Type Theory based on Dependent Inductive and Coinductive Types

Henning Basold
Radboud University
CWI, Amsterdam
h.basold@cs.ru.nl

Herman Geuvers
Radboud University
Technical University Eindhoven
herman@cs.ru.nl

Abstract

We develop a dependent type theory that is based purely on inductive and coinductive types, and the corresponding recursion and corecursion principles. This results in a type theory with a small set of rules, while still being fairly expressive. For example, all well-known basic types and type formers that are needed for using this type theory as a logic are definable: propositional connectives, like falsity, conjunction, disjunction, and function space, dependent function space, existential quantification, equality, natural numbers, vectors etc. The reduction relation on terms consists solely of a rule for recursion and a rule for corecursion. The reduction relations for well-known types arise from that. To further support the introduction of this new type theory, we also prove fundamental properties of its term calculus. Most importantly, we prove subject reduction and strong normalisation of the reduction relation, which gives computational meaning to the terms.

The presented type theory is based on ideas from categorical logic that have been investigated before by the first author, and it extends Hagino's categorical data types to a dependently typed setting. By basing the type theory on concepts from category theory we maintain the duality between inductive and coinductive types, and it allows us to describe, for example, the function space as a coinductive type.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic

Keywords Dependent Types, Inductive Types, Coinductive Types, Fibrations

1. Introduction

In this paper, we develop a type theory that is based solely on dependent inductive and coinductive types. By this we mean that the only way to form new types is by specifying the type of their corresponding constructors or destructors, respectively. From such a specification, we get the corresponding recursion and corecursion principles. One might be tempted to think that such a theory is relatively weak as, for example, there is no function space type. However, as it turns out, the function space is definable as a coinductive type. Other type formers, like the existential quantifier, that are needed in logic, are definable as well. Thus, the type theory we present in this paper encompasses intuitionistic predicate logic.

Why do we need another type theory, especially since Martin-Löf type theory (MLTT) (Martin-Löf 1975) or the calculus of inductive constructions (CoIC) (Paulin-Mohring 1993; Werner 1994; Bertot and Castéran 2004) are well-studied frameworks for intuitionistic logic? The main reason is that the existing type theories have no explicit dependent coinductive types. Giménez (Giménez 1995) discusses an extension of the CoIC with coinductive types and guarded recursive schemes but proves no properties about the conversion relation. On the other hand, Sacchini (Sacchini 2013) extended the CoC with streams, and proves subject reduction and strong normalisation. However, the problem of limited support for general coinductive types remains. Finally, we should also mention that general coinductive types are available in implementations like Coq (Coq Development Team 2012), which is based on (Giménez 1995), Agda (Agda 2015) and Nuprl (Constable 1997). Yet, none of these has a formal justification, and Coq's coinductive types are even known to have problems (e.g. related to subject reduction).

One might argue that dependent coinductive types can be encoded through inductive types, see (Ahrens et al. 2015; Basold 2015). However, it is not clear whether such an encoding gives rise to a good computation principle in an intensional type theory such as MLTT or CoIC, see (cLab 2016). This becomes an issue once we try to prove propositions about terms of coinductive type.

Other reasons for considering a new type theory are of foundational interest. First, taking inductive and coinductive types as core of the type theory reduces the number of deduction rules considerably compared to, for example, MLTT with W- and M-types. Second, it is an interesting fact that the (dependent) function space can be described as a coinductive type. This is well-known in category theory but we do not know of any treatment of this fact in type theories. Thus the presented type theory allows us to deepen our understanding of coinductive types.

Contributions Having discussed the raison d'être of this paper, let us briefly mention the technical contributions. First of all, we introduce the type theory and show how important logical operators can be represented in it. We also discuss some other basic examples, including one that shows the difference to existing theories with coinductive types. Second, we show that computations of terms, given in form of a reduction relation, are meaningful, in the sense that the reduction relation preserves types (subject reduction) and that all computations are terminating (strong normalisation). Thus, under the propositions-as-types interpretation, our type theory can serve as formal framework for intuitionistic reasoning.

Related Work A major source of inspiration for the setup of our type theory is categorical logic. Especially, the use of fibrations, brought forward in (Jacobs 1999), helped a great deal in understanding how coinductive types should be treated. Another source of inspiration is the view of type theories as internal language or even free model for categories, see for example (Lambek and Scott 1988). This view is especially important in topos theory, where final coalgebras have been used as foundation for predicative, constructive set theory (Aczel 1988; van den Berg and De Marchi 2007; van den Berg 2006). These ideas were extended in (Basold 2015), which discusses the categorical analogue of the type theory of this paper, see also Sec. 2.

Let us briefly discuss other type theories that the present work relates to. Especially close is the copattern calculus introduced in (Abel et al. 2013), as there the coinductive types are also specified by the types of their destructors. However, said calculus does not have dependent types, and it is based on systems of equations to define terms, whereas the calculus in the present paper is based on recursion and corecursion schemes.

To ensure strong normalisation, the copatterns have been combined with size annotations in (Abel and Pientka 2013). Due to the nature of the reduction relation in these copattern-based calculi, strong normalisation also ensures productivity for coinductive types or, more generally, well-definedness (Basold and Hansen 2015). As another way to ensure productivity, guarded recursive types were proposed and in (Bizjak et al. 2016) guarded recursion was extended to dependent types. Guarded recursive types are not only applicable to strictly positive types, which we restrict to in this paper, but also to positive and even negative types. However, it is not clear how one can include inductive types into such a type theory, which are, in the authors opinion, crucial to mathematics and computer science. Finally, in (Sacchini 2013) another type theory with type-based termination conditions and a type former for streams has been introduced. This type theory, however, lacks again dependent coinductive types.

Outline The rest of the paper is structured as follows. In Sec. 2, we briefly discuss the ideas from category theory that motivate the definition of the type theory. This section is strictly optional and can be safely skipped. The type theory itself is introduced in Sec. 3, and in Sec. 4 we give a host of examples and discuss the representation of logical operators. After that, we justify in Sec. 5 the definition of the typing rules of Sec. 3 by giving an untyped version of the calculus. Moreover, this section serves as the technical basis for the strong normalisation proof. Section 6 is devoted to proving important properties of the type theory, namely subject reduction in Sec. 6.1, and strong normalisation in Sec. 6.2. Finally, we make concluding remarks and discuss future work in Sec. 7.

2. Categorical Dependent Data Types

Before we introduce the actual calculus, let us briefly describe the structure the calculus shall capture. This is a short recap from (Basold 2015), to which we refer for more details. Note, that this section is completely optional and only serves as motivation for those familiar with category theory.

We begin with the definition of dialgebras and associated notions, see (Hagino 1987).

Definition 2.1. Let $\mathbf{C}$ and $\mathbf{D}$ be categories and $F, G : \mathbf{C} \to \mathbf{D}$ be functors. An (F, G)-*dialgebra* is morphism $d : FX \to GX$ in $\mathbf{D}$ for an object X in $\mathbf{C}$. We say that a morphism $f : X \to Y$ is a dialgebra *homomorphism* from the dialgebra $d : FX \to GX$ to $e : FY \to GY$, if $e \circ Ff = Gf \circ d$. This allows us to form the category $\mathrm{DiAlg}\,(F, G)$ of dialgebras and their homomorphisms. Finally, a dialgebra is an *initial* (resp. *final*) (F, G)-dialgebra if it is an initial (resp. final) object in $\mathrm{DiAlg}\,(F, G)$, see (Basold 2015).

Let us discuss an example of a dialgebra in the category of sets.

Example 2.2. Let $F, G : \mathbf{Set} \to \mathbf{Set} \times \mathbf{Set}$ be given by $F = \langle \mathbf{1}, \mathrm{Id} \rangle$ and $G = \langle \mathrm{Id}, \mathrm{Id} \rangle$, that is, F maps a set X to the pair $(\mathbf{1}, X)$ in the product category. Similarly, G, the diagonal functor, maps X to (X, X). Now, let $z : \mathbf{1} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ be the constant zero map and the successor on natural numbers, respectively. It is then easy to see that $(z, s) : F(\mathbb{N}) \to G(\mathbb{N})$ is an initial dialgebra. ◂

Initial and final dialgebras will allow us to describe dependent data types conveniently, where the dependencies are handled through the use of fibrations.

Definition 2.3. Let $P : \mathbf{E} \to \mathbf{B}$ be a functor, where $\mathbf{E}$ is called the *total* category and $\mathbf{B}$ the *base* category. A morphism $f : A \to B$ in $\mathbf{E}$ is said to be *cartesian over* $u : I \to J$ in $\mathbf{B}$, provided that i) $Pf = u$, and ii) for all $g : C \to B$ in $\mathbf{E}$ and $v : PC \to I$ with $Pg = u \circ v$ there is a unique $h : C \to A$ such that $f \circ h = g$. For P to be a *fibration*, we require that for every $B \in \mathbf{E}$ and $u : I \to PB$ in $\mathbf{B}$, there is a cartesian morphism $f : A \to B$ over u. Finally, a fibration is *cloven*, if it comes with a unique choice for A and f, in which case we denote A by $u^* B$ and f by $\overline{u}\, B$, as displayed in the diagram on the right.

$$\begin{tikzcd}
C \arrow[dr, dashed, "!h"'] \arrow[drr, bend left, "g"] & & & \\
& u^* B \arrow[r, "\overline{u}\, B"'] & B & \mathbf{E} \arrow[dd, "P"] \\
PC \arrow[dr, "v"'] \arrow[drr, bend left, "Pg"] & & & \\
& I \arrow[r, "u"'] & PB & \mathbf{B}
\end{tikzcd}$$

On cloven fibrations, we can define for each $u : I \to J$ in $\mathbf{B}$ a functor, the *reindexing* along u, as follows. Let us denote by $\mathbf{P}_I$ the category having objects A with $P(X) = I$ and morphisms $f : A \to B$ with $P(f) = \mathrm{id}_I$. We call $\mathbf{P}_I$ the *fibre above* I. The assignment of $u^* B$ to B for a cloven fibration can then be extended to a functor $u^* : \mathbf{P}_J \to \mathbf{P}_I$. Moreover, one can show that $\mathrm{id}_I^* \cong \mathrm{Id}_{\mathbf{P}_I}$ and $(v \circ u)^* \cong u^* \circ v^*$. In this work, we are mostly interested in *split fibrations*, which are cloven fibrations such that the above isomorphisms are equalities, that is, $\mathrm{id}_I^* = \mathrm{Id}_{\mathbf{P}_I}$ and $(v \circ u)^* = u^* \circ v^*$.

Example 2.4 (See (Jacobs 1999))**.** Important examples of fibrations arise from categories with pullbacks. Let $\mathbf{C}$ be a category and $\mathbf{C}^{\to}$ be the arrow category with morphisms $f : X \to Y$ of $\mathbf{C}$ as objects and commutative squares as morphisms. We can then define a functor $\mathrm{cod} : \mathbf{C}^{\to} \to \mathbf{C}$ by $\mathrm{cod}(f : X \to Y) = Y$. This functor turns out to be a fibration, the *codomain fibration*, if $\mathbf{C}$ has pullbacks. If we are given a choice of pullbacks, then cod is cloven.

The split variant of this construction is given by the category of *set-indexed families* over $\mathbf{C}$. Let $\mathrm{Fam}(\mathbf{C})$ be the category that has families $\{X_i\}_{i \in I}$ of objects X_i in $\mathbf{C}$ indexed by a set I. The morphisms $\{X_i\}_{i \in I} \to \{Y_j\}_{j \in J}$ in $\mathrm{Fam}(\mathbf{C})$ are pairs (u, f) where $u : I \to J$ is a function and f is an I-indexed family of morphisms in $\mathbf{C}$ with $\{f_i : X_i \to Y_{u(i)}\}_{i \in I}$. It is then straightforward to show that the functor $p : \mathrm{Fam}(\mathbf{C}) \to \mathbf{Set}$, given by projecting on the index set, is a split fibration. ◂

To model *dependent* data types, we consider dialgebras in the fibres of a fibration $P : \mathbf{E} \to \mathbf{B}$. Before giving a general account, let us look at an important example: the dependent function space.

Example 2.5. Suppose that $\mathbf{B}$ has a final object $\mathbf{1}$, and let $I \in \mathbf{B}$. Thus, there is a morphism $!_I : I \to \mathbf{1}$, which gives rise to the *weakening* functor $!_I^* : \mathbf{P_1} \to \mathbf{P}_I$. We can then show that for each $X \in \mathbf{P}_I$ the dependent function space $\Pi_I X$ is the final $(!_I^*, K_X)$-dialgebra, whereby K_X is the functor mapping every object to X and morphism to id_X. That is to say, there is a dialgebra $\mathrm{ev}_X : !_I^*(\Pi_I X) \to X$ that evaluates a function on an argument from I, such that for each dialgebra $f : !_I^*(U) \to X$ there is a unique $\lambda f : U \to \Pi_I X$ with $\mathrm{ev}_X \circ !_I^*(\lambda f) = f$. ◂

From a categorical perspective, the dependent function space is actually a functor $\mathbf{P}_I \to \mathbf{P_1}$ that is, moreover, right adjoint to the weakening functor : $!_I^* \dashv \Pi_I$. To capture this, we allow data types to have parameters.

Definition 2.6. Let $\mathbf{C}$, $\mathbf{D}$, $\mathbf{X}$ be categories, and $F : \mathbf{X} \times \mathbf{C} \to \mathbf{D}$ be a functor. We define a functor $\widehat{F} : \mathbf{C}^{\mathbf{X}} \to \mathbf{D}^{\mathbf{X}}$ between functor categories by

$$\widehat{F}(H) = F \circ \langle \mathrm{Id}_{\mathbf{X}}, H \rangle. \tag{1}$$

Let $G : \mathbf{X} \times \mathbf{C} \to \mathbf{D}$ be another functor. A *parameterised* (F, G)-dialgebra is an $(\widehat{F}, \widehat{G})$-dialgebra, that is, a natural transformation $\delta \colon \widehat{F}(H) \Rightarrow \widehat{G}(H)$ for a functor $H : \mathbf{X} \to \mathbf{C}$.

Example 2.7. The dependent function space Π_I functor is a final, parameterised (G, π_1)-dialgebra, where $G, \pi_1 : \mathbf{P}_I \times \mathbf{P_1} \to \mathbf{P}_I$, $G = !_I^* \circ \pi_2$ and π_1 is the product projection. This is a consequence of the fact that $\pi_1(X, U) = K_X(U)$, $G(X, U) = !_I^*(U)$, and that for each X the function space $\Pi_I X$ is a final $(!_I^*, K_X)$-dialgebra. This allows us to prove that the evaluation ev_X is natural in X and that Π_I is final in $\mathrm{DiAlg}\left(\widehat{G}, \widehat{\pi_1}\right)$. ◂

Let $P : \mathbf{E} \to \mathbf{B}$ be a cloven fibration, $I \in \mathbf{B}$ and u a tuple $u = (u_1, \ldots, u_n)$ of morphisms $u_k : J_k \to I$ in $\mathbf{B}$. Then for every $\mathbf{X}$ there is a functor $G_u : \mathbf{X} \times \mathbf{P}_I \to \prod_{k=1}^n \mathbf{P}_{J_k}$ given by

$$G_u = \langle u_1^*, \ldots, u_n^* \rangle \circ \pi_2. \tag{2}$$

Now we are in the position to define what it means for a category to have strictly positive, dependent data types.

Definition 2.8. Given a cloven fibration $P : \mathbf{E} \to \mathbf{B}$ we define by mutual induction data type completeness, the class $\mathcal{S}$ of strictly positive signatures and the class $\mathcal{D}$ of strictly positive data types.

We say that P is *data type complete*, if for all $(F, u) \in \mathcal{S}$ an initial $(\widehat{F}, \widehat{G_u})$- and final $(\widehat{G_u}, \widehat{F})$-dialgebra exists. We denote their carriers by $\mu(\widehat{F}, \widehat{G_u})$ and $\nu(\widehat{G_u}, \widehat{F})$, respectively. A pair (F, u) is a *strictly positive signature*, if $(F, u) \in \mathcal{S}$ by the first rule in Fig. 1. Finally, a *strictly positive data type* is a functor $F \in \mathcal{D}$, as given by the other rules in Fig. 1.

3. Typed Syntax

We introduce our type theory through its typing rules, following the categorical syntax just given. All definitions of this section are given by mutual induction, which we justify in Sec. 5.

Before we formally introduce the typing rules, let us give an informal overview of the syntax. First of all, we will have two kinds of variables: type constructor variables and term variables. This leads us to use well-formedness judgements of the form

$$\Theta \mid \Gamma_1 \vdash A : *,$$

which states that A is a type in the type constructor variable context Θ and the term variable context Γ_1.

The type constructor variables in Θ are meant to capture types with terms as parameters (dependent types), thus we need a means to deal with these parameter dependencies. The way we chose to do this here is by introducing *parameter contexts* and *instantiations* thereof.So we generalise the above judgement to

$$\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \multimap *,$$

in which A is a type constructor in the combined context $\Theta \mid \Gamma_1$ with parameters in Γ_2. Suppose that $\Gamma_2 = x_1 : B_1, \ldots, x_n : B_n$ and that we are given $\Gamma_1 \vdash t_k : B_k[t_1/x_1, \ldots, t_{k-1}/x_{k-1}]$,[1] then the instantiation of A with these terms is denoted by

$$\Theta \mid \Gamma_1 \vdash A \mathbin{@} t_1 \mathbin{@} \cdots \mathbin{@} t_n : *.$$

Note, however, that the arrow $\multimap$ is *not* meant to be the function space in a higher universe, rather parameter contexts are a syntactic tool to deal elegantly with parameters of type constructors. We illustrate this with a small example. Let Γ_2 and $t_1, \ldots, t_n$ be as above, and let X be a type constructor variable. The type system will allow us to form the judgement

$$X : \Gamma_2 \multimap * \mid \Gamma_1 \vdash X : \Gamma_2 \multimap *,$$

and then instantiate X with the terms $t_1, \ldots, t_n$ to obtain

$$X : \Gamma_2 \multimap * \mid \Gamma_1 \vdash X \mathbin{@} t_1 \mathbin{@} \cdots \mathbin{@} t_n : *.$$

Besides parameter instantiation, we also allow variables to be moved from the term variable context into the parameter context by parameter abstraction. Through these two mechanisms we can deal smoothly with type constructor variables, which are dependent types with parameters. As an example we will be able to form

$$X : (x : B, y : B) \multimap * \mid \varnothing \vdash (z).(X \mathbin{@} z \mathbin{@} z) : (z : B) \multimap *.$$

Similar to type constructors with parameters, we also have terms with parameters, and instantiations for them. A term with parameters will be typed by a *parameterised type* of the shape $\Gamma_2 \multimap A$. A term s with $\Gamma_1 \vdash s : \Gamma_2 \multimap A$ can be instantiated with arguments, just like type constructors: If $\Gamma_2 = x_1 : B_1, \ldots, x_n : B_n$ and $\Gamma_1 \vdash t_k : B_k[t_1/x_1, \ldots, t_{k-1}/x_{k-1}]$ for $1 \leqslant k \leqslant n$, then

$$\Gamma_1 \vdash s \mathbin{@} t_1 \mathbin{@} \cdots \mathbin{@} t_n : A[\overrightarrow{t}/\overrightarrow{x}],$$

where $A[\overrightarrow{t}/\overrightarrow{x}]$ denotes the simultaneous substitution of the t_k for the term variables x_k. In the case of terms, however, we do not allow parameter abstraction. We will rather be able to define the (dependent) function space as coinductive type, thus we do not need an explicit type constructor for it and also no explicit λ-abstraction.

Having set up how we deal with type constructor variables, we come to the heart of the calculus. Since it shall have inductive and coinductive types, we give ourselves type constructors that resemble the initial and final dialgebras for strictly positive signatures in Sec. 2. These type constructors are written as

$$\mu(X : \Gamma \multimap * \,; \overrightarrow{\sigma}; \overrightarrow{A}) \qquad \text{and} \qquad \nu(X : \Gamma \multimap * \,; \overrightarrow{\sigma}; \overrightarrow{A}),$$

where $\overrightarrow{\sigma} = \sigma_1, \ldots, \sigma_n$ are tuples of terms, which we will use for substitutions, and $\overrightarrow{A} = A_1, \ldots, A_n$ are types with a free type constructor variable X. In view of the categorical development, the σ_k are the analogue of the morphisms u_k in the base category that were used for reindexing, and the types A_k correspond to the projections of the functor F. Thus $(\overrightarrow{A}, \overrightarrow{\sigma})$ will take the role of a strictly positive signature in the type theory.

Accordingly, we will associate constructors and a recursion scheme to inductive types, and destructors and a corecursion scheme to coinductive types. Suppose, for example, that we have $X : \Gamma \multimap * \mid \Gamma_k \vdash A_k : *$, $\Gamma = x_1 : B_1, \ldots, x_n : B_n$ and $\sigma_k = (t_1, \ldots, t_n)$. The kth constructor of $\mu(X : \Gamma \multimap * \,; \overrightarrow{\sigma}; \overrightarrow{A})$ will have the type, using the shorthand $\mu = \mu(X : \Gamma \multimap * \,; \overrightarrow{\sigma}; \overrightarrow{A})$,

$$\vdash \alpha_k : (\Gamma_k, z : A_k[\mu/X]) \multimap (\mu \mathbin{@} t_1 \mathbin{@} \cdots \mathbin{@} t_m)\,.$$

α_k can now be instantiated according to the parameter context: Suppose, for simplicity, that $\Gamma_k = y : C$ for some type C that does not depend on X,[2] and that we are given a term $\Gamma \vdash s : C$. For a recursive argument $\Gamma \vdash u : A_k[\mu/X][s/y]$, we obtain $\Gamma \vdash \alpha_k \mathbin{@} s \mathbin{@} u : (\mu \mathbin{@} t_1 \mathbin{@} \cdots \mathbin{@} t_m)[s/y]$.

The rest of the type and term constructors are the standard rules one would then expect. It should also be noted that there is a strong similarity in the use of destructors for coinductive types to those in the copattern language in (Abel et al. 2013). Moreover, the

[1] Read: In the term variable context Γ_1, t_k is a term of type B_k.

[2] This is the strict positivity condition we will impose.

$$\frac{\mathbf{D} = \prod_{i=1}^{n} \mathbf{P}_{J_i} \qquad F \in \mathcal{D}_{\mathbf{C}\times\mathbf{P}_I \to \mathbf{D}} \qquad u = (u_1 : J_1 \to I, \ldots, u_n : J_n \to I)}{(F, u) \in \mathcal{S}_{\mathbf{C}\times\mathbf{P}_I \to \mathbf{D}}}$$

$$\frac{A \in \mathbf{P}_J}{K_A^{\mathbf{P}_I} \in \mathcal{D}_{\mathbf{P}_I \to \mathbf{P}_J}} \qquad \frac{\mathbf{C} = \prod_{i=1}^{n} \mathbf{P}_{I_i}}{\pi_k \in \mathcal{D}_{\mathbf{C} \to \mathbf{P}_{I_k}}} \qquad \frac{f : J \to I \text{ in } \mathbf{B}}{f^* \in \mathcal{D}_{\mathbf{P}_I \to \mathbf{P}_J}} \qquad \frac{F_i \in \mathcal{D}_{\mathbf{P}_I \to \mathbf{P}_{J_i}} \qquad i = 1,2}{\langle F_1, F_2 \rangle \in \mathcal{D}_{\mathbf{P}_I \to \mathbf{P}_{J_1} \times \mathbf{P}_{J_2}}}$$

$$\frac{F_1 \in \mathcal{D}_{\mathbf{P}_I \to \mathbf{P}_K} \qquad F_2 \in \mathcal{D}_{\mathbf{P}_K \to \mathbf{P}_J}}{F_2 \circ F_1 \in \mathcal{D}_{\mathbf{P}_I \to \mathbf{P}_J}} \qquad \frac{(F,u) \in \mathcal{S}_{\mathbf{C}\times\mathbf{P}_I \to \mathbf{D}}}{\mu(\widehat{F}, \widehat{G_u}) \in \mathcal{D}_{\mathbf{C} \to \mathbf{P}_I}} \qquad \frac{(F,u) \in \mathcal{S}_{\mathbf{C}\times\mathbf{P}_I \to \mathbf{D}}}{\nu(\widehat{G_u}, \widehat{F}) \in \mathcal{D}_{\mathbf{C} \to \mathbf{P}_I}}$$

Figure 1. Closure rules for data type complete categories

definition scheme for generalised abstract data types in (Hamana and Fiore 2011) describes the same *inductive* types.

We now define the well-formed types and terms of the calculus through several judgements, each of which has its own set of derivations rules. It is understood that the derivability of these judgments is defined by simultaneous induction. So, Definitions 3.1, 3.3, 3.4 and 3.7 should be seen as one simultaneous definition.

It is assumed that we are given two disjoint, countably infinite sets Var and TyVar of term variables and type constructor variables. Term variables will be denoted by $x, y, z, \ldots$, whereas type constructor variables are denoted by capital letters $X, Y, Z, \ldots$. The judgements we are going to use are are the following.

- $\vdash \Theta \;\; \mathbf{TyCtx}$ – The type constructor variable context Θ is well-formed.
- $\vdash \Gamma\ \mathbf{Ctx}$ – The term variable context Γ is well-formed.
- $\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \rightarrowtriangle *$ – The type constructor A is well-formed in the combined context $\Theta \mid \Gamma_1$ and can be *instantiated* with terms according to the *parameter context* Γ_2, where it is implicitly assumed that Θ, Γ_1 and Γ_2 are well-formed.
- $\Gamma_1 \vdash t : \Gamma_2 \rightarrowtriangle A$ – The term t is well-formed in the term variable context Γ_1 and, after instantiating it with arguments according to parameter context Γ_2, is of type A with the arguments substituted into A.
- $\sigma : \Gamma_1 \rhd \Gamma_2$ – The context morphism σ is a well-formed substitution for Γ_2 with terms in context Γ_1.

Definition 3.1 (Well-formed contexts)**.** The judgements for singling out well-formed contexts (type variable contexts and term variable contexts) are given by the following rules.

$$\frac{}{\vdash \varnothing \;\; \mathbf{TyCtx}} \qquad \frac{\vdash \Theta \;\; \mathbf{TyCtx} \qquad \vdash \Gamma\ \mathbf{Ctx}}{\vdash \Theta, X : \Gamma \rightarrowtriangle * \;\; \mathbf{TyCtx}}$$

$$\frac{}{\vdash \varnothing\ \mathbf{Ctx}} \qquad \frac{\varnothing \mid \Gamma \vdash A : *}{\vdash \Gamma, x : A\ \mathbf{Ctx}}$$

Remark 3.2. It is important to note that whenever a term variable declaration is added into the context, its type is not allowed to have any free type constructor variables, which ensures that all types are strictly positive. For example, we are not allowed to form the term context $\Gamma = x : X$ in which X occurs freely. This prevents us, as we will see in Ex. 4.5, from forming function spaces $X \to A$. ◂

Besides the usual notion of context, we also use *parameter contexts*, to bind arguments for which no free variable exists. We borrow the notation from the built-in dependent function space of Agda, only changing the regular arrow used there into $\rightarrowtriangle$ to emphasise that in our calculus this is *not* the function space.

Definition 3.3 (Context Morphism)**.** We introduce the notion of context morphisms as a shorthand notation for sequences of terms. Let Γ_1 and Γ_2 be contexts. A *context morphism* $\sigma : \Gamma_1 \rhd \Gamma_2$ is given by the following two rules.

$$\frac{}{() : \Gamma_1 \rhd \varnothing} \qquad \frac{\sigma : \Gamma_1 \rhd \Gamma_2 \qquad \Gamma_1 \vdash t : A[\sigma]}{(\sigma, t) : \Gamma_1 \rhd (\Gamma_2, x : A)}$$

where $\varnothing \mid \Gamma_2 \vdash A : *$, and $A[\sigma]$ denotes the simultaneous substitution of the terms in σ for the corresponding variables, which is often also denoted by $A[\sigma] = A[\sigma/\vec{x}]$. ◂

Definition 3.4 (Well-formed Type Constructor)**.** The judgement for type constructors is given inductively by the following rules, where it is understood that all involved contexts are well-formed.

$$\frac{}{\vdash \top : *}\ (\top\textbf{-I})$$

$$\frac{\vdash \Theta \;\; \mathbf{TyCtx} \qquad \vdash \Gamma\ \mathbf{Ctx}}{\Theta, X : \Gamma \rightarrowtriangle * \mid \varnothing \vdash X : \Gamma \rightarrowtriangle *}\ (\textbf{TyVar-I})$$

$$\frac{\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \rightarrowtriangle * \qquad \vdash \Gamma\ \mathbf{Ctx}}{\Theta, X : \Gamma \rightarrowtriangle * \mid \Gamma_1 \vdash A : \Gamma_2 \rightarrowtriangle *}\ (\textbf{TyVar-Weak})$$

$$\frac{\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \rightarrowtriangle * \qquad \Theta \mid \Gamma_1 \vdash B : *}{\Theta \mid \Gamma_1, x : B \vdash A : \Gamma_2 \rightarrowtriangle *}\ (\textbf{Ty-Weak})$$

$$\frac{\Theta \mid \Gamma_1 \vdash A : (x : B, \Gamma_2) \rightarrowtriangle * \qquad \Gamma_1 \vdash t : B}{\Theta \mid \Gamma_1 \vdash A \mathbin{@} t : \Gamma_2[t/x] \rightarrowtriangle *}\ (\textbf{Ty-Inst})$$

$$\frac{\Theta \mid \Gamma_1, x : A \vdash B : \Gamma_2 \rightarrowtriangle *}{\Theta \mid \Gamma_1 \vdash (x).\, B : (x : A, \Gamma_2) \rightarrowtriangle *}\ (\textbf{Param-Abstr})$$

$$\frac{\sigma_k : \Gamma_k \rhd \Gamma \qquad \Theta, X : \Gamma \rightarrowtriangle * \mid \Gamma_k \vdash A_k : *}{\Theta \mid \varnothing \vdash \rho(X : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{A}) : \Gamma \rightarrowtriangle *}\ (\textbf{FP-Ty})$$

where in the **(FP-Ty)**-rule, $\rho \in \{\mu, \nu\}$, $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$, $\vec{A} = (A_1, \ldots, A_n)$, and $1 \leqslant k \leqslant n$. ◂

Note that type constructor variables come with a parameter context. This context determines the parameters of an initial/final dialgebra, which essentially bundle the *local* context, the domain and the codomain of their constructors respectively destructors.

This brings us to the rules for term constructions. To introduce them, we need further notations. Part of the definition of the typing rules is a conversion rule, which uses a reduction relation on types.

Definition 3.5. The reduction relation on types consists of two types of reductions: a computation $\longrightarrow$ on terms, which is defined at the end of this section, and β-reduction for parameters. Essentially, parameter abstraction and instantiation for types corresponds to a simply typed λ-calculus on the type level. Thus the β-reduction for parameter instantiations is given by

$$((x).\, A) \mathbin{@} t \longrightarrow_p A[t/x].$$

The reduction relation on terms is lifted to types by taking the compatible closure of reduction of parameters, which is given by

$$\frac{t \longrightarrow t'}{A \,@\, t \longrightarrow A \,@\, t'} \tag{3}$$

We combine these relations into one reduction relation on types: $\longrightarrow_T = \longrightarrow_p \cup \longrightarrow$. One-step conversion of types is given by

$$A \longleftrightarrow_T B \iff A \longrightarrow_T B \text{ or } B \longrightarrow_T A. \tag{4}$$

In the typing rules for terms, we will use the following notation.

Notation 3.6. First, we denote the identity context morphism by $\mathrm{id}_\Gamma := (x_1, \dots, x_n)$ for $\Gamma = x_1 : A_1, \dots, x_n : A_n$. Second, given a type A with parameter context $x_1 : B_1, \dots, x_n : B_n$ and a context morphism $\sigma = (t_1, \dots, t_n)$, we denote by $A \,@\, \sigma$ the instantiation $A \,@\, t_1 \,@ \cdots @\, t_n$.

We continue with the term constructors.

Definition 3.7 (Well-formed Terms)**.** The judgement for terms is given by the rules in Fig. 2. To improve readability, we use the shorthand $\rho = \rho(X : \Gamma \multimap * \,; \overrightarrow{\sigma}; \overrightarrow{A})$, $\rho \in \{\mu, \nu\}$, and implicitly assume all involved types and contexts are well-formed. ◂

We will often leave out the type information in the superscript of constructors and destructors. The domain of a constructor $\alpha_k^{\mu(X:\Gamma \multimap * \,; \overrightarrow{\sigma}; \overrightarrow{A})}$ is determined by A_k and its codomain by the instantiation σ_k. Dually, the domain of a destructor ξ_k is given by the instantiation σ_k and its codomain by A_k.

Finally, we come to the reduction relation. Let us agree on the following notations. Given a context $\Gamma = x_1 : A_1, \dots, x_n : A_n$ and a type $\Theta \mid \Gamma \vdash B : *$, we denote the full parametrisation of B by $(\Gamma).\, B := (x_1). \cdots (x_n).\, B$, giving us

$$\Theta \mid \varnothing \vdash (\Gamma).\, B : \Gamma \multimap *.$$

Moreover, if we are given a sequence $\overrightarrow{B}$ of such types, that is, if $\Theta \mid \Gamma_i \vdash B_i : *$ for each B_i in that sequence, we denote by $\overrightarrow{(\Gamma_i).\, B}$ the sequence of types that arises by fully abstracting each type B_i separately. Finally, we denote by $\overrightarrow{B}[C/X]$ the substitution of C for X in each B_i separately.

For C and $\overrightarrow{A}$ with $\Theta \mid \Gamma' \vdash C : \Gamma \multimap *$, where $\Theta = X_1 : \Gamma_1 \multimap *, \dots, X_n : \Gamma_n \multimap *$, and $\Gamma_i \vdash A_i : *$, we define

$$\widehat{C}(\overrightarrow{A}) = C[\overrightarrow{(\Gamma_i).\, A} / \overrightarrow{X}] \,@\, \mathrm{id}_\Gamma \,.$$

The definition of the reduction relations requires an action of a type constructor C with free type constructor variables on terms. Since this action will be used like a functor, we define it (in Def. 5.2) in such a way that the following typing rule holds.

$$\frac{X : \Gamma_1 \multimap * \mid \Gamma_2' \vdash C : \Gamma_2 \multimap * \qquad \Gamma_1, x : A \vdash t : B}{\Gamma_2', \Gamma_2, x : \widehat{C}(A) \vdash \widehat{C}(t) : \widehat{C}(B)} \tag{5}$$

In the definition of the reduction relation we need to compose context morphisms. This composition is for $\Gamma_3 \overset{\sigma}{\rhd} \Gamma_2 \overset{\tau}{\rhd} \Gamma_1$ and $\Gamma_1 = x_1 : A_1, \dots, x_n : A_n$ defined by

$$\tau \bullet \sigma := (\tau_1[\sigma], \dots, \tau_n[\sigma]). \tag{6}$$

We can now define the reduction relation on terms.

Definition 3.8. The reduction relation $\longrightarrow$ on terms is defined as compatible closure of the contraction relation $\succ$ given in Fig. 3. We introduce in the definition of contraction a fresh variable x, for which we immediately substitute (either u or g_k). This is necessary for the use of the action of types on terms, see (5).

Remark 3.9. On terms, the reduction relation for a destructor-corecursor pair essentially emulates the commutativity of the following diagram (in context Γ_k). The dual reduction relation for a recursor-constructor pair emulates the dual of this diagram.

$$\begin{tikzcd}
C \,@\, \sigma_k \arrow[r, "\mathrm{corec}\ \overrightarrow{(\Gamma_k, y).\, g_k} \,@\, \sigma_k \,@\, x"] \arrow[d, "g_k"'] & \nu \,@\, \sigma_k \arrow[d, "\xi_k \,@\, \mathrm{id}_{\Gamma_k} \,@\, y"] \\
A_k[C/X] \arrow[r, "\widehat{A_k}(\mathrm{corec}\ \overrightarrow{(\Gamma_k, y).\, g} \,@\, \sigma_k \,@\, x)"] & A_k[\nu/X]
\end{tikzcd}$$

This concludes the definition of our proposed calculus. Note that there are no primitive type constructors for →-, Π- or ∃-types, all of these are, together with the corresponding introduction and elimination principles, definable in the above calculus.

Remark 3.10. As the alert reader might have noticed, our calculus does not have dependent recursion and corecursion, that is, the type C in **(Ind-E)** and **(Coind-I)** cannot depend on elements of the corresponding recursive type. This clearly makes the calculus weaker than if we had the dependent version: In the case of inductive types we do not have an induction principle, c.f. Ex. 4.6. For coinductive types, on the other hand, one cannot even formulate a dependent version of **(Coind-I)**, rather one would expect a coinduction rule that turns a bisimulation into an equality proof. This would imply that we have an extensional function space, see Ex. 4.5.

4. Examples

In this section, we illustrate the calculus given in Sec. 3 on a variety of examples. We begin with a few basic ones, then work our way through the encoding of logical operators, and finish with data types that are actually recursive. Those recursive data types are lists indexed by their length (vectors) and their dual, partial streams indexed by their definition depth. This last example illustrates how our dependent coinductive types generalise existing ones.

Before we go through the examples, let us introduce a notation for sequences of empty context morphisms. We denote such a sequence of k empty context morphisms by

$$\varepsilon_k := ((), \dots, ()).$$

In the first example, we explain the role of the basic type $\top$.

Example 4.1 (Terminal Object)**.** We first note that, in principle, we can encode $\top$ as a coinductive type by $\mathbf{1} := \nu(X : * \,; \varepsilon_1; X)$:

$$\frac{X : * \mid \varnothing \vdash X : *}{\vdash \nu(X : * \,; \varepsilon_1; X) : *}$$

This gives us the destructor $\xi_1 : (y : \mathbf{1}) \multimap \mathbf{1}$ and the inference

$$\frac{\vdash C : * \qquad y : C \vdash y : C}{\vdash \mathrm{corec}\ ((y).\, y) : (y : C) \multimap \mathbf{1}}\ \textbf{(Coind-I)}$$

So the analogue of the categorical concept of the morphism into a final object is given by $!_C := \mathrm{corec}\ ((y).\, y)$. Note that it is not possible to define a closed term of type $\mathbf{1}$ directly, rather we only get one with the help of $\top$ by $\diamondsuit' := !_\top \,@\, \diamondsuit$. Thus the puropose of $\top$ is to allow the formation of closed terms. Now, these definitions and $\widehat{X}(t) = t$, see Def. 5.2, give us the following reduction.

$$\begin{aligned}
\xi_1 \,@\, \diamondsuit' &= \xi_1 \,@\, (\mathrm{corec}\ ((y).\, y) \,@\, \diamondsuit) \\
&\longrightarrow \widehat{X}(\mathrm{corec}\ ((y).\, y) \,@\, x')[y/x'][\diamondsuit/y] \\
&= (\mathrm{corec}\ ((y).\, y) \,@\, x')[y/x'][\diamondsuit/y] \\
&= \mathrm{corec}\ ((y).\, y) \,@\, \diamondsuit \\
&= \diamondsuit'
\end{aligned}$$

Thus $\diamondsuit'$ is the canonical element with no observable behaviour. ◂

Dual to the terminal object $\mathbf{1}$, we can form an initial object.

Example 4.2. We put $\mathbf{0} := \bot := \mu(X : * \,; \varepsilon_1; X)$, dual to the definition of $\mathbf{1}$. If we define $E_C^\bot := \mathrm{rec}\ ((y).\, y)$, we get the usual

$$\frac{}{\vdash \diamondsuit : \top}\,(\top\textbf{-I}) \qquad \frac{\Gamma_1 \vdash t : (x : A, \Gamma_2) \twoheadrightarrow B \qquad \Gamma_1 \vdash s : A}{\Gamma_1 \vdash t \,@\, s : \Gamma_2[s/x] \twoheadrightarrow B[s/x]}\,\textbf{(Inst)} \qquad \frac{\Gamma \vdash t : A \qquad A \longleftrightarrow_T B}{\Gamma \vdash t : B}\,\textbf{(Conv)}$$

$$\frac{\Gamma \vdash A : *}{\Gamma, x : A \vdash x : A}\,\textbf{(Proj)} \qquad \frac{\Gamma_1 \vdash t : \Gamma_2 \twoheadrightarrow A \qquad \Gamma_1 \vdash B : *}{\Gamma_1, x : B \vdash t : \Gamma_2 \twoheadrightarrow A}\,\textbf{(Term-Weak)}$$

$$\frac{\vdash \mu(X : \Gamma \twoheadrightarrow * \,; \vec{\sigma}; \vec{A}) : \Gamma \twoheadrightarrow * \qquad 1 \leqslant k \leqslant |\vec{A}|}{\vdash \alpha_k^{\mu(X:\Gamma\twoheadrightarrow *\,;\vec{\sigma};\vec{A})} : (\Gamma_k, y : A_k[\mu/X]) \twoheadrightarrow \mu \,@\, \sigma_k}\,\textbf{(Ind-I)} \qquad \frac{\vdash \nu(X : \Gamma \twoheadrightarrow * \,; \vec{\sigma}; \vec{A}) : \Gamma \twoheadrightarrow * \qquad 1 \leqslant k \leqslant |\vec{A}|}{\vdash \xi_k^{\nu(X:\Gamma\twoheadrightarrow *\,;\vec{\sigma};\vec{A})} : (\Gamma_k, y : \nu \,@\, \sigma_k) \twoheadrightarrow A_k[\nu/X]}\,\textbf{(Coind-E)}$$

$$\frac{\vdash C : \Gamma \twoheadrightarrow * \qquad \Delta, \Gamma_k, y_k : A_k[C/X] \vdash g_k : (C \,@\, \sigma_k) \qquad \forall k = 1, \ldots, n}{\Delta \vdash \mathrm{rec}\ \overrightarrow{(\Gamma_k, y_k).\, g_k} : (\Gamma, y : \mu \,@\, \mathrm{id}_\Gamma) \twoheadrightarrow C \,@\, \mathrm{id}_\Gamma}\,\textbf{(Ind-E)}$$

$$\frac{\vdash C : \Gamma \twoheadrightarrow * \qquad \Delta, \Gamma_k, y_k : (C \,@\, \sigma_k) \vdash g_k : A_k[C/X] \qquad \forall k = 1, \ldots, n}{\Delta \vdash \mathrm{corec}\ \overrightarrow{(\Gamma_k, y_k).\, g_k} : (\Gamma, y : C \,@\, \mathrm{id}_\Gamma) \twoheadrightarrow \nu \,@\, \mathrm{id}_\Gamma}\,\textbf{(Coind-I)}$$

Figure 2. Judgements for well-formed terms

$$\mathrm{rec}\ \overrightarrow{(\Gamma_k, y_k).\, g_k} \,@\, (\sigma_k \bullet \tau) \,@\, (\alpha_k \,@\, \tau \,@\, u) \succ g_k\Big[\widehat{A_k}(\mathrm{rec}\ \overrightarrow{(\Gamma_k, y_k).\, g_k} \,@\, \mathrm{id}_\Gamma \,@\, x)\Big/ y_k\Big][\tau, u]$$

$$\xi_k \,@\, \tau \,@\, (\mathrm{corec}\ \overrightarrow{(\Gamma_k, y_k).\, g_k} \,@\, (\sigma_k \bullet \tau) \,@\, u) \succ \widehat{A_k}\left(\mathrm{corec}\ \overrightarrow{(\Gamma_k, y_k).\, g_k} \,@\, \mathrm{id}_\Gamma \,@\, x\right)[g_k/x][\tau, u]$$

Figure 3. Contraction of Terms.

elimination principle for falsum:

$$\frac{\vdash C : *}{\vdash E_C^\perp : (y : \perp) \twoheadrightarrow C}$$

Another example of a basic type are the natural numbers.

Example 4.3. We can define the type of natural numbers by

$$\mathbf{N} := \mu(X : * \,; \varepsilon_2; (\mathbf{1}, X)),$$

with contexts $\Gamma = \Gamma_1 = \Gamma_2 = \varnothing$. We get the usual constructors:

$$0 = \alpha_1^{\mathbf{N}} \,@\, \diamondsuit : \mathbf{N} \quad \text{and} \quad s = \alpha_2^{\mathbf{N}} : (y : \mathbf{N}) \twoheadrightarrow \mathbf{N}\,.$$

Moreover, we obtain the usual recursion principle:

$$\frac{\vdash t_0 : C \qquad y : C \vdash t_s : C}{\vdash \mathrm{rec}\,(t_0, (y).\, t_s) : (y : \mathbf{N}) \twoheadrightarrow C}$$

Let us now move to logical operators.

Example 4.4 (Binary Product and Coproduct)**.** Suppose we are given types $\Gamma \vdash A_1, A_2 : *$, then their binary product is fully specified by the two projections and pairing. Thus, we can use the following coinductive type for $A_1 \times_\Gamma A_2$.

$$\frac{\Gamma \vdash A_1 : * \qquad \Gamma \vdash A_2 : *}{\Gamma \vdash \nu(X : \Gamma \twoheadrightarrow * \,; (\mathrm{id}_\Gamma, \mathrm{id}_\Gamma); (A_1, A_2)) \,@\, \mathrm{id}_\Gamma : *}$$

Let us abbreviate $P := \mathrm{corec}\,((\Gamma, \_).\, t_1, (\Gamma, \_).\, t_2)$, then the projections are then given by $\pi_k := \xi_k \,@\, \mathrm{id}_\Gamma$, and pairing by $(t_1, t_2) := P \,@\, \mathrm{id}_\Gamma \,@\, \diamondsuit$. We have

$$\cfrac{\cfrac{\vdash (\Gamma).\, \top : \Gamma \twoheadrightarrow * \qquad \cfrac{\Gamma \vdash t_k : A_k}{\Gamma, \_ : \top \vdash t_k : A_k}}{\vdash P : (\Gamma, \_ : \top) \twoheadrightarrow A_1 \times_\Gamma A_2}}{\Gamma \vdash (t_1, t_2) : A_1 \times_\Gamma A_2}$$

This setup will give us the expected reduction:

$$\begin{aligned} \pi_k \,@\, (t_1, t_2) &= \xi_k \,@\, \mathrm{id}_\Gamma \,@\, (P \,@\, \mathrm{id}_\Gamma \,@\, \diamondsuit) \\ &\longrightarrow \widehat{A_k}(P \,@\, \mathrm{id}_\Gamma \,@\, x)[t_k/x][(\mathrm{id}_\Gamma, \diamondsuit)] \\ &= x[t_k/x][(\mathrm{id}_\Gamma, \diamondsuit)] \\ &= t_k, \end{aligned}$$

where the third step is given by $\widehat{A_k} = x$, since A_k does not use type constructor variables, see Def. 5.2.

Dually, the binary coproduct of A_1 and A_2 is given by

$$A_1 +_\Gamma A_2 := \mu(X : \Gamma \twoheadrightarrow * \,; (\mathrm{id}_\Gamma, \mathrm{id}_\Gamma); (A_1, A_2)) \,@\, \mathrm{id}_\Gamma,$$

the corresponding injections by $\kappa_i := \alpha_i \,@\, \mathrm{id}_\Gamma$, and we can form the case distinction $[t_1, t_2] := \mathrm{rec}\,((\Gamma, x).\, t_1, (\Gamma, x).\, t_2)$ subject to the following typing rule.

$$\frac{\Gamma \vdash C : * \qquad \Gamma, x : A_k \vdash t_k : C}{\Gamma \vdash [t_1, t_2] : (x : A_1 +_\Gamma A_2) \twoheadrightarrow C}$$

Moreover, we get the expected reduction:

$$[t_1, t_2] \,@\, (\kappa_i \,@\, s) \longrightarrow t_i[s/x].$$

We now show that the function space arises as coinductive type.

Example 4.5 (Dependent Product, Π-types, Function Space)**.** We use $\varnothing$ as global context and $\Gamma_1 = x : A$ as local context, thus

$$\cfrac{\cfrac{\varnothing \mid x : A \vdash B : *}{\_ : * \mid x : A \vdash B : *} \qquad \varepsilon_1 : \Gamma_1 \rhd \varnothing}{\vdash \nu(\_ : * \,; \varepsilon_1; B) : *}\,\textbf{(FP-Ty)}$$

If we define $\Pi x : A.B := \nu(\_ : * \,; \varepsilon_1; B)$, then $\vdash \Pi x : A.B : *$. We get λ-abstraction by putting $\lambda x.g := \mathrm{corec}\ ((x, \_).\, g) \,@\, \diamondsuit$:

$$\cfrac{\cfrac{\cfrac{\Gamma, x : A \vdash g : B \qquad \Gamma, x : A \vdash \top : *}{\Gamma, x : A, \_ : \top \vdash g : B}\,\textbf{(Weak)}}{\Gamma \vdash \mathrm{corec}\ ((x, \_).\, g) : (y : \top) \twoheadrightarrow \Pi x : A.B}\,\textbf{(Coind-I)}}{\Gamma \vdash \lambda x.g : \Pi x : A.B}\,\textbf{(Inst)}$$

Application is given by $t\,a := \xi_1 @ a @ t$. Since we have that $B[\Pi x : A.B/\_] = B$ and $(\Pi x : A.B)[\varepsilon_1] = \Pi x : A.B$, we can derive the usual typing rule for application:

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash t : \Pi x : A.B}{\Gamma \vdash t\,a : B[a/x]}$$

In particular, we have that $(\lambda x.g)\,a$ is well-typed, and we can derive the usual β-reduction:

$$\begin{aligned}(\lambda x.g)\,a &= \xi_1 @ a @ (\text{corec } ((x, \_).\, g) @ \diamondsuit) \\ &\longrightarrow \widehat{B}(\text{corec } ((x, \_).\, g) @ x')[g/x'][\diamondsuit/\_, a/x] \\ &= x'[g/x'][\diamondsuit/\_, a/x] \\ &= g[a/x],\end{aligned}$$

where we again use that $\_ \notin \text{fv}(B)$. As usual, we can derive from the dependent function space also the non-dependent one by

$$A \to B := \Pi x : A.B \text{ if } x \notin \text{fv}(B).$$

We can now extend the usual correspondence between variables in context and terms of function type to parameters as follows. First, from $\Gamma \vdash r : (x : A) \twoheadrightarrow B$, we get $\Gamma, x : A \vdash r @ x : B$. Next, for $\Gamma, x : A \vdash s : B$ we can form $\lambda x.s$, and finally a term $\Gamma \vdash t : A \to B$ gives rise to $\Gamma, x : A \vdash t\,x : B$. This situation can be summarised as follows.

$$\begin{array}{c}\Gamma \vdash r : (x : A) \twoheadrightarrow B \\ \hline \Gamma, x : A \vdash s : B \\ \hline\hline \Gamma \vdash t : \Pi x : A.B\end{array}$$

Here, a single line is a downwards correspondence, and the dashed double line signifies a two-way correspondence. These correspondences allow us, for example, to give the product projections the function type $A_1 \times_\Gamma A_2 \to A_k$, and to write $\pi_k\, t$ instead of $\pi_k @ t$.

Finally, let us illustrate how the type system only allows the formation of strictly positive types. Suppose we would want to form the type $X \to B$ for some variable X and type B. Recall that $X \to B = \nu(\_ : *; \varepsilon_1; B)$ with $\varepsilon_1 : (x : X) \rhd \varnothing$. However, for this to be a valid context morphism, we would need to derive $\vdash x : X$ **Ctx** (note the empty context), which is not possible according to Def. 3.1. Hence, $X \to B$ cannot be formed. ◂

Example 4.6 (Coproducts, Existential Quantifier)**.** Recall that we do not have dependent recursion, hence no induction principle. This means that we are not able to encode Σ-types à la Martin-Löf. Instead, we can define intuitionistic existential quantifiers, see 11.4.4 and 10.8.2 in (Troelstra and van Dalen 1988). In fact, ∃-types occur as the dual of Π-types (Ex. 4.5) as follows.

Let $x : A \vdash B : *$ and put $\exists x : A.B := \mu(\_ : *; \varepsilon_1; B)$. The pairing of terms t and s is given by $(t, s) := \alpha_1 @ t @ s$. One can easily derive that $\vdash \exists x : A.B : *$ and

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash s : B[t/x]}{\Gamma \vdash (t, s) : \exists x : A.B}$$

from **(Ind-I)** and **(Inst)**. Equally easy is also the derivation that the elimination principle for existential quantifiers, defined by

$$E^{\exists}_{x,y}(t, p) := \text{rec } ((x : A, y : B).\, p) @ t,$$

can be formed by the following rule.

$$\frac{\vdash C : * \qquad \Gamma, x : A, y : B \vdash p : C \qquad \Gamma \vdash t : \exists x : A.B}{\Gamma \vdash E^{\exists}_{x,y}(t, p) : C}$$

Finally, we get the usual reduction rule

$$E^{\exists}_{x,y}((t, s), p) \longrightarrow p\,[\,t/x \;\; s/y\,]\,. \qquad ◂$$

Example 4.7 (Generalised Dependent Product and Coproduct)**.** From a categorical perspective, it makes sense to not just consider product and coproducts that bind a variable in a type but also to allow the restriction of terms we allow as values for this variable. We can achieve this by replacing ε_1 in Ex. 4.5 and Ex. 4.6 by an arbitrary term $x : I \vdash f : J$. This gives us type constructors with

$$y : J \vdash \textstyle\coprod_f A : * \quad \text{and} \quad y : J \vdash \prod_f A : *$$

that are weakly adjoint $\coprod_f \dashv f^* \dashv \prod_f$, where f^* substitutes f. Similarly, propositional equality arises as left adjoint to contraction δ^*, where $\delta : (x : A) \rhd (x : A, y : A)$ is the diagonal substitution $\delta := (x, x)$, c.f. (Jacobs 1999, Def. 10.5.1). ◂

The next example is a standard inductive dependent type.

Example 4.8 (Vectors)**.** We define vectors $\text{Vec}\,A : (n : \mathbf{N}) \twoheadrightarrow *$, which are lists over A indexed by their length, by

$$\begin{aligned}&\text{Vec}\,A := \mu(X : \Gamma \twoheadrightarrow *; (\sigma_1, \sigma_2); (\mathbf{1}, A \times X @ k)) \\ &\Gamma = n : \mathbf{N} \quad \text{and} \quad \Gamma_1 = \varnothing \quad \text{and} \quad \Gamma_2 = k : \mathbf{N} \\ &\sigma_1 = (0) : \Gamma_1 \rhd (n : \mathbf{N}) \quad \text{and} \quad \sigma_2 = (s @ k) : \Gamma_2 \rhd (n : \mathbf{N}) \\ &X : (n : \mathbf{N}) \twoheadrightarrow * \mid \Gamma_1 \vdash \mathbf{1} : * \\ &X : (n : \mathbf{N}) \twoheadrightarrow * \mid \Gamma_2 \vdash A \times X @ k : *\end{aligned}$$

This yields the usual constructors $\text{nil} := \alpha_1 @ \diamondsuit$ and $\text{cons} := \alpha_2$, which have the expected types, namely $\alpha_1 : \mathbf{1} \twoheadrightarrow \text{Vec}\,A\ 0$ and $\alpha_2 : (k : \mathbf{N}, y : A \times \text{Vec}\,A\ k) \twoheadrightarrow \text{Vec}\,A @ (s @ k)$. The induced recursion scheme is then also the expected one. ◂

The dependent coinductive types of the present calculus differ from other calculi in that destructors can be restricted in the domain they may be applied to. We illustrate this by defining partially defined streams, which are the dual of vectors. A preliminary definition is necessary though.

Example 4.9. The *extended naturals*, which are to be thought of as natural numbers extendend with an element ∞, are given by the following coinductive type.

$$\mathbf{N}^\infty = \nu(X : *; \varepsilon_1; \top + X) : *$$

On this type, we can define the successor $s_\infty : y : \mathbf{N}^\infty \to \mathbf{N}^\infty$ by primitive corecursion.

Using the extended naturals, we can define partial streams, which are streams that might not be fully defined.

Example 4.10. Intuitively, we define partial streams as coinductive type indexed by the definition depth, and destructors that can only be applied to streams that are defined in at least the first position:

```
codata PStr (A : Set) : (n : N∞) → Set where
  hd : (k : N∞) → PStr A (s∞ k) → A
  tl : (k : N∞) → PStr A (s∞ k) → PStr A k
```

This co-datatype translates into our language by putting

$$\text{PStr}\,A := \nu(X : \Gamma \twoheadrightarrow *; (s_\infty\, k, s_\infty\, k); (A, X @ k)),$$

for contexts $\Gamma = n : \mathbf{N}^\infty$ and $\Gamma_1 = \Gamma_2 = k : \mathbf{N}^\infty$, context morphisms $(s_\infty\, k) : \Gamma_i \rhd \Gamma$ for $i = 1, 2$, and destructor codomains $X : \Gamma \twoheadrightarrow * \mid \Gamma_1 \vdash A : *$ and $X : \Gamma \twoheadrightarrow * \mid \Gamma_1 \vdash X @ k : *$. ◂

5. Pre-Types and Pre-Terms

The pre-types and pre-terms, we introduce in this section, have two purposes: First, they allow us to justify the simultaneous definition of the typing judgement and the reduction relation in Sec. 3. Second, we use them as a tool in Sec. 6.2 to prove strong normalisation.

Pre-types and -terms are introduced as follows. First, we define raw terms that mix types and terms, and raw contexts whose only purpose is to ensure that arities for instantiations match. These raw terms can then be split into pre-types and pre-terms.

Definition 5.1 (Raw Syntax)**.** The raw contexts and terms are given by the follows grammars.

$$
\begin{aligned}
\Gamma &:= \varnothing \mid \Gamma, x \quad x \in \mathrm{Var} \\
\Theta &:= \varnothing \mid \Theta, X : \Gamma \twoheadrightarrow * \\
M, N &:= \top \mid \Diamond \mid x \in \mathrm{Var} \mid M @ N \mid (x).\, M \mid X \in \mathrm{TyVar} \\
&\quad \mid \alpha_k \mid \xi_k \mid \rho(X : \Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{M}),\ \rho \in \{\mu, \nu\} \\
&\quad \mid \mathrm{rec}^{\mu(X:\Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{M})}\ \overrightarrow{(\Gamma_k, y_k).\, N_k} \\
&\quad \mid \mathrm{corec}^{\nu(X:\Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{M})}\ \overrightarrow{(\Gamma_k, y_k).\, N_k}
\end{aligned}
$$

Pre-types and pre-terms are defined through two judgements

$$\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow * \qquad \text{and} \qquad \Gamma_1, x \vdash t : \Gamma_2 \twoheadrightarrow \Box.$$

The rules for these judgements follow essentially those in Sec. 3, only that the type for terms is erased. For that reason, we leave their definition out. However, this definition and that of the reduction relation below have been implemented in Agda (Basold 2016).

Recall that the contraction of terms, see Fig. 3, requires an action of (pre-)types with free type variables on terms. We define it so that (5) on page 5 holds, see Lem. 6.1. In fact, the definition for recursive types just follows how functors arise from parameterised initial and final dialgebras, see Def. 2.6 and (Basold 2015).

Definition 5.2 (Type action)**.** Let $\Theta \mid \Gamma' \vdash C : \Gamma \twoheadrightarrow *$ be a pre-type with $\Theta = X_1 : \Gamma_1 \twoheadrightarrow *, \dots, X_n : \Gamma_n \twoheadrightarrow *$, $\vec{A}$ and $\vec{B}$ be sequences of pre-types with $\Gamma_i \vdash A_i : *$ and $\Gamma_i \vdash B_i : *$ for all $1 \leqslant i \leqslant n$. For a sequence $\vec{t}$ of pre-terms with $\Gamma_i, x \vdash t : \Box$ for all $1 \leqslant i \leqslant n$, we define $\widehat{C}(\vec{t})$ as follows. If $n = 0$, we simply put $\widehat{C}(\varepsilon) = x$. If $n > 0$, we define $\widehat{C}(\vec{t})$ by induction in the derivation of C.

$$
\begin{aligned}
&\widehat{C}(\vec{t}, t_{n+1}) = \widehat{C}(\vec{t}) && \text{for } \textbf{(TyVar-Weak)} \\
&\widehat{X_i}(\vec{t}) = t_i \\
&\widehat{C' @ s}(\vec{t}) = \widehat{C'}(\vec{t})[s/y], && \text{for } \Theta \mid \Gamma' \vdash C' : (y, \Gamma) \twoheadrightarrow * \\
&\widehat{(y).\, C'}(\vec{t}) = \widehat{C'}(\vec{t}), && \text{for } \Theta \mid (\Gamma', y) \vdash C' : \Gamma \twoheadrightarrow * \\
&\widehat{\mu(Y : \Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{D})}(\vec{t}) = \mathrm{rec}^{R_A}\ \overrightarrow{(\Delta_k, x).\, g_k} @ \mathrm{id}_\Gamma @ x, \\
&\quad \text{with } g_k = \alpha_k @ \mathrm{id}_{\Delta_k} @ \left(\widehat{D_k}\left(\vec{t}, y\right)\right) \\
&\quad \text{and } R_A = \mu(Y : \Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{D}[\overrightarrow{(\Gamma_i).\, A/X}]) \\
&\quad \text{for } \Theta, Y : \Gamma \twoheadrightarrow * \mid \Delta_k \vdash D_k : * \\
&\widehat{\nu(Y : \Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{D})}(\vec{t}) = \mathrm{corec}^{R_B}\ \overrightarrow{(\Delta_k, x).\, g} @ \mathrm{id}_\Gamma @ x, \\
&\quad \text{with } g_k = \widehat{D_k}\left(\vec{t}, x\right)[(\xi_k @ \mathrm{id}_{\Delta_k} @ x)/x] \\
&\quad \text{and } R_B = \nu(Y : \Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{D}[\overrightarrow{(\Gamma_i).\, B/X}])
\end{aligned}
$$

6. Meta properties

6.1 Subject Reduction

The proof of subject reduction is based on the following key lemma, which essentially states that the action of types on terms acts like a functor.

Lemma 6.1 (Type correctness of type action)**.** *Given the action of types on terms, see Def. 5.2, the following inference rule holds.*

$$\frac{X : \Gamma_1 \twoheadrightarrow * \mid \Gamma_2' \vdash C : \Gamma_2 \twoheadrightarrow * \qquad \Gamma_1, x : A \vdash t : B}{\Gamma_2', \Gamma_2, x : \widehat{C}(A) \vdash \widehat{C}(t) : \widehat{C}(B)}$$

Proof. We leave the proof details out. Let us only mention that the proof works by generalising the statement to types in arbitrary type constructor contexts Θ, and then proceeding by induction in C. $\Box$

The following is now an easy consequence of Lem. 6.1.

Theorem 6.2 (Subject reduction)**.** *If $\Gamma \vdash t_1 : A$ and $t_1 \longrightarrow t_2$, then $\Gamma \vdash t_2 : A$.*

6.2 Strong Normalisation

This section is devoted to show that all terms $\Gamma \vdash t : A$ are strongly normalising, which is intuitively what one would expect, given that we introduced the reduction relation by following the homomorphism property of (co)recursion for initial and final dialgebras.

The proof uses the saturated sets approach, see (Geuvers 1994), as follows. First, we define what it means for a set of pre-terms to be saturated, where, most importantly, all terms in a saturated set are strongly normalising. Next, we give an interpretation $[\![A]\!]$ of dependent types A as families of saturated sets. Finally, we show that if $\Gamma \vdash t : A$, then for all assignments ρ of terms to variables in Γ, we have $t \in [\![A]\!](\rho)$. Since $[\![A]\!](\rho) \subseteq \mathbf{SN}$, strong normalisation for all typed terms follows.

We begin with a few simple definitions.

Definition 6.3. We use the following notations.

- Λ is the set of pre-terms.
- $\mathbf{SN}$ is the set of strongly normalising pre-terms.
- $\lceil \Gamma \rceil$ is the set of variables in context Γ.

For simplicity, we identify context morphisms $\sigma : \Gamma_1 \rhd \Gamma_2$ and valuations $\rho : \lceil \Gamma_2 \rceil \to \Lambda$, if we know that the terms of ρ live in Γ_1. This allows us to write $\sigma(x)$ for $x \in \lceil \Gamma_2 \rceil$, and $M @ \rho$ for pre-terms M. It is helpful though to make the action of context morphisms on valuations, essentially given by composition, explicit and write

$$
\begin{aligned}
&[\![\sigma : \Gamma_1 \rhd \Gamma_2]\!] : \Lambda^{\lceil \Gamma_1 \rceil} \to \Lambda^{\lceil \Gamma_2 \rceil} \\
&[\![\sigma : \Gamma_1 \rhd \Gamma_2]\!](\gamma)(y) = \sigma(y)[\gamma].
\end{aligned} \tag{7}
$$

Saturated sets are defined by containing certain open terms (base terms) and by being closed under key reductions. We introduce these two notions in the following two definitions.

Definition 6.4 (Base Terms)**.** The set of *base terms* $\mathcal{B}$ is defined inductively by the following three closure rules.

- $\mathrm{Var} \subseteq \mathcal{B}$
- $\mathrm{rec}\ \overrightarrow{(\Gamma_k, x).\, N_k} @ \sigma @ M \in \mathcal{B}$, provided that $M \in \mathcal{B}$, $N_k \in \mathbf{SN}$ and $\sigma \in \mathbf{SN}$.
- $\xi_k^{\nu(X:\Gamma \twoheadrightarrow * ; \vec{\tau}; \vec{A})} @ \sigma @ M \in \mathcal{B}$, provided that $M \in \mathcal{B}$, $\sigma \in \mathbf{SN}$ and $\exists \gamma.\, (\sigma = [\![\tau_k]\!](\gamma))$.

Definition 6.5 (Key Redex)**.** A pre-term M is a *redex*, if there is a P with $M \succ P$. M is the *key redex*

1. of M itself, if M is a redex,
2. of $\mathrm{rec}\ \overrightarrow{(\Gamma_k, y_k).\, N_k} @ \sigma @ N$, if M the key redex of N, or
3. of $\xi_k @ \sigma @ N$, if M the key redex of N.

We denote by $\mathrm{red}_k(M)$ the term that is obtained by contracting the key redex of M.

Definition 6.6 (Saturated Sets)**.** A set $X \subseteq \Lambda$ is *saturated*, if

1. $X \subseteq \mathbf{SN}$
2. $\mathcal{B} \subseteq X$
3. If $\mathrm{red}_k(M) \in X$ and $M \in \mathbf{SN}$, then $M \in X$.

We denote by $\mathbf{SAT}$ the set of all saturated sets.

It is easy to see that $\mathbf{SN} \in \mathbf{SAT}$, and that every saturated set is non-empty. Moreover, it is easy to show that $\mathbf{SAT}$ is a complete lattice with set inclusion as order. Besides these standard facts, we will use the following constructions on saturated sets.

Definition 6.7. Let Γ be a context. We define a form of semantical context extension (comprehension) of pairs (E, U) with $E \subseteq \Lambda^{\lceil\Gamma\rceil}$ and $U : E \to \mathbf{SAT}$ with respect to a given variable $x \notin \lceil\Gamma\rceil$ by

$$\{(E, U)\}_x = \{\rho[x \mapsto M] \mid \rho \in E \text{ and } M \in U(\rho)\}, \qquad (8)$$

where $\rho[x \mapsto M] : \lceil\Gamma\rceil \cup \{x\} \to \Lambda$ extends ρ by mapping x to M. Moreover, we define a semantical version of the typing judgement:

$$E \Vdash U = \{M \mid \forall \gamma \in E.\ M[\gamma] \in U(\gamma)\}. \qquad (9)$$

We now show that we can give a model of well-formed types by means of saturated sets. To achieve this, we define simultaneously an interpretation of contexts and the interpretation of types. The intention is that we have that

- if $\vdash \Gamma\ \mathbf{Ctx}$, then $[\![\Gamma]\!] \subseteq \Lambda^{\lceil\Gamma\rceil}$,
- if $\vdash \Theta\ \ \mathbf{TyCtx}$, then $[\![\Theta]\!](X) \in \mathbf{SAT}^{[\![\Gamma]\!]}$ for all $X : \Gamma \twoheadrightarrow *$ in Θ, and
- if $\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow *$, then $[\![A]\!] : [\![\Theta]\!] \times [\![\Gamma_1, \Gamma_2]\!] \to \mathbf{SAT}$.

Definition 6.8 (Interpretations)**.** We interpret type variable contexts, term variable contexts and types simultaneously. First, we assign to each term context is a set of allowed possible valuations:

$$\begin{aligned}
[\![\varnothing]\!] &= \{! : \varnothing \to \Lambda\} \\
[\![\Gamma, x : A]\!] &= \{([\![\Gamma]\!], [\![A]\!])\}_x \\
&= \{\rho[x \mapsto M] \mid \rho \in [\![\Gamma]\!] \text{ and } M \in [\![A]\!](\rho)\}
\end{aligned}$$

For $\Theta = X_1 : \Gamma_1 \twoheadrightarrow *, \ldots, X_n : \Gamma_n \twoheadrightarrow *$ we define

$$[\![\Theta]\!] = \prod_{X_i \in \lceil\Theta\rceil} I_{\Gamma_i},$$

where I_Γ is the set of valuations that respect convertibility:

$$I_\Gamma = \{U : [\![\Gamma]\!] \to \mathbf{SAT} \mid \forall \rho, \rho'.\ \rho \longrightarrow_T \rho' \Rightarrow U(\rho) = U(\rho')\}$$

Finally, we define in Fig. 4 the interpretation of types as families of term sets. In the clause for inductive types, A_k^{Δ} denotes the type that is obtained by weakening $\Gamma_k \vdash A_k : *$ to $\Delta, \Gamma_k \vdash A_k^{\Delta} : *$, $\pi : (\Gamma_k, y : A_k^{\Delta}) \rhd \Gamma_k$ projects y away, and $[\![\sigma_k \bullet \pi]\!]^*(U) := U \circ [\![\sigma_k \bullet \pi]\!]$ is the reindexing for set families. ◂

Before we continue stating the key results about this interpretation of types, let us briefly look at an example.

Example 6.9. Suppose A, B are closed types. Recall that the function space was $A \to B = \nu(X : * ; \varepsilon_1 : (x : A) \rhd \varnothing; B)$, and that application was defined by $t\,a = \xi_1 @ a @ t$. Note that the condition $\gamma \in [\![\varepsilon_1]\!]^{-1}(\rho)$ reduces to $\gamma(x) \in [\![A]\!]$ because $[\![\varepsilon_1]\!](\gamma)(y \in \varnothing) = \rho(y)[\varepsilon_1]$ holds for any $\gamma \in [\![(x : A)]\!]$. So we write N instead of $\gamma(x)$. We further note that, since A, B and thus $A \to B$ are closed, we can leave out the valuation δ for the type variables. Taking all of this into account, we have

$$\begin{aligned}
[\![A \to B]\!](\gamma) &= \{M \mid \forall N \in [\![A]\!].\ \xi_1 @ \gamma @ M \in [\![B]\!]\} \\
&= \{M \mid \forall N \in [\![A]\!].\ M\,N \in [\![B]\!]\},
\end{aligned}$$

which is the usual definition definition, see (Geuvers 1994). ◂

Remark 6.10. One interesting result, used to prove the following lemmas, is that the interpretation of types is monotone in δ, and that the interpretation of coinductive types is the largest set closed under destructors. This suggests that it might be possible to formulate the definition of the interpretation in categorical terms.

We just state here the key lemmas and leave their proofs out.

Lemma 6.11 (Soundness of type action)**.** *Suppose C is a type with $\Theta \mid \Gamma \vdash C : \Gamma' \twoheadrightarrow *$ and $\Theta = X_1 : \Gamma_1 \twoheadrightarrow *, \ldots, X_n : \Gamma_n \twoheadrightarrow *$, such that for all parameters $\Delta \vdash r : C'$ occurring in C and $\tau : \Delta' \rhd \Delta$, we have $r[\tau] \in [\![C']\!](\tau)$. Let $\delta_A, \delta_B \in [\![\Theta]\!]$ and $\Gamma_i, x : A_i \vdash t_i : B_i$, such that for all $\sigma \in [\![\Gamma_i]\!]$, $t_i \in \delta_B(X_i)(\tau)$. Then for all contexts Δ, all $\sigma : \Delta \rhd \Gamma, \Gamma'$ and all $s \in [\![C]\!](\delta_A, \sigma)$*

$$\widehat{C}(\vec{t})[\sigma, s] \in [\![C]\!](\delta_B, \sigma). \qquad (10)$$

Lemma 6.12. *The interpretation of types $[\![-]\!]$ given in Def. 6.8 is well-defined and $[\![A]\!](\delta, \rho) \in \mathbf{SAT}$ for all A, δ, ρ.*

Lemma 6.13 (Soundness)**.** *If $\Gamma \vdash t : A$, then for all $\rho \in [\![\Gamma]\!]$ we have $t[\rho] \in [\![A]\!](\rho)$.*

From the soundness, we can easily derive strong normalisation.

Theorem 6.14. *All well-typed terms are strongly normalising, that is, if $\Gamma_1 \vdash t : \Gamma_2 \twoheadrightarrow A$ then $t \in \mathbf{SN}$.*

Proof. We first note that terms only reduce if $\Gamma_2 = \varnothing$. In that case we can apply we can apply Lem. 6.13 with ρ being the identity, so that $t \in [\![A]\!](\rho)$. Thus, by Lem. 6.12 and the definition of saturated sets, we can conclude that $t \in \mathbf{SN}$. Since t does not reduce if Γ_2 is non-trivial, we also have in that case that $t \in \mathbf{SN}$. Hence every well-typed term is strongly normalising. ☐

7. Conclusion

We have introduced a type theory that is solely based on inductive and coinductive types, in contrast to other type theories that usually have separate type constructors for, for example, the function space. This results in a a small set of rules for the judgements of the theory and the corresponding reduction relation. To justify the use of our type theory as logic, we also proved that the reduction relation preserves types and is strongly normalising on well-typed terms. Combining the present theory with that in (Norell 2007) would give us a justification for a large part Agda's current type system, especially including coinductive types.

There are still some open questions, regarding the present type theory, that we wish to settle in the future. First of all, a basic property of the reduction relation that is still missing is confluence. Second, we have constructed the type theory with certain categorical structures in mind, and it is easy to interpret the types and terms in a data type closed category, see (Basold 2015). However, one has to be careful in showing the soundness of such an interpretation, that is, if $A \longleftrightarrow_T B$ then we better have $[\![A]\!] = [\![B]\!]$ because of the conversion rule. We can indeed define a generalised Beck-Chevalley condition, c.f. (Jacobs 1999), but it remains to be checked if this condition is strong enough to ensure soundness.

In Remark 3.10, we mentioned already that we have no dependent recursion, i.e., no induction. This could be added to the theory, using a lifting of the types A_k in $\mu(X : \Gamma \twoheadrightarrow * ; \vec{\sigma}; \vec{A})$ to predicates, and we think that the proof of strong normalisation can be adopted accordingly. We did not develop this in the present paper in order to keep matters simple for the time being.

Moreover, we would like to investigate at the same time the principle dual to induction, namely proving an equality by means of a bisimulation. It is, however, not clear how this can be properly integrated. There are several options, all of which are not completely satisfactory: Equip all types with bisimilarity $\sim$ as specific equality (setoid approach), which makes the theory hard to use; add a generalised replacement rule that, given a proof $p : t \sim s$, allows us to infer $r : P @ s$ from $r : P @ t$, but this makes type checking undecidable; add a cast operator like in observational type theory, but this introduces non-canonical terms (Altenkirch et al. 2007); treat coinductive types similar to higher inductive types (The Univalent Foundations Program 2013), but it is not clear how such a system can be set up, though it seems to be the most promising approach.

Finally, certain acceptable facts are not provable in our theory, since we do not have universes. Another common feature that is

$$[\![\vdash \top : *]\!](\delta, \rho) = \bigcap\{X \in \mathbf{SAT} \mid \diamondsuit \in X\}$$
$$[\![\Theta, X : \Gamma \twoheadrightarrow * \mid \varnothing \vdash X : \Gamma \twoheadrightarrow *]\!](\delta, \rho) = \delta(X)(\rho)$$
$$[\![\Theta, X \mid \Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow *]\!](\delta, \rho) = [\![\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow *]\!](\delta|_{\lceil\Theta\rceil}, \rho)$$
$$[\![\Theta \mid \Gamma_1, x : B \vdash A : \Gamma_2 \twoheadrightarrow *]\!](\delta, \rho) = [\![\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \twoheadrightarrow *]\!](\delta, \rho|_{\lceil\Gamma_1\rceil})$$
$$[\![\Theta \mid \Gamma_1 \vdash A \,@\, t : \Gamma_2[t/x] \twoheadrightarrow *]\!](\delta, \rho) = [\![\Theta \mid \Gamma_1 \vdash A : (x : B, \Gamma_2) \twoheadrightarrow *]\!](\delta, \rho[x \mapsto t[\rho]])$$
$$[\![\Theta \mid \Gamma_1 \vdash (x).\, A : (x : B, \Gamma_2) \twoheadrightarrow *]\!](\delta, \rho) = [\![\Theta \mid \Gamma_1, x : B \vdash A : \Gamma_2 \twoheadrightarrow *]\!](\delta, \rho)$$
$$[\![\Theta \mid \varnothing \vdash \mu(X : \Gamma \twoheadrightarrow * \,;\, \vec{\sigma}; \vec{A}) : \Gamma \twoheadrightarrow *]\!](\delta, \rho) = \{M \mid \forall U \in I_\Gamma.\ \forall \Delta.\ \forall k.\ \forall N_k \in \{[\![\Gamma_k]\!], [\![A_k^{\triangle}]\!](\delta[X \mapsto U])\}_y \Vdash [\![\sigma_k \bullet \pi]\!]^*(U).$$
$$\mathrm{rec}\,\overrightarrow{(\Gamma_k, y).\, N_k} \,@\, \rho \,@\, M \in U(\rho)\}$$
$$[\![\Theta \mid \varnothing \vdash \nu(X : \Gamma \twoheadrightarrow * \,;\, \vec{\sigma}; \vec{A}) : \Gamma \twoheadrightarrow *]\!](\delta, \rho) = \{M \mid\ \exists U \in I_\Gamma.\ \forall k.\ \forall \gamma \in [\![\sigma_k]\!]^{-1}(\rho).\ \xi_k \,@\, \gamma \,@\, M \in [\![A_k]\!](\delta[X \mapsto U], \gamma)\}$$

Figure 4. Interpretation of types as families of saturated sets

missing from the type theory, is predicative polymorphism, which is in fact justified and desirable from the categorical point of view.

References

A. Abel and B. Pientka. Wellfounded recursion with copatterns: a unified approach to termination and productivity. In *ICFP*, pages 185–196, 2013. doi: 10.1145/2500365.2500591.

A. Abel, B. Pientka, D. Thibodeau, and A. Setzer. Copatterns: Programming Infinite Structures by Observations. In *Proc. of POPL*, pages 27–38. ACM, 2013. doi: 10.1145/2429069.2429075.

P. Aczel. *Non-well-founded Sets*. Number 14 in Lecture Notes. Center for the Study of Language and Information, Stanford University, 1988.

Agda. Agda Documentation. Technical report, Programming Logic group, Chalmers and Gothenburg University, 2015. URL `http://wiki.portal.chalmers.se/agda/`. Version 2.4.2.5.

B. Ahrens, P. Capriotti, and R. Spadotti. Non-wellfounded trees in Homotopy Type Theory. *ArXiv150402949 Cs Math*, Apr. 2015.

T. Altenkirch, C. McBride, and W. Swierstra. Observational Equality, Now! In *Proc. of PLPV '07*, pages 57–68. ACM, 2007. doi: 10.1145/1292597.1292608.

H. Basold. Dependent Inductive and Coinductive Types are Fibrational Dialgebras. In R. Matthes and M. Mio, editors, *Proceedings of FICS '15*, volume 191 of *EPTCS*, pages 3–17. Open Publishing Association, Sept. 2015. doi: 10.4204/EPTCS.191.3.

H. Basold. Code Repository, 2016. URL `http://cs.ru.nl/~hbasold/code/`.

H. Basold and H. H. Hansen. Well-definedness and Observational Equivalence for Inductive-Coinductive Programs. *To Appear in JLC*, 2015.

Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

A. Bizjak, R. Clouston, H. B. Grathwohl, R. E. Møgelberg, and L. Birkedal. Guarded Dependent Type Theory with Coinductive Types. In *Proceedings of FOSSACS'16*, 2016.

cLab. Type Theoretic Interpretation of the Final Chain, 2016. URL `https://coalg.org/clab/Type_Theoretic_Interpretation_of_the_Final_Chain`.

R. L. Constable. The Structure of Nuprl's Type Theory. In *Logic of Computation*, number 157 in NATO ASI Series, pages 123–155. Springer Berlin Heidelberg, 1997.

Coq Development Team. The Coq proof assistant reference manual. Technical report, LogiCal Project, 2012. URL `http://coq.inria.fr`. Version 8.4.

H. Geuvers. A short and flexible proof of strong normalization for the calculus of constructions. In P. Dybjer, B. Nordström, and J. Smith, editors, *Types for Proofs and Programs*, number 996 in LNCS, pages 14–38. Springer Berlin Heidelberg, June 1994. doi: 10.1007/3-540-60579-7_2.

E. Giménez. Codifying Guarded Definitions with Recursive Schemes. In *Selected Papers from the TYPES '94 Workshop*, pages 39–59, London, UK, 1995. Springer-Verlag.

T. Hagino. A typed lambda calculus with categorical type constructors. In *Category Theory in Computer Science*, pages 140–157, 1987.

M. Hamana and M. Fiore. A Foundation for GADTs and Inductive Families: Dependent Polynomial Functor Approach. In *Proceedings of the Seventh WGP*, WGP '11, pages 59–70, New York, NY, USA, 2011. ACM. doi: 10.1145/2036918.2036927.

B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge University Press, Mar. 1988.

P. Martin-Löf. About Models for Intuitionistic Type Theories and the Notion of Definitional Equality. In *3rd Scandinavian Logic Symposium*, pages 81–109. North Holland and American Elsevier, 1975.

U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, Göteborg, Sweden, Sept. 2007.

C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In M. Bezem and J. F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications, TLCA, Proceedings*, volume 664 of *LNCS*, pages 328–345. Springer, 1993.

J. Sacchini. Type-Based Productivity of Stream Definitions in the Calculus of Constructions. In *2013 28th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS)*, pages 233–242, June 2013. doi: 10.1109/LICS.2013.29.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

A. S. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction*. North-Holland, 1988.

B. van den Berg. *Predicative topos theory and models for constructive set theory*. Phd, University of Utrecht, 2006.

B. van den Berg and F. De Marchi. Non-well-founded trees in categories. *Annals of Pure and Applied Logic*, 146(1):40–59, Apr. 2007. ISSN 0168-0072. doi: 10.1016/j.apal.2006.12.001.

B. Werner. *Une théorie des Constructions Inductives*. Phd, Université Paris VII, 1994.

A. Examples

Example A.1. In Ex. 4.2, we have defined the bottom type (initial object) to be $\bot = \mathbf{0} = \mu(X : * \,; \varepsilon_1; X)$. Note that this gave us the elimination principle

$$\frac{\vdash C : *}{\vdash E_C^{\bot} : (y : \bot) \twoheadrightarrow C}$$

in which C is not allowed to have any dependencies. So, to make $\bot$ usable with dependent types, we need to switch to fibred initial objects:

$$\bot_\Gamma := \mu(X : \Gamma \twoheadrightarrow * \,; \mathrm{id}_\Gamma; X \mathbin{@} \mathrm{id}_\Gamma) \mathbin{@} \mathrm{id}_\Gamma,$$

which corresponds to the following generalised algebraic data type.

data $\bot_\Gamma : (x_1 : A_1) \cdots (x_n : A_n) \rightarrow$ **Set where**
$\quad \alpha : (x_1 : A_1) \cdots (x_n : A_n) \rightarrow \bot_\Gamma\, x_1 \cdots x_n \rightarrow \bot_\Gamma\, x_1 \cdots x_n$

Similar to before, we define $E_C^{\bot} := \mathrm{rec}\ ((\Gamma, y).\, y) \mathbin{@} \mathrm{id}_\Gamma$, to obtain the elimination principle for falsum in context Γ:

$$\frac{\Gamma \vdash C : *}{\Gamma \vdash E_C^{\bot} : (x : \bot_\Gamma) \twoheadrightarrow C}$$

The corresponding derivation is given as follows.

$$\cfrac{\cfrac{\cfrac{\cfrac{\Gamma \vdash C : *}{\vdash (\Gamma).\, C : \Gamma \twoheadrightarrow *} \qquad \Gamma, y : C \vdash y : C}{\vdash \mathrm{rec}\ ((\Gamma, y).\, y) : (\Gamma, y : \bot_\Gamma) \twoheadrightarrow ((\Gamma).\, C) \mathbin{@} \mathrm{id}_\Gamma}\ \textbf{(Ind-E)}}{\Gamma \vdash \mathrm{rec}\ ((\Gamma, y).\, y) \mathbin{@} \mathrm{id}_\Gamma : (x : \bot_\Gamma) \twoheadrightarrow ((\Gamma).\, C) \mathbin{@} \mathrm{id}_\Gamma}\ \textbf{(Inst)}}{\Gamma \vdash \mathrm{rec}\ ((\Gamma, y).\, y) \mathbin{@} \mathrm{id}_\Gamma : (x : \bot_\Gamma) \twoheadrightarrow C}\ \textbf{(Conv)}$$

Note that the abstraction and instantiation steps consist of several steps, one for each variable in Γ. ◂

Example A.2 (Propositional Equality)**.** In Ex. 4.7, we remarked that the propositional equality arises as left adjoint to the contraction δ^*, hence can be represented in the current type system. Let us elaborate this a bit more. First, note that the common way of implementing propositional equality in Agda is by means of the following data type.

data $\mathrm{Eq}_A : A \rightarrow A \rightarrow$ **Set where**
$\quad \mathrm{refl} : (x : A) \rightarrow \mathrm{Eq}_A\ x\ x$

So it is clear that this type can be represented in our type system by

$$\mathrm{Eq}_A(x, y) := \mu(X : (x : A, y : A) \twoheadrightarrow * \,; \delta; \top) \mathbin{@} x \mathbin{@} y,$$

where $\delta : (x : A) \rhd (x : A, y : A)$ is the diagonal $\delta = (x, x)$. This type has the usual constructor $\mathrm{refl} := \alpha_1 : (x : A) \twoheadrightarrow \mathrm{Eq}_A(x, x)$ and (weak) elimination rule

$$\frac{x : A, y : A \vdash C : * \qquad x : A \vdash p : C[x/x, x/y]}{x : A, y : A, z : \mathrm{Eq}_A(x, y) \vdash E_{x,y}^{\mathrm{Eq}}(p, z) : C}$$

This is of course not the full J-rule (or path induction) but it is already strong enough to prove, for example, the replacement (or substitution or transport) rule:

$$\frac{x : A \vdash P : * \qquad x : A \vdash p : P \qquad x, y : A \vdash t : \mathrm{Eq}_A(x, y)}{y : A \vdash \mathrm{repl}(p, t) : P[y/x]}$$

by using $C = P[y/x]$, weakening this type to $x, y : A \vdash C : *$, and putting

$$\mathrm{repl}(p, t) := E_{x,y}^{\mathrm{Eq}}(p, t). \qquad ◂$$

Example A.3 (Primitive corecursion and successor)**.** In Ex. 4.9, we claimed that the successor map $s_\infty : (y : \mathbf{N}^\infty) \twoheadrightarrow \mathbf{N}^\infty$ on the extended naturals can be defined by means of primitive corecursion. Let us introduce this principle and the definition of s_∞.

First, by primitive corecursion (for $\mathbf{N}^\infty$) we mean that for any C and d, we can find an h as in the following diagram

$$\begin{tikzcd} C \arrow[r, "h"] \arrow[d, "d"'] & \mathbf{N}^\infty \arrow[d, "\xi"] \\ (\top + C) + \mathbf{N}^\infty \arrow[r, "{[\mathrm{id}_\top + h, \xi]}"] & \top + \mathbf{N}^\infty \end{tikzcd}$$

We can derive this principle as follows. Note that if we define

$$a = [[\kappa_1 \mathbin{@} x, \kappa_2 \mathbin{@} \kappa_1 \mathbin{@} x], \kappa_2 \mathbin{@} \kappa_2 \mathbin{@} x]$$

we have $a : (\top + C) + \mathbf{N}^\infty \twoheadrightarrow \top + (C + \mathbf{N}^\infty)$. Now suppose we are given $\Delta, y : C \vdash d : (\top + C) + \mathbf{N}^\infty$, if we define $d' = a \mathbin{@} ([d, \kappa_2 \mathbin{@} y] \mathbin{@} y)$, then we find

$$\cfrac{\cfrac{\Delta, y : \mathbf{N}^\infty \vdash \kappa_2 \mathbin{@} y : (\top + C) + \mathbf{N}^\infty}{\Delta, y : C + \mathbf{N}^\infty \vdash [d, \kappa_2 \mathbin{@} y] \mathbin{@} y : (\top + C) + \mathbf{N}^\infty}}{\Delta, y : C + \mathbf{N}^\infty \vdash d' : \top + (C + \mathbf{N}^\infty)}$$

This gives us, by $\mathbf{N}^\infty$ being a coinductive type,

$$\Delta \vdash \mathrm{corec}\ \big((y).\, d'\big) : y : C + \mathbf{N}^\infty \twoheadrightarrow \mathbf{N}^\infty$$

and thus we can define $h = \mathrm{corec}\ \big((y).\, d'\big) \mathbin{@} (\kappa_1 \mathbin{@} y)$ to get

$$\Delta, y : C \vdash h : \mathbf{N}^\infty.$$

Finally, we can use primitive corecursion to define s_∞ by taking the extension of

$$[\kappa_1 \circ \kappa_2 \circ \kappa_2, \kappa_2] : \mathbf{N}^\infty + \mathbf{N}^\infty \rightarrow (\top + (\mathbf{N}^\infty + \mathbf{N}^\infty)) + \mathbf{N}^\infty,$$

giving us $h : \mathbf{N}^\infty + \mathbf{N}^\infty \rightarrow \mathbf{N}^\infty$. Thus we put $s_\infty = h \circ \kappa_1$. ◂

B. Meta Properties

B.1 Basic Meta Properties

Proposition B.1. *The following rules holds for the calculus given in Sec. 3.*

- *Substitution*

$$\frac{\Theta \mid \Gamma_1, x : A, \Gamma_2 \vdash B : \Gamma_3 \twoheadrightarrow * \qquad \Gamma_1 \vdash t : A}{\Theta \mid \Gamma_1, \Gamma_2[t/x] \vdash B[t/x] : \Gamma_3[t/x] \twoheadrightarrow *}$$

$$\frac{\Gamma_1, x : A, \Gamma_2 \vdash s : T \qquad \Gamma_1 \vdash t : A}{\Gamma_1, \Gamma_2[t/x] \vdash s[t/x] : T[t/x]}$$

- *Exchange*

$$\frac{\Theta \mid \Gamma_1, x : A, y : B, \Gamma_2 \vdash C : \Gamma_3 \twoheadrightarrow * \qquad x \notin \mathrm{fv}(B)}{\Theta \mid \Gamma_1, y : B, x : A, \Gamma_2 \vdash C : \Gamma_3 \twoheadrightarrow *}$$

$$\frac{\Gamma_1, x : A, y : B, \Gamma_2 \vdash t : \Gamma_3 \twoheadrightarrow C \qquad x \notin \mathrm{fv}(B)}{\Gamma_1, y : B, x : A, \Gamma_2 \vdash t : \Gamma_3 \twoheadrightarrow C}$$

- *Contraction*

$$\frac{\Theta \mid \Gamma_1, x : A, y : A, \Gamma_2 \vdash C : \Gamma_3 \twoheadrightarrow *}{\Theta \mid \Gamma_1, x : A, \Gamma_2[x/y] \vdash C[x/y] : \Gamma_3[x/y] \twoheadrightarrow *}$$

$$\frac{\Gamma_1, x : A, y : A, \Gamma_2 \vdash t : \Gamma_3 \twoheadrightarrow C}{\Gamma_1, x : A, \Gamma_2[x/y] \vdash t[x/y] : \Gamma_3[x/y] \twoheadrightarrow C[x/y]}$$

Proof. In each case, the rules are straightforwardly proved by simultaneous induction over types and terms. It should be noted that for types only the instantiation and weakening rules appear as cases, since the other rules have only types without free variables in the conclusion. Similarly, only terms constructed by means of the the projection, weakening or the instantiation rule appear as cases in the proofs. ☐

Analogously, the substitution, exchange and contraction rules for type variables are valid in the calculus, as well.

B.2 Subject Reduction

Proof of Lemma 6.1. Recall that we have to prove

$$\frac{X : \Gamma_1 \rightarrowtriangle * \mid \Gamma_2' \vdash C : \Gamma_2 \rightarrowtriangle * \qquad \Gamma_1, x : A \vdash t : B}{\Gamma_2', \Gamma_2, x : \widehat{C}(A) \vdash \widehat{C}(t) : \widehat{C}(B)}$$

We want to prove this by induction in the derivation of C, thus we need to generalise the statement to arbitrary type constructor contexts Θ. So let $\Theta = X_1 : \Gamma_1 \rightarrowtriangle *, \ldots, X_n : \Gamma_n \rightarrowtriangle *$ be a context, $\Theta \mid \Gamma' \vdash C : \Gamma \rightarrowtriangle *$ a type and $\Gamma_i, x : A_i \vdash t_i : B_i$ terms for $i = 1, \ldots, n$. We show that $\Gamma', \Gamma, x : \widehat{C}(\vec{A}) \vdash \widehat{C}(\vec{t}\,) : \widehat{C}(\vec{B})$ holds by induction in the derivation of C.

The induction base has two cases. First, it is clear that if $\Theta = \emptyset$, then $\vec{A} = \varepsilon$ and $\widehat{C}(\vec{A}) = C$, thus the definition is thus well-typed. Second, if $C = X_i$ for some i, then we immediately have

$$\widehat{C}(\vec{A}) = C[\overrightarrow{(\Gamma_i).\, A/X}]\ \mathrm{id}_{\Gamma_i} = ((\Gamma_i).\, A_i)\ \mathrm{id}_{\Gamma_i} \longrightarrow_p A_i,$$

thus, by **(Conv)** and the type of t_i, we have $\Gamma_i, x : \widehat{C}(\vec{A}) \vdash t_i : \widehat{C}(\vec{B})$ as required.

In the induction step, we have five cases for C.

- The type correctness for $\widehat{C}$ in case C has been constructed by Weakening for type and term variables is immediate by induction and the definition of F in these cases.
- $C = C' \mathbin{@} s$ and $\Gamma = \Delta[s/y]$ with
$$\frac{\Theta \mid \Gamma' \vdash C' : (y : D, \Delta) \rightarrowtriangle * \qquad \Gamma' \vdash s : D}{\Theta \mid \Gamma' \vdash C' \mathbin{@} s : \Delta[s/y] \rightarrowtriangle *}$$
By induction we have then that $\Gamma', y : D, \Delta, x : \widehat{C'}(\vec{A}) \vdash \widehat{C'}(\vec{t}\,) : \widehat{C'}(\vec{A})$, thus, since
$$\begin{aligned}\widehat{C'}(\vec{A}) &= C'[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} \mathrm{id}_{y:D,\Delta} \\ &= C'[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} y \mathbin{@} \mathrm{id}_{\Delta},\end{aligned}$$
we get by Prop. B.1
$$\begin{aligned}&\Gamma', \Delta[s/y], x : C'[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} s \mathbin{@} \mathrm{id}_{\Delta[s/y]} \\ &\qquad \vdash \widehat{C'}(\vec{t}\,)[s/y] : C'[\overrightarrow{(\Gamma_1).\, B/X}] \mathbin{@} s \mathbin{@} \mathrm{id}_{\Delta[s/y]}.\end{aligned}$$
As we now have
$$\begin{aligned}F_{C' @ s}(\vec{A}) &= (C' \mathbin{@} s)[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} \mathrm{id}_{\Delta[s/y]} \\ &= C'[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} s \mathbin{@} \mathrm{id}_{\Delta[s/y]}\end{aligned}$$
and $F_{C' @ s}(\vec{t}\,) = \widehat{C'}(\vec{t}\,)[s/y]$, we find that
$$\Gamma', \Delta[s/y], x : F_{C' @ s}(\vec{A}) \vdash F_{C' @ s}(\vec{t}\,) : F_{C' @ s}(\vec{B})$$
as expected.
- $C = (y).\, C'$ with $\Theta \mid \Gamma', y : D \vdash C' : \Gamma \rightarrowtriangle *$. This gives us, by induction, $\Gamma', y : D, \Gamma, x : \widehat{C'}^{\Theta}(\vec{A}) \vdash \widehat{C'}^{\Theta}(\vec{t}\,) : \widehat{C'}^{\Theta}(\vec{B})$. Now we observe that
$$\begin{aligned}\widehat{C'}^{\Theta}(\vec{A}) &= C'[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} \mathrm{id}_{\Gamma} \\ &\longleftarrow_p ((y).\, C'[\overrightarrow{(\Gamma_1).\, A/X}]) \mathbin{@} y \mathbin{@} \mathrm{id}_{\Gamma} \\ &= C[\overrightarrow{(\Gamma_1).\, A/X}] \mathbin{@} \mathrm{id}_{y:D,\Gamma} \\ &= F_C^{\Theta}(\vec{A}),\end{aligned}$$
which gives us, by **(Conv)**, that $\Gamma', y : D, \Gamma, x : F_C^{\Theta}(\vec{A}) \vdash \widehat{C'}^{\Theta}(\vec{t}\,) : F_C^{\Theta}(\vec{B})$. Thus the definition $F^{\Theta}_{(x).\, C'}(\vec{t}\,) = \widehat{C'}^{\Theta}(\vec{t}\,)$ is well-typed.
- $C = \mu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D})$ with
$$\frac{\Theta, Y : \Gamma \rightarrowtriangle * \mid \Delta_k \vdash D_k : * \qquad \sigma_k : \Delta_k \rhd \Gamma}{\Theta \mid \emptyset \vdash \mu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D}) : \Gamma \rightarrowtriangle *}$$
For brevity, we define $R_{\vec{B}} = \mu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D}[\vec{B}/\vec{X}])$. Then, by induction, we have
$$\Gamma, x : F^{\Theta,Y}_{D_k}(\vec{A}, R_{\vec{B}}) \vdash F^{\Theta,Y}_{D_k}(\vec{t}, x) : F^{\Theta,Y}_{D_k}(\vec{B}, R_{\vec{B}})$$
Now we note that $F^{\Theta,Y}_{D_k}(\vec{A}, R_{\vec{B}}) = D_k[\overrightarrow{(\Gamma_i).\, A/X}][R_{\vec{B}}/Y]$.[3] If we define
$$g_k = \alpha_k \mathbin{@} \mathrm{id}_{\Delta_k} \mathbin{@} \left(F^{\Theta,Y}_{D_k}\left(\vec{t}, \mathrm{id}_{R_{\vec{B}}}\right)\right),$$
where α_k refers to $\alpha_k^{\mu(Y:\Gamma \rightarrowtriangle *\,;\vec{\sigma};\vec{D}[\overrightarrow{(\Gamma_i).\, B/X}])}$ (see the definition of F), then we can derive the following.
$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \alpha_k : \overline{\Delta_k}(F^{\Theta,Y}_{D_k}(\vec{A}, R_{\vec{B}}) \rightarrowtriangle R_{\vec{B}} \mathbin{@} \sigma_k)}{\Delta_k \vdash \alpha_k \mathbin{@} \mathrm{id}_{\Delta_k} : F^{\Theta,Y}_{D_k}(\vec{A}, R_{\vec{B}}) \rightarrowtriangle R_{\vec{B}} \mathbin{@} \sigma_k}\,\textbf{(Inst)}}{\Delta_k, x : D_k[\overrightarrow{(\Gamma_i).\, A/X}][R_{\vec{B}}/Y] \vdash g_k : R_{\vec{B}}}\,\textbf{(Inst)}}{\Gamma \vdash \mathrm{rec}\ x\ \vec{g} \mathbin{@} \mathrm{id}_{\Gamma} : R_{\vec{A}} \mathbin{@} \mathrm{id}_{\Gamma} \rightarrowtriangle R_{\vec{B}} \mathbin{@} \mathrm{id}_{\Gamma}}\,\textbf{(Ind-E)}}{\Gamma, x : R_{\vec{A}} \mathbin{@} \mathrm{id}_{\Gamma} \vdash \mathrm{rec}\ x\ \vec{g} \mathbin{@} \mathrm{id}_{\Gamma} \mathbin{@} x : R_{\vec{B}} \mathbin{@} \mathrm{id}_{\Gamma}}\,\textbf{(Inst)}$$
Finally, we have
$$\begin{aligned}\widehat{C}^{\Theta}(\vec{A}) &= \mu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D})[\overrightarrow{(\Gamma_i).\, A/X}] \mathbin{@} \mathrm{id}_{\Gamma} \\ &= R_{\vec{A}} \mathbin{@} \mathrm{id}_{\Gamma},\end{aligned}$$
which implies, by the above derivations, that we indeed have
$$\Gamma, x : \widehat{C}^{\Theta}(\vec{A}) \vdash \widehat{C}^{\Theta}(\vec{t}\,) : \widehat{C}^{\Theta}(\vec{B}).$$
- $C = \nu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D})$. This case is treated analogously to that for inductive types.

This concludes the induction, thus (5) indeed holds for all C, $\vec{A}$, $\vec{B}$ and $\vec{t}$. ☐

C. Strong Normalisation

Pre-Types and -Terms

Definition C.1 (Pre-Types)**.** See Fig. 5

Definition C.2 (Pre-Terms)**.** See Fig. 6.

Remark C.3. The intuition for Def. 5.2 can be better understood in terms of the diagrams that correspond to, for example, the definition on initial dialgebras. Put $R_{\vec{A}} = \mu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D}[\overrightarrow{(\Gamma_i).\, A/X}])$ and analogous for $R_{\vec{B}}$. Then $\mu(Y : \Gamma \rightarrowtriangle *\,; \vec{\sigma}; \vec{D})(\vec{t}\,)$ is defined as the morphism h in the following diagram.

$$\begin{tikzcd}
\widehat{D_k}\left(\vec{A}, R_{\vec{A}}\right) \arrow[r, "\widehat{D_k}\left(\vec{\mathrm{id}}, h\right)"] \arrow[dd, "\alpha_k"] & \widehat{D_k}\left(\vec{A}, R_{\vec{B}}\right) \arrow[d, "\widehat{D_k}(\vec{t}, \mathrm{id})"] \\
 & \widehat{D_k}\left(\vec{B}, R_{\vec{B}}\right) \arrow[d, "\alpha_k"] \\
R_{\vec{A}} \mathbin{@} \sigma_k \arrow[r, "h[\sigma_k]"] & R_{\vec{B}} \mathbin{@} \sigma_k
\end{tikzcd}$$

C.1 Soundness proof for saturated sets model

Lemma C.4. *For all $\sigma : \Gamma_1 \rhd \Gamma_2$ and $\tau : \Gamma_2 \rhd \Gamma_3$ we have $[\![\tau \bullet \sigma]\!] = [\![\tau]\!] \circ [\![\sigma]\!]$.*

[3] Note that the second substitution does not contain a parameter abstraction, as $R_{\vec{B}}$ is closed.

$$\frac{}{\vdash \top : *}\ \textbf{(PT)-}\top$$

$$\frac{}{\Theta, X : \Gamma \multimap * \mid \varnothing \vdash X : \Gamma \multimap *}\ \textbf{(PT-TyVar)}$$

$$\frac{\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \multimap *}{\Theta, X : \Gamma \multimap * \mid \Gamma_1 \vdash A : \Gamma_2 \multimap *}\ \textbf{(PT-TyWeak)}$$

$$\frac{\Theta \mid \Gamma_1 \vdash A : \Gamma_2 \multimap *}{\Theta \mid \Gamma_1, x \vdash A : \Gamma_2 \multimap *}\ \textbf{(PT-Weak)}$$

$$\frac{\Theta \mid \Gamma_1 \vdash A : (x, \Gamma_2) \multimap * \qquad \Gamma_1 \vdash t : \Box}{\Theta \mid \Gamma_1 \vdash A \mathbin{@} t : \Gamma_2 \multimap *}\ \textbf{(PT-Inst)}$$

$$\frac{\Theta \mid \Gamma_1, x : A \vdash B : \Gamma_2 \multimap *}{\Theta \mid \Gamma_1 \vdash (x).\, B : (x, \Gamma_2) \multimap *}\ \textbf{(PT-Param-Abstr)}$$

$$\frac{\Theta, X : \Gamma \multimap * \mid \Gamma_k \vdash A_k : * \qquad \sigma_k : \Gamma_k \rhd \Gamma \qquad k = 1, \dots, n \qquad \rho \in \{\mu, \nu\}}{\Theta \mid \varnothing \vdash \rho(X : \Gamma \multimap * ; \vec{\sigma}; \vec{A}) : \Gamma \multimap *}\ \textbf{(PT-FP)}$$

Figure 5. Pre-Types

$$\frac{}{\vdash \diamondsuit : \Box}\ \textbf{(PO-}\top\textbf{-I)} \qquad \frac{\Gamma_1 \vdash t : (x, \Gamma_2) \multimap \Box \qquad \Gamma_1 \vdash s : \Box}{\Gamma_1 \vdash t \mathbin{@} s : \Gamma_2 \multimap \Box}\ \textbf{(PO-Inst)}$$

$$\frac{x \in \mathrm{Var}}{\Gamma, x \vdash x : \Box}\ \textbf{(PO-Proj)} \qquad \frac{\Gamma_1 \vdash t : \Gamma_2 \multimap \Box}{\Gamma_1, x \vdash t : \Gamma_2 \multimap \Box}\ \textbf{(PO-Weak)}$$

$$\frac{k \in \mathbb{N}}{\vdash \alpha_k : \Gamma, x \multimap \Box}\ \textbf{(PO-Ind-I)} \qquad \frac{k \in \mathbb{N}}{\vdash \xi_k : \Gamma, x \multimap \Box}\ \textbf{(PO-Coind-E)}$$

$$\frac{\vdash \mu(X : \Gamma \multimap * ; \vec{\sigma}; \vec{A}) : \Gamma \multimap * \qquad \forall k = 1, \dots, n.\ (\Delta, \Gamma_k, y_k \vdash g_k : \Box)}{\Delta \vdash \mathrm{rec}^{\mu(X : \Gamma \multimap * ; \vec{\sigma}; \vec{A})}\ \overrightarrow{(\Gamma_k, y_k).\, N_k} : (\Gamma, y) \multimap \Box}\ \textbf{(PO-Ind-E)}$$

$$\frac{\vdash \nu(X : \Gamma \multimap * ; \vec{\sigma}; \vec{A}) : \Gamma \multimap * \qquad \forall k = 1, \dots, n.\ (\Delta, \Gamma_k, y_k \vdash g_k : \Box)}{\Delta \vdash \mathrm{corec}^{\nu(X : \Gamma \multimap * ; \vec{\sigma}; \vec{A})}\ \overrightarrow{(\Gamma_k, y_k).\, N_k} : (\Gamma, y) \multimap \Box}\ \textbf{(PO-Coind-I)}$$

Figure 6. Pre-Terms

Proof. For all $\rho : \lceil \Gamma_1 \rceil \to \Lambda$ and $x \in \lceil \Gamma_3 \rceil$ we have

$$\begin{aligned} [\![\tau \bullet \sigma]\!](\rho)(x) &= (\tau \bullet \sigma)(x)[\rho] = \tau(x)[\sigma][\rho] \\ &= [\![\tau]\!]([\![\sigma]\!](\rho))(x) = ([\![\tau]\!] \circ [\![\sigma]\!])(\rho)(x) \end{aligned}$$

as required. ∎

Lemma C.5. *If $\Gamma_1 \vdash A : \Gamma_2 \multimap *$, $\sigma : \Gamma_1 \rhd \Gamma_2$ and $\rho \in [\![\Gamma_1]\!]$, then $[\![A \mathbin{@} \sigma]\!](\rho) = [\![A]\!]([\rho, [\![\sigma]\!](\rho)])$, where $[\rho, [\![\sigma]\!](\rho)] \in [\![\Gamma_1, \Gamma_2]\!]$ is given by*

$$[\rho, [\![\sigma]\!](\rho)]\,(x) = \begin{cases} \rho(x), & x \in \lceil \Gamma_1 \rceil \\ [\![\sigma]\!](\rho)(x), & x \in \lceil \Gamma_2 \rceil \end{cases}.$$

Proof. Simply by repeatedly applying the case of the semantics of type instantiations. ∎

The following four lemmas C.6-C.9 are easily proved by induction in the derivation of the corresponding type A.

Lemma C.6. *If $\Gamma_1, x : B, \Gamma_2 \vdash A : *$ and $\Gamma_1 \vdash t : B$, then for all $\rho \in [\![\Gamma_1, \Gamma_2[t/x]]\!]$ we have $[\![A[t/x]]\!] = [\![A]\!](\rho[x \mapsto t])$.*

Lemma C.7. *If $\Theta \mid \Gamma_2 \vdash A : *$ and $\sigma : \Gamma_1 \rhd \Gamma_2$, then for all $\delta \in [\![\Theta]\!]$ and $\rho \in [\![\Gamma_1]\!]$ we have $[\![A[\sigma]]\!](\delta, \rho) = [\![A]\!](\delta, [\![\sigma]\!](\rho))$.*

Lemma C.8. *If $\Theta_1, X : \Gamma \multimap *, \Theta_2 \mid \Gamma_1 \vdash A : *$ and $\vdash B : \Gamma \multimap *$, then $[\![A[B/X]]\!](\delta, \rho) = [\![A]\!](\delta[X \mapsto [\![B]\!]], \rho)$.*

Lemma C.9. *If $\Theta \mid \Gamma \vdash A : *$ and $\delta, \delta' \in [\![\Theta]\!]$ with $\delta \sqsubseteq \delta'$ (point-wise order), then for all $\rho \in [\![\Theta]\!]$*

$$[\![A]\!](\delta, \rho) \subseteq [\![A]\!](\delta', \rho).$$

Lemma C.10. *Let $\mu = \mu(X : \Gamma \multimap * ; \vec{\sigma}; \vec{A})$ where we have $\Theta \mid \varnothing \vdash \mu : \Gamma \multimap *$. If $\delta \in [\![\Theta]\!]$, $\rho \in [\![\Gamma_k]\!]$ and $P \in [\![A_k]\!](\delta[X \mapsto [\![\mu]\!](\delta)], \rho)$, then*

$$\alpha_k \mathbin{@} \rho \mathbin{@} P \in [\![\mu]\!](\delta, [\![\sigma_k]\!](\rho)).$$

Proof. Let δ, ρ and P be given as in the lemma, and put $M = \alpha_k \mathbin{@} \rho \mathbin{@} P$. We need to show for any choice of $U \in I_\Gamma$ and

$$N_k \in \{[\![\Gamma_k]\!], [\![A_k^{\triangle}]\!](\delta[X \mapsto U])\}_y \Vdash [\![\sigma_k \bullet \pi]\!]^*(U)$$

that

$$K = \mathrm{rec}\ \overrightarrow{(\Gamma_k, y).\, N_k} @ (\sigma_k \bullet \rho) @ M$$

is in $U(\llbracket \sigma_k \rrbracket(\rho))$. Now we define $r = \mathrm{rec}\ \overrightarrow{(\Gamma_k, y).\, N_k}$ and

$$K' = N_k\Big[\widehat{A_k}(r @ \mathrm{id}_\Gamma @ x') \Big/ y\Big][\rho, P]$$

so that $K \succ K'$. Let us furthermore put

$$V = U(\llbracket \sigma_k \rrbracket(\rho)).$$

By $V \in \mathbf{SAT}$, it suffices to prove that $K' \in V$. Note that we can rearrange the substitution in K' to get $K' = N_k[\rho, P']$ with $P' = \widehat{A_k}(r @ \mathrm{id}_\Gamma @ x')[\rho, P]$.

We get $K' \in V$ from $N_k \in \{\llbracket \Gamma_k \rrbracket, \llbracket A_k \rrbracket(\delta[X \mapsto U])\}_y \Vdash \llbracket \sigma_k \bullet \pi \rrbracket^*(U)$, provided that $\rho \in \llbracket \Gamma_k \rrbracket$ and $P' \in \llbracket A_k \rrbracket(\delta[X \mapsto U], \rho)$. The former is given from the assumption of the lemma. The latter we get from Lem. 6.11, since we have assumed soundness for the components of μ and $P \in \llbracket \widehat{A_k} \rrbracket(\rho)$. Thus we have $K' = N_k[\rho, P'] \in V$.

So by saturation we have $K \in V = U(\llbracket \sigma_k \rrbracket(\rho))$ for any choice of U and N_k, thus if follows that $M \in \llbracket \mu \rrbracket(\delta, \llbracket \sigma_k \rrbracket(\rho))$. □

Lemma C.11. *Let $\nu = \nu(X : \Gamma \twoheadrightarrow *; \vec{\sigma}; \vec{A})$ where we have $\Theta \mid \varnothing \vdash \nu : \Gamma \twoheadrightarrow *$. If $U \in I_\Gamma$ and $\delta \in \llbracket \Theta \rrbracket$, such that for all $M \in U(\rho)$, all k, all $\rho \in \llbracket \Gamma \rrbracket$ and all $\gamma \in \llbracket \sigma_k \rrbracket^{-1}(\rho)$, $\xi_k @ \gamma @ M \in \llbracket A_k \rrbracket(\delta[X \mapsto U], \gamma)$, then*

$$\forall \rho.\, U(\rho) \subseteq \llbracket \nu \rrbracket(\delta, \rho).$$

Proof. This follows immediately from the definition of $\llbracket \nu \rrbracket$, just instantiate the definition with the given U. Then all all $M \in U(\rho)$ are in $\llbracket \nu \rrbracket(\delta, \rho)$. □

Lemma C.12. *Let $\nu = \nu(X : \Gamma \twoheadrightarrow *; \vec{\sigma}; \vec{A})$ where we have $\Theta \mid \varnothing \vdash \nu : \Gamma \twoheadrightarrow *$. If $\delta \in \llbracket \Theta \rrbracket$, $U \in I_\Gamma$, $\rho \in \llbracket \Gamma_k \rrbracket$ and*

$$N_k \in \{\llbracket \Gamma_k \rrbracket, \llbracket \sigma_k \rrbracket^*(U)\}_y \Vdash \llbracket A_k \rrbracket(\delta[X \mapsto U])$$

for $k = 1, \ldots, n$, then

$$\mathrm{corec}\ \overrightarrow{(\Gamma_k, y).\, N_k} @ \rho @ M \in \llbracket \nu \rrbracket(\delta, \rho).$$

Proof. Similar to the proof of Lem. C.10 by using that the interpretation of ν-types is a largest fixed point Lem. C.11 and that the interpretation is monotone Lem. C.9. □

Lemma C.13. *Suppose C is a type with $\Theta \mid \Gamma_1 \vdash C : \Gamma_2 \twoheadrightarrow *$. If $\rho, \rho' \in \llbracket \Gamma_1, \Gamma_2 \rrbracket$ with $\rho \longrightarrow \rho'$, then $\forall \delta.\, \llbracket C \rrbracket(\delta, \rho) = \llbracket C \rrbracket(\delta, \rho')$. Furthermore, if $C \longrightarrow C'$, then $\llbracket C \rrbracket = \llbracket C' \rrbracket$.*

Proof. The first part follows by an easy induction, in which the only interesting case $C = X$ is. Here we have

$$\begin{aligned}\llbracket C \rrbracket(\delta, \rho) &= \delta(X)(\rho) \\ &= \delta(X)(\rho') \\ &= \llbracket C \rrbracket(\delta, \rho'),\end{aligned}$$

since $\delta(X) \in I_{\Gamma_i}$ and thus respects conversions.

For the second part, let D be given by replacing all terms in parameter position in C by variables, so that $C = D[\rho]$ for some substitution ρ. But then there is a ρ' with $\rho \longrightarrow \rho'$ and $C' = D[\rho']$, and the claim follows from the first part. □

Proof of Lemma 6.11. We proceed by induction in the derivation of $\Theta \mid \Gamma \vdash C : \Gamma' \twoheadrightarrow *$.

- $\vdash \top : *$ by **($\top$-I)**. In this case we have that $\widehat{\top}(\varepsilon) = x \in \mathcal{B}$, thus $\widehat{\top}(\varepsilon) \in \llbracket C \rrbracket(\delta)$ by by saturation.
- $\Theta, X_{n+1} : \Gamma_{n+1} \twoheadrightarrow * \mid \varnothing \vdash X : \Gamma_{n+1} \twoheadrightarrow *$ by **(TyVar-I)**. Note that $\widehat{X_{n+1}}(\vec{t}, t_{n+1}) = t_{n+1}$ and $\llbracket X_{n+1} \rrbracket(\delta, \sigma) = \delta(X_{n+1})(\sigma) = \llbracket B_{n+1} \rrbracket(\sigma)$. thus the claim follows directly from the assumption of the lemma.
- $\Theta, X : \Gamma'' \twoheadrightarrow * \mid \Gamma \vdash C : \Gamma' \twoheadrightarrow *$ by **(TyVar-Weak)**. Immediate by induction.
- $\Theta \mid \Gamma, y : D \vdash C : \Gamma' \twoheadrightarrow *$ by **(Ty-Weak)**. Again immediate by induction.
- $\Theta \mid \Gamma \vdash C @ r : \Gamma'[s/x] \twoheadrightarrow *$ with $\Gamma \vdash r : C'$ by **(Ty-Inst)**. First, we note that $\sigma = (\sigma_1, \sigma_2)$ with $\sigma_1 : \Delta \rhd \Gamma$ and $\sigma_2 : \Delta \rhd \Gamma'[\sigma_1, r]$. Let us put $\tau = (\sigma_1, r[\sigma_1], \sigma_2)$, so that we have

$$\begin{aligned}\widehat{(C @ r)}(\vec{t})[\sigma, s] &= \widehat{C}(\vec{t})[s/x][\sigma, s] \\ &= \widehat{C}(\vec{t})[\sigma_1, r[\sigma_1], \sigma_2, s] \\ &= \widehat{C}(\vec{t})[\tau, s].\end{aligned}$$

  By the assumption of the lemma on parameters we have $r[\sigma_1] \in \llbracket C' \rrbracket(\sigma_1)$, and thus $\tau \in \llbracket \Gamma, x : C'[\sigma_1], \Gamma'[\sigma_1, r[\sigma_1]] \rrbracket$, which gives $\llbracket C @ r \rrbracket(\delta, \sigma) = \llbracket C \rrbracket(\delta, \tau)$. By induction, we have $\widehat{C}(\vec{t})[\tau, s] \in \llbracket C \rrbracket(\delta, \tau)$, and

$$\widehat{(C @ r)}(\vec{t})[\sigma, s] \in \llbracket C @ r \rrbracket(\delta, \sigma)$$

  follows.
- $\Theta \mid \Gamma_1 \vdash (x).\, B : (x : A, \Gamma_2) \twoheadrightarrow *$ by **(Param-Abstr)**. Immediate by induction.
- $\Theta \mid \varnothing \vdash \mu(Y : \Gamma \twoheadrightarrow *; \vec{\tau}; \vec{D}) : \Gamma \twoheadrightarrow *$ by **(FP-Ty)**. We abbreviate, as before, this type just by μ. Recall the definition of $\widehat{\mu}$:

$$\begin{aligned}\widehat{\mu}(\vec{t}) &= \mathrm{rec}^{R_A}\ \overrightarrow{(\Delta_k, x).\, g_k} @ \mathrm{id}_\Gamma @ x \\ &\quad \text{with } g_k = \alpha_k @ \mathrm{id}_{\Delta_k} @ \left(\widehat{D_k}\ (\vec{t}, y)\right) \\ &\quad \text{and } R_A = \mu[\overrightarrow{(\Gamma_i).\, A_i}/\vec{X}] \\ &\quad \text{for } \Theta, Y : \Gamma \twoheadrightarrow * \mid \Delta_k \vdash D_k : *\end{aligned}$$

  Now, put $\delta' = \delta[Y \mapsto \llbracket R_B \rrbracket]$, then we have by induction that $\widehat{D_k}(\vec{t}, y)[\mathrm{id}_{\Delta_k}, y] \in \llbracket D_k \rrbracket(\delta', \mathrm{id}_{\Delta_k})$. Since $\mathrm{id}_{\Gamma_k} \in \mathbf{SN}$ we have by Lem. C.10 for all $\rho \in \llbracket \Delta_k, \widehat{D_k}(\vec{A}, R_B) \rrbracket$ that $g_k[\rho] \in \llbracket \mu \rrbracket(\delta, \llbracket \tau_k \rrbracket(\rho))$. By assumption, we have $s \in \llbracket R_A \rrbracket(\sigma)$, hence by choosing $U = \llbracket R_B \rrbracket$ in the definition of $\llbracket R_A \rrbracket$ we find $\widehat{\mu}(\vec{t})[\sigma, s] \in \llbracket R_B \rrbracket(\sigma) = \llbracket \mu \rrbracket(\delta, \sigma)$.
- $\Theta \mid \varnothing \vdash \nu(Y : \Gamma \twoheadrightarrow *; \vec{\tau}; \vec{D}) : \Gamma \twoheadrightarrow *$ by **(FP-Ty)**. Analagous to the inductive case, only that we use Lem. C.12.

□

Proof of Lemma 6.13. We proceed by induction in the type derivation for t. Since t does not have any parameters, we only have to deal with fully applied terms and will thus leave out the case for instantiation in the induction. Instead, we will have cases for fully instantiated α, ξ, etc. So let $\rho \in \llbracket \Gamma \rrbracket$ and proceed by the cases for t.

- $\diamondsuit \in \llbracket \top \rrbracket(\rho)$ by definition.
- For $\Gamma, x : A \vdash x : A$ we have $x[\rho] = \rho(x)$. By definition of $\llbracket \Gamma, x : A \rrbracket$, we have $\rho(x) \in \llbracket \Gamma \vdash A : * \rrbracket(\rho|_{\lceil \Gamma \rceil}) = \llbracket \Gamma, x : A \vdash A : * \rrbracket(\rho)$. Thus $x[\rho] \in \llbracket A \rrbracket(\rho)$ as required.
- Weakening is dealt with immediately by induction.
- If t is of type B by **(Conv)**, then by induction $t \in \llbracket A \rrbracket(\rho)$. Since by Lem. C.13 $\llbracket B \rrbracket(\rho) = \llbracket A \rrbracket(\rho)$, we have $t \in \llbracket B \rrbracket(\rho)$.
- Suppose we are given $\mu = \mu(X : \Gamma \twoheadrightarrow *; \vec{\sigma}; \vec{A})$ and $\Delta \vdash \alpha_k @ \tau @ t : \mu[\sigma_k \bullet \tau]$ with $\tau : \Delta \rhd \Gamma$ and $\Delta \vdash t : A_k[\mu/X][\tau]$.

Then, by induction, we have $t \in [\![A_k]\!](X \mapsto [\![\mu]\!], \tau)$ and soundness for the components of μ, thus by Lem. C.10

$$\alpha_k @ \tau @ t \in [\![\mu]\!]([\![\sigma_k \bullet \tau]\!](\rho)) = [\![\mu[\sigma_k \bullet \tau]]\!](\rho).$$

- Suppose we have $\mu = \mu(X : \Gamma \twoheadrightarrow *; \vec{\sigma}; \vec{A})$ and $\Delta \vdash \mathrm{rec}^{\mu}\ \overrightarrow{(\Gamma_k, x).\, g_k} @ \tau @ t : C @ \tau$. Then by induction we get from $\Delta \vdash t : \mu[\tau]$ that $t \in [\![\mu[\tau]]\!](\rho)$, hence if we chose $U = [\![C @ \tau]\!]$ and $N_k = g_k$ the definition of $[\![\mu]\!]$ yields $\mathrm{rec}^{\mu}\ \overrightarrow{(\Gamma_k, x).\, g_k} @ \tau @ t \in [\![C @ \tau]\!](\rho)$.
- Suppose $\nu = \nu(X : \Gamma \twoheadrightarrow *; \vec{\sigma}; \vec{A})$ and $\Delta \vdash \xi_k @ \tau @ t : A_k[\nu/X][\tau]$ with $\tau : \Delta \rhd \Gamma_k$ and $\Delta \vdash t : \nu[\sigma_k \bullet \tau]$. By induction, $t \in [\![\nu[\sigma_k \bullet \tau]]\!](\rho)$ thus there is a U such that $\xi_k @ \tau @ t \in [\![A_k]\!](X \mapsto U, [\![\tau]\!](\rho))$. By Lem. C.11 and Lem. C.9 we then have $\xi_k @ \tau @ t \in [\![A_k]\!](X \mapsto [\![\nu]\!], [\![\tau]\!](\rho))$. Since $[\![A_k]\!](X \mapsto [\![\nu]\!], [\![\tau]\!](\rho)) = [\![A_k[\nu/X][\tau]]\!](\rho)$, the claim follows.
- For corec-terms we just apply Lem. C.12, similar to the α_k-case.

This concludes the induction, thus the interpretation of types is sound with respect to the typing judgement for terms. $\square$