Axiomatizations of Compositional Inductive-Recursive Definitions

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Stephan Alexander Spahn
aus Karlsruhe

Tübingen
2018

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	25.10.2018
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Peter Schroeder-Heister
2. Berichterstatter:	Reader Dr. Anton Setzer

Contents

Zusammenfassung

Induktion-Rekursion ist ein Definitionsprinzip in Martin-Löf Typentheorie das Familien $(\mathsf{U}, \mathsf{T} : \mathsf{U} \to D) : \mathsf{Fam}(D)$, für eine beliebige (große) Menge $D : \mathsf{Set}_1$ (wie beispielsweise $D = \mathsf{Set}$, was den in motivierenden Beispielen auftretenden Fall darstellt), definiert, wobei simultan $\mathsf{U} : \mathsf{Set}$ durch Induktion-, und $\mathsf{T} : \mathsf{U} \to D$ durch Rekursion auf U definiert ist; die Qualifikation "simultan" meint hier dass U von Funktionswerten von T abhängen darf. Zwei äquivalente[1] Axiomatisierungen dieser Situation wurden von Dybjer-Setzer [38][40] vorgeschlagen. In beiden Fällen wird eine (große) Menge $\mathsf{DS}\ D\ D$ (respektive $\mathsf{DS}'\ D\ D$) von *Codes für induktiv-rekursive Definitionen* definiert sodass jeder Code $c : \mathsf{DS}\ D\ D$ Anlass gibt zu einem Endofunktor $[\![\ c\ ]\!] : \mathsf{Fam}(D) \to \mathsf{Fam}(D)$ zwischen Kategorien von Familien dessen initiale Algebra die Familie ist welche durch diesen Code c definiert ist.

Diese Axiomatisierungen ([38][40]) sind jedoch nicht die einzigen vernünftigen Aximatisierungen für Induktion-Rekursion. Es gibt mindestens zwei Wege zu diesem Schluss zu kommen: der eher praktisch orientierte Zugang ist motiviert durch die Beobachtung dass während es in der Referenz-Theorie der induktiven Definitionen immer möglich ist (in semantisch bedeutungsvoller Weise) die Komposition von zwei Codes zu einem einzigen-, neuen Code zu bilden, dies kaum der Fall zu sein scheint für die bisher existierenden Axiomatisierungen von induktiv-rekursiven Definitionen. Die zweite-, eher konzeptuelle Beobachtung über die bereits existierenden Axiomatisierungen ist dass diese keine Konstruktoren für abhängige Produkte (oder Potenzen[2]) von Codes, sondern nur für abhängige Summen von Codes enthalten. In der Tat zeigen wir dass beide Beobachtungen in Zusammenhang stehen, indem wir beweisen dass Kompositionalität für Dybjer-Setzer inductiv-rekursive Definitionen äquivalent ist zur Existenz von Potenzen dieser Codes durch Mengen.

In der Folge definieren-, und untersuchen wir zwei neue Axiomatisierungen für induktiv-rekursive Definitionen welche die erwähnten äquivalenten charakterisierenden Eigenschaften erfüllen und für die wir Kompositionalität beweisen. Die Erste ist erreicht indem wir ein Untersystem[3] UF, bestehend aus *uniformen Codes*, des Dybjer-Setzer induktiv-rekursive Definitionen definierenden Systems DS identifizieren für das Potenzen von Codes existieren.

Die zweite Axiomatisierung PN die *polynomielle Codes* (so genannt da diese auf der Idee

[1]Diese Axiomatisierungen sind äquivalent falls das "logical framework" entsprechend gewählt ist.
[2]Es besteht eine enge Beziehung zwischen abhängigen Produkten und Potenzen.
[3]Gemeint ist hier dass eine Semantik-erhaltende Übersetzung von UF nach DS existiert.

Polynome[4] zu iterieren basieren) definiert ein System in das DS eingebettet werden kann und das einen Konstruktor für abhängige Produkte (und insbesondere Potenzen) von Codes enthält. Während für DS die Existenz eines (mengentheoretischen) Modells durch eine Einbettung in ein bereits existierende Modell für DS erhalten werden kann, können wir nicht in dieser Weise ein Modell für PN erhalten, und konstruieren daher stattdessen ein neues Modell das aber beinahe mit denselben, große Kardinalzahlen betreffenden, mengentheoretischen Annahmen wie das Modell für DS auskommt: während Dybjer-Setzer induktiv-rekursive Definitionen ein Modell in ZFC erweitert durch eine Mahlo Kardinalzahl und eine 0-unerreichbare Kardinalzahl haben, benötigen wir hier ZFC erweitert durch eine Mahlo Kardinalzahl und eine 1-unerreichbare Kardinalzahl.

Da das System PN nicht einfach caeteris paribus durch Hinzunahme eines Konstruktors für abhängige Produkte von Codes entsteht, sondern eine Neudefinition aller Konstruktoren erfordert, stellt sich weiterhin die Frage nach Konstruktoren die das Bild der Einbettung $\mathsf{DS} \hookrightarrow \mathsf{PN}$ erzeugen; wir nähern uns dieser Frage indem wir ein Zwischensystem das zwischen DS und PN liegt, und eine Übersetzung dieses Zwischensystems nach DS definieren. Dieses Zwischensystem entsteht nicht allein durch Entfernen des Konstruktors für abhängige Produkte, denn dies ermöglichte uns noch nicht eine gewünschte Übersetzung zu erhalten, sondern durch zusätzliches Einführen einer Uniformitätsbedingung die mithilfe einer Annotation der Codes mit Binärbäumen realisiert ist.

Eine gemeinsame Eigenschaft beider neuer Systeme UF und PN ist, dass diese eine flexiblere Beziehung zwischen Codes und Untercodes besitzen als dies für DS der Fall ist: unter-DS-Codes eines gegebenen DS-codes haben alle denselben Typ während unter-UF-, und unter-PN-Codes dieser Einschränkung nicht unterliegen. Die letzgenannte Eigenschaft enthüllt einen abstrakteren Weg zu dem Schluss zu kommen dass Dybjer-Setzer Induktion-Rekursion nicht die allgemeinste Formulierung induktiv-rekursiver Definitionen ist: in ähnlicher Weise wie induktive Definitionen durch eine Menge von Operationen auf Mengen[5] definiert werden können, so sind induktiv-rekursive Definitionen durch eine Menge von Operationen auf Familien bestimmt (diese Operationen sind repräsentiert durch die Funktoren die durch die Codes definiert sind) und Dybjer-Setzer's Systeme DS und DS' zogen Operationen welche die indizierende Menge D ändern nicht in Betracht während die Syteme UF und PN das tun.

In Konsonanz mit der Idee Induktion-Rekursion als zum Vorhaben Typentheorie in Typentheorie zu formalisieren beitragend vorzustellen, kehren wir zu Dybjer-Setzer's ursprünglicher Formulierung zurück und geben ein *relational-parametrisches Modell* das als ein *Kategorien-mit-Familien* Modell auf der Kategorie der reflexiven Graphen formuliert ist; Kategorien-mit-Familien wurden in [34] als eine Formalisierung von Typentheorie in Typentheorie vorgeschlagen die auch einen kategorientheoretischen Zugang ermöglicht. Relationale-Parametrizität ist eine wohlbekannte und wichtige Beweistechnik um metatheoretische Eigenschaften von Typentheorien zu untersuchen.

[4]Polynome in dem hier zu verstehenden Sinne sind auch als "Container" bekannt und sind eine Formalisierung von induktiven Definitionen.

[5]Diese Operationen werden "strikt positive Operationen" genannt (siehe [36]).

Abstract

Induction-recursion is a definitional principle in Martin-Löf Type Theory defining families $(\mathsf{U}, \mathsf{T} : \mathsf{U} \to D) : \mathsf{Fam}D$ where $D : \mathsf{Set}_1$ is an arbitrary fixed (large) set (which in motivating examples is chosen to be $D = \mathsf{Set}$), and $\mathsf{U} : \mathsf{Set}$ is defined by induction while T is simultaneously defined by recursion on U; the qualifier "simultaneously" means here that U may depend on values of the function $\mathsf{T} : \mathsf{U} \to D$. Two equivalent[6] axiomatizations of this situation were proposed by Dybjer-Setzer in [38][40]. In both cases a (large) set of *codes* $\mathsf{DS}\ D\ D$ (respectively DS') for inductive-recursive definitions is defined such that each code $c : \mathsf{DS}\ D\ D$ decodes to an endofunctor $[\![\ c\ ]\!] : \mathsf{Fam}D \to \mathsf{Fam}D$ between categories of families whose initial algebra is the family defined by this code c. The authors proved the consistency of their axiomatizations be giving a set-theoretic model.

These axiomatizations DS and DS' are however not the only reasonable axiomatizations of induction-recursion. There are at least two ways to come to this conclusion: the more practical one is motivated by the observation that while in the reference theory of inductive definitions it is always possible to compose (in a semantically sound way) two inductive definitions to a single new one, this seems hardly to be the case for the preexisting axiomatizations of Induction-Recursion. The second-, more conceptual observation about the existing axiomatizations of induction-recursion is that it does not contain constructors for dependent products (or powers[7]) of codes but only for dependent sums of codes. Indeed, we show that these two observations are related by characterizing compositionality of Dybjer-Setzer induction-recursion in terms of the existence of powers of codes by sets.

Departing from this characterization, we define- and explore two new axiomatizations of induction-recursion satisfying the mentioned characterization and for which we prove compositionality. In the first one, this is achieved by restricting to a subsystem[8] UF of DS of *uniform codes* for which powers of codes exist. Consistency of this system is established by a semantics-preserving embedding into the system DS.

The second axiomatization PN of *polynomial codes* (so called since they are based on the idea of iterating polynomials[9]) defines a system into which DS can be embedded and which contains a constructor for dependent products- (and in particular powers) of codes. While for UF the existence of a model can be obtained by embedding it into DS for which Dybjer-Setzer themselves devised a (set-theoretic) model, we cannot argue in this

[6]These axiomatizations are equivalent if the underlying logical framework is chosen appropriately.
[7]There is a close relationship between dependent products-, and powers of codes.
[8]By "subsystem" we mean here that there is a semantics-preserving translation from UF to DS.
[9]Polynomials are also called *containers* and are a formalization for inductive definitions.

way for a model of PN and instead we provide a new model for the latter having almost the same set-theoretical assumptions concerning large cardinals: while Dybjer-Setzer induction-recursion can be modeled in ZFC supplemented by a Mahlo cardinal and a 0-inaccessible, we need ZFC plus a Mahlo cardinal and a 1-inaccessible.

Since the system PN does not simply arise by adding a constructor for dependent products of codes to DS caeteris paribus, but additionally requires redefining all other constructors, the question about constructors generating the image of the inclusion $\mathsf{DS} \hookrightarrow \mathsf{PN}$ imposes itself; we approach this question by defining an intermediary system lying between DS and PN and give a translation of this intermediary system into DS. This intermediary system does not only arise by removal of the constructor for dependent products of codes since this did not yet enable us to define a desired translation to DS, but by introduction of an additional uniformity constraint realized by an annotation of codes by binary trees.

A common feature of both new systems UF and PN is that they admit a more flexible relation between codes and their subcodes than this is the case for DS: sub-DS-codes of a given DS-code do all have the same type while sub-UF-, and sub-PN-codes are not constrained in this way. This latter feature reveals a more abstract way of arriving at the conclusion that Dybjer-Setzer induction-recursion is not the most general formulation of induction-recursion: like inductive definitions can be characterized by a set of operations on sets[10], inductive-recursive definitions are also determined by a set of operations on families (namely those represented by the set of functors defined by codes) and Dybjer-Setzer's systems did not take into account operations that change the (large) set D indexing families while the new systems UF and PN do so.

In line with the idea of considering induction-recursion as contributing to the pursue of the project of formalizing the meta theory of type theory in type theory itself [34, p.1], we return to Dybjer-Setzer's original formalization DS and provide a *relationally-parametric* model for it that we articulate as a categories-with-families model in the category of reflexive graphs; categories-with-families were proposed in loc.cit. as a formalization of type theory inside type theory that is additionally well adapted to category theoretic reasoning. Relational parametricity is an established and important proof technique to establish meta-theoretic properties of type theories.

[10]Theses operations are called "stictly positive operations" (see [36]).

Acknowledgement

I would like to thank Neil Ghani for having brought the fascinating topic of induction-recursion to my attention, for his supervision and other support during the time in which the present work has been carried out. I also would like to thank Fredrik Nordvall Forsberg for having taught me Agda, patience, and many other things. Conor McBride, Robert Atkey, and Peter Hancock I thank for interesting and helpful discussions. Furthermore I thank Peter Schroeder-Heister for having accepted my dissertation as supervisor, Reinhard Kahle for serving as a co-supervisor, and Anton Setzer for functioning as rapporteur.

Chapter 0

Introduction

Induction-recursion (IR)[1] is the appropriate notion of induction for families $(U, T) : \mathsf{Fam}(D)$, i.e. for pairs where $U : \mathsf{Set}$ is a (small) set, and $T : U \to D$ is a function from U to the arbitrary[2] large set $D : \mathsf{Set}_1$. The term "induction" traditionally has been restricted to refer to inductively defined sets as opposed to a more general meaning referring generally to objects in categories that are "inductively" defined in the sense of arising as initial algebras Section 2.2.4.4 which arguably makes sense in set-theoretic foundations since there everything is reducible to sets (and the element relation). In type-theoretic foundations like Martin-Löf type theory —with which we will be concerned in this thesis— functions like $T : U \to D$ are however not realized as a sets but enjoy more ontological autonomy. The operation to define functions "inductively" —if we understand this term for the moment as meaning "step-by-step, bottom-up, incrementally" or the like— is usually called "recursion" and is available in case the function's domain U is defined inductively. Here "U is defined..." before the invention of induction-recursion meant more precisely that the definition of U had to be completed before that of T could

[1]We use the acronym IR to stand for 'induction-recursion' in general. As it stands, induction-recursion is an informal notion. We do not present a formal system 'IR' in this thesis which would attribute some universality to this system whereas this thesis is just a step towards finding such an universal or canonical system.

[2]See [88, p.2] for the use of the term "arbitrary".

be begun. But with induction-recursion, both U, and T can be defined simultaneously, i.e. U may depend on values of T (in structurally smaller arguments of U).

The idea of simultaneous induction-recursion is implicit already in [88] where the first consistent version of Martin-Löf Type Theory was presented, for which normalization is shown by using inductive-recursively defined "computability predicates" (loc.cit. §4). More programatically, loc.cit. expresses the possibility to provide a formal definitional principle to define *all* definition of a certain kind:

> "The type $\mathbb{N}$ is just the prime example for a type introduced by an *ordinary inductive definition*. However, it seems preferable to treat this special case rather than to give the necessarily much more complicated formulation which would include [all other inductive definitions, and] $\mathbb{N}$ as special cases." [88, p.6]

Indeed, Martin-Löf had given such a "much more complicated formulation" already in his earlier paper [91] -however not in intuitionistic type theory but in the language of first order predicate logic[3]-, but the above quoted passage is usually read pars pro toto as a motivation for a definitional schema not only for "ordinary inductive definitions" but one replacing preferably the set of *all* rules defining particular types and type families (such as Tarski universes) by a single generic rules defining all particulars [32, p.2].

0.1 Martin-Löf Type Theory and Inductive Types

Martin-Löf Type Theory (MLTT) [88] —in which the theory of inductive-recursive definitions is situated— is a constructive foundational theory of mathematics formulated as a natural-deduction calculus. As such it implements the very concept of induction[4]: every type is specified by a *formation rule* supplying a name in form of a logical constant symbol for the type defined, moreover every nonempty type is introduced by rules positing certain terms called *constructors* for this type and these *introduction rules* are complemented by one *elimination rule* that states that a function having the defined type as domain is fully determined by specifying its values on constructors only, as well as one *computation rule* for every introduction rule relating the introduction- and elimination rules. Together these rules enforce that the defined type is constructed only using its constructors, and moreover that it is the least type[5] defined in this way.

The universal example of an inductive definition of a set[6] —in the sense that every inductive definition can be expressed in this form— is the type of trees (also called

[3]The above quotation from [88, p.6] continues: "See Martin-Löf 1971 [that is [91]] for a general formulation of inductive definitions in the language of first order predicate logic."

[4]We are rather brief on induction in general in this introduction and focus here more on IR specific matters. More on induction is in Section 2.2 and Section 2.1 containing a short historic account of induction.

[5]The notion of "the least type" can be made precise by the initial algebra semantics of type theory explained further below.

[6]Unless otherwise stated, we mean by 'set' a term of type Set in MLTT. We will briefly comment on Set in Section 0.5.

W-type Section 2.2.4.2[7]) of a given branching signature, where a branching signature $(A : \mathsf{Set}, B : A \to \mathsf{Set})$ is simply a family of sets where A can be regarded as the set of available nodes of for the trees, and for every node a, the set $B(a)$ is interpreted as the set of available branches departing from node a. The the set $\mathsf{W}(A, B)$ of all trees that can be formed with these data can be defined by the constructor

$$\sup : (a : A)(B(a) \to \mathsf{W}(A, B)) \to \mathsf{W}(A, B) \; .$$

Equivalently it is the least fixpoint of the function $X \mapsto (a : A)(B(a) \to X)$. We give a self-contained discussion of MLTT in Section 2.2 including all rules for the just sketched W-types.

0.2 Basic Examples of Induction-Recursion

Before commenting on formal notions of induction-recursion, the definitional principles defining families[8] $(\mathsf{U}, \mathsf{T}) \in \mathsf{Fam}(D) :\equiv \Sigma_{U:\mathsf{Set}} U \to D$ which we will mainly be interested in this thesis, we informally list a few basic examples.

Every inductive definition of a set U is a (degenerate) example of an inductive-recursive definition: U can be regarded as the family $(\mathsf{U}, \mathsf{T} : \mathsf{U} \to 1)$ indexed over the one element set which is the terminal object in the category Set_1, and hence there is by definition an isomorphism $\mathsf{Fam}\ 1 \simeq \mathsf{Set}$. That the formalism defining inductive-recursive definitions (see Section 3.2.1) reduces to that defining inductive definitions (see Remark 2.2.4.6) is easily observed once we have introduced them.

Maybe less obvious but nevertheless true is that also every inductive-recursively defined family $(\mathsf{U}, \mathsf{T} : \mathsf{U} \to A)$ where $A : \mathsf{Set}$ is a *small* set —as opposed to a large set in Set_1— can be translated to an equivalent, merely inductive definition[9].

A further degenerate example of an inductive-recursive definition is given by the family $(\mathbb{N}, \mathsf{Fin})$ where the function $\mathsf{Fin} : \mathbb{N} \to \mathsf{Set}$ assigns to every natural number n a set with exactly n elements (so, essentially $\mathsf{Fin}\ n = \{1, \dots, n\}$, see also Example 3.2.1.7). This family can be defined by the constructors

$$\mathsf{zero} : \mathbb{N}$$
$$\mathsf{suc} : \mathbb{N} \to \mathbb{N}$$

(or equivalently as least fixpoint of the functor $X \mapsto X + 1$ on the semantical side) and the recursive definition of the function Fin given by

[7]Where the letter 'W' stands for 'well-order', reminding us of the fact that a tree is exactly a well-ordered relation.

[8]Here and in the following we will pay attention to use the font (U, T) especially for families defined by induction-recursion while we use the font (U, T) for families in general; likewise, we use U for inductively defined sets and U for sets in general.

[9]Albeit not necessarily in Set but in the topos Set/A —a different model of the type Set, see the point "small induction-recursion" in Section 3.1.1 or [58][**gh**].

$$\mathsf{Fin}(\mathsf{zero}) \coloneqq \emptyset$$

$$\mathsf{Fin}(\mathsf{suc}\ n) \coloneqq \mathsf{Fin}(n) \cup \{n\}\ .$$

So, in particular Fin (and more generally every recursively defined function) is defined by specifying values on all constructors having no arguments and all terms that —as expressions— have the form of a constructor binding other constructors or generic terms (e.g. zero is a constructor binding no terms, and $\mathsf{suc}\ n$ is the the term consisting of the constructor suc binding the generic term n) of its inductively defined domain (in this case $\mathbb{N}$) such that the assigned value contains no terms of the inductively defined domain, or only such terms of it that are *structurally smaller* than their preimages (for instance $\emptyset$ does not contain any term of $\mathbb{N}$, and n is structurally smaller that $\mathsf{suc}\ n$).

In this example the indexing object $\mathsf{Set} : \mathsf{Set}_1$ is not small and there is some recursion taking place, but the recursion is not taking place simultaneously with the induction: the constructors zero and suc for $\mathbb{N}$ make no reference to Fin. Nevertheless, both parts of the definition can be integrated such that the family $(\mathbb{N}, \mathsf{Fin})$ —as *one* object— is an[10] initial algebra[11] of the functor[12]

$$\begin{aligned}
&F : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Fam}(\mathsf{Set})\\
&F = \langle F_0, F_1 \rangle\\
\\
&F_0 : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Set}\\
&F_0(U, T) = U + \{*\}\\
\\
&F_1 : ((U, T) : \mathsf{Fam}(\mathsf{Set})) \to F_0(U, T) \to \mathsf{Set}\\
&F_1(U, T)(\mathsf{inl}\ u) = T(u) + \{u\}\\
&F_1(U, T)(\mathsf{inr}\ *) = \emptyset\ .
\end{aligned}$$

This example is interesting because of another aspect that we mention in passing before we come to the next example: it highlights the constructive nature of type theoretic definitions: the axiom of infinity of ZF set theory $\exists\ I(\emptyset \in I \wedge \forall\ x((x \cup \{x\}) \in I))$ does merely assert the existence of an infinite set while the type theoretic definition gives an explicit construction of this infinite set. Indeed there is an explicit set-theoretic construction of $\mathbb{N}$ as well: the von Neumann ordinal has as underlying set exactly $\{\mathsf{Fin}(0), \mathsf{Fin}(1), \mathsf{Fin}(2), \dots\}$ and it is the minimal set satisfying the axiom of infinity. Of course this construction presupposes

[10]We write here cautiously "an" initial algebra, but since by Lambek's theorem all initial algebras of a functor are isomorphic, "the" (up to isomorphism uniquely determined) initial algebra is also a standard terminology that we will use after having stated the theorem.

[11]We will explain initial algebras in Section 2.2.4.4. For the moment, this notion can be understood as the appropriate generalization of "least fixpoint".

[12]Here *inl* and *inr* denote the coproduct inclusions, and by $*$ we denote he unique inhabitant of the unit type. Like every function into a Sigma type like $\mathsf{Fam}(\mathsf{Set})$, also F can be written as $F = \langle F_0, F_1 \rangle$.

the existence of some version of natural numbers to define Fin and can be proven to exist in ZF only by the axiom of infinity.

The motivating example for (nondegenerate) induction-recursion is an axiomatic description of a universe closed under application of a number of specified set-formers; this axiomatic description is “generic” in the sense that it does not depend on what concretely these set-formers are, but only their type matters for the definition. A universe[13] is here conceived as a pair $(\mathsf{U}, \mathsf{T} : \mathsf{U} \to \mathsf{Set})$ where U is thought of as a set of codes for the objects the universe contains, and T as a function assigning to each such code u the actual set Tu encoded by u. A universe (U, T) closed under Σ types can then be defined by the following pair consisting of a constructor Σ' for U and a recursive definition of T that depend on each other:

$$\Sigma' : (u : \mathsf{U})(\mathsf{T}\ u \to \mathsf{U}) \to \mathsf{U}$$

$$\mathsf{T}(\Sigma'(u,t)) \coloneqq \Sigma(\mathsf{T}u)(\mathsf{T} \circ t)\ .$$

Again, this can equivalently be expressed as an initial algebra of the endofunctor on $\mathsf{Fam}(\mathsf{Set})$ defined by $(U,T) \mapsto (\Sigma_{u:U} T(u) \to U,\ \lambda(u,t) \to \Sigma(Tu)(T \circ t))$. As we said (cf “generic”), this definition works for any constructor $\mathsf{K} : (U : \mathsf{Set}) \to (U \to \mathsf{Set}) \to \mathsf{Set}$ (not only Σ) in the sense that we can define K' in the same way.

0.3 Axiomatizations of Induction-Recursion

An axiomatization of induction-recursion is a set of axioms that can be used to define *all* pairs $(\mathsf{U} : \mathsf{Set}, \mathsf{T} : \mathsf{U} \to D)$ such that U is defined by induction, and (possibly simultaneously) T by recursion on U. The discussion of induction-recursion is usually situated in Martin-Löf Type Theory and also we shall be interested only in this setup throughout this thesis.

0.3.1 DS

One example of such an axiomatization DS for inductive-recursive definitions was devised by Dybjer-Setzer [40][14] (see Section 3.2.1). The system DS consists for each pair of arbitrary large sets $D\ E\ : \mathsf{Set}_1$ of three constructors ι, σ, δ inductively defining a large set $\mathsf{DS}\ D\ E : \mathsf{Set}_1$ whose terms are called *codes.* On this large set is recursively defined a *decoding function*[15]

[13]This type of universe is called a Tarski universe. Another style of universe is that of a Russel universe which is not interesting from the viewpoint of induction recursion since it is only a set, and not a family.

[14]They used the notation OP in place of DS.

[15]In a bit more detail, $[\![\ ]\!]$ is a function in its first argument and a functor in its second argument. We do not consider morphisms of codes which would turn a functor also in its first argument. For a discussion of morphisms of codes, see [49] or Remark 3.2.1.17.

$$[\![\ ]\!] : \mathsf{DS}\ D\ E \to \mathsf{Fam}(D) \to \mathsf{Fam}(E)\ .$$

The meaning of these constructors and their names is that ι $\underline{\text{i}}$ncludes terms of E into $\mathsf{DS}\ D\ E$ and the functor defined by this constructor assigns a constant functor, σ defines an indexed $\underline{\text{s}}$um of functors, and δ defines a sum of functors whose indexing set may $\underline{\text{d}}$epend on earlier stages of the decoding — as such only δ can encode non-degenerate inductive-recursive definitions, i.e. those having inductive arguments like the above mentioned paradigmatic case of the closure of the Tarski universe under Σ. The final step (after the definition of codes, and decoding) in the three component machinery of DS consists of rules asserting that each *endo*functor defined by a code $c : \mathsf{DS}\ D\ D$ has an initial algebra $(\mathsf{U}, \mathsf{T})_c$ defined by the code.

0.3.2 DS'

Dybjer-Setzer [38] gave one more axiomatization of IR: DS' is slightly more complicated than DS and historically preceded DS. We found it however more apt to generalizations for our purposes than DS. Again for every pair of large sets $D\ E\ : \mathsf{Set}_1$, a large set $\mathsf{DS}'\ D\ E$ of codes is defined where each code defines a functor between categories of families and endofunctors are asserted to have initial algebras. Unlike DS, however, the part of the DS' machinery defining codes consists itself of two components $(\mathsf{SP}, \mathsf{Arg})$.

SP which takes only one argument D, stands for "*strictly positive*" referring to the fact that inductive definitions can equivalently be characterized as those definitions arising as initial algebras for endofunctors out of a class of endofunctors defined as the closure under certain set-formers such as binary product and the eponymous operation $A \longrightarrow \_$ where (A is an arbitrary but fixed set and) the underscore indicates that the inductive argument may occur only in strictly positive position to the right of an arrow. This constraint of strict positivity is informed by Cantor's theorem of ZF set theory in an appropriate extension of which the theory of induction-recursion is supposed to have a model: would an inductive occurrence to the left be allowed, the putatively defined object would not be a set in the sense of ZF; the basic example of a non strictly positive functor is the power-set functor $X \mapsto (X \longrightarrow 2)$ (where 2 is the two-element set) which has no fixpoint in ZF sets according to Cantor's theorem and as such does not define any ZF set. Dybjer-Setzer (following other authors) tried to carry over a notion of strict positivity from sets to families by requiring that in the constructors for a family (U, T) defined by DS' (or DS) all occurrences of U in its constructors must be strictly positive while T (and values of T) may appear in negative position (i.e. to the left of an arrow); for this style of definition they adopted the terminology of "half positivity".

$\mathsf{Arg} : \mathsf{SP}\ D \to \mathsf{Set}_1$ is then (for every $D : \mathsf{Set}_1$[16]) defined by recursion on $\mathsf{SP}\ D$. The large set E is finally incorporated in the system by taking a code $c : \mathsf{DS}'\ D\ E$ to consists of two parts $c = (Q : \mathsf{SP}\ D, f : \mathsf{Arg}\ Q \to E)$. The system DS' is thus a *container* (see Definition 1.2.4.1) which determines some of its amenable properties. Since $\mathsf{SP}\ D$ is

[16]The definition of Arg depends on D but we do not use a subscript Arg_D to indicate this.

defined by induction, and Arg by recursion on $\mathsf{SP}\ D$, the definition of DS' is a degenerate inductive-recursive one since $\mathsf{SP}\ D$ does not depend on values of Arg.

⋆ ⋆ ⋆

Dybjer-Setzer showed in [40] that both systems DS and DS' are equivalent under reasonable assumptions.

0.4 Compositionality (Outline of the Thesis)

While Dybjer-Setzer's axioms cover a wide range of examples of inductive-recursive definitions, it is still unclear whether indeed all inductive-recursive definitions are covered by them and this thesis will focus on one way to approach the question whether they do.

It is difficult, though, to find an explicit example of a family (U, T) that —by some reasonable standard[17]— is defined inductive-recursively but cannot be expressed in the system DS (or the equivalent DS').

The next best attempt to decide whether all conceivable inductive-recursive definitions are expressible in DS is to consider not examples, but properties and structure of the system DS. In this thesis we focus on one significant property relevant to every definitional principle whose semantics is given in terms of initial algebras for endofunctors: while it is well known that for any two inductive definitions c and c' there exists a composite inductive definition $c \bullet c'$ such that $[\![\ c \bullet c'\ ]\!] = [\![\ c\ ]\!] \circ [\![\ c'\ ]\!]$ where $\circ$ denotes composition of functors, for inductive-recursive definitions it is unknown whether for every pair of codes there exist such a composite code. This implies that if there are other systems of induction-recursion that satisfy the compositionality property while DS does not, we would have found evidence that not all inductive-recursive definitions are definable in DS.

Thus a program for this thesis is set: in Section 4.1 —after dedicating earlier sections to basics— we give a new equivalent characterization of compositionality for DS in terms of powers of codes by sets: DS satisfies compositionality if and only if for every code c and every set $A : \mathsf{Set}$ there exists a power code $A \longrightarrow c$ with semantics $[\![\ A \longrightarrow c\ ]\!]X = A \longrightarrow_{\mathsf{Fam}} [\![\ c\ ]\!]X$[18]. While it is very unlikely that powers are definable in DS, we could not show that they are indeed not definable and so we took the next best approach to define new systems for induction-recursion for which we can show that they satisfy compositionality. There are two possible ways to arrive at such new systems: first we can restrict the system DS to those definitions that do compose and axiomatise this system; the result is the system UF of *uniform codes* for induction-recursion (see Chapter 5). The other way is to reflect more conceptually on what is needed to define composition for all codes and axiomatize a new system that extends DS by supplementing its constructors

[17]That is: by a sufficiently formal non-syntactical / semantical notion of induction-recursion, see also below in Section 8.2.

[18]Here $\longrightarrow_{\mathsf{Fam}}$ constructs powers (see Section 1.2.1) in the category whose object (large) set is the union $\Sigma_{D:\mathsf{Set}_1}\mathsf{Fam}(D)$ of all families for all index (large) sets $D : \mathsf{Set}_1$.

with new axioms implying compositionality; the result is the system of *polynomial codes*[19] for inductive-recursive definitions (see Chapter 6).

0.4.1 Why Composing Codes?

Before we proceed to an overview of the new axiomatizations of compositional induction-recursion developed in this thesis, one might wonder whether, and —if so— which intrinsic reasons there are to be interested in composing IR codes other than regarding compositionality as a classifying criterion for defintional principles.

On the practical side, compositionality offers the option to split more complicated inductive-recursive definitions into simpler ones and to study these parts in separation. A simple example for this is the type of "node-A-labeled finitely-branching trees with finite lists of subtrees" —usually briefly called "rose trees"— which is an initial algebra of

$$F_A(X) \coloneqq 1 + A \times \mathsf{List}(X) \ ,$$

where A is the set providing the node-labels and List is the functor sending a set $X : \mathsf{Set}$ to the set of finite list with entries taken from X. If we write

$$\begin{aligned} &L : \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set} \\ &X \mapsto (Y \mapsto 1 + X \times Y) \ , \end{aligned}$$

$\mathsf{List}(X)$ is an initial algebra of the functor $L(X)(-)$, and $F_A = L(A)(-) \circ \mathsf{List}$. Thus one can describe the type of rose trees by only referring to the simpler type of lists and the generic composition operation and it is natural to want to express this also more generally on the level of codes.

From the theoretical point of view, the problem of characterizing induction-recursion semantically for example by closure properties of the class of functors IR codes define is interesting. As for comparison to the reference theory of inductive definitions, the class of functors defining inductive definitions is the smallest class of functors between the appropriate categories containing the pullback functors and their adjoints, and is closed under composition and natural isomorphism [45, Corollary 1.14]. Of course one can construct categories whose class of morphisms is supposed to be that on IR functors only if the latter are closed under composition. Thus compositionality can form the necessary basis of this line of research.

0.4.2 Uniform Codes for Induction-Recursion (UF)

We come to an overview of the alternative axiomatizations of IR allowing for composition of codes we are going to present in Chapter 5 and Chapter 6.

[19]Not to be confused with codes for polynomial functors; the latter are exactly inductive definitions.

The system UF of *uniform codes* can be regarded —via a non-trivial translation— as a subsystem of DS consisting of codes that have a uniform structure. It is derived from the conjecture that one can compose every *explicitly given* pair of DS codes all of whose paths have the same length up to an isomorphism of their semantics. Since $\mathsf{DS}\ D\ E$ is defined by induction, it is (as recalled in Remark 2.2.4.4) a set $\mathsf{W}(A, B)$ of trees that share a common signature (A, B) (see Remark 3.2.1.2). If composition of codes is —or involves— grafting of trees[78, Proposition 1.1.19], it is not clear that the composite of two trees has the same signature (A, B). It has however some plausibility —backed up by computing examples— that application of semantics-preserving operations (see Example 3.2.1.12) on the trees to be composed can bring them in a form allowing for their composition and such that the composite is in a form complying with the prescribed signature. The problem is that these operations cannot be defined inductively on the system DS. In other words, there is a difference between being able to compose any two given codes, and recursively defining a function

$$\_ \bullet \_ \ :\ \mathsf{DS}\ D\ E \to \mathsf{DS}\ C\ D \to \mathsf{DS}\ C\ E.$$

We resolve this problem in the passage from DS to UF by constraining the "tree shape" of the code to a more uniform structure. The axioms of $\mathsf{UF}\ D\ E$ ensure that its codes correspond to trees all whose branches have the same length; this succeeds essentially by reversing the *nestings* of DS codes which has the effect that the existence of later stages in one branch is aligned with the existence of later stages in all other branches.

From the categorical point of view, UF is similar to DS' in that it is a (large) container; it differs however from DS' in that it is itself defined by non-degenerate induction-recursion.

The system UF is a priori smaller than DS and cannot directly express DS-codes with unbounded branch length (see Remark 3.2.3.4). But there is an elementary procedure to replace a DS code whose maximal branch length is explicitly given, by a DS-code with uniform branch length having isomorphic semantics such that one can find a UF-code with isomorphic semantics corresponding to this DS-code. $\mathsf{UF} \hookrightarrow \mathsf{DS}$ being a semantics-preserving subsystem of DS exempts us from providing a new model for the axioms of UF since the existence of such a model is implied by the inclusion of UF into DS for which by Dybjer-Setzer already gave a model in [40].

0.4.3 Polynomial Codes for Induction-Recursion (PN)

To motivate the system PN of *polynomial codes* which we call by this name because like polynomials[20] it is constructed from sums and powers[21]: while DS has only constructors forming sums of codes -in the sense of forming codes that decode to sums of functors-, PN has an additional constructor forming powers of codes in this sense. For further

[20] Polynomials in category theory (which are generalizations of polynomials over a ring as studied in algebra) are also called containers and encode inductive definitions (see Section 2.2.4.5. This motivates the name "polynomial codes" also in a more technical sense explained in Remark 6.1.0.1, namely as an inductive definition having as base case an inductive definition, i.e. as an "iterated container".

[21] Or more generally, dependent products.

explication, we return to the characterization of composable DS codes as codes for which powers of codes by sets exist. Even if we could not prove that powers are definable in DS, we still can try to pursue to option to stipulate the existence of powers by way of a new axiom. Simply adding a constructor

$$\_ \longrightarrow \_ : (A : \mathsf{Set}) \to \mathsf{DS}\ D\ E \to \mathsf{DS}\ D\ (A \to E)$$

encounters however a problem: it seems to be impossible to extend the inductive definition of $\bullet$ to this new constructor. The problem is related to the observation that the '$(A \to E)$' in the codomain of the constructor $\longrightarrow$ seems to break functoriality of the system DS in the second argument: the operation $\mathsf{DS}\ D\ \_ : \mathsf{Set}_1 \to \mathsf{Set}_1$ is not only a functor but even a monad, and the latter fact was used in the definition of composition $\bullet$ on DS under the assumption of the existence of powers of codes. If it does indeed break functoriality, DS and $\mathsf{DS}{+}\!\longrightarrow$ are different since DS is functorial.

The definition of PN proceeds consequently by giving axioms for a system that 1) subsumes the axioms of DS, 2) has a constructor for powers[22], and 3) makes $\mathsf{PN}\ D\ \_$ a monad (for every $D : \mathsf{Set}_1$). All these requirements can be achieved by designing the new system (like DS' and UF) as a container ; PN is like UF itself defined by non-degenerate induction-recursion. Thus defined PN indeed enjoys compositionality and we give a model (see Section 6.4) proving the consistency of its axioms since this time we have only an inclusion $\mathsf{DS}\ D\ E \hookrightarrow \mathsf{PN}\ D\ E$ and no inclusion in the other direction at our disposal and thus cannot obtain a model for PN from a model for DS. This model has almost the same set-theoretical assumptions concerning large cardinals: while Dybjer-Setzer induction-recursion can be modeled in ZFC supplemented by a Mahlo cardinal and a 0-inaccessible (see Definition 1.1.0.8), we need ZFC plus a Mahlo cardinal (see Corollary 1.1.0.21) and a 1-inaccessible.

Finally, since we did not arrive at the system PN by simply adding one constructor for powers but had to set up the whole system entirely different to accommodate a constructor for powers, it is a natural question whether the system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ obtained by removing the constructor for powers from PN has a translation back to DS. We show in Section 6.7 that there is at least a translation if codes of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ are annotated with additional information about their nesting.

0.5 Formalization in Agda, and Notation

Most of the content of this thesis has been formalized in the proof assistant Agda. This formalization is available at `https://bitbucket.org/stephanspahn/thesis-3/src/master/thesis-agda/agda/`.

In the present text we treat Agda informally as a version of MLTT with Σ-types (aka dependent pair types), Π-types (aka dependent function types), (non-dependent) function

[22]Even though the construction goes through with a constructor for powers, we mainly consider the slightly more general case of dependent products of codes in place of powers.

types, an empty type, a unit type, a cumulative hierarchy of two Russel style universes $\mathsf{Set} : \mathsf{Set}_1$, identity types, judgemental equality, and inductive- , as well as inductive-recursive definitions.

In addition to the syntax of type theory introduced in Section 2.2.3, we synonymously use the notations of the forms '$\lambda\ x \to f(x)$' and '$x \mapsto f(x)$' for anonymous assignments. We use both notations $f(x)$ and $(f\ x)$ for the application of a function to one of its arguments. For families $Z = (U, T) : \mathsf{Fam}(D)$, we use the notation $\mathsf{ind}\ Z = U$ and $\mathsf{fib}\ Z = T$. We use Agda convention to put arguments in type signatures in braces $\{\ ,\}$ if we do not want to display them as arguments of the terms of the types of this signature, and we use the Agda notation $\forall\{X\}$ in a type signature if we do not wish to make explicit the type of X; for this, an expression containing the expression $\{X = Y\}$ means that the (not to be displayed) variable X is substituted by (the not to be displayed) Y in this expression.

0.6 Structure of the Thesis

The rest of this thesis is structured as follows:

- Chapter 1 reviews the foundations we shall need: category theory and basics about ordinals are needed for initial algebra semantics of type theory (the latter will be introduced in the subsequent chapter). Families are the objects defined by induction-recursion. Monads play an important role for composition of inductive-recursive definitions.

- Chapter 2 opens by giving a brief historical overview on induction. Martin-Löf type theory, the foundational theory in which the main part of the thesis is situated is introduced with a focus on inductive types.

- Chapter 3 reviews Dybjer-Setzer's two axiomatizations DS and DS' of induction-recursion as well as decoding of codes to functors between categories of families.

- Chapter 4 develops an equivalent characterization of compositionality for DS codes in terms of powers of codes by sets.

- Chapter 5 gives a new axiomatization of inductive-recursive definitions that enjoys compositionality. This system UF can be regarded as a subsystem of DS. Codes and their decoding as functors between categories of families are defined. Differences and commonalities to the system DS are discussed and illustrated by examples.

- Chapter 6 defines a second new axiomatization of induction-recursion with composition for all codes. The system PN subsumes DS. Syntax, decoding of codes, and a set-theoretic model are given. Section 6.7 discusses a new system of codes lying between DS and PN a translation of this intermediary system into DS is given.

- Chapter 7 is independent from the chapters on compositionality of induction-recursion and can be regarded as a separate part of the thesis which contains its own motivation-, introduction-, and foundations sections. Here a relational-parametric model of Dybjer-Setzer's version DS of induction-recursion is given. Relational parametricity is a standard property expected from type systems.

0.6.1 Publications and Contributions

Parts of the material concerning the characterization of compositionality of DS, and the systems UF and PN were published in [52] and [51](together with Neil Ghani, Conor McBride, and Fredrik Nordvall Forsberg).

I collaborated to the general development of all the material present in [52] and [51]. In more detail, my main contributions were the following:

- I had-, and developed the idea to characterize compositionality for DS in terms of the bind operation on DS, and proved Theorem 4.1.0.2.
- I gave the definition of UF its final form departing from a preexisting definition, which was "right nested" and a "one-level system" (as opposed to a container); this preexisting system was developed by Conor McBride building on ideas of Peter Hancock. I gave the appropriate composition operation departing from the composition operation for the previous system. The significance of this contribution is that it provided a simplification which made it possible to prove that the composition operation of this new system commutes with semantics while it is unclear whether the composition operation for the previous system is semantically sound. I contributed to the proofs of Lemma 5.9.1.3, and Theorem 5.9.2.1 and their formalization in Agda; also Fredrik Nordvall Forsberg contributed to this formalization.
- I contributed to the formalization of the proof of Theorem 6.5.2.1. The system PN is inspired by a definitional principle for inductive definitions [24] by Conor McBride (and coauthors) that contains a constructor for products of codes. Neil Ghani extended this system to a version of induction-recursion with composition ($\mathsf{PN}_{\mathsf{cont}}$) which is one of the systems of compositional induction-recursion we discuss here; a second one (PN) which is a slight modification of $\mathsf{PN}_{\mathsf{cont}}$ was introduced to the case of induction-recursion by Fredrik Nordvall Forsberg and myself but the changes made are already implicit in Conor McBride's work. I redefined the embedding of DS into PN (see Proposition 6.7.1.4. and Section 6.7.2) in a way revealing that and embedding does not necessarily use π-codes which is the point of departure of Section 6.7.

The following parts were not published in [52] or [51].

- The study of the intermediary systems Section 6.7 is my own work entirely. The significance of the intermediary system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont\,bt}}$ is that there is a translation $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont\,bt}} \to \mathsf{DS}$ while it was not possible to find such a translation to DS from the system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ from which $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont\,bt}}$ is derived by annotation of codes by binary trees; the choice of exactly this form of annotation is not obvious (we found that an annotation with e.g. natural numbers did not work in the desired way). More interesting than the system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont\,bt}}$ itself is perhaps the annotation we found which establishes a way to enforce uniformity conditions on codes that is different from the one used for UF; this can be helpful for future research on the comparison between different axiomatizations of induction-recursion.

- Chapter 7 is my own work entirely. I presented an earlier version of this material at the workshop [116]. The significance of a relational-parametric model for induction-recursion is that it will make it possible to prove metatheoretical properties of MLTT extended by induction-recursion. For example, I am presently working on a relational-parametricity proof of the conjecture that for all definable DS-codes there exists an UF-code with isomorphic semantics (this work is not contained in this thesis).

Chapter 1

Foundations

While we assume some familiarity with ZFC set theory and category theory, we will review the material that is of explicit importance for us in this section. There is a separate chapter containing an introduction to type theory (see Section 2.2).

Section 1.3 on families, and Section 1.2.4 on containers are of most direct relevance for IR. The other sections in this chapter reviewing cardinals, ordinals, and initial algebras for endofunctors are needed to explain the semantics-, and set-theoretic model of induction-recursion; Mahlo cardinals which are of integral importance for models of IR are best introduced directly in the context where they are used (see Section 6.4). Monads are relevant for composition of inductive-recursive definitions, the central topic of this thesis.

1.1 Ordinals and Cardinals

Cardinals and ordinals are used for the construction of initial algebras for endofunctors and thus for establishing models of type theory by initial-algebra semantics. An application of the definitions in this section will be the construction of a model for a version of IR in Section 6.4; initial algebra semantics will be recalled in Subsection 1.2.3. A good reference for the topic is [71]. A general reference for set theory is [68]; in particular an introduction to ordinals and cardinals (which we do not repeat here) is in loc. cit. §I.2 - I.3.

Set-theoretic models of IR are not models in sets defined only by the axioms of ZFC but the latter set theory needs to be supplemented by an axiom asserting the existence by an appropriate *strongly inaccessible cardinal.* Intuitively these cardinals are larger than any set that can be constructed by ordinary set operations (e.g. union, formation of limits).

Definition 1.1.0.1 (Ordinal). *An* ordinal *is defined to be a transitive set that is totally ordered by set inclusion. We denote the class of all ordinals by* Ord*. For* $\alpha, \beta \in \mathsf{Ord}$*, we write* $|\alpha| = |\beta|$ *to express that there exists a bijection between* α *and* β*. (see [68, Definitin 2.10]).*

Definition 1.1.0.2 (Cardinal). *An ordinal* α *is called a* cardinal *if* $|\alpha| \neq |\beta|$ *for all* $\beta < \alpha$*. ([68, p.29]).*

Definition 1.1.0.3 (Limit of a sequence of ordinals). *For an increasing sequence of ordinals* $(\gamma_\xi)_{\xi \in \beta}$*, where* $\alpha \in \mathsf{Ord}$*, and either* $\beta \in \mathsf{Ord}$ *or* $\beta = \mathsf{Ord}$*, we define* $\lim_{\xi \to \alpha} \gamma_\xi := \sup\{\gamma_\xi | \xi < \alpha\}$*.*

Definition 1.1.0.4 (Cofinality).

1. *Let* α *be a limit ordinal. The* cofinality of α *is defined to be the minimum* $\mathsf{cf}(\alpha) = \mathsf{min}\{\beta | \exists$ *increasing* β*-sequence* $(\alpha_\xi)_{\xi<\beta}, \lim_{\xi\to\beta} \alpha_\xi = \alpha\}$ *(see [68, p.33]).*
2. *We call* β *to be* cofinal in α *if there exists an increasing* β*-sequence* $(\alpha_\xi)_{\xi<\beta}$ *such that* $\lim_{\xi\to\beta} \alpha_\xi = \alpha$*.*

For Definition 1.1.0.6 we need to recall ordinal arithmetic:

Definition 1.1.0.5 (Odinal arithmetic). *On ordinals three binary operations are defined by transfinite recursion:*

1. *For all* α
   - *(a)* $\alpha + 0 = \alpha$*.*
   - *(b)* $\alpha + (\beta + 1) = (\alpha + \beta) + 1$*, for all* β*.*
   - *(c)* $\alpha + \beta = \lim_{\xi\to\beta}(\alpha + \xi)$ *for all limit* $\beta > 0$*.*
2. *(a)* $\alpha \cdot 0 = 0$*.*
   - *(b)* $\alpha \cdot (\beta + 1) = (\alpha \cdot \beta) + \alpha$*, for all* β*.*
   - *(c)* $\alpha \cdot \beta = \lim_{\xi\to\beta}(\alpha \cdot \xi)$ *for all limit* $\beta > 0$*.*
3. *(a)* $\alpha^0 = 1$*.*
   - *(b)* $\alpha^{\beta+1} = \alpha^\beta \cdot \alpha$ *for all* β*.*
   - *(c)* $\alpha^\beta = \lim_{\xi\to\beta} \alpha^\xi$ *for all limit* $\beta > 0$*.*

We shall sometimes write 1 *for the empty set, and* $2 = 1 + 1$*. (See [68, p.23-24] for more on ordinal arithmetic.)*

Definition 1.1.0.6 (Regular cardinal, (weakly-, and strongly-) inaccessible cardinal). *1.*
A cardinal λ *is* (weakly) inaccessible *if it is infinite, (a limit cardinal), and* regular *(i.e. it is equal to its cofinality (see Definition 1.1.0.4)* $\mathsf{cf}(\lambda) = \lambda$*).*

2. *A cardinal λ is* (strongly) inaccessible *if it is weakly inaccessible, and $2^\kappa < \lambda$ for all $\kappa < \lambda$ (see [68, p.58]).*

Since we shall not use the notion of weakly inaccessible cardinal, for us "inaccessible cardinal" shall always mean "strongly inaccessible cardinal".

Lemma 1.1.0.7. 1. *The existence of inaccessible cardinals is not provable in ZFC (see [68, Theorem 12.12].*

2. *The* Generalized Continuum Hypothesis *(see [68, p.55]) implies that every weakly inaccessible cardinal is strongly inaccessible.*

Definition 1.1.0.8 (α-inaccessible cardinal, unbounded set). *Let α be any cardinal. A cardinal κ is α-*inaccessible *if it is inaccessible, and for every $\beta < \alpha$, the set of β-inaccessibles less than κ is* unbounded *in κ. Here, a set $X \subset \alpha$ of a limit cardinal α is unbounded if the supremum of X equals α.*

It obviously follows from the previous definition that a 0-inaccessible cardinal is just an inaccessible cardinal.

Terminology 1.1.0.9 (Large cardinal). *There is also the terminology of "large cardinal". There is no precise definition of this term (e.g. it is used without definition in [68]) but a large cardinal should at least be inaccessible. We will be interested only in cardinals which are i-inaccessible, smaller than that, or* Mahlo *(which we will introduce later in Corollary 1.1.0.21).*

Ordinals can be used to define the so-called *cumulative hierarchy* which is a family of sets indexed by ordinals in the following way:

Definition 1.1.0.10 (The cumulative hierarchy). *[68, p.64]*

The cumulative hierarchy is defined by

$$\begin{aligned} &\mathsf{V}_0 = \emptyset \\ &\mathsf{V}_{i+1} = \mathsf{P}\ \mathsf{V}_i \\ &\mathsf{V}_i = \cup_{j<i} \mathsf{V}_j \quad \textit{if } i \textit{ is a limit ordinal} \end{aligned}$$

where P is the operation assigning to a set A, its (decidable) powerset $\mathsf{P}\ A : A \to 2$.

Lemma 1.1.0.11. *The union $\bigcup_{\alpha \in \mathsf{Ord}} \mathsf{V}_\alpha$ ranging over the class Ord of all ordinals equals the* universal class $\mathsf{V} = \{\ x \mid x\ =\ x\ \}$ *(see [68, Lemma 6.3]).*

Definition 1.1.0.12 (Rank). *By Lemma 1.1.0.11 every set A is contained in an V_ρ for some ρ. The minimal such ρ is called the* rank $\mathsf{rank}(A)$ *of A.*

We will need the following lemma in Chapter 7.

Lemma 1.1.0.13. *Let κ be a regular cardinal, $(U^\alpha)_{\alpha<\kappa}$ be an increasing series of sets of rank $\alpha < \kappa$, $f : A \to \bigcup_{\alpha<\kappa} U^\alpha$, and $\mathsf{rank}(A) < \kappa$.*

Then there exists a $\beta < \kappa$ such that the image of f is contained in U^β — in symbols $f : A \to U^\beta$.

Proof. Define

$$\begin{aligned}&g : \mathsf{rank}(A) \to \kappa\\ &g(\lambda) = \min\{\beta|(\forall\lambda' < \lambda)(g(\lambda') < \beta) \wedge \forall\, x \in A \cap \mathsf{V}_\lambda, f(x) \in \bigcup_{\alpha<\beta} U^\alpha\}\ ,\end{aligned}$$

and notice that $g(\lambda) < \kappa$, and g is increasing (for this the first conjunct is needed). Assume $\mathsf{lim}_{\lambda\to\mathsf{rank}(A)}\, g(\lambda) = \kappa$. This implies $\mathsf{rank}(A)$ is cofinal in κ. It follows $\mathsf{rank}(A) \geq \mathsf{cf}(\kappa) = \kappa > \alpha$, contradiction. □

Important about the cumulative hierarchy is the following fact.

Lemma 1.1.0.14. *If I is a strongly inaccessible cardinal, V_I is a model of ZFC (see [68, Lemma 12.13]).*

Remark 1.1.0.15 (Reflection Principle). *Sometimes, statements of the form of previous lemma are called reflection principles (see [79] for a discussion). The idea is that properties we believed to hold for the universal class V (or some other totality) is* reflected *by an "initial segment" like V_I.*

This idea of reflection principles motivates the notion of a universe in type theory*, which are in turn the original motivation for induction-recursion (compare in particular Section 3.3).*

The just explained philosophical idea of reflection principles should not be confused with a more technical meaning of the term "reflection principle" in set-theoretic model theory explained in [68, p.186] which will be of no direct interest to us in this thesis.

So, in particular, if we write "ZFC $+\mathsf{I}$" for some inaccessible cardinal, we mean the theory ZFC supplemented by an axiom asserting the existence of I, and thereby that V_I is a set in the theory ZFC + I.

In particular, as will be explained in Subsection 1.2.3 the result of iteratively applying an endofunctor (on Set) to the empty set is contained in the cumulative hierarchy if the iteration is indexed by an ordinal smaller than the ordinal defining the cumulative hierarchy.

We will refer to the following definition in the discussion of codes for induction-recursion in Example 3.2.1.14 and Remark 3.2.3.4.

Definition 1.1.0.16 (Continuous sequence, normal sequence, normal function). *Let β be a ordinal with uncountable cofinality, or $\beta = \mathsf{Ord}$. A sequence $(\gamma_\xi)_{\xi\in\beta}$ of ordinals is called a* normal sequence *if it is increasing and continuous. In symbols, continuity of this sequence means that for every limit $\alpha < \beta$ we have $\gamma_\alpha = \lim_{\xi\to\alpha} \gamma_\xi$.*

Of course, we call a function $f : \beta \to \beta$ a normal function *if the associated sequence $\gamma_\xi := f(\xi)$ for $\xi \in \beta$ is a normal sequence. ([83, Definition 4.11, (iii)], Normal sequences are also defined in [68, Definition 2.17], but only for functions $f : \mathsf{Ord} \to \mathsf{Ord}$.)*

Definition 1.1.0.17 (Club set). *Let κ be a limit. A subset $S \subseteq \kappa$ is called a* club set *(this is a contraction of <u>cl</u>osed <u>unb</u>ounded) if*

1. *S is* closed in κ*, i.e. if:* $\forall\, \alpha \leq \kappa\ (\sup(S \cap \alpha) = \alpha \neq 0 \implies \alpha \in S)$.
2. *S is* unbounded in κ*, i.e. if:* $\forall\, \alpha < \kappa\ (\exists \beta \in S \wedge \alpha < \beta)$.

Definition 1.1.0.18 (Stationary set). *Let κ be a limit with $\mathsf{cf}(\kappa)$ being uncountable. A subset $S \subseteq \alpha$ is a* stationary subset of κ *if S has a nonempty intersection with every club set in κ.*

Definition 1.1.0.19 (Mahlo cardinal).
1. *A limit cardinal M is a* weakly Mahlo cardinal *if it is weakly inaccessible and the set of all regular cardinals below M is stationary. (Then also the set of all weakly inaccessibles below κ is stationary in κ.)*
2. *A limit cardinal M is a* strongly Mahlo cardinal *if it is strongly inaccessible and the set of all regular cardinals below M is stationary. (Then also the set of all strongly inaccessibles below κ is stationary in κ.)*

[68, pp.95-96]

Lemma 1.1.0.20 (Fixed-point theorem for normal functions). *[83] Let κ be an ordinal with uncountable cofinality. Then the set of fixed points of every normal function $f : \kappa \to \kappa$ is a club set. (This follows from [83, Proposition 4.12])*

Corollary 1.1.0.21 (Mahlo property). *Let M be a Mahlo cardinal (weakly or strongly), and $f : \mathsf{M} \to \mathsf{M}$ be normal. Then f has a regular fixed point. If M is weakly Mahlo, f has a weakly inaccessible fixed point, and if M is strongly Mahlo, f has a strongly inaccessible fixed point.*

Proof. By Lemma 1.1.0.20, the set of fixed points of f is a club set. Since the set *Reg* of regular cardinals below M is stationary, the intersection of *Reg* with the set of fixed points of f is nonempty. The claims in the second sentence follow analogously. □

We will usually refer to a Mahlo cardinal via the Mahlo property.

1.2 Category Theory

We assume familiarity with the basic definitions of category, functor, and natural transformation. In this section we review the notions that are of particular importance for us. A general reference useful for our purposes is [13] —in particular §9, Monads and Algebras. Additional references are [19, §6.5, Tensors and cotensors], and [73, §3.7, Tensor and cotensor products].

1.2.1 Power Objects

As we will see in Section 4.1, there is a close relation between compositionality of codes and *power objects*. Power objects are for example discussed in [69, A2 1 1] in categories with products.

Definition 1.2.1.1. *Let $(C, \otimes)$ be a monoidal category, $c \in C$ an object, and $A : \mathsf{Set}$ a (small) set. The* power (object) or cotensor c^A of c by A *is defined to be an object such that there is an isomorphism*

$$\hom_C(x, c^A) \simeq \hom_{\mathsf{Set}}(A, \hom_C(x, c))$$

which is natural in $x \in C$. By the Yoneda lemma, c^A is essentially unique in case it exists.

Lemma 1.2.1.2. *If C is a locally small cartesian closed category, all power objects exist, and —for $c \in C$ an object, and $A : \mathsf{Set}$— can be defined by $c^A := \Pi_{a:A} c$.*

1.2.2 Exponentials

We shall need the following lemma in Chapter 7. Before stating it, we recall the following definition:

Definition 1.2.2.1 (Global element). *If U is an object in a category with terminal object, a morphism $x : 1 \to U$ is called a* global element.

Lemma 1.2.2.2 (Evaluation of exponential objects).

1. *Recall that if A^X is an exponential object, then there is a map $ev : A^X \times X \to A$. If $f : 1 \to A^X$ and $a : 1 \to X$ are global elements, we write $fa : 1 \to A$ for the global element $ev \circ (f, a)$ defined by f and a.*
2. *There is a composition operation for exponential objects which we denote by $\circ$. It is obtained from the map ev and the adjunction $\_ \times A \dashv \_^A$ via the following transpositions:*

$$\dfrac{\dfrac{\dfrac{\circ : X^Y \times Z^X \to Z^Y}{X^Y \to (Z^Y)^{(Z^X)}}}{X^Y \to Z^{Y \times Z^X}}}{ev \circ (ev \times id) : X^Y \times Y \times Z^X \to Z}$$

We will use the previous lemma often without further mentioning.

1.2.3 Algebras for Endofunctors

The importance of *algebras for endofunctors $F : C \to C$* in regard to semantics of type theory is that their initial algebras interpret inductively defined types since the system of canonical maps out of the initial algebra to any other algebra represent the principle of recursion on this inductive type. Which endofunctors have initial algebras depends on the category C as well as on assumptions on the underlying set theory. An introduction to this subject is [65]. A reference for initial algebras for polynomial functors (which are of particular interest for us) is [1, §5]. We will briefly return to the relation of initiality and type definitions in general in Section 2.2.4.4.

As a caveat we mention that the terminological distinction between of induction and recursion is potentially confusing - in particular when talking about induction-recursion: in context of categorical semantics of type theory one usually identifies types with initial algebras for endofunctors on Set, since however the object representing an inductive-recursive definition is itself a family (see Section 1.3) such as $(\mathsf{U}, \mathsf{T}) : \mathsf{Fam}(\mathsf{Set})$ where $\mathsf{T} : \mathsf{U} \to \mathsf{Set}$ is regarded as to be defined by recursion (on the simultaneously inductively defined U), one is entitled to call the object (U, T) to be defined by induction (in the category $\mathsf{Fam}(\mathsf{Set})$). Now, even though $\mathsf{U} : \mathsf{Set}$, it is not defined by induction in the sense that there would be a polynomial functor of which it is initial algebra.

The purpose of this (and the following) section is not to give a comprehensive introduction to initial algebra semantics (which can be found in the given references) but mainly to recall that polynomial endofunctors (on Set) do have initial algebras in ZF [1] ; we will see later in Section 6.4 that the existence of initial algebras for endofunctors defining inductive-recursive definitions needs the additional assumption of large cardinals (see Terminology 1.1.0.9).

Definition 1.2.3.1 ((Initial) algebra of an endofunctor). *An algebra for an endofunctor $F : \mathcal{C} \to \mathcal{C}$ is a pair (X, α) where $X \in Ob\,\mathcal{C}$, and $\alpha : F(X) \to X$ is a morphism. A morphism $H : (X, \alpha) \to (X', \alpha')$ between F-algebras consists of a morphism $h : X \to X'$ making the following diagram commute:*

$$\begin{tikzcd}
F(X) \arrow[r, "\alpha"] \arrow[d, "F(h)"'] & X \arrow[d, "h"] \\
F(X') \arrow[r, "\alpha'"] & X'
\end{tikzcd}$$

An initial F-algebra is an initial object in the category of F-algebras.

We will sometimes leave the algebra-structure map α being part of the initial algebra (X, α) implicit and refer to X as "the initial algebra".

Lemma 1.2.3.2 (Lambek's lemma [80]). *If X is an initial algebra for an endofunctor, then there is an isomorphism $F(X) \simeq X$.*

1.2.3.1 Initial chains for endofunctors and fixpoint theorems

The most common way to construct initial algebras for endofunctors is by iterative application of the endofunctor to an initial object.

[1] In the category Set, there is a canonical choice of an initial object, and thus the initial algebra defined by the initial sequence of an endofunctor is a canonical choice for an initial algebra. In categories other than Set, one needs in general the axiom of choice to choose one out of possibly infinitely many isomorphic initial algebras.

Definition 1.2.3.3 (Initial (finitary) ω-chain, colimit of an initial ω-chain). *Let C be a category with an initial object $0 \in Ob\ C$, let ω be a countable ordinal, let $F : C \to C$ be an endofuctor, and for all $n < \omega$ let F^n be defined by $F^1 = F$ and $F^{n+1} = F \circ F^n$. The chain*

$$0 \xrightarrow{!} F0 \xrightarrow{F(!)} F^2 0 \xrightarrow{F^2(!)} \dots \xrightarrow{F^{n-1}(!)} F^n 0 \xrightarrow{F^n(!)} \dots$$

is called the initial ω-chain of F. A cocone for the initial ω-chain of F *is defined to be a pair $(A, (\alpha_n : F^n(0) \to A)_{n<\omega})$ such that $\alpha_n = \alpha_{n+1} \circ F^n$ for all $n < \omega$. A* colimit for the initial ω-chain of F *is defined to be a cocone $(C, (c_n : F^n(0) \to A)_{n<\omega})$ together with a map $f : C \to A$ for every cocone $(A, (\alpha_n : F^n(0) \to A)_{n<\omega})$ factoring in the obvious way.*

Theorem 1.2.3.4 (Construction of an initial algebra via (finitary) initial ω-chains). *Let C be a category with an initial object and colimits of (finitary) initial ω-chains. Then, if $F : C \to C$ preserves colimits of initial ω-chains, then F has an initial algebra μF given by $\mu F = colim_{n<\omega} F^n 0$.*

Proof. [65, Theorem 2.1.9].

Finitary initial chains are not sufficient for obtaining all inductive definitions of interest in ZFC. But fortunately, there are versions of initial chains and an assorted fixpoint theorem for uncountable limit ordinals.

Definition 1.2.3.5 (Initial chain, colimit of an initial chain). *Definition 1.2.3.3 can be extended to limit stages [65, §3.1] by putting*

$$F^j 0 = \mathsf{colim}_{i<j} F^i 0$$

for every limit ordinal j where the morphisms in the extended chain are induced by the universal property of the colimit.

The following theorem is a corollary of a theorem of Zermelo-Knaster-Tarski stating that least fixpoints for monotone functions of directed complete partial orders exists.

Theorem 1.2.3.6 (Construction of an initial algebra via initial chains). *[65, Theorem 3.1.1, Theorem 3.1.4] Let C be a category with an initial object 0 and with colimits of chains. If the initial chain of an endofunctor $F : C \to C$ converges in α steps (i.e. $F^\alpha(0) = F^{\alpha+1}(0)$), then $F^\alpha 0$ is an initial algebra of F where the algebra structure map is given by the inverse of the chain map $F(F^\alpha 0) \to F^\alpha$.*

Corollary 1.2.3.7. *Let C be a category with an initial object 0 and with colimits of chains. If an endofunctor $F : C \to C$ preserves colimits of α-chains, then the initial chain of F converges in α steps and the initial algebra of F is $F^\alpha(0)$. [65, Corollary 3.1.5].*

1.2.4 Polynomials / Containers, Trees, and their Associated Functors

We inherit the synonymous terms *polynomial* and *container*, and the closely related term of *tree*; while the first-mentioned two terms are really exact synonyms, they provide a formalization of the informal idea of a (wellfounded) tree. We follow the convention to talk about "containers" (rather than of "polynomials") if not followed by "functor", of "polynomial functors" (rather than of "container functors"), and of "trees" when referring to terms of the type defined by a polynomial functor.

The most important two facts about containers are that 1) there is an equivalence between the category of containers and the category of polynomial functors on Set (see [45, p.11]), and 2) every polynomial functor has an initial algebra and these initial algebras are exactly inductively defined sets.

Polynomial functors are a generalization of polynomial functions from rings to sets [78, p.1], and more generally to other locally cartesian closed categories [45][44], and even more generally to categories with only pullbacks [127]. The notion (in one variant or another) has been introduced many times according to loc.cit. where the first mention of the term is in Eilenberg-McLane's 1942 paper [41]. A more systematic theory of polynomial functors has been developed by Joyal, Kock, and Gambino [77][76][45]; [78] is a draft of a book on the topic. The theory of polynomial functors is already a well developed one: a collection of the most important results is summarized in [45, p.11] characterizing polynomial endofunctors of Set in several ways; a partial generalization (from Set to locally cartesian closed categories) of these results is given in [75]. From the viewpoint of topos theory there is work by Moerdijk, Palmgren, and van den Berg on the topic [95][125]. From the viewpoint of type theory, relevant references are [1] which is a study of the category of polynomial functors, and [3] showing that this category is locally cartesian closed.

The theory IR functors is —compared to that of polynomial functors— still at its very beginning and this thesis is merely about finding a 'correct' definition of inductive-recursive definitions - or at least about studying several alternative versions of the notion. And even though the theory of polynomial functors differs in a several aspects from the emerging theory of IR functors, polynomial functors play (at least) two different roles in the theory of IR: firstly, polynomial functors[2] appear as degenerate cases of IR functors since they encode inductive definitions; and secondly, containers will play an organizational role in our axiomatizations of IR.

Definition 1.2.4.1 (Container, (wellfounded) tree, polynomial functor). *1. A* container *is a term* $(A, B) : \Sigma_{A:\mathsf{Set}}(A \to \mathsf{Set})$*. A morphism* $f : (A, B) \to (A', B')$ *of containers is a pair* $(f_0 : A \to A', f_1 : (a : A) \to B'(f_0(a)) \to B(a))$*. We denote the category of containers in* Set *by* Cont*; we will see in Section 1.3 that* $\mathsf{Cont} = \mathsf{Fam}(\mathsf{Set}^{op})$ *is just the category of families of opposite sets. See also [2] for a discussion of categories of containers.*

[2]This can also be seen on the level of containers which are special cases of familiesSection 1.3 (namely families of sets), the objects defined by IR, where, if we consider morphisms of containers and -of families, we have to take care of the direction of morphisms, see below.

2. *Every container (A, B) defines an endofunctor on* Set *by*

$$\begin{aligned}
&[\![\ (A,B)\ ]\!] : \mathsf{Set} \to \mathsf{Set} \\
&X \mapsto \Sigma_{a:A}(B(a) \to X) \\
&(f : X \to Y) \mapsto ((a,t) \mapsto (a, f \circ t)) \ .
\end{aligned}$$

3. *If $H = (H_0, H_1) : (A, B) \to (A', B')$ is a morphism of containers, and $E : \mathsf{Set}$, then there is a function*

$$\begin{aligned}
&[\![\ H\ ]\!]E : [\![\ (A,B)\ ]\!]E \to [\![\ (A',B')\ ]\!]E \\
&[\![\ H\ ]\!]\ E\ (Q,\varphi) = (H_0(Q), \varphi \circ H_1(Q)) \ .
\end{aligned}$$

The above definitions of course also make with Set_1 *in place of* Set*.*

Remark 1.2.4.2. *We can understand a container (A, B) as encoding a set of* trees *with signature (A, B) in the following way: A is the set of nodes (marked with one ingoing branch) of the tree, and for each $a : A$, the set $B(a)$ is the set of branches directly extending from the node a. A tree is* wellfounded *if every branch has finite length. Notice that since $B(a)$ can be infinite, the lengths of paths in a wellfounded tree can by unbounded. Conversely, given one tree, one can define a set A consisting of the nodes of the tree marked with one ingoing branch, and if for each $a : A$, $B(a)$ is the set of edges extending from a, we can define B as the union of all the $B(a)$.*

The category Cont is of central importance for us. Further properties of it will be explained in Section 1.3 in more general context. In particular Cont is equivalent to the category of families of opposite sets and thus inherits some properties of the monad structure of Fam (see Lemma 1.3.0.6), and it is equivalent to the category of inductive definitions (see Section 2.2.4.5). We sometimes use the notation $\mathsf{con} : \mathsf{Set} \to \mathsf{Cont}$, $A \mapsto (A, \lambda\ \_ \to 0)$.

We have the following corollary to Theorem 1.2.3.6.

Corollary 1.2.4.3 (Initial algebras for Polynomial Functors). *Polynomial endofunctors always have initial algebras (in ZF set theory).*

Proof. [65, Theorem 3.1.12] gives details of the proof that such an initial algebra is the set of well-founded trees arguing via initial ω-chains Definition 1.2.3.5 and using Theorem 1.2.3.6 for the existence statement. The proof shows that —for the polynomial functor defined by $f : B \to A$, the initial chain converges in at most α steps where α is the first regular cardinal larger than the cardinality of B.

Remark 1.2.4.4. *For later reference (in Section 6.4), we notice here that for establishing initial algebras for IR functors, the assumptions proving Corollary 1.2.4.3 are too weak since unlike in case of a container (A, B) where B can be used to obtain an upper bound for the cardinality of the sets occurring in the initial chain, in the definition of an IR functor* *such a bound needs to be computed by induction on the code (see Section 3.2.1) defining the functor, and is much higher. Indeed establishing initial algebras for IR functors depends on the assumption of a large cardinal M which —very roughly speaking— is larger than any such set 'B' that can occur in the inductive definition of the code defining the IR functor.*

The content of this subsection continues in Section 2.2.4, in particular in Section 2.2.4.6 is recalled that the class of polynomial functors is closed under composition.

1.2.5 Monads

The notion of monad is a generalization of that of monoid known from algebra where it provides a conceptualization of composition. This is exactly the purpose for which monads will be important for us. We do not give an introduction to monads here which can be found in [13, §10] or [81, §VI] but only review the relation of monads to the binding operation that will be important for us.

Definition 1.2.5.1 (Monad, Kleisli extension). *A monad on a category C is given by an endofunctor M on C and transformations $\eta : id_C \to M$ called the* unit *and $\mu : M \circ M \to M$ called the* multiplication *of the monad, satisfying well known constraints.*

Every monad induces a function

$$\mathsf{kl} : Hom_C(D, M(E)) \to Hom_C(M(D), M(E))$$
$$m \mapsto \mu_{M(E)} \circ M(m) \circ \eta_D$$

called Kleisli extension*.*

Remark 1.2.5.2 (Extension system). *Based on Kleisli extension, [87] (following earlier work of [86]) gave an equivalent characterization of the notion of monad.*

Maps of the type of the Kleisli extension will be more important for us than monad structures themselves. In those cases of interest to us, one can uncurry kl:

Definition 1.2.5.3 (Bind). *If M is a monad on Set (or on Set_1), it induces the operation*

$$\mathsf{bind} : M(D) \to (D \to M(E)) \to M(E)$$
$$\mathsf{bind}\ X\ h = \mu((M(h))X)$$

called the bind operation *associated to M.*

The following example is instructive for regarding monads as a mechanism facilitating a composition operation.

Example 1.2.5.4 (Free-monoid monad (aka list monad)). *The functor $M : \mathsf{Set} \to \mathsf{Set}$ assigning to a set A, the set of finite (possibly empty) lists of elements of A as entries carries a monad structure where the component $\eta_A : A \to M(A)$ of the unit sends $a : A$ to the list of length one with a as the only entry, and the component of the multiplication $\mu_A : M(M(A)) \to A$ sends a list of lists of elements of A to the list obtained by removing inner brackets — e.g. $((a_1, a_2), (a_3, a_4))$ is sent to (a_1, a_2, a_3, a_4). The bind operation bind X f thus first substitutes every entry a of X by the list being the value $f(a)$ and then removes inner brackets to obtain a list with elements of the codomain of f as entries.*

1.3 Families

The objects defined by inductive-recursive definitions are families $(U : \mathsf{Set}, T : U \to D) : \mathsf{Fam}(D)$ (for $D : \mathsf{Set}_1$) where each code encoding an inductive-recursive definition decodes to a functor $\mathsf{Fam}(D) \to \mathsf{Fam}(E)$ (called *IR-functors*) between categories of families. The interest in families as such is explained by the fact that families of (opposite) sets, containers (aka polynomials), and inductive definitions are equivalent notions (in the sense explained in Section 2.2.4.5) which are of great importance for dependent theory and its semantics.

In this section we collect some material on families in general; a separate Section 1.2.4 on families of $(U, T) : \mathsf{Fam}(\mathsf{Set})$ of sets which are also called *containers* already preceded this section. Since we are interested in families mainly in connection to IR, we will make some anticipatory remarks about this connection which we will of course not assume in any formal sense later on.

Definition 1.3.0.1 (Fam). *We define a functor*

$$\begin{aligned}
&\mathsf{Fam} : \mathsf{Set}_1 \to \mathsf{Set}_1 \\
&D \mapsto \Sigma_{S:\mathsf{Set}}(S \to D) \\
&f \mapsto ((S, h) \mapsto (S, f \circ h))
\end{aligned}$$

Terminology 1.3.0.2 ($\mathsf{Fam}(D)$). *The definition of the functor* Fam *of the previous definition makes sense more generally as a functor $\mathsf{Fam} : \mathsf{Cat} \to \mathsf{Cat}$. Of interest to us is here that in this case $\mathsf{Fam}(D)$ can carry two different category structures:*

1. *For $D : \mathsf{Set}_1$ a discrete category, we call a morphism $h : (U, T) \to (U', T')$ defined as a pair $(h_0 : U \to U', h_1 : h_1 : (u : U) \to T(u) = T'(h(u)))$, a* cartesian morphism, *and the category with object set $\mathsf{Fam}(D)$ and cartesian morphisms as morphisms we call the* cartesian fragment of $\mathsf{Fam}(D)$. *Since we will mainly be interested in this category of families since in particular the notions of IR we discuss in this thesis induce functors only on this kind categories of families , we usually refer to this category just as "category of families".*
2. *More generally[3], if D is any category (such as $D = \mathsf{Set}$), we can take morphisms $(U, T) \to (U', T')$ to be pairs $(h_0 : U \to U', h_1 : (u : U) \to T(u) \to T'(h(u)))$.*

[3]For a version of induction recursion defining an action on non-cartesian morphisms, see [49].

For $\mathsf{Fam}(D)$ *as a fibration (as defined in [117]) over* D*, the more restricted definition of morphisms in the previous definition are the* split cartesian morphisms*, and if* $T = T' \circ h$ *is only a propositional equality, the morphisms* h *is a* cartesian morphism*.*

Remark 1.3.0.3. *As we mentioned in Definition 1.2.4.1, we have* $\mathsf{Cont} = \mathsf{Fam}(\mathsf{Set}^{op})$ *for the more general class of (not-necessarily cartesian) morphisms defined in Terminology 1.3.0.2 .2.*

Cartesian morphisms between families are closely related to a partial order on $\mathsf{Fam}(D)$. That $\mathsf{Fam}(D)$ carries such a partial order is a similarity of $\mathsf{Fam}(D)$ to Set that obtains for all $D : \mathsf{Set}_1$. This partial order is used for the construction of initial algebras for endofunctors of $\mathsf{Fam}(D)$, see e.g. Section 6.4.

Remark 1.3.0.4 (Partial order on $\mathsf{Fam}(D)$ and cartesian morphisms between families). *For every* $D : \mathsf{Set}_1$ *the large set* $\mathsf{Fam}D$ *is partially ordered where* $(U', T') \leq (U, T)$ *iff* $U' \subseteq U$ *and* $T \restriction U' = T'$ *is the restriction of* T *to* U'*.*

Obviously $(U', T') \leq (U, T)$ *induces a split cartesian morphisms. Conversely, if* $h : (U', T') \to (U, T)$ *is a split cartesian morphism, we have* $(im(h_0), \hat{T}) \leq (U, T)$ *where* $im(h_0) = \Sigma_{x:U}\Sigma_{u:U'}(hu = x)$ *and* $\hat{T}(x, u, p) := Tx$*.*

Induction-recursion in the style of Dybjer-Setzer Chapter 3 defines an action on cartesian morphisms between families and only on these and the functors arising as semantics of Dybjer-Setzer's IR are in particular monotonous functions with respect to the partial order on families. We will comment on some problems arising from the limitations of notions of IR acting only on cartesian morphisms in Remark 3.2.1.17.

Remark 1.3.0.5 (Properties of Fam and of $\mathsf{Fam}(D)$). Fam *has several interesting properties which are important for the theory of induction-recursion and for the semantics of type theory more generally:*

$\mathsf{Fam}(\mathsf{Set}^{op})$ *is equivalent to the category of polynomial functors (see Section 1.2.4) [44] and thus to the category of inductive definitions.*

- *As an operation* Fam *has itself the form of a (large) polynomial functor defined by the container* $(\mathsf{Set}, \mathsf{id})$ *(see also Section 1.2.4). In particular* $\mathsf{Fam} : X \mapsto \Sigma_{U:\mathsf{Set}}(U \to X)$*, as a functor arises as a sum of representable functors, is an example of a* familially representable functor *[128][21].*
- *More generally than the previous point,* Fam *is (as a 2-functor) the paradigmatic example of a familial 2-functor [128, §5.9], and (as a 1-functor) it is a parametric right adjoint (see loc. cit.). The latter implies that unit and multiplication of the* Fam *monad are cartesian natural transformations, i.e. the naturality squares are pullback squares. (We will not need to use this fact, though.)*
- $\mathsf{Fam}(D)$ *is a (Grothendieck) fibration and thus provides a basic model for type dependency [66].*

Lemma 1.3.0.6 (Monad structure on Fam). *1. For* $D : \mathsf{Set}_1$*, the* D*-component of the unit of the monad (see Section 1.2.5)* Fam *is given by* $\eta^{\mathsf{Fam}}_D : D \to \mathsf{Fam}\ D$*,* $d \mapsto (1,\ * \mapsto d)$*.*

2. *For $D : \mathsf{Set}_1$, the D-component of the multiplication of the monad* Fam *is given by* $\mu^{\mathsf{Fam}}_D : \mathsf{Fam}(\mathsf{Fam}(D)) \to \mathsf{Fam}(D)$, $(U, T) \mapsto (\Sigma_{u:U}\, \mathsf{proj}_1\, Tu),\ (u, k) \mapsto (\mathsf{proj}_2(Tu))\,(k))$.
3. η^{Fam} *and* μ^{Fam} *satisfy the triangle equalities.*
4. *Like every monad on* Set_1, $\mathsf{Fam}\ D$ *admits a bind operation (see Definition 1.2.5.3) given by* $\gg\!= := \mu^{\mathsf{Fam}} \circ \mathsf{Fam}$.

In particular composition of inductive definitions can be expressed via this monad structure (see Corollary 2.2.4.13 and Section 2.2.4.6).

Remark 1.3.0.7 (Colimits and limits in $\mathsf{Fam}(D)$). *For every $D : \mathsf{Set}_1$, the category $\mathsf{Fam}D$ is the free set-indexed-coproduct cocompletion of D see [117, §6][4]. In particular binary coproducts exist in $\mathsf{Fam}(D)$. The existence of set-indexed coproducts in $\mathsf{Fam}(D)$ will facilitate the definition of IR-functors as sums of already defined IR functors.*

For an arbitrary $D : \mathsf{Set}_1$, the category $\mathsf{Fam}(D)$ is not well endowed with limits. It has no terminal object or other binary products, and no exponential objects.

The situation is different if $D : \mathsf{Set}$ is small since then $\mathsf{Fam}(D) \simeq \mathsf{Set}/D$ is as a slice category of a topos itself a topos and as such has all limits. From the viewpoint of induction-recursion this special case is however less interesting since it has been shown in [58] that so-called small induction-recursion *is equivalent to (just) induction.*

$\mathsf{Fam}(D)$ is also more amenable if $D = \mathsf{Set}$ in which case $\mathsf{Fam}D \simeq \mathsf{Set}^{\{0\to 1\}}$ is equivalent to the arrow category of Set *which is a presheaf topos called* Sierpinski topos *[69, A2.1.12, B3.2.11]. This topos (like the above mentioned Set/D for small D) is moreover a* cohesive topos *[110] and as such more similar to* Set *than just an arbitrary topos.*

Remark 1.3.0.8 (The comma category i/Set_1). *The problems posed by the lack of limits in the categories $\mathsf{Fam}(D)$ for arbitrary $D : \mathsf{Set}_1$ can in some situations answered by instead considering the category $\Sigma_{D:\mathsf{Set}_1}\mathsf{Fam}(D)$ (with morphisms defined in the obvious way). This category is equivalent to the comma category i/Set_1 where $i : \mathsf{Set} \to \mathsf{Set}_1$ is the cumulativity map. Like every comma category of a topos, this category is itself a topos and has all limits; in particular the operation*

$$\begin{aligned} \_ \longrightarrow_{\mathsf{Fam}} \_ &: (S : \mathsf{Set}) \to \mathsf{Fam}\ D \to \mathsf{Fam}\ (S \to D) \\ S \longrightarrow_{\mathsf{Fam}} (A, P) &= (S \to A, g \mapsto P \circ g) \end{aligned}$$

of powering (see Definition 1.2.1.1) families that can be used to define composition of IR codes (see Section 4.1), can be understood as taking place in this category.

[4] A dual discussion, i.e. of the set-indexed-product completion can be found in [21, above Proposition 2.2].

Chapter 2

Inductive Definitions and MLTT

Induction constitutes [1] constructive mathematics. Non-technically speaking, as a proof method it reasons from properties of parts to properties of the partitioned[2] whole, and is thus available only if the whole is structured appropriately. Martin-Löf Type Theory (MLTT) makes this interplay of an object that is structured in a 'positive' way by parts, and a formalism proving properties of this object precise as we will explain in Subsection 2.2.2.

We will start this section by a short overview on induction in general, followed by an introduction to MLTT focusing on induction and some comments on the semantics of inductive definitions in MLTT.

Induction-recursion (see, Chapter 3) which has induction as an important special case is formulated by a set of rules extending MLTT presented here.

2.1 A Brief History of Induction

2.1.1 Induction in general

Acerbi [5] reviews some of the scholarship on the history of induction ("complete induction" in his diction): it is commonly understood that the first *conscious* use of induction is found in Blaise Pascal's "Traité du triangle arithmétique", but that the use of inductive arguments (broadly construed) in mathematics can be found much earlier — for example in Arab mathematics, and possibly in Euclid. Moreover loc. cit. claims to have found evidence for an inductive argument in the Platonic dialog "Parmenides".

[1]It does this together with its dual notion of coinduction with which we shall not be concerned much in this thesis (except for a few remarks in Subparagraph 2.2.3.3.3.1 Paragraph 2.2.3.3.6, and Paragraph 2.2.3.3.7). Even though coinduction was not yet discussed by Brouwer (see, e.g. [20]) who is a main reference point in the historical development of intuitionism and constructivism, almost all constructivists today agree that coinduction is a constructive notion as well.

[2]We do not assume here that partitions are disjoint.

2.1.2 Natural Numbers

In a more formal, axiomatic context, the natural numbers were characterized as an object purely in terms of induction probably for the first times in Hermann Grassmann's textbook on arithmetic [57]; this was followed by a definition of Richard Dedekind [30] (who also proved that any two models of his[3] natural numbers are isomorphic) and a definition equivalent to Dedekind's by Giuseppe Peano [102]. Peano's axioms contain also an axiom (the induction axiom) asserting that proofs by induction on natural numbers are possible.

2.1.3 In Constructive Mathematics following Brouwer

The idea to establish more general mathematical objects themselves — as opposed to only properties of them — by induction is an artefact of *intuitionism*, the kind of mathematics introduced by L. E. J. Brouwer.

The philosophical- and ideological[4]- background of Brouwer is largely informed by Immanuel Kant who formulated his constructive ideas maybe most concisely in his opus posthumum "He who would know the world must first manufacture it in his own self, indeed." ([72][p.240], see also [56][p.2] for more on Kant's ideas of synthesis). Also Brouwer's terminology of *intuition* as constitutive for the kind of mathematical activity conceived by him (see his notion of "creative subject" [121][§16]) originates with Kant [72], see also [43] for more on Brouwer's intuitionism and conception of constructive mathematics.

More precisely, an inductive object (such as an inductive type) is an object that allows to prove (all of) its properties by induction on it and thus the shift of terminology from inductive proofs to inductive objects was effected by the desire to do inductive proofs of properties of other objects than the linearly ordered natural numbers.To make this possible, the object whose properties shall be proved inductively needs to be structured, i.e. ordered, in a way facilitating such proofs. The most important such structures for our purposes are *species* — analogues of classical sets [121][§4] which inform Martin-Löf's *wellorderings* [92].

2.2 Martin-Löf Type Theory

2.2.1 Intuitionistic Logic

We emphasized the close relation of induction and constructive mathematics. One proof method that distinguishes "classical" (i.e. non-constructive) mathematics from constructive mathematics is that of *proof by contradiction (reduction ad absurdum)* which

[3]The induction axiom of natural numbers can be formulated either in first-, or in second order logic. That any two models are isomorphic holds however only for the second-order variant.

[4]Another source of influence for Brouwer was his Protestant faith and the assorted emphasis of the subject as the centre of his constructivism.

uses the law of excluded middle (LEM) that Brouwer started to reject [96] from 1908 as at odds with his constructivist program since the idea that understanding an object means for him understanding it positively by constructing it whereas LEM can be used to prove the existence of objects negatively, i.e. by establishing the impossibility of their non-existence without giving any intrinsic description of the asserted object. Classical logic minus the law of excluded middle is called *intuitionistic logic*.

We do not attempt here to comment on the question to what extend other formal systems such as Hilbert-style ones used e.g. in the Principia Mathematica, which also devises a type theory, conceives the notion of type as an object structured in a way facilitating proofs of its properties by induction.

2.2.2 MLTT as a Natural Deduction Calculus

Martin-Löf type theory [88][92] is a type theory that incorporates an intuitionistic higher order logic. Martin-Löf [92][p.13] understands the types defined in his type theory as constructive sets or as sets about which one can reason constructively within the system.

Given Brouwer's scepticism regarding formalised mathematics based on notational systems as well as his separation of logic and mathematics it might surprise that Brouwer's ideas of intuitionistic mathematics survive today mainly in MLTT which is formulated as a natural deduction system and as such does *not* separate logic from the theory of types but instead implements the *propositions-as-types and proofs-as-terms paradigm*.

To treat objects and proofs of properties of these objects in one integrated formal system such as MLTT became possible by Gentzen's *natural deduction calculus*[47] a notation system for logical inferences. This is realised by giving *for each set/type* rules for its *formation* (defining constant symbols possibly binding terms of the premise of the respective rules), *introduction* (defining constant symbols which stand for generic terms of the types formed), *elimination* (allowing to define functions having the formed type as domain; this subsumes in particular propositions —as traditionally understood— which are functions with codomain the set with two elements ("booleans")), and a *computation rule* (also called equality rule, which relates the introduction and elimination rule; these equality rules are instances of reduction rules in natural deduction which express that subsequent application of an introduction and elimination rules can be simplified [105]) by discharging intermediary premises.

2.2.3 Elements of MLTT

2.2.3.1 The Judgements of MLTT

Each rule consists of finite set of *(hypothetical) judgements* (see [93] for an explanation of this terminology) which are called the *premises* and one (hypothetical) judgement called the *conclusion* of the rule. Judgements can have the following forms

Judgement	Intuitive meaning
Γ ctxt	Γ is a well-formed context
$\Gamma \vdash A$ type	A is a well-formed type in the context Γ
$\Gamma \vdash M : A$	the term M has type A in the context Γ
$\Gamma \vdash A = B$ type	A and B are equal types in the context Γ
$\Gamma \vdash M = N : A$	the terms M and N are equal at the type A in the context Γ

All judgments in this table except the first one are called *hypothetical judgments* where the context (here Γ) is the hypothesis. This means that all types and term defined in a context may depend on terms occurring in this context; e.g. in $\Gamma \vdash M : A$ both M, and A may depend on terms of Γ.

2.2.3.2 Why Dependent Types?

Since MLTT is supposed to be a "one-level system" (i.e a *polymorphic* type theory) treating a syntactical symbolism and its meaning both on the same object level, it necessarily incorporates quantifiers of first-order logic ($\forall$, and $\exists$) since it becomes necessary to be able to express "for all objects of what type" a proposition is in question to hold. This is resolved by the notion of a *dependent type* written $a : A \vdash B(a)$ type where B may depend on terms of A.

2.2.3.3 The Rules of MLTT

After the discussion of the general forms of judgements used to express type dependency, we come more specifically to the types of MLTT. There are different versions of dependent type theory presented by Martin-Löf in his papers [92] [88] but we refer here to the latter version. In the following paragraphs we will give the rules of MLTT that define instances of the judgement forms listed in Section 2.2.3.1.

2.2.3.3.1 Rules for Context Formation The rules for context formation, and for type formation apply simultaneously: a context can be regarded as an ordered list of types while a (dependent) type is defined in a context; this is an example for *induction-induction* [98][5]. Defining the type theory by induction implies in particular that all the defined constructs are wellfounded, e.g. contexts (subsequently defined) have finite length, and types are generated by finitely many applications of constructors. The base case of this induction-induction is the axiom (i.e. a rule having no premise) displayed to the left below stating that there is an *empty context*, and the second mentioned rule for context formation states that a context can be *extended* by a type in (this) context to form a uniquely determined new context.

$$\frac{}{1 \text{ ctxt}}\ \textsc{Emp} \qquad\qquad \frac{\Gamma \text{ ctxt} \qquad \Gamma \vdash A \text{ type}}{\Gamma, x : A \text{ ctxt}}\ \textsc{Ext}$$

[5]We do not introduce induction-induction formally in this thesis.

Notice that x in the conclusion of the context-extension rule on the right is a *generic term* (aka arbitrary term) and does not appear in the premise of this rule. It is therefore not possible to extend contexts by *specific terms* (i.e. terms specified in the premise of a rule), but only by *generic* ones (for more on this point see Subparagraph 2.2.3.3.7.1).

2.2.3.3.2 Rules for Equality: Judgmental Equality The rules for *judgemental equality* introduce a constant symbol '=' for an equivalence relation on (the union of all) terms and types that moreover satisfies the rules

$$\frac{\Gamma \vdash x : A}{\Gamma \vdash x = x : A}\ \textsc{eq-form} \qquad \frac{\Gamma \vdash A \ \mathsf{type}}{\Gamma \vdash A = A}\ \textsc{eq-Ty-form}$$

$$\frac{\Gamma \vdash x = y : A}{\Gamma \vdash y = x : A}\ \textsc{eq-sym} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A}\ \textsc{eq-Ty-sym}$$

$$\frac{\Gamma \vdash x = y : A \qquad \Gamma \vdash y = z : A}{\Gamma \vdash x = z : A}\ \textsc{eq-trans} \qquad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash B = C}{\Gamma \vdash A = C}\ \textsc{eq-Ty-trans}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A = B}{\Gamma \vdash a : B}\ \textsc{eq-Ty} \qquad \frac{\Gamma \vdash a = b : A \qquad \Gamma \vdash A = B}{\Gamma \vdash a = b : B}\ \textsc{eq-Tm-trans}$$

where in the second rule '$a = b : B$' should be read as "a and b are of type B and are related by =" and not as "$a = b$ is a term of type B". Moreover there are rules expressing that all the rules defining types are congruences with respect to the equivalence relation =.

There is a further notion of equality realized as a type which we explain in Paragraph 2.2.3.3.9.

⋆ ⋆ ⋆

2.2.3.3.3 The Structure of the Rules Defining Types The rules for type formation given in the following paragraphs[6] always consists of one *formation rule*, zero- or

[6]For negative types (see, Subparagraph 2.2.3.3.7.1) there can be more than one elimination rule. An example for this is the alternative exhibition of Σ-types as negative types (see Paragraph 2.2.3.3.6).

finitely many *introduction rules*, one *elimination rule*, and one *computation rule* for each introduction rule:

The formation rule for a types introduces a constant symbol, say K, for the type to be defined which may bind some arguments.

An introduction rule states that a constructor applied to its argument(s) (if any) yields a term of the type being defined, where the arguments are terms specified in the premise of the rule. It is not compulsory that a type has an introduction rule since the case of an empty type is not excluded.

The premise of an elimination rule[7] has three parts: firstly the specification of a *motif* $x : \mathsf{K} \vdash M(x)\ \mathsf{type}$ which is a type family dependent on the type being defined, secondly the judgment that M evaluated in a constructor is an inhabited type $x : C \vdash d : M(cx)$ (this is called the *inductive hypothesis*), and thirdly the provision of an arbitrary term of the type to be eliminated:

The premise of computation rule shares the first two premises with the elimination rule and states that the term in the evaluated motif —which asserts the inductive hypothesis— is *equal* (in the sense explained in Paragraph 2.2.3.3.2) to the eliminator applied to the constructor.

We usually suppress contexts that are shared by all premises and the conclusion in a rule, e.g. in place of

$$\frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma,\ x : A \vdash B\ \mathsf{type}}{\Gamma \vdash \mathsf{K}(A, B)\ \mathsf{type}}\ \textsf{K-}\textsc{form} \quad .$$

we write

$$\frac{A\ \mathsf{type} \qquad x : A \vdash B\ \mathsf{type}}{\mathsf{K}(A, B)\ \mathsf{type}}\ \textsf{K-}\textsc{form} \quad .$$

We also write $x_1,\ \ldots,\ x_n\ : A$ instead of a list of judgements $x_1 : A,\ \ldots,\ x_n : A$ containing distinct variables of the same type (this notation should not be confused with a context since the $x_1,\ \ldots, x_n$ are not assumed to be generic but can be specific, see Paragraph 2.2.3.3.1).

2.2.3.3.3.1 Positive-, and Negative Type definitions Generally an elimination rule is so called because it eliminates some assumptions in the premise of the rule. This can however happen in different ways. One can distinguish two kinds of elimination-, and assorted computation rules: the case of *positive type definitions* where the conclusion of the computation rule describes the result of applying an eliminator to a constructor,

[7]At least for positively defined types (see Subparagraph 2.2.3.3.3.1 for the distinction to negatively defined types).

and *negative type definitions* where the conclusion of the computation rule describes the result of applying a constructor to an eliminator. Positive type definitions can express inductive types (see Section 2.2.4) while negative type definitions can express coinductive definitions[8]. The reason why both terminologies —positive types and inductive types (respectively negative types and coinductive types)— are in use is that the definition of inductive types already assumes some (positive) type definitions. So, the positive / negative terminology is somewhat more fundamental.

We will give two examples for negative type definitions in Subparagraph 2.2.3.3.3.1 and Paragraph 2.2.3.3.6 but we will be mainly interested in positive types since the axiomatizations of IR we present in this thesis do not define negative types.

2.2.3.3.4 The Empty Type The empty type 0 has formation rule

$$\frac{}{0\ \mathsf{type}}\ \textsc{intro-0} \quad .$$

It has no introduction rule since it has no terms. The elimination rule

$$\frac{x : 0 \vdash M(x)\ \text{type} \qquad z : 0}{\mathsf{elim}(M, z) : M(z)}\ \textsc{elim-0}$$

is trivial in the sense that $\mathsf{elim}(M, z) : M(z)$ regardless what M is. This elimination rule is traditionally called *ex falso sequitur quodlibet.*

There is no computation rule for the empty type since it has no introduction rule.

2.2.3.3.5 The Unit Type The unit type 1 has the formation rule and introduction rules

$$\frac{}{1\ \mathsf{type}}\ \textsc{form-1} \qquad\qquad \frac{}{* : 1}\ \textsc{intro-1} \quad .$$

The elimination rule is

$$\frac{x : 1 \vdash M(x)\ \text{type} \qquad m : M(*) \qquad z : 1}{\mathsf{elim}(M, m, z) : M(z)}\ \textsc{elim-1}$$

and the computation rule is

[8] The categorical dual to inductive definitions in the algebra semantics of type theory which we do not discuss in this thesis.

$$\frac{x : 1 \vdash M(x) \text{ type} \qquad m : M(*)}{\mathsf{elim}(M, m, *) = m : M(*)} \text{ COMP-1} \quad .$$

2.2.3.3.6 Σ-Types (as Positive-, and as Negative Types A Σ-type has as terms pairs where the type of the second component may depend on the first component.

$$\frac{A \ \mathsf{type} \qquad x : A \vdash B \text{ type}}{\Sigma AB \ \ \mathsf{type}} \text{ FORM-}\Sigma$$

$$\frac{a : A \vdash B(a) \ \mathsf{type} \qquad a : A \vdash b : B(a)}{(a, b)_\Sigma : \Sigma AB} \text{ INTRO-}\Sigma$$

We will usually omit the subscript of $(\, , \,)_\Sigma$.

$$\frac{x : \Sigma AB \vdash M(x) \ \mathsf{type} \qquad x : A, y : B(x) \vdash m(x, y) : M((x, y)_\Sigma) \qquad z : \Sigma AB}{\mathsf{elim}(M, m, z) : M(z)} \text{ ELIM-}\Sigma$$

$$\frac{x : \Sigma AB \vdash M(x) \ \mathsf{type} \qquad x : A, y : B(x) \vdash m(x, y) : M((x, y)_\Sigma)}{\mathsf{elim}(M, m, (x, y)_\Sigma) = m(x, y) : M((x, y)_\Sigma)} \text{ COMP-}\Sigma$$

From these rules, the following two *projection rules* are derivable

$$\frac{q : \Sigma_{x:A} B(x)}{\mathsf{proj}_0(q) : A} \ \mathsf{proj}_0\text{-}\Sigma \qquad\qquad \frac{q : \Sigma_{x:A} B(x)}{\mathsf{proj}_1(q) : B(\mathsf{proj}_0(q))} \ \mathsf{proj}_1\text{-}\Sigma$$

(where we have not typed out the premises we already know — a practise that we will adopt frequently in the following).

Alternatively, one can define Σ-types negatively by assuming the formation-, and introduction rules, the two projection rules (as elimination rules) complemented by the computation rules

$$\frac{x : A \vdash y : B(x)}{\mathsf{proj}_0(x, y)_\Sigma = x} \ \mathsf{proj}_0\text{-}\textsc{Comp} \qquad\qquad \frac{x : A \vdash y : B(x)}{\mathsf{proj}_1(x, y)_\Sigma = y} \ \mathsf{proj}_1\text{-}\textsc{Comp} \quad .$$

With ELIM-Σ and COMP-Σ one can furthermore derive

$$\frac{q : \Sigma_{x:A} B(x)}{(\mathsf{proj}_0(q), \mathsf{proj}_1(q))_\Sigma = q} \text{ COMP'-}\Sigma \quad .$$

Notice that the elimination rule using a motif has in the left-hand-side of the equation in the conclusion of the computation rule an application of an eliminator to a constructor, whereas we have here in the computation rule for the elimination via projection an application of the constructor to the eliminators. The first mentioned style is called a presentation of the type a *positive type*, and the second one a presentation as a *negative type*; positive types are types definable by induction (see Section 2.2.4) whereas negative types can be defined by coinduction (for which we will only have a few passing remarks this thesis). For Σ types one can show (see e.g. [46]) that both presentations are equivalent but not all types can be presented positively and negatively.

2.2.3.3.7 Π-Types as Negative Types (Elimination via an Application Rule) While Σ types can equivalently be presented as positive and negative types, this is not possible in case of Π-types without modifying the whole type theory somewhat (see Subparagraph 2.2.3.3.7.1). We start therefor with the negative presentation which can be formulated in the version of MLTT as we have presented it so far.

The formation rule

$$\frac{A \ \mathsf{type} \qquad x : A \vdash B(x) \ \mathsf{type}}{\Pi_{x:A} B(x) \ \mathsf{type}} \text{ FORM-}\Pi$$

where we use the notation $A \to B$ in place of $\Pi_{x:A} B(x)$ if $B(x)$ does not depend on the choice of x, and the introduction rule for Π-types

$$\frac{x : A \vdash B(x) \ \mathsf{type} \qquad x : A \vdash f(x) : B(x)}{\lambda x.f(x) : \Pi_{x:A} B(x)} \text{ INTRO-}\Pi$$

share the same pattern with the other types of MLTT, whereas the elimination rule

$$\frac{f : \Pi_{x:A} B(x) \qquad a : A}{\mathsf{app}(f, a) : B(a)} \text{ APP-}\Pi$$

that is also called *application rule* is here not formulated via a motif $f : \Pi_{x:A} B(x) \vdash M(f)$ type. again, the application rule is still an elimination rule in the sense that it can be used to discharge premises of the introduction rule (here $a : A$); it is complemented by the computation rule

$$\frac{f : \Pi_{a:A} B(a) \qquad x : A}{\mathsf{app}(\lambda x.f(x)\ ,\ a) = f(a)}\ \textsc{comp-}\Pi \quad .$$

qualifying the whole definition as a *negative* one in the sense explained in Paragraph 2.2.3.3.6).

2.2.3.3.7.1 Π-Types as Positive Types (Elimination via a Motif Using Higher Order Hypothetical Judgements) This paragraph briefly comments on what modifications of MLTT are necessary to present Π-types positively. Since we will not use this presentation in the following sections, this paragraph is optional, but it serves to describe the scope of induction (as opposed to coinduction), and as such of induction-recursion.

The problem in defining the elimination rule using a motif for Π-types arises since we need to instantiate the motif in the premise of this elimination rule by an evaluated constructor, i.e. we would need to find a context in which $M(\lambda x.f(x))$ is defined. Examining the introduction rule, we see that we would need to transform $x : A \vdash f(x) : B(x)$ into a context which is however not possible: the only candidate context available in the polymorphic presentation would be "x : A , f(x) : B(x)" but this is not a context since $f(x)$ does not denote a *generic term* of $B(x)$ but a *specific* one; in comparison, the construction does works for Σ-types since in "x : A , y : B(x)", the term y is a generic term of $B(x)$. The problem is resolved (see [122]) by defining a new construct that can turn a hypothetical judgment like $x : A \vdash f(x) : B(x)$ into a context $[x : A]f(x) : (x : A)B(x)$; inference rules using this construct are called *higher order inference rules* (see [111]). With this we obtain the elimination rule

$$\frac{\begin{array}{c} g : \Pi_{a:A} B(a) \vdash M(g) \text{ type} \\ [x : A]f(x) : (x : A)B(x) \vdash d(f) : M(\lambda f) \qquad m : \Pi_{a:A} B(a) \end{array}}{\mathsf{elim}(d, m) : M(m)}\ \textsc{elim-}\Pi \quad .$$

We shall not use this construct in the following.

Remark 2.2.3.1 (Recursion, non-dependent elimination). *[123, p.30] uses the alternative terminology of* dependent elimination *and* non-dependent elimination *for "induction" and "recursion" respectively. The explanation for this wording is: let T be a type defined by induction, then:*

1. *The non-dependent elimination principle for T is defining a function $f : T \to M$ by recursion.*
2. *If $t : T \vdash M(t)$ is a dependent type, "dependent elimination" means to define a term $f : \Pi_{t:T} M(t)$.*

Before we come to inductive definitions at the end of this section about which we shall say a few more words because of the central role they play in this thesis, we explicate two sets of rules that each define not only an isolated element of the type theory but whose definition involve all other types defined in the type theory: universes and rules for equality.

2.2.3.3.8 Universes Universes generally are intended to formalize the idea of a totality of objects. Naïve imagination tends to be somewhat rash in positing totalities of "all" objects of some kind —such as all sets or all types— which repeatedly resulted in inconsistent systems: Russel's paradox, consisting in the assertion (and simultaneous negation) of the statement that the "set" $\{x \mid \neg(x \in x)\}$ defined by unrestricted comprehension contains itself, led to the development of ramified type theory in [108]. Similarly, the first version of Martin-Löf type theory [88] was inconsistent since it entailed Girard's paradox since the judgment $\mathsf{type} : \mathsf{type}$ is derivable in it.

More specifically, $\mathsf{type} : \mathsf{type}$ is a strongly impredicative definition (see [100] for a discussion of this definition) and as such at odds with most conceptions of constructive mathematics and induction. Therefore it does make sense to consider a universe, such as Set, either within some other sort like $\mathsf{Set}\ \mathsf{type}$ (read: "Set is a type" as opposed to "Set is of type type") such that a noncircular positive definition within type becomes possible, or (additionally) to define a hierarchy $\mathsf{Set} : \mathsf{Set}_1$ (where $\mathsf{Set}\ \mathsf{type}$ as well as $\mathsf{Set}_1\ \mathsf{type}$) were both Set as well as Set_1 are defined within type together with additional rules establishing the relation $\mathsf{Set} : \mathsf{Set}_1$.

2.2.3.3.8.1 Russel Universes The style of universes admitting the direct typing relation $\mathsf{Set} : \mathsf{Set}_1$ we have just explained is called *Russel universe.* The rules for a Russel universe U are:

$$\frac{}{\mathsf{U}\ \mathsf{type}}\ \textsc{form-}\mathsf{U}$$

and for every type under which U shall be closed, we need an introduction rule: for not defining the empty type, we can require that it e.g. contains the type of natural numbers by

$$\frac{}{\mathbb{N} : \mathsf{U}}\ \textsc{intro-}\mathsf{U} \quad .$$

For closure under a type constructor K binding a family of types (such as Σ or Π)

$$\frac{A : \mathsf{U} \qquad x : A \vdash B : \mathsf{U}}{\mathsf{K}(A, B) : \mathsf{U}}\ \mathsf{K}\text{-}\mathsf{U} \quad .$$

Moreover, there is a cumulativity rule

$$\frac{A : \mathsf{U}}{A\ \mathsf{type}}\ \textsc{cum-}\mathsf{U} \qquad .$$

In this thesis we will assume a hierarchy of two Russel universes Set and Set_1, i.e. we have now two versions of U which we denote by Set, and Set_1. To negotiate the relation between these universes, there are two further rules:

$$\frac{}{\mathsf{Set} : \mathsf{Set}_1}\ \textsc{intro-}\mathsf{Set}_1 \qquad\qquad \frac{A : \mathsf{Set}}{A : \mathsf{Set}_1}\ \textsc{cum-}\mathsf{SetSet}_1 \qquad .$$

2.2.3.3.8.2 Tarski Universes The style of universes we are interested in this thesis is however of a different kind. *Tarski universes* are not single set-like objects (of whatever size) but certain families (see Section 1.3). As such, and assuming that they are definable by an inductive- or constructive process, they are examples of inductive-recursive definitions; in fact they are even the motivating example of inductive-recursive definitions and were already part of Martin-Löf type theory as presented in [92]. The reason why one considers Tarski universes instead of the seemingly simpler Russel ones are a number of type theoretical disadvantages —e.g. with respect to subtyping and canonicity— of the latter; nevertheless it is true that they are easier to use for other purposes, see [84] for a discussion.

A Tarski universe (U, T) (in Set) is understood as having as first component $\mathsf{U} : \mathsf{Set}$ a set of "codes" [9] representing elements contained by the universe, and a decoding function[10] $\mathsf{T} : \mathsf{U} \to \mathsf{Set}$ sending a term $u : \mathsf{U}$ to the actual set that u encodes.

Formally, the definition of a Tarski universe proceeds (after we have simultaneously introduced the constant symbols U, and T) by specifying the operations under which one wants the universe to be closed. These are the premises of n introduction rules $C_1 \dots C_n : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Set}$. As abbreviation, we define the set of *internal families in* (U, T) by $\mathsf{IFam}(\mathsf{U}, \mathsf{T}) := \Sigma_{u:\mathsf{U}}(\mathsf{T}(u) \to \mathsf{U})$, then for each $1 \leq k \leq n$ we give the introduction rule

$$\frac{(u, t) : \mathsf{IFam}(\mathsf{U}, \mathsf{T})}{C_k^I(u, t) : \mathsf{U}}\ \textsc{intro-}\mathsf{U} \qquad .$$

and a rule

$$\frac{C_k : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Set} \qquad (u, t) : \mathsf{IFam}(\mathsf{U}, \mathsf{T})}{\mathsf{T}(C_k^I(u, t)) = C_k(\mathsf{T}(u), \mathsf{T} \circ t) : \mathsf{Set}}\ \textsc{rec-}\mathsf{T} \qquad .$$

[9]Not to be confused with the codes that will define inductive-recursive definitions later on.
[10]This function T is often denoted by El in the literature.

stating that T is defined by recursion on U.Of interest in this definition is the occurrence of T: in the constructor $C^I_k : (u : U)(T(u) \to U) \to U$, the expression $T(u)$ occurs in negative expression. Since in the definition of the pair (U, T) only U appears in only positive position, this definition —and inductive-recursive definitions generally— are called *half positive definitions*; we will return to this notion in Chapter 3 Chapter 5Chapter 6. [89] and [92] also considers a hierarchy of universes $(\mathsf{U}_i, \mathsf{T}_i)$ for $i \in \mathbb{N}$ with the two new (i.e. additional to the rules stating that the $(\mathsf{U}_i, \mathsf{T}_i)$ are universes) rules stating that each U_{i+1} contains a code for U_i, i.e.

$$\frac{}{u_i : \mathsf{U}_{i+1}} \qquad\qquad \frac{}{\mathsf{T}_{i+1}(u_i) = \mathsf{U}_i}$$

and rules stating that the hierarchy thus obtained is cumulative in the sense that

$$\frac{a : \mathsf{U}_i}{k_i(a) : \mathsf{U}_{i+1}} \qquad\qquad \frac{a : \mathsf{U}_i}{\mathsf{T}_{i+1}(k_i(a)) = \mathsf{T}_i(a)}$$

Which we can summarize by

$$\begin{array}{ll} \mathsf{U} : (n : \mathbb{N})) \to \mathsf{Set} & \mathsf{T} : (n : \mathbb{N}) \to U(n) \to \mathsf{Set} \\ u_n : \mathsf{U}(n) & \mathsf{T}\ n\ u_n = \mathsf{U}(n) \\ k : (n : \mathbb{N}) \to \mathsf{U}(n) \to \mathsf{U}(n+1) & \mathsf{T}(n+1)(k\ n\ u_n) = \mathsf{T}\ n\ u_n \end{array}$$

Martin-Löf does not give elimination rules for Tarski universes ([92][88][90, p.182][11]). The reasons for this were both, philosophical and technical: Martin-Löf liked to think of MLTT as an open system that always can accommodate new type constructors while the elimination rules must of course account for all constructors under which the universe is supposed to be closed. On the technical side, a rule in natural deduction calculus must not contain transfinitely many premises which implies that in case one wants to formulate closure under transfinitely many constructors, or a transfinite hierarchy of universes, one must package the transfinitely many premises into a finite number by, e.g, internally indexing the required sequences; this has been done by other authors (see, [101]). The elimination rule (see e.g. [64, §2.1.6]) for a single universe closed under the set operation $C : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Set}$ is given by:

$$\frac{M : \mathsf{U} \to \mathsf{Set}_1 \qquad (u_0, t_0) : \mathsf{IFam}(\mathsf{U}, \mathsf{T}) \vdash k : M(C^I(u_0, t_0)) \qquad u : \mathsf{U}}{\mathsf{elim}(M, (u_0, t_0), u) : M(u)}\ \text{UTELIM} \quad .$$

We close this paragraph with a remark on a motivation of universes relevant to induction-recursion and another remark on the use of universes in constructive set theory.

[11]In the last mentioned reference the rules for universe cumulativity are called "elimination rules".

Remark 2.2.3.2 (Reflection Principles). *Universes are (or are closely related to)* reflection principles*[79]. The latter were invented by Gödel in context of ZFC set theory as a means to reduce incompleteness. The name "reflection principle" reflects the intention that the totality of all sets (which is inconsistent) should contain a sets itself satisfying the axioms of ZFC: Gödel believed that an axiom asserting the existence of such a set would be justified*[12] *by the fact that we believed the would-be set of all ZFC sets to satisfy this axiom in first place.*

This idea of reflecting structure of a containing totality into a contained object is also active in the definition of the system DS' *Section 3.3 for induction-recursion.*

2.2.3.3.9 Rules for Equality: Identity Type

In addition to judgmental equality (see Paragraph 2.2.3.3.2), there is one more notion of identity as a type (family). The so-called *(intuitionistic) identity type* $x\ y\ : A \vdash \mathsf{Id}_A(x,y)$ type of a type $A : \mathsf{type}$ is actually a type family which has as introduction rule

$$\frac{x : A}{\mathsf{refl}_x : \mathsf{Id}_A(x,x)}\ \textsc{intro-Id-A}$$

which can semantically be regarded as the least equivalence relation on A. We sometimes omit the subscript and write refl in place of refl_x (in particular we do so in case we apply the rules in Paragraph 2.2.3.3.12).The assorted elimination-, and computation rules are

$$\frac{\begin{array}{c} x,\ y : A,\ p : \mathsf{Id}_A(x,y) \vdash C(p)\ \mathsf{type} \\ z : A \vdash c(z) : C(\mathsf{refl}_z) \qquad a,\ b : A \qquad q : \mathsf{Id}_A(a,b) \end{array}}{\mathsf{elim}_{\mathsf{Id}_A}(x,y,p,C(p),z,c(z),a,b,q) : C(q)}\ \textsc{elim-Id-A}$$

$$\frac{x,\ y : A,\ p : \mathsf{Id}_A(x,y) \vdash C(p)\ \mathsf{type} \qquad z : A \vdash c(z) : C(\mathsf{refl}_z) \qquad a : A}{\mathsf{elim}_{\mathsf{Id}_A}(x,y,p,C(p),z,c(z),a,a,\mathsf{refl}_a) = c(a) : C(\mathsf{refl}_a)}\ \textsc{comp-Id-A} \quad .$$

In case there is a $p : \mathsf{Id}_A(x,y)$, then x and y are also called to be *propositionally equal.* We will also use the notation $x = y$ for propositional equality and indicate whether we mean propositional equality of judgemental equality explained in Paragraph 2.2.3.3.2.

2.2.3.3.10 Reasoning About Propositional Equalities

2.2.3.3.10.1 Dependent Pattern Matching

For defining a function on a dependent type, it is desirable to be able to specify the function only on constructors of the type as opposed to using the elimination rule. Unfortunately this is not in general possible without the assumption of further axioms (such as UIP, see Paragraph 2.2.3.3.12, or Streicher's

[12]This justification strategy is called "intrinsic justification".

axiom K, see [55] and [118]). Generally, assuming pattern matching for all dependent types implies a form of extensionality, see [29]. In some cases it is however possible to define functions by pattern matching without assuming further axioms (see [26]), and we will freely do so subsequently. One example of this are certain functions for reasoning about propositional equalities which we will define in the following paragraph.

2.2.3.3.10.2 Id as an Equivalence Relation, Substition, Congruence For the purpose of reasoning with propositional equalities (which we denote here by =) we shall use the following functions. Since we want to give this definitions for small-, and for large sets, we let a, b, c be either 0 or 1.

The introduction rule of the identity type says that the propositional-identity relation is reflexive. Moreover, it is an equivalence relation since we have:

Transitivity:

$$\begin{aligned}&\mathsf{trans} : \{A : \mathsf{Set}_a\}\{x\ y\ z : A\} \to x = y \to y = z \to x = z\\&\mathsf{trans\ refl\ refl} = \mathsf{refl}\ .\end{aligned}$$

Symmetry:

$$\begin{aligned}&\mathsf{sym} : \{A : \mathsf{Set}_a\}\{x\ y\ : A\} \to (p : x = y) \to (y = x)\\&\mathsf{sym\ refl} = \mathsf{refl}\ .\end{aligned}$$

Every function is a congruence with respect to the equivalence relation of propositional equality:

$$\begin{aligned}&\mathsf{cong} : \{A : \mathsf{Set}_a\} \to \{B : \mathsf{Set}_b\} \to\\&\qquad (f : A \to B) \to \{x\ y : A\} \to x = y \to f x = f y\\&\mathsf{cong}\ f\ \mathsf{refl} = \mathsf{refl}\ .\end{aligned}$$

We have a *substitution* function:

$$\begin{aligned}&\mathsf{subst} : \{A\ :\ \mathsf{Set}_a\} \to (B\ :\ A \to \mathsf{Set}_b) \to\\&\qquad \{x\ y : A\} \to x = y \to B x \to B y\\&\mathsf{subst}\ P\ \mathsf{refl}\ p = p\ .\end{aligned}$$

There are many more such functions that we use in the Agda files but elide here.

2.2.3.3.11 Function Extensionality The rule for *function extensionality* states that functions are propositionally equal (see Paragraph 2.2.3.3.9) if they send propositionally equal arguments to propositionally equal values:

$$\frac{p : \Pi_{x:A}\mathsf{Id}_{B(x)}(f(x), g(x))}{\mathsf{ext}(f, g, p) : \mathsf{Id}_{\Pi_{x:A}B(x)}(f, g)}\ \textsc{ext-}\Pi \quad .$$

Notice that some of the Agda formalizations we rely on make use of funcion extensionality. This is the case for most proofs concerning preservation of semantics in Chapter 5 and Chapter 6.

2.2.3.3.12 Rules for Extensional Equality The rules for IR (see Chapter 3) extending those for MLTT require some rules for *extensionality* [40, p.6]

$$\frac{A\ \mathsf{type} \qquad x : A \qquad y : A \qquad p : \mathsf{Id}_A(x, y)}{x = y : A}\ \textsc{eq-reflection}$$

$$\frac{A\ \mathsf{type} \qquad x : A \qquad y : A \qquad p : \mathsf{Id}_A(x, y)}{p = \mathsf{refl} : \mathsf{Id}_A(x, y)}\ \textsc{uip} \quad .$$

Lemma 2.2.3.3 (Extensionality implies function extensionality). *The previous two rules imply that the types* $\mathsf{id}_{\Pi_{x:A}B(x)}(f, g)$ *and* $\Pi_{x:A}\mathsf{Id}_{B(x)}(f(x), g(x))$ *are equivalent (see [40, p.6], and [123] for a definition of equivalence of types).*

Some of the Agda formalizations we rely on make use of UIP. We believe that this is just a choice of convenience and not of necessity and that with (considerable) more work it is possible without this assumption but we have not checked this.

2.2.4 Inductive Types

2.2.4.1 Introduction: What Exactly is now an Inductive Definition?

We said that MLTT is the appropriate theory to pursue constructive mathematics since its types are inductively[13] defined. Then we proceeded to write down a list of type definitions the choice of whose members is so far not justified by anything but their semblance to popular standard operations of set formation. Informal mathematics would proceed from here by producing further definitions in prose style using types from the list we have given. Formal mathematics requires the production of further definitions however to follow the same schema as that for the standard operations above, i.e. to consist of a formation-, introduction-, elimination-, and elimination rule. This raises the questions 1) which types can we possibly define 2) in which sense are the standard examples above not only standard but moreover canonical or universal, 3) can we formally define how a rule defining a type must look like?

All these questions have satisfying answers that can be found in the well developed literature on the topic. There are at least three possible ways to axiomatize the notion of inductive definition:

1. Martin-Löf [89] introduced a type of *wellorderings* —today usually called a W-type— formalizing constructively the notion of a wellfounded set. The axiom of foundation of ZF states that the element relation for sets is acyclic and loc.cit. implements this in form of trees which are exactly acyclic connected graphs. We will see in Section 2.2.4.5 how this answers 2) and 3) above.
2. On the other hand, one can explicitly write down a sequence of constructors capturing an intuitive notion of inductive definition that prima facie resembles rather a definition of an ordered sequence of sets of trees. This answers 1) above.
3. On the third hand, one can define an inductive type as being the initial algebra —if it exists— for an endofunctor where one is left with the task of specifying a class of endofunctors that really do have initial algebras. This essentially takes the standard operations as canonical-by-definition and as such addresses 2) and 3).

In the subsequent subsections we will discuss the three equivalent conceptions of inductive definitions. The questions 1)2)3) are for us not of historical interest but are mutatis mutandis relevant for axiomatizations of inductive-recursive definitions as well where they have not yet found as satisfying answers as in the simpler case of inductive types.

2.2.4.2 W-types

Like the types in the preceding subsection, W-types were introduced (under the name *wellfounded types*) in [88]. W types are often also called *tree types* since the introduction rule can be read as a schema how to construct a tree inductively: for a node $a : A$, the set $B(a)$ is regarded as the set of possible ingoing edges into the node a and the constructor sup $a\ t$ 'grafts' on every branch $x : B(a)$ of the tree under construction the tree $t(x)$ to

[13]If we bracket the question about coinduction for now.

obtain a new tree. $\mathsf{W}(A, B)$ is then the set of all possible trees with A as set of nodes and the 'branching signature' B that can be obtained in this way.

$$\frac{x : A \vdash B \text{ type}}{\mathsf{W}AB \ \mathsf{type}} \ \textsc{form-}\mathsf{W}$$

$$\frac{x : A \vdash B \text{ type} \qquad x : A \vdash t : B \to \mathsf{W}AB}{\text{sup } \ a\ t : \mathsf{W}AB} \ \textsc{intro-}\mathsf{W}$$

$$\frac{\begin{array}{c} w : \mathsf{W}AB \vdash M(w) \text{ type} \\ a : A,\ t : B \to \mathsf{W}AB \vdash m : M(\text{sup } \ a\ t) \qquad x : \mathsf{W}AB \end{array}}{\mathsf{elim}(M, m, x) : M(x)} \ \textsc{elim-}\mathsf{W}$$

$$\frac{w : \mathsf{W}AB \vdash M(w) \text{ type} \qquad a : A, t : B \to \mathsf{W}AB \vdash m : M(\text{sup } \ a\ t)}{\mathsf{elim}(M, m, \text{sup } \ a\ t) = m : \mathsf{W}AB} \ \textsc{comp-}\mathsf{W}$$

W types can for instance be used to formulate the well-known n-th number classes.

Example 2.2.4.1 (Constructive number classes and ordinal notation systems). *One can take the cardinality of the set $\mathbb{N}$ of natural numbers which is usually denoted by $\aleph_0$ —or ω_0 if it is meant as an ordinal— as basis of the "induction"*

$$\begin{aligned} \aleph_{\alpha+1} &= \omega_{\alpha+1} = \textit{least cardinal greater than } \aleph_\alpha \\ \aleph_\alpha &= \omega_\alpha = \sup\{\beta \mid \beta < \alpha\} \ . \end{aligned}$$

*where ω_α is called the α-*th number class *(see, [68, p.30]). We have put the word 'induction' in quotation marks since it is prima facie not clear that this definition is inductive in the sense of constructive mathematics; e.g. determining the supremum of an infinite set is not in general possible by induction. The underlying set of the* second number class *ω_1 can for example be regarded as the set of all countable ordinals (see also [42, p.9]).* Finiteness *of members of $\mathbb{N}$ can be defined by supplying a system of* notations *for them. For $\mathbb{N}$ this is accomplished by representing an element by the constructors*[14] zero *and* suc*. Likewise, we can reduce the problem of finding a system of notations for the elements of the ordinals ω_α to exhibiting the definitions of the the latter as an inductive definition. The second number class ω_1 for example arises as initial algebra of the functor*

[14]Notice that writing a (natural) number in a certain basis is a different problem —namely that of finding a *convenient* system of notation— while we are here only interested in finding any notation.

$$F(X) \coloneqq 1 + X + (\mathbb{N} \to X)) \ .$$

The second number class has been the subject of study of Church [25] and the encoding of ordinals in the style of loc. cit. is sometimes called church encoding.

More generally, the n-th number classes $\mathcal{O}\ n$ can be obtained as the sequence of iterated W*-types*

$$\begin{aligned} \mathcal{O} \ &: \ \mathbb{N} \ \to \ \mathsf{Set} \\ \mathcal{O}\ n \ &= \ W\ (Fin\ n)\ (T_{\mathcal{O}}\ n) \end{aligned}$$

where

$$\begin{aligned} &T_{\mathcal{O}} \ : \ (n \ : \ \mathbb{N}) \ \to \ Fin\ n \ \to \ \mathsf{Set} \\ &T_{\mathcal{O}}\ zero\ n \ = \ \emptyset \\ &T_{\mathcal{O}}\ (suc\ n)\ zero \quad = \ W\ (Fin\ n)\ (T_{\mathcal{O}}\ n) \\ &T_{\mathcal{O}}\ (suc\ n)\ (suc\ m) \ = \ T_{\mathcal{O}}\ n\ m \ . \end{aligned}$$

In particular $\mathcal{O}\ 1 \ = \ 1$, $\mathcal{O}\ 2 \ = \ \mathbb{N}$, $\mathcal{O}\ 3 \ =$ Kleene's second number class [74].

Ordinal notation systems *are inductively defined systems of denotations for (certain) ordinals [74]. Ordinals for which there is a notation are also called* recursive ordinals *or* computable ordinals *where 'recursive' and 'computable' means here to be computable by an algorithm. Since a notation for an ordinal is a finite word over a finite alphabet (for example the alphabet $\{zero,\ suc\}$, (well-formed) words over which are natural numbers — or rather numerals*[15]*), the set of all ordinal notations.*

Ordinal notation systems —for which the above defined number classes are examples— play an important role for determining (lower bounds of) the proof-theoretical strength of theories (see Section 3.1.1) in that the theory in question is used to define an ordinal notation system attached to this theory. In other words, the ordinal notations are here those ordinals that are computable by the theory in question, and the proof-theoretical strength of the theory (also called its proof theoretical ordinal) is (roughly) the supremum of those ordinals (it supplies notations for, and) up to which the theory can prove transfinite induction.

Remark 2.2.4.2 (Constructive Set Theory). *Another application of* W*-types in conjunction with Tarski universes (see Subparagraph 2.2.3.3.8.2) is constructive set theory. [6] uses a universe* (U, T) *to interpret constructive set theory into Martin Löf type theory by considering the* W*-type (see Section 2.2.4.2)* $\mathsf{W}(\mathsf{U}, \mathsf{T})$ *of "transfinite types".*

[15]By definition, a natural number is a term $n : \mathbb{N}$, the terminology of 'numeral' is used to indicate that this n is explicitly given as the application of a specified, finite member of applications of suc to $zero$.

2.2.4.3 W-types as Normal Forms of Inductive Types

Even though W types explained in the previous subsection doubtlessly match the intuition of what an inductively type should be, one might come upon the idea that there is a more general notion of inductive type consisting of a repetition of the schema '$(a : A) \to (B(a) \to X) \to X$'.

Definition 2.2.4.3 (Inductive type (type theoretic definition)). *An* inductive type *is defined by a finite sequence of constructors*

$$C^1 : (a_1^1 : A_1^1) \to (f_1^1 : B_1^1(a_1^1) \to X) \to (a_2^1 : A_2^1(a_1^1)) \to (f_2^1 : B_2^1(a_1^1, a_2^1) \to X) \to$$
$$\cdots \to (a_{n^1}^1 : A_{n_1}^1(a_1^1, \ldots, a_{n^1-1}^1)) \to (f_{n^1}^1 : B_{n^1}^1(a_1^1, \ldots, a_{n^1}^1) \to X) \to X$$

$$C^2 : (a_1^2 : A_1^2) \to (f_1^2 : B_1^2(a_1^2) \to X) \to (a_2^2 : A_2^2(a_1^2)) \to (f_2^2 : B_2^2(a_1^2, a_2^2) \to X) \to$$
$$\cdots \to (a_{n^2}^2 : A_{n^2}^2(a_1^2, \ldots, a_{n^2-1}^2)) \to (f_{n^2}^2 : B_{n^2}^2(a_1^2, \ldots, a_{n^2}^2) \to X) \to X$$
$$\vdots$$
$$C^k : (a_1^k : A_1^k) \to (f_1^k : B_1^k(a_1^k) \to X) \to (a_2^k : A_2^k(a_1^k)) \to (f_2^k : B_2^k(a_1^k, a_2^k) \to X) \to$$
$$\cdots \to (a_{n^k}^k : A_{n^k}^k(a_1^k, \ldots, a_{n^k-1}^k)) \to (f_{n^k}^k : B_{n^k}^k(a_1^k, \ldots, a_{n^k}^k) \to X) \to X$$

where

- A_j^i *may depend on terms of* A_l^i *for* $l < j$.
- B_j^i *may depend on terms of* A_l^i *for* $l \leq j$.
- A_j^i *and* B_j^i *must not depend on any* f_l^i.
- *The type of* C^i *may depend on* C^l *for* $l < i$.

An elimination rule would have as inductive hypothesis the obvious judgements

$$\cdots \vdash m^1 : M(C^1 a_1^1 f_1^1 \ldots a_{n^1}^1 f_{n^1}^1)$$
$$\vdots$$
$$\cdots \vdash m^k : M(C^k a_1^k f_1^k \ldots a_{n^k}^k f_{n^k}^k)$$

There is however an algorithm to compute a normal form for every inductive definition reducing it to the basic form of a W type. This reduction crucially relies on the constraints on dependency between the arguments of the constructors of inductive types together with the universal properties of (dependent) sums allowing to commute non dependent summands.

Remark 2.2.4.4 (Normal forms for inductive definitions are W-types). *One can encode every inductive definition Definition 2.2.4.3 as a* W*-type: since the types of the* a_j^i *must not depend on any* f_l^i*, we can exchange the order of the occurrence of the typing judgments involving these expressions; more precisely, we can transform* C^i *to:*

$$\underline{C^i} : (a_1^i : a_1^i) \to (a_2^i : A_2^i(a_1^i)) \to (a_{n^i}^i : A_{n_1}^i(a_1^i, \dots, a_{n^i-1}^i)) \to$$

$$\dots \to (f_1^i : B_1^i(a_1^i) \to X) \to (f_2^i : B_2^i(a_1^i, a_2^i) \to X) \to \dots \to (f_{n^i}^i : B_{n^i}^i(a_1^i, \dots, a_{n^i}^i) \to X) \to X$$

Next we can sum the arguments in the following way

$$\underline{\underline{C^i}} : \Sigma(\dots(\Sigma(\Sigma(a_1^i : a_1^i)(a_2^i : A_2^i(a_1^i)))\dots(a_{n^i}^i : A_{n_1}^i(a_1^i, \dots, a_{n^i-1}^i))\dots) \to$$

$$\Sigma(\dots(\Sigma(\Sigma(B_1^i(a_1^i))(B_2^i(a_1^i, a_2^i))\dots(B_{n^i}^i(a_1^i, \dots, a_{n^i}^i))\dots) \to X) \to X$$

such that the type of $\underline{\underline{C^i}}$ *is a already a constructor type for a* W*-type. We abbreviate this type by* $dom'\underline{\underline{C^i}}$*. Finally we sum the series of constructors obtained in this way, to obtain a single constructor for a* W*-type*

$$\underline{\underline{C}} : (\Sigma dom'\underline{\underline{C^1}}(dom'\underline{\underline{C^2}}(\dots(\Sigma dom'\underline{\underline{C^{k-1}}} dom'\underline{\underline{C^k}})\dots) \to X) \to X\ .$$

In the above some care is to be taken what we mean by "encode [inductive definitions]". While it is true that the above procedure works without further assumption, we obtain from it merely an injective map from the set of inductive definitions to the set of W*-types; e.g. in case* $n = 2$*, the maps* $(B \to A) \times (B' \to A) \to (B + B' \to A)$ *and back, are given by sending* $(f, g) \mapsto [f, g] \mapsto (\mathsf{inl}_{[f,g]}, \mathsf{inr}_{[f,g]})$*. However, the intended roundtrip starting from the coproduct map* $h : B + B' \to A$*, would be* $h \mapsto (\mathsf{inl}_h, \mathsf{inr}_h) \mapsto [\mathsf{inl}_h, \mathsf{inr}_h]$*, but the latter function is not in general equal to* h *unless we assume for example function extensionality (see Paragraph 2.2.3.3.11).*

2.2.4.4 The Equivalence between Initial Algebras and Type Definitions

The remaining way to characterize inductive definitions is by construing them as initial algebras for a sufficiently general form of endofuctor, i.e. without the presumption that such an endofunctor is defined by certain constructors we have specified beforehand. This amounts to solving the problem of giving a more conceptual description of which endofunctors (on Set) should have initial algebras.

Before we can enter this discussion, we need to explain how type definitions by introduction- and elimination rules can be expressed as initial algebras for endofunctors on Set. We take here the (so far generic) example of W-types.

2.2.4.4.1 Formation- and Introduction Rule From the premise $x : A$, $t : B(x) \to \mathsf{W}\ A\ B$ of the introduction rule, we define the polynomial functor (see Definition 1.2.4.1) whose action on objects is $P : X \mapsto \Sigma_{a:A}(B(a) \to X)$ by sending a set X to the set being the Σ-type obtained from summing the types in the premise where all occurrences of the type to be defined (i.e. $\mathsf{W}\ A\ B$) is replaced by X.

We assume that the endofunctor obtained in this way has an initial algebra which we denote by $\mathsf{W}\ A\ B$.

The introduction rule supplies the algebra map $\mathsf{in} : P(\mathsf{W}\ A\ B) \to \mathsf{W}\ A\ B$ since an element $(x, t) : P(\mathsf{W}\ A\ B)$ is an instance of the premise of this rule, such that we can define $\mathsf{in}(x, t) \coloneqq \sup(x, t)$.

Conversely, if we have an initial algebra for a polynomial functor, we can introduce a constant symbol for it. Evaluating this functor in the initial algebra yields the set of all ordered pairs being terms of the types in the premise of an introduction rule. The algebra map applied to one such pair (x, t) gives a term which we denote by $\sup(a, t)$.

2.2.4.4.2 Elimination Rule For interpreting the premise of the elimination rule, we interpret the judgment defining the motif

$$w : \mathsf{W}AB \vdash M(w) : \mathsf{Set}$$

by the morphism $\mathsf{proj}_1 : \Sigma_{w:\mathsf{W}AB} M(w) \to \mathsf{W}AB$. With $\overline{M} \coloneqq \Sigma_{w:\mathsf{W}AB} M(w)$, the judgment

$$x : A,\ t : B(x) \to \mathsf{W}\ A\ B \vdash d : M(\sup(a, t))$$

is encoded by the set $P(\overline{M})$, and there is the map $P(\mathsf{proj}_1) : P(\overline{M}) \to P(\mathsf{W}\ A\ B)$. Then, if we assume a term $w : \mathsf{W}\ A\ B$, the conclusion of the elimination rule gives us a term $\mathsf{elim}(x, t, d, w) : M(w)$ which defines a map $s : \mathsf{W}\ A\ B \to \overline{M}$. Thus we can define the map $a \coloneqq s \circ \mathsf{in} \circ P(\mathsf{proj}_1) : P(\overline{M}) \to \overline{M}$ making the diagram

$$\begin{tikzcd}
P(\overline{M}) \arrow[r, dashed, "a"] \arrow[d, dashed, "P(\mathsf{proj}_1)"'] & \overline{M} \arrow[d, "\mathsf{proj}_1"] \\
P(\mathsf{W}AB) \arrow[r, "in"] & \mathsf{W}AB
\end{tikzcd} .$$

commute, i.e. the elimination rule is encode by the statement that $\overline{M}$ carries a P-algebra structure.

Conversely, assuming that $\overline{M}$ carries a P-algebra structure, initiality of $\mathsf{W}\ A\ B$ implies via Lambeks's lemma (see Lemma 1.2.3.2) that $\mathsf{proj}_1 : \overline{M} \to \mathsf{W}\ A\ B$ has a section which evaluated in $w : \mathsf{W}\ A\ B$ yields a term of $M(w)$ which satisfies the conclusion of the elimination rule.

2.2.4.4.3 Computation Rule

The computation rule, finally is equivalent to the statement that s is a P-algebra homomorphism.

⋆ ⋆ ⋆

Having seen that initial algebras for polynomial endofunctors correspond to the rules defining W-types and thus —via the reduction in Remark 2.2.4.4— to inductive definition in the type theoretical sense (as defined in Remark 2.2.4.4), it is natural to ask whether initial algebras for (general) endofunctors on Set correspond to a more general notion of inductive type that can be characterized in MLTT. As we will recall in the next paragraph, this question has been answered in the negative.

2.2.4.4.4 Strict Positivity

Inspired by the following informal definition

Terminology 2.2.4.5 (Inductive definition (functorial definition)). *An inductive definition of a set is given by an endofunctor on* Set *definable in a metatheory which is a constructively acceptable version of ZFC having an initial algebra.*

Dybjer [36] established that all inductive definitions in the sense of being defined by an initial algebra of an 'admissible endofunctor' are equivalently W-types; these results in [36] were strengthened [4] to what is called there *strictly positive types* which subsume nested inductive- and coinductive types.. As an exact rendition of the notion of admissible endofunctor, loc.cit. uses the notion of a *strictly positive endofunctor*.

The idea is that an endofunctor F is *strictly positive* if it is defined from the standard operations for set formation, i.e. for a set X, the set $F(X)$ must iteratively be constructed from finite (possibly empty) sums, finite (possibly empty) products, and —so one is tempted to add– function types. In the last case of the constructor for function types, however, special care is to be taken: the term “strictly positive” means to indicate that in the type expression $F(X)$, the variable X must not occur to the left of '$\to$'; there is also a weaker notion of *positivity* were X may occur only to the left of an even number of $\to$ symbols such as in $(X \to 2) \to 2$. To see why this constraint is necessary, we recall Lambek's theorem (see Lemma 1.2.3.2), X being an initial algebras for F implies that there is an isomorphism $F(X) \simeq X$, and thus any reasonable definition of F should not try to enforce an impossible isomorphism.

More specifically, the constraint of strictly positive occurrence is motivated by Cantor's theorem stating that there is no ZF set isomorphic to its power set, in symbols there is no X such that $X \simeq X \to 2$ and thus admitting of $X \to 2$ as a possible value of F in X would mean that the resulting type theory has no model in ZF. A more constructively charged example (than Cantor's theorem) why one may wish to rule out non strictly-positive definitions that double negation of a proposition is intuitionistically not equivalent to the proposition itself, i.e. since negation is defined as $\neg X := (X \to \bot)$, then $\neg\neg A$ is intuitionistically not equivalent to A.

A category theoretic rendering of [36] (that strictly positive type definitions are equivalent to W types) is given in [45], namely that the class of polynomial functors is the least class of functors containing the pullback functors and their adjoints and which is closed under *composition*. In particular, this more abstract gloss becomes available only if inductive definitions are closed under composition; that they are we will recall in Section 2.2.4.6 for which we need some more definitions given in Section 2.2.4.5.

2.2.4.5 The Equivalence of Inductive Types, Families of (opposite) Sets, and Containers

The set Ind of all inductive definitions (as defined in Definition 2.2.4.3) can itself be defined inductively. Each *code* $c : \mathsf{Ind}$ for an inductive definition defines an endofunctor $[\![\, c \,]\!]$ on Set having an initial algebra by Corollary 1.2.4.3.

Remark 2.2.4.6 (The inductive type of inductive definitions). *There is an inductive type* Ind *of all inductive definitions defined by the constructors*

$$\begin{aligned}
&\mathsf{base} : \mathsf{Ind}\\
&\mathsf{sum} : (A : \mathsf{Set}) \to (A \to \mathsf{Ind}) \to \mathsf{Ind}\\
&\mathsf{dsum} : \mathsf{Set} \to \mathsf{Ind} \to \mathsf{Ind}
\end{aligned}$$

One can recursively define a function[16] $[\![\ ]\!] : \mathsf{Ind} \to \mathsf{Set} \to \mathsf{Set}$ *by*

$$\begin{aligned}
&[\![\, \mathsf{base} \,]\!]\ X := 1\\
&[\![\, \mathsf{sum}\ A\ f \,]\!]\ X := \Sigma_{a:A}[\![\, f(a) \,]\!]X\\
&[\![\, \mathsf{dsum}\ A\ c \,]\!]\ X := \Sigma_{k:A\to X}[\![\, c \,]\!]X \quad .
\end{aligned}$$

There are mutual semantics preserving translations:

Lemma 2.2.4.7 (Translation $\mathsf{Ind} \to \mathsf{Cont}$).

$$\begin{aligned}
&\mathsf{tr} : \mathsf{Ind} \to \mathsf{Cont}\\
&\mathsf{tr}(\mathsf{base}) := (1, \lambda\_ \to 0)\\
&\mathsf{tr}(\mathsf{sum}Af) := (\Sigma_{a:A}(\mathsf{proj}_1\mathsf{tr}(f(a))), \lambda(a,x) \to (\mathsf{proj}_2\mathsf{tr}(fa))x)\\
&\mathsf{tr}(\mathsf{dsum}Ac) := (\mathsf{proj}_1\mathsf{tr}(c), \lambda x \to A + (\mathsf{proj}_2\mathsf{tr}(c))x)
\end{aligned}$$

Proof. Let $X : \mathsf{Set}$.

$$\begin{aligned}
[\![\, \mathsf{tr}\ \mathsf{base} \,]\!]X &= [\![\, 1, \lambda x \to 0 \,]\!]X\\
&= \Sigma_{x:1}(0 \to X)\\
&=\simeq 1\\
&= [\![\, \mathsf{base} \,]\!]X
\end{aligned}$$

[16] We use the notation $[\![\ ]\!]$ also for other "decoding functions" and let the context indicate which one is meant.

$$\begin{aligned}
[\![\ \mathsf{tr\ sum}Af\ ]\!]X &= [\![\ (\Sigma_{a:A}(\mathsf{proj}_1\mathsf{tr}(f(a))), \lambda(a,x) \to (\mathsf{proj}_2\mathsf{tr}(fa))x)\ ]\!]X \\
&\simeq \Sigma_{(a,x):\Sigma_{a:A}\mathsf{proj}_1(\mathsf{tr}(f(a)))}((\mathsf{proj}_2(\mathsf{tr}(f(a)))x) \to X) \\
&\simeq \Sigma_{a:A}\Sigma_{\mathsf{proj}_1(\mathsf{tr}(fa))}((\mathsf{proj}_2(\mathsf{tr}(f(a)))x) \to X) \\
&= \Sigma_{a:A}[\![\ \mathsf{tr}(f(a))\ ]\!]X \\
&\simeq \Sigma_{a:A}[\![\ f(a)\ ]\!]X \\
&= [\![\ \mathsf{sum}Af\ ]\!]X
\end{aligned}$$

$$\begin{aligned}
[\![\ \mathsf{tr}(\mathsf{dsum}Ac)\ ]\!]X &= [\![\ (\mathsf{proj}_1\mathsf{tr}(c), \lambda x \to A + (\mathsf{proj}_2\mathsf{tr}(c))x)\ ]\!]X \\
&= \Sigma_{x:\mathsf{proj}_1\mathsf{tr}(c)}((A + \mathsf{proj}_2\mathsf{tr}(c))x \to X) \\
&\simeq \Sigma_{x:\mathsf{proj}_1\mathsf{tr}(c)}\Sigma_{k:(A+\mathsf{proj}_2\mathsf{tr}(c))x\to X}1 \\
&\simeq \Sigma_{x:\mathsf{proj}_1\mathsf{tr}(c)}\Sigma_{k:A\to X}\Sigma_{k':\mathsf{proj}_2\mathsf{tr}(c)x\to X}1 \\
&\simeq \Sigma_{k:A\to X}\Sigma_{x:\mathsf{proj}_1\mathsf{tr}(c)}(\mathsf{proj}_2\mathsf{tr}(c)x \to X) \\
&\simeq \Sigma_{k:A\to X}[\![\ \mathsf{tr}c\ ]\!]X \\
&= \Sigma_{k:A\to X}[\![\ c\ ]\!]X \\
&= [\![\ \mathsf{dsum}Ac\ ]\!]X \ \ .
\end{aligned}$$

Lemma 2.2.4.8 (Translation Cont → Ind).

$$\begin{aligned}
\mathsf{rt} &: \mathsf{Cont} \to \mathsf{Ind} \\
\mathsf{rt}(S,T) &\coloneqq \mathsf{sum}\ S(\lambda\ s \to \mathsf{dsum}T(s)(\mathsf{base}))
\end{aligned}$$

Proof. Let X : Set.

$$\begin{aligned}
[\![\ \mathsf{rt}(S,T)\ ]\!]X &= [\![\ \mathsf{sum}\ S(\lambda\ s \to \mathsf{dsum}T(s)(\mathsf{base}))\ ]\!] \\
&= \Sigma_{s:S}\Sigma_{k:T(s)\to X}[\![\ \mathsf{base}\ ]\!] \\
&\simeq [\![\ (S,T)\ ]\!]X \ \ .
\end{aligned}$$

Corollary 2.2.4.9. Cont *is a retract of* Ind *with retraction* tr*.*

Proof.

$$\begin{aligned}
\mathsf{tr}(\mathsf{rt}(S,T)) &= \mathsf{tr}(\mathsf{sum}\ S(\lambda\ s \to \mathsf{dsum}T(s)(\mathsf{base}))) \\
&= (\Sigma_{s:S}(\mathsf{proj}_1(\mathsf{tr}(\mathsf{dsum}T(s)(\mathsf{base})))), \lambda(s,x) \to \mathsf{proj}_2(\mathsf{tr}(\mathsf{dsum}T(s)(\mathsf{base})))x) \\
&= (\Sigma_{s:S}(\mathsf{proj}_1(1, \lambda x \to T(s) + 0)), \lambda(s,x) \to \mathsf{proj}_2(1, \lambda x \to T(s) + 0)x) \\
&\simeq (\Sigma_{s:S}1, \lambda s \to T(s)) \\
&= (S,T)
\end{aligned}$$

Remark 2.2.4.10. *Corollary 2.2.4.9Lemma 2.2.4.7, and Lemma 2.2.4.8 together justify to call the containers* tr(c) *a* normal forms *of the inductive definition c. (Compare this with Remark 2.2.4.4.)*

2.2.4.6 Composition of Inductive Definitions

A central point of this thesis will be composability for inductive-recursive definitions which we will begin to discuss in Chapter 4. In this section we will recall the special case that inductive definitions are composable. The statement that they do compose, as well as elementary description of the composition operation as such are well known (see e.g. [45]), and we will give a characterization focusing on two components that will be important for our generalizations later on: the bind operation (defined in Definition 1.2.5.3), and power objects (defined in Definition 1.2.1.1). To do so, we make use the semantical equivalence between Cont and Ind (see Section 2.2.4.5), and show composability for Cont.

Definition 2.2.4.11 (Composition for containers).

$$\begin{aligned} \_ \circ \_ &: \mathsf{Cont} \to \mathsf{Cont} \to \mathsf{Cont} \\ (S,P) \circ (U,T) &\coloneqq (\Sigma_{s:S}(P(s) \to U), \lambda(s,f) \to \Sigma_{x:P(s)} T(f(x))) \end{aligned}$$

In container notation $(\llbracket\ (S,P)\ \rrbracket\ U, \lambda(s,f) \to \llbracket\ P(s),\ \lambda x \to 1\ \rrbracket\ (T(f(x))$.

Lemma 2.2.4.12 (Container composition commutes with container evaluation). *In the situation of the previous definition we have isomorphisms* $\llbracket\ (S,P) \circ (U,T)\ \rrbracket \simeq \llbracket\ (S,P)\ \rrbracket \circ \llbracket\ (U,T)\ \rrbracket$.

Since Cont is by definition $\mathsf{Fam}(\mathsf{Set}^{op})$, we can use the bind operation

$$\begin{aligned} \_ \ggg\!= \_ &: \mathsf{Fam}(D) \to (D \to \mathsf{Fam}(E)) \to \mathsf{Fam}(E) \\ (U,T) \ggg\!= h &= ((\mu^{\mathsf{Fam}} \circ \mathsf{Fam}^{\to})(h))(U,T) \\ &= \mu^{\mathsf{Fam}}(U, h \circ T) \\ &= (\Sigma_{u:U} proj_1(h \circ T)(u), \lambda(u,k) \mapsto proj_2((h \circ T)u)(k)) \end{aligned}$$

and find the following corollary

Corollary 2.2.4.13 (Composition of containers in terms of bind and powers). *For* (S,P) $(U,T) : \mathsf{Cont}$ *we have*

$$(S,P) \circ (U,T) = (S,P) \ggg\!= e^{(U,T)}$$

for the map

$$\begin{aligned} e^{(U,T)} &: \mathsf{Set} \to \mathsf{Fam}(\mathsf{Set}) \\ e^{(U,T)}(X) &\coloneqq (X \to U, \lambda h \to \Sigma_{x:X} T(hx)) \ . \end{aligned}$$

This map $e^{(U,T)}$ essentially sends a set $S : \mathsf{Set}$ to the power of the family (U,T) by this set S. It is, however, important to take the word 'essentially' literally here: in Section 1.3 we defined an operation

$$\begin{aligned} \_ \longrightarrow_{\mathsf{Fam}} \_ &: \big(S : \mathsf{Set}\big) \to \mathsf{Fam}\ D \to \mathsf{Fam}\ (S \to D) \\ S \longrightarrow_{\mathsf{Fam}} (A,P) &= (S \to A, g \mapsto P \circ g) \end{aligned}$$

but the resulting object is —for an arbitrary D— only a power in the category of elements $\big(\Sigma D : \mathsf{Set}_1\big)(\mathsf{Fam}\ D)$ of the functor Fam[17]. In the latter category one can show that there is an isomorphism

$$(\mathsf{Set}/U, (X \to U, \lambda h \to (\mathsf{proj}_2 : \Sigma_{x:X} T(hx) \to U)))) \simeq (X \to \mathsf{Set}, (X \to U, \lambda h \to T \circ h))\ .$$

Remark 2.2.4.14 (Composition and initial algebras). *The fact that one can compose inductive definitions raises the question about the effect of composition of the types they define via initial algebra semantics, i.e. given inductive definitions I, J, and $K = I \circ J$, how do their initial algebras $\mu[\![\ I\ ]\!]$, $\mu[\![\ J\ ]\!]$, and $\mu[\![\ K\ ]\!]$ relate? [14] provides a collection of general elementary answers to this question in the situation of more general endofunctors. We are not going to pursue this in this thesis but leave it for future work.*

⋆ ⋆ ⋆

2.3 Conclusion and Outlook

In this subsection we have presented W types as a universal coding scheme for inductively defined types in MLTT. It is however not universal enough since we have already seen definitions in MLTT which are not covered by it: Tarski universes (see Paragraph 2.2.3.3.8) and identity types (see Paragraph 2.2.3.3.9). This is because both definitions are not defining single types $U : \mathsf{Set}$ but families $(U, T : U \to \mathsf{Set})$ of types. It is not possible to construe these families as e.g. a collection of independent W types since for example in the definition of a Tarski universe T cannot properly be accommodated and the definition of U depends on values of T which in turn refer to terms of U. Induction-recursion that we start to present in the next chapter solves such problems by providing a more general coding schema that can define families $(U, T) : \mathsf{Fam}(D)$ (for arbitrary $D : \mathsf{Set}_1$).

[17]This category (where we omit the obvious definition of the morphisms) is equivalent to the comma category i/Set_1 of the inclusion $i : \mathsf{Set} \to \mathsf{Set}_1$)see Remark 1.3.0.8).

Chapter 3

Inductive-Recursive Definitions

In this chapter we will review the basic theory of induction-recursion as developed by Peter Dybjer and Anton Setzer. We will start with an overview on the literature on the topic, continue with an explanation of the basic motivational ideas (predicativity, universes à la Tarski). We then explain one [40] of two (a priori) different axiomatization of induction-recursion by (large) sets of codes, their decoding to (endo)functors on categories of families (DS functors), give some examples and a discussion of DS functors. We close this chapter by reviewing another axiomatization of induction-recursion given by Dybjer [38] which historically precedes their first-mentioned one but which is more important for the other variants of induction-recursion we present later in Chapter 5 and Chapter 6.

3.1 Introduction

We will begin this introductory section by briefly commenting on the (in our opinion) major articles about introduction-recursion available so far (see Section 3.1.1), we will then (in Section 3.1.2) motivate induction-recursion from the viewpoint of *predicativism* (which we already briefly mentioned in the discussion of universes in Paragraph 2.2.3.3.8) supplemented by *reflection principles* (see, Remark 1.1.0.15), we will also give a second motivation about extending techniques for defining inductive set to inductive-recursive families (see Section 3.1.3).

3.1.1 Bibliographic History of Induction-Recursion

Universes in type theory which are the motivating examples for induction-recursion are described in:

1. Intuitionistic type theory, Per Martin-Löf. Defines a Tarski universe, the (probably first) example of an inductive-recursive definition. [88].
2. Predicative type universes and primitive recursion, Nax Paul Mendler, 1991. [94].

Indexed types, like families of types, consist of a type together with a function from this type to Set (or another universe of type). The difference to families as defined by induction-recursion is, firstly, that one considers collections of types indexed by a *fixed* type while for families one considers collections where the indexing type is not fixed, and, secondly, for inductive indexed types, the indexing type does not need to be an inductively defined type, i.e. the indexing function is not defined by recursion on the indexing type, but the fibers of the indexed types are instead defined inductively. An example of an indexed type is the identity type (see Remark 3.2.1.13). *Indexed containers* are generalizations of containers '(A, B)' carrying additional indexings in two sets; indexed containers define endofunctors on the slice categories Set/X for small sets X and via initial algebra semantics define indexed types.

1. Inductive sets and families in Martin Löf's type theory and their set theoretic semantics, Peter Dybjer, 1991. [35].
2. Inductive families, Peter Dybjer, 1994. [33].
3. Indexed containers, Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris, 2015. [8].

The next-mentioned three papers define what is usually understood by the term induction-recursion: the object of inductive-recursive definitions are families $(\mathsf{U}, \mathsf{T}) \in \mathsf{Fam}D$ (for arbitrary large $D : \mathsf{Set}_1$) where U and T are defined simultaneously. The fourth paper introduces indexed inductive-recursive definitions — an extension of induction-recursion where not a family of types but an indexed family of types $(\mathsf{U}_i, T_i : \mathsf{U}_i \to D_i)_{i:I}$ is defined; an important example of such a definition is Martin-Löf's propositional identity type.

4. A general formulation of simultaneous inductive-recursive definitions in type theory, Peter Dybjer, 2000. [32]. The first general formulation of induction-recursion.

5. A finite axiomatization of inductive-recursive definitions, Peter Dybjer and Anton Setzer, 1999. [38].
6. Induction-recursion and initial algebras, Peter Dybjer and Anton Setzer. [40]. Defines a large set of codes $\mathsf{DS}\ D\ E$ each of which encodes an inductive-recursive definitions, together with an initial-algebra semantics and a set-theoretical model.
7. Indexed induction-recursion, Peter Dybjer and Anton Setzer, 2006. [39].

There are several possibilities how to advance the theory of induction-recursion regarding scope and expressibility of inductive-recursive definitions.

8. Fibered data types, Neil Ghani, Lorenzo Malatesta, Fredrik Nordvall Forsberg, Anton Setzer, 2013. [112]. The category $\mathsf{Fam}(\mathsf{Set})$ is a fibered category, and Dybjer-Setzer induction-recursion recursion defines certain families. In this paper, this is generalized to a version defining elements of more general fibered categories in place of $\mathsf{Fam}(\mathsf{Set})$.
9. Small induction-recursion, Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, Thorsten Altenkirch, 2013 [58]. It is shown that the set $\mathsf{DS}\ D\ E$ of Dybjer-Setzer codes for induction-recursion for *small* sets D, $E : \mathsf{Set}$ (as opposed to large sets $D\ E : \mathsf{Set}_1$) defines exactly inductive types aka polynomials (see Section 1.2.4). From the equivalence of small induction-recursion and polynomials, a notion of morphisms between codes is derived from the existing notion of morphisms between polynomials.
10. Positive inductive-recursive definitions, Neil Ghani, Lorenzo Malatesta, Fredrik Nordvall Forsberg, 2013. [50]. Dybjer-Setzer codes define endofunctors on the categories $\mathsf{Fam}D$ where D is a discrete category. This is generalized here to non-discrete D. A notion of morphism between codes is considered.
11. Containers, monads, and induction-recursion, Neil Ghani, Peter Hancock, 2016. [48]. (A previous version is titled “an algebraic foundation and implementation of induction-recursion”). Develops decoding of DS codes in terms of initial algebra semantics of the large inductive type DS D E. Presents an equivalent axiomatization of DS by encoding σ and δ in a single constructor $\sigma\delta$.
12. Variations on inductive-recursive definitions, Neil Ghani, Conor McBride, Fredrik Nordvall Forsberg, Stephan Spahn, 2017. [52]. Defines systems of induction recursion where codes are composable.

On the proof theory of inductive-recursive definitions.

13. Proof Theory of Martin-Löf Type Theory – An Overview, Anton Setzer, 2004. [114]
14. Extending Martin-Löf Type Theory by one Mahlo-Universe, Anton Setzer, 2000. [113].
15. Proof Theory and Martin-Löf Type Theory, Anton Setzer, 2007. [115].

The theory of inductive-inductive definitions is related to inductive recursive definitions. In both definitional two components are defined simultaneously. ¡while in case of induction-recursion, one component is inductively defined and the other by recursion, in case of induction-induction both components are defined by induction. Also on the level of the definition of the definitional principles themselves there are relations: for example the sets of codes for positive inductive-recursive definitions (mentioned above in this bibliography) are definable by induction-induction.

16. Inductive-inductive Definitions, Fredrik Nordvall Forsberg, Anton Setzer, 2010. [99].
17. A categorical semantics for inductive-inductive definitions, Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg and Anton Setzer, 2011.[9].
18. Inductive-inductive definitions, Fredrik Nordvall Forsberg. [98].

3.1.2 Motivation I: Universes, Reflection Principles, and Predicative Definitions

Martin-Löf [88] intends the type theory defined in this paper to be a "full scale system for formalizing intuitionistic mathematics as developed, for example, in the book by Bishop" (see [18] for a newer edition of that book). This means in particular that it should be a formalization of set theory. Since universes and large cardinals whose existence is not provable in plain MLTT are a part of set theory as currently pursued, it is a natural question to which extend these extension can be formalized in MLTT (or extensions thereof) as well. The notion of a universe closed under the set-formers for dependent sums-, and products, as well as inductive definitions were formalized already in the first version of MLTT. This did however not include notions of universes closed under universe-formation as occurring in the definition of Mahlo cardinals, or other closure properties induced by large cardinal axioms. We already mentioned that one way to justify such kinds of universes are reflection principles Remark 1.1.0.15. The idea of reflection principles also explicitly informs one axiomatization of induction recursion that we will recall in Section 3.3 and we already explained what characterizes inductive/constructive definition and we will discuss a further aspect that enters with the passage to induction-recursion in the next paragraph Section 3.1.3. Before coming to this, we briefly mention one further "taxon" in the collection of properties characterizing definitional principles to which induction-recursion, or more specifically the formulation of Mahlo universes [1] in type theory via induction-recursion is often assigned to; we are talking about *predicativity*:

According to Bertrand Russell's conception [109](see also Poincaré [103]), a definition (of a totality) is *predicative* if an element of this totality is conceived not by reference to the totality it belongs to (and impredicative else). Unfortunately, taken in full generality, this is an empty concept: can we define the empty set predicatively? We would need to be able to discuss an element of it without reference to the (empty) totality, but we cannot express the idea of 'no element' without referring to the absence of any object which is the only characterization of the empty set we know of. Also a natural number is defined by being reachable from zero by the successor function in a finite number of steps; but

[1]an important example of a kind of universe that can be formulated by induction-recursion but which is not definable in the original version of MLTT in [88].

what is "a finite number" of steps? Well, it is synonymously a *natural number* of steps. More generally we cannot define any object in this way.

Predicativity as it is conventionally understood is thus rather a definitional principle starting from an accepted object (such as the natural numbers), together with an admissible operation of *predication* —usually formalized as a power-set operation, i.e. either $\mathbb{N} \to 2$ a term (i.e. a "predicate" or proposition) of which is a decidable subset, or $\mathbb{N} \to U$ where U is a universe in need of its own justification— that may only be applied a finite number of times. In this respect, the idea of predicative definitions is similar to that of induction. This can also be seen by making a connection to the complementarity of introduction-, and elimination rules in type theory: for example the induction axiom of Peano arithmetic —asserting that one can prove propositions on the natural numbers by reduction to their constructors— expresses the inference from the acceptability of $\mathbb{N}$ as constituted by its constructors to computability of terms of its object of predications ($\mathbb{N} \to 2$ or $\mathbb{N} \to U$) by 'elimination' (in the sense of and elimination rule). In still other words, the acceptability of the induction axiom on natural numbers is *grounded* in the inductive structure of the natural numbers while the proposition that the natural numbers are exhausted by this inductive structure is guaranteed by the induction axiom.

Returning to the topic of induction-recursion —in particular to Mahlo universes— we see that it neither matches the informal original definition of predicativity, nor the relaxed conception of the notion as "a type that is built up incrementally": while a Tarski universe closed under Σ-type, say, can still be imagined as being an incremental construction since one can "add the Σ-type of a randomly chosen dependent type", the definition of the Mahlo universe makes explicit reference to all endofunctions of the universe itself:

Definition 3.1.2.1. Set *has the* (external) Mahlo property *if for every function*

$$h : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Fam}(\mathsf{Set})$$

there is a family $(U_h, T_h) : \mathsf{Fam}(\mathsf{Set})$ *that is closed under* h *in the sense that there is a function*

$$\hat{h} : [(U_h, T_h)]U_h \to [(U_h, T_h)]U_h$$

such that

$$T_h \hat{h}_0(a, b) = h_0(T_h(a), T_h \circ b)$$
$$T_h(\hat{h}_1(a, b, c) = h_1(T_h a, T_h a, T_h \circ b, c)$$

where $(h_0, h_1) = h$, $(\hat{h}_0, \hat{h}_1) = \hat{h}$*), and we again used container notation* $[(A, B)]V = \Sigma_{v:A}(B(a) \to V)$.

Remark 3.1.2.2 (Internal Mahlo Universe). *While the external Mahlo universe can be defined by induction-recursion and is usually considered to be predicative (see [40, §6.3], we will also recall this in Example 3.2.1.10), there is also the notion of an internal Mahlo universe: briefly, while we in Definition 3.1.2.1 considered families* $(U_h, T_h) : \mathsf{Fam}(\mathsf{Set})$ *for* $h : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Fam}(\mathsf{Set})$ *that were closed under* h *in the described way, for the internal Mahlo universe, this definition is relativized to a universe* $(V, S) : \mathsf{FamSet}$ *and*

$h : \mathsf{Fam}(V) \to \mathsf{Fam}(V)$ *such* (U_h, T_h) *is a subuniverse of* (V, S) *that is closed under this* h *in an analogous way (see [40, p.31]). This definition of an internal Mahlo universe is however -as explained in loc.cit.- not an inductive-recursive definition since one of the constructors is not strictly positive (compare Paragraph 2.2.4.4.4 and Section 3.1.3), and not considered to be predicative:*

> *"[...] the usual [i.e. internal] construction in type theory, where the Mahlo universe has a constructor that refers to all total functions from families of sets in the Mahlo universe into itself; such a construction is, in the absence of a further analysis,* impredicative.*"([70], emphasis added)*

See [124] for a general discussion of different notions of predicativity in constructive- and non-constructive mathematics unrelated to induction-recursion.

3.1.3 Motivation II: "Strict Positivity for Families"

In Section 2.2.4.1 we said that one abstract way to conceive inductive definitions is as initial algebras for an 'admissible' endofunctor $F : \mathsf{Set} \to \mathsf{Set}$ where the class of 'admissible' endofunctors is described by a list of sufficiently formal, constructively acceptable conditions implying that an admissible functor indeed has an initial algebra. As we explained in Paragraph 2.2.4.4.4, 'admissiblity' for endofunctors on Set can be taken to be strict positivity, i.e. closure under the standard operators of set formation.

The same idea of induction as initial algebras is apparently available for endofunctors on more general categories such as $\mathsf{Fam}(D)$ which is the relevant category for induction-recursion. It is however prima facie unclear what an admissible endofunctor of $\mathsf{Fam}(D)$ should be:

Firstly, since for arbitrary $D : \mathsf{Set}$, the category $\mathsf{Fam}(D)$ is poorly endowed with limits (see Remark 1.3.0.7), there are less operators available on $\mathsf{Fam}(D)$ (than on the topos Set) under which $\mathsf{Fam}(D)$ is closed: e.g. $\mathsf{Fam}(D)$ does not generally have products and power objects; taking a power of a family in $\mathsf{Fam}(D)$ by a set A yields a family in $\mathsf{Fam}(A \to D)$. This change of the index set by some operators is a major difference between sets and families. As a consequence fixating us on one 'D' and endofunctors $\mathsf{Fam}(D) \to \mathsf{Fam}(D)$ in a single inductive-recursive definition might limit the space of inductive-recursive possible in this way[2].

Secondly, the problem which choice of operators matches the mutual dependency of U and T in the intended inductive-recursive definition requires some more detailed thought.

Dybjer-Setzer [40] addressed in their axiomatization DS Section 3.2.1 of induction-recursion mainly the second mentioned concern by providing constructors for DS that can encode

[2]A middle way between the extremes of fixing one D once and for all, and studying endofunctors on the category i/Set_1 (see Remark 1.3.0.8) of all families for all index (large) sets D which we shall pursue later on (see Remark 5.6.0.1) is to consider endofunctors on $\mathsf{Fam}(D)$ that may factor through $\mathsf{Fam}(E)$ for an E different from D in a way related to the construction of the inductive-recursive definition.

constructors for (U, T) that are "half positive"[3] in the sense that for an argument (U, T)[4] of the endofunctor, U must occur only strictly positive in the definition of the pair (U, T) while T may appear in negative position.

The axiomatization DS' [38] which we will review in Section 3.3 is similar in its emphasis on strictly positive operations on sets but is somewhat more flexible in regard to the indexing set.

3.2 DS

Dybjer-Setzer introduced the system DS axiomatizing inductive-recursive definitions in [40]. The authors regarded this system as a simplification of another system DS' serving the same purpose of axiomatizing inductive-recursive definitions that they had published earlier in [38]. Both systems are equivalent as shown in [40] in case the logical framework is chosen appropriately.

The systems defining inductive-recursive definitions we consider in this thesis always have three parts: 1) a (large) set of *codes* (such as $\mathsf{DS}\ D\ E$ or $\mathsf{DS}'\ D\ E$ for every $D\ E\ : \mathsf{Set}_1$) is defined, 2) for each code, $c : \mathsf{DS}\ D\ E$ say, a decoding functor $[\![\ c\ ]\!] : \mathsf{Fam}(D) \to \mathsf{Fam}(E)$ is recursively defined, 3) finally, in case $D = E$, to c is associated a family $(\mathsf{U}_c, \mathsf{T}_c)$ (where we usually omit this subscript c) which is exhibited by a generic elimination rule as the initial algebra[5] of $[\![\ c\ ]\!]$.

3.2.1 DS Codes and Their Decoding

For $D, E : \mathsf{Set}_1$, the large set $\mathsf{DS}\ D\ E : \mathsf{Set}_1$ of Dybjer-Setzer codes is inductively defined by the three constructors

$$\begin{aligned}
\iota &: E \to \mathsf{DS}\ D\ E \\
\sigma &: (A : \mathsf{Set}) \to (A \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E \\
\delta &: (A : \mathsf{Set}) \to ((A \to D) \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E
\end{aligned}$$

Recursively on $\mathsf{DS}\ D\ E$ is defined the decoding function $[\![\ \ ]\!] : \mathsf{DS}\ D\ E \to \mathsf{Fam}(D) \to \mathsf{Fam}(E)$ which —like every function into a category of families— can be given in two components $[\![\ \ ]\!] = \langle [\![\ \ ]\!]_0, [\![\ \ ]\!]_1 \rangle$

[3]See loc. cit. p.4, footnote 2. They follow other authors in the terminology of "half positivity".

[4]Notice the different fonts (U, T) for an argument of the endofunctor and (U, T) for a family that is an initial algebra for this functor.

[5]The semantics of DS' was originally (i.e. in [38]) given in another (but equivalent) way without explicit reference to endofunctors on categories of families.

$$
\begin{aligned}
&[\![\ ]\!]_0 : \mathsf{DS}\ D\ E \to \mathsf{Fam}(D) \to \mathsf{Set}\\
&[\![\ \iota\ e\ ]\!]_0(U,T) \coloneqq 1\\
&[\![\ \sigma\ A\ f\ ]\!]_0(U,T) \coloneqq \Sigma_{a:A}[\![\ f(a)\ ]\!]_0(U,T)\\
&[\![\ \delta\ A\ f\ ]\!]_0(U,T) \coloneqq \Sigma_{G:A\to U}[\![\ f(T\circ G)\ ]\!]_0(U,T)
\end{aligned}
$$

$$
\begin{aligned}
&[\![\ ]\!]_1 : (c : \mathsf{DS}\ D\ E) \to ((U,T) : \mathsf{Fam}(D)) \to [\![\ c\ ]\!]_0(U,T) \to E\\
&[\![\ \iota\ e\ ]\!]_1(U,T)* \coloneqq e\\
&[\![\ \sigma\ A\ f\ ]\!]_1(U,T)(a,x) \coloneqq [\![\ f(a)\ ]\!]_1(U,T)x\\
&[\![\ \delta\ A\ f\ ]\!]_1(U,T)(G,x) \coloneqq [\![\ f(T\circ G)\ ]\!]_1(U,T)x
\end{aligned}
$$

The action on (cartesian) morphisms $h : (U,T) \to (U',T')$ between families in $\mathsf{Fam}(D)$ which are determined by the action on the indexing sets (see Terminology 1.3.0.2) is given by:

$$
\begin{aligned}
&[\![\ ]\!]^{\to}\ h : [\![\ c\ ]\!]_0(U,T) \to [\![\ c\ ]\!]_0(U',T')\\
&[\![\ \iota\ e\ ]\!]^{\to}\ h = \mathsf{id}_1\\
&[\![\ \sigma\ A\ f\ ]\!]^{\to}\ h\ (a,x) = (a, [\![\ f(a)\ ]\!]^{\to}\ h\ x)\\
&[\![\ \delta\ A\ f\ ]\!]^{\to}\ h\ (g,x) = (h\circ g, [\![\ f(T'\circ h\circ g)\ ]\!]^{\to}\ h\ x)
\end{aligned}
$$

where the last line is correctly types since by the assumption of h to be a cartesian morphism we have $T'\circ h = T$.

We will give a prose explanation of these constructors and their decoding just below Remark 3.2.1.1, and a justification of decoding in categorical terms in Section 3.2.3.4.

The third, still outstanding, layer in the formalization DS given by initial-algebra semantics for the DS functors $[\![\ c\ ]\!]$, we will explain later in Section 3.2.2.

Remark 3.2.1.1 (Explanation of the constructors of DS and their decoding). *Let us assume that (U,T) is an initial algebra for a DS code. The set U is inductive-recursively defined in the sense that in its definition do not only appear arguments $x : \mathsf{U}$ of itself (this would be the degenerate case of an inductive definition) but also values $\mathsf{T}x$ of T which is defined simultaneously (with U) by recursion on U. What we have just explained on the side of the initial algebra $(\mathsf{U}\ ,\ \mathsf{T})$ is reflected on the side of codes by three possibilities of what such an argument of a constructor can be:*

- *It can be either trivial — case ι which forms the basis of the induction, in which no real induction takes place beyond the definition of a constant functor $\lambda\_ \to (1, \lambda* \to e)$ having as initial algebra the object $(1, \lambda* \to e)$ the functor is constant on.*

- *Or it can be a non-inductive argument (followed by some further arguments) — case σ, where in $\llbracket\ f a\ \rrbracket(U, T)$ the argument a does neither depend on U nor on T. This results in a definition without non-degenerate simultaneity of the induction and the recursion taking place; i.e. the whole definition could instead be carried out by first completing the definition of* U *and, in a second step, defining* T *by recursion on* U*. In fact the constructors ι and σ together without a further constructor define only constant functors: while σ allows us to form set-indexed sums of functors, ι allows to assign only point values such that the most interesting code*[6] *we can form is of the form $\sigma A\lambda a \to \iota B(a)$ which decodes to the functor constant on the family* $(A, B) : \mathsf{Fam}(D)$*.*
- *Or it can be an inductive argument (followed by further arguments) — case δ, where the second argument of δ allows for assignment to a code of a family of values of* T*. Here the argument $T \circ G$ in $\llbracket\ f(T \circ G)\ \rrbracket$ depends on T. While $\llbracket\ \iota e\ \rrbracket_0(U, T)$ and $\llbracket\ \sigma A f\ \rrbracket_0(U, T)$ are obviously strictly positive expressions in U (under the assumption that their subcodes are), this is not the case for the δ-constructor for which we have*

$$\begin{aligned}\llbracket\ \delta A F\ \rrbracket_0(U, T) &= \Sigma_{G:A\to U}\llbracket\ F(G \circ T)\ \rrbracket(U, T)\\ &= \Sigma_{G:A\to U}\llbracket\ ((A \to \_)(F) \circ (\_ \to D)(G))(T)\ \rrbracket(U, T)\\ &= \Sigma_{G:A\to U}\llbracket\ ((A \to \_)(F) \circ (\_ \to D)(G))(T)\ \rrbracket(U, T)\\ &= \Sigma_{G:A\to U}\llbracket\ ((A \to \_)(F))(T \circ G \to D))\ \rrbracket\end{aligned}$$

  i.e. there occurs a reference to a term of U (of "half" of (U, T)), albeit only as an argument of T in negative position as can be seen in the last line of the above equations.

  Remark 2.2.4.6 implies that $\mathsf{DS}\ 1\ 1$ *where the δ constructor is modified to* $\delta' : \mathsf{Set} \to \mathsf{DS}\ 1\ 1 \to \mathsf{DS}\ 1\ 1$ *—with decoding $\llbracket\ \delta'\ A\ c\ \rrbracket X :\!= \Sigma_{k:A\to X}\llbracket\ c\ \rrbracket X$ for X :* Set*— does allow only for degenerate inductive-recursive definition equivalent to just inductive ones. Also, the stronger statement, that* DS *D E for small sets D E :* Set *is equivalent to a set of inductive definitions is true [58].*

It is instructive to go through this explanation one more time in the motivating example for induction-recursion, namely the Tarski universe (U, T) *closed under Σ-types which we described in Subparagraph 2.2.3.3.8.2. The endofunctor defining the family* (U, T) *can be encoded by the code —i.e. by the constructors of* DS *D E— as $\delta 1\ \lambda X.\ \delta X(*)\lambda Y.\ \iota\Sigma X(*)Y$ and the decoding recovers the defining functor:*

$$\llbracket\ \delta 1\ \lambda X.\ \delta X(*)\lambda Y.\ \iota\Sigma X(*)Y\ \rrbracket_0(U, T) = \Sigma_{p:1\to U}\Sigma_{b:T(p*)\to U}\llbracket\ \iota\Sigma(T \circ p)(T \circ b)\ \rrbracket_0(U, T)$$

[6]This is true only up to equivalence since, firstly, merging two subsequent σ codes $\sigma A\lambda a.\sigma B(a)\lambda b.R(a, b)$ to one sigma code $\sigma(\Sigma_{a:A}B(a)\lambda(a, b)R(a, b)$ does usually not hold definitionally, and secondly, in $\sigma A f$, the subcodes $f(a)$ can be a σ code for one a and ι for others - but also for this one can find an equivalent code decoding to a constant functor.

$$[\![\ \delta 1 \lambda X.\ \delta X(*) \lambda Y.\ \iota \Sigma X(*) Y\ ]\!](U, T) \simeq (\Sigma_{u:U}(T(u) \to U), (u, b) \mapsto \Sigma T(u)(T \circ b))$$

Here the constructor $\overline{\Sigma} : (u : U) \to (\mathsf{T}(u) \to \mathsf{U}) \to \mathsf{U}$ encoded by this code is said to take an "inductive argument" $h : T(u) \to \mathsf{U}$ depending upon the hypothetical judgement $\mathsf{U} : \mathsf{Set},\ u : \mathsf{U} \vdash \mathsf{T}(u) : \mathsf{Set}$ whose premises are not yet proven and themselves depend on the type whose definition they support.

Of course, one can also regard $\mathsf{DS}\ D\ E$ itself as a universe of codes and its constructors as reflecting operations under which it shall be closed: the base case ι gives us for every term $e : E$ a (code for) a ground type ιo) while σ gives us for every family $(A, f : A \to \mathsf{DS}\ D\ E)$ of codes indexed by a set a new code $\sigma A f$; together ι, σ, and δ' cover all ordinary inductive definitions. δ finally allows to go beyond ordinary induction by indexing families of codes by families of objects of D.

Notice that there is a difference between the definition as we give it and the original one given in [40]: loc.cit. only considered systems of the form $\mathsf{DS}\ D\ D$, i.e. where $E = D$, which is seemingly sufficient for the discussion of initial algebras. It is however conceptually advantageous to consider DS as an operation taking two arguments, since then it is functorial in its second argument and opfunctorial in the first. This would not be the case if we consider DS as an operation taking only one argument because the ι constructor would indicate functoriality while the δ constructor would indicate op-functoriality, and as a result, the system as a whole would be neither functorial nor op-functorial. Funcoriality of DS (and the other systems of induction-recursion we will consider in later sections) becomes relevant for example in the discussion of the question of composability of codes. The following remark should be compared to the other systems of induction-recursion Chapter 5 Chapter 6 where the analogous statement does not obtain.

Remark 3.2.1.2 (DS as W-type). *For all $D\ E : \mathsf{Set}_1$, the large set $\mathsf{DS}\ D\ E$ is inductively defined and thus we can encode it as a large W-type by $\mathsf{DS} := \mathsf{W}\ X\ Y$ where $X := E + \mathsf{Set} + \mathsf{Set}$, $Y(\mathsf{in}_0\ e) = 0$, $Y(\mathsf{in}_1\ A) = A$, $Y(\mathsf{in}_2\ A) = A \to D$.*

The following remark will receive further discussion in Remark 3.2.1.17.

Remark 3.2.1.3 (Monotonicity of decoding in the partial order on families). *We have seen above that decoding of DS-codes sends cartesian morphism to cartesian morphisms. We have also seen that the partial order on $\mathsf{Fam}(D)$ defines a special class of cartesian morphisms (see Section 1.3). One can prove by induction on codes that DS functors are monotonous operators on this partial order. This monotonicity is important for the existence of initial algebras of DS functors*[7]*. Remark 3.2.1.17.*

[7]We do not recall this proof here which can be found in [40] but we will give a proof of a more general statement in Section 6.4.

3.2.1.1 Subcodes

One difference between DS-codes and codes of the other system of induction-recursion we shall define in Chapter 5 and Chapter 6 are *subcodes*; the latter correspond to subtrees if we consider $\mathsf{DS}\ D\ E$ as W-type (see Remark 3.2.1.2).

Remark 3.2.1.4 (Monotonicity of decoding in subcode ordering). *There is an obvious partial order on codes given by*

$$
\begin{aligned}
&\_ \prec \_ \;:\; \{D\ E\ :\ \mathsf{Set}_1\} \;\to\; \mathsf{DS}\ D\ E \;\to\; \mathsf{DS}\ D\ E \;\to\; \mathsf{Set}_1\\
&c\ \prec\ (\iota\ e)\ =\ \bot\\
&c\ \prec\ (\sigma\ A\ f)\ =\ \exists(a:A)((f\ a)\ \equiv\ c)\\
&c\ \prec\ (\delta\ A\ F)\ =\ \exists(X:(A\to D)))(F\ X\ \ \equiv\ c))
\end{aligned}
$$

A partial order for codes is also discussed in [40, Lemma 5.3.7].

Remark 3.2.1.5 (Types of subcodes). *With an eye to the variations of induction-recursion we will present in Chapter 5 and Chapter 6, we highlight that if $c' \prec c$ is a subcode of $c : \mathsf{DS}\ D\ E$, the type of this subcode is always $\mathsf{DS}\ D\ E$, too. If we denote by IR such another system the type of a subcode could also be $IR\ D'\ E'$ where $D'\ E'$ do not necessarily equal D respectively E.*

3.2.1.2 Examples of DS Codes

We give some examples of codes.

Example 3.2.1.6 (Coproducts of codes). *If $c,\ c'\ :\ \mathsf{DS}\ D\ E$ are two codes, we can define a new code $\sigma 2 f$, where 2 is the set with 2 elements, and $f : 0_2 \mapsto c$ and $f : 1_2 \mapsto c'$. This code decodes to the coproduct of the functors defined by c and c', i.e. there are isomorphisms $[\![\ c + c'\ ]\!](U,T) \simeq [\![\ c\ ]\!](U,T) + [\![\ c\ ]\!](U,T)$ for every $(U,T) : \mathsf{Fam}(D)$, and since by [13, Proposition 8.8] colimits in functor categories are computed pointwise, this defines a coproduct in $\mathsf{Fam}(D) \to \mathsf{Fam}(E)$. Since for every D, $\mathsf{Fam}\ D$ can be characterised as the free Set-indexed coproduct cocompletion of D (see Remark 1.3.0.7), taking coproducts is one of the most basic and native operations to be performed on them.*

We can now give an encoding of the endofunctor defining the family of finite sets we mentioned in the introduction Section 0.2.

Example 3.2.1.7 (The family of finite sets). *The family $(\mathbb{N}, \mathsf{Fin})$ where Fin assigns to a natural number n a set with n elements is initial algebra for the functor $F : \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Fam}(\mathsf{Set})$ where $F_0(U,T) = U+1$, and $F_1(U,T)(\mathsf{in}_l\ u) = T(u)+1$, and $F_1(U,T)(\mathsf{in}_r\ *) = \emptyset$. This functor has code $(\delta\ 1\ \lambda X.\iota X(*)) + (\iota\emptyset)$.*

Example 3.2.1.8 (W-types). *1. By choosing $D = E = 1$, we can use* $\mathsf{DS}\ 1\ 1$ *to represent inductive definitions. Let us encode Martin-Löf's type* $\mathsf{W}\ S\ P : \mathsf{Set}$ *of wellfounded trees, where* $S : \mathsf{Set}$ *encodes the set of shapes of the tree, and* $P : S \to \mathsf{Set}$ *maps each shape to its branching degree. Recall that this type is inductively defined by the constructor*

$$\mathsf{sup} : \big(s : S\big) \to (P(s) \to \mathsf{W}\ S\ P) \to \mathsf{W}\ S\ P$$

Here we see that sup *takes one non-inductive argument $s : S$, followed by an inductive argument $P(s) \to$* $\mathsf{W}\ S\ P$*, which depends on the first non-inductive one.* $\mathsf{W}\ S\ P$ *can be represented by the code* $c_{\mathsf{W}\ S\ P} : \mathsf{DS}\ 1\ 1$ *with* $c_{\mathsf{W}\ S\ P} = \sigma\, S\, (s \mapsto \delta\, P(s)\, (\_ \mapsto \iota \star))$ *where σ is used for the non-inductive argument and δ for the inductive one, and finally a closing ι.*

For the decoding note that $\mathsf{Fam}\ 1 \cong \mathsf{Set}$ *since the second component of such a family is trivial. Thus, if $(W, T) :$* $\mathsf{Fam}\ 1$*, then*

$$[\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_0(W, T) = \big(\Sigma s : S\big)\big((P(s) \to W) \times 1\big) \tag{3.1}$$

such that indeed $\mathsf{sup} : [\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_0(\mathsf{W}\ S\ P, \_) \to \mathsf{W}\ S\ P$ *(up to isomorphism), and initial algebras of* $[\![\ c_{\mathsf{W}\ S\ P}\ ]\!] : \mathsf{Fam}\ 1 \to \mathsf{Fam}\ 1$ *are* W*-types.*

2. Instead of leaving the fibres of the family trivial, we can "upgrade" the given code to do something interesting in the whole family. For instance, if we redefine $c_{\mathsf{W}\ S\ P} : \mathsf{DS\ Set\ Set}$ *by*

$$c_{\mathsf{W}\ S\ P} = \sigma\, S\, (s \mapsto \delta\, P(s)\, (Y \mapsto \iota\, \Big(\big(x : P(s)\big) \to Y\ x\Big)))$$

the index set decoding (3.1) *stays the same, but the decoding* $[\![\ c_{\mathsf{W}\ S\ P}\ ]\!]_1(W, T)$ *applies T everywhere in the tree. In particular, if we choose $S = \mathbb{N}$ and $P =$* Fin*, where* $\mathsf{Fin}\ n$ *is a finite type with n elements, then* $[\![\ c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin}}\ ]\!](X, T) \cong (\mathsf{List}\, X,\ [x_1, \ldots, x_n] \mapsto T\, x_1 \times \ldots \times T\, x_n)$ *. We will see a use of this upgraded code later in Example 5.10.0.1.*

We have mentioned the example of a Tarski universe closed under a set former such as Σ already several times as motivating examples for IR. But only closing under such a set former without assuming that the universe contains something is an induction without base case and as such defines only the empty set. We can however use coproducts of codes add codes of set formers and a code assuming a set the universe shall contain.

Example 3.2.1.9 (A universe containing 2 which is closed under W-types). *We get considerably more power by choosing $D = E =$* Set*. Now we can represent a universe containing* 2 *that is* closed under W-types *by the code* $c_{2\mathsf{W}} : \mathsf{DS\ Set\ Set}$*, where*

$$c_{2\mathsf{W}} = \sigma\ \{\mathsf{bool}, \mathsf{w}\}\ (\mathsf{bool} \mapsto \iota\ 2; \mathsf{w} \mapsto \delta\ 1\ (X \mapsto (\delta\ (X \star)\ (Y \mapsto \iota\ (\mathsf{W}\ (X \star)\ Y)))))$$

First we offer a choice between two constructors: bool *and* w *using σ. In the* bool *case, we use an ι code to ensure the name* bool *decodes to* 2*; in the* w *case, we ask for a name a for the shapes of the* W*-type using δ* 1*, and for every element in the decoding of that name, we ask for a name for the branching degrees using δ $(X\star)$ — here $X : 1 \to$* Set

represents the decoding of the name a. The rest of the code gets to depend on the decoding $Y : X\star \to \mathsf{Set}$ of this family, and we finish by declaring that this constructor decodes to $\mathsf{W}\ (X\star)\ Y$. Note that this code can be written as a coproduct of codes $c_2 +_{\mathsf{DS}} c_{\mathsf{W}}$: generally for $c\ d : \mathsf{DS}\ D\ E$, we define their coproduct $c +_{\mathsf{DS}} d = \sigma\ 2\ (\mathsf{ff} \mapsto c\ ;\ \mathsf{tt} \mapsto d)$. We will return to this in Example 5.5.0.1.

The decoding of the code $c_{2\mathsf{W}} : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}$ from Example 3.2.1.9 satisfies $[\![\ c_{2\mathsf{W}}\ ]\!]_0(U,T) \cong 1 + \big(\Sigma a : U\big)(T(a) \to U)$ with $[\![\ c_{2\mathsf{W}}\ ]\!]_1(U,T)\ (\mathsf{inl}\,\star) = 2$ and $[\![\ c_{2\mathsf{W}}\ ]\!]_1(U,T)\ (\mathsf{inr}\,(a,b)) = \mathsf{W}\ (T\,a)\ (T \circ b)$ which are the equations for a universe closed under W-types.

Example 3.2.1.10 (Mahlo universe). *A code for the family (U_h, T_h) participating in the definition of the Mahlo universe described Definition 3.1.2.1 is*

$$\iota N + \delta 1 \lambda A.\delta A(*)\lambda B.\iota f(A(1),B)) + \delta 1 \lambda A.\delta A(*)\lambda B.\sigma f(A(1),B))\lambda C.g(A(1),B,C)\ .$$

Remark 3.2.1.11 (Products of codes). *As we mentioned before, $\mathsf{Fam}(D)$ does not have (all) products. There is thus no definition of the product of DS codes and more generally, there is no system of codes for induction-recursion in which products of codes (in case they exist) decode to a product in $\mathsf{Fam}(D)$. A step towards a pointwise products of codes has been taken in ([58][49]).*

Example 3.2.1.12 (Padding of a code). *If $c : \mathsf{DS}\ D\ E$ is any code, the code $\sigma 1\ \lambda_- \to c$ is a code whose semantics is isomorphic to that of c.*

Remark 3.2.1.13 (Fitting the identity type into the taxonomy of definition principles). *The so-called "identity type" (see Paragraph 2.2.3.3.9) is special among the definitions in MLTT: its type signature $\mathsf{Id} : (A : \mathsf{Set}) \to ((a,a') : A \times A) \to \mathsf{Set}$ indicates that it is not definable by DS. It is however definable by* indexed induction-recursion *[39].*

On the other hand, since the constructors $\mathsf{refl} : (x : A) \to \mathsf{Id}_A(x,x)$ do not take inductive arguments, the identity type is covered by the framework for defining inductive families defined in [33, §5.1.2].

Example 3.2.1.14 (An unbounded DS-code). *For sets $X, Y : \mathsf{Set}$, we define*

$$\begin{aligned}
&z : \mathbb{N} \to \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}\\
&z(\mathsf{zero}) = \iota X\\
&z(\mathsf{suc}(n)) = \sigma\ Y\ \lambda_- \to \delta Y \lambda_- \to z(n)
\end{aligned}$$

and the code $Z = \sigma\mathbb{N}\ z : \mathsf{DS}\ D\ E$. The code $z(n)$ has maximal path length $2n+1$ (where by "path length" we mean the number of constructors in a path of his code), and the code Z has consequently unbounded path length.

It is however important to notice here what is necessary to define this code: since $\mathsf{DS}\ D\ E : \mathsf{Set}_1$ is a large set different from Set, defining z requires large elimination which is not part of MLTT (without DS). In this sense, the definition of Z is one in iterated induction-recursion. Comparing this situation to inductive definition, we recall that there is a *difference between inductive definitions in* MLTT *[36], and iterated inductive definitions (aka* nested inductive definitions*) in* MLTT *[4] which assumes an additional fixpoint combinator corresponding to the elimination machinery for* DS*.*

Example 3.2.1.15 (Realizability and computability predicates). *C. Coquand [27] shows normalization of a fragment of MLTT by a realizability argument which can be formalized as an inductive-recursive definition. A similar method using an inductive-recursively defined* computability predicate *was used by Martin-Löf [88] to show normalization for the (inconsistent) system defined in loc. cit.*

Example 3.2.1.16 (Reflexive-transitive closure of a graph). *A graph is a family* $(E, P) : \mathsf{Fam}(V \times V)$*. The free category* $(\mathsf{E}', \mathsf{P}') : \mathsf{Fam}(V \times V)$ *on* (E, P) *is given by the following inductive-recursive definition:*

$$\begin{aligned}
&\mathsf{E}' : \mathsf{Set}_1\\
&\mathsf{ids} : V \to \mathsf{E}'\\
&\mathsf{edg} : E \to \mathsf{E}'\\
&\mathsf{comp} : (e : \mathsf{E}') \to (e' : \mathsf{E}') \to (\mathsf{proj}_1\,(\mathsf{P}'\,e') \equiv \mathsf{proj}_2\,(\mathsf{P}'\,e)) \to \mathsf{E}'
\end{aligned}$$

$$\begin{aligned}
&\mathsf{P}' : \mathsf{E}' \to (V \times V)\\
&\mathsf{P}'\,(\mathsf{ids}\,v) = (v\,,\,v)\\
&\mathsf{P}'(\,\mathsf{edg}\,e) = P\,e\\
&\mathsf{P}'(\mathsf{comp}\,p\,q\,r) = (\mathsf{proj}_1\,(\mathsf{P}'\,p)\,,\,\mathsf{proj}_2\,(\mathsf{P}'\,q))
\end{aligned}$$

This functor is represented by the coproduct $\mathsf{DS}\ V\ (V \times V)$ *code with summands*

$$\begin{aligned}
&\sigma V \lambda v \to \iota(v, v)\\
&\sigma E \lambda e \to \iota(Pe)\\
&\delta 1 \lambda X \to \delta 1 \lambda Y \to \sigma(\mathsf{proj}_1 Y(*) \equiv \mathsf{proj}_2 X(*)))\lambda z \to \iota(\mathsf{proj}_1 X(*), \mathsf{proj}_2 Y(*))
\end{aligned}$$

So, the reflexive-transitive-closure functor is the functor that sends (E, P) *to the initial algebra of the above* DS*-functor.*

Remark 3.2.1.17 (Discussion of DS Functors). *What can we say about* DS *functors in their own right and how would we approach the problem to characterise them categorically? A possible class of functors to which we could wish to compare them are* parametric right adjoints *(pra) as studied by [128](and others).*

A functor $F : A \to B$ *between categories with terminal object is a parametric right adjoint if it factors as* $A \to B/F(1) \to C$ *into the by* F *induced action on slice categories (where* A *is identified with* $A/1$*) followed by dependent sum.*

DS *functors and pras are both generalizations of polynomial functors. An example for a pra which is not a polynomial functor but whose action can still be defined by induction-recursion (see Example 3.2.1.16) is the functor assigning to a graph the free category on it (loc.cit. p.671).*

More generally, many functors as they appear in higher category theory are describable in terms of pras. Unfortunately, however, the theory of induction-recursion we presented here is inadequate with regard to the theory of pras since, for example, terminal objects and non cartesian morphisms are essential for the latter. That $\mathsf{Fam}(D)$ *for an arbitrary* D *does not have a terminal object is as such not a problem since as soon as* D *has a terminal object, also* $\mathsf{Fam}(D)$ *has one. But the fact that* DS *codes define only an action on cartesian morphisms (in Dybjer-Setzer's account even only on split cartesian morphisms) is a problem since the morphisms to the terminal objects are not in general cartesian. For example* $\mathsf{Fam}(\mathsf{Set})$ *has a terminal object (see Remark 1.3.0.7) given by* $(1, \lambda * \to 1)$*, but the unique morphisms* $t : (U, T) \to (1, \lambda * \to 1)$ *are obviously not cartesian unless for all* $u :$ *we have* $T(u) = 1$ *which is not generally the case.*

To see why DS *functors only act on cartesian morphisms, we recall (from Section 3.2.1) how this action is defined. Let* $c : \mathsf{DS}\ D\ E$ *and write*[8] $[\![\ c\ ]\!]^{\to}$ *for this action where for* $h : (U, T) \to (U', T')$ *, (i.e.* $h = (h_0 : U \to U', h_1 : T \Rightarrow T')$ *where* $T \Rightarrow T' :\equiv (u : U) \to T(u) \equiv T'(hu)$*)*

$$[\![\ c\ ]\!]^{\to} h = ([\![\ c\ ]\!]_0^{\to} h, [\![\ c\ ]\!]_1^{\to} h : (x : [\![\ c\ ]\!]_0(U, T)) \to [\![\ c\ ]\!]_1(U, T)x \equiv [\![\ c\ ]\!]_1(U', T')([\![\ c\ ]\!]^{\to} hx)$$

is defined by

$$\begin{aligned}
&[\![\ \iota e\ ]\!]^{\to} h : (1, \lambda x \to e) \to (1, \lambda x \to e)\\
&[\![\ \iota e\ ]\!]^{\to} h :\equiv (\mathsf{id}_1, \mathsf{refl})
\end{aligned}$$

$$\begin{aligned}
&[\![\ \sigma A f\ ]\!]^{\to} h : (\Sigma_{a:A}[\![\ fa\ ]\!]_0(U, T), \lambda ax.[\![\ fa\ ]\!]_1(U, T)x) \to (\Sigma_{a:A}[\![\ fa\ ]\!]_0(U', T'), \lambda ax.[\![\ fa\ ]\!]_1(U', T')x)\\
&[\![\ \sigma A f\ ]\!]^{\to} h :\equiv (\langle id, ([\![\ fa\ ]\!]^{\to} h)_0\rangle, \lambda(a, x).([\![\ fa\ ]\!]^{\to} h)_0 x : [\![\ fa\ ]\!]_1(U, T)x \equiv [\![\ fa\ ]\!]_1(U, T)([\![\ fa\ ]\!]^{\to} h)_0 x))
\end{aligned}$$

and we display the δ *case in the diagram*

$$\begin{tikzcd}
\Sigma_{k:A\to U}[\![\ F(T\circ k)\ ]\!]_0(U,T) \arrow[rr, "\langle \lambda k.h\circ k, \Psi\rangle"] \arrow[dr, "(t,x)\mapsto [\![\ F(T\circ k)\ ]\!]_1(U,T)\ x"'] & & \Sigma_{k':A\to U'}[\![\ F(T'\circ k')\ ]\!]_0(U',T') \arrow[dl, "(t',x')\mapsto [\![\ F(T'\circ k')\ ]\!]_1(U',T')\ x'"] \\
& D &
\end{tikzcd}$$

where

[8] Notice that we changed the notation here slightly compared to Section 3.2.1.

$$\Psi := (subst \llbracket F(\_ \circ k) \rrbracket_0(U', T')(h_1)) \circ (\llbracket F(T \circ k) \rrbracket^{\to} h)_0$$

for

$$subst \llbracket F(\_ \circ k) \rrbracket_0(U', T')(h_1) : \llbracket F(T \circ k) \rrbracket_0 U'T' \to \llbracket F(T' \circ h \circ k) \rrbracket_0 U'T'$$

and the filling cell of type

$$\lambda(k, x) \to \llbracket F(T \circ k) \rrbracket_1(U, T)\ x \equiv \llbracket F(T' \circ \circ h \circ k) \rrbracket_1(U', T')x$$

is the inductive hypothesis.

A term $\llbracket F(T \circ k) \rrbracket_0(U', T') \to \llbracket F(T' \circ h \circ k) \rrbracket_0(U'T')$ we can however only obtain if there is really an equality $T \Rightarrow T' \circ h$. A solution to this problem was suggested in [49]: by introducing morphisms between codes one obtains a morphism $F(T \circ k) \to F(T' \circ h \circ k)$ and can define a functor $\llbracket \_ \rrbracket_0(U'T')$, but this necessitates F to be a proper functor —as opposed to a mere function— as well.

We thus conclude that for studying the category theory of inductive recursive definitions it is recommended to choose a system defining morphisms between codes as well. This system is likely to be a different one from that of [49] since for example products of codes decode only to pointwise products and not in general to products in categories of families as would be desirable.

We will not discuss morphisms of codes any further in this thesis.

3.2.2 The Rules for Dybjer-Setzer Families

Having defined Dybjer-Setzer codes $c : \mathsf{DS}\ D\ E$ and their decoding $\llbracket c \rrbracket$, we come to the definition of the families they define, where —since these families will be initial algebras for endofunctors— we have to choose $D = E$.

The first two rules are the formation rules for a family $(\mathsf{U}_c, \mathsf{T}_c)$[9] defined by a code $c : \mathsf{DS}\ D\ D$

$$\frac{c : \mathsf{DS} D\ D}{\mathsf{U}_c : \mathsf{Set}}\ \mathsf{U}\text{-}\textsc{form} \qquad\qquad \frac{c : \mathsf{DS} D\ D}{\mathsf{T}_c : \mathsf{U}_c \to D}\ T\text{-}\textsc{form}$$

and the next two rules assert that (U, T) is an algebra for the endofunctor $\llbracket c \rrbracket : \mathsf{Fam}(D) \to \mathsf{Fam}(D)$ by introducing a structure map intro. The second rule states that T is defined by recursion on U.

$$\frac{c : \mathsf{DS} D\ D}{\mathsf{intro} : \llbracket c \rrbracket_0(\mathsf{U}_c, \mathsf{T}_c) \to \mathsf{U}_c}\ \mathsf{U}\text{-}\textsc{intro} \qquad\qquad \frac{c : \mathsf{DS} D\ D \qquad x : \llbracket c \rrbracket_0(\mathsf{U}_c, \mathsf{T}_c)}{\mathsf{T}_c\ (\mathsf{intro}\ x) = \llbracket c \rrbracket_1(\mathsf{U}_c, \mathsf{T}_c)\ x}\ \mathsf{T}\text{-}\textsc{rec}$$

[9]We avoid here to write subscripts $(\mathsf{U}_c, \mathsf{T}_c)$.

The elimination rule for inductive-recursive definition has a form similar to that for ordinary inductive definitions: for every motif $P : \mathsf{U} \to \mathsf{Set}_1$ (notice that we eliminate into large sets), for every $x : \mathsf{U}$ we want to obtain a term $t : Px$ by using the induction principle for U which has as argument functions (called *step functions*) of the form

$$\mathsf{step} : (x : [\![c]\!]_0(\mathsf{U}, \mathsf{T})) \to \mathsf{IH}\ c\ P\ x \to P(\mathsf{intro}\ x))\ .$$

This says that given the assumption $(x : [\![\mathsf{U}]\!]_0(\mathsf{U}, \mathsf{T}))$, we can —by invoking the inductive hypothesis $\mathsf{IH}\ c\ P\ x$— draw the conditional (under the assumptions made) conclusion that $P(\mathsf{intro}\ x)$. From this, the induction principle elim gives us the desired term $\mathsf{elim}\ P\ \mathsf{step}\ x\ :\ P\ x$. More formally we have the typing

$$\begin{aligned}\mathsf{elim}\ :\ & (P\ :\ \mathsf{U}_c\ \to\ \mathsf{Set}_1)\ \to\\ & (\mathsf{step}\ :\ (x\ :\ [\![\ c\ ]\!]_0\ (\mathsf{U}_c\ ,\ \mathsf{T}_c))\ \to\ \mathsf{IH}\ c\ P\ x\ \to\ P\ (\mathsf{intro}\ x))\ \to\\ & (x\ :\ \mathsf{U}_c)\ \to\ P\ x\end{aligned}$$

and defining elim by induction on U_c means that we have to give a definition for $\mathsf{elim}\ P\ \mathsf{step}\ (\mathsf{intro}\ x)$ and it is apparent that this definition needs to be an instance of $\mathsf{step}\ x$. Thus we need a function that produces (recursively) from the data we already have, an instance of the inductive hypothesis, where the latter is (for $(U, T) : \mathsf{Fam}(D)$) defined by:

$$\begin{aligned}&\mathsf{IH}\ :\ (c\ :\ \mathsf{DS}\ D\ E)\ \to\\ &\quad (P\ :\ U\ \to\ \mathsf{Set}_1)\ \to\ [\![\ c\ ]\!]_0\ (U, T)\ \to\ \mathsf{Set}_1\\ &\mathsf{IH}\ (\iota\ e)\ P\ \_\ = 1\\ &\mathsf{IH}\ (\sigma\ A\ f)\ P\ (a\ ,\ x)\ =\ \mathsf{IH}\ (f\ a)\ P\ x\\ &\mathsf{IH}\ (\delta\ A\ F)\ P\ (h\ ,\ x)\ =\ ((x\ :\ A)\ \to\ P\ (h\ x))\ \times\ \mathsf{IH}\ (F\ (T\ \circ\ h))\ P\ x\ .\end{aligned}$$

Such a function is

$$\begin{aligned}&\mathsf{mapIH}\ :\ (c\ :\ \mathsf{DS}\ D\ E)\ \to\\ &\quad (P\ :\ U\ \to\ \mathsf{Set}_1)\ \to\ (g\ :\ (x\ :\ U)\ \to\ P\ x)\ \to\\ &\quad (x\ :\ [\![\ c\ ]\!]_0(U, T))\ \to\ \mathsf{IH}\ c\ P\ x\\ &\mathsf{mapIH}\ (\iota\ e)\ P\ g\ \_\ =\ *\\ &\mathsf{mapIH}\ (\sigma\ A\ f)\ P\ g\ (a\ ,\ x)\ =\ \mathsf{mapIH}\ (f\ a)\ P\ g\\ &\mathsf{mapIH}\ (\delta\ A\ F)\ P\ g\ (h\ ,\ x)\ =\ (g\ \circ\ h\ ,\ \mathsf{mapIH}\ (F\ (T \circ h))\ P\ g\ x)\end{aligned}$$

Now we can recursively call elim, and for $c\ :\ \mathsf{DS}\ D\ D$ give the definition

$$\mathsf{elim}\ P\ \mathsf{step}\ (\mathsf{intro}\ x)\ =\ \mathsf{step}\ x\ (\mathsf{mapIH}\ c\ P\ (\mathsf{elim}\ P\ \mathsf{step})\ x)\ .$$

Remark 3.2.2.1 (Alternative elimination rules). *[39, §5.4] points out that for some purposes it poses a problem that* $\mathsf{IH}\ c\ P\ x : \mathsf{Set}_1$ *in the previous definition is a large set, and a solution is presented (see [39, Appendix B]) that replaces* IH *by a definition returning only a small set, as well as assorted definitions replacing* mapIH, *and* elim. *These alternative rules are equivalent in the sense of [39, Lemma B.2] to the original set of rules* IH, mapIH, *and* elim *we explained above. Since for our purposes the size of* $\mathsf{IH}\ c\ P\ x$ *is not relevant we shall stick to the original rules.*

Remark 3.2.2.2 (In which sense can U be understood as an inductively defined set?). *The terminology of induction-recursion might be misleading in the following sense: in previous sections we have seen that an inductive sets are initial algebras for polynomial functors where a set is here understood to be a set in ZFC. However both terms "inductive" and "set" are strictly speaking undefined terms in the present more general setting. As we will see later,* $\mathsf{MLTT} + \mathsf{DS}$ *does not posses a model in ZFC, but in (at least) ZFC + M + I where M and I are large cardinals. Thus it is unclear (and unlikely) that the set* U *can be defined by a polynomial functor on* Set.

3.2.3 Functorial Aspects of DS

We start this subsection by collecting some properties of $\mathsf{DS}\ D\ \_ : \mathsf{Set}_1 \to \mathsf{Set}_1$ as functor in its second argument. These properties will be of interest in subsequent chapters. Of more intrinsic interest at this point is Section 3.2.3.3 exhibiting $\mathsf{DS}\ D\ \_$ as a free monad, and Section 3.2.3.4 using the universal property of free monads to compute decoding of codes.

The basic observation is that DS is a functor in its second argument[10], i.e.

$$\begin{aligned}
&\mathsf{DS} \ : \ \{D\ E\ E' \ : \ \mathsf{Set}_1\} \ \to \ (E \ \to \ E') \ \to \ \mathsf{DS}\ D\ E \ \ \to \ \mathsf{DS}\ D\ E' \\
&\mathsf{DS}\ h\ (\iota\ e) \ = \ \iota\ (h\ e) \\
&\mathsf{DS}\ h\ (\sigma\ A\ f) \ = \ \sigma\ A\ (\lambda\ a \ \to \ \mathsf{DS}\ h\ (f\ a)) \\
&\mathsf{DS}\ h\ (\delta\ A\ F) \ = \ \delta\ A\ (\lambda\ g \ \to \ \mathsf{DS}\ h\ (F\ g))\ .
\end{aligned}$$

One can also show that it is op-functorial in its first argument.

3.2.3.1 $\mathsf{DS}D\_$ and Decoding

We also record that the functor $\mathsf{DS}\ D\_$ commutes with decoding, i.e. for $C\ E\ E' \ : \ \mathsf{Set}_1$, $c \ : \ \mathsf{DS}\ C\ E$, $P \ : \ \mathsf{Fam}\ C$, $h \ : \ E \ \to \ E'$ there is an equality

$$[\![\ \mathsf{DS}\ h\ c \ ]\!]\ P \ = \ \mathsf{Fam}^{\to}\ h\ ([\![\ c \ ]\!]\ P)\ .$$

[10]Recall that we use Agda convention to put arguments in type signatures in braces { , } if we do not want to display them as arguments of the terms of the types of this signature.

3.2.3.2 Monad Structure of DS

Moreover, $\mathsf{DS}\ D\ \_$ is a monad whose unit is given by its first constructor.

$$\begin{aligned}
&\eta\mathsf{DS}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ E\ \to\ \mathsf{DS}\ D\ E\\
&\eta\mathsf{DS}\ e\ =\ \iota\ e
\end{aligned}$$

And multiplication by:

$$\begin{aligned}
&\mu\mathsf{DS}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{DS}\ D\ (\mathsf{DS}\ D\ E)\ \to\ \mathsf{DS}\ D\ E\\
&\mu\mathsf{DS}\ (\iota\ c)\ =\ c\\
&\mu\mathsf{DS}\ (\sigma\ A\ f)\ =\ \sigma\ A\ \lambda\ a\ \to\ \mu\mathsf{DS}\ (f\ a)\\
&\mu\mathsf{DS}\ (\delta\ A\ F)\ =\ \delta\ A\ \lambda\ h\ \to\ \mu\mathsf{DS}\ (F\ h)
\end{aligned}$$

Like every monad, there is a bind operation derivable from it.

$$\begin{aligned}
\_ \gg= \_\ &:\ \{D\ E\ E'\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{DS}\ D\ E\ \to\ (E\ \to\ \mathsf{DS}\ D\ E')\ \to\ \mathsf{DS}\ D\ E'\\
c\ \gg=\ h\ &=\ \mu\mathsf{DS}\ (\mathsf{DS}\ h\ c)
\end{aligned}$$

3.2.3.3 DS as a Free Monad, $\sigma\delta$-Codes

One can moreover show [48] that the monad structure on DS is universal in the following sense. The implications of this are useful for the generalization of DS in Chapter 6.

We first define the following operation which is a covariant functors in its second argument and a contravariant functor in its first argument.

Definition 3.2.3.1. *For $D, E : \mathsf{Set}_1$ we define*

$$\mathsf{CFam}\ D\ E \coloneqq \Sigma_{(U,T):\mathsf{Fam}(\mathsf{Set})}(\llbracket\ (U,T)\ \rrbracket D \to E)\ .$$

where we use the container notation $\llbracket\ (U,T)\ \rrbracket D = \Sigma_{u:U}(T(u) \to D)$ (see also Definition 1.2.4.1).

Definition 3.2.3.2 ($\sigma\delta$-codes). *One can encode the two constructors*

$$\begin{aligned}
&\sigma : (A : \mathsf{Set}) \to (A \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E\\
&\delta : (A : \mathsf{Set}) \to ((A \to D) \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E
\end{aligned}$$

into one:

$$\sigma\delta : \mathsf{CFam}\ D(\mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$$

with the decoding

$$[\![\ \sigma\delta Q\ h\ ]\!]_0(U,T) \coloneqq \Sigma_{(s,f):[\![\ Q\ ]\!]U}[\![\ h(s, T \circ f)\ ]\!]_0(U,T)\ .$$

and it is not difficult to see how to define $[\![\ \sigma\delta Q\ h\ ]\!]_1$*.*

Proof. We have the semantics-preserving transformations:

$$\begin{aligned}
&\sigma\ A\ f \mapsto \sigma\delta(A, \lambda\_.0)(\lambda(a,\_).f(a))\\
&\delta\ A\ f \mapsto \sigma\delta(1, \lambda\_.A)(\lambda(*,g).F(g))\\
&\sigma\delta\ (A,B)\ h \mapsto \sigma A(\lambda a.(\delta B(a)\ \lambda g \to h(a,g)))
\end{aligned}$$

Remark 3.2.3.3 (DS D _ as a free monad). *The constructors ι and $\sigma\delta$ express that* DS $D\_$ *is the free monad of the functor* CFam $D\_$*.*

Proof. We instantiate [45, Proposition A.4]:

For all $D\ E$: Set_1, we define functors $P = \mathsf{CFam}\ D\ \_$ and $P_E = (Y \mapsto E + P(Y))$. By Definition 3.2.3.2, DS $D\ E$ is initial algebra of P_E. By the implication (iii) to (i) of [45, Proposition A.4], P has a pointwise free monad. [45, Proposition A.3] implies that this pointwise free monad is the initial algebra of P_E. □

Remark 3.2.3.4 (Failure of uniformization of unbounded DS-codes). *Returning to Example 3.2.1.12 where we have seen that one can prolong a code by padding without changing its semantics, one might come upon the idea that —using the $\sigma\delta$ presentation of* DS *(see Example 3.2.1.14)— one could replace every* DS *code by a code with isomorphic semantics all of whose branches have the same length. This is however not true as the Example 3.2.1.14 of a code with unbounded path length shows: by Item 2 we would produce by padding a "code" with $\mathbb{N}$ as maximal path length but this is impossible since* DS $D\ E$ *is a wellfounded set and as such all its term have finite path length.*

Remark 3.2.3.5 (Non-constructivity of uniformization of DS-codes). *A different problem the attempt of applying the padding procedure with the aim of giving codes a uniform structure has to face, is that this procedure cannot be defined by recursion on* DS $D\ E$ *—even not on the subsystem consisting of codes with bounded maximal path length— since in one branch there is no information available whether other branches already ended.*

Remark 3.2.3.6 (Uniformization of DS-codes with explicitly given bounded branch length
If a DS*-code has bounded path length, it is obviously possible to apply the padding procedure in all branches, prolonging all branches to the given bound for the branch length. Translating this code to the $\sigma\delta$-presentation of* DS *(see Definition 3.2.3.2) gives a code with uniform structure. Translating the latter code to a* UF*-code with isomorphic semantics is then just a matter of reversing the nesting (see for example Example 5.3.0.1).*

3.2.3.4 A Categorical Explanation of Decoding

From exhibiting $\mathsf{DS}\ D\ \_$ as free monad, we can obtain even more insight into this system. As we will see now, one can infer the definition of decoding from the universal property of the free monad: the type $\mathsf{DS}\ D\ E$ is defined by induction and as such it is initial algebra for a functor — here $F(X) = E + \mathsf{Fam}\ D\ X$; thus to obtain a map $f : \mathsf{DS}\ D\ E \to \mathsf{Fam}D \to \mathsf{Fam}E$ it is sufficient to give an algebra structure on the (large) set $\mathsf{Fam}D \to \mathsf{Fam}E$. Now, there is a canonical map

$$\begin{aligned}
&\rho : \mathsf{CFam}\ D\ E \to \mathsf{Fam}D \to \mathsf{Fam}E\\
&\rho(X, h)(U, T) = (\llbracket\ X\ \rrbracket\ U, h \circ \llbracket\ X\ \rrbracket_1 T)
\end{aligned}$$

which can readily be extended to an algebra α of the desired form fitting in

$$\begin{tikzcd}
E + \mathsf{Fam}\ D\ (\mathsf{DS}\ D\ E) \arrow[r, "in"] \arrow[d] & \mathsf{DS}\ D\ E \arrow[d, "\llbracket\ \rrbracket"] \\
E + \mathsf{Fam}\ D\ E \arrow[r, "\alpha"] & (\mathsf{Fam}D \to \mathsf{Fam}E)
\end{tikzcd}$$

given by the coproduct map $\alpha = [e \mapsto ((U, T) \mapsto (1, * \mapsto e)),\ \rho)]$.

3.3 DS′

An equivalent system of codes for induction-recursion which will be important for the other system we discuss in later sections Chapter 5 Chapter 6 was defined by Dybjer-Setzer in [38]: DS' is a two-level system whose parts SP and Arg form for every $D : \mathsf{Set}_1$ a large container[11] $(\mathsf{SP}\ D, \mathsf{Arg}) : \mathsf{Fam}(\mathsf{Set}_1)$ where the large sets $\mathsf{SP}\ D$ of *codes for strictly-positive functors* is defined by the following constructors:

$$\begin{aligned}
&\mathsf{nil} : \mathsf{SP}\ D\\
&\mathsf{ni} : (A : \mathsf{Set}) \to (A \to \mathsf{SP}D) \to \mathsf{SP}\ D\\
&\mathsf{ia} : (A : \mathsf{Set}) \to ((A \to D) \to \mathsf{SP}D) \to \mathsf{SP}\ D
\end{aligned}$$

where the name 'ni' stands for "non-inductive [arguments]", and 'ia' stands for"inductive arguments" for similar reasons as explained in regard to the constructors of $\mathsf{DS}\ D\ D$ (see Remark 3.2.1.1) since these constructors play similar roles. The constructors have the same shape as those for $\mathsf{DS}\ D\ D$ except the base case.

[11]Here we define $\mathsf{Fam}(\mathsf{Set}_1) :- \Sigma_{X:\mathsf{Set}_2}(X \to \mathsf{Set}_1)$ Definition 1.2.4.1. Dybjer-Setzer do not use "container language" anywhere in their discussion.

The second component of the system is given by a function $\mathsf{Arg}\ D : \mathsf{SP}\ D \to \mathsf{Set}_1$ defined by recursion on $\mathsf{SP}\ D$.

$$\begin{aligned}&\mathsf{Arg\ nil} \coloneqq 1\\&\mathsf{Arg\ ni}\ A\ f \coloneqq \Sigma_{a:A}\mathsf{Arg}\ f(a)\\&\mathsf{Arg\ ia}\ A\ g \coloneqq \Sigma_{h:A\to D}\mathsf{Arg}\ f(h)\ .\end{aligned}$$

such that both parts form a large container $(\mathsf{SP}\ D, \mathsf{Arg}\ D) : \Sigma_{R:\mathsf{Set}_1} R \to \mathsf{Set}_1$ such that we can define $\mathsf{DS}'\ D\ E = [\![\ (\mathsf{SP}\ D, \mathsf{Arg}\ D)\ ]\!] E$ by container evaluation; in particular a code $(\phi, g) : \mathsf{DS}'\ D\ E$ has now two components.

The decoding of codes is organized in two components

$$\begin{aligned}&\mathsf{arg} : \mathsf{SP}\ D \to \mathsf{Fam}(D) \to \mathsf{Set}\\&\mathsf{arg}\ (\mathsf{nil})(U,T) \coloneqq 1\\&\mathsf{arg}\ (\mathsf{ni}\ A\ f)(U,T) \coloneqq \Sigma_{a:A}\mathsf{arg}(f(a))(U,T)\\&\mathsf{arg}\ (\mathsf{ia}\ A\ g)(U,T) \coloneqq \Sigma_{h:A\to U}\mathsf{arg}(g(T\circ g))(U,T)\end{aligned}$$

$$\begin{aligned}&\mathsf{map} : (\phi : \mathsf{SP}\ D) \to \mathsf{Fam}(D) \to \mathsf{Arg}(\phi)\\&\mathsf{map}\ (\mathsf{nil})\ (U,T) * \coloneqq *\\&\mathsf{map}\ (\mathsf{ni}\ A\ f)(U,T)\ (a,b) \coloneqq (a, \mathsf{map}(fa)(U,T)b)\\&\mathsf{map}\ (\mathsf{ia}\ A\ f)(U,T)(f,b)\ \coloneqq (T\circ h, \mathsf{map}(f(T\circ h))(U,T)b)\end{aligned}$$

This means we[12] obtain an operation from container evaluation in a large set $E : \mathsf{Set}_1$ by pairs $(\phi, g) : [\![\ (\mathsf{SP}\ D, \mathsf{Arg}\ D)\ ]\!] E$ each of which defines a function

$$\begin{aligned}&[\![\ (\phi, g)\ ]\!] : \mathsf{Fam}(D) \to \mathsf{Fam}(E)\\&(U,T) \mapsto (\mathsf{arg}(\phi)(U,T), g \circ \mathsf{map}(\phi)(U,T))\end{aligned}$$

which —as the authors show in the following paper [40]— is equivalent to the decoding function of the system $\mathsf{DS}\ D\ E$. In [38] however develop the the theory DS' of induction-recursion purely in terms of $(\mathsf{SP}, \mathsf{Arg})$, i.e. they give (for every $D : \mathsf{Set}_1$, every phi : $\mathsf{SP}\ D$, and every map $d : \mathsf{Arg}(\phi) \to D$) introduction and elimination rules for a pair $(\mathsf{U}(\phi, d), \mathsf{T}(\phi, d) : \mathsf{U}(\phi, d) \to D)$ such that

$$\begin{tikzcd}\mathsf{arg}(\phi)(\mathsf{U}(\phi,d), \mathsf{T}(\phi,d)) \arrow[r, "intro"] \arrow[d, "map(\phi))(\mathsf{U}(\phi,d),\mathsf{T}(\phi,d))"'] & U(\phi,d) \arrow[d, "\mathsf{T}(\phi,d)"] \\ \mathsf{Arg}(\phi) \arrow[r, "d"] & D\end{tikzcd}$$

[12]Dybjer-Setzer considered only the case $D = E$.

The above diagram explains (see [40, §5.1]) the terminology of induction-recursion as a *reflection principle* (compare Remark 1.1.0.15): the formalism reflects the operation d (on D) into U. An illustrative example is the case of closure of the Tarski universe under Σ types in which d becomes $\Sigma : (A : \mathsf{Set})(A \to \mathsf{Set}) \to \mathsf{Set}$, and *intro* becomes $\hat{\Sigma} : (u : \mathsf{U})(\mathsf{T}u \to \mathsf{U}) \to \mathsf{U}$. This special case is also the instance where reflection principles originally have been studied: Gödel advertised a justification strategy called *intrinsic justification* for large cardinal axioms in set theory essentially saying that we can accept an axiom positing a set having a number of properties (like being closed under some operations) if we believed that Set has these properties in first place (see [79] for a technical as well as philosophical discussion of reflection principles).

Remark 3.3.0.1. *It is possible to code the two constructors* ni *and* ia *into a single one like in the case of Definition 3.2.3.2.*

Remark 3.3.0.2 (Elimination). *Dybjer-Setzer define of course elimination for the system* DS' *which we omit here since the system is equivalent to* DS *as we will see in the next subsection.*

Remark 3.3.0.3 (Functorial aspects of DS'). DS' *is functorial in its second arument which follows easily from the fact that for* $E : \mathsf{Set}_1$*, the system* $\mathsf{DS}'\ D\ E$ *is by definition the value* $[\![\ (\mathsf{SP}\ D, \mathsf{Arg}\ D)\ ]\!]\ E$ *of a container.*

One can moreover show that DS' *is a monad in its second argument.*

3.3.1 Equivalence Between DS and DS′

Remark 3.3.1.1. *There are translations between the systems* DS' *and* DS*. For all* $D\ E : \mathsf{Set}_1$ *we have:*

$$\begin{aligned}
&\sharp\ :\ \mathsf{DS}\,D\ E\ \to\ \mathsf{DS}'\,D\ E\\
&\sharp\,(\iota\ e)\ =\ (1\ ,\ \lambda\,x\ \to\ e)\\
&\sharp\,(\sigma\ A\ f)\ =\ (\mathsf{ni}\,A\ (\lambda\,a\ \to\ (\mathsf{proj}_1\ (\sharp\ (f\ a))))\ ,\ \lambda\ (a\ ,\ x)\ \to\ (\mathsf{proj}_2\ (\sharp\ (f\ a)))\ x)\\
&\sharp\,(\delta\ A\ f)\ =\ (\mathsf{ia}\,A\ (\lambda\,a\ \to\ (\mathsf{proj}_1\ (\sharp\ (f\ a))))\ ,\ \lambda\ (a\ ,\ x)\ \to\ (\mathsf{proj}_2\ (\sharp\ (f\ a)))\ x)
\end{aligned}$$

$$\begin{aligned}
&\flat\ :\ \mathsf{DS}'\,D\ E\ \to\ \mathsf{DS}\,D\ E\\
&\flat\,(\mathsf{nil}\ ,\ f)\ =\ \iota\ (\ f\ \_)\\
&\flat\,(\mathsf{ni}\,A\ h\ ,\ f)\ =\ \sigma\ A\ \lambda\,a\ \to\ \flat(\ h\ a\ ,\ \lambda\,x\ \to\ f\ (a\ ,\ x))\\
&\flat\,(\mathsf{ia}\,A\ h\ ,\ f)\ =\ \delta\ A\ \lambda\,a\ \to\ \flat(\ h\ a\ ,\ \lambda\,x\ \to\ f\ (a\ ,\ x))
\end{aligned}$$

Moreover one can show roundtrips. A more detailed discussion of the equivalence can be found in [40]

Chapter 4

Characterization of Compositionality for DS

Motivated by composibility for inductive definitions (see Section 2.2.4.6), the main interest of this thesis will be whether a composition operation can be defined for a version of IR definitions. In this chapter we try to answer this question by attempting to adapt the composition formula for inductive definitions to DS. In the course we will find a new equivalent characterization of compositionality for DS codes in terms of powers of codes by sets. A bit more generally than adding powers, we will define an extesnion $\mathsf{DS} + \pi$ (see Section 4.2) of the axiomatization DS by a constructor for dependent products of codes and present a class of examples (see Section 4.2.1) of codes in this new system that are not directly expressible in plain DS.

4.1 Characterization of Compositionality of DS-Codes in Terms of Powers

Given DS-codes $c : \mathsf{DS}\ C\ D$ and $d : \mathsf{DS}\ D\ E$, is there a code $d \bullet c : \mathsf{DS}\ C\ E$ such that

$$\llbracket\, d \bullet c \,\rrbracket(U,T) \cong \llbracket\, d \,\rrbracket(\llbracket\, c \,\rrbracket(U,T))\ ?$$

One observes that it is easy to define postcomposition of any code by a ι or a σ code: the functor $\llbracket\, \iota\ e \,\rrbracket$ ignores its argument, hence so must $\llbracket\, (\iota\ e) \bullet c \,\rrbracket$, and for σ codes, we can just proceed structurally.

The δ case, however, requires more thought. To exercise this thought we first recall the reference case of composition for inductive definitions (see Section 2.2.4.6 and Corollary 2.2.4.13) which —in the convenient incarnation of the type Ind of inductive definitions as $\mathsf{Fam}(\mathsf{Set})$[1]— was given by

$$\begin{aligned}
\_ \circ \_ &: \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Fam}(\mathsf{Set}) \to \mathsf{Fam}(\mathsf{Set}) \\
(S,P) \circ (U,T) &:= (\Sigma_{s:S}(P(s) \to U), \lambda(s,f) \to \Sigma_{x:P(s)} T(f(x)) \\
&= (S,P) \ggg= e^{(U,T)}
\end{aligned}$$

where, to be explicit, the bind operation (see Lemma 1.3.0.6) is given by

$$\begin{aligned}
\_ \ggg= \_ &: \mathsf{Fam}(D) \to (D \to \mathsf{Fam}(E)) \to \mathsf{Fam}(E) \\
(U,T) \ggg= h &= ((\mu^{\mathsf{Fam}} \circ \mathsf{Fam}^{\to})(h))(U,T) \\
&= \mu^{\mathsf{Fam}}(U, h \circ T) \\
&= (\Sigma_{u:U} \mathsf{proj}_1(h \circ T)(u), \lambda(u,k) \mapsto \mathsf{proj}_2((h \circ T)u)(k)) \ .
\end{aligned}$$

and

$$\begin{aligned}
e^{(U,T)} &: \mathsf{Set} \to \mathsf{Fam}(\mathsf{Set}) \\
e^{(U,T)}(X) &:= (X \to U, \lambda h \to \Sigma_{x:X} T(hx))
\end{aligned}$$

was the appropriation —via the isomorphism

$$(\mathsf{Set}/U, (X \to U, \lambda h \to (\mathsf{proj}_2 : \Sigma_{x:X} T(hx)))) \simeq (X \to \mathsf{Set}, (X \to U, \lambda h \to T \circ h)) \ .$$

— of the power operation

$$\begin{aligned}
\_ \longrightarrow_{\mathsf{Fam}} \_ &: \big(S : \mathsf{Set}\big) \to \mathsf{Fam}\ D \to \mathsf{Fam}\ (S \to D) \\
S \longrightarrow_{\mathsf{Fam}} (A,P) &= (S \to A, g \mapsto P \circ g)
\end{aligned}$$

for families taken in the category i/Set_1 (see Remark 1.3.0.8) because of $\mathsf{Fam}(D)$'s poverty of limits in cases other than $D = \mathsf{Set}$.

Returning to the problem of precomposing by a δ code we first consider the action on index sets of families and find:

[1]To be precise, the category Ind is equivalent to $\mathsf{Fam}(\mathsf{Set}^{op})$, but the op-ing is relevant only for morphisms in the latter category and since the latter play no role in the present discussion, we suppress the *op* in notation.

$$\begin{aligned}
[\![\delta\ A\ F]\!]_0([\![c]\!]_0 Z) &= \big(\Sigma g : A \to [\![c]\!]_0 Z\big)\big([\![F\big([\![c]\!]_1(Z) \circ g\big)]\!]_0([\![c]\!] Z)\big) \\
&= \Big(\big(A \longrightarrow_{\mathsf{Fam}} [\![c]\!] Z\big) \gg\!\!=_{\mathsf{Fam}} \big(g \mapsto [\![F\big([\![c]\!]_1(Z) \circ g\big)]\!]_0([\![c]\!] Z)\big)\Big)_0
\end{aligned}$$

This suggests that to define $(\delta\ A\ F) \bullet c$ we need to internalize $\gg\!\!=_{\mathsf{Fam}}$ and $\longrightarrow_{\mathsf{Fam}}$ in the system DS. The first is readily achievable, because $\mathsf{DS}\ C$ is a monad (see Section 3.2.3.2) (and a fortiori a functor), too, and the implied bind operation is moreover compatible with decoding:

Proposition 4.1.0.1. *The bind operation on* DS

$$\_ \gg\!\!= \_ : \mathsf{DS}\ C\ D\ \to (D\ \to \mathsf{DS}\ C\ E)\ \to \mathsf{DS}\ C\ E$$

defined by $c\ \gg\!\!=\ h := \mu(\mathsf{DS}\ h\ c)$ *satisfies*

$$[\![c \gg\!\!= g]\!]\, Z \cong [\![c]\!]\, Z \gg\!\!=_{\mathsf{Fam}} (e \mapsto [\![g\, e]\!]\, Z)$$

for every $Z : \mathsf{Fam}\ C$, $c : \mathsf{DS}\ C\ D$ *and* $g : D \to \mathsf{DS}\ C\ E$. □

Thus it would remain to define powers of codes. Here, however, problems arise if we try to define $S \longrightarrow c$ by induction on c: to apply the inductive hypothesis on $f\ a$ in the following S-fold power of a σ code

$$S \to [\![\sigma\ A\ f]\!]_0\, Z = S \to \big(\Sigma a : A\big)([\![f\ a]\!]_0\, Z) \cong \big(\Sigma g : S \to A\big)\big(\big(x : S\big) \to [\![f\ (g\ x)]\!]_0\, Z\big) \tag{4.1}$$

we would need to generalize our construction to *dependent products* $\big(x : S\big) \to c(x)$ where $c : S \to \mathsf{DS}\ D\ E$. But, if we do so, we can no longer do an induction on c, and we are stuck (compare this to Definition 6.5.1.3 and to Remark 5.1.0.2 where solutions to this problem are suggested). Even worse, any definition of composition necessarily involves powers:

Theorem 4.1.0.2. *There is a composition operator for* DS *if and only if there is a power operator for* DS*. Here, by composition and power operators we mean terms*

$$\begin{aligned}
\_ \bullet \_ &: \mathsf{DS}\ D\ E \to \mathsf{DS}\ C\ D \to \mathsf{DS}\ C\ E \\
\_ \longrightarrow \_ &: \big(S : \mathsf{Set}\big) \to \mathsf{DS}\ D\ E \to \mathsf{DS}\ D\ (S \to E)
\end{aligned}$$

respectively such that $[\![c \bullet d]\!]\, Z \cong [\![c]\!]([\![d]\!]\, Z)$ *and* $[\![S \longrightarrow c]\!]\, Z \cong (S \longrightarrow_{\mathsf{Fam}} [\![c]\!]\, Z)$*.*

Proof. Given $\_ \longrightarrow \_$, we can define $\_ \bullet \_$ by

$$\begin{aligned}
(\iota\ e) \bullet d &= \iota\ e \\
(\sigma\ A\ f) \bullet d &= \sigma\ A\ (a \mapsto (f\, a) \bullet d) \\
(\delta\ A\ F) \bullet d &= (A \longrightarrow d) \gg\!\!= (g \mapsto (F\, g) \bullet d)
\end{aligned}$$

using Proposition 4.1.0.1. Conversely, $A \longrightarrow c := (\delta\ A\ (h \mapsto \iota\ h)) \bullet c$ is a power operator. □

Remark 4.1.0.3. *A desired second part of the previous theorem would show that powers are indeed not definable in* DS*. Unfortunately, a proof of this statement has remained elusive so far. Reasons why we believe that this statement is nevertheless true are —besides the failed attempt to define powers— the fact that the universal property of* $\longrightarrow$ *is negative while the constructors of* DS *decode to operations with positive universal properties.*

Both systems of compositional IR we will define do have powers for codes: in UF they are definable by Remark 5.9.1.4, and the system PN (see Chapter 6) is defined in a way that can accommodate a constructor for powers directly.

As expected, for small DS the power operator is definable.

Remark 4.1.0.4 (For small DS, powers are definable). *In [58] it has been shown that* small DS, *i.e. the system of all* $\mathsf{DS}\ D\ E$ *where* $D\ E : \mathsf{Set}$ *are small sets is equivalent to the system of polynomial functors between the slices* $\mathsf{Set}/D \to \mathsf{Set}/E$*. Let* $c_{(S,T)}$ *denote the code representing a container* $(S,T) : \mathsf{Cont}$ *(see Definition 1.2.4.1), and let* $A : \mathsf{Set}$*, then*

$$\begin{aligned}
[\![\ A \longrightarrow c_{(S,T)}\ ]\!]X &= A \to \Sigma_{s:S}(T(s) \to X) \\
&\simeq \Sigma_{f:A\to S}\Pi_{a:A}(T(fa) \to X) \\
&\simeq \Sigma_{f:A\to S}((\Sigma_{a:A}T(fa)) \to X) \\
&= [\![\ c_{(A\to S,\lambda f.\Sigma_{a:A}T(fa))}\ ]\!]X
\end{aligned}$$

where the first isomorphism is due to the axiom of intensional choice and the second one is currying.

It is perhaps instructive to go through the previous construction of powers for small DS in case of (unrestricted) DS to see where problems arise:

Remark 4.1.0.5 (For general (large) DS this approach fails). *Let us try to raise a code* δBF *to the power* A *using the Ansatz above:*

$$\begin{aligned}
[\![\ A \longrightarrow \delta BF\ ]\!]_0(U,T) &:= A \to [\![\ \delta BF\ ]\!]_0(U,T) \\
&= A \to \Sigma_{k:B\to U}[\![\ F(T\circ k)\ ]\!]_0(U,T) \\
&\simeq \Sigma_{k':A\to B\to U}(\Pi_{a:A}[\![\ F(T\circ(k'a))\ ]\!]_0(U,T))
\end{aligned}$$

The only[2] *way to potentially encode this is*

$$[\![\ \delta A\times BF'\ ]\!]_0(U,T) \simeq \Sigma_{k'':A\times B\to U}[\![\ F'(T\circ k'')\ ]\!]_0(U,T)$$

[2]The "only" way means here the only way to encode this strictly and not up to an arbitrary isomorphism; the isomorphisms indicated by $\simeq$ in the previous and the following formula refer only to the canonical isomrphisms given by the axiom of intensional choice respectively currying.

where $[\![F'(T \circ k'')]\!]_0(U,T) = \Pi_{a:A}[\![F(T \circ (k'a))]\!]_0(U,T)$ *which is already not nice since we would need a dependent* π *(which we will revisit a bit further below Section 4.2) even if we only want to define powers, i.e.* $\cdots = [\![\pi A f']\!]_0(U,T) = \Pi_{a:A}[\![f'a]\!]_0(U,T)$ *with* $f'a = F(T \circ (k'a))$*. On the other hand does the prefix of the attempted code give us*

$$[\![\delta(A \times B)\lambda X.\pi A\lambda a.F(X)]\!]_0 = \Sigma_{k:A\times B\to U}\Pi_{a:A}[\![F(T \circ k)]\!]_0(U,T)$$

and there is no possibility to effect on the level of codes that k *be replaced by* $k'a$ *in the last step since* π *does not take dependent arguments.*

For clarity an emphasis we mention that the previous remark does of course *not* show that it is impossible to encode powers in some other way by supplying a DS code with isomorphic semantics.

Two natural options suggest themselves as solutions: (i) restrict codes to ensure that no dependency arises in the definition of powers; (ii) devise a system with dependent products of codes. In the next two chapters, we investigate new systems of codes for both of these solutions.

4.2 DS+π

A bit more generally (at least prima facie so) than equipping DS with powers, one can extend the system DS also by a constructor

$$\pi : (A : \mathsf{Set})(A \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ (A \to E)$$

for dependent products of codes having semantics

$$[\![\pi\ A\ f]\!](UT) := (\Pi_{a:A}[\![fa]\!](U,T), \lambda f \lambda a.[\![fa]\!]_1(U,T)) \ .$$

As we explained in the last sentence above Theorem 4.1.0.2, also simply adding this constructor (instead of a constructor for powers) to the system DS does not facilitate composition for the system thus supplemented since the induction cannot deal with the dependency. We will return to the project of adding such a constructor more systematically in Chapter 6.

For the present section, the main application we have in mind for this new system is a set of codes whose decoding relates to that of (plain) DS codes like power series relate to polynomials.

4.2.1 Examples of DS+π Codes

We present a class of codes whose members do not (prima facie) reduce to DS codes without π since their decoding is given by infinitely (in the example we give, ω_0-many) iterated Σ types. For the construction of such codes, it is helpful to notice that it is always possible to replace an iterated σ code, i.e. a code of the form $\sigma A \lambda a.(\sigma B(a) f(a))$, by a code $\sigma(\Sigma_{a:A} B(a))\lambda(a,b).f(a)(b)$ which has isomorphic semantics, this fails for size reasons in case of δ codes.

The codes we shall define all have the form $Z\ A := \pi\mathbb{N}^+(z\ A)$ where $z\ A$ is a sequence f codes defined by recursion on $\mathbb{N}$, $\mathbb{N}^+$ are the positive natural numbers (i.e. 1, 2, ...) and A is an arbitrary set.

Example 4.2.1.1.

$$\begin{aligned}
&\pi\ \mathbb{N}\ (\lambda\ n \to\ z\ A\ (n+1)\ ) : \mathsf{DS\ Set\ Set}\\
&z\ :\ (A : \mathsf{Set}) \to \mathbb{N} \to \mathsf{DS\ Set\ Set}\\
&z\ A\ 0\ := \delta\ 0\ (\lambda\_ \to \sigma\ 1(\lambda\ \_ \to\ \iota\ A))\\
&z\ A\ (n+1)\ :=\ \delta\ A\ (\lambda\ X\ \to\ \sigma\ A\ (\lambda\ a\ z\ (X\ a)\ n))
\end{aligned}$$

The indexing set of the semantics for z in case $n+2$ can be computed as

$$[\![\ z\ A\ (n+2)\ ]\!]_0\ (U,T) = \Sigma_{k:A\to U}\Sigma_{a:A}\Sigma_{l:T(k\ a)\to U}\Sigma_{b:T(ka)}[\![\ z\ (T\ (l\ b))\ n\ ]\!]_0\ (U,T)$$

and thus

$$\begin{aligned}
&[\![\ \pi\ \mathbb{N}\ (\lambda n \to\ z\ A\ (n+1))\ ]\!] = \Pi_{n:\mathbb{N}}\Sigma_{k:A\to U}\Sigma_{a:A}[\![\ z\ (T(k\ a))(n+1)\ ]\!]_0(U,T)\\
&\simeq \Sigma_{\phi:\mathbb{N}\to A\to U}\Pi_{n:\mathbb{N}}\Sigma_{a:A}[\![\ z\ (T(\phi\ n)\ a)(n+1)\ ]\!]_=(U,T)\\
&\simeq \Sigma_{\phi:\mathbb{N}\to A\to U}\Sigma_{\psi:\mathbb{N}\to A}\Pi_{n:\mathbb{N}}[\![\ z\ (T(\phi\ n)(\psi\ n))(n+1)\ ]\!]_0(U,T)\ .
\end{aligned}$$

Here we used twice the intensional “axiom” of choice where the quotation marks indicate that this statement of derivable in MLTT:

$$\Pi_{x:X}\Sigma_{y:Y}C(x,y) \simeq \Sigma_{\phi:X\to Y}\Pi_{x:X}C(x,\phi\ x)\ .$$

This example has siblings sharing the same organization of defining a transfinite sequence of codes and packaging this sequence into a single code with a final π. The code we just presented is the same as in Example 3.2.1.14 of an unbounded DS-code except that the final σ is replaced by a final π.

The following is a simpler variant:

Example 4.2.1.2.

$$\begin{aligned}
&\pi\ \mathbb{N}\ (\lambda\ n.\ z\ A\ n)\ \ : \mathsf{DS\ Set\ Set}\\
&z\ :\ (A : \mathsf{Set}) \to \mathbb{N} \to \mathsf{DS\ Set\ Set}\\
&z\ A\ 0\ := \iota\ A\\
&z\ A\ (n+1)\ :=\ \delta\ A\ (\lambda\ X\ \to\ z\ (\Sigma\ A\ X)\ n)
\end{aligned}$$

A version where the recursive function takes an infinite sequence (stream) of sets as first argument is.

Example 4.2.1.3.

$$\begin{aligned}
&\pi\ \mathbb{N}\ \lambda\ n \to\ z\ A\ n\ : \mathsf{DS}\ \mathsf{Set}\ \mathsf{Set}\\
&z\ :\ (A : \mathbb{N} \to \mathsf{Set}) \to \mathbb{N} \to \mathsf{DS}\ \mathsf{Set}\ (\mathbb{N} \to \mathsf{Set})\\
&z\ A\ 0\ := \iota\ A\\
&z\ A\ (n+1)\ :=\ \delta\ (\Sigma\ \mathbb{N}\ A)\ \lambda\ X \to \sigma\ (\mathbb{N} \to (\Sigma\ \mathbb{N}\ A))(\lambda\ a \to z\ (X \circ a)\ n)
\end{aligned}$$

The codes we just presented are intended to provide examples of codes that lie in $\mathsf{DS} + \pi$ but not in DS. The decoding of Example 4.2.1.1 indicates that we have a "δ-code of infinite path length" which cannot really be a DS-code since $\mathsf{DS}\ D\ E$ —as an inductively defined type— is a wellfounded set and its terms cannot be defined by infinitely many constructors. Thus a possible proof that $Z\ A$ cannot correspond to any DS-code could e.g. show that there is no DS-code having as semantics the decoding of $Z\ A$. This would be implied by a proof that a code containing infinitely many adjacent δ-constructors cannot be compressed into a code with only finitely many constructors. One step in the direction of such a proof is the following counterexample showing that at least in the case where in all branches there are two subsequent δ-codes, there is no code replacing the two δ-codes everywhere by only one.

Remark 4.2.1.4. *We give a counter-example of a code $\delta A\lambda X.\delta B(X)\lambda Y.\iota d(X,Y)$ for which there is no code of the form $\delta S\lambda X.\sigma C(X)\lambda Y.\iota d'(X,Y)$, i.e. we show that*

$$\exists(D : \mathsf{Set}_1, A : \mathsf{Set}, B : (A \to D) \to \mathsf{Set}) \tag{4.2}$$
$$\neg\exists(S : \mathsf{Set}, C : (S \to D) \to \mathsf{Set}) \tag{4.3}$$
$$\forall(U : \mathsf{Set}, T : U \to D) \tag{4.4}$$
$$\Sigma(k : A \to U)(B(T \circ k) \to U) \simeq \Sigma(l : S \to U)C(T \circ l) \tag{4.5}$$

Indeed, we have to show that no pair $(S : \mathsf{Set}, C : (S \to D) \to \mathsf{Set})$ can satisfy $\Sigma(k : A \to U)(B(T \circ k) \to U) \simeq \Sigma(l : S \to U)C(T \circ l)$ for all pairs $(U : \mathsf{Set}, T : U \to D)$ for the $(D : \mathsf{Set}_1, A : \mathsf{Set}, B : (A \to D) \to \mathsf{Set})$ defined below.

We define $D := \mathsf{Set}, A := 2, Bh := h0$ for $h : A \to D$.

Instantiating (4) in (U,T) with $(1, T_0)$ where $T_0 := \lambda\_.2$ gives $1^2 \times 1^2 \simeq 1^S \times C(\lambda\_.2)$ which implies that $C(\lambda\_.2) \simeq 1$.

Instantiating (4) in (U,T) with (U, T_0) gives $U^2 \times U^2 \simeq U^S \times C(\lambda\_.2)$ which implies $S = 4$ by $C(\lambda\_.2) \simeq 1$ by the previous result.

Instantiating (4) in (U,T) with $(1, T_1)$ where $T_1 := \lambda\_.1$ gives $1^2 \times 1^1 \simeq 1^4 \times C(\lambda\_.1)$ which implies $C(\lambda\_.1) \simeq 1$.

Instantiating (4) in (U,T) with $(2, T_1)$ gives $2^2 \times 2^1 \simeq 2^4 \times 1$ which is not true and gives the desired contradiction concluding the proof. □

4.3 Conclusion and Outlook

Since we conclude with the end of this chapter our review-, and analysis DS-codes, a few sentences of summary and outlook are in place.

As we indicated in Remark 4.1.0.3 and Section 4.1, it is still an open problem to decide whether composition —or by Theorem 4.1.0.2 equivalently powering of codes— is possible in DS. Among the possibilities to decide this problem would be the more concrete approach to find a counterexample in the style of Section 4.2.1, i.e. a proof that there is indeed no DS code with isomorphic semantics, or more abstract approaches like showing that the functors $\mathsf{DS}\ D\ \_$ and $(\mathsf{DS} + \pi)\ D\ \_$ have different properties together with arguments explaining why this implies that that there are $(\mathsf{DS} + \pi)$ codes with semantics different from that of any DS code, or, alternatively, a justification why from a constructivist viewpoint difference of the functors is significant enough to separate the systems.

To obtain further insight into the problem of compositionality of IR codes, and alternative or different axiomatizations of IR in general, we will present in Chapter 5 and Chapter 6 two new axiomatizations for which we can show that they enjoy compositionality.

Chapter 5

Uniform Codes for Induction-Recursion

This section presents our first new system for induction-recursion with a native composition operation. The system UF of *uniform codes* can be regarded a subsystem of DS (Proposition 5.7.0.1).

5.1 Motivation: Uniformity-, and Definability of DS Codes

Informally, a uniform code is a DS code where, for every constructor in a term, all immediate subterms have the same root-constructor. To recognize this as a sound intuition, we recall that DS D E as an inductive definition is in particular equivalent to a W-type, and as such codes can be regarded as trees. More visually, a non-uniform DS code can look like

$$\begin{tikzcd}
\iota e_1 \arrow[dr] & \iota e_2 \arrow[d] & & \iota e_4 \arrow[d] \\
& \sigma A^1 f^1 \arrow[dr] & \iota e_3 \arrow[d] & \sigma A^2 f^2 \arrow[dl] \\
& & \sigma A f \arrow[d] & \\
& & {} &
\end{tikzcd}$$

whereas a uniform DS code has on every level the same kind of code and all branches have the same length.

$$\begin{tikzcd}
\iota e_1 \arrow[dr] & \iota e_2 \arrow[d] & \iota e' \arrow[d] & \iota e_4 \arrow[d] \\
& \sigma A^1 f^1 \arrow[dr] & \sigma A' f' \arrow[d] & \sigma A^2 f^2 \arrow[dl] \\
& & \sigma A f \arrow[d] & \\
& & {} &
\end{tikzcd}$$

Remark 5.1.0.1 (Uniform codes as definable DS Codes). *The above informal characterization of uniform codes tells us that the only thing that can prevent a code from being uniform is that the functions taken as second argument of σ and δ are functions whose values may be codes starting with arbitrary constructors, and in particular different arguments may be sent to codes staring with different constructors, such that in the presence of coproducts —allowing to define functions by case distinction such as $\sigma\ A\ f +_{\mathsf{DS}} \delta\ B\ G = \sigma\ 2\ (0_2 \mapsto \sigma\ A\ f; 1_2 \mapsto \delta\ B\ G)$ where one subcode is a σ code while the other is a δ one— codes can be non-uniform. But this suggests that codes being expressions over the alphabet consisting of the letters ι, σ, δ, lambda expressions, and variable symbols (allowed to be bound by the lambda expressions) ranging over sets —i.e.* definable[1] DS codes— *are uniform* DS *codes in the above sense.*

Remark 5.1.0.2 (Why uniformity intuitively implies compositionality). *A partial explanation why we can expect that uniformity or definability of codes implies compositionality of codes can be observed from Eq.* (4.1) *where we encountered the dilemma that —attempting to inductively define powers upon* DS *codes— on the one hand would we be necessitated to more generally define dependent products of codes to be able to call the inductive hypothesis, but on the other hand would deprive us the presence of the family $(x : S) \to c(x)$ of codes in this dependent product in place of a single code, of the possibility to do induction on this code c. This is different if for all $x : S$ the $c(x)$ start with the same constructor as ensured by uniformity.*

[1]As a warning in regard to this terminology, we should mention that there is a notion 'definability' in (set theoretic) model theory. That notion has the same intention that something is expressible within the formal language under discussion. Our notion of 'definabilty' is however of course different because we are working in a different setup.

5.2 UF Codes and Their Decoding

Even though the informal motivation for uniformity of codes given in the previous subsection might convince to some extent, it does not give a constructive recipe how to inductively define a (large) set containing only such uniform codes. The notion of definability is more promising in this regard since it proceeds structurally. The axiomatization UF of uniform codes presented below should thus —as explained in a bit more detail in Section 5.2— rather be seen as an axiomatization of definable DS codes; this is to be understood as saying that *all* UF codes can be regarded as (and as we shall see: translated to) definable DS codes.

Definition 5.2.0.1 (The (large) set of UF codes and their decoding). *For all $D, E : \mathsf{Set}_1$, we define the (large) set $\mathsf{UF}\ D\ E : \mathsf{Set}_1$ of* uniform codes for induction-recursion *by $\mathsf{UF}\ D\ E := \big(\Sigma c : \mathsf{Uni}\ D\big)(\mathsf{Info}\ c \to E)$ where the (large) family $(\mathsf{Uni}\ D : \mathsf{Set}_1, \mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1)$ is mutually defined in the following way:*

$$
\begin{aligned}
&\mathsf{Uni}\ D : \mathsf{Set}_1\\
&\iota_{\mathsf{UF}} : \mathsf{Uni}\ D\\
&\sigma_{\mathsf{UF}} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D\\
&\delta_{\mathsf{UF}} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Info}\ :\ \mathsf{Uni}\ D \to \mathsf{Set}_1\\
&\mathsf{Info}\ \iota_{\mathsf{UF}} = 1\\
&\mathsf{Info}\ (\sigma_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma)\\
&\mathsf{Info}\ (\delta_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma \to D)
\end{aligned}
$$

The decoding function is defined by:

$$
\begin{aligned}
&[\![\ ]\!] : \mathsf{UF}\ D\ E \to \mathsf{Fam}\ D \to \mathsf{Fam}\ E\\
&[\![\ c, \alpha\ ]\!]\ Z = \mathsf{Fam}(\alpha)\ ([\![\ c\ ]\!]_{\mathsf{Uni}}\ Z, [\![\ c\ ]\!]_{\mathsf{Info}}\ Z)
\end{aligned}
$$

$$
\begin{aligned}
&[\![\ ]\!]_{\mathsf{Uni}} : \mathsf{Uni}\ D \to \mathsf{Fam}\ D \to \mathsf{Set}\\
&[\![\ \iota_{\mathsf{UF}}\ ]\!]_{\mathsf{Uni}}\ (U, T) = 1\\
&[\![\ \sigma_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Uni}}\ (U, T) = \big(\Sigma x : [\![\ c\ ]\!]_{\mathsf{Uni}}\ (U, T)\big)(A([\![\ c\ ]\!]_{\mathsf{Info}}\ (U, T)\ x))\\
&[\![\ \delta_{\mathsf{UF}}\ c\ A\ ]\!]_{\mathsf{Uni}}\ (U, T) = \big(\Sigma x : [\![\ c\ ]\!]_{\mathsf{Uni}}\ (U, T)\big)(A([\![\ c\ ]\!]_{\mathsf{Info}}\ (U, T)\ x) \to U)
\end{aligned}
$$

$$
\begin{aligned}
&[\![\ ]\!]_{\mathsf{Info}} : \big(c : \mathsf{Uni}\ D\big) \to \big(Z : \mathsf{Fam}\ D\big) \to [\![\ c\ ]\!]_{\mathsf{Uni}}\ Z \to \mathsf{Info}\ c\\
&[\![\ \iota_{\mathsf{UF}}\ ]\!]_{\mathsf{Info}}\ (U, T)\ \star = \star\\
&[\![\ \sigma_{\mathsf{UF}}\ c\ S\ ]\!]_{\mathsf{Info}}\ (U, T)\ (x, a) = ([\![\ c\ ]\!]_{\mathsf{Info}}\ (U, T)\ x, a)\\
&[\![\ \delta_{\mathsf{UF}}\ c\ S\ ]\!]_{\mathsf{Info}}\ (U, T)\ (x, g) = ([\![\ c\ ]\!]_{\mathsf{Info}}\ (U, T)\ x, T \circ g)
\end{aligned}
$$

We give a prose explanation of Explanation of UF codes and their decoding: we defined a set $\mathsf{Uni}\ D : \mathsf{Set}_1$ determining the code shapes simultaneously with a function $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$ assigning to each code shape the information available for indexing code shapes depending on it in a uniform way. Uniformity is reached here by realizing Uni as a left-nested rather than a right-nested definition [104].

We can understand Uni as a set of 'contexts' where each uniform code is defined in a context of what has been defined earlier. To inductively incorporate "what has been defined earlier" we somehow need to extract the 'variables' available in an earlier stage of the context —this is accomplished by the function $\mathsf{Info} : \mathsf{Uni}\ D \to \mathsf{Set}_1$— and to allow the next stage of the context to consist of sets defined with these variables —this is done by the constructor(s) of type $\big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D$; here the plural (constructors) of two identical constructors is necessary because their 'internal decodings' via Info differ: compared to the reference system DS, $\mathsf{Info}\ c$ should (at least) cover the cases where as indexing object either a set A-, or the large set $(A \to D)$ of families of terms of D is chosen where a constructor σ_{UF} respectively δ_{UF} effects that *all* variables in $\sigma_{\mathsf{UF}}\ Q\ h$ respectively $\delta_{\mathsf{UF}}\ Q\ h$ available for the next stage are exclusively "of type σ" respectively "of type δ", and ι_{UF} means that there won't be any next stage; here the difference between σ_{UF} and δ_{UF} is steered via their different internal decoding.

So, we may understand the object $\mathsf{Info}\ Q$ as the '(large) set of formulas with variables drawn from the context Q', a function $h : \mathsf{Info}\ Q \to \mathsf{Set}$ as a family of sets where each fiber $h(\varphi)$ is the set defined by the formula φ and an argument pair $(Q, h : \mathsf{Info}\ Q \to \mathsf{Set})$ as the data we need to define a new context with prefix Q and appended by new sets $(h(\varphi))_\varphi$ of variable, and applying a constructor $\sigma_{\mathsf{UF}}\ Q\ h$ or $\delta_{\mathsf{UF}}\ Q\ h$ finally determines whether the context we form shall be "of type σ" or "of type δ". This confirms our understanding of uniform codes as definable DS codes (see Remark 5.1.0.1).

An alternative way of understanding the construction of uniform codes is to recognize it as a version of DS where the dependencies in the constructor arguments are inverted to the effect that later stages in tree corresponding the code cannot depend on earlier stages and thus can be fixed in a uniform way upfront (see also Remark 5.2.0.2 and Section 5.7). While this alternative explanation also convinces to some degree, it should not lead one to the false conclusion that it would not be possible to define an equivalent version of UF in "right nested" fashion (see Section 5.8).

Decoding of uniform codes $\mathsf{UF}\ D\ E$ is again given by functors $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$. The definition is very similar to the decoding of DS codes except that UF codes have two components. We use the same notation $[\![\ ]\!]$ for decoding a uniform code as for decoding a DS code; this convention is reasonable since we will give a semantics-preserving translation from DS to UF in Section 5.7.

A few immediate observation about the systems of IR we have seen so far are in place.

Remark 5.2.0.2 (Preliminary comparison between UF, DS and DS'). *Since* $\mathsf{UF}\ D\ \_$ *is defined as the action of a container [3], it is automatically functorial. This two-level presentation of codes* $(\mathsf{Uni}, \mathsf{Info})$ *is similar to* DS' *(see Section 3.3) in that both systems are (large) containers (see Definition 1.2.4.1). A difference is however that the family* $(\mathsf{Uni}, \mathsf{Info})$ *is defined by non-degenerate induction-recursion whereas* SP *can be defined* *without reference to* Arg *which was defined by simple recursion on* SP*.*

A further difference between UF *and* DS *is the chirality of the nestings of their inductive arguments:* Uni *is left-nested while* SP *as well as* DS *are right-nested, e.g.* $\sigma_{\mathsf{UF}} : (c : \mathsf{Uni}) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Set}$ *has* $(\mathsf{Info}\ c)$ *on the left in* $(\mathsf{Info}\ c \to \mathsf{Set})$ *whereas* DS *is right nested in this sense. One can however define* UF *right-nested as well (see Section 5.8). See also [104] for this sort of chiralities.*

Remark 5.2.0.3. *Like for* DS *codes (see Section 3.2.3.3) it is again possible to subsume* σ_{UF} *and* δ_{UF} *in one constructor:*

$$\begin{aligned}
&\mathsf{sd} : (c : \mathsf{Uni}) \to (\mathsf{Info}\ c \to \mathsf{Cont}) \to \mathsf{Uni} \\
&\mathsf{Info}\ \mathsf{sd}\ c\ A \ = \Sigma(\mathsf{Info}\ c)(\llbracket\ A\ \rrbracket\ D)\ \ .
\end{aligned}$$

5.3 Examples

Example 5.3.0.1 (W-types, again). *For the sake highlighting differences between* UF*-, and* DS*-codes, we return to the* W*-types of Example 3.2.1.8. A uniform code in* $\mathsf{UF}\ 1\ 1$ *representing the* W*-type* $\mathsf{W}\ (S,\ P)$ *is* $c_{\mathsf{W}\ (S,\ P),\mathsf{UF}} = \delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\star, s) \mapsto P(s)) :$ $\mathsf{Uni}\ 1$*, together with the terminal map* $\mathsf{Info}\ c_{\mathsf{W}\ (S,\ P),\mathsf{UF}} \to 1$*. If we compare this to the Dybjer-Setzer code from Example 3.2.1.8, we see that the order of the (non-base-case) constructors is reversed:*

$$\begin{aligned}
(\delta_{\mathsf{UF}}\ \big(\sigma_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\_ \mapsto S)\big)\ ((\star, s) \mapsto P(s))\ ,\ \_ \mapsto \star) &: \mathsf{UF}\ 1\ 1 \\
\sigma\, S\, (s \mapsto \delta\, P(s)\, (\_ \mapsto \iota\, \star)) &: \mathsf{DS}\ 1\ 1
\end{aligned}$$

Also this code can be promoted to a more interesting $\mathsf{UF}\ \mathsf{Set}\ \mathsf{Set}$ *code applying a given* T *everywhere in the tree. We get the same decoding as in Example 3.2.1.8 if we replace the trivial map* $(\_ \mapsto \star) : \mathsf{Info}\ c_{\mathsf{W}\ S\ P,\mathsf{UF}} \to 1$ *by the map* $((s, \star), Y) \mapsto \big((c : P(s)\big) \to Y\, x)$*.*

Decoding $c_{\mathsf{W}\ S\ P,\mathsf{UF}}$ *we see that*

$$\begin{aligned}
\llbracket\ c_{\mathsf{W}\ S\ P,\mathsf{UF}}\ \rrbracket_{\mathsf{Uni}}\,(U, T) &= \big(\Sigma(\star, s) : 1 \times S\big)(P(s) \to U) \\
\llbracket\ c_{\mathsf{W}\ S\ P,\mathsf{UF}}\ \rrbracket_{\mathsf{Info}}\,(U, T)((\star, s), Y) &= ((\star, s), T \circ Y)
\end{aligned}$$

where $\llbracket\ c_{\mathsf{W}\ S\ P,\mathsf{UF}}\ \rrbracket_{\mathsf{Uni}}\,(U, T)$ *is isomorphic to the domain of the* W*-type constructor* sup*, but this time nested the other way compared to the decoding of* $c_{\mathsf{W}\ S\ P}$ *in Example 3.2.1.8.*

Example 5.3.0.2. *A class of examples of DS codes whose members are* not *uniform in the intuitive sense motivated above are those obtained from Section 4.2.1 by replacing the leading* π *by a* σ*.*

To recover the expected main example of a universe closed under operators Example 3.2.1.9 in the system UF Example 5.5.0.1, we need to introduce coproducts of codes Section 5.4 first.

5.4 Coproducts of Uniform Codes

The coproduct $c +_{\mathsf{DS}} d :\equiv \sigma\ 2\ (0_2 \mapsto c; 1_2 \mapsto d)$ of two DS codes is not in general the embedding of a uniform code —even if c and d are— as c and d may still have different shapes and we can thus not generally immediately use the same construction to define coproducts of uniform codes we used for DS codes. However, if c and d do have the same shape, this construction still works. Our plan for constructing coproducts of uniform codes is then to find equivalent replacements of the summands, such that the new pair has a common shape, and then to use the standard coproduct construction upon these replacement codes.

5.4.1 Graded UF Codes

To facilitate the replacement procedure, we first introduce an $\mathbb{N}$-indexed variant $\mathsf{UF}^+\ D\ E\ n = \big(\Sigma c : \mathsf{Uni}^+\ D\ n\big)(\mathsf{Info}^+\ c \to E)$ of UF which we will later show to be equivalent to UF. We inductively define:

```
$\mathsf{Uni}^+\ D\ :\ \mathbb{N}\ \to\ \mathsf{Set}_1$
$\iota_+\ :\ \mathsf{Uni}^+\ D\ 0$
$\delta\sigma\ :\ \{n\ :\ \mathbb{N}\}\ \to\ (G\ :\ \mathsf{Uni}^+\ D\ n\ ) \to\ (\mathsf{Info}^+\ G\ \to\ \mathsf{Cont})\ \to\ \mathsf{Uni}^+\ D\ (\mathsf{suc}\ n)$

$\mathsf{Info}^+\ :\ \{n\ :\ \mathbb{N}\}\ \to\ \mathsf{Uni}^+\ D\ n\ \to\ \mathsf{Set}_1$
$\mathsf{Info}^+\ \iota_+\ =\ 1$
$\mathsf{Info}^+\ (\delta\sigma\ G\ H)\ =\ (\Sigma\ x\ :\ \mathsf{Info}^+\ G\ )\ (\ [\![\ (H\ x)\ ]\!]\ D)$
```

```
$\mathsf{UF}^+\ :\ (D\ E\ :\ \mathsf{Set}_1)\ \to\ \mathbb{N}\ \to\ \mathsf{Set}_1$
$\mathsf{UF}^+\ D\ E\ n\ =\ (\Sigma\ c\ :\ \mathsf{Uni}^+\ D\ n\ )\ (\mathsf{Info}^+\ c\ \to\ E)$
```

Remark 5.4.1.1. *Like in the case of* DS *(see Section 3.2.3.3) and* UF *(see Remark 5.2.0.3) there are two two equivalent presentations for* UF^+*, i.e. we can separate the* $\delta\sigma$ *constructor by:*

```
$\sigma_+\ :\ \{n\ :\ \mathbb{N}\}\ \to\ (G\ :\ \mathsf{Uni}^+\ D\ n\ ) \to\ (\mathsf{Info}^+\ G\ \to\ \mathsf{Set})\ \to\ \mathsf{Uni}^+\ D\ (\mathsf{suc}\ n)$
$\sigma_+\ G\ A\ =\ \delta\sigma\ G\ (\lambda\ \gamma\ \to\ (A\ \gamma\ ,\ (\lambda\ \_\ \to\ 0))$

$\delta_+\ :\ \{n\ :\ \mathbb{N}\}\ \to\ (G\ :\ \mathsf{Uni}^+\ D\ n\ ) \to\ (\mathsf{Info}^+\ G\ \to\ \mathsf{Set})\ \to\ \mathsf{Uni}^+\ D\ (\mathsf{suc}\ n)$
$\delta_+\ G\ A\ =\ \delta\sigma\ G\ (\lambda\ \gamma\ \to\ (1\ ,\ (\lambda\ \_\ \to\ A\ \gamma)))$
```

Definition 5.4.1.2 (degree of a context). *The degree of the context part of a code is recursively defined by:*

$$
\begin{aligned}
&\nu \;:\; \mathsf{Uni}\; D \;\to\; \mathbb{N}\\
&\nu\; \iota \;=\; 0\\
&\nu\; (\sigma\; G\; A) \;=\; \mathsf{suc}\; (\nu\; G)\\
&\nu\; (\delta\; G\; A) \;=\; \mathsf{suc}\; (\nu\; G)
\end{aligned}
$$

We have translations between the graded and the ungraded versions of UF.

Definition 5.4.1.3. *The passage from* UF *to the graded system* UF$^+$ *is given by the following function:*

$$
\begin{aligned}
&\sharp \;:\; (c \;:\; \mathsf{Uni}\; D) \;\to\; \mathsf{Uni}^+\; D\; (\nu\; c)\\
&\sharp\; \iota \;=\; \iota_+\\
&\sharp\; (\sigma\; G\; A) \;=\; \sigma_+\; (\sharp\; G)\; (A \;\circ\; \sharp\mathsf{Info}\; G)\\
&\sharp\; (\delta\; G\; A) \;=\; \delta_+\; (\sharp\; G)\; (A \;\circ\; \sharp\mathsf{Info}\; G)
\end{aligned}
$$

$$
\begin{aligned}
&\sharp\mathsf{Info} \;:\; (c \;:\; \mathsf{Uni}\; D) \;\to\; \mathsf{Info}^+\; (\sharp\; c) \;\to\; \mathsf{Info}\; c\\
&\sharp\mathsf{Info}\; \iota \;=\; \mathsf{id}\\
&\sharp\mathsf{Info}\; (\sigma\; G\; A) \;=\; \mathit{map}\; (\sharp\mathsf{Info}\; G)\; \mathsf{proj}_1\\
&\sharp\mathsf{Info}\; (\delta\; G\; A) \;=\; \mathit{map}\; (\sharp\mathsf{Info}\; G)\; \mathsf{proj}_2
\end{aligned}
$$

$$
\begin{aligned}
&\sharp\mathsf{UF} \;:\; \{E \;:\; \mathsf{Set}_1\} \;\to\; (R \;:\; \mathsf{UF}\; D\; E) \;\to\; \mathsf{UF}^+\; D\; E\; (\nu\; (\mathsf{proj}_1\; R))\\
&\sharp\mathsf{UF}\; (c\; ,\; \alpha) \;=\; (\sharp\; c\; ,\; \alpha \;\circ\; \sharp\mathsf{Info}\; c)
\end{aligned}
$$

Conversely, there is a "forgetful function" removing the degree:

$$
\begin{aligned}
&\flat \;:\; \{n \;:\; \mathbb{N}\} \;\to\; (c \;:\; \mathsf{Uni}^+\; D\; n) \;\to\; \mathsf{Uni}\; D\\
&\flat\; \iota_+ \;=\; \iota\\
&\flat\; (\delta\sigma\; G\; H) = \;\delta\; (\sigma\; (\flat\; G)\; (\lambda\; \gamma \;\to\; \mathsf{proj}_1\; (H\; (\flat\mathsf{Info}\; G\; \gamma))))\\
&(\lambda\; \{\; (\gamma\; ,\; x) \;\to\; \mathsf{proj}_2\; (H\; (\flat\mathsf{Info}\; G\; \gamma))\; x\})
\end{aligned}
$$

$$
\begin{aligned}
&\flat\mathsf{Info} \;:\; \{n \;:\; \mathbb{N}\} \;\to\; (c \;:\; \mathsf{Uni}^+\; D\; n) \;\to\; \mathsf{Info}\; (\flat\; c) \;\to\; \mathsf{Info}^+\; c\\
&\flat\mathsf{Info}\; \iota_+\; x \;=\; x\\
&\flat\mathsf{Info}\; (\delta\sigma\; G\; H)\; ((\; \gamma\; ,\; x)\; ,\; y) \;=\; \flat\mathsf{Info}\; G\; \gamma\; ,\; x\; ,\; y
\end{aligned}
$$

$$
\begin{aligned}
&\flat\mathsf{UF}^+ \;:\; \{E \;:\; \mathsf{Set}_1\}\{n \;:\; \mathbb{N}\} \;\to\; \mathsf{UF}^+\; D\; E\; n \;\to\; \mathsf{UF}\; D\; E\\
&\flat\mathsf{UF}^+\; (c\; ,\; \alpha) \;=\; (\flat\; c\; ,\; \alpha \;\circ\; \flat\mathsf{Info}\; c)
\end{aligned}
$$

Remark 5.4.1.4. *The graded and the ungraded system of uniform codes are equivalent. It is important, though, that this is only so because* UF *codes are uniform: a* DS *code* $\sigma A f$ *does not have a well defined degree since branches can have different length. This remark should be compared to Section 6.7.*

A decoding $[\![-]\!]^+$ can be defined for UF^+ along the lines for the one for UF (alternatively, Proposition 5.4.1.5(ii) below can be used as a definition).

Proposition 5.4.1.5. *Let* $D, E : \mathsf{Set}_1$ *and* $Z : \mathsf{Fam}\ D$*. If* $c : \mathsf{UF}\ D\ E$ *and* $d : \mathsf{UF}^+\ D\ E\ n$*, then (i)* $[\![\sharp\mathsf{UF}\, c]\!]^+ Z \cong [\![c]\!] Z$*; and (ii)* $[\![\flat\mathsf{UF}^+\, d]\!] Z \cong [\![d]\!]^+ Z$*.* □

This proposition can be summed up in the following commuting diagram:

$$\begin{tikzcd}
\mathsf{UF}\ D\ E \arrow[rr, bend left, "\langle \nu, \sharp\mathsf{UF} \rangle"] \arrow[dr, "{[\![-]\!]}"'] & & (\Sigma n : \mathbb{N})(\mathsf{UF}^+\ D\ E\ n) \arrow[ll, bend left, "\flat\mathsf{Info} \circ \mathsf{proj}_2"] \arrow[dl, "{[\![-]\!]^+}"] \\
& \mathsf{Fam}\ D \to \mathsf{Fam}\ E &
\end{tikzcd}$$

Next, note $[\![\delta\sigma\ c\ 1\ (\_ \mapsto 0)]\!]^+ Z \cong [\![c]\!]^+ Z$. Thus we can pad out $c : \mathsf{UF}^+\ D\ E\ n$ to $\mathsf{pad}_k\ c : \mathsf{UF}^+\ D\ E\ (n + k + 1)$ without changing the meaning of the code:

Lemma 5.4.1.6. *Let* $k : \mathbb{N}$*. There is an operation* $\mathsf{pad}_k : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ (n + k + 1)$ *such that* $[\![\mathsf{pad}_k\ c]\!]^+ Z \cong [\![c]\!]^+ Z$ *for every* $Z : \mathsf{Fam}\ D$ *which is given by*

```
pad : {m : ℕ} → (n : ℕ) → (c : Uni⁺ D m) → Uni⁺ D (suc (m + n))
pad {m} zero    c = subst (Uni⁺ D ∘ suc) (sym (+ − rightId m))
                     (δσ c (λ _ → (1, λ _ → 0)))
pad {m} (suc n) c = subst (Uni⁺ D ∘ suc) (sym (+ − suc m n))
                     (δσ (pad n c) (λ _ → (1, λ _ → 0)))

padInfo : {m : ℕ} → (n : ℕ)(c : Uni⁺ D m) → Info⁺ (pad n c) → Info⁺ c
padInfo {m} zero c UIP (+ − rightId m) = proj_1
padInfo {m} (suc n) c UIP (+ − suc m n) = padInfo n c ∘ proj_1

padUFR : {E : Set₁}{m : ℕ} → (n : ℕ) → UF⁺ D E m → UF⁺ D E (suc (m + n))
padUFR n (c , α) = pad n c , α ∘ padInfo n c
```

where

$$
\begin{aligned}
&+-\mathsf{rightId} : (n : \mathbb{N}) \to n + 0 = n\\
&+-\mathsf{rightId}\ \mathsf{zero} = \mathsf{refl}\\
&+-\mathsf{rightId}\ (\mathsf{suc}\ n) = \mathsf{cong}\ \mathsf{suc}\ (+-\mathsf{rightId}\ n)\\
\\
&+-\mathsf{suc} : (m\ n : \mathbb{N}) \to m + \mathsf{suc}\ n = \mathsf{suc}\ (m\ +\ n)\\
&+-\mathsf{suc}\ \mathsf{zero}\ n = \mathsf{refl}\\
&+-\mathsf{suc}(\mathsf{suc} m) n = \mathsf{cong}\ \mathsf{suc}\ (+-\mathsf{suc}\ m\ n)\ ,
\end{aligned}
$$

and UIP *invokes uniqueness of identity proofs (see Paragraph 2.2.3.3.12).* □

5.4.2 Definition of Coproducts for UF

Since all UF^+ codes of the same length also are of the same shape, it is now easy to form coproducts of such codes by:

$$
\begin{aligned}
&\_ +_+ \_ : \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ n \to \mathsf{UF}^+\ D\ E\ (\mathsf{suc}\, n)\\
&(c, \alpha) +_+ (d, \beta) = (c +_{\mathsf{Uni}^+} d, [\alpha, \beta] \circ (c +_{\mathsf{Info}^+} d))\\
\\
&\_ +_{\mathsf{Uni}^+} \_ : \mathsf{Uni}^+\ D\ n \to \mathsf{Uni}^+\ D\ n \to \mathsf{Uni}^+\ D\ n\\
&\iota_+ +_{\mathsf{Uni}^+} \iota_+ = \sigma_+\ \iota_+\ (\_ \mapsto 2)\\
&(\delta\sigma\ c\ A\ B) +_{\mathsf{Uni}^+} (\delta\sigma\ d\ A'\ B') = \delta\sigma\ (c +_{\mathsf{Uni}^+} d)\ ([A, A'] \circ (c +_{\mathsf{Info}^+} d))\ ([B, B'] \circ (c +_{\mathsf{Info}^+} d))
\end{aligned}
$$

$$(c +_{\mathsf{Info}^+} d) : \mathsf{Info}^+\ (c +_{\mathsf{Uni}^+} d) \to \mathsf{Info}^+\ c + \mathsf{Info}^+\ d$$

Note that we did not need to consider the definition of e.g. $\iota_+ +_{\mathsf{Uni}^+} (\delta\sigma\ c\ A\ B)$ as these summands cannot possibly have the same length.

Lemma 5.4.2.1. *For all* $c, d : \mathsf{UF}^+\ D\ E\ n$ *and* $Z : \mathsf{Fam}\ D$ *we have* $[\![\ c +_+ d\ ]\!]^+ Z \cong [\![\ c\ ]\!]^+ Z + [\![\ d\ ]\!]^+ Z$*, where the right hand side is a coproduct of families.* □

Putting everything together, we have:

Theorem 5.4.2.2. *Let* $D, E : \mathsf{Set}_1$*. Define* $\_ +_{\mathsf{UF}} \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E \to \mathsf{UF}\ D\ E$ *by* $c +_{\mathsf{UF}} d = \mathsf{forget}\ (\mathsf{canon}^+\ c +_+ \mathsf{canon}^+\ d)$*. Then* $[\![\ c +_{\mathsf{UF}} d\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z + [\![\ d\ ]\!]\ Z$*.* □

⋆ ⋆ ⋆

5.5 Example of a Coproduct Code

Having coproducts 5.4 at our disposal, we can reproduce the example of a universe closed under several operations 3.2.1.9.

Example 5.5.0.1 (A universe closed under W-types, again). *We construct the code* $c_{2,\mathsf{UF}} +_{\mathsf{UF}} c_{\mathsf{W},\mathsf{UF}} : \mathsf{UF\ Set\ Set}$ *from the following summands — again note that the nesting is the other way around compared to the* DS *code in Example 3.2.1.9:*

$$\begin{aligned} c_{2,\mathsf{UF}} &= (\iota_{\mathsf{UF}}, \star \mapsto 2) : \mathsf{UF\ Set\ Set} \\ c_{\mathsf{W},\mathsf{UF}} &= \big(\delta_{\mathsf{UF}}\ \big(\delta_{\mathsf{UF}}\ \iota_{\mathsf{UF}}\ (\star \mapsto 1)\big)\ ((\star, A) \mapsto A\star), ((\star, A), B) \mapsto \mathsf{W}\ (A\star)\ B\big) : \mathsf{UF\ Set\ Set} \end{aligned}$$

By Theorem 5.4.2.2, we will have $[\![\ c +_{\mathsf{UF}} d\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z\ + [\![\ d\ ]\!]\ Z$, *where the right hand side uses the coproduct of families. Hence* $c_{2,\mathsf{UF}} +_{\mathsf{UF}} c_{\mathsf{W},\mathsf{UF}}$ *decodes correctly.*

5.6 Subcodes

Remark 5.6.0.1 (subcodes of uniform codes). *Compare Remark 3.2.1.5 Again we have a notion of subcodes which we can compare to* DS *subcodes. In the example of a code for a polynomial functor, for the subtree being the branch over* $s : S$ *respectively, we could take*

$$\delta P(s)(\lambda\_ \to \iota\star) \prec \sigma S(\lambda s \to (\delta P(s)(\lambda\_ \to \iota\star)))$$

$$(\delta(\iota_{\mathsf{UF}})(\lambda\_ \to P(s)), \lambda(\star, Y) \to \star) \prec (\delta(\sigma(\iota_{\mathsf{UF}})(\lambda\_ \to S))(\lambda(\star, s) \to P(s)), \lambda((\star, s), Y) \to \star)\ .$$

If we consider the subcode for the "upgraded" code which has a modified map in the second component not ignoring its arguments, we see that we that we have further possibilities how to define a subcode: we can take the second-component map in the subcode to be the composite of the sum inclusion $in_s : \mathsf{Info}(\delta(\iota_{\mathsf{UF}})(\lambda\_ \to P(s))) \to \mathsf{Info}(c_{\mathsf{W}\ S\ P,\mathsf{UF}})$ *followed by the* s*-component of the map which is the second component of the "upgraded" code*

$$(\delta(\iota_{\mathsf{UF}})(\lambda\_ \to P(s)), \lambda(\star, Y) \to (\big(c : P(s)\big) \to Y\ x)) \prec$$

$$(\delta(\sigma(\iota_{\mathsf{UF}})(\lambda\_ \to S))(\lambda(\star, s) \to P(s)), \lambda((\star, s), Y) \to (\big(c : P(s)\big) \to Y\ x))\ .$$

We could however also take only the sum-inclusion in_s *as second component in which case the subcode has a different type than the code it is subcode of*

$$(\delta(\iota_{\mathsf{UF}})(\lambda\_ \to P(s)), \lambda(\star, Y) \to (\big(c : P(s)\big) \to Y\ x))\ :\ \mathsf{UF}\ D\ \mathsf{Info}(c_{\mathsf{W}\ S\ P,\mathsf{UF}})$$

Since in general a subcode of a subcode should not depend on the top-most code, the latter suggested definition is better. Notice that in particular a subcode of a DS *code always has the same type as the code it is subcode of.*

5.7 Embedding of UF into DS and Consistency of UF

We embed UF into DS, i.e. we give a translation of codes which is *semantics-preserving* in that the decoding of a code is isomorphic to the decoding of its translation. Since UF codes are "backwards" compared to DS codes, this embedding resembles the well-known accumulator based algorithm for reversing a list. Define $\mathsf{accUFtoDS} : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$ (the second argument is the accumulator) by

$$\begin{aligned}
&\mathsf{accUFtoDS}\ \iota_{\mathsf{UF}}\ F = F\ \star\\
&\mathsf{accUFtoDS}\ (\sigma_{\mathsf{UF}}\ c\ A)\ F = \mathsf{accUFtoDS}\ c\ (\gamma \mapsto \sigma\ (A\ \gamma)\ (a \mapsto F\ (\gamma, a)))\\
&\mathsf{accUFtoDS}\ (\delta_{\mathsf{UF}}\ c\ A)\ F = \mathsf{accUFtoDS}\ c\ (\gamma \mapsto \delta\ (A\ \gamma)\ (h \mapsto F\ (\gamma, h)))
\end{aligned}$$

and define $\mathsf{UFtoDS} : \mathsf{UF}\ D\ E \to \mathsf{DS}\ D\ E$ by starting with a ι:

$$\mathsf{UFtoDS}\ (c, \alpha) = \mathsf{accUFtoDS}\ c\ (\iota \circ \alpha)\ .$$

Proposition 5.7.0.1. *The translation* UFtoDS *is a (semantics-preserving) embedding, i.e. for every* $c : \mathsf{UF}\ D\ E$ *and* $Z : \mathsf{Fam}\ D$*, we have* $[\![\ \mathsf{UFtoDS}\ c\ ]\!]\ Z \cong [\![\ c\ ]\!]\ Z$. □

Corollary 5.7.0.2 (Consistency of UF). *The system* UF *is consistent via the embedding it posits and the existing model of the system* DS *[38] into which it embeds.*

5.8 Right Nested Uniform Codes

We said that the inclusion of UF into DS works by "reversing" the order of dependency of codes. We mention for the sake of completeness that there exists also a right-nested version $\mathsf{UF}^{\mathsf{op}}$ of UF defined by the constructors

$$\begin{aligned}
&\mathsf{UF}^{\mathsf{op}}\ (D\ E\ : \mathsf{Set}_1)\ (Q\ :\ \mathsf{Uni}\ D)\ :\ \mathsf{Set}_1\\
&\iota^{op}_Q\ :\ (\mathsf{Info}\ Q\ \to\ E)\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ Q\\
&\sigma^{op}_Q\ :\ (A\ :\ \mathsf{Info}\ Q\ \to\ \mathsf{Set})\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ \ (\sigma_{\mathsf{UF}}\ Q\ A)\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ Q\\
&\delta^{op}_Q\ :\ (A\ :\ \mathsf{Info}\ Q\ \to\ \mathsf{Set})\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ \ (\delta_{\mathsf{UF}}\ Q\ A)\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ Q\ \ .
\end{aligned}$$

There is a translation[2] of this system parametrized by 'contexts' into DS

$$\begin{aligned}
&\sharp\ :\ \ \{Q\ :\ \mathsf{Uni}\ D\}\ \to\ \mathsf{Info}\ Q\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ Q\ \to\ \mathsf{DS}\ D\ E\\
&\sharp\ e\ (\iota^{op}_Q\ d)\ =\ \iota_{\mathsf{UF}}\ (d\ e)\\
&\sharp\ e\ (\sigma^{op}_Q\ A\ h)\ =\ \sigma_{\mathsf{UF}}\ (A\ e)\ (\lambda\ a\ \to\ \sharp\ (e\ ,\ a)\ h)\\
&\sharp\ e\ (\delta^{op}_Q\ A\ h)\ =\ \delta_{\mathsf{UF}}\ (A\ e)\ (\lambda\ g\ \to\ \sharp\ (e\ ,\ g)\ h)\ \ .
\end{aligned}$$

[2] We use here the "enharmonic" notation ♯ to indicate that $\mathsf{UF}^{\mathsf{op}}$ (parametrized by 'contexts') is a subsystem of DS.

which obviously provides a translation $\mathsf{UF}^{\mathsf{op}}\ D\ E\ Q\ \to\ \mathsf{DS}\ D\ E$ in case $Q = \iota_{\mathsf{UF}}$.

Parametrized decoding of $\mathsf{UF}^{\mathsf{op}}$, i.e. a map

$$[\![\ ]\!]\ :\ \{Q\ :\ \mathsf{Uni}\,D\}\ \to\ \mathsf{Info}\,Q\ \to\ \mathsf{UF}^{\mathsf{op}}\ D\ E\ Q\ \to\ \mathsf{Fam}(D) \to \mathsf{Fam}(E)$$

can obviously be defined by precomposing decoding of DS with $\sharp$. Since however composition (our main focus) for $\mathsf{UF}^{\mathsf{op}}$-codes is much more difficult than for UF-codes, we do not study the former system any further in this thesis; in fact we studied the right nested version of uniform codes before we turned to the left-nested one because of the just-mentioned reason.

5.9 Composition for UF codes

5.9.1 A Combined Power-Bind Operator for UF

Composition of DS codes —as we recall from section 4.1— can be inferred from a power operation which we could not define because of the dependency arising in its attempted construction. At the beginning of this chapter (in Remark 5.1.0.2) we gave an intuitive explanation why uniformity of codes is able to resolve this problem. In this section we will substantiate this idea with a formal proof.

Composition for UF will —as for DS— be facilitated by a combination of a power-, and a bind operation. Unfortunately, however, the functor $\mathsf{UF}\ D_{-}$ is not a monad from whose structure we could obtain the bind operation. More generally, since the relation between the bind operation and monad structure is a very close one, (see [87]), we cannot obtain a bind operation at all. The failure of defining both operations can be recognized by remembering the geometric interpretation of these operations: Monad multiplication would send a tree of trees T to a tree t by taking the root of t to be the root of the root of T, and recursively grafting the trees of T onto this root; in the instance of trees: the result of grafting uniform tress upon a uniform tree may not be a uniform tree since, first the trees may differ in height, and second one would need to graft trees of the same height on *all* leaves of the first tree and not only on some while leaving other branches too short. The bind operation does the grafting more selectively according to its two arguments.

This description fortunately entails a way how to resolve the problem: we need to define an operation that chooses the appropriate number of trees all of which have the same height and does the grafting only with these. To this end it comes in handy that a 'context' (see the explanation after Definition 5.2.0.1) c already has a layer structure capturing this pre-formal idea since it exactly encodes the kind of matching of a prefix of context to an appropriate postfix. We come to the following formalization of our bespoke bind operation.

We can think of the operation $- \ggg\!\!=\!\![- \longrightarrow -]$ defined below either as a combination of a bind operation, binding families of sets derived from a power operation of a context by a family of sets indexed over the environment set of that code, i.e. a function

$c \mapsto (\lambda d.(\lambda x.\mathsf{Info}\ D\ c\ x \to d)))$: $\mathsf{Uni} \to (\mathsf{Uni} \to \mathsf{Set}) \to \mathsf{Uni}$; or as an operation $((-,-) \to -)$ which is an exponential function where for $((c,E) \to d)$ the exponent consists not of a code but of the two arguments which would give a code when the appropriate constructor would be applied to them.

Definition 5.9.1.1 ($- \ggg\!\!=\![- \longrightarrow -]$). *We define*

$$
\begin{aligned}
&- \ggg\!\!=\![- \longrightarrow -] : \big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D \to \mathsf{Uni} D\\
&c \ggg\!\!=\![E \longrightarrow \iota_{\mathsf{UF}}] = c\\
&c \ggg\!\!=\![E \longrightarrow \sigma_{\mathsf{UF}}\ d\ A] = \sigma_{\mathsf{UF}}\ (c \ggg\!\!=\![E \longrightarrow d])(\gamma \mapsto \big(e : E(\ggg\!\!=_{\mathsf{Info},0}\ \gamma)\big) \to A(\ggg\!\!=_{\mathsf{Info},1}\ \gamma\, e))\\
&c \ggg\!\!=\![E \longrightarrow \delta_{\mathsf{UF}}\ d\ A] = \delta_{\mathsf{UF}}\ (c \ggg\!\!=\![E \longrightarrow d])(\gamma \mapsto \big(\Sigma e : E(\ggg\!\!=_{\mathsf{Info},0}\ \gamma)\big) A(\ggg\!\!=_{\mathsf{Info},1}\ \gamma\, e))
\end{aligned}
$$

$$
\begin{aligned}
&(c \ggg\!\!=\![E \longrightarrow d])_{\mathsf{Info}} : \mathsf{Info}\ (c \ggg\!\!=\![E \longrightarrow d]) \to \big(\Sigma x : \mathsf{Info}\ c\big)(E\, x \to \mathsf{Info}\ d)\\
&(c \ggg\!\!=\![E \longrightarrow \iota_{\mathsf{UF}}])_{\mathsf{Info}}\ x = (x, (\_ \mapsto \star))\\
&(c \ggg\!\!=\![E \longrightarrow \sigma_{\mathsf{UF}}\ d\ A])_{\mathsf{Info}}\ (x,g) = (\ggg\!\!=_{\mathsf{Info},0}\ x, e \mapsto (\ggg\!\!=_{\mathsf{Info},0}\ x\ e, g\ e))\\
&(c \ggg\!\!=\![E \longrightarrow \delta_{\mathsf{UF}}\ d\ A])_{\mathsf{Info}}\ (x,g) = (\ggg\!\!=_{\mathsf{Info},0}\ x, e \mapsto (\ggg\!\!=_{\mathsf{Info},0}\ x\ e, (a \mapsto g\ (e,a)))
\end{aligned}
$$

where $\ggg\!\!=_{\mathsf{Info},0}$ *and* $\ggg\!\!=_{\mathsf{Info},1}$ *denote the first-, and second projection of* $(c \ggg\!\!=\![E \longrightarrow d])_{\mathsf{Info}}$ *respectively, inferring the other arguments from context.*

This definition is validated by the following proposition:

Proposition 5.9.1.2. *There is an equivalence*

$$[\![\ c \ggg\!\!=\![E \longrightarrow d], (d \ggg\!\!=\![E \longrightarrow d])_{\mathsf{Info}}\ ]\!] \cong ([\![\ c, \mathsf{id}\ ]\!]) \ggg\!\!=_{\mathsf{Fam}} (e \mapsto ((E\ e) \longrightarrow_{\mathsf{Fam}} [\![\ d, \mathsf{id}\ ]\!]))\ .$$

Proof. We do not not reproduce the Agda proof of the previous Proposition 5.9.1.2 here but rather state the following lemma collecting properties of the $\ggg\!\!=\![\longrightarrow]$ operation in diagrammatic form that can be helpful in understanding the corresponding Agda proof. □

Lemma 5.9.1.3 (Decoding of $\_ \ggg\!\!=\![\_ \longrightarrow \_]$).

1. *For all c, E, d, Z for which the following expressions are defined, there is a map $Rt1\ d$ as well as a family of propositional equalities $LemRt1\ d\ x$ indexed over x : $[\![\ c \ggg\!\!=[E \longrightarrow d]\ ]\!]_{\mathsf{Uni}}\ Z$ commuting the following square up to this family of propositional equalities.*

$$\begin{tikzcd}
{[\![\ c \ggg\!\!=\![E \longrightarrow d]\ ]\!]_{\mathsf{Uni}}\ Z} \arrow[r, "Rt1\ d"] \arrow[d, "{[\![\ c \ggg\!\!=\![E \longrightarrow d]\ ]\!]_{\mathsf{Info}}\ Z}"'] & {[\![\ c\ ]\!]_{\mathsf{Uni}} Z} \arrow[d, "{[\![\ c\ ]\!]_{\mathsf{Info}} Z}"] \\
{\mathsf{Info}\ \{D\}(c \ggg\!\!=\![E \longrightarrow z])} \arrow[r, "\ggg\!\!=_{\mathsf{Info},0}"] & {\mathsf{Info}\ \{D\}\ c}
\end{tikzcd}$$

2. *For all c, E, d, Z for which the following expressions are defined, there is a map $Rt2\ d\ x$ as well as a family of propositional equalities $LemRt2\ d\ x\ e$ indexed over*

$e : E(\llbracket\, c \,\rrbracket_{\mathsf{Info}} Z(Rt1\ d\ x))$ commuting the following square up to this family of propositional equalities.

$$\begin{tikzcd}
E(\llbracket\, c \,\rrbracket_{\mathsf{Info}} Z(Rt1\ d\ x)) \arrow[r, "Rt2"] \arrow[d, "\mathsf{subst}\ E\ (\mathsf{sym}\ (Lem\ Rt1\ d\ x))"'] & \llbracket\, d \,\rrbracket_{\mathsf{Uni}} Z \arrow[d, "\llbracket\, d \,\rrbracket_{\mathsf{Info}} Z"] \\
E\ (\gg\!=_{\mathsf{Info},0}\ \llbracket\, c \gg\!= [E \longrightarrow d] \,\rrbracket_{\mathsf{Info}}\ Z\ x) \arrow[r, "\gg\!=_{\mathsf{Info},1}\ d"] & \mathsf{Info}\ \{D\}\ d
\end{tikzcd}$$

3. *For all c, E, d, Z for which the following expressions are defined, there is a map $Lt1\ d$ as well as a family of propositional equalities $LemLtfst\ d$ indexed over $x : \llbracket\, c \,\rrbracket_{\mathsf{Uni}} Z$ and $g : E\llbracket\, c \,\rrbracket_{\mathsf{Info}} Zx \to \llbracket\, d \,\rrbracket_{\mathsf{Uni}} Z$ commuting the following diagram up to this family of propositional equalities.*

$$\begin{tikzcd}
(x : \llbracket\, c \,\rrbracket_{\mathsf{Uni}} Z) \arrow[rr, "in(x{,}g)"] \arrow[d, "\llbracket\, c \,\rrbracket_{\mathsf{Info}} Z"'] & & \Sigma_{[x \in \llbracket\, c \,\rrbracket_{\mathsf{Uni}} Z]}(E(\llbracket\, c \,\rrbracket_{\mathsf{Info}} Zx) \to \llbracket\, d \,\rrbracket_{\mathsf{Uni}} Z) \arrow[d, "Lt1d"] \\
\mathsf{Info}\ \{D\} c & \mathsf{Info}\ \{D\}\ [c ++ [E \longrightarrow d]] \arrow[l, "\gg\!=_{\mathsf{Info},0}"'] & \llbracket\, c \gg\!= [E \longrightarrow d] \,\rrbracket_{\mathsf{Uni}}\ Z \arrow[l, "\llbracket\, c \gg\!= [E \longrightarrow d] \,\rrbracket_{\mathsf{Info}}\ Z"']
\end{tikzcd}$$

4. *For all c, E, d, Z for which the following expressions are defined, there is a family of propositional equalities $LemLtsnd\ d$ indexed over $x : \llbracket\, c \,\rrbracket_{\mathsf{Uni}} Z$ and $g : E\llbracket\, c \,\rrbracket_{\mathsf{Info}} Zx \to \llbracket\, d \,\rrbracket_{\mathsf{Uni}} Z$ commuting the following diagram up to this family of propositional equalities.*

$$\begin{tikzcd}
E(\ \gg\!=_{\mathsf{Info},0}\ d\ (\llbracket\, c \gg\!= [E \longrightarrow d] \,\rrbracket_{\mathsf{Info}}\ Z(Lt1\ d\ (x, g)))) \arrow[r, "\mathsf{subst}\ E\ Lemfst\ d\ (x{,}g)"] \arrow[dr, "\gg\!=_{\mathsf{Info},1}\ d\ (\gg\!=_{\mathsf{Info},0}\ d\ \llbracket\, c \gg\!= [E \longrightarrow d] \,\rrbracket_{\mathsf{Info}}\ Z(Lt1\ d\ (x{,}g)))"'] & E\ (\llbracket\, c \,\rrbracket_{\mathsf{Info}} Zx) \arrow[d, "g"] \\
& \llbracket\, d \,\rrbracket_{\mathsf{Uni}} Z
\end{tikzcd}$$

5.

```
LemLtboth : ∀ {D c E} d → {Z : Fam D} →
      (x : ⟦ c ⟧Uni Z)(g : (e : E (⟦ c ⟧Info Z x)) → ⟦ d ⟧Uni Z) →
      (c ≫=[E ⟶ d])Info (⟦ c ≫=[E ⟶ d] ⟧Info Z (Lt1 d (x , g)))
                              ≡ (⟦ c ⟧Info Z x , ⟦ d ⟧Info Z ∘ g)
LemLtboth {c = c} {E} d {Z} x g
   = Σ ≡ (LemLtfst {c = c} d x g)
      (trans (subst − dom {A′ = E} (LemLtfst {c = c} d x g))
          (ext (λ e →
           trans (LemLtsnd d x g (subst E (sym (LemLtfst d x g)) e))
              (cong (⟦ d ⟧Info Z ∘ g)
                 (subst − sym − subst (LemLtfst d x g))))))
```

Where the functions for equational reasoning used here are defined in Paragraph 2.2.3.3.10 and Paragraph 2.2.3.3.11.

Remark 5.9.1.4. *While is not possible to derive a bind operator from* $\_ \gg\!\!=\![\_ \longrightarrow \_]$*, we do obtain a power operator with the correct universal property by:*

$$A \longrightarrow (c, f) \coloneqq (\iota_{\mathsf{UF}} \gg\!\!=\![(\_ \mapsto A) \longrightarrow c], (\gamma \mapsto f \circ (\mathsf{proj}_2((\iota_{\mathsf{UF}} \gg\!\!=\![(\_ \mapsto A) \longrightarrow c])_{\mathsf{Info}}))\gamma) \enspace .$$

(This fact will not be needed in the proof of composition in Theorem 5.9.2.1.)

5.9.2 Definition of Composition for UF

We can now define composition for UF codes in a fashion similar to Theorem 4.1.0.2, except that we separate the action of the first component of a code and take care of the second component in a second step:

Theorem 5.9.2.1. *The operations*

$$\begin{aligned}
&\_ \bullet_{\mathsf{Uni}} \_ : \mathsf{Uni}\ D \to \mathsf{UF}\ C\ D \to \mathsf{Uni}\ C \\
&(\_ \bullet_{\mathsf{Info}} \_) : \big(c : \mathsf{Uni}\ D\big) \to \big(R : \mathsf{UF}\ C\ D\big) \to \mathsf{Info}\ (c \bullet_{\mathsf{Uni}} R) \to \mathsf{Info}\ c
\end{aligned}$$

simultaneously defined by

$$\begin{aligned}
&\iota_{\mathsf{UF}} \bullet_{\mathsf{Uni}} R = \iota_{\mathsf{UF}} \\
&(\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} R = \sigma_{\mathsf{UF}}\ (c \bullet_{\mathsf{Uni}} R)\ (A \circ (c \bullet_{\mathsf{Info}} R)) \\
&(\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Uni}} (d, \beta) = (c \bullet_{\mathsf{Uni}} (d, \beta)) \gg\!\!=\![(A \circ (c \bullet_{\mathsf{Info}} (d, \beta))) \longrightarrow d]
\end{aligned}$$

$$\begin{aligned}
&(\iota_{\mathsf{UF}} \bullet_{\mathsf{Info}} R)\ x = x \\
&((\sigma_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Info}} R)\ (x, y) = ((c \bullet_{\mathsf{Info}} R)\ x, y) \\
&((\delta_{\mathsf{UF}}\ c\ A) \bullet_{\mathsf{Info}} (d, \beta))\ x = ((c \bullet_{\mathsf{Info}} (d, \beta))\ (\gg\!\!=_{\mathsf{Info},0}\ x, \beta \circ (\gg\!\!=_{\mathsf{Info},1}\ x))
\end{aligned}$$

make the following a composition operation for UF *codes*

$$\begin{aligned}
&\_ \bullet \_ : \mathsf{UF}\ D\ E \to \mathsf{UF}\ C\ D \to \mathsf{UF}\ C\ E \\
&(c, \alpha) \bullet (d, \beta) = (c \bullet_{\mathsf{Uni}} (d, \beta), \alpha \circ (c \bullet_{\mathsf{Info}} (d, \beta)))
\end{aligned}$$

and this operation commutes with decoding in the sense that for every $(U, T) : \mathsf{Fam}(C)$

$$[\![\, c \bullet d \,]\!](U, T) \simeq [\![\, c \,]\!]([\![\, d \,]\!](U, T))$$

is an isomorphism. □

⋆ ⋆ ⋆

5.10 Example: Composing a Universe with A W-Type

We can now generate examples by the new composition operation Theorem 5.9.2.1.

Example 5.10.0.1 (Postcomposing a universe by a W-type). *Composing the code* $c_{2,\,\mathsf{W}} : \mathsf{UF\ Set\ Set}$ *for a universe closed under* W*-types, and containing* 2 *from Example 5.5.0.1 with the "upgraded" code* $c_{\mathsf{W}\ (\mathbb{N},\ \mathsf{Fin})} : \mathsf{UF\ Set\ Set}$ *from Example 5.3.0.1, we get a code for a universe where each constructor now takes a list of inductive arguments, with decoding the product of the decodings. Up to an isomorphism relating coproducts of compositions with compositions of coproducts, the resulting code is* $c_{2\mathsf{W}} \bullet c_{\mathsf{W}\ (\mathbb{N},\ \mathsf{Fin})} \cong c_{2,\mathsf{UF}} +_{\mathsf{UF}} c'_{W,\mathsf{UF}}$*, where* $c_{2,\mathsf{UF}}$ *is as before, and*

$$\begin{aligned} c'_{W,\mathsf{UF}} = (\delta_{\mathsf{UF}}\ (\sigma_{\mathsf{UF}}\ c_{\mathsf{W}\ (\mathbb{N},\ \mathsf{Fin})}\ ((\star, n, Y) \mapsto (\big(x : \mathsf{Fin}\, n\big) \to Y\, x) \to \mathbb{N}))& \\ ((\star, n, Y, e) \mapsto \big(\Sigma y : \big(x : \mathsf{Fin}\, n\big) \to Y\, x\big) \mathsf{Fin}\,(e\, y)),& \\ ((\star, n, Y, e, B) \mapsto \big(\Sigma y : \big(x : \mathsf{Fin}\, n\big) \to Y\, x\big)\big(w : \mathsf{Fin}\,(e\, y)\big) \to B\,(y, w))) \end{aligned}$$

where

$$\begin{aligned} &[\![\ (\sigma_{\mathsf{UF}}\ c_{\mathsf{W}\ (\mathbb{N},\ \mathsf{Fin})}\ ((\star, n, Y) \mapsto (\big(x : \mathsf{Fin}\, n\big) \to Y\, x) \to \mathbb{N}))\ ]\!]_{\mathsf{Uni}} = \\ &\big(\Sigma((\star, n), Y) : [\![\ c_{\mathsf{W}\ (\mathbb{N},\ \mathsf{Fin})}\ ]\!]_{\mathsf{Uni}}\big)(((x : \mathsf{Fin}(n)) \to (T \circ Y)(x)) \to \mathbb{N}) \end{aligned}$$

(since $[\![\ c_{\mathsf{W}\ (\mathbb{N},\ \mathsf{Fin})}\ ]\!]_{\mathsf{Info}}(U, T)((\star, n), Y) = ((\star, n), T \circ Y)$*) arranges for the list of inductive arguments.*

5.11 Conclusion and Outlook

We have introduced a system UF of codes for composable induction-recursion . This system can be regarded as a subsystem of DS and this embedding settles the consistency of UF's axioms. Even though every 'definable' DS code —which includes all main examples like Tarski universes (after some workaround arrange for the necessary coproducts) and inductive definitions— corresponds to a UF code having the same semantics, the system may be criticized because of not subsuming all of DS. We address the latter shortcoming in the following chapter by presenting one more axiomatization of induction-recursion that does subsume DS while enjoying composability for all of its codes.

Chapter 6

Polynomial Codes for Induction-Recursion

We saw in Section 4.1 that composition for Dybjer-Setzer codes requires a power operator. However, simply adding a code for powers results in a system for which we could not decide whether it carries a monad structure and we have seen that for the other systems we studied a bind operation (which would follow from the monad structure) was crucial for constructing composition. In fact it is not even clear (but very unlikely) that $\mathsf{DS} + \pi$ is functorial. This last problem was treated in Chapter 5 by creating a new system that is a container (see Definition 1.2.4.1). Thus, our program for this chapter consists in defining a new system that is a container and contains a constructor for powers (or more generally dependent products of codes); it will turn out that the system PN of *polynomial codes* so constructed is a monad.

There are two ways to justify the name 'polynomial codes' for the present system: firstly, codes are —like polynomials— (either trivial or) constructed as sums-, or products of codes. Secondly, in an equivalent formulation of it, the base case constructor takes polynomials of sets (aka containers) as base case. Since a system whose base case has already nontrivial structure is less amenable for computations, we defer a more detailed presentation of this alternative to Section 6.7 after briefly introducing the equivalent system as a motivation in Section 6.1 and carrying out the main part of the chapter with the version Section 6.2 where the container base case is split in two cases that are easier to handle.

6.1 Motivation: Iterating Containers

We start with a motivation for the kind of extension of DS we are aiming at. Considering the systems DS' Section 3.3 (or UF Chapter 5), we noticed that the organization as a container addresses the possible obstruction to functoriality posed by a constructor like

$$\pi : (A : \mathsf{Set}) \to \mathsf{DS}\ D\ E \to \mathsf{DS}\ D\ (A \to E)$$

where the argument 'D' changes to '$(A \to D)$' depending on an argument taken by this constructor, by disentangling the second argument 'E' of $\mathsf{DS}'\ D\ E$ from the inductively defined $\mathsf{SP}\ D$ that uses only the first. Would we not know that DS and DS' are equivalent, the just repeated observation could nourish hope that adding a constructor for powers to DS' could be more successful than the failed attempt in Section 4.1 to add it to DS. Indeed, in this chapter, we pursue the idea to add a constructor for powers to a modified version of DS':

In the attempt to add a constructor for powers to DS' and to verify functoriality or monadicity, one notices that the problem arising lies in that —since the argument Q respectively $[\![\ Q\ ]\!]D$ in the constructor

$$\sigma\delta' : (Q : \mathsf{Cont}) \to ([\![\ Q\ ]\!]D \to \mathsf{SP}\ D) \to \mathsf{SP}\ D$$

is too inflexible— one is not able to find a container Q complying the requirements imposed by the recursion calling it. One gets the idea that, like in case of UF where we also defined $Q : \mathsf{Uni}$ by induction, and $\mathsf{Info}\{D\} : \mathsf{Uni}\ D \to \mathsf{Set}_1$ by recursion, a pair (Q, h) defined by induction-recursion might be successful also here. The objectives in the present chapter are however to define a system that 1) subsumes all of DS while —as mentioned above— maintaining the property 2) that this system be functorial-, 3) a monad in its second component, and 4) satisfies composability. In the following remark we describe a schema for defining a system satisfying 1) and 2), and it will turn out that resulting system automatically satisfies 3) (this is implied by the conjunction of Section 6.5.1 and Proposition 6.7.1.3), and 4) (see Section 6.5) as well.

Remark 6.1.0.1 ("Inductive Container"). *We will describe now the idea to define the "inductive container"* $(\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}, \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P})$ *satisfying the above described desiderata (1-3); we chose the superscript* $-\mathsf{pi}$ *to indicate that that this system extended by a constructor for dependent products- (or powers) of codes will be (equivalent to) the system that satisfies also composability.*

For every $D : \mathsf{Set}_1$*, the (large) set of containers* $(\mathsf{Cont}\{\mathsf{zero}\}\ ,\ \lambda H \to\ [\![\ H\ ]\!]\ D)\ :\ \mathsf{Cont}\{\mathsf{suc}\ 1\}$[1] *is itself a container one level higher in the universe hierarchy. Now , if we would like to use this as the basis of an induction, we require a base case*

$$\mathsf{bs} : \mathsf{Cont}\{\mathsf{zero}\} \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ D$$

and for an inductive step, we require that for the "decoding" $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\ Q$ *of an "inductive container" Q, for every family of inductive containers indexed over the decoding of Q, we obtain a new inductive container*

$$\mathsf{it} : (Q : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ D) \to (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\ D\ Q \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ D) \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ D$$

in a way that, if we decode the base case, we obtain simple container decoding i.e. $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\ D\ (\mathsf{bs}\ X) = [\![\ X\ ]\!] D$*, and for the* it *constructor we can define* $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ \mathsf{it}\ Q\ f\ =\ \Sigma(x : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\ Q)(\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\ (f\ x))$.

Recalling Remark 3.3.0.1, we can regard DS' *as a reduced version of* $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$*:*

$$\begin{aligned}
&\mathsf{bs}^{-} : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\\
&\mathsf{it}^{-} : (Q : \mathsf{Cont}) \to ([\![\ Q\ ]\!] D \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}) \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ .
\end{aligned}$$

Even if we will not pursue this in this thesis, we mention that one can iterate this process on our "inductive container" $(\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}, \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P})$ *to obtain:*

$$\begin{aligned}
&{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}}' : (D : \mathsf{Set}_1) \to \mathsf{Set}_1\\
&\mathsf{bs}'\ :\ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ D \to {\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}}'\ D\\
&\mathsf{it}'\ :\ (Q : {\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}}'\ D) \to ({\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}}'\ D\ Q \to {\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}}'\ D) \to {\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}}'\ D
\end{aligned}$$

$$\begin{aligned}
&{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}}'\ : {\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}}'\ D \to \mathsf{Set}_1\\
&{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}}'\ \mathsf{bs}'\ X = \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\ X\\
&{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}}'\mathsf{it}'\ Q\ f = \Sigma(x : {\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}}'\ Q)({\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}}'\ (f\ x))\ .
\end{aligned}$$

[1]Here $\mathsf{Cont}\{\mathsf{suc}\ 1\} \coloneqq \Sigma_{A:\mathsf{Set}_1}(A \to \mathsf{Set}_1)$.

One can the define a sequence $(\mathsf{Ct}^n DE)_{n:\mathbb{N}}$ *of systems obtained in this way by*

- $\mathsf{Ct}^1\ D\ E = [\![\ (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\ D,\ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P})\ ]\!] E$
- $\mathsf{Ct}^2\ D\ E = [\![\ (\mathsf{PN}^{-\mathsf{pi}\ \prime}_{\mathsf{Cont}C}\ D,\ \mathsf{PN}^{-\mathsf{pi}\ \prime}_{\mathsf{Cont}P})\ ]\!] E$
- $\mathsf{Ct}^{n+1}\ D\ E = (\mathsf{Ct}^n)'\ D\ E$ *for* $n > 1$ *by repeating the priming procedure described above.*

One obtains inclusion maps $\mathsf{Ct}^n\ D\ E \to \mathsf{Ct}^{n+1}\ D\ E$ *induced by* bs^{n+1} *such that each system subsumes the previous step as a subsystem. And all systems in this sequence are functorial.*

We will return to the relation of DS' *and* Ct^1 *in Section 6.7.*

The idea of “inductive containers” we have just sketched is related[2] to that of “levitation” which appeared in [24] in context of inductive definitions.

Of interest for the purpose of compositionality is here in particular that $\mathsf{Ct}^1 DE$ is a functorial system subsuming DS'. In this chapter we will be interested in a system equivalent to $\mathsf{Ct}^1\ D\ E$ extended by a constructor for dependent products of codes, and we will see that this system carries a monad structure and enjoys composability.

6.2 PN Codes and Their Decoding

In the first presentation PN of the system of polynomial codes for induction recursion, we split the base case bs of the system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ in the motivation of Section 6.1 in two cases $\mathsf{id}_{\mathsf{PN}}$, and con and add a constructor for dependent products of codes (the same construction works with powers in place of dependent products (see Remark 6.2.0.2); we will see that that the splitting of the base case is inessential on the sense that we can add the constructor pi to $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ itself to obtain a system $\mathsf{PN}_{\mathsf{cont}}$ equivalent to PN .

Definition 6.2.0.1. *Let* $D, E : \mathsf{Set}_1$*. The large set* $\mathsf{PN}\ D\ E : \mathsf{Set}_1$ *of* polynomial codes for induction-recursion *is defined by* $\mathsf{PN}\ D\ E := \big(\Sigma c : \mathsf{Poly\,D}\big)(\mathsf{Info\,c} \to E)$*, where* $\mathsf{Poly\,D} : \mathsf{Set}_1$ *and* $\mathsf{Info} : \mathsf{Poly\,D} \to \mathsf{Set}_1$ *are mutually defined by the constructors*

$$\begin{aligned}
&\mathsf{id}_{\mathsf{PN}} : \mathsf{Poly\,D}\\
&\mathsf{con} : (A : \mathsf{Set}) \to \mathsf{Poly\,D}\\
&\mathsf{sigma} : (S : \mathsf{Poly\,D}) \to (\mathsf{Info\,S} \to \mathsf{Poly\,D}) \to \mathsf{Poly\,D}\\
&\mathsf{pi} : (A : \mathsf{Set}) \to (A \to \mathsf{Poly\,D}) \to \mathsf{Poly\,D}
\end{aligned}$$

[2]The idea of “levitation” is rather to define a hierarchy of sets of codes where each one contains a code encoding (via initial algebra semantics) the set of codes on the previous stage. It is likely that one can make the relation of “levitation” and the sequence we just presented more precise but we did not pursue this.

and the recursively defined function

$$\begin{aligned}
&\mathsf{Info} : \mathsf{Poly\,D} \to \mathsf{Set}_1\\
&\mathsf{Info\,id_{PN}} = D\\
&\mathsf{Info\,(con\,A)} = A\\
&\mathsf{Info\,(sigma\,S\,F)} = \big(\Sigma x : \mathsf{Info\,S}\big)(\mathsf{Info\,(Fx)})\\
&\mathsf{Info\,(pi\,A\,F)} = \big(x : A\big) \to (\mathsf{Info\,(Fx)})\ .
\end{aligned}$$

Polynomial codes in $\mathsf{PN}\ D\ E$ *decode to functors* $\mathsf{Fam}\ D \to \mathsf{Fam}\ E$ *in the following way: let* $c : \mathsf{Poly}\ D$ *and* $\alpha : \mathsf{Info\,c} \to E$. *This polynomial code induces a functor* $[\![\ c, \alpha\ ]\!] = ([\![\ c\ ]\!]_0 -, [\![\ c, \alpha\ ]\!]_1 -) : \mathsf{Fam}\ D \to \mathsf{Fam}\ E$ *where* $[\![\ c, \alpha\ ]\!]_1 = \alpha \circ [\![\ c\ ]\!]_{\mathsf{info}}$ *are defined by:*

$$\begin{aligned}
&[\![\ \ ]\!]_0 : \mathsf{Poly\,D} \to \mathsf{Fam}\ D \to \mathsf{Set}\\
&[\![\ \mathsf{id_{PN}}\ ]\!]_0\,(U,T) = U\\
&[\![\ \mathsf{con}\,A\ ]\!]_0\,(U,T) = A\\
&[\![\ \mathsf{sigma}\,S\,F\ ]\!]_0\,(U,T) = \big(\Sigma s : [\![\ S\ ]\!]_0\,(U,T)\big)([\![\ F([\![\ S\ ]\!]_{\mathsf{info}}\,(U,T)\,s)\ ]\!]_0\,(U,T))\\
&[\![\ \mathsf{pi}\,A\,F\ ]\!]_0\,(U,T) = \big(x : A\big) \to [\![\ Fx\ ]\!]_0\,(U,T)
\end{aligned}$$

$$\begin{aligned}
&[\![\ \ ]\!]_{\mathsf{info}} : (c : \mathsf{Poly\,D}) \to ((U,T) : \mathsf{Fam}\ D) \to [\![\ c\ ]\!]_0\,(U,T) \to \mathsf{Info\,c}\\
&[\![\ \mathsf{id_{PN}}\ ]\!]_{\mathsf{info}}\,(U,T)\,x = T\,x\\
&[\![\ \mathsf{con}\,A\ ]\!]_{\mathsf{info}}\,(U,T)\,a = a\\
&[\![\ \mathsf{sigma}\,S\,F\ ]\!]_{\mathsf{info}}\,(U,T)\,(s,x) = ([\![\ S\ ]\!]_{\mathsf{info}}\,(U,T)\,s, [\![\ (F\,([\![\ S\ ]\!]_{\mathsf{info}}\,(U,T)\,s))\ ]\!]_{\mathsf{info}}\,(U,T)\,x\\
&[\![\ \mathsf{pi}\,A\,F\ ]\!]_{\mathsf{info}}\,(U,T)\,g = a \mapsto [\![\ (Fa)\ ]\!]_{\mathsf{info}}\,(U,T)\,(g\,a)
\end{aligned}$$

Warning: polynomial codes should not be confused with polynomial functors [44, 45] or codes for them. Codes decoding to polynomial functors are exactly the terms of Ind (see Remark 2.2.4.6).

We use the same name Info as in uniform codes for the function computing the information represented by a code. The code $\mathsf{id_{PN}}$ represents the identity functor, con A the functor whose index set is constantly A, sigma S F represents a dependent coproduct of functors, and pi A F represents an$A$-indexed dependent product of functors. Observe that PN D _ is again, like UF D _, functorial by function composition.

We observe that while DS codes are right-nested (see Section 3.2.1), i.e. allow for a nesting of terms depending on earlier steps in the second ("right") argument (of the constructors), and UF codes are nested in the first argument (see Section 5.2), PN codes are nested in both arguments.

Remark 6.2.0.2. *One obtains a weaker system by replacing the* pi *constructor with a constructor* $\mathsf{power} : \mathsf{Set} \to \mathsf{Poly}\ D \to \mathsf{Poly}\ D$ *with* $\mathsf{Info\,(power\,A\,c)} = A \to \mathsf{Info\,c}$. *In the full* *system, such an operator can be defined by* $\mathsf{power}\ A\ c \coloneqq \mathsf{pi}\ A\ (\_ \mapsto c)$*. The weaker system also enjoys composition, and the embedding of Dybjer-Setzer codes in Proposition 6.7.1.4 factors through the system with powers only. Semantically, the stronger system is just as easy to handle.*

6.3 Examples

Example 6.3.0.1 (W-types, again). *We revisit Examples 3.2.1.8 and 5.3.0.1. For* $S : \mathsf{Set}$*,* $P : S \to \mathsf{Set}$ *the polynomial code for the* W*-type* $\mathsf{W}\ S\ P$ *is* $(c_{\mathsf{W}\ S\ P,\mathsf{PN}}, \_ \mapsto \star) : \mathsf{PN}\ 1\ 1$ *where* $c_{\mathsf{W}\ S\ P,\mathsf{PN}} = \mathsf{sigma}\ (\mathsf{con}\ S)(s \mapsto\ \mathsf{pi}\ (P\ s)\ (\_ \mapsto \mathsf{id}_{\mathsf{PN}}))$*. Again this can be upgraded to a* $\mathsf{PN}\ \mathsf{Set}\ \mathsf{Set}$ *code applying* $T : U \to \mathsf{Set}$ *everywhere in the tree by replacing the trivial map* $(\_ \mapsto \star) : \mathsf{Info}\, c_{\mathsf{W\ S\ P,PN}} \to 1$ *by the map* $((s, Y) \mapsto \big(c : P(s)\big) \to Y\ x) : \mathsf{Info}\, c_{\mathsf{W\ S\ P,PN}} \to \mathsf{Set}$*.*

Decoding $c_{\mathsf{W}\ S\ P,\mathsf{PN}}$ *we get*

$$[\![\ c_{\mathsf{W}\ S\ P,\mathsf{PN}}\ ]\!]_0\,(U, T) = \big(\Sigma s : S\big)((P\ s) \to U)$$

this time matching the domain of the W*-type constructor* sup *strictly.*

Example 6.3.0.2 (A universe closed under W-types, again). *We also revisit Example 3.2.1.9 again. A polynomial code* $(c_{\mathsf{2W,PN}}, \alpha) : \mathsf{PN}\ \mathsf{Set}\ \mathsf{Set}$ *for a universe containing* 2*, closed under* W*-types is given by* $c_{\mathsf{2W,PN}} : \mathsf{Poly}\ \ \mathsf{Set}$ *where*

$$c_{\mathsf{2W,PN}} = \mathsf{sigma}\ (\mathsf{con}\ \{\mathsf{bool}, \mathsf{w}\})(\mathsf{bool} \mapsto \mathsf{con}\ 1; \mathsf{w} \mapsto \mathsf{sigma}\ \mathsf{id}_{\mathsf{PN}}\ (X \mapsto \mathsf{pi}\ X\ (\_ \mapsto \mathsf{id}_{\mathsf{PN}})))$$

together with $\alpha_{\mathsf{2W,PN}} : \mathsf{Info}\, c_{\mathsf{2W,PN}} \to \mathsf{Set}$ *defined by* $\alpha_{\mathsf{2W,PN}}(\mathsf{bool}, x) = 2$ *and* $\alpha_{\mathsf{2W,PN}}(\mathsf{w}, (A, B)) = \mathsf{W}\ A\ B$*.*

Decoding $(c_{\mathsf{2W,PN}}, \alpha_{\mathsf{2W,PN}})$ *we again get the same result as in Example 3.2.1.9.*

6.4 Existence of Initial Algebras: A Set-Theoretic Model of MLTT + PN

Since we do not exhibit PN as a subsystem of DS, we cannot rely on Dybjer and Setzer's proof that initial algebras of the corresponding functors exist. We can, however, extend their proof to polynomial codes. Recall that, as we saw in Lemma 1.1.0.14, every inaccessible cardinal I can be used to define the universe V_{I} which can be proved to be a model of ZFC, and hence by Gödel's second incompleteness theorem, the existence of I cannot be proved in ZFC (see [68, Theorem 12.12]. Since all versions of IR can define universes, it is not surprising that a model of IR has to assume inaccessibles. More precisely in this section we will show:

Theorem 6.4.0.1. *In ZFC+M+I (where M is a Mahlo cardinal (see Corollary 1.1.0.21), and I is a 1-inaccessible (see Definition 1.1.0.8) containing M,) all functors $[\![c]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ associated to a polynomial code $c : \mathsf{PN}\ D\ D$ have initial algebras.*

To prove this theorem, we adapt Dybjer-and-Setzer's set-theoretic model [38, 39] in ZFC+M+I' and the Generalised Continuum Hypothesis[3] to a model of PN. The assumptions of our model differ from Dybjer-Setzer's only in that I has to be 1-inaccessible and not 0-inaccessible as I' where we recall that a cardinal I is 1-inaccessible if the set X of 0-inaccessibles less than I is unbounded in I, i.e. -if the supremum of X equals I.

6.4.1 Interpreting the Elements of $\mathsf{MLTT} + \mathsf{PN}$

We begin in the first subsection with the interpretation of the version of MLTT which is necessary for the extension by PN.

6.4.1.1 Interpreting the Elements of MLTT

Like Dybjer-Setzer, we interpret Set as V_{M} and Set_1 as V_{I}, where V_α is the cumulative hierarchy. Dybjer and Setzer [39] require I to be 0-inaccessible (and containing M) only, since their set of codes is only inductively defined.

The interpretation of the types and terms in MLTT is the same as given by Dybjer-Setzer in [38, p.9-10] (see also [35, §2.3] and the further reference given there). We repeat it here for the sake of completeness.

The interpretation of types is straightforward: In particular if $x : A \dashv B(x) : \mathsf{Set}$ is interpreted as 'if $x \in (\!|A|\!)$ then $(\!|B(x)|\!) \in \mathsf{Set}$', and we interpret

$$
\begin{aligned}
&(\!|(A, B)|\!) := \{\{A\}, \{A, B\}\},\\
&(\!|\lambda\ (x : A) \to B(x)|\!) := \{(x, B(x)) | x \in A\},\\
&(\!|\Pi_{x \in A} B(x)|\!) := \{f : A \to \cup_{x \in A} B(x) | \forall (x \in A). f(x) \in B(x)\}\\
&(\!|\Sigma_{x:A} B(x)|\!) := \{(c, d) | c \in A \wedge d \in B(c)\},\\
&(\!|0|\!) := \varnothing,\\
&(\!|1|\!) := \{0\},\\
&(\!|2|\!) := \{0, 1\},\\
&(\!|A_0 + \cdots + A_n|\!) := \Sigma_{i \in \{0,\dots,n\}} Ai\ \ .
\end{aligned}
$$

[3]Notice that that the Generalized Continuum Hypothesis implies the axiom of choice by a theorem of Sierpinsky's (see [53]). The axiom of choice is used explicitly in the construction of the model.

6.4.1.2 Interpretation of $\mathsf{Poly\,D}$

We now need to verify that $\mathsf{Poly\,D}$ can be constructed in the model, and that initial algebras exist also for PN functors.

Lemma 6.4.1.1. *The interpretation of the type* $\mathsf{Poly\,D}$ *is in* V_{I}*, and so is the interpretation of* $\mathsf{Info}\,(X)$ *for every* X *in* $\mathsf{Poly\,D}$.

Proof. First, to see that this lemma has the correct implications: since the interpretation of D is required to be an element of the interpretation V_{I} of Set_1, also the interpretation of $\mathsf{Poly\,D}$ must be in V_{I} since $(\mathsf{Poly\,D}, \mathsf{Info}\,) : \mathsf{FamSet}_1$.

That this is the case can be seen by observing that $(\mathsf{Poly\,D}, \mathsf{Info}\,) : \mathsf{FamSet}_1$ is exactly a Tarski universe in Set_1 closed under Σ and Π, and containing a code for D. Such a Tarski universe can be modeled by assuming a 0-inaccessible cardinal. Since we, however, need to have such a Tarski universe for *all* $D : \mathsf{Set}_1$, we need to assume at least that the set of all such 0-inaccessibles is contained in our model; by definition the least cardinal allowing for this is a 1-inaccessible.□

Notice that arguing that $(\mathsf{Poly\,D}, \mathsf{Info}\,)$ is defined by induction-recursion on FamSet_1 and thus has a model by shifting the size of the model of [38] by one, would require a second Mahlo cardinal containing the first one which would be an unnecessarily strong assumption.

Notice also that for the model of DS the assumption of a 0-inaccessible above M is sufficient since for a fixed D, the definition of $\mathsf{DS}\ D\ D$ does not require an inaccessible beyond M. And again the necessity of a 0-inaccessible comes about since we need to define $\mathsf{DS}\ D\ D$ for all D.

6.4.2 Construction of Initial Algebras for PN Functors

The remainder of the construction of the model shows that initial algebras of PN functors exist. As we have sketched in Section 1.2.3, the strategy is also here to iterate the application of PN functors to an initial object(here: of $\mathsf{Fam(Set)}$). But now we have —in addition to convergence of the iteration— to take care that for a family (U, T), and (U', T') the result of sufficiently many application of an PN-functor to it, the interpretation of U' is still contained in the universe V_{I}. Apparently, the (maximal) size of the interpretation of U' depends on the inductive-recursive structure of the index-set $[\![\ c\ ]\!]_0$ part of the PN-functor as well as on the family (U, T); this size is computed by the following definition of$\mathsf{Aux}(c, (U, T))$:

Definition 6.4.2.1. *Given a polynomial code* $c : \mathsf{Poly\,D}$ *and an object* (U, T) *of* $\mathsf{Fam}D$*, the set* $\mathsf{Aux}(c, U, T)$ *of premises of inductive arguments of* c *with respect to* U*,* T *is defined* *by induction over c:*

$$\begin{aligned}
\mathsf{Aux}(\mathsf{id}_{\mathsf{PN}}, U, T) &= \mathsf{Aux}(\mathsf{con}\, A, U, T) = \emptyset \\
\mathsf{Aux}(\mathsf{sigma}\, S\, F, U, T) &= \mathsf{Aux}(S, U, T) \cup \bigcup_{x\colon [\![\, S \,]\!]_0(U,T)} \mathsf{Aux}(F([\![\, S \,]\!]_{\mathsf{info}}(U,T)\, x), U, T) \\
\mathsf{Aux}(\mathsf{pi}\, A\, F, U, T) &= \{A\} \cup \bigcup_{x\colon A} \mathsf{Aux}(F x, U, T)
\end{aligned}$$

We now observe that if for certain (U, T), all $A \in \mathsf{Aux}(c, U, T)$ are "small" in a suitable sense then $[\![\, c \,]\!]$ is κ-continuous for some inaccessible $\kappa < \mathsf{M}$. Hence, by a standard argument — see e.g. Adámek et al [7] — we can conclude that $[\![\, c \,]\!]$ has an initial algebra.

6.4.2.1 Continuity of PN-Functors in Monotone κ-Sequences

We first show that PN-functors are monotone in the partial order on Fam(Set) (see Remark 1.3.0.4).

Lemma 6.4.2.2. *Let $c : \mathsf{Poly}\, \mathsf{D}$, and (U, T), (U', T') be objects of* Fam *D. Assume $U \subseteq U'$ and $T' \restriction U = T$. Then*

1. $[\![\, c \,]\!]_0(U, T) \subseteq [\![\, c \,]\!]_0(U', T')$, *and*
2. $[\![\, c \,]\!]_{\mathsf{info}}(U', T') \restriction [\![\, c \,]\!]_0(U, T) = [\![\, c \,]\!]_{\mathsf{info}}(U, T)$.

Proof. Induction on c:

- If $c = \mathsf{id}_{\mathsf{PN}}$, then the assertion follows by assumption.
- If $c = \mathsf{con}\, A$, then the assertion is trivial.
- If $c = \mathsf{sigma}\, S\, F$, then $[\![\, c \,]\!]_0(U, T) = \big(\Sigma s : [\![\, S \,]\!]_0\, (U, T)\big)([\![\, F([\![\, S \,]\!]_{\mathsf{info}}\, (U, T)\, s) \,]\!]_0\, (U, T))$. By the induction hypothesis, $[\![\, S \,]\!]_0\, (U, T) \subseteq [\![\, S \,]\!]_0\, (U', T')$ and $[\![\, S \,]\!]_{\mathsf{info}}\, (U', T')\, s = [\![\, S \,]\!]_{\mathsf{info}}\, (U, T)\, s$, hence by the induction hypothesis again
$$[\![\, F([\![\, S \,]\!]_{\mathsf{info}}\, (U, T)\, s) \,]\!]_0\, (U, T) \subseteq [\![\, F([\![\, S \,]\!]_{\mathsf{info}}\, (U', T')\, s) \,]\!]_0\, (U', T')$$
for every $s \in [\![\, S \,]\!]_0\, (U, T)$, and the assertion follows from the monotonicity of sigma-types.
- If $c = \mathsf{pi}\, S\, F$, then the assertion follows from the induction hypothesis and the monotonicity of pi-types in the codomain. □

Notation 6.4.2.3. *Write $\bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha)$ for $(\bigcup_{\alpha<\kappa} U^\alpha, \bigcup_{\alpha<\kappa} T^\alpha)$.*

Lemma 6.4.2.4 (Monotonicity of PN-functors in κ-sequences). *Let κ be inaccessible and $(U^\alpha, T^\alpha)_{\alpha<\kappa}$ be a monotone κ-sequence of objects of* Fam*D, i.e. if $\alpha < \beta$ then $U^\alpha \subseteq U^\beta$ and $T^\beta \restriction U^\alpha = T^\alpha$. Assume for some $\alpha_0 < \kappa$ that*

$$\mathsf{Aux}(c, U^\alpha, T^\alpha) \subseteq V_\kappa \tag{6.1}$$

for all $\alpha_0 \leq \alpha < \kappa$. Then $[\![\, c \,]\!]_0$ is κ-continuous in (U, T), i.e.

$$[\![\, c \,]\!]_0(\bigcup_{\alpha<\kappa} U^\alpha, \bigcup_{\alpha<\kappa} T^\alpha) = \bigcup_{\alpha<\kappa} [\![\, c \,]\!]_0(U^\alpha, T^\alpha) \ .$$

Proof. The direction $\supseteq$ follows immediately from Lemma 6.4.2.2. We prove $\subseteq$ by induction over c:

- If $c = \mathsf{id_{PN}}$, then

$$[\![\mathsf{id_{PN}}]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha) = \bigcup_{\alpha<\kappa} U^\alpha = \bigcup_{\alpha<\kappa} [\![\mathsf{id_{PN}}]\!]_0(U^\alpha, T^\alpha) \ .$$

- If $c = \mathsf{con}\, A$, then

$$[\![\mathsf{con}\, A]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha) = A = \bigcup_{\alpha<\kappa} A = \bigcup_{\alpha<\kappa} [\![\mathsf{con}\, A]\!]_0(U^\alpha, T^\alpha) \ .$$

- If $c = \mathsf{sigma}\, S\, F$, then assume $a \in [\![\mathsf{sigma}\, S\, F]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha)$. We want to find $\alpha < \kappa$ such that $a \in [\![\mathsf{sigma}\, S\, F]\!]_0(U^\alpha, T^\alpha)$. We know $a = (x, y)$ for some $x \in [\![S]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha)$ and

$$y \in [\![F([\![S]\!]_{\mathsf{info}} \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha)\, x)]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha) \ .$$

  By the induction hypothesis, $x \in [\![S]\!]_0(U^\beta, T^\beta)$ for some $\beta < \kappa$. Without loss of generality, $\alpha_0 \le \beta$ (if not, choose $\beta := \alpha_0$: we still have $x \in [\![S]\!]_0(U^\beta, T^\beta)$ by monotonicity of $[\![S]\!]_0$), and by Lemma 6.4.2.2, $[\![S]\!]_{\mathsf{info}} \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha)\, x = [\![S]\!]_{\mathsf{info}}(U^\beta, T^\beta)\, x$, hence in fact

$$y \in [\![F([\![S]\!]_{\mathsf{info}}(U^\beta, T^\beta)\, x)]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha) \ .$$

  Now for every $\beta \le \alpha < \kappa$, $\mathsf{Aux}(F([\![S]\!]_{\mathsf{info}}(U^\beta, T^\beta)\, x), U^\alpha, T^\alpha) \subseteq V_\kappa$ by the definition of Aux and the inductive hypothesis, and so

$$y \in [\![F([\![S]\!]_{\mathsf{info}}(U^\beta, T^\beta)\, x)]\!]_0(U^{\beta'}, T^{\beta'})$$

  for some $\beta' < \kappa$ by the induction hypothesis. We have $a = (x, y) \in [\![\mathsf{sigma}\, S\, F]\!]_0(U^\alpha, T^\alpha)$ where $\alpha = \max\{\beta, \beta'\}$.

- If $c = \mathsf{pi}\, A\, F$, then assume $f \in [\![\mathsf{pi}\, A\, F]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha)$, i.e.

$$f \in \big(x : A\big) \to [\![Fx]\!]_0 \bigcup_{\alpha<\kappa}(U^\alpha, T^\alpha) = \big(x : A\big) \to \bigcup_{\alpha<\kappa} [\![Fx]\!]_0(U^\alpha, T^\alpha)$$

  by the induction hypothesis. Hence for every $x \in A$, there is an α_x such that $f(x) \in [\![Fx]\!]_0(U^{\alpha_x}, T^{\alpha_x})$. By the definition of Aux, $A \in V_\kappa$, hence $|A| < \kappa$, and by the inaccessibility of κ, $\alpha' := \sup_{x\in A} \alpha_x < \kappa$, and we have $f \in \big(x : A\big) \to [\![Fx]\!]_0(U^{\alpha'}, T^{\alpha'})$ as required. $\square$

6.4.2.2 Continuity of PN-functors in Their Initial Sequences

We now need to ensure that the hypotheses of Lemma 6.4.2.4 hold for the initial sequence of an PN-functor, i.e. we need to show that 1) this initial sequence is a monotone κ-sequence (of objects of $\mathsf{Fam}D$), and 2) that $\mathsf{Aux}(c, U^\alpha, T^\alpha) \subseteq V_\kappa$. We first define this initial sequence:

Definition 6.4.2.5. *For $c : \mathsf{Poly\,D}$ and $f : \mathsf{Info\,c} \to D$ we define the initial sequence by*

$$\begin{aligned} U^0 &= \emptyset & T^0 &= \emptyset \\ U^{\alpha+1} &= [\![\, c \,]\!]_0(U^\alpha, T^\alpha) & T^{\alpha+1}(x) &= f([\![\, c \,]\!]_{\mathsf{info}}(U^\alpha, T^\alpha)\, x) \\ U^\lambda &= \bigcup_{\beta<\lambda} U^\beta & T^\lambda(x) &= T^\beta \quad \textit{where } x \in U^\beta \end{aligned}$$

Lemma 6.4.2.6. *Let c, f be a polynomial code and $([\![\, c, f \,]\!]^\alpha)_\alpha = (U^\alpha, T^\alpha)_\alpha$ the initial sequence of the associated functor. If $\alpha < \beta$ then $U^\alpha \subseteq U^\beta$ and $T^\beta \restriction U^\alpha = T^\alpha$.*

Proof. Induction on α, β, using Lemma 6.4.2.2 for the step cases.

The proof of the following lemma showing that the second premise of Lemma 6.4.2.4 is satisfied will be the only place where the existence of M is used in the model construction.

Lemma 6.4.2.7. *Let (c, f) be a PN code and $([\![\, (c, f) \,]\!]^\alpha)_\alpha = (U^\alpha, T^\alpha)_\alpha$ the initial sequence of the associated functor. There exists an inaccessible κ such that $\mathsf{Aux}(c, U^\alpha, T^\alpha) \subseteq V_\kappa$ for all $\alpha < \kappa$.*

Proof. The strategy for the proof is as follows: we define an increasing function $f : \mathsf{Ord} \to \mathsf{Ord}$, which tells us how much further up the cumulative hierarchy we need to go to contain one iteration of $[\![\, c \,]\!]_0$. The important property of f will be

$$\text{if } U^{\beta'} \subseteq V_\beta \text{ then } U^{\beta'+1} \cup \mathsf{Aux}(c, U^{\beta'}, T^{\beta'}) \subseteq V_{f(\beta)} \tag{6.2}$$

for all $\beta' < \mathsf{M}$. We then show that $f : \mathsf{M} \to \mathsf{M}$ and use the Mahlo property to find an inaccessible fixed point κ of f. Finally we show $\mathsf{Aux}(c, U^\alpha, T^\alpha) \subseteq V_\kappa$ by induction on α, using (6.2).

The function $f : \mathsf{Ord} \to \mathsf{Ord}$ is defined by transfinite recursion:

$$\begin{aligned} f(\beta) = \min\{\alpha \mid & (\forall \beta' < \beta)\big(f(\beta') < \alpha\big) \wedge \\ & (\forall \beta' < \mathsf{M})\big(U^{\beta'} \subseteq V_\beta \implies U^{\beta'+1} \cup \mathsf{Aux}(c, U^{\beta'}, T^{\beta'}) \subseteq V_\alpha\big)\} \end{aligned}$$

The first conjunct makes sure that f is increasing, and the second makes (6.2) true.

<u>Claim:</u> $f : \mathsf{M} \to \mathsf{M}$.
Proof of claim. Let $\beta < \mathsf{M}$ and note that

$$\begin{aligned} f(\beta) = \min\{\alpha \mid & (\forall \beta' < \beta)\big(f(\beta') < \alpha\big) \wedge \\ & (\forall \beta' \in \{\beta' \in \mathsf{M} \mid U^{\beta'} \subseteq V_\beta\})\big(U^{\beta'+1} \cup \mathsf{Aux}(c, U^{\beta'}, T^{\beta'}) \subseteq V_\alpha\big)\} \ , \end{aligned}$$

further that $B :\equiv \{\beta' \in \mathsf{M} \mid U^{\beta'} \subseteq V_\beta\} \in V_{\beta+1} \subseteq V_{\mathsf{M}}$ so that $|B| < \mathsf{M}$. For each $\beta' \in B$, we have $U^{\beta'+1} \cup \mathsf{Aux}(c, U^{\beta'}, T^{\beta'}) \subseteq V_{\mathsf{M}}$ and hence $U^{\beta'+1} \cup \mathsf{Aux}(c, U^{\beta'}, T^{\beta'}) \subseteq V_{\alpha_{\beta'}}$ for some $\alpha_{\beta'} < \mathsf{M}$ since M is inaccessible. Thus $f(\beta) \leq \sup_{\beta'} \alpha_{\beta'} < \mathsf{M}$ by the regularity of M. □

So f is an increasing function on M, however f does not need to be continuous at limits, hence not normal and the Mahlo property might not apply. To fix this, we define a new function $\theta : \mathsf{Ord} \to \mathsf{Ord}$ by $\theta(\alpha) = f^\alpha(0)$.

Claim: $\theta : \mathsf{M} \to \mathsf{M}$, and θ is normal.
Proof of claim. We prove that $\theta(\alpha) < \mathsf{M}$ for $\alpha < \mathsf{M}$ by transfinite induction over α. The base case and successor case are clear, since $f : \mathsf{M} \to \mathsf{M}$. If $\lambda < \mathsf{M}$ is a limit, then $\theta : \lambda \to \mathsf{M}$ is a normal function so that $\theta(\lambda) = \sup_{\beta<\lambda} \theta(\beta) < \mathsf{M}$ by the regularity of M. Finally θ is increasing since f is, and it is continuous at limits by definition. □

Hence by the Mahlo property, θ has an inaccessible fixed point $\kappa < \mathsf{M}$.

Claim: $f : \kappa \to \kappa$.
Proof of claim. Assume $\alpha < \kappa$. Since κ is inaccessible, $\alpha < \beta$ for some $\beta < \kappa$, and $\beta \leq \theta(\beta)$. Thus

$$f(\alpha) < f(\beta) \leq f(\theta(\beta)) = \theta(\beta + 1) < \theta(\kappa) = \kappa$$

i.e. $f : \kappa \to \kappa$. □

This combined with (6.2) gives us a useful fact:

$$\text{if } U^{\beta'} \subseteq V_\beta \text{ then } U^{\beta'+1} \cup \mathsf{Aux}(c, U^{\beta'}, T^{\beta'}) \subseteq V_\kappa \tag{2'}$$

for all $\beta < \kappa$ (since $f(\beta) < \kappa$, hence $V_{f(\beta)} \subseteq V_\kappa$).

Finally, we prove that $U^\alpha \subseteq V_\kappa$ for all $\alpha < \kappa$. By (2′), it then immediately follows that $\mathsf{Aux}(c, U^\alpha, T^\alpha) \subseteq V_\kappa$. The proof is by induction on α:

- If $\alpha = 0$, then $U^0 = \emptyset \subseteq V_\kappa$.
- If $\alpha = \beta + 1$, then $U^\beta \subseteq V_\kappa$ by the induction hypothesis, and we are done by (2′).
- If $\alpha = \lambda$ limit, then $U^\lambda = \bigcup_{\beta<\lambda} U^\beta \subseteq V_\kappa$ by the induction hypothesis. □

6.4.2.3 Completing the Proof

By combining Lemma 6.4.2.4 and Lemma 6.4.2.7, we get:

Theorem 6.4.2.8. *Assuming that a Mahlo cardinal exists in the meta-theory, all functors* $[\![\, c \,]\!] : \mathsf{Fam}\ D \to \mathsf{Fam}\ D$ *arising from a polynomial code* $c : \mathsf{PN}\ D\ D$ *have initial algebras.*

Proof. Feeding Lemma 6.4.2.6 and Lemma 6.4.2.7 into Lemma 6.4.2.4, we get that

$$\begin{aligned}
[\![\, c \,]\!]_0(\bigcup_{\alpha<\kappa} U^\alpha, \bigcup_{\alpha<\kappa} T^\alpha) &= \bigcup_{\alpha<\kappa} [\![\, c \,]\!]_0(U^\alpha, T^\alpha) \\
&= \bigcup_{\alpha<\kappa} U^{\alpha+1} \\
&= \bigcup_{\alpha<\kappa} U^\alpha \ .
\end{aligned}$$

By Lemma 6.4.2.6, $[\![\, c \,]\!]_{\mathsf{info}}(\bigcup_{\alpha<\kappa} U^{\alpha}, \bigcup_{\alpha<\kappa} T^{\alpha}) = \bigcup_{\alpha<\kappa} T^{\alpha}$, so that the initial sequence converges after κ steps. By Adamek's Theorem [7, Thm 3.1.4], $[\![\, c \,]\!]$ has an initial algebra.

⋆ ⋆ ⋆

6.5 Composition of PN Codes

Composition for PN codes can be defined following the same pattern as in Proposition 4.1.0.2, where we constructed composition for DS codes using the assumption of a power operation, and the monadicity of DS. The system PN has a power operation using the pi constructor, and is monadic thanks to the sigma constructor:

6.5.1 Monad Structure and Bind Operation for PN

Proposition 6.5.1.1. *1. For each $D : \mathsf{Set}_1$, $\mathsf{PN}\ D$ is a monad, i.e. there are terms*

$$\begin{aligned}
&\eta_{\mathsf{PN}} : E \to \mathsf{PN}\ D\ E \\
&\eta_{\mathsf{PN}}(e) = (\mathsf{con}\ 1, \_ \mapsto e) \\[1em]
&\mu_{\mathsf{PN}} : \mathsf{PN}\ D\ (\mathsf{PN}\ D\ E) \to \mathsf{PN}\ D\ E \\
&\mu_{\mathsf{PN}}(c, \alpha) = (\mathsf{sigma}\ c\ (\mathsf{proj}_1 \circ \alpha), (x, y) \mapsto \mathsf{proj}_2\ (\alpha\ x)\ y)
\end{aligned}$$

satisfying the monad laws. Moreover the equations in terms of the monad structure on Fam *hold strictly.*

2. Furthermore this monad structure is compatible with that of Fam*:*

Let $(U, T) : \mathsf{Fam}\ D$. Then $[\![\, \eta_{\mathsf{PN}}(e) \,]\!](U, T) = \eta_{\mathsf{Fam}}(e)$ for every $e : E$, and $[\![\, \mu_{\mathsf{PN}}(c) \,]\!](U, T) = \mu_{\mathsf{Fam}}(\mathsf{Fam}([\![\, - \,]\!](U, T))([\![\, c \,]\!](U, T)))$ for every $c : \mathsf{PN}\ D\ (\mathsf{PN}\ D\ E)$.

Definition 6.5.1.2 (Dependent bind operation). *Using the monad structure, we can define a "dependent bind" operation*

$$\begin{aligned}
\_ \gg\!\!=_{\mathsf{PN}} \_ &: \mathsf{PN}\ C\ D \to \big((x : D) \to \mathsf{PN}\ C\ (E\,x)\big) \to \mathsf{PN}\ C\ \big((\Sigma x : D)(E\,x)\big) \\
c \gg\!\!=_{\mathsf{PN}} h &= \mu_{\mathsf{PN}}(\mathsf{PN}(x \mapsto \mathsf{PN}(y \mapsto (x, y)))\,(h\,x)\ c)
\end{aligned}$$

The following definition which is possible by virtue of pi, solves the problem posed by the obstruction to composition described in the passage after Eq. (4.1), and in the penultimate sentence of Remark 5.1.0.2.

Definition 6.5.1.3 (Compatibility of $\mathsf{PN}\ D\ \_$ with dependent functions). *The* pi *constructor gives rise to the following operation that commutes the dependent function space and* $\mathsf{PN}\ D\ \_$ *for every* $S : \mathsf{Set}$ *and* $E : A \to \mathsf{Set}_1$*:*

$$\begin{aligned}\pi_{\mathsf{PN}} A &: \big((a : A) \to \mathsf{PN}\ D\ (E\, a)\big) \to \mathsf{PN}\ D\ \big((a : A) \to (E\, a)\big)\\ \pi_{\mathsf{PN}}\ A\ f &= (\mathsf{pi}\ A\ (\mathsf{proj}_1 \circ f), (g \mapsto (a \mapsto \mathsf{proj}_2\, (fa)\ (ga)))) \ .\end{aligned}$$

Using these ingredients, we can now define composition of PN codes:

6.5.2 Definition of Composition for PN

Theorem 6.5.2.1. *For* $c : \mathsf{Poly}\ D$ *and* $\alpha : \mathsf{Info}\,\mathsf{c} \to E$ *and* $R : \mathsf{PN}\ C\ D$*, define* $(c, \alpha) \bullet R = \mathsf{PN}(\alpha)\, (c/R) : \mathsf{PN}\ C\ E$*, where* $\_/\_ : \big(c : \mathsf{Poly}\ E\big) \to \mathsf{PN}\ D\ E \to \mathsf{PN}\ D\ (\mathsf{Info}\,\mathsf{c})$ *is defined by*

$$\begin{aligned}&\mathsf{id}_{\mathsf{PN}}/R = R\\ &(\mathsf{con}\ A)/R = (\mathsf{con}\ A, \mathsf{id})\\ &(\mathsf{sigma}\ c\ f)/R = (c/R) \gg\!\!=_{\mathsf{PN}} (p \mapsto (f\, p)/R)\\ &(\mathsf{pi}\ A\ f)/R = \pi_{\mathsf{PN}}\ A\ (a \mapsto (fa)/R)\end{aligned}$$

Then $[\![\ R \bullet Q\ ]\!]\ (U, T) \cong [\![\ R\ ]\!]\ ([\![\ Q\ ]\!]\ (U, T))$. □

⋆ ⋆ ⋆

6.6 Examples of Composed Codes

Example 6.6.0.1. *Let us compose* $c_{\mathsf{2W,PN}}$ *from Example 6.3.0.2 with the "upgraded" code* $c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}$ *from Example 6.3.0.1. This time we get the code*

$$\begin{aligned}&\mathsf{sigma}\ (\mathsf{con}\ \{\mathsf{bool}, \mathsf{w}\})\\ &\qquad (\mathsf{bool} \mapsto \mathsf{con}\ 1;\\ &\qquad\ \ \mathsf{w} \mapsto \mathsf{sigma}\ c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}\ ((n, Y) \mapsto \mathsf{pi}\ \big((x : \mathsf{Fin}\, n) \to (Y\ x)\big)\ (\_ \mapsto c_{\mathsf{W}\ \mathbb{N}\ \mathsf{Fin,PN}}))) \ .\end{aligned}$$

Example 6.6.0.2 (Composition of W types). *We recover the well known fact that* W*-types are closed under composition by composing the codes* $(c_{\mathsf{PN,W}\ (S,\ P)}, \alpha)(c_{\mathsf{PN,W}\ (R,\ Q)}, \alpha') :$ $\mathsf{PN}\ 1\ 1$ *of for two* W *types from Example 6.3.0.1. The first component (the second one is trivial) of* $(c_{\mathsf{PN,W}\ (S,\ P)}, \alpha) \bullet (c_{\mathsf{PN,W}\ (R,\ Q)}, \alpha')$ *is given by* $c_{\mathsf{PN,W}\ (S,\ P)}\ /\ (c_{\mathsf{PN,W}\ (R,\ Q)}, \alpha') =$

$$(\mathsf{sigma}\ (\mathsf{con}\ S)\ (s \mapsto \mathsf{pi}\ (P\ s)\ c_{\mathsf{PN,W}\ (R,\ Q)}\ ),\ (s, g) \mapsto (s, (p \mapsto \alpha'))) \ .$$

6.7 Intermediary Systems

In this section we discuss two systems of codes lying between DS and PN. One system we already sketched in Section 6.1 as a motivation for PN; the other system is a more uniform version of this system where codes are annotated with information about their nesting structure. We already mentioned in Chapter 3 that Dybjer-Setzer [40] defined a one level- and a two-level system of codes for induction-recursion and showed that these are equivalent [4]. This raises the question whether our two-level system PN can be reduced to a one-level systems as well. We will not address this question in full generality but restrict ourselves to the discussion of whether the system PN without the constructor for powers of codes can be reduced in his sense. We denote this system by $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ [5] and its intuition is to be constructed by iterating the constructors of DS without adding a constructor of a "different kind" (like π). This restriction is justified by the fact that —to our knowledge— there is no intrinsic relation between the two level system and a constructor for powers beyond the capacity of the former to accommodate the latter. It is rather that the flexibility of the two level systems allows subcodes to have a different type than the code they are a subcode of (compare Section 3.1.3), and as a consequence it allows constructors to generate subcodes of a larger variety of types. The π constructor is an example generating such subcodes of different type[6]. Thus, the question whether there is no translation of PN to DS could be answered negatively by giving translations of PN to $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ and $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ to DS.

6.7.1 $\mathsf{PN}_{\mathsf{cont}}$: an Equivalent Axiomatization of PN

We start by defining a system that is equivalent to PN.This system $\mathsf{PN}_{\mathsf{cont}}$ is defined by the following constructors:

[4]At least after strengthening their logical framework to comprise case distinction for the set 2 into Set [40, Definition 5.3.5] which we have generally assumed anyway.

[5]We can think of this acronym as standing for "iterated container" or "inductive container".

[6]Another possible example would be a constructor for binary products of codes. In principle, the strength of two level systems in the present sense is that one can add constructors changing the index set as long as one finds an appropriate decoding function

Definition 6.7.1.1.

$$
\begin{aligned}
&\mathsf{Poly}_{\mathsf{Cxt}}\ (D\ :\ \mathsf{Set}_1)\ :\ \mathsf{Set}_1\\
&\mathsf{eta}\ :\ \mathsf{Cont}\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D\\
&\mathsf{mu}\ \ :\ (R\ :\ \mathsf{Poly}_{\mathsf{Cxt}}\ D)\ \to\ (\mathsf{Info}_{\mathsf{Cxt}}\ R\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D)\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D\\
&\mathsf{pi}_{\mathsf{Cont}}\ \ :\ (A\ :\ \mathsf{Set})\ \to\ (A\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D)\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Info}_{\mathsf{Cxt}}\ :\ \{D\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D\ \to\ \mathsf{Set}_1\\
&\mathsf{Info}_{\mathsf{Cxt}}\ \{D\}\ (\mathsf{eta}\ (S\ ,\ P))\ \ =\ [\![\ S\ ,\ P\ ]\!]\ D\\
&\mathsf{Info}_{\mathsf{Cxt}}\ (\mathsf{mu}\ R\ f)\ =\ \Sigma\ (\mathsf{Info}_{\mathsf{Cxt}}\ R)\ (\lambda\ x\ \to\ \mathsf{Info}_{\mathsf{Cxt}}\ (f\ x))\\
&\mathsf{Info}_{\mathsf{Cxt}}\ (\mathsf{pi}_{\mathsf{Cont}}\ A\ f)\ =\ \Pi\ A\ (\lambda\ x\ \to\ \mathsf{Info}_{\mathsf{Cxt}}\ (f\ x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{PN}_{\mathsf{cont}} : \mathsf{Set}_1 \to \mathsf{Set}_1 \to \mathsf{Set}_1\\
&\mathsf{PN}_{\mathsf{cont}} D E = [\![\ (\mathsf{Poly}_{\mathsf{Cxt}} D, \mathsf{Info}_{\mathsf{Cxt}})\ ]\!] E
\end{aligned}
$$

The subscript of $\mathsf{PN}_{\mathsf{cont}}$ and its components shall indicate that the base case takes a container (see Definition 1.2.4.1) as argument.

Definition 6.7.1.2 (Monad structure of $\mathsf{PN}_{\mathsf{cont}}\ D\ \_$). *$\mathsf{PN}_{\mathsf{cont}}\ D\ \_$ is again a monad whose unit is given by $\eta_{\mathsf{PN}_{\mathsf{cont}}} : e \mapsto ((\mathsf{eta}\ (1,\ \_ \to 0),\ \_ \mapsto\ e))$, and whose multiplication is given by $\mu_{\mathsf{PN}_{\mathsf{cont}}} : (R,\ f) \mapsto (\mathsf{mu}\ R\ (\mathsf{proj}_1\ \circ\ f),\ (x,\ q) \mapsto (\mathsf{proj}_2\ (f\ x))\ q))$.*

We write in the following $\mathsf{con}_{\mathsf{Cont}}\ A \coloneqq \mathsf{eta}\ (A,\ a \mapsto 0)$.

We denoted the constructors by eta respectively mu because of the relation to the just mentioned monad structure.

Proposition 6.7.1.3. *The systems* PN *and* $\mathsf{PN}_{\mathsf{cont}}$ *can be translated into each other.*

$$
\begin{aligned}
&\mathsf{Tr}_{\mathsf{Cont}}\ :\ \{D\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{Poly}_{\mathsf{Cxt}}\ D\ \to\ \mathsf{Poly}\ D\\
&\mathsf{Tr}_{\mathsf{Cont}}\ (\mathsf{eta}\ (S\ ,\ P))\ =\ \mathsf{sigma}\ (\mathsf{con}\ S)\ (\lambda\ \{(s,\ \star)\ \to\ \mathsf{pi}\ (P\ s)\ (\lambda\ \_\ \to\ \mathsf{id}_{\mathsf{PN}})\ \})\\
&\mathsf{Tr}_{\mathsf{Cont}}\ (\mathsf{mu}\ R\ f)\ =\ \mathsf{sigma}\ (\mathsf{Tr}_{\mathsf{Cont}}\ R)\ (\lambda\ x\ \to\ \mathsf{Tr}_{\mathsf{Cont}}\ (f\ (\mathsf{Tr}_{\mathsf{Pos}}\ R\ x)))\\
&\mathsf{Tr}_{\mathsf{Cont}}\ (\mathsf{pi}_{\mathsf{Cont}}\ A\ f)\ =\ \mathsf{pi}\ A\ (\lambda\ x\ \to\ \mathsf{Tr}_{\mathsf{Cont}}\ (f\ x))
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Tr}_{\mathsf{Pos}}\ :\ \forall\ \{D\}\ \to\ (x\ :\ \mathsf{Poly}_{\mathsf{Cxt}}\ D)\ \to \mathsf{Info}\ \ (\mathsf{Tr}_{\mathsf{Cont}}\ x)\ \to\ \mathsf{Info}_{\mathsf{Cxt}}\ x\\
&\mathsf{Tr}_{\mathsf{Pos}}\ (\mathsf{eta}\ (S\ ,\ P))\ (\ s\ ,\ f)\ =\ s\ ,\ f\\
&\mathsf{Tr}_{\mathsf{Pos}}\ (\mathsf{mu}\ R\ f)\ (x\ ,\ y)\ =\ \mathsf{Tr}_{\mathsf{Pos}}\ R\ x\ ,\ \mathsf{Tr}_{\mathsf{Pos}}\ (f\ (\mathsf{Tr}_{\mathsf{Pos}}\ R\ x))\ y\\
&\mathsf{Tr}_{\mathsf{Pos}}\ (\mathsf{pi}_{\mathsf{Cont}}\ A\ f)\ g\ =\ \lambda\ a\ \to\ \mathsf{Tr}_{\mathsf{Pos}}\ (f\ a)\ (g\ a)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Tr}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{PN}_{\mathsf{cont}}\ D\ E\ \to\ \mathsf{PN}\ D\ E\\
&\mathsf{Tr}\ (c\ ,\ \alpha)\ =\ \mathsf{Tr}_{\mathsf{Cont}}\ c\ ,\ \alpha\ \circ\ (\mathsf{Tr}_{\mathsf{Pos}}\ c)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{rT}_{\mathsf{Cont}} \;:\; \{D \;:\; \mathsf{Set}_1\} \;\to\; \mathsf{Poly}\; D \;\to\; \mathsf{Poly}_{\mathsf{Cxt}}\; D\\
&\mathsf{rT}_{\mathsf{Cont}}\; \mathsf{id}_{\mathsf{PN}} \;=\; \mathsf{eta}\; (1\;,\; (\lambda\; \_ \;\to\; 1))\\
&\mathsf{rT}_{\mathsf{Cont}}\; (\mathsf{con}\; A) \;=\; \mathsf{con}_{\mathsf{Cont}}\; A\\
&\mathsf{rT}_{\mathsf{Cont}}\; (\mathsf{sigma}\; R\; f) \;=\; \mathsf{mu}\; (\mathsf{rT}_{\mathsf{Cont}}\; R)\; (\lambda\; x \;\to\; \mathsf{rT}_{\mathsf{Cont}}\; (f\; (\mathsf{rT}_{\mathsf{Pos}}\; R\; x)))\\
&\mathsf{rT}_{\mathsf{Cont}}\; (\mathsf{pi}\; A\; f) \;=\; \mathsf{pi}_{\mathsf{Cont}}\; A\; (\lambda\; a \;\to\; \mathsf{rT}_{\mathsf{Cont}}\; (f\; a))
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{rT}_{\mathsf{Pos}} \;:\; \forall\, \{D\} \;\to\; (x \;:\; \mathsf{Poly}\; D) \;\to\; \mathsf{Info}_{\mathsf{Cxt}}\; (\mathsf{rT}_{\mathsf{Cont}}\; x) \;\to\; \mathsf{Info}\; x\\
&\mathsf{rT}_{\mathsf{Pos}}\; \mathsf{id}_{\mathsf{PN}}\; (\_\;,\; d) \;=\; d\, \star\\
&\mathsf{rT}_{\mathsf{Pos}}\; (\mathsf{con}\; A)\; (a\;,\; \_) \;=\; a\\
&\mathsf{rT}_{\mathsf{Pos}}\; (\mathsf{sigma}\; S\; T)\; (x\;,\; y) \;=\; (\mathsf{rT}_{\mathsf{Pos}}\; S\; x)\;,\; (\mathsf{rT}_{\mathsf{Pos}}\; (T\; (\mathsf{rT}_{\mathsf{Pos}}\; S\; x))\; y)\\
&\mathsf{rT}_{\mathsf{Pos}}\; (\pi\; A\; T)\; f \;=\; \lambda\; a \;\to\; \mathsf{rT}_{\mathsf{Pos}}\; (T\; a)\; (f\; a)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{rT} \;:\; \{D\; E \;:\; \mathsf{Set}_1\} \;\to\; \mathsf{PN}\; D\; E \;\to\; \mathsf{PN}_{\mathsf{cont}}\; D\; E\\
&\mathsf{rT}\; (F\;,\; \alpha) \;=\; \mathsf{rT}_{\mathsf{Cont}}\; F\;,\; \alpha \;\circ\; \mathsf{rT}_{\mathsf{Pos}}\; F
\end{aligned}
$$

The interest of the system $\mathsf{PN}_{\mathsf{cont}}$ lies in that it reveals that the embedding of DS into PN essentially does not use $\mathsf{pi}_{\mathsf{Cont}}$ — a fact which is rather hidden in the presentation as PN:

Proposition 6.7.1.4. *The map*

$$
\begin{aligned}
&\mathsf{DStoPN} : \mathsf{DS}\; D\; E \to \mathsf{PN}\; D\; E\\
&\mathsf{DStoPN}\; c = (\mathsf{toP}\; c, \mathsf{toI}\; c)
\end{aligned}
$$

where

$$
\begin{aligned}
&\mathsf{toP} : \mathsf{DS}\; D\; E \to \mathsf{Poly\, D}\\
&\mathsf{toP}(\iota\, e) = \mathsf{con}\, 1\\
&\mathsf{toP}(\sigma\, A\, f) = \mathsf{sigma}\, (\mathsf{con} A)\, (\mathsf{toP} \circ f)\\
&\mathsf{toP}(\delta\, A\, F) = \mathsf{sigma}\, (\mathsf{pi}\, A\, (\_ \mapsto \mathsf{id}_{\mathsf{PN}}))\, (\mathsf{toP} \circ F)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{toI} : \big(c : \mathsf{DS}\; D\; E\big) \to \mathsf{Info}\, (\mathsf{toP\, c}) \to E\\
&\mathsf{toI}(\iota\, e)\, \star = e\\
&\mathsf{toI}(\sigma\, A\, f)\, (a, x) = \mathsf{toI}\, (f\, a)\, x\\
&\mathsf{toI}(\delta\, A\, F)\, (g, x) = \mathsf{toI}\, (F\, g)\, x
\end{aligned}
$$

is semantics-preserving. □

6.7.2 $\mathsf{DS} \hookrightarrow \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}} \hookrightarrow \mathsf{PN}_{\mathsf{cont}}$

Proposition 6.7.1.4 should be surprising because on the one hand, the translation of a δ-code is a code containing a pi code, but on the other hand, this translation is semantics preserving and thus the translation of a code decoding to a Σ-type should not suddenly need a Π-type to which pi decodes. This situation can be clarified by the following translation of DS into the system $\mathsf{PN}_{\mathsf{cont}}$ which does not use pi, and can be factored through the system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ that we motivated in Remark 6.1.0.1. In this section we use the $\sigma\delta$-presentations of the system DS (see Definition 3.2.3.2), and DS' (for which we have not introduced a separate notation, see Remark 3.3.0.1); we recall that these four systems are all mutually equivalent (see Section 3.3).

Definition 6.7.2.1.

$$
\begin{aligned}
&\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}(D : \mathsf{Set}_1) : \mathsf{Set}_1\\
&\mathsf{bs} : \mathsf{Cont} \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C} D\\
&\mathsf{it} : (R : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C} D) \to (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P} R \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C} D) \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C} D\\
\\
&\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P} : \{D : \mathsf{Set}_1\} \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C} D \to \mathsf{Set}_1\\
&\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\{D\}(\mathsf{bs}\ R) = [\![\ R\ ]\!] D\\
&\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}(\mathsf{it}\ R\ f) = \Sigma(\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P} R)(\lambda x \to \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}(fx))\\
\\
&\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}} : \mathsf{Set}_1 \to \mathsf{Set}_1 \to \mathsf{Set}_1\\
&\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}} DE = [\![\ (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C} D, \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P})\ ]\!] E\ .
\end{aligned}
$$

Lemma 6.7.2.2 (Translation of DS[7] into $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$). *There is a semantics-preserving translation*

$$
\begin{aligned}
&\mathsf{DStoPN}_{\mathsf{cont}}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{DS}\ D\ E\ \to\ \mathsf{PN}_{\mathsf{cont}}\ D\ E\\
&\mathsf{DStoPN}_{\mathsf{cont}}\ (\iota\ e)\ =\ \mathsf{eta}(1, \lambda_{\_} \to e)\\
&\mathsf{DStoPN}_{\mathsf{cont}}\ (\sigma\delta\ (Q\ ,\ f))\ =\ \mu_{\mathsf{PN}_{\mathsf{cont}}}\ (\mathsf{eta}\ Q\ ,\ (\lambda\ x\ \to\ \mathsf{DStoPN}_{\mathsf{cont}}\ (f\ x)))\ .
\end{aligned}
$$

In particular, this translation does not use π, and thus lands in $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$. In contrast, the translation $\mathsf{toP}(\delta\, A\, F) = \mathsf{sigma}\,(\mathsf{pi}\, A\,(\_ \mapsto \mathsf{id}_{\mathsf{PN}}))\,(\mathsf{toP} \circ F)$ into PN does use π.

It might seem obvious that $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ can be embedded into $\mathsf{PN}_{\mathsf{cont}}$. However, constructively it is not entirely trivial and we want to be absolutely precise about the distinction to be made between having a translation for every *explicitly given* code, and having a *constructively definable* map as in the statement of the following remark. Every explicitly given code in $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ is intuitively constructed by the same constructors as its image in PN - only the names of the constructors have changed and the translation of this code is intuitively its value under an identity map. But this argument does not quite go through constructively because of the partition of the code into two components and the fact that these parts are not defined by a simple induction but by induction-recursion. This distinction shall be our concern also in the remainder of this section.

Remark 6.7.2.3. *'Obviously'* $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ *is a subsystem of* $\mathsf{PN}_{\mathsf{cont}}$. *More formally, we can define*[8] *a map* $\sharp : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}\ D\ E \to \mathsf{PN}\ D\ E$ *by*

[8]We use here (and elsewhere in the Agda files) the "enharmonic" notations $\flat$ for functions that intuitively reduce something and $\sharp$ for functions that intuitively augment something.

$\sharp\mathsf{Cxt} \;:\; \{D \;:\; \mathsf{Set}_1\} \;\to\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\; D \;\to\; \mathsf{Poly}_{\mathsf{Cxt}}\; D$
$\sharp\mathsf{Cxt}\; (\mathsf{bs}\; X) \;=\; \mathsf{eta}\; X$
$\sharp\mathsf{Cxt}\; (\;\; R\, f) \;=\; \mathsf{mu}\; (\sharp\mathsf{Cxt}\; R)\; (\lambda\; z \;\to\; \sharp\mathsf{Cxt}\; (f\; (\flat\mathsf{Pos}\; R\; z)))$

$\sharp\mathsf{Pos} \;:\; \{D\} \;\to\; (Q \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\; D) \;\to\; (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\; Q) \;\to\; \mathsf{Info}_{\mathsf{Cxt}}\; (\sharp\mathsf{Cxt}\; Q)$
$\sharp\mathsf{Pos}\; (\mathsf{bs}\; K)\; x \;=\; x$
$\sharp\mathsf{Pos}\; (\;\; R\, f)\; (x \;,\; z) \;=\; (\sharp\mathsf{Pos}\; R\; x \;,\; \mathsf{subst}\; (\lambda\; y \;\to\; \mathsf{Info}_{\mathsf{Cxt}}\; (\sharp\mathsf{Cxt}\; (f\; y)))\; (\flat\sharp\mathsf{Pos}\; R\; x)\; (\sharp\mathsf{Pos}\; (f\; x)\; z))$

$\flat\mathsf{Pos} \;:\; \{D\} \;\to\; (Q \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\; D) \;\to\; \mathsf{Info}_{\mathsf{Cxt}}\; (\sharp\mathsf{Cxt}\; Q) \;\to\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\; Q$
$\flat\mathsf{Pos}\; (\mathsf{bs}\; K)\; x \;=\; x$
$\flat\mathsf{Pos}\; (\;\; R\, f)\; (x \;,\; z) \;=\; (\flat\mathsf{Pos}\; R\; x \;,\; (\flat\mathsf{Pos}\; (f\; (\flat\mathsf{Pos}\; R\; x)))\; z)$

$\flat\sharp\mathsf{Pos} \;:\; \{D\} \;\to\; (Q \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}C}\; D) \;\to\; (w \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}P}\; Q) \;\to\; (w \;=\; ((\flat\mathsf{Pos}\; Q)\; ((\sharp\mathsf{Pos}\; Q)\; w)))$
$\flat\sharp\mathsf{Pos}\; (\mathsf{bs}\; K)\; w \;=\; \mathsf{refl}$
$\flat\sharp\mathsf{Pos}\; (\;\; R\, f)\; (x \;,\; z) \;=\; \Sigma\equiv\; (\flat\sharp\mathsf{Pos}\; R\; x)\; (q)$

$\sharp : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}\; D\; E \to \mathsf{PN}\; D\; E$
$\sharp = [\![\; (\sharp\mathsf{Cxt}, \sharp\mathsf{Pos})\;]\!] E$

where in the last line we used 'container notation' Definition 1.2.4.1, and to spell out the term q requires more definitions for equational reasoning than we gave in Paragraph 2.2.3.3.10 and hence we refer to the Agda files.

6.7.3 $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \hookrightarrow \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$, an Annotated version of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$

6.7.3.1 Nesting of Codes

The observation that the inclusion of DS into $\mathsf{PN}_{\mathsf{cont}}$ does not use pi, and factors through the system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ which —like DS— does not contain a constructor for dependent products of codes, raises the question whether there is a translation $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}} DE \to \mathsf{DS} DE$. We could not decide this question in full generality but we found a translation of a subsystem $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$ of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ that is a more uniform version of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ that additionally carries an annotation of codes by information about their shape to DS. This partial solution is informed by the observation what is problematic in finding a translation from $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ to DS: $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ arises by iterating application of the $\sigma\delta$-constructor of DS in 'contexts'. Assuming that we base a translation on an operation reducing the degree of the nestings of 'contexts' arising from this iteration in steps of one, the translation would need to keep track of how often this operation needs to be applied until we arrive at a 'context' with a nesting degree of zero which amounts to it being just a container. But here again the problem arises that the nesting degrees may differ in different paths of a code. The system $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$ answers this problem by constraining the codes to have uniform path length an annotating every code by information about its nesting. It turns out that a convenient way to formalize the shape of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ codes is as binary trees $\mathsf{bt} : \mathsf{Set}$ defined by the constructors

$$\begin{aligned} &\mathsf{leaf} \ : \ \mathsf{bt} \\ &\mathsf{branch} \ : \ \mathsf{bt} \ \rightarrow \ \mathsf{bt} \ \rightarrow \ \mathsf{bt} \end{aligned}$$

Without further pursuing it, we mention that the set of binary trees carries a simple well-order relation and is an example of an ordinal notation system [82]. Thus, in principle, one could carry out the following also by annotating codes by natural numbers. This is what we tried in an earlier version of this section before we found the more convenient option of annotating by binary trees which is closer to the structure of codes.

This annotated version is subject of the following definition:

Definition 6.7.3.1. *We define the (large) container* $(\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}, \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}})$ *by the following inductive-recursive definition.*

$$\begin{aligned} &\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ (D \ : \ \mathsf{Set}_1) : \ \mathsf{bt} \ \rightarrow \ \mathsf{Set}_1 \\ &\mathsf{bs}_{\mathsf{bt}} \ : \ \mathsf{Cont} \ \rightarrow \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ D \ \mathsf{leaf} \\ &\mathsf{it}_{\mathsf{bt}} \ : \ (xx \ yy \ : \ \mathsf{bt}) \ \rightarrow \ (R \ : \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ D \ xx) \ \rightarrow \\ &(\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}} \ R \ \rightarrow \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ D \ yy) \ \rightarrow \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ D \ (\mathsf{branch} \ xx \ yy) \end{aligned}$$

$$\begin{aligned} &\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}} \ : \ \{D \ : \ \mathsf{Set}_1\} \ \{xx \ : \ \mathsf{bt}\} \ \rightarrow \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ D \ xx \ \rightarrow \ \mathsf{Set}_1 \\ &\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}} \ \{D\} \ (\mathsf{bs}_{\mathsf{bt}} \ R) \ = \ [\![\ R \ ]\!] \ D \\ &\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}} \ (\mathsf{it}_{\mathsf{bt}} \ xx \ yy \ Q \ h) \ = \ \Sigma \ (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}} \ Q) \ (\lambda \ x \ \rightarrow \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}} \ (h \ x)) \end{aligned}$$

The system of annotated uniform polynomial codes is again defined by container evaluation:

$$\begin{aligned} &\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \ : \{D \ E \ : \ \mathsf{Set}_1\} \ \rightarrow \ \mathsf{Set}_1 \ \rightarrow \ (xx \ : \ \mathsf{bt}) \ \rightarrow \ \mathsf{Set}_1 \\ &\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \ D \ E \ xx \ = \ [\![\ (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}} \ D \ xx \ , \ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}}) \ ]\!] \ E \end{aligned}$$

Decoding of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$ is given by simply forgetting the annotation and decoding as a $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ code. As a container $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$ is of course again a functor.

We can think of the constructor $\mathsf{it}_{\mathsf{bt}}$ (or it in the version without annotation) as producing iterations of *nestings* of the base case.

6.7.3.2 A Translation $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \to \mathsf{DS}$

We give now a translation $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \to \mathsf{DS}$ the components of which we summarize in the diagram at the end of Paragraph 6.7.3.2.4.

6.7.3.2.1 Reducing the Degree of Nesting of Codes We want to compare this system to an annotated version of Dybjer-Setzer's system DS' (aka $(\mathsf{SP}, \mathsf{Arg})$) Section 3.3. For codes with no nesting we already have a coincidence:

$$\begin{aligned}
&\flat_{\mathsf{leaf}} \;:\; \{D\;E \;:\; \mathsf{Set}_1\} \;\to\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; \mathsf{leaf} \;\to\; \mathsf{DS}'\; D\; E\\
&\flat_{\mathsf{leaf}}\; (\mathsf{bs}_{\mathsf{bt}}\; X\;,\; f) \;=\; (\iota'\; X\;,\; f)
\end{aligned}$$

This indicates that we can obtain a translation if we can reduce the degree of the nesting in $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$. The basic idea for this reduction is given by the following operation:

Definition 6.7.3.2 (A "differential operator" for codes).

$$\begin{aligned}
&\Delta \;:\; \{D\;E\}\{xx\;yy \;:\; \mathsf{bt}\}\{Q \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}}\; D\; xx\;\} \;\to\; (h \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}}\; Q \;\to\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContCbt}}\; D\; yy) \;\to\\
&\qquad (x \;:\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}}\; Q) \;\to\; (f \;:\; (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{ContPbt}}\; (\mathsf{it}_{\mathsf{bt}}\; xx\; yy\; Q\; h)) \;\to\; E) \;\to\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; yy\\
&\Delta\; h\; x\; f \;=\; (h\; x\;,\; \lambda\, q \;\to\; f\; (x\;,\; q))
\end{aligned}$$

We use the symbol 'Δ' (and below $\not{d}$, $\not{d}\not{d}$ etc.) as a reminiscience of "derivative": the intuition is here the rule $\frac{d}{dx}x^n = n \cdot x^{n-1}$ for differentiating powers of real numbers where the "iteration" of n-times applying x to itself is replaced by the number being the $(n-1)$-times iteration of the application of x to itself combined with a "scaling" by a factor of n. This will become even more clear in the following definition where a code $c : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; (\mathsf{branch}\; xx\; yy)$ with index $\mathsf{branch}\; xx\; yy$ is replaced by a code $\not{d}\; c : \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; yy\;)\; xx$ with index xx in a system "scaled" by yy.

The extension of Δ to codes is a dependent function; we first have to define the codomain $\mathsf{Cod}_{\not{d}}$:

Definition 6.7.3.3 ("Differentiating" codes).

$$\begin{aligned}
&\mathsf{Cod}_{\not{d}} \;:\; \{D\;E \;:\; \mathsf{Set}_1\} \;\to\; \mathsf{bt} \;\to\; \mathsf{Set}_1\\
&\mathsf{Cod}_{\not{d}}\; \{D\}\; \{E\}\; (\mathsf{leaf}) \;=\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; \mathsf{leaf}\\
&\mathsf{Cod}_{\not{d}}\; \{D\}\; \{E\}\; (\mathsf{branch}\; xx\; yy\;) \;=\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; yy\;)\; xx
\end{aligned}$$

$$\begin{aligned}
&\not{d} \;:\; \{D\;E \;:\; \mathsf{Set}_1\} \;\to\; \{ww \;:\; \mathsf{bt}\} \;\to\; \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\; D\; E\; ww \;\to\; \mathsf{Cod}_{\not{d}}\; \{D\}\; \{E\}\; ww\\
&\not{d}\; (\mathsf{bs}_{\mathsf{bt}}\; R\;,\; f) \;=\; (\mathsf{bs}_{\mathsf{bt}}\; R\;,\; f)\\
&\not{d}\; (\mathsf{it}_{\mathsf{bt}}\; xx\; yy\; Q\; h\;,\; f) \;=\; (\;Q\;,\; \lambda\, x \;\to\; \Delta\; \{Q\}\; h\; x\; f)
\end{aligned}$$

In prose, the previous definition says that we can reduce the 'outer' degree of a code from ($\mathsf{branch}\ xx\ yy$) to xx - albeit at the cost of transforming the set E to which c decodes, to a set of codes of degree yy. Notice that the definition of $đ$ is possible also for the systems $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ without the annotation of codes; the latter becomes relevant only later on. Our intuition of the yin-and-yang of differentiation and its reverse of integration is backed up by the following observation (which is true also for the system without annotation):

Lemma 6.7.3.4 (Justification of $đ$). *1. $đ$ acts as retraction of $\mu^{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}}$ on* it *codes, i.e.there is a definitional equality*

$$\mu^{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}}\ (đ\ (\mathsf{it}_{\mathsf{bt}}\ xx\ yy\ R\ h\ ,\ f))\ =\ (\mathsf{it}_{\mathsf{bt}}\ xx\ yy\ R\ h\ ,\ f)\ .$$

2. $đ$ acts as the identity on $\mathsf{bs}_{\mathsf{bt}}$ *codes.*

Using $đ$ as the basis of our translation, the idea is now to iterate this process of "differentiation" until all of the occurring sets of codes are indexed by leaf such that we have 'flattened' the set of codes $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ E\ ww$ to an iterated set of codes $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ (...\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ E\ \mathsf{leaf}...)\ \mathsf{leaf})\ \mathsf{leaf}$. This is accomplished by the following dependent function $đđ$ which iteratively applies $đ$.

Definition 6.7.3.5 (Iterating the "differentiation" of codes).

$$\begin{aligned}
&\mathsf{Cod}_{đđ}\ :\ (D\ E\ :\ \mathsf{Set}_1)\ \to\ \mathsf{bt}\ \to\ \mathsf{Set}_1\\
&\mathsf{Cod}_{đđ}\ D\ E\ \mathsf{leaf}\ =\ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ E\ \mathsf{leaf}\\
&\mathsf{Cod}_{đđ}\ D\ E\ (\mathsf{branch}\ xx\ yy)\ =\ \mathsf{Cod}_{đđ}\ D\ (\mathsf{Cod}_{đ}\ \{D\}\ \{E\}\ yy)\ xx
\end{aligned}$$

$$\begin{aligned}
&đđ\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ \{ww\ :\ \mathsf{bt}\}\ \to\ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ E\ ww\ \to\ \mathsf{Cod}_{đđ}\ D\ E\ ww\\
&đđ\ (\mathsf{bs}_{\mathsf{bt}}\ R\ ,\ f)\ =\ (\mathsf{bs}_{\mathsf{bt}}\ R\ ,\ f)\\
&đđ(\mathsf{it}_{\mathsf{bt}}\ xx\ yy\ Q\ h\ ,\ f)\ =\\
&\qquad đđ\ (\mathsf{proj}_1\ (đ\ (\mathsf{it}_{\mathsf{bt}}\ xx\ yy\ Q\ h\ ,\ f))\ ,\ \lambda\, x\ \to\ \ (đ\ ((\mathsf{proj}_2\ (đ\ (\mathsf{it}_{\mathsf{bt}}\ xx\ yy\ Q\ h\ ,\ f)))x)))\ .
\end{aligned}$$

6.7.3.2.2 Translating $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$-Codes into Annotated DS'-Codes

As an intermediary step in the intended translation, we define a function to an annotated version of the system DS'

$$\begin{aligned}
&\mathsf{DS}'_{\mathsf{bt}}\ :\ (D\ E\ :\ \mathsf{Set}_1)\ \to\ \mathsf{bt}\ \to\ \mathsf{Set}_1\\
&\mathsf{DS}'_{\mathsf{bt}}\ D\ E\ \mathsf{leaf}\ =\ \mathsf{DS}'\ D\ E\\
&\mathsf{DS}'_{\mathsf{bt}}\ D\ E\ (\mathsf{branch}\ xx\ yy)\ =\ \mathsf{DS}'_{\mathsf{bt}}\ D\ (\mathsf{DS}'_{\mathsf{bt}}\ D\ E\ yy)\ xx
\end{aligned}$$

$$\begin{aligned}
&\mathsf{redu}'\ :\ (D\ :\ \mathsf{Set}_1)\ \to\ \{E\ :\ \mathsf{Set}_1\}\ \to\ (xx\ yy\ :\ \mathsf{bt})\ \to\\
&\qquad \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ (\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ E\ yy\ )\ xx\ \ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D(\mathsf{DS}'_{\mathsf{bt}}\ D\ E\ yy)\ xx
\end{aligned}$$

(We omit here and in the following several definitions for simplicity of presentation since they involve further definitions distracting from the main line of the argument; the complete set of definitions can be found in the Agda files. The definition of redu' involves $\flat_{\mathsf{leaf}}$, $\mathscr{d}$, as well as functoriality of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ and $\mathsf{DS}'_{\mathsf{bt}}$.) From this we obtain

$$
\begin{array}{l}
\mathsf{redu}\ :\ (D\ :\ \mathsf{Set}_1)\ \to\ \{E\ :\ \mathsf{Set}_1\}\ \to\ (yy\ :\ \mathsf{bt})\ \to\ \mathsf{Cod}_{\mathscr{d}}\ \{D\}\ \{E\}\ yy\ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ yy\\
\mathsf{redu}\ D\ \mathsf{leaf}\ =\ \flat_{\mathsf{leaf}}\\
\mathsf{redu}\ D\ \{E\}(\mathsf{branch}\ xx\ yy)\ w\ =\ \mathsf{redu}'\ D\ \{E\}\ xx\ yy\ w
\end{array}
$$

and the corresponding version for the iterated version of the function reducing the nesting

$$\mathsf{Conv}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ (xx\ :\ \mathsf{bt})\ \to\ \mathsf{Cod}_{\mathscr{dd}}\ D\ E\ xx\ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ xx\ .$$

(The definition of Conv uses $\flat_{\mathsf{leaf}}$ as well as functoriality of $\mathsf{Cod}_{\mathscr{dd}}$.)

6.7.3.2.3 Defining the Last Component of the Translation The last step of our translation is to use the monad structure of $\mathsf{DS}'_{\mathsf{bt}}$; for $\mathsf{DS}'_{\mathsf{bt}}\ D_{\_}\ \mathsf{leaf}$ the monad multiplication is of course simply that of DS' (see Section 3.3)

$$
\begin{array}{l}
\mu^{\mathsf{leaf}}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D\ (\mathsf{DS}'_{\mathsf{bt}}\ D\ E\ \mathsf{leaf})\ \mathsf{leaf}\ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ \mathsf{leaf}\\
\mu^{\mathsf{leaf}}\ x\ =\ \mu\ x\ .
\end{array}
$$

For index trees other that leaf we obtain an appropriate multiplication by iterating μ^{leaf}

$$\mathsf{MU}\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\ \to\ \{xx\ :\ \mathsf{bt}\}\ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ xx\ \to\ \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ \mathsf{leaf}$$

(where the definition of MU involves functoriality of $\mathsf{DS}'_{\mathsf{bt}}$).

6.7.3.2.4 Completing the Translation $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont\,bt}} \to \mathsf{DS}'$ In total, a translation is given by:

Proposition 6.7.3.6.

$$
\begin{array}{l}
\flat\, Trans\ :\ \{D\ E\ :\ \mathsf{Set}_1\}\{xx\ :\ \mathsf{bt}\}\ \to\ \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont\,bt}}\ D\ E\ xx\ \to\ \mathsf{DS}'\ D\ E\\
\flat\, Trans\ \{D\}\{E\}\{xx\}\ =\ \mathsf{MU}\ \{xx\}\ \circ (\ \mathsf{Conv}\ xx)\ \circ\ \mathscr{dd}\ \{D\}\{E\}\{xx\}
\end{array}
$$

□

We summarize the components in the following commutative diagram:

$$\begin{tikzcd}
 & \mathsf{Cod}_{dd}\ D\ E\ xx \arrow[r, "\mathsf{Conv}"] & \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ xx \arrow[dr, "\mathsf{MU}"] & \\
\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}\ D\ Exx \arrow[ur, "dd"] \arrow[rrr, "\flat Trans"] & & & \mathsf{DS}'_{\mathsf{bt}}\ D\ E\ \mathsf{leaf} \simeq \mathsf{DS}\ D\ E
\end{tikzcd}$$

One way to read the statement that the translation $\flat Trans = \mathsf{MU} \circ \mathsf{Conv} \circ dd$ preserves semantics (up to equivalence) is the statement that "$\mathsf{MU} \circ \mathsf{Conv}$ is a retraction of dd in semantics", i.e. that $[\![(\mathsf{MU} \circ \mathsf{Conv})\ \_]\!] \circ [\![dd\,\_]\!] \simeq [\![\_]\!]$. This extends Lemma 6.7.3.4 saying that d has a retraction (given by $\mu^{\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}}$ on codes on which it does not act trivially). There was unfortunately no time to formalize this.

⋆ ⋆ ⋆

Chapter 7

Relational Parametricity for DS

This chapter is concerned with a relationally-parametric model of MLTT + DS (see Chapter 3 for a discussion of Dybjer-Setzer's axiomatization DS of induction-recursion) by establishing a category-with-families structure (see Section 7.5) on the category of reflexive graphs (see Section 7.6.2) — the latter being an abstraction of reflexive relations.

Because we shall need several general definitions that previous chapters did not need, this chapter contains a part introducing these basics; specifically, there is a discussion of the category of families of reflexive graphs (in Section 7.6.2), a category-with-families structure which will constitute a relational-parametric model of MLTT, Section 7.7 on unary-relational parametricity contains some definitions explaining how to obtain unary parametricity as a degenerate case of binary one. Paragraph 7.6.3.1.3 provides a binary-relational model of MLTT+DS by providing the mentioned category-with families structure and interpreting the rules of DS in it.

7.1 Studying Type Theory from Within Type Theory

In a previous paper [37] I introduced a general notion of *simultaneous inductive-recursive definition* in intuitionistic type theory. This notion subsumes various reflection principles and seems to pave the way for a natural development of what could be called "internal type theory", that is, the construction of models of (fragments of) type theory in type theory, and more generally, the formalization of the metatheory of type theory in type theory.
The present paper is a first investigation of such an internal type theory.

Peter Dybjer, "internal type theory", 1992, [34]

The metatheory of (fragments of) type theory, to which Peter Dybjer is referring in the passage quoted in the epigraph, is formulated in the paper from which the quote is taken in form of *categories with families* Section 7.5 as a vehicle to reason about metatheoretic properties that are —as the prefix (meta-) itself expresses– not accessible from within type theory itself.

One collection of such metatheoretic properties consists of *properties of substitution.* From this viewpoint the rules of type theory are regarded as rules regulating the use of variables in type definitions. Type definitions can from this operational point of view be seen as transformations of the "ingoing" variables (or terms) used in the premises of a rule to the "outgoing" variables (or terms) in the conclusion of the rule.

Thus, one metatheoretic question is whether and which relations between the ingoing variables or terms are preserved by the transformation associated to a type definition. For example, the rules for judgmental equality assert that type formers preserve the equality relation. If generally *all* relations are preserved by a type definition, it is called *relationally parametric*, and if all types in the type theory under discussion globally can be ascertained to have this property, it is called a relationally-parametric theory. For example, an inhabitant of the polymorphic endomorphism type $f : \Pi_{X:\mathsf{Type}}(X \to X)$ is relationally parametric, if for all $X, Y : \mathsf{Type}$, all relations $R \subseteq X \times Y$, if $(x, y) \in R$, then $(f\ X\ x, f\ Y\ y) \in R$. It is of interest that this property is satisfied since it rules out the possibility that the type $\Pi_{X:\mathsf{Type}}(X \to X)$ contains any other function than the polymorphic identity $\lambda X \to id_X$; without the parametricity property this type would also contain for example $\lambda X \to (\lambda x \to (0, \text{if } X = \{0, 1\}; x,\ \text{else}))$ which is not in the spirit of parametric polymorphism where definitions should be "uniform" and not depend on ad-hoc checking of input types.

We said that the question about the preservation of relations is metatheoretic since there is no infrastructure present on the level of MLTT itself to deal with these relations. One way to nevertheless pursue this inquiry is therefore to switch to models of type theory. A relationally-parametric model of the type theory in question is then a model where each type is modeled as a relation.

This model theoretic approach is also called external parametricity about which we say more in Section 7.3.0.1. There is also a notion of internal parametricity which is pursued by adding rules dealing with the relations to MLTT itself but we will not be interested in this option in this thesis.

7.1.1 The Significance of Ralational-Parametricity for DS

The interest of formulating relational-parametricity for DS lies of course in the desire not to have to formulate a parametricity statement (like the above-mentioned one that applies to function types only) for every type defined in the theory separately, but to be able to formulate one such statement that applies to all types defined by DS.

7.2 Further Motivations-, and Consequences of Relational-Parametricity

Having (informally) explained what relational parametricity is, we come to the question why it is interesting. Usually relational parametricity is introduced by a frame narrative woven together from category-theory inspired applications ("functoriality without composition" [15][62] allowing for what has been called "free theorems" [126]) together with an assorted import of universal properties ("parametric limits" [59][31]) whose equivalents can be defined by reference to functors in category theory, and foundational concerns related to impredicativity.

The first mentioned plot comes from the attempt to generalize, appropriate, or make a semblance between notions from category theory, the "abstract algebra of functional relations", and relational parametricity whose abstract theory one could call the "abstract algebra of reflexive relations".

The second mentioned story line around foundational problems posed by impredicativity is informed by the fact that to obtain interesting implications from relational-parametricity in (dependent) type theory, one needs a version of polymorphic type theory (see [106] where relational-parametricity was introduced), i.e. a version of type theory having dependent function types[1] ranging over all (or at least a type of some) types. In the course of considering relational parametricity in conjunction with impredicative quantification also paradoxes are discussed [28].

[1] [106] uses the terminology of *higher order functions*.

A third, rather meta-, or model-theoretic perspective of relational parametricity is related to the first two perspectives. One important example for a model theoretic problem about type theory is whether a semantic term[2] of a semantic type is of *canonical form* in the sense that it is composed of semantic terms of constructors of the type whose semantic type is in discussion. As such, a general way to construe relational-parametricity is to say that it bridges a possible mismatch between semantical expectations and syntactical truth and -provability in typed theories.

A derived sort of problems is whether canonicity (if it holds) is preserved by operations on types. For example [84] discusses the preservation of canonicity under subtyping and what happens to canonicity in presence of a cumulative hierarchy of universes. [120] calls properties like this *semantic properties*, elsewhere they are also called *metatheoretical properties* and to resolve them loc.cit. suggests ways to make the problem approachable on the level of the object theory.

There is no motivation to study relational parametricity for induction-recursion in excess of the motivation to study it for the types defined by induction-recursion.

7.3 External-, and Internal Parametricity

There are two varieties of (relational) parametricity: external-, and internal one. We are in this thesis only interested in the external variant but mention the internal one for the sake of completeness.

7.3.0.1 External Parametricity

Given the distinction of object-, and meta theory it is not surprising that problems of parametricity were discovered in cases where different levels are present already on the side of the object theory: Girard [54] and Reynolds [106] studied the so-called *polymorphic lambda calculus* (aka system F) where a type Type of (some) types (i.e. a universe) together with the possibility to quantify over Type is present. This leads Reynolds to talk about "higher order functions" —which are more commonly called polymorphic functions— by which he means functions mapping types to functions. One example is the so-called the "polymorphic endomorphism type" $\Pi_{X:\mathsf{Type}}(X \to X)$. Reynolds' theory of parametricity contains an *abstraction theorem* which is so called because parametricity statements about particular functions such as inhabitants of $X \to X$ are "abstracted" to inhabitants of $\Pi_{X:\mathsf{Type}}(X \to X)$, i.e. to endomorphism of *any generic* set. In the above-mentioned example of the polymorphic endomorphism type, that relational-parametricity is satisfied in a model, means that if $f : \Pi_{X:\mathsf{Type}}(X \to X)$, then Type is interpreted as the (large) set of all relations bewteen sets, and we will see in Paragraph 7.6.3.1.2 how to express the relational-parametricity statement mentioned above[3] can be expressed in this model.

[2]i.e. the interpretation of a term in the model. Likewise, a semantic type is the interpretation of a type in the model.

[3]i.e. f is relationally parametric, if for all $X, Y : \mathsf{Type}$, all relations $R \subseteq X \times Y$, if $(x, y) \in R$, then $(f\ X\ x, f\ Y\ y) \in R$.

Moreover Reynolds proves an *identity-extension theorem* expressing that the identity relation is respected by the semantic term of a semantic polymorphic function type.

Following the "algebraic" trend to discuss a type always together with relations (such as equality), propositions, predicates, etc. on them, Reynolds sets out to give a formal account of this tandem in terms of a relational-parametric model. Since he was concerned with the polymorphic lambda calculus, his work did not comprise a treatment of inductive types, but the extension to the latter has been addressed in later literature such as [61] sometimes under the name of *logical relations* or *predicate-, and relations lifting* where the latter terminology originates in the initial-algebra semantics practice of considering algebras for endofuctors $F : B \to B$ and then " lifting" this functor to a functor $\hat{F} : Prop(B) \to Prop(B)$ along appropriate fibrations $p : Prop(B) \to B$ (where Prop(B) denotes the category of propositions on elements of B) in a way that complies with the formation of initial algebras for both functors.

Even though [61] does treat relational-parametric ideas in a fibrational setting and thus in a rudimentary model of type dependency, it does not cover all aspects that are of type theoretical interest: 1) the discussion is situated entirely in fibrational category theory and the relation to type theory is left implicit or has example character, 2) the focus of the paper —as far as examples are presented— is on logic or simple type theory (where type variables are absent), 3) there are types in Martin-Löf type theory that are not (merely) inductive (such as universes) and hence are not discussed, 4) the fibrational framework used there does not in general model type theory strictly; moreover, while equalities between types can be dealt with on the relations level, there is not provided a formal way to deal with equalities between relations.

[12] addresses all the above mentioned points except 3) by giving a model of MLTT with a universe, Π-types, and a natural-numbers type (but without identity type) in form of a category-with-families structure (the notion the epigraph Section 7.1 was referring to) on the category of reflexive graphs. This models type theory strictly (i.e. substitution is strictly associative as opposed to be associative only up to an isomorphism).

7.3.0.2 Internal Parametricity

Instead of expressing parametricity statements in a model of type theory, one can also add rules to the type theory expressing parametricity statements. See [17][16][10] for discussion, bibliography, context and applications of the topic.

7.4 Bibliography on Relational Parametricity

To provide further background, we give a selective bibliography on relational parametricity, the selection criterion being a preference for theoretical rather than applied articles.

- Reynolds, "Types, Abstraction and Parametric Polymorphism" [106], 1983. Introduces relational-parametricity by giving a relational-parametric model of system F; formulated an *abstraction theorem*, and an *identity-extension lemma*.

- Bainbridge, Freyd, Scedrov, and Scott, “Functorial Polymorphism” [15], 1990, addresses certain foundational set-theoretic problems arising from impredicativity via relational parametricity.

- R. Hasegawa, “Categorical Data Types in Parametric Polymorphism” [60], 1994. Starts a category-theoretic approach to relational-parametricity.

- R. Hasagawa “Relational Limits in General Polymorphism” [59], 1994, studies a notion of relational-parametric limits.

- Robinson and Rosolini, 1994 “Reflexive Graphs and Parametric Poly- morphism” [107] studies Reynolds’ model via internal category theory motivated by foundational problems related to impredicativity.

- Dumphy and Reddy, “Parametric Limits” [31], 2004, formulates a general notion of *parametric limit* inspired by the eponymous notion from category theory.

- Takeuti, “The Theory of Parametricity in the Lambda Cube” [120], 2001. Extends the discussion of relational-parametricity to dependent types.

- Atkey, “Relational Parametricity for Higher Kinds” [11], 2012, Extends relational parametricity to higher kinded polymorphism, which allows quantification over type operators and types, and presents a model of relational parametricity for System $F\omega$, within the impredicative Calculus of Inductive interprets inductive types indexed by an arbitrary kind, and discusses initiality in terms of impredicative quantification.

- Atkey, Ghani, and Johann, “A Relationally Parametric Model of Dependent Type Theory” [12], 2014. First relational-parametric model of dependent type theory that can prove the existence for (internally defined) so-called “indexed functors” (i.e. functors between categories of families fixing the indexing set).

- Krishnaswami and Dreyer, “ Internalizing Relational Parametricity in the Extensional Calculus of Constructions” [97], 2013. Construct a realizability model of the Calculus of Constructions, using Quasi-PERs to simultanously define the underlying and relational interpretations of types.

- Hermida, “Logical Relations and Parametricity – A Reynolds Programme for Category Theory and Programming Languages” [62], 2014. A programmatic article emphasizing the category-theoretics viewpoint on relational-parametricity.

More on the side of applications and not directly aimed at furthering the theory of relational parametricity:

- Wadler, “Theorems for Free!” [126], 1989. Studies instances of the parametricity statement (called “free theorems”).

- Coquand, “A New Paradox in Type Theory” [28], 1995. Derives a paradox in type theory from impredicative quantification and instances of the parametricity statement.

7.5 Categories with families

Categories with families (CwFs) [34] are a well studied class of categorical models of dependent type theory providing sound and complete models of dependent type theories and a close and intuitive correspondence between object- and meta-theory (that is, between the theory being modeled and the theory in which the model is situated), e.g. dependent type theory can be exhibited as the initial category with families [23]. CwFs can be formalized in type theory itself and thus contribute —like induction-recursion— to the project of "internalising" type theory in itself. An advantage of CwFs as compared to other (categorical) models of type theory is that it models substitution in a strict way, whereas other fibrational models render substitution in a weak way up to the isomorphism granted by the universal property of pullbacks.

CwFs are only a model of type *dependency*, in particular the existence of particular types defined by introduction-, and elimination rules is not automatic in a CwF and has to be established separately. Formally, this is accomplished by exhibiting the CwF as a generalised algebraic theory (GAT) [22] and adding further operator symbols and equations to this GAT recovering the inference rules formulating the introduction and elimination of types. In case of MLTT, the underlying GAT has to be extended to cover dependent sum- and dependent product types, W-types, and universes (for the example of Π-types see, e.g. [34, §2.2]).

It is important to notice that we make two adaptions regarding the categories of families (see Section 1.3) in which our model will be situated: first we have to consider families of large sets i.e. $\mathsf{Fam}(\mathsf{Set}_1) := \Sigma_{A:\mathsf{Set}_1}(A \to \mathsf{Set}_1)$, and second, we are considering now also non-cartesian morphisms in the category $\mathsf{Fam}(\mathsf{Set}_1)$ in case it figures in a CwF structure.

Definition 7.5.0.1 (Category with families (CwF)). *A* category with families *consists of a category $\mathcal{C} \in \mathsf{Cat}(\mathsf{Set}_1)$ with a chosen terminal object, and a functor $\mathsf{T} = (\mathsf{Ty}, \mathsf{Tm}) : \mathcal{C}^{\mathsf{op}} \to \mathsf{Fam}(\mathsf{Set}_1)$ equipped with an operation*[4] $\_ \bullet \_ : (C \in Ob\,\mathcal{C}) \to \mathsf{Ty}\ C \to C$ *(called* comprehension*) such that*

$$Hom_{\mathcal{C}}(C', C \bullet A) \simeq \{(f, M) \mid f : C' \to C, M \in \mathsf{Tm}\ C'\ (\mathsf{Ty}\ f(A))\}\ .$$

The function $\mathsf{Ty}\ f$ *is also called* type substitution*, and the transformation* $\mathsf{Tm}\ f$ *is called* term substitution*. For emphasis, the morphism* $\mathsf{T} =$

$$\begin{tikzcd}
\mathsf{Ty}\ C' \arrow[rr, "\mathsf{Ty}\ f"] \arrow[dr, "\mathsf{Tm}\ C'"'] & \mathsf{Tm}\ f & \mathsf{Ty}\ C \arrow[dl, "\mathsf{Tm}\ C"] \\
& \mathsf{Set}_1 &
\end{tikzcd}$$

[4]Some properties of this operation can be expressed in terms of relating the monad structure of Fam and the functor T.

is not necessarily cartesian and as such $\mathsf{Tm}\ f\ : (x : \mathsf{Ty}\ C') \to \mathsf{Tm}\ C'\ x \to \mathsf{Tm}\ C\ (\mathsf{Ty}\ f x)$ *is not necessarily a family of identities.*

Given a type theory (i.e. in syntactical form), the *term model* is intended to be the closest representation possible of the type theory "in terms" of a CwF.

Definition 7.5.0.2 (Term model). *[63, Example 1] spells out the standard example of a CwF*[5] *- called the* term model C^T.

For a type theory T, the category C^T is defined to have as objects contexts of T, and as morphisms substitutions. The functor $(\mathsf{Ty}, \mathsf{Tm}) : C^{op} \to \mathsf{Fam}(\mathsf{Set})$ *is defined to have as value* $\mathsf{Ty}\ \Gamma$ *the set of types in context* Γ*, and for* $A \in \mathsf{Ty}\ \Gamma$*, the set* $\mathsf{Tm}\ \Gamma\ (A)$ *is that of terms of A. The action on morphisms applies substitutions to types and terms. Comprehension is defined by the rule for context extension in the type theory T.*

The term model of a type theory T is distinguished in that it is initial in the category of CwF models of T; i.e. if $F : C \to \mathsf{Fam}(\mathsf{Set}_1)$ *is any CwF model of T, then there is a functor* $H : C^T \to C$ *preserving the terminal object and a transformation* $h : F^T \to F \circ H$ *from the term model* $F^T : C^T \to \mathsf{Fam}(\mathsf{Set}_1)$.

7.6 A Reflexive-Graph Model of MLTT

In this section we will recall a reflexive-graph model for MLTT with Sigma type, Pi types, and two universes $\mathsf{Set} : \mathsf{Set}_1$; this model is the one presented in [12] except that loc.cit. deals with a version of MLTT with only one universe.

[5] To be precise, loc. cit. talks about so-called 'categories with attributes' - but this notion is equivalent to that of a CwF as stated in [34, §2.1] and proved in [64].

7.6.1 Reflexive Graphs

Definition 7.6.1.1 (RG)**.** *We define the category* RG *by* $Ob\,\mathsf{RG} = \{\mathsf{0}, \mathsf{1}\}$ *and* $Mor\,\mathsf{RG}$ *is generated by the set* $\{s : \mathsf{1} \to \mathsf{0}, t : \mathsf{1} \to \mathsf{0}, r : \mathsf{0} \to \mathsf{1}\}$*, identities, and the equation* $sr = tr = id_\mathsf{0}$*; in other words* $Mor\,\mathsf{RG} = \{\mathsf{id}_\mathsf{0}, \mathsf{id}_\mathsf{1}, r, s, t, r \circ s, r \circ t, s \circ r\}$*.*

Definition 7.6.1.2 ($\mathcal{RG}$**, Reflexive Graph).** *We define the functor category* $\mathcal{RG} :=$ $(\mathsf{RG} \to \mathsf{SET}_1)$ *objects of which we call* reflexive graphs[6]*. A* morphism of reflexive graphs *is defined to be a natural transformation of the underlying functors.*

A discrete reflexive graph *is a reflexive graphs that sends all morphisms to identities.*

For $\mathcal{A} : \mathcal{RG}_\Gamma$ we will sometimes use the notations $\mathcal{A}(\mathsf{0}) = \mathcal{A}_0$, $\mathcal{A}(\mathsf{1}) = \mathcal{A}_1$, $\mathcal{A}(s) = s_\mathcal{A}$, $\mathcal{A}(t) = t_\mathcal{A}$.

If we want to emphasize that we are talking about small respectively large reflexive graphs meaning that the functor $\mathcal{A}$ factors through SET respectively does not factor in this way, we use the notation: $\mathcal{RG}(\mathsf{SET})$, repsectively $\mathcal{RG}(\mathsf{SET}_1)$.

7.6.2 Families of Reflexive Graphs

Definition 7.6.2.1 (Category of elements (Grothendieck construction)). *Let C be a category. The* category of elements *of a functor $F : C \to \mathsf{SET}_1$ is defined to be the category $\int C$ with $Ob\,(\int C) = \{(c, x) | c \in Ob\,\mathcal{C}, x \in F(c)\}$. Morphism have the form $\phi : (c, x) \to (c', x')$ where ϕ is a morphism $\phi_0 : c \to c'$ in C such that $F(\phi_0)(x) = x'$.*

This construction is a special case of the situation where $F : C \to \mathsf{Cat}$ is a functor into the category of categories. Sometimes the category of elements is called Grothendieck construction as well.

Definition 7.6.2.2 ($\mathcal{RG}_\Gamma$)**.** *Let* $\Gamma : (\mathsf{RG}) \to \mathsf{SET}_1$ *be a reflexive graph. For* S *either* Set*, or* Set_1*, or* TYPE *we define the* category of (small, respectively large, respectively very large) reflexive graphs[7] over Γ *as the functor category* $\mathcal{RG}(\mathsf{S})_\Gamma := (\int \Gamma)^{\mathsf{op}} \to \mathsf{S}$*.*

In our model, a type in a context Γ will be interpreted as as a presheaf $\mathcal{A} \in \mathcal{RG}_\Gamma$

7.6.2.1 Comprehension

For the purpose of modeling a dependent type in context, we need a construction accounting for comprehension: if $\Gamma \vdash A$ type, we need to form the new context $\Gamma,\ a : A$ to define $\Gamma,\ a : A \vdash B(a)$ type. We first make the following definition:

[6]Notice that some authors refer to reflexive graphs as (contravariant) presheaves on the opposite category $\mathsf{RG}^{\mathsf{op}}$. While of course $(\mathsf{RG}^{\mathsf{op}})^{\mathsf{op}}$ is isomorphic to RG, some care is to be taken when discussing universes in the resulting functor categories since the construction of these universes as described [119] is given only for contravariant presheaves and it is not immediately clear that this carries over to covariant presheaves without further changes in addition to “or-ing”.

[7]Mind also here the remark concerning variance made in the footnote of Definition 7.6.1.2.

Definition 7.6.2.3. *For $\mathcal{A} : (\int \Gamma)^{\mathsf{op}} \to \mathsf{SET}_1$, we (overload notation and) define $\mathcal{RG}_{\mathcal{A}} :=$ $(\int \mathcal{A})^{op} \to \mathsf{SET}_1$*

As such, $\mathcal{B} \in \mathcal{RG}_{\mathcal{A}}$ is not yet an exact interpretation of the syntax $\Gamma,\ a : A \vdash B(a)$ type since $\mathcal{A}$ is no reflexive graph and thus no semantic context. This is rectified by the following definition and the subsequent lemma:

Definition 7.6.2.4 (Comprehension (Γ indexed family of reflexive graphs over a base)). *For $\mathcal{A} : \int \Gamma \to \mathsf{SET}_1$, we can promote the category $\int \mathcal{A}$ to a reflexive graph $\Gamma \bullet \mathcal{A} : \mathsf{RG} \to \mathsf{SET}_1$ by*

$$x \mapsto \{((x,\gamma),\zeta) \mid \gamma \in \Gamma(x), \zeta \in \mathcal{A}(x,\gamma)\}$$

where source, and target maps are defined in the obvious way, and call this reflexive graph the comprehension of $\mathcal{A}$ in Γ.

Lemma 7.6.2.5. *In the situation of the previous definition we have $\int(\Gamma \bullet \mathcal{A}) \simeq \int \mathcal{A}$.*

As an aside, we notice that the above definition and its assorted lemma obviously make sense for general presheaves, not only for reflexive graphs.

Lemma 7.6.2.6 (Slices of $\mathcal{RG}_\Gamma$ (families of reflexive graphs over a fixed base)). *There is an equivalence $\mathcal{RG}_\Gamma/\mathcal{A} \simeq \mathcal{RG}_{\mathcal{A}}$.*

Proof. Given $f : Hom_{\mathcal{RG}_\Gamma}(\mathcal{B}, \mathcal{A})$ we obtain a presheaf $X^f : \int \mathcal{A} \to \mathsf{Set}$. Since

$$Ob \int \mathcal{A} = \{((x,\gamma),\zeta) \mid x \in \mathsf{RG},\ \gamma \in \Gamma(x), \zeta \in \mathcal{A}(x,\gamma)\}$$

we define

$$X^f : ((x,\gamma),\zeta) \mapsto \{((x,\gamma),\xi) \mid \xi \in \mathcal{B}(x,\gamma),\ f(x,\gamma)(\xi) = \zeta\} = \mathcal{B}(x,\gamma)_\zeta$$

and if $h : ((x,\gamma),\zeta) \to ((x',\gamma'),\zeta')$ we have that

$$X^f(h) = (((x',\gamma'),\xi')) \mapsto ((x,\gamma), \mathcal{A}(h')(f(x',\gamma')(\xi')))$$

or, equivalently

$$X^f(h) = (((x',\gamma'),\xi')) \mapsto ((x,\gamma), f(x,\gamma)(\mathcal{B}(h')))$$

where $h' : (x,\gamma) \to (x',\gamma')$ is the morphism in $\int \Gamma$ underlying h.

Conversely, if $X : \int \mathcal{A} \to \mathsf{Set}$, we can define $\mathcal{B}$ by $\mathcal{B}(x,\gamma) = \Sigma_{\zeta \in \mathcal{A}(x,\gamma)} X((x,\gamma),\zeta)$. The maps $\mathcal{B}(x,\gamma) \to \mathcal{A}(x,\gamma)$ are the projections which also determine an action on morphisms.

7.6.3 Interpreting MLTT in $\mathbb{T} : \mathcal{RG} \to \mathsf{Fam}(\mathsf{SET}_1)$

The relational-parametric model of $\mathsf{MLTT} + \mathsf{DS}$ we are about to define consists in interpreting all types and terms in the following CwF

$$\begin{aligned}&\mathbb{T} : \mathcal{RG} \to \mathsf{Fam}(\mathsf{TYPE})\\&\mathbb{T}(\Gamma) = (\mathcal{RG}(\mathsf{TYPE})_\Gamma, \mathcal{A} \mapsto (1 \to \mathcal{A}))\ .\end{aligned}$$

Since a CwF model is in first place a model for type dependency, the definitions of concrete types have to be expressed with respect to this CwF structure which is standard (see e.g. [64, §3]) in case of Sigma and Pi types since they are characterized by universal properties. If we wish to interpret further types such as universes or IR we need to refer to additional structure of our particular model.

In our model such additional structure is that of presheaf topoi: we are defining a presheaf topos model in the sense that contexts and types are interpreted as (covariant) presheaves and terms as certain morphisms of presheaves, and thus the types we describe in our model are instances of the respective definitions in a general presheaf topos model and as such are validated as instances of these already existing models. General references for the type theory associated to a topos are [69, pp. D 4.3, D 4.4][66][85]; these references do not cover the interpretation of universes for which we will give references separately in Section 7.6.3.2.

7.6.3.1 Interpreting Σ-, and Π-Types

In this section we describe the types expressible in terms of the CwF structure alone; these are Σ-types, Π-types, and function types as degenerate Π-types. Π-types were described in [11] and [12].

For the case of Σ-, and Π-types which bind a dependent type we need to refer to the semantics of comprehension defined in Definition 7.6.2.4. On the syntactical level, comprehension forms from $\Gamma \vdash A\ \mathsf{type}$, and $\Gamma \vdash a : A$ the new context $\Gamma,\ a : A$ in which we can define $\Gamma,\ a : A \vdash B(a)\ \mathsf{type}$. Now, if $\mathbf{K}$ stands for the constructor of either Π, Σ (or any other constructor binding a dependent type such as W-types), we obtain $\Gamma \vdash \mathsf{K}(A, B)\ \mathsf{type}$. In terms of $\mathbb{T}$ this reads $A \in \mathbb{T}_0((\!|\Gamma|\!))$, $B \in \mathbb{T}_0((\!|\Gamma|\!) \bullet A)$, $\mathsf{K}(A, B) \in \mathbb{T}_0((\!|\Gamma|\!))$. We use these notational conventions for the following paragraphs. By overloading notation, we use the letters s, t, r to stand for the source,- target, and reflexivity maps of all reflexive graphs without annotating as subscripts which reflexive graph is meant.

7.6.3.1.1 Σ-Types in $\mathcal{RG}_\Gamma$

$$
\begin{aligned}
\Sigma\mathcal{A}\mathcal{B}(0,\gamma_0) &:= \{(a,b) | a : \mathcal{A}(0,\gamma_0), b : \mathcal{B}(0,\gamma_0,a)\}\\
\Sigma\mathcal{A}\mathcal{B}(1,\gamma_1) &:= \{((a^R,b^R))|\\
&\qquad a^R : \mathcal{A}(1,\gamma_1) \quad b^R : \mathcal{B}(1,\gamma_1,a^r)\}
\end{aligned}
$$

$$
\begin{aligned}
s(a^R,b^R) &:= (sa^R, sb^R)\\
t(a^R,b^R) &:= (ta^R, tb^R)\\
r(a,b) &:= (ra, rb)
\end{aligned}
$$

There are two projection morphisms which we denote with the same name as the set-level projections:

$$
\begin{aligned}
&\mathsf{proj}_1 : \Sigma\mathcal{A}\mathcal{B} \to \mathcal{A}\\
&\mathsf{proj}_1(x,\gamma)(l,r) := l
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{proj}_2 : \Sigma\mathcal{A}\mathcal{B} \to \mathcal{B}(\_,\_,\mathsf{proj}_1)\\
&\mathsf{proj}_2(x,\gamma)(l,r) := r
\end{aligned}
$$

The universal property of dependent sums is described in [64, Definition 3.18].

We thus interpret the Σ-type by $(\!|\Sigma\ A\ B|\!) = \Sigma(\!|A|\!)(\!|B|\!)$. Notice that we use the symbol 'Σ' for both: Σ-types of types, and Σ-types of families of reflexive graphs.

7.6.3.1.2 Π-Types in $\mathcal{RG}_\Gamma$

$$
\begin{aligned}
\Pi\mathcal{A}\mathcal{B}(0,x) := \{(f_0,f_r)|&\\
&f_0 : \forall a_0 : \mathcal{A}(0,x).\mathcal{B}(x,a_0)\\
&f_r : \forall a_r : \mathcal{A}(1,(rx)).\mathcal{B}(rx,a_r)\\
&\forall a_r : \mathcal{A}(1,(rx)).s(f_r a_r) = f_0(sa_r)\\
&\forall a_r : \mathcal{A}(1,(rx)).t(f_r a_r) = f_0(ta_r)\\
&\forall a_0 : \mathcal{A}(0,x).r(f_0 a_0) = f_r(ra_0)\}
\end{aligned}
$$

And

$$
\begin{aligned}
\Pi\mathcal{A}\mathcal{B}(1,y) := \{((f_0^s,f_r^s),(f_0^t,f_r^t),F)|&\\
&(f_0^s,f_r^s) : \Pi\mathcal{A}\mathcal{B}(0,sy)\\
&(f_0^t,f_r^t) : \Pi\mathcal{A}B(0,ty)\\
&F : \forall a_r : \mathcal{A}(1,y).\mathcal{B}(1,y,a_r)\\
&\forall a_r : \mathcal{A}(1,y).sFa_r = f_0^s sa_r\\
&\forall a_r : \mathcal{A}(1,y).tFa_r = f_0^t ta_r\}
\end{aligned}
$$

$$\begin{aligned}
s_{\Pi\mathcal{A}\mathcal{B}} y((f_0^s, f_r^s), (f_0^t, f_r^t), F) &:= (f_0^s, f_r^s)\\
t_{\Pi\mathcal{A}\mathcal{B}} y((f_0^s, f_r^s), (f_0^t, f_r^t), F) &:= (f_0^t, f_r^t)\\
r_{\Pi\mathcal{A}\mathcal{B}} x(f_0, f_r) &:= ((f_0, f_r), (f_0, f_r), f_r)
\end{aligned}$$

We interpret the Π-type by $(\!|\Pi\ A\ B|\!) = \Pi(\!|A|\!)(\!|B|\!)$. Also the symbol 'Π' we use for Π-types of types, and Π-types of families of reflexive graphs.

7.6.3.1.3 Exponentials in $\mathcal{RG}_\Gamma$ The interpretation of exponentials is a special case of that of Π-types

$$\begin{aligned}
(\mathcal{A} \Rightarrow \mathcal{B})(0, \gamma_0) = \{(f_0, f_1) \mid\ & f_0 : \mathcal{A}(0, \gamma_0) \to \mathcal{B}(0, \gamma_0),\ f_1 : \mathcal{A}(1, \Gamma(r)(\gamma_0)) \to \mathcal{B}(1, \Gamma(r)(\gamma_0))\\
& \mathcal{B}(s) \circ f_1 = f_0 \circ \mathcal{B}(s),\ \mathcal{B}(t) \circ f_1 = f_0 \circ \mathcal{B}(t)\\
& \mathcal{B}(r) \circ f_0 = f_1 \circ \mathcal{A}(r)\}
\end{aligned}$$

$$\begin{aligned}
(\mathcal{A} \Rightarrow \mathcal{B})(1, \gamma_1) = \{((f_0^{src}, f_1^{src}), (f_0^{tgt}, f_1^{tgt}), R) \mid\ & (f_0^{src}, f_1^{src}) : (\mathcal{A} \Rightarrow \mathcal{B})(0, \Gamma(s)(\gamma_1))\\
& (f_0^{tgt}, f_1^{tgt}) : (\mathcal{A} \Rightarrow \mathcal{B})(0, \Gamma(t)(\gamma_1))\\
& R : \mathcal{A}(1, \gamma_1) \to \mathcal{B}(1, \gamma_1)\\
& \mathcal{B}(s) \circ R = f_0^{src} \circ \mathcal{A}(s)\\
& \mathcal{B}(t) \circ R = f_0^{tgt} \circ \mathcal{A}(t)\}
\end{aligned}$$

$$(\mathcal{A} \Rightarrow \mathcal{B})(r) : (f_0, f_1) \mapsto ((f_0, f_1), (f_0, f_1), f_1)$$

$$(\mathcal{A} \Rightarrow \mathcal{B})(s) : ((f_0^{src}, f_1^{src}), (f_0^{tgt}, f_1^{tgt}), R) \mapsto (f_0^{src}, f_1^{src})$$

$$(\mathcal{A} \Rightarrow \mathcal{B})(t) : ((f_0^{src}, f_1^{src}), (f_0^{tgt}, f_1^{tgt}), R) \mapsto (f_0^{tgt}, f_1^{tgt})$$

We interpret the (non-dependent) function type by $(\!|A \to B|\!) = (\!|A|\!) \Rightarrow (\!|B|\!)$. The notation $(\!|A|\!) \Rightarrow (\!|B|\!)$ should not be confused with the notation $(\!|A|\!) \to (\!|B|\!)$ where the latter denotes the set of morphisms of families of reflexive graphs.

⋆ ⋆ ⋆

7.6.3.2 $(\!|\mathsf{Set}|\!)$, $(\!|\mathsf{Set}_1|\!)$

A translation of set-theoretic universes in presheaf topoi is in [119]. To model type theory strictly (and not only up to equivalence) the model of loc.cit. has to be reviewed which was done in [12].

Definition 7.6.3.1. *We interpret* $\Gamma \vdash \mathsf{Set}_1\ \mathsf{type}$ *as the following family of reflexive graphs:*

$$\begin{aligned}
&(\!|\mathsf{Set}_1|\!) : (\int \Gamma)^{\mathsf{op}} \to \mathsf{TYPE}\\
&(x,\gamma) \mapsto \big\{\mathcal{A}(x,\gamma) \mid \mathcal{A} \in \mathcal{RG}(\mathsf{Set}_1)_{(\!|\Gamma|\!)}\big\}\\
&s : \mathcal{A}(1,\gamma) \mapsto \mathcal{A}(0,s(\gamma))\\
&t : \mathcal{A}(1,\gamma) \mapsto \mathcal{A}(0,t(\gamma))\\
&r : \mathcal{A}(0,\gamma) \mapsto \mathcal{A}(1,r(\gamma))
\end{aligned}$$

The interpretation of the Russel universe Set has two manifestations — namely first via the interpretation of the judgment $\Gamma \vdash \mathsf{Set\ type}$ exhibiting Set as a type, and second via the judgment $\Gamma \vdash \mathsf{Set} : \mathsf{Set}_1$ (expressing cumulativity) exhibiting it as a term of Set_1. Since in our model types and terms are interpreted as materially different (i.e. not only by a formal distinction) kind of things —namely families of reflexive graphs and global elements of families of reflexive graphs— we have to give two different interpretation where we denote the interpretation of 'Set as a type' by $\mathcal{rg}(\mathsf{SET})_\Gamma$, and the interpretation of 'Set as a term' by $(\!|\mathsf{Set}|\!)$. Since these definitions are equivalent, we will usually write $(\!|\mathsf{Set}|\!)$ for both incarnations.

Definition 7.6.3.2 **($(\!|\ \Gamma \vdash \mathsf{Set\ type}\ |\!)$).** *We define $\mathcal{rg}(\mathsf{SET})_\Gamma \in \mathcal{RG}(\mathsf{SET}_1)_\Gamma$ by:*

$$\begin{aligned}
&\mathcal{rg}(\mathsf{SET})_\Gamma : (\int \Gamma)^{\mathsf{op}} \to \mathsf{SET}_1\\
&(x,\gamma) \mapsto \big\{\mathcal{A}(x,\gamma) \mid \mathcal{A} \in \mathcal{RG}(\mathsf{SET})_{(\!|\Gamma|\!)}\big\}\\
&s : \mathcal{A}(1,\gamma) \mapsto \mathcal{A}(0,s(\gamma))\\
&t : \mathcal{A}(1,\gamma) \mapsto \mathcal{A}(0,t(\gamma))\\
&r : \mathcal{A}(0,\gamma) \mapsto \mathcal{A}(1,r(\gamma))
\end{aligned}$$

Definition 7.6.3.3 **($(\!|\ \Gamma \vdash \mathsf{Set} : \mathsf{Set}_1\ |\!)$).** *We define the following global element:*

$$\begin{aligned}
&(\!|\mathsf{Set}|\!) : 1 \to (\!|\mathsf{Set}_1|\!)\\
&(\!|\mathsf{Set}|\!)(x,\gamma)(*) = \mathcal{rg}(\mathsf{SET})_{(\!|\Gamma|\!)}(x,\gamma)
\end{aligned}$$

where we obtain the s, t, r since $\mathcal{rg}(\mathsf{SET})_{(\!|\Gamma|\!)} \in \mathcal{RG}(\mathsf{SET}_1)_\Gamma$.

7.6.3.3 $(\!|\mathsf{Fam}|\!)$

We define

$$
\begin{aligned}
&(\!|\mathsf{Fam}|\!) : 1 \to (\!|\mathsf{Set}_1|\!) \Rightarrow (\!|\mathsf{Set}_1|\!)\\
&(\!|\mathsf{Fam}|\!)(0,\gamma_0)(*) := (f_0^{\mathsf{Fam},(0,\gamma_0)}, f_1^{\mathsf{Fam},(0,\gamma_0)})\\
&f_0^{\mathsf{Fam},(0,\gamma_0)} : \mathcal{D}(0,\gamma_0) \mapsto \Sigma_{\mathcal{A}(0,\gamma_0):(\!|\mathsf{Set}|\!)(0,\gamma_0)}(\mathcal{A}(0,\gamma_0) \to \mathcal{D}(0,\gamma_0))\\
&f_1^{\mathsf{Fam},(0,\gamma_0)} : \mathcal{D}(1,r(\gamma_0)) \mapsto \Sigma_{\mathcal{A}(1,r(\gamma_0)):(\!|\mathsf{Set}|\!)(1,r(\gamma_0))}(\mathcal{A}(1,r(\gamma_0)) \to \mathcal{D}(1,r(\gamma_0)))\\
&(\!|\mathsf{Fam}|\!)(1,\gamma_1)(*) := (((f_0^{\mathsf{Fam},src,(1,\gamma_1)}, f_1^{\mathsf{Fam},src,(1,\gamma_1)}), (f_0^{\mathsf{Fam},tgt,(1,\gamma_1)}, f_1^{\mathsf{Fam},tgt,(1,\gamma_1)})), f_R^{\mathsf{Fam},(1,\gamma_1)})\\
&f_0^{\mathsf{Fam},src,(1,\gamma_1)} : \mathcal{D}(0,s(\gamma_1)) \mapsto \Sigma_{\mathcal{A}(0,s(\gamma_1)):(\!|\mathsf{Set}|\!)(0,s(\gamma_1))}(\mathcal{A}(0,s(\gamma_1)) \to \mathcal{D}(0,s(\gamma_1)))\\
&f_1^{\mathsf{Fam},src,(1,\gamma_1)} : \mathcal{D}(1,r(s(\gamma_1))) \mapsto \Sigma_{\mathcal{A}(1,r(s(\gamma_1))):(\!|\mathsf{Set}|\!)(1,r(s(\gamma_1)))}(\mathcal{A}(1,r(s(\gamma_1))) \to \mathcal{D}(1,r(s(\gamma_1))))\\
&f_0^{\mathsf{Fam},tgt,(1,\gamma_1)} : \mathcal{D}(0,t(\gamma_1)) \mapsto \Sigma_{\mathcal{A}(0,t(\gamma_1)):(\!|\mathsf{Set}|\!)(0,t(\gamma_1))}(\mathcal{A}(0,st\gamma_1)) \to \mathcal{D}(0,t(\gamma_1)))\\
&f_1^{\mathsf{Fam},src,(1,\gamma_1)} : \mathcal{D}(1,r(t(\gamma_1))) \mapsto \Sigma_{\mathcal{A}(1,r(t(\gamma_1))):(\!|\mathsf{Set}|\!)(1,r(t(\gamma_1)))}(\mathcal{A}(1,r(t(\gamma_1))) \to \mathcal{D}(1,r(t(\gamma_1))))\\
&f_R^{\mathsf{Fam},(1,\gamma_1)} : \mathcal{D}(1,\gamma_1) \mapsto \Sigma_{\mathcal{A}(1,\gamma_1):(\!|\mathsf{Set}|\!)(1,\gamma_1)}(\mathcal{A}(1,\gamma_1) \to \mathcal{D}(1,r\gamma_1))
\end{aligned}
$$

By Lemma 1.2.2.2, we can evaluate the global element $(\!|\mathsf{Fam}|\!) : 1 \to (\!|\mathsf{Set}_1|\!) \Rightarrow (\!|\mathsf{Set}_1|\!)$ in a global element $\mathcal{D} : 1 \to (\!|\mathsf{Set}_1|\!)$ to obtain a global element $(\!|\mathsf{Fam}|\!)\mathcal{D} : 1 \to (\!|\mathsf{Set}_1|\!)$. One can define pointwise projection maps induced by the set-level Σ-types.

We can define $(\!|\mathsf{Fam}|\!)$ also as a function $(\!|\mathsf{Fam}|\!) : \mathcal{RG}(\mathsf{SET}_1)_\Gamma \to \mathcal{RG}(\mathsf{SET}_1)_\Gamma$. This definition is an instance of Σ-types (see Paragraph 7.6.3.1.1), and exponentials (see Paragraph 7.6.3.1.3). As a Σ-type, $(\!|\mathsf{Fam}|\!)(\!|D|\!)$ comes with two projections which is more convenient. We spell out some details of this for later reference:

Definition 7.6.3.4 ($(\!|\mathsf{Fam}|\!)$). *1. For $(\!|D|\!) : \mathcal{RG}(\mathsf{SET}_1)_\Gamma$ we define*

$$(\!|\mathsf{Fam}|\!)\ (\!|D|\!) \in (\int \Gamma) \to \mathsf{SET}_1$$

$$(\!|\mathsf{Fam}|\!)\ (\!|D|\!) = \Sigma(\!|\mathsf{Set}|\!)(\_ \Rightarrow (\!|D|\!))$$

$$
\begin{aligned}
&\Sigma((\!|\mathsf{Set}|\!)(\_ \Rightarrow (\!|D|\!)))(0,\gamma_0) := \{(u,t) | u : (\!|\mathsf{Set}|\!)(0,\gamma_0), t : (\_ \Rightarrow (\!|D|\!))(0,\gamma_0,u)\}\\
&\Sigma((\!|\mathsf{Set}|\!)(\_ \Rightarrow (\!|D|\!)))(1,\gamma_1) := \{((u^R,t^R))|\\
&\qquad\qquad u^R : (\!|\mathsf{Set}|\!)(1,\gamma_1) \quad t^R : (\_ \Rightarrow (\!|D|\!))(1,\gamma_1,u^R)\}
\end{aligned}
$$

$$s(u^R,t^R) := (su^R, st^R) \tag{7.1}$$

$$t(u^R,t^R) := (tu^R, tt^R) \tag{7.2}$$

$$r(u,t) := (ru, rt)\ . \tag{7.3}$$

2. We define $(\!|\mathsf{Fam}(D)|\!) := (\!|\mathsf{Fam}|\!)(\!|D|\!)$.

Remark 7.6.3.5 (Global element of $(\!|\mathsf{Fam}|\!)\mathcal{D}$). *For a global element $(\mathfrak{u}, \mathfrak{t}) : 1 \to (\!|\mathsf{Fam}|\!)\mathcal{D}$ we obtain the following more convenient description of the notation of the previous definition: if $u = \mathfrak{u}(0,\gamma_0)(*)$ in*

$$(\_ \Rightarrow (\!|D|\!))(0,\gamma_0,u) = (\mathcal{U} \Rightarrow (\!|D|\!))(0,\gamma_0) \ ,$$

then $u = \mathcal{U}(0,\gamma_0)$ for an $\mathcal{U} \in \mathcal{RG}(\mathsf{SET})_\Gamma$, and analogously for $u^R = \mathfrak{u}(1,\gamma_1)(*)$ in

$$(\_ \Rightarrow (\!|D|\!))(1,\gamma_1,u^R) = (\mathcal{U} \Rightarrow (\!|D|\!))(1,\gamma_1) \ ,$$

$u^R = \mathcal{U}(1,\gamma_1)$ for an $\mathcal{U} \in \mathcal{RG}(\mathsf{SET})_\Gamma$.

$$t \in (\mathcal{U} \Rightarrow (\!|D|\!))(0,\gamma_0) = \{(\mathfrak{t}_0,\mathfrak{t}_1) \mid \mathfrak{t}_0 : \mathcal{U}(0,\gamma_0) \to (\!|D|\!)(0,\gamma_0),\ \mathfrak{t}_1 : \mathcal{U}(1,\Gamma(r)(\gamma_0)) \to (\!|D|\!)(1,\Gamma(r)(\gamma_0))$$

$$(\!|D|\!)(s) \circ \mathfrak{t}_1 = \mathfrak{t}_0 \circ (\!|D|\!)(s), \tag{7.4}$$

$$(\!|D|\!)(t) \circ \mathfrak{t}_1 = \mathfrak{t}_0 \circ (\!|D|\!)(t) \tag{7.5}$$

$$(\!|D|\!)(r) \circ \mathfrak{t}_0 = \mathfrak{t}_1 \circ \mathcal{U}(r)\} \tag{7.6}$$

And

$$\begin{aligned} b^R \in (\mathcal{U} \Rightarrow (\!|D|\!))(1,\gamma_1) = \{((\mathfrak{t}_0^{src},\mathfrak{t}_1^{src}),(\mathfrak{t}_0^{tgt},\mathfrak{t}_1^{tgt}),\mathfrak{t}_R) \mid\ & (\mathfrak{t}_0^{src},\mathfrak{t}_1^{src}) : (\mathcal{U} \Rightarrow (\!|D|\!))(0,\Gamma(s)(\gamma_1)) \\ & (\mathfrak{t}_0^{tgt},\mathfrak{t}_1^{tgt}) : (\mathcal{U} \Rightarrow (\!|D|\!))(0,\Gamma(t)(\gamma_1)) \\ & \mathfrak{t}_R : \mathcal{U}(1,\gamma_1) \to (\!|D|\!)(1,\gamma_1) \end{aligned}$$

$$(\!|D|\!)(s) \circ \mathfrak{t}_R = \mathfrak{t}_0^{src} \circ \mathcal{U}(s) \tag{7.7}$$

$$(\!|D|\!)(t) \circ \mathfrak{t}_R = \mathfrak{t}_0^{tgt} \circ \mathcal{U}(t)\} \tag{7.8}$$

$$\begin{aligned} (\mathcal{U} \Rightarrow (\!|D|\!))(r) &: (\mathfrak{t}_0,\mathfrak{t}_1) \mapsto ((\mathfrak{t}_0,\mathfrak{t}_1),(\mathfrak{t}_0,\mathfrak{t}_1),\mathfrak{t}_1) \\ (\mathcal{U} \Rightarrow (\!|D|\!))(s) &: ((\mathfrak{t}_0^{src},\mathfrak{t}_1^{src}),(\mathfrak{t}_0^{tgt},\mathfrak{t}_1^{tgt}),\mathfrak{t}_R) \mapsto (\mathfrak{t}_0^{src},\mathfrak{t}_1^{src})) \\ (\mathcal{U} \Rightarrow (\!|D|\!))(t) &: ((\mathfrak{t}_0^{src},\mathfrak{t}_1^{src}),(\mathfrak{t}_0^{tgt},\mathfrak{t}_1^{tgt}),\mathfrak{t}_R) \mapsto (\mathfrak{t}_0^{tgt},\mathfrak{t}_1^{tgt}) \end{aligned}$$

We also unpack the following family of reflexive graphs for later reference:

Definition 7.6.3.6 ($(\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!)$).

$$\begin{aligned} ((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(0,\gamma_0) = \{(\mathcal{F}_0,\mathcal{F}_1) \mid\ & \\ & \mathcal{F}_0 : (\!|\mathsf{Fam}(D)|\!)(0,\gamma_0) \to (\!|\mathsf{Fam}(D)|\!)(0,\gamma_0), \\ & \mathcal{F}_1 : (\!|\mathsf{Fam}(D)|\!)(1,\Gamma(r)(\gamma_0)) \to (\!|\mathsf{Fam}(D)|\!)(1,\Gamma(r)(\gamma_0)) \end{aligned}$$

$$(\!|\mathsf{Fam}(D)|\!)(s) \circ \mathcal{F}_1 = \mathcal{F}_0 \circ (\!|\mathsf{Fam}(D)|\!)(s), \tag{7.9}$$

$$(\!|\mathsf{Fam}(D)|\!)(t) \circ \mathcal{F}_1 = \mathcal{F}_0 \circ (\!|\mathsf{Fam}(D)|\!)(t) \tag{7.10}$$
$$(\!|\mathsf{Fam}(D)|\!)(r) \circ \mathcal{F}_0 = \mathcal{F}_1 \circ (\!|\mathsf{Fam}(D)|\!)(r)\} \tag{7.11}$$

$$\begin{aligned}((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(1, \gamma_1) = \{((\mathcal{F}_0^{src}, \mathcal{F}_1^{src}), (\mathcal{F}_0^{tgt}, \mathcal{F}_1^{tgt}), \mathcal{F}_R) \mid \\ (\mathcal{F}_0^{src}, \mathcal{F}_1^{src}) : ((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(0, \Gamma(s)(\gamma_1)) \\ (\mathcal{F}_0^{tgt}, \mathcal{F}_1^{tgt}) : ((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(0, \Gamma(t)(\gamma_1)) \\ \mathcal{F}_R : (\!|\mathsf{Fam}(D)|\!)(1, \gamma_1) \to (\!|\mathsf{Fam}(D)|\!)(1, \gamma_1)\end{aligned}$$

$$(\!|\mathsf{Fam}(D)|\!)(s) \circ \mathcal{F}_R = \mathcal{F}_0^{src} \circ (\!|\mathsf{Fam}(D)|\!)(s) \tag{7.12}$$
$$(\!|\mathsf{Fam}(D)|\!)(t) \circ \mathcal{F}_R = \mathcal{F}_0^{tgt} \circ (\!|\mathsf{Fam}(D)|\!)(t)\} \tag{7.13}$$

$$((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(r) : (\mathcal{F}_0, \mathcal{F}_1) \mapsto ((\mathcal{F}_0, \mathcal{F}_1), (\mathcal{F}_0, \mathcal{F}_1), \mathcal{F}_1)$$
$$((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(s) : ((\mathcal{F}_0^{src}, \mathcal{F}_1^{src}), (\mathcal{F}_0^{tgt}, \mathcal{F}_1^{tgt}), R) \mapsto (\mathcal{F}_0^{src}, \mathcal{F}_1^{src})$$
$$((\!|\mathsf{Fam}(D)|\!) \Rightarrow (\!|\mathsf{Fam}(D)|\!))(t) : ((\mathcal{F}_0^{src}, \mathcal{F}_1^{src}), (\mathcal{F}_0^{tgt}, \mathcal{F}_1^{tgt}), R) \mapsto (\mathcal{F}_0^{tgt}, \mathcal{F}_1^{tgt})$$

In particular an argument taken by $\mathcal{F}_0$ is a tuple $(\mathfrak{U}(0, \gamma_0), (t_0, t_1))$ satisfying Eq. (7.1) *Eq.* (7.2)*Eq.* (7.3)*, and the pair (t_0, t_1) where $t_0 : \mathfrak{U}(0, \gamma_0) \to (\!|D|\!)(0, \gamma)_0$, and $t_1 : \mathfrak{U}(1, r(\gamma_0)) \to (\!|D|\!)(1, r(\gamma_0))$ has to satisfy Eq.* (7.4)*Eq.* (7.5)*Eq.* (7.6)*.*

An argument taken by $\mathcal{F}_1$ is a tuple $(\mathfrak{U}(1, r(\gamma_0)), ((t_0^{src}, t_1^{src}), (t_0^{tgt}, t_1^{tgt}), t_R))$ satisfying Eq. (7.1) *Eq.* (7.2)*Eq.* (7.3)*, and $((t_0^{src}, t_1^{src}), (t_0^{tgt}, t_1^{tgt}), t_R)$ must satisfy Eq.* (7.12) *where $(t_0^{src}, t_1^{src}) : (\mathfrak{U} \Rightarrow (\!|D|\!)(0, s(r(\gamma_0))))$, and $(t_0^{tgt}, t_1^{tgt}) : (\mathfrak{U} \Rightarrow (\!|D|\!)(0, t(r(\gamma_0))))$ (where the latter two types are identical since by the reflexive graph axioms $s \circ r = t \circ r$), and $t_R : \mathfrak{U}(1, r(\gamma_0)) \to (\!|D|\!)(1, r(\gamma_0))$.*

An argument taken by $\mathcal{F}_R$ is a tuple $(\mathfrak{U}(1, \gamma_1), ((t_0^{src}, t_1^{src}), (t_0^{tgt}, t_1^{tgt}), t_R))$ satisfying Eq. (7.12) *where $(t_0^{src}, t_1^{src}) : (\mathfrak{U} \Rightarrow (\!|D|\!)(0, s(\gamma_1)))$, and $(t_0^{tgt}, t_1^{tgt}) : (\mathfrak{U} \Rightarrow (\!|D|\!)(0, t(\gamma_1)))$, and $t_R : \mathfrak{U}(1, \gamma_1) \to (\!|D|\!)(1, \gamma_1)$.*

⋆ ⋆ ⋆

7.6.4 Interpreting DS

The idea to define binary relational parametricity for DS is to write down the same constructors as for DS but in the presheaf topos $\mathcal{RG}_\Gamma$. For simplicity we give the the model only for the case DS D D.

For $\mathcal{D} : \mathcal{RG}_\Gamma$ we intend to define $(\!|\mathsf{DS}|\!)\ \mathcal{D} : \mathcal{RG}_\Gamma$ by

$$
\begin{aligned}
&\iota' : \mathcal{D} \to (\!|\mathsf{DS}|\!)\ \mathcal{D}\\
&\sigma' : (\mathcal{A} : \mathcal{RG}_\Gamma)(\mathcal{A} \Rightarrow (\!|\mathsf{DS}|\!)\ \mathcal{D}) \to (\!|\mathsf{DS}|\!)\ \mathcal{D}\\
&\delta' : (\mathcal{A} : \mathcal{RG}_\Gamma)((\mathcal{A} \Rightarrow \mathcal{D}) \Rightarrow (\!|\mathsf{DS}|\!)\ \mathcal{D}) \to (\!|\mathsf{DS}|\!)\ \mathcal{D}
\end{aligned}
$$

Since we presently have no notion of strict positivity for objects in $\mathcal{RG}_\Gamma$ telling us whether the morphisms ι', σ', δ' really define something in line with our set theoretic foundations, we reduce the above definition to a mutual inductive definition of two (namely for $x = 0$, and $x = 1$) families of sets by two families of constructors $\iota^-, \sigma^-, \delta^-$ simultaneously with recursively defined families of maps r_{DS}, s_{DS}, t_{DS}.

Definition 7.6.4.1 ($(\!|\mathsf{DS}|\!)$). *We inductive-recursively define the family of sets $((\!|\mathsf{DS}|\!)\ \mathcal{D})$:*

$$
\begin{aligned}
&\iota^-(x,\gamma) : \mathcal{D}(x,\gamma) \to ((\!|\mathsf{DS}|\!)\ \mathcal{D})(x,\gamma)\\
&\sigma^-(x,\gamma) : (\mathcal{A} : \mathcal{RG}_\Gamma)(\mathcal{A} \Rightarrow (\!|\mathsf{DS}|\!)\ \mathcal{D})(x,\gamma) \to ((\!|\mathsf{DS}|\!)\ \mathcal{D})(x,\gamma)\\
&\delta^-(x,\gamma) : (\mathcal{A} : \mathcal{RG}_\Gamma)((\mathcal{A} \Rightarrow \mathcal{D}) \Rightarrow (\!|\mathsf{DS}|\!)\ \mathcal{D})(x,\gamma) \to ((\!|\mathsf{DS}|\!)\ \mathcal{D})(x,\gamma)
\end{aligned}
$$

simultaneously with three families[8] *of maps*

$$
\begin{aligned}
&r_{DS} : ((\!|\mathsf{DS}|\!)\ \mathcal{D})(0,\gamma_0) \to ((\!|\mathsf{DS}|\!)\ \mathcal{D})(1,\Gamma(r)(\gamma_0)), && r_{DS} : \delta^-(0,\gamma_0)\mathcal{A}F \mapsto \delta^-(1,\Gamma r(\gamma_0))\mathcal{A}(rF), \quad \dagger\\
&s_{DS} : ((\!|\mathsf{DS}|\!)\ \mathcal{D})(1,\gamma_1) \to ((\!|\mathsf{DS}|\!)\ \mathcal{D})(0,\Gamma(s)(\gamma_1)), && s_{DS} : \delta^-(1,\gamma_1)\mathcal{A}F \mapsto \delta^-(0,\Gamma s(\gamma_1))\mathcal{A}(sF), \quad \dagger\\
&t_{DS} : ((\!|\mathsf{DS}|\!)\ \mathcal{D})(1,\gamma_1) \to ((\!|\mathsf{DS}|\!)\ \mathcal{D})(0,\Gamma(t)(\gamma_1)), && t_{DS} : \delta^-(1,\gamma_1)\mathcal{A}F \mapsto \delta^-(0,\Gamma t(\gamma_1))\mathcal{A}(tF), \quad \dagger
\end{aligned}
$$

(The formulas marked with † are required to hold with 'iota'- and 'sigma' in place of 'delta', too.) Since $\mathcal{A}$ and the exponentials in the domain are by definition families of reflexive graphs, one observes that $s_{\mathsf{DS}} \circ r_{\mathsf{DS}} = t_{\mathsf{DS}} \circ r_{\mathsf{DS}} = id$.

Remark 7.6.4.2. *Even though, the definition of $(\!|\mathsf{DS}|\!)\mathcal{D}$ defines an indexed (over $\int \Gamma$) family of sets by a simultaneous indexed induction-recursion, there are some differences preventing it (prima facie) from being an instantiation of the coding scheme of indexed induction-recursion (IIRD) presented in [39], and thus we do not automatically get a set-theoretic model for 'parametric MLTT + $(\!|\mathsf{DS}|\!)$' (that is: including decoding (see Section 7.6.4.1 and initial algebras (see Section 7.6.5))) —proving consistency of the latter system— from the model of IIRD given in loc.cit.:*

1. *Our object of codes $(\!|\mathsf{DS}|\!)\mathcal{D}$ lives (as an object)(like all elements of our parametric model) not in the category of sets but in the category of families of reflexive graphs.*
2. *in our definition, the first argument '$\mathcal{A}$' of the constructors σ^-, and δ^- is is a not a set but a family of reflexive graphs, and is indexed over $\int \Gamma$.*

[8]We suppress here the the indices (x, γ) in $r_{DS}(x, \gamma)$, $s_{DS}(x, \gamma)$, and $t_{DS}(x, \gamma)$.

3. *We have an indexing* category *(namely $\int\Gamma$) and not only an indexing set and since the reflexive-graph laws are expressed in terms of morphisms in $\int\Gamma$ they are not directly expressible in IIRD*

Proof. The proof contained in the Agda files shows that this definition is strictly positive and terminating.

By construction we have now for $\mathcal{D} : \mathcal{RG}_\Gamma$ defined a family of large reflexive graphs $(\!|\mathsf{DS}|\!)\ \mathcal{D} : \mathcal{RG}_\Gamma$ as desired.

Definition 7.6.4.3. 1. *We call a global element $c : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}$ a $(\!|\mathsf{DS}|\!)\mathcal{D}$-*code.

2. *We can Definition 7.6.4.1 write in more compact form as:*

$$\frac{d : 1 \to \mathcal{D}}{\iota^-\ d : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}}\ \text{INTRO-}\iota^-$$

$$\frac{\mathcal{A} : \mathcal{RG}(\mathsf{SET})_\Gamma \qquad f : 1 \to \mathcal{A} \Rightarrow (\!|\mathsf{DS}|\!)\mathcal{D}}{\sigma^-\ \mathcal{A}\ f : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}}\ \text{INTRO-}\sigma^-$$

$$\frac{\mathcal{A} : \mathcal{RG}(\mathsf{SET})_\Gamma \qquad f : 1 \to (\mathcal{A} \Rightarrow \mathcal{D}) \Rightarrow (\!|\mathsf{DS}|\!)\mathcal{D}}{\sigma^-\ \mathcal{A}\ f : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}}\ \text{INTRO-}\delta^- \quad .$$

We will sometimes suppress the argument $*$ and write $d(x,\gamma)$ in place of $d(x,\gamma)(*)$, and likewise for f and $\mathcal{F}$.

Definition 7.6.4.4 (Interpretation of $\mathsf{DS}\ D\ D$). 1. *Of course we define $(\!|\mathsf{DS}\ D|\!) :=$ $(\!|\mathsf{DS}|\!)(\!|D|\!)$.*

2. *We interpret the constructor ι as the morphism of families of reflexive graphs with components $\iota^-(x,\gamma)$, i.e. $(\!|\iota|\!)(x,\gamma) = \iota^-(x,\gamma)$, and likewise for the constructors σ, and δ.*

3. *The $((x,\gamma) : \int\Gamma)$-indexed family of constructors $\iota^-(x,\gamma)$, $\sigma^-(x,\gamma)$, $\delta^-(x,\gamma)$ define by mutual induction a family of (large) sets $((\!|\mathsf{DS}|\!)\ \mathcal{D})(x,\gamma) : \mathsf{Set}_1$.*

7.6.4.1 Decoding of $(\!|\mathsf{DS}|\!)\mathcal{D}$-Codes

For a code $c : 1 \to (\!|\mathsf{DS}|\!)\ \mathcal{D}$, we want to define a global point, i.e. a morphism $\langle\!\langle c \rangle\!\rangle : 1 \to ((\!|\mathsf{Fam}|\!)\mathcal{D} \Rightarrow (\!|\mathsf{Fam}|\!)\mathcal{D})$. This amounts to defining

$$\langle\!\langle c \rangle\!\rangle(1,\gamma)(*) \in ((\!|\mathsf{Fam}|\!)\mathcal{D} \Rightarrow (\!|\mathsf{Fam}|\!)\mathcal{D})(1,\gamma)$$

$$\langle\!\langle c\rangle\!\rangle(0,\gamma)(*) \in ((\!|\mathsf{Fam}|\!)\mathcal{D} \Rightarrow (\!|\mathsf{Fam}|\!)\mathcal{D})(0,\gamma)$$

satisfying

$$s_\gamma(\langle\!\langle c\rangle\!\rangle(1,\gamma)(*)) = \langle\!\langle c\rangle\!\rangle(0,s(\gamma))(*) \tag{7.14}$$
$$t_\gamma(\langle\!\langle c\rangle\!\rangle(1,\gamma)(*)) = \langle\!\langle c\rangle\!\rangle(0,t(\gamma))(*) \tag{7.15}$$
$$r_\gamma(\langle\!\langle c\rangle\!\rangle(0,\gamma)(*)) = \langle\!\langle c\rangle\!\rangle(1,r(\gamma))(*) \tag{7.16}$$

The following two definitions are simultaneous.

Definition 7.6.4.5 ($\langle\!\langle c\rangle\!\rangle(0,\gamma)(*)$). *We define $\langle\!\langle c\rangle\!\rangle(0,\gamma)(*)$ by induction on c. We write*

$$\langle\!\langle c\rangle\!\rangle(0,\gamma)(*) =: (\mathcal{F}_0^c, \mathcal{F}_1^c)$$

and

$$\begin{aligned}
&\mathcal{F}_0^c(\mathcal{U}(0,\gamma),(t_0,t_1)) = (\mathcal{U}^c(0,\gamma),(t_0^c,t_1^c))\\
&(t_0^c,t_1^c) : (\mathcal{U}(c) \Rightarrow \mathcal{D})(0,\gamma)\\
&t_0^c : \mathcal{U}^c(0,\gamma) \to \mathcal{D}(0,\gamma)\\
&t_1^c : \mathcal{U}^c(1,r(\gamma)) \to \mathcal{D}(1,r(\gamma))
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}_1^c(\mathcal{U}(1,r(\gamma_0)),((t_0^{src},t_1^{src}),(t_0^{tgt},t_1^{tgt}),t_R)) = (\mathcal{U}^c(1,r(\gamma_0)),((t_0^{src,c},t_1^{src,c}),(t_0^{tgt,c},t_1^{tgt,c}),t^{R,c}))\\
&t_0^{src,c} : \mathcal{U}^c(0,s(r(\gamma_0))) \to \mathcal{D}(0,s(r(\gamma_0)))\\
&t_1^{src,c} : \mathcal{U}^c(1,r(s(r(\gamma_0)))) \to \mathcal{D}(1,r(s(r(\gamma_0))))\\
&t_0^{tgt,c} : \mathcal{U}^c(0,t(r(\gamma_0))) \to \mathcal{D}(0,t(r(\gamma_0)))\\
&t_1^{tgt,c} : \mathcal{U}^c(1,r(t(r(\gamma_0)))) \to \mathcal{D}(1,r(t(r(\gamma_0))))\\
&t^{R,c} : \mathcal{U}^c(1,r(\gamma_0)) \to \mathcal{D}(1,r(\gamma_0))
\end{aligned}$$

<u>For $c = \iota^-\ d$</u>, we define

$$\begin{aligned}
&\mathcal{U}^{\iota^-\ d}(0,\gamma_0) = 1\\
&t_0^{\iota^-\ d} : \mathcal{U}^{\iota^-\ d}(0,\gamma_0) \to \mathcal{D}(0,\gamma_0)\\
&t_0^{\iota^-\ d}(*) = d(0,\gamma_0)\\
&t_1^{\iota^-\ d} : \mathcal{U}^{\iota^-\ d}(1,r(\gamma_0)) \to \mathcal{D}(1,r(\gamma_0))\\
&t_1^{\iota^-\ d}(*) = r(d(0,\gamma_0))\ ,
\end{aligned}$$

and

$$
\begin{aligned}
&\mathcal{U}^{\iota^- d}(1, r(\gamma_0)) = 1\\
&(t_0^{src,\iota^- d}, t_1^{src,\iota^- d}) : (\mathcal{U} \Rightarrow (\!|D|\!))(0, s(r(\gamma_0))))\\
&t_0^{src,\iota^- d}(*) = const\ d(0, \gamma_0)\\
&t_1^{src,\iota^- d}(*) = const\ r d(0, \gamma_0)\\
&(t_0^{tgt,\iota^- d}, t_1^{tgt,\iota^- d}) := (t_0^{src,\iota^- d}, t_1^{src,\iota^- d})\\
&t^{R,\iota^- d} : \mathcal{U}^{\iota^- d}(1, r(\gamma_0)) \to (\!|D|\!)(1, r(\gamma_0))\\
&t^{R,\iota^- d}(*) = d(1, r(\gamma_0))
\end{aligned}
$$

<u>For $c = \sigma^-\ \mathcal{A}\ f$</u> we define

$$
\begin{aligned}
&\mathcal{U}^{\sigma^-\ \mathcal{A}\ f}(0, \gamma_0) := \Sigma_{a:1\to\mathcal{A}} \mathcal{U}^{fa}(0, \gamma_0)\\
&t_0^{\sigma^-\ \mathcal{A}\ f} : \mathcal{U}^{\sigma^-\ \mathcal{A}\ f}(0, \gamma_0) \to \mathcal{D}(0, \gamma_0)\\
&t_0^{\sigma^-\ \mathcal{A}\ f}\ a\ x = t_0^{f\ a}\ x\\
&t_1^{\sigma^-\ \mathcal{A}\ f} : \mathcal{U}^{\sigma^-\ \mathcal{A}\ f}(1, r(\gamma_0)) \to \mathcal{D}(1, r(\gamma_0))\\
&t_1^{\sigma^-\ \mathcal{A}\ f}\ a\ x = t_1^{f\ a}\ x
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{U}^{\sigma^-\ \mathcal{A}\ f}(1, r(\gamma_0)) := \Sigma_{a:1\to\mathcal{A}} \mathcal{U}^{f\ a}(1, r(\gamma_0))\\
&t_0^{src,\sigma^-\ \mathcal{A}\ f} : \mathcal{U}^{\sigma^-\ \mathcal{A}\ f}(0, s(r(\gamma_0))) \to \mathcal{D}(0, s(r(\gamma_0)))\\
&t_0^{src,\sigma^-\ \mathcal{A}\ f}\ a\ x = t_0^{src,f\ a}\ x\\
&t_1^{src,\sigma^-\ \mathcal{A}\ f}\ a\ x = t_1^{src,f\ a}\ x\\
&t_0^{tgt,\sigma^-\ \mathcal{A}\ f}\ a\ x = t_0^{tgt,f\ a}\ x\\
&t_1^{tgt,\sigma^-\ \mathcal{A}\ f}\ a\ x = t_1^{tgt,f\ a}\ x\\
&t^{R,\sigma^-\ \mathcal{A}\ f}\ a\ x = t^{R,f\ a}\ x
\end{aligned}
$$

<u>For $c = \delta^-\ \mathcal{A}\ \mathcal{F}$</u> we define

$$
\begin{aligned}
&\mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F}}(0, \gamma_0) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})} \mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G})}(0, \gamma_0)\\
&t_0^{\delta^-\ \mathcal{A}\ \mathcal{F}} : \mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F}}(0, \gamma_0) \to \mathcal{D}(0, \gamma_0)\\
&t_0^{\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t_0^{\mathcal{F}((t_0,t_1)\circ\mathcal{G})}\ x\\
&t_1^{\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t_1^{\mathcal{F}((t_0,t_1)\circ\mathcal{G})}\ x
\end{aligned}
$$

$$\begin{aligned}
&\mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F}}(1, r(\gamma_0)) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})}\mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G})}(1, r(\gamma_0))\\
&t_0^{src,\delta^-\ \mathcal{A}\ \mathcal{F}} : \mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F}}(0, s(r(\gamma_0))) \to \mathcal{D}(0, s(r(\gamma_0)))\\
&t_0^{src,\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t_0^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G})}\ x\\
&t_1^{src,\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t_1^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G})}\ x\\
&t_0^{tgt,\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t_0^{tgt,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G})}\ x\\
&t_1^{tgt,\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t_1^{tgt,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G})}\ x\\
&t^{R,\delta^-\ \mathcal{A}\ \mathcal{F}}\ \mathcal{G}\ x = t^{R,\mathcal{F}((t_0^{src},t_R)\circ\mathcal{G})}\ x
\end{aligned}$$

For this definition we recall Lemma 1.2.2.2, and that we have the global element $(t_0, t_1) : 1 \to (\mathcal{U} \Rightarrow \mathcal{D})(0, \_)$. *Notice that in the last line of the previous definition we also could have written* $t^{R,\mathcal{F}((t_0^{tgt},t_R)\circ\mathcal{G})}\ x$ *since* $s \circ r = t \circ r$.

Definition 7.6.4.6 ($\langle\!\langle c\rangle\!\rangle(1, \gamma)(*)$). *We define* $\langle\!\langle c\rangle\!\rangle(1, \gamma_1)(*)$ *by induction on* c. *We write*

$$\langle\!\langle c\rangle\!\rangle(1, \gamma)(*) =: ((\mathcal{F}_0^{src,c}, \mathcal{F}_1^{src,c}), (\mathcal{F}_0^{tgt,c}, \mathcal{F}_1^{tgt,c}), \mathcal{F}_R^c)$$

and

$$\begin{aligned}
&\mathcal{F}_0^{src,c}(\mathcal{U}(0, \gamma), (t_0, t_1)) = (\mathcal{U}^{c,SRC}(0, \gamma), (t_0^{c,SRC}, t_1^{c,SRC}))\\
&\mathcal{F}_1^{src,c}(\mathcal{U}(1, r(\gamma_0)), ((t_0^{src}, t_1^{src}), (t_0^{tgt}, t_1^{tgt}), t_R) =\\
&\qquad\qquad (\mathcal{U}^{c,SRC}(1, r(\gamma_0)), ((t_0^{src,c,SRC}, t_1^{src,c,SRC}), (t_0^{tgt,c,SRC}, t_1^{tgt,c,SRC}), t_R^{c,SRC}))
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}_0^{tgt,c}(\mathcal{U}(0, \gamma), (t_0, t_1)) = (\mathcal{U}^{c,TGT}(0, \gamma), (t_0^{c,TGT}, t_1^{c,TGT}))\\
&\mathcal{F}_1^{tgt,c}(\mathcal{U}(1, r(\gamma_0)), ((t_0^{src}, t_1^{src}), (t_0^{tgt}, t_1^{tgt}), t_R) =\\
&\qquad\qquad (\mathcal{U}^{c,TGT}(1, r(\gamma_0)), ((t_0^{src,c,TGT}, t_1^{src,c,TGT}), (t_0^{tgt,c,TGT}, t_1^{tgt,c,TGT}), t_R^{c,TGT}))
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}_R^c(\mathcal{U}(1, \gamma_1), ((t_0^{src}, t_1^{src}), (t_0^{tgt}, t_1^{tgt}), t_R) =\\
&\qquad\qquad (\mathcal{U}^{c,R}(1, \gamma_1)), ((t_0^{src,c,R}, t_1^{src,c,R}), (t_0^{R,c,R}, t_1^{R,c,R}), t_R^{c,R}))\\
&t_0^{src,c,R} : \mathcal{U}^{c,R}(0, s(\gamma_1))) \to \mathcal{D}(0, s(\gamma_1), R)\\
&t_1^{src,c,R} : \mathcal{U}^{c,R}(1, r(s(\gamma_1)))) \to \mathcal{D}(1, r(s(\gamma_1)), R)\\
&t_0^{R,c,R} : \mathcal{U}^{c,R}(0, t(\gamma_1))) \to \mathcal{D}(0, t(\gamma_1), R)\\
&t_1^{R,c,R} : \mathcal{U}^{c,R}(1, r(t(\gamma_1)))) \to \mathcal{D}(1, r(t(\gamma_1)), R)\\
&t_R^{c,R} : \mathcal{U}^{c,R}(1, \gamma_1)) \to \mathcal{D}(1, \gamma_1, R)
\end{aligned}$$

where these terms are defined by:

<u>*For $\mathscr{c} = \iota^- \ \mathscr{d}$*</u> *we define:*

$$\begin{aligned}
&\mathcal{F}_0^{src,\iota^-\ \mathscr{d}} : \_ \mapsto (1, const\ \mathscr{d}(0, s(\gamma_1)), const\ \mathscr{d}r(0, s(\gamma_1)))\\
&\mathcal{F}_1^{src,\iota^-\ \mathscr{d}} : \_ \mapsto (1, (const\ \mathscr{d}(0, s(\gamma_1)), (const\ r\mathscr{d}(0, s(\gamma_1)))),\\
&(const\ \mathscr{d}(0, s(\gamma_1)), (const\ r\mathscr{d}(0, s(\gamma_1)))), (const\ \mathscr{d}(0, r(s(\gamma_1))))))
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}_0^{tgt,\iota^-\ \mathscr{d}} : \_ \mapsto (1, const\ \mathscr{d}(0, t(\gamma_1)), const\ \mathscr{d}r(0, t(\gamma_1)))\\
&\mathcal{F}_1^{tgt,\iota^-\ \mathscr{d}} : \_ \mapsto (1, (const\ \mathscr{d}(0, t(\gamma_1)), (const\ r\mathscr{d}(0, t(\gamma_1)))),\\
&(const\ \mathscr{d}(0, t(\gamma_1)), (const\ r\mathscr{d}(0, t(\gamma_1)))), (const\ \mathscr{d}(0, r(t(\gamma_1))))))
\end{aligned}$$

$$\begin{aligned}
&\mathcal{F}_R^{\iota^-\ \mathscr{d}} : \_ \mapsto (1, (const\ \mathscr{d}(0, s(\gamma_1)), (const\ r\mathscr{d}(0, s(\gamma_1)))),\\
&(const\ \mathscr{d}(0, t(\gamma_1)), (const\ r\mathscr{d}(0, t(\gamma_1)))), (const\ \mathscr{d}(1, \gamma_1)))
\end{aligned}$$

<u>*For $\mathscr{c} = \sigma^- \ \mathcal{A}\ f$*</u> *we define*

$$\begin{aligned}
&\mathscr{U}^{\sigma^-\ \mathcal{A}\,f,SRC}(0,\gamma) = \Sigma_{\mathscr{a}:1\to\mathcal{A}} \mathscr{U}^{f\ \mathscr{a},SRC}(0,\gamma)\\
&\mathscr{t}_0^{\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_0^{f\ \mathscr{a},SRC}\ x\\
&\mathscr{t}_1^{\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_1^{f\ \mathscr{a},SRC}\ x
\end{aligned}$$

$$\begin{aligned}
&\mathscr{U}^{\sigma^-\ \mathcal{A}\,f,SRC}(1,r(\gamma_0)) = \Sigma_{\mathscr{a}:1\to\mathcal{A}} \mathscr{U}^{f\ \mathscr{a},SRC}(1,r(\gamma_0))\\
&\mathscr{t}_0^{src,\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_0^{src,f\ \mathscr{a},SRC}\ x\\
&\mathscr{t}_1^{src,\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_1^{src,f\ \mathscr{a},SRC}\ x\\
&\mathscr{t}_0^{tgt,\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_0^{tgt,f\ \mathscr{a},SRC}\ x\\
&\mathscr{t}_1^{tgt,\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_1^{tgt,f\ \mathscr{a},SRC}\ x\\
&\mathscr{t}_R^{\sigma^-\ \mathcal{A}\,f,SRC}\ \mathscr{a}\ x = \mathscr{t}_R^{f\ \mathscr{a},SRC}\ x
\end{aligned}$$

$$\begin{aligned}
&\mathscr{U}^{\sigma^-\ \mathcal{A}\,f,TGT}(0,\gamma) = \Sigma_{\mathscr{a}:1\to\mathcal{A}} \mathscr{U}^{f\ \mathscr{a},TGT}(0,\gamma)\\
&\mathscr{t}_0^{\sigma^-\ \mathcal{A}\,f,TGT}\ \mathscr{a}\ x = \mathscr{t}_0^{f\ \mathscr{a},TGT}\ x\\
&\mathscr{t}_1^{\sigma^-\ \mathcal{A}\,f,TGT}\ \mathscr{a}\ x = \mathscr{t}_1^{f\ \mathscr{a},TGT}\ x
\end{aligned}$$

$$
\begin{aligned}
&\mathcal{U}^{\sigma^-\ \mathcal{A}\ f,TGT}(1, r(\gamma_0)) = \Sigma_{a:1\to\mathcal{A}}\mathcal{U}^{f\ a,TGT}(1, r(\gamma_0))\\
&t_0^{src,\sigma^-\ \mathcal{A}\ f,TGT}\ a\ x = t_0^{src,f\ a,TGT}\ x\\
&t_1^{src,\sigma^-\ \mathcal{A}\ f,TGT}\ a\ x = t_1^{src,f\ a,TGT}\ x\\
&t_0^{tgt,\sigma^-\ \mathcal{A}\ f,TGT}\ a\ x = t_0^{tgt,f\ a,TGT}\ x\\
&t_1^{tgt,\sigma^-\ \mathcal{A}\ f,TGT}\ a\ x = t_1^{tgt,f\ a,TGT}\ x\\
&t_R^{\sigma^-\ \mathcal{A}\ f,TGT}\ a\ x = t_R^{f\ a,TGT}\ x
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{U}^{\sigma^-\ \mathcal{A}\ f,R}(1, r(\gamma_0)) = \Sigma_{a:1\to\mathcal{A}}\mathcal{U}^{f\ a,R}(1, r(\gamma_0))\\
&t_0^{src,\sigma^-\ \mathcal{A}\ f,R}\ a\ x = t_0^{src,f\ a,R}\ x\\
&t_1^{src,\sigma^-\ \mathcal{A}\ f,R}\ a\ x = t_1^{src,f\ a,R}\ x\\
&t_0^{R,\sigma^-\ \mathcal{A}\ f,R}\ a\ x = t_0^{R,f\ a,R}\ x\\
&t_1^{R,\sigma^-\ \mathcal{A}\ f,R}\ a\ x = t_1^{R,f\ a,R}\ x\\
&t_R^{\sigma^-\ \mathcal{A}\ f,R}\ a\ x = t_R^{f\ a,R}\ x
\end{aligned}
$$

<u>For $c = \delta^-\ \mathcal{A}\ \mathcal{F}$</u> we define

$$
\begin{aligned}
&\mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}(0, \gamma_0) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})}\mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),SRC}(0, \gamma_0)\\
&t_0^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC} : \mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}(0, \gamma_0) \to \mathcal{D}(0, \gamma_0)\\
&t_0^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_0^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),SRC}\ x\\
&t_1^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_1^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),SRC}\ x
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}(1, r(\gamma_0)) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})}\mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),SRC}(1, r(\gamma_0))\\
&t_0^{src,\delta^-\ \mathcal{A}\ \mathcal{F},SRC} : \mathcal{U}^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}(0, s(r(\gamma_0))) \to \mathcal{D}(0, s(r(\gamma_0)))\\
&t_0^{src,\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_0^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G}),SRC}\ x\\
&t_1^{src,\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_1^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G}),SRC}\ x\\
&t_0^{tgt,\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_0^{tgt,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G}),SRC}\ x\\
&t_1^{tgt,\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_1^{tgt,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G}),SRC}\ x\\
&t_R^{\delta^-\ \mathcal{A}\ \mathcal{F},SRC}\ \mathcal{G}\ x = t_R^{\mathcal{F}((t_0^{src},t_R)\circ\mathcal{G}),SRC}\ x
\end{aligned}
$$

$$\begin{aligned}
&\mathcal{U}^{\delta^- \mathcal{A}\ \mathcal{F},TGT}(0,\gamma_0) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})}\mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),TGT}(0,\gamma_0)\\
&t_0^{\delta^- \mathcal{A}\ \mathcal{F},TGT} : \mathcal{U}^{\delta^- \mathcal{A}\ \mathcal{F},TGT}(0,\gamma_0) \to \mathcal{D}(0,\gamma_0)\\
&t_0^{\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_0^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),TGT}\ x\\
&t_1^{\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_1^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),TGT}\ x
\end{aligned}$$

$$\begin{aligned}
&\mathcal{U}^{\delta^- \mathcal{A}\ \mathcal{F},TGT}(1,r(\gamma_0)) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})}\mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),TGT}(1,r(\gamma_0))\\
&t_0^{src,\delta^- \mathcal{A}\ \mathcal{F},TGT} : \mathcal{U}^{\delta^- \mathcal{A}\ \mathcal{F},TGT}(0,s(r(\gamma_0))) \to \mathcal{D}(0,s(r(\gamma_0)))\\
&t_0^{src,\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_0^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G}),TGT}\ x\\
&t_1^{src,\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_1^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G}),TGT}\ x\\
&t_0^{tgt,\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_0^{tgt,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G}),TGT}\ x\\
&t_1^{tgt,\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_1^{tgt,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G}),TGT}\ x\\
&t_R^{\delta^- \mathcal{A}\ \mathcal{F},TGT}\ \mathcal{G}\ x = t_R^{\mathcal{F}((t_0^{src},t_R)\circ\mathcal{G}),TGT}\ x
\end{aligned}$$

$$\begin{aligned}
&\mathcal{U}^{\delta^- \mathcal{A}\ \mathcal{F},R}(1,r(\gamma_0)) := \Sigma_{\mathcal{G}:1\to(\mathcal{A}\Rightarrow\mathcal{U})}\mathcal{U}^{\mathcal{F}((t_0,t_1)\circ\mathcal{G}),R}(1,r(\gamma_0))\\
&t_0^{src,\delta^- \mathcal{A}\ \mathcal{F},R} : \mathcal{U}^{\delta^- \mathcal{A}\ \mathcal{F},R}(0,s(r(\gamma_0))) \to \mathcal{D}(0,s(r(\gamma_0)))\\
&t_0^{src,\delta^- \mathcal{A}\ \mathcal{F},R}\ \mathcal{G}\ x = t_0^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G}),R}\ x\\
&t_1^{src,\delta^- \mathcal{A}\ \mathcal{F},R}\ \mathcal{G}\ x = t_1^{src,\mathcal{F}((t_0^{src},t_1^{src})\circ\mathcal{G}),R}\ x\\
&t_0^{R,\delta^- \mathcal{A}\ \mathcal{F},R}\ \mathcal{G}\ x = t_0^{R,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G}),R}\ x\\
&t_1^{R,\delta^- \mathcal{A}\ \mathcal{F},R}\ \mathcal{G}\ x = t_1^{R,\mathcal{F}((t_0^{tgt},t_1^{tgt})\circ\mathcal{G}),R}\ x\\
&t_R^{\delta^- \mathcal{A}\ \mathcal{F},R}\ \mathcal{G}\ x = t_R^{\mathcal{F}((t_0^{src},t_R)\circ\mathcal{G}),R}\ x
\end{aligned}$$

Definition 7.6.4.7 (Action of decoding on morphisms). *Like in the decoding of* DS*-codes, also the decoding of* (|DS|)*-codes has an action on morphisms Section 3.2.1 which is induced by the universal property of* Σ*-types. In fact there are two similar cases we can distinguish: the function describing the action of decoding on a transformation of a global element, and the action of decoding on a morphism between global elements. We shall only be interested to fix a notation for the former:*

If $\varphi : (\mathcal{U}, t)(x,\gamma) \to (\mathcal{U}, t)(x',\gamma')$*, then*

$$\begin{aligned}
&(\phi)^{\mathcal{F}\sigma^-\ \mathcal{A}\ f} : \Sigma_{a:1\to\mathcal{A}}(\mathcal{U})^{\mathcal{F}f\ a}(x,\gamma) \to \Sigma_{a:1\to\mathcal{A}}(\mathcal{U})^{\mathcal{F}f\ a}(x',\gamma')\\
&(\phi)^{\mathcal{F}(}\boldsymbol{a},x) = (a,(f)^{\mathcal{F}f\ a})
\end{aligned}$$

Lemma 7.6.4.8. *The terms* $\langle\!\langle c\rangle\!\rangle(1,\gamma)(*)$ *and* $\langle\!\langle c\rangle\!\rangle(0,\gamma)(*)$ *satisfy Eq.* (7.14)*Eq.* (7.15)*Eq.* (7.16)*.*

Proof. We mutually defined all terms $(\!|c|\!)(x,\gamma)(*)$ in Definition 7.6.4.5 and Definition 7.6.4.6. This implies that Eq. (7.14)Eq. (7.15)Eq. (7.16) hold. $\square$

Continuing Definition 7.6.4.4, we have the following:

Notation 7.6.4.9 (Evaluation of $(\!|\mathsf{Fam}|\!)(\!|D|\!) \Rightarrow (\!|\mathsf{Fam}|\!)(\!|D|\!)$ and projections). *1. We denote the component of a global element $\mathcal{F} : 1 \to (\!|\mathsf{Fam}|\!)(\!|D|\!) \Rightarrow (\!|\mathsf{Fam}|\!)(\!|D|\!)$ by*

$$\begin{aligned}
&\mathcal{F}_0(0,\gamma_0)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(0,\gamma_0) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(0,\gamma_0)\\
&\mathcal{F}_1(0,\gamma_0)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(1,r(\gamma_0)) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(1,r(\gamma_0))\\
&\mathcal{F}_{0,src}(1,\gamma_1)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(0,s(\gamma_1)) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(0,s(\gamma_1))\\
&\mathcal{F}_{1,src}(1,\gamma_1)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(1,r(s(\gamma_1))) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(1,r(s(\gamma_1)))\\
&\mathcal{F}_{0,tgt}(1,\gamma_1)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(0,t(\gamma_1)) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(0,t(\gamma_1))\\
&\mathcal{F}_{1,tgt}(1,\gamma_1)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(1,r(t(\gamma_1))) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(1,r(t(\gamma_1)))\\
&\mathcal{F}_R(1,\gamma_1)(*) : (\!|\mathsf{Fam}|\!)\mathcal{D}(1,\gamma_1) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(1,\gamma_1)
\end{aligned}$$

and for a global element $\mathfrak{x} : 1 \to (\!|\mathsf{Fam}|\!)\mathcal{D}$ with components

$$\begin{aligned}
&\mathfrak{x}(0,\gamma_0)(*) = (\mathcal{U}^{\mathfrak{x}}(0,\gamma_0), (t_0^{\mathfrak{x}}(0,\gamma_0), t_1^{\mathfrak{x}}(0,\gamma_0)))\\
&\mathfrak{x}(1,\gamma_1)(*) = (\mathcal{U}^{\mathfrak{x}}(1,\gamma_1), ((t_{0,src}^{\mathfrak{x}}(1,\gamma_1), t_{1,src}^{\mathfrak{x}}(1,\gamma_1)), (t_{0,tgt}^{\mathfrak{x}}(1,\gamma_1), t_{1,tgt}^{\mathfrak{x}}(1,\gamma_1))), t_R^{\mathfrak{x}}(1,\gamma_1))
\end{aligned}$$

where we sometimes suppress the superscripts '$\mathfrak{x}$' when they can be inferred from the context, we denote the components of the global element $\mathcal{F}\ \mathfrak{x} : 1 \to (\!|\mathsf{Fam}|\!)\mathcal{D}$ being the evaluation of $\mathcal{F}$ in $\mathfrak{x}$ by

$$\begin{aligned}
&\mathcal{F}\ \mathfrak{x}(0,\gamma_0) = ((\mathcal{U}^{\mathfrak{x}})^{\mathcal{F}}(0,\gamma_0), ((t_0^{\mathfrak{x}})^{\mathcal{F}}(0,\gamma_0), (t_1^{\mathfrak{x}})^{\mathcal{F}}(0,\gamma_0)))\\
&\mathcal{F}\ \mathfrak{x}(1,\gamma_1) =\\
&\quad ((\mathcal{U}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1), (((t_{0,src}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1), (t_{1,src}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1)), ((t_{0,tgt}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1), (t_{1,tgt}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1))), (t_R^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1))
\end{aligned}$$

2. We can write $\mathcal{F}$ ((x,γ)-component-wise) in two components

$$\begin{aligned}
&\mathcal{F}_{fst}(x,\gamma) : (1(x,\gamma) \to (\!|\mathsf{Fam}|\!)\mathcal{D}(x,\gamma)) \to (\!|\mathsf{Set}|\!)(x,\gamma)\\
&\mathcal{F}_{snd}(x,\gamma) : (\mathfrak{x} : 1(x,\gamma) \to (\!|\mathsf{Fam}|\!)\mathcal{D})(x,\gamma) \to (\mathcal{F}_{fst}\mathfrak{x} \Rightarrow \mathcal{D})(x,\gamma)
\end{aligned}$$

such that the evaluation of these components in $\mathfrak{x}$ return component-wise the first-, respectively second projections, i.e.

$$\begin{aligned}
&(\mathcal{F}_{fst}\mathfrak{x})(x,\gamma)(*) = (\mathcal{U}^{\mathfrak{x}})^{\mathcal{F}}(0,\gamma)(*)\\
&(\mathcal{F}_{snd}\mathfrak{x})(0,\gamma_0) = ((t_0^{\mathfrak{x}})^{\mathcal{F}}(0,\gamma_0), (t_1^{\mathfrak{x}})^{\mathcal{F}}(0,\gamma_0))\\
&(\mathcal{F}_{snd}\mathfrak{x})(1,\gamma_1) = ((((t_{0,src}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1), (t_{1,src}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1)), ((t_{0,tgt}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1), (t_{1,tgt}^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1))), (t_R^{\mathfrak{x}})^{\mathcal{F}}(1,\gamma_1))
\end{aligned}$$

3. *For a code* $c : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}$, *we write* $\mathcal{F}^c := \langle\!\langle c \rangle\!\rangle$.

Remark 7.6.4.10. *An component* $\mathcal{F}^c(x,\gamma)(*) : ((\!|\mathsf{Fam}|\!)\mathcal{D} \Rightarrow (\!|\mathsf{Fam}|\!)\mathcal{D})(x,\gamma)$ *is (a priori) not simply the decoding of a* DS D D *code since for example the* $(0,\gamma_0)$*-instance* ${\mathcal{F}^c}_1 \mathfrak{X}(0,\gamma_0)(*) = ((t_0)^{\mathcal{F}^c}_{(0,\gamma)}, (t_1)^{\mathcal{F}^c}_{(0,\gamma)})$ *refers to an* $(1, r(\gamma_0))$*-instance — namely* $(\mathcal{U}^{\mathfrak{X}})^{\mathcal{F}^c}(1, r(\gamma_0))$ *as domain of* $(t_1)^{\mathcal{F}^c,1}_{(0,\gamma_0)} : (\mathcal{U}^{\mathfrak{X}})^{\mathcal{F}^c}(1, r(\gamma_0)) \to \mathcal{D}(1, r(\gamma_0))$*— as well.*

7.6.5 Construction of Initial Algebras

The proof that initial algebras for $(\!|\mathsf{DS}|\!)$-codes exist is just a terminological variation of the one for DS-codes (see [38]) once we explained (= introduced notation for) how to deal with families in place of sets. Since we did not repeat this proof in Chapter 4 we spell out the adaption here. The structure of the proof is also similar to the one we gave in Section 6.4: first we show (in Lemma 7.6.5.5) that $(\!|\mathsf{DS}|\!)$-functors are κ-continuous under the assumption that their sets of argument $\mathsf{Aux}(c, (\mathcal{U}, t))$ are bounded by V_κ, and then we show that the initial sequence of a $(\!|\mathsf{DS}|\!)$-functor is a monotone κ-sequence satisfying this assumption; the existence of initial algebras then follows by the usual argument (see [7]).

7.6.5.0.1 Aux

Definition 7.6.5.1 (Aux). *Given a code* $c : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}$ *a global element* $(\mathcal{U}, t)$ *of* $(\!|\mathsf{Fam}D|\!)$, *the set* $\mathsf{Aux}(c, (\mathcal{U}, t))$ *of premises of inductive arguments of* c *with respect to* $(\mathcal{U}, t)$ *is defined by induction on* c*:*

$$\begin{aligned}
\mathsf{Aux}(\iota^-\, d, (\mathcal{U}, t)) &= \emptyset \\
\mathsf{Aux}(\sigma^-\, \mathcal{A}\, f, (\mathcal{U}, t)) &= \bigcup_{\mathfrak{X}:1\to\mathcal{A}} \mathsf{Aux}(f\, \mathfrak{X}, (\mathcal{U}, t)) \\
\mathsf{Aux}(\delta^-\, \mathcal{A}\, \mathcal{F}, (\mathcal{U}, t)) &= \{\mathcal{A}\} \cup \bigcup_{\mathcal{G}:1\to\mathcal{A}\Rightarrow\mathcal{U}} \mathsf{Aux}(\mathcal{F}(t \circ \mathcal{G})(\mathcal{U}, t))
\end{aligned}$$

The notation of the previous definition is a compromise between Definition 7.6.4.1 and its rewrite in terms of global elements Definition 7.6.4.3. The idea is to have one case for each constructor, but there are different options of how to write the constructors: they can either each define a family of sets or one set per index. The Agda formalization of Definition 7.6.4.1 is still different and has constructors e.g. σ_0 taking as first argument a $\gamma_0 : \Gamma 0$. Clearly these options are all equivalent and irrelevant for the following proof that initial algebras exist.

Notation 7.6.5.2. *If* $\mathcal{A}$ *is a family of reflexive graphs or an indexed set (in this section the reflexive graph structure is not relevant and only the underlying family of sets matters), we write* $\mathcal{A} \in V_\kappa$ *for* κ *an inaccessible, if* $\mathcal{A}(x,\gamma) \in V_\kappa$ *for all indices* (x,γ)*. Likewise for families of reflexive graphs or indexed sets* $\mathcal{A}$*, and* $\mathcal{B}$*, we write* $\mathcal{A} \subseteq \mathcal{B}$ *if* $\mathcal{A}(x,\gamma) \subseteq \mathcal{B}(x,\gamma)$ *for all* $(x,\gamma) \in \int \Gamma$*, and for a set* V*, we write* $\mathcal{A} \subseteq V$ *if* $\mathcal{A}(x,\gamma) \subseteq V$ *for all* $(x,\gamma) \in \int \Gamma$*. The union* $\bigcup_{\alpha<\kappa} \mathcal{A}_\alpha$ *of a family (of sets or reflexive graphs) is understood as a level-wise union of sets, and likewise for the union* $\mathcal{A} \cup \mathcal{B}$ *of two families.*

If we were to fashion Definition 7.6.5.1 in a style corresponding to the component-wise Definition 7.6.4.1 instead of corresponding to the global-element style of Definition 7.6.4.3, we would define sets $\mathsf{Aux}(c, (\mathcal{U}, t), (x, \gamma))$*, and regard* $\mathsf{Aux}(c, (\mathcal{U}, t))$ *as an indexed family of sets. We would apply the above notation to this family and the expression* $\mathsf{Aux}(c, (\mathcal{U}, t)) \in V_\kappa$ *would still be defined.*

7.6.5.0.2 Continuity We now show that if for certain $(\mathcal{U}, t)$, all $A \in \mathsf{Aux}(c, (\mathcal{U}, t))$ are bounded, then $\mathcal{F}^c$ is κ-continuous for an inaccessible $\kappa < \mathsf{M}$. And again by Adamek's theorem [7] it follows that $\mathcal{F}^c$ has an initial algebra.

The following is an adaption of the notation below [39, Theorem 8.1].

Definition 7.6.5.3 ($\leq_{(\!|\mathsf{Fam}|\!)\mathcal{D}}$). *For* $(\mathcal{U}, t)$, $(\mathcal{U}', t') : 1 \to (\!|\mathsf{Fam}|\!)\mathcal{D}$ *we define a partial order.* $(\mathcal{U}, t) \leq_{(\!|\mathsf{Fam}|\!)\mathcal{D}} (\mathcal{U}', t')$ *if for all* $(x, \gamma) : \int \Gamma$

$$\begin{aligned}
&\mathcal{U}(x, \gamma) \subseteq \mathcal{U}'(x, \gamma)\\
&t'(x, \gamma) \restriction \mathcal{U}(x, \gamma) = t(x, \gamma)
\end{aligned}$$

where the last line is understood component-wise, i.e. as the set-level restriction of the maps.

$$\begin{aligned}
&t_0{}'(0, \gamma_0) \restriction \mathcal{U}(0, \gamma_0) = t_0(0, \gamma_0)\\
&t_1{}'(0, \gamma_0) \restriction \mathcal{U}(1, r(\gamma_0)) = t_1(0, \gamma_0)\\
&t_{0,src}{}'(1, \gamma_1) \restriction \mathcal{U}(0, s(\gamma_1)) = t_{0,src}(1, s(\gamma_1))\\
&t_{1,src}{}'(1, \gamma_1) \restriction \mathcal{U}(1, r(s(\gamma_1))) = t_{1,src}(0, \gamma_0)\\
&t_{0,tgt}{}'(1, \gamma_1) \restriction \mathcal{U}(0, t(\gamma_1)) = t_{0,tgt}(0, \gamma_0)\\
&t_{1,tgt}{}'(1, \gamma_1) \restriction \mathcal{U}(1, r(t(\gamma_1))) = t_{1,tgt}(1, \gamma_1)\\
&t_R{}'(1, \gamma_1) \restriction \mathcal{U}(1, \gamma_1) = t_R(1, \gamma_1)
\end{aligned}$$

Definition 7.6.5.4 (Monotone κ-sequence).

1. *For an inaccessible κ, a sequence $(\mathcal{U}^\alpha, t^\alpha)_{\alpha<\kappa}$ of global elements of $(\!|\mathsf{Fam}(D)|\!)$ is called a* monotone κ-sequence *if for all $\alpha < \beta$ we have $(\mathcal{U}^\alpha, t^\alpha) \leq_{(\!|\mathsf{Fam}|\!)\mathcal{D}} (\mathcal{U}^\beta, t^\beta)$.*
2. *We understand the union $\bigcup_{\alpha<\kappa}(\mathcal{U}^\alpha, t^\alpha)$ of a monotone κ-sequence pointwise in the sense of Definition 7.6.5.3. We also use the notation $(\bigcup_{\alpha<\kappa} \mathcal{U}^\alpha, \bigcup_{\alpha<\kappa} t^\alpha)$ for the union.*

Lemma 7.6.5.5 (Bounded continuity of $(\!|\mathsf{DS}|\!)$-functors in monotone κ-sequences). *Let κ be inaccessible and $(\mathcal{U}^\alpha, t^\alpha)_{\alpha<\kappa}$ be a monotone κ-sequence, let c be a code.*

Assume for some $\alpha_0 < \kappa$ that

$$\mathsf{Aux}(c, (\mathcal{U}^\alpha, t^\alpha)) \subseteq V_\kappa \tag{7.17}$$

for all $\alpha_0 \leq \alpha < \kappa$. Then $\mathcal{F}^{c}_{fst}(\mathfrak{U}, t)$ is κ-continuous in $(\mathfrak{U}, t)$, i.e.

$$\mathcal{F}^{c}_{fst}(\bigcup_{\alpha<\kappa} U^\alpha, \bigcup_{\alpha<\kappa} T^\alpha) = \bigcup_{\alpha<\kappa} \mathcal{F}^{c}_{fst}(U^\alpha, T^\alpha) \ .$$

where the union $\bigcup_{\alpha<\kappa}(\mathfrak{U}^\alpha, t^\alpha)$ is understood pointwise in the sense of Definition 7.6.5.3.

Proof. The direction $\supseteq$ follows from Lemma 7.6.5.6. We prove $\subseteq$ by induction over c:

- If $c = \iota^-\ d$, then

$$\mathcal{F}^{\iota^-\ d}_{fst} \bigcup_{\alpha<\kappa}(\mathfrak{U}^\alpha, t^\alpha) = \bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha = \bigcup_{\alpha<\kappa} \mathcal{F}^{\iota^-\ d}_{fst}(\mathfrak{U}^\alpha, t^\alpha) \ .$$

- If $c = \sigma^-\ \mathcal{A}\, f$, the statement follows simply by induction like in case of DS.
- If $c = \delta^-\ \mathcal{A}\, \mathcal{F}$, the statement follows like for DS as well: we have to show that there exists an $\alpha < \kappa$ such that for all $(x, \gamma) \in \int \Gamma$, and all[9]

$$a : \mathcal{F}^{\delta^-\ \mathcal{A}\, \mathcal{F}}_{fst} \bigcup_{\alpha<\kappa}(\mathfrak{U}^\alpha, t^\alpha)(x, \gamma)$$

  we have

$$a : \mathcal{F}^{\delta^-\ \mathcal{A}\, \mathcal{F}}_{fst}(\mathfrak{U}^\alpha, t^\alpha)(x, \gamma) \ .$$

  Indeed, we have $a = (\mathcal{G}, \chi)$ where

$$\mathcal{G} : (\mathcal{A} \Rightarrow \bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha)(x, \gamma)$$
$$\chi : (\mathcal{F}^{\mathcal{F}((\bigcup_{\alpha<\kappa} t^\alpha)\circ\mathcal{G})}_{fst} \bigcup_{\alpha<\kappa}(\mathfrak{U}^\alpha, t^\alpha))(x, \gamma) \ .$$

  By assumption Eq. (7.17) $\mathcal{A} \in V_\kappa$, and since κ is inaccessible there exists a $\beta < \kappa$ such that $\mathcal{G} : (\mathcal{A} \Rightarrow \bigcup_{\alpha<\beta} \mathfrak{U}^\alpha)(x, \gamma)$: to see this, we apply Lemma 1.1.0.13 to the component functions: for an index of the form $(0, \gamma_0)$, we have $\mathcal{G} = (\mathcal{G}_0, \mathcal{G}_1)$ where

$$\mathcal{G}_0 : \mathcal{A}(0, \gamma_0) \to \bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha(0, \gamma_0)$$
$$\mathcal{G}_1 : \mathcal{A}(1, r(\gamma_0)) \to \bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha(1, r(\gamma_0)) \ ,$$

  and in each case Lemma 1.1.0.13 gives us $\beta_0 < \kappa$, respectively $\beta_1 < \kappa$ such that $\mathcal{G}_0$, respectively $\mathcal{G}_1$ restrict to

$$\mathcal{G}_0 : \mathcal{A}(0, \gamma_0) \to \bigcup_{\alpha<\beta_0} \mathfrak{U}^\alpha(0, \gamma_0)$$
$$\mathcal{G}_1 : \mathcal{A}(1, r(\gamma_0)) \to \bigcup_{\alpha<\beta_1} \mathfrak{U}^\alpha(1, r(\gamma_0)) \ ,$$

[9]Notice that we can here (and in similar situations in the following) not argue with global elements (e.g. assuming $a : 1 \to \mathcal{F}^{\delta^-\ \mathcal{A}\, \mathcal{F}}_{fst} \bigcup_{\alpha<\kappa}(\mathfrak{U}^\alpha, t^\alpha)$) since $\mathcal{RG}\Gamma$ is not in general a wellpointed topos.

and we chose $\beta := \mathsf{max}\{\beta_0, \beta_1\}$. Likewise for an index of the form $(1, \gamma_1)$, Lemma 1.1.0.13 gives us $\beta_{0.src}, \beta_{1.src}, \beta_{0.tgt}, \beta_{1.tgt}, \beta_R < \kappa$, and we choose β to be the maximum of these. So, in particular $\mathcal{G} : (\mathcal{A} \Rightarrow \mathcal{U}^\beta)(x, \gamma)$, and $\mathcal{F}((\bigcup_{\alpha<\kappa} t^\alpha) \circ \mathcal{G} = \mathcal{F}((t^\beta) \circ \mathcal{G}$. Without loss of generality $\alpha_0 \leq \beta$ (otherwise go on with $\beta := \alpha_0$) and we are again in the situation to apply the assumption: i.e. $\beta \leq \alpha < \kappa$ implies that $\mathsf{Aux}(\mathcal{F}(t^\alpha \circ \mathcal{G}), (\mathcal{U}^\alpha, t^\alpha)) \in V_\kappa$ and by inductive hypothesis there exists $\beta' < \kappa$ such that $\chi : (\mathcal{F}_{fst}^{\mathcal{F}(t^\alpha \circ \mathcal{G})}(\mathcal{U}^{\beta'}, t^{\beta'}))(x, \gamma)$. In total, we get

$$a = (\mathcal{G}, \chi) : (\mathcal{F}_{fst}^{\delta^- \mathcal{A}\, \mathcal{F}}(\mathcal{U}^\alpha, t^\alpha))(x, \gamma)$$

as desired by taking α to be the maximum of β, and β'.

Lemma 7.6.5.6. *Let $c : 1 \to (\!|\mathsf{DS}|\!)\mathcal{D}$, and $(\mathcal{U}, t) \leq_{(\!|\mathsf{Fam}|\!)\mathcal{D}} (\mathcal{U}', t')$. Then*

1. *$\mathcal{F}_{fst}^c(\mathcal{U}, t) \subseteq \mathcal{F}_{fst}^c(\mathcal{U}', t')$, and*
2. *$\mathcal{F}_{snd}^c(\mathcal{U}', t') \restriction \mathcal{F}_{fst}^c(\mathcal{U}, t) = \mathcal{F}_{snd}^c(\mathcal{U}, t)$.*

Proof. The proof follows by induction and monotonicity in a way analogous to the case of DS by arguing pointwise.

We want to apply Lemma 7.6.5.5 to the initial sequence of the corresponding functor:

Definition 7.6.5.7. *For $c : (\!|\mathsf{DS}|\!)\mathcal{D}$ we define the initial sequence of $\mathcal{F}^c$ by*

$$\begin{aligned} \mathcal{U}^0 &= \emptyset & t^0\, \chi &= \emptyset \\ \mathcal{U}^{\alpha+1} &= \mathcal{F}_{fst}^c(U^\alpha, t^\alpha) & t^{\alpha+1}\, \chi &= \mathcal{F}_{snd}^c(U^\alpha, T^\alpha)\, \chi \\ \mathcal{U}^\lambda &= \bigcup_{\beta<\lambda} \mathcal{U}^\beta & t^\lambda\, \chi &= t^\beta \chi \quad \textit{where } \chi : 1 \to \mathcal{U}^\beta \end{aligned}$$

Lemma 7.6.5.8. *Let c be a code and $(\mathcal{U}^\alpha, t^\alpha)_{\alpha \in \mathsf{Ord}}$ the initial sequence of the associated functor $\mathcal{F}^c$. If $\alpha < \beta$ then $\mathcal{U}^\alpha \subseteq \mathcal{U}^\beta$ and $t^\beta \restriction \mathcal{U}^\alpha = t^\alpha$.*

Proof. Induction on α, β, using Lemma 7.6.5.6 for the limit cases.

We now prove that the assumption of Lemma 7.6.5.5 holds. This proof is the only occasion where the Mahlo property of M is used.

Lemma 7.6.5.9. *Let c be a code and $(\mathcal{U}^\alpha, t^\alpha)_\alpha$ the initial sequence of the associated functor $\mathcal{F}^c$. There exists an inaccessible κ such that $\mathsf{Aux}(c, (\mathcal{U}^\alpha, t^\alpha)) \subseteq V_\kappa$ for all $\alpha < \kappa$.*

Proof. We define an increasing function $f : \mathsf{Ord} \to \mathsf{Ord}$, that will send an ordinal β to the rank $f(\beta)$ of the Aux-set of the value of the application of the functor $\mathcal{F}^c$ to an initial sequence of rank β; i.e. β and its value $f(\beta)$ will stand in the relation:

$$\text{for all } \beta' < \mathsf{M} \text{ (if } \mathcal{U}^{\beta'} \subseteq V_\beta \text{ then } \mathcal{U}^{\beta'+1} \cup \mathsf{Aux}(c, \mathcal{U}^{\beta'}, t^{\beta'})) \subseteq V_{f(\beta)}) \ . \tag{7.18}$$

More formally, we define f by transfinite recursion by:

$$\begin{aligned} f(\beta) = \min\{\alpha \mid & (\forall \beta' < \beta)\big(f(\beta') < \alpha\big) \wedge \\ & (\forall \beta' < \mathsf{M})\big(\mathcal{U}^{\beta'} \subseteq V_\beta \implies \mathcal{U}^{\beta'+1} \cup \mathsf{Aux}(c, (\mathcal{U}^{\beta'}, t^{\beta'})) \subseteq V_\alpha\big)\} \end{aligned}$$

The first conjunct ascertains that f is increasing, and the second makes (7.18) true.

Since in our model all sets are (supposed to be) of rank at most M, we are interested in the restriction of f to M. In fact we chose M precisely to find an inaccessible κ as desired in the statement of the lemma we are proving. Again, since all sets in our model shall be of rank at most M, we want to show now, that also the image of f is contained in M; we use for this function the same letter $f : \mathsf{M} \to \mathsf{M}$.

Finally we show $\mathsf{Aux}(\varphi, (\mathcal{U}^\alpha, t^\alpha)) \subseteq V_\kappa$ by induction on α, using (7.18).

<u>Claim:</u> $f : \mathsf{M} \to \mathsf{M}$.
Proof of this claim. Let $\beta < \mathsf{M}$ and note that

$$\begin{aligned} f(\beta) = \min\{\alpha \mid & (\forall \beta' < \beta)\big(f(\beta') < \alpha\big) \wedge \\ & (\forall \beta' \in \{\beta' \in \mathsf{M} \mid \mathcal{U}^{\beta'} \subseteq V_\beta\})\big(\mathcal{U}^{\beta'+1} \cup \mathsf{Aux}(c, (\mathcal{U}^{\beta'}, t^{\beta'})) \subseteq V_\alpha\big)\} \ , \end{aligned}$$

further that $B := \{\beta' \in \mathsf{M} \mid \mathcal{U}^{\beta'} \subseteq V_\beta\} \in V_{\beta+1} \subseteq V_\mathsf{M}$ so that $|B| < \mathsf{M}$. For each $\beta' \in B$, we have $U^{\beta'+1} \cup \mathsf{Aux}(\gamma, (\mathcal{U}^{\beta'}, t^{\beta'})) \subseteq V_\mathsf{M}$ and hence $\mathcal{U}^{\beta'+1} \cup \mathsf{Aux}(c, (\mathcal{U}^{\beta'}, t^{\beta'})) \subseteq V_{\alpha_{\beta'}}$ for some $\alpha_{\beta'} < \mathsf{M}$ since M is inaccessible. Thus $f(\beta) \leq \sup_{\beta'} \alpha_{\beta'} < \mathsf{M}$ by the regularity of M. $\square$

We finally would like to construct a candidate κ as desired in the statement of the lemma as an inaccessible fixed point of f which (by definition) entails that all further iterations of $\mathcal{F}^c$ have the same rank κ. To this end, we want to use the Mahlo property stating exactly that every increasing *normal* (see Definition 1.1.0.16) endofunction $f : \mathsf{M} \to \mathsf{M}$ has an inaccessible fixed point. However, f is —albeit increasing— not necessarily continuous, and thus not necessarily normal. We thus apply "Newton's fixed-point method" and derive the new function

$$\begin{aligned} \theta &: \mathsf{Ord} \to \mathsf{Ord} \\ \theta(\alpha) &= f^\alpha(0) \end{aligned}$$

which is continuous by construction, and increasing since f is, and thus normal[10]. So, even if our wish that f have an inaccessible fixed point is not fulfilled, we get a fixed point of θ, and since we start the iteration of $\mathcal{F}^c$ in the empty set, this still matches the intuitive proof idea.

But we still have to take care of that we need this fixed point to be $< \mathsf{M}$, i.e. we have to recover that:

<u>Claim:</u> $\theta : \mathsf{M} \to \mathsf{M}$
Proof of this claim. We prove that $\theta(\alpha) < \mathsf{M}$ for $\alpha < \mathsf{M}$ by transfinite induction over α.

[10]Newton's original fixed-point method which is formulated for real numbers would now go on to argue that if the sequence $x_0 := 0$, $x_{n+1} := f(x_n)$ (there are no limit steps in the original version) converges to an x, and f is continuous, then x is a fixed point of f.

The base case and successor case are clear, since $f : \mathsf{M} \to \mathsf{M}$. If $\lambda < \mathsf{M}$ is a limit, the statement follows from the conjunction of regularity of M, and continuity of θ: we have for $\lambda = \lim_{\xi\to\lambda} \gamma_\xi$ that $\theta(\lim_{\xi\to\lambda} \gamma_\xi) = \lim_{\xi\to\lambda} \theta(\gamma_\xi)$ by continuity of θ. Assume $\theta(\lambda) \geq \mathsf{M}$, then since $\mathsf{M} = \mathsf{cf}(\mathsf{M}) = \mathsf{min}\{\beta \mid \exists$ increasing β-sequence in M $(\alpha_\xi)_{\xi<\beta}, \lim_{\xi\to\beta} \alpha_\xi = \mathsf{M}\}$ it follows $\lambda \geq \mathsf{M}$, contradiction. □

Hence by the Mahlo property, θ has an inaccessible fixed point $\kappa < \mathsf{M}$ which we use in the last few steps to infer the actual statement of the lemma.

Claim: $f : \kappa \to \kappa$.
Proof of this claim. Assume $\alpha < \kappa$. Since κ is inaccessible, $\alpha < \alpha + 1 < (\alpha + 1) + 1 < \kappa$ Thus

$$f(\alpha) < f(\alpha + 1) \leq f(\theta(\alpha + 1)) = f(f^{\alpha+1}(0)) = \theta((\alpha + 1) + 1) < \theta(\kappa) = \kappa$$

i.e. $f : \kappa \to \kappa$. □

This combined with (7.18) implies:

$$\text{if } \mathfrak{U}^{\beta'} \subseteq V_\beta \text{ then } U^{\beta'+1} \cup \mathsf{Aux}(\mathfrak{c}, (\mathfrak{U}^{\beta'}, \mathfrak{t}^{\beta'})) \subseteq V_\kappa \tag{7.19}$$

for all $\beta < \kappa$ (since $f(\beta) < \kappa$ implies $V_{f(\beta)} \subseteq V_\kappa$).

Claim: $\mathfrak{U}^\alpha \subseteq V_\kappa$ for all $\alpha < \kappa$.
Proof of this claim. The proof is by induction on α:

- If $\alpha = 0$, then $U^0 = \emptyset \subseteq V_\kappa$.
- If $\alpha = \beta + 1$, then $U^\beta \subseteq V_\kappa$ by the induction hypothesis, and the case follows by (7.19).
- If $\alpha = \lambda$ limit, then $\mathfrak{U}^\lambda = \bigcup_{\beta<\lambda} \mathfrak{U}^\beta \subseteq V_\kappa$ since the $\mathfrak{U}^\beta \subseteq V_\kappa$ by the inductive hypothesis, and κ is inaccessible. □

The previous claim and (7.19) entail $\mathsf{Aux}(\varphi, (\mathfrak{U}^\alpha, \mathfrak{t}^\alpha)) \subseteq V_\kappa$ as desired which completes the proof. □

By combining Lemma 6.4.2.4 and Lemma 6.4.2.7, we get:

Theorem 7.6.5.10. *In ZFC + M + I, all functors $\mathcal{F}^\mathfrak{c}$ associated to a code $\mathfrak{c} : 1 \to (\!|\mathsf{DS}|\!)\mathfrak{D}$ have initial algebras.*

Proof. By Lemma 6.4.2.4 —using Lemma 7.6.5.8, and Lemma 7.6.5.9— we obtain

$$\begin{aligned}\mathcal{F}^\mathfrak{c}_{fst}(\bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha, \bigcup_{\alpha<\kappa} \mathfrak{t}^\alpha) &= \bigcup_{\alpha<\kappa} \mathcal{F}^\mathfrak{c}_{fst}(\mathfrak{U}^\alpha, \mathfrak{t}^\alpha) \\ &= \bigcup_{\alpha<\kappa} \mathfrak{U}^{\alpha+1} \\ &= \bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha \ .\end{aligned}$$

By Lemma 7.6.5.8, $\mathcal{F}^\mathfrak{c}_{snd}(\bigcup_{\alpha<\kappa} \mathfrak{U}^\alpha, \bigcup_{\alpha<\kappa} \mathfrak{t}^\alpha) = \bigcup_{\alpha<\kappa} \mathfrak{t}^\alpha$, so that the initial sequence converges after κ steps. By Adamek's Theorem [7, Thm 3.1.4], $\mathcal{F}^\mathfrak{c}$ has an initial algebra.

7.7 Unary-Relational Parametricity for DS

As shown by [120], n-ary parametricity for all $n : \mathbb{N}$ can be dealt with in terms of considering only the cases $n = 1$, and $n = 2$. Moreover, if we model parametricity via reflexive graphs, we can express the $n = 1$ case as a degenerate case of $n = 2$. In this section we will only define the appropriate category of presheaves for this degenerate unary case.

Definition 7.7.0.1 (SR). *We define the following category*

$$\mathsf{SR} = \{\mathsf{0}, \mathsf{1}, r : \mathsf{0} \to \mathsf{1}, s : \mathsf{1} \to \mathsf{0} \mid s \circ r = \mathsf{id}\} \subseteq \mathcal{SR} \times \mathcal{SR} \,.$$

The category $\mathcal{SR} = \mathsf{SR}^{\mathsf{op}} \to \mathsf{Set}$ *is the category of presheaves of subsets equipped with a specified retraction.*

Remark 7.7.0.2 ($Sub(\mathsf{Set})$). *Obviously* $\mathcal{SR}$ *equipped with the relation*

$$\{(\mathcal{A}, \mathcal{B}) \mid \mathcal{A}(\mathsf{0}) = \mathcal{B}(\mathsf{0}),\, \exists\, k : \mathcal{A}(\mathsf{1}) \to \mathcal{B}(\mathsf{1}),\, \mathcal{B}(s) \circ k = \mathcal{A}(s)\}$$

factored by isomorphisms (i.e. those instances where the k above is an isomorphism) is a full subcategory of $Sub(\mathsf{Set})$ *(the category of subobjects of sets) where* $\mathcal{SR}$ *consists of exactly those objects of* $Sub(\mathsf{Set})$ *which are split monomorphisms.* $Sub(\mathsf{Set})$ *is of central interest in classical texts on logical relations, see e.g. [67]*

Chapter 8

Epilogue

8.1 Summary

The main focus of this thesis has been the question of compositionality of codes for inductive-recursive definitions. First we addressed the question of compositionlity of classical Dybjer-Setzer codes.

While this question is still open, we developed a characterization of compositionality for these codes which lets us believe that it is unlikely that DS-codes are composable.

Based on the insights that the conjunction of powers of codes by sets and a monad structure —or at least a bind operation— on the system of codes plays a crucial role in defining composition of codes, we defined two new systems UF, and PN of codes for induction-recursion which relate by semantics preserving inclusions $\mathsf{UF} \hookrightarrow \mathsf{DS} \hookrightarrow \mathsf{PN}$. The constructors of the three systms are displayed in the following table:

UF	DS	PN
ι_{UF}: $\mathsf{Uni}\ \mathrm{D}$ σ_{UF}: $\big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D$ δ_{UF}: $\big(c : \mathsf{Uni}\ D\big) \to (\mathsf{Info}\ c \to \mathsf{Set}) \to \mathsf{Uni}\ D$	$\iota_{\mathsf{DS}} : E \to \mathsf{DS}\ D\ E$ $\sigma_{\mathsf{DS}} : \big(A : \mathsf{Set}\big) \to (A \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$ $\delta_{\mathsf{DS}} : \big(A : \mathsf{Set}\big) \to ((A \to D) \to \mathsf{DS}\ D\ E) \to \mathsf{DS}\ D\ E$	$\mathsf{id}_{\mathsf{PN}} : \mathsf{Poly}\,\mathsf{D}$ $\mathsf{con} : (A : \mathsf{Set}) \to \mathsf{Poly}\,\mathsf{D}$ $\mathsf{sigma} : (S : \mathsf{Poly}\,\mathsf{D}) \to (\mathsf{Info}\,\mathsf{S} \to \mathsf{Poly}\,\mathsf{D}) \to \mathsf{Poly}\,\mathsf{D}$ $\mathsf{pi} : (A : \mathsf{Set}) \to (A \to \mathsf{Poly}\,\mathsf{D}) \to \mathsf{Poly}\,\mathsf{D}$
$\mathsf{Info}\ \iota_{\mathsf{UF}} = 1$ $\mathsf{Info}\ (\sigma_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma)$ $\mathsf{Info}\ (\delta_{\mathsf{UF}}\ c\ A) = \big(\Sigma\gamma : \mathsf{Info}\ c\big)(A\ \gamma \to D)$		$\mathsf{Info}\,\mathsf{id}_{\mathsf{PN}} = \mathrm{D}$ $\mathsf{Info}\,(\mathsf{con}\,\mathsf{A}) = \mathrm{A}$ $\mathsf{Info}\,(\mathsf{sigma}\,\mathsf{S}\,\mathsf{F}) = \big(\Sigma x : \mathsf{Info}\,\mathsf{S}\big)(\mathsf{Info}\,(\mathsf{F}\,\mathsf{x}))$ $\mathsf{Info}\,(\mathsf{pi}\,\mathsf{A}\,\mathsf{F}) = \big(x : A\big) \to (\mathsf{Info}\,(\mathsf{F}\,\mathsf{x}))$

The sub-system of UF consisting of codes with a more uniform structure than that of DS-codes facilitating compositionality. UF has powers of codes by sets but it has no monad structure and only a 'combined power-and-bind operation' can be defined on it. Consistency of UF is implied by including it in the existing model of DS.

The super-system PN of polynomial codes —which we called this way since they are 'constructed from sums and products' like polynomials has powers of codes by sets and a monad structure. Consistency of PN is shown providing a model in ZFC supplemented by a Mahlo cardinal and a 1-inaccessible cardinal I containiing M; this model is similar to the existing model of DS but (among other differences) that the latter has the slightly weaker requirement of I to be 0-inaccessible.

Since the fact that one constructor of the system PN generates powers of codes invites the question about further differences between the systems DS and PN, we defined two further systems $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \hookrightarrow \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ such that $\mathsf{DS} \hookrightarrow \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \hookrightarrow \mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}} \hookrightarrow \mathsf{PN}$, where $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ is PN with the constructor for powers removed, and $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}}$ is an annotated uniform version of $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$. We could give a translation $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Contbt}} \to \mathsf{DS}$ but it is still unclear whether $\mathsf{PN}^{-\mathsf{pi}}_{\mathsf{Cont}}$ can be translated into DS as well.

Finally (and unrelated to compositionality), we returend to the original system DS and its model to address the question whether the latter can be extended to a relationally parametric model. This question is motivated by the idea to use relational parametricity to show that all definable DS-codes are uniform; this application is however not covered in this thesis and is left for future work.

8.2 Conclusion

To conclude, we want to give a more conceptual explanation of what is difference between Dybjer-Setzer's IR systems and the new ones. Asking what might be not be present in Dybjer-Setzer's system to facilitate composition, we need to refer to semantics of induction-recursion. As we already mentioned, strict positivity is a characterization for inductive definitions that can to some extend be considered to be a semantical one since it is informed by universal properties of set-operations in the intended model.

The term 'strict positivity' can be characterized by the property of being definable by a certain choice of operations. The eponymous operation, namely the constructor $\longrightarrow$ for exponentials that needs to be constrained to have an inductive argument only in *strictly positive position*, i.e. to the right of an arrow and a constant (set) to the left —thus becoming an operator forming powers and not exponentials— is only one of them.

In case of induction-recursion, we can thus consider the operations on families that can be arranged for by the different axiomatizations we want to compare. One difference between sets and families is that powering might change the indexing set[1] and this becomes a problem e.g. in case of a constructor

$$\_ \longrightarrow \_ : (A : \mathsf{Set}) \to \mathsf{DS}\ D\ E \to \mathsf{DS}\ D\ (A \to E)$$

(with the preponderant '$(A \to E)$') that we needed —or at least used— to realize compositionality. So, while the term 'strict positivity' focuses attention on the problematic operation $\longrightarrow$ and suggests as solution to put the inductive argument in the right position,

[1]If $X : \mathsf{Fam}\ D$, then $(A \longrightarrow_{\mathsf{Fam}} X) : \mathsf{Fam}\ (A \to D)$ and these types are obviously identical only if $D = 1$, i.e. if $\mathsf{Fam}\ D \simeq \mathsf{Set}$.

in case of families also this solution is not available if the family operations one is willing to admit must not change the (large) set D indexing families $\mathsf{Fam}(D)$. A similar disparity appears for the operation $\times$ forming binary products since Set_1 is closed under it while $\mathsf{Fam}(D)$ is not in general closed under it. Thus, to talk about 'strict positivity for families' is misleading.

Another viewpoint on this is that if $c : \mathsf{DS}\ D\ E$ is code, then all its subcodes of c are of the same type $\mathsf{DS}\ D\ E$ while in the system PN we have defined, the second argument 'E' need not be E for every subcode of a code $\mathsf{PN}\ D\ E$. Not insisting in this strict regiment on subcodes apparently gives more flexibility that can be used to define codes.

A categorical view on this is that while the categories $\mathsf{Fam}(D)$ are usually not well endowed with limits, the comma category i/Set_1 —where $i : \mathsf{Set} \to \mathsf{Set}_1$ is the inclusion (induced by cumulativity)— which is equivalent to the category $\Sigma_{D:\mathsf{Set}_1}\mathsf{Fam}(D)$ (with morphisms appropriately defined) is always a topos and as such (by locally cartesian closedness) has exponentials and powers. We did not explore the possibility to define a version of IR decoding to an endomorphism of the codomain fibration $i/\mathsf{Set}_1 \to \mathsf{Set}_1$.

8.3 Future Work

It is still unclear whether the systems of induction-recursion we have seen are really different and if so what the nature of their difference is. At the end of Section 4.1 we have presented a class of candidate codes which we believe cannot be definable in the system DS but even if we had by coincidence found an instantiation of one of these codes and a proof that no family defined by DS can be isomorphic to the family defined by this code, it would still be unclear what the intuitive difference between families defined by the different versions of IR is since e.g. uniformity is a property of codes — not necessarily of what they define.

Questions that are closely related or even equivalent to that whether the systems differ are whether $\mathsf{DS} + \pi$ is functorial (in its second argument), and if yes, if it is additionally monadic.

Regarding the system UF there is moreover a slight mismatch between the intuition we have of uniform codes and their technical realization: we motivated uniform codes as DS codes in which all branches have the same length. In the formalization however branch length does not explicitly occur but the motivating property is only implied by it. One can try to to approach this by defining a system equivalent to UF by annotating DS codes by their branch length.

Since the motivating example for induction-recursion is universes, it would be interesting to find kinds of universes that are characteristic to the different versions of IR.

As for developing the semantic theory of IR, we mentioned in Remark 3.2.1.17 that the absence of morphisms of codes might be an obstacle. As mentioned there, this might be interesting for applications of induction-recursion to higher category theory.

Bibliography

[1] Michael Abbott. “Categories of Containers”. PhD thesis. University of Leicester, 2003. URL: https://www.cs.le.ac.uk/people/ma139/docs/thesis.pdf (cit. on pp. 21, 23).

[2] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. “Categories of Containers”. In: *Foundations of Software Science and Computation Structures*. Ed. by Andrew D. Gordon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 23–38. ISBN: 978-3-540-36576-1 (cit. on p. 23).

[3] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. “Containers: Constructing strictly positive types”. In: *TCS* 342.1 (2005), pp. 3–27. ISSN: 0304-3975 (cit. on pp. 23, 92).

[4] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. “Representing Nested Inductive Types using W-types”. In: *Automata, Languages and Programming, 31st International Colloqium (ICALP)*. 2004, pp. 59–71 (cit. on pp. 52, 70).

[5] F. Acerbi. “Plato: Parmenides 149a7-c3. A Proof by Complete Induction?” In: *Archive for History of Exact Sciences* 55.1 (2000), pp. 57–76. ISSN: 00039519, 14320657. URL: http://www.jstor.org/stable/41134098 (cit. on p. 30).

[6] Peter Aczel. “The Type Theoretic Interpretation of Constructive Set Theory”. In: *Logic Colloquium ’77*. Ed. by Angus Macintyre, Leszek Pacholski, and Jeff Paris. Vol. 96. Studies in Logic and the Foundations of Mathematics Supplement C. Elsevier, 1978, pp. 55–66. DOI: https://doi.org/10.1016/S0049-237X(08)71989-X. URL: http://www.sciencedirect.com/science/article/pii/S0049237X0871989X (cit. on p. 48).

[7] Jiří Adámek, Stefan Milius, and Lawrence Moss. “Initial algebras and terminal coalgebras: a survey. In: Introduction to category theory, algebras and coalgebra (ESSLLI 2010”. Draft. June 2010. URL: https://www.tu-braunschweig.de/Medien-DB/iti/survey_full.pdf (cit. on pp. 113, 117, 155, 156, 160).

[8] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. “Indexed containers”. In: *Journal Functional Programming* 25 (2015) (cit. on p. 58).

[9] Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. “A categorical semantics for inductive-inductive definitions”. In: *Algebra and Coalgebra in Computer Science*. Ed. by Andrea Corradini and Bartek Klin. Vol. 6859. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 70–84 (cit. on p. 60).

[10] Thorsten Altenkirch Ambrus Kaposi. “A nominal syntax for internal parametricity”. In: *TYPES, Tallinn.* 2015. URL: http://cs.ioc.ee/types15/abstracts-book/contrib8.pdf (cit. on p. 133).

[11] Robert Atkey. “Relational Parametricity for Higher Kinds”. In: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL.* Ed. by Patrick Cégielski and Arnaud Durand. Vol. 16. Leibniz International Proceedings in Informatics (LIPIcs). 2012, pp. 46–61. DOI: 10.4230/LIPIcs.CSL.2012.46 (cit. on pp. 134, 139).

[12] Robert Atkey, Neil Ghani, and Patricia Johann. “A Relationally Parametric Model of Dependent Type Theory”. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2014).* 2014. DOI: 10.1145/2535838.2535852 (cit. on pp. 133, 134, 136, 139, 141).

[13] Steve Awodey. *Category Theory.* 2nd. New York, NY, USA: Oxford University Press, Inc., 2010. ISBN: 0199237182, 9780199237180. URL: http://angg.twu.net/MINICATS/awodey__category_theory.pdf (cit. on pp. 19, 25, 67).

[14] Roland Backhouse, Marcel Bijsterveld, Rik van Geldrop, and Jaap van der Woude. “Categorical fixed point calculus”. In: *Category Theory and Computer Science: 6th International Conference, CTCS '95 Cambridge, United Kingdom, August 7–11, 1995 Proceedings.* Ed. by David Pitt, David E. Rydeheard, and Peter Johnstone. Springer, 1995, pp. 159–179. DOI: 10.1007/3-540-60164-3_25 (cit. on p. 56).

[15] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. “Functorial polymorphism”. In: *Theoretical Computer Science* 70.1 (1990). Special Issue Fourth Workshop on Mathematical Foundations of Programming Semantics, Boulder, CO, May 1988, pp. 35–64. DOI: https://doi.org/10.1016/0304-3975(90)90151-7. URL: http://www.sciencedirect.com/science/article/pii/0304397590901517 (cit. on pp. 131, 134).

[16] Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. “A Presheaf Model of Parametric Type Theory”. In: *Electron. Notes Theor. Comput. Sci.* 319.C (Dec. 2015), pp. 67–82. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2015.12.006. URL: http://dx.doi.org/10.1016/j.entcs.2015.12.006 (cit. on p. 133).

[17] Jean-Philippe Bernardy and Guilhem Moulin. “A Computational Interpretation of Parametricity”. In: *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science.* LICS '12. New Orleans, Louisiana: IEEE Computer Society, 2012, pp. 135–144. ISBN: 978-0-7695-4769-5. DOI: 10.1109/LICS.2012.25. URL: http://dx.doi.org/10.1109/LICS.2012.25 (cit. on p. 133).

[18] E. Bishop and D.S. Bridges. *Constructive analysis.* Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985. ISBN: 9783540150664 (cit. on p. 60).

[19] Francis Borceux. *Handbook of Categorical Algebra.* Vol. 2. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511525865 (cit. on p. 19).

[20] L. E. J. Brouwer. *Brouwer's Cambridge Lectures on Intuitionism.* Cambridge University Press, 1981 (cit. on p. 30).

[21] Aurelio Carboni and Peter Johnstone. “Connected limits, familial representability and Artin glueing”. In: *Mathematical Structures in Computer Science* 5.4 (1995), pp. 441–459. DOI: 10.1017/S0960129500001183 (cit. on pp. 27, 28).

[22] John Cartmell. “Generalised algebraic theories and contextual categories”. In: *Annals of Pure and Applied Logic* 32 (1986), pp. 209–243. ISSN: 0168-0072. DOI: http://dx.doi.org/10.1016/0168-0072(86)90053-9. URL: http://www.sciencedirect.com/science/article/pii/0168007286900539 (cit. on p. 135).

[23] Simon Castellan. http://iso.mor.phis.me/archives/2011-2012/stage-2012-goteburg/report.pdf. 2014 (cit. on p. 135).

[24] James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. “The gentle art of levitation”. In: *ICFP 2010*. 2010, pp. 3–14 (cit. on pp. 12, 108).

[25] Alonzo Church. “The constructive second number class”. In: *Bull. Amer. Math. Soc.* 44.4 (Apr. 1938), pp. 224–232. URL: https://projecteuclid.org:443/euclid.bams/1183500400 (cit. on p. 48).

[26] Jesper Cockx, Dominique Devriese, and Frank Piessens. “Eliminating dependent pattern matching without K”. In: *Journal of Functional Programming* 26 (2016), e16. DOI: 10.1017/S0956796816000174 (cit. on p. 44).

[27] Catarina Coquand. *A realizability interpretation of Martin-Löf’s type theory*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.6695&rep=rep1&type=pdf (cit. on p. 70).

[28] Thierry Coquand. “A new paradox in type theory”. In: *Logic, Methodology and Philosophy of Science IX*. Ed. by Dag Prawitz, Brian Skyrms, and Dag Westerståhl. Vol. 134. Studies in Logic and the Foundations of Mathematics. Elsevier, 1995, pp. 555–570. DOI: https://doi.org/10.1016/S0049-237X(06)80062-5. URL: http://www.sciencedirect.com/science/article/pii/S0049237X06800625 (cit. on pp. 131, 134).

[29] Thierry Coquand. “Pattern Matching with Dependent Types”. In: *Proceedings of the logical frameworks workshop at Båstad*. June 1992. URL: http://www.cse.chalmers.se/~coquand/pattern.ps (cit. on p. 44).

[30] Richard Dedekind. *Was sind und was sollen die Zahlen?* Cambridge Library Collection - Mathematics. Cambridge University Press, 2012. DOI: 10.1017/CBO9781139237314 (cit. on p. 31).

[31] B. Dunphy and U. S. Reddy. “Parametric Limits”. In: *Logic in Computer Science*. 2004, pp. 242–251. URL: http://www.cs.bham.ac.uk/~udr/papers/parametric.pdf (cit. on pp. 131, 134).

[32] Peter Dybjer. “A general formulation of simultaneous inductive-recursive definitions in type theory”. In: *Journal of Symbolic Logic* 65.2 (2000) (cit. on pp. 2, 58).

[33] Peter Dybjer. “Inductive families”. In: *Formal aspects of computing* 6.4 (1994), pp. 440–465 (cit. on pp. 58, 69).

[34] Peter Dybjer. “Internal Type Theory”. In: *Lecture Notes in Computer Science*. Springer, 1996, pp. 120–134 (cit. on pp. x, xii, 130, 135, 136).

[35] Peter Dybjer. “Logical Frameworks”. In: ed. by Gérard Huet and Gordon Plotkin. New York, NY, USA: Cambridge University Press, 1991. Chap. Inductive Sets and Families in Martin-LöF’s Type Theory and Their Set-theoretic Semantics, pp. 280–306. ISBN: 0-521-41300-1. URL: http://dl.acm.org/citation.cfm?id=120477.120487 (cit. on pp. 58, 111).

[36] Peter Dybjer. “Representing inductively defined sets by wellorderings in Martin-Löf’s type theory”. In: *Theoretical Computer Science* 176.1 (1997), pp. 329–335. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(96)00145-4. URL: http://www.sciencedirect.com/science/article/pii/S0304397596001454 (cit. on pp. x, xii, 52, 70).

[37] Peter Dybjer. “Universes and a general notion of simultaneous inductive-recursive definitions in type theory”. In: *Proceedings of the workshop on types for proofs and programs.* 1992 (cit. on p. 130).

[38] Peter Dybjer and Anton Setzer. “A Finite Axiomatization of Inductive-Recursive Definitions”. In: *TLCA*. Springer Verlag, 1999, pp. 129–146 (cit. on pp. ix, xi, 6, 57, 59, 63, 77, 78, 99, 111, 112, 155).

[39] Peter Dybjer and Anton Setzer. “Indexed induction–recursion”. In: *Journal of logic and algebraic programming* 66.1 (2006), pp. 1–49 (cit. on pp. 59, 69, 74, 111, 146, 156).

[40] Peter Dybjer and Anton Setzer. “Induction–recursion and initial algebras”. In: *Annals of Pure and Applied Logic* 124.1-3 (2003), pp. 1–47 (cit. on pp. ix, xi, 5, 7, 9, 45, 57, 59, 61–63, 66, 67, 78, 79, 119).

[41] Samuel Eilenberg and Saunders MacLane. “Group Extensions and Homology”. In: *Annals of Mathematics* 43.4 (1942), pp. 757–831. ISSN: 0003486X. URL: http://www.jstor.org/stable/1968966 (cit. on p. 23).

[42] Thomas Forster. “A Tutorial on Countable Ordinals”. In: *Preprint* (2010). URL: https://www.dpmms.cam.ac.uk/~tf/fundamentalsequence.pdf (cit. on p. 47).

[43] Miriam Franchella. “Brouwer and Nietzsche: Views About Life, Views About Logic”. In: *History and Philosophy of Logic* 36.4 (2015), pp. 367–391 (cit. on p. 31).

[44] Nicola Gambino and Martin Hyland. “Wellfounded trees and dependent polynomial functors”. In: *Types for Proofs and Programs*. 2004, pp. 210–225. URL: http://www1.maths.leeds.ac.uk/%7Epmtng/Publications/gambino-hyland.pdf (cit. on pp. 23, 27, 109).

[45] Nicola Gambino and Joachim Kock. “Polynomial functors and polynomial monads”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 154 (2013), pp. 153–192 (cit. on pp. 8, 23, 52, 55, 76, 109).

[46] R. Garner. “On the strength of dependent products in the type theory of Martin-Löf”. In: *ArXiv e-prints* (Mar. 2008). arXiv: 0803.4466 [math.LO] (cit. on p. 38).

[47] G. Gentzen. “Untersuchungen über das logische Schließen I”. In: *Mathematische Zeitschrift* 39 (1935), pp. 176–210. URL: http://eudml.org/doc/168546 (cit. on p. 32).

[48] Neil Ghani and Peter Hancock. “Containers, monads and induction recursion”. In: *Mathematical Structures in Computer Science* 26.1 (2016), pp. 89–113. DOI: 10.1017/S0960129514000127 (cit. on pp. 59, 75).

[49] Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg. “Positive Inductive-Recursive Definitions”. In: *Algebra and Coalgebra in Computer Science: 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings.* Ed. by Reiko Heckel and Stefan Milius. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 19–33. ISBN: 978-3-642-40206-7. DOI: 10.1007/978-3-642-40206-7_3. URL: https://doi.org/10.1007/978-3-642-40206-7_3 (cit. on pp. 5, 26, 69, 72).

[50] Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg. “Positive Inductive-Recursive Definitions”. In: *Logical Methods in Computer Science* 11.1 (2015). DOI: 10.2168/LMCS-11(1:13)2015 (cit. on p. 59).

[51] Neil Ghani, Conor McBride, Fredrik Nordvall Forsberg, and Stephan Spahn. “Inductive-Recursive Definitions and Composition”. In: *23rd International Conference on Types for Proofs and Programs, TYPES 2017.* 2017 (cit. on p. 12).

[52] Neil Ghani, Conor McBride, Fredrik Nordvall Forsberg, and Stephan Spahn. “Variations on Inductive-Recursive Definitions”. In: *42 International Symposium on Mathematical Foundations of Computer Science (MFCS).* 2017 (cit. on pp. 12, 59).

[53] Leonard Gillman. “Two Classical Surprises Concerning the Axiom of Choice and the Continuum Hypothesis”. In: *The American Mathematical Monthly* 109.6 (2002), pp. 544–553. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2695444 (cit. on p. 111).

[54] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types.* New York, NY, USA: Cambridge University Press, 1989. ISBN: 0-521-37181-3. URL: http://www.paultaylor.eu/stable/prot.pdf (cit. on p. 132).

[55] Healfdene Goguen, Conor McBride, and James McKinna. “Eliminating Dependent Pattern Matching”. In: *Algebra, Meaning, and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday.* Ed. by Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 521–540. DOI: 10.1007/11780274_27. URL: https://doi.org/10.1007/11780274_27 (cit. on p. 44).

[56] I.H. Grant. *Philosophies of Nature after Schelling.* Transversals: New Directions in Philosophy. Bloomsbury Publishing, 2008. ISBN: 9781441147301. URL: https://books.google.co.uk/books?id=lSIdCgAAQBAJ (cit. on p. 31).

[57] H. Grassmann. *Lehrbuch der Arithmetik für höhere Lehranstalten.* Lehrbuch der Matematik für höhere lehranstalten ; 1. Th. Th. Chr. Fr. Enslin, 1861 (cit. on p. 31).

[58] Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. “Small Induction Recursion”. In: *Typed Lambda Calculi and Applications.* Ed. by Masahito Hasegawa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 156–172. ISBN: 978-3-642-38946-7. URL: https://link.springer.com/chapter/10.1007/978-3-642-38946-7_13 (cit. on pp. 3, 28, 59, 65, 69, 84).

[59] R. Hasegawa. “Relational Limits in General Polymorphism”. In: *Publications of the Research Institute for Mathematical Sciences*. Vol. 30. 1994, pp. 535–576. URL: https://pdfs.semanticscholar.org/fc1d/5b4fcd96d9a435c8923bf263ee61cea77bdf.pdf (cit. on pp. 131, 134).

[60] Ryu Hasegawa. “Categorical data types in parametric polymorphism”. In: *Mathematical Structures in Computer Science* 4.1 (1994), pp. 71–109. DOI: 10.1017/S0960129500000372 (cit. on p. 134).

[61] Claudio Hermida and Bart Jacobs. “Structural Induction and Coinduction in a Fibrational Setting”. In: *Information and Computation* 145.2 (1998), pp. 107–152. ISSN: 0890-5401. DOI: http://dx.doi.org/10.1006/inco.1998.2725. URL: http://www.sciencedirect.com/science/article/pii/S0890540198927250 (cit. on p. 133).

[62] Claudio Hermida, Uday S. Reddy, and Edmund P. Robinson. “Logical Relations and Parametricity – A Reynolds Programme for Category Theory and Programming Languages”. In: *Electronic Notes in Theoretical Computer Science* 303 (2014). Proceedings of the Workshop on Algebra, Coalgebra and Topology (WACT 2013), pp. 149–180. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2014.02.008. URL: http://www.sciencedirect.com/science/article/pii/S1571066114000346 (cit. on pp. 131, 134).

[63] Martin Hofmann. “On the interpretation of type theory in locally cartesian closed categories”. In: *Computer Science Logic*. Ed. by Leszek Pacholski and Jerzy Tiuryn. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 427–441. ISBN: 978-3-540-49404-1. URL: https://link.springer.com/chapter/10.1007/BFb0022273 (cit. on p. 136).

[64] Martin Hofmann. “Syntax and semantics of dependent types”. In: *Extensional Constructs in Intensional Type Theory*. London: Springer London, 1997, pp. 13–54. ISBN: 978-1-4471-0963-1. DOI: 10.1007/978-1-4471-0963-1_2. URL: https://doi.org/10.1007/978-1-4471-0963-1_2 (cit. on pp. 42, 136, 139, 140).

[65] “Introduction to category theory, algebras and coalgebra (ESSLLI 2010)”. In: June 2010. URL: https://www.tu-braunschweig.de/Medien-DB/iti/survey_full.pdf (cit. on pp. 20, 22, 24).

[66] Bart Jacobs. *Categorical Logic and Type Theory*. Vol. 141. Studies in Logic and the Foundations of Mathematics. North Holland, 1999 (cit. on pp. 27, 139).

[67] Bart Jacobs. “Comprehension categories and the semantics of type dependency”. In: *Theoretical Computer Science* 107.2 (1993), pp. 169–207. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/0304-3975(93)90169-T. URL: http://www.sciencedirect.com/science/article/pii/030439759390169T (cit. on p. 161).

[68] T. Jech. *Set Theory: The Third Millennium Edition, revised and expanded*. Springer Monographs in Mathematics. Springer Berlin Heidelberg, 2006. ISBN: 9783540440857. URL: https://books.google.co.uk/books?id=WTAl997XDb4C (cit. on pp. 15–19, 47, 111).

[69] Peter T Johnstone. *Sketches of an elephant: a Topos theory compendium.* Oxford logic guides. New York, NY: Oxford Univ. Press, 2002. URL: https://cds.cern.ch/record/592033 (cit. on pp. 19, 28, 139).

[70] Reinhard Kahle and Anton Setzer. "An Extended Predicative Definition of the Mahlo Universe". In: *Ways of Proof Theory.* De Gruyter. DOI: 10.1515/9783110324907.315. URL: https://doi.org/10.1515%2F9783110324907.315 (cit. on p. 62).

[71] A. Kanamori. *The Higher Infinite: Large Cardinals in Set Theory from Their Beginnings.* Springer Monographs in Mathematics. Springer Berlin Heidelberg, 2008. ISBN: 9783540888666. URL: https://books.google.co.uk/books?id=FH%5C_n84ocuSMC (cit. on p. 15).

[72] Immanuel Kant. *Opus Postumum.* Cambridge University Press, 1993 (cit. on p. 31).

[73] G. M. Kelly. "Basic concepts of enriched category theory". In: *Repr. Theory Appl. Categ.* 10 (2005). Reprint of the 1982 original [Cambridge Univ. Press, Cambridge; MR0651714], pp. vi+137 (cit. on p. 19).

[74] S. C. Kleene. "On notation for ordinal numbers". In: *Journal of Symbolic Logic* 3.4 (1938), pp. 150–155. DOI: 10.2307/2267778 (cit. on p. 48).

[75] A. Kock and J. Kock. "Local fibered right adjoints are polynomial". In: *ArXiv e-prints* (May 2010). arXiv: 1005.4236 [math.CT] (cit. on p. 23).

[76] J. Kock. "Polynomial functors and trees". In: *ArXiv e-prints* (July 2008). arXiv: 0807.2874 [math.CT] (cit. on p. 23).

[77] J. Kock, A. Joyal, M. Batanin, and J.-F. Mascari. "Polynomial functors and opetopes". In: *ArXiv e-prints* (June 2007). arXiv: 0706.1033 [math.QA] (cit. on p. 23).

[78] Joachim Kock. *Notes on Poloynomial functors, (draft available from the author's website).* 2009. URL: http://mat.uab.es/~kock/cat/polynomial.pdf (cit. on pp. 9, 23).

[79] Peter Koellner. "On reflection principles". In: *Annals of Pure and Applied Logic* 157.2 (2009). Kurt Gödel Centenary Research Prize Fellowships, pp. 206–219. ISSN: 0168-0072. DOI: https://doi.org/10.1016/j.apal.2008.09.007. URL: http://www.sciencedirect.com/science/article/pii/S0168007208001309 (cit. on pp. 18, 43, 79).

[80] Joachim Lambek. "A fixpoint theorem for complete categories". In: *Mathematische Zeitschrift* 103.2 (Apr. 1968), pp. 151–161. ISSN: 1432-1823. DOI: 10.1007/BF01110627. URL: https://doi.org/10.1007/BF01110627 (cit. on p. 21).

[81] S.M. Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1998. ISBN: 9780387984032. URL: https://books.google.co.uk/books?id=eBvhyc4z8HQC (cit. on p. 25).

[82] Hilbert Levitz. "Transfinite Ordinals and Their Notations: For The Uninitiated". Jan. 2006. URL: http://www.cs.fsu.edu/%5C~%7B%7Dlevitz/ords.ps (cit. on p. 124).

[83] A. Levy. *Basic Set Theory.* Basic set theory Bd. 13. Dover Publications, 2002 (cit. on pp. 18, 19).

[84] Zhaohui Luo. *Notes on universes in type theory.* 2012. URL: http://www.cs.rhul.ac.uk/~zhaohui/universes.pdf (cit. on pp. 41, 132).

[85] Maria Emilia Maietti. "The internal type theory of a Heyting pretopos". In: *Types for Proofs and Programs.* Ed. by Eduardo Giménez and Christine Paulin-Mohring. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 216–235. ISBN: 978-3-540-49562-8. URL: https://link.springer.com/chapter/10.1007%2FBFb0097794#citeas (cit. on p. 139).

[86] E.G. Manes. *Algebraic Theories.* Graduate Texts in Mathematics. Springer New York, 2012. URL: https://www.springer.com/gb/book/9781461298625 (cit. on p. 25).

[87] F. Marmolejo and R. J. Wood. "Monads as extension systems — no iteration necessary". In: *Theory and applications of categories* 24.4 (2010), pp. 84–113. URL: http://www.tac.mta.ca/tac/volumes/24/4/24-04abs.html (cit. on pp. 25, 100).

[88] Per Martin-Löf. *An intuitionistic theory of types.* 1972. URL: http://archive-pml.github.io/martin-lof/pdfs/An-Intuitionistic-Theory-of-Types-1972.pdf (cit. on pp. 1, 2, 32, 33, 40, 42, 46, 58, 60, 70).

[89] Per Martin-Löf. "An intuitionistic theory of types: predicative part". In: *Logic Colloquium '73, Proceedings of the Logic Colloquium.* Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. North-Holland, 1975, pp. 73–118. URL: http://archive-pml.github.io/martin-lof/pdfs/An-Intuitionistic-Theory-of-Types-Predicative-Part-1975.pdf (cit. on pp. 42, 46).

[90] Per Martin-Löf. "Constructive Mathematics and Computer Programming". In: *Logic, Methodology and Philosophy of Science VI.* Ed. by L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski. Vol. 104. Studies in Logic and the Foundations of Mathematics. Elsevier, 1982, pp. 153–175. DOI: https://doi.org/10.1016/S0049-237X(09)70189-2. URL: http://www.sciencedirect.com/science/article/pii/S0049237X09701892 (cit. on p. 42).

[91] Per Martin-Löf. "Hauptsatz for the intuitionistic theory of iterated inductive definitions". In: (1971). URL: http://archive-pml.github.io/martin-lof/pdfs/Hauptsatz-for-the-intuitionistic-theory-of-iterated-inductive-definitions-1971.pdf (cit. on p. 2).

[92] Per Martin-Löf. *Intuitionistic type theory. Notes by Giovanni Sambin.* Vol. 1. Studies in Proof Theory. Bibliopolis, 1984, pp. iv+91. ISBN: 88-7088-105-9. URL: http://archive-pml.github.io/martin-lof/pdfs/Bibliopolis-Book-retypeset-1984.pdf (cit. on pp. 31–33, 41, 42).

[93] Per Martin-Löf. "On the Meanings of the Logical Constants and the Justifications of the Logical Laws". In: *Nordic Journal of Philosophical Logic* 1.1 (1996), pp. 11–60. URL: http://archive-pml.github.io/martin-lof/pdfs/Meanings-of-the-Logical-Constants-1983.pdf (cit. on p. 32).

[94] Nax Paul Mendler. “Predicative Type Universes and Primitive Recursion”. In: *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991.* 1991, pp. 173–184. DOI: 10.1109/LICS.1991.151642. URL: https://doi.org/10.1109/LICS.1991.151642 (cit. on p. 58).

[95] Ieke Moerdijk and Erik Palmgren. “Wellfounded trees in categories”. In: *Annals of Pure and Applied Logic* 104.1 (2000), pp. 189–218. ISSN: 0168-0072. DOI: http://dx.doi.org/10.1016/S0168-0072(00)00012-9. URL: http://www.sciencedirect.com/science/article/pii/S0168007200000129 (cit. on p. 23).

[96] Joan Moschovakis. “Intuitionistic Logic”. In: *The Stanford Encyclopedia of Philosophy.* Ed. by Edward N. Zalta. Spring 2015. Metaphysics Research Lab, Stanford University, 2015. URL: https://plato.stanford.edu/archives/spr2015/entries/logic-intuitionistic/ (cit. on p. 32).

[97] Derek Dreyer Neelakantan R. Krishnaswami. “Internalizing Relational Parametricity in the Extensional Calculus of Constructions”. In: *Computer Science Logic (CSL).* Sept. 2013. URL: https://www.cl.cam.ac.uk/~nk480/final-csl-internalizing-parametricity.pdf (cit. on p. 134).

[98] Fredrik Nordvall Forsberg. “Inductive-inductive definitions”. PhD thesis. Swansea University, 2013. URL: https://personal.cis.strath.ac.uk/fredrik.nordvall-forsberg/thesis/thesis.pdf (cit. on pp. 33, 60).

[99] Fredrik Nordvall Forsberg and Anton Setzer. “Inductive-Inductive Definitions”. In: *Computer Science Logic.* Ed. by Anuj Dawar and Helmut Veith. Vol. 6247. Lecture Notes in Computer Science. doi: 10.1007/978-3-642-15205-4_35. Springer Berlin / Heidelberg, 2010, pp. 454–468. URL: http:/#/dx.doi.org/10.1007/978-3-642-15205-4%5C_35 (cit. on p. 60).

[100] Erik Palmgren. “A construction of type: type in Martin-Löf's partial type theory with one universe”. In: *Journal of Symbolic Logic* 56.3 (1991), pp. 1012–1015. DOI: 10.2307/2275068 (cit. on p. 40).

[101] Erik Palmgren. “On universes in type theory”. In: *Twenty five years of constructive type theory.* Ed. by Giovanni Sambin and Jan Smith. Oxford University Press, 1998, pp. 191–204. URL: http://www2.math.uu.se/~palmgren/universe.pdf (cit. on p. 42).

[102] G. Peano. *Arithmetices principia: nova methodo.* Fratres Bocca, 1889 (cit. on p. 31).

[103] Henri Poincaré. “Les Mathematiques et la Logique.” In: *Revue de Métaphysique et de Morale* 14 (), pp. 294–317 (cit. on p. 60).

[104] Robert Pollack. “Dependently Typed Records in Type Theory”. In: *Formal Aspects of Computing* 13.3 (July 2002), pp. 386–402. ISSN: 1433-299X. DOI: 10.1007/s001650200018. URL: https://doi.org/10.1007/s001650200018 (cit. on pp. 92, 93).

[105] D. Prawitz. “Natural deduction: a proof-theoretical study”. PhD thesis. Almqvist & Wiksell, 1965. URL: https://www.bibsonomy.org/bibtex/2e22cbf14263e2466815829957ac19b
arademaker (cit. on p. 32).

[106] John C. Reynolds. "Types, Abstraction and Parametric Polymorphism." In: *IFIP Congress.* Jan. 3, 2002, pp. 513–523. URL: http://dblp.uni-trier.de/db/conf/ifip/ifip83.html#Reynolds83 (cit. on pp. 131–133).

[107] E. Robinson and G. Rosolini. "Reflexive Graphs and Parametric Polymorphism". In: *Logic in Computer Science.* 1994, pp. 364–371. URL: https://www.computer.org/csdl/proceedings/lics/1994/6310/00/00316053.pdf (cit. on p. 134).

[108] B. Russell. "On Some Difficulties in the Theory of Transfinite Numbers and Order Types". In: *Proceedings of the London Mathematical Society* s2-4.1 (1907), pp. 29–53. ISSN: 1460-244X. DOI: 10.1112/plms/s2-4.1.29. URL: http://dx.doi.org/10.1112/plms/s2-4.1.29 (cit. on p. 40).

[109] Bertrand Russell. "Mathematical Logic as Based on the Theory of Types". In: *American Journal of Mathematics* 30.3 (1908), pp. 222–262 (cit. on p. 60).

[110] U. Schreiber. "Differential cohomology in a cohesive infinity-topos". In: *ArXiv e-prints* (Oct. 2013). arXiv: 1310.7930 [math-ph] (cit. on p. 28).

[111] Peter Schroeder-Heister. "A natural extension of natural deduction". In: *Journal of Symbolic Logic* 49.4 (1984), pp. 1284–1300. DOI: 10.2307/2274279 (cit. on p. 39).

[112] A Setzer, N Ghani, L Malatesta, F N Forsberg, and A Setzer. "Fibred Data Types". In: *Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium.* 2013, pp. 243–252. DOI: 10.1109/LICS.2013.30 (cit. on p. 59).

[113] Anton Setzer. "Extending Martin-Löf Type Theory by one Mahlo-Universe". In: *Arch. Math. Log.* 39 (2000), pp. 155–181. URL: http://dx.doi.org/10.1007/s001530050140 (cit. on p. 59).

[114] Anton Setzer. "Proof theory and Martin-Löf Type Theory". In: *One Hundred Years of Intuitionism (1907–2007): The Cerisy Conference.* Ed. by Mark van Atten, Pascal Boldini, Michel Bourdeau, and Gerhard Heinzmann. Basel: Birkhäuser Basel, 2008, pp. 257–279. ISBN: 978-3-7643-8653-5. DOI: 10.1007/978-3-7643-8653-5_16. URL: https://doi.org/10.1007/978-3-7643-8653-5_16 (cit. on p. 59).

[115] Anton Setzer. "Proof theory and Martin-Löf Type Theory". In: *One Hundred Years of Intuitionism (1907 – 2007).* Ed. by M. v. Atten, P. Boldini, M. Bourdeau, and G. Heinzmann. Birkhäuser, 2008, pp. 257–279. ISBN: 978-3764386528. DOI: 10.1007/978-3-7643-8653-5_16. URL: http://dx.doi.org/10.1007/978-3-7643-8653-5_16 (cit. on p. 59).

[116] Stephan Spahn. *A relationally-parametric model of Martin-Löf Type Theory with induction-recursion, Categorical Logic Workshop, Stockholm University.* http://staff.math.su.se/p.l.lumsdaine/catlogworkshop/abstracts.html. December 3-4 2015 (cit. on p. 13).

[117] Thomas Streicher. *Fibered categories á la Jean Bénabou.* 1999. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/FIBR/FibLec18.pdf (cit. on pp. 27, 28).

[118] Thomas Streicher. *Investigations into Intensional Type Theory, Habilitation Thesis.* 1993. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/HabilStreicher.pdf (cit. on p. 44).

[119] Thomas Streicher. “Universes in Toposes”. In: *From Sets and Types to Topology and Analysis: Towards practicable foundations for constructive mathematics.* Oxford University Press, 2004. URL: `http://www.oxfordscholarship.com/view/10.1093/acprof:oso/9780198566519.001.0001/acprof-9780198566519` (cit. on pp. 137, 141).

[120] Izumi Takeuti. “The Theory of Parametricity in Lambda Cube”. In: (Jan. 2001). URL: `http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1217-10.pdf` (cit. on pp. 132, 134, 161).

[121] A.S. Troelstra. *Principles of Intuitionism.* Lecture Notes in Mathematics no. 95. Springer, 1969. URL: `https://books.google.co.uk/books?id=39-VAQAACAAJ` (cit. on p. 31).

[122] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics.* Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2014. ISBN: 9780080955100. URL: `https://www.elsevier.com/books/constructivism-in-mathematics-vol-1/troelstra/978-0-444-70266-1` (cit. on p. 39).

[123] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* Institute for Advanced Study: `https://homotopytypetheory.org/book`, 2013 (cit. on pp. 39, 45).

[124] B. van den Berg. “Predicative toposes”. In: *ArXiv e-prints* (July 2012). arXiv: `1207.0959 [math.CT]` (cit. on p. 62).

[125] B. van den Berg and I. Moerdijk. “W-types in Homotopy Type Theory”. In: *ArXiv e-prints* (July 2013). arXiv: `1307.2765 [math.CT]` (cit. on p. 23).

[126] Philip Wadler. “Theorems for Free!” In: *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture.* FPCA ’89. Imperial College, London, United Kingdom: ACM, 1989, pp. 347–359. ISBN: 0-89791-328-0. DOI: `10.1145/99370.99404`. URL: `http://doi.acm.org/10.1145/99370.99404` (cit. on pp. 131, 134).

[127] M. Weber. “Polynomials in categories with pullbacks”. In: *ArXiv e-prints* (June 2011). arXiv: `1106.1983 [math.CT]` (cit. on p. 23).

[128] Mark Weber. “Familial 2-functors and parametric right adjoints”. In: *Theory and Applications of Categories* 18.22 (2007), pp. 665–732. ISSN: 1201-561X. URL: `http://www.tac.mta.ca/tac/volumes/18/22/18-22abs.html` (cit. on pp. 27, 70).

Index