Radboud Universiteit

Realizability semantics for type theory

Interpreting the Calculus of Constructions, and some inductive types, in the category of $\mathcal{D}$-sets.

Kelley Philip van Evert

hello@kelleyvanevert.nl

July 3, 2018

Master's thesis
Mathematical Foundations of Computing Science

Radboud University Nijmegen
Institute for Computing and Information Sciences

Supervisor

Prof. dr. J.H. (Herman) Geuvers
Radboud University Nijmegen
h.geuvers@cs.ru.nl

Second reader

N.M. (Niels) van der Weide
Radboud University Nijmegen
n.vanderweide@cs.ru.nl

Abstract

First, a concrete model is presented for the Calculus of Constructions, based on a realizability interpretation of logic (and hence type theory, over the Curry-Howard correspondence); essentially reducing types to specifications over computational objects of an arbitrary underlying partial combinatory algebra. Crucially, by means of an insight due to the insight of Moggi and Hyland, the category of $\mathcal{D}$-sets (of sets equipped with a realizability predicate) admits the (sub)category of partial equivalence relations as a 'universe object' of small types, allowing it to model polymorphism as well as dependent types.

Second, we show explicitly how the category of partial equivalence relations contains all initial F-algebras for non-dependent, as well as dependent polynomial functors F. This allows us (in principle) to interpret a decent class of inductive types, and moreover derive dependent eliminators for them by means of a little trick. We motivate this, avoiding the necessarily intricate mapping of inductive type specifications to polynomial functors, by working out some typical examples in detail. Finally, we mention the potential of (F, G)-dialgebras in simplifying this mapping.

Acknowledgements

I would have not been able to bring about this thesis, were it not for the following people, to which I am grateful. Cyril, for our friendship and adventures these last few years; Susan, for her confidence in me when mine was at its lowest; Frits, for the push he gave me when I really needed it; Herman, for his help, time and kinds words; and Julia, for her love and support.

Contents

1 Introduction

Realizability semantics

One of the basic premises of the field of theoretical computer science, is the notion of *effective computability* as some kind of natural law.[1] As a result, we have a number of 'real' systems on our hand which model this seemingly absolute notion of computation: the untyped lambda calculus, Turing machines, etc.—indeed having given rise to computers, and hence this field, in the first place. This thesis takes what one could describe as an 'analytical' approach to *type theory*, by using these 'real systems' to give a *semantics* to the Calculus of Constructions. I call this 'analytic' because, as opposed to studying types in an axiomatic fashion, this approach deals with *reducing* the structures of type theory to these 'real' structures of computability.

The method of reduction that we employ, is that of *realizability*, first introduced as an interpretation of first-order logic by Kleene [1945], and which can be understood as a formalized version of the *Brouwer-Heyting-Kolmogorov* interpretation of what it means to prove a statement. A more modern take in a similar vein is expressed in the well-known *Curry-Howard correspondence* that relates types and propositions, resp. programs and proofs. The idea is to assume some model of computation (a *partial combinatory algebra*), and then interpret logic, or type theory, in terms of it. In doing this, an explicit computational interpretation will be given to the more complex types of the type theory in question.

What kind of complex types will we be treating? From the conceptual perspective of Barendregt [1984], the Calculus of Constructions, as a syntactic model of types, contains three defining expressive features: *polymorphism* (allowing functions to be defined 'generically' over parameters of arbitrary type), *dependent types* (allowing type specifications to depend on terms), and *type operators* (allowing the formation of types of functions that operate on types themselves). In principle, each of these provides us with a model-theoretic challenge: how to understand the 'implementation' of (objects of) such types? Or, in our setting, how to understand such types as behavioral specifications of given computation objects?

It turns out that, although from a logical perspective polymorphic types may have seemed a bit weird, there is a relatively easy realizability characterization of polymorphic functions. This was first shown in Girard [1972], but the insight really gained traction after Hyland [1988] and Longo and Moggi [1991]. Informally, the behavioral specification (i.e. 'realizability type') of a polymorphic function is simply that it may not behaviorally depend on the parameter in question, which is stated as that is behaves 'the same on all types'. Formally, the construction relies on the existence of a *universe object* in the category of realizability types in which we will interpret CC, i.e. a 'type of types' that is large enough to allow for this impredicative definition, and it is this existence which was the main finding. Having this universe object subsequently allows for a rather straightforward interpretation of dependent types and type operators.

[1] Yet, as far as can be philosophically ascertained, this 'absoluteness' remains an empirical statement, and is voiced in the famous 'Church-Turing' thesis.

Modeling dependent inductive types

Inductive types, which also go under the name of *algebraic data types* in programming communities, are a well-understood class of structured (data) types that arise as freely generated algebras over a given list of constructors.[2] It is known that inductive types can be characterized as initial F*-algebras* for certain endofunctors F. If one takes F from the class of *non-dependent polynomial functors*, the so-called *well-founded trees* (or, W-types) are obtained. These types, originally studied by Per Martin-Löf, provide for a decent class of non-dependent inductive types. It is also known that any *Martin-Löf category* (locally cartesian closed, and supporting W-types) also supports all initial F-algebras for *dependent polynomials* F, allowing for the interpretation of a wide range of dependent inductive types (see for instance Hamana and Fiore [2011] and Abbott et al. [2005]).

We show (with an explicit construction similar to that of Kleene's fixed-point theorem) that the category of partial equivalence relations, which in our model forms the universe of 'small types', indeed contains all initial algebras for such polynomials. With a few detailed examples, we motivate that a decent class of inductive types can therefore be interpreted in the model. In order the obtain dependent eliminators for these types, we apply a simple 'trick', defining first the (inductive) 'type of dependent propositions' over them, whose non-dependent recursor then corresponds to the sought dependent eliminator.

A problem that occurs, when translating inductive type specifications to their respective polynomials, is that these polynomials get unwieldy quite fast. One approach to remedy this situation, is to describe inductive types in terms of (F, G)-dialgebras, which generalize F-algebras. We briefly discuss the reduction of a certain class of (F, G)-dialgebras to F-algebras, due to Basold [2015].

Outline

In chapter 2, we give a brief historical account to motivate the methodology of realizability semantics, and also provide some technical background material on (a) partial combinatory algebras, (b) the Calculus of Constructions, and (c) some category theory, mainly with regard to dependent types, polynomial functors, and F-algebras and (F, G)-dialgebras.

In chapter 3, we provide an introductory realizability model for the simply typed lambda calculus, which in particular also allows us to emphasize some key characteristics of the realizability interpretation.

Chapter 4 contains the 'meat' of the semantics, defining the categories of $\mathcal{D}$-sets and partial equivalence relations, and describing the interpretation-supporting structures that they admit. In particular, a 'universe object' is identified, and it is shown how this facilitates the interpretation of polymorphism.

In chapter 5, the technical machinery amounted so far is put to work in a relatively straightforward model of the Calculus of Constructions, and which we again highlight some characteristics of the realizability semantics.

In chapter 6, we show that the category of partial equivalence relations contains all initial algebras for (non-)dependent polynomials (and in particular contains W-types), and subsequently can interpret a decent class of dependent inductive types. Although we don't give an explicit interpretation in full generality, we cover a few typical cases in some detail, deriving their recursors as well as dependent eliminators. Furthermore, we mention how (F, G)-dialgebras may be used to describe dependent inductive types.

[2] Unless one works in a non-extensional setting (as is particularly interesting as of late given the emergence of Homotopy Type Theory), 'freely generated' just means inductively constructed, from which the concept derives its name. Our model will indeed be extensional.

2 Setting & background material

In this chapter, we will first briefly discuss the historical origin and setting of the realizability semantics approach, and then some background material necessary for subsequent theory development.

2.1 Origins and method of realizability

This thesis is concerned with giving a particular kind of *denotational semantics* for type theories, based on the notion of *realizability* (sometimes also called *recursive realizability*). This fascinating approach has its origin in the field of logic, in the various reconsiderations of logical truth of the first half of the 20^{th} century. This section will paint a quick picture of these origins, although it must be noted that for the matter of this thesis' theory development, most of the logical nuances will be of no import.

In the context of Brouwer's *intuitionism*, Heyting's *intuitionistic number theory* (or *Heyting arithmetic*) and the Russian school of recursion theory, Kleene [1945] initially introduced the notion of recursive realizability, which can be seen as a formalization of the informal *BHK interpretation* (although the historical record is more complicated and a bit murky). The BHK interpretation, named after Brouwer, Heyting, and Kolmogorov, proposed the view that logical statements should be viewed in terms of what is required to prove them:

- a proof of $\phi \wedge \psi$ would consist of both a proof of ϕ, and a proof of ψ;
- a proof of $\phi \vee \psi$ would consist of either of a proof of ϕ, or a proof of ψ, and including information as to which has been proved;
- a proof of $\phi \Rightarrow \psi$ would consist of a procedure by which any proof of ϕ could be transformed to a proof of ψ;
- a proof of $\exists x[\phi(x)]$ would consist of an object x together with a proof of $\phi(x)$;
- a proof of $\forall x[\phi(x)]$ would consist of a procedure by which, given any object x, a proof of $\phi(x)$ could be constructed;
- a proof of $\neg\phi$ is just a proof of $\phi \Rightarrow \bot$;
- no proof for $\bot$ exists.

Kleene's formalized version of this, was then:

Definition 2.1.1 (Kleene's Realizability)**.** Considering a first-order predicate language, including natural numbers (represented by constants $\overline{n}$) and whose function and relation symbols correspond to total recursive functions/relations on $\mathbb{N}$; a natural number e is said to *realize* a sentence ϕ iff:

- ϕ is atomic, which is found true after having evaluation the corresponding recursive functions denoted; and $e = 0$;
- $\phi = \psi \wedge \chi$, and $e = \langle n, m\rangle$ (where $\langle -, -\rangle$ is some effective pairing isomorphism) and n realizes ϕ and m realizes χ;
- $\phi = \psi \vee \chi$, and $e = \langle k, n\rangle$ and either $k = 0$ and n realizes ψ or $k = 1$ and n realizes χ;
- $\phi = \psi \Rightarrow \chi$, and for any $n \in \mathbb{N}$ that realizes ψ, the result $\{e\}n$ of the e^{th} recursive function applied to n, is defined (recursive functions may be partial), and realizes χ;

- $\phi = \neg\psi$, and e realizes $\psi \Rightarrow \overline{0} = \overline{1}$;
- $\phi = \exists x[\psi(x)]$, and $e = \langle n, m\rangle$ and n realizes $\psi(\overline{m})$;
- $\phi = \forall x[\psi(x)]$, and for every $n \in \mathbb{N}$, $e\{n\}$ is defined and realizes $\psi(\overline{n})$. //

This landmark insight led to a whole field of logical research into the subtleties and various alternative versions of realizability. It turns out, for instance, that there is a subtle difference between what can be called 'constructible' (provability in a constructible logic, say Heyting arithmetic), and 'computable' (interpretation of logic w.r.t. an underlying model of computation, say realizability). For whereas every HA provable sentence is realized (the realizer can be taken simply from the corresponding deduction), not every realizable sentence is HA provable. An interesting contestant is the so-called *Markov's principle*, the fourth postulate of the Russian school, stating that $[\forall n(P(n) \vee \neg P(n))] \Rightarrow [\neg\forall n \neg P(n)] \Rightarrow [\exists n P(n)]$ (in other words, if P is decidable, and holds on some n, then some n can indeed be found). This statement can be taken to convey the informal idea of *unbounded search*, indeed definable by a Turing machine, or through the minimalization operator of μ-recursion[1]. But it is clearly not HA provable, i.e. intuitionistically valid, due to the fact that no n can be logically provided. Another example is the negation of the Halting Problem: whereas this statement is vacuously realized due to the Halting Problem being unrealizable, HA takes no position on the matter. For more details, see e.g. Van Oosten [2002].

Where logic may be concerned with realizability due its interesting connection with constructivity (among others), the subject can be seen as a natural fit for computer science as well, due to the way it encompasses a way of reducing more complex notions, such as *data types* and *polymorphism*, to more elementary ones in an *a priori* given model of computation (be it the natural numbers and recursive functions thereover as in Kleene's initial formulation, or any other, as we'll see later). Such a reduction then amounts to a constructive (or, more precisely, computable) account of the more complex notion in terms on the underlying model, which is precisely what is needed in order to *implement* them.

As Longley [1995] and Van Oosten [2002] both note, this second strand of thought can be found in the discovery of the so-called *partial equivalence relation* (PER) construction, which can be traced back to Kreisel [1959], and in particular Girard [1972], who uses it to model System F (or, the second order polymorphic lambda calculus $\lambda 2$). If under the well-known *Curry-Howard-correspondence*, types are seen as propositions (and programs of these types as proofs of the respective propositions), then we can partition the realizers of a type-denoting proposition into a partial equivalence relation $(\sim) \subseteq \mathbb{N} \times \mathbb{N}$, and view the obtained set of equivalence classes as a data type. Essentially, such an equivalence class (denoting an object of the data type) is a collection of various derivations of the proposition involved, which are under some account related as equivalent.

The main work inciting a whole field of research into this account of data types was Hyland [1982]. In this breakthrough, Hyland constructed the *effective topos*, generalizing Kleene's realizability—which worked just for first-order predicate logic at most—to a *topos*, which is a kind of category that can be essentially seen as a constructive variant of the category of sets, and which can be used to model higher-order impredicative logic. The effective topos was also shown to contain a full subcategory equivalent to the PERs on $\mathbb{N}$.

Shortly thereafter, Hyland [1988] (due to an idea of Moggi) and subsequently Longo and Moggi [1991] provided the key insight(s) relating to polymorphism that allowed first System F and subsequently the Calculus of Constructions to be modeled. The crucial (final) idea is that the subcategory of *modest sets* within the effective topos is 'complete' in a sense that certain exponentials are isomorphic to 'polymorphic' function spaces, due to the ability to

[1] I.e. $\lambda d.\lambda e.\mathsf{let}\ n := \mu n((d\{n\})_0 = 0)\ \mathsf{in}\ \langle n, d\{n\}\rangle$, pardon the syntax enhancement. Note the subtle reliance on the meta-theoretical logic employed being classical, without which it cannot be said that the μ-expression is defined.

internalize the PER category as a single object, and the domain of such exponentials. (See theorem 4.3.2.)

These tools then, form the matter from which we'll be able to model the Calculus of Constructions. We will, however, construct our models from the slight generality of considering an (almost) arbitrary *partial combinatory algebra* (PCA) as our model of computation, instead of just the natural numbers and recursive functions thereover. Considering various different PCAs, the constructed models can vary quite interestingly, see e.g. Longley [1995], but this will not be of concern to our present endeavor.

2.2 Partial combinatory algebras

Partial combinatory algebras (PCAs) will form the basis on which our realizability models will be constructed. These can be regarded as our underlying models of computation, i.e. 'abstract machines', as they come equipped with a notion of computation. This notion of computation is then automatically incorporated into our models.

First, recall:

Definition 2.2.1. The *untyped lambda calculus* has terms freely generated over the following grammar, where x denotes any of an infinite supply of variables:

$$M, N ::= x \mid (MN) \mid \lambda x.M$$

The *equational theory* ($=_\beta$) of the untyped lambda calculus is axiomatized by the β-rule

$$(\lambda x.M)N =_\beta M[x := N]$$

and is closed under reflexivity, symmetry, transitivity, and moves into the grammar formation rules. Note that the latter includes the ξ-rule as well: $\big[M =_\beta N \Rightarrow \lambda x.N =_\beta \lambda x.M\big]$. Also note that we tacitly assume terms to be identical over mere renaming of variables (α-equivalence), and that variables are indeed renamed accordingly to prevent name capture on substitution. //

Now, for combinatory algebras.

Definition 2.2.2. A *partial applicative structure* (PA) is a set $\mathcal{D}$ along with a (partial) application operator $- \cdot - : \mathcal{D} \times \mathcal{D} \rightharpoonup \mathcal{D}$. //

There is a subtle distinction to be made between PCA *elements*, and PCA *terms* (or *expressions*). The latter are formed with both elements and variables, and containing no variables, may or may not actually denote PCA elements due to the partiality of application.

Definition 2.2.3. The set of terms/expressions $\mathsf{Terms}(\mathcal{D})$ over a partial applicative structure $\mathcal{D}$ is formed freely from the following grammar, where x denotes any of an infinite supply of variables:

$$t, u ::= x \mid d \mid (t \cdot u)$$

That is, they are formed from variables, elements, and juxtaposition around a binary application symbol. (i) A term t is called *closed* if it contains no variables, and else it is called *open*. (ii) We write $t[x := u]$ to denote the term that is obtained by substituting u for x in t (and renaming bound variables in order to avoid name capture). (iii) A closed term $t \in \mathsf{Terms}(\mathcal{D})$ may or may not denote an element of $\mathcal{D}$ (via the obvious interpretation), due to the partiality of $\mathcal{D}$'s application operator. If it does, we write $t\!\downarrow$ to mean that it is defined, but blur the distinction between the term and the element it denotes. (iv) This also means we interpret the statement $t = u$ to mean that $t\!\downarrow$ and $u\!\downarrow$ and $t = u$ holds for

the elements denoted. This is called *strict equality*. (v) For closed terms t and u, we write $t \asymp u$, for *Kleene equality*, if either $(t\!\uparrow \wedge u\!\uparrow)$ or $(t\!\downarrow \wedge u\!\downarrow) \wedge (t = u)$. (vi) These definitions are then generalized to open terms:

$$\begin{aligned} t\!\downarrow \quad &\text{iff} \quad t[\vec{x} := \vec{d}]\!\downarrow && \text{for all } \vec{d} = d_0 \dots d_n \in \mathcal{D} \\ t = u \quad &\text{iff} \quad t[\vec{x} := \vec{d}] = u[\vec{x} := \vec{d}] && \text{for all } \vec{d} = d_0 \dots d_n \in \mathcal{D} \\ t \asymp u \quad &\text{iff} \quad t[\vec{x} := \vec{d}] \asymp u[\vec{x} := \vec{d}] && \text{for all } \vec{d} = d_0 \dots d_n \in \mathcal{D} \end{aligned}$$

This notation is chosen for brevity, but is essentially equivalent to the *environment model* approach, which is often represented with equations such as $\forall_\rho \big[[\![t]\!]_\rho = [\![i]\!]_\rho \big]$. The difference is that instead of working with environments ρ and interpretation w.r.t. them, the propositional relations $(=)$, $(\asymp)$ and $(-)\!\downarrow$ are overloaded to terms.

For legibility, we omit parentheses and associate application to the left, writing e.g. $x{\cdot}z{\cdot}(y{\cdot}z)$ instead of $(x \cdot z) \cdot (y \cdot z)$. //

Definition 2.2.4. A *partial combinatory algebra* or *PCA*, is a partial applicative structure supplied with two distinguished elements k and s, such that the following term equations hold:

$$\begin{aligned} k \cdot x \cdot y &= x \\ s \cdot x \cdot y &\downarrow \\ s \cdot x \cdot y \cdot z &\asymp x \cdot z \cdot (y \cdot z) \end{aligned}$$

Note that x, y, z are variables, and the equations are open term equations. We abbreviate $i \triangleq s \cdot k \cdot k$. //

Definition 2.2.5. *Combinatory Logic* (CL) is the free (total) combinatory algebra generated over distinct $\{\mathsf{K}, \mathsf{S}\}$ and the axiom scheme that for all $a, b, c \in \mathrm{CL}$, we have $\mathsf{K} \cdot a \cdot b = a$ and $\mathsf{S} \cdot a \cdot b \cdot c = (a \cdot c) \cdot (b \cdot c)$. We write uppercase sans-serif and omit the application operator to emphasize that we're writing an element of (or term over) CL, as in $\mathsf{I} \triangleq \mathsf{SKK}$. //

This naming is due to the fact that such structures are indeed *combinatory complete*, as per the proposition below. This means, essentially, that any definable PCA *term* can be represented by a PCA *element*. Hence, it is *complete* with respect to such definability (in terms of k and s). The *combinatory* part refers to the fact that any effectively computable function (say, per a Turing machine) can be represented combinatorily in terms of such k and s.

Proposition 2.2.6 (Combinatory completeness)**.** *Every PCA $\mathcal{D}$ is combinatory complete, which is to say that for every term t and variable x, there is a term $\lambda\!\!\lambda x.t$ whose variables are those of t, excluding x, such that $(\lambda\!\!\lambda x.t)\!\downarrow$ and for every $d \in \mathcal{D}$, $(\lambda\!\!\lambda x.t) \cdot d \asymp t[x := d]$.*

Proof. We can define the well-known $\lambda\!\!\lambda = \lambda^*$ by induction on the structure of t:

$$\begin{aligned} \lambda^* x.x &\triangleq i & \qquad \lambda^* x.d &\triangleq k \cdot d \\ \lambda^* x.y &\triangleq k \cdot y & \qquad \lambda^* x.(t \cdot u) &\triangleq s \cdot (\lambda^* x.t) \cdot (\lambda^* x.u) \end{aligned}$$

The first three are easily seen. For the fourth, let us have arbitrary t, u, x, d. That $\lambda^* x.(t \cdot u)\!\downarrow$ follows directly by induction. Moreover,

$$\begin{aligned} (\lambda^* x.(t \cdot u)) \cdot d &\triangleq s \cdot (\lambda^* x.t) \cdot (\lambda^* x.u) \cdot d \\ &\asymp ((\lambda^* x.t) \cdot d) \cdot ((\lambda^* x.u) \cdot d) && \text{(definition of } s\text{)} \\ &\asymp t[x := d] \cdot u[x := d] && \text{(IH)} \\ &\asymp (t \cdot u)[x := d] && \text{(an easy induction proof)} \end{aligned}$$

Note that λ^* acts upon the syntactic structure of the term that it is applied to, and due to this *non-algebraic* form is not at all automatically suspectible to encoding within the PCA itself. □

Proposition 2.2.7. *More generally, for $u \in \mathsf{Terms}(\mathcal{D})$ instead of just $d \in \mathcal{D}$, we have that $(\mathbin{\curlywedge} x.t) \cdot u \asymp t[x := u]$ holds.*

Proposition 2.2.8. *A PCA $\mathcal{D}$ is a singleton set iff $s = k$.*

Proof. From the axioms we have $s \cdot x \cdot y \cdot z \asymp (x \cdot z) \cdot (y \cdot z)$ as well as $k \cdot x \cdot y \cdot z \asymp x \cdot z$. Now if we assume $s = k$, then $(x \cdot z) \cdot (y \cdot z) \asymp x \cdot z$. Substituting $x := i$ and $y := k \cdot q$, we then have that, crucially, $z \cdot q \asymp z$. This can be used to show that every two elements d, e are equal. Using the instance $k \cdot i \asymp k$, we have $e = i \cdot e \asymp k \cdot i \cdot d \cdot e \asymp k \cdot d \cdot e = d$. (And given that $d, e\!\downarrow$, this is a strict equality.) □

It is important to note that, although we can simulate the untyped lambda calculus in this way, the 'meta-syntax' that we obtain by denoting PCA terms with $\mathbin{\curlywedge}$ need not necessary obey the rules we would expect from the untyped lambda calculus' λ. To make this formal, we need the following.

Definition 2.2.9. The set of *meta-terms* over a PCA $\mathcal{D}$ will be denoted $\Lambda(\mathcal{D})$—as it is essentially just the untyped lambda calculus, over the set of constants $\mathcal{D}$—and is formed freely from the following grammar:

$$M, N ::= x \mid d \mid (M \cdot N) \mid \lambda x.M$$

We will allow ourselves to write $M =_\beta N$, interpreting the terms simply in the untyped lambda calculus (with added constants from $\mathcal{D}$). We will denote by $M_\mathcal{D}$ explicitly the PCA term obtained after having iteratively replaced all occurrences of $(\lambda x.N)_\mathcal{D}$ with $\mathbin{\curlywedge} x.N_\mathcal{D}$ (denoting a $\mathcal{D}$-term, given a particular choice of $\mathbin{\curlywedge}$). For example, we might use λ^*, the operation of which was given in proposition 2.2.6. //

Although combinatory completeness gives us that

$$\big((\lambda x.N) \cdot M\big)_\mathcal{D} \asymp (\lambda x.N)_\mathcal{D} \cdot M_\mathcal{D} \asymp (\mathbin{\curlywedge} x.N_\mathcal{D}) \cdot M_\mathcal{D} \asymp N_\mathcal{D}[x := M_\mathcal{D}] \cdots$$

we don't necessary have the final step that would have the β-rule behave as one would have it:

$$\cdots \asymp \big(N[x := M]\big)_\mathcal{D}$$

The following examples show this explicitly.

Example 2.2.10. The β-rule $\big[((\mathbin{\curlywedge} x.M) \cdot u)_\mathcal{D} \asymp (M[x := u])_\mathcal{D}\big]$ need not hold over the meta-syntax. For example, in CL, and using $\mathbin{\curlywedge} = \lambda^*$ we have the inequality:

$$\begin{aligned}(\lambda x.\lambda y.x)_{\mathrm{CL}}(\mathsf{SS}) &= (\lambda x.\mathsf{K}x)_{\mathrm{CL}}(\mathsf{SS}) \quad \neq \quad \big((\lambda y.x)[x := \mathsf{SS}]\big)_{\mathrm{CL}} = \lambda^* y.(\mathsf{SS}) \\ &= (\lambda^* x.\mathsf{K}x)(\mathsf{SS}) \qquad\qquad\qquad\qquad\qquad\quad = \mathsf{S}(\mathsf{KS})(\mathsf{KS}) \\ &= \mathsf{S}(\mathsf{KK})\mathsf{I}(\mathsf{SS}) \\ &= \mathsf{K}(\mathsf{SS})\end{aligned}$$ ◿

Example 2.2.11. The ξ-rule $\big[M_\mathcal{D} \asymp N_\mathcal{D} \Rightarrow (\mathbin{\curlywedge} x.M)_\mathcal{D} \asymp (\mathbin{\curlywedge} x.N)_\mathcal{D}\big]$ need not hold over the meta-syntax. For example, in CL, and using $\mathbin{\curlywedge} = \lambda^*$, we have $\mathsf{S} = \mathsf{KSS}$, although the following are not equal:

$$\begin{aligned}(\lambda x.\mathsf{S})_{\mathrm{CL}} &= (\lambda^* x.\mathsf{S}) \quad \neq \quad \big(\lambda x.(\mathsf{KSS})\big)_{\mathrm{CL}} = \lambda^* x.(\mathsf{KSS}) \\ &= \mathsf{KS} \qquad\qquad\qquad\qquad\quad = \mathsf{S}(\mathsf{S}(\mathsf{KK})(\mathsf{KS}))(\mathsf{KS})\end{aligned}$$ ◿

We can now take one of a few approaches to remedy this situation. On of them is to only consider certain 'well-behaved' PCAs. In the theory of combinatory algebras, it is common to distinguish two kinds of 'well-behavedness': *λ-algebra* and *λ-models* (cf. Barendregt [1984] for a thorough account).

Definition 2.2.12. A PCA $\mathcal{D}$ with a particular choice of $\mathbin{\curlywedge}$ is called:

- a *λ-algebra*, if for all $M, N \in \Lambda(\mathcal{D})$, we have $M_\mathcal{D} = N_\mathcal{D}$ whenever $M =_\beta N$, i.e. the meta-terms (seen as normal lambda terms) obey β conversion (i.e. examples such as 2.2.10 and 2.2.11 cannot occur);
- a *λ-model*, if additionally it satisfies the axiom of *weak extensionality*, for all $t, u \in \mathsf{Terms}(\mathcal{D})$:

$$\frac{t \asymp u}{\mathbin{\curlywedge} x.t \asymp \mathbin{\curlywedge} x.u}$$

//

A second approach, as is taken in Longley [1995], is to detail the various instances of the β-rule that are still valid without assuming anything on the side of the PCA. To just give a flavor of what we're dealing with, we include the following from Longley [1995, p. 32]:

Lemma 2.2.13 (Longley [1995])**.** *The following instances of the β-rule are valid, for $N, M \in \Lambda(\mathcal{D})$, and if we take $\mathbin{\curlywedge} = \lambda^*$:*

(β1) $\big((\lambda^* x.N) \cdot t\big)_\mathcal{D} \asymp \big(N[x := t]\big)_\mathcal{D}$

(β2) *If $M{\downarrow}$, x is not free in M, and no free occurrence of x in N appears under a λ^*-abstraction, then $(\lambda^* x.N)_\mathcal{D} \cdot M_\mathcal{D} \asymp \big(N[x := M]\big)_\mathcal{D}$*

Proof. Longley [1995, p. 33] □

A third approach becomes viable given our particular subject matter, being realizability models that are in themselves of a quite extensional nature. Hence, it will often suffice to talk of extensionally equal (meta-)terms. This saves us from having to often check the validity of applications of the β-rule, as well as unnecessarily restricting the range of PCAs considered.

Definition 2.2.14. For closed terms t and u, we write $t \sim u$ to denote that they are *extensionally equal*, i.e.

$$t \sim u \quad \text{iff} \quad \forall_{v \in \mathsf{Terms}(\mathcal{D})} \big[\, t \cdot v \asymp u \cdot v \,\big]$$

and this is generalized to open terms similarly as above. //

Lemma 2.2.15. *$k \cdot (N \cdot M)_\mathcal{D} \sim s \cdot (\lambda^* y.N_\mathcal{D}) \cdot (\lambda^* y.M_\mathcal{D})$, for all meta-terms N, M in which the variable y does not occur freely.*

Proof. A boring but lengthy proof by induction on the structure of N and M. □

Definition 2.2.16. A *context* $C\big[\,-\,\big]$ is a term (either meta- or not, which will be clear from the circumstances), with a 'hole' (which may occur multiple times). Note that this hole may occur multiple times. The term $C\big[\,N\,\big]$ then has N put in the place of these holes (allowing variables to be captured). //

Proposition 2.2.17. *For $N, M \in \Lambda(\mathcal{D})$, we have at least $((\lambda x.N) \cdot M)_\mathcal{D} \sim (N[x := M])_\mathcal{D}$.*

Proof Sketch. As described above, we only need to prove the last part, i.e. $N_\mathcal{D}[x := M_\mathcal{D}] \sim (N[x := M])_\mathcal{D}$. The crucial case is when N contains an abstraction term $\lambda y.N'$ and M contains an application $M = M_1 M_2$, as in example 2.2.10, because then M gets 'syntactically broken up' on the right side, whereas this does not happen on the left. The following, somewhat informal calculation shows how this does not affect function extensionality, for the 'base' case where $N = \lambda y.C[x]$ and $M = M_1 M_2$. The occurrences of $\big[\,-\,\big]$ just denote

underspecified contexts, the structure of which should be clear, but which are a unwieldy to spell out without resorting to a full induction proof: the idea is that λ^* just acts on it as it does, and we're only interested in the effects of this procedure on the hole. (With a slightly more careful analysis of the contexts involved, and a full induction proof, this calculation can be extended to prove the claim.)

$$\begin{aligned}
\big((\lambda x\, y.\lceil\, x\,\rceil)\cdot(M_1M_2)\big)_{\mathcal{D}} &\asymp (\lambda^* x.\lceil\, k\cdot x\,\rceil_{\mathcal{D}})\cdot(M_1M_2)_{\mathcal{D}} \\
&\asymp \lceil\, s\cdot(k\cdot k)\cdot i\,\rceil\cdot(M_1M_2)_{\mathcal{D}} \\
&\asymp \lceil\, k\cdot(M_1M_2)_{\mathcal{D}}\,\rceil \\
&\sim \lceil\, s\cdot(\lambda^* y.M_{1\mathcal{D}})\cdot(\lambda^* y.M_{2\mathcal{D}})\,\rceil \qquad \text{(by lemma 2.2.15)} \\
&\asymp \big(\lambda y.\lceil\, M_1\cdot M_2\,\rceil\big)_{\mathcal{D}} \\
&\asymp \big((\lambda y.\lceil\, x\,\rceil)[x := M_1M_2]\big)_{\mathcal{D}} \qquad \square
\end{aligned}$$

Proposition 2.2.18. *Any PCA admits (among others) pairing and projection, boolean logic, an encoding of natural numbers, composition, and a fixed-point combinator, as such:*

Pairing

$$\begin{aligned}
(\mathsf{pair}\cdot x\cdot y)\!\downarrow \\
\mathsf{proj}_0\cdot(\mathsf{pair}\cdot x\cdot y) &\asymp x \\
\mathsf{proj}_1\cdot(\mathsf{pair}\cdot x\cdot y) &\asymp y
\end{aligned}$$

Boolean logic

$$\begin{aligned}
(\mathsf{if}\cdot x\cdot y)\!\downarrow \\
\mathsf{if}\cdot\mathsf{true}\cdot y\cdot z &\asymp y \\
\mathsf{if}\cdot\mathsf{false}\cdot y\cdot z &\asymp z \\
\mathsf{not}\cdot\mathsf{false} &= \mathsf{true} \\
\mathsf{not}\cdot\mathsf{true} &= \mathsf{false}
\end{aligned}$$

Natural numbers

$$\begin{aligned}
\mathsf{iszero}\cdot\ulcorner 0\urcorner &= \mathsf{true} \\
\mathsf{iszero}\cdot\ulcorner n+1\urcorner &= \mathsf{false} \\
\mathsf{succ}\cdot\ulcorner n\urcorner &= \ulcorner n+1\urcorner \\
\mathsf{pred}\cdot\ulcorner n+1\urcorner &= \ulcorner n\urcorner
\end{aligned}$$

Composition

$$\begin{aligned}
\mathsf{comp}\cdot f\cdot g\!\downarrow \\
\mathsf{comp}\cdot f\cdot g\cdot x &\asymp f\cdot(g\cdot x)
\end{aligned}$$

Fixed-points

$$\begin{aligned}
\mathsf{Y}\cdot f\!\downarrow \\
f\cdot(\mathsf{Y}\cdot f) &\sim \mathsf{Y}\cdot f
\end{aligned}$$

Proof. By standard tricks for the untyped λ-calculus, and using our convention that the PCA is a λ-algebra. Denote $k' \triangleq \lambda^* x.\lambda^* y.\, y$.

$$\begin{aligned}
\mathsf{pair} &\triangleq \lambda^* xyz.\,(z\cdot x\cdot y) & \mathsf{if} &\triangleq \lambda^* xyz.\,(x\cdot y\cdot z) \\
\mathsf{proj}_0 &\triangleq \lambda^* x.\,(x\cdot k) & \mathsf{true} &\triangleq k \\
\mathsf{proj}_1 &\triangleq \lambda^* x.\,(x\cdot k') & \mathsf{false} &\triangleq k' \\
& & \mathsf{not} &\triangleq \lambda^* xy.\lambda^* z.\, x\cdot z\cdot y \\
\mathsf{comp} &\triangleq \lambda^* fgx.\, f\cdot(g\cdot x) & & \\
\mathsf{Y} &\triangleq \big(\lambda^* xfz.\, f\cdot(x\cdot x\cdot f)\cdot z\big)\cdot\big(\lambda^* xfz.\, f\cdot(x\cdot x\cdot f)\cdot z\big) & &
\end{aligned}$$

There are a number of well-known encodings for natural numbers. These are due to Barendregt [1984]:

$$\begin{aligned}
\ulcorner 0\urcorner &\triangleq i & \mathsf{iszero} &\triangleq \lambda^* x.\,(x\cdot k) \\
\ulcorner n+1\urcorner &\triangleq \mathsf{pair}\cdot\ulcorner n\urcorner\cdot\mathsf{false} & \mathsf{succ} &\triangleq \lambda^* x.\,\mathsf{pair}\cdot\mathsf{false}\cdot x \\
& & \mathsf{pred} &\triangleq \lambda^* x.\,(x\cdot k')
\end{aligned}$$

The not-so-standard part is in verifying that Longley's β-rules apply to these terms. This amounts to some boring computations.

$$\begin{aligned}
\mathsf{pair}\cdot x\cdot y &\asymp (\lambda^* z.\, z\cdot x\cdot y)\cdot x\cdot y \\
&\asymp \lambda^* z.\, z\cdot x\cdot y \quad\downarrow \qquad (\beta 1, \text{ twice})
\end{aligned}$$

$$\begin{aligned}
\mathsf{proj}_0 \cdot (\mathsf{pair} \cdot x \cdot y) &\asymp (\lambda^* x.\,(x \cdot k)) \cdot (\lambda^* z.\, z \cdot x \cdot y) \\
&\asymp (\lambda^* z.\, z \cdot x \cdot y) \cdot k && (\beta 2) \\
&\asymp k \cdot x \cdot y && (\beta 2) \\
&\asymp x
\end{aligned}$$

The booleans and natural numbers work similarly, as they amount to the same kind of pairings/projections, as does composition. For the fixed-point combinator, note that we've used a slightly different version than the usual Y, to ensure that $\mathsf{Y} \cdot f\!\downarrow$. For this one, we don't have $f \cdot (\mathsf{Y} \cdot f) \asymp \mathsf{Y} \cdot f$, but they are extensionally equal. Let $X = (\lambda^* x f z.\, f \cdot (x \cdot x \cdot f) \cdot z)$, and verify:

$$\begin{aligned}
\mathsf{Y} \cdot f &\asymp X \cdot X \cdot f \\
&\asymp (\lambda^* f z.\, f \cdot (X \cdot X \cdot f) \cdot z) \cdot f && (\beta 2) \\
&\asymp \lambda^* z.\, f \cdot (X \cdot X \cdot f) \cdot z \quad \downarrow && (\beta 1)
\end{aligned}$$

$$\begin{aligned}
\mathsf{Y} \cdot f \cdot z &\asymp (\lambda^* z.\, f \cdot (X \cdot X \cdot f) \cdot z) \cdot z && \text{(as above)} \\
&\asymp f \cdot (\mathsf{Y} \cdot f) \cdot z && (\beta 1, \text{ def. } \mathsf{Y})
\end{aligned}$$

□

Convention 2.2.19. *We will allow ourselves to write* $\langle t, u \rangle$ *instead of* $(\mathsf{pair} \cdot t \cdot u)$*, and* t_i *instead of* $(\mathsf{proj}_i \cdot t)$*, and* $(t \circ u)$ *instead of* $(\mathsf{comp} \cdot t \cdot u)$*.*

2.2.1 Examples of PCAs

Example 2.2.20 (Kleene's recursion-theoretic $\mathcal{K}_1$)**.** Kleene's original realizability was based on $\mathcal{K}_1$, also called *the first Kleene algebra*, which consists of the natural numbers $\mathbb{N}$, also taken to represent partial recursive functions through some choice of coding $\{\phi_n\}_{n\in\mathbb{N}}$, e.g. Gödel numbering. Partial application is called *Kleene application*, and written $\{n\}m \triangleq \phi_n(m)$, i.e. the n^{th} partial recursive function, applied to m. The effective topos arises from this PCA.

Properties. $\mathcal{K}_1$ is not (weakly) extensional, since there are an infinite number of (coded) programs that compute any particular partial function.[2] ◿

Example 2.2.21 (Term models)**.** One can also take *term models*. For instance, Λ/β denotes the (open) term model of the untyped lambda calculus modulo β-convertibility, and CL is already a PCA as well. Application is obviously total in both of these cases.

Properties. We have seen that CL is not weakly extensional. On the other hand, because we identify $\mathsf{\Lambda}$ with λ in Λ/β, this PCA is (weakly) extensional. ◿

Example 2.2.22 (Graph model P_ω)**.** A well-known class of PCAs are the so-called *graph models*, the first of which was discovered by D. Scott [ref] as an unexpected set-theoretical model of the untyped lambda calculus, within his domain-theoretical framework. The idea is that a suitable notion of *domain* leads to a category with a *reflexive* object D, i.e. for which $D^D \preccurlyeq D$. The cardinality of the function space D^D does not exceed that of D, due to the fact that only *continuous* functions are morphisms, which in this setting is related to computability.

[2] It is not possible to create a duplicate-avoiding recursive enumeration of partial recursive functions. Suppose, for contradiction, equality of partial recursive functions were semi-decidable. Then in particular, we could semi-decide whether $\{n\}0\!\downarrow \Leftrightarrow \{m\}0\!\downarrow$. But then, we could decide K_0, the problem whether a given partial recursive function halts on 0: if h denotes any partial recursive function that always halts, and ℓ any partial recursive function that always loops, then we can decide K_0 by semi-deciding $\{n\}0\!\downarrow \Leftrightarrow \{k\}0\!\downarrow$ and $\{n\}0\!\downarrow \Leftrightarrow \{\ell\}0\!\downarrow$ in parallel, one of which will return 1 eventually.

We will briefly outline the simplest graph model P_ω, independently due to both D. Scott and G. Plotkin. We take *directed-complete partial orders* (DCPOs) of the form $(D, \sqsubseteq, \bigsqcup)$ as our notion of domain, where $D \neq \varnothing$ and $\bigsqcup$ assigns a *supremum* to each *directed* set $X \subseteq D$, meaning that every two elements of X have a common greater-or-equal element, also in X. DCPO morphisms f are functions that respect these data, so that in particular they respect suprema: $f(\bigsqcup X) = \bigsqcup\{f(x) \mid x \in X\}$. Now define the DCPO $P_\omega \triangleq (\mathcal{P}(\mathbb{N}), \subseteq, \bigcup)$, and fix any encoding $(-\triangleright-) : \mathcal{P}_{\text{fin}}(\mathbb{N}) \times \mathbb{N} \cong \mathbb{N}$. Then we can define the *graph* of a morphism $f \in {P_\omega}^{P_\omega}$ by

$$\begin{aligned} \text{graph} &: {P_\omega}^{P_\omega} \to P_\omega \\ \text{graph}(f) &\triangleq \{x \triangleright n \mid x \text{ finite} \wedge n \in f(x)\} \end{aligned}$$

The trick employed here, is that due to the nature of these 'computable' functions, they can be fully represented by their set of finite approximations: $f(x) = f(\bigcup\{\text{finite } x' \subseteq x\}) = \bigcup\{f(x') \mid \text{finite } x' \subseteq x\}$. (Obviously the set of finite subsets of some fixed x is directed.) We can indeed decode them with $\text{decode}(g) \triangleq x \mapsto \{n \mid \exists x' \subseteq x,\ x' \triangleright n \in g\}$, noting that

$$\begin{aligned} \text{decode}(\text{graph}(f))(x) &= \{n \mid \exists x' \subseteq x,\ x' \triangleright n \in \text{graph}(f)\} \\ &= \{n \mid n \in f(x)\} \\ &= f(x) \end{aligned}$$

Thus, P_ω is a reflexive domain. We will now quickly work to making P_ω a PCA, instead of continuing the general construction on reflexive domains. Noting that not all x will necessary be of the form $\text{graph}(f)$, we define application as $a \cdot x \triangleq \{n \mid \exists x' \subseteq x,\ x' \triangleright n \in a\}$, for which indeed $\text{graph}(f) \cdot x = f(x)$. As for a choice of k and s, just define (where it is crucially of importance that $(- \cdot -)$ is continuous too, which is indeed the case):

$$\begin{aligned} k &\triangleq \{x \triangleright y \triangleright n \mid n \in x \wedge x, y \in \mathcal{P}_{\text{fin}}(\mathbb{N})\} \\ s &\triangleq \{x \triangleright y \triangleright z \triangleright n \mid n \in (x \cdot z) \cdot (y \cdot z) \wedge x, y, z \in \mathcal{P}_{\text{fin}}(\mathbb{N})\} \end{aligned}$$

Properties. Graph models such as P_ω turn out to be extensional precisely when the underling reflexive object has $D^D \cong D$, which isn't the case for P_ω. For more details, see e.g. the grand overview of 'webbed models' (which generalize graph models) in Berline [2000]. ◿

2.3 The Calculus of Constructions

The *Calculus of Constructions* (CC) is one of the more well-understood, expressive type theories around, enjoying a rather simple definition. Many proof assistants are built around extensions of CC, such as the Calculus of Inductive Constructions. Depending on one's point of departure, one usually presents CC in one of two (equivalent) 'flavors': by means of a *semantic* presentation, or as a *Pure Type System*. The former aims at aligning the syntax with the categorical constructs used to make sense of them (over the extended *types-as-propositions* paradigm), where the latter aims at syntactic coherence and conceptual clarity. Although in this thesis we will mainly adhere to the former, we will in this section briefly outline both presentations. This way, we can clear up some [...?].

2.3.1 The Pure Type System presentation of CC

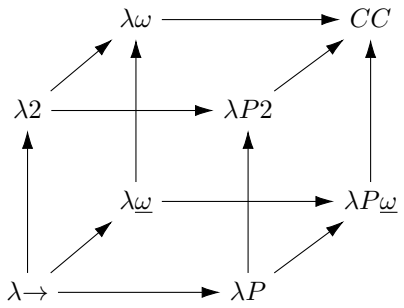
The Pure Type System approach is a particularly elegant way of conceptualizing and presenting the constructs of certain type theories (including the Calculus of Constructions),

and is due to Barendregt [1991]. We will not go into specific details of the general framework, but rather focus on the conceptual underpinnings of CC.

The idea, informally, is that, given that type theories are based on the distinction between *terms* and *types* (as this forms the basis of the *typing relation*), most proposed type theories lend their character to a particular combination of the following:

- allowing types to depend on terms (the result of which is a *dependent* type theory, and by the Curry-Howard correspondence this corresponds to the introduction of *predicates*);
- allowing terms to depend on types (the result of which is a *polymorphic* type theory);
- allowing types to depend on types (the result of which is a *higher-order* type theory);
- allowing terms to depend on terms—this is always the case.

The *lambda cube* (also introduced in Barendregt [1991]) shows how various well-known type theories result from these combinations, where the three axes correspond to the first three of the above items:



The formal definition of a Pure Type System is parametrized by three sets (*sorts* S, *axioms* A, and *rules* R), and if we take $S = \{\star, \Box\}$, $A = \{(\star : \Box)\}$, and R any subset of $\{(\star,\star),(\star,\Box),(\Box,\star),(\Box,\Box)\}$ including $(\star,\star)$, we get the 8 systems of the lambda cube. The Calculus of Constructions is the most expressive system of the cube, obtained by the R containing all four of these rules.

Definition 2.3.1. The following is a *Pure Type System* presentation of the Calculus of Constructions. The presentation is stated slightly different than usual, due to our purposes of aligning it with the semantic presentation. The difference, however, is superficial, and the presentation remains equivalent to the original exposition (see Barendregt [1991]).

The pre-syntax is formed by:

$$\begin{aligned} s &::= \star \mid \Box \\ T, U, M, N &::= s \mid x \mid \Pi(x : T).T \mid \lambda(x : T).T \mid TT \\ \Gamma, \Delta &::= \cdot \mid \Gamma, x^T \\ J &::= \Gamma \mid \Gamma \vdash T : T \end{aligned}$$

Definitional equivalence ($\equiv$) can be kept out of the judgment/inference structure, because it does not rely on ambient information, and is solely based on syntactic β-conversion (over pre-terms). Two pre-terms T, U are considered definitionally equivalent ($T \equiv U$) if they β-convertible (which moves into pre-term construction).

Valid judgments are those that can be derived from the inferences of figure 1.

(Well-)typed terms are pre-terms T that fit into a valid judgment $\Gamma \vdash T : U$ or $\Gamma \vdash U : T$, for a certain pre-term U and context Γ. //

Context formation

$$(\mathsf{cempty})\ \frac{}{\cdot} \qquad (\mathsf{cext})\ \frac{\Gamma \vdash T : s}{\Gamma, x^T}$$

Term formation

$$(\mathsf{axiom})\ \frac{\Gamma}{\Gamma \vdash \star : \Box} \qquad (\mathsf{var})\ \frac{\Gamma, x^T, \Delta}{\Gamma, x^T, \Delta \vdash x : T}$$

$$(\Pi)\ \frac{\Gamma \vdash T : s_1 \qquad \Gamma, x^T \vdash U : s_2}{\Gamma \vdash \Pi(x : T).U : s_2} \qquad (\lambda)\ \frac{\Gamma, x^T \vdash M : U \qquad \Gamma \vdash \Pi(x : T).U : s}{\Gamma \vdash \lambda(x : T).M : \Pi(x : T).U}$$

$$(\mathsf{app})\ \frac{\Gamma \vdash M : \Pi(x : T).U \qquad \Gamma \vdash N : T}{\Gamma \vdash MN : U[x := N]} \qquad (\mathsf{conv})\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash U : s \qquad T \equiv U}{\Gamma \vdash M : U}$$

Figure 1: Inference rules of PTS CC.

Remark 2.3.2. *The typeable terms of the PTS can be split into the following disjoint sets, displaying its 'leveled' typeability relation:*

- $\{\Box\}$
- *The set of* kinds*, which are terms K that can be typed as $\Gamma \vdash K : \Box$. This includes $\star$, but also terms such as $\Pi(x : s).\star$.*
- *The set of* constructors*, which are terms P that can be typed as $\Gamma \vdash P : K$. This set can be further divided into* types $A : \star$ *and* non-types C *such as the typical* $\mathsf{Vec} : \Pi(n : \mathbb{N}).\star$.
- *The set of* objects*, which are terms M that can be typed as $\Gamma \vdash M : A$.*

Example 2.3.3 (Polymorphic identity in PTS CC)**.** The polymorphic identity type and function can be derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{T_1}{A^\star \vdash A : \star}(\mathsf{var})}{A^\star, x^A}(\mathsf{cext})}{A^\star, x^A \vdash x : A}(\mathsf{var}) \qquad \dfrac{\dfrac{T_1}{A^\star \vdash A : \star}(\mathsf{var}) \qquad \dfrac{T_1}{A^\star, x^A \vdash A : \star}(\mathsf{var})}{A^\star \vdash A \to A : \star}(\Pi)}{A^\star \vdash \lambda(x : A).x : A \to A}(\lambda) \qquad T_2}{\vdash \lambda(A : \star).\lambda(x : A).x : \Pi(A : \star).\ A \to A}(\lambda)$$

where

$$T_1 \triangleq \left[\dfrac{\dfrac{\dfrac{}{\cdot}(\mathsf{cempty})}{\vdash \star : \Box}(\mathsf{axiom})}{A^\star}(\mathsf{cext})\right] \qquad T_2 \triangleq \left[\dfrac{\dfrac{\dfrac{}{\cdot}(\mathsf{cempty}) \qquad \dfrac{T_1}{A^\star \vdash A : \star}(\mathsf{var}) \qquad \dfrac{T_1}{A^\star, x^A \vdash A : \star}(\mathsf{var})}{\vdash \star : \Box}(\mathsf{axiom}) \qquad A^\star \vdash A \to A : \star}{\vdash \Pi(A : \star).\ A \to A : \star}(\Pi)\right]$$

◿

Example 2.3.4 (Leibniz equality in PTS CC)**.** The Leibniz equality type (for any $A : \star$)

$$(a \stackrel{L}{=}_A b) \triangleq \Pi(C : \Pi(x : A).\star).\, Ca \to Cb$$

can be derived as follows, where $\Gamma \triangleq A^\star, a^A, b^A$:

$$(\Pi)_{(\Box,\star)} \dfrac{T_4 \qquad (\Pi)_{(\star,\star)} \dfrac{T_5 \qquad T_6}{\Gamma, C^{\Pi(x:A).\star} \vdash Ca \to Cb : \star}}{\Gamma \vdash (a \stackrel{L}{=}_A b) : \star}$$

where

$$T_3 \triangleq \left[\begin{array}{c} (\mathsf{cempty}) \dfrac{}{\cdot} \\ (\mathsf{axiom}) \dfrac{}{\vdash \star : \Box} \\ (\mathsf{cext}) \dfrac{}{A^\star} \\ (\mathsf{var}) \dfrac{}{A^\star \vdash A : \star} \\ (\mathsf{cext}) \dfrac{}{A^\star, a^A} \\ (\mathsf{var}) \dfrac{}{A^\star, a^A \vdash A : \star} \\ (\mathsf{cext}) \dfrac{}{\Gamma} \end{array} \right] \qquad T_4 \triangleq \left[\begin{array}{c} (\mathsf{var}) \dfrac{T_3}{\Gamma \vdash A : \star} \qquad (\mathsf{cext}) \dfrac{(\mathsf{var}) \dfrac{T_3}{\Gamma \vdash A : \star}}{\Gamma, x^A} \qquad (\mathsf{axiom}) \dfrac{}{\Gamma, x^A \vdash \star : \Box} \\ (\Pi)_{(\star,\Box)} \dfrac{}{\Gamma \vdash \Pi(x : A).\star : \Box} \end{array} \right]$$

$$T_5 \triangleq \left[\begin{array}{c} (\mathsf{var}) \dfrac{T_4}{\Gamma, C^{\Pi(x:A).\star}} \qquad (\mathsf{var}) \dfrac{T_4}{\Gamma, C^{\Pi(x:A).\star}} \\ (\mathsf{app}) \dfrac{}{\Gamma, C^{\Pi(x:A).\star} \vdash C : \Pi(x : A).\star} \qquad \dfrac{}{\Gamma, C^{\Pi(x:A).\star} \vdash a : A} \\ \hline \Gamma, C^{\Pi(x:A).\star} \vdash Ca : \star \end{array} \right]$$

$$T_6 \triangleq \left[\begin{array}{c} (\mathsf{var}) \dfrac{\dfrac{T_5}{\Gamma, C^{\Pi(x:A).\star} \vdash Ca : \star}}{\Gamma, C^{\Pi(x:A).\star}, x^{Ca}} \qquad (\mathsf{var}) \dfrac{\dfrac{T_5}{\Gamma, C^{\Pi(x:A).\star} \vdash Ca : \star}}{\Gamma, C^{\Pi(x:A).\star}, x^{Ca}} \\ (\mathsf{app}) \dfrac{}{\Gamma, C^{\Pi(x:A).\star}, x^{Ca} \vdash C : \Pi(x : A).\star} \qquad \dfrac{}{\Gamma, C^{\Pi(x:A).\star}, x^{Ca} \vdash a : A} \\ \hline \Gamma, C^{\Pi(x:A).\star}, x^{Ca} \vdash Ca : \star \end{array} \right]$$

Note how, in the derivation, the (Π) rules have been annotated with their specific instance, and that forming the Leibniz equality type needs dependent types $(\star, \Box)$ to type the family of types C indexed over A, as well as polymorphism $(\Box, \star)$ to type the Leibniz equality type itself. Type operators $(\Box, \Box)$ are not required. ◿

2.3.2 The semantic presentation of CC

Whereas the Pure Type System presentation aims at reducing the syntax to a coherent conceptual minimum, the semantic presentation aims at adhering to the *types-as-propositions* paradigm, and as such is particularly useful for building categorical models.

The syntax clearly delineates *two* syntactical collections, assigning each construct to the realm of (*contexts* of) *types*, to be interpreted as the objects of a target category, or the realm of *terms* (or more generally *context morphisms* or *substitutions*), to be interpreted as certain morphisms of a target category. As such, the typing relation has just these two levels, relating the two.

Definition 2.3.5. The Calculus of Constructions (as per its semantic presentation) is given by the following data.

The pre-syntax and pre-judgments are freely generated over the grammar:

$$\begin{aligned}
\Gamma &::= \cdot \mid \Gamma, x^A \\
A, B &::= \mathsf{Prop} \mid \mathsf{Proof}(t) \mid \Pi(x : A).A \\
s, t, u &::= x \mid \lambda(x : A).t \mid \mathsf{app}_{\Pi(x:A).A}(t, t) \mid \forall(x : A).t \\
J &::= \Gamma \mid \Gamma \vdash A \mid \Gamma \vdash t : A \\
&\quad \mid \Gamma \equiv \Delta \mid \Gamma \vdash A \equiv B \mid \Gamma \vdash t \equiv t : A
\end{aligned}$$

And the derivations by which valid judgments are formed by the inferences of figure 2. //

Convention 2.3.6. *For clarity, instead of* $\forall(x : A)$ *and* $\lambda(x : A)$ *and* $\Pi(x : A)$*, we will often write the notationally more pleasing* $\forall x^A$*,* λx^A*, and resp.* $\Pi_{x:A}$*. As is common, we write* $A \to B$ *instead of* $\Pi_{x:A}.B$ *if* x *does not occur freely in* B*. Similarly, we will write* $A \Rightarrow b$ *instead of* $\forall x^A.b$ *(if* x *does not occur freely in* b*), and in particular also* $a \Rightarrow b$ *instead of* $\forall x^{\mathsf{Proof}(a)}.b$*.*

Example 2.3.7 (Polymorphism in semantic CC)**.** The polymorphic types in this presentation, are those small types of the form $\mathsf{Proof}(\forall a^{\mathsf{Prop}}.t)$. As an example, consider the polymorphic identity function.

$$\begin{aligned}
&\lambda(A : \star).\lambda(x : A).x : \Pi(A : \star).\ A \to A && \text{in PTS CC} \\
&\lambda a^{\mathsf{Prop}}.\ \lambda x^{\mathsf{Proof}(a)}.\ x : \mathsf{Proof}(\forall a^{\mathsf{Prop}}.\ \forall x^{\mathsf{Proof}(a)}.\ a) && \text{in semantic CC}
\end{aligned}$$

The context $\Gamma = a^{\mathsf{Prop}}, x^{\mathsf{Proof}(a)}$ can be easily created, say with deduction tree T_7. Then,

$$T_8 \triangleq \left[\begin{array}{c} (\lambda)\ \dfrac{(\mathsf{var})\ \dfrac{T_7}{a^{\mathsf{Prop}}, x^{\mathsf{Proof}(a)} \vdash x : \mathsf{Proof}(a)}}{a^{\mathsf{Prop}} \vdash \lambda x^{\mathsf{Proof}(a)}.\ x : \Pi_{x:\mathsf{Proof}(a)}\ \mathsf{Proof}(a)} \\ \hline \vdash \lambda a^{\mathsf{Prop}}.\ \lambda x^{\mathsf{Proof}(a)}.\ x : \Pi_{a:\mathsf{Prop}}\ \Pi_{x:\mathsf{Proof}(a)}\ \mathsf{Proof}(a) \end{array} \right]$$

Interestingly, at this point we can use the polymorphic identity function, but it's not clear that its type is indeed a small type. For that, we need the (proof coherence) rule. Let T_9 be the deduction tree that, using transitivity of $(\equiv)$ and multiple usages of (proof coherence)), proves the following:

$$\begin{aligned}
\mathsf{Proof}(\forall a^{\mathsf{Prop}}.\ \forall x^{\mathsf{Proof}(a)}.\ a) &\equiv \Pi_{a:\mathsf{Prop}}\ \mathsf{Proof}(\forall x^{\mathsf{Proof}(a)}.\ a) \\
&\equiv \Pi_{a:\mathsf{Prop}}\ \Pi_{x:\mathsf{Proof}(a)}\ \mathsf{Proof}(a)
\end{aligned}$$

Then, we can use (conv) to rewrite the polymorphic identity's type as:

$$(\mathsf{conv})\ \frac{T_9 \qquad T_8}{\vdash \lambda a^{\mathsf{Prop}}.\ \lambda x^{\mathsf{Proof}(a)}.\ x : \mathsf{Proof}(\forall a^{\mathsf{Prop}}.\ \forall x^{\mathsf{Proof}(a)}.\ a)}$$ ◿

Example 2.3.8. The Leibniz equality type (for any type A)

$$(a \overset{L}{=}_A b) \triangleq \Pi_{C:A\to\mathsf{Prop}}\ \mathsf{Proof}(Ca) \to \mathsf{Proof}(Cb)$$

Context formation

$$(\mathsf{cempty})\ \frac{}{\cdot} \qquad (\mathsf{cext})\ \frac{\Gamma \vdash A}{\Gamma, x^A} \qquad (\mathsf{ceq})\ \frac{\Gamma, x^A, \Delta \qquad \Gamma \vdash A \equiv B}{\Gamma, x^A, \Delta \equiv \Gamma, x^B, \Delta}$$

Type formation

$$(\mathsf{prop})\ \frac{\Gamma}{\Gamma \vdash \mathsf{Prop}} \qquad (\mathsf{proof})\ \frac{\Gamma \vdash p : \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p)} \qquad (\Pi)\ \frac{\Gamma, x^A \vdash B}{\Gamma \vdash \Pi(x : A).B}$$

$$(\mathsf{proof\ coherence})\ \frac{\Gamma, x^A \vdash p : \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(\forall(x : A).p) \equiv \Pi(x : A).\mathsf{Proof}(p)}$$

Term formation

$$(\mathsf{var})\ \frac{\Gamma, x^A, \Delta}{\Gamma, x^A, \Delta \vdash x : A} \qquad (\lambda)\ \frac{\Gamma, x^A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : \Pi(x : A).B}$$

$$(\mathsf{app})\ \frac{\Gamma \vdash t : \Pi(x : A).B \qquad \Gamma \vdash s : A}{\Gamma \vdash \mathsf{app}_{\Pi(x:A)B}(t, s) : B[x := s]} \qquad (\forall)\ \frac{\Gamma, x^A \vdash p : \mathsf{Prop}}{\Gamma \vdash \forall(x : A).p : \mathsf{Prop}}$$

$$(\mathsf{conv})\ \frac{\Gamma \vdash A \equiv B \qquad \Gamma \vdash t : A}{\Gamma \vdash t : B} \qquad (\mathsf{eq})\ \frac{\Gamma \vdash A \equiv B \qquad \Gamma \vdash s \equiv t : A}{\Gamma \vdash s \equiv t : B}$$

$$(\beta)\ \frac{\Gamma, x^A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash \mathsf{app}_{\Pi(x:A).B}(\lambda(x : A).t, s) \equiv t[x := s] : B[x := s]}$$

Convertibility moves into the syntactic operators

$$\frac{\Gamma, x^A \vdash t \equiv t' : B}{\Gamma \vdash \lambda(x : A).t \equiv \lambda(x : A).t' : \Pi_{x:A} B} \qquad \frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x^A \vdash B \equiv B'}{\Gamma \vdash \Pi(x : A).B \equiv \Pi(x : A').B'}$$

$$\frac{\Gamma \vdash p \equiv p' : \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p) \equiv \mathsf{Proof}(p')} \qquad \frac{\Gamma \vdash t \equiv t' : \Pi_{x:A} B \qquad \Gamma \vdash s \equiv s' : A}{\Gamma \vdash \mathsf{app}_{\Pi(x:A)B}(t, s) \equiv \mathsf{app}_{\Pi(x:A)B}(t', s') : B[x := s]}$$

$$\frac{\Gamma, x^A \vdash p \equiv p' : \mathsf{Prop}}{\Gamma \vdash \forall(x : A).p \equiv \forall(x : A).p' : \mathsf{Prop}}$$

Moreover, for $(\equiv)$ to be an equivalence relation, additional inference rules are needed to postulate this over the various syntactical categories, which we omit for brevity.

Figure 2: Inference rules of semantic CC.

can be derived as follows, where $\Gamma \triangleq A, a^A, b^A$, and T_{10} is an easy derivation tree that deduces the validity of the context $\Gamma, C^{A\to\mathsf{Prop}}, x^{\mathsf{Proof}(Ca)}$.

$$\dfrac{(\mathsf{app})\ \dfrac{(\mathsf{var})\ \dfrac{T_{10}}{\Gamma, C^{A\to\mathsf{Prop}}, x^{\mathsf{Proof}(Ca)} \vdash C : A \to \mathsf{Prop}} \qquad (\mathsf{var})\ \dfrac{T_{10}}{\Gamma, C^{A\to\mathsf{Prop}}, x^{\mathsf{Proof}(Ca)} \vdash b : A}}{(\mathsf{proof})\ \dfrac{\Gamma, C^{A\to\mathsf{Prop}}, x^{\mathsf{Proof}(Ca)} \vdash Cb : \mathsf{Prop}}{(\Pi)\ \dfrac{\Gamma, C^{A\to\mathsf{Prop}}, x^{\mathsf{Proof}(Ca)} \vdash \mathsf{Proof}(Cb)}{(\Pi)\ \dfrac{\Gamma, C^{A\to\mathsf{Prop}} \vdash \mathsf{Proof}(Ca) \to \mathsf{Proof}(Cb)}{\Gamma \vdash \Pi_{C:A\to\mathsf{Prop}}\ \mathsf{Proof}(Ca) \to \mathsf{Proof}(Cb)}}}}$$

Note that we've abbreviated Cx for $\mathsf{app}_{A\to\mathsf{Prop}}(C, x)$. ◿

Remark 2.3.9. *Note that in order to formulate Leibniz equality, we only really need polymorphism (the ↑ axis of the lambda cube) and dependent types (the → axis of the lambda cube). However, in semantic CC, where polymorphism is modeled with the use of the universe type* Prop*, and dependent types 'come along' by lifting small type names (in* Prop*), defined in terms of other types, via* Proof*—we then, interestingly get (small) type operators for free.*

2.4 Interpreting type theory in a category

Dependent types

There are two often-used representations of dependent types, that in appropriate settings can be considered simply of presentational difference: the *display map* presentation, and the *fibrational*, or *indexed objects* presentation. In the former, less machinery is needed, and a dependent type $a^A \vdash B_a$ is presented as a *display map* $f : B \to A$, essentially inducing the dependent type via $B_a = \{b \in B \mid f(b) = a\}$. In the latter, a fibration $\mathbf{E} \xrightarrow{P} \mathbf{B}$ is installed, where $\mathbf{B}$ is a 'base' category of indexing objects, and $\mathbf{E}$ is a 'total' category, in which the dependent types live. An easy example is $\mathsf{Fam}(\mathbf{Set}) \xrightarrow{\pi_0} \mathbf{Set}$. Due to the extensional and set-like nature of the constructions in this thesis, we will leave definitions and propositions for fibrations in full generality aside, and instead work with the specific case of *indexed category* fibrations of the form $\mathsf{Fam}(\mathbf{C}) \xrightarrow{\pi_0} \mathbf{Set}$ (or $\mathsf{Fam}(\mathbf{C}) \xrightarrow{\pi_0} \mathbf{C}$ if we can have an 'underlying set' $|C|$ for any $C \in \mathbf{C}$).

The category $\mathsf{Fam}(\mathbf{C})$ has:

$$\begin{aligned} \text{objects} \quad & \big(I, \{A_i\}_{i\in I}\big) \quad \text{for } I, A_i \in \mathbf{Set} \\ \text{morphisms} \quad & \{A_i\}_{i\in I} \xrightarrow{g} \{B_j\}_{j\in J} \quad \text{with } g = \big(I \xrightarrow{u} J, \{A_i \xrightarrow{g_i} B_{u(j)}\}_{i\in I}\big) \end{aligned}$$

We will often just write $\{A_i\}_i$ instead of $(I, \{A_i\}_i)$, and similarly for morphisms, if they do not reindex (i.e. $u = \mathsf{id}$).

The fibration π_0 just sends pairs (I, A) to I, and morphisms g as above, to u (the reindexing part of the morphism). Above any given set I, we can talk of the *fiber* $\mathbb{P}_I$, which is the subcategory of $\mathsf{Fam}(\mathbf{C})$ with families indexed over I and morphisms that don't reindex over their indexing sets ($\pi_0(g) = \mathsf{id}_I$). For any $u : I \to J$, we can define the *reindexing functor* $u^* : \mathbb{P}_J \to \mathbb{P}_I$, that does nothing else than reindex:

$$u^*(\{X_j\}_{j\in J}) \triangleq \{X_{u(i)}\}_{i\in I}$$

and mapping morphisms accordingly. Due to our restricted setting, fibers are equivalent to indexed categories: $\mathbb{P}_I \cong \mathbf{C}^I$, and we also have $\mathbf{C}^I \times \mathbf{C}^J \cong \mathbf{C}^{I+J}$, where $I + J$ denotes the disjoint sum of I and J.

1. Natural numbers

```
ind N : Type :=
  zero : N
  succ : N → N
```

2. Lists over some type A

```
ind List {A} : Type :=
  nil : List
  cons : A → List → List
```

3. Inductive identity

```
ind Eq {A} : A → A → Type :=
  refl : forall x:A, Eq x x
```

4. Countable ordinals

```
ind Ord : Type :=
  zero : Ord
  succ : Ord → Ord
  lim : (N → Ord) → Ord
```

5. Vectors of length $n \in \mathbb{N}$ over some type A

```
ind Vec {A} : N → Type :=
  vnil : Vec zero
  vcons : forall n:N,
    A → Vec n → Vec (succ n)
```

Figure 3: Some typical inductive types

Inductive types

Practical type theories are often extended with inductive types, corresponding to the idea of *free algebras*, i.e. data types, for a certain signature. Figure 3 lists some typical inductive types, which we will refer back to in subsequent examples.

The common way of understanding inductive types, is as initial F-algebras for certain endofunctors F. As not all functors F make sense, one often restricts attention to polynomial functors, *strictly positive* functors, etc. More on this later.

Definition 2.4.1. An *F-algebra*, for some endofunctor $F : \mathbf{C} \to \mathbf{C}$, is an object C together with a morphism $FC \xrightarrow{c} C$. The category **F-Alg**, or $\mathbf{Alg}(F)$, of F-algebras has such objects (C, c), and morphisms $(C, c) \xrightarrow{g} (D, d)$ with underlying morphisms $g : C \to D$ in $\mathbf{C}$ such that the following square commutes:

$$\begin{array}{ccc} FC & \xrightarrow{c} & C \\ \downarrow{\scriptstyle Fg} & & \downarrow{\scriptstyle g} \\ FD & \xrightarrow{d} & D \end{array}$$

//

Now, inductive types arise as initial F-algebras for polynomial functors, if they exist. Lambek's lemma gives us that for such an initial F-algebra, c is in fact an isomorphism $c : FC \cong C$, justifying the title of *structure map*:

Lemma 2.4.2 (Lambek's Lemma)**.** *If (C, c) is an initial F-algebra, then $c : FC \cong C$.*

Proof. Because (C, c) is an initial F-algebra, there exists a unique algebra morphism $u : (C, c) \to (D, d)$ for any (D, d). Taking $(D, d) = (FC, Fc)$, we get a unique $u : C \to FC$ such that $u \circ c = Fc \circ Fu$. Now consider the morphism $c \circ u : C \to C$, which is an F-algebra map because $u \circ c = Fc \circ Fu = F(c \circ u)$ and hence $(c \circ u) \circ c = c \circ F(c \circ u)$. Because of C's initiality, and $\mathsf{id}_C : C \to C$ being an F-algebra as well, we must have $c \circ u = \mathsf{id}_C$. But then, $Fc \circ Fu = F(c \circ u) = \mathsf{id}_{FC}$, and by definition of u then $u \circ c = F(c \circ u) = \mathsf{id}_{FC}$, and hence we have our isomorphism. □

Example 2.4.3. If they exist, we can find the types of natural numbers, resp. lists over some

type A, as the initial F-algebras for the functors:

$$F_N(X) \triangleq \mathbf{1} + X$$
$$F_{\mathrm{List}(A)}(X) \triangleq \mathbf{1} + A \times X$$

If the initial F_N-algebra exists, it is some $(N, (z, s))$ such that $(z, s) : \mathbf{1} + N \to N$, i.e. $z : \mathbf{1} \to N$ and $s : N \to N$; and initiality ensures that $s \circ z \neq z$.

Moreover, we can define functions by recursion, using initiality. Suppose we want to define some function $f : N \to D$. It then suffices to define $z_f : \mathbf{1} \to D$ and $s_f : D \to D$, together making $(z_f, s_f) : F_N D \to D$ an F_N-algebra, and then let $f : N \to D$ be the unique morphism such that $(z, s) \circ f = F_N f \circ (z_f, s_f)$, by N's initiality. ◿

For dependent inductive types, the above does not suffice. One way of describing dependent inductive (and coinductive) types that is gaining traction, is by way of (F, G)-dialgebras, a generalization of the above. Although in full generality we need fibrations or indexed categories, it will for present purposes suffice to present the definition for just the category of families (indexed by sets) over a certain category.

Definition 2.4.4. An (F, G)-dialgebra, for certain functors $F, G : \mathsf{Fam}(\mathbf{C}) \to \mathsf{Fam}(\mathbf{C})$, is an object C and a morphism $FC \xrightarrow{c} GC$. The category of (F, G)-dialgebras contains such objects (C, c), and has morphisms $(C, c) \xrightarrow{g} (D, d)$ where

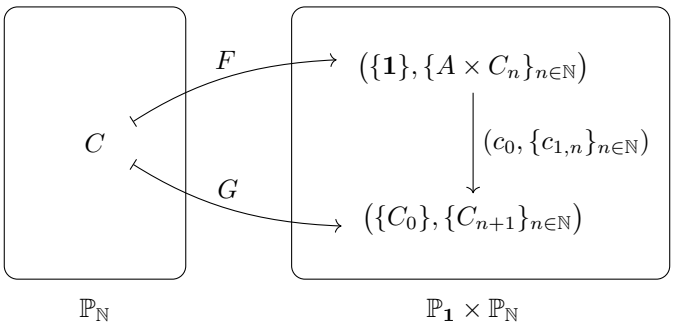
$$\begin{array}{ccc} C & & FC \xrightarrow{\;c\;} GC \\ \downarrow{\scriptstyle g} & \text{and} & \downarrow{\scriptstyle Fg} \qquad\qquad \downarrow{\scriptstyle Gg} \\ D & & FD \xrightarrow{\;d\;} GD \end{array}$$

Denote the category of (F, G)-dialgebras and -morphisms as $\mathbf{DiAlg}(F, G)$.

Note that F-algebras are (isomorphically) instances (F, G)-dialgebras for which $F, G : \mathbf{C^1} \to \mathbf{C^1}$ and with $G = \mathsf{Id}$. //

Now, dependent inductive types, if they exist, arise as initial (F, G)-dialgebras for certain F and G (as we will see).

Example 2.4.5. Assume some object A, and work in $\mathsf{Fam}(\mathbf{Set})$. Then define $G = (z^*, s^*) : \mathbb{P}_\mathbb{N} \to \mathbb{P}_\mathbf{1} \times \mathbb{P}_\mathbb{N}$ for the zero morphism $z : \mathbf{1} \to \mathbb{N}$ and the successor morphism $s : \mathbb{N} \to \mathbb{N}$. This means that $GX \triangleq (\{X_0\}, \{X_{n+1}\}_{n\in\mathbb{N}})$. Furthermore, let $FX \triangleq (\{\mathbf{1}\}, \{A \times X_n\}_{n\in\mathbb{N}})$. Any (F, G)-dialgebra (C, c) will be of the form:



Then, $(\mathsf{Vec}_A, (\mathsf{nil}, \mathsf{cons}))$ is the initial (F, G)-dialgebra, where

$$\begin{array}{ll} \mathsf{Vec}_A \in \mathbb{P}_\mathbb{N} & \mathsf{Vec}_A \triangleq \{A^n\}_{n\in\mathbb{N}} \\ \mathsf{nil} : \{\mathbf{1}\} \to \{A^0\} & \mathsf{nil}_\star([m]_\mathbf{1}) \triangleq \bot \\ \mathsf{cons} : \{A \times A^n\}_{n\in\mathbb{N}} \to \{A^{n+1}\}_{n\in\mathbb{N}} & \mathsf{cons}_n(a, v) \triangleq v[n+1 := a] \end{array}$$ ◿

Example 2.4.6. Assume some object A, and work in $\mathsf{Fam}(\mathbf{C})$. Consider the diagonal map $\delta = \mathsf{id} \times \mathsf{id} : A \to A \times A$. This induces the reindexing functor, called the *contraction functor*, $\delta^* : \mathbf{C}^{A \times A} \to \mathbf{C}^A$, which maps $\delta^*(C) = \{C_{(a,a)}\}_{a \in A}$. Now suppose the contraction functor has a left adjoint $\mathsf{Diag} \dashv \delta^*$, which means we have the natural hom-set isomorphism:

$$\frac{\mathsf{Diag}\big(\{C_a\}_{a\in A}\big) \to \{D_{(a,b)}\}_{(a,b)\in A\times A} \quad \text{in } \mathbf{C}^{A\times A}}{\{C_a\}_{a\in A} \to \delta^*(D) = \{D_{(a,a)}\}_{a\in A} \quad \text{in } \mathbf{C}^A}$$

Then it must be the case that

$$\mathsf{Diag}(C)_{(a,b)} = \begin{cases} C_a & \text{if } a = b \\ \mathbf{0} & \text{else} \end{cases}$$

Consider any $C = \{C_a\}_{a\in A}$, let $G = \delta^*$, and let $F = K_C$, the constant functor sending all objects to C and all morphisms to id_C. Then, pretty much trivially, $(\mathsf{Diag}(C), \mathsf{id})$ is the initial (F, G)-dialgebra. This can be easily seen, because the UP stipulates that for every (D, d), there must be a unique $u : \mathsf{Diag}(C) \to D$ such that the square below commutes. There is obviously only one choice.

$$\begin{array}{ccc} C & \xrightarrow{\mathsf{id}_C} & C \\ \downarrow{\scriptstyle \mathsf{id}_C} & & \downarrow{\scriptstyle d} \\ C & \xrightarrow{\{u_a\}_{a\in A}} & \{D_{(a,a)}\}_{a\in A} \end{array}$$

If we set $C = \{\mathbf{1}\}_{a\in A}$, we get the inductive equality type over A, as the initial (K_C, δ^*)-dialgebra. ◿

The following theorem gives us a reduction from (F, G)-dialgebras to F-algebras within a fiber:

Theorem 2.4.7 (Basold [2015])**.** *(Noting that* $\mathsf{Fam}(\mathbf{C})$ *has fibrewise coproducts:) If we have* $F, G_u : \mathbf{C}^I \to \mathbf{C}^{J_1} \times \cdots \times \mathbf{C}^{J_n}$ *where* $G_u = u^*$ *for* $u = (u_1, \dots u_n)$ *with each* $u_i : J_i \to I$*, then we have*

$$\mathbf{DiAlg}\big(F, G_u\big) \cong \mathbf{Alg}\big((\Sigma_{u_1} \circ F_1) +_I \cdots +_I (\Sigma_{u_n} \circ F_n)\big)$$

Proof. We can simply compute:

$$\begin{array}{rll} & \dfrac{FX \to G_u X}{} & \text{in } \mathbf{C}^{J_1} \times \cdots \times \mathbf{C}^{J_n} \\ (=) & \dfrac{(F_1X, \dots F_nX) \to (u_1^*(X), \dots u_n^*(X))}{} & \text{in } \mathbf{C}^{J_1} \times \cdots \times \mathbf{C}^{J_n} \\ (\text{over each } \Sigma_{u_i} \dashv u_i^*) & \dfrac{(\Sigma_{u_1}(F_1X), \dots \Sigma_{u_n}(F_nX)) \to (X, \dots X)}{} & \text{in } \big(\mathbf{C}^I\big)^n \\ (\text{over } \amalg_{k=1}^n \dashv \Delta_n) & \dfrac{\amalg_{k=1}^n \Sigma_{u_k}(F_kX) \to X}{} & \text{in } \mathbf{C}^I \end{array}$$

Preservation of the respective (di)algebra morphisms follows directly from the naturality of the hom-set isomorphisms above, which can be easily picked up from an illustration:

$$\begin{array}{ccc} C & FC \xrightarrow{c} G_uC & \\ \downarrow{\scriptstyle g} & \downarrow{\scriptstyle Fg} \quad\quad \downarrow{\scriptstyle G_ug} & \Leftrightarrow \\ D & FD \xrightarrow{d} G_uD & \end{array} \qquad \begin{array}{ccc} \amalg_{k=1}^n \Sigma_{u_k}(F_kC) & \xrightarrow{c'} & C \\ \downarrow{\scriptstyle \amalg_{k=1}^n \Sigma_{u_k}(F_kg)} & & \downarrow{\scriptstyle g} \\ \amalg_{k=1}^n \Sigma_{u_k}(F_kD) & \xrightarrow{d'} & D \end{array} \qquad \square$$

Polynomial functors

What then do we mean by polynomial functors, in the above? A number of notions can be used, but a typical computer science approach is the following:

Definition 2.4.8. A *non-dependent polynomial* (endofunctor) P has the shape

$$P(X) = \Sigma_{a \in A} X^{Y_a}$$

for a certain set A and family of objects $\{Y_a\}_{a \in A} \in \mathsf{Fam}(\mathbf{C})$. Noting that dependent types such as Y can be formulated as *display maps* as well, and using the assumption that $\mathbf{C}$ is sufficiently similar to **Set**, we would write this as

$$P_f(X) = \Sigma_{a \in A} X^{f^{-1}(a)}$$

for certain $f : B \to A$. This latter presentation also corresponds to the notion of *containers* $A \triangleright f$ of Abbott et al. [2005]. //

Initial algebras for non-dependent polynomials are also known as W-types (in which the 'W' stands for 'well-founded trees') and the following definition will be useful.

Definition 2.4.9 (Abbott et al. [2005])**.** A *Martin-Löf category* is a locally cartesian closed category with disjoint coproducts and closed under the formation of W-types. //

Example 2.4.10. Following up on example 2.4.3, we get the natural numbers functor F_N when we set $f = \mathsf{inr} : \mathbf{1} \to \mathbf{1} + \mathbf{1}$:

$$P_f(X) = \Sigma_{x \in \mathbf{1}+\mathbf{1}} X^{f^{-1}(x)} = X^{\mathbf{0}} + X^{\mathbf{1}} \cong \mathbf{1} + X$$

And we can add parameters as well, and get the functor $F_{\mathrm{List}(A)}$ for the type of lists over A when we set $f = \mathsf{inr} : A \to \mathbf{1} + A$:

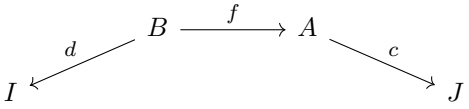
$$P_f(X) = \Sigma_{x \in \mathbf{1}+A} X^{f^{-1}(x)} \cong X^{\mathbf{0}} + \Sigma_{a \in A} X^{\mathbf{1}} \cong \mathbf{1} + A \times X$$

We can also get the 'countable ordinals' for the $f : \mathbf{1} + \mathbb{N} \to \{0, 1, 2\}$ with $f(\star) = 1$ and for all $n \in \mathbb{N}$, $f(n) = 2$:

$$P_f(X) = \Sigma_{i \in \{0,1,2\}} X^{f^{-1}(i)} \cong X^{\mathbf{0}} + X^{\mathbf{1}} + X^{\mathbb{N}} \cong \mathbf{1} + X + X^{\mathbb{N}}$$ ◿

A generalization of this can be given as (Basold [2015]):

Definition 2.4.11. A *dependent polynomial diagram* is a diagram of the shape



A *dependent polynomial* is any functor that is isomorphic to the composition

$$P \triangleq \mathbf{C}^I \xrightarrow{d^*} \mathbf{C}^B \xrightarrow{\Pi_f} \mathbf{C}^A \xrightarrow{\Sigma_c} \mathbf{C}^J$$ //

If $J = I = 1$, we have

$$\begin{aligned}
\Sigma_!\big(\Pi_f\big(!^*\big(\{X\}\big)\big)\big) &= \Sigma_!\big(\Pi_f\big(\{X\}_{b \in B}\big)\big) \\
&= \Sigma_!\big(\{\Pi_{b \in f^{-1}(a)} X\}_{a \in A}\big) \\
&= \Sigma_!\big(\{X^{f^{-1}(a)}\}_{a \in A}\big) \\
&= \big\{\Sigma_{a \in A} X^{f^{-1}(a)}\big\} \\
&= \big\{P_f(X)\big\}
\end{aligned}$$

in which we can see that the polynomial is *non-dependent*.

Although we will not prove the following theorem in generality, we do state it here, and will come back to it in chapter 6 for our specific instance of **Per**.

Theorem 2.4.12 (e.g. Gambino and Hyland [2004])**.** *Any Martin-Löf category, which in particular has initial algebras for non-dependent polynomials, has initial algebras for dependent polynomials as well.*

Proposition 2.4.13. *(Noting that* $\mathsf{Fam}(\mathbf{C})$ *has fibre-wise coproducts:) Dependent polynomials are closed under coproducts, composition, and finite products.*

Proof. For any set K, let $\{P_k\}_{k\in K}$ be a family of dependent polynomials for diagrams $\left[\, I \xleftarrow{d_k} A_k \xrightarrow{f_k} B_k \xrightarrow{c_k} J \,\right]_{k\in K}$. We can then define the polynomial functor

$$\Sigma_k P_k \qquad \text{for the diagram} \qquad \begin{array}{ccccc} & & \Sigma_k B_k & \xrightarrow{\Sigma_k f_k} & \Sigma_k A_k \\ & {\scriptstyle [d]=[d_k]_k}\swarrow & & & \searrow{\scriptstyle [c]=[c_k]_k} \\ I & & & & J \end{array}$$

And we can compute that

$$\begin{aligned}
\Sigma_k\big(P_k X\big) &= \sum_{k\in K} \Big\{ \sum_{a\in c_k^{-1}(j)} \prod_{b\in f_k^{-1}(a)} X_{d_k(b)} \Big\}_j \\
&= \Big\{ \sum_{k\in K} \sum_{a\in c_k^{-1}(j)} \prod_{b\in f_k^{-1}(a)} X_{d_k(b)} \Big\}_j && (\Sigma_k \text{ acts fibre-wise}) \\
&\cong \Big\{ \sum_{(k,a)\in[c]^{-1}(j)} \prod_{(-,b)\in(\Sigma f)^{-1}(k,a)} X_{d_k(b)} \Big\}_j \\
&= \big(\Sigma_k P_k\big) X
\end{aligned}$$

For the isomorphism, note that the lifted Σf does not 'reindex' over K, and hence the k involved is kept constant throughout the 'pipeline'.

For composition and finite products, it gets quite messy, and we refer the interested reader to Gambino and Kock [2013]. □

Example 2.4.14. Consider the dependent polynomial with diagram $\mathbb{N} \xleftarrow{!} \mathbf{0} \xrightarrow{!} \mathbf{1} \xrightarrow{z} \mathbb{N}$ (note how this relates to F_z from example 2.4.5):

$$\begin{aligned}
P_z &\triangleq \mathbf{C}^{\mathbb{N}} \xrightarrow{!^*} \mathbf{C}^{\mathbf{0}} \xrightarrow{\Pi_!} \mathbf{C}^{\mathbf{1}} \xrightarrow{\Sigma_s} \mathbf{C}^{\mathbb{N}} \\
&= (!^* \,;\, \Pi_!) \,;\, \Sigma_z \\
&= \big(\{X_n\}_{n\in\mathbb{N}} \mapsto \{\mathbf{1}\}\big) \,;\, \Big(\{X_\star\} \mapsto \begin{Bmatrix} X_\star & n=0 \\ \mathbf{0} & n\geq 1 \end{Bmatrix}_{n\in\mathbb{N}}\Big) \\
&= \{X_n\}_{n\in\mathbb{N}} \mapsto \begin{Bmatrix} \mathbf{1} & n=0 \\ \mathbf{0} & n\geq 1 \end{Bmatrix}_{n\in\mathbb{N}} \\
&= F_z \,;\, \Sigma_z
\end{aligned}$$

Similarly, consider the dependent polynomial for $\mathbb{N} \xleftarrow{\pi_1} A\times\mathbb{N} \xrightarrow{\mathsf{id}} A\times\mathbb{N} \xrightarrow{s\circ\pi_1} \mathbb{N}$ (and

again, note how this relates to F_s from example 2.4.5):

$$\begin{aligned}
P_s &\triangleq \mathbf{C}^{\mathbb{N}} \xrightarrow{\pi_1} \mathbf{C}^{A\times\mathbb{N}} \xrightarrow{\mathsf{id}} \mathbf{C}^{A\times\mathbb{N}} \xrightarrow{\Sigma_{(s\circ\pi_1)}} \mathbf{C}^{\mathbb{N}} \\
&= \mathbf{C}^{\mathbb{N}} \xrightarrow{\pi_1} \mathbf{C}^{A\times\mathbb{N}} \xrightarrow{\mathsf{id}} \mathbf{C}^{A\times\mathbb{N}} \xrightarrow{\Sigma_{\pi_1}} \mathbf{C}^{\mathbb{N}} \xrightarrow{\Sigma_s} \mathbf{C}^{\mathbb{N}} \\
&= (\pi_1^* \,;\, \Pi_{\mathsf{id}} \,;\, \Sigma_{\pi_1}) \,;\, \Sigma_s \\
&\cong \big(\{X_n\}_n \mapsto \{\Sigma_{a\in A} X_n\}_n\big) \,;\, \Sigma_s \\
&= \big(\{X_n\}_n \mapsto \{A \times X_n\}_n\big) \,;\, \Big(\{X_n\}_{n\in\mathbb{N}} \mapsto \begin{Bmatrix} \mathbf{0} & n=0 \\ X_{n-1} & n\geq 1 \end{Bmatrix}_{n\in\mathbb{N}}\Big) \\
&= \{X_n\}_{n\in\mathbb{N}} \mapsto \begin{Bmatrix} \mathbf{0} & n=0 \\ A\times X_{n-1} & n\geq 1 \end{Bmatrix}_{n\in\mathbb{N}} \\
&= F_s \,;\, \Sigma_s
\end{aligned}$$

If we combine these two polynomials over proposition 2.4.13,

$$\begin{aligned}
&F : \mathbf{C}^{\mathbb{N}} \to \mathbf{C}^{\mathbb{N}} \\
&F \triangleq P_z +_{\mathbb{N}} P_s \\
&\quad\cong \big(F_z; \Sigma_z\big) +_{\mathbb{N}} \big(F_s; \Sigma_s\big)
\end{aligned}$$

which then is a polynomial for the diagram

$$\begin{array}{ccccc}
 & & \mathbf{0} + (A\times\mathbb{N}) & \xrightarrow{!+\mathsf{id}} & \mathbf{1} + (A\times\mathbb{N}) \\
 & \overset{[!,\pi_1]}{\swarrow} & & & \overset{[z,\pi_1;s]}{\searrow} \\
\mathbb{N} & & & & \mathbb{N}
\end{array}$$

we can see that they describe, over theorem 2.4.7, the (F, G_u)-dialgebras for which the initial one was the inductive type of vectors. Explicitly, an initial algebra for this F would amount to some pair (W, w) with

$$\begin{Bmatrix} \mathbf{1} & n=0 \\ A\times W_{n-1} & n\geq 1 \end{Bmatrix}_{n\in\mathbb{N}} \xrightarrow{\{w_n\}_{n\in\mathbb{N}}} \{W_n\}_{n\in\mathbb{N}}$$

and obeying the initiality requirement. ◿

The question then remains, which pairs (F, G) correspond to dependent polynomials. Basold [2015] shows that Martin-Löf categories admit initial (F, G)-dialgebras for those (F, G_u) that are *strictly positive (data type) signatures*, which includes initial-algebra modeling $F = \mu(\hat{F}, \widehat{G_u})$. However, in our more specific setting, we can short-cut this machinery, noting that Martin-Löf categories already admit initial algebras for dependent polynomials, and noting that the fibration $\mathsf{Fam}(\mathbf{C}) \xrightarrow{\pi_0} \mathbf{C}$ has *comprehension*:

Proposition 2.4.15. *The fibration* $\mathsf{Fam}(\mathbf{C}) \xrightarrow{\pi_0} \mathbf{C}$ *is said to be a* comprehension category with unit *(CCU), in the terminology of Jacobs [1999], because each fiber has a final object, these are preserved by reindexing, and there is a* comprehension *functor* $\{-\} : \mathsf{Fam}(\boldsymbol{C}) \to \boldsymbol{C}$ *that is right adjoint to the final object functor* $\mathbf{1}_{(-)} : \boldsymbol{C} \to \mathsf{Fam}(\boldsymbol{C})$*:*

$$\mathbf{1}_{(-)} \dashv \{-\}$$

Proof. Because $\mathbf{C}$ is a Martin-Löf category, each fiber has a final object, and we can simply give comprehension in terms of the dependent sum: $\{A\} \triangleq \Sigma_{i\in I} A_i$. This gives rise to a functor $\mathscr{P} : \mathsf{Fam}(\mathbf{C}) \to \mathbf{C}^{\to}$, which maps any family A to the projection $\pi_A = \pi_0 = \pi_0(\varepsilon_A) : \Sigma_{i\in I} A_i \to I$. □

Essentially, comprehension gives us a way to 'encode families of objects of the total category as objects in the base category; the arrow category machinery is necessary to make sure that the notion of morphism is left intact, because of the correspondence

$$\begin{array}{ll} \{A_i\}_{i\in I} \to \{B_j\}_{j\in J} & \text{in } \mathsf{Fam}(\mathbf{C}) \\ \hline\hline \Sigma_{i\in I} A_i \longrightarrow \Sigma_{j\in J} B_j & \text{in } \mathbf{C}^{\to} \\ \downarrow{\scriptstyle \pi_0} \qquad\qquad \downarrow{\scriptstyle \pi_0} & \\ I \longrightarrow J & \end{array}$$

Now, we can state:

Definition 2.4.16. The pair (F, G) is *simple* if $F, G_u : \mathbf{C}^I \to \mathbf{C}^{J_1} \times \cdots \times \mathbf{C}^{J_n}$ where $G_u = u^*$ for $u = (u_1, \ldots u_n)$ with each $u_i : J_i \to I$, and F simple can be derived from the following deduction rules:

$$\frac{A \in \mathbf{C}^J}{K_A \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^J}} \qquad \frac{u : J \to I}{u^* \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^J}} \qquad \frac{F_1 \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^J} \qquad F_2 \ \mathsf{simple}_{\mathbf{C}^J \to \mathbf{C}^K}}{F_1; F_2 \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^K}}$$

$$\frac{F_1 \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^J} \qquad F_2 \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^K}}{\langle F_1, F_2 \rangle \ \mathsf{simple}_{\mathbf{C}^I \to \mathbf{C}^J \times \mathbf{C}^K}} \qquad \frac{}{\pi_i \ \mathsf{simple}_{\mathbf{C}^{J_1} \times \mathbf{C}^{J_2} \to \mathbf{C}^{J_i}}} \qquad /\!/$$

Proposition 2.4.17. *For simple (F, G), we can reduce (F, G)-dialgebras to dependent polynomial functors.*

Proof. First, use theorem 2.4.7 to reduce (F, G)-dialgebras to H-algebras, and then, by induction over the derivation that F is simple, noting that H indeed has the shape of the dependent polynomials of proposition 2.4.13:

$$\begin{aligned} HX &= \coprod_{k=1}^{n} \sum_{u_k} F_k(X) && \text{(theorem 2.4.7)} \\ &\cong \sum_{[u]} \langle F_1, \ldots F_n \rangle X && \text{(using } \mathbf{C}^{I+J} \simeq \mathbf{C}^I \times \mathbf{C}^J\text{)} \\ &= \Big\{ \sum_{(k,j) \in [u]^{-1}(i)} \big(\langle F_1, \ldots F_n \rangle X\big)_j \Big\}_i && \text{(fibrewise } \Sigma\text{)} \end{aligned}$$

- If $F = h^*$ for some $h : J \to I$, then we have $HX \cong \Sigma_u h^*(X) = \Sigma_u\big(\{X_{h(j)}\}_j\big) = \big\{\Sigma_{j \in u^{-1}(i)} X_{h(j)}\big\}_i \cong P$ for the diagram $\Big[\, I \xleftarrow{h} J \xrightarrow{\mathsf{id}} J \xrightarrow{u} I \,\Big]$.
- If $F = K_A$ for some $A \in \mathbf{C}^J$, then we have $HX \cong \Sigma_u A = \big\{\Sigma_{j \in u^{-1}(i)} A_j\big\}_i \cong \big\{\Sigma_{j \in u^{-1}(i)} \Sigma_{(j,a) \in (\pi_0)^{-1}(j)} \mathbf{1}\big\}_i \cong \big\{\Sigma_{(j,a) \in (\pi_0; u)^{-1}(i)} \Pi_{- \in !^{-1}} X_-\big\}_i = P$ for the diagram $\Big[I \xleftarrow{!} \mathbf{0} \xrightarrow{!} \{A\} \xrightarrow{\pi_0; u} I \Big]$.
- Assuming $F_1 : \mathbf{C}^I \to \mathbf{C}^J$ and $F_2 : \mathbf{C}^J \to \mathbf{C}^K$ are such that for any $u_1 : J \to I$ resp. $u_2 : K \to J$, we have (F_1, u_1) and (F_2, u_2) represented by polynomials, we can apply proposition 2.4.13 to get a dependent polynomial for $F_1; F_2$ (which will be quite complicated).
- Similarly for $\langle F_1, F_2 \rangle$.
- The most interesting (and hard) case is when $F = \pi_i$, and it corresponds to the situation in which the resulting inductive types' constructors can reference one another (recursively). We refer the reader to Basold [2015, thm 4.6] for a proof, and won't further pursue constructor cross-referencing in this thesis. □

3 Introducing realizability semantics

In this chapter, we will give an introductory realizability model for the simply typed lambda calculus ($\lambda\!\to$), which only requires very little structure of the D-sets used to build them. (We will postpone describing additional structure for the time being.) After giving the construction, we'll briefly discuss some key characteristics of our model that will be relevant for later sections.

3.1 $\mathcal{D}$-sets

The idea behind the following definitions, is that if a PCA $\mathcal{D}$ is thought of as a model of computation, in the sense that its elements denote data and computable functions, then a $\mathcal{D}$-set is just a normal set, but supplied with a notion of *realizability* for its elements of $\mathcal{D}$ for each of its elements, which can be thought of as 'machine implementations', or 'codes'.

Definition 3.1.1. A *$\mathcal{D}$-set*, or *assembly* $(A, \Vdash_A)$ consists of a set A together with a *realizability relation* $(\Vdash_A) \subseteq \mathcal{D} \times A$, where $n \Vdash_A a$ is read as 'n realizes a', and such that $\forall_{a\in A}\exists_{n\in\mathcal{D}}[\, n \Vdash_A a\,]$. We will just write $(\Vdash)$ instead of $(\Vdash_A)$ if A is clear from the context. When necessary, we write $|A|$ to explicitly refer to the carrier set instead of the $\mathcal{D}$-set itself.

A $\mathcal{D}$-set morphism between $(A, \Vdash_A) \xrightarrow{f} (B, \Vdash_B)$ is a function $f : A \to B$ such that f is *realized*, or *tracked*, by some $e \in \mathcal{D}$:

$$\forall_{a\in A}\forall_{n\in\mathcal{D}}\big[\, n \Vdash_A a \Rightarrow (e \cdot n) \Vdash_B f(a)\,\big]$$

We denote the category of $\mathcal{D}$-sets and $\mathcal{D}$-set morphisms by **$\mathcal{D}$-Set**. //

The notion that a morphism must be tracked ensures that the category of syntactical constructions to be built is one with a computable notion of morphisms, so that its natural exponential objects (as internal homs) can serve to interpret function types.

The assumption that every $a \in A$ is realized may informally be taken to express the idea that all syntax will be interpretable, and plays a quite subtle role in extensionality details. Note also that even if we had allowed partial functions $f : |A| \rightharpoonup |B|$ to be eligible for tracking, none would be tracked, i.e. $\mathcal{D}$-set morphisms are always total.

Lemma 3.1.2. *Every $\mathcal{D}$-set morphism $A \xrightarrow{f} B$ is total, i.e. for all $a \in A$, $f(a) = b$ for some $b \in B$.*

Proof. Due to the requirement that $\forall_{a\in A}\exists_{n\in\mathcal{D}}[\, n \Vdash_A a\,]$. Let $a \in A$ and e track f. We can take any such $n \Vdash_A a$, and then use the definition of tracking to find some $(e \cdot n) \Vdash_B f(a)$. Hence, $f(a)\!\downarrow$. □

Definition 3.1.3. The *realizability predicate* $E_A : |A| \to \mathcal{P}(\mathcal{D})$ of a $\mathcal{D}$-set A, sometimes also called the *existence predicate* in the literature, is defined as

$$E_A(a) \triangleq \{n \in \mathcal{D} \mid n \Vdash_A a\}$$

It is not hard to see that one could equivalently state the definition of $\mathcal{D}$-sets in terms of such realizability predicates, and then take the realizability relation as a derived concept. Hence, we will allow ourselves to define / use $\mathcal{D}$-set in terms of their realizability predicates as well as in terms of their realizability relations. //

Remark 3.1.4 (Functions and their realizers relate many-to-many)**.** *Note that any total map $e \in \mathcal{D}$ realizes $A \xrightarrow{!} \top$, which tells us that $\mathcal{D}$-set morphisms may be multiply realized. Also, an $e \in \mathcal{D}$ may realize distinct morphisms $A \xrightarrow{f \neq g} B$, because we can have $\{e \cdot n \mid n \Vdash_A a\} \subseteq E_B(f(a)) \cap E_B(g(a))$ in those cases where $f(a) \neq g(a)$, i.e., if B's realizability predicate is 'underspecified'. The simplest example is of course when $B = \nabla|B|$ is fully realized.*

Remark 3.1.5. *If we take $\mathcal{D} = \mathcal{K}_1$, we obtain the so-called ω-sets as discussed before, initially investigated by Kleene.*

Definition 3.1.6. For any set X, the *full realizability structure* is the $\mathcal{D}$-set $\nabla X \triangleq (X, - \mapsto \mathcal{D})$. Note that any function f from a $\mathcal{D}$-set A to X is automatically a $\mathcal{D}$-set morphism $f : X \to \nabla X$ as well, because it is vacuously tracked by any $d \in \mathcal{D}$. //

Proposition 3.1.7. *This definition automatically induces a functor $\nabla : \mathbf{Set} \to \mathcal{D}\textbf{-Set}$. Moreover, there is a trivial forgetful functor $\mathsf{Forget} : \mathcal{D}\textbf{-Set} \to \mathbf{Set}$, and we have an adjoint situation $\mathsf{Forget} \dashv \nabla$ and also $\mathsf{Forget} \circ \nabla = \mathsf{id}$.*

Proof. Define $\nabla f = f$, which is vacuously tracked by definition of ∇, and define $\mathsf{Forget}(A) = |A|$ and $\mathsf{Forget}(f) = f$. The required functoriality equalities, as well as $\mathsf{Forget} \circ \nabla = \mathsf{id}$, of course follow automatically. For the adjoint we need the natural isomorphism:

$$\frac{\mathsf{Forget}(A) \xrightarrow{f} Y}{A \xrightarrow{g} \nabla Y} \qquad \begin{array}{l} \text{in } \mathbf{Set} \\ \text{in } \mathcal{D}\textbf{-Set} \end{array}$$

($\Downarrow$) Because any such g is vacuously tracked, we can take $g := f$. ($\Uparrow$) Here we lose the tracking requirement, and hence we can take $f := g$. □

Proposition 3.1.8. *$\mathcal{D}$-Set is a cartesian closed category, and has an initial object.*

Proof. It is easy to see that we have an initial and final $\mathcal{D}$-set with

$$\begin{array}{ll} \mathbf{0} \triangleq \varnothing & \Vdash_{\mathbf{0}} \triangleq \varnothing \\ \mathbf{1} \triangleq \{\star\} & \Vdash_{\mathbf{1}} \triangleq \{\star\} \times \mathcal{D} \end{array}$$

Furthermore, $\mathcal{D}$-sets admit straightforward set-theoretical products $|A \times B| = |A| \times |B|$ and exponentials $|B^A| = \{f : |A| \to |B| \mid f \text{ tracked}\}$ with:

$$\begin{array}{rcl} \langle n, m \rangle \Vdash_{A \times B} (a, b) & \text{iff} & n \Vdash_A a \wedge m \Vdash_B b \\ e \Vdash_{B^A} f & \text{iff} & e \text{ tracks } f \end{array}$$

That these object indeed satisfy the required universal properties is not too hard to see. Essentially, we're still just working with sets, and functions like the ones we need here are easily tracked because they act (PCA-)algebraically on their inputs.

The evaluation morphism $B^A \times A \xrightarrow{\mathsf{eval}} B$ is just $(f, a) \mapsto f(a)$, and tracked by $(\lambda\!\!\lambda d.\, d_0 \cdot d_1) \in \mathcal{D}$, for if $(f, a) \in B^A \times A$ and $n \Vdash_{(B^A \times A)} (f, a)$ (i.e. $n_0 \Vdash_{B^A} f$ and $n_1 \Vdash_A a$), then indeed $(\lambda\!\!\lambda d.\, d_0 \cdot d_1) \cdot n = n_0 \cdot n_1 \Vdash_B f(a)$, because by assumption n_0 tracks f and $n_1 \Vdash_A a$. Similarly, the currying morphism $C^{A \times B} \xrightarrow{\mathsf{curry}} (C^B)^A$ is just $f \mapsto (a \mapsto (b \mapsto f(a, b)))$ and tracked by $(\lambda\!\!\lambda d.\lambda\!\!\lambda n.\lambda\!\!\lambda m.\; d \cdot \langle n, m \rangle) \in \mathcal{D}$, the pairing morphism $A \xrightarrow{\mathsf{pair}} (A \times B)^B$ is just $a \mapsto (b \mapsto (a, b))$ and tracked by $(\lambda\!\!\lambda d.\lambda\!\!\lambda e.\langle d, e \rangle) \in \mathcal{D}$, and projection morphisms $(A \times B) \xrightarrow{\pi_0} A$ and $(A \times B) \xrightarrow{\pi_1} \to B$ are respectively realized by $(\lambda\!\!\lambda d.d_0)$, $(\lambda\!\!\lambda d.d_1) \in \mathcal{D}$. □

Proposition 3.1.9. *If B is fully realized, then any $B^A = \big(|A| \to |B|\big)$ is fully realized too.*

Proof. Because the tracking requirement is vacuously satisfied. □

3.2 A note on data types and uniform realizability

As noted in the introduction and section 2.1, the notion of *realizability* originally had a mainly logical motivation. However, the previous section introduced $\mathcal{D}$-sets from a simple computer science theoretical perspective / level of abstraction. Not necessary for subsequent theory development, this section attempts to illustrate how the definition of $\mathcal{D}$-sets was arrived at (historically) through logical considerations, and shed some informal light on the 'shape' of the definition of $\mathcal{D}$-sets. This perspective may also allow for some intuition on the automatic extensional flavor of these realizability models, and how *uniform realizability* is a key characteristic underpinning the *data types* interpretation.

The starting point for realizability was Kleene's definition of the form

$$e \text{ realizes } \phi$$

Hyland [1982, §1] mentions that Dana Scott was the first to view realizability 'model-theoretically', in terms of $\{e \mid e \text{ realizes } \phi\}$ being the *non-standard truth value* of ϕ. The set of non-standard truth values is then $\mathcal{P}(\mathcal{D})$, on which the semantics of the logical connectives can already be defined:

Definition 3.2.1 (Semantics of the connectives on non-standard truth values)**.** Let $X, Y \in \mathcal{P}(\mathcal{D})$ be non-standard truth-values.

$$\begin{aligned} X \wedge Y &\triangleq \{\langle n, m\rangle \mid n \in X \wedge m \in Y\} \\ X \vee Y &\triangleq \{\langle k, n\rangle \mid n \in X\} \cup \{\langle k^*, m\rangle \mid m \in Y\} \\ X \Rightarrow Y &\triangleq \{e \mid \forall_{n \in \mathcal{D}}[\, n \in X \Rightarrow (e \cdot n)\!\downarrow \wedge\, (e \cdot n) \in Y\,]\} \\ \bot &\triangleq \varnothing \\ \top &\triangleq \mathcal{D} \end{aligned}$$ //

For any set A, we can then defined the set of *non-standard predicates* (with a free variable ranging over A): $\phi, \psi \in \mathcal{P}(\mathcal{D})^A$. These are of course our earlier introduced realizability predicates $E_1, E_2 : A \to \mathcal{P}(\mathcal{D})$ included in the definition of $\mathcal{D}$-sets, over a fixed carrier set A. However, instead of leaping directly to morphisms between $\mathcal{D}$-sets with different carrier sets, the logical approach would prescribe building up the theory from this finer setting. First we can lift the semantics of the connectives to these predicates, component-wise, as follows (where we write $\phi = \{\phi_a \mid a \in A\}$):

Definition 3.2.2 (Semantics of the connectives on non-standard predicates)**.**

$$\begin{aligned} (\phi \wedge \psi)_a &\triangleq \phi_a \wedge \psi_a \\ (\phi \vee \psi)_a &\triangleq \phi_a \vee \psi_a \\ (\phi \Rightarrow \psi)_a &\triangleq \phi_a \Rightarrow \psi_a \\ \bot_a &\triangleq \bot \quad \text{(overloaded)} \\ \top_a &\triangleq \top \quad \text{(overloaded)} \end{aligned}$$ //

These definitions correspond to our own $\mathcal{D}$-set constructions:

$$\begin{aligned} (E_1 \wedge E_2) &= E_{(A, E_1) \times (A, E_2)} \\ (E_1 \Rightarrow E_2)_a &= \{e \mid e \text{ tracks } \mathsf{id} \text{ 'at' } a\} \end{aligned}$$

Subsequently, one can define *entailment* between non-standard predicates by the following:

Definition 3.2.3 (Non-standard entailment / uniform realization)**.**

$$\phi \vdash_A \psi \quad \text{iff} \quad \bigcap_{a \in A} (\phi \Rightarrow \psi)_a \neq \varnothing$$

Note that an element $e \in \bigcap(\phi \Rightarrow \psi)$ is a recursive function that, regardless of $a \in A$, always sends elements of ϕ_a to elements of ψ_a, and hence we say that *a uniformly realizes* $\phi \vdash_A \psi$. //

This definition corresponds to our notion of the identity function on A being a $\mathcal{D}$-set morphism (i.e. being tracked):

$$\phi \vdash_A \psi \quad \text{iff} \quad (A, \phi) \xrightarrow{\mathsf{id}} (A, \psi) \text{ exists}$$

If we now view this structure $(\mathcal{P}(\mathcal{D})^A, \vdash_A)$ as the category of non-standard predicates over A, with morphisms provided by the entailment relation (or, stated alternatively, the subcategory of $\mathcal{D}$-sets over a fixed carrier set A), then we can introduce quantifications via the standard method in categorical logic (due to Lawvere) by taking the quantifiers as the adjoints to *substitution*.

Proposition 3.2.4. *Any function $f : A \to B$ can then be lifted to a so-called* reindexing *functor $f^* : (\mathcal{P}(\mathcal{D})^B, \vdash_B) \to (\mathcal{P}(\mathcal{D})^A, \vdash_A)$, which 'substitutes along f'—defined as:*

$$(f^*(\psi))_a \triangleq \psi_{f(a)}$$

…which stated in $\mathcal{D}$-set terms corresponds to:

$$f^*(B, \psi) \triangleq (A, a \mapsto \psi_{f(a)})$$

Proof. We need to check two things: (1) that $f^*(\psi)$ is indeed a $\mathcal{D}$-set (i.e. every element is realized), and (2) that we can even send the identity morphism in the first place (i.e. the resulting identity morphism is also tracked). Item (1) follows directly from ψ being a $\mathcal{D}$-set. For (2), consider the entailment $\psi_1 \vdash_B \psi_2$, say uniformly tracked by some $e \in \mathcal{D}$. Then e automatically tracks the resulting entailment as well. □

Proposition 3.2.5. *The reindexing functor has left and right adjoints $\exists f \dashv f^* \dashv \forall f$.*

Proof. These can be given by:

$$(\exists f.\phi)_b \triangleq \bigcup_{a \in A} (f(a) \equiv b) \wedge \phi_a$$
$$(\forall f.\phi)_b \triangleq \bigcap_{a \in A} (f(a) \equiv b) \Rightarrow \phi_a$$

where we define

$$f(a) \equiv b \triangleq \begin{cases} \top & \text{if } f(a) = b \\ \bot & \text{else} \end{cases}$$

To prove that $\exists f$ is a functor, consider an entailment $\phi_1 \vdash_A \phi_2$ uniformly tracked by some $e \in \mathcal{D}$, and let $b \in B$ and $n \in (f(a) \equiv b) \wedge \phi_1(a)$ for some $a \in A$, i.e. $f(a) = b$ and $n \in \phi_1(a)$. Then $(e \cdot n) \in \phi_2(a) \subseteq (\exists f.\phi_2)_b$ as well, so we're OK.

To prove $\exists f \dashv f^*$, we need to have, for all ϕ over A and ψ over B, the bi-implication:

$$\frac{\exists f.\phi \vdash_B \psi}{\phi \vdash_A f^*(\psi)}$$

(⇓) Suppose e uniformly tracks the top entailment, i.e. for all $b \in B$ and $n \in \mathcal{D}$, if $n \in \phi_a$ for any a such that $f(a) = b$, then $(e \cdot n) \in \psi_b$. Now let $a \in A$ and $n \in \phi_a$. If we set $b = f(a)$, then by the previous we have $(e \cdot n) \in \psi_{f(a)} = f^*(\psi)_a$, and hence we see that a uniformly tracks the bottom entailment as well.

(⇑) Suppose e uniformly tracks the bottom entailment, i.e. for all $a \in A$ and $n \in \mathcal{D}$, if $n \in \phi_a$, then $(e \cdot n) \in \psi_{f(a)}$. Now let $b \in B$ and $n \in \phi_a$ for some a such that $f(a) = b$. But then $(e \cdot n) \in \psi_{f(a)} = \psi_b$, so e uniformly tracks the top entailment as well.

To prove that $\forall f$ is a functor, consider an entailment $\phi_1 \vdash_A \phi_2$ uniformly tracked by some $g \in \mathcal{D}$ and let $b \in B$ and $e \in (f(a) \equiv b) \Rightarrow \phi_1(a)$ for all $a \in A$, i.e. whenever $f(a) = b$, $(e \cdot n) \in \phi_1(a)$ for all $n \in \mathcal{D}$. Then, $g \circ e \in (f(a) \equiv b) \Rightarrow \phi_2(a)$ for all $a \in A$, because whenever $f(a) = b$ is the case, then for all $n \in \mathcal{D}$, for which we already know that $(e \cdot n) \in \phi_1(a)$, we also have $g \cdot (e \cdot n) \in \phi_2(a)$ by our original premiss. Hence, the recursive function $\Lambda e.g \circ e$ uniformly realizes $\forall f.\phi_1 \vdash_B \forall f.\phi_2$.

To prove $f^* \dashv \forall f$, we need to have, for all ϕ over A and ψ over B, the bi-implication:

$$\frac{\psi \vdash_B \forall f.\phi}{f^*(\psi) \vdash_A \phi}$$

(⇓) Suppose $g \in \mathcal{D}$ uniformly tracks the top entailment, i.e. for all $b \in B$ and $e \in \psi_b$, if no a exists with $f(a) = b$, then $((g \cdot e) \cdot n)\!\uparrow$ for all n, and else, for all a such that $f(a) = b$, $((g \cdot e) \cdot n) \in \phi_a$ for all n. Define $h \triangleq \Lambda e.((g \cdot e) \cdot k)$. Then h uniformly tracks the bottom entailment, for if $a \in A$, and $e \in \psi_{f(a)}$, then indeed $h \cdot e = (g \cdot e) \cdot k \in \phi_a$. [Note that k just an arbitrary choice here, anything will work.]

(⇑) Suppose $h \in \mathcal{D}$ uniformly tracks the bottom entailment, i.e. for all $a \in A$ and $e \in \psi_{f(a)}$, we have $h \cdot e \in \phi_a$. Define $g \triangleq \Lambda e.(k \cdot (h \cdot e))$, which will then uniformly track the top entailment. Suppose $b \in B$ and $e \in \psi_b$. We must prove that $g \cdot e \in \big((f(a) \equiv b) \Rightarrow \phi_a\big)$ for all a, so let $a \in A$. If $f(a) \neq b$, then this vacuously holds. If $f(a) = b$, we have $e \in \psi_b = \psi_{f(a)}$ and hence $g \cdot e = k \cdot (h \cdot e) \in \big((f(a) \equiv b) \Rightarrow \phi_a\big)$ because $k \cdot (h \cdot e) \cdot n = h \cdot e \in \phi_a$ for any n.

Note that whereas we can also define $\exists f$ by the simpler formula $(\exists f.\phi)_b \triangleq \bigcup \big\{\phi_a \mid f(a) = b\big\}$, we cannot do the same for $\forall f$. The problem is that if f is not surjective, i.e. when there is some b that is not in f's range, then it might be the case that for some $e \in \psi_b$ for which $e \notin f^*(\psi)_a$ for all a, the h that uniformly tracks $f^*(\psi) \vdash_A \phi$ has $(h \cdot e)\!\uparrow$. If this is the case, then h will not be able to uniformly track the top entailment, simply because of its undefinedness at e. To insulate for this situation, the definition includes the implication. □

Noteworthy is that if we follow the above-mentioned definitional correspondence between logical entailment $\phi \vdash_A \psi$ and the existence of the identity morphism $(A, \phi) \xrightarrow{\mathsf{id}} (A, \psi)$, we get the following characterization of our previous notion of a $\mathcal{D}$-set morphism:

Corollary 3.2.6. *The following are equivalent:*

- $(A, \phi) \xrightarrow{f} (B, \psi)$
- $\phi \vdash_A f^*(\psi)$
- $\exists f.\phi \vdash_B \psi$

◿

3.3 A note on extensionality

Note that from the start, the model includes a strong kind of extensionality, in the sense that we identity morphisms with their entire set of realizers, instead of, say, defining morphisms as pairs of functions and their respective realizers.

The following proposition shows how, regardless of whether the underlying PCA obeys the (η) rule, the definition of the tracking requirement automatically identifies behaviorally PCA elements under their tracked morphisms, and hence it could be said that the $\mathcal{D}$-set function space does indeed obey (η). This last statement will be made precise in subsection 3.4, after we have given the $\lambda\!\!\rightarrow$ model.

Proposition 3.3.1. *For any $\mathcal{D}$-set morphism $A \xrightarrow{f} B$,*

1. *e tracks f iff $(\curlywedge n.\, e \cdot n)$ tracks f;*
2. *d tracks f iff e tracks f — whenever $d \sim e$.*

Proof. 1) Let $e \in \mathcal{D}$, $a \in A$, and $n \Vdash_A a$. Obviously $e \cdot n = (\curlywedge n.\, e \cdot n) \cdot n$, and hence we have our bi-implication. 2) By the definition of tracking. □

3.4 Modeling $\lambda\!\!\rightarrow$

Now we will turn to giving a model of $\lambda\!\!\rightarrow$ in the category of $\mathcal{D}$-sets, as an initial sketch of the idea of giving a realizability semantics for type theory. For the simply typed lambda calculus, products and exponentials give us enough structure for such a model.

Definition 3.4.1 (Simply typed lambda calculus)**.** Recall that the simply typed lambda calculus has an enumerable supply of variables $x, y, \ldots$, terms $t, u ::= x \mid t^{A\to B} u^A \mid (\lambda x^A.t)$, types $A, B ::= A \to B \mid A \times B$, contexts $\Gamma ::= \cdot \mid \Gamma, x^A$, typing judgments $\Gamma \vdash t : A$, definitional equality judgments $\Gamma \vdash t \equiv u : A$, and the following inference rules. (For practical purposes, these data would be extended with assumed constant types and terms such as `int`, `bool`, etc.)

The inference rules are given in figure 4. //

Remark 3.4.2. *There are at least 3 different notions of 'equality' floating around at the moment. To clarify (our notation):*

- α-equivalence $(\equiv_\alpha)$*, where two lambda-terms are considered α-equivalent iff they are syntactically equal up to renaming of bound variables. Note that, in the definition above, such renaming is in fact assumed, in order for the substitution in the $(\equiv \beta)$ rule to work properly. The theory of names in terms is quite subtle, but largely irrelevant for our present purposes, and hence we will glaze over these details, and consider α-equivalent terms as if they were syntactically equal.*
- Definitional equality $(\equiv)$*, or* conversion*, which is a syntactic construct within the presented type theory. A definitional equality can only be said to hold in the shape of $\Gamma \vdash t \equiv u : A$, and proven to hold meta-theoretically, by means of constructing a valid derivation in terms of the presented inferences. The main driver, here, is of course the $(\equiv \beta)$ inference. One might also include the $(\equiv \eta)$ rule, and to distinguish the two resulting equivalences, one would speak of β-conversion $(\equiv_\beta)$ resp. $\beta\eta$-conversion $(\equiv_{\beta\eta})$.*
- Regular mathematical equality $(=)$ *which, between terms, would mean syntactic equality (and hence include α-equivalence per our convention). Definitional equality is of course stronger—it includes a computably enumerable set of non-trivial equations*

Typing inferences

$$(\mathsf{var})\ \frac{x^A \in \Gamma}{\Gamma \vdash x : A} \quad (\mathsf{app})\ \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t^{A\to B} u^A : B} \quad (\mathsf{abs})\ \frac{\Gamma, x^A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B}$$

Computation

$$(\equiv \beta)\ \frac{\Gamma, x^A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x^A.t)s \equiv t[x := s] : B}$$

Convertibility moves into application / abstraction

$$(\equiv \mathsf{app})\ \frac{\Gamma \vdash t \equiv u : A \to B \qquad \Gamma \vdash s \equiv r : A}{\Gamma \vdash ts \equiv ur : B} \quad (\equiv \mathsf{abs})\ \frac{\Gamma, x^A \vdash t \equiv u : B}{\Gamma \vdash \lambda x^A.t \equiv \lambda x^A.u : A \to B}$$

Convertibility is an equivalence relation

$$(\equiv \mathsf{refl})\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \equiv t : A} \quad (\equiv \mathsf{symm})\ \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \quad (\equiv \mathsf{trans})\ \frac{\Gamma \vdash t \equiv u : A \quad \Gamma \vdash u \equiv w : A}{\Gamma \vdash t \equiv w : A}$$

Figure 4: Inference rules of the simply typed lambda calculus.

between terms—which is one of the main interests of doing type theory in the first place.

Definition 3.4.3 ($\mathcal{D}$-set model of $\lambda\to$)**.** See figure 5. For the interpretation, the idea is that we interpret contexts and types as $\mathcal{D}$-sets, and terms as $\mathcal{D}$-set elements. //

Informally, terms are interpreted to realizable functions that first take their context as a list (realized by an iterated pairing $(\cdots((-,-),-),-,\cdots)$), and then return the computed combination. I.e. they are the classic verifying proof terms per the Curry-Howard correspondence.

Example 3.4.4. Suppose we take $t = \lambda x^A.\lambda f^{A\to B}.fx$. Let $\Gamma = x^A, f^{A\to B}$. We can derive the well-typedness of this term in the simply typed lambda calculus:

$$\dfrac{\dfrac{\dfrac{\dfrac{f^{A\to B} \in \Gamma}{\Gamma \vdash f : A \to B} \quad \dfrac{x^A \in \Gamma}{\Gamma \vdash x : A}}{\Gamma \vdash fx : A \to B}}{x^A \vdash \lambda f^{A\to B}.fx : A \to B}}{\vdash \lambda x^A.\lambda f^{A\to B}.fx : (A \to B) \to A \to B}$$

And the interpretation that is given to it in the $\mathcal{D}$-set model is:

$$\begin{aligned}
[\![\vdash \lambda x^A.\lambda f^{A\to B}.fx]\!] &= \star \mapsto a \mapsto [\![x^A \vdash \lambda f^{A\to B}.fx]\!](\star, a) \\
&= \star \mapsto a \mapsto \big(g \mapsto (b \mapsto [\![\Gamma \vdash fx]\!](g, b))\big)(\star, a) \\
&= \star \mapsto a \mapsto b \mapsto [\![\Gamma \vdash fx]\!]((\star, a), b) \\
&= \star \mapsto a \mapsto b \mapsto \Big(g \mapsto [\![\Gamma \vdash f]\!](g)\big([\![\Gamma \vdash x]\!](g)\big)\Big)((\star, a), b) \\
&= \star \mapsto a \mapsto b \mapsto \Big(g \mapsto \mathsf{proj}_2^3(g)\big(\mathsf{proj}_1^3(g)\big)\Big)((\star, a), b) \\
&= \star \mapsto a \mapsto b \mapsto b(a)
\end{aligned}$$

I.e. the term is interpreted to its untyped counterpart, but with the catch that the empty context is treated as a type as well, inhabitant of which has to be given to the proof term to start computing in the first place. ◿

Interpretation signature

$$[\![\Gamma]\!], [\![A]\!] \in \mathcal{D}\textbf{-set} \qquad\qquad [\![\Gamma \vdash s^A]\!] \in [\![A]\!]$$

Interpretation of contexts and types

$$[\![\,\cdot\,]\!] \triangleq \top \qquad\qquad [\![A \times B]\!] \triangleq [\![A]\!] \times [\![B]\!]$$
$$[\![\Gamma, x^A]\!] \triangleq [\![\Gamma]\!] \times [\![A]\!] \qquad\qquad [\![A \to B]\!] \triangleq [\![B]\!]^{[\![A]\!]}$$

Interpretation of terms

$$[\![x_1{}^{A_1}, \ldots x_i{}^{A_i}, \ldots x_n{}^{A_n} \vdash x_i]\!] \triangleq \mathsf{proj}_{i+1}^{n+1}$$
$$[\![\Gamma \vdash tu]\!] \triangleq (g \in [\![\Gamma]\!]) \;\mapsto\; [\![\Gamma \vdash t]\!](g)\big([\![\Gamma \vdash u]\!](g)\big)$$
$$[\![\Gamma \vdash \lambda x^A.t]\!] \triangleq \mathsf{curry}([\![\Gamma, x^A \vdash t]\!])$$

Figure 5: Interpretation of the simply typed lambda calculus.

Proposition 3.4.5 (Type soundness)**.** *Derivability of* $\Gamma \vdash s : A$ *implies* $[\![\Gamma \vdash s]\!] \in [\![A]\!]^{[\![\Gamma]\!]}$.

Proof. By induction on the structure of derivable type judgments.

Suppose $\Gamma \vdash x : A$. Then we must have $x^A \in \Gamma$, say at position i of n. Then indeed $[\![\Gamma \vdash x]\!] = \mathsf{proj}_{i+1}^{n+1} \in [\![A]\!]^{\cdots \times [\![A]\!] \times \cdots} = [\![A]\!]^{[\![\Gamma]\!]}$.

Suppose $\Gamma \vdash t\,s : B$. We may assume, for certain A, that $\Gamma \vdash t \in [\![A \to B]\!]^{[\![\Gamma]\!]}$ and $[\![\Gamma \vdash s]\!] \in [\![A]\!]^{[\![\Gamma]\!]}$. Then, for any $g \in [\![\Gamma]\!]$, we have $[\![\Gamma \vdash t]\!](g) \in [\![A \to B]\!]$, $[\![\Gamma \vdash s]\!](g) \in [\![A]\!]$, and $[\![\Gamma \vdash ts]\!](g) \in [\![B]\!]$.

Similar for abstraction. □

Proposition 3.4.6 (Equational soundness)**.** *Derivability of* $\Gamma \vdash s \equiv t : A$ *implies* $[\![\Gamma \vdash s]\!] = [\![\Gamma \vdash t]\!]$.

Proof. The only rule that requires non-trivial checking is of course the β-rule. Suppose that $\Gamma \vdash (\lambda x^A.t)s \equiv t[x := s] : B$ has been derived from $\Gamma, x^A \vdash t : B$ and $\Gamma \vdash s : A$. We may assume that $[\![\Gamma, s^A \vdash t]\!] \in [\![B]\!]^{[\![\Gamma, x^A]\!]} = [\![B]\!]^{[\![\Gamma]\!] \times [\![A]\!]}$ and $[\![\Gamma \vdash s]\!] \in [\![A]\!]^{[\![\Gamma]\!]}$.

Proceed by induction on the structure of t. Suppose $t = y \neq x$. Then $t[x := s] = y$, and $[\![\Gamma, x^A \vdash y]\!] = (g, a) \mapsto [\![\Gamma \vdash y]\!](g)$ by construction, and

$$\begin{aligned}
[\![\Gamma \vdash (\lambda x^A.y)s]\!] &= g \mapsto [\![\Gamma \vdash \lambda x^A.y]\!](g)\big([\![\Gamma \vdash s]\!](g)\big) \\
&= g \mapsto [\![\Gamma, x^A \vdash y]\!]\big((g, [\![\Gamma \vdash s]\!](g))\big) && \text{by def } [\![\;]\!] \text{ and } \mathsf{curry} \\
&= g \mapsto [\![\Gamma \vdash y]\!]
\end{aligned}$$

Suppose that $t = x$. Then $t[x := s] = s$, and $[\![\Gamma, x^A \vdash t]\!] = \mathsf{proj}_{n+1}^{n+1}$ by construction, and:

$$\begin{aligned}
[\![\Gamma \vdash (\lambda x^A.t)s]\!] &= g \mapsto [\![\Gamma \vdash \lambda x^A.y]\!](g)\big([\![\Gamma \vdash s]\!](g)\big) \\
&= g \mapsto \mathsf{proj}_{n+1}^{n+1}\big((g, [\![\Gamma \vdash s]\!](g))\big) \\
&= [\![\Gamma \vdash s]\!]
\end{aligned}$$

Suppose that $t = uv$ or $t = \lambda y^C.u$. Then $t[x := s] = (u[x := s])(v[x := s])$ or $\lambda y^C.u[x := s^*]$ or just $\lambda x^A.u$ (in case $y = x$), and the result follows by induction. Here, s^* is of course just s in which y is renamed to a fresh variable. □

Extensionality

Proposition 3.4.7. *The interpretation satisfies the η-rule, as well as function extensionality, regardless of whether the underlying PCA does.*

Proof. By the following simple computations (noting that x does not occur freely in f):

$$\begin{aligned}
[\![\Gamma \vdash \lambda x^A.\, fx]\!] &= \mathsf{curry}([\![\Gamma, x^A \vdash fx]\!]) \\
&= g \mapsto a \mapsto [\![\Gamma, x^A \vdash fx]\!](g, a) \\
&= g \mapsto a \mapsto [\![\Gamma, x^A \vdash f]\!](g, a)\big([\![\Gamma, x^A \vdash x]\!](g, a)\big) \\
&= g \mapsto a \mapsto [\![\Gamma \vdash f]\!](g)(a) \\
&= [\![\Gamma \vdash f]\!]
\end{aligned}$$

$$\begin{aligned}
[\![\Gamma, x^A \vdash fx]\!] &= g \mapsto [\![\Gamma, x^A \vdash f]\!](g)\big([\![\Gamma, x^A \vdash x]\!](g)\big) \\
&= (g', a) \mapsto [\![\Gamma, x^A \vdash f]\!](g', a)(a) \\
&= (g', a) \mapsto [\![\Gamma \vdash f]\!](g', a) \\
&= [\![\Gamma \vdash f]\!]
\end{aligned}$$

□

Type-freeness

Note also, that the interpretation essentially erases the type information: the realizers of the interpreted elements are certainly type-less objects. In particular, if we set $\mathcal{D} = \Lambda$, it can be easily seen that the interpretation amounts to type erasure.

(Although, it must also be noted that because we don't interpret elements to single elements of $\mathcal{D}$ but rather entire sets of realizers, the interpretation is way more extensional than just an intensional type erasure, as noted above.)

4 Modest sets, PERs, and some constructions

4.1 Modest sets

Definition 4.1.1. A $\mathcal{D}$-set A is called *modest*, when its codes do not realize multiple distinct elements: $\forall_{a,b\in A}\forall_{n\in\mathcal{D}}[\, d \Vdash a \wedge d \Vdash b \Rightarrow a = b\,]$. In terms of the realizer predicate, this means that for all $a \neq b \in A$, we have $E(a) \cap E(b) = \varnothing$.

Note that modest $\mathcal{D}$-sets form a full subcategory **Mod** of $\mathcal{D}$-**Set**, simply because nothing extra is required of morphisms. We will also just refer to modest $\mathcal{D}$-sets as simply modest sets, because there will be no ambiguity. //

Modest sets have some nice properties. For instance, the fact that codes may not multiple realize different elements has as a consequence that every morphism between modest sets is uniquely determined by some $e \in \mathcal{D}$.

Proposition 4.1.2. *Any morphism between two given modest sets is uniquely determined by any of its realizers $e \in \mathcal{D}$.*

Proof. Crucial though, is the $\mathcal{D}$-set requirement that every element of A is realized. Let $f : A \to B$ and $g : A \to B$ be two distinct morphisms between modest sets, both realized by $e \in \mathcal{D}$. Because f and g are distinct, there is some $a \in A$ for which $f(a) \neq g(a)$, for which we can then find some $n \Vdash_A a$ for which we know that $e \cdot n \Vdash_B f(a)$ as well as $e \cdot n \Vdash_B g(a)$. By modesty of B, we must have $g(a) = f(a)$, contradicting our assumption. □

Convention 4.1.3. *We will allow ourselves to write $[e] : A \to B$ to denote the unique morphism that e realizes from A to B, if it exists. (Because of its correspondence with the notation $[e]_{B^A}$ for pers A and B that will be introduced below.)*

4.1.1 Partial equivalence relations

An alternative characterization of modest $\mathcal{D}$-sets is by means of partial equivalence relations on $\mathcal{D}$.

Definition 4.1.4. A *partial equivalence relation* (or *per*) on $\mathcal{D}$ is a symmetric and transitive relation $R \subseteq \mathcal{D} \times \mathcal{D}$. We write $n \sim_R m$ for $(n, m) \in R$. We define the *domain* or *carrier* of R as $\operatorname{dom} R \triangleq \{n \in \mathcal{D} \mid n \sim_R n\}$, and denote by $\mathcal{Q}(R) \triangleq \{[n]_R \mid n \in \operatorname{dom} R\}$ the *quotient*, i.e. set of equivalence classes of the elements of R's domain. Note that indeed R restricted to its domain is a normal equivalence relation—the rest of R is fully disrelated.

A *per morphism* $R \xrightarrow{f} S$ is a function $f : \mathcal{Q}(R) \to \mathcal{Q}(S)$ that is *tracked* (or, *realized*) by some $e \in \mathcal{D}$:

$$\forall_{n\in\operatorname{dom} R}\big[\, f([n]_R) = [e \cdot n]_S \,\big]$$

We denote the category of pers by **Per**. //

Although pers do not even have the shape of $\mathcal{D}$-sets, they naturally describe modest sets via their quotient: if R is a per, then $\mathcal{Q}(R)$ is a modest set, with its realizability relation defined as

$$n \Vdash_{\mathcal{Q}(R)} [m]_R \quad \text{iff} \quad n \in [m]_R$$

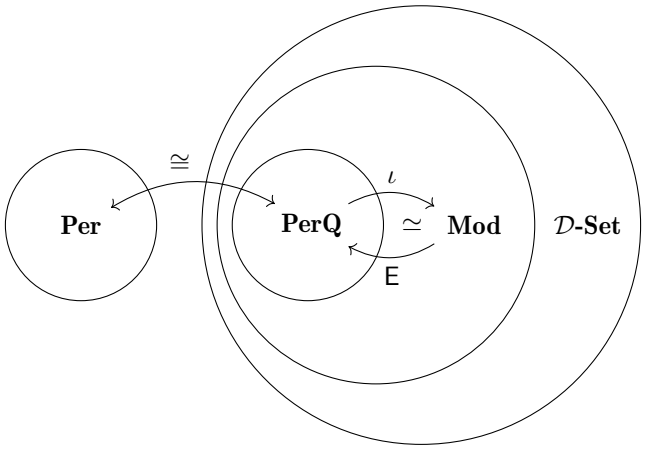


Figure 6: The relationships between the categories of **Per**, **PerQ**, **Mod**, and $\mathcal{D}$-**Set**.

This is the same as stating that it has the realizability predicate $E_{\mathcal{Q}(R)}(n) \triangleq [n]_R$. Modesty follows from the fact that equivalence classes are disjunct.

Definition 4.1.5. Let **PerQ** denote the category of quotients of pers $\mathcal{Q}(R)$ and per morphisms between them, i.e. $f : \mathcal{Q}(R) \to \mathcal{Q}(S)$ in **PerQ** iff $f : R \to S$ in **Per**. Let $\mathcal{Q} : \mathbf{Per} \cong \mathbf{PerQ}$ denote the isomorphism between the two categories. //

On the one hand, describing these objects as partial equivalence relations will turn out to be very useful from the standpoint of writing definitional equations. On the other hand, viewing them as their quotients allows them to 'physically' be modest $\mathcal{D}$-sets. Hence, we adopt the following convention, which allows us to sloppily use both methods of description interchangeably.

Convention 4.1.6. *We will, when no ambiguity can arise, identify pers with their quotients. This means that when we say 'A is a per', we are speaking of its modest set quotient, but allow ourselves to write $n \sim_A m$ as well as $[n] \in A$, or even $X \in A$ for $X \subseteq \mathcal{D}$. When necessary, we will write $\mathcal{Q}(A)$, or $A \subseteq \mathcal{D} \times \mathcal{D}$, or mention* **PerQ***, to be explicit.*

Although pers are just a subclass of modest sets, with carrier $\mathcal{D}$, they do characterize, up to isomorphism, the entire class of modest sets, due to the natural way a quotient of $\mathcal{D}$ can be extracted from any modest set.

Proposition 4.1.7. *The category of pers is a full subcategory of, and in fact equivalent to, the category of modest sets.*

Proof. A per is already modest, as we've seen, and hence we have an inclusion $\iota : \mathbf{PerQ} \to \mathbf{Mod}$. **Per** is indeed a full subcategory because the $\mathcal{D}$-set tracking requirement is fulfilled by the per tracking requirement.

The other way around, we can easily define $\mathsf{E} : \mathbf{Mod} \to \mathbf{PerQ}$ simply as the functor that maps a modest set to the range of its realizability predicate $\mathsf{E}(A) \triangleq \{E_A(n) \mid n \in \operatorname{dom} E_A\} = \{[n] \mid n \in \operatorname{dom} E_A\}$, and $\mathsf{E}(A \xrightarrow{f} B)(E_A(a)) = E_B(f(a))$. Seen as a partial equivalence relation, this means that

$$n \sim_{\mathsf{E}(A)} m \quad \text{iff} \quad \exists_{a \in A}\big[\, n \Vdash a \wedge m \Vdash a \,\big]$$

We have an obvious equality $\mathsf{E}(\iota(A)) = A$ for pers A, and an isomorphism $\iota(\mathsf{E}(A)) \cong A$ for modest sets A. Hence, we have our equivalence of categories. □

Remark 4.1.8. *One might wonder whether a non-modest $\mathcal{D}$-set X can also be made to resemble a per, using the same construction as for modest sets. It may be informative to note that this doesn't just fail on the fact that $\mathsf{E}(X)$ will not be a partial partition of $\mathcal{D}$, but also in that it might occur that multiple elements of X are sent to the same subset of $\mathcal{D}$.*

4.1.2 Modest sets form a cartesian closed category

We will work with pers, and hence the result holds for modest sets in general, as well.

Proposition 4.1.9. *Modest sets are closed under $\mathcal{D}$-set products and exponentials.*

Proof. It is easily see that products of modest sets are modest as well. Suppose A and B are modest sets, and consider their $\mathcal{D}$-set exponential B^A. We can then apply proposition 4.1.2 to see that any code can at most denote one morphism. Hence B^A is modest. □

Proposition 4.1.10. **Mod** *is cartesian closed (and hence* **Per** *too).*

Proof. The previous proposition shows that **Mod** has products and exponentials. Define the final modest set $\mathbf{1}$ as the per with quotient $\{\mathcal{D}\}$, i.e. fully related. Indeed any modest set morphism $f : A \to \mathbf{1}$ must send any element to $\mathcal{D}$, hence there is only one such morphism. Define the initial modest set $\mathbf{0}$ as the per with quotient $\varnothing$. Indeed there is always only one modest set morphism out of it, i.e. the empty one. □

Proposition 4.1.11. *For* **Per**, *explicitly, these constructions amount to:*

$$\begin{aligned}
\mathbf{0} &\triangleq \varnothing \\
\mathbf{1} &\triangleq \mathcal{D} \times \mathcal{D} \\
n \sim_{(A \times B)} m &\Leftrightarrow n_0 \sim_A m_0 \wedge n_1 \sim_B m_1 \\
d \sim_{B^A} e &\Leftrightarrow \forall_{n,m \in \mathcal{D}} \big[\, n \sim_A m \Rightarrow (d \cdot n) \sim_B (e \cdot m) \,\big]
\end{aligned}$$

Proof. For $\mathbf{0}$ and $\mathbf{1}$ this is obvious.

For products, first let A and B denote modest sets, so that $\mathsf{E}A$ and $\mathsf{E}B$ are pers. Then,

$$\begin{aligned}
n \sim_{(\mathsf{E}A \times \mathsf{E}B)} m &\Leftrightarrow n_0 \sim_{\mathsf{E}A} m_0 \wedge n_1 \sim_{\mathsf{E}B} m_1 \\
&\Leftrightarrow \exists_{X \in A} \big[\, \{n_0, m_0\} \subseteq X \,\big] \wedge \exists_{Y \in B} \big[\, \{n_1, m_1\} \subseteq Y \,\big] \\
&\Leftrightarrow \exists_{X \in A, Y \in B} \big[\, \{n_0, m_0\} \subseteq X \wedge \{n_1, m_1\} \subseteq Y \,\big] \\
&\Leftrightarrow \exists_{X \in A, Y \in B} \big[\, \{n, m\} \subseteq \{\langle d, e\rangle \mid d \in X, e \in Y\} \,\big] \\
&\Leftrightarrow \exists_{X \in (A \times B)} \big[\, \{n, m\} \subseteq X \,\big] \\
&\Leftrightarrow n \sim_{\mathsf{E}(A \times B)} m
\end{aligned}$$

The other way around, let R and S be pers, so that $\mathcal{Q}R$ and $\mathcal{Q}S$ are modest sets.

$$\begin{aligned}
\mathcal{Q}(R) \times \mathcal{Q}(S) &= \big\{ \{\langle d', e'\rangle \mid d' \in X, e' \in Y\} \mid X \in \mathcal{Q}(R),\, Y \in \mathcal{Q}(S) \big\} \\
&= \big\{ \{\langle d', e'\rangle \mid d' \in [d]_R, e' \in [e]_S\} \mid d \in \operatorname{dom} R,\, e \in \operatorname{dom} S \big\} \\
&= \big\{ [\langle d, e\rangle] \mid d \in \operatorname{dom} R,\, e \in \operatorname{dom} S \big\} \\
&= \big\{ [n] \mid n_0 \in \operatorname{dom} R \wedge n_1 \in \operatorname{dom} S \big\} \\
&= \big\{ [n] \mid n_0 \sim_R n_0 \wedge n_1 \sim_S n_1 \big\} \\
&= \big\{ [n] \mid n \sim_{(R \times S)} n \big\} \\
&= [n]_{n \in \operatorname{dom}(R \times S)} \\
&= \mathcal{Q}(R \times S)
\end{aligned}$$

For the exponential, let A and B denote modest sets, so that $\mathsf{E}A$ and $\mathsf{E}B$ are pers. Then,

$$\begin{aligned} d \sim_{\mathsf{E}(B^A)} e &\Leftrightarrow \exists_{X \in B^A}\big[\, \{d, e\} \subseteq X \,\big] \\ &\Leftrightarrow \exists_{f:A\to B \text{ in } \mathbf{Per}}\big[\, d, e \text{ both track } f \,\big] \\ &\Leftrightarrow \exists_{f:A\to B \text{ in } \mathbf{Per}}\, \forall_{n\in\mathcal{D}} \forall_{X\in A}\big[\, n \in X \Rightarrow (d\cdot n) \in f(X) \wedge (e\cdot n) \in f(X) \,\big] \quad \text{(a)} \\ d \sim_{(\mathsf{E}B)^{(\mathsf{E}A)}} e &\Leftrightarrow \forall_{n,m\in\mathcal{D}}\big[\, n \sim_{\mathsf{E}A} m \Rightarrow (d\cdot n) \sim_{\mathsf{E}B} (e\cdot m) \,\big] \\ &\Leftrightarrow \forall_{n,m\in\mathcal{D}}\big[\, \exists_{X\in A}\big[\, \{n, m\} \in X \,\big] \Rightarrow \exists_{Y\in B}\big[\, \{(d\cdot n), (e\cdot m)\} \in Y \,\big]\,\big] \qquad \text{(b)} \end{aligned}$$

Suppose (a), and let $n, m \in \mathcal{D}$, and $X \in A$ such that $\{n, m\} \in X$. Set $Y := f(X)$, then indeed both $(d \cdot n) \in Y$ and $(e \cdot m) \in Y$. For the other way around, suppose (b). Then define $f(X) \triangleq \{(d \cdot n) \mid n \in X\} \cup \{(e \cdot n) \mid n \in X\}$. This is a morphism in **Per** because it is tracked by (both) d and e, and obviously it fulfills its requirement.

The other way around, let R and S be pers. Compute:

$$\begin{aligned} \mathcal{Q}(S^R) &= \big\{ [e]_{S^R} \mid e \in \operatorname{dom} S^R \big\} \\ \mathcal{Q}(S)^{\mathcal{Q}(R)} &= \big\{ \{d \in \mathcal{D} \mid d \text{ realizes } f\} \mid \mathcal{Q}(R) \xrightarrow{f} \mathcal{Q}(S) \text{ in } \mathbf{Per} \big\} \end{aligned}$$

(I∩) Let $e \in \operatorname{dom} S^R$ and consider $[e]_{S^R} = \{d \mid d \sim_{S^R} e\} = \{d \mid \forall_{n,m\in\mathcal{D}}[n \sim_R m \Rightarrow (d\cdot n) \sim_S (e\cdot m)]\}$. Define $f : \mathcal{Q}(R) \to \mathcal{Q}(S)$ as $f(X) \triangleq \{e \cdot n \mid n \in X\}$. Then f is a morphism in **Per** tracked by e. Consider $\{d \in \mathcal{D} \mid d \text{ realizes } f\}$. Indeed this set will be exactly $[e]_{S^R}$, hence we have our inclusion.

(∪I) Consider some morphism $f : \mathcal{Q}(R) \to \mathcal{Q}(S)$ in **Per**, and the set $\{d \in \mathcal{D} \mid d \text{ realizes } f\}$. Any two elements d, e hereof will be S^R-related, and moreover every element $e \in \mathcal{D}$ that is S^R-related to any such d, will also be an element of the set. Hence, we have an equivalence class $[e]_{S^R}$ on our hands. □

Corollary 4.1.12. *The last part of the proof includes an isomorphism (for $A \in$ **Mod** and $B \in$ **Per**):*

$$\mathsf{fn} : \mathsf{E}(B^A) \cong B^A$$

◿

Proof. To be precise, we can define

$$\begin{aligned} \mathsf{fn}(X) &\triangleq a \mapsto \bigcup \big\{ [e\cdot n]_B \mid n \Vdash_A a \wedge e \in X \big\} \\ \mathsf{fn}^{-1}(A \xrightarrow{f} B) &\triangleq \big\{ e \in \mathcal{D} \mid e \text{ realizes } f \big\} \\ &= \big\{ e \in \mathcal{D} \mid \forall_{a\in A} \forall_{n\Vdash_A a}\big[\, e\cdot n \in f(a) \,\big] \big\} \end{aligned}$$

noting that

$$\begin{aligned} \mathsf{fn}\big(\mathsf{fn}^{-1}(A \xrightarrow{f} B)\big)(a) &= \mathsf{fn}\big(\{e \in \mathcal{D} \mid e \text{ realizes } f\}\big)(a) \\ &= f(a) \qquad \text{see below} \\ \mathsf{fn}^{-1}\big(\mathsf{fn}(X)\big) &= \big\{ e \in \mathcal{D} \mid e \text{ realizes } \mathsf{fn}(X) \big\} \\ &= \big\{ e \in \mathcal{D} \mid e \text{ realizes } \big(a \mapsto \{e\cdot n \mid n \Vdash_A a \wedge e \in X\}\big) \big\} \\ &= \big\{ e \in \mathcal{D} \mid \forall_{a\in A} \forall_{n\Vdash_A a}\, [n \Vdash_A a \wedge e \in X] \big\} \\ &= X \end{aligned}$$

□

Remark 4.1.13. *Note how we cannot simply define $\mathsf{fn}(X)(a) = \{e \cdot n \mid n \Vdash_A a \wedge e \in X\}$, because the the codomain could be such that for some a, the set of realizers of f is not*

surjective unto $f(a)$. To give a simple counterexample, let $A = \{\{n\} \mid n \in \mathcal{D}\}$ and $B = \{\{\mathsf{K}n\} \mid n \in \mathcal{D}\}$ and $f(\{n \neq \mathsf{K}\}) = \{\mathsf{K}n\}$ and $f(\{\mathsf{K}\}) = \{\mathsf{KK}, \mathsf{S}\}$. Obviously, only K tracks f, and S will never be reached. To be clear, what we see here, is that that a **Mod** *morphisms can be unambiguously encoded to a subset of $\mathcal{D}$ only with respect to its domain and codomain modest sets A and B (the latter of which, in addition, can of course only be unambiguously 'resolved' as a per).*

Remark 4.1.14. *The notation $d \sim_A e$ used for pers, as opposed to say $d\, A\, e$, is chosen specifically because of the extensional nature of the category of pers, see e.g. proposition 4.1.2. We have, for any two pers A and B:*

$$d \sim e \quad \Rightarrow \quad d \sim_{B^A} e \quad \textit{i.e.} \quad [d] = [e] : A \to B$$

Definition 4.1.15. A per A is called *discrete* if each equivalence class is a singleton set. For any subset X of $\mathcal{D}$, denote by ΔX the discrete per over domain X, i.e. with $n \sim_{\Delta X} m$ iff $n = m \in X$. //

Proposition 4.1.16. *The model construction for the simply typed lambda calculus can be carried over to work exclusively with modest sets, resp. partial equivalence relations.*

Proof. This is not too hard to see: modest sets (resp. pers) admit the CCC structure necessarily to interpret $\lambda\!\to$'s types. □

4.1.3 Coproduct

Defining coproducts, we have to pay a little more attention: we need the indexing objects to be distinguishable within $\mathcal{D}$, if we want to be able to track the required outgoing morphism of the universal property.

Definition 4.1.17. If A and B are modest sets, we can define the coproduct $A + B$ together with its injections as

$$\begin{aligned} A + B &\triangleq \big\{\{\langle \mathsf{true}, n\rangle \mid n \in X\} \mid X \in A\big\} \cup \big\{\{\langle \mathsf{false}, n\rangle \mid n \in X\} \mid X \in B\big\} \\ \mathsf{inl} &\triangleq [\mathit{\Lambda} n.\langle \mathsf{true}, n\rangle] : A \to A + B \\ \mathsf{inr} &\triangleq [\mathit{\Lambda} n.\langle \mathsf{false}, n\rangle] : B \to A + B \end{aligned}$$

For any two morphisms $[d] : A \to C$ and $[e] : B \to C$, we can define the (unique) morphism $[\,[d], [e]\,] \triangleq [\mathit{\Lambda} p.\,(\mathsf{if} \cdot p_0 \cdot d \cdot e) \cdot p_1)] : A + B \to C$ such that indeed $[\,[d], [e]\,] \circ \mathsf{inl} \sim [d]$ and $[\,[d], [e]\,] \circ \mathsf{inr} \sim [e]$. //

Of course this definition, too, can be specified to the case of modest sets, resp. pers.

4.2 Dependent sums and products

Definition 4.2.1. Given a $\mathcal{D}$-set A and a family of $\mathcal{D}$-sets $B : |A| \to \mathcal{D}$**-Set**. We say that some function $f : (a \in |A|) \to |B(a)|$ is *tracked* by $e \in \mathcal{D}$ iff $\forall_{a \in A}\, \forall_{n \in \mathcal{D}}\, [\, n \Vdash_A a \Rightarrow (e \cdot n) \Vdash_{B(a)} f(a)\,]$, analogous to the definition of $\mathcal{D}$-set morphisms.

Then, define the $\mathcal{D}$-sets

$$\begin{aligned} &\Sigma_{a \in A} B(a) \triangleq \Sigma_{a \in |A|}\, |B(a)| \qquad \text{with} \quad \langle n, m\rangle \Vdash_{\Sigma_{a \in A} B(a)} (a, b) \Leftrightarrow n \Vdash_A a \wedge m \Vdash_{B(a)} b \\ &\Pi_{a \in A} B(a) \triangleq {}^{\mathsf{tracked}}\Pi_{a \in |A|}\, |B(a)| \quad \text{with} \qquad e \Vdash_{\Pi_{a \in A} B(a)} f \Leftrightarrow f \text{ is tracked by } e \end{aligned}$$

(Here, the superscript '$\mathsf{tracked}$' is meant to indicate that only those functions as included that are indeed tracked by some e.) Moreover, we have: //

We have to be a bit careful though, for this rendering of the dependent sum need not actually have to satisfy the universal property that one would expect.

Proposition 4.2.2. *We do not always have, for any given family of morphisms* $\{B(a) \xrightarrow{c_a} C\}_a$*, a morphism* $\Sigma_{a\in A}B(a) \xrightarrow{f} C$ *such that* $f(a,b) = c_a(b)$ *for all* $(a,b) \in \Sigma_{a\in A}B(a)$*.*

Proof. Obviously, due to decidability issues. Define the per $A = \{K, \overline{K}\}$, i.e. the Halting set and its complement, and simply let $B_{(-)} = \mathbf{1}$. Then, for the per $C = \{\{\mathsf{true}\}, \{\mathsf{false}\}\}$, and the two morphisms $c_K = k \cdot \mathsf{true}$ and $c_{\overline{K}} = k \cdot \mathsf{false}$. Then we cannot find an f such that $f \circ \mathsf{in}_a = c_a$ for both $a = K, \overline{K}$, for any e tracking this morphism would solve the Halting problem. □

As mentioned already, above, the key here is *uniform realizability.*

Definition 4.2.3. If A is a $\mathcal{D}$-set, and $\{B \xrightarrow{f_a} C\}_a$ is a family of morphisms such that some $g \in \Pi_{a\in A}(B \to C)$ exists with $g(a) = f_a$, then we say that the family is *uniformly tracked / realized.* //

Convention 4.2.4. *We will write* $\{B \xrightarrow{f(a)} C\}_a$ *for a uniformly tracked family of morphisms, identifying* $f(a) = f_a$*, and write* $\{B \xrightarrow{f_a} C\}_a$ *for a family of morphisms that is not, or not necessarily known to be, uniformly tracked.*

Proposition 4.2.5. *For any uniformly realizable family of morphisms* $\{B(a) \xrightarrow{c(a)} C\}_a$*, meaning that* $c \in \Pi_{a\in A}\big(B(a) \to C\big)$*, there is a morphism* $\Sigma_{a\in A}B(a) \xrightarrow{f} C$ *such that for all* $(a,b) \in \Sigma_{a\in A}B(a)$*, we have* $f(a,b) = c(a)(b)$*.*

Proof. By currying. □

The reverse transformation also holds, and in fact the transformation itself is tracked:

Proposition 4.2.6. *Every morphism* $\Sigma_{a\in A}B(a) \xrightarrow{f} C$ *can be 'factored' to a uniformly tracked family of morphisms* $\{B(a) \xrightarrow{f(a)} C\}_a$*; and moreover this, and the previous transformation, are tracked.*

Proof. By uncurrying, and the procedure of currying and uncurrying is realizable. □

Proposition 4.2.7. *There is an easy map* $\mathsf{in} \in \Pi_{a\in A}\big(B(a) \to \Sigma_{a\in A}B(a)\big)$*, which can also be said to be a uniformly realized family* $\{B(a) \xrightarrow{\mathsf{in}(a)} \Sigma_{a\in A}B(a)\}_a$*.*

Proof. Trivial. □

Proposition 4.2.8. *The* $\mathcal{D}$*-sets* $\Sigma_{a\in A}B(a)$ *and* $\Pi_{a\in A}B(a)$ *are indeed the categorical dependent sum and product in* $\mathcal{D}$**-Set***—modulo uniform realizability.*

Proof. Indeed, the dependent product's eval is tracked by the same PCA element as was the (non-dependent) exponential, and similarly has an abstraction morphism (whose type gets a bit messy). That they satisfy the required universal properties follows yet again from the fact that we're essentially still working with sets, the required functions are indeed tracked easily, and the antagonist functions of the UP, if anything, have a smaller range. We've also just seen that the dependent sum satisfies the required properties. □

Proposition 4.2.9. *If* $B : |A| \to$ **Mod***, then* $\Pi_{a\in A}B(a)$ *is modest.*

Proof. Suppose we have e, f, g such that $e \Vdash f$ and $e \Vdash g$. Let $a \in A$. Then $f(a) = g(a)$ follows from the modesty of $B(a)$ after having found any $n \in \mathcal{D}$ such that $n \Vdash_A a$. That we can find such an n follows from the requirement of $\mathcal{D}$-sets that every element must indeed be coded. Thus $f = g$ by extensionality. □

Proposition 4.2.10. *If $B : |A| \to \mathbf{Mod}$, then we can characterize the per-form that the definition of Π takes as as*

$$d \sim_{\mathsf{E}(\Pi_{a\in A}B(a))} e \quad \Leftrightarrow \quad \forall_{a\in A}\big[\, n \Vdash_A (a) \wedge m \Vdash_A (a) \Rightarrow (d\cdot n) \sim_{\mathsf{E}(B(a))} (e\cdot m)\,\big]$$

Proof. (Unfolding definitions. Note that this is just a slightly more complicated form of the per exponential that we've defined before.) □

Proposition 4.2.11. *If A is modest, and $B : A \to \mathbf{Mod}$, then so is $\Sigma_{a\in A}B(a)$.*

Proof. This follows directly from the definition. □

Now, we will relativize these data to a given context $\Gamma \in \mathcal{D}$-**Set**, which we will need in order to interpret the Calculus of Constructions.

Definition 4.2.12. Given a $\mathcal{D}$-set Γ, and families $A : |\Gamma| \to \mathcal{D}$-**Set** and $B : |\Sigma_{\gamma\in\Gamma}A(\gamma)| \to \mathcal{D}$-**Set**, define the $\mathcal{D}$-sets $\Sigma_\Gamma(A,B)$ and $\Pi_\Gamma(A,B)$ as the dependent product spaces that send $\gamma \in \Gamma$ to the dependent sum / product of A and B at γ:

$$\Sigma_\Gamma(A,B) \triangleq \Pi_{\gamma\in\Gamma}\Sigma_{a\in A(\gamma)}B(\gamma,a) \qquad \sigma_\Gamma(A,B) \triangleq \gamma \mapsto \Sigma_{a\in A(\gamma)}B(\gamma,a)$$
$$\Pi_\Gamma(A,B) \triangleq \Pi_{\gamma\in\Gamma}\Pi_{a\in A(\gamma)}B(\gamma,a) \qquad \pi_\Gamma(A,B) \triangleq \gamma \mapsto \Pi_{a\in A(\gamma)}B(\gamma,a) \qquad /\!/$$

Proposition 4.2.13. *If B is a modest family, then each $P \in \Pi_\Gamma(A,B)$ is a modest family as well.*

Proof. This follows directly from applying proposition 4.2.9 twice. □

Proposition 4.2.14. *If A and B are modest families, the each $S \in \Sigma_\Gamma(A,B)$ is a modest family as well.*

Proof. This follows directly from applying proposition 4.2.11 and 4.2.9. □

Definition 4.2.15. Let Γ be a $\mathcal{D}$-set, $A : |\Gamma| \to \mathcal{D}$-**Set**, and $B : |\Sigma_{\gamma\in\Gamma}A(\gamma)| \to \mathcal{D}$-**Set**. Define:

$$\mathsf{eval}_{\Gamma,A,B} \in \Pi_{\left[(f,a)\in \Pi_\Gamma(A,B)\times\Pi_{\gamma\in\Gamma}A(\gamma)\right]}\ \Pi_{\gamma\in\Gamma}B(\gamma,a(\gamma))$$
$$\mathsf{eval}_{\Gamma,A,B}(f,a)(\gamma) \triangleq f(\gamma)(a(\gamma))$$

This definition is valid, because $\mathsf{eval}_{\Gamma,A,B}(f,a) \in \Pi_{\gamma\in\Gamma}B(\gamma,a(\gamma))$ is tracked by $\boldsymbol{\lambda} z.\,(e\cdot z)\cdot(n\cdot z)$, where e tracks f and $n \Vdash_{A(\gamma)} a$, and subsequently $\mathsf{eval}_{\Gamma,A,B}$ is tracked by $\boldsymbol{\lambda} m.\boldsymbol{\lambda} z.\,(m_0\cdot z)\cdot(m_1\cdot z)$.

Informally, this is of course just a messy internal $\mathcal{D}$-**Set** equivalent of the following picture:

$$\mathsf{eval}_{\Gamma,A,B} : \left({B_{(\gamma,a)}}^{A_{(\gamma)}}\right)^\Gamma \times {A_{(\gamma)}}^\Gamma \to {B_{(\gamma,a)}}^\Gamma \qquad /\!/$$

Definition 4.2.16. Let Γ be a $\mathcal{D}$-set, $A : |\Gamma| \to \mathcal{D}$-**Set**, and $B : |\Sigma_{\gamma\in\Gamma}A(\gamma)| \to \mathcal{D}$-**Set**. Define:

$$\mathsf{curry}_{\Gamma,A,B} \in \Pi_{\left[t\in \Pi_{(\gamma,a)\in\Sigma_{\gamma\in\Gamma}A(\gamma)}B(\gamma,a)\right]}\ \Pi_\Gamma(A,B)$$
$$\mathsf{curry}_{\Gamma,A,B}(t)(\gamma)(a) \triangleq t(\gamma,a)$$

This definition is valid, because $\mathsf{curry}_{\Gamma,A,B}(t) \in \Pi_{\gamma\in\Gamma}\Pi_\Gamma(A,B)$ is tracked by $\boldsymbol{\lambda} z.\boldsymbol{\lambda} n.\,t\cdot\langle z,n\rangle$, and subsequently $\mathsf{curry}_{\Gamma,A,B}$ is tracked by $\boldsymbol{\lambda} m.\boldsymbol{\lambda} z.\boldsymbol{\lambda} n.\,m\cdot\langle z,n\rangle$.

Informally, this is of course just a messy internal $\mathcal{D}$-**Set** equivalent of the following picture:

$$\mathsf{curry}_{\Gamma,A,B} : {B_{(\gamma,a)}}^{\Gamma\times A_{(\gamma)}} \to \left({B_{(\gamma,a)}}^{A_{(\gamma)}}\right)^\Gamma \qquad /\!/$$

Definition 4.2.17. Similarly, we can define

$$\mathsf{uncurry}_{\Gamma,A,B} \in \Pi_{\left[t\in\Pi_\Gamma(A,B)\right]}\ \Pi_{(\gamma,a)\in\Sigma_{\gamma\in\Gamma}A(\gamma)}B(\gamma,a)$$
$$\mathsf{uncurry}_{\Gamma,A,B}(t)(\gamma,a) \triangleq t(\gamma)(a) \qquad /\!/$$

Per-specified counterparts

Let $\Gamma \in \mathcal{D}\textbf{-Set}$, $A : |\Gamma| \to \mathcal{D}\textbf{-Set}$, $M : |\Gamma| \to \textbf{Mod}$, and $B : |\Sigma_{\gamma\in\Gamma} A(\gamma)| \to \textbf{Mod}$.

For reasons that will be explained in the following chapter (on the interpretation of CC), we will need versions of previously introduced constructions that, over the equivalence $\textbf{Per} \simeq \textbf{Mod}$, obtain and/or work specifically with objects of **Per**, i.e. applying this equivalence of proposition 4.1.7, and the isomorphism $\mathsf{fn} : \mathsf{E}(M^\Gamma) \cong M^\Gamma$ of corollary 4.1.12 at certain places.

Definition 4.2.18. Define the *per-specified* Π^{per} and $\Pi^{\mathsf{per}}_\Gamma$ as:

$$\Pi^{\mathsf{per}}_{\gamma\in\Gamma} M(\gamma) \in \textbf{Per} \qquad \Pi^{\mathsf{per}}_\Gamma(A,B) \in \textbf{Per}$$
$$\Pi^{\mathsf{per}}_{\gamma\in\Gamma} M(\gamma) \triangleq \mathsf{E}\big(\Pi_{\gamma\in\Gamma} M(\gamma)\big) \qquad \Pi^{\mathsf{per}}_\Gamma(A,B) \triangleq \Pi_{\gamma\in\Gamma}\Pi^{\mathsf{per}}_{a\in A(\gamma)} B(\gamma,a)$$

$$\sigma^{\mathsf{per}}_\Gamma(A,B) \triangleq \gamma \mapsto \Sigma^{\mathsf{per}}_{a\in A(\gamma)} B(\gamma,a)$$
$$\pi^{\mathsf{per}}_\Gamma(A,B) \triangleq \gamma \mapsto \Pi^{\mathsf{per}}_{a\in A(\gamma)} B(\gamma,a) \qquad /\!/$$

Note that the isomorphism of corollary 4.1.12 can easily be lifted over proposition 4.2.9:

$$\mathsf{fn}_\Pi : \Pi^{\mathsf{per}}_{\gamma\in\Gamma} M(\gamma) \cong \Pi_{\gamma\in\Gamma} M(\gamma)$$

Definition 4.2.19. Define the *per-specified* curry and eval as:

$$\mathsf{curry}^{\mathsf{per}}_{\Gamma,A,B} \in \Pi_{\left[t \in \Pi^{\mathsf{per}}_{(\gamma,a)\in\Sigma_{\gamma\in\Gamma}A(\gamma)} B(\gamma,a)\right]} \Pi^{\mathsf{per}}_\Gamma(A,B)$$
$$\mathsf{curry}^{\mathsf{per}}_{\Gamma,A,B}(t)(\gamma)(a) \triangleq \mathsf{fn}^{-1}\big(\mathsf{fn}_\Pi(t)(\gamma,a)\big)$$

$$\mathsf{eval}^{\mathsf{per}}_{\Gamma,A,B} \in \Pi_{\left[(f,a) \in \Pi^{\mathsf{per}}_\Gamma(A,B)\times\Pi_{\gamma\in\Gamma}A(\gamma)\right]} \Pi^{\mathsf{per}}_{\gamma\in\Gamma} B(\gamma,a(\gamma))$$
$$\mathsf{eval}^{\mathsf{per}}_{\Gamma,A,B}(f,a)(\gamma) \triangleq \mathsf{fn}^{-1}\big(\mathsf{fn}_\Pi\big(f(\gamma)\big)\big(a(\gamma)\big)\big)$$

Note that because these functions operate solely isomorphically on the structure of their operands (over fn), they are still tracked by the same elements of $\mathcal{D}$ as before. $/\!/$

4.3 A universe object and polymorphism

Definition 4.3.1. We can define the following $\mathcal{D}$-set, which will come to play the role of the *universe object*, i.e. the interpretation of the universe of (names of) small types Prop:

$$\mathcal{U} \triangleq \nabla\textbf{Per} \qquad /\!/$$

As mentioned earlier, this is useful due to the following theorem:

Theorem 4.3.2 (Longo and Moggi [1991])**.** *For any set X and family $A : X \to \textbf{Mod}$,*

$$\prod_{x\in\nabla X} A(x) \cong \bigcap_{x\in X} \mathsf{E}(A(x)) \qquad \textit{in } \textbf{Mod}$$

Proof. Let $f \in \Pi_{x\in\nabla X} A(x)$, i.e. a function $f : \Pi_{x\in X}|A(x)|$ tracked by some $e \in \mathcal{D}$. Because ∇X carries the full realizability structure, $n \Vdash_{\nabla X} x$ always holds, and hence the tracking requirement of e degenerates to $\forall_{n\in\mathcal{D}}[\,(e\cdot n) \Vdash_{A(x)} f(x)\,]$. Informally, what is happening here, is that f is forced to 'ignore' n; ∇X is essentially a singleton in **Mod**, and thus f, as a

generalized element in **Mod**, is fully defined on its only output. Define $Y_f \triangleq \{e \cdot n \mid n \in \mathcal{D}\}$, and note that it is indeed an equivalence class $Y_f = [e \cdot -]_{\mathsf{E}(A(x))} \in A(x)$, for any x. Hence $Y_f \in \bigcap_{x \in X} \mathsf{E}(A(x))$. Note that this map $f \mapsto Y_f$ is realized by $\lambda\!\!\lambda e.\, e \cdot z$ (for any choice of z) and hence a morphism in **Mod**.

For the other way around, let $Y \in \bigcap_{x \in X} \mathsf{E}(A(x))$, i.e. $Y = [n]_{\mathsf{E}(A(x))} \in \mathsf{E}(A(x))$ for all $x \in X$. If the left hand would would have been a partial equivalence relation, we could have simply defined the constant map $f_Y : \Pi_{x \in X} \mathsf{E}(A(x))$ sending $x \mapsto Y$. Because this map is tracked by $k \cdot n$, we have a modest set morphism $f_Y : \Pi_{x \in \nabla X} \mathsf{E}(A(x))$, realized by $\lambda\!\!\lambda n.\, k \cdot n$. But, the left hand side is a modest exponential, so although the realizer is identical, the morphism realized is in fact $x \mapsto [n]_{A(x)}$.

Summarizing:

$$f \mapsto \{e \cdot n \mid n \in \mathcal{D}\} \qquad \text{(for any e tracking f)}$$
$$\big(x \mapsto [n]_{A(x)}\big) \leftarrow\!\shortmid [n]$$

These maps are indeed inverses in **Mod**, which we may calculate most easily at the PCA level, using proposition 3.3.1 to guarantee the equality of morphisms tracked by extensionally equivalent PCA elements. For one way, for any $n \in \mathcal{D}$, we have $\big((\lambda\!\!\lambda e.\, e \cdot z) \circ (\lambda\!\!\lambda n.\, k \cdot n)\big) \cdot n \asymp (\lambda\!\!\lambda e.\, e \cdot z) \cdot (k \cdot n) \asymp k \cdot n \cdot z = n$, hence the PCA composition is extensionally equal to the identity morphism, and the morphism composition is the categorical identity morphism.

For the other way, compute:

$$\begin{aligned}\big((\lambda\!\!\lambda n.\, k \cdot n) \circ (\lambda\!\!\lambda e.\, e \cdot z)\big) \cdot d \cdot q &\asymp (\lambda\!\!\lambda n.\, k \cdot n) \cdot (d \cdot z) \cdot q \\ &\asymp k \cdot (d \cdot z) \cdot q \\ &\asymp d \cdot z\end{aligned}$$

Now we have to first use the fact that ∇X is fully realized, and hence any morphism $[d] : \Pi_{x \in \nabla X} A(x)$ indeed has $d \cdot q \asymp d \cdot z$; and then apply proposition 3.3.1, to see that the PCA composition tracks the identity morphism.

Hence, we have an isomorphism in **Mod**. □

Polymorphism

As the typical insight goes, in order to consistently interpret polymorphic types without running in to cardinality issues, these types must be seen to represent the 'smallest type possible for the given specification. E.g., the type $\forall A.A \to A$ denotes not the collection of functions from A to A, for all A, taken together, but rather the collection of functions that, for any A, could be characterized/typed as a functions from A to A.

Example 4.3.3. Consider the polymorphic identity type

$$\vdash \mathsf{PolyId} \triangleq \Lambda A.\lambda x^A.x : \forall A.A \to A$$

Here, we see that the type specification $\forall A$ really means that PolyId is a function that, for any conceivable type A, can be made of the type $A \to A$, i.e. we want (informally):

$$[\![\vdash \mathsf{PolyId}]\!] \in \bigcap_A [\![A \to A]\!]$$

This is where we invoke theorem 4.3.2 and set, for $F(A) \triangleq A^A$:

$$[\![\vdash \mathsf{PolyId}]\!] \triangleq \prod_{A \in \mathcal{U}} F(A) \cong \bigcap_{A \in \mathcal{U}} F(A)$$

To see what's really going on in $\Pi_{A\in\mathcal{U}}F(A)$, note that an element of it is just a function $f : \Pi_{A\in|\mathcal{U}|}|F(A)|$, that is tracked by some $e \in \mathcal{D}$, which means:

$$\begin{aligned} & \forall_{A\in\mathcal{U}}\forall_{n\in\mathcal{D}}\big[\, n \Vdash_{\mathcal{U}} A \Rightarrow (e\cdot n) \Vdash_{A^A} \mathsf{id}_A \,\big] \\ \text{iff} \quad & \forall_{A\in\mathcal{U}}\forall_{n\in\mathcal{D}}\big[\, (e\cdot n) \Vdash_{A^A} \mathsf{id}_A \,\big] \\ \text{iff} \quad & \forall_{A\in\mathcal{U}}\forall_{n\in\mathcal{D}}\forall_{a\in A}\forall_{m\in\mathcal{D}}\big[\, m \Vdash_A a \Rightarrow (e\cdot n\cdot m) \Vdash_A a \,\big] \end{aligned}$$

What we see is that n is essentially disregarded, and e takes on the role of having to track id_A regardless of the A in question. ◿

Definition 4.3.4. Motivated by the previous, we define the per $\forall F$, for any function $F : \mathbf{Per} \to \mathbf{Per}$, as:

$$\forall F \triangleq \bigcap_{A\in\mathbf{Per}} F(A)$$

//

We already have for each such $F = \Pi_{A\in\mathcal{U}}\, F_A \in \Pi_{A\in\mathcal{U}}\,\mathcal{U}$, the $\mathcal{D}$-set dependent product evaluation map $\mathsf{eval} : \Pi_{(f,A)\in F\times\mathcal{U}}\, F(A)$. The per-specified version of this will give us

$$\begin{aligned} & \mathsf{eval}^{\mathsf{per}} \in \Pi_{(-,A)\in\forall F\times\mathcal{U}}\, A \\ & \mathsf{eval}^{\mathsf{per}}([n]_{\forall F}, A) = [n]_{F(A)} \end{aligned}$$

and is realized by the first projection.

In the model construction to be given in chapter 5, we will have $[\![\vdash \mathsf{PolyId}]\!] \cong \forall(A \mapsto A^A)$.

4.4 Fibrational structure

We've seen that we are able to form dependent sums and products, and more, for $\mathcal{D}$-sets generally, and modest sets specifically. A common categorical way of describing this structure is via fibrations, and it can be insightful to structure the previous in such a fibrational setting.

Specifically the last result, has shown that $\mathcal{D}$ objects, due to the internal category of **Per**, are expressive enough to encode families of pers, indexed by any $\mathcal{D}$-set, which motivates the following definitions.

We can mirror the standard fibration $\mathsf{Fam}(\mathbf{Set}) \xrightarrow{\pi_0} \mathbf{Set}$ to work with pers, using the fact that each family $B : |A| \to \mathbf{Per}$ of pers, indexed by a per A, can indeed be found as an element in the per $\Pi_{a\in A}\mathcal{U}$, which can then be treated as the 'fiber' of pers above A. All these objects/families can already by found in the CC interpretation we gave.

Definition 4.4.1. Define the category $\mathsf{Fam}(\mathcal{D}\text{-}\mathbf{Set})$ as having:

$$\begin{aligned} \text{objects} \quad & \big(I, \{A_i\}_{i\in I}\big) \quad \text{for } I, A_i \in \mathcal{D}\text{-}\mathbf{Set} \\ \text{morphisms} \quad & \{A_i\}_{i\in I} \xrightarrow{g} \{B_j\}_{j\in J} \quad \text{with } g = \big(I \xrightarrow{u} J, \{A_i \xrightarrow{g(i)} B_{u(j)}\}_{i\in I}\big) \\ & \qquad\qquad \textit{uniformly tracked: } \exists_e \forall_{i\in I}\forall_{n\Vdash_I i}\big[\, e\cdot i \Vdash g(i) \,\big] \end{aligned}$$

I.e., families of pers indexed by $\mathcal{D}$-sets as objects, and morphisms that consist of a reindexing $\mathcal{D}$-set morphism and a uniformly tracked family of per morphisms thereover.

Define moreover the (full) subcategory $\mathsf{Fam}(\mathbf{Per})$ of $\mathsf{Fam}(\mathcal{D}\text{-}\mathbf{Set})$ where all A_i and $g(i) : A_i \to B_{u(j)}$ are required to live in **Per**. //

Denote by $\mathbb{P}_I$ the 'fiber over I' with objects and morphisms that are sent to I and id_I, i.e. the subcategory of $\mathsf{Fam}(\mathcal{D}\textbf{-Set})$ or $\mathsf{Fam}(\textbf{Per})$ (which will be clear from the context) with families indexed by the $\mathcal{D}$-set I and with no reindexing. Note that all such objects are essentially contained in the D-set $\Pi_{i\in I}\mathcal{U}$, i.e. we will come to find these objects in the accessible range of our interpretation of CC, as $[\![\Gamma \vdash a : \mathsf{Prop}]\!]$.

Definition 4.4.2. A functor $F : \mathbb{P}_I \to \mathbb{P}_J$ is called *realizable* if it comes equipped with a uniformly tracked family of morphisms $\left\{FA \xrightarrow{p(f)} FB\right\}_{f:A\to B}$. //

Convention 4.4.3. *Whenever we talk about functors over* $\mathcal{D}$**-Set** *and* **Per***, we mean realizable functors (and if not, we will explicitly mention so).*

Definition 4.4.4. Suppose $u : I \to J$ is a $\mathcal{D}$-set morphism. Then we have a *reindexing functor* $u^* : \mathbb{P}_J \to \mathbb{P}_I$ defined by

$$u^*(\{B_j\}_{j\in J}) \triangleq \{B_{u(i)}\}_{i\in I}$$
$$u^*(B \xrightarrow{g} C) \triangleq \left(\mathsf{id}_I, \{g(u(i))\}_{i\in I}\right)$$

Note that $u^*(g)$ is uniformly tracked by $e \circ d$, if g is uniformly tracked by e and u is tracked by d, and subsequently u^* itself is realizable. //

Proposition 4.4.5. *The functor* $\begin{array}{c}\mathsf{Fam}(\mathcal{D}\textbf{-Set})\\ \downarrow{\scriptstyle \pi_0}\\ \mathcal{D}\textbf{-Set}\end{array}$ *is a (cloven) fibration.*

Proof. Suppose we have some $I \xrightarrow{u} J$ in $\mathcal{D}$**-Set** and some J-indexed family of pers $B = \{B_j\}_{j\in J}$. We can then define $u^*(B) \xrightarrow{\overline{u}} B$ in $\mathsf{Fam}(\mathcal{D}\textbf{-Set})$ given by $\overline{u} = \left(u, \{\mathsf{id} : B_{u(i)} \to B_{u(i)}\}_{i\in I}\right)$. Now to show that this is a cartesian morphism over u. Consider any object $C = \{C_k\}_{k\in K}$, morphism $v : K \to I$, and morphism $C \xrightarrow{g} B$ with $g = (u \circ v, \{g(k)\}_{k\in K})$. We can then define $C \xrightarrow{h} u^*(B)$ reindexing over v and with $C_k \xrightarrow{h(k)} u^*(B)_{v(k)} = B_{u(v(k))}$ by setting $h(k) = g(k)$. Thus, the functor is a fibration. Using these choices for u^* and $\overline{u}$, we then have a cloven fibration. □

Proposition 4.4.6. *Restricted to* $\begin{array}{c}\mathsf{Fam}(\textbf{Per})\\ \downarrow{\scriptstyle \pi_0}\\ \mathcal{D}\textbf{-Set}\end{array}$ *, we still have a (cloven) fibration.*

Proof. Obviously, if B lives in $\mathsf{Fam}(\textbf{Per})$, then so will $u^*(B)$, and $\overline{u}$ as well. □

Definition 4.4.7. If we have pers A and B and a morphism $f : A \to B$, we can find for each $b \in B$ the per $f^{-1}(b) \triangleq \{a \in A \mid f(a) = b\}$, inheriting A's realizability predicate. //

Proposition 4.4.8. *The reindexing functor has left and right adjoints* $\Sigma_f \dashv f^* \dashv \Pi_f$ *(both in* $\mathsf{Fam}(\textbf{Per})$ *as in* $\mathsf{Fam}(\mathcal{D}\textbf{-Set})$*).*

Proof. With the usual constructions:

$$\Sigma_f(A) \triangleq \left\{\Sigma_{i\in f^{-1}(j)} A_i\right\}_{j\in J}$$
$$\Pi_f(A) \triangleq \left\{\Pi_{i\in f^{-1}(j)} A_i\right\}_{j\in J}$$

We will prove this for the left adjoint. We need the natural hom-set isomorphism:

$$\frac{\Sigma_f(A) \to B \quad \text{in } \mathbb{P}_J}{A \to f^*(B) \quad \text{in } \mathbb{P}_I}$$

Any morphism $\{g(j)\}_j : \Sigma_f(A) \to B$, say uniformly tracked by d, can be mapped naturally to $\{a \mapsto g(f(i))(i,a)\}_{i\in I} : A \to f^*(B)$, uniformly tracked by $\lambda n\, \ell.\, d \cdot (\overline{f} \cdot n) \cdot \langle n, \ell\rangle$. The other way around, any morphism $\{h(i)\}_i : A \to f^*(B)$, say uniformly tracked by e, can be mapped naturally to $\{(i,a) \mapsto h(i(a))\}_j : \Sigma_f(A) \to B$, uniformly tracked by $\lambda m\, p.\, e \cdot p_0 \cdot m$. Indeed then Σ_f is itself realized.

To be explicit, we can provide the natural transformations thus (both obviously easily tracked as well):

$$\varepsilon_B \triangleq \{\pi_1\}_j : \Sigma_f\big(f^*(B)\big) \to B$$
$$\eta_A \triangleq \{a \mapsto (i,a)\}_i : A \to f^*\big(\Sigma_f(A)\big)$$

□

Lastly, the following will be mainly useful later on, in chapter 6.

Proposition 4.4.9. *Both fibrations* $\mathsf{Fam}(\mathcal{D}\textbf{-Set}) \xrightarrow{\pi_0} \mathcal{D}\textbf{-Set}$ *and* $\mathsf{Fam}(\textbf{Per}) \xrightarrow{\pi_0} \mathcal{D}\textbf{-Set}$ *are* comprehension categories with unit.

Proof. This is really not too hard, and lends its easy construction to the set-theoretical nature of these categories. Each fiber $\mathbb{P}_I$ has a final object $\{\mathbf{1}\}_{i\in I}$, and these are trivially preserved over reindexing. For the right adjoint to the final object functor (sending each J to $\{\mathbf{1}\}_{j\in J}$), we need to satisfy the natural hom-set isomorphism:

$$\frac{\{\mathbf{1}\}_{i\in I} \to \{B_j\}_{j\in J} \quad \text{in } \mathsf{Fam}(\mathbf{C})}{I \to \{\!|\{B_j\}_{j\in J}|\!\} \quad \text{in } \mathbf{C}}$$

This is achieved by setting $\{\!|B|\!\} = \Sigma_{j\in J} B_j$, because every morphism of the shape $\{\mathbf{1}\}_{i\in I} \to B$ corresponds isomorphically to dependent function $f : \Pi_{i\in I} B_{u(i)}$, for some $u : I \to J$, which then again corresponds isomorphically to a morphism $f' : I \to \Sigma_{j\in J} B_j$ (that combines f and u).

For the counit of this adjunction, we have

$$\frac{\begin{array}{ccc} & \{\mathbf{1}\}_{(j,b)\in\Sigma_{j\in J}B_j} & \\ \nearrow & & \searrow^{\varepsilon_B = (\pi_0,\pi_1)} \\ \{\mathbf{1}\}_{i\in I} & \longrightarrow & \{B_j\}_{j\in J} \end{array} \quad \text{in } \mathsf{Fam}(\mathbf{C})}{I \to \{\!|\{B_j\}_{j\in J}|\!\} \quad \text{in } \mathbf{C}}$$

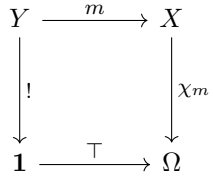
□

4.5 $\mathcal{D}$-Set is not a topos

A topos is a category that can be considered as a universe within which *constructive set theory* can be done. It has all finite limits, a subobject classifier, and exponentials. It can be instructive to note how $\mathcal{D}$-**Set** fails to be a topos (it is, rather, a quasi-topos).

Definition 4.5.1. A *topos* is a category with all finite limits, exponentials, and a subobject classifier, which means an object Ω and a map $\top : \mathbf{1} \to \Omega$ such that for every mono $m : Y \rightarrowtail X$, there is a *classifying map* $\chi_m : X \to \Omega$ such that the following square is a

pullback:



//

Essentially, toposes equate monos with a notion of subobjects, and Ω plays the role of the set **2** in **Set**. Here, we have Y as a subobject of X. Also, morphisms that are both epi and mono are isomorphisms, just as it is in **Set**:

Proposition 4.5.2. *In a topos, every morphism that is both epi and mono, is a isomorphism.*

Proof. Suppose $m : Y \rightarrowtail X$ is a mono, and hence we have the pullback square as above. Now, drawing the diagonal arrow $t = X \xrightarrow{!} \mathbf{1} \xrightarrow{\top} \Omega$, we have both $\chi_m, t : X \to \Omega$, and from the square we know that m is their equalizer. Because m is epi as well, we then know that $\chi_m = t$. Then from the square's commutation we get $Y \xrightarrow{!} \mathbf{1} = Y \xrightarrow{m} X \xrightarrow{!} \mathbf{1}$, and hence $m = \mathsf{id}$ is an isomorphism. □

Proposition 4.5.3. *In $\mathcal{D}$-**Set**, every morphism is (1) epi iff the underlying function is surjective, and (2) mono iff the underlying function is injective.*

Proof. That surjectivity of $|f|$ entails epi-ness of f holds simply because $|f|$ is a set-theoretical function. Now suppose $f : A \to B$ is epi. Then it must be surjective, because if it were not, and say $b \in B$ is not in its range, then we can easily construct $g, h : B \to \nabla\mathbf{2}$ that only disagree on b, invalidating f's epi-ness. For the second item again, the simple set-theoretical underpinning of morphisms in $\mathcal{D}$-**Set** ensure this. □

Proposition 4.5.4. *There are morphisms in $\mathcal{D}$-**Set** that are both epi and mono, but not iso.*

Proof. Because of decidability. For any non-recursively enumerable subset $K \subseteq \mathbb{N}$, define the $\mathcal{D}$-set $\mathsf{Decide}(K) = (\mathbb{N}, \Vdash)$ with

$$n \Vdash k \quad \text{iff} \quad \begin{cases} n = \langle 0, k\rangle & \text{if } k \notin K \\ n = \langle 1, k\rangle & \text{if } k \in K \end{cases}$$

where 0 and 1 are two distinct elements of $\mathcal{D}$. Then, the second projection $\pi_1 \in \mathcal{D}$ trivially tracks a morphism $i : \mathsf{Decide}(K) \to (\mathbb{N}, n \Vdash k \Leftrightarrow n = k)$, whose underlying function is surjective as well as injective, and hence the morphism is epi and mono per the above. But, of course we cannot construct a realizable inverse, because the realizer would have to decide membership of K. □

What we could conclude here, is that the default categorical notions of mono / subobject do not concur with the computational underpinning of morphisms in $\mathcal{D}$-**Set**. There is a known way of 'adding more' objects and morphisms that results in a topos (giving rise to the well-known *effective topos* if $\mathcal{D} = \mathcal{K}_1$), but it is noteworthy that there is also a 'natural' way of defining partial maps and subobjects in such a way that it 'fits well' with the computational aspects of $\mathcal{D}$-**Set**. This will be described in the following section.

4.6 Partial maps and extensional PERs

Recall from the definition of $\mathcal{D}$-sets, that $\mathcal{D}$-set morphisms are necessarily total, even though their realizers may be partial (lemma 3.1.2). In this section we investigate a 'natural'

notion of partial maps and subobjects, that 'fit' better with the computable nature of the underlying PCA, which in this section will be taken to be $\mathcal{K}_1$. A subsequent construction that we briefly outline, is that of *extensional PERs*, although admittedly these constructions play no further role in the theory development of this thesis. Interestingly, the definition of extensional pers hinges on the decidability of equality in $\mathcal{K}_1$.

Convention 4.6.1. *We will assume $\mathcal{D} = \mathcal{K}_1$ for the rest of this chapter.*

First, let us fix some notation for working with $\mathcal{K}_1$. For $d, e, n \in \mathbb{N}$, we have normal Kleene application $\{d\}n$ denoting the d^{th} partial recursive function applied to n. This may be defined ($\{d\}n\!\downarrow$) or undefined ($\{d\}n\!\uparrow$). 'Parallel' search, $d \mathbin{/\!\!/} e$, results in

$$\{d \mathbin{/\!\!/} e\}n = \begin{cases} \{d\}n & \text{if } \{d\}n\!\downarrow \wedge \neg\exists_{m<n}\ \{e\}m\!\downarrow \\ \{e\}n & \text{if } \{d\}n\!\downarrow \wedge \neg\exists_{m\leq n}\ \{d\}m\!\downarrow \\ \textit{undefined} & \text{if } \{d\}n\!\uparrow \wedge \{e\}n\!\uparrow \end{cases}$$

'Parallel' conjunction, $d \wedge e$, results in

$$\{d \wedge e\}n = \begin{cases} 1 & \text{if } \{d\}n\!\downarrow \wedge \{e\}n\!\downarrow \\ \textit{undefined} & \text{else} \end{cases}$$

We also allow ourselves to write ($\mathsf{if}\ t\ \mathsf{then}\ u$) where t, u are (closed) expressions over $\mathcal{K}_1$, noting that if either t or u is undefined, then the entire expression is so as well; and we allow $\{d\}n\!\downarrow$ and $t \stackrel{?}{=} u$ in expressions as well, where its value is considered to be either 1, or 0, or undefined (i.e., we won't be accidentally solving the halting problem with our syntax).

4.6.1 Partial maps and Σ-predicates

The following constructions are originally due to Freyd et al. [1992], but this formulation is taken mainly from Reus [1999].

Definition 4.6.2. In **Per**, a *partial map* $A \overset{f}{\rightharpoonup} B$ *with r.e. domain*, is a partial function $f : |A| \rightharpoonup |B|$ such that it is *realized* by some $e \in \mathbb{N}$ in the following sense:

$$\forall_{n \in \operatorname{dom} A} \begin{cases} \{e\}n\!\downarrow & \Rightarrow f([n]_A) = [\{e\}n]_B \\ f([n]_A)\!\uparrow & \Rightarrow \{e\}n\!\uparrow \end{cases}$$

I.e., for all $n \in \operatorname{dom} A$, we have $\{e\}\!\downarrow$ iff $f([n]_A)$, and if defined, $f([n]_A) = [\{e\}n]_B$ as usual. //

The addition 'with r.e. domain' refers to the property of these maps, that for there is a r.e. subset $W \subseteq \mathbb{N}$, we have $\operatorname{dom} f = (\operatorname{dom} A) \cap W$. In fact, W can be simply taken to be the *domain of definition* of e, i.e. ($n \in W \Leftrightarrow \{e\}n\!\downarrow$).

Although this definition is OK, we will follow Reus [1999] and characterize the maps alternatively, in a more 'categorical' way, as well. The benefit is that this alternative characterization includes the explicit definition of Σ-predicates—the notion of subobject that corresponds with our notion of partial map.

Definition 4.6.3. An $\mathcal{M}$*-partial map* from A to B is a span $A \overset{m}{\leftarrowtail} D \overset{g}{\to} B$ where g is total and $m \in \mathcal{M}$ and $\mathcal{M}$ is an *admissible* set of monos—a collection of monos containing all identities, closed under composition and pullbacks. //

We can indeed define a set $\mathcal{M}$ such that the partial maps with r.e. domain are precisely the $\mathcal{M}$-partial maps, as follows.

Definition 4.6.4. The set of Σ-predicates over a $\mathrm{PER}_{\mathbb{N}}$ object is defined as the following set of $\mathrm{PER}_{\mathbb{N}}$ objects:

$$\Sigma(A) \triangleq \{B \subseteq A \mid \exists_{W \subseteq_{\mathrm{r.e.}} \mathbb{N}} [(\mathrm{dom}\, A) \cap W = \mathrm{dom}\, B]\} \qquad /\!/$$

Proposition 4.6.5. *The following class of monos $\mathcal{M}_p$ in* **Per** *is admissible.*

$$\mathcal{M}_p \triangleq \{A \overset{m}{\rightarrowtail} B \mid m \text{ is an inclusion} \wedge B \in \Sigma(A)\}$$

Proof. Obviously $\mathcal{M}_p$ contains the identities, simply taking $W = \mathbb{N}$. Also, it is closed under composition and pullback, because the intersection of two recursively enumerable sets is again recursively enumerable. (Here, pullback is simply defined as intersection.) $\square$

Theorem 4.6.6. *The category of $\mathrm{PER}_{\mathbb{N}}$ objects with $\mathcal{M}_p$-partial maps is equivalent to the category of $\mathrm{PER}_{\mathbb{N}}$ objects with partial maps with r.e. domain.*

Proof. Let the span $A \overset{m}{\leftarrowtail} D \overset{g}{\to} B$ (for some total, i.e. **Per**-morphism g) define some $\mathcal{M}_p$-partial map. We know that $(\mathrm{dom}\, A) \cap W = \mathrm{dom}\, D$ for some r.e. W. Suppose W is encoded by some $w \in \mathbb{N}$, in the sense that $(n \in W \Leftrightarrow \{w\}n\!\downarrow)$. Suppose d tracks g. Construct the partial map $A \overset{f}{\rightarrowtail} B$ with r.e. domain as:

$$f([n]_A) \triangleq \begin{cases} g([n]_A) & \text{if } n \in W \\ \textit{undefined} & \text{else} \end{cases}$$

This function is realized by

$$\lambda^* m.\ \mathsf{if}\ \{w\}m\!\downarrow\ \mathsf{then}\ \{d\}m$$

The other way around, suppose we have some partial $A \overset{f}{\rightarrowtail} B$ map with r.e. domain, say realized by e. Then e automatically defines the r.e. set W with $(n \in W \Leftrightarrow \{e\}n\!\downarrow)$. Define $D \subseteq A$ as $D \triangleq \{X \in A \mid \exists_{n \in X}\ [\{e\}n\!\downarrow]\} = \{X \in A \mid \forall_{n \in X}\ [\{e\}n\!\downarrow]\}$. We then trivially have our $\mathcal{M}_p$-partial map $A \overset{m}{\leftarrowtail} D$, and we can define $D \overset{g}{\to} A$ as $g \triangleq f \upharpoonright D$, which is still tracked by e.

Hence we have the one-to-one correspondence (for any two $\mathrm{PER}_{\mathbb{N}}$ objects A and B), proving the equivalence.

$$\frac{\mathcal{M}_p\text{-partial map } A \overset{m}{\leftarrowtail} D \overset{g}{\to} B}{\text{partial map } A \overset{f}{\rightarrowtail} B \text{ with r.e. domain}}$$

$\square$

Now that we have partial maps, we will define a way to work with partly unspecified $\mathrm{PER}_{\mathbb{N}}$ objects much like the 'maybe' monad known to functional computer programmers.

Definition 4.6.7. Define

$$\begin{aligned} \mathsf{def}_0 &\triangleq \{n \in X \mid \{n\}0\!\downarrow\} \\ \mathsf{undef}_0 &\triangleq \{n \in X \mid \{n\}0\!\uparrow\} \end{aligned} \qquad /\!/$$

Definition 4.6.8. Define, for any $A \in$ **Per** over $\mathbb{N}$,

$$n \sim_{A_\perp} m \quad \Leftrightarrow \quad \big(\{n\}0\!\downarrow \wedge \{m\}0\!\downarrow \wedge \{n\}0 \sim_A \{m\}0\big) \ \vee\ \big(\{n\}0\!\uparrow \wedge \{m\}0\!\uparrow\big) \qquad /\!/$$

Proposition 4.6.9. *This corresponds to the* **Per** *definition*

$$A_\perp \triangleq \Big(\big\{\{\lambda^* z.n \mid n \in X\} \mid X \in A\big\} \cup \big\{\mathsf{undef}_0\big\}\Big) \setminus \{\varnothing\}$$

Proof. This requires quite some paper, but is easily seen when one notes that $A_\perp$ contains exactly (1) those equivalence classes of natural numbers that denote lambdas that retain, as a whole, an equivalence class of A, and in addition (2) a single equivalence class that is the set of non-value-retaining numbers (undef_0). □

Informally, the trick being applied here, is that we treat elements $n \in \mathbb{N}$ not as values themselves, but 'maybes' that either retain a value ($\{n\}0\!\downarrow$) or do not ($\{n\}0\!\uparrow$). The functor $(-)_\perp$ treats $n, m \in A$ as the 'underlying values, and lifts A to what we could call the 'maybe structure $A_\perp$ for the values of A.

Definition 4.6.10. This in fact defines a functor $(-)_\perp : \mathbf{Per} \to \mathbf{Per}$, if we additionally define

$$(A_\perp \xrightarrow{f} B_\perp) \triangleq [m]_{A_\perp} \mapsto \begin{cases} \mathsf{undef}_0 & \text{if } [m]_{A_\perp} = \mathsf{undef}_0 \\ [\lambda^* z.\{e\}(\{m\}0)]_{B_\perp} & \text{else, where } e \text{ tracks } f \end{cases}$$

…which is of course realized by $\lambda^* m.\lambda^* z.\{e\}(\{m\}0)$. //

Definition 4.6.11. Define the **Per**-morphism $A \xrightarrow{\mathsf{up}} A_\perp$ by

$$[n]_A \mapsto [\lambda^* z.n]_{A_\perp} \quad \text{realized by} \quad k \qquad //$$

If we consider $A_\perp$ to be a 'maybe structure' for the values of A, as remarked above, then up is the morphism that embeds A into $A_\perp$ in the natural way.

Proposition 4.6.12. *In fact, this defines a natural transformation $\mathsf{up} : \mathsf{Id} \Rightarrow (-)_\perp$.*

Proof. By showing that the following square commutes.

$$\begin{array}{ccc} A & \xrightarrow{\quad f \quad} & B \\ \downarrow{\scriptstyle \mathsf{up}_A} & & \downarrow{\scriptstyle \mathsf{up}_B} \\ A_\perp & \xrightarrow{\quad f_\perp \quad} & B_\perp \end{array}$$

$$\begin{aligned} f_\perp\big(\mathsf{up}_A([n]_A)\big) &= f_\perp\big([\lambda^* z.n]_{A_\perp}\big) \\ &= [\lambda^* z.\{e\}(\{\lambda^* q.n\}0)]_{B_\perp} \\ &= [\lambda^* z.\{e\}n]_{B_\perp} \\ &= \mathsf{up}_B([\{e\}n]_B) \\ &= \mathsf{up}_B(f([n]_A)) \end{aligned}$$

□

Definition 4.6.13. Define $\Sigma \triangleq \top_\perp = \{\mathbb{N}\}_\perp = \{\mathsf{def}_0, \mathsf{undef}_0\}$. //

Lemma 4.6.14. *The function $t : \Sigma \to \Sigma$ with $t(\mathsf{def}_0) = \mathsf{undef}_0$ and $t(\mathsf{undef}_0) = \mathsf{def}_0$, is not realized.*

Proof. Suppose it were, say by e. We could then decide the halting problem $K = \{n \mid \{n\}0\!\downarrow\}$ with the program $\curlywedge n.\{n \mathbin{/\!/} \{t\}n\}0$. □

4.6.2 Extensional PERs

We will now introduce a generalization of the 'maybe' structures of the form $A_\perp$, that we introduced in the previous subsection, and of which the elements take on a 'maybe'-like form due to their either retaining a value, or not, via their definedness on 0. Instead of retaining a value only on 0, we will now have the realizers be partial functions from a 'base' set $S \subseteq \mathbb{N}$ to $\mathbb{N}$. The set S can be though of as a certain degree of 'definedness' of the equivalence classes involved.

Definition 4.6.15. An *extensional per* is a per A that can be given some *base* $S \subseteq \mathbb{N}$ for which

$$\forall_{n \in \mathrm{dom}\, A, m \in \mathbb{N}} \big[\, n \sim_A m \;\Leftrightarrow\; \forall_{s \in S}\, \{n\}s \asymp \{m\}s \,\big]$$

Denote the full subcategory of **Per** that has ex-pers as objects, as **ExPer**. //

Remark 4.6.16. *Note the subtle range of the outer quantifier. If if would have ranged over all $n, m \in \mathbb{N}$, then any base S would uniquely determine an extensional per. This is not, however, the definition, and the case is rather that a base and a domain together uniquely determine an extensional per. In fact, $\exists_S \forall_{n,m \in \operatorname{dom} A}[\dots] \not\Rightarrow \exists_S \forall_{n \in \operatorname{dom} A, m \in \mathbb{N}}[\dots] \not\Rightarrow \exists_S \forall_{n,m \in \mathbb{N}}[\dots]$. For the first non-implication, note that even though $n \in \operatorname{dom} A$ and $m \in \mathbb{N}$ may have $\forall_{s \in S}[\{n\}s \asymp \{m\}s]$, there is no way of guaranteeing that $m \in \operatorname{dom} A$. For the second non-implication, the first part of this remark applies. Moreover, the $\exists_S$ addition does not alter this.*

Lemma 4.6.17. *Note that the definition implies, in particular, that for all $n, m \in \mathbb{N}$, whenever $\forall_{s \in S}[\{n\}s \asymp \{m\}s]$, it is also the case that $n \in \operatorname{dom} A \Leftrightarrow m \in \operatorname{dom} A$.*

Example 4.6.18. $\mathbf{N}_\perp$ is an ex-per, with base $\{0\}$. ◿

Proposition 4.6.19. *An ex-per A can be uniquely defined by giving a base and a domain.*

Proof. Let A and B be two ex-pers both having S as base, and the same domain. Then,

$$\begin{aligned} n \sim_A m &\Rightarrow \forall_{s \in S}\ \{n\}s \asymp \{m\}s \\ &\Rightarrow n \sim_B m \qquad \text{(noting that } \operatorname{dom} A = \operatorname{dom} B\text{)} \end{aligned}$$

This applies both ways, and hence $n \sim_A m$ iff $n \sim_B m$, and hence $A = B$. □

Lemma 4.6.20. *Equality in $\mathcal{K}_1$ is decidable: $\forall_{n,m \in \mathbb{N}}[n \neq m \Rightarrow \exists_{d \in \mathbb{N}}\ \{d\}n \neq \{d\}m]$.*

Proposition 4.6.21. *In fact, we can directly find a base of any ex-per A by $S \triangleq \{s \in \mathbb{N} \mid \forall_{n,m \in \mathbb{N}}[n \sim_A m \Rightarrow \{n\}s \asymp \{m\}s]\}$, and unless $A = \varnothing$, this is the unique base for A, which we will denote* $\mathsf{Base}(A)$.

Proof. Let use have an ex-per A, say with base T. Let S be the contestant base for A as defined above.

1. First, note that $T \subseteq S$. For, every $t \in T$ indeed satisfies $\forall_{n,m \in \mathbb{N}}[n \sim_A m \Rightarrow \{n\}t \asymp \{m\}t]$ by T being a base for A.
2. Now, let us prove that S is a base for A. Let $n \in \operatorname{dom} A$ and $m \in \mathbb{N}$. ($\Rightarrow$) If $n \sim_A m$, then indeed for every $s \in S$ we have $\{n\}s \asymp \{m\}s$ by definition of S. ($\Leftarrow$) Suppose $\forall_{s \in S}[\{n\}s \asymp \{m\}s]$. Suppose, for contradiction, that $\neg(n \sim_A m)$. Then, we can find some $t \in T$ for which $\{n\}t \not\asymp \{m\}t$. This t cannot be in S, because we just assumed that $\forall_{s \in S}[\{n\}s \asymp \{m\}s]$. But this contradicts our earlier finding that $T \subseteq S$. Hence, $n \sim_A m$, and indeed S is a base for A.
3. Now, for $S \subseteq T$. Suppose, for contradiction, that $s \in S \setminus T$. Let $n \in \operatorname{dom} A, m \in \mathbb{N}$. But, we know that $\forall_{t \in T}[\{n\}t \asymp \{m\}t] \Rightarrow n \sim_A m \Rightarrow \forall_{s \in S}[\{n\}s \asymp \{m\}s]$ (the other way around holds simply because $T \subseteq S$).

   The only way this is possible, is if there is some $t \in T$ such that s and t are indistinguishable, i.e. $\forall_{n \in \mathbb{N}}[\{n\}s \asymp \{n\}t]$. This is, in fact, certainly possible in PCAs in general. But, in $\mathcal{K}_1$ it is not, because $\mathbb{N}$ is decidable. Hence, we have our contradiction, and can conclude that $S \subseteq T \subseteq S$, there is only one base for A. □

Remark 4.6.22. *(Note how the decidability in $\mathbb{N}$, then, plays a crucial role in the definition of extensional PERs.)*

Informally, in an extensional per, the realizers represent 'partially defined' values, defined 'up to S'. Each of the equivalence classes of an extensional per equates all realizers up to this degree. This will allow us to create chains of ex-pers that are defined up to incremental degrees $S \subseteq S' \subseteq S'' \dots$

Lemma 4.6.23. *For discrete pers A we have*

$$\begin{aligned} m \sim_{\Sigma^A} n \quad &\textit{iff} \quad \forall_{a,b \in \mathrm{dom}\, A}\big[\, a \sim_A b \Rightarrow \{m\}a \sim_\Sigma \{n\}b \,\big] \\ &\textit{iff} \quad \forall_{a \in \mathrm{dom}\, A}\big[\, \{m\}a \sim_\Sigma \{n\}a \,\big] \end{aligned}$$

Definition 4.6.24. On any extensional per A, we can define a partial order $(\leq) \subseteq (\mathrm{dom}\, A)^2$ on its realizers as:

$$n \leq m \quad \Leftrightarrow \quad \forall_{s \in \mathsf{Base}(A)}\big[\, \{n\}s\!\downarrow \Rightarrow \{n\}s = \{m\}s \,\big] \qquad /\!/$$

Lemma 4.6.25. *Suppose $q > p$ in some ex-per A. Then there is a morphism $[b] : A \to \Sigma$ such that*

$$[b]([q]_A) = \mathsf{def}_0 \qquad\qquad [b]([p]_A) = \mathsf{undef}_0$$

Proof. That $q > p$ means that there is some $s \in \mathsf{Base}(A)$ such that $\{q\}s\!\downarrow$ and $\{q\}s \not\simeq \{p\}s$, either because $\{p\}s\!\uparrow$, or because $\{p\}s \neq \{q\}s$. Define $[b] : A \to \Sigma$ using this s as

$$b \triangleq \lambda\!\!\lambda r.\lambda\!\!\lambda z.\mathsf{if}\ [\{r\}s \stackrel{?}{=} \{q\}s]\ \mathsf{then}\ 0$$

Indeed, $\{\{b\}q\}0\!\downarrow$ and hence $\{b\}q \in \mathsf{def}_0$, whereas $\{\{b\}p\}0\!\uparrow$ and hence $\{b\}p\!\downarrow \in \mathsf{undef}_0$. □

Lemma 4.6.26. *Suppose $q > p$ in some ex-per A. Then there is a morphism $[a] : \Sigma \to A$ such that*

$$[a](\mathsf{def}_0) = [q]_A \qquad\qquad [a](\mathsf{undef}_0) = [p]_A$$

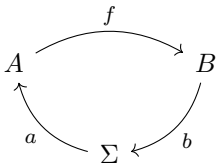
Proof. Define $[a] : \Sigma \to A$ as

$$\begin{aligned} q' &\triangleq q \wedge (\lambda\!\!\lambda z.\{x\}0) \\ a &\triangleq \lambda\!\!\lambda x.\lambda\!\!\lambda s.\{q' \mathbin{/\!/} p\}s \end{aligned}$$

Suppose $x \in \mathsf{undef}_0$, then $\{x\}0\!\uparrow$, and the parallel conjunction of q' will never be defined, and thus q'' neither, and thus $\{a\}x$ denotes a partial recursive function that behaves similarly as p, and hence is in $[p]_A$. Otherwise, if $x \in \mathsf{def}_0$, then $\{x\}0\!\downarrow$, and thus q' and q behave similarly, and $\{a\}x$ as well, because $q > p$ ensures that $\{q\}s$ will always be defined when $\{p\}s$ is, and the parallel search of a prefers the former. Hence $\{a\}x \in [q]_A$. □

Theorem 4.6.27. *Any map between ex-pers is order-preserving.*

Proof. Using the two previous lemma's, and supposing we have some counterexample $[f] : A \to B$, i.e. such that for some $p < q$ in A it is the case that $f(p) > f(q)$, we can construct the following situation:



$$\begin{aligned} &[b \circ f \circ a] : \Sigma \to \Sigma \\ &\{b \circ f \circ a\}(\mathsf{undef}_0) = \mathsf{def}_0 \\ &\{b \circ f \circ a\}(\mathsf{def}_0) = \mathsf{undef}_0 \end{aligned}$$

Clearly this violates lemma 4.6.14, allowing us to solve the halting problem. □

Definition 4.6.28. Define $\mathbf{N} \triangleq \Delta\mathbb{N} = \{\{n\} \mid n \in \mathbb{N}\}$. $/\!/$

Remark 4.6.29. *Note that $\mathbf{N}_\perp = \{\{\lambda^* z.n\} \mid n \in \mathbb{N}\} \cup \{\mathsf{undef}_0\}$ then defines the natural numbers with the 'flat' information ordering.*

Proposition 4.6.30. *Extensional pers A obey a certain separation condition. For $X, Y \in A$:*

$$X \neq Y \quad \Rightarrow \quad \exists f \in {\mathbf{N}_\perp}^E\big[\, f(X) \neq f(Y) \,\big]$$

Proof. By the definition of extensional pers, noting that X and Y are disjunct and hence we have $X \ni n \neq m \in Y$, we can find some $s \in \mathbb{N}$ in its base such that $\{n\}s \not\asymp \{m\}s$. (Informally, this is a degree to which E is not yet specified.) Define f as the map realized by $\lambda^* e.\lambda^* z.\{e\}s$, so that $f([n]_E) = [\lambda^* z.\{n\}s]_{\mathbf{N}_\perp}$ and $f([m]_E) = [\lambda^* z.\{m\}s]_{\mathbf{N}_\perp}$. These are by definition of $\mathbf{N}_\perp$ indeed different equivalence classes. [Note that $\{n\}s \not\asymp \{m\}s$ implies that at least one of the two is indeed defined.] □

In all of the previous, an extensional per may be well though of a being a set of componentwise 'nullable' values over S dimensions (with S its base). There is another way to think about ex-pers though, due to Rosolini [1991], that characterizes them as a collection of subsets of some per X. This is shown in the following theorem.

Theorem 4.6.31. *Any ex-per E is of the form $E \subseteq \Sigma^X$ for some per X, and the other way around, for any per $A \subseteq \Sigma^X$ we can find some ex-per that is isomorphic to it.*

Proof. (⇒) Let E be an ex-per, and let $S = \mathsf{Base}(E)$. Then, whenever $n \sim_E m$, we have $\forall_{s \in \mathsf{Base}(E)}\, [\{m\}s \asymp \{n\}s]$, which implies $\forall_{s \in S}\, [\{m\}s \sim_\Sigma \{n\}s]$ (for, given any such s, either both sides are undefined, and then undefined on 0 too, or equal, and then in particular Σ-equivalent as well). This last statement is equivalent to $m \sim_{\Sigma^{\Delta S}} n$ over lemma 4.6.23. Hence, every equivalence class of E corresponds to a function of $\Sigma^{\Delta S}$ (to wit, the one realized by m and n).

(⇐) Suppose A is some per with $A \subseteq \Sigma^X$ for some per X. This is to say, A can be though of as a collection of subsets of X, each encoded by some $f \in \operatorname{dom} A$. We will construct the ex-per E. Set $\mathsf{Base}(E) = S = \operatorname{dom} X$. For each $f \in \operatorname{dom} A$, let e_f denote the unique $e \in \mathbb{N}$ such that

$$\forall_{x \in \operatorname{dom} X} \big[\, \big(\{e\}x = 0 \Leftrightarrow \{f\}x \in \mathsf{def}_0\big) \;\wedge\; \big(\{e\}x\!\uparrow \Leftrightarrow \{f\}x \in \mathsf{undef}_0\big) \,\big]$$

In fact, e_f is not only unique to f, but also to $[f]_A$, because whenever $f \sim_A g$, they must agree on their value on each $x \in \operatorname{dom} X$ simply by the tracking requirement of pers. So, let us write $e_{[f]}$. Then, define the domain of E as

$$\operatorname{dom} E \triangleq \{e_{[f]} \mid [f] \in A\} = \{e_f \mid f \in \operatorname{dom} A\}$$

That $A \cong E$ is clear from the construction. Moreover, the construction of e_f from from f can indeed be done within $\mathcal{K}_1$, and hence we can track the constructed isomorphism with e. □

5 A realizability model for semantic CC

Now we have enough materials to build a realizability model for CC. We've seen that the category of D-sets admits an internal category **Per**, and that this internal category will allow us to model dependent types and polymorphism, and hence we essentially simply interpret CC along the following lines:

- Types are D-sets, and *small types*, in particular, will be modest D-sets. (We will in fact force them to be PER_D objects.)
- Contexts are just products of types, as before.
- Terms are D-set elements, and hence terms of small types are equivalence classes of their type's realizability predicate.

However, a few technical problems prevent us from writing this definition down in just a few lines. The first is a common one, namely that, due to the fact that CC types are closed under β-conversion, multiple derivations may exist for any particular object-denoting judgment. The typical solution to this problem is to first give an *a priori* interpretation to the fill *a priori* syntax, and then include a correctness proof, that all such derivations for the same judgment lead to the same interpretation. [Alternatively, one might define an interpretation on *entire derivations*, and then prove that interpretations of derivations of the same judgment concur.]

The second problem is specific to our case, and has to do with the fact that in the equivalence of **Per** and **Mod**, their objects are only isomorphic, and not exactly equal. This problem, for us, mainly manifests in the exponential object. Take for instance the type of the polymorphic identity, for which we have

$$\mathsf{Proof}(\forall a^{\mathsf{Prop}}.\, a \Rightarrow a) \;\equiv\; \Pi_{a:\mathsf{Prop}}\, \mathsf{Proof}(a \Rightarrow a) \;\equiv\; \Pi_{a:\mathsf{Prop}}\, \mathsf{Proof}(a) \to \mathsf{Proof}(a)$$

The leftmost of these types will be interpreted as an element of the internal category **Per** (lifted to $\mathcal{D}$-**Set**), and hence will be a **Per** exponential, whereas the other two will be interpreted as normal $\mathcal{D}$-**Set** exponentials. Elements of the former will be equivalence classes (denoting a D-set morphism), whereas elements of the latter will already be D-set morphisms. Although isomorphic, we are bound to interpreting them to distinct object in **Mod**. But we cannot allow ourselves to interpret ($\equiv$) as isomorphism, because it would yield too weak a notion of 'model'.

The solution, then, is to 'canonicalize' interpreted types (and contexts, subsequently) to be always either non-modest, or else in **Per** (and never in **Mod** $\setminus$ **Per**). This leads to using the *per-specified* versions Π^{per}, $\Pi^{\mathsf{per}}_{(-)}$, $\mathsf{curry}^{\mathsf{per}}$, and $\mathsf{eval}^{\mathsf{per}}$ at the right places.

Definition 5.0.1. An *a priori* partial interpretation $[\![-]\!]$ from the pre-syntax of (semantic) CC *object-denoting judgments* to various realizability structures is defined, to be found in figure 7. //

Definition 5.0.2. A D-set A (resp. an element a thereof) is called *canonical* iff

- if A is modest, and then $A \in$ **Per**
  (resp. $a \in \mathcal{P}(D)$);
- if $A = B \times C$ or $A = \sigma(B, C)$, then B and C are (component-wise) canonical
  (resp. $a \in \mathcal{P}(D) \times \mathcal{P}(D)$);
- if $A = C^B$ or $A = \pi(B, C)$, then C is (component-wise) canonical
  (resp. $\mathrm{dom}\, a \subseteq \mathcal{P}(D)$). //

Signature

$$[\![\Gamma]\!] \in \mathcal{D}\textbf{-Set} \qquad [\![\Gamma \vdash A]\!] : [\![\Gamma]\!] \to \mathcal{D}\textbf{-Set} \qquad [\![\Gamma \vdash t : A]\!] \in \Pi_{\gamma \in [\![\Gamma]\!]} [\![\Gamma \vdash A]\!]_\gamma$$

Interpretation of contexts

$$\begin{aligned} [\![\cdot]\!] &\triangleq \top \\ [\![\Gamma, x^A]\!] &\triangleq \Sigma_{\gamma \in [\![\Gamma]\!]} [\![\Gamma \vdash A]\!]_\gamma \end{aligned}$$

Interpretation of types

$$\begin{aligned} [\![\Gamma \vdash \mathsf{Prop}]\!]_\gamma &\triangleq \mathcal{U} \quad \big(= \nabla\textbf{PerQ}\big) \\ [\![\Gamma \vdash \mathsf{Proof}(t)]\!]_\gamma &\triangleq \iota\big([\![\Gamma \vdash t]\!]_\gamma\big) \\ [\![\Gamma \vdash \Pi_{x:A} B]\!]_\gamma &\triangleq f_{[\![\Gamma]\!]}\big([\![\Gamma \vdash A]\!], [\![\Gamma, x^A \vdash B]\!]\big)_\gamma \\ &\qquad \text{where } f \triangleq \begin{cases} \pi^{\mathsf{per}}_{(\,-\,)} & \text{if range}\big([\![\Gamma, x^A \vdash B]\!]\big) \subseteq \textbf{Mod} \\ \pi_{(\,-\,)} & \text{else} \end{cases} \end{aligned}$$

Interpretation of terms

$$\begin{aligned} [\![\Gamma, x_i^A, \Delta \vdash x_i : A]\!]_\gamma &\triangleq \mathsf{proj}_i^n(\gamma) \\ [\![\Gamma \vdash \lambda x^A.t : \Pi_{x:A} B]\!]_\gamma &\triangleq f_{[\![\Gamma]\!],[\![\Gamma \vdash A]\!],[\![\Gamma, x^A \vdash B]\!]}([\![\Gamma, x^A \vdash t : B]\!])_\gamma \\ &\qquad \text{where } f = \begin{cases} \mathsf{curry}^{\mathsf{per}} & \text{if range}\big([\![\Gamma, x^A \vdash B]\!]\big) \subseteq \textbf{Mod} \\ \mathsf{curry} & \text{else} \end{cases} \\ [\![\Gamma \vdash \mathsf{app}_{\Pi(x:A)B}(t, s) : B']\!]_\gamma &\triangleq f_{[\![\Gamma]\!],[\![\Gamma \vdash A]\!],[\![\Gamma, x^A \vdash B]\!]}([\![\Gamma \vdash t : \Pi(x : A).B]\!], [\![\Gamma \vdash s : A]\!])_\gamma \\ &\qquad \text{where } f = \begin{cases} \mathsf{eval}^{\mathsf{per}} & \text{if range}\big([\![\Gamma, x^A \vdash B]\!]\big) \subseteq \textbf{Mod} \\ \mathsf{eval} & \text{else} \end{cases} \\ [\![\Gamma \vdash \forall x^A.t : \mathsf{Prop}]\!]_\gamma &\triangleq f_{[\![\Gamma]\!]}\big([\![\Gamma \vdash A]\!], [\![\Gamma, x^A \vdash t : \mathsf{Prop}]\!]\big)_\gamma \\ &\qquad \text{where } f \triangleq \begin{cases} \pi^{\mathsf{per}}_{(\,-\,)} & \text{if range}\big([\![\Gamma, x^A \vdash t]\!]\big) \subseteq \textbf{Mod} \\ \pi_{(\,-\,)} & \text{else} \end{cases} \end{aligned}$$

(Implicitly, we only assume expressions to be defined if all sub-expressions are defined.)

Figure 7: Interpretation of semantic CC.

Lemma 5.0.3. *For any derivable context Γ, type A, and term t,*

1. $[\![\Gamma]\!]$ *is canonical;*
2. $[\![\Gamma \vdash A]\!]$ *is canonical;*
3. $[\![\Gamma \vdash t : A]\!]$ *is canonical.*

Proof. By simultaneous induction on the structure of the derivation, by force of the use of the per-specified versions of the constructions employed. □

Proposition 5.0.4. *The interpretations (and sub-interpretations) of all derivable object-denoting judgments sub-interpretations are defined, and furthermore derivable convertibility judgments are equationally sound. This means that:*

1. *if Γ is derivable, then $[\![\Gamma]\!]$ is defined;*
2. *if $\Gamma \equiv \Delta$ is derivable, then $[\![\Gamma]\!] = [\![\Delta]\!]$;*
3. *if $\Gamma \vdash A$ is derivable, then $[\![\Gamma]\!]$ is defined, and $[\![\Gamma \vdash A]\!]$ is defined;*
4. *if $\Gamma \vdash A \equiv B$ is derivable, then $[\![\Gamma]\!]$, $[\![\Gamma \vdash A]\!]$, and $[\![\Gamma \vdash B]\!]$ are defined, and moreover $[\![\Gamma \vdash A]\!] = [\![\Gamma \vdash B]\!]$;*
5. *if $\Gamma \vdash t : A$ is derivable, then $[\![\Gamma]\!]$, $[\![\Gamma \vdash A]\!]$, and $[\![\Gamma \vdash t : A]\!]$ are defined;*
6. *if $\Gamma \vdash s \equiv t : A$ is derivable, then $[\![\Gamma]\!]$, $[\![\Gamma \vdash A]\!]$, $[\![\Gamma \vdash s : A]\!]$ and $[\![\Gamma \vdash t : A]\!]$ are defined, and $[\![\Gamma \vdash s : A]\!] = [\![\Gamma \vdash t : A]\!]$.*

Proof. The proof is by simultaneous induction on the (sizes of the) structure of the involved derivation. (1), (2), and (3) are easily seen to be true. □

The interpretation is consistent

Proposition 5.0.5. *The interpretation is consistent, in the sense that the model admits an uninhabited 'false' type.*

Proof. We can syntactically define the small type $\mathsf{False} \triangleq \mathsf{Proof}(\forall a^{\mathsf{Prop}}.\, a)$, and then compute that the interpretation is empty:

$$\begin{aligned}
[\![\Gamma \vdash \mathsf{False}]\!]_\gamma &= [\![\Gamma \vdash \forall a^{\mathsf{Prop}}.\, a]\!]_\gamma \\
&= \pi^{\mathsf{per}}_{[\![\Gamma]\!]}\big([\![\Gamma \vdash \mathsf{Prop}]\!], [\![\Gamma, a^{\mathsf{Prop}} \vdash a]\!]\big)_\gamma \\
&= \Pi^{\mathsf{per}}_{A \in \nabla \mathbf{PerQ}} [\![\Gamma, a^{\mathsf{Prop}} \vdash a]\!]_{(\gamma, A)} \\
&\cong \bigcap_{A \in \mathbf{PerQ}} A \qquad \text{(by thm. 4.3.2)} \\
&= \varnothing
\end{aligned}$$

□

Fibrational structure

The following proposition may be taken to serve as a 'sanity check' for the definition of $\mathsf{Fam}(\mathbf{Per})$ as given in section 4.4 (which notably includes the notion of *uniform tracking*).

Proposition 5.0.6. *For derivable $[\Gamma \vdash a : \mathsf{Prop}]$, $[\Gamma \vdash b : \mathsf{Prop}]$, and $[\Gamma \vdash f : \mathsf{Proof}(a) \to \mathsf{Proof}(b)]$, indeed the interpretation of $[\![\Gamma \vdash f : \mathsf{Proof}(a) \to \mathsf{Proof}(b)]\!] \in \mathbb{P}_{[\![\Gamma]\!]}$ is (essentially) a uniformly tracked morphism in the fiber $\mathbb{P}_{[\![\Gamma]\!]}$, from $[\![\Gamma \vdash a : \mathsf{Prop}]\!] \in \mathbb{P}_{[\![\Gamma]\!]}$ to $[\![\Gamma \vdash b : \mathsf{Prop}]\!] \in \mathbb{P}_{[\![\Gamma]\!]}$.*

Proof. Let $I = [\![\Gamma]\!]$. Note that the interpretations of the small types are indeed objects in

$\mathbb{P}_I$, i.e. they have the form $\{A_\gamma\}_{\gamma\in I} \,/\, \{B_\gamma\}_{\gamma\in I}$.

$$A = [\![\Gamma \vdash a : \mathsf{Prop}]\!] \in \Pi_{\gamma\in I}\mathcal{U}$$
$$B = [\![\Gamma \vdash b : \mathsf{Prop}]\!] \in \Pi_{\gamma\in I}\mathcal{U}$$

The function space is of course not equal to a hom-set of $\mathbb{P}_{[\![\Gamma]\!]}$, but it is isomorphic to it. If we interpret the function, we get

$$\begin{aligned} g = [\![\Gamma \vdash f : \mathsf{Proof}(a) \to \mathsf{Proof}(b)]\!] &\in \Pi_{\gamma\in I}[\![\Gamma \vdash \mathsf{Proof}(a) \to \mathsf{Proof}(b)]\!]_\gamma \\ &= \Pi_{\gamma\in I}\pi_I\big([\![\Gamma \vdash \mathsf{Proof}(a)]\!], [\![\Gamma \vdash \mathsf{Proof}(b)]\!]\big)_\gamma \\ &= \Pi_{\gamma\in I}\Pi^{\mathsf{per}}_{a\in A_\gamma} B_\gamma \\ &\cong \Pi_{\gamma\in I}\Pi_{a\in A_\gamma} B_\gamma \\ &= \mathsf{Tracked}\big(\Pi_{\gamma\in|I|} \,\big|\, \Pi_{a\in A_\gamma} B_\gamma\big|\big) \end{aligned}$$

I.e., there is some $e \in D$ such that, for all $\gamma \in |[\![\Gamma]\!]|$ and $n \Vdash_{[\![\Gamma]\!]} \gamma$, we have $e\cdot n \Vdash_{(\Pi_{a\in A_\gamma} B_\gamma)} g$. Hence, we essentially have a morphism on our hand, that does not reindex over $[\![\Gamma]\!]$, and is tracked by e. □

A note on equality

Definition 5.0.7. A *propositional equality type* E assigns to every $\mathcal{D}$-set A, a family E_A of pers indexed over $A \times A$, is called *normal* if $A_E(a,b) \neq \varnothing \Leftrightarrow a = b \in A$, and is called *trivial* if it is normal and contains exactly one element when non-empty. //

Proposition 5.0.8. *The model admits a trivial propositional equality type.*

Proof. We can simply set

$$E_A(a,b) \triangleq \begin{cases} \top = \{\mathcal{D}\} & \text{if } a = b \\ \bot = \varnothing & \text{else} \end{cases}$$

Note that we can also provide a (trivially tracked) constructor refl. Less ad-hoc, this type (and a constructor) flow automatically from the notion of inductive types that can be provided for the model, see the following chapter. □

Corollary 5.0.9. *As a result, the model admits the uniqueness of identity proofs (UIP) axiom, as well as Streicher's K.* ◿

Proposition 5.0.10. *However, we can also tweak the model a bit, to make the above equality type inexistent.*

Proof. If we pick our pairing operation $\langle -,-\rangle$ such that $\langle 0,0\rangle = 0$, pick our enumeration of partial recursive functions such that $0 \cdot n = 0$, define $\mathbf{PerQ}_0 \triangleq \{A \in \mathbf{PerQ} \mid 0 \sim_A 0\}$, and use $\mathcal{U}_0 = \nabla\mathbf{PerQ}_0$ in the interpretation of Prop, then we have forced every interpreted per to always have an equivalence class $[0]$. Although makes our model inconsistent, it also makes the above equality type impossible. □

Proposition 5.0.11. *Any function $C : A \to \mathcal{U}$ is vacuously tracked (for any $\mathcal{D}$-set A), and hence we have (for any derivable $[\Gamma \vdash A \to \mathsf{Prop}]$)*

$$\begin{aligned} [\![\Gamma \vdash A \to \mathsf{Prop}]\!]_\gamma &= \mathcal{U}^{[\![\Gamma\vdash A]\!]_\gamma} \\ &= \mathbb{P}_{[\![\Gamma\vdash A]\!]_\gamma} \qquad (\text{in } \mathsf{Fam}(\mathbf{Per})) \end{aligned}$$

Proof. Because $\mathcal{U} = \nabla\mathbf{PerQ}$ is fully realized, the tracking requirement is vacuous. □

Proposition 5.0.12. *The interpretation satisfies*

$$[\![\Gamma, x^A \vdash x \overset{L}{=} x]\!] = [\![\Gamma, x^A \vdash \mathsf{PolyId}]\!]$$

Proof. Because both types are of the form $\Pi_{C:X} S$, where the interpretation of X is fully realized, and S is a small type, and hence interpreted in **Per**—this ensures us that the entire type is interpreted as an intersection of pers. In particular, the range of C is all of **Per** in both cases, even though Leibniz' C technically ranges over families of pers, due to proposition 5.0.11. Finally, type-freeness of the interpretation guarantees us that in both cases, the realizers are just of the form $\mathbf{\Lambda} n\, m.m$, no matter the types involved. Hence, we have the equality. □

Hence, we see that the low-level extensionality combined with the type-freeness of the model results in the fact that Leibniz equality is only non-trivial, in the case that we have multiple elements in the interpretation of the polymorphic identity type. This if course not the case, and is very unnatural to 'force' into the model.

Remark 5.0.13. *It is interesting to note that this equation also holds in Stefanova and Geuvers [1995] (see the computation below), where we also have a type-free interpretation, and where types also only play the role of 'specifications of behavior', arguably even more clearly so than in our case. However, their model is intensional, where ours is extensional, and hence (although still unnatural) one can in principle provide as many polymorphic identities as one would wish.*

$$\begin{aligned}
t \in [\![\mathsf{PolyId}]\!]_{\xi,\rho} \quad &\text{iff} \quad t \in \bigcap_{A \in \mathcal{U}^*} \Pi_\star\, m \in \overline{\mathbf{A}}.\, [\![A \to A]\!]^*_{\xi[A:=A],\rho[A:=m]} \\
&\text{iff} \quad \forall_{A \in \mathcal{U}^*} \forall_{m \in \overline{\mathbf{A}}} \forall_{a \in A} \big[\, t \cdot m \cdot a \in A \,\big] \\
&\Updownarrow \\
t \in [\![x \overset{L}{=}_A x]\!]_{\xi,\rho} \quad &\text{iff} \quad t \in \bigcap_{C \in \left(\widetilde{\prod}^{a \in [\![A]\!]^*.\mathcal{U}^*}\right)} \Pi_{(\Pi_{x \in A} \star)}\, m_2 \in \big(\Pi_{(\Pi_{x \in A} \star)} m' \in [\![A]\!]^*.\overline{\mathbf{A}}\big). \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\![Cx \to Cx]\!]_{\xi[C:=C],\rho[C:=m_2]} \\
&\text{iff} \quad \forall_{C \in \left(\widetilde{\prod}^{a \in [\![A]\!]^*.\mathcal{U}^*}\right)} \forall_{m_2 \in \left(\Pi_k m' \in [\![A]\!]^*.\overline{\mathbf{A}}\right)} \forall_{c \in C(\!|x|\!)_\rho} \big[\, t \cdot m_2 \cdot c \in C(\!|x|\!)_\rho \,\big]
\end{aligned}$$

(⇓) *follows directly by the substitution* $A := C(\!|x|\!)_\rho, m := m_2, a := c$. *Crucial though, is that the lower* m_2 *is indeed contained in* $\overline{\mathbf{A}}$, *but this is ensured by (item 2 of) their definition of 'sufficient subset', which, informally, is any set of PCA terms that suffices to 'stand in' as syntactic descriptions of types.* (⇑) *already follows synthetically, without recourse to model specifics: suppose we have* A, m, a. *Then we can define* $C(-) \triangleq A, m_2 := \Lambda x.m$ *and* $c := a$, *and the conclusion follows.*

Proposition 5.0.14. *The interpretation admits definitional, as well as Leibniz function extensionality, i.e.*

$$[\![\Gamma, x^A \vdash fx : B]\!] = [\![\Gamma, x^A \vdash gx : B]\!] \;\Rightarrow\; [\![\Gamma \vdash f : \Pi_{x:A} B]\!] = [\![\Gamma \vdash g : \Pi_{x:A} B]\!]$$
$$[\![\Gamma, x^A \vdash fx \overset{L}{=}_B gx]\!] \neq \varnothing \;\Rightarrow\; [\![\Gamma \vdash f \overset{L}{=}_{\Pi(x:A).B} g]\!] \neq \varnothing$$

Proof. The first follows from a simple calculation akin to proposition 3.3.1. The second uses the fact that the family C over which the Leibniz equality type ranges is unrestricted (proposition 5.0.11), and hence one can use the typical proof method that one would also use to prove, e.g. Leibniz equality's symmetry, but then in the model itself. □

6 Inductive types

In this chapter, we will show how **Per** is contains all initial algebras for non-dependent as well as dependent polynomial functors, which shows that **Per** is closed under the formation of W-types (with realizable signature), as well as a dependent variant thereof. These can be used to describe inductive types, and we will conclude the chapter with a description of how to interpret a certain class of dependent inductive types, having constructors that do not cross-reference each other. This is done by way of an elegant description in terms of (F, G)-dialgebras, and then a reduction to a dependent polynomial, which in turn has an initial algebra in **Per**.

It must be noted that, in fact, a way stronger result holds for **Per**: Stekelenburg [2011] has shown that **Per** is closed under arbitrary *realizable* functors, so that the category is said to be 'algebraically compact'. We nonetheless 'get out hands dirty', and obtain the initial algebras in a more 'low-level' fashion.

6.1 For non-dependent polynomials

Per definition 2.4.8, and knowing that **Per** has dependent sums and inverse images, a non-dependent polynomial functor $P_f : \mathbf{Per} \to \mathbf{Per}$, for a certain *signature* $f : B \to A$ in **Per**, can be described as

$$P_f(X) = \Sigma_{a \in A} X^{f^{-1}(a)}$$

Note that it maps morphisms $[d]$ to morphisms $[\lambda p.\, \langle p_0, d \circ p_1 \rangle]$, and hence the functor is easily realized itself.

In this section we will prove that **Per** contains all initial P_f-algebras, i.e. pers of well-founded trees.

Proposition 6.1.1. *For a given per morphism $f : B \to A$, the non-dependent polynomial functor P_f can be characterized as having*

$$\langle i, d \rangle \sim_{P_f(C)} \langle j, e \rangle \quad \textit{iff} \quad i \sim_A j \wedge \forall_{n,m} \big[\, n \sim_B m \wedge (\overline{f} \cdot n) \sim_A i \Rightarrow (d \cdot n) \sim_C (e \cdot m) \,\big]$$

Proof. Unfolding definitions. □

There are two fixed-point theorem that are often used to show that initial algebras exist. Tarski's fixed-point theorem applies to cases where one has a complete lattice and a monotonic function over it, and it not necessarily constructive. A more generally applicable, and constructive variant is Kleene's fixed-point theorem, which applies to *directed-complete partial orders* (DCPOs) and Scott-continuous functions (which preserve suprema of directed sets) thereover.

Now, the structure $(\mathbf{Per}, \subseteq, \bigcup, \bigcap)$ is not a complete lattice, as it can easily be seen that unions of pers are not always pers again (transitivity can be broken readily). One might come up with 'smarter' joins $(\bigvee) : \mathcal{P}(\mathbf{Per}) \to \mathbf{Per}$ and accompanying orders $(\subseteq)$ (for instance, $A \leq B$ iff $\exists i : A \rightarrowtail B$) but simple cardinality issues will prevent these from giving a complete lattice. However, it is a DCPO:

Lemma 6.1.2. *The structure $(\mathbf{Per}, \subseteq, \bigcup, \varnothing)$ is a directed-complete partial order (DCPO).*

Proof. I.e. **Per** is partially ordered by inclusion, and every directed set has a supremum. Suppose $\mathcal{X}$ is a family of pers. Then $\bigcup\mathcal{X}$ is a per too, for if $d \sim_{(\bigcup\mathcal{X})} e \sim_{(\bigcup\mathcal{X})} f$, then we can be find $X, Y \in \mathcal{X}$ with $d \sim_X e \sim_Y f$. Because $\mathcal{X}$ is directed, we have some $Z \in \mathcal{X}$ such that $X, Y \subseteq Z$. Then, $d \sim_Z e \sim_Z f$ and by transitivity $d \sim_Z f$ and hence $d \sim_{(\bigcup\mathcal{X})} f$. □

A further complication arises when the signature f describes a infinitely branching tree (in breadth), for then (as far as the author understands) P_f cannot be Scott-continuous. For finitely branching tree signatures, though, P_f is easily seen to be Scott-continuous:

Proposition 6.1.3. *If $f : B \to A$ is such that every $f^{-1}(a)$ is finite (i.e. is a finite amount of equivalence classes), then P_f is Scott-continuous, i.e. preserves limits of directed sets.*

Proof. The crucial thing here, is of course that morphisms $[\ell] : f^{-1}(a) \to \bigcup\mathcal{X}$ are characterized by $n \sim_{f^{-1}(a)} m \Rightarrow \exists_{X\in\mathcal{X}}\big[\,(\ell\cdot n) \sim_X (\ell\cdot m)\,\big]$, while morphisms $[\ell] \in \bigcup_{X\in\mathcal{X}} X^{f^{-1}(a)}$ are characterized by $\exists_{X\in\mathcal{X}}\big[\,n \sim_{f^{-1}(a)} m \Rightarrow (\ell\cdot n) \sim_X (\ell\cdot m)\,\big]$. However, if $f^{-1}(a)$ is always finite, then we can take $X = \bigcup_{[n]\in f^{-1}(a)} X_{[n]}$ where $X_{[n]}$ is guaranteed by the first characterization, which is still in $\mathcal{X}$ due to its directedness, for the second characterization. Then, we can compute

$$\begin{aligned} P_f\big(\bigcup\mathcal{X}\big) &= \sum_{a\in A} \big(\bigcup\mathcal{X}\big)^{f^{-1}(a)} \\ &= \sum_{a\in A} \bigcup_{X\in\mathcal{X}} X^{f^{-1}(a)} && \text{(as described)} \\ &= \bigcup_{X\in\mathcal{X}} \sum_{a\in A} X^{f^{-1}(a)} \\ &= \bigcup_{X\in\mathcal{X}} P_f(X) \end{aligned}$$ □

Corollary 6.1.4. *If $f^{-1}(a)$ is always finite, we can readily apply Kleene's fixed-point theorem, to see that every functor P_f admits an initial P_f-algebra, which will be*

$$\mathsf{W}_f = \bigcup_{n\in\mathbb{N}} {P_f}^n(\varnothing) \qquad \text{with} \qquad w = \mathsf{id} : P_f(\mathsf{W}_f) \to \mathsf{W}_f$$

◿

However, **Per** also admits initial algebras for P_f describing infinitely branching trees. As the author does now know P_f to be Scott-continuous in this case, a new proof is required. The following proof is based on the one given in Bauer [2000], but slightly reworked so that the construction with Q (used in the unicity part of the UP) in the proof is clearer.

Theorem 6.1.5. *For every per morphism $f : B \to A$, the initial P_f-algebra exists in* **Per**.

Proof. We simply apply the same construction as in Kleene's fixed-point theorem, but then prove the universal property by hand. First, construct the following sequence of inclusions:

$$\varnothing \subseteq P_f(\varnothing) \subseteq P_f\big(P_f(\varnothing)\big) \subseteq \cdots \subseteq {P_f}^n(\varnothing) \subseteq \cdots \subseteq \textstyle\bigcup_n {P_f}^n(\varnothing) \subseteq P_f\big(\bigcup_n {P_f}^n(\varnothing)\big) \subseteq \cdots$$

The inclusions can be proven to hold by (transfinite) induction. The first inclusion is of course trivial. Now suppose (IH) that we already have $C \subseteq P_f(C)$ for some C, and let $\langle i, d\rangle \sim_{P_f(C)} \langle j, e\rangle$. For $\langle i, d\rangle \sim_{P_f(P_f(C))} \langle j, e\rangle$ to hold, we need it for $n \sim_B m$ with $(\overline{f}\cdot n) \sim_A i$ to hold that $(d\cdot n) \sim_{P_f(C)} (e\cdot m)$. But, we already know that $(d\cdot n) \sim_C (e\cdot m)$, and then we can just apply (IH). Hence $P_f(C) \subseteq P_f\big(P_f(C)\big)$. That limits are pers again follows from the directedness of the preceding part of the sequence.

Hence, the entire sequence is directed, and we can find the least fixed-point of P_f by taking the supremum of the sequence. (Note that Lambek's Lemma is in this case indeed not

even required, as we're working with a set-theoretical fixed-point.)

$$\mathsf{W}_f \triangleq \bigcup_{\gamma} {P_f}^{\gamma}(\varnothing)$$

To show that this is a P_f-algebra, we can simply take the identity $\mathsf{W}_f : \mathsf{W}_f \to \mathsf{W}_f$ as the structure map, because W_f is the fixed-point of P_f. To show that it has the universal property, let $(S, [s])$ be some P_f-algebra, and we must satisfy:

$$\begin{array}{ccc} P_f(\mathsf{W}_f) & \xrightarrow{[\lambda x.x]} & \mathsf{W}_f \\ \Big\downarrow{\scriptstyle [\lambda p.\langle p_0, u \circ p_1\rangle]} & & \vdots{\scriptstyle [u]} \\ P_f(S) & \xrightarrow{[s]} & S \end{array}$$

(The u is this diagram is then said to be defined by recursion, with specification s.)

Unicity. The following diagram illustrates the morphisms and objects involved in this part of the proof:

$$\begin{array}{ccccccc} Q & \xrightarrow{\subseteq} & P_f(\mathsf{W}_f) & \xrightarrow{=} & \mathsf{W}_f & \xrightarrow{\subseteq} & Q \\ & & {\scriptstyle P_f[u]}\big\downarrow\big\downarrow{\scriptstyle P_f[v]} & & {\scriptstyle [u]}\big\downarrow\big\downarrow{\scriptstyle [v]} & & \\ & & P_f(S) & \xrightarrow{[s]} & S & & \end{array}$$

Suppose we have two such P_f-morphisms $[u]$ and $[v]$.

Consider the equalizer of $P_f[u]$ and $P_f[v]$, which is the per $Q \subseteq P_f(\mathsf{W}_f)$ with

$$Q \triangleq \left\{ \left[\langle i, d\rangle\right]_{P_f(\mathsf{W}_f)} \mid \left[\langle i, u \circ d\rangle\right]_{P_f(S)} = \left[\langle i, v \circ d\rangle\right]_{P_f(S)} \right\}$$

First, note that every $[\langle i, d\rangle] \in Q$ is sent to the same element over $[u]$ and $[v]$ as well, which means that $[u] = [v] : Q \to S$. Stated differently, elements that agree over $P_f[u]$ and $P_f[v]$, also agree over $[u]$ and $[v]$.

$$\begin{aligned} [u]\big([\langle i, d\rangle]_Q\big) &= [u \cdot \langle i, d\rangle]_S \\ &= [s \cdot \langle i, u \circ d\rangle]_S && \text{(by commutation of } [u]) \\ &= [s \cdot \langle i, v \circ d\rangle]_S && \text{(by definition of } Q, \text{ applied over } [s]) \\ &= [v \cdot \langle i, d\rangle]_S && \text{(by commutation of } [v]) \\ &= [v]\big([\langle i, d\rangle]_Q\big) \end{aligned}$$

Second, we show that in fact $\mathsf{W}_f \subseteq Q$, so that we can can conclude that $[u] = [v] : \mathsf{W}_f \to S$. We prove $\mathsf{W}_f \subseteq Q$ by showing that $P_f(Q) \subseteq Q$. Suppose $[\langle i, d\rangle]_{P_f(Q)} \in P_f(Q)$. We want to show that this equivalence class is in Q as well, i.e. is sent to the same element over $P_f[u]$ and $P_f[v]$:

$$P_f[u]\big([\langle i, d\rangle]_{P_f Q}\big) = [\langle i, u \circ d\rangle]_{P_f S} \overset{\text{want}}{=} [\langle i, v \circ d\rangle]_{P_f S} = P_f[v]\big([\langle i, d\rangle]_{P_f Q}\big)$$

By definition of P_f, this means proving that for all $n \in \operatorname{dom} B$ such that $\overline{f} \cdot n \sim_A i$, we have $[(u \circ d) \cdot n]_S = [(v \circ d) \cdot n]_S$. But, applying the definition of P_f to $[\langle i, d\rangle]_{P_f(Q)}$,

we know that for all such n, we have $[d \cdot n]_Q \in Q$, and over our previous finding that $[u] = [v] : Q \to S$, we indeed have

$$[(u \circ d) \cdot n)]_S = [u \cdot (d \cdot n)]_S = [u]\big([d \cdot n]_Q\big) = [v]\big([d \cdot n]_Q\big) = [v \cdot (d \cdot n)]_S = [(v \circ d) \cdot n)]_S$$

Stated differently, subsequently constructed elements agree again over $P_f[u]$ and $P_f[v]$.

This then concludes the proof, because we have $Q \subseteq P_f(\mathsf{W}_f) = \mathsf{W}_f \subseteq Q$, and hence $[u] = [v] : \mathsf{W}_f \to S$.

Existence. To provide a u that satisfies the commutation, we simply use a fixed-point combinator Y of our underlying PCA $\mathcal{D}$:

$$u \triangleq \mathsf{Y} \cdot \big(\lambda\!\!\lambda v.\lambda\!\!\lambda p.\ s \cdot \langle p_0, v \circ p_1\rangle\big) \qquad \text{so that} \qquad u \cdot \langle i, d\rangle \asymp s \cdot \langle i, u \circ d\rangle$$

By construction, if u tracks any (uniquely determined) morphism, it makes the square commute and hence is a W_f-morphism as well; and because it is the unique such morphism due to the previous construction, we are then done. To check that u indeed tracks a morphism $[u] : \mathsf{W}_f \to S$, we need to prove that $\langle i, d\rangle \sim_{\mathsf{W}_f} \langle j, e\rangle$ implies $(u \cdot \langle i, d\rangle) \sim_S (u \cdot \langle j, e\rangle)$. By definition of u, and s being a per morphism, this is the case when

$$\langle i, u \circ d\rangle \sim_{P_f(S)} \langle j, u \circ e\rangle \tag{6.1}$$

which then by definition of P_f is the case when:

$$i \sim_A j \wedge \forall_{n,m}\big[\, n \sim_B m \wedge (\overline{f} \cdot n) \sim_A i \Rightarrow (u \cdot (d \cdot n)) \sim_S (u \cdot (e \cdot m))\,\big] \tag{6.2}$$

Here we have our recursion, and hence we do a (transfinite) induction on the stage of inclusion γ that both $\langle i, d\rangle, \langle j, e\rangle \in P_f^\gamma(\varnothing) \subseteq \mathsf{W}_f$, i.e. for the induction hypothesis

$$\langle i, d\rangle \sim_{P_f^\gamma(\varnothing)} \langle j, e\rangle \quad \Rightarrow \quad s \cdot \langle i, u \circ d\rangle \sim_S s \cdot \langle j, u \circ e\rangle$$

The case of $\gamma = 0$ is vacuous. Assuming (IH) for certain γ, and letting $\langle i, d\rangle \sim_{P_f^{\gamma+1}(\varnothing)} \langle j, e\rangle$, we get $(d \cdot n) \sim_{P_f^\gamma(\varnothing)} (e \cdot m)$ (for appropriate n, m) by definition of P_f, and then $u \cdot d \cdot n \sim_S u \cdot e \cdot m$ by (IH), so that we have $\langle i, u \circ d\rangle \sim_{P_f(S)} \langle j, u \circ e\rangle$, and then we can apply s to get $s \cdot \langle i, u \circ d\rangle \sim_S s \cdot \langle j, u \circ e\rangle$. The limit goes similarly. □

Corollary 6.1.6. *This means that* **Per** *is closed under the formation of W-types, and hence is a Martin-Löf category.* ◿

Example 6.1.7. Suppose we define the natural numbers with P_f for $f = \mathsf{inr} : \mathbf{1} \to \mathbf{1} + \mathbf{1}$. Note that $\mathbf{1} + \mathbf{1} = \big\{[\langle \mathsf{true}, -\rangle], [\langle \mathsf{false}, -\rangle]\big\}$.

$$P_f(C) = \Sigma_{i \in \mathbf{1}+\mathbf{1}} C^{f^{-1}(i)} = \big\{[\langle\langle \mathsf{true}, -\rangle, -\rangle]\big\} \cup \big\{[\langle\langle \mathsf{false}, -\rangle, d\rangle] \mid [d] : \mathbf{1} \to C\big\}$$

Then, denoting $\underline{0} \triangleq [\langle\langle \mathsf{true}, -\rangle, -\rangle]$ and $\underline{n+1} \triangleq [\langle\langle \mathsf{false}, -\rangle, \lambda\!\!\lambda z.\underline{n}\,\rangle]$, we get a chain of inclusions $\varnothing \subseteq \{\,\underline{0}\,\} \subseteq \{\,\underline{0}, \underline{1}\,\} \subseteq \ldots$, and finally $\mathsf{W}_f = \{\,\underline{n} \mid n \in \mathbb{N}\}$.

We get a (non-dependent) recursor u from the UP of initial F-algebras, which we can explicitly extract from the 'existence' part of theorem 6.1.5 above. Recall that, type-theoretically, the recursor is presented (informally) as

$$\frac{C\ \mathsf{type} \qquad z : C \qquad s : C \to C}{\mathsf{rec} : \mathbb{N} \to C}$$

and having computation rules $\mathsf{rec}\,0 \equiv z$ and $\mathsf{rec}\,(n+1) \equiv s\,(\mathsf{rec}\,n)$.

If we have any per D, and two morphisms $[z] : \mathbf{1} \to D$ and $[s] : D \to D$, then we can have the coproduct morphism $[\,[z, s]\,] : P_f(D) \to D$, so that D is a P_f-algebra. The recursor, being defined using a fixed-point computer in the underling PCA, is such that

$$\mathsf{rec} \cdot \langle i, d\rangle \asymp [z, s] \cdot \langle i, \mathsf{rec} \circ d\rangle$$

and hence

$$\begin{aligned} \mathsf{rec} \cdot \underline{0} &\asymp z \cdot (-) \\ \mathsf{rec} \cdot (\underline{n+1}) &\asymp s \cdot (\mathsf{rec} \cdot \underline{n}) \end{aligned}$$

We cannot have a dependent eliminator just yet, though. For this we will have to wait until the next section, where dependent inductive types are defined, and then use a trick. ◿

Example 6.1.8. Suppose we define the lists over some per A with P_f for $f = \mathsf{inr} : A \to \mathbf{1}+A$.

$$P_f(C) = \Sigma_{i \in \mathbf{1}+A} C^{f^{-1}(i)} = \big\{[\langle\langle \mathsf{true}, -\rangle, -\rangle]\big\} \cup \big\{[\langle\langle \mathsf{false}, n\rangle, d\rangle] \mid [n] \in A, [d] : \mathbf{1} \to C\big\}$$

If we denote

$$\begin{aligned} <> &\triangleq \langle\langle \mathsf{true}, -\rangle, -\rangle \\ <n_1> &\triangleq \langle\langle \mathsf{false}, n_1\rangle, \curlywedge z.\langle\langle \mathsf{true}, -\rangle, -\rangle\rangle \\ <n_2, n_1> &\triangleq \langle\langle \mathsf{false}, n_2\rangle, \curlywedge z.\langle\langle \mathsf{false}, n_1\rangle, \curlywedge z.\langle\langle \mathsf{true}, -\rangle, -\rangle\rangle\rangle \\ <n_3, n_2, n_1> &\triangleq \langle\langle \mathsf{false}, n_3\rangle, \curlywedge z.\langle\langle \mathsf{false}, n_2\rangle, \curlywedge z.\langle\langle \mathsf{false}, n_1\rangle, \curlywedge z.\langle\langle \mathsf{true}, -\rangle, -\rangle\rangle\rangle\rangle \\ &\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \end{aligned}$$

Then, we get a chain of inclusions like so:

$$\varnothing \subseteq \big\{[<>]\big\} \subseteq \cdots \subseteq \big\{[<n_k, \ldots, n_1>] \mid k \geq 0, n_1, \ldots, n_k \in \operatorname{dom} A\big\} = \mathsf{W}_f$$

Similar to the previous example, we can find the recursor for this per. Note that, for any per A, having a pair of morphisms $[z] : \mathbf{1} \to D$ and $[c] : A \times D \to D$ is (realizably) equivalent to having a morphism $[d] : P_f(D) \to D$ (essentially a uniformly tracked family of morphisms, uncurried), say with some appropriate $r \in \mathcal{D}$ acting as in the equations below, and then the initiality of W_f gives us the recursor again, which will behave as expected:

$$\begin{aligned} \mathsf{rec} \cdot <> &\asymp d \cdot \langle \mathsf{true}, -\rangle \cdot (-) && \asymp r \cdot z \cdot (-) \\ \mathsf{rec} \cdot <n_k, \ldots> &\asymp d \cdot \langle \mathsf{false}, n_k\rangle \cdot (\curlywedge z.\mathsf{rec} \cdot <\ldots>) && \asymp r \cdot c \cdot \langle n_k, \mathsf{rec} \cdot <\cdots>\rangle \end{aligned}$$

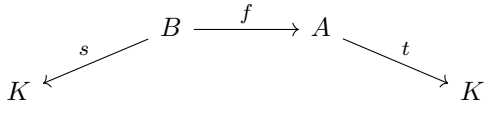
Again, we have to wait for the dependent case to get a dependent eliminator. ◿

6.2 For dependent polynomials

Now, let us consider algebras for dependent polynomials in **Per**. Recall that a dependent polynomial is (any functor F that is isomorphic to) P, which is given (but setting $I = J = K$) as

$$P \triangleq \mathbf{Per}^K \xrightarrow{s^*} \mathbf{Per}^B \xrightarrow{\Pi_f} \mathbf{Per}^A \xrightarrow{\Sigma_t} \mathbf{Per}^K$$

for the diagram



in **Per**

First let us compute explicitly that

$$P\big(\{C_k\}_k\big) = \big\{\ \Sigma_{a\in t^{-1}(k)}\Pi_{b\in f^{-1}(a)}C_{s(b)}\ \big\}_k$$
$$P\big(\{h_k : C_k \to D_k\}_k\big) = \big\{\ (a,g)\mapsto \big(a, b\mapsto h_{s(b)}\big(g(b)\big)\big) : (FC)_k \to (FD)_k\ \big\}_k$$

Note that if h is uniformly tracked by e, then Ph is uniformly tracked by

$$\lambda\!\!\lambda p.\, \langle p_0, \lambda\!\!\lambda n.\, e\cdot(\overline{s}\cdot n)\cdot(p_1\cdot n)\rangle$$

(Compare this with the $\lambda\!\!\lambda p.\, \langle p_0, u\circ p_1\rangle$ that we had in the non-dependent case.) In particular, P is itself realized as well. We can also characterize F's equivalence relation again:

$$\begin{aligned}\langle i,d\rangle \sim_{(PC)_i} \langle j,e\rangle \quad \text{iff} \quad & i\sim_A j \wedge (\overline{t}\cdot i)\sim_K \overline{k}\\ & \wedge\, \forall_{n,m}\big[\, n\sim_B m \wedge (\overline{f}\cdot n)\sim_A i \Rightarrow (d\cdot n)\sim_{C_{s([n])}} (e\cdot m)\,\big]\end{aligned}$$

Indeed we can simply perform the same argument as in the non-dependent case: Kleene's fixed-point theorem applies to the situation in which P is Scott-continuous, which is at least so when every $f^{-1}(a)$ is finite in size. For full generality though, we have to provide a new proof, but this is easy, because of the component-wise nature of the construction.

Lemma 6.2.1. *For each per I, the structure $(\mathbf{Per}^I, \underset{c.w.}{\subseteq}, \underset{c.w.}{\bigcup}, \{\varnothing\}_i)$ is a DCPO.*

Proof. This follows directly from lemma 6.1.2, and the component-wiseness of the data. □

Proposition 6.2.2. *If $f : B \to A$ is such that every $f^{-1}(a)$ is finite (i.e. is a finite amount of equivalence classes), then P is Scott-continuous, i.e. preserves limits of directed sets.*

Proof. Similar as before, we have the inequivalence of:

$$[\ell] \in \prod_{b\in f^{-1}(a)} \big(\overset{c.w.}{\bigcup}\mathcal{X}\big)_{s(n)} \qquad \text{iff} \qquad n\sim_{f^{-1}(a)} m \Rightarrow \exists_{X\in\mathcal{X}}\big[\,\ell\cdot n\sim_{X_{s([n])}}\ell\cdot m\,\big]$$
$$[\ell] \in \bigcup_{X\in\mathcal{X}}\prod_{b\in f^{-1}(a)} X_{s(n)} \qquad \text{iff} \qquad \exists_{Y\in\mathcal{X}}\Big[\, n\sim_{f^{-1}(a)} m \Rightarrow \big[\,\ell\cdot n\sim_{Y_{s([n])}}\ell\cdot m\,\big]\,\Big]$$

And, if $f^{-1}(a)$ is always finite, we can define with finite union $Y \triangleq \bigcup_{[n]\in f^{-1}(a)} X_{s([n])}$—which will be $Y\in\mathcal{X}$ again due to the latter's directedness—so that we have ($\Downarrow$) as well as ($\Uparrow$). Then, we can compute

$$\begin{aligned}P\big(\overset{c.w.}{\bigcup}\mathcal{X}\big) &= \Big(\sum_{a\in t^{-1}(k)}\prod_{b\in f^{-1}(a)}\big(\overset{c.w.}{\bigcup}\mathcal{X}\big)_{s(b)}\Big)_k\\ &= \Big(\sum_{a\in t^{-1}(k)}\bigcup_{Y\in\mathcal{X}}\prod_{b\in f^{-1}(a)} Y_{s(b)}\Big)_k \qquad \text{(as described)}\\ &= \Big(\bigcup_{Y\in\mathcal{X}}\sum_{a\in t^{-1}(k)}\prod_{b\in f^{-1}(a)} Y_{s(b)}\Big)_k\\ &= \overset{c.w.}{\bigcup_{Y\in\mathcal{X}}} P(Y)\end{aligned}$$

□

Corollary 6.2.3. *If $f^{-1}(a)$ is always finite, we can readily apply Kleene's fixed-point theorem, obtaining an initial P-algebra in the fiber $\mathbf{Per}^K$, as:*

$$\mathsf{W}_F = \overset{c.w.}{\bigcup_{n\in\mathbb{N}}} P^n(\{\varnothing\}_k) \qquad \text{with} \qquad \{w_k\}_k = \mathsf{id} : P(\mathsf{W}_F)\to\mathsf{W}_F$$

◿

Theorem 6.2.4. *For every such polynomial P, the initial P-algebra exists in* $\mathsf{Fam}(\mathbf{Per})$.

Proof Sketch. Essentially the same way as for the non-dependent case, this time using the component-wise structure of the fibers. Construct the sequence (writing F for $[P]$):

$$\{\varnothing\}_k \underset{\text{c.w.}}{\subseteq} P\big(\{\varnothing\}_k\big) \underset{\text{c.w.}}{\subseteq} \dots \underset{\text{c.w.}}{\subseteq} \overset{\text{c.w.}}{\bigcup}_n\big(P^n\big(\{\varnothing\}_k\big)\big) \underset{\text{c.w.}}{\subseteq} \dots \underset{\text{c.w.}}{\subseteq} P\Big(\overset{\text{c.w.}}{\bigcup}_n\big(P^n\big(\{\varnothing\}_k\big)\big)\Big) \underset{\text{c.w.}}{\subseteq} \dots$$

Again we prove that the inclusions hold by (transfinite) induction. The first inclusion is of course trivial. Now suppose (IH) that we already have $C \underset{\text{c.w.}}{\subseteq} PC$, i.e.

$$\forall_{k\in K}\big[\, n \sim_{C_k} m \Rightarrow n \sim_{(PC)_k} m \,\big] \tag{IH}$$

Then, for any $k \in K$, assuming $\langle i, d\rangle \sim_{(PC)_k} \langle j, e\rangle$, we need to prove the recursive component $(d \cdot n) \sim_{(PC)_{s([n])}} (e \cdot m)$ to get $\langle i, d\rangle \sim_{(P^2C)_k} \langle j, e\rangle$, which we easily get from (IH). That we can take limit unions is because the sequence (up to any given point) is directed.

We again take the supremum:

$$\mathsf{W}_F \triangleq \overset{c.w.}{\bigcup_{\gamma}} P^\gamma\big(\{\varnothing\}_k\big)$$

Again, that W_F is an F-algebra follows directly, because we can take the identity morphism as its structure map. To show that W_F is initial, we have to put in a little more work. Let R be some P-algebra with map $r = \{r_k\}_k : PR \to R$ uniformly tracked by $q \in \mathcal{D}$ (and we will write $[q]$ to denote r), so that we must provide some $h = \{h_k\}_k$ uniformly tracked by $u \in \mathcal{D}$, to satisfy:

$$\begin{array}{ccc} P(\mathsf{W}_F) & \xrightarrow{[\lambda x.x]} & \mathsf{W}_F \\ {\scriptstyle [\lambda p.\langle p_0, \lambda n.u\cdot(\overline{s}\cdot n)\cdot(p_1\cdot n)\rangle]}\downarrow & & \vdots\,{\scriptstyle [u]} \\ P(R) & \xrightarrow{[q]} & R \end{array}$$

Again we provide u by using a fixed-point combinator of $\mathcal{D}$:

$$\begin{aligned} & u \triangleq \mathsf{Y}\cdot\big(\lambda v.\lambda p.\, q\cdot\langle p_0,\, \lambda n.\, v\cdot(\overline{s}\cdot n)\cdot(p_1\cdot n)\rangle\big) \\ \text{so that}\quad & u\cdot\langle i, d\rangle \asymp q\cdot\langle i,\, \lambda n.\, u\cdot(\overline{s}\cdot n)\cdot(d\cdot n)\rangle \end{aligned}$$

That this map exists, and is unique, follows similarly as before. That it then makes W_F an F-algebra is by construction. □

Example 6.2.5. Let us consider vectors of length $n \in \mathbf{N}$ over some per A (where $\mathbf{N}$ is the per of natural numbers defined in 6.1.7). We've seen the dependent polynomial characterization of this type in example 2.4.14, the morphisms of the diagram of which are obviously realized and hence in **Per**, hence resulting in a polynomial functor $F : \mathbf{Per}^{\mathbf{N}} \to \mathbf{Per}^{\mathbf{N}}$ mapping

$$FX \cong \begin{Bmatrix} \mathbf{1} & n = \underline{0} \\ A \times X_{\underline{k}} & n = \underline{k+1} \end{Bmatrix}_{n\in\mathbf{N}}$$

Applying Kleene's fixed-point construction, which is the same as the construction in the explicit proof given above (because lists are not infinitely branching), we get

$$\mathsf{W}_F = \overset{c.w.}{\bigcup_{n\in\mathbb{N}}} F^n(\{\varnothing\}_{n\in\mathbf{N}}) = \big\{\, \overbrace{(A \times \cdots \times (A \times (A}^{n} \times \mathbf{1}))\cdots)\big\}_{\underline{n}} \cong \{A^n\}_{\underline{n}}$$

For any per family $\{D_n\}_{n\in\mathbf{N}}$, morphism $[z] : \mathbf{1} \to D_{\underline{0}}$, and uniformly tracked family of morphisms $\left\{A \times D_{\underline{n}} \xrightarrow{c(n)} D_{\underline{n+1}}\right\}_{\underline{n}}$, we can form an F-algebra $(D, [d])$ with $d \triangleq \mathsf{rec}_{\mathbf{N}}(z, c)$. We then get a recursor $[\mathsf{rec}] : \mathsf{W}_F \to D$ from initiality, which is a uniformly tracked family $\left\{\left(\mathsf{W}_F\right)_{\underline{n}} \xrightarrow{[\mathsf{rec}\cdot n]} D_{\underline{n}}\right\}_{\underline{n}}$, and for which we have

$$\begin{aligned} \mathsf{rec} \cdot \underline{0} \cdot (-) &\asymp z \cdot (-) \\ \mathsf{rec} \cdot \underline{n+1} \cdot r &\asymp c \cdot \underline{n} \cdot r \end{aligned}$$

◿

Note that although there is indeed a lot of type-dependency on $\mathbf{N}$, the recursor of this last example really is the 'non-dependent' recursor for vectors, which would type-theoretically typically be presented as:

$$\frac{n^{\mathbb{N}} \vdash C_n\ \mathsf{type} \qquad z : C_0 \qquad c_a : A \times C_n \to C_{n+1}}{\mathsf{rec} : \Pi_{n:\mathbb{N}}\, \mathsf{Vec}_n \to C_n}$$

and must not be confused with the 'dependent eliminator' for vectors, which is rather:

$$\frac{n^{\mathbb{N}}, v^{\mathsf{Vec}_n} \vdash C_n(v)\ \mathsf{type} \qquad z : C_0(\mathsf{nil}) \qquad c_n : \Pi_{a:A, v:\mathsf{Vec}_n}\, A \times C_n(v) \to C_{n+1}(\mathsf{cons}(a, v))}{\mathsf{rec} : \Pi_{n:\mathbb{N}}\, \Pi_{v:\mathsf{Vec}_n} C_n(v)}$$

As before, with the natural numbers and lists, we cannot directly have a dependent eliminator as well. However, we do now have dependent inductive types, which provide us with the machinery to construct them. The following section shows how.

6.3 Dependent eliminators

As we saw in example 6.1.7, where we had constructed the natural numbers as the initial algebra for a non-dependent polynomial, giving the per $\mathbf{N} = \{\underline{0}, \underline{1}, \dots\}$ and the structure map $P_f(\mathbf{N}) = \mathbf{1} + \mathbf{N} \xrightarrow{c} \mathbf{N}$, we could not yet have a dependent eliminator due to the fact that the UP of P_f-initiality was not expressive enough.

Now that we have initial algebras for dependent polynomials, we apply a simple trick to get a dependent eliminator anyway, for any inductive type. Essentially, we construct the 'type of proofs', which is an inductive type that is term-dependent over the objects of the inductive type in question, and whose main feature is not its inhabitance, but its dependency structure.

For non-dependent inductive types

Recall that, type-theoretically, the dependent eliminator for natural numbers is presented (informally) as

$$\frac{n^{\mathbb{N}} \vdash C(n)\ \mathsf{type} \qquad z : C(0) \qquad s : \Pi_{n:\mathbb{N}}\, C(n) \to C(n+1)}{\mathsf{elim} : \Pi_{n:\mathbb{N}}\, C(n)}$$

and having computation rules $\mathsf{elim}\, 0 \equiv z$ and $\mathsf{elim}\, (n+1) \equiv s\, n\, (\mathsf{elim}\, n)$.

Basically, we'll be constructing the following inductive type:

```
ind dN : N → Type :=
  dzero : dN zero
  dsucc : forall n:N, dN n → dN (succ n)
```

This corresponds, in the same way as for the vectors in example 2.4.14, to the dependent polynomial:

$$F = \quad \mathbf{N} \xleftarrow{[!,\mathsf{id}]} \mathbf{0}+\mathbf{N} \xrightarrow{!+\mathsf{id}} \mathbf{1}+\mathbf{N} \xrightarrow{c} \mathbf{N}$$

for which an initial algebra would amount to some pair (W, w) with

$$\begin{Bmatrix} \mathbf{1} & n=0 \\ W_{\underline{n-1}} & n \geq 1 \end{Bmatrix}_{\underline{n}} \xrightarrow{\{w_{\underline{n}}\}_{\underline{n}}} \{W_{\underline{n}}\}_{\underline{n}}$$

If we then take the comprehension of this object, we get $\{W\} = \Sigma_{n \in \mathbf{N}} \mathbf{1}$, in which we have equivalence classes of the form $[\langle \underline{n}, - \rangle]$, for $\underline{n} \in \mathbf{N}$. Now, for the dependent eliminator, suppose we have some family $\{P_{\underline{n}}\}_{\underline{n}}$, and some morphism $[c_z] : \mathbf{1} \to P_{\underline{0}}$, and a uniformly tracked family of morphisms $\left\{P_{\underline{n}} \xrightarrow{[c_s \cdot \underline{n}]} P_{\underline{n+1}}\right\}$. Then we can have the following $\mathsf{Fam}(\mathbf{Per})$-morphism, because we can indeed uniformly define $p \cdot \underline{n}$ to be c_z when $n = 0$ and $c_s \cdot \underline{n}$ when $n \geq 1$ using $\mathsf{rec}_{\mathbf{N}}$.

$$\begin{Bmatrix} \mathbf{1} & n=0 \\ P_{\underline{n-1}} & n \geq 1 \end{Bmatrix}_{\underline{n}} \xrightarrow{\{p \cdot \underline{n}\}_{\underline{n}}} \{P_{\underline{n}}\}_{\underline{n}}$$

Again, we use the unique map guaranteed by the proof, defined in terms in a fixed-point combinator of the underlying PCA, and for which

$$\mathsf{elim} \cdot \langle i, d \rangle \asymp \overline{p} \cdot \langle i, \curlywedge n.\, \mathsf{elim} \cdot (\overline{[!, \mathsf{id}]} \cdot n) \cdot (d \cdot n) \rangle$$

hence

$$\begin{aligned} \mathsf{elim} \cdot \langle \underline{0}, - \rangle &\asymp c_z \cdot (-) \\ \mathsf{elim} \cdot \langle \underline{n+1}, - \rangle &\asymp c_s \cdot \underline{n+1} \cdot (\mathsf{elim} \cdot \langle \underline{n}, - \rangle) \end{aligned}$$

Similarly, we could continue from 6.1.8, to get a dependent eliminator for lists. This eliminator would type-theoretically be presented as:

$$\frac{\ell^{\mathsf{List}_A} \vdash C(\ell)\ \mathsf{type} \qquad z : C(\mathsf{nil}) \qquad c : \Pi_{a:A, \ell:\mathsf{List}_A} C(\ell) \to C(\mathsf{cons}(a, \ell))}{\mathsf{elim} : \Pi_{\ell:\mathsf{List}_A}\, C(\ell)}$$

To get this eliminator, we would construct the dependent inductive type:

```
ind dList {A} : (List A) → Type :=
  dnil : dList nil
  dcons : forall a:A, forall l:List,
    dList l → dList (cons a l)
```

Using the constructed initial list algebra $\mathbf{L}$ over some per A, with structure map $P_f(\mathbf{L}) = \mathbf{1} + (A \times \mathbf{L}) \xrightarrow{c} \mathbf{L}$, this corresponds to the dependent polynomial:

$$F = \quad \mathbf{L} \xleftarrow{[!,\pi_1]} \mathbf{0}+(A \times \mathbf{L}) \xrightarrow{!+\mathsf{id}} \mathbf{1}+(A \times \mathbf{L}) \xrightarrow{c} \mathbf{L}$$

For dependent inductive types

For the vectors (example 6.2.5), recall that the dependent eliminator should be:

$$\frac{n^{\mathbb{N}}, v^{\mathsf{Vec}_n} \vdash C_n(v)\ \mathsf{type} \qquad z : C_0(\mathsf{nil}) \qquad c_n : \Pi_{a:A, v:\mathsf{Vec}_n} A \times C_n(v) \to C_{n+1}(\mathsf{cons}(a, v))}{\mathsf{rec} : \Pi_{n:\mathbb{N}}\, \Pi_{v:\mathsf{Vec}_n} C_n(v)}$$

To get such an eliminator, we construct the dependent inductive type:

```
ind dVec {A} : forall n:N, (Vec A n) → Type :=
  dvnil : dVec zero vnil
  dvcons : forall n:N, forall a:A, forall v:Vec n,
    dVec n v → dVec (succ n) (vcons n a v)
```

Using the already constructed per family $\mathbf{V} = \{\mathbf{V}_{\underline{n}}\}_{\underline{n}}$ of vectors (over some per A), we use comprehension to get

$$\{\mathbf{V}\} = \Sigma_{n \in \mathbf{N}} \mathbf{V}_n$$

which will have equivalence classes of the form $[\langle \underline{n}, v \rangle]$, where $\underline{n} \in \mathbf{N}$ and $v \in \mathbf{V}_{\underline{n}}$. For $[k] \in A$, let us write $[k :: v]$ to denote the equivalence class denoting the 'cons'-ed vector with k prepended to v.

Then, we use the dependent polynomial for the diagram:

$$\begin{array}{ccccc}
 & & & & \{F_{\mathsf{Vec}}(\mathbf{V})\} \\
 & & & & \| \\
 & & & \overset{!+\mathsf{id}}{\{+\mathsf{id}\}} & \\
 & & & \overset{+\cdots}{\longrightarrow} & \\
\overset{[!,\pi_1}{\{,\pi_1\}}\overset{,\ldots]}{} & \swarrow & \left\{\begin{matrix}\mathbf{0} & n=0 \\ A \times \mathbf{V}_{\underline{n-1}} & n \geq 1\end{matrix}\right\}_{\underline{n}} & & \left\{\begin{matrix}\mathbf{1} & n=0 \\ A \times \mathbf{V}_{\underline{n-1}} & n \geq 1\end{matrix}\right\}_{\underline{n}} & \searrow^{\{c\}} \\
\{\mathbf{V}\} & & & & \{\mathbf{V}\}
\end{array}$$

This dependent polynomial is a functor over the fiber $\mathbf{Per}^{\{\mathbf{V}\}}$, and maps families of pers as follows (modulo isomorphism):

$$F\Big(\{X_{[\langle \underline{n}, v\rangle]}\}_{[\langle \underline{n}, v\rangle]}\Big) = \left\{\begin{matrix}\mathbf{1} & n=0 \\ X_{[\underline{n-1}, v']} & n \geq 1 \wedge v = k :: v'\end{matrix}\right\}_{[\langle \underline{n}, v\rangle]}$$

Hence, an initial algebra for this polynomial is found as simply

$$\mathsf{W}_F = \bigcup_{n \in \mathbb{N}}^{c.w.} F^n\big(\{\varnothing\}_{[\langle \underline{n}, v\rangle]}\big) \cong \{\mathbf{1}\}_{[\langle \underline{n}, v\rangle]}$$

But the clue lies not in this object, and rather in the recursor that follows from initiality, which can be used as eliminator for the vectors due to the type dependency structure of this 'proof type', which can be seen from the fact that every F-algebra $(P, [p])$ will have the shape

$$\left\{\begin{matrix}\mathbf{1} & n=0 \\ P_{[\langle \underline{n-1}, v'\rangle]} & n \geq 1 \wedge v = k :: v'\end{matrix}\right\}_{[\langle \underline{n}, v\rangle]} \xrightarrow{[p \cdot \langle \underline{n}, v\rangle]} \{P_{[\langle \underline{n}, v\rangle]}\}_{[\langle \underline{n}, v\rangle]}$$

The morphism $[p]$ is an uncurried version of a uniformly tracked family of morphisms, and can be constructed given certain $[z] : \mathbf{1} \to P_{[\langle 0, -\rangle]}$ and $\big\{P_{[\langle \underline{n}, v\rangle]} \xrightarrow{[c \cdot \langle \underline{n+1}, k::v\rangle]} P_{[\langle \underline{n+1}, k::v\rangle]}\big\}$. Given a $[p]$ constructed this way, the recursor will give us a dependent eliminator for the vectors:

$$\begin{aligned}
\mathsf{elim} \cdot \langle \langle \underline{0}, v_{\mathsf{nil}} \rangle, - \rangle &\asymp z \cdot (-) \\
\mathsf{elim} \cdot \langle \langle \underline{n+1}, k :: v \rangle, - \rangle &\asymp c \cdot \langle \underline{n+1}, k :: v \rangle \cdot (\mathsf{elim} \cdot \langle \langle \underline{n}, v \rangle, - \rangle)
\end{aligned}$$

6.4 For (F, G)-dialgebras

Wrapping up this chapter, we would like to briefly mention that (F, G)-dialgebras can be used as a more elegant and descriptively intuitive way of specifying (dependent) inductive types. Using theorem 2.4.7 (as was shown in Basold [2015]) one can subsequently

mechanically reduce such dialgebras to polynomial functor algebras, alleviating the need to describe the mapping from inductive type specifications to their respective polynomials (which can get quite messy), as we did in the previous sections.

Proposition 6.4.1. *All simple pairs (F, G) (see definition 2.4.16) are realized.*

Proof. This is easily seen. As a reindexing functor $G = G_u$ is realized. And as a constant, reindexing, or composition, pairing, or projection of previous simple functors, F is also easily realized. □

Example 6.4.2. Example 2.4.5 showed how the type of vectors over a given type A arises as the initial (F, G)-dialgebra for

$$\begin{aligned} FX &\triangleq \big(\{\mathbf{1}\}, \{A \times X_n\}_n\big) \\ GX &\triangleq \big(\{X_0\}, \{X_{n+1}\}_n\big) \end{aligned}$$

These functors can, due to proposition 6.4.1 above, and by the fact that we already have a natural numbers inductive type $\mathbf{N}$, and over the (realizable) equivalence $\mathbf{Per}^{\mathbf{1}+\mathbf{N}} \cong \mathbf{Per}^{\mathbf{1}} \times \mathbf{Per}^{\mathbf{N}}$, be seen as realizable functors $F, G : \mathbf{Per}^{\mathbf{N}} \to \mathbf{Per}^{\mathbf{1}+\mathbf{N}}$. And indeed, as example 2.4.14 showed how, over theorem 2.4.7, we can then get the same class of algebras, modulo isomorphism, as the P-algebras for

$$\begin{aligned} P &= \big(F_z; \Sigma_z\big) +_{\mathbf{N}} \big(F_s; \Sigma_s\big) \\ &\cong X \mapsto \begin{Bmatrix} \mathbf{1} & n = 0 \\ A \times \underline{X_{n-1}} & n \geq 1 \end{Bmatrix}_{\underline{n}} \end{aligned}$$

Example 6.2.5 showed how the initial algebra for this dependent polynomial could be found, and subsection 6.3 showed how, in addition, its dependent eliminator could be derived. ◿

Example 6.4.3. Reconsider example 2.4.6, where we had

$$\big(\mathsf{Diag}(C)\big)_{(a,b)} = \begin{cases} C_a & \text{if } a = b \\ \mathbf{0} & \text{else} \end{cases}$$

We can define this as a family of pers indexed over any $\mathcal{D}$-**Set** A, and then we can recover our adjoint situation in $\mathsf{Fam}(\mathbf{Per})$, per the following.

Every $f : \mathsf{Diag}(C) \to D$, say uniformly tracked by d, can be mapped naturally to the $g : C \to f^*(D)$ with $g_a(c) \triangleq f_{(a,a)}(c)$, which is uniformly tracked by $e \triangleq \lambda n.d \cdot \langle n, n \rangle$. Every $g : C \to f^*(D)$, say uniformly tracked by e, can be mapped naturally to the $f : \mathsf{Diag}(C) \to D$ with $f_{(a,a)}(c) \triangleq g_a(c)$ and $f_{(a,b)} \triangleq ! : \mathbf{0} \to D_{(a,b)}$ for $a \neq b$.

Interestingly, this last morphism is indeed uniformly tracked by $d \triangleq \lambda n.e \cdot n_0$, because whenever $a = b$, we can use e, and whenever $a \neq b$, the tracking requirement itself degenerates to vacuously hold — there is no need to actually be able to effectively decide whether a and b are equal.

Also, note that, although the functor $\mathsf{Diag} : \mathbf{Per}^{A \times A} \to \mathbf{Per}^{A}$ need not necessarily be realizable, this does not hinder the construction of the inductive equality type, which arises simply as the initial $(K_{\{\mathbf{1}\}_{a \in A}}, \delta^*)$-dialgebra, which is $\mathsf{Eq}_A = \mathsf{Diag}(\{\mathbf{1}\}_{a \in A})$, indeed the trivial propositional equality type from proposition 5.0.8. ◿

7 Conclusion

We have seen the construction of a concrete model of the Calculus of Constructions, based on the realizability interpretation of logic, originally due to Kleene. In essence, the interpretation models types as certain 'specifications' what may hold over objects of an underlying system of computation, for which an arbitrary partial combinatory algebra $\mathcal{D}$ sufficed.

It is noteworthy that in doing this, an extensional flavor is 'automatically' applied to the function type. This turns out to be somewhat hard-wired into the original idea that a logical implication is realized if *there exists* some realizing operation (proof) justifying it. The model simply carries this over to the function type, over the Curry-Howard correspondence.

A key insight, due to Moggi and elaborated by Hyland, is that if a $\mathcal{D}$-set X is equipped with the 'full realizability structure' (meaning that every term of the underlying partial combinatory algebra realizes every element of the set), then the dependent function type $\Pi_{x\in X} A(x)$, where A is a family of partial equivalence relation, is isomorphic to the partial equivalence relation $\bigcap_{x\in X} A(x)$—effectively, because the function type is forced to 'ignore' its input. This can then be used to model polymorphic functions, of which it is traditionally known that they should be interpreted 'as small as possible'. Additionally, taking X to be entire category of **Per**, a 'universe object' of small types is obtained, allowing one to define dependent types and type operators internally, hence allowing for the Calculus of Constructions to be interpreted in its semantic presentation.

Subsequently, we went on to show that **Per** a Martin-Löf category, allowing for the construction of W-types as initial algebras for non-dependent polynomial functors. Also initial algebras for dependent polynomials were obtained by an explicit construction, an instance of the known fact that any Martin-Löf category admits them. These initial algebras were found in the fashion of the well-known Kleene fixed-point theorem, although explicit proofs were given due to the fact that the author does not know the polynomials to be Scott-continuous in full generality (in the case of infinitely branching tree specifications).

Using these initial algebras, is was shown, by way of some typical cases, how (dependent) inductive types can be interpreted in the model, and how dependent eliminators for them may be derived (requiring a little 'meta-trick' whereby the inductive 'type of dependent propositions' is first formulated).

Finally, it was mentioned how a reduction of a certain class of (F, G)-dialgebras to dependent polynomials, described by Basold [2015], may allow for easier categorical specification of dependent inductive types. (The dependent polynomials associated with inductive type specifications can get very messy.)

References

M. Abbott, T. Altenkirch, and N. Ghani. Containers: constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005. [*Cited on pages 5 and 24.*]

H. Barendregt. Introduction to generalized type systems. *Journal of functional programming*, 1(2):125–154, 1991. [*Cited on page 15.*]

H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984. [*Cited on pages 4, 11, and 12.*]

H. Basold. Dependent inductive and coinductive types are fibrational dialgebras. In *Matthes, R.(ed.), FICS 2015: Proceedings Tenth International Workshop on Fixed Points in Computer Science Berlin, Germany, September 11-12, 2015*, pages 3–17. [Sl]: EPTCS, 2015. [*Cited on pages 5, 23, 24, 26, 27, 70, and 72.*]

A. Bauer. *The Realizability Approach to Computable Analysis and Topology*. PhD thesis, Carnegie Mellon University, 2000. [*Cited on page 62.*]

C. Berline. From computation to foundations via functions and application: The λ-calculus and its webbed models. *Theoretical Computer Science*, 249(1):81–161, 2000. [*Cited on page 14.*]

P. Dybjer. Internal type theory. In *International Workshop on Types for Proofs and Programs*, pages 120–134. Springer, 1995. [*Not cited.*]

P. Freyd, P. Mulry, G. Rosolini, and D. Scott. Extensional pers. *Information and Computation*, 98(2):221–227, 1992. [*Cited on page 50.*]

N. Gambino and M. Hyland. Wellfounded trees and dependent polynomial functors. In *Types for Proofs and Programs, LNCS 3085*, pages 210–225. Springer, 2004. [*Cited on page 25.*]

N. Gambino and J. Kock. Polynomial functors and polynomial monads. In *Mathematical proceedings of the cambridge philosophical society*, volume 154, pages 153–192. Cambridge University Press, 2013. [*Cited on page 25.*]

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, University of Paris VII, 1972. [*Cited on pages 4 and 7.*]

M. Hamana and M. Fiore. A foundation for gadts and inductive families: Dependent polynomial functor approach. In *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming*, pages 59–70. ACM, 2011. [*Cited on page 5.*]

J. M. E. Hyland. The effective topos. In *Studies in Logic and the Foundations of Mathematics*, volume 110, pages 165–216. Elsevier, 1982. [*Cited on pages 7 and 30.*]

J. M. E. Hyland. A small complete category. *Annals of pure and applied logic*, 40(2): 135–165, 1988. [*Cited on pages 4 and 7.*]

B. Jacobs. *Categorical logic and type theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999. [*Cited on page 26.*]

S. C. Kleene. On the interpretation of intuitionistic number theory. *The journal of symbolic logic*, 10(4):109–124, 1945. [*Cited on pages 4 and 6.*]

G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, Amsterdam, 1959. [*Cited on page 7.*]

J. R. Longley. *Realizability toposes and language semantics*. PhD thesis, University of Edinburgh, 1995. [*Cited on pages 7, 8, and 11.*]

G. Longo and E. Moggi. Constructive natural deduction and its 'ω-set' interpretation. *Mathematical Structures in Computer Science*, 1(2):215–254, 1991. [*Cited on pages 4, 7, and 44.*]

J. van Oosten. Realizability: A historical essay. *Mathematical Structures in Computer Science*, 12(3):239–263, 2002. [*Cited on page 7.*]

W. Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Technical report, University of Edinburgh, 1992. [*Not cited.*]

B. Reus. Realizability models for type theories. *Electronic Notes in Theoretical Computer Science*, 23(1):128–158, 1999. [*Cited on page 50.*]

G. Rosolini. An exper model for quest. In *International Conference on Mathematical Foundations of Programming Semantics*, pages 436–445. Springer, 1991. [*Cited on page 55.*]

M. Stefanova and H. Geuvers. A simple model construction for the calculus of constructions. In *International Workshop on Types for Proofs and Programs*, pages 249–264. Springer, 1995. [*Cited on page 60.*]

W. P. Stekelenburg. Algebraically compact categories in the effective topos. Master's thesis, Utrecht University, 2011. [*Cited on page 61.*]

S. Vermeeren. Realizability Toposes, 2009. Accessed from `https://stijnvermeeren.be/download/mathematics/essay.pdf` on April 26, 2018. [*Not cited.*]