troublesome. Categorically, bang appears to be a comonad, but this becomes delicate syntactically. Two approaches have been successful, the more traditional Intuitionistic Linear Logic [5] and Dual Intuitionistic Linear Logic, a calculus with contexts divided into two areas [1]. Alternatively the comonad can be factored into a pair of adjoint functors. This leads to a calculus with two kinds of sequent, Linear/Non-Linear Logic [4]. The three approaches have been carefully compared in [24]. Interestingly, Linear/Non-Linear Logic produces a monad as well as a comonad.

Monads have been used to model type theories with explicit computation types such as the Computational Lambda Calculus [27]. More recently, it has been found useful to factor the monad into a pair of adjoint functors leading to a calculus with two kinds of sequent, Call-By-Push-Value [23].

On the categorical side, indexed comonads called *structural actions* have been used to abstract from the action of extending a linear context with a banged type [6]. A unified categorical framework for the algebra of computational monads, including an abstract account of the action of extending a computational context with a valued type [32], has been developed in [33].

1.2 Main Results

This thesis contains some original mathematics. These results are not difficult, but appear to be new and may have applications beyond their use here.

1.2.1 Recursive Morphisms

Given a monotonic endofunction F on an ordered set, it can be useful to consider the ordered set of pre fixed points for the endofunction: elements a such that $Fa \leq a$. For example, the endofunction has a least fixed point if and only if there is a least pre fixed point. The notion of pre fixed point appears quite technical but the generalization for endofunctors on categories has proven to be very useful. An *algebra* for an endofunctor F is an object a together with a map $s : Fa \to a$. For certain endofunctors, the notion of algebra is directly meaningful. Moreover, the dual notion of coalgebra is also meaningful.

The ordered set of pre fixed points generalises to a category of algebras. Because morphisms in the category of algebras must commute with the maps s, they are much more expressive than the order on the set of pre fixed points. Likewise, coalgebra morphisms are more expressive than the order on the set of post fixed points. Equally expressive is the notion of morphism from coalgebra to algebra. Such morphisms have received little attention in the literature. They are, however, natural mathematical objects and might have been discovered, for example, by asking what an algebra is for $\tilde{F} : \tilde{C} \to \tilde{C}$, where F is an endofunctor on C and $\tilde{C}$ is the category of twisted arrows Mac Lane describes in Exercise IX.6.3 of [22].

We call such morphisms 'recursive' because their defining condition, $f = s \circ F f \circ p$, says they may be rewritten in terms of themselves and because they seem to turn up whenever one looks at recursion categorically. In Freyd's theory of algebraic compactness, for example, they can be found in the proof of the Iterated Square Theorem. Actually, the proof carefully steps around them, but they are there if one looks.

Another place they turn up is in the proof of the dinaturality of initial algebra delivery. We observe that this follows from a remarkable property of recursive morphisms. Given a functor $G : A \to B$, dinaturality says any functor $H : B \to A$ takes initial algebras for GH to initial algebras for HG. Lemma 2.1.4 says that recursive morphisms from $p \to Hs$ correspond, in the manner of an adjunction, with recursive morphisms $Gp \to s$. For algebra morphisms or, dually, coalgebra morphims, any adjunction $G \dashv H$ is known to lift to such a correspondence, but for recursive morphisms the correspondence does not require an adjunction.

1.2.2 Structural Algebraic Compactness

It is fair to say that enriched category theory, on its own, does not support the basic theory of algebraic completeness. The problem concerns structure on the function delivering invariants. Given an invariant for every endofunctor on D, then for every functor $B \times D \to D$ we have a *delivery function* taking each object b of B to an object of D that is invariant under the action of b. Given an *initial* invariant for every endofunctor, we can use the initiality of each invariant to extend the delivery function to a functor $B \to D$.

In the motivating examples of such categories, however, we are only given invariants for enriched endofunctors. These invariants are initial so, for every enriched functor $B \times D \to D$, we obtain a functor $B \to D$, but initiality does not give an enrichment. In the motivating examples, it is known that these functors do enrich so one might simply strengthen the notion of algebraic completeness for enriched categories by asking for an enrichment of each delivery functor. Alternatively, it has been observed that in the motivating examples the given invariants satisfy a strengthened form of initiality that does give an enrichment. This suggests a strengthening of the notion of algebraic completeness for enriched categories by asking for invariants with strengthened initiality. When it is observed that the strengthened initiality follows from plain initiality

when the enrichment of D has cotensors, which it does in the motivating examples, this provides a workable notion of algebraic completeness for enriched categories [10].

A more sophisticated account of this solution has been developed using indexed and internal category theory [34]. It is observed that with mild conditions on the enriching category, which are satisfied in the motivating examples, enriched categories, functors and transformations give rise to indexed categories, functors and transformations which can be viewed as internal categories, functors and transformations. The strengthened form of initiality is then seen to be the externalization of plain internal initiality. Again, it is observed that internal initiality follows from plain, external initiality when the original enrichment has cotensors and so in the motivating examples the whole issue disappears.

When algebraically compact, and hence complete, categories of domains, are actually constructed they tend to come out enriched and compact in the strengthened sense described above. Either the construction starts with an enriched category or, as demonstrated in Axiomatic Domain Theory, an enrichment may be derived. Given a recursive type theory, however, we would like to describe compact models directly in terms of the structure used to interpret the type theory. From the enriched point of view, we do this using the cotensors that simplify the enriched theory of completeness.

The action of cotensors can be described without reference to enrichment, in which case they are called costructural actions. Functors and transformations that respect costructural actions are then called costructural. We observe that in the motivating examples enriched endofunctors correspond to costructural endofunctors and so we have an initial invariant for every costructural endofunctor. Initiality allows us to extend the delivery function to a functor, but it also gives us transformations making the functor costructural. Unlike the enriched setting, then, the costructural setting supports the basic theory of algebraic completeness.

Actually, we base our theory on the dual notions of tensor, structural action, final invariant and cocompleteness because structural actions are covariant in both arguments and because structural actions may be viewed as a mechanism for representing parameterized maps. Lemma 4.1.4 says that the delivery of final coalgerbas for structural endofunctors extends to a structural functor.

1.2.3 Adjoint Types and Free Distributor Representations

Benton decomposes the bang of Linear Logic into a sequence of two type constructors, U and L [3]. We see this as a nod towards domain models which have a lift operation: bang is LU and lift is UL. With L left adjoint to U, LU is a comonad and UL is a monad.

Syntactically, Benton's approach uses two forms of sequent, linear and non-linear. Semantically, these correspond to two categories, D and C. The constructor U corresponds to a functor $D \to C$ with a left adjoint corresponding to L. The category D is monoidal closed, C is cartesian closed and the adjunction is monoidal. We call such adjunctions LNL structures. It can be shown that Benton's two-sequent calculus LNL generates the free LNL structure (over a graph of basic types) and that the category of LNL theories is equivalent to the category of LNL structures.

We observe that, despite their conceptual origin as factors of LU or UL, U and L can be treated individually, both syntactically, which is easy, and semantically, which is less obvious. Our aim is to make the theory of adjoint types easier to understand conceptually and easier to use technically.

Categorically, our analysis is based on distributors (formerly known as profunctors). We appeal to a simple picture of free distributors lifted from the paths-in-a-directed-graph picture of free categories (see Appendix A). Distributors may be represented by functors—hence the term profunctor—in two ways, fore and aft. An adjunction is a pair of functors representing the same distributor in two ways. Proposition 7.2.3 says that when Benton's calculus is restricted to just U, it generates the free distributor with an aft-representation. Restricted to L, it generates the

free profunctor with a fore-representation. Together, therefore, the rules for U and L generate an adjunction, but this is an artifact of the double representation.

1.2.4 Quotient Relations

One of the principle motives behind the use of categorical structure is the prospect of precise definitions capturing intuitions about uniformity. In models of LNL terms are interpreted by transformations. The notion of transformation itself requires very little structure. Transformations, in general, are given by collections of edges $Fa \to Ga$ in one graph indexed by the nodes a of another. This includes domain and codomain node functions F and G. A given transformation, or class of transformations, often carries extra structure that allows us to express some notion of uniformity. Identifying a good notion of uniformity for some class of transformations is important because it allows us to abstract from particular transformations to any uniform transformation.

In many mathematical transformations the node functions lift to functors and the transformations are natural, but in some transformations interpreting terms in a type theory the node functions simply don't lift to functors and we must look beyond naturality. Here we consider a similar but more general notion of uniformity known as binary relational parametricity. The definitions we use can be found in Appendix C. Abstractly, binary relational parametricity is based on the observation that the node functions for term transformations, which interpret type constructions, lift to operators on graph categories, which are typically given by categories of binary relations. When a transformation lifts to a graph operator transformation we say it is *parametric* or, when the graph operators have been diagonalized, *diparametric*. In models of type theories, diparametricity accomodates the mixed variance of some type constructors, which breaks functorality and, hence, naturality.

Given such an abstract categorical framework for parametricity, we are expected to provide our own category of binary relations or some method of constructing such categories to obtain a useful notion of parametricity. We show that the uniformity that characterizes fixed point operators induced by fixed point objects corresponds to a form of relational (di)parametricity obtained from a particular category of relations we call 'quotient relations'.

Parametricity with respect to quotient relations is defined using pull-backs so any functor that preserves pull-backs lifts to quotient relations. Corollary 9.1.9 shows that function spaces automatically lift to quotient relations even though they generally don't preserve pull-backs

1.3 Additional Contributions

This dissertation also contains a number of conceptual and technical innovations. These are manifest in various original definitions that either convey the author's personal view of the subject or are useful in organising the mathematics.

1.3.1 Minimal Coherence

In addition to their stronger uses, universal properties often ensure that the equalities we least want to mention take care of themselves. Category theory has been very useful in helping to express and identify universal properties and category theory has also helped identify the next best thing to universal properties: coherence. Like universal properties, coherence provides the equalities that allow us to be glib. Unlike universal properties, coherence requires a small language of its own which involves a choice of primitives. One method of choosing these primitives is to borrow from an adjacent universal property. For example, when monoidal coherence was discovered the coherence conditions were borrowed from products. Later it was observed that the conditions can be simplified.

This use of coherence, as a means of ellipsis, requires a coherence theorem: from minimal conditions, maximum coherence. The point we would like to make here is that the choice of primitives and coherence conditions may begin and end with a specific application and be useful even without a coherence theorem. In Chapter 3 we sketch a 2-categorical picture of coherence conditions and their immediate application to the preservation of internal structures. When an internal structure is externalized as 2-cells, it is not hard to see what coherence, in the form of 2-cells, is required for a given 1-cell (functor) to preserve the given internal structure. We illustrate the procedure with lax left monoid structure. To our surprise this leads straight to the simplified monoidal coherence conditions, although the 2-categorical picture does not appear to suggest the coherence theorem that follows from these conditions.

For the notion of structural action, we use this 2-categorical perspective to identify the appropriate coherence conditions and to obtain a fine analysis of the canonical structural action on a monoidal category.

1.3.2 Fixed Point Algebras

An important consequence of algebraic compactness, which Freyd demonstrates in [14], is the fixed point operation it induces in the compact category and, more importantly, in related categories. Not only does this guarantee the existence of fixed points but it provides a canonical choice of fixed points analogous to least fixed points in categories of partial orders. In our axiomatics, the compact category D is related to a category C by an adjunction $L \dashv U : D \to C$. In Chapter 6 we use the compactness of D to construct a family of maps $\mathrm{fix}_d : (Ud \to\!\!\triangleright Ud) \to Ud$ in C that internalizes a fixed point operation for endomaps on objects of the form Ud. We show, following Freyd, that in categories of domains the resulting fixed point operation is characterized by a uniformity property analogous to Plotkin's Axiom for least fixed points.

A very similar account has appeared in [31]. We work with an adjunction instead of a comonad and provide details where [31] refers the reader to Freyd [14]. In particular, we introduce a notion of fixed point algebra that includes both Freyd algebras and fixed point objects and prove a transposition theorem that includes the transposition of Freyd algebras to fixed point objects and to parameterized fixed point objects.

A fixed point algebra induces a family of recursive morphisms. Given certain conditions, this is the unique uniform family of recursive morphisms. In the case of a Freyd algebra for the identity endofunctor on D, we obtain a family of maps analogous to bottom maps. In the case of a fixed point algebra for the identity endofunctor on C we obtain a family of maps analogous to least fixed points.

1.3.3 Exponential Structure

Our emphasis on structural actions plays down the role of enrichment because we concentrate on the structure used to model type constructions. However there is one aspect of enrichment that is important from this point of view as it is closely connected with the construction of structural actions in a monoidal adjunction. That is the enrichment of the monoidal category D in the cartesian category C via the monoidal functor $U : D \to C$. The important property of this enrichment is that the underlying category of D enriched in itself, via the function space $(e \multimap d)$, is isomorphic to the underlying category of D enriched in C, via the function space $U(e \multimap d)$. From the structrual point of view, the operation $U(e \multimap d)$ provides an alternative representation of parameterized maps in D.

We therefore introduce a notion of *exponential* that abstracts from this operation and from the hom objects of enriched structure generally. This leads to a pleasantly symmetrical definitons describing the structure on Kleisli aductions, which don't carry the full monoidal structure but still give models of recursive types.

1.3.4 Oblique Types

The calculus shown in Figure 8.4 matches a fragment of Levy's Call-By-Push-Value with recursive computation types and we use Levy's syntax to emphasise the match. Likewise, the translations given in Section 8.2.2 correspond to the call-by-name and call-by-value translations Levy studies in [23]. Levy's carefully motivated operational semantics, however, is wasted on the models we consider. Our models abstract from adjunctions given by the lift functor on categories of partial orders and lift can be viewed as a model of nontermination, but, as computational effects go, nontermination is extremely degenerate.

The point we would want to make here is that, independently of Levy's operational analysis, the importance of the oblique function space is indicated categorically by the notion of costructural action and type theoretically by the idioms used to type fixed point combinators. The latter point alone is enough to motivate a type theory with oblique function spaces in which fixed point combinators are more succinctly derived. We extend the minimal calculus that matches CBPV to a calculus with types modelled by structural actions and exponentials.

1.3.5 Directors

In our analysis of adjoint types we use various notions of free distributor. This views distributors not as 1-cells in bicategories generalizing categories of sets and relations, but as objects with structure that is respected by distributor morphisms. The idea of a free distributor then requires a forgetful functor includes distributors among objects with less structure.

We therefore use a notion of *director* which is a distributor without any composition. The situation is directly analogous to categories and directed graphs. Indeed the free distributor construction is given by a free category construction. The usefulness of these definitions results from the two different ways functors (or directed graph morphisms) induce distributors (or directors).

1.4 Overview

The idea of the thesis is to extend LNL with recursive types and to look at the interpretation of fixed point combinators in algebraically compact models. This is a fairly mundane undertaking and the bulk of the thesis concerns interesting things that turn up along the way. The majority of these things, those presented in Chapters 2, 3, 4, 5, and 6, are essentially categorical. Chapters 7 and 8 present some findings in type theory and Chapter 9 brings the two subjects closest with results about categorical models of types. Chapters 4 and 6 are an amplification of [14]. Chapter 3 can—and perhaps, given the mathematical tone of the thesis, should—be replaced by Appendix B. Personally, my favourite bits are Chapter 2 and Appendix A.

1.4.1 Category Theory

As LNL models are adjunctions, we want to say that an algebraically compact model is a an adjunction in which the one category is algebraically compact. For our definition of algebraically compact category, we use categories with structural actions. The notions of structural action and of structural endofunctor include natural transformations that must satisfy certain coherence conditions. We have therefore developed a general picture of coherence which is described in Chapter 3. Since we look at coherence conditions as polyhedra that may be pasted together, we use explicit 2-categorical language in that chapter, albeit in a very elementary way. The chapter includes a very condensed account of structural actions and the indexed category construction, but the 2-categorical picture suggests a more general and complicated notion of structural functor than we need for our theory of algebraic compactness. We therefore include an elementary account of structural actions and functors in Appendix B. The real interest of Chapter 3 is the integrated picture of internal and external structure that emerges.

In Chapter 4 we lift the theory of algebraic compactness from ordinary categories to categories with structural actions. Here we play down the 2-categorical perspective because the elementary theory is quite pretty. Also, the actions are directly meaningful as a mechanism for representing parameterized maps. We include some examples of compact categories from domain theory, but these are best viewed in the context of an adjuntion as described in Chapter 5. Although we are keen to show here that nothing more than the structural setting is *needed* for a precise description of compact categories, the more traditional enriched and internal settings are very closely related.

Equipped with a theory of compactness that says 'look for structural actions', it is interesting to revisit traditional descriptions of domain theoretic models. In Chapter 5 we first revisit monoidal adjunctions and then consider purely structural descriptions. We are encouraged in this by the Kleisli models which are missing parts of the monoidal structure but carry all the necessary structural actions. By the end of Chapter 5, we can say what an algebraically compact model is and are in a position to interpret an extended LNL.

An interpretation of LNL with recursive types is going to include fixed point operations so, before we consider such interpretations, we look at the native fixed point operations. We expect to have fixed points because the luff subcategory in which Freyd constructs fixed points is a special case of our adjunction. In Chapter 6 we dissassemble Freyd's construction and adapt it to the adjunctions we consider. We give an abstract description that includes fixed points for parameterized maps, and here the structural actions fit in very nicely, but the really interesting thing to come out of our analysis is actually used very weakly in the fixed point construction itself. That is a result concerning recursive morphisms.

Chapter 2 describes this result and gives an application to ordinary (non domain-theoretic) recursion theory. As recursive morphisms relate objects in one category to objects in an other, it is well to view them in terms of distributors. Because distributors also appear in our analysis of LNL, we have included in Appendix A a brief introduction to the aspects of distributors we employ.

Once we have described an extended LNL and played a little with the resulting type theory, we look at its interpretation in algebraically compact models. In the first section of Chapter 9 we develop a theory of uniform transformations that corresponds directly to the uniformity characterizing the fixed point operation derived in Chapter 6. The theory is presented in terms of a categorical framework for relational parametricity but without mention of the logical content usually associated with relational parametricy. The framework, or at least the part we use, is described in Appendix C. In the second section of Chapter 9 we apply the theory to the interpretation of fixed point combinators. This requires added conditions on our notion of algebraically compact model, but when these conditions are satisfied, as in concrete domain-theoretic adjunctions, the interpretation of any fixed point combinator is uniform and hence must coincide with the canonical fixed point operation. The coincidence result is somewhat contrived and it more the notion of quotient parametricity and its relation to fixed point uniformity we would like to emphasise.

1.4.2 Type Theory

In algebraically compact adjunctions it is the functors modelling linear types that have invariants, so in Chapter 7 we extend LNL with recursive linear types. Then, in Chapter 8, we find recursive linear types that allow us to type fixed point combinators.

In Chapter 7 we also take a close look at the main feature of LNL: mixed derivations of two kinds of sequent. Two special type constructors L and U are used to pass between the two kinds of sequent within a derivation. On the face of it, the constructors L and U are modelled by, and generate, adjoint functors. We show that these adjoint constructors can be understood separately if the term model for the calculus is viewed as a distributor. The concepts and terminology we

use are developed in Appendix A.

In Chapter 8 we pause to consider what sort of calculus might be useful in place of LNL. We have already seen in Chapter 5 that the monoidal adjunctions that motivated LNL are better viewed as structural adjunctions in the case of algebraically compact models. Moreover, The derivations of fixed point combinators use an idiom that corresponds to the construction of the costructural action in the structrual adjunction. We therefore propose a calculus motivated by structural adjunctions. This calculus is tentative and, as we observe in Chapter 10, other type idioms and categorical structure suggest variant forms of recursive type.

Chapter 2

Recursive Morphisms

Given an endofunctor $F : C \to C$, it is now commonplace to consider the category Alg$F$ of algebras and algebra morphisms for F or the category Coa$F$ of coalgebras and coalgebra morphisms [17]. In both cases, morphisms are given by maps that commute with 'constructors' or 'destructors' on objects of C. Here we investigate the analogous notion of morphism from the objects of Coa$F$ to the objects of Alg$F$.

Definition 2.0.3 *Given a coalgebra p for F and an algebra s for F, a* recursive morphism *from p to s is given by a map f such that $f = s \circ F f \circ p$.*

$$p \overset{f}{\dashrightarrow} s$$

$$\begin{array}{ccc} Fb & \xrightarrow{Ff} & Fa \\ {\scriptstyle p}\uparrow & & \downarrow{\scriptstyle s} \\ b & \xrightarrow{f} & a \end{array}$$

We call such maps 'recursive' because the equation $f = s \circ F f \circ p$ allows f to be rewritten in terms of itself. For certain choices of p, F and s, we recover familiar recursive function equations. If we take, for example, $F f = c + f$ and $s = \langle g, h \rangle$, we obtain the recursive equation

$$f(x) = \begin{cases} g(u) & \text{when } p(x) = \text{inl(u)} \\ h(f(y)) & \text{when } p(x) = \text{inr(y)} \end{cases}$$

$$\begin{array}{ccc} c + b & \xrightarrow{c+f} & c + a \\ {\scriptstyle p}\uparrow & & \downarrow{\scriptstyle \langle g,h \rangle} \\ b & \xrightarrow{f} & a \end{array}$$

and, if $c = 1$, so that g is just an element of a, and p is given by the predecessor on the naturals, we obtain the equation for primitive recursion.

$$f(n) = \begin{cases} g & \text{when } n = 0 \\ h(f(m)) & \text{when } n = m + 1 \end{cases}$$

$$\begin{array}{ccc} 1 + N & \xrightarrow{1+f} & 1 + a \\ {\scriptstyle \text{pred}}\uparrow & & \downarrow{\scriptstyle \langle g,h \rangle} \\ N & \xrightarrow{f} & a \end{array}$$

By taking power-set functors for F, more general forms of recursion can be obtained. This approach to recursion has been developed by Taylor [42, Section 6.3] and we apply some of our results to Taylor's framework in Section 2.2.9.

Originally, however, we came upon recursive morphisms as fixed points. If we take the identity on 1 for p, so that f is just an element of a, and $Ff = f$ we obtain the simple fixed point equation for f.

$$f = s(f) \qquad \begin{array}{ccc} 1 & \xrightarrow{f} & a \\ {\scriptstyle \mathrm{id}}\uparrow & & \downarrow{\scriptstyle s} \\ 1 & \xrightarrow{f} & a \end{array}$$

If we take $Ff = c \times f$ and $p = (\mathrm{id}_c, \mathrm{id}_c)$, we obtain a parameterized fixed point equation for f.

$$f(u) = s(u, f(u)) \qquad \begin{array}{ccc} c \times c & \xrightarrow{c \times f} & c \times a \\ {\scriptstyle (\mathrm{id},\mathrm{id})}\uparrow & & \downarrow{\scriptstyle s} \\ c & \xrightarrow{f} & a \end{array}$$

This is the sort of recursive morphism we actually use in our analysis of fixed point objects in Chapter 6. The broader importance of recursive morphisms is suggested in Section 2.2.5 where Freyd's diagram chase proof of the Square Theorem is reconstructed in terms of recursive morphisms and also by the simplicity of Lemma 2.1.4 in Section 2.1.2.

Mathematically, the important result is Lemma 2.1.4. In Section 2.2.3, we show how the (object level) dinaturality of initial algebra delivery can be understood in terms of the (object level) dinaturality of recursive coalgebra delivery which follows directly from the Lemma. In Section 2.2.9, we show how the Lemma applies to the transposition of recursive coalgebras and corecursive algebras. Recursive coalgebras feature prominently in Taylor's framework for recursion. Our use of corecursive algebras in Chapter 6 is fairly weak but a stronger use has recently turned up in work by Simpson and Escardo characterizing the real interval [38].

2.1 Oblique Adjunctions for Free

While it is known that adjunctions lift to categories of algebras and coalgebras for certain pairs of endofunctors including the pair given by the adjunction [16, Section 2.5], for recursive morphisms we obtain a form of adjunction even if the underlying functors are not adjoint.

2.1.1 The Distributor of Recursive Morphisms

Recursive morphisms do not form a category. Coalgebras are distinct from algebras, and so we have no identity recursive morphisms and no composable pairs on which to define composition of recursive morphisms. However, recursive morphisms do compose in the usual way with coalgebra morphisms on the one side and with algebra morphisms on the other.

$$\begin{array}{ccccccc} Fb' & \xrightarrow{Fg} & Fb & \xrightarrow{Ff} & Fa & \xrightarrow{Fh} & Fa' \\ {\scriptstyle p'}\uparrow & & {\scriptstyle p}\uparrow & & \downarrow{\scriptstyle s} & & \downarrow{\scriptstyle s'} \\ b' & \xrightarrow{g} & b & \xrightarrow{f} & a & \xrightarrow{h} & a' \end{array} \quad \mapsto \quad \begin{array}{ccc} Fb' & \xrightarrow{F(h \circ f \circ g)} & Fa' \\ {\scriptstyle p'}\uparrow & & \downarrow{\scriptstyle s'} \\ b' & \xrightarrow{h \circ f \circ g} & a' \end{array}$$

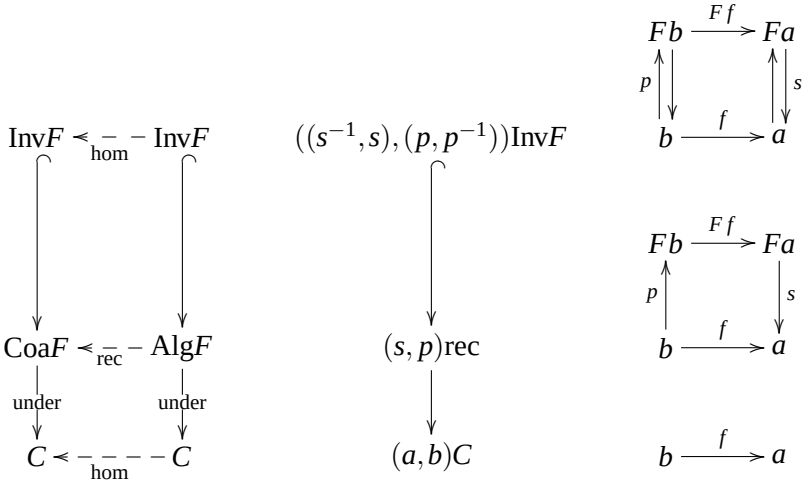
This gives a bimodular action

$$\mathrm{Coa}(p', p) \times (p, s)\mathrm{rec} \times \mathrm{Alg}(s, s') \longrightarrow (p', s')\mathrm{rec},$$

where $(s, p)\mathrm{rec}$ is the set of recursive morphisms to s from p. In other words, recursive morphisms give the oblique arrows of a distributor from $\mathrm{Alg}F$ down to $\mathrm{Coa}F$ (see Appendix A).

$$\mathrm{Coa}F \xleftarrow[\mathrm{rec}]{\;-\;} \mathrm{Alg}F$$

Both Coa$F$ and Alg$F$ include the category Inv$F$ of F invariants, which has mutually inverse coalgebra-algebra pairs for objects and coalgebra/algebra morphisms for arrows. When the distributor rec, is restricted to Inv$F$, it coincides with the hom distributor on Inv$F$. Both the hom distributors on Inv$F$ and rec lie over the hom distributor on C.



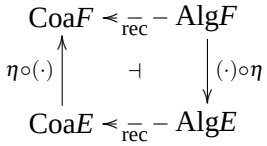
2.1.2 Two Oblique Adjunctions

Suppose we have a natural transformation $\eta : E \Rightarrow F$ from one endofunctor to another. Composition with η gives functors $\eta \circ (\cdot) : \text{Coa}E \to \text{Coa}F$ and $(\cdot) \circ \eta : \text{Alg}F \to \text{Alg}E$. By the naturality of η we have $f = s \circ (F f \circ \eta) \circ p$ iff $f = s \circ (\eta \circ E f) \circ p$, which means f is a morphism from $\eta \circ p$ to s iff f is a morphism from p to $s \circ \eta$. With respect to recursive morphisms, we have something of an adjunction

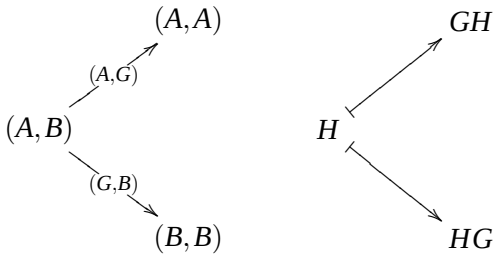
Proposition 2.1.3 *Given a coalgebra p for E, an algebra s for F and a natural transformation $\eta : E \Rightarrow F$, there is a bijection, natural in the choice of p and s, between morphisms from $\eta \circ p$ to s and morphisms from p to $s \circ \eta$.*

$$\frac{\eta \circ p \dashrightarrow s}{p \dashrightarrow s \circ \eta}$$

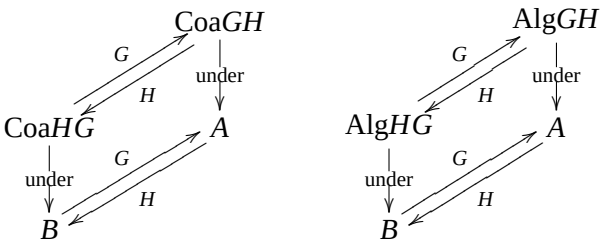
The proposition gives an oblique adjunction (see Section A.6).



Here is another one. Suppose we have a functor $G : B \to A$ that we compose with functors $H : A \to B$ to obtain pairs of endofunctors GH and HG.



The functor G lifts to algebras: it carries algebras for HG to algebras for GH and carries algebra morphisms for HG to algebra morphisms for GH. The functors H lift in the same way, taking algebras for GH to algebras for HG. The functors G and H also lift to coalgebras.



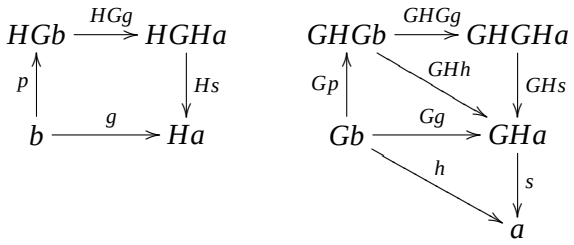
It turns out that an oblique adjunction exists between G on coalgebras and H on algebras, even if G and H are not themselves adjoint.

$$\begin{array}{ccc} \mathrm{Coa}GH & \prec_{\mathrm{rec}} - & \mathrm{Alg}GH \\ {\scriptstyle G}\uparrow & \dashv & \downarrow{\scriptstyle H} \\ \mathrm{Coa}HG & \prec_{\mathrm{rec}} - & \mathrm{Alg}HG \end{array}$$

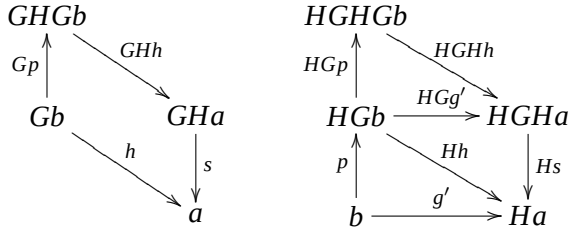
Lemma 2.1.4 *Given a coalgebra p for HG and an algebra s for GH, there is a bijection, natural in the choice of p and s, between morphisms from Gp to s and morphisms from p to Hs.*

$$\frac{Gp \dashrightarrow s}{p \dashrightarrow Hs}$$

Proof. The bijection is given by $g \mapsto s \circ Gg$ and $h \mapsto Hh \circ p$. If g is a recursive morphism from p to Hs, we see that $h = s \circ Gg$ is a recursive morphism from Gp to s by applying G to the diagram for g.



Likewise by applying H to the diagram for h, we see that $g' = Hh \circ p$ is a recursive morphism from p to Hs.



But then

$$g' = Hh \circ p = H(s \circ Gg) \circ p = Hs \circ HGg \circ p = g,$$

the morphism we started with. Similarly, $h' = s \circ G(Hh \circ p) = h$ starting with any morphism h from Gp to s. □

The Lemma gives an isomorphism between $G_* \circ \mathrm{rec}_{GH}$ and $\mathrm{rec}_{HG} \circ H^*$ in **Dist** (see Section A.6).

$$\begin{array}{ccc} \mathrm{Coa}GH & \prec_{\mathrm{rec}} & \mathrm{Alg}GH \\ \downarrow G_* & \cong & \downarrow H^* \\ \mathrm{Coa}HG & \prec_{\mathrm{rec}} & \mathrm{Alg}HG \end{array} \quad \mapsto \quad \begin{array}{ccc} \mathrm{Inv}GH & \prec_{\mathrm{hom}} & \mathrm{Inv}GH \\ \downarrow G_* & \cong & \downarrow H^* \\ \mathrm{Inv}HG & \prec_{\mathrm{hom}} & \mathrm{Inv}HG \end{array} \quad \mapsto \quad \begin{array}{c} \mathrm{Inv}GH \\ G \uparrow \dashv \downarrow H \\ \mathrm{Inv}GH \end{array}$$

If the isomorphism is restricted to the categories InvGH and InvHG of GH and HG invariants, we obtain an isomorphism directly between G_* and H^* and hence an adjunction $G \dashv H : \mathrm{Inv}GH \to \mathrm{Inv}HG$ (see Proposition A.6.4).

Now if we swap G and H in Lemma 2.1.4 and restrict again to invariants, we obtain a second adjunction $H \dashv G : \mathrm{Inv}HG \to \mathrm{Inv}GH$. The unit of this adjunction at (p, p^{-1}) is p while the counit of $G \dashv H$ at the same invariant is p^{-1}. Likewise, the counit of $H \dashv G$ is inverse to the unit of $G \dashv H$. The functors G and H therefore lift to form an equivalence between InvGH and InvHG. Freyd appeals to this equivalence for a direct proof that G preserves initial invariants [15, Section 5].

Proposition 2.1.5 *The functor G takes initial invariants for HG to initial invariants for GH.*

Note however that this is really a direct consequence of Lemma 2.1.4 and, using Proposition 2.2.2, should be viewed as a special case of Corollary 2.2.4.

2.2 Recursive Coalgebras and Corecursive Algebras

Recursive coalgebras (Definition 2.2.1) occur in Taylor's treatment of recursion where they are the coalgebras that 'obey the recursion scheme' [42, Section 6.3].

Definition 2.2.1 *An algebra σ is* corecursive *if for every coalgebra p there exists a unique morphism z_p from p to σ. Dually, a coalgebra π is* recursive *if there is a unique r_s from π to every algebra s.*

$$\begin{array}{ccc} Fb & \xrightarrow{Fz_p} & F\phi \\ {\scriptstyle p}\uparrow & & \downarrow{\scriptstyle \sigma} \\ b & \xrightarrow{z_p} & \phi \end{array} \qquad \begin{array}{ccc} F\omega & \xrightarrow{Fr_s} & Fa \\ {\scriptstyle \pi}\uparrow & & \downarrow{\scriptstyle s} \\ \omega & \xrightarrow{r_s} & a \end{array}$$

If t is a terminal object, then the unique algebra $Ft \to t$ is a corecursive algebra. Dually, if i is an initial object, then the unique coalgebra $i \to Fi$ is a recursive coalgebra. The following embellishment of Lambek's Lemma [21, Lemma 2.2] provides more examples.

Proposition 2.2.2 (after Lambek) *The following are equivalent:*

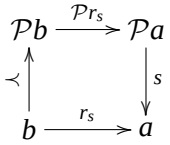
1. *p is a final coalgebra.*
2. *p is the inverse of a corecursive algebra.*

Dually, the following are equivalent:

1. *s is an initial algebra.*
2. *s is the inverse of a recursive coalgebra.*

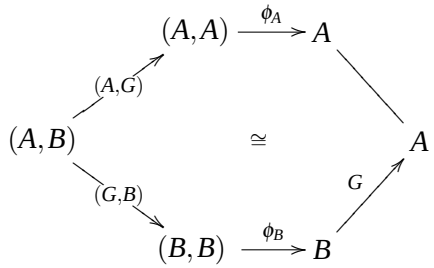
However, not every corecursive algebra or recursive coalgebra has an inverse. Also, unlike final coalgebras and initial algebras, corecursive algebras and recursive coalgebras are not unique up to isomorphism. If $n \mapsto n-1$ is taken as a functor on the partial order $(\mathbb{Z}, \leq)$, then every algebra is corecursive (there are no coalgebras), no algebra has an inverse and no two are isomorphic.

By the recursion theorem, every well-founded binary relation $\prec$ corresponds to a recursive coalgebra for the covariant power-set functor $\mathcal{P}$ on Set, but none of these is the inverse of an initial algebra for $\mathcal{P}$. For an account of well-foundedness and the general recursion theorem in a categorical context, see Taylor [42, Chapter 6].



2.2.3 Preservation of Recursive Coalgebras

Proposition 2.1.5 is the basis of a categorical generalisation of the dinaturality of the least fixed point operator: If ϕ_A and ϕ_B are functors giving (the carriers of) initial invariants for categories of endofunctors on A and B, then, because initial objects are unique up to coherent isomorphism, we get a natural isomorphism from the functor $G \circ \phi_B \circ (G,B)$ to the functor $\phi_A \circ (A,G)$. This is the dinaturality hexagon for ϕ_A and ϕ_B.



We now consider recursive coalgebras for G although, by symmetry, the same arguments apply to recursive coalgebras for H and, by duality, to corecursive algebras for G and for H. From the bijection of Lemma 2.1.4 we see that Gp is a recursive coalgebra whenever p is.

Corollary 2.2.4 *The functor G takes recursive coalgebras for HG to recursive coalgebras for GH*

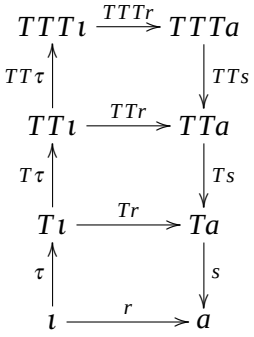
However, because recursive coalgebras are not unique up to coherent isomorphism, the provision of recursive coalgebra structure on the object part of a functor from (A,A) to A does not fix the arrow part of the functor. Similarly, even if ϕ_A and ϕ_B have been chosen to give (the carriers of) recursive coalgebras, the fact that $G\phi_B(HG)$ carries recursive coalgebra structure, does not fix a comparison with $\phi_A(GH)$. The uniqueness of the recursive morphism into algebras requires a coherent choice of arrows and a coherent choice of comparisons, but then ensures that these choices will be mutually coherent (when tested with recursive morphisms). Corollary 2.2.4 only ensures that dinaturality of recursive coalgebra delivery cannot fail at the object level.

2.2.5 Freyd's Square for Recursive Coalgebras

A weak form of Freyd's Square (Theorem 2.2.8 below) holds for recursive coalgebras.

Proposition 2.2.6 *If a recursive coalgebra π for TT is of the form $T\tau \circ \tau$, then τ is a recursive coalgebra for T.*

Proof. Suppose $T\tau \circ \tau$ is a recursive coalgebra for TT and s is an algebra for T. If r is a morphism from $T\tau \circ \tau$ to $s \circ Ts$, then so is $s \circ Tr \circ \tau$, but there is just one such morphism and so $r = s \circ Tr \circ \tau$, i.e. r is a morphism from τ to s. On the other hand, if $r = s \circ Tr \circ \tau$ then $Tr = Ts \circ TTr \circ T\tau$ and we have $r = s \circ Ts \circ TTr \circ T\tau \circ \tau$, i.e. r is a morphism from $T\tau \circ \tau$ to $s \circ Ts$.

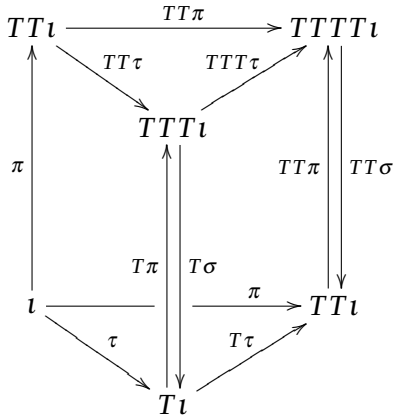


□

This proof sits inside the proof of the Square Theorem given in [14]. The following proposition provides a sufficient condition for the premise.

Proposition 2.2.7 *If a recursive coalgebra π for TT has an inverse, then π is of the form $T\tau \circ \tau$ where τ has an inverse.*

Proof. Let σ be inverse to π and let τ be the unique morphism from π to $T\sigma$. The map $T\tau$ is then a morphism from $T\pi$ to $TT\sigma$. Now $T\pi \circ T\sigma$ is the identity on $TTTι$, so $T\tau \circ \tau$ is a morphism from π to $TT\sigma$, but π is also a morphism from π to $TT\sigma$ because $TT\sigma \circ TT\pi$ is the identity on $TTι$ and so we must have $\pi = T\tau \circ \tau$.



Now we show τ has an inverse. By Corollary 2.2.4 (with T for G and the identity for H), if π is recursive, then so is $T\pi$. Let ς be the unique morphism from $T\pi$ to σ. Now the identity is a morphism from π to σ but, again because $T\pi \circ T\sigma$ is the identity, $\varsigma \circ \tau$ is also a morphism from π to σ and so must be the identity. Likewise, because $\pi \circ \sigma$ is the identity on $TTι$ and the identity on $Tι$ is a morphism from $T\pi$ to $T\sigma$, $\tau \circ \varsigma$ must be the identity. □

Using Proposition 2.2.2, Propositions 2.2.6 and 2.2.7 combine to form an important theorem.

Theorem 2.2.8 (Freyd's Iterated Square) *Initial algebras for TT are of the form $\varsigma \circ T\varsigma$ where ς is an initial algebra for T.*

2.2.9 Transposition of Corecursive Algebras and Recursive Coalgebras

Proposition 2.1.3 has the following corollary.

Corollary 2.2.10 *If σ is a corecursive algebra for F, then $\sigma \circ \eta$ is a corecursive algebra for E.*

If F is of the form HG we can put Lemma 2.1.4, together with Proposition 2.1.3, to obtain a correspondence between certain recursive morphisms into an algebra s for GH and arbitrary recursive morphisms into $Hs \circ \eta$.

$$\frac{\dfrac{G(\eta \circ p) \dashrightarrow s}{\eta \circ p \dashrightarrow Hs}}{p \dashrightarrow Hs \circ \eta}$$

We apply this correspondence to the transposition of corecursive algebras. Given a natural transformation $\eta : \mathrm{Id} \Rightarrow RL$, we call $\bar{s} = Rs \circ \eta$ the *transpose* of s, even if η is not the unit of an adjunction.

Corollary 2.2.11 *If σ is a corecursive algebra for LR, then its transpose, $R\sigma \circ \eta$, is a corecursive algebra for the identity endofunctor.*

Corollary 2.2.11 produces corecursive algebras for the identity endofunctor. If we also have a natural transformation $\iota : S \to \mathrm{Id}$ (not necessarily the counit of a comonad), then Corollary 2.2.10, with ι for η, takes corecursive algebras for the identity to corecursive algebras for S. Alternatively, we can produce corecursive algebras for S starting with a corecursive algebra τ for LSR, in which case the transformation ι is superfluous. With R for H, LS for G and $\eta_S : S \Rightarrow RLS$ for η, we see that corecursive algebras for LSR transpose to corecursive algebras for S.

$$\frac{\dfrac{LS(\eta_S \circ p) \dashrightarrow \sigma}{\eta_S \circ p \dashrightarrow R\sigma}}{p \dashrightarrow R\sigma \circ \eta_S}$$

This is so useful we make it a lemma and view Corollary 2.2.11 as the special case $S = \mathrm{Id}$.

Lemma 2.2.12 *Given functors $R : D \to C$ and $L : C \to D$, a natural transformation $\eta : \mathrm{Id} \to RL$ and an endofunctor $S : C \to C$, corecursive algebras for LSR transpose to corecursive algebras for S.*

The dual of Lemma 2.2.12 applies to the transposition of recursive coalgebras. Taylor's version of the recursion theorem is based on a categorical formulation of well-foundedness and says well-founded coalgebras are recursive [42, Section 6]. When the endofunctor in question is the covariant power-set functor on Set, we recover the notion of a well-founded relation and the theorem says recursive definitions based on well-founded relations are well-defined. This particular endofunctor carries monad structure that factors through a variety of adjunctions [41]. The typical situation is an adjunction $F \dashv U : D \to C$ topped by a monad T on D such that the original monad, possibly the T of another factorization, is given by the composite UTF. We may consider the notions of well-foundedness and recursiveness with respect to both the factor T and the original UTF. The dual of Lemma 2.2.12 says that recursive coalgebras for UTF transpose to recursive coalgebras for T.

Corollary 2.2.13 *If π is a recursive coalgebra for UTF then its transpose, $F\pi \circ \varepsilon$, is a recursive coalgebra for T.*

Corollary 2.2.13 uses the natural transfomation $\varepsilon : FU \Rightarrow \mathrm{Id}$ that comes with the adjunction $F \dashv U$ to transpose coalgebras for UTF into coalgebras for T. In the case of the power-set-like endofunctors studied in [41] there is another natural transformation $\kappa : T \Rightarrow TFU$ that can be used to transform coalgebras for T into coalgebras for UTF: compose $p : b \to Tb$ with $\kappa : Tb \to TFUb$ and apply U to obtain a coalgebra $U(\kappa \circ p) : Ub \to UTFUb$. With TF for H,

U for G and κ for η, we see that recursive coalgebras for T transform to recursive coalgebras for UTF.

$$\frac{\frac{U(\kappa \circ \pi) \dashrightarrow s}{\kappa \circ \pi \dashrightarrow TFs}}{\pi \dashrightarrow TFs \circ \kappa}$$

Corollary 2.2.14 *If π is a recursive coalgebra for T then $U(\kappa \circ \pi)$. is a recursive coalgebra for UTF.*

Note that the preservation of recursive coalgebras results from the mere existence of ε and κ and not from the laws of the adjunction. However, with power-set-like endofunctors, much more is true. The above transform reflects recursive coalgebras and both the transform and transposition preserve and reflect well-founded coalgebras [41].

Chapter 3

Monoidal Categories and Structural Actions

In Section 3.1 we illustrate our approach to coherence conditions using a notion of lax left monoid. We set up a unified account of monoids and monoidal structure which serves as basis for an account of comonoids and comonoidal structure. Our lax 2-categorical setting requires careful attention to the exact forms of duality relating monoidal and comonoidal structures. We show how comonoids in lax monoid categories are preserved by oplax monoidal functors. In Section 3.2 we describe how, despite our very weak approach to coherence, Kelly's simplified coherence conditions for monoidal categories may be derived from the notion of lax monoid.

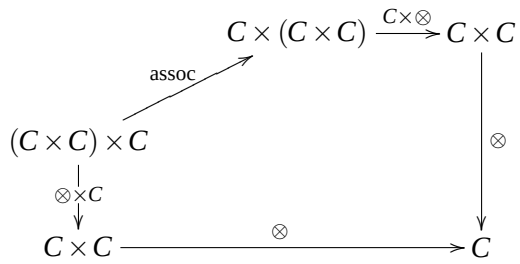
In Section 3.3 we indicate how the theory of structural actions fits into our picture of coherence and the preservation of comonoids. As in our derivation of monoidal coherence conditions, the conditions for structural actions correspond to structure on the transpose of the action. For an elementary account of structural actions see Appendix B.

3.1 Lax Monoids and Comonoids

In a category C with products, the notion of a monoid object can be given either by a product theory or directly in terms of various commutative diagrams. In the later case we don't use the universal properties of products so much as the structure induced by some choice of products. This amounts to monoidal structure which replaces the universal properties of products with the coherent isomorphisms those properties would induce were the monoidal multiplication actually to give products.

3.1.1 Lax Monoid 0-Cells

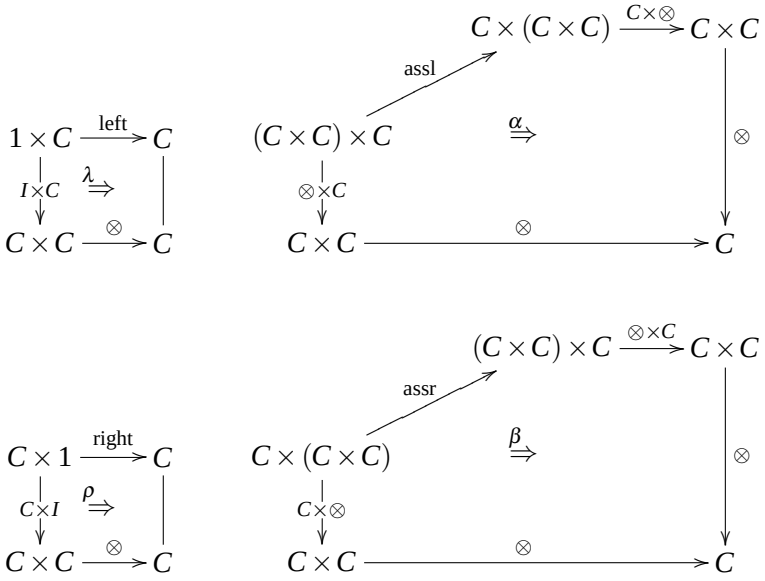
Given a monoidal 2-category C (with monoidal structure written as product structure), we can replace the commutative diagrams in the definition of monoid structure with 2-cells. We look at a diagram



as giving a 1-cell $(C \times C) \times C \to C$ that decomposes both as $\otimes \circ (\otimes \times C)$ and as $\otimes \circ (C \times \otimes) \circ \text{assoc}$ for some 1-cell $\otimes$. In a 2-category, this is to say the domain of the identity 2-cell on this 1-cell is

given as $\otimes \circ (\otimes \times C)$ and the codomain as $\otimes \circ (C \times \otimes) \circ \mathrm{assoc}$ for some 1-cell $\otimes$. Our definition of lax monoid structure allows, in place of the identity some 1-cell $(C \times C) \times C \to C$, any 2-cell with the same domain and codomain.

Definition 3.1.2 Lax monoid structure *on a 0-cell C of a monoidal 2-category C, is given by 2-cells λ, ρ, α and β whose domains and codomains are given by 1-cells $I : 1 \to C$ and $\otimes : C \times C \to C$ assembled as shown.*



We distinguish between left-hand and right-hand associativity because we are not assuming the monoidal structure on the ambient 2-category is symmetric and we don't want to rely on inverses to assl and assr. Much of this Chapter goes through without, say, the ρ and β, but this leads to categories without right identities in the indexed category construction in Section 3.3.6.

Definition 3.1.2 does not include coherence conditions. A lax monoid in the monoidal 2-category of categories is therefore not necessarily a monoidal category even if λ, ρ, α and β are all natural isomorphisms. Likewise, a lax monoid in the monoidal 2-category of 2-categories is not necessarily a monoidal 2-category. Note however that Definition 3.1.2 only uses the monoidal structure on C as lax monoid structure. So given a lax monoid C in the monoidal 2-category of 2-categories, we can recycle Definition 3.1.2 as a definition of lax monoid in C. Here we are only interested in the simpler case where we have a lax monoid C in the monoidal 2-category of categories. We make C into a 2-category by adding identity 2-cells and apply Definition 3.1.2 which now gives a definition of monoid in C. Because C is a degenerate 2-category whose only 2-cells are identities, we drop the 'lax' and the 2-cell diagrams of Definition 3.1.2 amount to commutative diagrams in C. When the lax monoid structure on C is given by monoidal structure, assl and assr are mutual inverses, the associativity diagrams imply one another and we recover the usual definition of monoid object in a monoidal category.

3.1.3 Lax Monoid 1-Cells

Say we have lax monoid structure on 0-cells C and C' of a monoidal 2-category C and we want a definition of compatible structure on 1-cells from C to C'. Rather than thinking in terms of morphisms that lax commute with the operations of our lax monoids, we obtain our definition by viewing a 1-cell from C to C' as a 0-cell in an arrow 2-category constructed over C and applying Definition 3.1.2 to this 0-cell.

Definition 3.1.4 *The* lax arrow 2-category $\mathrm{Arr}\mathsf{C}$ *is constructed over a 2-category C by taking 1-cells $F : C_0 \to C_1$ of C for the 0-cells of* $\mathrm{Arr}\mathsf{C}$*, pairs of 1-cells $G_0 : C_0 \to C_0'$ and $G_1 : C_1 \to C_1'$*

together with a 2-cell $\psi_G : G_1 \circ F \Rightarrow F' \circ G_0$ *for the 1-cells of* ArrC *and pairs of 2-cells* α_0 *and* α_1 *such that* $(F' \circ \alpha_0) \cdot \psi_G = \psi_{G'} \cdot (\alpha_1 \circ F)$ *(see Figure 3.1) for the 2-cells of* ArrC.

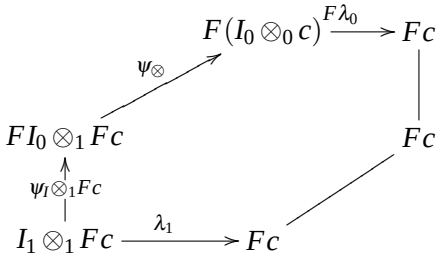
Interpreted in ArrC, Definition 3.1.2 gives our definition of monoidal structure on 1-cells of the original 2-category C. Monoidal structure on C lifts to monoidal structure on ArrC in the obvious way. Multiplication in ArrC is given by multiplying the corresponding diagrams in C, the identity on 1 in C becomes the unit for ArrC, and the naturality squares in C for the transformations left, right, assl and assr become the components of the corresponding natural transformations for ArrC. Clearly, structure on ArrC lifted in this way from C is strictly preserved by the domain and codomain projections back to C. Any lax monoid in ArrC defined with respect to monoidal structure lifted from C therefore projects to a pair of lax monoids[1] in C and so we can take lax monoid structure on a 1-cell F between two lax monoids C and C' to be given by any lifting of the pair of lax monoids to a lax monoid $F : C \to C'$ in ArrC.

Definition 3.1.5 Lax monoid structure *on a 1-cell* $F : C \to C'$ *in a 2-category* C *is given by lax monoid structure on* F *taken as a 0-cell in the 2-category* ArrC.

In ArrC, the four 2-cell diagrams in Definition 3.1.2 become four prisms in C. Figures 3.5 and 3.6 show the prisms for left identity and left associativity. These are drawn as cubes underlying lax squares in ArrC with each cube split into two hemi-cubes to match the compatibility cube shown in Figure 3.2. By stacking such prisms it is clear that lax monoid 1-cells compose to form composite lax monoid 1-cells. Lax monoids in C thus form a category over (the plain category part of) the 2-category C.

Definition 3.1.6 *The category* MonC *has lax monoid 0-cells in* C *for 0-cells and lax monoid 1-cells for 1-cells.*

The compatibility conditions on 2-cells in ArrC result in coherence conditions between the structure 2-cell for a lax monoid 1-cell $F : C_0 \to C_1$ and the structure 2-cells for the lax monoid structures on C_0 and C_1. Concretely, in Cat, lax monoid structure on a functor F requires natural transformations $\psi_\otimes$ and ψ_I making the following diagrams commute.



These are the standard coherence conditions on a monoidal functor in the theory of monoidal categories (which normally associates the natural transformations $\psi_\otimes$ and ψ_I with the functor F). So while Definition 3.1.2 does not give a definition of monoidal category when interpreted in Cat, Definition 3.1.5 does give the familiar definition of monoidal functor.

Just as Definition 3.1.2 can be recycled to give a definition of monoid in a category C that is itself a lax monoid in Cat, Definition 3.1.5 can be recycled to give a definition of monoid map in C. The arrow 2-category Arr$C$,—again treating C as a degenerate 2-category whose only 2-cells are the identities—is the degenerate 2-category obtained from the familiar arrow category $C^{\to}$. Just as monoidal structure lifts to the arrow category, the lax monoid structure on C lifts to Arr$C$

[1] Not to be confused with a lax monoid in $\mathsf{C} \times \mathsf{C}$. Definition 3.1.2 requires monoidal structure but we have not assumed the monoidal structure on C is symmetric and so the usual construction of monoidal structure on $\mathsf{C} \times \mathsf{C}$ is not available.

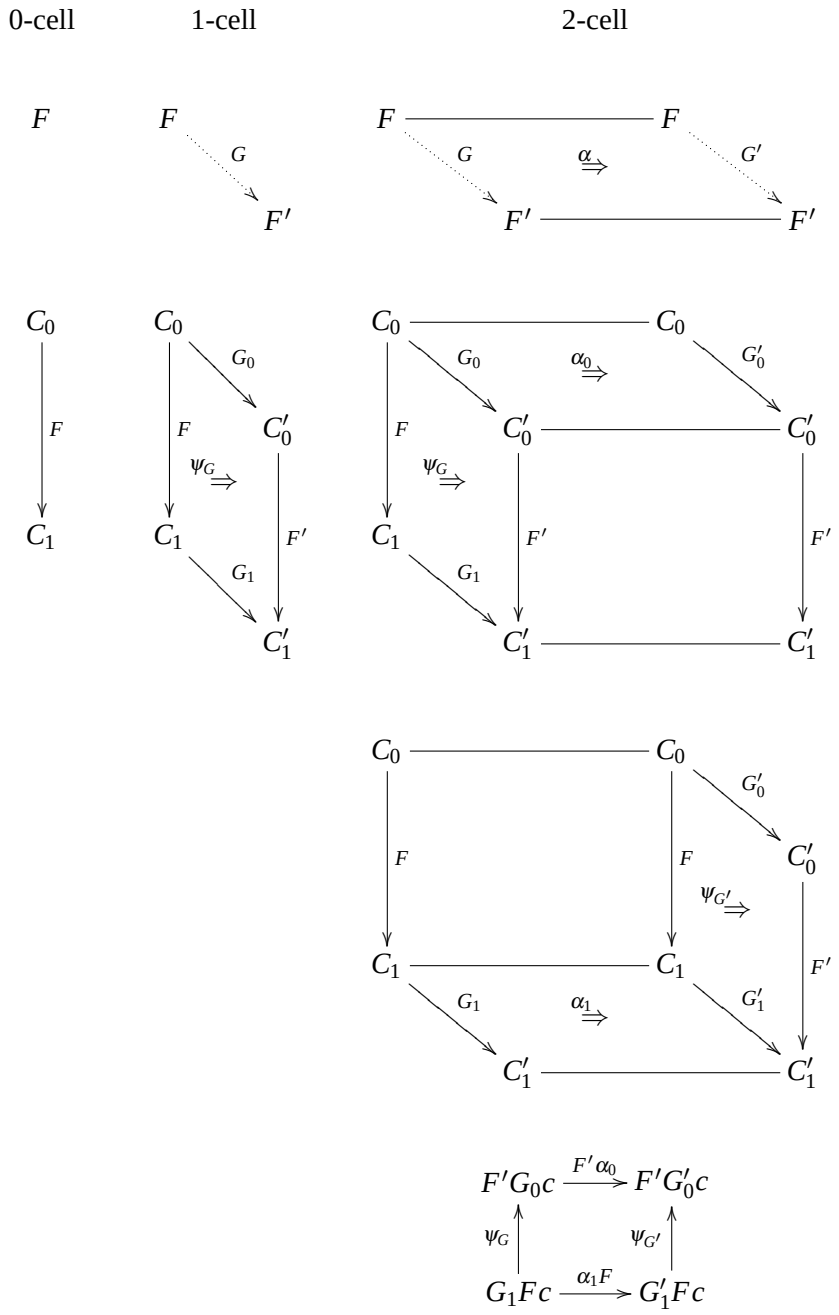


Figure 3.1: The arrow 2-category Arr$\mathsf{C}$ over C.

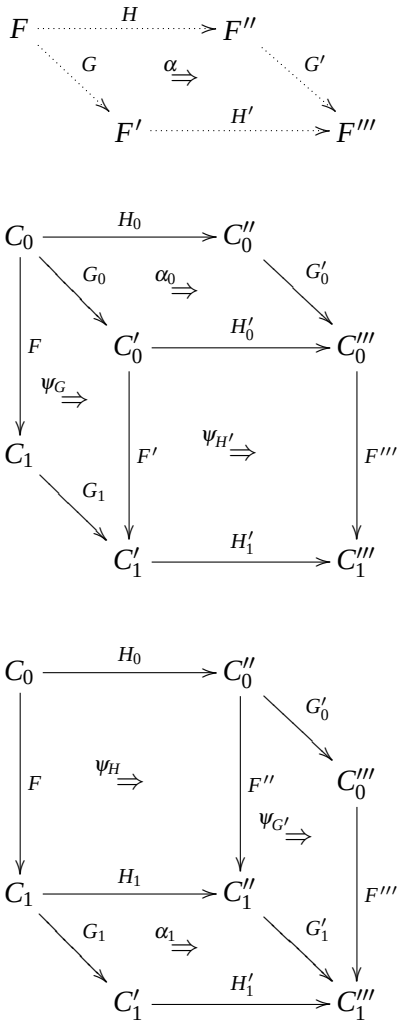


Figure 3.2: A lax square in ArrC.

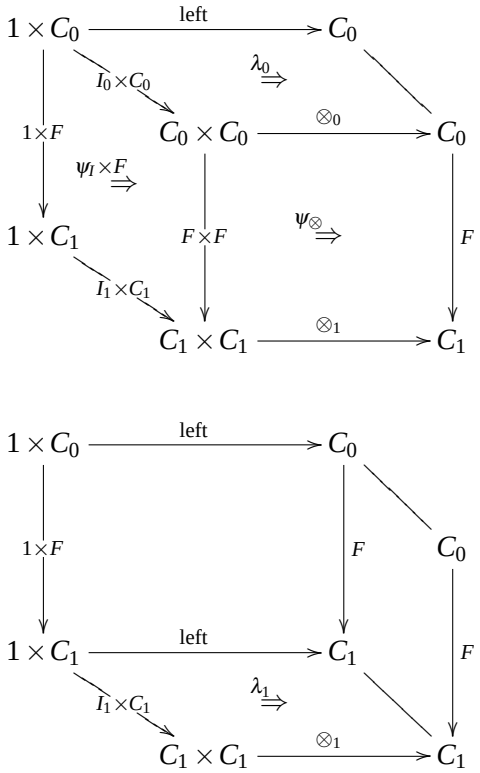


Figure 3.3: Compatibility with left identity.

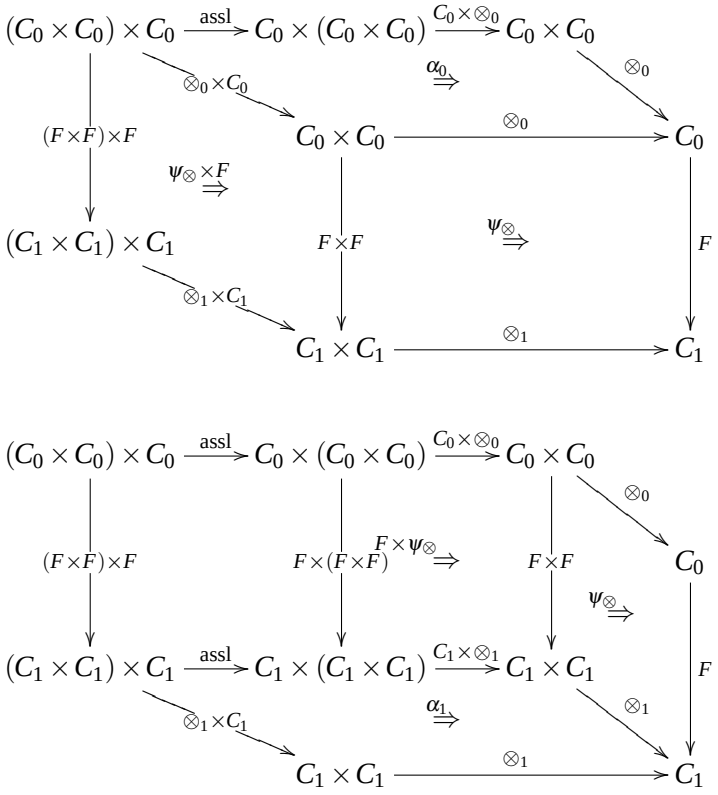


Figure 3.4: Compatibility with left associativity.

and we can apply Definitions 3.1.5 and 3.1.6 to obtain the category Mon$C$. When the lax monoid structure on C is given by monoidal structure, this is the usual category of internal monoids C.

As with monoidal structure, it is not necessary to preserve lax monoid structure strictly in order to carry internal structure from to one lax monoid to another.

Proposition 3.1.7 *Lax monoid functors lift to internal monoids.*

$$\begin{array}{ccc} \mathrm{Mon}C_0 & \xrightarrow{\mathrm{Mon}F} & \mathrm{Mon}C_1 \\ \downarrow{\scriptstyle \text{forget}} & & \downarrow{\scriptstyle \text{forget}} \\ C_0 & \xrightarrow{F} & C_1 \end{array}$$

Proof. The functor Mon$F$ takes a monoid

$$I_0 \xrightarrow{o} c \qquad\qquad c \otimes_0 c \xrightarrow{m} c$$

in C_0 to the monoid

$$I_1 \xrightarrow{Fo\circ\psi_I} Fc \qquad Fc \otimes_1 Fc \xrightarrow{Fm\circ\psi_\otimes} Fc$$

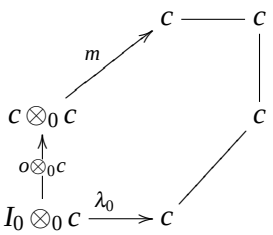
in C_1. To see this, think of c as a 1-cell $1 \to C_0$. Then o is a 2-cell

$$\begin{array}{ccc} 1 & \xleftarrow{\text{uniq}} & 1 \\ \| & \overset{o}{\Rightarrow} & \downarrow{\scriptstyle c} \\ 1 & \xrightarrow{I_0} & C_0 \end{array}$$

and m is a 2-cell

$$\begin{array}{ccc} 1 \times 1 & \xleftarrow{\text{diag}} & 1 \\ \downarrow{\scriptstyle c\times c} & \overset{m}{\Rightarrow} & \downarrow{\scriptstyle c} \\ C_0 \times C_0 & \xrightarrow{\otimes_0} & C_0 \end{array}$$

and the diagrams that make these a monoid in C_0 are given by four prisms (two of which are shown in Figures 3.5 and 3.6). For example, the prism shown in Figure 3.5 gives the diagram



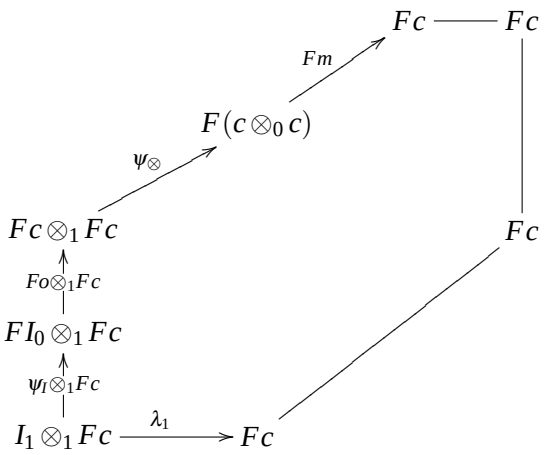
which is

$$\begin{array}{ccc} I_0 \otimes_0 c & \xrightarrow{\lambda_0} & c \\ \downarrow{\scriptstyle o\otimes_0 c} & & \| \\ c \otimes_0 c & \xrightarrow{m} & c \end{array}$$

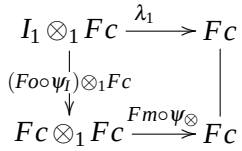
When these four prisms are stacked on the four prisms making F a lax monoid functor (two of which are shown in Figures 3.5 and 3.6), the four resulting prisms (one of which is shown in Figure 3.7) give the diagrams making

$$I_1 \xrightarrow{\psi_I} FI_0 \xrightarrow{Fo} Fc \qquad Fc \otimes_1 Fc \xrightarrow{\psi_\otimes} F(c \otimes_0 c) \xrightarrow{Fm} Fc$$

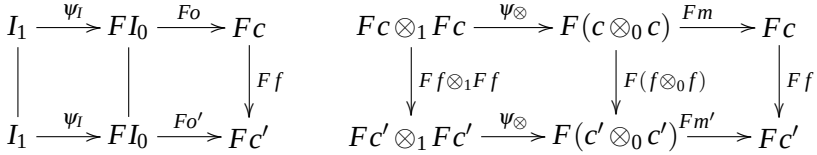
a monoid in C_1. For example, the stack shown in Figure 3.7 gives the diagram



which is



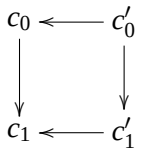
Monoid maps $f : c \to c'$ in C_0 become monoid maps $Ff : Fc \to Fc'$ in C_1 by the naturality of ψ_I and $\psi_\otimes$ and the funtorality of F.



□

3.1.8 Lax Comonoids

In a monoidal category C, comonoid structure on an object c is defined as monoid structure on c taken as an object of the opposite category C^{op}. A little more care is required for the definition of comonoid maps which are the opposites of monoid maps in the opposite category: the category of comonoids in C is given by $(\mathrm{Mon}C^{\mathrm{op}})^{\mathrm{op}}$. In terms of arrow categories, comonoid maps are monoid objects in the opposite of the arrow category over C^{op}. This has arrows for objects and op-squares



for arrows.

In a monoidal 2-category everything is more delicate. In a 2-category we can take opposites of either the 1-cells, the 2-cells or both and this leads to even more possibilities in the construction of arrow categories.

Definition 3.1.9 *Given a 2-category* C*, we write* $\mathrm{Op}\mathsf{C}$ *for the 2-category with op-2-cells (opposite hom categories).*

Taking the opposite of a category is therefore a 2-functor $(\cdot)^{\mathrm{op}} : \mathsf{Cat} \to \mathrm{Op}\mathsf{Cat}$.

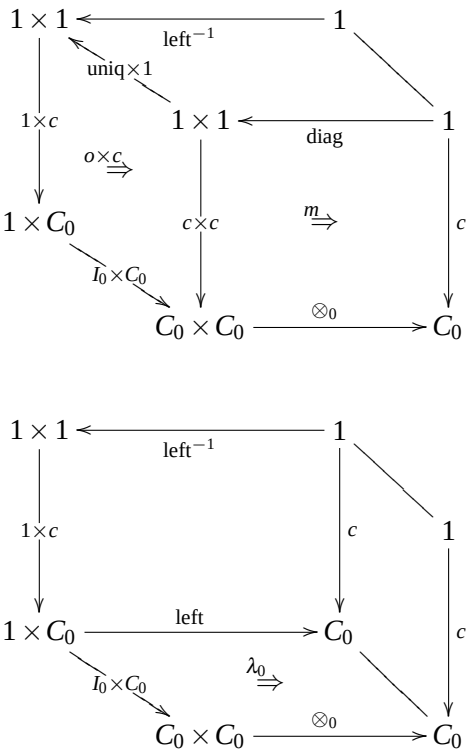


Figure 3.5: Internal left identity.

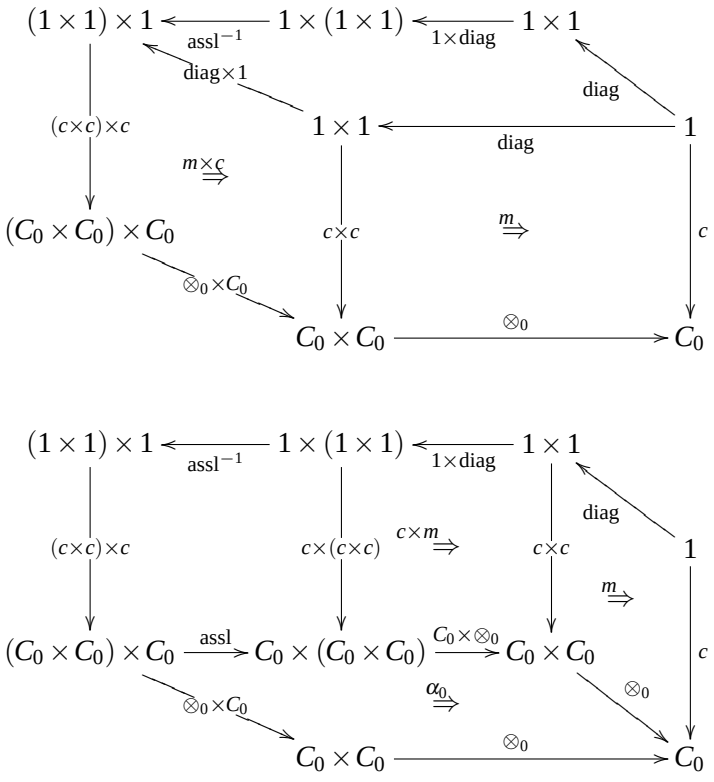


Figure 3.6: Internal left associativity.

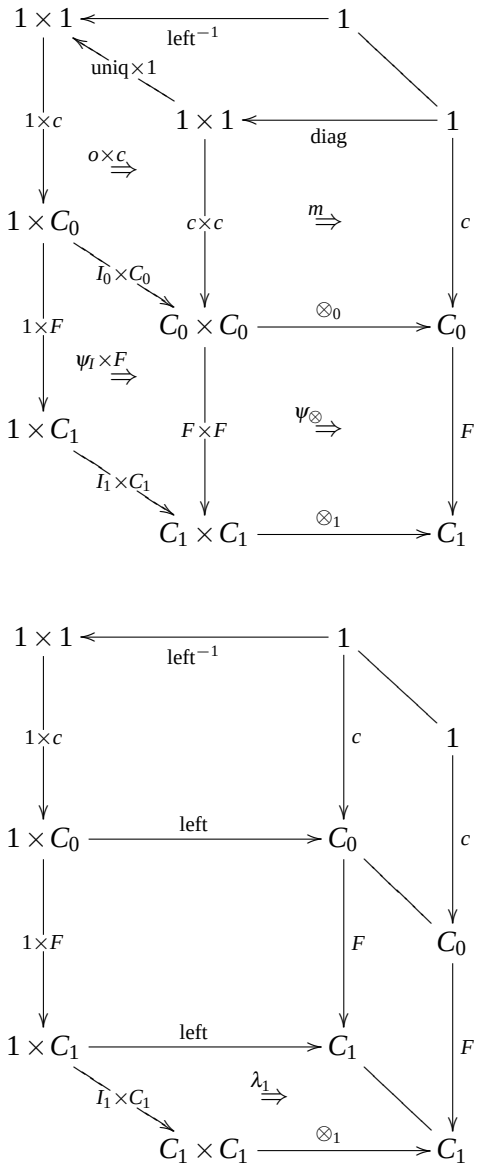


Figure 3.7: Internal left identity.

3.1.10 Preservation of Internal Comonoids

Lax monoid functors $F : C_0 \to C_1$ from one lax monoid category to another do not necessarily lift to carry comonoids in C_0 to comonoids in C_1. Taken as 2-cells, the counit and comultiplication of a comonoid are not confluent with the ψ in Arr so, although the identity and associativity diagrams for comonoid structure still give four prisms, these cannot be stacked on the correponding prisms for lax monoid structure on F. If we want to lift F to internal comonoids, we require *op*lax monoid structure on F.

Definition 3.1.11 Oplax monoid structure *on a 1-cell F is given by lax monoid structure on F taken as a 0-cell in* OpArrOpC.

The 2-category OpArrOpC is ArrC with the structure 2-cells ψ reversed. The hemi-cubes in Figure 3.1 must be taken as a complete cube to remain meaningful; with the ψ reversed, the cube falls into a different pair of confluent pasting diagrams. Note that the α_0 and α_1 are not reversed in OpArrOpC so oplax structure on $F : C_0 \to C_1$ projects to *lax* structure on C_0 and C_1. These projections are given by Opdom and Opcod from OpArrOpC to OpC, which is C, and preserve monoidal structure lifted from C. With the ψ reversed our prisms stack and we can lift F to internal comonoids.

Proposition 3.1.12 *Oplax monoid functors lift to comonoids.*

$$\begin{array}{ccc} \mathrm{CoMon}C_0 & \overset{\mathrm{CoMon}F}{\dashrightarrow} & \mathrm{CoMon}C_1 \\ \Big\downarrow{\scriptstyle \text{forget}} & & \Big\downarrow{\scriptstyle \text{forget}} \\ C_0 & \xrightarrow{\quad F \quad} & C_1 \end{array}$$

3.2 Monoidal Categories

Interpreted in Cat, Definition 3.1.2 does not give a definition of monoidal category because it does not include coherence conditions relating the 2-cells λ, ρ and α. Definition 3.1.5, however, does give the usual definition of monoidal functor including the usual coherence conditions and we can use this 'local coherence' to recover the familiar coherence conditions for monoidal categories. A monoidal category is a lax monoid whose multiplication $\otimes : C \times C \to C$ is monoidal when transposed to a functor $\hat{\otimes} : C \to [C,C]$.

3.2.1 The Monoidal Category of Endofunctors

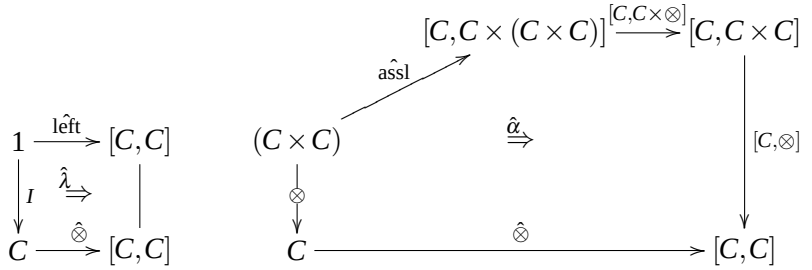
Each category $[C,C]$ of endofunctors and natural transformations carries two monoidal structures given by composition (carried out left-to-right or right-to-left). In symmetric monoidal 2-categories such as Cat these are inter-derivable, but here we take the left closed structure which fits our definition of lax monoid.

$$1 \xrightarrow{\text{ident}} [C,C] \qquad [C,C] \times [C,C] \xrightarrow{\text{comp}} [C,C]$$

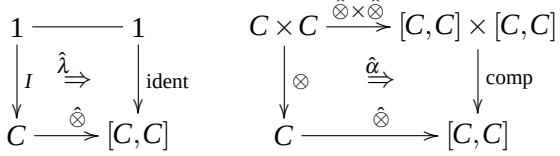
$$\cdot \longmapsto \mathrm{id}_C \qquad \langle g, f \rangle \longmapsto g \circ f$$

3.2.2 The Transpose of Lax Monoid Structure

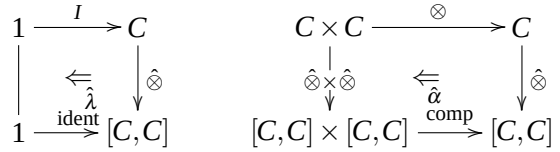
In a left closed monoidal 2-category such as Cat, the left identity and left associativity 2-cells of Definition 3.1.2 transpose to 2-cells with domain and codomain as shown.



In Cat the codomains can be rewritten.



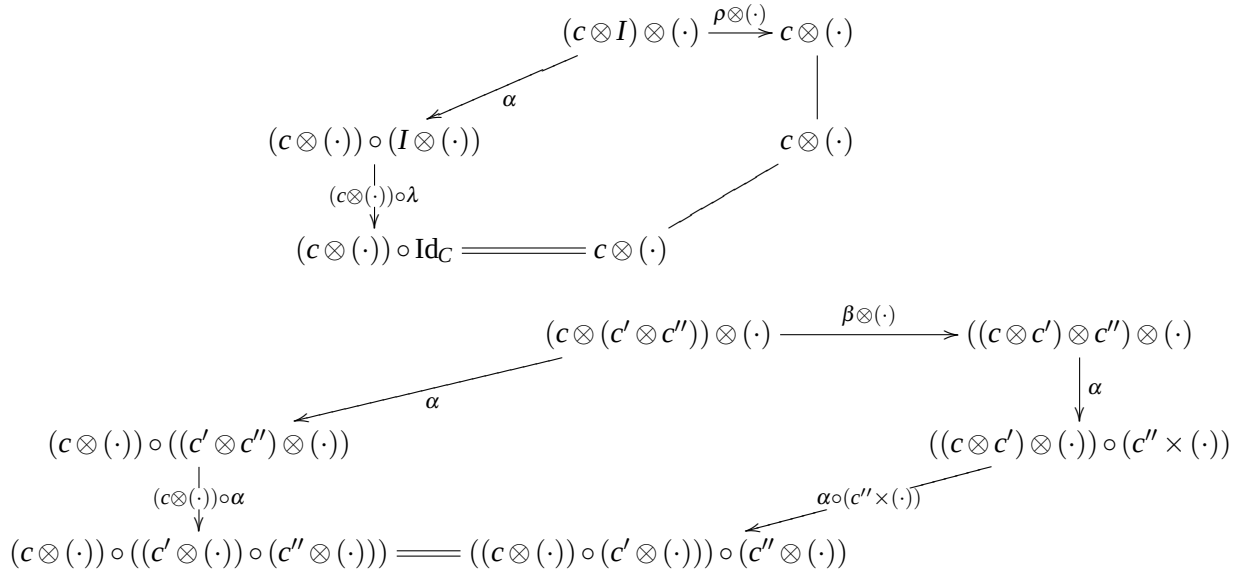
These diagrams can be viewed as unit and multiplication 1-cells in the arrow category OpArrOpCat lying over the unit and multiplication 1-cells of the lax monoids C and $[C,C]$ in Cat.



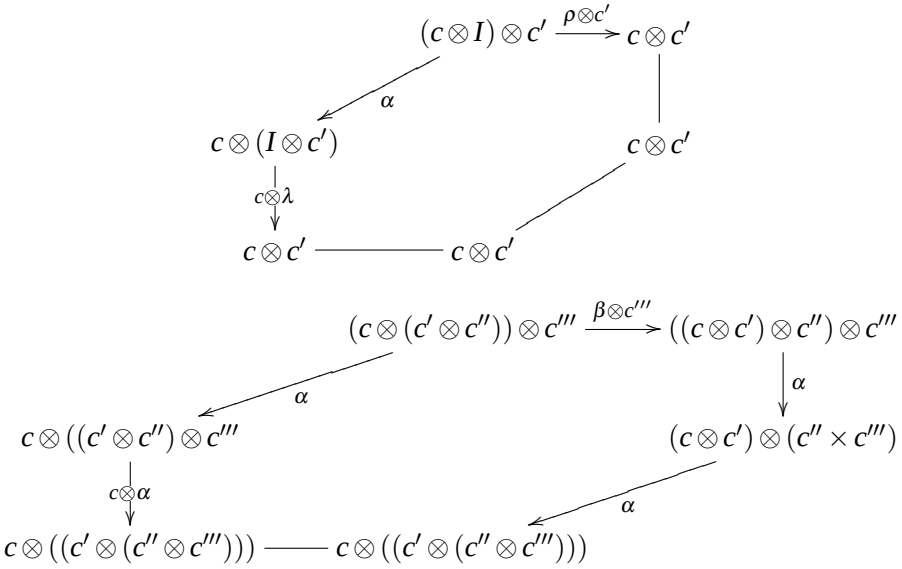
Lax monoid structure on a category C thus transposes to give potential oplax monoid structure on $\hat{\otimes}$. The natural transformation λ transposes to a potential $\psi_{\otimes}$ and the natural transformation α to a potential ψ_I.

3.2.3 Coherent Lax Monoids and Monoidal Categories

Given a lax monoid in Cat, the left transposed natural transformations constitute oplax monoid structure on $\hat{\otimes}$ exactly when the following diagrams commute in $[C,C]$. Double lines indicate equalities in $[C,C]$ which would be isomorphisms were C a bicategory.



These commute in $[C,C]$ exactly when the following diagrams commute in C for all c' and c'''.



When λ, ρ, α and β are natural isomorphisms, β is α^{-1} and we have the usual coherence diagrams for a monoidal category.

Definition 3.2.4 *A lax monoid in a left closed monoidal 2-category is* left coherent *if the left transpose of multiplication is monoidal.*

For coherent lax monoids then, the functor $\hat{\otimes}$ is oplax monoidal and so lifts to the categories of comonoids in C and $[C,C]$.

Proposition 3.2.5 *The 2-category of monoidal categories, monoidal functors and monoidal natural transformations is the sub-2-category of* StrMonCat *given by the coherent strong monoids.*

3.3 Structural Actions

Structural actions may be presented in 'curried form' as indexed comonads. If $\mathrm{Com}[A,A]$ is the category of comonads on A then structural actions of C on A correspond to functors from C to $\mathrm{Com}[A,A]$. This point of view simplifies the construction of the associated indexed category: the Kleisli construction gives a functor from $\mathrm{Com}[A,A]$ to Cat which can be composed with any structural action of C on A to obtain a C-indexed category. Note that this account of structural actions and their associated indexed categories relies on the closed structure of the 2-category Cat.

Structural actions may also be presented in 'uncurried form' as actions with extra structure. Given an action $\oslash : C \times A \to A$, we ask for a natural transformation with components $\delta_{ca} : c \oslash a \to c \oslash (c \oslash a)$ which we call pseudo-diagonals, and a natural transformation with components $\iota_{ca} : c \oslash a \to a$, which we call pseudo-terminals. These transformations must make the action of each object in C a comonad. The Kleisli category A_c for the action of an object c is then the fibre over c in the associated C-indexed category. We think of A_c as the category A with maps parameterized by c. Note that the action may be viewed as an endofunctor on A parameterized by C.

Structural actions may also be presented in terms of the canonical structural meta-action on Cat. The motivating example of a structural action is (the self-action of) cartesian multiplication: multiplication by an object C carries canonical comonad structure. The 2-category Cat is cartesian and the Kleisli construction for multiplication by a category C lifts to 2-cells, so we may construct the 2-category Cat_C. A structural action is a comonad in Cat_C.

Definition 3.3.1 *Given a category C, the 2-category* Str$C$ *has structural actions for 0-cells, (constant) structural functors for 1-cells and structural natural transformations for 2-cells.*

Definition 3.3.2 *Given a structural action C on a 2-category* K*, the 2-category* Str$C$ *has comonads in* K_C *for 0-cells, (constant) comonad functors in* K_C *for 1-cells and comonad transformations in* K_C *for 2-cells.*

A comonad is a comonoid in the functor category $[D,D]$, but can be defined without constructing $[D,D]$.

3.3.3 Diagonal Structure

In the proof of Proposition 3.1.12, coherence is only used on the diagonal of $C \times C$. This appears in the orientation of the arrow $1 \to 1 \times 1$: the necessary diagrams in C are indexed by singletons. In that proof, lax squares of the form

$$\begin{array}{ccc} X & \longleftarrow & Y \\ \downarrow & \Leftarrow & \downarrow \\ C & \longrightarrow & D \end{array}$$

are pasted vertically onto lax squares of the form

$$\begin{array}{ccc} C & \longrightarrow & D \\ \downarrow & \Leftarrow & \downarrow \\ C' & \longrightarrow & D' \end{array}$$

to obtain a lax square of the first form. When viewed as a horizontal arrow this first form can be interpreted as a 1-cell with pointing structure: the top arrow allows us to define Y based points $I(R(y)) \to c$ in C which the bottom arrow carries to $FIR(y) \to Fc$ and the 2-cell restores to a Y based point $y \to FIR(y) \to Fc$ in D. Now the structure we obtained by externalizing internal commutative comonoid structure can be generalized to commutative lax monoid structure in a 2-category of cells with pointing structure.

Definition 3.3.4 *The* lax pointing *2-category* PC *is constructed over the 2-category* C *by taking 1-cells $F : C \to C'$ of* C *for the 0-cells of* ArrC*, pairs of 1-cells $H : D \to C$ and $H' : C' \to D'$ together with a 2-cell $\psi_H : G \Rightarrow H' \circ F \circ H$ for the 1-cells of* ArrC *and pairs of compatible 2-cells α and α' for the 2-cells of* ArrC.

Commutative lax monoid structure in a 2-category of the form PC induces internal diagonal structure on its second projection.

Definition 3.3.5 Diagonal structure *is commutative lax monoid structure on a 0-cell in the 2-category* PCat.

Inside a category with commutative lax monoid structure, 'diagonal structure' refers to internal commutative comonoid structure that is coherent with respect to the monoid structure of the category.

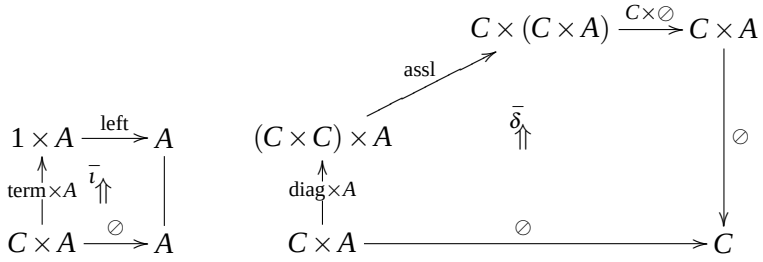
3.3.6 The Indexed Category Construction

Diagonal structure on $C \to [A,A]$ determines a functor from C to the category of comonoids in $[A,A]$, otherwise known as the category of comonads on A, and hence gives a sort of indexed comonad on A. The Kleisli construction takes a comonad T on A to a category on the objects of A with arrows from a to a' given by maps $Ta \to a'$ in A. The construction is contravariantly

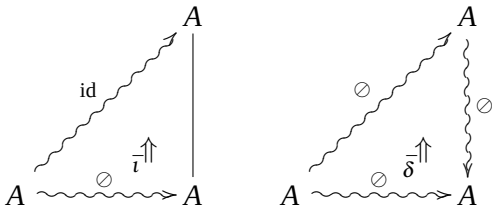
functorial and so composed with the functor from C gives an indexed category on C. Writing $c \oslash (\cdot)$ for the comonad over an object c of C, the arrows in the category over c are given by maps $c \oslash a \to a'$. Change of fibre over a map f from c' to c in C^{op} is the identity on objects and takes an arrow $g : c' \oslash a \to a'$ over c' to the arrow $g \circ (f \oslash a) : c \oslash a \to a'$ over c.

Diagonal structure on $[A,A]$ can be described independently of closed structure on the ambient 2-category.

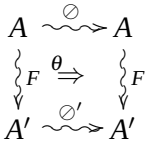
Writing $\oslash$ for the transpose $\overline{\Delta}$ of Δ, we transpose the unit and multiplication 1-cells in PCat that make Δ a lax monoid in PCat to obtain the diagrams below.



These correspond to the counit and comultiplication 2-cells for a comonad 1-cell $\oslash : A \rightsquigarrow A$ on the 0-cell A in the 2-category Cat_C of C indexed functors, which is the 2-category over C in the simple 2-fibration of Cat over itself, meaning that $\oslash$ is given by a functor $\oslash : C \times A \to A$.



If we would like to take $\Delta : C \to [A,A]$ and $\Delta' : C \to [A',A']$ for 0-cells in some 2-category, this correspondence suggests a natural notion of structure preserving 1-cell from Δ to Δ': any 1-cell $F : A \rightsquigarrow A'$ in Cat_C that is a comonad 1-cell with respect to the comonads $\oslash$ and $\oslash'$ corresponding to Δ and Δ'.

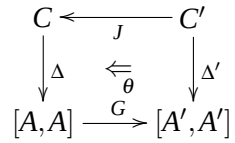


(Lax) comonad 1-cell in Cat_C means comonad 0-cell in $\mathrm{Arr}\mathsf{Cat}_C$. Blute, Cockett and Seely [6] refer to a comonad $\oslash : A \rightsquigarrow A$ as a 'structural action' and to a functor $F : A \to A'$ as 'strong structural' functor when

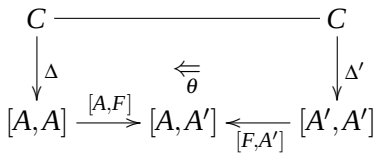
$$C \times A \xrightarrow{\mathrm{uniq} \times A} 1 \times A \xrightarrow{\mathrm{left}} A \xrightarrow{F} A'$$

gives a comonad 1-cell.

Note that the natural 1-cells from $\Delta : C \to [A,A]$ to $\Delta' : C' \to [A',A']$ taken as a lax monoid 0-cells in PCat are lax or oplax monoid 1-cells in PCat, which are lax monoid 0-cells in $\mathrm{Arr}\mathsf{PCat}$ or $\mathrm{OpArrOp}\mathsf{PCat}$,



but the transpose of the comonad square for F gives a pentagon.



There should be a natural 2-category containing both sorts of 1-cell, but we do not pursue this here.

Chapter 4

Structural Algebraic Compactness

The motivating examples of algebraically compact categories are not algebraically compact. Indeed Freyd warns in [15] that the notion of algebraic compactness 'should be understood in a 2-category setting'. To simplify the exposition, Freyd develops a theory of ordinary algebraic compactness, which requires a Freyd algebra for *every* endofunctor, even though domain theory constructs Freyd algebras only for endofunctors carrying certain extra structure. To admit the domain theoretic categories that motivate the theory, we are expected to replace the 2-category Cat of ordinary categories with a 2-category of categories carrying certain extra structure. The 2-categories that jump to mind derive from enriched and internal category theory, but we have found an interesting alternative.

Although our elementary presentation does not mention the ambient 2-category, we develop a theory of algebraic compactness in the 2-categories Str and Bis of structural and bistructural actions. Structural actions provide an account of maps parameterized by objects in the acting category C and, as it happens, the structural theory of algebraic compactness admits the motivating examples. With a category of predomains for C, our setting includes the enriched setting where the established theory is conducted [43, 39]. When C is the the unit category, we recover Freyd's attractive theory of ordinary algebraic compactness. For a broad perspective on the notion of structural action see Chapter 3 or, for an elementary account, see Appendix B.

Lemma 4.1.4 reveals the real strength of the structural setting. It allows us to work with a naïve definition of structurally algebraically compact category.

Definition 4.0.7 *A structural category D is* structurally algebraically compact *if it has a Freyd algebra for every structural endofunctor.*

In general, definitions that treat the existence of extra structure as a property are not good practice. The definition above ignores the possibility that an endofunctor may be structural in more than one way, so we appear to be losing information. We get away with this here because Lemma 4.1.4 produces the extra structure when it is needed. Otherwise, we would have to ask explicitly for parameterized invariants as in [10] (see Section 4.2.8).

Over ordinary categories, every delivery of Freyd algebras extends to a functor $\mathsf{Cat}(D,D) \to D$ which, using the closed structure of Cat, gives us a family of functors $\mu_B : \mathsf{Cat}(B \times D, D) \to \mathsf{Cat}(B,D)$ that interpret parameterized recursive types. In the structural setting, a delivery of ordinary Freyd algebras for the functors in $\mathsf{Str}(D,D)$ doesn't necessarily lift to a structural functor $\mathsf{Str}(D,D) \to D$ but does give us a family of functors $\mu_B : \mathsf{Str}(B \times D, D) \to \mathsf{Str}(B,D)$. So, despite our strengthened notion of functor, there is no need to strengthen the notion of Freyd algebra. In fact, the construction of μ only uses the final coalgebra property of Freyd algebras. This is a striking feature of the structural setting.

In the structural setting, however, one must take care with opposites. Over ordinary categories, the definition of free invariant Freyd uses in [14] is visibly self-dual: The algebra part of an invariant for F is a Freyd algebra if and only if the coalgebra part is a Freyd algebra for F^{op}. In the structural setting, we cannot, in general, construct opposite categories and functors. However, we can if the action on D has a right adjoint, in which case we say that the action is bistructural. For bistructural actions we obtain the dualities of algebraic compactness: A category D is compact if and only if the opposite category D^{op} is compact if and only if the doubled category $D^{\mathrm{op}} \times D$ is compact.

Our account of structural compactness makes no explicit use of enriched, internal or indexed category theory. However, these more sophisticated perspectives are important conceptually and technically. The indexed perspective abstracts from both the structural and enriched settings and guides our understanding of many constructions. Technically, enriched categories have dominated abstract domain theory ever since Wand introduced the use of O-categories [43]. Recent work by Fiore [10, 11, 12], in which compact categories are actually constructed, is motivated by questions of enrichment. So before we consider examples in Section 4.3, we explain how to set up a correspondence between the enriched and structural settings in Section 4.2.

Terminology. In this chapter, all use of the terms 'compact', 'complete' and 'cocomplete' is in the algebraic sense.

4.1 Algebraic Compactness in the Structural Setting

Our basic result, Lemma 4.1.4, actually applies to structural cocompleteness.

Definition 4.1.1 *A structural category D is* structurally cocomplete *if it has a final coalgebra for every structural endofunctor.*

We use the Lemma to show that structural cocompleteness implies structural parameterized cocompleteness.

Definition 4.1.2 *A structural category D is* structurally parameterized cocomplete *if it has a structural parameterized final coalgebra for every structural parameterized endofunctor. Given a structural parameterized endofunctor $F : B \times D \to D$, a* structural parameterized final coalgebra *is a structural functor $\phi : B \to D$ together with a transformation whose components $\pi_b : \phi b \to F b \phi b$ are final coalgebras.*

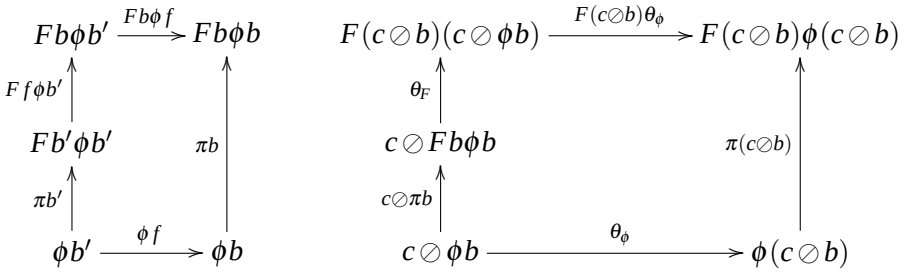
To handle compactness properly, we require opposites, but opposites are not a natural feature of the purely structural setting. Therefore, in Section 4.1.12, we strengthen the notion of structural action to that of bistructural action and check that the dualities of ordinary compactness lift to the (bi)structural setting. The whole thing works out very nicely.

4.1.3 The structurality of delivery functors.

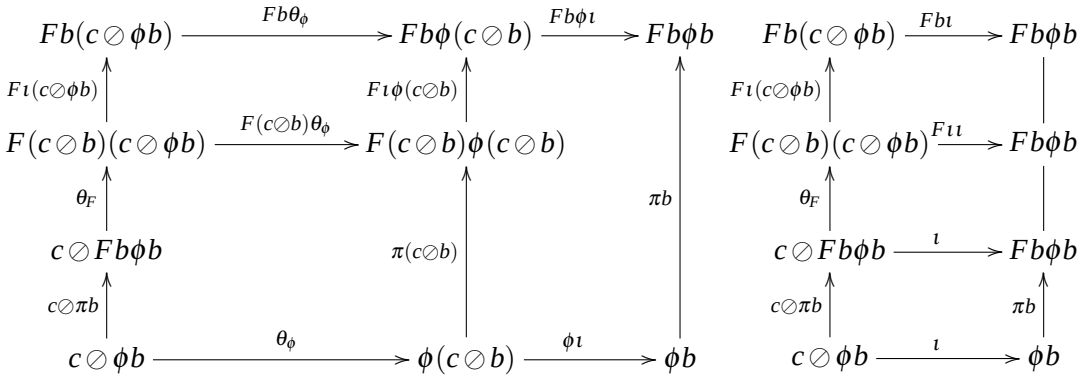
The universal properties of final coalgebras allow us to build both a delivery functor and a natural transformation that makes the delivery functor structural.

Lemma 4.1.4 *Given a structural functor $F : B \times D \to D$ and a final coalgebra $\pi b : \phi b \to F b \phi b$ for each endofunctor $F b : D \to D$, final coalgebra delivery lifts to a structural functor $\phi : B \to D$.*

Proof. Using the final coalgebra property, the functor part of the structural endofunctor F determines the functor part of ϕ while the transformation θ_F determines the transformation θ_ϕ.



The functorality of ϕ and the naturality and coherence of θ_ϕ follow from the uniqueness of induced maps. Take, for example, the coherence of θ_ϕ with respect to the ι. The components of the composite $\phi\iota\circ\theta_\phi$ and the components of ι give coalgebra morphisms which must both equal the unique morphism induced by the final coalgebra.



□

If we have a final coalgebra for every structural endofunctor on D, then we have one for every endofunctor of the form Fb, where F is a structural functor on $B\times D$. Therefore, in the structural setting, we get parameterized cocompleteness for free.

Corollary 4.1.5 *Every structurally cocomplete category is structurally parameterized cocomplete.*

Given a structurally cocomplete category D, we build a family of functors from $\mathsf{Str}(B\times D,D)$ to $\mathsf{Str}(B,D)$ indexed by B, the structural category of parameters.

Theorem 4.1.6 *For structurally cocomplete D, parameterized final coalgebra delivery gives a family of functors*

$$\mu_B : \mathsf{Str}(B\times D,D) \to \mathsf{Str}(B,D)$$

that is natural in B up to a canonical isomorphism.

Note that, although we do have a functor $Y : \mathsf{Str}(D,D) \to D$, we are not simply composing this with the transpose of F to obtain a functor $Y\circ\overline{F}$ because this does not tell us how to make the composite structural. The category $\mathsf{Str}(D,D)$ and the functors $\overline{F}$ and Y are not, in general, structural.

Using Lemma 4.1.4 and the same square root construction Freyd uses in [14], it can be shown that the product of structurally cocomplete categories is structurally cocomplete. If the product is structurally cocomplete, the factors are structurally cocomplete, and so the structurally cocompleteness of the product is equivalent to the structurally cocompleteness of the factors.

Theorem 4.1.7 *Structural categories A and B are structurally cocomplete if and only if their product is structurally cocomplete.*

Every Freyd algebra has an associated final coalgebra, so Lemma 4.1.4 specializes to Freyd algebra delivery.

Corollary 4.1.8 *Given a structural functor $F : B \times D \to D$ and a Freyd algebra $\sigma b : Fb\phi b \to \phi b$ for each endofunctor Fb, Freyd algebra delivery lifts to a structural functor $\phi : B \to D$.*

The results for cocompleteness specialize to results for compactness.

Theorem 4.1.9 *If D is structurally compact then it is structurally parameterized compact and parameterized Freyd algebra delivery gives a family of functors*

$$\mu_B : \mathsf{Str}(B \times D, D) \to \mathsf{Str}(B, D)$$

natural in B up to a canonical isomorphism.

Theorem 4.1.10 *Structural categories A and B are structurally compact if and only if their product is structurally compact.*

The dual of Lemma 4.1.4 applies to the parameterized initial algebra construction, but produces a costructural delivery functor from a costructural parameterized endofunctor.

Corollary 4.1.11 *Given a costructural functor $F : B \times D \to D$ and a initial algebra $\sigma b : Fb\omega b \to \omega b$ for each endofunctor $Fb : D \to D$, then initial algebra delivery lifts to a costructural functor $\omega : B \to D$.*

Given a functor $F : B \times D \to D$ and a Freyd algebra $\sigma b : Fb\phi b \to \phi b$ for each endofunctor $Fb : D \to D$, we are spoiled for choice: we can build a delivery functor using either the initial algebra construction or the final coalgebra construction. But the first lifts to the costructural setting while the second lifts to the structural setting, and the two settings are distinct. A structural action on D does not, in general, give a structural action on the opposite category D^{op}.

4.1.12 The structural compactness of opposites and doubles.

However, if the action part of a structural action on D has a right adjoint, then there is a corresponding structural action on D^{op}.

Definition 4.1.13 *An* adjunction of parameterized functors *between $\oslash : C \times A \to B$ and $\to : C^{\mathrm{op}} \times B \to A$ is given by a natural bijection $B(c \oslash a, b) \to A(a, c \to b)$. The* counit *$\varepsilon : c \oslash (c \to b) \to b$ at c and b corresponds to the identity on $c \to b$.*

For our purposes, $A = B = D$ and we work in terms of the action $\to^{\mathrm{op}} : C \times D^{\mathrm{op}} \to D^{\mathrm{op}}$. We therefore use the corresponding bijection

$$r : D(c \oslash d', d) \to D^{\mathrm{op}}(c \to^{\mathrm{op}} d, d').$$

An adjunction between actions then sets up a correspondence between structurality and costructurality.

Proposition 4.1.14 *Given adjoint actions $\oslash \dashv \to$, transformations ι and δ that make $\oslash$ structural correspond to transformations $r\iota$ and $r(\varepsilon \circ (c \oslash \varepsilon) \circ \delta)$ that make $\to^{\mathrm{op}}$ structural.*

This allows us to make the following definition.

Definition 4.1.15 *A* bistructural action *is a structural action together with a right adjoint costructural action with corresponding transformations.*

A similar correspondence exists at the level of functors.

Proposition 4.1.16 *Given bistructural actions on A and B each transformation θ that makes a functor $F : A \to B$ structural corresponds to a transformation $r(F\varepsilon \circ \theta)$ that makes the opposite functor F^{op} structural.*

Definition 4.1.17 *A* bistructural functor *is a functor between bistructural categories equipped with a transformation making it structural together with a corresponding transformation making it costructural.*

For bistructural D, structural cocompleteness, completeness and compactness are equivalent to costructural cocompleteness, completeness and compactness.

Corollary 4.1.18 *A bistructural category D is structurally (co)complete (compact) if and only if it is costructurally (co)complete (compact).*

This allows the following definition.

Definition 4.1.19 *A bistructural category is* bistructurally (co)complete (compact) *if it has a (final)initial (co)algebra (Freyd algebra) for every bistructural endofunctor.*

Bistructural categories have bistructural opposites, bistructural cocompleteness is dual to bistructural completeness and bistructural compactness is self-dual.

Corollary 4.1.20 *A bistructural category D is bistructurally cocomplete (compact) if and only if the corresponding D^{op} is bistructurally complete (compact).*

Here we must be careful. As we mentioned at the end of the previous section, there are two ways to construct the delivery functor for Freyd algebras. At the level of ordinary categories, they give the same functor, but here we must also check that the double delivery is bistructural.

Proposition 4.1.21 *Given a bistructurally compact category D, the structural final coalgebra and the costructural initial algebra constructions produce corresponding delivery functors.*

With this sanity check out of the way, we are free to use the doubling trick.

Corollary 4.1.22 *A bistructural category D is structurally compact if and only if $D^{\mathrm{op}} \times D$ is structurally compact.*

4.2 Compactness in Various Settings

The indexed setting provides a useful perspective on both the structural and the enriched settings. An indexed category associates a category $\mathbb{D}_c$ to each object c in C. Arrows $d' \to d$ in $\mathbb{D}_c$ correspond to arrows $c \oslash d' \to d$ in the structural setting and to arrows $c \to \mathcal{D}(d', d)$ in the enriched setting. We think of the indexed setting as neutral and view the structural and enriched settings as parameterized and curried, respectively. For the notion of indexed category see [18].

4.2.1 Indexed Compactness

The 2-category Ind of C-indexed categories replaces the individual categories and functors Cat with families of categories and functors indexed by the objects of C. Products and opposites lift componentwise from Cat. The notion of algebraic compactness also lifts. The indexed definition requires a Freyd algebra in each component of the indexed category. These must be preserved by the reindexing functors, which lift to algebras and coalgebras via their commutativity with the components of the indexed endofunctor.

Definition 4.2.2 *An indexed category $\mathbb{D}$ is* indexedly compact *if, for every indexed endofunctor $\mathbb{F} : \mathbb{D} \to \mathbb{D}$, each component of F has a Freyd algebra and reindexing preserves Freyd algebras.*

The delivery construction also lifts, so indexedly compact categories are indexedly parameterized compact and we obtain a family of functors

$$\mu_{\mathbb{B}} : \mathsf{Ind}(\mathbb{B} \times \mathbb{D}, \mathbb{D}) \to \mathsf{Ind}(\mathbb{B}, \mathbb{D}).$$

While Definition 4.2.2 is the natural notion of compactness in the indexed setting, a weaker, technical definition is sufficient to obtain structural compactness from enriched compactness.

Definition 4.2.3 *An indexed category $\mathbb{D}$ is* compact at *c if, for every indexed endofunctor $\mathbb{F} : \mathbb{D} \to \mathbb{D}$, the component $\mathbb{F}_c$ has a Freyd algebra.*

4.2.4 Structural Compactness and Indexed Compactness

As described in Chapter 3 (and in Appendix B), given a structural action on D we may construct an indexed category $\mathbb{D}$. Locally, the construction gives functors $i : \mathsf{Str}(B, D) \to \mathsf{Ind}(\mathbb{B}, \mathbb{D})$. Globally, we have a 2-functor $i : \mathsf{Str} \to \mathsf{Ind}$. We would like to find D in Str for which i reflects compactness. For our purposes, the existence of (what we will call) a unit will suffice.

Definition 4.2.5 *A structural action on D has unit u if the maps $\iota : u \oslash d \to d$ are isomorphisms.*

When D has unit u, D is isomorphic to $\mathbb{D}_u$ and the isomorphism matches each structural endofunctor F on D with the endofunctor $\mathbb{F}_u$ on $\mathbb{D}_u$.

Proposition 4.2.6 *If D has unit u and $\mathbb{D}$ is compact at u, then D is structurally compact.*

Corollary 4.2.7 *If D has a unit and $\mathbb{D}$ is indexedly compact, then D is structurally compact.*

Note that we are being somewhat glib. The universal properties of Freyd algebras and Corollary 4.1.8 ensure that the details take care of themselves.

The 2-functor i preserves products. so, locally, we have functors

$$i_B : \mathsf{Str}(B \times D, D) \to \mathsf{Ind}(\mathbb{B} \times \mathbb{D}, \mathbb{D})$$

and, when D has a unit and $\mathbb{D}$ is indexedly compact, the i commute with the μ (up to a canonical isomorphism).

$$\begin{array}{ccc} \mathsf{Str}(B \times D, D) & \xrightarrow{\mu_B} & \mathsf{Str}(B, D) \\ \downarrow{\scriptstyle i_B} & & \downarrow{\scriptstyle i} \\ \mathsf{Ind}(\mathbb{B} \times \mathbb{D}, \mathbb{D}) & \xrightarrow{\mu_{\mathbb{B}}} & \mathsf{Ind}(\mathbb{B}, \mathbb{D}) \end{array}$$

Similar results hold for bistructural compactness. Also, the restriction of i to bistructural actions preserves opposites and so, for indexed categories corresponding to bistructural categories, we can play any games we like with opposites and doubles.

4.2.8 Enriched Compactness and Indexed Compactness

In the enriched setting, we cannot base a theory of compactness on the sort of naïve definition we use in the structural setting. Take, for example, a naïve definition of enrichedly complete category.

> A $\mathcal{V}$-category $\mathcal{A}$ is *$\mathcal{V}$-algebraically complete* if for every $\mathcal{V}$-functor F, the underlying functor F_0 on $\mathcal{A}_0$ has an initial algebra. [10, Definition 6.1.4]

Because the delivery of initial algebras does not, in general, enrich, it is necessary to either strengthen the definition of initial algebra [34, Condition 2.3] or ask explicitly for an enrichment.

> Let $\mathcal{V}$ be cartesian. A $\mathcal{V}$-category $\mathcal{B}$ is *parameterised $\mathcal{V}$-algebraically complete* if it is $\mathcal{V}$-algebraically complete and for every $\mathcal{V}$-functor $F : \mathcal{A} \times \mathcal{B} \to \mathcal{B}$ and every indexed family $\{\iota_A^{F_0} : F(A, F_0^\dagger A) \to F_0^\dagger A\}_{A \in |A|}$ of initial $F_0(A,-)$-algebras, the induced functor $F_0^\dagger A : \mathcal{A}_0 \to \mathcal{B}_0$ $\mathcal{V}$-enriches. [10, Definition 6.1.7]

The problem revolves around the underlying category construction. It helps to observe that the underlying category is just one component of an indexed category [37, 34]. Briefly, to construct an indexed category $\mathbb{D}$ from a $\mathcal{V}$-enriched category $\mathcal{D}$, the maps $c \to \mathcal{D}(d', d)$ in $\mathcal{V}$ are collected to form the arrows $d' \to d$ of $\mathbb{D}_c$. We define composition and identities in $\mathbb{D}_c$ using composition and identities in D together with maps $c \to c \otimes c$ and $c \to I$.

$$\begin{array}{ccc} c \otimes c & \longleftarrow & c \\ \downarrow & & \vdots \\ D(d',d) \otimes D(d'',d') & \longrightarrow & D(d'',d) \end{array} \qquad \begin{array}{ccc} I & \longleftarrow & c \\ \downarrow & & \vdots \\ I & \longrightarrow & D(d,d) \end{array}$$

If these maps make c a comonoid, we obtain a category. Because the comonoid structure on c affects the behaviour of arrows in $\mathbb{D}_c$, we must distinguish between the categories produced with different structures. In general then, we obtain a category indexed over $\mathrm{Com}\mathcal{V}$, the category of comonoids in $\mathcal{V}$. When $\mathcal{V}$ has a good category of comonoids, the indexed category retains much of the information in the enriched category.

The monoidal category $\mathcal{V}$ always has at least one comonoid, the canonical comonoid on the unit I. The ordinary underlying category, rather unfortunately denoted $\mathcal{D}_0$, is the component $\mathbb{D}_I$ of the associated indexed category $\mathbb{D}$ and so the functor part of enriched endofunctors on $\mathcal{D}$ is the component at I of the associated indexed endofunctor. The naïve definition of enrichedly compact category thus amounts to compactness at I.

Definition 4.2.9 *An enriched category $\mathcal{D}$ is* enrichedly compact *if, for every enriched endofunctor $\mathcal{F}$, the underlying endofunctor $\mathcal{F}_0$ on $\mathcal{D}_0$ has a Freyd algebra.*

Proposition 4.2.10 *An enriched category $\mathcal{D}$ is enrichedly compact if the indexed category $\mathbb{D}$ is compact at I.*

When the monoidal structure on $\mathcal{V}$ gives products, the forgetful functor from $\mathrm{Com}\mathcal{V}$ to $\mathcal{V}$ has an inverse. So when $\mathcal{V}$ is cartesian each object carries a canonical comonoid structure which can be used in the above construction to obtain a $\mathcal{V}$-indexed category. Globally this gives a 2-functor $j : \mathsf{Enr} \to \mathsf{Ind}$ from the 2-category of $\mathcal{V}$-enriched categories to the 2-category of $\mathcal{V}$-indexed categories. Moreover, with $\mathcal{V}$ cartesian, every indexed functor from $\mathbb{E}$ to $\mathbb{D}$ is obtained from an enriched functor from $\mathcal{E}$ to $\mathcal{D}$.

Proposition 4.2.11 *If the enriching category $\mathcal{V}$ is cartesian and $\mathcal{D}$ is enrichedly compact then the $\mathcal{V}$-indexed category $\mathbb{D}$ is compact at* 1.

Remark. In the indexed category construction, the comonoid c is being used as a enriched *co*category. Given a C^{op}-enriched category $\mathcal{B}$ with the same objects as $\mathcal{D}$ (so $\mathcal{B}$ and $\mathcal{D}$ together form a $C^{\mathrm{op}} \times C$-enriched category), we take $\mathbb{D}_{\mathcal{B}}(d', d) = C(\mathcal{B}(d', d), \mathcal{D}(d', d))$ and define composition and identities using the cocategory structure on the hom objects of B.

$$\begin{array}{ccc} B(d',d) \otimes B(d'',d') & \longleftarrow & B(d'',d) \\ \downarrow & & \vdots \\ D(d',d) \otimes D(d'',d') & \longrightarrow & D(d'',d) \end{array} \qquad \begin{array}{ccc} I & \longleftarrow & B(d,d) \\ \downarrow & & \vdots \\ I & \longrightarrow & D(d,d) \end{array}$$

4.2.12 Structural Compactness Meets Enriched Compactness

The established notions of tensors and cotensors for enriched categories (see [7, Section 6.5]) provide a connection with structural and costructrual actions.

Proposition 4.2.13 *Given a $\mathcal{V}$ enriched category $\mathcal{D}$, $\mathcal{V}$ tensors give a structural action on the underlying category $D = \mathcal{D}_0$ and $\mathcal{V}$ cotensors give a costructural action.*

When $\mathcal{V}$ is cartesian, the structural action has unit 1 and gives the same indexed category $\mathbb{D}$ as the enriched category. So, when $\mathcal{V}$ is cartesian, we can apply Propositions 4.2.11 and 4.2.6.

Theorem 4.2.14 *If $\mathcal{V}$ is cartesian and $\mathcal{D}$ has $\mathcal{V}$ tensors and is enrichedly compact, then $D = \mathcal{D}_0$ is structurally compact.*

To return from the structural or costructural action to the enriched category, we might introduce a notion of $\mathcal{V}$ exponential.

Definition 4.2.15 $\mathcal{V}$ exponentials *for a structural (or costructural) action on D consist of a functor $\wedge : D \times D^{\mathrm{op}} \to \mathcal{V}$ together with a natural isomorphism $D(c \oslash e, d) \cong \mathcal{V}(c, d \wedge e)$ (or a natural isomorphism $D(e, c \to d) \cong \mathcal{V}(c, d \wedge e)$).*

Note however that in general $\mathcal{V}$ exponentials do not give anything like a $\mathcal{V}$-enriched category.

It has been observed that for $\mathcal{V}$-enriched categories with $\mathcal{V}$ cotensors, $\mathcal{V}$-algebraic completeness implies parameterised $\mathcal{V}$-algebraic completeness [10]. We understand this in terms of the corresponding costructural action, for which costructural completeness implies parameterized costructural completeness.

4.3 Structurally Algebraically Compact Categories of Domains

The notion of Freyd algebra was motivated by the properties of invariants constructed in domain theory. We therefore expect to find compact categories among the naturally occuring categories of domain theory. These categories are either given concretely as categories of partial orders or abstractly as (categories constructed from) categories satisfiying certain axioms. The concrete categories are then viewed as (categories constructed from) the abstract categories.

It is generally accepted that the language of abstract domain theory should be based on the monad structure carried by the lift functor. However, different perspectives on the monad structure have emerged. One account observes that, in the concrete examples, the category of monad algebras represents the category of strict maps. The lift functor is then viewed as the signature of a theory of domains (in a categorical theory of algebraic theories). Another account observes that the Kliesli category represents the category of partial maps which suggests that the lift functor is connected with the representation of partiality.

The notion of compactness does not clarify the role of the monad. In both the concrete and abstract cases, both the categories of strict and of partial maps are compact. Compactness provides a common axiomatics for the interpretation of recursive types in a variety of domain theoretic models.

In the abstract examples below, accessibility and local presentability are size conditions which ensure that the base category internalizes certain constructions from the ambient set theory (or topos). The canonical actions of the base category are described in Section 5.3 of Chapter 5.

4.3.1 Categories of Pointed Objects

Let CPO be the category of *predomains* and *continuous maps*. Objects are partial orders with colimits of ω chains. Arrows are maps that preserve colimits of ω chains. Let $\mathrm{CPPO}_\perp$, be the category of *domains* and *strict maps*. Objects are pointed predomains with all elements strictly greater than the point. Arrows are continuous maps that preserve the point. The motivating example of an algebraically compact category is $\mathrm{CPPO}_\perp$ [14].

Fact 4.3.2 *The category* $\mathrm{CPPO}_\perp$ *is not algebraically compact.*

The usual fix is based on the canonical CPO-enrichment of $\mathrm{CPPO}_\perp$: the category $\mathrm{CPPO}_\perp$ has Freyd algebras for all CPO-enriched endofunctors. Because CPO is cartesian and $\mathrm{CPPO}_\perp$ has CPO tensors and cotensors, the enrichment corresponds to a bistructural action of CPO on $\mathrm{CPPO}_\perp$ and structural, indexed and enriched compactness are all equivalent. Concretely, the structural action of c on d is the quotient of $c \times d$ that identifies pairs containing the least element of d and the corresponding costructural action is just the function space $c \to d$.

Example 4.3.3 *The category* $\mathrm{CPPO}_\perp$ *is bistructurally algebraically compact with respect to the bistructural action of* CPO.

The lift functor may be viewed abstractly as a monad with extra structure. Fiore *et al* use the following definition. A *domain-theoretic commutative monad* is a commutative strong monad T on a cartesian closed category C, with an initial object 0 and an inductive fixed point object. For the details see [12, Section 1]. A fixed point object is inductive if it is the colimit of the chain

$$0 \xrightarrow{b} T0 \xrightarrow{Tb} TT0 \xrightarrow{TTb} TTT0 \xrightarrow{TTTb} \cdots$$

The lift functor is a domain-theoretic commutative monad on CPO. The object of vertical natural numbers is the inductive fixed point object. The category of monad algebras for the lift monad is isomorphic to the category $\mathrm{CPPO}_\perp$ of domains and strict maps, so Example 4.3.3 may be viewed as an instance of the following.

Example 4.3.4 *The category* C^T *of monad algebras for a domain-theoretic commutative monad* T *on* C *is bistructurally algebraically compact with respect to the canonical action of* C *(assuming* C *is locally presentable and* T *is accessible).*

4.3.5 Categories of Partial Maps

Let pCPO be the category of *predomains* and *continuous partial maps*. Objects are partial orders with colimits of ω chains. Arrows are partial maps, defined on sets that contain colimits of ω chains, that preserve colimits of ω chains. The category pCPO carries a structural action of CPO and is structurally isomorphic to the category $\mathrm{CPPO}_\perp$. The action of c on d is simply the product $c \times d$ and the corresponding costructrual action is simply the predomain of functions from $c \to d$.

Example 4.3.6 *The category* pCPO *is bistructurally algebraically compact with respect to the bistructural action of* CPO.

Again the lift functor may be viewed abstractly, this time as a means of representing partiality. Pull-backs are used to set up a correspondence between certain partial maps and maps in the Kliesli category. Fiore *et al* use the following definition. A *lifting monad* is a commutative strong monad on a category C with a terminal object, such that the unit η is cartesian, C has all pull-backs of η_1 and η classifies partial maps with domains given by pull-backs of η_1 [12]. See [12, Appendix A] for details. The lift functor is a lifting monad on CPO and the Kliesli category for this monad is isomorphic to pCPO, so Example 4.3.6 may be viewed as an instance of the following.

Example 4.3.7 *The Kliesli category* C_T *for a domain-theoretic lifting monad* T *on* C *is bistructurally algebraically compact with respect to the canonical action of* C *(assuming* C *is locally presentable and* T *is accessible).*

Chapter 5

Structural Adjunctions

Algebraically compact categories do not exist in a vacuum. They typically participate in an adjunction and enrich in a category with fixed points. The precise status of these structures is still the subject of active research [11, 12]. Here we show how known categories of domains may be presented in terms of the structural setting. This allows us to apply the theory of structural compactness developed in Chapter 4. The structural setting, which allows us to present the structure of categories of domains with no mention of enrichment, reduces the technical overhead associated with the use of algebraic compactness.

In Section 5.1, we examine monoidal structures from the perspective of the structural setting. To apply our theory of structural compactness, it is enough to show that monoidal adjunctions may always be viewed as structural adjunctions. Additional structure—closed, symmetric, cartesian—however, can also be viewed in terms of structural actions. Symmetric structure, for example, makes the monoidal structure structural. This allows the monoidal structure to participate in endofunctors with invariants given by structural compactness.

Kleisli categories of domains, which represent categories of partial maps, have costructural actions (known as Kleisli exponentials) without having the closed structure that would give us the costructural action in a monoidal adjunction. Therefore, in Section 5.2, we give definitions that describe categories of domains directly in terms of structural actions.

Eilenberg-Moore categories of domains, which represent categories of pointed objects, then provide our standard examples of monoidal adjunctions with all the trimmings, while Kleisli categories of domains provide examples of (more) purely structural adjunctions.

5.1 Monoidal Adjunctions as Structural Adjunctions

A *monoidal adjunction* is an adjunction given by monoidal functors and monoidal transformations between monoidal categories. Likewise, a *structural adjunction* is given by structural functors and structural transformations between categories with structural actions (of some fixed category). If we ignore the functors and transformations that make the adjoints monoidal or structural we obtain the *underlying adjunction*.

Theorem 5.1.1 *For every monoidal adjunction there is a canonical structural adjunction with the same underlying adjunction and acting category* Com$C$.

Proof. Given a monoidal adjunction $L \dashv U : D \to C$, a general theorem of 2-algebraic structure [19] (established by Day in the special case of monoidal structure) constructs inverses to the transformations that make the left adjoint L monoidal. For example, the transpose of the transformation $I_C \to U(I_D)$ gives an inverse to the transformation $I_D \to L(I_C)$. Using this opmonoidal structure, L lifts to comonoids (by Proposition 3.1.12) and so precomposition with

L carries the canonical structural action of Com$D$ on D to a structural action of Com$C$ on D given by $c \oslash d = Lc \otimes d$, where here c is a comonoid in C. In more detail, the transformation $\psi_L : c \oslash Lc' = Lc \otimes_D Lc' \to L(c \otimes_C c')$ that makes L monoidal also makes it structural, and the transformation ψ_U that makes U monoidal gives us a transformation $\psi_U \circ (\eta \otimes_C Ud) : c \otimes_C Ud \to U(Lc \otimes_D d) = U(c \oslash d)$ that makes U structural. It can be checked that the unit and counit are then structural tranformations. □

We view monoidal adjunctions as structural adjunctions so that we can ask for structural compactness.

Definition 5.1.2 *A* cocomplete (compact) monoidal adjunction *is a monoidal adjunction in which D is structurally cocomplete (compact) with respect to the canonical structural action of* Com$C$.

The notion of compact monoidal adjunction captures enough of the structure of categories of domains to proceed directly to a discussion of canonical fixed points (see Section 6.3.1), but our full theory of canonical fixed points makes use of additional structure.

5.1.3 Closed Structure

Given a monoidal closed adjunction, we could construct a bistructural adjunction, but parts of a monoidal closed adjunction can be dropped and we are left with structures that still look sensible from the structural perspective.

Monoidal structure is *closed* if its multiplication, viewed as a parameterized endofunctor, has a right adjoint (Definition 4.1.13).

Definition 5.1.4 *A* semiclosed monoidal adjunction *is a monoidal adjunction in which the monoidal structure on D is closed.*

With D closed monoidal, the canonical structural action on D has a right adjoint and the corresponding costructural action on D is given by $c \to d = Lc \multimap d$. This makes D bistructural.

Definition 5.1.5 *A* (co)complete (compact) semiclosed monoidal adjunction *is a semiclosed monoidal adjunction in which D is bistructurally (co)complete (compact).*

Closed structure can also be described independently of monoidal structure. A closed category D has an object I and an operation $\multimap\colon D^{\mathrm{op}} \times D \to D$ together with natural transformations satisfying coherence laws.

Definition 5.1.6 *A* semimonoidal closed adjunction *is a closed adjunction $L \dashv U : D \to C$ in which the closed structure on C is adjoint to monoidal structure.*

We ask for monoidal structure on C so that the closed adjunction can be viewed as a costructural adjunction with acting category Com$C$.

Definition 5.1.7 *A* complete (compact) semimonoidal closed adjunction *is a semiclosed monoidal adjunction in which D is costructurally complete (compact).*

5.1.8 Cartesian Structure

Monoidal structure is *cartesian* if its multiplication gives products and its unit is final.

Definition 5.1.9 *A* monoidal/cartesian adjunction *is a monoidal adjunction in which the monoidal structure on C is cartesian.*

The properties of products induce a symmetry on the multiplication and commutative comonoid structure on objects. Given a symmetric monoidal category C, we write CCom$C$ for the full subcategory of Com$C$ spanned by the commutative comonoids (the symmetry is used to express the commutativity of a comonoid).

Proposition 5.1.10 (after Fox [13]) *Cartesian monoidal structure carries a unique symmetry and the forgetful functor* forget : CCom$C \to C$ *has a unique section* com : $C \to$ CCom$C$ *(meaning* forget $\circ$ com $= \mathrm{Id}_C$*).*

The inclusion forget : CCom$C \to$ Com$C$ restricts the cononical structural action of Com$C$ on C to a structural action of CCom$C$ and, when the monoidal structure is cartesian, precomposition with com : $C \to$ CCom$C$ carries this to a structural action of C, but this action is just the original multiplication on C, so the multiplication of a cartesian monoidal category C is a structural action of C on itself. Similarly, the canonical action of Com$C$ on D is carried to an action of C on D given by $c \oslash d = Lc \otimes d$, where c is now an object of C. By Theorem 5.1.1

Corollary 5.1.11 *For every monoidal/cartesian adjunction there is a canonical structural adjunction with the same underlying adjunction and acting category C.*

Definition 5.1.12 *A* cocomplete (compact) monoidal/cartesain adjunction *is a monoidal/cartesian adjunction in which D is structurally cocomplete (compact) with respect to the canonical structural action of C.*

5.1.13 Symmetric Structure

In a monoidal adjunction, the adjunction is structural. In a symmetric monoidal adjunction, everything in sight is structural, the adjunction, the multiplications, the actions and all the natural transformations, and in a symmetric closed monoidal adjunction, everything in sight is bistructural.

In more detail, suppose the monoidal structure on C is symmetric, as in a monoidal/cartesian adjunction. We can use the symmetry $\gamma : c \otimes b \to b \otimes c$ to produce a transformation θ_b, with components $c \otimes (b \otimes a) \to b \otimes (c \otimes a)$. This transformation satisfies the laws for the structurality of the action of b. Note that these laws involve θ_b and the transformations that make the action of c structural and are independent of any transformations that may make the action of b a comonad.

When C is symmetric closed monoidal, the opposite of the function space gives a structural action on C^{op} and the symmetry, which makes the action of b on C a structural endofunctor, also makes the corresponding action on C^{op} a structural endofunctor. It can also be checked that the adjunction between these endofunctors has structural unit and counit.

Definition 5.1.14 *A* symmetric monoidal/cartesian closed adjunction *is a semiclosed monoidal adjunction in which D is symmetric and C is cartesian closed.*

Remark. Given a symmetry for the structure on D, semiclosed monoidal adjunctions have duals. Choose an object r in D, and observe that, via the symmetry, the functor $\neg : D^{\mathrm{op}} \to D$ given by $\neg d = d \multimap r$ is right adjoint to its own opposite $\neg^{\mathrm{op}}$. The self adjunction is monoidal and, composed with the adjunction $L \dashv U$, produces a second semiclosed monoidal adjunction $(\neg^{\mathrm{op}} \circ L) \dashv (U \circ \neg) : D^{\mathrm{op}} \to C$.

5.2 Structural Adjunctions Proper

Since we view a monoidal adjunction, as a structural adjunction, we could simply ask for a compact structural adjunction and proceed directly to a discussion of canonical fixed points (see Section 6.3.1).

Definition 5.2.1 *A* compact structural adjunction *is a structural adjunction $L \dashv U : D \to C$ with D structurally compact.*

However, our theory of fixed points makes use of additional structure and, as we saw in Section 5.1, the additional structure can be described directly in terms of structural actions.

5.2.2 Structurality and Balance

As we observed in Section 5.1.13, when a structural adjunction is given by a monoidal adjunction the structurality of the actions, taken as functors, can be derived from a symmetry. If we do not ask for monoidal structure, we must ask for the structurality of the actions directly. For example, our account of parameterized fixed points requires the structurality of the action of C on itself.

In the case of a self-action, structurality is like a cross between symmetry and associativity. If we ask for structurality at one point b of a self-action $\boxslash$, we are asking for a transformation θ_b with components $c \boxslash (b \boxslash a) \to b \boxslash (c \boxslash a)$, which we think of as a form of (lax) symmetry on C. If we ask for structurality of the whole action, we are asking for a transformation $\theta_{\boxslash}$ with components $c \boxslash (b \boxslash a) \to (c \boxslash b) \boxslash (c \boxslash a)$. Given $\theta_{\boxslash}$ we obtain a θ_b for each b by composition with $\lambda \boxslash (c \boxslash a)$. On the other hand, by composition with $(c \boxslash b) \boxslash \lambda$ we obtain an α_b with components $\alpha_b : c \boxslash (b \boxslash a) \to (c \boxslash b) \boxslash a$, which is a lax associativity.

To avoid the expression 'structural structural action', we use the word 'balance'.

Definition 5.2.3 *A structural action or structural functor is* balanced *if the functors and transformations that make it structural are themselves structural.*

As it stands, this definition is a bit slippery. For our purposes, we will assume that structural actions on products are given by products of structural actions and that the acting category C carries a fixed self-action $\boxslash$. A balanced structural action $\oslash : C \times D \to D$ therefore includes a natural transformation $\theta_{\oslash}$ with components $c \oslash (b \oslash d) \to (c \boxslash b) \oslash (c \oslash d)$.

A *balanced adjunction* is given by balanced functors and natural transformations between categories with balanced actions. For example, symmetric monoidal adjunctions give balanced adjunctions, balanced monoidal adjunctions in fact. For parameterized fixed points, however it is really the balanced *self*-action of C that is most useful. We therefore make the following definition.

Definition 5.2.4 *A* suitable structural adjunction *is a structural adjunction $L \dashv U : D \to C$ where C is the acting category and the action on C is balanced.*

For example, a monoidal/cartesian adjunction gives a suitable structural adjunction. This is enough for an account of parameterized fixed points, but to get the most out of the structural compactness of D, we can ask for everything to be balanced.

Definition 5.2.5 *A* suitable balanced adjunction *is a balanced adjunction $L \dashv U : D \to C$ where C is the acting category (and the balance is with respect to the action on C).*

For example, a symmetric monoidal/cartesian adjunction gives a suitable balanced adjunction.

5.2.6 Exponential Structure

Suppose we view a monoidal/cartesian adjunction as a suitable structural adjunction with the action of C on itself given by the product and the action on D given by $c \oslash d = Lc \otimes d$. With D monoidal, we have $d \cong d \otimes I$ and so $Lc \cong Lc \otimes I = c \oslash I$. The unit I allows us to express L in terms of $\oslash$.

If we intend to derive L from an object I, we might just ask for a structural functor U into the acting category.

Definition 5.2.7 *A* suitable structural functor *is a structural functor $U : D \to C$ where C is the acting category.*

Given a suitable structural functor U, we could ask for an object I such that L, given by $Lc = c \oslash I$, is left adjoint to U. Now suppose the action on D is bistructural, with adjoint action $\to : C^{\mathrm{op}} \times D \to D$, and consider the functor $P : C \to D^{\mathrm{op}}$ given by $Pc = c \to^{\mathrm{op}} I$. When U arises

from a semiclosed monoidal adjunction, P is (monoidal) left adjoint to $R : D^{\mathrm{op}} \to C$ given by $Rd = U(d \multimap I)$. Considering that Ud may be isomorphically expressed as $U(I \multimap d)$, we generalize from $U : D \to C$ to an exponential $- : D \times D^{\mathrm{op}} \to C$.

Definition 5.2.8 *A* balanced exponential *is a balanced bistructural category D with a balanced action on the acting category and with an exponential given by a structural functor $- : D \times D^{\mathrm{op}} \to C$ and structural units and counits.*

A suitable exponential is determined by a balanced structural action

- $\boxtimes : C \times C \to C$, $\delta_{\boxtimes} : c \boxtimes a \to c \boxtimes (c \boxtimes a)$, $\iota_{\boxtimes} : c \boxtimes a \to a$
- $\theta_{\boxtimes} : c \boxtimes (b \boxtimes d) \to (c \boxtimes b) \boxtimes (c \boxtimes d)$

and a balanced structural action

- $\oslash : C \times D \to D$, $\delta_{\oslash} : c \oslash d \to c \oslash (c \oslash d)$, $\iota_{\oslash} : c \oslash d \to d$
- $\theta_{\oslash} : c \oslash (b \oslash d) \to (c \boxtimes b) \oslash (c \oslash d)$

that has a right adjoint as a functor parameterized by C,

- $\to : C^{\mathrm{op}} \times D \to D$, $\varepsilon_{\to} : c \oslash (c \to d) \to d$, $\eta_{\to} : d \to (c \to (c \oslash d))$

and a structural right adjoint as a functor parameterized by D.

- $- : D \times D^{\mathrm{op}} \to C$, $\varepsilon : (d - e) \oslash e \to d$, $\eta : c \to ((c \oslash d) - d)$
- $\theta_{-} : c \boxtimes (d - e) \to ((c \oslash d) - (c \oslash e))$

Any category D has a balanced exponential with acting category $C = 1$. With respect to the canonical actions, any semiclosed monoidal/cartesian adjunction $L \dashv U$ has an exponential given by $d - e = U(e \multimap d)$. If the adjunction is symmetric, then the exponential is balanced.

An exponential includes isomorphisms

$$D(c \oslash e, d) \cong C(c, d - e) \cong D(e, c \to d)$$

from which we may obtain adjunctions both from D to C and from D^{op} to C. When the exponential is balanced, we obtain suitable balanced adjunctions.

Proposition 5.2.9 *Given a balanced exponential on D and an object e in D, the functors defined by $Lc = c \oslash e$ and $Ud = d - e$ give a suitable balanced adjunction.*

The adjoint functors in a semiclosed monoidal/cartesian adjunction are recovered from the canonical exponential by taking $e = I$.

Exponentials on bistructural categories have opposites. If $(C, D, \oslash, \to, -)$ is a balanced exponential, then so is $(C, D^{\mathrm{op}}, \to^{\mathrm{op}}, \oslash^{\mathrm{op}}, -^{\sigma})$, where $d -^{\sigma} e = e - d$. If Proposition 5.2.9 is applied to the opposite of a balanced exponential, we obtain a suitable balanced adjunction between D^{op} and C. If the exponential is built from a semiclosed monoidal/cartesian model and we take $e = r$ we obtain the dual adjunction mentioned at the end of Section 5.1.13.

A *cocomplete suitable exponential* has $D^{\mathrm{op}} \times D$ structurally cocomplete. The opposite of a cocomplete suitable exponential is cocomplete. Likewise for complete suitable exponentials. A *compact suitable exponential* has $D^{\mathrm{op}} \times D$ or, equivalently, D structurally compact. Compact suitable exponentials have compact opposites.

5.3 Structural Adjunctions on Categories of Domains

The examples of compact categories given in Section 4.3 are characterized by adjunctions with the base category. For categories of pointed objects the adjunction is monoidal closed and hence bistructural. For categories of partial maps it is a proper bistructural adjunction.

5.3.1 Categories of Eilenberg-Moore Algebras

Given a monad T on a category C, the category C^T of Eilenberg-Moore algebras for T is characterized by an adjunction with C. When C is cartesian closed and T has a commutative strength, C^T is symmetric monoidal closed and so is the adjunction [20].

The abstract category of pointed objects C^T in Example 4.3.4, therefore participates in a monoidal/cartesian adjunction. From this we obtain a canonical structural action of C on C^T. Due to the cartesian structure on C, enriched endofunctors on D correspond to structural endofunctors and so enriched compactness for D is equivalent to structural compactness.

Example 5.3.2 *The Eilenberg-Moore adjunction constructed from a domain-theoretic commutative monad is a compact symmetric monoidal/cartesian closed adjunction.*

The categories $D = \mathrm{CPPO}_\perp$ and $C = \mathrm{CPO}$ carry the standard concrete example of such an adjunction. The left adjoint is the lift functor which adds a least element and the right adjoint is the forgetful functor which includes domains among predomains. It can be checked that $\mathrm{CPPO}_\perp$ is isomorphic to CPO^T, where $T = \mathrm{forget} \circ \mathrm{lift}$.

Example 5.3.3 *The adjunction* $\mathrm{lift} \dashv \mathrm{forget} : \mathrm{CPPO}_\perp \to \mathrm{CPO}$ *is a compact symmetric monoidal/cartesian closed adjunction.*

5.3.4 Categories of Kleisli Maps

Given a monad T on a category C, the category C_T of Kleisli maps for T is also characterized by an adjunction with C. A *structural monad* is given by a structural endofunctor and structural transformations.

Theorem 5.3.5 *Given a structural monad $T : A \to A$, the Kleisli adjunction is structural with the canonical action of c on a map given by $f : a' \to Ta$ in the Kleisli category C_T given by $\theta_a \circ f : c \oslash a' \to T(c \oslash a)$.*

Proof. Because a structural action is an indexed comonad, a structural monad is a monad with an indexed distributive law over the indexed comonad. Just as a distributive law allows one to lift a Kleisli adjunction to a comonad adjunction (see [35]), the indexed distributive law allows us to lift the Kleisli adjunction to an indexed comonad adjunction, which is a structural adjunction. □

The notion of a strong monad on a cartesian category coincides with the notion of a structural monad with respect to the canonical structural action given by the cartesian structure.

Corollary 5.3.6 *Given a cartesian category C with a strong monad T, the Kleisli adjunction is a suitable structural adjunction with the canonical actions of C.*

The abstract category of partial maps C_T in Example 4.3.7 therefore participates in a suitable structural adjunction. Again, due to the cartesian structure on C, the structural algebraic compactness of the Kleisli category follows from its enriched compactness.

Example 5.3.7 *The Kleisli adjunction constructed from a domain-theoretic commutative lifting monad is a compact suitable structural adjunction.*

The categories $D = \mathrm{pCPO}$ and $C = \mathrm{CPO}$ carry the standard concrete example of such an adjunction. The left adjoint is the inclusion of total maps among partial maps and the right adjoint is the lift functor that represents partial maps using the extra element. It can be checked that pCPO is isomorphic to CPO_T, where $T = \mathrm{lift} \circ \mathrm{incl}$.

Example 5.3.8 *The adjunction* $\mathrm{incl} \dashv \mathrm{lift} : \mathrm{pCPO} \to \mathrm{CPO}$ *is a compact suitable structural adjunction.*

Chapter 6

Canonical Fixed Points

It is observed in [14] that certain Freyd algebras correspond to fixed point objects in a related category. The idea of a fixed point object is to generate fixed points for some class of endomaps via some generic endomap equipped with a fixed point. By generic, we mean equipped with a canonical endomap morphism into any endomap of the given class. When the fixed point and canonical endomap morphisms are unique in some sense, and the generic endomap belongs to the given class, the family of fixed points that it generates is characterized by a uniformity property. This is what happens in categories of domains and in categories of complete partial orders the characterization is known as Plotkin's Axiom.

In our setting the fixed point object is given by transposition.

Theorem 6.0.9 *Given an adjunction $L \dashv U : D \to C$, Freyd algebras for LU transpose to fixed point objects in C (with respect to the monad UL).*

This gives us a generic endomap $\mathrm{suc} : U\omega \to U\omega$ that induces ordinary fixed points. For c-parameterized fixed points, we transpose a Freyd algebra $L(c \times U\psi_c) \to \psi_c$ to a c-parameterized endomap $\mathrm{suc}_c : c \times U\psi_c \to U\psi_c$.

In the tradition of abstract nonsense, we view suc_c as an algebra for the endofunctor S taking a to $c \times a$ and introduce a notion of fixed point algebra, an artifical device that subsumes Freyd algebras and Mulry's notion of fixed point object. This allows us to be particularly glib in the statement of our general transposition result. Given an adjunction $L \dashv U : D \to C$ and an endofunctor $S : C \to C$, fixed point algebras for LSU transpose to fixed point algebras for S.

The notion of fixed point algebra, together with the transposition result, falls neatly into two halves. The clean half is the based on the notion of corecusive algebra. Given a natural transformation $\eta : \mathrm{Id}_C \Rightarrow UL$ and an endofunctor $S : C \to C$, corecursive algebras for LSU transpose to corecursive algebras for S. Note that this does not use the laws of the adjunction, just one of the natural transformations.

The dirty half depends on notions of 'pointed' object and 'strict' map and the full adjunction laws. In an effort to tease apart the exact conditions used, we work with abstract classes of pointed objects and strict maps. In applications we adopt definitions that abstract from adjunctions between categories of partial orders where these terms originated.

6.1 Corecursive Algebras and Unique Fixed Points

Algebras for the identity endofunctor are endomaps. A corecursive algebra for the identity endofunctor is an endomap with a strong form of unique fixed point.

Definition 6.1.1 *An endomap has a* very unique fixed point *if it is a corecursive algebra for the identity endofunctor.*

If by point we mean a map from some fixed object b, an endomap s with a very unique fixed point has a unique fixed point z_b.

$$\begin{array}{ccc} b & \xrightarrow{z_b} & \phi \\ \| & & \downarrow{\scriptstyle \mathsf{s}} \\ b & \xrightarrow{z_b} & \phi \end{array}$$

Typically b is a terminal object or a monoidal unit, but because z_b exists uniquely not just for some b but for *all* b, our endomap has a unique fixed point in a generalised sense.

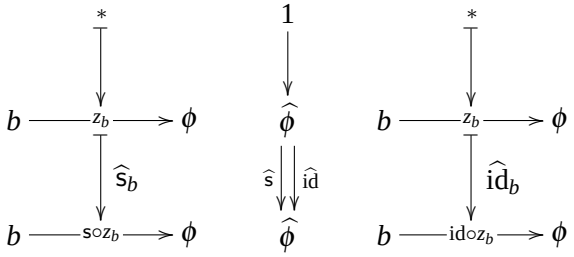
In Section 6.1.2, we examine a natural generalization of the notion of unique fixed point, but we also show that the notion of very unique fixed point is strictly stronger. In Sections 6.1.7 and 6.1.10, we derive definitions for unique parameterized fixed points by interpreting Definition 6.1.1 in the Kleisli categories of comonads.

6.1.2 Very Unique Fixed Points and Unique Fixed Generalized Points

A *b-point in* ϕ is just any map from b to ϕ and a *fixed b-point for* $\mathsf{s} : \phi \to \phi$ is a b-point z such that $\mathsf{s} \circ z = z$.

Definition 6.1.3 *An endomap* s has a unique fixed point *if it has a unique fixed b-point* z_b *for each b.*

In Set, s has a unique fixed point when there is a unique element $z \in \phi$ such that $\mathsf{s}(z) = z$. In a general category C, the maps z_b behave as maps sending everything to one particular point, a sort of virtual element z. In terms of the Yoneda representation of C, which represents a map f as the transformation $\widehat{f}$ taking z to $f \circ z$ in the category $[C^{\mathrm{op}}, \mathrm{Set}]$ of presheaves on C, b-points in ϕ are elements of $\widehat{\phi}$ at b and s has a unique fixed point when the transformation $\widehat{\mathsf{s}}$ fixes exactly one element of $\widehat{\phi}$ at each b, which is to say the equalizer of $\widehat{\mathrm{id}}$ and $\widehat{\mathsf{s}}$ is isomorphic to the constant presheaf 1.



More generally, a diagram d in C *has a unique fixed point* if $\lim \widehat{\mathrm{d}} \cong 1$ in $[C^{\mathrm{op}}, \mathrm{Set}]$. When the diagram consists of an endomap $\mathsf{s} : \phi \to \phi$ in parallel with the identity on ϕ we recover Definition 6.1.3. If C has a generating object, such a one element set in Set, our general notion of unique fixed point simplifies. If b is a generating object, then

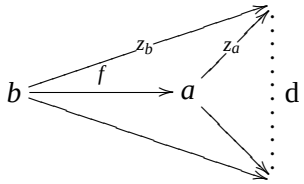
d has a unique fixed point *iff* there is a unique cone on d with vertex b.

Another useful characterization involves the category $\mathrm{Cone}(\mathrm{d})$ of cones and cone morphisms over the diagram d.

Lemma 6.1.4 *The following are equivalent:*

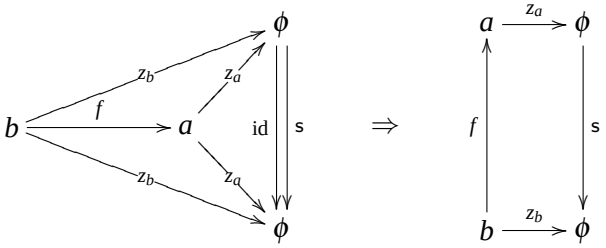
1. d *has a unique fixed point.*
2. *The vertex projection from* $\mathrm{Cone}(\mathrm{d})$ *to C is an isomorphism.*

When d has a unique fixed point, the inverse of the vertex projection is $b \mapsto z_b$ and every map $f : b \to a$ is a morphism from the cone z_b to the cone z_a



or, from the point of view of the cones, precomposition with an arbitrary map f from b to a carries the cone z_a to the cone z_b. When f is an endomap, precomposition with f has no effect at all.

Corollary 6.1.5 *If an endomap* s *has a unique fixed point, then for any* $f : b \to a$ *we have* $z_b = \mathsf{s} \circ z_a \circ f$.



In the case of a corecursive endomap, this means the maps z_p depend solely on the carrier of p. It does not mean an endomap with a unique fixed point is a corecursive endomap. The notion of corecursive endomap is stronger because $\mathsf{s} \circ z \circ p = z$ may hold for z other than z_b.

Counterexample 6.1.6 If $\mathsf{s} : 3 \to 3$ fixes one element and swaps the other two, then $\mathsf{s} \circ \mathsf{s} = \mathrm{id}$. This gives $\mathsf{s} \circ \mathrm{id} \circ \mathsf{s} = \mathrm{id}$ and $\mathsf{s} \circ \mathsf{s} \circ \mathsf{s} = \mathsf{s}$ and so, if s is taken for p, both id and s can be taken for z_p, but $\mathsf{s} \neq \mathrm{id}$.

More generally, if s is a corecursive endomap and there exists s^- such that $\mathsf{s} \circ \mathsf{s}^- = \mathrm{id}$, then s is an identity and hence the identity on a terminal object. In particular, if s is a corecursive isomorphism, then it is the identity on a terminal object.

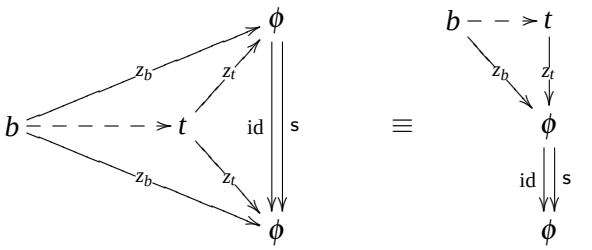
Limits. The relationships between limits, terminal objects and unique fixed points can get a little confusing. The Yoneda representation preserves limits, so if d has limit t and t is terminal, then $\widehat{\mathsf{d}}$ has limit $\widehat{t}$ and $\widehat{t}$ is terminal which means d has a unique fixed point. Converses follow from Lemma 6.1.4. When d has a unique fixed point,

$$t \text{ is terminal } \textit{iff } z_t \text{ is a limit cone}$$

or, looked at another way, when C has a terminal object t,

$$\mathsf{d} \text{ has a unique fixed point } \textit{iff } \mathsf{d} \text{ has limit } t \text{ in } C.$$

In particular, in the presence of an endomap s with a unique fixed point, an object t is terminal *iff* z_t equalizes id_ϕ and s or, looked at the second way, in the presence of a terminal object t, an endomap s has a unique fixed point *iff* t equalizes id_ϕ and s.



6.1.7 Comonoids and Parameterized Unique Fixed Points

Our treatment of parameterized maps is based on comonad structure. The comonads we have in mind are given by multiplication, multiplication by any object in C if C has products or multiplication by a comonoid object if C is monoidal. By Proposition 5.1.10, the first is a special case of the second, so suppose C carries monoidal structure and c carries comonoid structure given by the maps

$$\delta : c \to c \otimes c \quad \text{and} \quad \kappa : c \to I .$$

If $\otimes$ actually gives products, δ and κ will be the diagonal and terminator for c. The functor $c \otimes (\cdot)$ carries comonad structure with comultiplication and counit given by the composite natural transformations

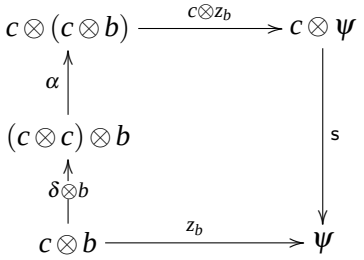
$$\alpha \circ (\delta \otimes (\cdot)) : c \otimes (\cdot) \Rightarrow c \otimes (c \otimes (\cdot)) \quad \text{and} \quad \lambda \circ (\kappa \otimes (\cdot)) : c \otimes (\cdot) \Rightarrow (\cdot),$$

where α and λ are the associativity and left identity natural transformations for the monoidal structure on C. We think of the Kleisli category C_c for this comonad as a category of parameterized maps: arrows in C_c are given by maps $c \otimes a \to a'$ in C.

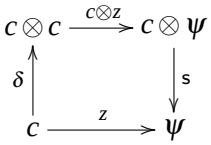
For a definition of parameterized very unique fixed point we interpret our definition of very unique fixed point in such a Kleisli category. When c is a monoidal unit, $c \otimes (\cdot)$ is isomorphic to the identity, C_c is isomorphic to C and the following reduces to Definition 6.1.1.

Definition 6.1.8 *A c-parameterized endomap $\mathsf{s} : c \otimes \psi \to \psi$ has a c-*parameterized very unique fixed point *if the endomap it gives in C_c is corecursive algebra.*

Taken as an endomap in C_c, a c-parameterized map with a very unique fixed point has a unique fixed point in the sense of Section 6.1. This means for each object b, there is a unique fixed b-point in the category C_c for the endomap given by s, which is to say, a unique c-parameterized map $z_b : c \otimes b \to a$ such that $f \circ (c \otimes z_b) \circ \nu_b = x$, using the definition of composition in C_c. Expanding ν in terms of the monoidal structure, this becomes $f \circ (c \otimes z_b) \circ \alpha \circ (\delta \otimes b) = z_b$.



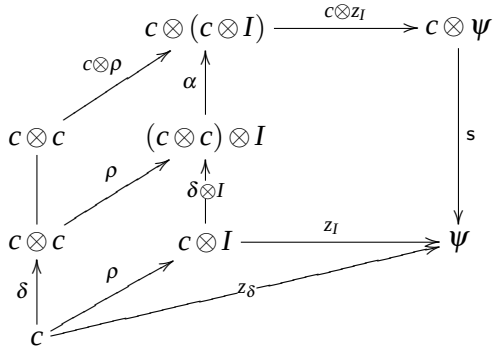
Compare this with a naïve definition based on the example of products in Set. If we define a 'c-parameterized fixed point' for s to be a map $z : c \to \psi$ such that $\mathsf{s} \circ (c \otimes x) \circ \delta = x$



and ask for a unique such z, we are implicitly fixing $b = I$ where Definition 6.1.8 quantifies over arbitrary b. This can be seen in the following proof.

Proposition 6.1.9 *A c-parameterized endomap with a c-parameterized unique fixed point has a unique 'c-parameterized fixed point'.*

Proof. By coherence and naturality, ρ gives a coalgebra morphism from δ to ν_I, so we have a 'c-parameterized fixed point' z_δ given by $z_I \circ \rho$, where z_I gives the unique fixed I-point for the endomap s gives in C_c. In a monoidal category, ρ is an isomorphism and so z_δ must be unique: a second parameterized fixed point for s would give a second fixed I-point.



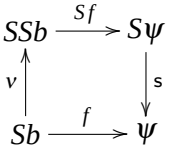
□

6.1.10 Comonads and Parameterized Unique Fixed Points

More generally, suppose $S : C \to C$ carries comonad structure with comultiplication $\nu : S \Rightarrow SS$ and counit $\iota : S \Rightarrow \mathrm{Id}$.

Definition 6.1.11 *An S-parameterized endomap $\mathsf{s} : S\psi \to \psi$ has a S-parameterized very unique fixed point if the endomap it gives in the Kleisli category C_S is a corecursive algebra.*

Now the diagram in C that says f is a morphism from ν_b to s is the same diagram that says f gives a fixed b-point for the arrow given by s in the Kleisli category C_S.



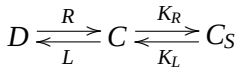
This means that, as endomaps in C_S, corecursive algebras for S have unique fixed points in the sense of Section 6.1. Actually, they have very unique fixed points.

Proposition 6.1.12 *Taken as an endomap in the Kleisli category for S, a corecursive algebra for S has a very unique fixed point.*

This is Corollary 2.2.11 with the left and right Kleisli adjoints, K_L and K_R, for L and R. In this adjunction, the transpose of a map $s : Sa \to a'$ in C is the map from a to a' in C_S given by s. Note that we do not use the fact that K_L is left adjoint to K_R. The comonad structure is used to build the natural transformation $\mathrm{Id}_{C_S} \Rightarrow K_R K_L$, which is all we need. Here is another instance of Corollary 2.2.11.

Proposition 6.1.13 *Given functors R and L, a natural transformation $\eta : \mathrm{Id} \to RL$ and a comonad S, corecursive algebras for LSR transpose to give endomaps in C_S with very unique fixed points.*

This can be seen either as Lemma 2.2.12 followed by Proposition 6.1.12 or as Corollary 2.2.11 with $L \circ K_L$ and $K_R \circ R$ for L and R.



As we pointed out after Corollary 2.2.11, the transformation ι can also be used to obtain corecursive algebras for S from corecursive endomaps in C. When ι is the counit of comonad structure on S, this amounts to using K_R to carry endomaps in C to endomaps in C_S. Using $\eta : \mathrm{Id} \to RL$, corecursive endomaps in C can be obtained from corecursive algebras for LR. We therefore have two ways of obtaining corecursive endomaps C_S, starting either with a corecursive algebra for LSR or one for LR. In applications, the second is seen to be a degenerate form of the first, with weaker properties.

6.2 Fixed Point Algebras

In this section, each category is equipped with a class of pointed objects and a class of strict maps. We use boldface variables to range over these maps and objects. A *strict algebra morphism* is one given by a strict map $\mathbf{h}$ and a *pointed algebra* is one with a pointed carrier $\mathbf{a}$ (indicated by a dot on the structure map when the carrier is implicit). Note that a pointed algebra does not necessarily have a strict structure map and, although the terms 'strict' and 'pointed' suggest certain concrete examples, the results below hold with fairly weak conditions on the choice of pointed objects and strict maps. Note in particular that strict maps are not necessarily between pointed objects.

Definition 6.2.1 *A* fixed point algebra *is a corecursive algebra from which there is a unique strict algebra morphism into any pointed algebra.*

$$\begin{array}{ccccc} Fb & \xrightarrow{Fz_p} & F\psi & \xrightarrow{F\mathbf{r}_f} & F\mathbf{a} \\ {\scriptstyle p}\uparrow & & \downarrow{\scriptstyle \mathsf{s}} & & \downarrow{\scriptstyle f} \\ b & \xrightarrow{z_p} & \psi & \xrightarrow{\mathbf{r}_f} & \mathbf{a} \end{array}$$

Note that fixed point algebras do not necessarily have strict structure maps and are not necessarily pointed.

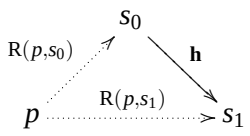
In the presence of a fixed point algebra, each set $\mathrm{rec}(p,f)$ of recursive morphisms contains a *canonical recursive morphism* $\mathrm{crm}(p,f) = \mathbf{r}_f \circ z_p$. The existence of such morphisms also follows from a weaker form of fixed point algebra that appears in our analysis of transposition.

Definition 6.2.2 *A* weak fixed point algebra *is a corecursive algebra equipped with a strict algebra morphism into each pointed algebra.*

6.2.3 Uniform Families of Recursive Morphisms

A fixed point algebra induces a well-behaved family of recursive morphisms into pointed algebras. Given a coalgebra p and an algebra s on a pointed object $\mathbf{c}$, we compose $\mathbf{r}_s$ with the unique recursive morphism z_p for s to obtain a canonical recursive morphism $\mathrm{crp}(p,s)$ into s.

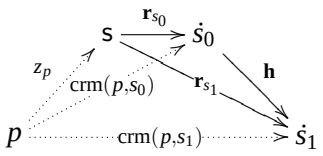
Definition 6.2.4 *A family* $\mathrm{R}(p,s)$ *of recursive morphisms is* uniform (with respect to strict maps) *if, given any strict morphism* $\mathbf{h}$ *between pointed algebras* s_0 *and* s_1 *we find that* $\mathrm{R}(p,s_1) = \mathbf{h} \circ \mathrm{R}(p,s_0)$.



In applications, strict maps are closed under composition and families of canonical recursive morphims are uniform.

Proposition 6.2.5 *If strict maps are closed under composition, the family of recursive morphisms induced by a fixed point algebra is uniform.*

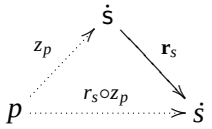
Proof. If strict maps compose, then strict endomap morphisms compose. So, given a strict endomap morphism $\mathbf{h}$ from s_0 to s_1, $\mathbf{h} \circ \mathbf{r}_{s_0}$ is a strict endomap morphism from s to s_1 which must then equal the unique strict recursive morphism $\mathbf{r}_{s_1}$. From $\mathbf{h} \circ \mathbf{r}_{s_0} = \mathbf{r}_{s_1}$ we have $\mathbf{h} \circ \mathbf{r}_{s_0} \circ z_b = \mathbf{r}_{s_1} \circ z_b$, which means $\mathbf{h} \circ \mathrm{crm}(p, s_0) = \mathrm{crm}(p, s_1)$.



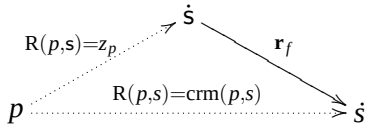
□

If the fixed point algebra is pointed, uniformity characterizes its family of recursive morphisms.

Proposition 6.2.6 *In the presence of a* pointed *fixed point algebra* s *for* F*, any uniform family of recursive morphisms into pointed algebras* s *is given by* $\mathrm{crm}(p,s) = r_s \circ z_p$.



Proof. Let $\mathrm{R}(p,s)$ be a uniform family of recursive morphisms for pointed algebras s. Our fixed point algebra s is pointed so this family includes a recursive morphism $\mathrm{R}(p,\mathsf{s})$ which must then equal the unique recursive morphism z_b. The algebra morphism $\mathbf{r}_s$ is strict, so uniformity gives $\mathrm{R}(p,s) = \mathbf{r}_s \circ \mathrm{R}(p,\mathsf{s})$ and hence $\mathrm{R}(p,s) = \mathbf{r}_s \circ z_p$ which means $\mathrm{R}(p,s) = \mathrm{crm}(p,s)$.



□

Lemma 6.2.7 *If strict maps are closed under composition, a pointed fixed point algebra induces the unique uniform family of recursive morphisms (for a given endofunctor).*

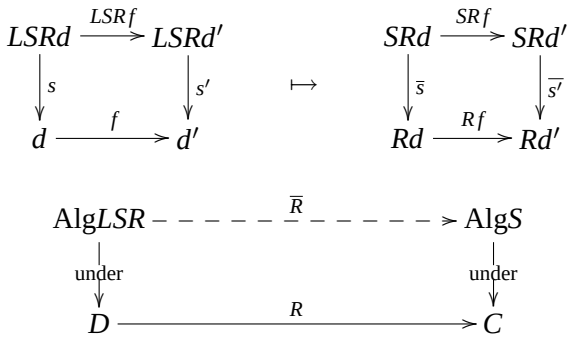
Applied to the identity endofunctor on categories of complete partial orders, we obtain the characterization of least fixed points known as Plotkin's Axiom.

6.2.8 Transposition of Fixed Point Algebras

Theorem 2.2.12 says corecursive algebras $LSR\phi \to \phi$ transpose to corecursive algebras $SR\phi \to R\phi$. We want to extend this to fixed point algebras. Whereas Theorem 2.2.12 holds with no conditions on R, L, η and S, the transposition of a fixed point algebra s to a fixed point algebra $\bar{\mathsf{s}}$ is less general. Given a pointed algebra s for S, we need to show that there exists a unique strict morphism $\mathbf{r}_s$ from $\bar{\mathsf{s}}$ to s. We expect to obtain this morphism from the unique strict morphism $\mathbf{r}_{\widehat{s}}$ from s to some pointed algebra $\widehat{s}$.

$$\begin{array}{ccc} LSR\phi & \xrightarrow{LSR\mathbf{r}_{\widehat{s}}} & LSR\mathbf{d} \\ \downarrow{\scriptstyle \mathsf{s}} & & \downarrow{\scriptstyle \widehat{s}} \\ \phi & \xrightarrow{\mathbf{r}_{\widehat{s}}} & \mathbf{d} \end{array} \quad \mapsto \quad \begin{array}{ccc} SR\phi & \xrightarrow{S\mathbf{r}_s} & S\mathbf{a} \\ \downarrow{\scriptstyle \bar{\mathsf{s}}} & & \downarrow{\scriptstyle s} \\ R\phi & \xrightarrow{\mathbf{r}_s} & \mathbf{a} \end{array}$$

If s is an algebra for LSR then its transpose $\bar{s}$ is an algebra for S, without conditions on R, L, η and S. Also, if f gives an algebra morphism from s to s', then Rf gives an algebra morphism from $\bar{s}$ to $\overline{s'}$. In other words, the functor R lifts to a functor $\overline{R}$ from AlgLSR to Alg$S$.



Suppose the algebra $\widehat{s}$ is pointed and transposes to s. Suppose $\overline{R}$ preserves strictness. Taking $\mathbf{r}_s = \overline{R}\mathbf{r}_{\widehat{s}}$ then gives us a strict morphism from $\bar{\mathsf{s}}$ to s. So, if $\overline{R}$ preserves strictness and is *surjective on pointed objects*, meaning every pointed object in Alg$S$ is the image of some pointed object in AlgLSR, then $\bar{\mathsf{s}}$ is a weak fixed point algebra. Now if R preserves strictness, then so does $\overline{R}$ (which does not mean strict algebras transpose to strict algebras) and this is a natural condition to place on R. On the other hand, suppose R is surjective on pointed objects. The existence of $\mathbf{d}$ with $\mathbf{a} = R\mathbf{d}$ does not ensure the existence of $\mathbf{d}$ such that $D(LSR\mathbf{d}, \mathbf{d})$ contains $\widehat{s}$ with $s = \overline{R}\widehat{s}$. But suppose R is also *semi right adjoint to* L. By this we mean a natural transformation ε is given such that $f = R(\varepsilon_d \circ Lf) \circ \eta_a$ for any map f into any $a = Rd$. This is equivalent to the triangle law $R\varepsilon \circ \eta_a = \mathrm{id}_a$. Now, taking $\widehat{s}_{\mathbf{d}} = \varepsilon_{\mathbf{d}} \circ Ls$ we have $s = R\widehat{s} \circ \eta_{\mathbf{a}} = \overline{R}\widehat{s}$. Note that different choices of $\mathbf{d}$ will give different $\widehat{s}_{\mathbf{d}}$.

Proposition 6.2.9 *If R preserves strictness, is surjective on pointed objects and is semi right adjoint to L, then a fixed point algebra for LSR transposes to a weak fixed point algebra for S.*

To show that the $\mathbf{r}_s$ are unique requires further conditions. Given an arbitrary strict morphism $\mathbf{m}$ from $\bar{\mathsf{s}}$ to s, we need to show that $\mathbf{m} = \mathbf{r}_s$. This follows from the uniqueness of $\mathbf{r}_{\widehat{s}}$, if we assume that $\overline{R}$ is *full on strict maps* meaning every strict morphism from $\overline{R}s$ to $\overline{R}s'$ is the image of some strict morphism from s to s'. Our strict morphism $\mathbf{m}$ is then of the form $\overline{R}\widehat{\mathbf{m}}$, where $\widehat{\mathbf{m}}$ is some strict morphism from s to $\widehat{s}$, but $\mathbf{r}_{\widehat{s}}$ is the unique such morphism so $\widehat{\mathbf{m}} = \mathbf{r}_{\widehat{s}}$ and $\mathbf{m} = \overline{R}\widehat{\mathbf{m}} = \overline{R}\mathbf{r}_{\widehat{s}} = \mathbf{r}_s$.

When is $\overline{R}$ full on strict morphisms? It is not enough for R to be full on strict maps because we may have $\mathbf{m} = R\widehat{\mathbf{m}}$ without $\widehat{\mathbf{m}}$ being a morphism from s to $\widehat{s}$. Suppose, however, that R is full on strict maps and L is *semi left adjoint* to R. This means $f = \varepsilon \circ L(Rf \circ \eta)$ for all $f : Lc \to d$. Now if $\mathbf{m} = \overline{R}\widehat{\mathbf{m}}$ and $\overline{R}\widehat{\mathbf{m}}$ is a morphism from $\bar{\mathsf{s}}$ to s then $\widehat{\mathbf{m}}$ is a morphism from s to $\widehat{s}$.

Theorem 6.2.10 *If R preserves strictness, is surjective on pointed objects, full on strict maps and right adjoint to L, then a fixed point algebra for LSR transposes to a fixed point algebra for S.*

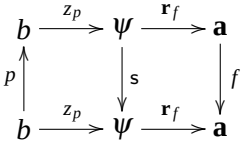
In applications, we find ourselves with a category D equipped with a class of pointed objects and a class of strict maps and are given a functor $R : D \to C$ along which we would like to transpose fixed point algebras. What objects and maps in C should we take as pointed and strict if we would like R to satisfy the conditions of Theorem 6.2.10? If we equip C with the class of pointed objects given by applying R to pointed objects in D, then R is surjective on pointed objects by definition. If we equip C with the class of strict maps given by applying R to strict maps in D, R preserves strictness. However, despite being surjective on strict maps, R may not be full on strict maps. If $Rd_0 = Rd_1$ with $d_0 \neq d_1$, then $\mathbf{m} : d \to d_0$ and, hence, $R\mathbf{m} : Rd \to Rd_0$ may be strict without there being a strict $\mathbf{m}' : d \to d_1$ such that $R\mathbf{m}' = R\mathbf{m}$. However, if we want R to preserve strictness, we cannot force fullness on strict maps by defining $m : c \to c'$ to be strict

if for all d over c and d' over c' there is a strict map $\widehat{\mathbf{m}} : d \to d'$ over m. Fortunately, this problem does not arise in the adjunctions we consider between concrete categories of domains. In these adjunctions, R is injective on objects and the above definitions coincide.

6.2.11 Fixed Point Objects as Fixed Point Algebras

As we saw in Section 6.1, a recursive morphism out of an identity coalgebra for the identity endofunctor is a fixed point and so the canonical recursive morphisms induced by a fixed point algebra for the identity endofunctor include canonical fixed points for endomaps on pointed objects. A fixed point algebra for the identity endofunctor therefore serves as fixed point object.

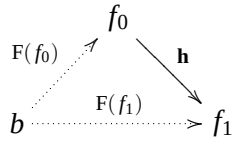
Definition 6.2.12 *A* (weak) fixed point object *is a (weak) fixed point algebra for the identity endofunctor.*



As observed in Section 6.1, such endomaps have unique fixed points in a strong sense.

Specialised to the identity endofunctor, the results of Section 6.2.3 become results about fixed point objects and families of fixed points for endomaps on pointed objects. For each b, a fixed point object induces a well-behaved family of fixed b-points. Given an endomap f on a pointed object $\mathbf{c}$, we compose $\mathbf{r}_f$ with the unique fixed b-point z_b for s to obtain a canonical fixed b-point $\mathrm{cfp}(f)$ for f.

Definition 6.2.13 *A family* $\mathrm{F}(f)$ *of fixed b-points is* uniform (with respect to strict maps) *if, given any strict map* $\mathbf{h}$ *such that* $f_1 \circ \mathbf{h} = \mathbf{h} \circ f_0$*, we find that* $\mathrm{F}(f_1) = \mathbf{h} \circ \mathrm{F}(f_0)$.



Corollary 6.2.14

- *If strict maps are closed under composition, the family of fixed b-points induced by a fixed point object is uniform.*
- *In the presence of a* pointed *fixed point object, any uniform family of fixed b-points for pointed endomaps f is given by* $\mathrm{cfp}(f) = \mathbf{r}_f \circ z_b$.
- *If strict maps are closed under composition, a pointed fixed point object induces the unique uniform family of fixed b-points for each b.*

Note that, because a fixed point object has a very unique fixed point, the family $\mathrm{cfp}(f)$ is also uniform with respect to change of b. Composition with any map $g : b_0 \to b_1$ takes canonical b_1-points to canonical b_0-points. In the presence of a final object 1, this means the canonical fixed b-points are determined by the canonical fixed 1-points.

Taking $S = \mathrm{Id}$, Proposition 6.2.9 produces weak fixed point objects and Theorem 6.2.10 produces fixed point objects.

Corollary 6.2.15 *If R preserves strictness, is surjective on pointed objects*

- *and is semi right adjoint to L, then a fixed point algebra for LR transposes to a weak fixed point object.*
- *and is full on strict maps and right adjoint to L, then a fixed point algebra for LR transposes to a fixed point object.*

Now, suppose we have an endofunctor S with comonad structure and we construct its Kleisli adjunction $K_L \dashv K_R$. The right adjoint K_R is the identity on objects and hence injective on objects. It carries a map $f : a \to b$ to the map in C_S given by $f \circ \iota_a : Sa \to b$. Strict maps in C_S are therefore maps of the form $\mathbf{h} \circ \iota$ where $\mathbf{h}$ is a strict map in C. Corollary 6.2.15, with K_L and K_R for L and R, says fixed point algebras for $S = K_L \circ K_R$ transpose to fixed point objects in the Kleisli category *C_S with respect to the above definition of strict map and pointed object in C_S*. In Section 6.1 we only needed the natural transformation $\eta : \mathrm{Id} \Rightarrow K_R K_L$, but here we are using both triangle laws of the adjunction $K_L \circ K_R$. The one law gives the existence of r_f and, hence, the existence of a canonical family of c-parameterized fixed b-points in C_S. The other, the uniqueness of this family with respect to uniformity (assuming the fixed point algebra for S is pointed).

Definition 6.2.16 *A* parameterized fixed point object *is a fixed point algebra for a comonad.*

When S is $c \otimes (\cdot)$, maps of the form $f \circ \iota_a : c \otimes a \to b$ give 'constant' maps in the Kleisli category C_c. The behaviour of composition with such maps simplifies the initiality property of a fixed point object in C_c. Being a strict morphism from an endomap given by $s' : c \otimes a' \to a'$ to an endomap given by $s : c \otimes a \to a$ is equivalent to being of the form $f \circ \iota_a$ with f an algebra morphism from s' to s, and so the initiality property reduces, as expected, to the algebra initiality property of the original fixed point algebra. Given any map $f : c \otimes \mathbf{a} \to \mathbf{a}$, with $\mathbf{a}$ pointed, a fixed point algebra s_c for $c \otimes (\cdot)$ induces both canonical fixed b-points in C_c, as described above, and a 'c-parameterized fixed point' $r_f \circ z_I \circ \rho$ (see Proposition 6.1.9).

$$\begin{array}{ccccccc}
c \otimes c & \xrightarrow{c\otimes\rho} & c \otimes (c \otimes I) & \xrightarrow{c\otimes z_I} & c \otimes \psi & \xrightarrow{c\otimes \mathbf{r}_f} & c \otimes \mathbf{a} \\
\delta \uparrow & & \nu \uparrow & & \downarrow \mathsf{s}_c & & \downarrow f \\
c & \xrightarrow{\rho} & c \otimes I & \xrightarrow{z_I} & \psi & \xrightarrow{\mathbf{r}_f} & \mathbf{a}
\end{array}$$

6.2.17 Freyd Algebras as Fixed Point Algebras

The notion of free algebra introduced by Freyd in [14] is equivalent to our notion of fixed point algebra in the special case where all objects are pointed and all maps strict.

Proposition 6.2.18 *For all algebras* s*, the following are equivalent:*

1. s *is initial and the inverse of a final coalgebra.*
2. s *is initial and corecursive.*

$$\begin{array}{ccccc}
Fb & \xrightarrow{Fz_p} & F\psi & \xrightarrow{F\mathbf{r}_f} & F\mathbf{a} \\
p \uparrow & & \downarrow \mathsf{s} & & \downarrow f \\
b & \xrightarrow{z_p} & \psi & \xrightarrow{\mathbf{r}_f} & \mathbf{a}
\end{array}$$

Definition 6.2.19 *A* Freyd algebra *is an algebra that is initial and corecursive.*

Suppose that in D all objects are pointed and all maps are strict. Then, with conditions on R, Corollary 6.2.15 says a Freyd algebra for LR transposes to a fixed point object in C,

If we are also given an endofunctor S on C, Theorem 6.2.10 then says a Freyd algebra for LSR transposes to fixed point algebra for S. If S carries comonad structure, we obtain a fixed point object in C_S as described in the previous section or we can transpose the Freyd algebra for LSR directly to a fixed point object in C_S using Corollary 6.2.15 with $K_R \circ R$ and $L \circ K_L$ for L and R.

6.3 Canonical Fixed Points in Compact Structural Adjunctions

Given a structurally compact category D, we take all objects and maps in D to be pointed and strict. If anyone asks for actual points and strictness, we can turn to the structural compactness of D. Given any structural action on D, the identity endofuctor on D is trivially structural, so when D is structurally compact it contains an initial/final object $\bot = \mu d.d$. As $\bot$ is initial, every object d in D has a canonical point $\bot : \bot \to d$ given by the unique map from $\bot$. As $\bot$ is final, we also obtain canonical generalized points $\bot : e \to d$ given by the unique maps that factor though $\bot$. In this sense, every object in D is pointed. Moreover, by the universal properties of $\bot$, every map in D preserves the canonical points induced by $\bot$ and so, in this sense, every map is strict.

Given a functor R, we take pointed objects and strict maps in the codomain of R as follows.

1. c is pointed if $c = R\mathbf{d}$ for some pointed $\mathbf{d}$
2. f is strict if $f = R\mathbf{g}$ for some strict $\mathbf{g}$

If asked for points and strictness, we can obtain these from the domain of R: asked for a point in Rd, we apply R to a point in d. Strictness in the codomain then follows from strictness in the domain.

With these classes of pointed object and strict map the conditions of Proposition 6.2.9 are satisfied by the various forms of compact structural adjunction described in Chapter 5. The weak fixed point object produced by Proposition 6.2.9 is enough to identify canonical fixed points.

On categories of domains, with conditions on the domain-theoretic monad satisfied by concrete monads, R is injective on objects, and the conditions of theorem 6.2.10 are satisfied. In these adjunctions the canonical fixed points are characterized by their uniformity.

6.3.1 Ordinary Fixed Points

Given a compact structural adjunction $L \dashv U : D \to C$, the composite LU is structural and so we have a Freyd algebra $\sigma : LU\omega \to \omega$. The map σ transposes to a weak fixed point object in C by Corollary 6.2.15. This weak fixed point object induces a canonical family of fixed points for endomaps on objects of the form Ud. Likewise for a compact costructural adjunction.

In particular, every compact monoidal adjunction and every compact closed adjunction has a canonical family of generalized fixed points, as does every compact suitable structural adjunction, including the adjunctions derived from a balanced exponential on a compact category, in which case we have a canonical family of fixed points for endomaps on objects of the form $d-e$.

Although the fixed point object induces fixed b-points for any b in C, these are generated by the fixed 1-points, where $1 = U\bot$. As right adjoint, U takes the final object $\bot$ in D to a final object in C which generates the unique fixed b-points of the fixed point object. Similarly, the canonical 1-point $U\bot : U\bot \to Ud$ generates canonical b-points in Ud. It can be checked that these are the points the fixed point object induces for the identity endomap on Ud.

6.3.2 Parameterized Fixed Points

Given a compact structural adjunction $L \dashv U : D \to C$ with a balanced action $\boxslash$ on C, the functors taking d to $L(x \boxslash Ud)$ are structural and so we have Freyd algebras $\sigma_x : L(x \boxslash U\psi_x) \to \psi_x$. By

Corollary 6.2.15, the σ_x transpose to weak x-parameterized fixed point objects in C which induce families of parameterized fixed b-points $x \boxtimes b \to Ud$ for parameterized endomaps $x \boxtimes Ud \to Ud$ on objects of the form Ud. Again, these are generated by the x-parameterized fixed 1-points.

In particular, every compact suitable adjunction has canonical families of parameterized fixed points $c \boxtimes b \to Ud$ for endomaps $c \boxtimes Ud \to Ud$. When the adjunction derives from a suitable exponential on a compact category we have a canonical family of parameterized fixed points for endomaps $c \boxtimes (d - e) \to (d - e)$. When the adjunction or exponential is given by a monoidal/cartesian adjunction these fixed b-points are generated by the family of 'c-parameterized fixed points' $c \to Ud$ corresponding to the c-parameterized fixed 1 points $c \times 1 \to Ud$.

The compact category of pointed objects C^T for a domain-theoretic commutative monad on C, is characterised by a monoidal/cartesian adjunction with C and the compact category of partial maps C_T for a domain-theoretic commutative lifting monad, by a suitable structural adjunction (see Section 5.3). Either way, the category C has a (parameterized) fixed point object which induces canonical (parameterized) fixed points.

In CPO, the parameterized fixed point object $U\psi_c$ is an infinite c-branching tree and the generic parameterized endomap suc_c carries nodes up along the branch indicated by the element of c. In particular, ψ_1 is the vertical natural numbers and suc_1 is the successor map. The tree $U\psi_c$ also has a point for each c-stream. In the order, each stream lies just above the path it indicates up through the tree. For $c = 1$, there is the one stream above the one path, but in general there are many. In can be checked that the fixed points induced by these fixed point objects are least fixed points.

6.3.3 Internal Fixed Points

If the monoidal structure on C in a compact suitable adjunction is closed, we can internalise the operation that produces canonical (parameterized) fixed points in C. We use the existence of parameterized fixed points. For each d in D, we take the evaluation map $(Ud \multimap Ud) \boxtimes Ud \to Ud$ as a $(Ud \multimap Ud)$-parameterized endomap and construct the $(Ud \multimap Ud)$-parameterized fixed point $(Ud \multimap Ud) \to Ud$. These maps make up a transformation with components

$$\mathrm{cfp}_d : (Ud \multimap Ud) \to Ud$$

that internalise the canonical fixed point operation. By internalising parameterized evaluation to obtain a map $(c \boxtimes Ud \multimap Ud) \boxtimes (c \boxtimes Ud) \to (c \boxtimes Ud)$, we may even internalise the operation producing canonical parameterized fixed points. This produces a transformation

$$\mathrm{cfp}^b_d : (b \boxtimes Ud \multimap Ud) \to (b \multimap Ud).$$

In particular, every compact monoidal/cartesian closed adjunction has an internal (parameterized) fixed point operation and, if the acting category is bistructural, so does a compact suitable adjunction. For example, CPO is cartesian closed, and hence bistructural, so the adjunctions with pCPO and with $\mathrm{CPPO}_\perp$ both induce internal (parameterized) fixed point operations.

Given a balanced exponential, without a costructural action on the acting category C, we cannot internalise the fixed point operation directly, but we can construct a corresponding transformation with components $d - ((d - e) \oslash e) \to (d - e)$ or, for parameterized fixed points, $d - ((c \boxtimes (d - e) \oslash e) \to (d - (c \boxtimes e))$.

6.3.4 Uniform Fixed Points

Given a compact structural adjunction $L \dashv U : D \to C$. Suppose U is injective on objects. The image of U then forms a subcategory of C. In particular, strict maps in C are closed under composition and a fixed point object induces a uniform family of fixed points by Corollary 6.2.14.

Moreover, because the original Freyd algebra is pointed (all objects are pointed in D) and transposition preserves pointedness, our fixed point object is pointed and hence, by Corollary 6.2.14, the fixed point object induces the unique uniform family of fixed points in C (with respect to strict morphisms between endomaps on pointed objects).

On categories of partial orders the lift monad is injective on objects and so the the monad's right Kleisli adjoint, which does all the 'object work', is injective on objects. Also, partial orders carry at most one lift monad algebra structure, which amounts to a least element, and so the monad's right Eilenberg-Moore adjoint is also injective on objects. In categories of partial orders then, canonical fixed points, which coincide with least fixed points, are characterized by uniformity.

The right Kleisli for any comonad is injective on objects, so if R is such an adjoint, or an injective-on-objects functor U followed by such an adjoint, a fixed point object in the codomain of R induces the unique uniform family of fixed points (assuming strict maps in the domain of R are closed under composition).

If the comonad on C given by the action of some x is a structural, then the structural compactness of D gives us an x-parameterized fixed point object ψ_x that induces the unique uniform family of fixed points in the Kliesli category C_x.

Chapter 7

Recursive Linear Types

In [3], Benton studies a linear/nonlinear logic with mixed derivations of two kinds of sequent. Various forms of the logic are presented including a natural deduction system with a calculus of assigned terms called LNL. This calculus has models in symmetric monoidal/cartesian closed adjunctions. The monoidal category interprets linear sequents and the cartesian category, nonlinear sequents. Examples include adjunctions constructed from models of Intuitionistic Linear Logic and the adjunctions used in denotational semantics, which we describe in Chapter 5. In the domain theoretic examples, the monoidal category is algebraically compact and this can be used to model recursive types. The existence of invariant algebras for endofunctors on the monoidal category, suggests that we allow type recursion on linear type expressions containing a free linear type variable, which are interpreted as endofunctors on the monoidal category. With recursive linear types, even the pure calculus, with no base types, has a rich collection of types and terms including fixed point combinators.

In Section 7.1, we introduce our syntax for LNL, which includes derivations of types in context to regulate the formation of recursive types, and discuss the linearity of its logic of derivable sequents. We then add rules for recursive linear types. Our syntax for function types, units and pairing is standard. Our syntax for adjoint types matches Levy's CBPV and for recursive types, Plotkin's FPC.

In Section 7.2, we analyse adjoint types in LNL theories and their term models. The special feature of LNL is the decomposition of bang, which controls weaking and conraction in Linear Logic, into two type constructors. We observe that the term model is naturally viewed as a distributor and that, from this point of view, the two constructors may be understood in isolation. Also, using linear and nonlinear function types, the isomorphism giving the adjunction, is definable in the closed fragment of LNL.

7.1 A Recursive Linear/NonLinear Calculus

LNL has two kinds of sequent which, following Benton, we refer to as 'linear' and 'nonlinear'. The rules and equations for LNL are given in Figures 7.1, 7.2 and 7.3. We have added explicit type rules and linear type contexts to control the construction of recursive linear types, but this has no effect on the LNL rules. In addition to the beta reductions and commuting conversions Benton gives in [3], we include eta laws and the application equation

$$(g)(b \texttt{ to x in } a) \quad =_{\mathrm{a}} \quad b \texttt{ to x in } (g)a$$

to obtain a full equational theory. The application equation is used in Section 7.2.5 where we internalize the adjunction LNL generates and is required for completeness with respect to cate-

gorical models.

We use upright X for nonlinear base types, italic X and Y for nonlinear types, upright x and y for variables of nonlinear type, Γ for lists of nonlinear typings and italic x, y and f for terms of nonlinear type. We use upright A for linear type variables, Ψ for lists of linear type variables, italic A and B for linear types, upright a and b for variables of linear type, Π for lists of linear typings and italic a, b, and g for terms of linear type.

Syntactically, type contexts are lists of distinct type variables. and term contexts are lists of variable typings with distinct term variables. Comma and the little bar, or 'stile', both mean concatenation. By convention, we use the stile when joining a nonlinear list to a linear list.

For the reader familiar with [3], we note some cosmetic changes. In place of '$\vdash_{\mathcal{C}}$', and '$\vdash_{\mathcal{L}}$' we use '$\rhd \ldots \vdash$' and '$\blacktriangleright \ldots \vdash$' to distinguish nonlinear and linear sequents. In place of a semicolon, which would clash with the notation used in seperation logics, we use a stile to divide contexts into nonlinear and linear halves. In place of Benton's syntax for mixing, we follow the syntax of Levy's CBPV [23]. This admits operational readings that can be used to incorporate computational effects, although the only effect we have in mind is the possiblity of nonterminating recursions. As effects go, nontermination is very degenerate which allows for more structure on linear types than is found in CBPV. To remind ourselves of this specialization, we use L in place of the F in CBPV.

Benton [3]	here	thinking
F	L	lift
G	U	underlying domain
F(x)	$\texttt{produce}(x)$	produce result x
G(a)	$\texttt{thunk}(a)$	treat a as data (address of instructions a)
let F(b) = x in a	b `to` x `in` a	bind the result of b to x and proceed with a
derelict(x)	$\texttt{force}(x)$	treat x as instructions (load program counter with x)

Also, seeing as our semantics requires a list of typings and interprets exchange explicitly, our derivations make explicit use of the exchange rules shown in Figure 7.4.

The equations we write all have implied free occurence and derivablity conditions which may be reconstructed as described in [23]. We use arrows to remind ourselves how certain equations are used in reduction systems, but the theory is equational.

We take the collection of nonlinear base types to be some countable infinity. In place of a collection of base types, we may as well parameterize our calculus on a graph F with base types for nodes and (unary) basic operations for edges. In this case we consider the calculus LNL(F) which extends LNL with a rule and a term constructor f( ) for each basic operation f.

$$\frac{\rhd \Gamma \vdash x : \mathrm{X}}{\rhd \Gamma \vdash \mathtt{f}(x) : \mathrm{Y}}$$

We assume that the base types of F are included in our infinite collection of base types so that LNL and LNL(F) have the same types. Note that the categorical semantics of the above rule costs us nothing because X and Y are not derived types. In applications, more expressive basic operations involving derived types could be added.

We use square brackets for substitution into expressions with holes. The maximal decomposition of B into an expression with holes filled with a variable A is written $B[\mathrm{A}]$. The expression with just the holes (and no free occurence of A) is $B[]$ and the same expression with the holes filled with some expression C is written $B[C]$.

7.1.1 Linear and Nonlinear Lambda Calculi

The closed fragment of LNL, shown in Figure 7.1, contains the simply typed lambda calculus and a linear version of the same calculus.

$$\frac{\triangleright X \quad \ldots \quad \triangleright Y}{\triangleright \mathrm{x} : X, \ldots, \mathrm{y} : Y \vdash \mathrm{y} : Y} \qquad \frac{\blacktriangleright A}{\blacktriangleright \mathrm{a} : A \vdash \mathrm{a} : A}$$

$$\frac{\triangleright X \quad \triangleright Y}{\triangleright (X \multimap Y)} \qquad \frac{\blacktriangleright A \quad \blacktriangleright B}{\blacktriangleright (A \multimap B)}$$

$$\frac{\triangleright \Gamma, \mathrm{x} : X \vdash y : Y}{\triangleright \Gamma \vdash \lambda \mathrm{x}.y : (X \multimap Y)} \qquad \frac{\blacktriangleright \Pi, \mathrm{a} : A \vdash b : B}{\blacktriangleright \Pi \vdash \lambda \mathrm{a}.b : (A \multimap B)}$$

$$\frac{\triangleright \Gamma \vdash x : X \quad \triangleright \Gamma \vdash f : (X \multimap Y)}{\triangleright \Gamma \vdash (f)x : Y} \qquad \frac{\blacktriangleright \Pi \vdash a : A \quad \blacktriangleright \Pi' \vdash g : (A \multimap B)}{\blacktriangleright \Pi, \Pi' \vdash (g)a : B}$$

The linear calculus differs from the nonlinear calculus in the rule for variables: the context contains just the linear variable being introduced. This exact match between linear variables in the context and free linear variables in the term is preserved by the other linear rules, which are multiplicative in the linear part of the context. Consequently, the collection of derivable linear sequents admits no weakening or contraction on linear types whatsoever, while the collection of nonlinear sequents admits all weakenings and all contractions.

7.1.2 Admissible Weakenings and Contractions

The elimination rule for L

$$\frac{\Psi \blacktriangleright \Gamma, \mathrm{x} : X \mid \Pi \vdash a : A \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash b : \mathrm{L}X}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a : A}$$

uses the nonlinear part of the context. Together with the mixed version of the linear variable introduction rule,

$$\frac{\Psi \triangleright X \quad \ldots \quad \Psi \triangleright Y \quad \Psi \blacktriangleright A}{\Psi \blacktriangleright \mathrm{x} : X, \ldots, \mathrm{y} : Y \mid \mathrm{a} : A \vdash \mathrm{a} : A}$$

this expands the collection of derivable linear sequents to admit weakening on types of the form $\mathrm{L}X$. Given a derivation of $\blacktriangleright \Pi \vdash a : A$, each variable introduction

$$\frac{\blacktriangleright A}{\blacktriangleright \mathrm{a} : A \vdash \mathrm{a} : A} \quad \text{is replaced with} \quad \frac{\triangleright X \quad \blacktriangleright A}{\blacktriangleright \mathrm{x} : X \mid \mathrm{a} : A \vdash \mathrm{a} : A}$$

to obtain a derivation of $\blacktriangleright \mathrm{x} : X \mid \Pi \vdash a : A$. We then use the elimination rule for L.

$$\frac{\blacktriangleright \mathrm{x} : X \mid \Pi \vdash a : A \quad \blacktriangleright \mathrm{b} : \mathrm{L}X \vdash \mathrm{b} : \mathrm{L}X}{\blacktriangleright \Pi, \mathrm{b} : \mathrm{L}X \vdash \mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a : A}$$

Together with the introduction rule for L,

$$\frac{\Psi \triangleright \Gamma \vdash x : X}{\Psi \blacktriangleright \Gamma \vdash \mathtt{produce}(x) : \mathrm{L}X}$$

the elimination rule admits contraction on types of the form $\mathrm{L}X$. Given a derivation of $\blacktriangleright \Pi, \mathrm{b}_0 : \mathrm{L}X, \mathrm{b}_1 : \mathrm{L}X \vdash a$ each variable introduction

$$\frac{\dfrac{\triangleright X}{\blacktriangleright \mathrm{L}X}}{\blacktriangleright \mathrm{b}_i : \mathrm{L}X \vdash \mathrm{b}_i : \mathrm{L}X} \quad \text{is replaced with} \quad \frac{\triangleright X}{\dfrac{\triangleright \mathrm{x} : X \vdash \mathrm{x} : X}{\blacktriangleright \mathrm{x} : X \vdash \mathtt{produce}(\mathrm{x}) : \mathrm{L}X}}$$

to obtain a derivation of $\blacktriangleright \mathrm{x} : X \mid \Pi \vdash a[\mathtt{produce}(\mathrm{x}), \mathtt{produce}(\mathrm{x})] : A$. We then use the elimination rule for L.

$$\frac{\blacktriangleright \mathrm{x} : X \mid \Pi \vdash a[\mathtt{produce}(\mathrm{x}), \mathtt{produce}(\mathrm{x})] : A \qquad \blacktriangleright \mathrm{b} : \mathrm{L}X \vdash \mathrm{b} : \mathrm{L}X}{\blacktriangleright \Pi, \mathrm{b} : \mathrm{L}X \vdash \mathrm{b} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a[\mathtt{produce}(\mathrm{x}), \mathtt{produce}(\mathrm{x})] : A}.$$

Unlike the mixed linear variable rule, the elimination rule for U and the introduction rule for L, introduce linear sequents with purely nonlinear term context.

Note that variables in the nonlinear part of the context can appear free in terms of linear type, but that the linear part of the context is always empty for terms of nonlinear type.

The elimination rule for the unary type constructor L uses cut-and-bind machinery: the rule cuts to a so-called parasitic type A via a nonlinear variable x which is bound in the resulting term. Such terms are known as let-expressions and appear in programming languages, computational lambda calculi, linear lambda calculi and mixed calculi such as LNL [3] and CBPV [23].

Let-expressions require so-called commuting conversions, equations of the form

$$c[b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a] =_{\mathrm{cc}} b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ c[a]\,.$$

At the level of proofs, commuting conversions allow us to permute an instance of the cut-and-bind rule with parasitic type A past the introduction and elimination rules for the type A. At least one equation is required for each type constructor that may be used to produce the type of the parasitic formula—in this case, one for each linear type constructor. If new linear type constructors are added, new commuting conversions must be added.

The equational theory of closed LNL is generated by the equations in Figure 7.1. It has the usual beta and eta laws for the two kinds of function type together with beta and eta laws for U and L. It has two commuting conversions for L elimination, one with respect to $(\multimap)$ elimination and one with respect to L elimination.

7.1.3 Pairing and Units

LNL has pairing and units for both nonlinear and linear types. For nonlinear types it has additive rules and for linear types, multiplicative rules. This reflects the intended semantics in cartesian and monoidal categories Syntactically, the multiplicative rules maintain the exact, context-term match for linear variables.

In the logic of derivable terms, these rules give us intuitionistic conjunction and truth and linear tensor and unit. At level of terms, both the introduction and elimination rules for nonlinear pairing and unit can be understood in terms of basic operations, whereas the elimination rules for linear pairing and unit use cut-and-bind machinery.

The linear pairing and unit elimination rules have their own commuting conversions as well as commuting conversions with the other cut-and-bind rules.

7.1.4 Recursive Linear Types

Notice that our LNL type expressions contain two kinds of identifier, 'base types' of nonlinear type and 'type variables' of linear type. The terminology reflects the intended use. The collection of nonlinear base types is global and is chosen prior to type formation while the collection of linear type variables is local and is given for each sequent. While we may indulge in substitution on base types meta-syntactically, the rules for recursive linear types (Figure 7.5) only use substitution of linear type expressions for linear type variables.

We extend LNL with a type constructor that binds one linear type variable in a linear type to produce a new linear type. The rules and equations for the new type construction are shown in Figure 7.5. A type of the form $\mu \mathrm{A}.B$ is recursive in the sense that the associated introduction and elimination rules use substitution of $\mu \mathrm{A}.B$ for A in B.

As recursive types are linear and may occur as a parasitic type, we include a commuting conversion with respect to each of the cut-and-bind rules.

The simplest recursive linear type is the origin, $\bot = \mu \mathrm{A}.\mathrm{A}$. In our domain model, this is the one point domain, which is initial and final in the category of strict maps. Perhaps the next simplest is the vertical naturals, $\overline{\mathrm{N}} = \mu \mathrm{A}.\mathrm{LUA}$. In our domain model, this is $\leq$ on the natural numbers together with an element greater than every number. Generalizing the vertical naturals, are the vertical lists, $X^* = \mu \mathrm{A}.\mathrm{L}(X \times \mathrm{UA})$. In our domain model, this is the set of finite and infinite lists of elements of X ordered by list extension and alphabetically using the order on X. This set may also be viewed as a tree. Recursive types of the form $\mathrm{H}_B = \mu \mathrm{A}.(\mathrm{LUA} \multimap B)$ are used to type fixed point combinators in Sections 8.1.2 and 8.1.2.

$$\frac{}{\Psi \rhd \mathrm{X}} \qquad \frac{}{\Psi, \mathrm{A} \blacktriangleright \mathrm{A}}$$

$$\frac{\Psi \rhd X \quad \ldots \quad \Psi \rhd Y}{\Psi \rhd \mathrm{x}:X, \ldots, \mathrm{y}:Y \vdash \mathrm{y}:Y} \qquad \frac{\Psi \rhd X \quad \ldots \quad \Psi \rhd Y \quad \Psi \blacktriangleright A}{\Psi \blacktriangleright \mathrm{x}:X, \ldots, \mathrm{y}:Y \mid \mathrm{a}:A \vdash \mathrm{a}:A}$$

$$\frac{\Psi \rhd X \quad \Psi \rhd Y}{\Psi \rhd (X \multimap Y)} \qquad \frac{\Psi \blacktriangleright A \quad \Psi \blacktriangleright B}{\Psi \blacktriangleright (A \multimap B)}$$

$$\frac{\Psi \rhd \Gamma, \mathrm{x}:X \vdash y:Y}{\Psi \rhd \Gamma \vdash \lambda \mathrm{x}.y : (X \multimap Y)} \qquad \frac{\Psi \blacktriangleright \Gamma \mid \Pi, \mathrm{a}:A \vdash b:B}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash \lambda \mathrm{a}.b : (A \multimap B)}$$

$$\frac{\Psi \rhd \Gamma \vdash x:X \quad \Psi \rhd \Gamma \vdash f : (X \multimap Y)}{\Psi \rhd \Gamma \vdash (f)x : Y} \qquad \frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a:A \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash g : (A \multimap B)}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash (g)a : B}$$

$$\frac{\Psi \blacktriangleright A}{\Psi \rhd \mathrm{U}A} \qquad \frac{\Psi \rhd X}{\Psi \blacktriangleright \mathrm{L}X}$$

$$\frac{\Psi \blacktriangleright \Gamma \vdash a:A}{\Psi \rhd \Gamma \vdash \mathtt{thunk}(a) : \mathrm{U}A} \qquad \frac{\Psi \rhd \Gamma \vdash x:X}{\Psi \blacktriangleright \Gamma \vdash \mathtt{produce}(x) : \mathrm{L}X}$$

$$\frac{\Psi \rhd \Gamma \vdash x : \mathrm{U}A}{\Psi \blacktriangleright \Gamma \vdash \mathtt{force}(x) : A} \qquad \frac{\Psi \blacktriangleright \Gamma, \mathrm{x}:X \mid \Pi \vdash a:A \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash b : \mathrm{L}X}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a : A}$$

$$\begin{array}{rcl}
(\lambda \mathrm{x}.f[\mathrm{x}])y & \to_\beta & f[y] \\
(\lambda \mathrm{a}.g[\mathrm{a}])b & \to_\beta & g[b] \\
\mathtt{force}(\mathtt{thunk}(a)) & \to_\beta & a \\
\mathtt{produce}(y)\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a[\mathrm{x}] & \to_\beta & a[y] \\
f & =_\eta & \lambda \mathrm{x}.(f)\mathrm{x} \\
g & =_\eta & \lambda \mathrm{a}.(g)\mathrm{a} \\
x & =_\eta & \mathtt{thunk}(\mathtt{force}(x)) \\
a & =_\eta & a\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{produce}(\mathrm{x}) \\
(g)(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a) & =_\mathrm{a} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (g)a \\
(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ g)a & \to_\mathrm{cc} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (g)a \\
(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ c & \to_\mathrm{cc} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ c)
\end{array}$$

Figure 7.1: Rules and equations for closed LNL.

$$\frac{\Psi \rhd X \quad \Psi \rhd Y}{\Psi \rhd (X \times Y)} \qquad \frac{\Psi \blacktriangleright A \quad \Psi \blacktriangleright B}{\Psi \blacktriangleright (A \otimes B)}$$

$$\frac{\Psi \rhd \Gamma \vdash x : X \quad \Psi \rhd \Gamma \vdash y : Y}{\Psi \rhd \Gamma \vdash (x,y) : (X \times Y)} \qquad \frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a : A \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash b : B}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash (a.b) : (A \otimes B)}$$

$$\frac{\Psi \rhd \Gamma \vdash z : (X \times Y)}{\Psi \rhd \Gamma \vdash \mathtt{first}(z) : X} \quad \frac{\Psi \rhd \Gamma \vdash z : (X \times Y)}{\Psi \rhd \Gamma \vdash \mathtt{second}(z) : Y} \qquad \frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash c : (A \otimes B) \quad \Psi \blacktriangleright \Gamma \mid \Pi', \mathrm{a} : A, \mathrm{b} : B \vdash d : D}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash c \ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ d : D}$$

$$\begin{array}{rcl}
\mathtt{first}((x,y)) & \to_\beta & x \\
\mathtt{second}((x,y)) & \to_\beta & y \\
(e.f)\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ d[\mathrm{a},\mathrm{b}] & \to_\beta & d[e,f] \\
z & =_\eta & (\mathtt{first}(z), \mathtt{second}(z)) \\
c & =_\eta & c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ (\mathrm{a}.\mathrm{b}) \\
(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathrm{c}\ \mathtt{and}\ \mathrm{d}\ \mathtt{in}\ e & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathrm{c}\ \mathtt{and}\ \mathrm{d}\ \mathtt{in}\ e) \\
(c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ g)e & \to_{\mathrm{cc}} & c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ (g)e \\
(c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ d)\ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ e & \to_{\mathrm{cc}} & c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ (d\ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ e) \\
(c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ d)\ \mathtt{to}\ \mathrm{e}\ \mathtt{and}\ \mathrm{f}\ \mathtt{in}\ g & \to_{\mathrm{cc}} & c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ (d\ \mathtt{to}\ \mathrm{e}\ \mathtt{and}\ \mathrm{f}\ \mathtt{in}\ g)
\end{array}$$

Figure 7.2: Rules and equations for pairing.

$$\frac{}{\Psi \rhd 1} \qquad \frac{}{\Psi \blacktriangleright \mathrm{I}}$$

$$\frac{\Psi \rhd \Gamma}{\Psi \rhd \Gamma \vdash () : 1} \qquad \frac{\Psi \rhd \Gamma}{\Psi \blacktriangleright \Gamma \vdash * : \mathrm{I}}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash b : \mathrm{I} \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash a : A}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ a : A}$$

$$\begin{array}{rcl}
* \ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ a & \to_\beta & a \\
x & =_\eta & () \\
(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ e & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ e) \\
(c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ d)\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ e & \to_{\mathrm{cc}} & c\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ (d\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ e) \\
(b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ g)e & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ (g)e \\
(b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ e & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ e) \\
(b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathrm{e}\ \mathtt{and}\ \mathrm{f}\ \mathtt{in}\ g & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathrm{e}\ \mathtt{and}\ \mathrm{f}\ \mathtt{in}\ g) \\
(b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ e & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathtt{nix}\ \mathtt{in}\ e)
\end{array}$$

Figure 7.3: Rules and equations for units.

$$\frac{\Psi, \Psi' \rhd X}{\Psi', \Psi \rhd X} \qquad \frac{\Psi, \Psi' \blacktriangleright A}{\Psi', \Psi \blacktriangleright A}$$

$$\frac{\Psi \rhd \Gamma, \Gamma' \vdash x : X}{\Psi \rhd \Gamma', \Gamma \vdash x : X} \quad \frac{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash a : A}{\Psi \blacktriangleright \Gamma \mid \Pi', \Pi \vdash a : A}$$

Figure 7.4: Exchange rules.

$$\frac{\Psi, \mathrm{A} \blacktriangleright B}{\Psi \blacktriangleright \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a : B[\mu \mathrm{A}.B]}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash \mathtt{fold}(a) : \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a : \mu \mathrm{A}.B}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash \mathtt{unfold}(a) : B[\mu \mathrm{A}.B]}$$

$$\begin{array}{rcl}
\mathtt{unfold}(\mathtt{fold}(a)) & \to_\beta & a \\
a & =_\eta & \mathtt{fold}(\mathtt{unfold}(a)) \\
\mathtt{unfold}(b\ \mathtt{to\ x\ in}\ a) & \to_{\mathrm{cc}} & b\ \mathtt{to\ x\ in}\ \mathtt{unfold}(a) \\
\mathtt{unfold}(c\ \mathtt{to\ a\ and\ b\ in}\ d) & \to_{\mathrm{cc}} & c\ \mathtt{to\ a\ and\ b\ in}\ \mathtt{unfold}(d) \\
\mathtt{unfold}(b\ \mathtt{to\ nix\ in}\ a) & \to_{\mathrm{cc}} & b\ \mathtt{to\ nix\ in}\ \mathtt{unfold}(a)
\end{array}$$

Figure 7.5: Rules and equations for recursive linear types.

7.2 Oblique Terms and Types

We analyse the term model of LNL in terms of distributors (see Appendix A). First we observe that substitution over the terms of LNL—and this applies to other mixed calculi—can be read directly as a distributor. The adjoint type constructors L and U are shown to function independently as representations for this term ditributor. The function types of LNL provide these representations of the term distributor with a pair of internalizations that we call the oblique function types.

7.2.1 The Term Distributor

The theory of LNL terms with one free variable forms the total category of a distributor. Objects are type expressions which are classified as nonlinear or linear.

$$\rhd X \qquad \blacktriangleright A$$

Arrows are equivalence classes of derivable terms in singleton context. The equivalence is generated by the equational theory and alpha conversion. Composition is given by well-typed substitution and identities by the variable introduction rules. The arrows are classified as nonlinear, oblique or linear.

$$\rhd \mathrm{x} : X \vdash y[\mathrm{x}] : Y \qquad \blacktriangleright \mathrm{y} : Y \vdash a[\mathrm{y}] : A \qquad \blacktriangleright \mathrm{a} : A \vdash b[\mathrm{a}] : B$$

The nonlinear and linear arrows, together with nonlinear and linear objects, form the term categories C and D as in an ordinary term model of LNL, while the oblique arrows give a distributor between these categories. Oblique arrows cannot be composed with one-another but can be composed with linear and nonlinear arrows to produce new oblique arrows. For example, the arrows given by the three terms above can be composed to produce an oblique arrow from X to B.

$$\blacktriangleright \mathrm{x} : X \vdash b[a[y[\mathrm{x}]]] : B$$

Remark. Barber's Dual Intuitionistic Linear Logic [1] can be translated into the logic of derivable linear sequents in LNL (for details see [24]). The types appearing in the nonlinear part of a DILL context are translated to linear LNL types according to their DILL type structure and then to nonlinear types LNL using U so as to appear in the nonlinear part of an LNL sequent $\Psi \blacktriangleright \mathrm{U}\Gamma \mid \Pi \vdash a : A$. Ordinarily a translation of one type theory into another induces a functor between the corresponding term categories, but here we have a problem with identities. What we actually obtain from our translation is a distributor morphism. This suggests that the distributor structure is more fundamental than either of the categories.

7.2.2 The Adjoint Fragments of LNL

Given a graph F of nonlinear basic types and operations, the term category over the rules

$$\frac{}{\rhd \mathrm{X}}, \qquad \frac{\rhd X \quad \ldots \quad \rhd Y}{\rhd \mathrm{x} : X, \ldots, \mathrm{y} : Y \vdash \mathrm{y} : Y} \quad \text{and} \quad \frac{\rhd \Gamma \vdash x : \mathrm{X}}{\rhd \Gamma \vdash \mathtt{f}(x) : \mathrm{Y}}$$

is the free category on F. This term category only uses the unary forms of the variable introduction rule and the operation rule,

$$\frac{\rhd Y}{\rhd \mathrm{y} : Y \vdash \mathrm{y} : Y} \qquad \frac{\rhd \mathrm{x} : X \vdash x : \mathrm{X}}{\rhd \mathrm{x} : X \vdash \mathtt{f}(x) : \mathrm{Y}}$$

but, if we add the rules for nonlinear pairing, units and function types, we need to use the general forms to obtain the free cartesian closed category on F.

Given a graph G of basic linear types and operations, we could obtain the free category on G using a linear base type rule, the linear variable rule from LNL, and a linear operation rule analogous to the unary rule above. We could then obtain the free symmetric monoidal closed category by adding the rules for linear pairing, units and function types. However, given both basic nonlinear and basic linear operations, we are in a position to add basic operations from basic nonlinear types to basic linear types.

Given a director K from G to F (see Appendix A) and using W for basic linear operations, k for oblique edges from X to W and g for basic linear operations from V to W, the term distributor over the rules

$$\frac{}{\triangleright \mathrm{X}} \qquad \frac{}{\blacktriangleright \mathrm{W}}$$

$$\frac{\triangleright X \quad \ldots \quad \triangleright Y}{\triangleright \mathrm{x} : X, \ldots, \mathrm{y} : Y \vdash \mathrm{y} : Y} \qquad \frac{\blacktriangleright A}{\blacktriangleright \mathrm{a} : A \vdash \mathrm{a} : A}$$

$$\frac{\triangleright \Gamma \vdash x : \mathrm{X}}{\triangleright \Gamma \vdash \mathtt{f}(x) : \mathrm{Y}} \qquad \frac{\triangleright \Gamma \vdash x : \mathrm{X}}{\blacktriangleright \Gamma \vdash \mathtt{k}(x) : \mathrm{W}} \qquad \frac{\blacktriangleright \Gamma \mid \Pi \vdash a : \mathrm{V}}{\blacktriangleright \Gamma \mid \Pi \vdash \mathtt{g}(a) : \mathrm{W}}$$

is the free distributor on K. This free distributor only uses the unary forms of the operation rules,

$$\frac{\triangleright \mathrm{z} : Z \vdash x : \mathrm{X}}{\triangleright \mathrm{z} : Z \vdash \mathtt{f}(x) : \mathrm{Y}} \qquad \frac{\triangleright \mathrm{z} : Z \vdash x : \mathrm{X}}{\blacktriangleright \mathrm{z} : Z \vdash \mathtt{k}(x) : \mathrm{W}} \qquad \frac{\blacktriangleright \mathrm{z} : Z \vdash a : \mathrm{V}}{\blacktriangleright \mathrm{z} : Z \vdash \mathtt{g}(a) : \mathrm{W}} \qquad \frac{\blacktriangleright \mathrm{c} : C \vdash a : \mathrm{V}}{\blacktriangleright \mathrm{c} : C \vdash \mathtt{g}(a) : \mathrm{W}}$$

but, again, we need the general forms if we want to obtain the free distributor between categories with more structure.

When the rules and equations for U are added, the term distributor P becomes the free distributor on K with an aft representation by a functor $U : D \to C$ (see Section A.5). In particular, P is isomorphic to $C(-, U(+))$ and every (oblique) morphism from K to a distributor of the form $A(-, F(+))$, factors uniquely via the canonical (oblique) morphism from K to P.

The term category D is the free category on G, whereas C is the free category on F with a copy of D glued via P, which is to say, C is the total category of P.

On the other hand, when the rules and equations for L are added, the term distributor P becomes the free distributor with a fore representation by a functor $L : C \to D$, in which case P is isomorphic to $D(L(-), +)$. The term category C is the free category on F, while D is a copy of C glued via P to the free category on G, which is to say, D is the total category of P.

Proposition 7.2.3 *The term distributor P is isomorphic to* $C(-, U(+))$ *over LNL theories including the rules and equations for* U *and is isomorphic to* $C(L(-), +)$ *over LNL theories including the rules and equations for* L.

Proof. For each object (X, B) in $C^{\mathrm{op}} \times D$, we require the components of natural isomorphims from $C(-, U(+))$ and $D(L(-), +)$ to $P(-, +)$. Composition with any natural oblique transformation from U to Id_D gives a natural transformation from $C(-, U(+))$ to $P(-, \mathrm{Id}(+))$. Likewise, composition with a transformation from Id_C to L gives a transformation from $D(L(-), +)$

to $P(\mathrm{Id}(-),+)$. Natural transformations are thus given by Composition with the arrows given by the sequents

$$\blacktriangleright \mathrm{x} : \mathrm{U}B \vdash \mathtt{force}(\mathrm{x}) : B \quad \text{and} \quad \blacktriangleright \mathrm{x} : X \vdash \mathtt{produce}(\mathrm{x}) : \mathrm{L}X$$

which, by the substitution lemma, form an oblique transformation, give us one half of the isomorphism.

$$C(X,UB) \underset{(\mathrm{id},\mathtt{thunk}(\mathrm{b}))}{\overset{(\mathrm{id},\mathtt{force}(\mathrm{x}))}{\rightleftarrows}} P(X,B) \underset{(\mathtt{produce}(\mathrm{x}),\mathrm{id})}{\overset{(\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathrm{b},\mathrm{id})}{\rightleftarrows}} D(LX,B)$$

Inverses are given by substitution for b in the raw terms '$\mathtt{thunk}(\mathrm{b})$' and 'a `to` x `in` b', where x is the free variable in the term giving an arrow in $P(X,B)$. Although these terms are not in the type theory, and hence do not give arrows, substitution into them still produces natural transformations and the equations that make them inverses to the representable natural transformations above are instances of the beta and eta laws for L and U. □

If the rules for both U and L are added, P is the free distributor with both such representations, in which case we have an adjunction $L \vdash U$.

Corollary 7.2.4 *Over theories including the rules and equations for both* U *and* L*, the term functor L is left adjoint to the term functor U.*

The rules for U and L interact to produce ω copies of both free categories in both term categories and, generally, neither L nor U has a retract.

For Benton's LNL, we take K to be the degenerate director given by F with no oblique edges and empty G. The free distributor on K is then just the empty distributor from the free category C on F to the empty category. The free distributor of the form $C(-,U(+))$ has U the empty functor into C. On the other hand, the free distributor of the form $D(L(-),+)$ has D isomorphic, via L, to C, the free category on F.

The free distributor with both representations, however, is less degenerate. The category C contains ω copies of the free category on F and, while L still gives an isomorphism between C and D (assuming no other linear type constructors), U is a proper inclusion.

Generally however, we see no reason not to use a director of basic operations. We will write LNL(K) for the theory generated by the rules of LNL together with a rule for each edge in K.

7.2.5 The Oblique Function Types

In the full equational theory of closed LNL, there is a correspondence between terms (in context) of nonlinear type $(X \multimap \mathrm{U}B)$ and terms (in context) of linear type $(\mathrm{L}X \multimap B)$ *with no free variables of linear type*. The nonlinear-to-linear direction can be expressed using substitution on a free nonlinear variable.

$$\dfrac{\dfrac{\triangleright \mathrm{f} : (X \multimap \mathrm{U}B), \mathrm{x} : X \vdash \mathrm{f} : (X \multimap \mathrm{U}B) \quad \triangleright \mathrm{f} : (X \multimap \mathrm{U}B), \mathrm{x} : X \vdash \mathrm{x} : X}{\triangleright \mathrm{f} : (X \multimap \mathrm{U}B), \mathrm{x} : X \vdash (\mathrm{f})\mathrm{x} : \mathrm{U}B}}{\blacktriangleright \mathrm{f} : (X \multimap \mathrm{U}B), \mathrm{x} : X \vdash \mathtt{force}((\mathrm{f})\mathrm{x}) : B}$$

$$\dfrac{\dfrac{\blacktriangleright \mathrm{f} : (X \multimap \mathrm{U}B), \mathrm{x} : X \vdash \mathtt{force}((\mathrm{f})\mathrm{x}) : B \quad \blacktriangleright \mathrm{f} : (X \multimap \mathrm{U}B) \mid \mathrm{b} : \mathrm{L}X \vdash \mathrm{b} : \mathrm{L}X}{\blacktriangleright \mathrm{f} : (X \multimap \mathrm{U}B) \mid \mathrm{b} : \mathrm{L}X \vdash \mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((\mathrm{f})\mathrm{x}) : B}}{\blacktriangleright \mathrm{f} : (X \multimap \mathrm{U}B) \vdash \underbrace{\lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((\mathrm{f})\mathrm{x})}_{\mathrm{lin}[\mathrm{f}]} : (\mathrm{L}X \multimap B)}$$

In the other direction, we use substitution on a linear *meta*-variable g because terms of nonlinear type cannot have free linear variables. The following typing requires that g contain no free linear variables: the introduction rule for U requires a purely nonlinear context.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{derivation weakened with } \mathrm{x} : X}{\blacktriangleright\, \Gamma, \mathrm{x} : X \vdash g : (\mathrm{L}X \multimap B)} \qquad \dfrac{\triangleright\, \Gamma, \mathrm{x} : X \vdash \mathrm{x} : X}{\blacktriangleright\, \Gamma, \mathrm{x} : X \vdash \mathtt{produce}(\mathrm{x}) : \mathrm{L}X}}{\blacktriangleright\, \Gamma, \mathrm{x} : X \vdash (g)\mathtt{produce}(\mathrm{x}) : B}}{\triangleright\, \Gamma, \mathrm{x} : X \vdash \mathtt{thunk}((g)\mathtt{produce}(\mathrm{x})) : \mathrm{U}B}}{\triangleright\, \Gamma \vdash \underbrace{\lambda \mathrm{x}.\mathtt{thunk}((g)\mathtt{produce}(\mathrm{x}))}_{\mathsf{non}[g]} : (X \mathrel{-\!\!\triangleright} \mathrm{U}B)}$$

Given any nonlinear term (in context) f of type $(X \mathrel{-\!\!\triangleright} \mathrm{U}B)$, we have

$$\begin{aligned}
\mathsf{non}[\mathsf{lin}[f]] &=_{\mathrm{def}} \lambda \mathrm{x}.\mathtt{thunk}((\mathsf{lin}[f])\mathtt{produce}(\mathrm{x})) \\
&=_{\mathrm{def}} \lambda \mathrm{x}.\mathtt{thunk}((\lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((f)\mathrm{x}))\mathtt{produce}(\mathrm{x})) \\
&=_{\beta} \lambda \mathrm{x}.\mathtt{thunk}(\mathtt{produce}(\mathrm{x})\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((f)\mathrm{x})) \\
&=_{\beta} \lambda \mathrm{x}.\mathtt{thunk}(\mathtt{force}((f)\mathrm{x})) \\
&=_{\eta} \lambda \mathrm{x}.(f)\mathrm{x} \\
&=_{\eta} f.
\end{aligned}$$

The other direction is less straightforward. Given any linear term (in context) g of type $(\mathrm{L}X \multimap B)$, we have

$$\begin{aligned}
\mathsf{lin}[\mathsf{non}[g]] &=_{\mathrm{def}} \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((\mathsf{non}[g])\mathrm{x}) \\
&=_{\mathrm{def}} \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((\lambda \mathrm{x}.\mathtt{thunk}((g)\mathtt{produce}(\mathrm{x})))\mathrm{x}) \\
&=_{\beta} \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}(\mathtt{thunk}((g)\mathtt{produce}(\mathrm{x}))) \\
&=_{\beta} \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (g)\mathtt{produce}(\mathrm{x}).
\end{aligned}$$

Here we make use of the *application law*

$$(g)(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a) \quad =_{\mathrm{a}} \quad b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (g)a$$

with b for b and $\mathtt{produce}(\mathrm{x})$ for a.

$$\begin{aligned}
&=_{\mathrm{n}} \lambda \mathrm{b}.(g)(\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{produce}(\mathrm{x})) \\
&=_{\eta} \lambda \mathrm{b}.(g)\mathrm{b} \\
&=_{\eta} g.
\end{aligned}$$

The nonlinear-to-linear direction can be abstracted to a linear combinator

$$\blacktriangleright \vdash \underbrace{\lambda \mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{lin}[\mathrm{f}]}_{\mathsf{lin}_{\mathrm{lin}}} : (\mathrm{L}(X \mathrel{-\!\!\triangleright} \mathrm{U}B) \multimap (\mathrm{L}X \multimap B))$$

or to a nonlinear combinator

$$\blacktriangleright \vdash \underbrace{\lambda \mathrm{f}.\mathtt{thunk}(\mathsf{lin}[\mathrm{f}])}_{\mathsf{lin}_{\mathrm{non}}} : ((X \mathrel{-\!\!\triangleright} \mathrm{U}B) \mathrel{-\!\!\triangleright} \mathrm{U}(\mathrm{L}X \multimap B)),$$

and we have

$$\begin{array}{rll} \mathsf{non}[\mathsf{lin}_{\mathsf{lin}}] & =_{\mathrm{def}} & \lambda \mathrm{x}.\mathtt{thunk}((\mathsf{lin}_{\mathsf{lin}})\mathtt{produce}(\mathrm{x})) \\ & =_{\mathrm{def}} & \lambda \mathrm{x}.\mathtt{thunk}((\lambda \mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{lin}[\mathrm{f}])\mathtt{produce}(\mathrm{x})) \\ & =_{\beta} & \lambda \mathrm{x}.\mathtt{thunk}(\mathtt{produce}(\mathrm{x})\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{lin}[\mathrm{f}]) \\ & =_{\beta} & \lambda \mathrm{x}.\mathtt{thunk}(\mathsf{lin}[\mathrm{x}]) \\ & =_{\mathrm{def}} & \mathsf{lin}_{\mathsf{non}} \end{array}$$

and

$$\begin{array}{rll} \mathsf{lin}[\mathsf{lin}_{\mathsf{non}}] & =_{\mathrm{def}} & \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}((\mathsf{lin}_{\mathsf{non}})\mathrm{x}) \\ & =_{\beta} & \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}(\mathtt{thunk}(\mathsf{lin}[\mathrm{x}])) \\ & =_{\beta} & \lambda \mathrm{b}.\mathrm{b}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{lin}[\mathrm{x}] \\ & =_{\mathrm{def}} & \mathsf{lin}_{\mathsf{lin}}. \end{array}$$

The nonlinear combinator has an inverse

$$\blacktriangleright \vdash \underbrace{\lambda \mathrm{y}.\mathsf{non}[\mathtt{force}(\mathrm{y})]}_{\mathsf{non}_{\mathsf{non}}} : (\mathrm{U}(\mathrm{L}X \multimap B) \rightarrowtriangle (X \rightarrowtriangle \mathrm{U}B)).$$

We have

$$\begin{array}{rll} \lambda \mathrm{f}.(\mathsf{non}_{\mathsf{non}})(\mathsf{lin}_{\mathsf{non}})\mathrm{f} & =_{\mathrm{def}} & \lambda \mathrm{f}.(\lambda \mathrm{y}.\mathsf{non}[\mathtt{force}(\mathrm{y})])(\mathsf{lin}_{\mathsf{non}})\mathrm{f} \\ & =_{\beta} & \lambda \mathrm{f}.\mathsf{non}[\mathtt{force}((\mathsf{lin}_{\mathsf{non}})\mathrm{f})] \\ & =_{\mathrm{def}} & \lambda \mathrm{f}.\mathsf{non}[\mathtt{force}((\lambda \mathrm{f}.\mathtt{thunk}(\mathsf{lin}[\mathrm{f}]))\mathrm{f})] \\ & =_{\beta} & \lambda \mathrm{f}.\mathsf{non}[\mathtt{force}(\mathtt{thunk}(\mathsf{lin}[\mathrm{f}]))] \\ & =_{\beta} & \lambda \mathrm{f}.\mathsf{non}[\mathsf{lin}[\mathrm{f}]] \\ & = & \lambda \mathrm{f}.\mathrm{f} \end{array}$$

$$\begin{array}{rll} \lambda \mathrm{y}.(\mathsf{lin}_{\mathsf{non}})(\mathsf{non}_{\mathsf{non}})\mathrm{y} & =_{\mathrm{def}} & \lambda \mathrm{y}.(\lambda \mathrm{f}.\mathtt{thunk}(\mathsf{lin}[\mathrm{f}]))(\mathsf{non}_{\mathsf{non}})\mathrm{y} \\ & =_{\beta} & \lambda \mathrm{y}.\mathtt{thunk}(\mathsf{lin}[(\mathsf{non}_{\mathsf{non}})\mathrm{y}]) \\ & =_{\mathrm{def}} & \lambda \mathrm{y}.\mathtt{thunk}(\mathsf{lin}[(\lambda \mathrm{y}.\mathsf{non}[\mathtt{force}(\mathrm{y})])\mathrm{y}]) \\ & =_{\beta} & \lambda \mathrm{y}.\mathtt{thunk}(\mathsf{lin}[\mathsf{non}[\mathtt{force}(\mathrm{y})]]) \\ & = & \lambda \mathrm{y}.\mathtt{thunk}(\mathtt{force}(\mathrm{y})) \\ & =_{\beta} & \lambda \mathrm{y}.\mathrm{y} \end{array}$$

So, internally, the type $(\mathrm{X} \rightarrowtriangle \mathrm{UB})$ is naturally isomorphic to the type $\mathrm{U}(\mathrm{LX} \multimap \mathrm{B})$. We refer to the 'type' represented by $(X \rightarrowtriangle \mathrm{U}B)$ and $(\mathrm{L}X \multimap B)$ as the *oblique function type* on X and B.

Chapter 8

Recursive Types for Fixed Point Combinators

A significant feature of the untyped lambda calculus are the recursive combinators. These include divergent combinators which reduce to themselves and fixed point combinators Y with the property that $(Y)f$ reduces to $(f)(Y)f$. Fixed point combinators are also known as 'paradoxical combinators' because they provide a semantics for logical paradoxes. For example, the meaning of the statement 'this statement is false' would be the fixed point of the operation that swaps 'true' with 'false'.

In the untyped calculus, these combinators are part of the base theory and are inherited by any extension and/or refinement that respects the beta-law. The simply typed lambda calculus, however, has models without fixed points and so fixed point combinators are excluded from the base theory. If we require a fixed point combinator of type $(\tau \to \tau) \to \tau$ in a simply typed calculus, we can add a constant $\mathtt{Y}_\tau$ together with an equation $f(\mathtt{Y}_\tau(f)) =_\tau \mathtt{Y}_\tau(f)$. The equation says $\mathtt{Y}_\tau$ must be interpreted as a fixed point operator but doesn't say which (and some models have more than one fixed point operator).

Recursive combinators may reappear in the base theory when recursive types are added. The laws governing the contents of certain recursive types require that certain terms satisfy the fixed point equation. We derive some of these in Section 8.1. In Chapter 9 we use fixed point combinators to construct recursive maps into and out of recursive types in term models of RLNL.

Looking at the derivation of recursive combinators in RLNL we notice certain idioms involving oblique function types. This suggests a calculus with an oblique function type which would be interpreted directly by a costructural action. This, in turn, suggests we replace L and U with type constructors for the structural action and exponential on D which we view as parameterized versions of the adjoint types. We sketch the resulting type theory in Section 8.2.

8.1 Recursive Combinators in Recursive Types

The following three paragraphs set out the somewhat stylized treatments of divergence, the Curry fixed point scheme and the Turing fixed point combinator which we lift to FPC in Section 8.1.1 and to RLNL in Section 8.1.2. All three are based on the same self-application scheme: in the untyped lambda calculus we are free to apply any term x to itself.

$$\mathsf{D}[x] = (x)x$$

Divergence. Abstracting, we obtain a self-application combinator which is applied to itself to obtain a divergent combinator.

$$\begin{aligned} \mathsf{X} &= \lambda \mathrm{x}.\mathsf{D}[\mathrm{x}] \\ \Omega &= (\mathsf{X})\mathsf{X} \end{aligned}$$

The term Ω is of the form $\mathsf{D}[x]$ and beta-reduces to itself in one step.

$$\begin{aligned} \Omega &=_{\mathrm{def}} \\ (\mathsf{X})\mathsf{X} &=_{\mathrm{def}} \\ (\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])\mathsf{X} &\to_{\beta} \mathsf{D}[\mathsf{X}] \\ &=_{\mathrm{def}} (\mathsf{X})\mathsf{X} \\ &{}_{\mathrm{def}}= \Omega \end{aligned}$$

Operationally, we have an infinite loop and this can be used to iterate the application of some term f. Equationally, the result of such iteration gives a fixed point of f.

Curry. If we apply a term f to the self-application scheme before abstracting, we obtain a fixed point for f.

$$\begin{aligned} \mathsf{X}'[f] &= \lambda \mathrm{x}.(f)\mathsf{D}[\mathrm{x}] \\ \mathsf{C}[f] &= (\mathsf{X}'[f])\mathsf{X}'[f] \end{aligned}$$

The term $\mathsf{C}[f]$ is of the form $\mathsf{D}[x]$ and, assuming x is not free in f, reduces to $(f)\mathsf{C}[f]$ in one step.

$$\begin{aligned} \mathsf{C}[f] &=_{\mathrm{def}} (\mathsf{X}'[f])\mathsf{X}'[f] \\ &=_{\mathrm{def}} \\ (\lambda \mathrm{x}.(f)\mathsf{D}[\mathrm{x}])\mathsf{X}'[f] &\to_{\beta} (f)\mathsf{D}[\mathsf{X}'[f]] \\ &=_{\mathrm{def}} (f)(\mathsf{X}'[f])\mathsf{X}'[f] \\ &{}_{\mathrm{def}}= (f)\mathsf{C}[f] \end{aligned}$$

Abstracting on this scheme, we obtain a fixed point combinator.

$$\mathsf{Y} = \lambda \mathrm{f}.\mathsf{C}[\mathrm{f}]$$

Given some term f, the term $(\mathsf{Y})f$ beta-reduces to $\mathsf{C}[f]$, which beta-reduces to $(f)\mathsf{C}[f]$, which beta-expands to $(f)(\mathsf{Y})f$. Note that this reasoning uses beta-expansion in the argument of f.

Turing. By first applying the self-application scheme to f and then applying f, we can abstract on f before abstracting the self-application to obtain a fixed point combinator of the form $\mathsf{D}[x]$.

$$\begin{aligned} \mathsf{X}'' &= \lambda \mathrm{x}.\lambda \mathrm{f}.(\mathrm{f})(\mathsf{D}[\mathrm{x}])\mathrm{f} \\ \mathsf{Z} &= (\mathsf{X}'')\mathsf{X}'' \end{aligned}$$

Given some term f, the term $(\mathsf{Z})f$ beta-reduces directly to $(f)(\mathsf{Z})f$.

$$\begin{aligned} (\mathsf{Z})f &=_{\mathrm{def}} \\ ((\mathsf{X}'')\mathsf{X}'')f &=_{\mathrm{def}} \\ ((\lambda \mathrm{x}.\lambda \mathrm{f}.(\mathrm{f})(\mathsf{D}[\mathrm{x}])\mathrm{f})\mathsf{X}'')f &=_{\beta} \\ (\lambda \mathrm{f}.(\mathrm{f})(\mathsf{D}[\mathsf{X}''])\mathrm{f})f &\to_{\beta} (f)(\mathsf{D}[\mathsf{X}''])f \\ &=_{\mathrm{def}} (f)((\mathsf{X}'')\mathsf{X}'')f \\ &{}_{\mathrm{def}}= (f)(\mathsf{Z})f \end{aligned}$$

Note that the reduction requires two steps. The Turing combinator trades the beta-expansion required to reason about the Curry combinator for an extra evaluation step.

First we lift our untyped combinators to a fragment of the metalanguage FPC. Then we lift these to a fragment of our type theory RLNL. The fragments we use have function spaces and recursive types. Although our primary concern is the existence of a fixed point combinator in the equational theory of RLNL, we avoid reductions in the arguments of applications as these raise operational questions.

8.1.1 Recursive Combinators in FPC

We use the fragment of FPC shown in Figure 8.1. This has a function space $(\multimap)$ and a recursion operator μ that binds one type variable. Any type may be substituted for a type variable. We use the equivalences generated by the beta-reductions shown.

In FPC, every type T contains a divergent combinator and every type of the form $((T \multimap T) \multimap T)$ contains a fixed point combinator. The recursive type $\mu \mathrm{S}.(\mathrm{S} \multimap T)$ allows us to lift the divergent combinator Ω to T and the Curry fixed point $\mathsf{C}[f]$ to $((T \multimap T) \multimap T)$. The recursive type $\mu \mathrm{S}.(\mathrm{S} \multimap ((T \multimap T) \multimap T))$ allows us to lift the Turing combinator to $((T \multimap T) \multimap T)$.

Divergence. Let T be a type variable. We can apply terms of type $M = \mu \mathrm{S}.(\mathrm{S} \multimap \mathrm{T})$ to themselves by unfolding one copy to the type $(M \multimap \mathrm{T})$.

$$\dfrac{\dfrac{\dfrac{}{\mathrm{S}, \mathrm{T} \square \mathrm{T}}}{\mathrm{T}, \mathrm{S} \square \mathrm{T}}}{\dfrac{\dfrac{}{\mathrm{T}, \mathrm{S} \square \mathrm{S}} \quad \mathrm{T}, \mathrm{S} \square \mathrm{T}}{\mathrm{T}, \mathrm{S} \square (\mathrm{S} \multimap \mathrm{T})}}{\mathrm{T} \square \underbrace{\mu \mathrm{S}.(\mathrm{S} \multimap \mathrm{T})}_{M}}}$$

$$\dfrac{\mathrm{T} \square M \qquad \dfrac{\dfrac{\mathrm{T} \square M}{\mathrm{T} \square \mathrm{x} : M \vdash \mathrm{x} : M}}{\mathrm{T} \square \mathrm{x} : M \vdash \mathtt{unfold}(\mathrm{x}) : (M \multimap \mathrm{T})}}{\mathrm{T} \square \mathrm{x} : M \vdash \underbrace{(\mathtt{unfold}(\mathrm{x}))\mathrm{x}}_{\mathsf{D}[\mathrm{x}]} : \mathrm{T}}$$

Abstracting, we obtain a combinator of type $(M \multimap \mathrm{T})$.

$$\dfrac{\mathrm{T} \square \mathrm{x} : M \vdash \mathsf{D}[\mathrm{x}] : \mathrm{T}}{\mathrm{T} \square \vdash \underbrace{\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}]}_{\mathsf{X}_0} : (M \multimap \mathrm{T})}$$

We can apply this to itself by folding one copy into type M.

$$\dfrac{\dfrac{\mathrm{T} \square \vdash \mathsf{X}_0 : (M \multimap \mathrm{T})}{\mathrm{T} \square \vdash \mathtt{fold}(\mathsf{X}_0) : M} \qquad \mathrm{T} \square \vdash \mathsf{X}_0 : (M \multimap \mathrm{T})}{\mathrm{T} \square \vdash \underbrace{(\mathsf{X}_0)\mathtt{fold}(\mathsf{X}_0)}_{\Omega_0} : \mathrm{T}}$$

This term beta-reduces to itself in two steps.

$$\begin{array}{rll}
\Omega_0 & =_{\text{def}} & \\
(\mathsf{X}_0)\mathtt{fold}(\mathsf{X}_0) & =_{\text{def}} & \\
(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])\mathtt{fold}(\mathsf{X}_0) & \to_\beta & \mathsf{D}[\mathtt{fold}(\mathsf{X}_0)] \\
& =_{\text{def}} & (\mathtt{unfold}(\mathtt{fold}(\mathsf{X}_0)))\mathtt{fold}(\mathsf{X}_0) \\
& =_\beta & (\mathsf{X}_0)\mathtt{fold}(\mathsf{X}_0) \\
& {}_{\text{def}}= & \Omega_0
\end{array}$$

The term Ω_0 is not of the form $\mathsf{D}[x]$. The self-application combinator X_0 acts on terms of type M while the self-application of self-application in Ω_0 is performed on a term of type $(M \rightharpoonup \mathrm{T})$. Note however that Ω_0 reduces to $\mathsf{D}[\mathtt{fold}(\mathsf{X}_0)]$ which is of the form $\mathsf{D}[x]$.

$$\begin{array}{rcl}
\mathsf{X} & = & \mathtt{fold}(\mathsf{X}_0) \\
\Omega & = & (\mathtt{unfold}(\mathsf{X}))\mathsf{X}
\end{array}$$

Now Ω is of the form $\mathsf{D}[x]$. and reduces to itself.

$$\begin{array}{rll}
\Omega & =_{\text{def}} & \\
(\mathtt{unfold}(\mathsf{X}))\mathsf{X} & =_{\text{def}} & \\
(\mathtt{unfold}(\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])))\mathsf{X} & =_\beta & \\
(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])\mathsf{X} & \to_\beta & \mathsf{D}[\mathsf{X}] \\
& =_{\text{def}} & (\mathtt{unfold}(\mathsf{X}))\mathsf{X} \\
& {}_{\text{def}}= & \Omega
\end{array}$$

Here is the cycle in terms of the self-application scheme.

$$\begin{array}{rlcrlcl}
\Omega & =_\beta & \Omega_0 & \to_\beta & \Omega & =_\beta & \cdots \\
\mathsf{D}[\mathsf{X}] & =_\beta & (\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])\mathsf{X} & \to_\beta & \mathsf{D}[\mathsf{X}] & =_\beta & \cdots \\
\mathsf{D}[\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])] & =_\beta & (\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}]) & \to_\beta & \mathsf{D}[\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])] & =_\beta & \cdots
\end{array}$$

Here is the cycle with expanded definitions.

$$\begin{array}{rlcrlcl}
\Omega & =_\beta & \Omega_0 & \to_\beta & \cdots & & \\
(\mathtt{unfold}(\mathsf{X}))\mathsf{X} & =_\beta & (\mathsf{X}_0)\mathtt{fold}(\mathsf{X}_0) & \to_\beta & \cdots & & \\
(\mathtt{unfold}(\mathtt{fold}(\mathsf{X}_0)))\mathtt{fold}(\mathsf{X}_0) & =_\beta & (\mathsf{X}_0)\mathtt{fold}(\mathsf{X}_0) & \to_\beta & \cdots & & \\
(\mathtt{unfold}(\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])))\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}]) & =_\beta & (\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}])\mathtt{fold}(\lambda \mathrm{x}.\mathsf{D}[\mathrm{x}]) & \to_\beta & \cdots & & \\
(\mathtt{unfold}(\mathtt{fold}(\lambda \mathrm{x}.(\mathtt{unfold}(\mathrm{x}))\mathrm{x})))\mathtt{fold}(\lambda \mathrm{x}.(\mathtt{unfold}(\mathrm{x}))\mathrm{x}) & =_\beta & (\lambda \mathrm{x}.(\mathtt{unfold}(\mathrm{x}))\mathrm{x})\mathtt{fold}(\lambda \mathrm{x}.(\mathtt{unfold}(\mathrm{x}))\mathrm{x}) & \to_\beta & \cdots & &
\end{array}$$

Erasing $\mathtt{fold}()$ and $\mathtt{unfold}()$, both Ω_0 and Ω become the untyped Ω and beta-equivalence becomes an identity.

Curry. Let f be a term variable of type $(\mathrm{T} \rightharpoonup \mathrm{T})$. First we weaken the context in our derivation for self-application to include f.

$$\dfrac{\dfrac{\dfrac{\overline{\mathrm{T} \sqcap \mathrm{T}} \quad \overline{\mathrm{T} \sqcap \mathrm{T}}}{\mathrm{T} \sqcap (\mathrm{T} \rightharpoonup \mathrm{T})} \quad \mathrm{T} \sqcap M}{\mathrm{T} \sqcap \mathrm{f} : (\mathrm{T} \rightharpoonup \mathrm{T}), \mathrm{x} : M \vdash \mathrm{x} : M} \quad \dfrac{\dfrac{\overline{\mathrm{T} \sqcap \mathrm{T}} \quad \overline{\mathrm{T} \sqcap \mathrm{T}}}{\mathrm{T} \sqcap (\mathrm{T} \rightharpoonup \mathrm{T})} \quad \mathrm{T} \sqcap M}{\mathrm{T} \sqcap \mathrm{f} : (\mathrm{T} \rightharpoonup \mathrm{T}), \mathrm{x} : M \vdash \mathtt{unfold}(\mathrm{x}) : (M \rightharpoonup \mathrm{T})}}{\mathrm{T} \sqcap \mathrm{f} : (\mathrm{T} \rightharpoonup \mathrm{T}), \mathrm{x} : M \vdash \underbrace{(\mathtt{unfold}(\mathrm{x}))\mathrm{x}}_{\mathsf{D}[\mathrm{x}]} : \mathrm{T}}$$

As in the untyped X', we apply f to the result of self-application before abstracting on x.

$$\dfrac{\dfrac{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M\vdash \mathsf{D}[\mathrm{x}]:\mathrm{T} \qquad \dfrac{\mathrm{T}\square M \qquad \dfrac{\overline{\mathrm{T}\square\mathrm{T}} \quad \overline{\mathrm{T}\square\mathrm{T}}}{\mathrm{T}\square(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square\mathrm{x}:M,\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M\vdash\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M\vdash(\mathrm{f})\mathsf{D}[\mathrm{x}]:\mathrm{T}}$$
$$\dfrac{}{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\underbrace{\lambda\mathrm{x}.(\mathrm{f})\mathsf{D}[\mathrm{x}]}_{\mathsf{X}'_0[\mathrm{f}]}:(M\rightharpoonup\mathrm{T})}$$

We can apply this term to itself by folding one copy into type M.

$$\dfrac{\dfrac{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\mathsf{X}'_0[\mathrm{f}]:(M\rightharpoonup\mathrm{T})}{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\mathtt{fold}(\mathsf{X}'_0[\mathrm{f}]):M} \qquad \mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\mathsf{X}'_0[\mathrm{f}]:(M\rightharpoonup\mathrm{T})}{\mathrm{T}\square\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\underbrace{(\mathsf{X}'_0[\mathrm{f}])\mathtt{fold}(\mathsf{X}'_0[\mathrm{f}])}_{\mathsf{C}_0[\mathrm{f}]}:\mathrm{T}}$$

Now, given any term f of type $(\mathrm{T}\rightharpoonup\mathrm{T})$, $\mathsf{C}_0[f]$ is beta-equivalent to $(f)\mathsf{C}_0[f]$.

$$\begin{aligned}
\mathsf{C}_0[f] &=_{\mathrm{def}} \\
(\mathsf{X}'_0[f])\mathtt{fold}(\mathsf{X}'_0[f]) &=_{\mathrm{def}} \\
(\lambda\mathrm{x}.(f)\mathsf{D}[\mathrm{x}])\mathtt{fold}(\mathsf{X}'_0[f]) &\to_\beta (f)\mathsf{D}[\mathtt{fold}(\mathsf{X}'_0[f])] \\
&=_{\mathrm{def}} (f)(\mathtt{unfold}(\mathtt{fold}(\mathsf{X}'_0[f])))\mathtt{fold}(\mathsf{X}'_0[f]) \\
&=_\beta (f)(\mathsf{X}'_0[f])\mathtt{fold}(\mathsf{X}'_0[f]) \\
&\ {}_{\mathrm{def}}{=} (f)\mathsf{C}_0[f]
\end{aligned}$$

Erasing f in $\mathsf{C}_0[f]$ gives Ω_0. We can avoid the beta-reduction in the argument of f by using $\mathsf{D}[\mathtt{fold}(\mathsf{X}'_0[f])]$ in place of $\mathsf{C}_0[f]$.

$$\begin{aligned}
\mathsf{X}'[\mathrm{f}] &= \mathtt{fold}(\mathsf{X}'_0[\mathrm{f}]) \\
\mathsf{C}[\mathrm{f}] &= (\mathtt{unfold}(\mathsf{X}'[\mathrm{f}]))\mathsf{X}'[\mathrm{f}]
\end{aligned}$$

The term $\mathsf{C}[f]$ is of the form $\mathsf{D}[x]$ and erasing f gives Ω. Without the f, the terms Ω and Ω_0 beta-reduce to each another, but here the rearrangement reorders the two beta-reductions with respect to the application of f.

$$\begin{aligned}
\mathsf{C}[f] &=_{\mathrm{def}} \\
(\mathtt{unfold}(\mathsf{X}'[f]))\mathsf{X}'[f] &=_{\mathrm{def}} \\
(\mathtt{unfold}(\mathtt{fold}(\lambda\mathrm{x}.(f)\mathsf{D}[\mathrm{x}])))\mathsf{X}'[f] &=_\beta \\
(\lambda\mathrm{x}.(f)\mathsf{D}[\mathrm{x}])\mathsf{X}'[f] &\to_\beta (f)\mathsf{D}[\mathsf{X}'[f]] \\
&=_{\mathrm{def}} (f)(\mathtt{unfold}(\mathsf{X}'[f]))\mathsf{X}'[f] \\
&\ {}_{\mathrm{def}}{=} (f)\mathsf{C}[f]
\end{aligned}$$

Abstracting, we obtain a combinator of type $((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})$.

$$\mathsf{Y} \;=\; \lambda \mathrm{f}.\mathsf{C}[f]$$

Given some term f of type $(\mathrm{T}\rightharpoonup\mathrm{T})$, the term $(\mathsf{Y})f$ beta-reduces to $\mathsf{C}[f]$, which beta-reduces to $(f)(\mathsf{Y})f$, which beta-expands to $f(\mathsf{Y}(f))$. As in the untyped calculus, this uses beta-expansion in the argument of f.

Turing. To type the Turing combinator, we replace T with $((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})$ in our recursive type M.

$$\dfrac{\dfrac{\dfrac{}{\mathrm{T},\mathrm{S}\sqsubset\mathrm{S}} \qquad \dfrac{\dfrac{\dfrac{}{\mathrm{S},\mathrm{T}\sqsubset\mathrm{T}} \qquad \dfrac{}{\mathrm{S},\mathrm{T}\sqsubset\mathrm{T}}}{\mathrm{S},\mathrm{T}\sqsubset(\mathrm{T}\rightharpoonup\mathrm{T})} \qquad \dfrac{}{\mathrm{S},\mathrm{T}\sqsubset\mathrm{T}}}{\mathrm{S},\mathrm{T}\sqsubset((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})}}{\mathrm{T},\mathrm{S}\sqsubset((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})}}{\mathrm{T},\mathrm{S}\sqsubset(\mathrm{S}\rightharpoonup((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T}))}}{\mathrm{T}\sqsubset\underbrace{\mu\mathrm{S}.(\mathrm{S}\rightharpoonup((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T}))}_{M'}}$$

The result of self-application is now of type $((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{T}\sqsubset\mathrm{T}} \qquad \dfrac{}{\mathrm{T}\sqsubset\mathrm{T}}}{\mathrm{T}\sqsubset(\mathrm{T}\rightharpoonup\mathrm{T})} \qquad \mathrm{T}\sqsubset M'}{\mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash\mathrm{x}:M'} \qquad \dfrac{\dfrac{\dfrac{}{\mathrm{T}\sqsubset\mathrm{T}} \qquad \dfrac{}{\mathrm{T}\sqsubset\mathrm{T}}}{\mathrm{T}\sqsubset(\mathrm{T}\rightharpoonup\mathrm{T})} \qquad \mathrm{T}\sqsubset M'}{\mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash\mathrm{x}:M'}}{\mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash\mathrm{unfold}(\mathrm{x}):(M'\rightharpoonup((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T}))}}{\mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash\underbrace{(\mathrm{unfold}(\mathrm{x}))\mathrm{x}}_{\mathsf{D}'[\mathrm{x}]}:((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})}$$

With type $((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})$, we can apply the result of self-application to our endofunction f.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{T}\sqsubset\mathrm{T}} \qquad \dfrac{}{\mathrm{T}\sqsubset\mathrm{T}}}{\mathrm{T}\sqsubset(\mathrm{T}\rightharpoonup\mathrm{T})} \qquad \mathrm{T}\sqsubset M'}{\mathrm{T}\sqsubset\mathrm{x}:M',\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})\vdash\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})} \qquad \mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash\mathsf{D}'[\mathrm{x}]:((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})}{\mathrm{T}\sqsubset\mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}),\mathrm{x}:M'\vdash(\mathsf{D}'[\mathrm{x}])\mathrm{f}:\mathrm{T}}$$

Now we can proceed as for X', except we abstract on f before we abstract on x.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{T}\square\mathrm{T}} \quad \dfrac{}{\mathrm{T}\square\mathrm{T}}}{\mathrm{T}\square(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square M' \quad \mathrm{T}\square(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square \mathrm{x}:M', \mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}) \vdash \mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square \mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}), \mathrm{x}:M' \vdash (\mathsf{D}'[\mathrm{x}])\mathrm{f}:\mathrm{T} \quad \mathrm{T}\square \mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}), \mathrm{x}:M' \vdash \mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T})}}{\mathrm{T}\square \mathrm{f}:(\mathrm{T}\rightharpoonup\mathrm{T}), \mathrm{x}:M' \vdash (\mathrm{f})(\mathsf{D}'[\mathrm{x}])\mathrm{f}:\mathrm{T}}$$
$$\dfrac{\mathrm{T}\square \mathrm{x}:M' \vdash \lambda\mathrm{f}.(\mathrm{f})(\mathsf{D}'[\mathrm{x}])\mathrm{f}:((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T})}{\mathrm{T}\square \vdash \lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})(\mathsf{D}'[\mathrm{x}])\mathrm{f}:(M'\rightharpoonup((\mathrm{T}\rightharpoonup\mathrm{T})\rightharpoonup\mathrm{T}))}$$
$$\mathrm{T}\square \vdash \underbrace{\mathtt{fold}(\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})(\mathsf{D}'[\mathrm{x}])\mathrm{f})}_{\mathsf{X}''}:M'$$

We apply this combinator to itself by unfolding one copy.

$$\dfrac{\mathrm{T}\square \mathrm{x}:M' \quad \dfrac{\mathrm{T}\square \vdash \mathsf{X}'':M}{\mathrm{T}\square \vdash \mathtt{unfold}(\mathsf{X}''):(M\rightharpoonup\mathrm{T})}}{\mathrm{T}\square \vdash \underbrace{(\mathtt{unfold}(\mathsf{X}''))\mathsf{X}''}_{\mathsf{Z}}:\mathrm{T}}$$

Given any term f of type $(\mathrm{T}\rightharpoonup\mathrm{T})$, $(\mathsf{Z})f$ reduces directly to $(f)(\mathsf{Z})f$.

$$\begin{aligned}
(\mathsf{Z})f &=_{\mathrm{def}} \\
((\mathtt{unfold}(\mathsf{X}''))\mathsf{X}'')f &=_{\mathrm{def}} \\
((\mathtt{unfold}(\mathtt{fold}(\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})(\mathsf{D}'[\mathrm{x}])\mathrm{f})))\mathsf{X}'')f &=_{\beta} \\
((\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})(\mathsf{D}'[\mathrm{x}])\mathrm{f})\mathsf{X}'')f &=_{\beta} \\
(\lambda\mathrm{f}.(\mathrm{f})(\mathsf{D}'[\mathsf{X}''])\mathrm{f})f &\rightarrow_{\beta} (f)(\mathsf{D}'[\mathsf{X}''])f \\
&=_{\mathrm{def}} (f)((\mathtt{unfold}(\mathsf{X}''))\mathsf{X}'')f \\
&{}_{\mathrm{def}}{=} (f)(\mathsf{Z})f
\end{aligned}$$

Erasing the operations $\mathtt{fold}()$ and $\mathtt{unfold}()$ in the FPC combinator

$$\mathsf{Z} = (\mathtt{unfold}(\mathtt{fold}(\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})((\mathtt{unfold}(\mathrm{x}))\mathrm{x})\mathrm{f})))\mathtt{fold}(\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})((\mathtt{unfold}(\mathrm{x}))\mathrm{x})\mathrm{f}),$$

we recover the untyped combinator

$$\mathsf{Z} = (\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})((\mathrm{x})\mathrm{x})\mathrm{f})\lambda\mathrm{x}.\lambda\mathrm{f}.(\mathrm{f})((\mathrm{x})\mathrm{x})\mathrm{f}.$$

8.1.2 Recursive Combinators in RLNL

In our FPC derivations, free variables of recursive type appear twice in some expressions, whereas in RLNL recursive types are linear and so variables of recursive type must appear exactly once. Any typing of these combinators in RLNL will therefore pass through types of the form U*A*. In place of variables of recursive type $\mu\mathrm{x}.B$ we use variables of type $\mathrm{U}\mu\mathrm{x}.B$. This requires that B contain applications of U and L so that we can fold and unfold the terms we build into and out of type $\mu\mathrm{x}.B$.

Divergence. Linearity prevents us from applying a variable of recursive type M directly to itself. We therefore start with a variable x of type UM. The destructor for U gives us a term $\mathtt{force}(x)$ of type M which we can unfold to a function type and apply to some form of x. Suppose M unfolds to type $(M \multimap \mathrm{B})$. We can then apply $\mathtt{unfold}(\mathtt{force}(x))$ to $\mathtt{force}(x)$ to obtain a term of type B. Before we can abstract we must apply the constructor for U to obtain a term of type UB. Now we would like to apply the resulting self-application combinator

$$\mathsf{X}_0 \quad = \quad \lambda \mathrm{x}.\mathtt{thunk}((\mathtt{unfold}(\mathtt{force}(x)))\mathtt{force}(x))$$

of type $(\mathrm{U}M \mathrel{-\!\!\triangleright} \mathrm{UB})$ to itself, but we find there is no way to fold this term into type UM. However, observe that $(\mathrm{U}M \mathrel{-\!\!\triangleright} \mathrm{UB})$ is an oblique function space isomorphic to $\mathrm{U}(\mathrm{LU}M \multimap \mathrm{B})$ which suggests that M should unfold to $(\mathrm{LU}M \multimap \mathrm{B})$. So take $M = \mu \mathrm{x}.(\mathrm{LUx} \multimap \mathrm{B})$. Now $\mathtt{unfold}(\mathtt{force}(x))$ must be applied to a term of type LUM, so, instead of the destructor for U, we apply the constructor for L to our variable x of type UM:

$$\mathsf{D}[\mathrm{x}] \quad = \quad (\mathtt{unfold}(\mathtt{force}(x)))\mathtt{produce}(x)$$

Applying the constructor for U and abstracting, we obtain the combinator

$$\mathsf{X}_1 \quad = \quad \lambda \mathrm{x}.\mathtt{thunk}(\mathsf{D}[\mathrm{x}])$$

of type $(\mathrm{U}M \mathrel{-\!\!\triangleright} \mathrm{UB})$, but now this type is isomorphic to the type UM via $\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\,]))$. and so our combinator can be applied to itself: $\Omega_1 = (\mathsf{X}_1)\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1]))$. This term beta-reduces to $\mathtt{thunk}(\mathsf{D}[\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1]))])$ and $\mathsf{D}[\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1]))]$ beta-reduces to $\mathtt{force}(\Omega_1)$, so Ω_1 is beta-equivalent to $\mathtt{thunk}(\mathtt{force}(\Omega_1))$.

$$\begin{aligned}
\Omega_1 \quad &=_{\mathrm{def}} \quad (\mathsf{X}_1)\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])) \\
&\to_\beta \quad \mathtt{thunk}(\mathsf{D}[\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1]))]) \\
&=_{\mathrm{def}} \quad \mathtt{thunk}((\mathtt{unfold}(\mathtt{force}(\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])))))\mathtt{produce}(\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])))) \\
&=_\beta \quad \mathtt{thunk}((\mathtt{unfold}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])))\mathtt{produce}(\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])))) \\
&=_\beta \quad \mathtt{thunk}((\mathrm{lin}[\mathsf{X}_1])\mathtt{produce}(\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])))) \\
&=_\beta \quad \mathtt{thunk}(\mathtt{produce}(\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1]))) \ \mathtt{to}\ x\ \mathtt{in}\ \mathtt{force}((\mathsf{X}_1)x)) \\
&=_\beta \quad \mathtt{thunk}(\mathtt{force}((\mathsf{X}_1)\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1])))) \\
&=_{\mathrm{def}} \quad \mathtt{thunk}(\mathtt{force}(\Omega_1))
\end{aligned}$$

We can avoid the extra thunk-force, as well as the reduction under a thunk, by using the term $\mathtt{force}(\Omega_1)$ of type B. This term beta-reduces to itself and, along the way, to

$$(\mathrm{lin}[\mathsf{X}_1])\mathtt{produce}(\mathtt{thunk}(\mathtt{fold}(\mathrm{lin}[\mathsf{X}_1]))).$$

The term $\mathrm{lin}[\mathsf{X}_1]$ is beta-equivalent to λa.a $\mathtt{to}$ x $\mathtt{in}$ $\mathsf{D}[\mathrm{x}]$ so, putting

$$\mathsf{X} = \lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}],$$

$\mathtt{force}(\Omega_1)$ is beta-equivalent to

$$\Omega = (\mathsf{X})\mathtt{produce}(\mathtt{thunk}(\mathtt{fold}(\mathsf{X}))).$$

The derivation of this term completely avoids the type $(\mathrm{U}M \multimap \mathrm{U}\mathrm{B})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{B},\mathrm{X} \blacktriangleright \mathrm{X}}}{\mathrm{B},\mathrm{X} \triangleright \mathrm{U}\mathrm{X}} \quad \dfrac{\dfrac{}{\mathrm{X},\mathrm{B} \blacktriangleright \mathrm{B}}}{\mathrm{B},\mathrm{X} \blacktriangleright \mathrm{B}}}{\mathrm{B},\mathrm{X} \blacktriangleright \mathrm{L}\mathrm{U}\mathrm{X}}}{\mathrm{B},\mathrm{X} \blacktriangleright (\mathrm{L}\mathrm{U}\mathrm{X} \multimap \mathrm{B})}}{\mathrm{B} \blacktriangleright \underbrace{\mu\mathrm{X}.(\mathrm{L}\mathrm{U}\mathrm{X} \multimap \mathrm{B})}_{M}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright M}{\mathrm{B} \triangleright \mathrm{U}M}}{\mathrm{B} \triangleright \mathrm{x} : \mathrm{U}M \vdash \mathrm{x} : \mathrm{U}M}}{\mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M \vdash \mathtt{produce}(\mathrm{x}) : \mathrm{L}\mathrm{U}M} \quad \dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright M}{\mathrm{B} \triangleright \mathrm{U}M}}{\mathrm{B} \triangleright \mathrm{x} : \mathrm{U}M \vdash \mathrm{x} : \mathrm{U}M}}{\mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M \vdash \mathtt{force}(\mathrm{x}) : M}}{\mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M \vdash \mathtt{unfold}(\mathtt{force}(\mathrm{x})) : (\mathrm{L}\mathrm{U}M \multimap \mathrm{B})}}{\mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M \vdash \underbrace{(\mathtt{unfold}(\mathtt{force}(\mathrm{x})))\mathtt{produce}(\mathrm{x})}_{\mathsf{D}[\mathrm{x}]} : \mathrm{B}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M \vdash \mathsf{D}[\mathrm{x}] : \mathrm{B} \quad \mathrm{B} \blacktriangleright ; \mathrm{a} : \mathrm{L}\mathrm{U}M \vdash \mathrm{a} : \mathrm{L}\mathrm{U}M}{\mathrm{B} \blacktriangleright ; \mathrm{a} : \mathrm{L}\mathrm{U}M \vdash \mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}] : \mathrm{B}}}{\mathrm{B} \blacktriangleright \vdash \lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}] : (\mathrm{L}\mathrm{U}M \multimap \mathrm{B})}}{\mathrm{B} \blacktriangleright \vdash \mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}]) : M}}{\mathrm{B} \blacktriangleright \vdash \underbrace{\mathtt{thunk}(\mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}]))}_{\mathsf{X}} : \mathrm{U}M}$$

$$\dfrac{\dfrac{\dfrac{\mathrm{B} \triangleright \vdash \mathsf{X} : \mathrm{U}M}{\mathrm{B} \blacktriangleright \vdash \mathtt{produce}(\mathsf{X}) : \mathrm{L}\mathrm{U}M} \quad \dfrac{\dfrac{\mathrm{B} \triangleright \vdash \mathsf{X} : \mathrm{U}M}{\mathrm{B} \blacktriangleright \vdash \mathtt{force}(\mathsf{X}) : M}}{\mathrm{B} \blacktriangleright \vdash \mathtt{unfold}(\mathtt{force}(\mathsf{X})) : (\mathrm{L}\mathrm{U}M \multimap \mathrm{B})}}{\mathrm{B} \blacktriangleright \vdash \underbrace{(\mathtt{unfold}(\mathtt{force}(\mathsf{X})))\mathtt{produce}(\mathsf{X})}_{\Omega} : \mathrm{B}}$$

This term beta-reduces to itself in four steps.

$$\begin{aligned}
\Omega &=_{\mathrm{def}} \\
(\mathtt{unfold}(\mathtt{force}(\mathsf{X})))\mathtt{produce}(\mathsf{X}) &=_{\mathrm{def}} \\
(\mathtt{unfold}(\mathtt{force}(\mathtt{thunk}(\mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}])))))\mathtt{produce}(\mathsf{X}) &=_{\beta} \\
(\mathtt{unfold}(\mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}])))\mathtt{produce}(\mathsf{X}) &=_{\beta} \\
(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}])\mathtt{produce}(\mathsf{X}) &\to_{\beta} \\
\mathtt{produce}(\mathsf{X})\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{D}[\mathrm{x}] &\to_{\beta} \quad \mathsf{D}[\mathsf{X}]
\end{aligned}$$

$$\begin{aligned}
\mathsf{D}[\mathsf{X}] &=_{\mathrm{def}} (\mathtt{unfold}(\mathtt{force}(\mathsf{X})))\mathtt{produce}(\mathsf{X}) \\
&\ {}_{\mathrm{def}}{=} \ \Omega
\end{aligned}$$

Curry. Let f be a variable of type U(LUB⊸B), which is isomorphic to the endofunction type (UB⇸UB).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{B} \blacktriangleright \mathrm{B}}}{\mathrm{B} \triangleright \mathrm{UB}}}{\mathrm{B} \blacktriangleright \mathrm{LUB}} \quad \dfrac{}{\mathrm{B} \blacktriangleright \mathrm{B}}}{\mathrm{B} \blacktriangleright \underbrace{(\mathrm{LUB} \multimap \mathrm{B})}_{E}}$$

$$\dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \mathsf{D}[\mathrm{x}] : \mathrm{B}}{\mathrm{B} \triangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \mathtt{thunk}(\mathsf{D}[\mathrm{x}]) : \mathrm{UB}}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \mathtt{produce}(\mathtt{thunk}(\mathsf{D}[\mathrm{x}])) : \mathrm{LUB}} \quad \dfrac{\dfrac{\mathrm{B} \blacktriangleright E}{\mathrm{B} \triangleright \mathrm{U}E} \quad \dfrac{\mathrm{B} \blacktriangleright M}{\mathrm{B} \triangleright \mathrm{U}M}}{\mathrm{B} \triangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \mathrm{f} : \mathrm{U}E}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \mathtt{force}(\mathrm{f}) : E}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \underbrace{(\mathtt{force}(\mathrm{f}))\mathtt{produce}(\mathtt{thunk}(\mathsf{D}[\mathrm{x}]))}_{\mathsf{B}[\mathrm{f},\mathrm{x}]} : \mathrm{B}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{x} : \mathrm{U}M \vdash \mathsf{B}[\mathrm{f},\mathrm{x}] : \mathrm{B} \quad \mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E; \mathrm{a} : \mathrm{LU}M \vdash \mathrm{a} : \mathrm{LU}M}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E; \mathrm{a} : \mathrm{LU}M \vdash \mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[\mathrm{f},\mathrm{x}] : \mathrm{B}}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[\mathrm{f},\mathrm{x}] : (\mathrm{LU}M \multimap \mathrm{B})}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \mathtt{fold}(\lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[\mathrm{f},\mathrm{x}]) : M}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \underbrace{\mathtt{thunk}(\mathtt{fold}(\lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[\mathrm{f},\mathrm{x}]))}_{\mathsf{X}'[\mathrm{f}]} : \mathrm{U}M}$$

$$\dfrac{\dfrac{\mathrm{B} \triangleright \mathrm{f} : \mathrm{U}E \vdash \mathsf{X}'[\mathrm{f}] : \mathrm{U}M}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \mathtt{produce}(\mathsf{X}'[\mathrm{f}]) : \mathrm{LU}M} \quad \dfrac{\mathrm{B} \triangleright \mathrm{f} : \mathrm{U}E \vdash \mathsf{X}'[\mathrm{f}] : \mathrm{U}M}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \mathtt{force}(\mathsf{X}'[\mathrm{f}]) : M}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \underbrace{(\mathtt{unfold}(\mathtt{force}(\mathsf{X}'[\mathrm{f}])))\mathtt{produce}(\mathsf{X}'[\mathrm{f}])}_{\mathsf{C}[\mathrm{f}]} : \mathrm{B}}$$

$$\begin{aligned}
\mathsf{C}[f] &=_{\mathrm{def}} \\
(\mathtt{unfold}(\mathtt{force}(\mathsf{X}'[f])))\mathtt{produce}(\mathsf{X}'[f]) &=_{\mathrm{def}} \\
(\mathtt{unfold}(\mathtt{force}(\mathtt{thunk}(\mathtt{fold}(\lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[f,\mathrm{x}])))))\mathtt{produce}(\mathsf{X}'[f]) &=_{\beta} \\
(\mathtt{unfold}(\mathtt{fold}(\lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[f,\mathrm{x}])))\mathtt{produce}(\mathsf{X}'[f]) &=_{\beta} \\
(\lambda \mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[f,\mathrm{x}])\mathtt{produce}(\mathsf{X}'[f]) &\to_{\beta} \\
\mathtt{produce}(\mathsf{X}'[f])\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathsf{B}[f,\mathrm{x}] &\to_{\beta} \quad \mathsf{B}[f,\mathsf{X}'[f]]
\end{aligned}$$

$$\begin{aligned}
\mathsf{B}[f,\mathsf{X}'[f]] &=_{\mathrm{def}} (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}(\mathsf{D}[\mathsf{X}'[f]])) \\
&=_{\mathrm{def}} (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}((\mathtt{unfold}(\mathtt{force}(\mathsf{X}'[f])))\mathtt{produce}(\mathsf{X}'[f]))) \\
&=_{\mathrm{def}} (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}(\mathsf{C}[f]))
\end{aligned}$$

$$\dfrac{\dfrac{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E \vdash \mathsf{C}[\mathrm{f}] : \mathrm{B} \quad \mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M; g : \mathrm{LU}E \vdash g : \mathrm{LU}E}{\mathrm{B} \blacktriangleright \mathrm{x} : \mathrm{U}M; g : \mathrm{LU}E \vdash \mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{C}[\mathrm{f}] : \mathrm{B}}}{\mathrm{B} \blacktriangleright \vdash \underbrace{\lambda \mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{C}[\mathrm{f}]}_{\mathsf{Y}} : (\mathrm{LU}E \multimap \mathrm{B})}$$

Given any term f of type $\mathrm{U}E$, $\mathtt{thunk}((\mathsf{Y})\mathtt{produce}(f))$ is a fixed point of $\mathrm{non}(f)$ and, given any term f of type $(\mathrm{UB}{\to\triangleright}\mathrm{UB})$, $(\mathrm{non}(\mathsf{Y}))\mathrm{lin}(f)$ is a fixed point of f.

Turing. We replace B with $(\mathrm{LU}E{\multimap}\mathrm{B})$ in our recursive type M.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{x},\mathrm{B}\blacktriangleright\mathrm{x}}}{\mathrm{x},\mathrm{B}\triangleright\mathrm{Ux}}}{\mathrm{x},\mathrm{B}\blacktriangleright\mathrm{LUx}} \quad \dfrac{\dfrac{\dfrac{\mathrm{B}\blacktriangleright E}{\mathrm{B}\triangleright\mathrm{U}E}}{\mathrm{B}\blacktriangleright\mathrm{LU}E} \quad \dfrac{}{\mathrm{B}\blacktriangleright\mathrm{B}}}{\mathrm{B}\blacktriangleright(\mathrm{LU}E{\multimap}\mathrm{B})}}{\mathrm{x},\mathrm{B}\blacktriangleright(\mathrm{LUx}{\multimap}(\mathrm{LU}E{\multimap}\mathrm{B}))}}{\mathrm{B}\blacktriangleright\underbrace{\mu\mathrm{x}.(\mathrm{LUx}{\multimap}(\mathrm{LU}E{\multimap}\mathrm{B}))}_{M'}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{B}\blacktriangleright M'}{\mathrm{B}\triangleright\mathrm{U}M'}}{\mathrm{B}\triangleright\mathrm{x}:\mathrm{U}M'\vdash\mathrm{x}:\mathrm{U}M'}}{\mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M'\vdash\mathtt{produce}(x):\mathrm{LU}M'} \quad \dfrac{\dfrac{\dfrac{\mathrm{B}\blacktriangleright M'}{\mathrm{B}\triangleright\mathrm{U}M'}}{\mathrm{B}\triangleright\mathrm{x}:\mathrm{U}M'\vdash\mathrm{x}:\mathrm{U}M'}}{\mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M'\vdash\mathtt{force}(x):M'}}{\mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M'\vdash\mathtt{unfold}(\mathtt{force}(x)):(\mathrm{LU}M'{\multimap}(\mathrm{LU}E{\multimap}\mathrm{B}))}}{\mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M'\vdash\underbrace{(\mathtt{unfold}(\mathtt{force}(x)))\mathtt{produce}(x)}_{\mathsf{D}'[\mathrm{x}]}:(\mathrm{LU}E{\multimap}\mathrm{B})}$$

$$\dfrac{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathsf{D}'[\mathrm{x}]:(\mathrm{LU}E{\multimap}\mathrm{B}) \quad \dfrac{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathrm{f}:\mathrm{U}E}{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathtt{produce}(\mathrm{f}):\mathrm{LU}E}}{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash(\mathsf{D}'[\mathrm{x}])\mathtt{produce}(\mathrm{f}):\mathrm{B}}$$

$$\dfrac{\dfrac{\dfrac{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash(\mathsf{D}'[\mathrm{x}])\mathtt{produce}(\mathrm{f}):\mathrm{B}}{\mathrm{B}\triangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathtt{thunk}((\mathsf{D}'[\mathrm{x}])\mathtt{produce}(\mathrm{f})):\mathrm{UB}}}{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathtt{produce}(\mathtt{thunk}((\mathsf{D}'[\mathrm{x}])\mathtt{produce}(\mathrm{f}))):\mathrm{LUB}} \quad \dfrac{\dfrac{\mathrm{B}\blacktriangleright E}{\mathrm{B}\triangleright\mathrm{U}E} \quad \dfrac{\mathrm{B}\blacktriangleright M'}{\mathrm{B}\triangleright\mathrm{U}M'}}{\mathrm{B}\triangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash f:\mathrm{U}E}}{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathtt{force}(f):E}}{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\underbrace{(\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}((\mathsf{D}'[\mathrm{x}])\mathtt{produce}(\mathrm{f})))}_{\mathsf{B}'[\mathrm{f},\mathrm{x}]}:\mathrm{B}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{B}\blacktriangleright\mathrm{f}:\mathrm{U}E,\mathrm{x}:\mathrm{U}M'\vdash\mathsf{B}'[\mathrm{f},\mathrm{x}]:\mathrm{B} \quad \mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M';g:\mathrm{LU}E\vdash g:\mathrm{LU}E}{\mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M';g:\mathrm{LU}E\vdash\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{B}'[\mathrm{f},\mathrm{x}]:\mathrm{B}}}{\mathrm{B}\blacktriangleright\mathrm{x}:\mathrm{U}M'\vdash\lambda\mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{B}'[\mathrm{f},\mathrm{x}]:(\mathrm{LU}E{\multimap}\mathrm{B})} \quad \mathrm{B}\blacktriangleright;\mathrm{a}:\mathrm{LU}M'\vdash\mathrm{a}:\mathrm{LU}M'}{\mathrm{B}\blacktriangleright;\mathrm{a}:\mathrm{LU}M'\vdash\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \lambda\mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{B}'[\mathrm{f},\mathrm{x}]:(\mathrm{LU}E{\multimap}\mathrm{B})}}{\mathrm{B}\blacktriangleright\vdash\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \lambda\mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{B}'[\mathrm{f},\mathrm{x}]:(\mathrm{LU}M'{\multimap}(\mathrm{LU}E{\multimap}\mathrm{B}))}}{\mathrm{B}\blacktriangleright\vdash\mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \lambda\mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{B}'[\mathrm{f},\mathrm{x}]):M'}}{\mathrm{B}\triangleright\vdash\underbrace{\mathtt{thunk}(\mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \lambda\mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ \mathsf{B}'[\mathrm{f},\mathrm{x}]))}_{\mathsf{X}''}:\mathrm{U}M'}$$

$\mathsf{X}'' = \mathtt{thunk}(\mathtt{fold}(\lambda\mathrm{a}.\mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \lambda\mathrm{g}.\mathrm{g}\ \mathtt{to}\ \mathrm{f}\ \mathtt{in}\ (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}(((\mathtt{unfold}(\mathtt{force}(x)))\mathtt{produce}(x))\mathtt{produce}(\mathrm{f})))))$

$$\dfrac{\dfrac{\mathrm{B} \triangleright \vdash \mathsf{X}'' : \mathrm{U}M'}{\mathrm{B} \blacktriangleright \vdash \mathtt{produce}(\mathsf{X}'') : \mathrm{L}\mathrm{U}M'} \qquad \dfrac{\dfrac{\mathrm{B} \triangleright \vdash \mathsf{X}'' : \mathrm{U}M'}{\mathrm{B} \blacktriangleright \vdash \mathtt{force}(\mathsf{X}'') : M'}}{\mathrm{B} \blacktriangleright \vdash \mathtt{unfold}(\mathtt{force}(\mathsf{X}'')) : (\mathrm{L}\mathrm{U}M' \multimap (\mathrm{L}\mathrm{U}E \multimap \mathrm{B}))}}{\mathrm{B} \blacktriangleright \vdash \underbrace{(\mathtt{unfold}(\mathtt{force}(\mathsf{X}'')))\mathtt{produce}(\mathsf{X}'')}_{\mathsf{Z}} : (\mathrm{L}\mathrm{U}E \multimap \mathrm{B})}$$

$$\begin{aligned}
(\mathsf{Z})\mathtt{produce}(f) &=_{\mathrm{def}} \\
((\mathtt{unfold}(\mathtt{force}(\mathsf{X}'')))\mathtt{produce}(\mathsf{X}''))\mathtt{produce}(f) &=_{\mathrm{def}} \\
((\mathtt{unfold}(\mathtt{force}(\mathtt{thunk}(\mathtt{fold}(\lambda \mathrm{a.a\ to\ x\ in\ } \lambda \mathrm{g.g\ to\ f\ in\ } \mathsf{B}'[\mathrm{f},\mathrm{x}])))))\mathtt{produce}(\mathsf{X}''))\mathtt{produce}(f) &=_{\beta} \\
((\mathtt{unfold}(\mathtt{fold}(\lambda \mathrm{a.a\ to\ x\ in\ } \lambda \mathrm{g.g\ to\ f\ in\ } \mathsf{B}'[\mathrm{f},\mathrm{x}])))\mathtt{produce}(\mathsf{X}''))\mathtt{produce}(f) &=_{\beta} \\
((\lambda \mathrm{a.a\ to\ x\ in\ } \lambda \mathrm{g.g\ to\ f\ in\ } \mathsf{B}'[\mathrm{f},\mathrm{x}])\mathtt{produce}(\mathsf{X}''))\mathtt{produce}(f) &=_{\beta} \\
(\mathtt{produce}(\mathsf{X}'')\ \mathtt{to\ x\ in\ } \lambda \mathrm{g.g\ to\ f\ in\ } \mathsf{B}'[\mathrm{f},\mathrm{x}])\mathtt{produce}(f) &=_{\beta} \\
(\lambda \mathrm{g.g\ to\ f\ in\ } \mathsf{B}'[\mathrm{f},\mathsf{X}''])\mathtt{produce}(f) &\to_{\beta} \\
\mathtt{produce}(f)\ \mathtt{to\ f\ in\ } \mathsf{B}'[\mathrm{f},\mathsf{X}''] &\to_{\beta} \\
\mathsf{B}'[f,\mathsf{X}'']
\end{aligned}$$

$$\begin{aligned}
\mathsf{B}'[f,\mathsf{X}''] &=_{\mathrm{def}} (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}((\mathsf{D}'[\mathsf{X}''])\mathtt{produce}(f))) \\
&=_{\mathrm{def}} (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}(((\mathtt{unfold}(\mathtt{force}(\mathsf{X}'')))\mathtt{produce}(\mathsf{X}''))\mathtt{produce}(f))) \\
&=_{\mathrm{def}} (\mathtt{force}(f))\mathtt{produce}(\mathtt{thunk}((\mathsf{Z})\mathtt{produce}(f)))
\end{aligned}$$

Given any term f of type UE, $\mathtt{thunk}((\mathsf{Z})\mathtt{produce}(f))$ is a fixed point of $\mathrm{non}(f)$ and, given any term f of type $(\mathrm{UB} \multimap \triangleright \mathrm{UB})$, $(\mathrm{non}(\mathsf{Z}))\mathrm{lin}(f)$ is a fixed point of f.

$$\frac{}{\Theta, \mathrm{T} \square \mathrm{T}}$$

$$\frac{\Theta \square S \quad \ldots \quad \Theta \square T}{\Theta \square \mathrm{s} : S, \ldots, \mathrm{t} : T \vdash \mathrm{t} : T}$$

$$\frac{\Theta \square S \quad \Theta \square T}{\Theta \square (S \rightharpoonup T)}$$

$$\frac{\Theta \square \Gamma, \mathrm{s} : S \vdash t : T}{\Theta \square \Gamma \vdash \lambda \mathrm{s}.t : (S \rightharpoonup T)}$$

$$\frac{\Theta \square \Gamma \vdash s : S \quad \Theta \square \Gamma \vdash f : (S \rightharpoonup T)}{\Theta \square \Gamma \vdash (f)s : T}$$

$$\frac{\Theta, \mathrm{S} \square T}{\Theta \square \mu \mathrm{S}.T}$$

$$\frac{\Theta \square \Gamma \vdash t : T[\mu \mathrm{S}.T]}{\Theta \square \Gamma \vdash \mathtt{fold}(t) : \mu \mathrm{S}.T}$$

$$\frac{\Theta \square \Gamma \vdash t : \mu \mathrm{S}.T}{\Theta \square \Gamma \vdash \mathtt{unfold}(t) : T[\mu \mathrm{S}.T]}$$

$$\begin{array}{rcl} (\lambda \mathrm{x}.f[\mathrm{x}])y & \rightarrow_\beta & f[y] \\ \mathtt{unfold}(\mathtt{fold}(a)) & \rightarrow_\beta & a \\ f & =_\eta & \lambda \mathrm{x}.(f)\mathrm{x} \\ a & =_\eta & \mathtt{fold}(\mathtt{unfold}(a)) \end{array}$$

Figure 8.1: Rules and equations for closed FPC

.

$$\frac{}{\Psi \rhd \mathrm{X}} \qquad \frac{}{\Psi \blacktriangleright \mathrm{A}}$$

$$\frac{\Psi \rhd X \quad \ldots \quad \Psi \rhd Y}{\Psi \rhd \mathrm{x} : X, \ldots, \mathrm{y} : Y \vdash \mathrm{y} : Y} \qquad \frac{\Psi \rhd X \quad \ldots \quad \Psi \rhd Y \quad \Psi \blacktriangleright A}{\Psi \blacktriangleright \mathrm{x} : X, \ldots, \mathrm{y} : Y \mid \mathrm{a} : A \vdash \mathrm{a} : A}$$

$$\frac{\Psi \rhd X \quad \Psi \rhd Y}{\Psi \rhd (X \mathrel{-\!\!\triangleright} Y)} \qquad \frac{\Psi \blacktriangleright A \quad \Psi \blacktriangleright B}{\Psi \blacktriangleright (A \multimap B)}$$

$$\frac{\Psi \rhd \Gamma, \mathrm{x} : X \vdash y : Y}{\Psi \rhd \Gamma \vdash \lambda \mathrm{x}.y : (X \mathrel{-\!\!\triangleright} Y)} \qquad \frac{\Psi \blacktriangleright \Gamma \mid \Pi, \mathrm{a} : A \vdash b : B}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash \lambda \mathrm{a}.b : (A \multimap B)}$$

$$\frac{\Psi \rhd \Gamma \vdash x : X \quad \Psi \rhd \Gamma \vdash f : (X \mathrel{-\!\!\triangleright} Y)}{\Psi \rhd \Gamma \vdash (f)x : Y} \qquad \frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a : A \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash g : (A \multimap B)}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash (g)a : B}$$

$$\frac{\Psi, \mathrm{A} \blacktriangleright B}{\Psi \blacktriangleright \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a : B[\mu \mathrm{A}.B]}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash \mathtt{fold}(a) : \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash a : \mu \mathrm{A}.B}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash \mathtt{unfold}(a) : B[\mu \mathrm{A}.B]}$$

$$\frac{\Psi \blacktriangleright A}{\Psi \rhd \mathrm{U}A} \qquad \frac{\Psi \rhd X}{\Psi \blacktriangleright \mathrm{L}X}$$

$$\frac{\Psi \blacktriangleright \Gamma \vdash a : A}{\Psi \rhd \Gamma \vdash \mathtt{thunk}(a) : \mathrm{U}A} \qquad \frac{\Psi \rhd \Gamma \vdash x : X}{\Psi \blacktriangleright \Gamma \vdash \mathtt{produce}(x) : \mathrm{L}X}$$

$$\frac{\Psi \rhd \Gamma \vdash x : \mathrm{U}A}{\Psi \blacktriangleright \Gamma \vdash \mathtt{force}(x) : A} \qquad \frac{\Psi \blacktriangleright \Gamma, \mathrm{x} : X \mid \Pi \vdash a : A \quad \Psi \blacktriangleright \Gamma \mid \Pi' \vdash b : \mathrm{L}X}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a : A}$$

Figure 8.2: Rules for closed RLNL.

$$
\begin{array}{rcl}
(\lambda \mathrm{x}.\, f[\mathrm{x}])y & \to_\beta & f[y] \\
(\lambda \mathrm{a}.g[\mathrm{a}])b & \to_\beta & g[b] \\
\mathtt{unfold}(\mathtt{fold}(a)) & \to_\beta & a \\
\mathtt{force}(\mathtt{thunk}(a)) & \to_\beta & a \\
\mathtt{produce}(y)\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a[\mathrm{x}] & \to_\beta & a[y] \\
f & =_\eta & \lambda \mathrm{x}.(f)\mathrm{x} \\
g & =_\eta & \lambda \mathrm{a}.(g)\mathrm{a} \\
a & =_\eta & \mathtt{fold}(\mathtt{unfold}(a)) \\
x & =_\eta & \mathtt{thunk}(\mathtt{force}(x)) \\
a & =_\eta & a\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{produce}(\mathrm{x}) \\
(g)(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a) & =_{\mathrm{a}} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (g)a \\
(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ g)c & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (g)c \\
\mathtt{unfold}(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a) & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{unfold}(a) \\
(b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a)\ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ c & \to_{\mathrm{cc}} & b\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ (a\ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ c)
\end{array}
$$

Figure 8.3: Equations for closed RLNL.

8.2 A Linear Fixed Point Calculus

The idioms that appear in our derivations of recursive combinators and the structural view of the motivating models of RLNL suggest new types.

8.2.1 Fixed Point Idioms

Looking at the derivations of fixed point combinators in Section 8.1.2 we see that we have used RLNL in a stylized way. The rules for the function space ($\multimap$) and the constructor L are used only within three particular idioms.

Idiom	Sugar
$\dfrac{\dfrac{\Psi \rhd T}{\Psi \blacktriangleright \mathrm{L}T} \quad \Psi \blacktriangleright R}{\Psi \blacktriangleright (\mathrm{L}T \multimap R)}$	$\dfrac{\Psi \rhd T \quad \Psi \blacktriangleright R}{\Psi \blacktriangleright (T \to R)}$
$\dfrac{\Psi \blacktriangleright \Gamma, \mathrm{a} : T \vdash r : R \quad \Psi \blacktriangleright \Gamma; \mathrm{y} : \mathrm{L}T \vdash \mathrm{y} : \mathrm{L}T}{\dfrac{\Psi \blacktriangleright \Gamma; \mathrm{y} : \mathrm{L}T \vdash \mathrm{y}\ \mathtt{to}\ \mathrm{a}\ \mathtt{in}\ r : R}{\Psi \blacktriangleright \Gamma \vdash \lambda \mathrm{y}.\mathrm{y}\ \mathtt{to}\ \mathrm{a}\ \mathtt{in}\ r : (\mathrm{L}T \multimap R)}}$	$\dfrac{\Psi \blacktriangleright \Gamma, \mathrm{a} : T \vdash r : R}{\Psi \blacktriangleright \Gamma \vdash \lambda \mathrm{a}.r : (T \to R)}$
$\dfrac{\dfrac{\Psi \rhd \Gamma \vdash t : T}{\Psi \rhd \Gamma \vdash \mathtt{produce}(t) : \mathrm{L}T} \quad \Psi \blacktriangleright \Gamma \vdash f : (\mathrm{L}T \multimap R)}{\Psi \blacktriangleright \Gamma \vdash (f)\mathtt{produce}(t) : R}$	$\dfrac{\Psi \rhd \Gamma \vdash t : T \quad \Psi \blacktriangleright \Gamma \vdash f : (T \to R)}{\Psi \blacktriangleright \Gamma \vdash (f)t : R}$

Each rule shown to the right can be read as sugar for the idiom to the left. Sugared derivations and terms are shorter than the original. Here is the sugared form of the derivation of Ω.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{A},\mathrm{B} \blacktriangleright \mathrm{A}}}{\mathrm{A},\mathrm{B} \rhd \mathrm{U}\mathrm{A}} \quad \dfrac{}{\mathrm{A},\mathrm{B} \blacktriangleright \mathrm{B}}}{\mathrm{A},\mathrm{B} \blacktriangleright (\mathrm{U}\mathrm{A} \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \underbrace{\mu \mathrm{A}.(\mathrm{U}\mathrm{A} \to \mathrm{B})}_{M}}}{\mathrm{B} \rhd \mathrm{U}M} \quad \dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright M}{\mathrm{B} \rhd \mathrm{U}M}}{\mathrm{B} \rhd \mathrm{a} : \mathrm{U}M \vdash x : \mathrm{U}M}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M \vdash \mathtt{force}(a) : M}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M \vdash \mathtt{unfold}(\mathtt{force}(a)) : (\mathrm{U}M \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M \vdash (\mathtt{unfold}(\mathtt{force}(a)))a : \mathrm{B}}}{\mathrm{B} \blacktriangleright \vdash \underbrace{\lambda \mathrm{a}.(\mathtt{unfold}(\mathtt{force}(a)))a}_{\mathrm{A}} : (\mathrm{U}M \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \vdash \mathrm{A} : (\mathrm{U}M \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \vdash \mathtt{fold}(\mathrm{A}) : M}}{\mathrm{B} \rhd \vdash \mathtt{thunk}(\mathtt{fold}(\mathrm{A})) : \mathrm{U}M} \quad \mathrm{B} \blacktriangleright \vdash \mathrm{A} : (\mathrm{U}M \to \mathrm{B})}{\mathrm{B} \blacktriangleright \vdash \underbrace{(\mathrm{A})\mathtt{thunk}(\mathtt{fold}(\mathrm{A}))}_{\Omega} : \mathrm{B}}$$

Here is the sugared form of the Turing combinator.

$$\dfrac{\dfrac{\dfrac{}{\mathrm{A},\mathrm{B} \blacktriangleright \mathrm{A}}}{\mathrm{A},\mathrm{B} \rhd \mathrm{UA}} \qquad \dfrac{\dfrac{\mathrm{B} \blacktriangleright E}{\mathrm{B} \rhd \mathrm{U}E}\qquad \dfrac{}{\mathrm{B} \blacktriangleright \mathrm{B}}}{\mathrm{B} \blacktriangleright (\mathrm{U}E \to \mathrm{B})}}{\dfrac{\mathrm{A},\mathrm{B} \blacktriangleright (\mathrm{UA} \to (\mathrm{U}E \to \mathrm{B}))}{\mathrm{B} \blacktriangleright \underbrace{\mu \mathrm{A}.(\mathrm{UA} \to (\mathrm{U}E \to \mathrm{B}))}_{M'}}}$$

$$\dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright M'}{\mathrm{B} \rhd \mathrm{U}M'}}{\mathrm{B} \rhd \mathrm{a} : \mathrm{U}M' \vdash x : \mathrm{U}M'} \qquad \dfrac{\dfrac{\mathrm{B} \blacktriangleright M'}{\mathrm{B} \rhd \mathrm{U}M'}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M' \vdash \mathtt{force}(a) : M'}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M' \vdash \mathtt{unfold}(\mathtt{force}(a)) : (\mathrm{U}M' \to (\mathrm{U}E \to \mathrm{B}))}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M' \vdash \underbrace{(\mathtt{unfold}(\mathtt{force}(a)))a}_{\mathsf{D}'[\mathrm{a}]} : (\mathrm{U}E \to \mathrm{B})}$$

$$\dfrac{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash f : \mathrm{U}E \quad \mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash \mathsf{D}'[\mathrm{a}] : (\mathrm{U}E \to \mathrm{B})}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash \underbrace{(\mathsf{D}'[\mathrm{a}])\mathrm{f}}_{\mathsf{C}[\mathrm{f},\mathrm{a}]} : \mathrm{B}}$$

$$\dfrac{\dfrac{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash \mathsf{C}[\mathrm{f},\mathrm{a}] : \mathrm{B}}{\mathrm{B} \rhd \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash \mathtt{thunk}(\mathsf{C}[\mathrm{f},\mathrm{a}]) : \mathrm{UB}} \qquad \dfrac{\dfrac{\mathrm{B} \blacktriangleright E}{\mathrm{B} \rhd \mathrm{U}E}\qquad \dfrac{\mathrm{B} \blacktriangleright M'}{\mathrm{B} \rhd \mathrm{U}M'}}{\mathrm{B} \rhd \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash f : \mathrm{U}E}}{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash \underbrace{(\mathtt{force}(f))\mathtt{thunk}(\mathsf{C}[\mathrm{f},\mathrm{a}])}_{\mathsf{B}'[\mathrm{f},\mathrm{a}]} : \mathrm{B}}$$

$$\dfrac{\dfrac{\mathrm{B} \blacktriangleright \mathrm{f} : \mathrm{U}E, \mathrm{a} : \mathrm{U}M' \vdash \mathsf{B}'[\mathrm{f},\mathrm{a}] : \mathrm{B}}{\mathrm{B} \blacktriangleright \mathrm{a} : \mathrm{U}M' \vdash \lambda \mathrm{f}.\mathsf{B}'[\mathrm{f},\mathrm{a}] : (\mathrm{U}E \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \vdash \underbrace{\lambda \mathrm{a}.\lambda \mathrm{f}.\mathsf{B}'[\mathrm{f},\mathrm{a}]}_{\mathsf{A}''} : (\mathrm{U}M' \to (\mathrm{U}E \to \mathrm{B}))}$$

$$\dfrac{\dfrac{\dfrac{\mathrm{B} \blacktriangleright \vdash \mathsf{A}'' : (\mathrm{U}M' \to (\mathrm{U}E \to \mathrm{B}))}{\mathrm{B} \blacktriangleright \vdash \mathtt{fold}(\mathsf{A}'') : M'}}{\mathrm{B} \rhd \vdash \mathtt{thunk}(\mathtt{fold}(\mathsf{A}'')) : \mathrm{U}M'} \qquad \mathrm{B} \blacktriangleright \vdash \mathsf{A}'' : (\mathrm{U}M' \to (\mathrm{U}E \to \mathrm{B}))}{\mathrm{B} \blacktriangleright \vdash \underbrace{(\mathsf{A}'')\mathtt{thunk}(\mathtt{fold}(\mathsf{A}''))}_{\mathsf{Z}} : (\mathrm{U}E \to \mathrm{B})}$$

The sugared derivations inhabit the calculus shown in Figure 8.4 which matches a fragment of Levy's Call-By-Push-Value. In the RLNL derivations of our recursive combinators the linear part of the context is only used in the introduction idiom (and only with one variable), so we can drop this part of the context.

This calculus has rules for just the one constructor U, which Levy refers to as the jumping fragment of CBPV. Following the analysis in Section 7.2, we understand the rules for this constructor as generating the free distributor with an aft representation.

In the presence of a proper oblique function type, the isomorphism between the two representations described in Section 7.2.5 factors into two isomorphisms denoted by terms in two closed fragments of LNL, one without linear function types, the other without nonlinear. In models without one or the other function type, much of the expressivity is retained.

8.2.2 Translating FPC into RLNL

The untyped and FPC combinators are sensitive to the choice of operational semantics. In particular, call-by-value theories require special, new fixed point combinators because the combinators inherited from the base theory are bottomized in call-by-value semantics. Fortunately, the structure of our models allows us to side-step the issue of call-by-value *versus* call-by-name. In fact this was one of the original reasons for choosing a mixed linear/nonlinear calculus, the intuition being that if the recursive types live in the linear part of the calculus their use should be insensitive to the evaluation sequence.

A price is paid in type complexity. We now view the situation in terms of Levy's operational analysis: the type theory forces us to specify, with a 'thunk', the points in the term where evaluation is defered. This sits well with the idea of linear use. A parameter whose meaning requires evaluation must appear in exactly one place so as to have a well defined evaluation context. Note however that our sugared derivations don't require linear identifiers.

Our translations of (our fragment of) FPC into RLNL factor through (our fragment of) CBPV with recursive computation types. The translations into CPBV match Levy's translations of call-by-name and call-by-value calculi (which use the rules for L). The translation of (our fragment of) CPBV into RLNL is given by the above sugaring.

The equations we have given for FPC generate a 'call-by-name theory' because a call-by-value semantics induces fewer equivalences. We translate the rules of the closed/recursive fragment of FPC into CBPV derivations as shown in Figure 8.5. FPC sequents are translated into CBPV sequents of the form

$$\Psi \blacktriangleright a_1 : \mathrm{U}A_1, , \ldots, , a_n : \mathrm{U}A_n \vdash t : B$$

with context types and term types translated slightly differently. This corresponds to the idea that parameters are passed as thunks. Note that the translation carries our derivations of recursive combinators in FPC to our (sugared) derivations in RLNL.

For a call-by-value semantics, we translate the rules of the closed/recursive fragment of FPC into CBPV derivations as shown in Figure 8.6. We give two, simultaneous translations of types into non-linear types and into linear types. The linear translation is used in the translation of recursive types. FPC sequents are translated into CBPV sequents of the form

$$\Psi \blacktriangleright a_1 : A_1, , \ldots, , a_n : A_n \vdash t : \mathrm{F}B.$$

Again we translate context types and term types slightly differently.

The fixed point combinators we have given for the lambda calculus and FPC misbehave in a call-by-value semantics. Evaluated under call-by-value, both $\mathsf{Y}(f)$ and $f(\mathsf{Y}(f))$ diverge, so while $\mathsf{Y}(f)$ is a fixed point of f in the sense that $f(\mathsf{Y}(f)) = \mathsf{Y}(f)$, it is not useful. The same holds for $\mathsf{Z}(f)$. There is a work-around for this problem: in A'[f], replace D[a] with $\lambda g.\mathsf{D}[\mathrm{a}](g)$ and, in A'', we replace $\mathsf{D}[\mathrm{a}](f)$ with $\lambda g.\mathsf{D}[\mathrm{a}](f)(g)$. This delays the evaluation of the argument passed to f: the argument is passed as a 'thunk' and is only evaluated if 'forced' by f. Note that the call-by-value image of these call-by-value fixed point combinators in RLNL is equivalent to our RLNL combinators.

8.2.3 Parameterized Adjoint Types

The oblique function type is naturally interpreted by a costructural action and the calculus shown in Figure 8.4 is naturally interpreted in a suitable structural functor (Definition 5.2.7), on a structurally compact bistructural category. This suggests the addition of rules for a type for the structural action on D. It also suggests a linear variable introduction rule which is interpreted by the transformation $\iota : c \oslash d \to d$. If a linear area is added to the other rules, this allows linear identifiers to appear in derivations but only one at a time: a multiplication on D would be needed to interpret multiple linear identifiers.

Since the right Kleisli adjoint for a domain-theoretic lifting monad, which is our leading example of a suitable structural functor, may be viewed as one component of an exponential, we propose the calculus shown in Figure 8.7 with a type for the exponential. This calculus is naturally interpreted by a suitable exponential on a structurally compact bistructural category. A translation into RLNL can be derived from the construction of a suitable exponential from a monoidal/cartesian closed adjunction.

$$\frac{}{\blacktriangleright \mathrm{A}}$$

$$\frac{\triangleright X \quad \ldots \quad \triangleright Y}{\triangleright \mathrm{x} : X, \ldots, \mathrm{y} : Y \vdash \mathrm{y} : Y}$$

$$\frac{\triangleright X \quad \blacktriangleright B}{\blacktriangleright (X \to B)}$$

$$\frac{\blacktriangleright \Gamma, \mathrm{x} : X \vdash b : B}{\blacktriangleright \Gamma \vdash \lambda \mathrm{x}.b : (A \to B)}$$

$$\frac{\blacktriangleright \Gamma \vdash x : X \quad \blacktriangleright \Gamma \vdash g : (X \to B)}{\blacktriangleright \Gamma \vdash (g)x : B}$$

$$\frac{\Psi, \mathrm{A} \blacktriangleright B}{\Psi \blacktriangleright \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma; \Pi \vdash a : B[\mu \mathrm{A}.B]}{\Psi \blacktriangleright \Gamma; \Pi \vdash \mathtt{fold}(a) : \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma; \Pi \vdash a : \mu \mathrm{A}.B}{\Psi \blacktriangleright \Gamma; \Pi \vdash \mathtt{unfold}(a) : B[\mu \mathrm{A}.B]}$$

$$\frac{\blacktriangleright A}{\triangleright \mathrm{U}A}$$

$$\frac{\blacktriangleright \Gamma \vdash a : A}{\triangleright \Gamma \vdash \mathtt{thunk}(a) : \mathrm{U}A}$$

$$\frac{\triangleright \Gamma \vdash x : \mathrm{U}A}{\blacktriangleright \Gamma \vdash \mathtt{force}(x) : A}$$

Figure 8.4: Jumping fragment of CBPV with recursive computation types.

$$\frac{}{\Theta^{\mathrm{n}}, \mathrm{T}^{\mathrm{n}} \blacktriangleright \mathrm{T}^{\mathrm{n}}}$$

$$\frac{\dfrac{\Theta^{\mathrm{n}} \blacktriangleright S^{\mathrm{n}}}{\Theta^{\mathrm{n}} \rhd \mathrm{U}S^{\mathrm{n}}} \quad \ldots \quad \dfrac{\Theta^{\mathrm{n}} \blacktriangleright T^{\mathrm{n}}}{\Theta^{\mathrm{n}} \rhd \mathrm{U}T^{\mathrm{n}}}}{\Theta^{\mathrm{n}} \rhd \mathrm{s}^{\mathrm{n}} : \mathrm{U}S^{\mathrm{n}}, \ldots, \mathrm{t}^{\mathrm{n}} : \mathrm{U}T^{\mathrm{n}} \vdash \mathrm{t}^{\mathrm{n}} : \mathrm{U}T^{\mathrm{n}}}}{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{s}^{\mathrm{n}} : \mathrm{U}S^{\mathrm{n}}, \ldots, \mathrm{t}^{\mathrm{n}} : \mathrm{U}T^{\mathrm{n}} \vdash \mathtt{force}(\mathrm{t}^{\mathrm{n}}) : T^{\mathrm{n}}}$$

$$\frac{\dfrac{\Theta^{\mathrm{n}} \blacktriangleright S^{\mathrm{n}}}{\Theta^{\mathrm{n}} \rhd \mathrm{U}S^{\mathrm{n}}} \quad \Theta^{\mathrm{n}} \blacktriangleright T^{\mathrm{n}}}{\Theta^{\mathrm{n}} \blacktriangleright (\mathrm{U}S^{\mathrm{n}} \to T^{\mathrm{n}})}$$

$$\frac{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}}, \mathrm{s}^{\mathrm{n}} : \mathrm{U}S^{\mathrm{n}} \vdash t^{\mathrm{n}} : T^{\mathrm{n}}}{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash \lambda \mathrm{s}^{\mathrm{n}}.t^{\mathrm{n}} : (\mathrm{U}S^{\mathrm{n}} \to T^{\mathrm{n}})}$$

$$\frac{\dfrac{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash s^{\mathrm{n}} : S^{\mathrm{n}}}{\Theta^{\mathrm{n}} \rhd \mathrm{U}\Gamma^{\mathrm{n}} \vdash \mathtt{thunk}(s) : \mathrm{U}S^{\mathrm{n}}} \quad \Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash f^{\mathrm{n}} : (\mathrm{U}S^{\mathrm{n}} \to T^{\mathrm{n}})}{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash (f^{\mathrm{n}})\mathtt{thunk}(s^{\mathrm{n}}) : T^{\mathrm{n}}}$$

$$\frac{\Theta^{\mathrm{n}}, \mathrm{S}^{\mathrm{n}} \blacktriangleright T^{\mathrm{n}}}{\Theta^{\mathrm{n}} \blacktriangleright \mu \mathrm{S}^{\mathrm{n}}.T^{\mathrm{n}}}$$

$$\frac{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash t^{\mathrm{n}} : T^{\mathrm{n}}[\mu \mathrm{S}^{\mathrm{n}}.T^{\mathrm{n}}]}{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash \mathtt{fold}(t^{\mathrm{n}}) : \mu \mathrm{S}^{\mathrm{n}}.T^{\mathrm{n}}}$$

$$\frac{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash t^{\mathrm{n}} : \mu \mathrm{S}^{\mathrm{n}}.T^{\mathrm{n}}}{\Theta^{\mathrm{n}} \blacktriangleright \mathrm{U}\Gamma^{\mathrm{n}} \vdash \mathtt{unfold}(t^{\mathrm{n}}) : T^{\mathrm{n}}[\mu \mathrm{S}^{\mathrm{n}}.T^{\mathrm{n}}]}$$

Figure 8.5: The call-by-name image of FPC in CBPV

$$\frac{\dfrac{}{\Theta^{\mathrm{v}}, \mathrm{T}^{\mathrm{v}} \blacktriangleright \mathrm{T}^{\mathrm{v}}}}{\Theta^{\mathrm{v}}, \mathrm{T}^{\mathrm{v}} \rhd \mathrm{U}\mathrm{T}^{\mathrm{v}}} \quad \frac{}{\Theta^{\mathrm{v}}, \mathrm{T}^{\mathrm{v}} \blacktriangleright \mathrm{T}^{\mathrm{v}}}$$

$$\frac{\Theta^{\mathrm{v}} \rhd S^{\mathrm{v}} \quad \ldots \quad \Theta^{\mathrm{v}} \rhd T^{\mathrm{v}}}{\dfrac{\Theta^{\mathrm{v}} \rhd \mathrm{s}^{\mathrm{v}} : S^{\mathrm{v}}, \ldots, \mathrm{t}^{\mathrm{v}} : T^{\mathrm{v}} \vdash \mathrm{t}^{\mathrm{v}} : T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \rhd \mathrm{s}^{\mathrm{v}} : S^{\mathrm{v}}, \ldots, \mathrm{t}^{\mathrm{v}} : T^{\mathrm{v}} \vdash \mathtt{produce}(\mathrm{t}^{\mathrm{v}}) : \mathrm{F}T^{\mathrm{v}}}}$$

$$\frac{\dfrac{\Theta^{\mathrm{v}} \rhd S^{\mathrm{v}} \quad \dfrac{\Theta^{\mathrm{v}} \rhd T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \mathrm{F}T^{\mathrm{v}}}}{\dfrac{\Theta^{\mathrm{v}} \blacktriangleright (S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}})}{\Theta^{\mathrm{v}} \rhd \mathrm{U}(S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}})}} \quad \frac{\Theta^{\mathrm{v}} \rhd S^{\mathrm{v}} \quad \dfrac{\Theta^{\mathrm{v}} \rhd T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \mathrm{F}T^{\mathrm{v}}}}{\Theta^{\mathrm{v}} \blacktriangleright (S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}})}$$

$$\frac{\dfrac{\dfrac{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}}, \mathrm{s}^{\mathrm{v}} : S^{\mathrm{v}} \vdash t^{\mathrm{v}} : \mathrm{F}T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash \lambda \mathrm{s}^{\mathrm{v}}.t^{\mathrm{v}} : (S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}})}}{\Theta^{\mathrm{v}} \rhd \Gamma^{\mathrm{v}} \vdash \mathtt{thunk}(\lambda \mathrm{s}^{\mathrm{v}}.t^{\mathrm{v}}) : \mathrm{U}(S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}})}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash \mathtt{produce}(\mathtt{thunk}(\lambda \mathrm{s}^{\mathrm{v}}.t^{\mathrm{v}})) : \mathrm{FU}(S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}})}$$

$$\frac{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash s^{\mathrm{v}} : \mathrm{F}S^{\mathrm{v}} \qquad \dfrac{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash f^{\mathrm{v}} : \mathrm{FU}(S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}}) \quad \Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}}, \mathrm{x} : S^{\mathrm{v}}, \mathrm{y} : \mathrm{U}(S^{\mathrm{v}} \to \mathrm{F}T^{\mathrm{v}}) \vdash (\mathtt{force}(\mathrm{y}))\mathrm{x} : \mathrm{F}T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}}, \mathrm{x} : S^{\mathrm{v}} \vdash f^{\mathrm{v}} \ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ (\mathtt{force}(\mathrm{y}))\mathrm{x} : \mathrm{F}T^{\mathrm{v}}}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}}, \mathrm{x} : S^{\mathrm{v}} \vdash s^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ f^{\mathrm{v}} \ \mathtt{to}\ \mathrm{y}\ \mathtt{in}\ (\mathtt{force}(\mathrm{y}))\mathrm{x} : \mathrm{F}T^{\mathrm{v}}}$$

$$\frac{\dfrac{\Theta^{\mathrm{v}}, \mathrm{S}^{\mathrm{v}} \blacktriangleright T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}}}{\Theta^{\mathrm{v}} \rhd \mathrm{U}\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}} \quad \frac{\Theta^{\mathrm{v}}, \mathrm{S}^{\mathrm{v}} \blacktriangleright T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}}$$

$$\frac{\dfrac{\dfrac{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash t^{\mathrm{v}} : \mathrm{F}T^{\mathrm{v}}[\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}] \quad \Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}}, \mathrm{x} : \mathrm{U}T^{\mathrm{v}}[\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}] \vdash \mathtt{fold}(\mathtt{force}(\mathrm{x})) : \mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash t^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{fold}(\mathtt{force}(\mathrm{x})) : \mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash \mathtt{thunk}(t^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{fold}(\mathtt{force}(\mathrm{x}))) : \mathrm{U}\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash \mathtt{produce}(\mathtt{thunk}(t^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{fold}(\mathtt{force}(\mathrm{x})))) : \mathrm{FU}\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}}$$

$$\frac{\dfrac{\dfrac{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash t^{\mathrm{v}} : \mathrm{F}\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}} \quad \Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}}, \mathrm{x} : \mathrm{U}\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}} \vdash \mathtt{unfold}(\mathtt{force}(\mathrm{x})) : T^{\mathrm{v}}[\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}]}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash t^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{unfold}(\mathtt{force}(\mathrm{x})) : T^{\mathrm{v}}[\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}]}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash \mathtt{thunk}(t^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{unfold}(\mathtt{force}(\mathrm{x}))) : \mathrm{U}T^{\mathrm{v}}[\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}]}}{\Theta^{\mathrm{v}} \blacktriangleright \Gamma^{\mathrm{v}} \vdash \mathtt{produce}(\mathtt{thunk}(t^{\mathrm{v}} \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{unfold}(\mathtt{force}(\mathrm{x})))) : \mathrm{FU}T^{\mathrm{v}}[\mu \mathrm{S}^{\mathrm{v}}.T^{\mathrm{v}}]}$$

Figure 8.6: The call-by-value image of FPC in CBPV

$$\frac{}{\Psi \rhd \mathrm{X}}$$

$$\frac{}{\Psi \blacktriangleright \mathrm{A}}$$

$$\frac{\Psi \rhd X \quad \ldots \quad \Psi \rhd Y}{\Psi \rhd \mathrm{x}:X,\ldots,\mathrm{y}:Y \vdash \mathrm{y}:Y}$$

$$\frac{\Psi \rhd X \quad \ldots \quad \Psi \rhd Y \quad \Psi \blacktriangleright A}{\Psi \blacktriangleright \mathrm{x}:X,\ldots,\mathrm{y}:Y \mid \mathrm{a}:A \vdash \mathrm{a}:A}$$

$$\frac{\rhd X \quad \blacktriangleright B}{\blacktriangleright (X \to B)}$$

$$\frac{\blacktriangleright \Gamma, \mathrm{x}:X \mid \Pi \vdash b:B}{\blacktriangleright \Gamma \mid \Pi \vdash \lambda \mathrm{x}.b : (A \to B)}$$

$$\frac{\rhd \Gamma \vdash x:X \quad \blacktriangleright \Gamma \mid \Pi \vdash g : (X \to B)}{\blacktriangleright \Gamma \mid \Pi \vdash (g)x : B}$$

$$\frac{\Psi, \mathrm{A} \blacktriangleright B}{\Psi \blacktriangleright \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma; \Pi \vdash a : B[\mu \mathrm{A}.B]}{\Psi \blacktriangleright \Gamma; \Pi \vdash \mathtt{fold}(a) : \mu \mathrm{A}.B}$$

$$\frac{\Psi \blacktriangleright \Gamma; \Pi \vdash a : \mu \mathrm{A}.B}{\Psi \blacktriangleright \Gamma; \Pi \vdash \mathtt{unfold}(a) : B[\mu \mathrm{A}.B]}$$

$$\frac{\Psi \blacktriangleright A \quad \Psi \blacktriangleright B}{\Psi \rhd (A - B)}$$

$$\frac{\Psi \rhd X \quad \Psi \blacktriangleright B}{\Psi \blacktriangleright (X \oslash B)}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \mathrm{b}:B \vdash a:A}{\Psi \rhd \Gamma \vdash \lambda \mathrm{b}.a : (A - B)}$$

$$\frac{\Psi \rhd \Gamma \vdash x:X \quad \Psi \blacktriangleright \Gamma \mid \Pi \vdash b:B}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash (x \mid b) : (X \oslash B)}$$

$$\frac{\Psi \rhd \Gamma \vdash f : (A - B) \quad \Psi \blacktriangleright \Gamma \mid \Pi \vdash b : B}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash (f)b : A}$$

$$\frac{\Psi \blacktriangleright \Gamma, \mathrm{x}:X \mid \mathrm{b} :\vdash d : D \quad \Psi \blacktriangleright \Gamma \mid \Pi \vdash c : (X \oslash B)}{\Psi \blacktriangleright \Gamma \mid \Pi \vdash c \ \mathtt{to}\ \mathrm{x}\ \mathtt{and}\ \mathrm{b}\ \mathtt{in}\ d : D}$$

Figure 8.7: Rules for closed LFPC.

Chapter 9

Quotient Relations and Parametric Models

In compact models of RLNL, it is natural to compare the fixed points defined by the combinators derived in Chapter 8 with the canonical fixed points obtained from compactness in Chapter 6. Here we use a notion of quotient diparametricity to make this comparison. We observe that, in the concrete categories of domains that motivate the subject, the two coincide and we derive conditions on compact adjunctions providing an abstract form of this result. These conditions may be seen as a typed analogue of Morris's characterization of lambda-calculus models in which the Park Coincidence holds [2].

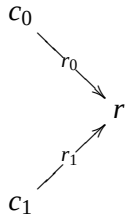
9.1 Quotient Diparametricity

The type of uniformity that characterizes the canonical fixed point operation in Chapter 6 can be viewed as a particular form of parametricity. We use the graph category framework for relational parametricity described in Appendix C together with a particular form of relation called a 'quotient relation'. For a theory of parametricity to apply to the interpretation of typed terms, the operations that interpret the type constructions must lift to the structure used to define parametricity, in this case graphs categories of quotient relations. One way of doing this uses an internal logic, but here we describe our liftings directly in terms of the categorical structure. This brings out just how little structure is required in the case of quotient relations.

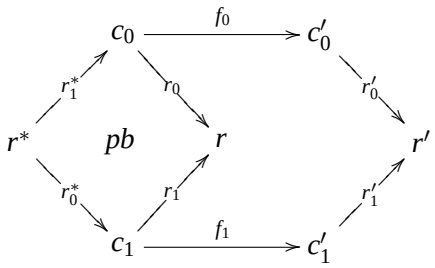
9.1.1 Quotient Relations

Categorically, binary relations may be represented by spans. Here we develop a theory of relations represented by cospans. We develop this theory using pull-backs, but a more general theory could be developed using Yoneda representations. The important point is the specialized notion of relation that results.

Definition 9.1.2 *Given a category C with pull-backs, the category* $\mathsf{Q}C$ *of* cospan relations *over C has cospans r_i in C for objects*

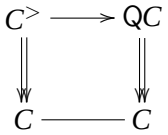


and pairs of maps f_i *such that* $r'_0 \circ f_0 \circ r^*_1 = r'_1 \circ f_1 \circ r^*_0$ *for arrows,*

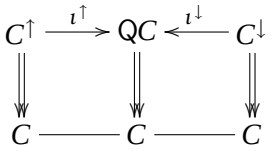


where r^*_1 *is a pull-back of* r_1 *along* r_0 *and* r^*_0 *is the corresponding pull-back of* r_0 *along* r_1*. Composition and identities are given by composition and identities in* $C \times C$*.*

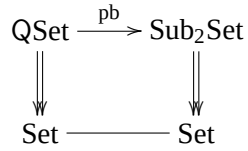
We think of a cospan $r_i : c_i \to r$ as relating those pairs in $c_0 \times c_1$ sent to the same element of r. The definition of arrow does not depend on the choice of pull-back r^*. In fact, any weak pull-back will do, so C need only have weak pull-backs. The properties of (weak) pull-backs ensure that composition in $C \times C$ lifts to $\mathsf{Q}C$. The categories $\mathsf{Q}C$ and C form the edge and vertex categories of a graph category with source and target functors as in $C^{>}$.



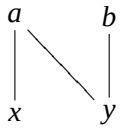
There is an identity-on-objects functor from $C^{>}$ to $\mathsf{Q}C$, and although it is not generally full or faithful it extends the full and faithful graph embeddings of $C^{\downarrow}$ and $C^{\uparrow}$ into $C^{>}$ to full and faithful graph embeddings into $\mathsf{Q}C$.



While in Appendix C we present the objects of $\mathrm{Sub}_2\mathsf{Set}$ as subobjects of products, these correspond to equivalence classes of jointly monic spans. If we use pull-backs to cast the objects of QSet as jointly monic spans, we obtain a full and faithful embedding of QSet into $\mathrm{Sub}_2\mathsf{Set}$.



This means that the notion of arrow in QSet matches the notion of arrow in $\mathrm{Sub}_2\mathsf{Set}$ which corresponds to the usual logical relations definition. Note, however, that not every binary relation over Set is represented by the pull-back of a cospan. For example, no cospan relation relates



without also relating b and x. This is a generic counter-example in that a binary relation is represented by the pull-back of a cospan *iff* it is zig-zag complete.

Takeyama and Tennent proposed zig-zag completeness [36] as a characterization of relations induced between different concrete data types that represent the same abstract data type.

Definition 9.1.3 (Takeyama and Tennent) *An n-ary relation r is* zig-zag complete *if, for each permutation* σ*,*

$$\sigma r(a,\mathbf{x}) \quad \text{and} \quad a \sim b \quad \text{imply} \quad \sigma r(b,\mathbf{x}),$$

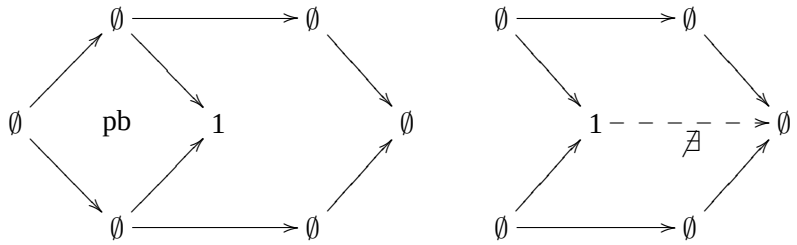
where $a \sim b$ *if there exists y such that* $\sigma r(a,\ldots,y,\ldots)$ *and* $\sigma r(b,\ldots,y,\ldots)$.

Proposition 9.1.4 *Over* Set*, an n-ary relation r is represented by the wide pull-back of an n-ary cospan* iff *it is zig-zag complete.*

When $n = 2$, zig-zag completeness can be described in terms of composition of binary relations. In a category R with an involution $(\cdot)^{\mathrm{o}} : R^{\mathrm{op}} \to R$, an arrow r, is *difunctional* [26] if $r \circ r^{\mathrm{o}} \circ r = r$. This condition is well known in the context of relations between algebras. Ordinarily R is the category of relations Rel$C$ over some regular category C. See Meisen [26] for a study of the relationship between pull-back spans and difunctional relations over categories other than Set.

A regular category C is *Malt'cev* if every arrow in Rel$C$ is difunctional. For example, the category of groups is Malt'cev. Another example is $\mathrm{Set}^{\mathrm{op}}$ [30]. This is interesting because, while $\mathrm{Rel}(\mathrm{Set}^{\mathrm{op}})$ uses spans in $\mathrm{Set}^{\mathrm{op}}$ to represent arrows, we are using spans in $\mathrm{Set}^{\mathrm{op}}$ to represent objects in $\mathrm{Sub}_2\mathrm{Set}$ which correspond to arrows in RelSet and, either way, the arrows we get are all difunctional. What about the category we haven't mentioned, $\mathrm{Sub}_2(\mathrm{Set}^{\mathrm{op}})$? The category QSet is equivalent to its full subcategory of jointly epic cospans, and this is equivalent to the opposite of $\mathrm{Sub}_2(\mathrm{Set}^{\mathrm{op}})$.

There is nearly an isomorphism between QSet and $\exists_{ver}\mathrm{Set}^{>}$, the category of cospan diagrams with the vertex component of arrows existentially quantified. The exceptions occur with arrows to the identity cospan on empty sets from other cospans on empty sets.

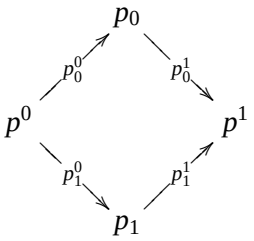


When C is the category of non-empty sets or the category of pointed sets and point preserving functions, the category of cospan relations QC is isomorphic to the cospan category $\exists_{ver}C$ which is equivalent to the opposite of $\mathrm{Sub}_2(C^{\mathrm{op}})$.

9.1.5 Push-me-pull-you's

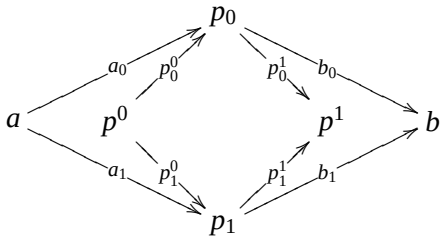
There exists a pleasantly symmetrical construction K which is equivalent to Q when applied to pull-back categories.

Definition 9.1.6 *Given any category C, the category* KC *of* push-me-pull-you's *over C has, for objects, commutative diamonds* p_i^j

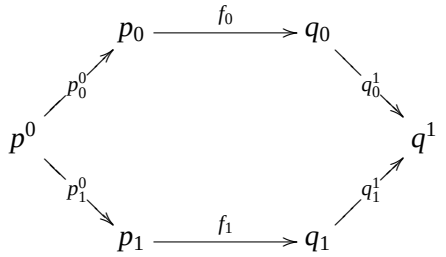


such that any span a_i that commutes with the cospan p_i^1 commutes with any cospan b_i that commutes with the span p_i^0,

$$p_0^1 \circ a_0 = p_1^1 \circ a_1 \quad \text{and} \quad p_0^0 \circ b_0 = p_1^0 \circ b_1 \quad \Rightarrow \quad b_0 \circ a_0 = b_1 \circ a_1,$$



and, for arrows, pairs of maps f_i such that $q_0^1 \circ f_0 \circ p_0^0 = q_1^1 \circ f_1 \circ p_1^0$.



Composition and identities are given by composition and identities in $C \times C$.

The push-me-pull-you condition abstracts the property of (weak) pull-backs that ensures composition lifts from $C \times C$. Identities lift from $C \times C$ because the diamonds commute. Informally, the span generates pairs, while the cospan tests pairs. Commutativity says every pair generated must test good. The push-me-pull-you condition says every pair that tests good must be generated. The objects of $\mathsf{K}C$ include all (weak) pull-backs and (weak) push-outs in C. Over Set or any C with both pull-backs and push-outs, a diamond is a push-me-pull-you *iff* the pull-back of the cospan commutes with the push-out of the span.

Proposition 9.1.7 (Robinson) *A choice of pull-backs in C gives an equivalence between the categories $\mathsf{Q}C$ and $\mathsf{K}C$ which is a graph equivalence over the identity on C.*

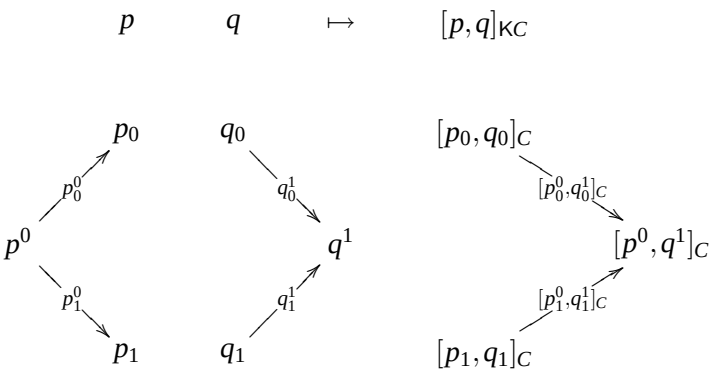
$$\begin{array}{ccc} \mathsf{Q}C & \xrightarrow{\text{pb}} & \mathsf{K}C \\ \Downarrow & & \Downarrow \\ C & = & C \end{array}$$

Proof. One direction of the equivalence takes a cospan in $\mathsf{Q}C$ to its chosen pull-back diamond in $\mathsf{K}C$ and arrow pairs to themselves. The other direction takes a diamond in $\mathsf{K}C$ to its cospan part in $\mathsf{Q}C$ and arrows pairs to themselves. These two functors give the identity on $\mathsf{Q}C$ and the endofunctor on $\mathsf{K}C$ that normalizes push-me-pull-you's to pull-back's. Any push-me-pull-you is isomorphic to any (weak) pull-back diamond with the same cospan. □

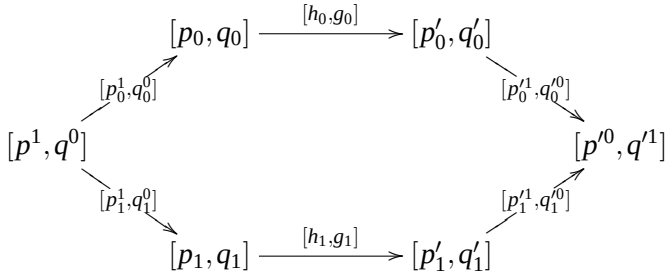
Proposition 9.1.8 *Given a choice of pull-backs in C, closed structure on C lifts to closed structure on $\mathsf{K}C$.*

$$\begin{array}{ccc} (\mathsf{K}C)^{\text{op}} \times \mathsf{K}C & \xrightarrow{[-,+]_{\mathsf{K}C}} & \mathsf{K}C \\ \Downarrow & & \Downarrow \\ C^{\text{op}} \times C & \xrightarrow{[-,+]_C} & C \end{array}$$

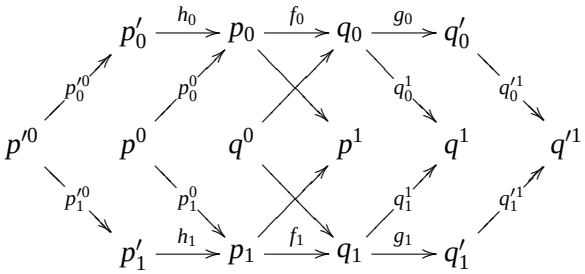
Proof. The cospan part of $[-,+]_{\mathsf{K}C}$ is given by applying $[-,+]_C$ to the cospan of the covariant argument and the span of the contravariant argument. The span part is given by the pull-back of the cospan part.



The cospan $[p_i^0, q_i^1]$ internalizes the test on arrow pairs in the definition of $\mathsf{K}C$. The pull-back span then generates all pairs that test good. We must check that $[h_i, g_i]$ carries these to good pairs according to the cospan $[p_i'^0, q_i'^1]$. Note that the span $[p_i^1, q_i^0]$ does not generally generate all the pairs in $[p_i^0, q_i^1]^*$—the contravariant place in the function space does not generally preserve pull-backs—so we cannot appeal to the diagram



Instead, consider a pair $f_i : p_i \to q_i$ in $[p^0, q^1]^*$. We know this gives an arrow from p to q because $[p_i^0, q_i^1]$ internalises the arrow test.

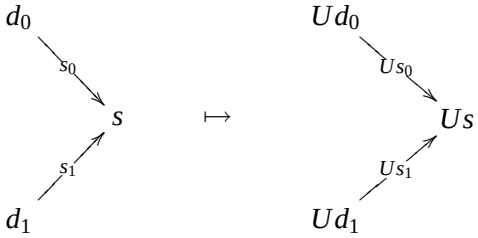


By the push-me-pull-you properties of p_i^j and q_i^j, the composite pair $h_i \circ f_i \circ g_i$, which is $[h_i, g_i]$ applied to f_i, gives an arrow from p' to q', but then this pair tests good according to the cospan $[p_i'^0, q_i'^1]$ which internalizes the arrow test from p' to q'. □

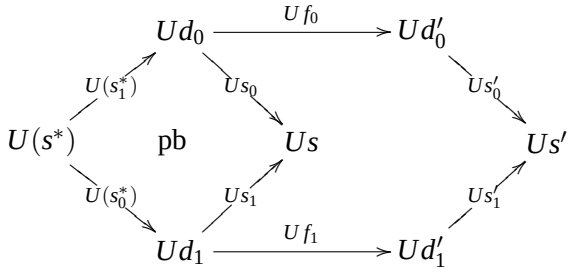
Corollary 9.1.9 *Given a choice of pull-backs in C, closed structure on C lifts to* Q*C.*

9.1.10 Componentwise Liftings

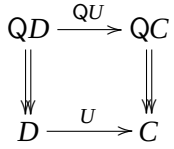
If $U : D \to C$ is a pull-back functor (functor preserving pull-backs between categories with pull-backs), then it lifts componentwise to a functor $\mathsf{Q}U : \mathsf{Q}D \to \mathsf{Q}C$.



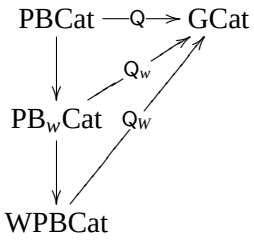
Given an arrow pair f_i in $\mathsf{Q}D$, we check that the image Uf_i is an arrow pair in $\mathsf{Q}C$ by taking, for our pull-back of Us_i, the image of a pull-back of s_i.



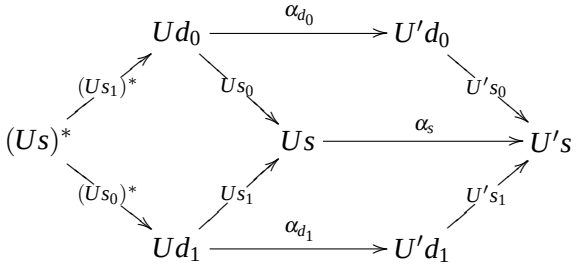
Taken with U, the functor $\mathsf{Q}U$ gives a graph functor.



Definition 9.1.2 thus gives the object part of a functor $\mathsf{Q} : \mathrm{PBCat} \to \mathrm{GCat}$ from the category of pull-back categories to the category of graph categories. In fact, the construction extends to functors taking pull-backs to weak pull-backs and to functors preserving weak pull-backs between categories with weak pull-backs.



These Q extend to 2-functors: the functors $\mathsf{Q}U$ act componentwise on the cospans that make up the objects of $\mathsf{Q}D$, so the components of any natural transformation $\alpha : U \Rightarrow U'$ also give a graph transformation from $\mathsf{Q}U$ to $\mathsf{Q}U'$.



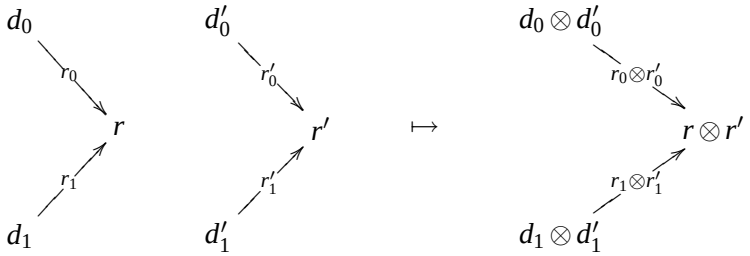
When D has pull-backs, $D \times D$ has pull-backs. These are computed componentwise, so $\mathsf{Q}(D \times D)$ is isomorphic to $\mathsf{Q}D \times \mathsf{Q}D$.

$$\begin{array}{ccc} \mathrm{GCat} \times \mathrm{GCat} & \xrightarrow{\times} & \mathrm{GCat} \\ \uparrow{\scriptstyle \mathsf{Q}\times\mathsf{Q}} & \cong & \uparrow{\scriptstyle \mathsf{Q}} \\ \mathrm{PBCat} \times \mathrm{PBCat} & \xrightarrow{\times} & \mathrm{PBCat} \end{array}$$

Any binary operation M on $D \times D$ that preserves pull-backs therefore lifts componentwise to a binary operation on $\mathsf{Q}D \times \mathsf{Q}D$.

$$\mathsf{Q}D \times \mathsf{Q}D \cong \mathsf{Q}(D \times D) \xrightarrow{\mathsf{Q}M} \mathsf{Q}C$$

For example, the smash product $\otimes$ on $\mathrm{Cppo}_\perp$ can be expressed as a pull-back (the image under lifting of the product expressed as a pull-back in Cpo) and so preserves pull-backs. It therefore lifts componentwise to $\mathrm{QCppo}_\perp$.



If pull-backs are used to embed $\mathrm{QCppo}_\perp$ into $\mathrm{Sub}_2\mathrm{Cppo}_\perp$, then $\mathsf{Q}\otimes$ corresponds to the smash product $\boxtimes$ [9] which is adjoint to the parametric function space $[-,+]_{\mathrm{Sub}_2\mathrm{Cppo}_\perp}$.

$$\begin{array}{ccc} \mathrm{QCppo}_\perp \times \mathrm{QCppo}_\perp & \xrightarrow{\mathsf{Q}\otimes} & \mathrm{QCppo}_\perp \\ \downarrow{\scriptstyle \mathrm{pb}\times\mathrm{pb}} & & \uparrow{\scriptstyle \mathrm{po}} \downarrow{\scriptstyle \mathrm{pb}} \\ \mathrm{Sub}_2\mathrm{Cppo}_\perp \times \mathrm{Sub}_2\mathrm{Cppo}_\perp & \xrightarrow{\boxtimes} & \mathrm{Sub}_2\mathrm{Cppo}_\perp \end{array}$$

While $\mathsf{Q}\otimes$ is equivalent to the composite $\mathrm{po} \circ \boxtimes \circ (\mathrm{pb} \times \mathrm{pb})$, the functor $\mathsf{Q}\otimes$ is given by a natural, componentwise definition, while $\boxtimes$ is defined using an existential quantification and a completion or, more abstractly, certain coequalizers in the category of lift algebras.

How might the above apply to operations of mixed variance? When D and D^{op} have pull-backs, $D^{\mathrm{op}} \times D$ has pull-backs. If F is a pull-back functor on $D^{\mathrm{op}} \times D$, we obtain a graph functor $\mathsf{Q}F$ on $\mathsf{Q}(D^{\mathrm{op}}) \times \mathsf{Q}D \cong \mathsf{Q}(D^{\mathrm{op}} \times \mathsf{Q}D)$. The following proposition then gives us a graph functor on $(\mathsf{Q}D)^{\mathrm{op}} \times \mathsf{Q}D$.

Proposition 9.1.11 *When D and D^{op} have pull-backs, $\mathsf{Q}(D^{\mathrm{op}})$ is equivalent to $(\mathsf{Q}D)^{\mathrm{op}}$.*

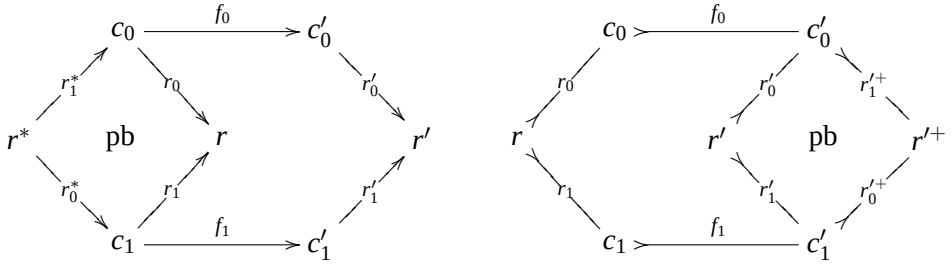
Proof. The pull-back operations on the objects of $\mathsf{Q}(D^{\mathrm{op}})$ and $(\mathsf{Q}D)^{\mathrm{op}}$ give the equivalence. □

In other words, on those D that have both pull-backs and push-outs, the 2-functor Q commutes up to an equivalence with the opposite category 2-functor $(\cdot)^{\mathrm{op}} : \mathrm{Cat} \to \mathrm{Cat}^{\mathrm{co}}$.

$$\begin{array}{ccc} \mathrm{GCat} & \xrightarrow{(\cdot)^{\mathrm{op}}} & \mathrm{GCat}^{\mathrm{co}} \\ \uparrow{\scriptstyle \mathsf{Q}} & \simeq & \uparrow{\scriptstyle \mathsf{Q}^{\mathrm{co}}} \\ \mathrm{PBPOCat} & \xrightarrow{(\cdot)^{\mathrm{op}}} & \mathrm{PBCat}^{\mathrm{co}} \end{array}$$

This is a little surprising. It means $\mathsf{Q}C$ is equivalent to $(\mathsf{Q}(C^{\mathrm{op}}))^{\mathrm{op}}$ when C has both pull-backs and push-outs. Both $\mathsf{Q}C$ and $(\mathsf{Q}(C^{\mathrm{op}}))^{\mathrm{op}}$ have pairs of maps for arrows, but the objects are different

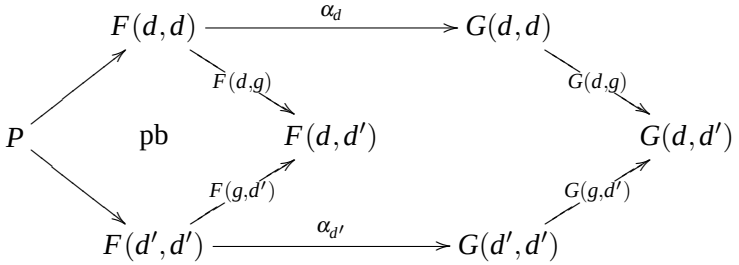
and arrows are tested differently. Take for example QSet. In comparison with the category $\mathrm{Sub}_2\mathrm{Set}$ of all binary relations over Set, QSet has the same notion of arrow but only represents some relations (the difunctional ones), while $(\mathsf{Q}(\mathrm{Set}^{\mathrm{op}}))^{\mathrm{op}}$, having spans for objects, represents all relations but has a more liberal notion of arrow tested using a push-out (pull-back in $\mathrm{Set}^{\mathrm{op}}$) of the codomain span.



Note, however, that this method of lifting mixed variance functors goes all wrong when applied to closed structure. Assuming C^{op} has pull-backs, these are not generally preserved by the contravariant parts $[-,c]_C : C^{\mathrm{op}} \to C$ of a function space functor $[-,+]_C$. However, given just pull-backs in C, any function space functor $[-,+]_C : C^{\mathrm{op}} \times C \to C$ lifts to a reflexive graph functor $[-,+]_{\mathsf{Q}}C : (\mathsf{Q}C)^{\mathrm{op}} \times \mathsf{Q}C \to \mathsf{Q}C$, with or without pull-backs in C^{op} and whether or not those in C are preserved.

9.1.12 Uniformity as Quotient Diparametricity

In [28], Mulry internalizes uniformity (for transformations of fixed point type) using a notion of 'strong dinaturality'. A transformation α is *strongly dinatural* if, for all $g : d \to d'$,

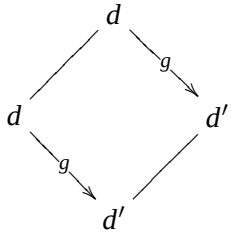


Assuming C has pull-backs, this is parametricity with respect to certain objects in $\mathsf{Q}C$.

Proposition 9.1.13 *Strong dinatural transformations from F to G are identical with diparametric transformations from $\mathsf{Q}F \circ ((\imath^{\uparrow} \circ \Downarrow) \times \imath^{\downarrow})$ to $\mathsf{Q}G \circ ((\imath^{\uparrow} \circ \Downarrow) \times \imath^{\downarrow})$.*

$$(D^{\downarrow})^{\mathrm{op}} \times D^{\downarrow} \xrightarrow{\Downarrow \times D^{\downarrow}} (D^{\mathrm{op}})^{\uparrow} \times D^{\downarrow} \xrightarrow{\imath^{\uparrow} \times \imath^{\downarrow}} \mathsf{Q}(D^{\mathrm{op}}) \times \mathsf{Q}D \xrightarrow[\text{iso}]{} \mathsf{Q}(D^{\mathrm{op}} \times D) \underset{\mathsf{Q}F}{\overset{\mathsf{Q}G}{\rightrightarrows}} \mathsf{Q}C$$

Diamonds of the form



are pull-backs, This means the above functor $(\iota^{\uparrow} \circ \Downarrow)$ can be written as $\iota^{\downarrow}$ followed by a pull-back operation from $(\mathsf{Q}D)^{\mathrm{op}}$ to $\mathsf{Q}(D^{\mathrm{op}})$.

$$\begin{array}{ccc} (D^{\downarrow})^{\mathrm{op}} & \xrightarrow{(\iota_D^{\downarrow})^{\mathrm{op}}} & (\mathsf{Q}D)^{\mathrm{op}} \\ \downarrow \Downarrow & & \downarrow \mathrm{pb} \\ (D^{\mathrm{op}})^{\uparrow} & \xrightarrow{\iota^{\uparrow}_{D^{\mathrm{op}}}} & \mathsf{Q}(D^{\mathrm{op}}) \end{array}$$

The above functors on $(D^{\downarrow})^{\mathrm{op}} \times D^{\downarrow}$ therefore factor through functors on $(\mathsf{Q}D)^{\mathrm{op}} \times \mathsf{Q}D$.

$$(D^{\downarrow})^{\mathrm{op}} \times D^{\downarrow} \xrightarrow{(\iota^{\downarrow})^{\mathrm{op}} \times \iota^{\downarrow}} (\mathsf{Q}D)^{\mathrm{op}} \times \mathsf{Q}D \xrightarrow{\mathrm{pb} \times \mathsf{Q}D} \mathsf{Q}(D^{\mathrm{op}}) \times \mathsf{Q}D \xrightarrow[\mathrm{iso}]{} \mathsf{Q}(D^{\mathrm{op}} \times D) \overset{\mathsf{Q}G}{\underset{\mathsf{Q}F}{\rightrightarrows}} \mathsf{Q}C$$

Corollary 9.1.14 *A family of maps is strongly dinatural if it gives a diparametric transformation from* $\mathsf{Q}F \circ (\mathrm{pb} \times \mathsf{Q}D)$ *to* $\mathsf{Q}G \circ (\mathrm{pb} \times \mathsf{Q}D)$.

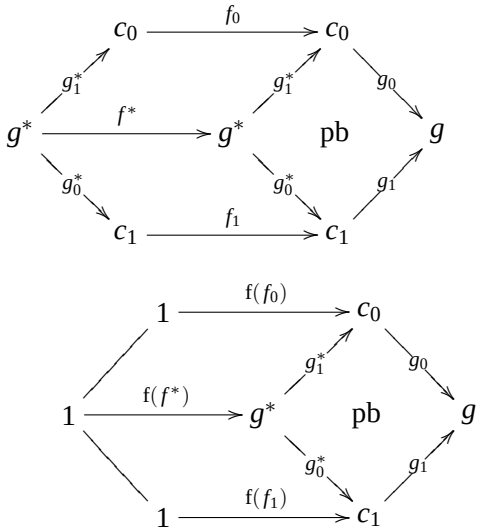
Now suppose transformation α has the type of a fixed point operator. That is, suppose F is the function space $[-,+]$ of some closed category C and G is the second projection π_C on $C^{\mathrm{op}} \times C$. The function space lifts to a functor $[+,-]_{\mathsf{Q}C}$ given by $\mathsf{Q}[-,+] \circ (\mathrm{pb} \times \mathsf{Q}C)$. The second projection $\pi_{\mathsf{Q}C}$ on $(\mathsf{Q}C)^{\mathrm{op}} \times \mathsf{Q}C$ throws away its first argument and so equals $\mathsf{Q}\pi_C \circ (\mathrm{pb} \times \mathsf{Q}C)$. Therefore, by the Corollary, a diparametric transformation from $[+,-]_{\mathsf{Q}C}$ to $\pi_{\mathsf{Q}C}$ gives a strong dinatural transformation from the functor $[+,-]$ to the functor π_C. In this case, however, the converse holds.

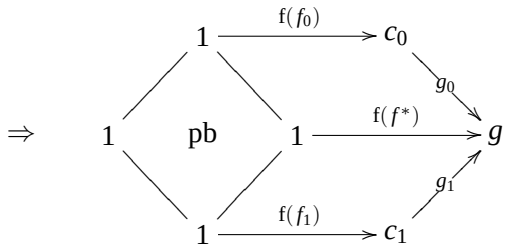
Proposition 9.1.15 *A family of maps* $\mathrm{f}_c : [c,c] \to c$ *is strongly dinatural* iff *it gives a diparametric transformation from* $[-,+]_{\mathsf{Q}C} : (\mathsf{Q}C)^{\mathrm{op}} \times \mathsf{Q}C \to \mathsf{Q}C$ *to* $\pi_{\mathsf{Q}C} : (\mathsf{Q}C)^{\mathrm{op}} \times \mathsf{Q}C \to \mathsf{Q}C$.

Proof. We check that the *a priori* stronger parametricity condition is implied by strong dinaturality. Strong dinaturality for these two functors says, modulo an internalisation, that we have $g(\mathrm{f}(f_0)) = \mathrm{f}(f_1)$, for any f_0, f_1 and g such that $g \circ f_0 = f_1 \circ g$.

$$\begin{array}{ccc} c_0 & \xrightarrow{f_0} & c_0 \\ \downarrow g & & \downarrow g \\ c_1 & \xrightarrow{f_1} & c_1 \end{array} \quad \Rightarrow \quad \begin{array}{ccc} 1 & \xrightarrow{\mathrm{f}(f_0)} & c_0 \\ \| & & \downarrow g \\ 1 & \xrightarrow{\mathrm{f}(f_1)} & c_1 \end{array}$$

Diparametricity says, modulo an internalisation, that we have $g_0(\mathrm{f}(f_0)) = g_1(\mathrm{f}(f_1))$, for any f_0, f_1 and cospan (g,h) such that $g \circ f_0 \circ g_1^* = h \circ f_1 \circ g_0^*$.





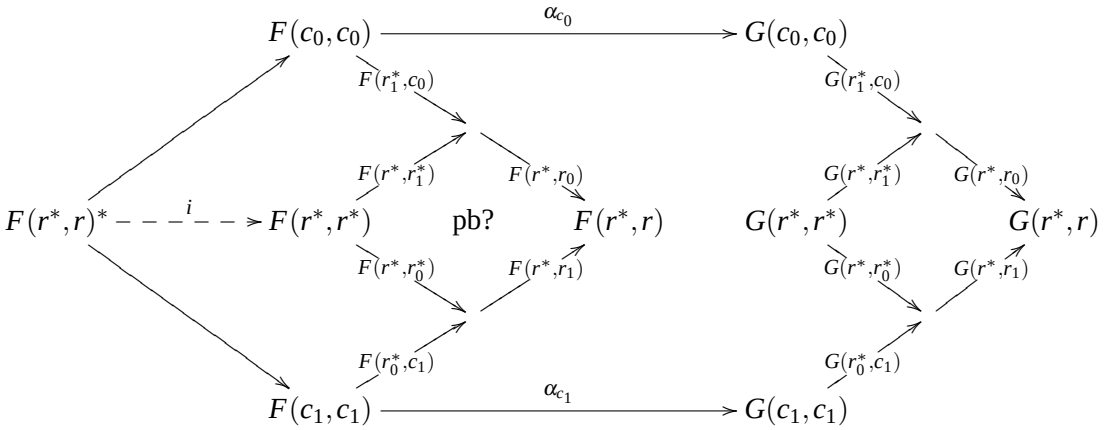
This follows from strong dinaturality applied to f^*, f_0 and g_1^* and to f^*, f_1 and g_0^*, where f^* is the unique span map from $(f_0 \circ g_1^*, f_1 \circ g_0^*)$ to (g_1^*, g_0^*). □

.

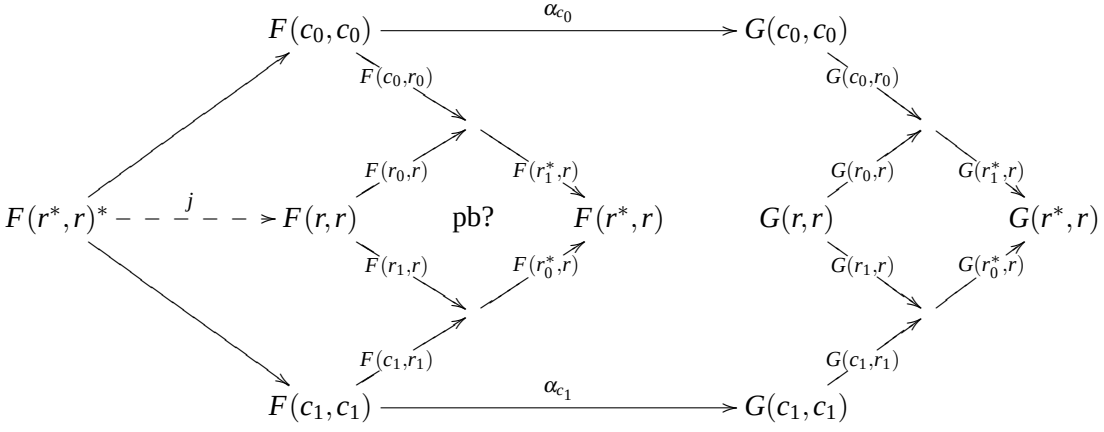
We have given a direct proof, but the Proposition is just one consequence of the following Lemma.

Lemma 9.1.16 *If $F : C^{\mathrm{op}} \times C \to C$ preserves pull-backs in it's covariant component or takes push-outs to pull-backs in it's contravariant component, then a family of maps $\alpha_c : F(c,c) \to G(c,c)$ is strong dinatural if and only if it is diparametric with respect to $\mathbb{Q}C$.*

Proof. Suppose the covariant functor $F(r^*,+)$ preserves pull-backs. Then $(F(r^*,r_1^*),F(r^*,r_0^*))$ is a pull-back of $(F(r^*,r_0),F(r^*,r_1))$ and we obtain a map i from the pull-back $F(r^*,r)^*$ to $F(r^*,r^*)$. Diparametricity at r_i then follows from strong dinaturality at r_1^* (upside-down in the diagram below) pasted along α_{r^*} to strong dinaturality at r_0^*.



On the other hand, if $F(-,r)$ takes push-outs to pull-backs. Then $(F(r_0,r),F(r_1,r))$ is a pull-back of $(F(r_1^*,r),F(r_0^*,r))$ and we obtain a map j from $F(r^*,r)^*$ to $F(r,r)$. Diparametricity at r_i then follows from strong dinaturality at r_0 and at r_1.



□

Note that neither half of Lemma 9.1.16 uses the full bifunctorality of F and G. If $F(\mathrm{id}, r_0) \circ F(r_1^*, \mathrm{id}) \neq F(r_1^*, \mathrm{id}) \circ F(\mathrm{id}, r_0)$, so that $F(r_1^*, r_0)$ is not well defined, then we can take $\mathsf{Q}F((r_0, r_1), (s_0, s_1))$ to be the cospan $(F(\mathrm{id}, r_0) \circ F(r_1^*, \mathrm{id})), F(\mathrm{id}, r_1) \circ F(r_0^*, \mathrm{id})$, in which case the first half of the proof goes though, or we can take it to be $(F(r_1^*, \mathrm{id}) \circ F(\mathrm{id}, r_0), F(r_0^*, \mathrm{id}) \circ F(\mathrm{id}, r_1))$ in which case the second half goes though.

When $F(-.+)$ is the function space $[-,+]$ associated with some tensor product $\otimes$, so that $F[r,+]$, is a right adjoint to $r \otimes (\cdot)$ and, hence, preserves pull-backs, then, by the Lemma, strong dinaturality from $[-,+]$ to $G(-,+)$ is equivalent to diparametricity with respect to $\mathsf{Q}C$. Also, in the case of function spaces, the operator $\mathsf{Q}[-,+]$ is functorial.

9.2 Domain Theoretic Models of Recursive Linear Types

The addition of recursive types to LNL is motivated by the existence of invariants in domain-theoretic adjunctions. The fact that, in these adjunctions, the invariants are actually Freyd algebras allows us to interpret nested and mixed variance recursive types. However, we do not see that the equational theory of RLNL *requires* that a recursive type be interpreted by a Freyd algebra. The term model does not then form a compact adjunction and so domain-theoretic models are special.

To get around a lack of Freyd algebras in the term model, the general notion of RLNL model we give in Section 9.2.1 uses a technical notion of pre Freyd algebra. In Section 9.2.5 we describe how a fixed point combinator can be used to show that the term model has pre Freyd algebras.

In the notion of uniformity defined in Chapter 6 variation is limited to strict maps. In Section 9.2.7 we observe that this can be built into a modified notion of quotient relation, although in RLNL models variation is already limited in this way.

This means that the transformation interpreting any fixed point combinator is uniform, so long as the structure modeling the term derivation lifts to quotient relations. Since compact adjunctions have canonical fixed points and these are characterized by uniformity when the right adjoint is injective on objects, In Section 9.2.9 we give a definition of domain-theoretic adjunction that ensures that all structure lifts to quotient relations. This ensures that fixed points defined by combinators coincide with the canonical fixed points of the model, which, in partial order models, coincide with least fixed points.

9.2.1 RLNL Models

The type theory LNL is interpreted in a symmetric monoidal/cartesian closed adjunction $L \dashv U : D \to C$. Linear types and terms in context are interpreted using objects and transformations in the monoidal closed category D and linear types and terms, using functors and transformations in the cartesian closed category C.

Definition 9.2.2 (Benton [3]) *An* LNL model *in a symmetric monoidal/cartesian closed $L \dashv U : D \to C$ is given by a function from types in context to families of objects indexed by tuples of objects in D that respects the equations*

$$\begin{aligned}
[\![\mathrm{A}_1, \ldots, \mathrm{A}_k \blacktriangleright \mathrm{A}_k]\!] &= d_k, \text{ at } (d_1, \ldots, d_k) \\
[\![\Psi \rhd (X \multimap Y)]\!] &= [[\![\Psi \rhd X]\!], [\![\Psi \rhd Y]\!]]_C \\
[\![\Psi \blacktriangleright (A \multimap B)]\!] &= [[\![\Psi \blacktriangleright A]\!], [\![\Psi \blacktriangleright B]\!]]_D \\
[\![\Psi \blacktriangleright \mathrm{L}X]\!] &= L[\![\Psi \rhd X]\!] \\
[\![\Psi \rhd \mathrm{U}A]\!] &= U[\![\Psi \blacktriangleright A]\!]
\end{aligned}$$

$$[\![\Psi \rhd (X \times Y)]\!] = [\![\Psi \rhd X]\!] \times [\![\Psi \rhd Y]\!]$$

$$\begin{aligned}\llbracket \Psi \blacktriangleright (X \otimes Y) \rrbracket &= \llbracket \Psi \blacktriangleright X \rrbracket \otimes \llbracket \Psi \blacktriangleright Y \rrbracket \\ \\ \llbracket \Psi \rhd 1 \rrbracket &= 1 \\ \llbracket \Psi \blacktriangleright \mathrm{I} \rrbracket &= I\end{aligned}$$

and a function from terms in context to transfomations that respects the equations

$$\begin{aligned}\mathrm{dom}\llbracket \Psi \rhd x : \mathrm{X} \vdash \mathtt{f}(x) : \mathrm{Y} \rrbracket &= \llbracket \Psi \rhd \mathrm{X} \rrbracket \\ \mathrm{cod}\llbracket \Psi \rhd x : \mathrm{X} \vdash \mathtt{f}(x) : \mathrm{Y} \rrbracket &= \llbracket \Psi \rhd \mathrm{Y} \rrbracket \\ \llbracket \Psi \rhd \mathrm{x}_1 : X_1, \ldots, \mathrm{x}_n : X_n \vdash \mathrm{x}_n : X_n \rrbracket &= \pi_n = \lambda_C \circ t_{\llbracket \Psi \rhd X_1 \rrbracket \times \cdots \times \llbracket \Psi \rhd X_n \rrbracket} \\ \llbracket \Psi \blacktriangleright \mathrm{x}_1 : X_1, \ldots, \mathrm{x}_n : X_n \shortmid \mathrm{a} : A \vdash \mathrm{a} : A \rrbracket &= \lambda_D \circ ((u \circ L t_{\llbracket \Psi \rhd X_1 \rrbracket \times \cdots \times \llbracket \Psi \rhd x_n \rrbracket}) \otimes \mathrm{id}_{\llbracket \Psi \blacktriangleright A \rrbracket}) \\ \llbracket \Psi \rhd \Gamma \vdash \mathtt{thunk}(a) : \mathrm{U}A \rrbracket &= U \llbracket \Psi \blacktriangleright \Gamma \vdash a : A \rrbracket \circ \eta \\ \llbracket \Psi \blacktriangleright \Gamma \vdash \mathtt{force}(x) : A \rrbracket &= \varepsilon \circ L \llbracket \Psi \rhd \Gamma \vdash x : \mathrm{U}A \rrbracket \\ \llbracket \Psi \blacktriangleright \Gamma \vdash \mathtt{produce}(x) : \mathrm{L}X \rrbracket &= L \llbracket \Psi \rhd \Gamma \vdash x : X \rrbracket \\ \llbracket \Psi \blacktriangleright \Gamma \shortmid \Pi, \Pi' \vdash b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a : A \rrbracket &= \llbracket \Psi \blacktriangleright \Gamma, \mathrm{x} : X \shortmid \Pi \vdash a : A \rrbracket \circ \\ & \quad d \circ (L\mathrm{id}_{\llbracket \Psi \rhd X \rrbracket} \otimes \llbracket \Psi \blacktriangleright \Gamma \shortmid \Pi' \vdash b : \mathrm{L}X \rrbracket) \circ \alpha_D\end{aligned}$$

By adding interpretations for the recursive linear type construction and the rules for introducing and eliminating recursive types, we extend the definition of LNL model to a definition of RLNL model. As we control the construction of recursive types using type contexts, our interpretation of recursive types is based on an interpretation of types in context. We interpret the nonlinear and linear type sequents $\Psi \rhd X$ and $\Psi \blacktriangleright A$ derivable in RLNL, as functors on $|D|^k$ where $|D|$ is the discreet category of objects of D, and k is the number of type variables in Ψ. These functors lift to functors on $(D^{\mathrm{op}} \times D)^k$. Except for the function types, the operations used to interpret types extend to functors on multiples of D. To accomodate function types, we can use functors on multiples of $D^{\mathrm{op}} \times D$. All these functors are structural with respect to the canonical structural actions.

Proposition 9.2.3 *In an LNL model, the interpretation of each type in context extends to a structural functor on a multiple of $D^{\mathrm{op}} \times D$.*

To interpret the introduction and elimination rules for recursive types we require an invariant for the endofunctor interpreting the type construction that appears in the recursive type. As the calculus allows recursive types to be nested, it is also necessary that the delivery of these invariants lifts to a functor with invariants of its own. The theory of structural algebraic compactness presented in Chapter 4 provides just such invariants; however, we would also like the term category to have the structure we require for our models and although we see how to construct canonical recursive maps to and from invariants in the term category, we do not see that these are unique. Let us say that a *pre Freyd algebra* is an algebra with a given recursive morphism from any coalgebra and a given algebra morphism to any algebra.

Definition 9.2.4 *An* RLNL(F) model *is an LNL(F) model in which* $\Psi \blacktriangleright \mathrm{b} : B[\mu \mathrm{A}.B[\mathrm{A}]] \vdash \mathtt{fold}(\mathrm{b}) : B[\mathrm{A}]$ *is interpreted as a pre Freyd algebra for the (parameterized) functor that interprets* $\Psi, \mathrm{A} \blacktriangleright B$.

9.2.5 Recursive Types in Term Models

The type theory LNL can be used to build a categorical model of itself. In the symmetric monoidal closed category D, objects are linear type expressions and arrows are equivalence

classes of derivable terms in singleton linear context. The equivalence is generated by the equational theory and alpha conversion on the variable appearing in the context. Composition is given by well-typed substitution of term for free variable and identities are given by the linear variable introduction rule.

We define a functor $\otimes : D \times D \to D$ by taking an arrow given by the pair of terms $\Psi \blacktriangleright \mathrm{a} : A \vdash b : B$ and $\Psi \blacktriangleright \mathrm{a}' : A' \vdash b' : B'$ to the arrow given by

$$\Psi \blacktriangleright \mathrm{c} : (A \otimes A') \vdash \mathrm{c}\ \mathtt{to}\ \mathrm{a}\ \mathtt{and}\ \mathrm{a}'\ \mathtt{in}\ (b.b') : (B \otimes B').$$

We define a functor $[-,+] : D^{\mathrm{op}} \times D \to D$ by taking an arrow given by $\Psi \blacktriangleright \mathrm{b} : B \vdash a : A$ and $\Psi \blacktriangleright \mathrm{a}' : A' \vdash b'[\mathrm{a}'] : B'$ to the arrow given by

$$\Psi \blacktriangleright \mathrm{g} : (A \multimap A') \vdash \lambda \mathrm{b}.b'[(\mathrm{g})a] : (B \multimap B').$$

We check that these operations are functorial. A cartesian closed category C is defined in the same way using nonlinear types and nonlinear sequents with singleton context, except the functor giving products takes an arrow given by the pair of terms $\Psi \rhd \mathrm{x} : X \vdash y : Y$ and $\Psi \rhd \mathrm{x}' : X' \vdash y' : Y'$ to the arrow given by $\Psi \rhd \mathrm{z} : (X \times X') \vdash (\mathtt{first}(\mathrm{z}), \mathtt{second}(\mathrm{z})) : (Y \times Y')$.

We define a functor $L : C \to D$ by applying the type constructor L to objects and taking an arrow given by $\rhd \mathrm{x} : X \vdash y[\mathrm{x}] : Y$ to the arrow given by $\blacktriangleright \mathrm{a} : \mathrm{L}X \vdash \mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{produce}(y[\mathrm{x}]) : \mathrm{L}Y$ and we define a functor $U : D \to C$ by applying the type constructor U to objects and taking an arrow given by $\blacktriangleright \mathrm{a} : A \vdash b[\mathrm{a}] : B$ to the arrow given by $\rhd \mathrm{x} : \mathrm{U}A \vdash \mathtt{thunk}(b[\mathtt{force}(\mathrm{x})]) : \mathrm{U}B$. We check that these operations are functorial.

By Corollary 7.2.4, L is left adjoint to U and from the proof of Proposition 7.2.3 we see that the unit at X is given by

$$\rhd \mathrm{x} : X \vdash \mathtt{thunk}(\mathtt{produce}(\mathrm{x})) : \mathrm{UL}X$$

and the counit at B is given by

$$\blacktriangleright \mathrm{a} : \mathrm{LU}B \vdash \mathrm{a}\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{force}(\mathrm{x}) : B.$$

In [24] it is shown that this construction gives the object part of an equivalence between a category of LNL theories and a category of symmetric monoidal/cartesian closed adjunctions, which means that LNL is *the* language for such adjunctions. We do not have a similar result for RLNL and compact adjunctions. However, in the term model of RLNL, we can use fixed point combinators to construct the morphisms into and out of an invariant, even if this does not make the invariant a Freyd algebra as there is nothing to say that these are the unique such morphisms.

Given a functor defined by a linear type expression $\mathrm{A} \blacktriangleright B[\mathrm{A}]$, consider the arrow given by

$$\blacktriangleright \mathrm{b} : B[\mu \mathrm{A}.B[\mathrm{A}]] \vdash \mathtt{fold}(\mathrm{b}) : \mu \mathrm{A}.B[\mathrm{A}]$$

as the structure morphism of an algebra. First, given an arrow p from C to $B[C]$ we require a recursive morphism given by an arrow from C to $\mu \mathrm{A}.B[\mathrm{A}]$. Such an arrow is a fixed point of the endofunction on $D(C, \mu \mathrm{A}.B[\mathrm{A}])$ given by composition with $p[\mathrm{c}]$ and $\mathtt{fold}(\mathrm{b})$. Internally, this endofunction is represented by

$$\lambda \mathrm{f}.\mathtt{thunk}(\lambda \mathrm{c}.\mathtt{fold}((B[\mathtt{force}(\mathrm{f})])p[\mathrm{c}])) : (\mathrm{U}(C \multimap \mu \mathrm{A}.B[\mathrm{A}]) \mathbin{\&} \mathrm{U}(C \multimap \mu \mathrm{A}.B[\mathrm{A}])),$$

where $B[f]$ is determined by the type expression $B[\mathrm{A}]$. Applying a fixed point combinator, we obtain a term of type $\mathrm{U}(C \multimap \mu \mathrm{A}.B[\mathrm{A}])$ which gives a term of type $(C \multimap \mu \mathrm{A}.B[\mathrm{A}])$ which uncurries to give the desired arrow from C to $\mu \mathrm{A}.B[\mathrm{A}]$. We check that this gives a recursive morphism. The arrow given by $\mathtt{fold}(\mathrm{b})$ has an inverse given by $\mathtt{unfold}(\mathrm{a})$ and a similar construction gives us a recursive morphism from $\mathtt{unfold}(\mathrm{a})$ to any algebra for B.

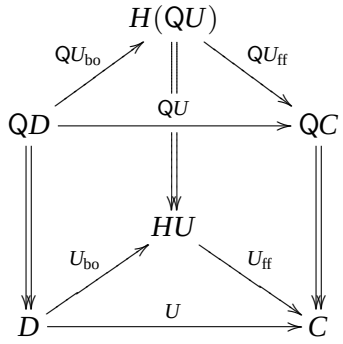
Proposition 9.2.6 *The monoidal category of the term model for RLNL(K) has a pre Freyd algebra for every endofunctor defined by a type expression in RLNL(K).*

9.2.7 Uniformity in RLNL Models

For uniformity to characterize canonical fixed points, it is necessary that g vary over *strict* maps only. In Mulry's setting, the notion of strong dinaturality must be correspondingly weakened. In that setting, strict maps are algebra morphisms and so Mulry uses the notion of strong dinatural transformation with variation restricted to algebra morphisms [28, Def. 3.11].

Before we consider the interpretation of RLNL, we observe that a factorization system in Cat can be used to describe the correspondingly weakened parametricity condition. Strict maps will be those in the image of some functor $U : D \to C$. In Mulry's setting, U is the forgetful functor from the category of LU-algebras.

In Cat every functor F factors as a bijective-on-objects functor F_{bo} followed by a full-and-faithful functor F_{ff}. If D is the domain of F, the interpolating category HF has the objects of D for objects and has hom sets given by $\mathrm{Hom}(\mathrm{F}(-),\mathrm{F}(+))$. This factorization system lifts to graph functors, because the operation H is the object part of a functor from Fun, the category of functors and functor squares commuting up to given natural isomorphisms, to Cat. A graph morphism gives a graph in Fun which H then sends to a graph category. Assuming U is a pull-back functor, we therefore obtain a graph $H(\mathsf{Q}U)$ by factoring the graph functor $\mathsf{Q}U$. Note that $H(\mathsf{Q}U)$ is not $\mathsf{Q}(HU)$. Our definition of the latter requires HU to have pull-backs (which it does, by the way, if U creates pull-backs).



When $L \dashv U$ is a closed adjunction, the closed structure on C lifts along U_{ff} to closed structure on HU by defining

$$[-,+]_{HU} \stackrel{\mathrm{def}}{=} [LU-,+]_D.$$

This closed structure commutes with U_{ff} up to a natural isomorphism $\tau : U[LU-,+]_D \Rightarrow [U-,U+]_C$ which is the internalisation of transposition. Similarly, closed structure on $\mathsf{Q}C$ lifts to closed structure on $H(\mathsf{Q}U)$ and we obtain a graph functor $[-,+]_{H(\mathsf{Q}U)} : H(\mathsf{Q}U)^{\mathrm{op}} \times H(\mathsf{Q}U) \to H(\mathsf{Q}U)$. This is the image under H of closed structure on the graph $\mathsf{Q}U$ over U in Fun.

Proposition 9.2.8 *A family of maps $\mathrm{f}_d : [Ud,Ud] \to Ud$ in C indexed by the objects of D is strongly dinatural with variation restricted to D* iff *it gives a diparametric transformation from $[-,+]_{H(\mathsf{Q}U)}$ to $\pi_{H(\mathsf{Q}U)}$.*

Remark. More generally, Section 6.2 uses uniformity to characterize the canonical recursive morphisms $\mathrm{R}(p,s)$ induced by a fixed point algebra. When the operation that produces canonical recursive morphisms is internalized to a collection of maps $\mathrm{R} : [b,Fb] \times [Fa,a] \to [b,a]$, this uniformity can also be viewed as diparametricity with respect to quotient relations. This generalizes the situation with the canonical fixed point operations.

In an RLNL model, types are interpreted by functors on (multiples of) D, so parametricity is already with respect to quotient relations over D. In the case of the fixed point type $(Ud \to Ud) \to Ud$ this amounts to restricting variation to strict maps.

9.2.9 Fixed Point Combinators in Domain Theoretic Models

Consider a compact symmetric monoidal/cartesian closed adjunction. The right adjoint is not obliged to be injective on objects, so we ask for this to ensure that the canonical fixed point operation is the unique uniform fixed point operation. The majority of the structure in the compact adjunction lifts to quotient relations: products and right adjoints preserve pull-backs, the closed structure lifts by Corollary 9.1.9, natural transformations are automatically (di)parametric and, using identity relations, Freyd algebras for (diagonalized) endofunctors that lift also lift. This leaves the left adjoint and monoidal multiplication. In concrete adjunctions between categories of partial orders these preserve pull-backs, so we ask for this explicitly.

Definition 9.2.10 *A* domain-theoretic monoidal adjunction *is a compact symmetric monoidal/cartesian closed adjunction whose right adjoint is injective on objects and whose left adjoint and monoidal multiplication preserve pull-backs.*

Consider an RLNL model in a compact adjunction. Although the compactness provides a model of recursive types, we do not see that our weakened notion of RLNL model is not obliged to follow this interpretation. We therefore ask for this explicitly. One might say that the Freyd algebras provide the canonical interpretation of recursive types.

Definition 9.2.11 *A* domain-theoretic model of RLNL *is a model in a domain-theoretic adjunction with recursive types interpreted by Freyd algebras.*

The natural transformations that interpret RLNL are quotient diparametric and all the structure that interprets the rules of RLNL preserves quotient diparametricity so every term is interpreted by a quotient diparametric transformation.

Proposition 9.2.12 *In a domain-theoretic model of RLNL the interpretation of any term gives a quotient diparametric transformation.*

Corollary 9.2.13 *In a domain-theoretic model of RLNL the interpretation of any fixed point combinator coincides with the canonical fixed point operator.*

Chapter 10

Directions for Future Work

Limits on space and time have forced us to ignore, as best we could, a number of interesting possibilities. We sketch those that seem most promising or intriguing, beginning with those we most hope to pursue in future.

10.1 Fox's Construction and Lemma 2.1.4

Loosely speaking Lemma 2.1.4 uses the structure maps of the algebras and coalgebas as unit and counit maps at the level of algebra/coalgebra morphisms. Similarly, Fox's construction uses the structure of commutative comonoids to build the maps required by products and a final object.

Given a monoidal category with mulitplication $\otimes : D \times D \to D$, the multiplication gives products if $\otimes$ is right adjoint to the diagonal $\delta : D \to D \times D$. If we apply Lemma 2.1.4 to these two functors we obtain an oblique adjunction between algebra/coalgebras for $\Delta\otimes$ and $\otimes\Delta$. Coalgebras for the latter are given by maps $p : d \to d \otimes d$ which is a step towards comonoids and parts of the oblique adjunction are reminiscent of Fox's construction.

This is only part of a complete picture however. While the products and final objects are described by independent adjunctions, there is a curious symbiosis between multiplications and units in Fox's construction. So, for a start, it would seem that Lemma 2.1.4 would have to be applied to a pair of functors involving both multiplication and unit.

We find the possibility of a connection particularly intriguing as we have used Lemma 2.1.4 so closely with Fox's construction in our account of parameterized fixed point objects.

10.2 The Free Adjunction on the Distributor Classifier

The free adjunction over a director looks to be very interesting. The simplest case is that of the free adjuntion over the director with just the one oblique edge between two kinds of object (which happens to be the distributor classifier). This has objects of the form $(UL)^n\bot$, $(UL)^nU\top$, $(LU)^n\top$ and $(LU)^nL\bot$, with $n \in N$, and lots of maps. It appears to be some sort of vertical naturals glued to itself upside down. Computationally, it would seem to be the free computational calculus on one 'tick'.

10.3 Logical Relations

We have used the graph category framework for binary relational parametricity in a very elementary way. Really we should have built a graph compact adjunction to obtain a theory of logical relations for RLNL. Using a more sophisticated form of quotient relation it may be possible to show that *any* compact adjunction is a parametric model (with respect to the more sophisticated

notion of relation) and therefore interprets fixed point combinators as canonical fixed. In fact, this was the original aim of this thesis. The sticking point seems to be the left adjoint (which is one point of tensor multiplication on D and so there is trouble there too).

10.4 Other Fixed Point Transformations

As we mentioned in Section 6.3.3, in a purely exponential model we have an alternative internalization of the fixed point operator and it is not difficult to derive a fixed point combinator to inhabit the corresponding type expression in LFPC. Because exponential models have opposites, we also obtain a dual fixed point operator with components $((e{-}d) \to d){-}e \to (e{-}d)$, but we do not see the corresponding combinator (in the absence of an action type $\oslash$).

10.5 Other Recursive Types

There is another idiom staring out at us from the fixed point derivations. The recursive types are always used in conjuntion with the type constructor U in one of three idioms.

Idiom	Sugar
$\dfrac{\dfrac{\dfrac{\Psi, \mathrm{A} \blacktriangleright B[\mathrm{UA}]}{\Psi \blacktriangleright \mu\mathrm{A}.B[\mathrm{UA}]}}{\Psi \rhd \mathrm{U}\mu\mathrm{A}.B[\mathrm{UA}]}}$	$\dfrac{\Phi, \mathrm{Y} \blacktriangleright B[\mathrm{Y}]}{\Phi \rhd \mu\mathrm{Y}.B[\mathrm{Y}]}$
$\dfrac{\dfrac{\dfrac{\Psi \blacktriangleright \Gamma \vdash a : B[\mathrm{U}\mu\mathrm{A}.B[\mathrm{UA}]]}{\Psi \blacktriangleright \Gamma \vdash \mathtt{fold}(a) : \mu\mathrm{A}.B[\mathrm{UA}]}}{\Psi \rhd \Gamma \vdash \mathtt{thunk}(\mathtt{fold}(a)) : \mathrm{U}\mu\mathrm{A}.B[\mathrm{UA}]}}$	$\dfrac{\Phi \blacktriangleright \Gamma \vdash a : B[\mu\mathrm{Y}.B[\mathrm{Y}]]}{\Phi \rhd \Gamma \vdash \mathtt{fold}(a) : \mu\mathrm{Y}.B[\mathrm{Y}]}$
$\dfrac{\dfrac{\dfrac{\Psi \rhd \Gamma \vdash x : \mathrm{U}\mu\mathrm{A}.B[\mathrm{UA}]}{\Psi \blacktriangleright \Gamma \vdash \mathtt{force}(x) : \mu\mathrm{A}.B[\mathrm{UA}]}}{\Psi \blacktriangleright \Gamma \vdash \mathtt{unfold}(\mathtt{force}(x)) : B[\mathrm{U}\mu\mathrm{A}.B[\mathrm{UA}]]}}$	$\dfrac{\Phi \rhd \Gamma \vdash x : \mu\mathrm{Y}.B}{\Phi \blacktriangleright \Gamma \vdash \mathtt{unfold}(x) : B[\mu\mathrm{Y}.B]}$

Here is the sugared form of the derivation of Ω.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathrm{Y}, \mathrm{B} \rhd \mathrm{Y}} \quad \dfrac{}{\mathrm{Y}, \mathrm{B} \blacktriangleright \mathrm{B}}}{\mathrm{A}, \mathrm{B} \blacktriangleright (\mathrm{Y} \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \underbrace{\mu\mathrm{A}.(\mathrm{UA} \to \mathrm{B})}_{M}}}{\mathrm{B} \rhd M}}{\mathrm{B} \rhd \mathrm{x} : M \vdash \mathrm{x} : M} \quad \dfrac{\dfrac{\mathrm{B} \rhd M}{\mathrm{B} \rhd \mathrm{x} : M \vdash \mathrm{x} : M}}{\mathrm{B} \blacktriangleright \mathrm{x} : M \vdash \mathtt{unfold}(\mathrm{x}) : (M \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \mathrm{x} : M \vdash (\mathtt{unfold}(\mathrm{x}))\mathrm{x} : \mathrm{B}}}{\mathrm{B} \blacktriangleright \vdash \underbrace{\lambda\mathrm{x}.(\mathtt{unfold}(\mathrm{x}))\mathrm{x}}_{\mathsf{A}} : (M \to \mathrm{B})}}{\mathrm{B} \blacktriangleright \vdash \mathsf{A} : (\mathrm{U}M \to \mathrm{B})}}{\mathrm{B} \rhd \vdash \mathtt{fold}(\mathsf{A}) : M} \quad \mathrm{B} \blacktriangleright \vdash \mathsf{A} : (M \to \mathrm{B})}{\mathrm{B} \blacktriangleright \vdash \underbrace{(\mathsf{A})\mathtt{fold}(\mathsf{A})}_{\Omega} : \mathrm{B}}$$

This derivation inhabits LFPC together with the rules shown in Figure 10.1.

10.6 Parametricity and Compactness

In view of Plotkin's observations on parametricity and compactness the term model of RLNL could probably be made compact if the calculus were extended in some way that would allow parametricity to be expressed, for example by including a type constructor that internalises the collection of quotient relations or allows it to be internalised. Or perhaps it is enough to add structure that ensures the left adjoint and tensor preserve pull-backs as in domain-theoretic models.

10.7 Models of closed LNL.

While closed LNL has an notion of model in symmetric closed adjunctions inherited from the notion of LNL model in symmetric monoidal closed adjunctions, the question of term models and, hence, categorical completeness requires other structure.

Without product types, there are [sigh] at least two ways to build a categorical term model out of a simply typed lambda calculus. We can abandon the notion of types as objects and build a cartesian closed category of contexts and simultaneous substitutions, or we can use types as objects, but index our model with a category of contexts. In the first case we obtain a simple category with complex objects and a contrived function space, while in the second case we obtain an indexed category with simple objects and natural function spaces.

As the category of types acts on the category of contexts by context extension, the notion of structural action might still be useful. For closed relevant calculi the notion of structural action might be generalized from indexed comonad to indexed structure that models the relevant form of context extension.

10.8 2-Categories for Types

In Chapter 3 we have a definition of lax monoid in categories with lax monoid structure. This is an example of what Baez calls the microcosm principle: internal structure requires like external structure. Does this apply to fixed point structure? Is there a 2-category in which the invariants of compact categories are induced by a generic endofunctor in a 2-categorical analogue of a fixed point object? Something like this seems to be happening in Wraith's construction of a natural number object in the category of bounded toposes which is then used to identify internal iterates (which makes essential use of distributors). A starting point might be the 2-category of compact adjunctions or, to connect with Wraith's construction, Vicker's Geometric Domain Theory.

$$\frac{\Phi, \mathrm{Y} \blacktriangleright B}{\Phi \rhd \mu \mathrm{Y}.B}$$

$$\frac{\Phi \blacktriangleright \Gamma \vdash a : B[\mu \mathrm{Y}.B]}{\Phi \rhd \Gamma \vdash \mathtt{fold}(a) : \mu \mathrm{Y}.B}$$

$$\frac{\Phi \rhd \Gamma \vdash x : \mu \mathrm{Y}.B}{\Phi \blacktriangleright \Gamma \vdash \mathtt{unfold}(x) : B[\mu \mathrm{Y}.B]}$$

$$\begin{array}{rcl} \mathtt{unfold}(\mathtt{fold}(a)) & \to_\beta & a \\ x & =_\eta & \mathtt{fold}(\mathtt{unfold}(x)) \end{array}$$

Figure 10.1: Rules and equations for linear recursive nonlinear types.

$$\frac{\Psi, \mathrm{A} \rhd X}{\Psi \blacktriangleright \mu \mathrm{A}.X}$$

$$\frac{\Psi \rhd \Gamma \vdash b : X[\mu \mathrm{A}.X]}{\Psi \blacktriangleright \Gamma \vdash \mathtt{fold}(b) : \mu \mathrm{A}.X}$$

$$\frac{\Psi \blacktriangleright \Gamma \mid \Pi \vdash b : \mu \mathrm{A}.X \quad \Psi \blacktriangleright \Gamma, \mathrm{x} : X[\mu \mathrm{A}.X] \mid \Pi' \vdash a : A}{\Psi \blacktriangleright \Gamma \mid \Pi, \Pi' \vdash b \ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a : A}$$

$$\begin{array}{rcl} \mathtt{fold}(y)\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ a[\mathrm{x}] & =_\beta & a[y] \\ a & =_\eta & a\ \mathtt{to}\ \mathrm{x}\ \mathtt{in}\ \mathtt{fold}(\mathrm{x}) \end{array}$$

Figure 10.2: Rules and equations for nonlinear recursive linear types.

Bibliography

[1] A. Barber and G. Plotkin. Dual intuitionistic linear logic. Technical report ECS-LFCS-96-347, LFCS, University of Edinburgh, 1996.

[2] H. P. Barendrecht. *The Lambda Calculus – Its Syntax and Semantics*. North-Holland, 1984.

[3] P. Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical Report 352, Cambridge University, Cambridge, England, October 1994.

[4] P. Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135, Berlin, 1995. Springer Verlag.

[5] P.N. Benton, G.M. Bierman, V.C.V. de Paiva, and J.M.E. Hyland. Term assignment for intuitionistic linear logic. Technical Report 262, Computer Laboratory, University of Cambridge, 1992.

[6] Richard Blute, J. R. B. Cockett, and R. A. G. Seely. Categories for computation in context and unified logic. *Journal of Pure and Applied Algebra*, 116:49–98, 1997.

[7] Francis Borceux. *Handbook of Categorical Algebra*. Cambridge University Press, 1994.

[8] Roy L. Crole and Andrew M. Pitts. New foundations for fixpoint computations: Fix-hyperdoctrines and the fix-logic. *Information and Computation*, 98(2):171–210, 1992.

[9] Brian Dunphy. *Parametricity as a Notion of Uniformity in Reflexive Graphs*. PhD thesis, Department of Mathematics, University of Illinois, 2002.

[10] Marcelo P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. PhD thesis, University of Edinburgh, 1994.

[11] Marcelo P. Fiore. An enrichment theorem for an axiomatisation of categories of domains and continuous functions. *Mathematical Structures in Computer Science*, 7(5):591–618, 1997.

[12] Marcelo P. Fiore, Gordon D. Plotkin, and A. John Power. Complete cuboidal sets in axiomatic domain theory. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages 268–279. IEEE Computer Society Press, 1997.

[13] T. Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976.

[14] Peter Freyd. Algebraically complete categories. In A. Carboni et al., editors, *Proc. 1990 Como Category Theory Conference*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer Verlag, 1991.

[15] Peter J. Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science: Proceedings of the LMS Symposium, Durham, 1991*, number 177 in LMS Lecture Notes. Cambridge University Press, 1992.

[16] Claudio Hermida and Bart Jacobs. An algebraic view of structural induction. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 412–426. Springer Verlag, 1995.

[17] Jesse Hughes. *A Study of Categories of Algebras and Coalgeras*. PhD thesis, Department of Philosophy, Carnegie Mellon University, May 2001.

[18] Bart Jacobs. *Categorical Type Theory*. PhD thesis, University of Nijmegen, 1991.

[19] G. M. Kelly. Coherence theorems for lax algebras and for distributive laws. *Lecture Notes in Mathematics*, 420:281–375, 1974.

[20] Anders Kock. Monads on symmetric monoidal closed categories. *Arc. Math. (Basel)*, pages 1–10, 1970.

[21] Joachim Lambek. A fixpoint theorem for complete catgories. *Math. Zeitschr.*, 103:151–161, 1968.

[22] Saunders Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer Verlag, New York, New York, 1971.

[23] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. *TLCA 1999*, pages 228–242, 1999.

[24] M. Maietti, P. Maneggia, V. de Paiva, and E. Ritter. Relating categorical semantics for intuitionistic linear logic. Technical Report CSR-01-07-2001, University of Birmingham, 2001.

[25] Guy McCusker. Games and full abstraction for fpc. *LICS 1996*, pages 174–183, 1996.

[26] Jeanne Meisen. Relations in regular categories. *Lecture Notes in Mathematics*, 418, 1974.

[27] Eugenio Moggi. Computational lambda-calculus and monads. *LICS 1989*, pages 14–23, 1989.

[28] P. S. Mulry. Strong monads, algebras and fixed points. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science: Proceedings of the LMS Symposium, Durham, 1991*, volume 177 of *LMS Lecture Notes*. Cambridge University Press, 1992.

[29] David Park. The Y-combinator in scott's lambda-calculus models (revised version). Theory of Computation Report 13, Department of Computer Science, University of Warwick, June 1976.

[30] M. Cristina Pedicchio. Maltsev categories and maltsev operations. *Journal of Pure and Applied Algebra*, 98, 1995.

[31] Gordon D. Plotkin and Alex K. Simpson. Complete axioms for categorical fixed-point operators. *LICS 2000*, pages 30–41, 2000.

[32] A. John Power and Hayo Thielecke. Closed freyd- and kappa-categories. *ICALP 1999*, pages 625–634, 1999.

[33] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997.

[34] John Power and Giuseppe Rosolini. Fixpoint operators for domain equations. *Theoretical Computer Science*, 278:323–333, 2002.

[35] John Power and Hiroshi Watanabe. Combining a monad and a comonad. *Theoretical Computer Science*, 280:137–162, 2002.

[36] Robert Tennent (rdt@qucis.queensu.ca). What is a data refinement relation? Electronic mail to data-refinement@etl.go.jp, 5 January 1996.

[37] E. P. Robinson. The simple fibration. Unpublished manuscript, April 1996.

[38] Alex Simpson and Martín Escardó. A universal characterization of the closed euclidean interval (extended abstract). In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science*, pages 115–125, 2001.

[39] Michael B. Smyth and Gordon D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4):761–783, 1982.

[40] Joseph E. Stoy. *Denotational Semantics : the Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[41] Paul Taylor. Intuitionistic sets and ordinals. *The Journal of Symbolic Logic*, 61(3):705–744, 1996.

[42] Paul Taylor. *Practical Foundations of Mathematics*. Number 59 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1999.

[43] Mitchell Wand. Fixed-point constructions in order-enriched categories. *Theoretical Computer Science*, 8:13–30, 1979.

Appendix A

Distributors and Directors

Distributors—the structures formerly known as profunctors—are commonly presented as a form of binary relation between categories with an emphasis on relational composition, but this is not the view we describe here. Our picture of distributors is based on an elementary notion of oblique arrow, an arrow from one kind of object to another, strictly incomparable kind of object. The two kinds of object live in categories of their own, with plain arrows, but there are no plain arrows between the two kinds of object. Moreover, in a single distributor, all the oblique arrows go from the one kind of object to the other and so oblique arrows are never composed with one another.

We are being coy about the order of the two kinds because there are two ways of orienting these structures. The way of the profunctor agrees with the orientation of the oblique arrows and the way of the distributor disagrees. Although we focus on the oblique arrows, the 2-theory of these structures works better the distributor way (see [7]). To help with the orientation we think of the structure vertically: the oblique arrows *up* from *A* to *B*, which form a profunctor *up* from *A* to *B*, form a distributor *down* from *B* to *A*.

If composition is dropped, categories become directed graphs and distributors become, for lack of a better word, *directors*. Directors have oblique edges from one kind of node up to another kind of node. As with distributors, there are no edges back down from the other kind of node and so a path in a director contains at most one oblique edge. Just as our picture of categories begins with directed graphs, our picture of distributors begins with directors.

A.1 Distributors over Directors.

We picture a director as a collection of special edges connecting nodes in one graph to the nodes in another. By 'graph' we mean '(small) directed graph'. As in a graph, the notion of composable pair of edges is well defined, but no composite is given.

Definition A.1.1 *A* (small) director *from a graph* G *down to a graph* F *is a (small) collection of oblique edges from nodes in* F *up to nodes in* G.

Definition A.1.2 *A* director morphism *from* K *to* K′ *is a pair of graph morphisms from* F *to* F′ *and from* G *to* G′ *together with a function from the oblique edges of* K *to the oblique edges of* K′ *that is compatible with the graph morphisms on nodes.*

Directors and director morphisms form a category Dir.

Definition A.1.3 *The* total graph *of a director* K *from* G *to* F *is* F + G *with the oblique edges of* K *attached as plain edges.*

This extends to a functor Dir → Gra, where Gra is the category of graphs.

Definition A.1.4 *A* (small) distributor *from a (small) category D down to a (small) category C is a (small) collection of oblique arrows from objects in C up to objects in D that is closed under composition with arrows in C and D.*

Definition A.1.5 *A* morphism of distributors *from P to P′ is a pair of functors from C to C′ and from D to D′ together with a function from the oblique arrows of P to the oblique arrows of* P′ *that commutes with composition.*

Distributors and distributor morphisms form a category Dis, which should not be confused with the bicategory **Dist** of distributors where distributors are composed. Another representation of the category Dis is obtained by pulling the underlying graph functor from Cat to Gra back along the total graph functor.

Definition A.1.6 *The* total category *of a distributor P from C to D is C + D with the oblique arrrows of P attached as plain arrows.*

This extends to a functor Dis → Cat that lies over the total graph functor. The square is a pullback.

$$\begin{array}{ccc} \text{Dis} & \xrightarrow{\text{total}} & \text{Cat} \\ \downarrow{\scriptstyle \text{under}} & & \downarrow{\scriptstyle \text{under}} \\ \text{Dir} & \xrightarrow{\text{total}} & \text{Gra} \end{array}$$

A.2 Free Distributors

Just as we picture an arrow in the free category on a graph F as a (possibly empty) path in F (together with start and finish nodes), we picture an oblique arrow in the free distributor on a director K as a (possibly empty) path in F followed by an oblique edge in K followed by a (possibly empty) path in G. The paths in *G* and the paths in *F* are the arrows in the domain and codomain of the path distributor. In both the path category and the path distributor, the empty paths (together with their start and finish node) are the identity arrows. There are no oblique identity arrows because a path that contains an oblique edge is not empty.

Definition A.2.1 *The* path distributor *of a director* K *from* G *to* F *is the director from* pathG *to* pathF *with each oblique edge given by a path in* F *followed by an oblique edge in* K *followed by a path in* G.

The path construction commutes with the total graph/category construction. Rather than construct the oblique arrows seperately from the arrows, we can take the paths in the total graph and pick out those containing an (ex)oblique edge as the oblique arrows.

$$\begin{array}{ccc} \text{Dis} & \xrightarrow{\text{total}} & \text{Cat} \\ \uparrow{\scriptstyle \text{path}} & & \uparrow{\scriptstyle \text{path}} \\ \text{Dir} & \xrightarrow{\text{total}} & \text{Gra} \end{array}$$

The path construction for directors is left adjoint to the underlying director functor from distributors to directors and so produces the free distributor with respect to underlying director morphisms. The adjunction gives us a path monad on Dir analogous to the path monad on Gra.

A.3 The Distributor/Director Classifier

So far, our picture of distributors is compositive: we begin with two, distinct categories and add certain oblique arrows to obtain a composite structure. Alternatively, we might begin with one category and identify two kinds of object. If we then prohibit arrows down from the one kind to the other, we can identify the arrows up from the other kind as oblique. This second point of view is neatly expressed using the notion of a classifier.

The category of directors is equivalent to the slice category $\mathrm{Gra}/\uparrow$, where $\uparrow$ is the graph that has two nodes with reflexive edges and one irreflexive edge between them. The graph $\uparrow$ classifies directors: the oblique edges are singled out and sent to the one irreflexive edge while the domain and codomain graphs are sent to the two reflexive subgraphs. Note that edges back down from the domain are prohibited by the lack of a second irreflexive edge.

Likewise, the category of distributors is equivalent to the slice category $\mathrm{Cat}/\uparrow$, where $\uparrow$ is now the category with two objects and one non-identity arrow between them. Interestingly, $\mathrm{Cat}/\uparrow$ is the same as the comma category $(\mathrm{under},\uparrow)$, where under is the underlying graph functor and $\uparrow$ is the graph classifying directors. In a sense, the graph $\uparrow$ actually classifies both directors and distributors.

Note that a Grothendieck construction for categories gives us a functor $\mathrm{Cat}^{\uparrow} \to \mathrm{Cat}/\uparrow$, where $\mathrm{Cat}^{\uparrow}$ is the category of functors (pointing up) and functor squares.

A.4 Locally Small Distributors

Whether we view oblique arrows as connecting two categories or as singled out by a classifying functor, oblique arrows can only be composed with plain arrows to produce other oblique arrows. This suggests a third point of view that allows for a more subtle treatment of size than the one-size-fits-all approach adopted (implicitly) above.

Just as the notion of category can be expressed in terms of hom' sets, the notion of a distributor can be expressed in terms of het' sets, where by 'heteromorphism' we mean 'oblique arrow'.

Definition A.4.1 *A* locally small distributor *from a locally small category D to a locally small category C has a set* $\mathrm{het}(d,c)$ *of oblique arrows for each object d in D and each object c in C together with bimodular actions*

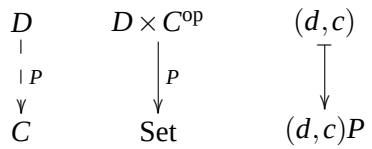
$$\mathrm{comp} : \mathrm{hom}(c',c) \times (d,c)\mathrm{het} \times \mathrm{hom}(d,d') \longrightarrow (d',c')\mathrm{het}\,.$$

Note that for het sets the covariant argument is written first and the contravariant second. Ideally we would write one above the other, but this looks odd in print. To remind ourselves of the reversal, we write the 'het' after its arguments. To distinguish between different categories and distributors, we often write the symbol for the category or distributor in place of the 'hom' or 'het'.

Given a distributor P, the bimodular actions can be rearranged to give the local components of a functor from the locally small category $D \times C^{\mathrm{op}}$ to the locally small category Set.

$$\begin{aligned}
C(X,Y) \times (A,Y)P \times D(A,B) &\longrightarrow (B,X)P \\
C(X,Y) \times D(A,B) &\longrightarrow \mathrm{Set}((A,Y)P,(B,X)P) \\
C^{\mathrm{op}}(Y,X) \times D(A,B) &\longrightarrow \mathrm{Set}((A,Y)P,(B,X)P) \\
D(A,B) \times C^{\mathrm{op}}(Y,X) &\longrightarrow \mathrm{Set}((A,Y)P,(B,X)P) \\
(D \times C^{\mathrm{op}})((A,Y),(B,X)) &\longrightarrow \mathrm{Set}((A,Y)P,(B,X)P)
\end{aligned}$$

The bimodularity of the actions is equivalent to the functorality of these components. Hence the snappy definition often found in the literature: a distributor is a functor from $D \times C^{\mathrm{op}}$ to Set.



Given a locally small category C, by reversing the arguments to the hom sets for C we obtain the het sets $(b,a)C = C(a,b)$ of a distributor from C to itself. Note that the hom distributor uses three copies of each arrow in C, two for the two copies of C that provide the domain and codomain and a third copy for the collection of oblique arrows.

A.5 Fore and Aft

Functors induce distributors—which explains the term 'profunctor'—and graph morphisms induce directors. Given a graph morphism G from C up to D, we can take oblique edges from c to d to be given by plain arrows from Gc to d. Note that we require two copies of each edge from Gc to d. The original stays in D where it may well follow other edges of the same form. A second copy is formally detached (more precisely, one end is detached) from D to become an oblique edge and follow edges to c in C. We call the resulting director the *fore director* and write G_*. Given a functor, the same construction produces a distributor.

Definition A.5.1 *Given a functor G from C to a locally small category D, the* fore distributor G_* *is given by $G_*(d,c) = D(Gc,d)$.*

This extends to a functor $\mathrm{Cat}^{\uparrow} \to \mathrm{Dis}$ which is equivalent to the functor $\mathrm{Cat}^{\uparrow} \to \mathrm{Cat}/\uparrow$ given by a Grothendieck construction.

Given a graph morphism H from D down to C we can take the edges from c to Hd as oblique edges from c to d to obtain a director H^* which we call the aft director. This aft construction can be expressed in terms of the fore construction using various dualities, but we prefer to continue with elementary definitions until it is clear that one construction is more fundamental.

Definition A.5.2 *Given a functor H from D to a locally small category C, the* aft distributor H^* *is given by $H^*(d,c) = C(c,Hd)$.*

This extends to a functor $\mathrm{Cat}^{\downarrow} \to \mathrm{Dis}$ which is equivalent to the functor $\mathrm{Cat}^{\downarrow} \to \mathrm{Cat}/\uparrow$ given by another Grothendieck construction.

The categories For and Aft are the categories of distributors-with-a-fore-representation and distributors-with-an-aft-representation. The free distributor-with-a-fore-representation over a director K from G down to F is the distributor induced by the inclusion functor from path$F$ up into path$K$. The free distributor-with-an-aft-representation over K is the distributor induced by the inclusion functor from path$G$ down into path$K$. The requirement that the distributor be induced by a functor generates extra objects.

A.6 Distributors and Adjunctions

If we replace categories with distributors we obtain a generalization of the notion of adjunction. Instead of a natural bijection between elements of hom sets, we ask for a natural bijection between elements of het sets.

Definition A.6.1 *An* oblique adjunction *is given by a natural isomorphism* $R(d, Gc) \cong Q(Hd, c)$*, where* R *and* Q *are distributors and* G *and* H *are functors.*

$$\begin{array}{ccc} D_1 & \overset{}{\underset{R}{\leftarrow\!-\,-}} & D_0 \\ {\scriptstyle G}\uparrow & \dashv & \downarrow{\scriptstyle H} \\ C_1 & \underset{Q}{\leftarrow\!-\,-} & C_0 \end{array}$$

Ordinary adjunctions correspond to the special case of an oblique adjunction between hom distributors.

$$\begin{array}{ccc} D & \underset{\mathrm{hom}}{\leftarrow\!-\,-} & D \\ {\scriptstyle G}\uparrow & \dashv & \downarrow{\scriptstyle H} \\ C & \underset{\mathrm{hom}}{\leftarrow\!-\,-} & C \end{array}$$

In this case the isomorphism can be viewed as an isomorphism between the fore distributor induced by G and the aft distributor induced by H.

Proposition A.6.2 *If* G *is left adjoint to* H*, then* G_* *is isomorphic to* H^* *in* Dis.

The converse fails because, in general, an isomorphism between G_* and H^* in Dis may include automorphisms of D and C.

Proposition A.6.3 G *left adjoint to* H *if and only if* G_* *is isomorphic to* H^* *in* Dis *over identities on* D *and* C.

In the bicategory **Dist**, where distributors are composed and hom distributors are identities, the oblique adjunction isomorphism can be written $G_* \circ D \cong C \circ H^*$.

$$\begin{array}{ccc} D_1 & \underset{D}{\leftarrow\!-\,-} & D_0 \\ {\scriptstyle G_*}\downarrow & \cong & \downarrow{\scriptstyle H^*} \\ C_1 & \underset{C}{\leftarrow\!-\,-} & C_0 \end{array}$$

With identities for C and D, we obtain a characterization of ordinary adjunctions.

Proposition A.6.4 G *is left adjoint to* H *in* **Cat** *if and only if* G_* *is isomorphic to* H^* *in* **Dist**.

$$\begin{array}{ccc} D & \longrightarrow\!\!\!\!\!\!\!\!- & D \\ {\scriptstyle G_*}\downarrow & \cong & \downarrow{\scriptstyle H^*} \\ C & \longrightarrow\!\!\!\!\!\!\!\!- & C \end{array}$$

This is a direct consequence of the definitions of G_* and H^*, but also follows from bicategorical properties of the inclusion of bicategories **Cat** → **Dist** given by the aft construction (Propositions 7.8.5 and 7.9.1 in Borceux [7]). Proposition A.6.4 should not be confused with the following proposition.

Proposition A.6.5 *Given any functor* F*, the distributor* F_* *is left adjoint to* F^* *in* **Dist**.

The category Adj of adjunctions and adjunction morphisms is a category of distributors-with-both-fore-and-aft-representations. It is equivalent to the pull-back of the projection For → Dis along the projection Aft → Dis.

$$\begin{array}{ccc} \mathrm{Adj} & \longrightarrow & \mathrm{For} \\ \downarrow & & \downarrow \\ \mathrm{Aft} & \longrightarrow & \mathrm{Dis} \end{array}$$

The free adjunction over a director exists but is more complex than the free distributors described above.

Appendix B

Structural Actions

Here is an elementary account of structural actions and the indexed category construction. While we developed this material independently, another, differently motivated account has appeared in the work of Blute, Cockett and Seely [6] and we use their terminology.

In a nutshell, a structural action is an indexed comonad, and the Kleisli construction produces an indexed category. Chapter 3 develops an even more abstract perspective which views a structural action as the transpose of a functor that preserves comonoid objects. Although such abstract points of view help to get the definitions and coherence conditions right, Chapter 4, which actually uses the definitions and conditions, can be understood on the basis of the following elementary account.

B.1 Categories with Structural Actions

An *action* of a category C on a category D is a functor from $C \times D$ to D. Functors on product categories are sometimes said to be bifunctorial, so we are taking actions to be bifunctorial. We are interested in actions with certain extra structure.

Definition B.1.1 *A* structural action *is an action* $\oslash : C \times D \to D$*, together with natural transformations,* duplication *and* elimination*, with components* $\delta : c \oslash d \to c \oslash (c \oslash d)$ *and* $\iota : c \oslash d \to d$ *such that* $\iota \circ \delta = \mathrm{id}$, $(\mathrm{id} \oslash \iota) \circ \delta = \mathrm{id}$ *and* $\delta \circ \delta = (\mathrm{id} \oslash \delta) \circ \delta$.

We may think of a structural action as an indexed comonad: the action of each object in C has the structure of a comonad on D and, importantly, the transformation given by each map in C is a comonad transformation[1].

Proposition B.1.2 *Structural actions of C on D correspond to functors from C to the category of comonads on D.*

With this in mind, the category of comonads on D and comonad transformations has a structural action on D, called the *standard structural action*, corresponding to the identity on the category of comonads.

Given any functor $R : C \to C'$ and a structural action $\oslash : C' \times D \to D$, the composite $\oslash \circ (R \times \mathrm{Id}) : C \times D \to D$ is a structural action. This is the reindexing along R of the indexed comonad given by $\oslash$. Taking 1 for C and the category of comonads on D for C', the unit category 1 has a structural action on D for each comonad on D, which includes the identity comonad. Taking 1

[1] The definition of structural action in [6] overlooks this second condition, although it follows from the naturality of the duplication and elimination transformations.

for C' with the identity action on D, reindexing along the unique functor from C to 1 gives the constant identity structural action on D.

The theory of algebraic compactness we present in Chapter 4 requires products of categories with structural actions. Given structural actions $\oslash_A$ and $\oslash_B$ on categories A and B, the action $\oslash_{A\times B}$ at c takes (a,b) to $(c\oslash_A a, c\oslash_B b)$. This gives a structural action on the product of A and B. It can be checked that this is a product in the category of structural actions of C. The unit is the constant identity action of C on the unit category 1.

We also require a notion of costructural action which we define using the opposite of the category acted upon.

Definition B.1.3 *A* costructural action *of C on D is a structural action of C on D^{op}.*

This differs inessentially from [6] which uses the opposite of the action of the opposite of the acting category: our costructural action of C is the opposite of their costructural action of C^{op}.

B.2 The Indexed Category Construction

Suppose we have a structural action $\oslash : C\times D\to D$. We may use the action to interpret the notion of a parameterized map with parameterizing objects taken from C. We consider a map from $c\oslash d$ to d' as a c-parameterized map from d to d'. Each map from $c\oslash d$ to d' is meant to represent a collection of maps from d to d' indexed by the contents of c.

The duplication transformation allows us compose such parameterized maps. The composite of a c-parameterized maps given by $g : c\oslash d\to d'$ and $g' : c\oslash d'\to d''$ is given by $g'\circ((\mathrm{id}\oslash g)\circ\delta) : c\oslash d\to d''$. This is Kleisli composition for the comonad given by the action of c.

We use the elimination transformation to interpret the notion of a constant parameterized map, one that ignores its parameter. Given a map $g : d\to d'$, we think of the c-parameterized map given by $g\circ\iota : c\oslash d\to d'$ as constantly g for all elements of c. The constant identities, which are given by the components of elimination, behave as identities under composition of parameterized maps.

Given a map $r : c\to c'$, c'-parameterized maps, represented by maps from $c'\oslash d$, can be reparameterized to c-parameterized maps, represented by maps from $c\oslash d$, by composition with $r\oslash d : c\oslash d\to c'\oslash d$ in D. Reparameterization commutes with composition and identities because the natural transformation from the action of c to the action of c' given by f is a comonad transformation.

To sum up, for each object c of C there is a category of c-parameterized maps. This category is the Kleisli category for the comonad given by the action of c. For each map $r : c\to c'$ of in C there is a functor from the category of c'-parameterized maps to the category of c-parameterized maps. In other words, we have an indexed category.

Proposition B.2.1 *Given a structural action $\oslash : C\times D\to D$, the Kleisli category for the action of an object c gives the category over c in an indexed category over C.*

B.3 Structural Functors and Natural Transformations

A functor between categories with structural actions does not have to commute with the actions in order to lift to an indexed functor between the indexed categories constructed from the actions. All that is required is a well placed natural transformation.

Definition B.3.1 *A* structural functor *between categories with structural actions $\oslash$ and $\oslash'$ is a functor F equipped with a natural transformation with components $\theta : c\oslash' Fd\to F(c\oslash d)$ such that $F\iota\circ\theta=\iota$ and $F\delta\circ\theta=\theta\circ(\mathrm{id}\oslash\theta)\circ\delta$.*

If the functor part of a structural transfomation is the identity on D, then it corresponds to an indexed comonad transformation, a natural transformation between two functors from C to the category of comonads on D. Just as comonad transformations lift to functors between Kleisli categories, indexed comonad trasformations lift to indexed functors between indexed Kleisli categories. If the acting category C is the unit category 1, so that structural actions correspond to comonads, then a structural functor is a lax comonad functor, a functor between comonad carriers that lifts to the Kleisli categories for those comonads by means of a coherent natural transformation. In general then, structural functors correspond to a form of indexed comonad transformation/functor, although this is not very helpful conceptually.

A natural transformation between structural functors lifts to an indexed natural transformation if it coheres with the transformations that make the functors structural.

Definition B.3.2 *A* structural natural transformation *between structural functors F and F' is any natural transformation with components $\alpha : Fd \to F'd$ such that $\alpha \circ \theta = \theta' \circ (\mathrm{id} \oslash \alpha)$.*

Suppose we are given a structural action $\oslash : C \times D \to D$. Objects in D correspond to functors from 1 to D and there is a unique structural action of C on the unit category 1, the constant identity action, so we may ask if these pointing functors corresponding to objects in D are structural. We are asking for a transformation with components $\theta : c \oslash d \to d$, where d is a fixed object of D. We observe that the elimination transformation $\iota : c \oslash d \to d$, which is natural, satisfies the conditions for a structural functor. In addition, maps in D correspond to structural natural transformations between these structural pointing functors. Note that we are exploiting the degeneracy of the category 1.

Again using opposites, we define costructural functors and natural transformations.

Definition B.3.3 *A* costructural functor *F is given by a structural functor F^{op} and a* costructural natural transformation *is the opposite of a structural natural transformation.*

Appendix C

Directed Graph Categories

If the notion of directed graph is interpreted in the category of categories or, equivalently, the notion of category is interpreted in the category of directed graphs we obtain an abstract framework for binary relational parametricity. Our definitions and terminology follow [9], but the ideas go back to Reynolds and company.

C.1 Graph Categories, Functors and Transformations

A *graph category* consists of a category of edges R_e together with a *source* functor $(\cdot)_0$ and a *target* functor $(\cdot)_1$ to a category of vertices R_v.

$$\begin{array}{c} R_e \\ {\scriptstyle (\cdot)_0}\Big\Vert\Big\Vert{\scriptstyle (\cdot)_1} \\ R_v \end{array}$$

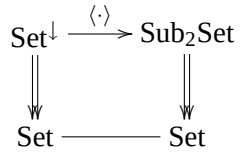
For example, the arrow category $C^{\downarrow}$ can be viewed as the edge category of a graph category with source and target given by domain and codomain, respectively. Another example is $\mathrm{Sub}_2\mathrm{Set}$, the pull-back of the subset fibration SubSet along binary products in Set. This category has binary relations for objects and parametric pairs of functions for arrows (meaning pairs that preserve relatedness).

$$\begin{array}{ccc} \mathrm{Sub}_2\mathrm{Set} & & \mathrm{Sub}_2\mathrm{Set} \longrightarrow \mathrm{SubSet} \\ \Big\Vert & \longleftarrow & \Big\downarrow \quad \mathrm{pb} \quad \Big\downarrow \\ \mathrm{Set} & & \mathrm{Set}\times\mathrm{Set} \xrightarrow{\times} \mathrm{Set} \end{array}$$

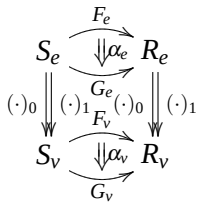
A *graph functor* consists of an edge functor F_e and a vertex functor F_v such that $(F_e s)_0 = F_v(s_0)$ and $(F_e s)_1 = F_v(s_1)$.

$$\begin{array}{ccc} S_e & \xrightarrow{F_e} & R_e \\ {\scriptstyle (\cdot)_0}\Big\Vert\Big\Vert{\scriptstyle (\cdot)_1} & & {\scriptstyle (\cdot)_0}\Big\Vert\Big\Vert{\scriptstyle (\cdot)_1} \\ S_v & \xrightarrow{F_v} & R_v \end{array}$$

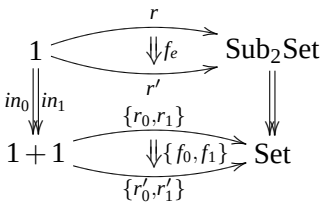
For example, there is the *graph* graph functor $\langle\cdot\rangle$ from $\mathrm{Set}^{\downarrow}$ to $\mathrm{Sub}_2\mathrm{Set}$. The vertex functor is the identity on Set and the edge functor takes a function f to its graph $\langle f\rangle$, the set of pairs (x, fx).



A *graph transformation* consists of an edge natural transformation α_e and a vertex natural transformation α_v such that $(\alpha_{es})_0 = \alpha_{v(s_0)}$ and $(\alpha_{es})_1 = \alpha_{v(s_1)}$.

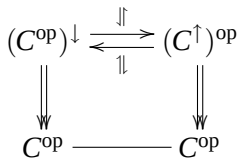


For example, given binary relations r and r' over Set, each pair of functions that is parametric with respect to r and r' corresponds to a graph transformation f thus:

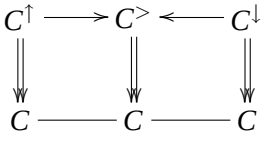


A graph category R can be viewed either as an internal directed graph in the large category of categories, as presented above, or as an internal category in the large category of directed graphs, in which case it has an arrow graph R^1 and an object graph R^0 together with graph morphisms for domain, codomain, composition and identities. Similarly, graph functors and graph transformations can be viewed as internal, directed graph morphisms and directed graph transformations (replace 1-cells with 2-cells in the definition of directed graph morphism) or as internal functors and internal natural transformations. Either way, graph categories, graph functors and graph transformations form a large 2-category GCat.

Note that we must be careful to distinguish $C^{\downarrow}$ from the graph category $C^{\uparrow}$ with source and target given by codomain and domain, respectively. The graph category $C^{\downarrow}$ of *down arrows* is not generally isomorphic to the graph category $C^{\uparrow}$ of *up arrows*. The two are equivalent as graph categories over the identity on C *iff* C is a groupoid. Note that there *is* a graph isomorphism between the graph of down arrows on the opposite of C and opposite of the graph of up arrows on C.



The graphs $C^{\downarrow}$ and $C^{\uparrow}$ both embed fully and faithfully into $C^{>}$, the graph of cospans over C. This has the domain of the first cospan component for source and the domain of the second for target.



C.2 Graph Operators and Parametric Transformations

The notion of graph category is useful because it comes apart in ways that plain categories and plain graphs do not.

Definition C.2.1 *A* graph operator *from S to R consists of an edge function F_e from the objects of S_e to the objects of R_e and a vertex function F_v from the objects of S_v to the objects of R_v such that $(F_e s)_0 = F_v(s_0)$ and $(F_e s)_1 = F_v(s_1)$.*

In other words, a graph operator is a graph morphism from the object graph of S to the object graph of R. For example, the identity on R^0, the object graph of R, gives a graph operator from R^{op} to R and the diagonal graph morphism $\Delta : R^0 \to R^0 \times R^0$ gives a graph operator from R to $R^{\mathrm{op}} \times R$. Note that, while every graph functor restricts to a graph operator, operators such as those above do not generally extend to graph functors.

Given graph operators F and G on S, a *parametric transformation* from F to G is a family of maps $\alpha_d : Fd \to Gd$ indexed by the objects of S_v that lifts to a family of maps $\alpha_s : Fs \to Gs$ indexed by the objects of S_e, meaning $(\alpha_s)_0 = \alpha_{s_0}$ and $(\alpha_s)_1 = \alpha_{s_1}$ for all s.

$$\begin{array}{ccccc}
Fs_0 & \text{——} & \alpha_{s_0} & \longrightarrow & Gs_0 \\
\uparrow & & \uparrow & & \uparrow \\
Fs & \text{——} & \alpha_s & \longrightarrow & Gs \\
\downarrow & & \downarrow & & \downarrow \\
Fs_1 & \text{——} & \alpha_{s_1} & \longrightarrow & Gs_1
\end{array}$$

For graph categories with at most one such α_s we just draw a *parametricity square*:

$$\begin{array}{ccc}
Fs_0 & \xrightarrow{\alpha_{s_0}} & Gs_0 \\
\uparrow Fs & & \uparrow Gs \\
Fs_1 & \xrightarrow{\alpha_{s_1}} & Gs_1
\end{array}$$

Given a construction that lifts categories and object functions to graph categories and graph operators, we can ask when a (not necessarily natural) transformation lifts to a parametric transformation. For example, the arrow graph construction leads to the notion of natural transformation.

Proposition C.2.2 *Natural transformations from F to G are identical with parametric transformations from $F^{\downarrow}$ to $G^{\downarrow}$.*

$$\begin{array}{ccc}
Fs_0 & \xrightarrow{\alpha_{s_0}} & Gs_0 \\
\downarrow Fs & & \downarrow Gs \\
Fs_1 & \xrightarrow{\alpha_{s_1}} & Gs_1
\end{array}$$

Within this framework of binary relational parametricity, the fundamental notions are functorality: when does an object function lift to a functor? And parametricity: When does a transformation lift to a parametric transformation? As the Proposition shows, naturality is a derived notion. Another derived notion is that of diparametricity.

Definition C.2.3 *A* diparametric transformation *between operators F and G on $S^{\mathrm{op}} \times S$ is a parametric transformation between the operators $F \circ \Delta$ and $G \circ \Delta$ on S.*

$$S \xrightarrow{\Delta} S^{\mathrm{op}} \times S \underset{F}{\overset{G}{\rightrightarrows}} R$$

Just as dinaturality weakens naturality, diparametricity weakens parametricity. Note, however, that diparametricity is expressed in terms of parametricity, unlike dinaturality which cannot, in general, be expressed in terms of naturality. Also, because diparametricity squares compose, diparametrics compose.

$$\begin{array}{ccccc} F(s_0,s_0) & \xrightarrow{\alpha_{s_0}} & G(s_0,s_0) & \xrightarrow{\alpha'_{s_0}} & H(s_0,s_0) \\ \updownarrow{\scriptstyle F(s,s)} & & \updownarrow{\scriptstyle G(s,s)} & & \updownarrow{\scriptstyle H(s,s)} \\ F(s_1,s_1) & \xrightarrow{\alpha_{s_1}} & G(s_1,s_1) & \xrightarrow{\alpha'_{s_1}} & H(s_1,s_1) \end{array}$$

Remark. Stretching our framework, dinaturality can be expressed in terms of diparametricity. Given a category C, the quasi-category $\mathsf{D}C$ of *diamonds* has commutative diamonds for objects and commuting pairs of maps (as in $\mathsf{K}C$) for arrows. By 'quasi' we mean that composition is a partial operation on composable pairs and the equations of category theory hold just where both sides exist. A 'graph quasi-category' has a category of vertices and quasi-category of edges. The definitions of 'graph operator' and 'parametric transformation' are unaffected because they ignore composition. Without extra conditions (see Section 9.1.5), there is no guarantee that componentwise composition of arrows in $\mathsf{D}C$ gives arrows in $\mathsf{D}C$. On the other hand, every functor F lifts componentwise to a quasi-functor $\mathsf{D}F$ which preserves what composites exist. Also, we have an isomorphism between the graph quasi-categories $(\mathsf{D}C)^{\mathrm{op}}$ and $\mathsf{D}(C^{\mathrm{op}})$. Dinaturality is parametricity with respect to certain objects in the quasi-category of diamonds.

$$\begin{array}{ccccc} & & F(d,d) & \xrightarrow{\alpha_d} & G(d,d) & & \\ & \nearrow^{F(g,d)} & & & & \searrow^{G(d,g)} & \\ F(d',d) & & & & & & G(d,d') \\ & \searrow_{F(d',g)} & & & & \nearrow_{G(g,d')} & \\ & & F(d',d') & \xrightarrow{\alpha_{d'}} & G(d',d') & & \end{array}$$

Proposition C.2.4 *Dinatural transformations from F to G are identical with diparametric transformations from $\mathsf{D}F \circ ((\iota^{\uparrow} \circ \Downarrow) \times \iota^{\downarrow})$ to $\mathsf{D}G \circ ((\iota^{\uparrow} \circ \Downarrow) \times \iota^{\downarrow})$.*

$$(D^{\downarrow})^{\mathrm{op}} \times D^{\downarrow} \xrightarrow{\Downarrow \times D^{\downarrow}} (D^{\mathrm{op}})^{\uparrow} \times D^{\downarrow} \xrightarrow{\iota^{\uparrow} \times \iota^{\downarrow}} \mathsf{D}(D^{\mathrm{op}}) \times \mathsf{D}D \xrightarrow[\text{iso}]{} \mathsf{D}(D^{\mathrm{op}} \times D) \underset{\mathsf{D}F}{\overset{\mathsf{D}G}{\rightrightarrows}} \mathsf{D}C$$