Investigations into Inductive-Recursive Definitions

Lorenzo Malatesta

Doctor of Philosophy

University of Strathclyde
Department of Information and Computer Science

2015

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Abstract

The theory of recursive functions where the domain of a function is inductively defined at the same time as the function is called induction-recursion. This theory has been introduced in Martin-Löf type theory by Dybjer [37] and further explored in a series of papers by Dybjer and Setzer [38, 39, 40]. Important data types like universes closed under dependent type operators are instances of this theory.

In this thesis we study the class of data types arising from inductive-recursive definitions, taking the seminal work of Dybjer and Setzer as our starting point. We show how the theories of inductive and indexed inductive types arise as sub-theories of induction-recursion, by revealing the role played by a notion of of size within the theory of induction-recursion. We then expand the expressive power of induction-recursion, showing how to extend the theory of induction-recursion in two different ways: in one direction we investigate the changes needed to obtain a more flexible semantics which gives rise to a more comprehensive elimination principle for inductive-recursive types. In another direction we generalize the theory of induction-recursion to a fibrational setting. In both extensions we provide a finite axiomatization of the theories introduced, we show applications and examples of these theories not previously covered by induction-recursion, and we justify the existence of data types built within these theories.

Contents

Chapter 1

Introduction

1.1 Dependent types for proofs and programs

Both in computer science and in constructive mathematics dependency is a ubiquitous and central notion. For example, whenever we refer to a family of sets indexed by some other set we are exploiting the concept of a type dependent on values of some other type. The possibility to explicitly refer to dependent types considerably broadens the expressive power of a formal language. The paradigm of such a dependent system is Martin-Löf type theory [74, 73, 80]. Two other ingredients, beside the notion of dependency, make dependent type theories expressive and versatile as formal languages: the first is the possibility to use these theories both as programming languages and as formal systems to reason about constructive mathematics. And, indeed, dependent type systems have proved themselves extremely successful in fulfilling these two purposes: proof assistants based on dependent type theories are nowadays used both to certify programs [26] and to formalize mathematics [56]. This feature of dependent systems comes directly from the Brouwer-Heyting-Kolmogorov explanation of the logical operations and it is validated by the Curry-Howard isomorphism: we can regard an inhabitant a of a type A either as a proof of a *proposition* A, or as a program satisfying a *specification* A.

The other ingredient making dependent type theories so expressive and therefore ideal languages to develop both constructive mathematics and writing

correct programs is given by inductive types. We focus on them in the next section.

1.2 A theory of inductive definitions

We could try to introduce inductive types by looking at their historical development but, as noted by Coquand and Dybjer [28], it is difficult to trace back the origin of this notion since it permeates the history of proof theory, a large part of computer science and, to a large extent, all constructive mathematics. So we will rather try to introduce them informally and by means of examples.
An inductive definition of a type consists in giving some rules to generate elements of the type and then specifying that terms in the type have been generated according to these rules only. The inductive type is then defined as the least type closed under the rules; put otherwise, this means that the type has been freely generated by the rules.
In logic and computer science the use of inductive definitions is pervasive. Consider the following simple examples in logic:

- the syntax of a formal language: we define the terms of a language as the smallest set of expressions containing variables and constants and closed under the term formation rules.

- The theorems of a formal deductive system. Given a formal system H, let $\mathsf{Th}(\mathsf{H})$ be the collection of its theorems. Then $\mathsf{Th}(\mathsf{H})$ is the smallest set of well-formed formulas closed under the inference rules.

In computer science inductive types are the bricks of a dependently typed programming language: they represent the building blocks on which any other type is built[1]. Inductive definitions are therefore fundamental to dependent type theories which are both logical systems and frameworks for programming. And, indeed, Martin-Löf type theory can be seen as an intuitionistic

[1]Pushing on this metaphor we could say that recursion is the mortar the dependently typed programmer has at her disposal for computing.

theory of iterated inductive definitions developed in a framework of dependent types: each type is defined inductively by its introduction rule which prescribes how canonical terms of the type are generated, while the elimination and computational rules, derived from the introduction rule, express a recursion principle specifying its action on canonical elements.
In classical set theory we can represent an inductive type as the set obtained as the intersection of all collections satisfying the condition of closure under the given rules. But this informal description of an inductive type *from above* does not do justice to the intuition that inductive types are essentially built *from below* as the union of stages in a process. Indeed the former is a rather impredicative description: it defines an inductive set quantifying over the collection of all sets satisfying the closure condition. Nevertheless, inductive definitions can be understood directly in their own terms, and they are first-class citizens in generalized predicative systems such as Martin-Löf type theory.
The simplest inductive definition is that of the set of natural numbers. From a constructivist perspective we define the natural numbers, also known from Cantor as the first number class, as an inductive type as follows:

Example 1.1 (First number class)**.** The set of natural numbers N is built according to the following rules:

- $\mathsf{0}$ is a natural number,
- if n is a natural number then so is $\mathsf{s}(n)$. ■

Once we have defined $\Omega_0 =_{\mathsf{def}} \mathsf{N}$, we can go further and inductively define also the second number class Ω_1, i.e. the set of countably infinite ordinals, also known as Brouwer ordinals. The rules are the following:

Example 1.2 (Second number class)**.** The type of Brouwer ordinals Ω is built according to the following rules:

- $\mathsf{0}$ is a Brouwer ordinal,
- if α is a Brouwer ordinal then so is $\mathsf{s}(\alpha)$,
- if $f : \mathsf{Nat} \to \Omega$ is a countable sequence of Brouwer ordinals then so is $\mathsf{lim}(f)$. ■

We could be tempted to indulge this Cantorian initiative. And, indeed, we can iterate the process just described: if we have build the n-th number class Ω_n we get the $(n+1)$-th number class Ω_{n+1} by adding on top of the rules for Ω_n the following rule

- if $f : \Omega_n \to \Omega_{n+1}$ is an Ω_n sequence of elements of the $(n+1)$-th number class Ω_{n+1}, then so is $\mathsf{lim_{n+1}}\,(f)$.

But a pattern seems now to emerge. We can use the the set of natural numbers N, to index number classes. This way we could define a whole collection of inductive types representing number classes in one go. We show in the next section how to do it.

1.2.1 Indexed inductive types

Indexed inductive definitions exploit two distinctive features of Martin-Löf type theory already mentioned: type dependency and inductive types. Indexed inductive types represent the next level of sophistication of an inductive type where we allow for the simultaneous definition of a family of inductive types indexed over some other type. This extra feature allows to store relevant computational information in the indices of an inductive type. Let us illustrate by means of some examples how indexed inductive definitions can be used to build more sophisticated data types.

Example 1.3 (Finite sets)**.** The type of finite sets is a data type indexed over N. For every $n : \mathsf{N}$, the set $\mathsf{fin}(n)$ represents the set of natural numbers below n. It is built according to the following rules:

- for every natural number n, zero is an element in $\mathsf{fin}(\mathsf{s}(n))$.
- If we have an element x of $\mathsf{fin}(n)$ then $\mathsf{succ}(x)$ is an element of $\mathsf{fin}(\mathsf{s}(n))$.

■

When $\mathsf{fin}(n)$ has been defined for an arbitrary natural number n, we can use it for example as a set of variables for the untyped λ- calculus. This way we can build the set of well-scoped untyped λ-term as follows:

Example 1.4 (untyped λ-terms)**.** The type of untyped λ-terms is a N-indexed inductive type, constructed according to the following rules:

- variables, represented as elements of $\mathsf{fin}(n)$, are λ-terms;
- if we have two λ-terms t, u with at most n free variables then we can use function application, $\mathsf{App}\,(t, u)$, to build a λ-term with at most n free variables;
- if we have a λ-term with at most $n + 1$ free variables we can use λ-abstraction to build a term with at most n free variables. ■

Note that λ-abstraction uses lambda terms with $n+1$ free variables to build terms with n free variable so that the entire family needs to be built simultaneously. We finally come back to the N-indexed type of number classes mentioned in the previous section.

Example 1.5 (A family of number classes)**.** The type of number classes is a type indexed over N. For every $n : \mathsf{N}$, the type $\Omega(n)$ represents the n-th number class. It is built according to the following rules:

- $\mathsf{0}$ is an element of $\Omega(n)$;
- if we have an element x in $\Omega(\mathsf{s}(n))$ then $\mathsf{succ}(x)$ is an element of $\Omega(\mathsf{s}(n))$;
- if we have a sequence $f : \Omega(n) \to \Omega(\mathsf{s}(n))$, then $\mathsf{lim}_{n+1}(f)$ is an element of $(\Omega(\mathsf{s}(n)))$. ■

At this point we already engaged in a Cantorian initiative and we now want to pursue this construction. We have used natural numbers to build a N-indexed family of number classes. But we could refine our construction by using any already given number class as index of our families to get larger families of number classes, and carry on this way. But, this would not be really satisfactory: maybe we do not want to fix in advance an index of this family. Can we build a family whose indices are the number classes we are currently building up? If this is possible then we would have to build the index of the family at the same time of the family we are currently building. In the next section we explain how this is indeed possible using inductive-recursive definitions (see also Section 3.2.7 for more details on the above example).

1.2.2 Inductive-recursive definitions

When defining an indexed inductive type, indexed over A, we think of A as an index for a family of inductive types $B(a)$ where a is an element of A. Another approach is to regard a family as a (large valued) recursive function $B : A \to \mathsf{Set}$. That is, we exploit the inductive definition of A to recursively define B. But A might not be already given in advance as hinted at the end of the previous section. That is, the inductive definition of A might depend on the recursively defined function B. The theory of such functions $B : A \to \mathsf{Set}$, and more generally of functions with codomain any (possibly large) type D, where as we build up the function B recursively, we also build up the type A inductively is called induction-recursion. Data types built using these functions are called inductive-recursive types. This thesis is about this class of types.

The theory of induction-recursion has been introduced by Dybjer [36] and then further refined by Dybjer and Setzer in a series of papers [38, 39, 40]. Important data types like universes closed under dependent type operators are instances of induction-recursion and the theory is so expressive that almost every known type is an instance of it (an exception is given by the internal Mahlo universe constructed by Setzer [88]).

Data types as presented in this thesis line up in an increasing hierarchy of complexity: at the bottom lie simple inductive types like the natural numbers, lists and trees. Then, with indexed inductive types we can build data types indexed by extra information that can be used to express properties of data having those types. Examples are given by vectors, untyped lambda terms, general tree types [80]. Finally, inductive-recursive types represent the most advanced form of data types where we allow for the definition of indexed data type where the indices are generated at the same time as the data.

1.2.3 Syntax and semantics of inductive types

So far we have seen *examples* of (indexed) inductive types. We may be tempted to add rules for each specific inductive type we can come up with to our dependent type theory. This is not an economic approach and, arguably,

nor a satisfactory one. By contrast, different approaches, in form of schemes, have been proposed to capture the notion of an (indexed) inductive type in a dependent type system. A precursor of these schemes can be traced back to Martin-Löf [72] who formulated it in first order logic. Dybjer [35] gave a scheme for indexed inductive types in Martin-Löf type theory while Coquand and Paulin-Mohring [29] gave one for the Calculus of Constructions. More recently Benke et al. [17] gave a scheme for indexed inductive types inspired by those given by Dybjer and Setzer [38, 39, 40] which are the first containing schemes covering inductive-recursive types.
A more semantical approach is given by indexed containers (Section 2.2.3) and dependent polynomial functors (Section 2.2.3) and it is inspired by the language of category theory. In this thesis we deliberately take this approach by using initial algebra semantics (see Section 2.2.1) which nowadays has become one of the cornerstones of the theory of modern functional programming languages. This approach provides a well-developed theory of data types which is modular, expressive and usable at different levels: it is used as a framework both by researchers in the mathematical foundations of programming language semantics, by programming language designers and even by programmers themselves (under the guise of the *Algebra of Programming School* [18]). In this thesis, a theory of data type will always be given by a syntax (which we sometimes refer to as a grammar) and a corresponding functorial semantics which assigns to each syntactic object a functor (which we also refer to as its extension).

1.3 Type, sets and categories

Martin Löf type theory has proved to be a robust foundational framework. In our opinion this claim is validated not so much by the evidence that a large amount of mathematics can be developed within it (reductionism), but rather by the deep connections that this theory has interweaved with other branches of mathematics and in particular with other foundational theories (structuralism). Using a metaphor, we can think of Martin-Löf type theory as the base of a prism of glass. The light coming from outside can hit a type and refract it as a set, as a proposition, as an object of a category, or as

a specification for a program. Though simple, this metaphor is imprecise. Trying to make precise the connection between type theory, set theory, and category theory has been a major theme of recent research. The development of homotopy type theory [92] has redesigned this pyramid by inserting a new face to this solid represented by homotopy theory which allows us to refract a type as a space.

Type theory will be the basis of this thesis. We will mainly exploit its relationship with category theory by using type theoretic reasoning to talk about categories, and vice versa. We will work in extensional type theory since this will greatly simplify this relationship. However, it is important to notice that the theory of induction-recursion makes perfect sense in intensional type theory as well, and alternatively, we could have used setoids and the category of setoids to overcome the difficulties arising from doing category theory within intensional type theory [60, 63, 78].

To build models of type theory augmented with schemata for inductive-recursive definitions we turn our pyramid and let it rest on classical set theory. To emphasize this we use different fonts for the categories of sets: (i) Set for the category of constructive sets (or small types) built from the Set-judgments of the type theory we present in Section 2.1; (ii) $\mathbf{Set}$ for the category of (not-constructive) sets, built from ZFC possibly augmented with strong axiom of infinity (see Section 3.1.2 and 3.3).

All of the consistency proofs presented in this thesis will use the categories of sets built from the classical theory of sets ZFC. Nonetheless, since this thesis is developed in a constructive settings, we are naturally led to work with the category of constructive sets Set whenever possible. To minimize confusion we always explicitly point the reader's attention to which categories are in use.

We refer the reader to [39], Section 2.4 and in [40] Section 4 for details of the construction of the syntactic category Set and of other categories used to model induction-recursion. We do not pursue investigation into categorical properties of the category Set since this would lead us out of the scope of this thesis. For our purposes it will suffice to assume that Set is a locally cartesian closed category with W-types as ensured here by the rules for Set-judgments (2.2) - (2.18) in Section 2.1 and Section 2.2. Notice, that, with abuse of

notation, we use the same font both for the category of (constructive) sets and for the type of small types.

1.4 Overview

The rest of this thesis is structured as follows:

- Chapter 2 introduces Martin-Löf type theory formulated in a Logical Framework. We focus on the theories of inductive and indexed inductive types in Section 2.2, also covering standard material on initial algebra semantics.

- Chapter 3 introduces the theory of induction-recursion as originally formulated by Dybjer and Setzer. We begin with an introduction to universes as a paradigmatic example of an inductive-recursive type in Section 3.1. We develop a syntax and a corresponding semantics for the theory of induction-recursion in Section 3.2. We recast the set-theoretical model built by Dyber and Setzer to prove the consistency of the theory in Section 3.3.

- Chapter 4 is based on joint work with T.Altenkirch, N. Ghani, P. Hancock and C. McBride [57]. In this chapter we introduce a sub-theory of induction-recursion that we dub "small induction-recursion". We build a category of small inductive-recursive definitions and prove full and faithfulness of the interpretation functor. The main result in the chapter is an equivalence between the category of dependent polynomial functors and the category of small inductive-recursive functors.

- Chapter 5 is based on joint work with Fredrik Nordvall Forsberg and Neil Ghani [53]. In this chapter we introduce a variant of induction-recursion, which we call "positive induction-recursion", motivated by an extension of the original semantics of induction-recursion. We introduce the syntax and the semantics of positive induction-recursion in Section 5.2. We gave examples of applications of our theory in Sections 5.3 and 5.4. We justify the existence of positive inductive-recursive types in Section 5.6.

- Chapter 6 is based on joint work with Neil Ghani, Fredrik Nordvall Forsberg and Anton Setzer [54]. In this chapter we abstract the theory of induction-recursion to a fibrational setting. We obtain a theory of generalized inductive-recursive types which we call "fibered data types". We give the syntax and the semantics for this theory in Section 6.3 and 6.4 and we give examples of fibered data types in Section 6.5. We justify their existence in Section 6.6.

Contribution

Parts of the the thesis has been published in peer reviewed conferences. My contribution to the publications are:

[57] Working on initial ideas by Thorsten Altenkirch I developed the mathematical content of the paper and wrote a first draft. The final version of the paper was edited by Peter Hancock and Conor McBride.

[53] Working on an initial idea of Neil Ghani I developed the mathematical content of the paper jointly with Fredrik Nordvall Forsberg. The final version was edited by Fredrik Nordvall Forsberg and me.

[54] The paper was entirely written by Fredrik Nordvall Forsberg and me after an initial idea of Neil Ghani.

Chapter 2

Background

Abstract In this chapter we lay down the basis on which the rest of the thesis is built by recalling some standard material. We start by introducing the theory of types which will serve as a framework for our investigations (Section 2.1). We then analyze the notion of an (indexed) inductive definition (Section 2.2) focusing on the categorical approach given by initial algebra semantics.

2.1 Type theory in a Logical Framework

In this Section we introduce the type theory we will use as a basis for the rest of the thesis. We adopt a presentation of Martin-Löf type theory within a Logical Framework as in [80] part III. This formulation will turn out to be essential in Chapter 3, where we extend the type theory presented here with rules for inductive-recursive definitions. We use the extensional version of Martin-Löf type theory, rather than the intensional one (see Rule 2.13 and Rule 2.14 and the discussion on page 19 for more on this).

The core of Martin-Löf type theory is a λ-calculus with dependent types. The first versions [74, 73] were formulated as free-standing theories of types. The Logical Framework was introduced later, suggesting the perspective of conceiving type theory as a meta-theory where other theories can be interpreted, allowing for a modular specification of different theories within it.

From our perspective the main advantage of the introduction of the Logical

Framework is the possibility of having a theory of sets internal to a theory of types. Notice that we do not follow the presentation of the theory of sets as presented in [80] part III. There, sets are introduced by defining constants of different types and asserting equalities between elements in the sets. Here, instead, we introduce sets in the more readable polymorphic style (see Section 2 in [48] for more on the polymorphic vs monomorphic presentation of Martin-Löf type theory). We hope that, even if concise, our presentation will help the reader not familiar with type theory to get a better understanding of the theory.

Martin-Löf type theory is a formal system about types and terms. Its primary purpose is to allow one to derive judgments about these entities. There are four basic forms of judgments:

$$\begin{array}{llll} (1) & A : \mathsf{type} & (2) & a : A \\ (3) & A = B : \mathsf{type} & (4) & a = b : A \end{array}$$

whose intended reading is the following: (1) A is a type, (2) a is a term of type A, (3) A and B are equal types (4) a and b are equal terms of type A. The theory allows us to derive hypothetical judgments, i.e. judgments depending on some assumptions. A collections of assumptions is called a context Γ and consists of a list of typing judgments:

$$x_1 : A_1, x_2 : A_2(x_1), \ldots, x_n : A_n(x_1, x_2 \ldots x_{n-1})$$

A well-formed context is one for which for every $i < n$ the judgment $A_i : \mathsf{type}$ holds in the preceding context $x_1 : A_1, \ldots, x_{i-1} : A_{i-1}$.

We write hypothetical judgments depending on a context Γ as follows:

$$\begin{array}{ll} \Gamma \vdash A : \mathsf{type} & \Gamma \vdash a : A \\ \Gamma \vdash A = B : \mathsf{type} & \Gamma \vdash a = b : A \end{array}$$

where the turnstile separates the context Γ from the judgment. To increase readability, we omit both of them if the context Γ is not discharged in a rule.

Given judgments

$$\Gamma \vdash a_1 : A_1$$
$$\Gamma \vdash a_2 : A_2(a_1)$$
$$\vdots$$
$$\Gamma \vdash a_n : A_n(a_1, a_2 \dots a_{n-1})$$
$$\Gamma, x_1 : A_1, x_2 : A_2(x_1), \dots, x_n : A_n(x_1, x_2 \dots x_{n-1}) \vdash B(x_1, x_2 \dots x_n) : \mathsf{type}$$
$$\Gamma, x_1 : A_1, x_2 : A_2(x_1), \dots, x_n : A_n(x_1, x_2 \dots x_{n-1}) \vdash b(x_1, x_2 \dots x_n) : B(x_1, x_2 \dots x_n)$$

we simply write

$$\Gamma \vdash B(a_1, a_2 \dots a_n) : \mathsf{type}$$
$$\Gamma \vdash b(a_1, a_2 \dots a_n) : B(a_1, a_2 \dots a_n)$$

for the simultaneous substitutions $B[a_1/x_1, a_2/x_2, \dots, a_n/x_n]$ and $b[a_1/x_1, a_2/x_2, \dots, a_n/x_n]$ of the terms $a_1, a_2 \dots a_n$ in $B(x_1, x_2 \dots x_n)$ and $b(x_1, x_2 \dots x_n)$ respectively. We identify types and terms up to α-conversion.
We are now led to add judgments expressing that Γ is a valid context and that two contexts Γ and Δ are equal:

$$\vdash \Gamma \ \mathsf{context} \qquad \vdash \Gamma = \Delta \ \mathsf{context}$$

Valid contexts are formed according to the following rules:

Rule 2.1 (Contexts)**.**

$$\frac{}{(\,)\ \mathsf{context}} \qquad \frac{\Gamma\ \mathsf{context} \qquad \Gamma \vdash A : \mathsf{type}}{\vdash (\Gamma, x : A)\ \mathsf{context}}$$

where $(\,)$ denotes the empty context and x is assumed to be fresh for Γ. ■

To simplify the presentation we split the rules of the system in different groups:

- *general rules* forming the equality, substitution and variable assumption rules;
- rules for *the type of small types*. These rules allow for the definition of the (large) type Set of all small types (we also refer to small types simply as sets);

- rules for certain *constant types* consisting of the minimal collections of basic types needed to start with to build all the other types. In particular at this stage we do not need to include any infinite set we will define them later via induction-recursion.
- rules for *set/type formers* needed to introduce dependent products and coproducts types;
- rules for *equality types* which allow for the formation of a type whose inhabitants witness that elements of a certain type are equal.

Both the rules for constant sets and the rules for set/type formers follow a common pattern. They are given in four steps:

- *formation rules* which introduce new sets/types;
- *introduction rules* which describe how canonical elements of the type are formed;
- *elimination rules* which tell us how to use elements of the type to define recursive functions out of it.
- *computation rules* which explain how functions defined by using the elimination rules act on canonical elements of the type.

General rules

Rule 2.2 (Equality)**.** Equality is an equivalence relation on types and terms:

$$\frac{A : \mathsf{type}}{A = A} \qquad \frac{B = A : \mathsf{type}}{A = B : \mathsf{type}} \qquad \frac{A = B : \mathsf{type} \quad B = C : \mathsf{type}}{A = C : \mathsf{type}}$$

$$\frac{a : A}{a = a : A} \qquad \frac{b = a : A}{a = b : A} \qquad \frac{a = b : A \quad b = c : A}{a = c : A}$$

■

Rule 2.3 (Conversion)**.**

$$\frac{a : A \quad A = B : \mathsf{type}}{a : B} \qquad \frac{a = b : A \quad A = B : \mathsf{type}}{a = b : B}$$

■

Rule 2.4 (Assumption, weakening)**.** In the following we indicate with $\mathcal{I}$ any judgment which can appear on the right of the turnstile

$$\frac{\Gamma \vdash A : \mathsf{type}}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma \vdash \mathcal{I}}{\Gamma, x : A \vdash \mathcal{I}}$$

■

Rule 2.5 (Substitution rules)**.** Substitution in types:

$$\frac{x : A \vdash B(x) : \mathsf{type} \quad a : A}{B(a) : \mathsf{type}} \qquad \frac{x : A \vdash B(x) = D(x) : \mathsf{type} \quad a = c : A}{B(a) = D(c) : \mathsf{type}}$$

Substitution in terms:

$$\frac{x : A \vdash b(x) : B(x) \quad a : A}{b(a) : B(a)} \quad \frac{x : A \vdash b(x) = d(x) : B(x) \quad a = c : A}{b(a) = d(c) : B(a)}$$

■

The type of small sets

We now give rules for the type Set of small sets. Further down we will give rule asserting closure of Set under certain set formers

Rule 2.6 (Set)**.**

$$\frac{}{\mathsf{Set} : \mathsf{type}} \qquad \frac{A : \mathsf{Set}}{\mathsf{El}(A) : \mathsf{type}} \qquad \frac{A = B : \mathsf{Set}}{\mathsf{El}(A) = \mathsf{El}(B) : \mathsf{type}}$$

Given $A : \mathsf{Set}$, we simply write $a : A$ as a shorthand for $a : \mathsf{El}(A)$ if no ambiguity can arise; in other words we allow ourselves to treat the large universe $(\mathsf{Set}, \mathsf{El})$ as if it was formulated à la Russell (cf. Section 3.1.3). ■

Constant types

We now introduce rules to postulate the existence of basic sets. Notice that we use large elimination for the set N_2, i.e. we allow elimination into type (rather then Set) for predicates defined on the set N_2: this will be needed later in Remark 2.12 and Notation 3.25 to define $+$-types and to define the sum of two inductive-recursive definitions.

Rule 2.7 (N_0, N_1, N_2)**.** Formation rules:

$$\frac{}{\mathsf{N}_0 : \mathsf{Set}} \qquad \frac{}{\mathsf{N}_1 : \mathsf{Set}} \qquad \frac{}{\mathsf{N}_2 : \mathsf{Set}}$$

Introduction rules: there is no introduction for N_0.

$$\frac{}{0_1 : \mathsf{N}_1} \qquad \frac{}{0_2 : \mathsf{N}_2}$$

$$\frac{}{1_2 : \mathsf{N}_2}$$

Elimination rules:

$$\frac{x : \mathsf{N}_0 \vdash P(x) : \mathsf{Set} \quad a : \mathsf{N}_0}{\mathsf{case}_0(a) : P(a)}$$

$$\frac{x : \mathsf{N}_1 \vdash P(x) : \mathsf{Set} \quad \begin{array}{r} a : \mathsf{N}_1 \\ b : P(0_1) \end{array}}{\mathsf{case}_1(a, b) : P(a)} \qquad \frac{x : \mathsf{N}_2 \vdash P(x) : \mathsf{type} \quad \begin{array}{r} a : \mathsf{N}_2 \\ b_0 : P(0_2) \\ b_1 : P(1_2) \end{array}}{\mathsf{case}_2(a, b_0, b_1) : P(a)}$$

Computation rule:

$$\frac{x : \mathsf{N}_1 \vdash P(x) : \mathsf{Set} \quad b : P(0_1)}{\mathsf{case}_1(0_1, b) = b : P(0_1)} \qquad \frac{x : \mathsf{N}_2 \vdash P(x) : \mathsf{type} \quad \begin{array}{r} i_2 : \mathsf{N}_2 \\ b_0 : P(0_2) \\ b_1 : P(1_2) \end{array}}{\mathsf{case}_2(i_2, b_1, b_2) = b_i : P(i_2)} \; i = \{0, 1\}$$

■

Type/set formers

Next we introduce Π-sets and Π-types. We deliberately use the same notation for both since this allows us to avoid duplicating the same rules.

Rule 2.8 (Dependent products)**.** Formation rules:

$$\frac{A : \mathsf{type} \quad x : A \vdash B(x) : \mathsf{type}}{(\Pi\, x : A) B(x) : \mathsf{type}}$$

$$\frac{A : \mathsf{Set} \quad x : A \vdash B(x) : \mathsf{Set}}{\vdash (\Pi\, x : A) B(x) : \mathsf{Set}}$$

Introduction rule:

$$\frac{x : A \vdash b(x) : B(x)}{\lambda\, x : A.\, b(x) : (\Pi\, x : A) B(x)}$$

Elimination rule:

$$\frac{f : (\Pi\, x : A) B(x) \quad a : A}{f(a) : B(a)}$$

Computation rules:

$$\frac{x : A \vdash b(x) : B(x) \quad a : A}{(\lambda\, x : A.\, b(x))(a) = b(a) : B(a)}\ \beta\,\text{pi}$$

$$\frac{x : A \vdash b(x) = c(x) : (\Pi\, x : A)B(x)}{\lambda\, x : A.\, b(x) = \lambda\, x : A.\, c(x) : (\Pi\, x : A)B(x)}\ \xi\,\text{pi}$$

$$\frac{b : (\Pi\, x : A)B(x)}{\lambda\, x : A.\, b(x) = b : (\Pi\, x : A)B(x)}\ \eta\,\text{pi}$$

■

We introduce now the rules for Σ-sets and Σ-types using the same notation for both as we did for Π-sets and Π-types.

Rule 2.9. (Dependent coproducts) Formation rule:

$$\frac{A : \mathsf{type} \quad x : A \vdash B(x) : \mathsf{type}}{(\Sigma\, x : A)B(x) : \mathsf{type}}$$

$$\frac{A : \mathsf{Set} \quad x : A \vdash B(x) : \mathsf{Set}}{(\Sigma\, x : A)B(x) : \mathsf{Set}}$$

Introduction rule:

$$\frac{a : A \quad b : B(a)}{(a, b) : (\Sigma\, x : A)B(x)}$$

Eliminations rule:

$$\frac{p : (\Sigma\, x : A)B(x)}{\pi_0(p) : A}\ \text{fst pr} \qquad \frac{p : (\Sigma\, x : A)B(x)}{\pi_1(p) : B(\pi_0(p))}\ \text{snd pr}$$

Computation rule:

$$\frac{a : A, \quad a : A \vdash b : B(a)}{\pi_0(a, b) = a : A} \qquad \frac{a : A \quad a : A \vdash b : B(a)}{\pi_1(a, b) = b : B(a)}$$

$$\frac{p : (\Sigma\, x : A)B(x)}{p = (\pi_0(p), \pi_1(p)) : (\Sigma\, x : A)B(x)}\ \eta\,\text{sig}$$

■

Notation 2.10. In the rest of the thesis we adopt some conventions about dependent product and coproducts which we summarize here: when the type A can be inferred from the context we write $\lambda\, x.\, b$ for $\lambda\, x : A.\, b(x)$. When writing codes for inductive, indexed inductive and inductive-recursive definitions we write a function $\lambda\, a.\, \gamma(a)$ simply as $a \mapsto \gamma(a)$. For simultaneous abstractions $\lambda\, x.\, \lambda\, y.\, b(x, y)$ and applications $f(a)(b)$ we write $\lambda\, x, y.\, b(x, y)$ and $f(a, b)$ respectively.
We sometimes abbreviate $(\Pi\, x : A)B(x)$ and $(\Sigma\, x : A)B(x)$ simply as $(\Pi\, A)B$ and $(\Sigma\, A)B$; if B does not depend on $x : A$ we write $A \to B$ and $A \times B$ for $(\Pi\, a : A)B$ and $(\Sigma\, x : A)B$ respectively.

Since we have introduced function types and the universe Set we can now use an alternative notation for a family of sets $x : A \vdash B(x) : \mathsf{Set}$.

Notation 2.11 (families)**.** A family of sets $x : A \vdash B(x) : \mathsf{Set}$ can be equivalently presented as given by a pair $(A, B) : (\Sigma\, A : \mathsf{Set})(A \to \mathsf{Set})$. Indeed, we can derive the latter judgment form the former by using introduction for Π-types, the rule $\eta\,\mathrm{pi}$ and the introduction rule for Σ-types. Vice versa we can derive the former from the latter by elimination for Σ-types and Π-types and the rule 2.4.

We do not include $+$-types among our set formers simply because they can be defined by using Σ-sets and large elimination for N_2.

Remark 2.12 (Sums)**.** Given $A : \mathsf{Set}$, $B : \mathsf{Set}$ we can define $A + B$ as

$$A + B =_{\mathsf{def}} (\Sigma\, x : \mathsf{N}_2)\, \mathsf{case}_2(x, A, B)$$

With this definition, the injections $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$ are defined as $\mathsf{inl} =_{\mathsf{def}} \lambda\, a : A.\, (\mathsf{0}_2, a)$ and $\mathsf{inr} =_{\mathsf{def}} \lambda\, b : B.\, (\mathsf{1}_2, b)$ respectively. Given maps $f : A \to C$ and $g : B \to C$ the coproduct map $[f, g] : A + B \to C$ is given by $[f, g] =_{\mathsf{def}} \lambda\, x : A + B.\, \mathsf{case}_2(\pi_0(x), f(\pi_1(x)), g(\pi_1(x)))$.
Similarly note that we have not included any finite sets apart from N_0, N_1 and N_2. All the other finite sets N_n can be defined from N_1 and N_2 using sums: for example $\mathsf{N}_3 = \mathsf{N}_2 + \mathsf{N}_1$.

Extensional rules

We finish this presentation with rules for identity types and a brief discussion about extensional rules which may or may not be added to the system considered so far. Identity types represent a notion of equality internal to the system. They come in two shapes, intensional or extensional. The difference between these is so relevant that they actually give rise to two different systems respectively: intensional type theory and extensional type theory. The latter identifies internal and judgmental equality, with far-reaching consequences. Models for extensional type theory have been intensively studied [87, 27, 59, 30], but there are also drawbacks: type-checking is undecidable, and the theory fails to be strongly normalizing. Most of this thesis will use categorical reasoning and this will be greatly simplified by working in the extensional setting.

Rule 2.13 (Identity types)**.** Formation rule:

$$\frac{A : \mathsf{type} \quad a : A \quad b : A}{a \equiv_A b : \mathsf{type}}$$

Introduction rule:

$$\frac{A : \mathsf{type} \quad a : A}{\mathsf{refl}(a) : a \equiv_A a}$$

Elimination rule:

$$\frac{\begin{array}{c} A : \mathsf{type} \quad x, y : A, q : x \equiv_A y \vdash P(x, y, q) : \mathsf{Set} \\ a, b : A \quad p : a \equiv_A b \quad x : A \vdash c(x) : P(x, x, \mathsf{refl}(x)) \end{array}}{\mathsf{J}(a, b, p, c) : P(a, b, p)}$$

Computation rule:

$$\frac{\begin{array}{c} A : \mathsf{type} \quad x, y : A, q : x \equiv_A y \vdash P(x, y, q) : \mathsf{Set} \\ a : A \qquad x : A \vdash c(x) : P(x, x, \mathsf{refl}(x)) \end{array}}{\mathsf{J}(a, a, \mathsf{refl}(a), c) = c(a) : P(a, a, \mathsf{refl}(a))}$$

■

Notice that within the proposition-as-types perspective the above elimination rule states that in order to prove a statement about an identity type it suffices to prove it on diagonal elements, i.e. on terms of the form $(x, x, \mathsf{refl}(x))$.
We now state the rules turning intensional type theory into its extensional versions.

Rule 2.14 (Equality reflection rule)**.**

$$\frac{p : x \equiv_A y}{x = y : A}\ \text{eq-reflection} \qquad \frac{p : a \equiv_A a}{p = \mathsf{refl}(a) : a \equiv_A a}\ \text{UIP} \quad \blacksquare$$

As proved by Hofmann and Streicher in [62] these rules are not derivable from the rules already stated. The counter model built in [62] to prove this has paved the way for further investigations of homotopical models for the intensional version [92].
Finally let us present function extensionality asserting that pointwise propositionally equal functions are themselves propositionally equal

Rule 2.15 (Function extensionality)**.**

$$\frac{f, g : (\Pi\, x : A)B(x) \quad p : (\Pi\, x : A) f(x) \equiv_{B(x)} g(x)}{\mathsf{funext}(\mathsf{f}, \mathsf{g}, \mathsf{p}) : f \equiv_{(\Pi\, x : A)B(x)} g} \quad \blacksquare$$

This rule which reflects common usage of functions in mathematics can be safely added without breaking decidability of type-checking but it seem harder to justify from a computational point of view where terms are regarded as algorithms: two functions which are pointwise equal can have quite different algorithmic content (see e.g. Section 5 in [48] for more on this). This rule is derivable from Rule 2.14.

2.2 Inductive definitions and their categorical semantics

The theory of dependent types described so far is a solid system for reasoning about dependent types but the range of sets and types which can be built within it is particularly narrow: for example, observe that the theory described so far does not allow us to build any infinite set. The aim of this Section is to increase expressiveness of the system by adding scheme for inductive types. We start by formulating in a natural deduction style the rules for the basic examples presented in Section 1.2 and then by introducing comprehensive schemata for inductive types. While introducing examples of inductive types and schemata for defining them we will pay attention to

the categorical approach for describing them represented by initial algebra semantics.
The rules in Example 1.1 translate directly to the introduction rules of the type N of natural numbers from which we can derive the corresponding elimination and computation rules:

Rule 2.16 (Natural numbers)**.** Formation rule:

$$\frac{}{\mathsf{N} : \mathsf{Set}}$$

Introduction rules:

$$\frac{}{\mathsf{0} : \mathsf{N}} \qquad \frac{n : \mathsf{N}}{\mathsf{s}\,(n) : \mathsf{N}}$$

Elimination rules

$$\frac{a : \mathsf{N} \quad d : P(\mathsf{0}) \quad \begin{array}{c} n : \mathsf{N} \vdash P(n) : \mathsf{Set} \\ n : \mathsf{N}, y : P(n) \vdash e(n, y) : P(\mathsf{s}(n)) \end{array}}{\mathsf{natrec}(a, d, e) : P(a)}$$

Computation rules:

$$\frac{d : P(\mathsf{0}) \quad \begin{array}{c} n : \mathsf{N} \vdash P(n) : \mathsf{Set} \\ n : \mathsf{N}, y : P(n) \vdash e(n, y) : P(\mathsf{s}(n)) \end{array}}{\mathsf{natrec}(\mathsf{0}, d, e) = d : P(\mathsf{0})}$$

$$\frac{a : \mathsf{N} \quad d : C(\mathsf{0}) \quad \begin{array}{c} n : \mathsf{N} \vdash P(n) \\ n : \mathsf{N}, y : P(n) \vdash e(n, y) : P(\mathsf{s}(n)) \end{array}}{\mathsf{natrec}(\mathsf{s}(a), d, e) = e(a, \mathsf{natrec}(a, d, e)) : P(\mathsf{s}(a))}$$ ■

Similarly, we can turn the rules given in Example 1.2 in the introduction rules for a type Ω (for simplicity we omit the corresponding elimination and computation rules).

Rule 2.17 (Brouwer ordinals)**.** Introduction rules:

$$\frac{}{\mathsf{0} : \Omega} \qquad \frac{\alpha : \Omega}{\mathsf{s}\,(\alpha) : \Omega} \qquad \frac{f : \mathsf{N} \to \Omega}{\mathsf{lim}\,(f) : \Omega}$$ ■

From a constructive viewpoint, where induction is a basic principle, it is natural to allow for a wide class of inductively defined structures. As a

paradigm of such a broadly comprehensive inductively generated structure in Martin-Löf type theory we have the set of well-orderings or, shortly, W-types [73]. A type $(\mathrm{W}\, x : A)B(x)$, also indicated $(\mathrm{W}A)B$, represents the set of well-founded trees whose nodes are elements $a : A$ and whose branches over such a node a are given by elements of a set $a : A \vdash B(a)$. The rules describing W-types in Martin-Löf type theory are the following:

Rule 2.18 (W-types)**.** Formation rule

$$\frac{x : A \vdash B(x)}{(\mathrm{W}\, x : A)B(x) : \mathsf{Set}}$$

Introduction rule

$$\frac{a : A \quad f : B(a) \to (\mathrm{W}\, x : A)B(x)}{\mathsf{sup}\,(a, f) : (\mathrm{W}\, x : A)B(x)}$$

Elimination rule

$$\frac{\begin{array}{c} w : (\mathrm{W}\, x : A)B(x) \\ w : (\mathrm{W}\, x : A)B(x) \vdash P(w) : \mathsf{Set} \\ a : A, f : B(a) \to (\mathrm{W}\, x : A)B(x), g : (\Pi\, b : B(a))P(f(b)) \vdash h(a, f, g) : P(\mathsf{sup}\,(a, f)) \end{array}}{\mathsf{wrec}(w, h) : P(w)}$$

Computation rule

$$\frac{\begin{array}{c} w : (\mathrm{W}\, x : A)B(x) \\ w : (\mathrm{W}\, x : A)B(x) \vdash P(w) \\ a : A, f : B(a) \to (\mathrm{W}\, x : A)B(x), g : (\Pi\, b : B(a))\, P(f(b)) \vdash h(a, f, g) : P(\mathsf{sup}\,(a, f)) \end{array}}{\mathsf{wrec}(\mathsf{sup}\,(a, f), h) = h(a, f, \lambda t.\, \mathsf{wrec}(f(t), h))}$$

■

An insightful perspective from which looking at a W-types $(\mathrm{W}A)B$, is as a direct generalization of the free term algebra construction from finite arities to arbitrary arities specified by a signature: a set A of term constructors or operators $a : A$, each with a small arity given by the set $B(a)$.
In extensional type theory W-types can been used to encode inductively defined types. For example we can encode the types N and Ω defined in Rules 2.16 and in Rule 2.17 respectively:

Example 2.19. The type of natural numbers can be encoded by the W-type

$$(\mathrm{W}\, x : \mathsf{N}_2)B_{\mathsf{N}}(x)$$

where

$$B_{\mathsf{N}}(0_2) = \mathsf{N}_0 \qquad B_{\mathsf{N}}(1_2) = \mathsf{N}_1$$
$$0 = \mathsf{sup}(0_2, \lambda\, x.\, \mathsf{case}_0(x)) \qquad \mathsf{s}(n) = \mathsf{sup}(1_2, \lambda\, x.\, \mathsf{case}_1(x, n))$$

and recursion operator defined by

$$\mathsf{natrec}(a, d, e) = \mathsf{wrec}(a, \lambda\, x, y, z.\, \mathsf{case}_2(a, d, e(y(0_1), z(0_1))))$$

Similarly the type of Brouwer ordinals is given by

$$(\mathrm{W}\, x{:}\, \mathsf{N}_3) B_\Omega(x)$$

where

$$B_\Omega(0_3) = \mathsf{N}_0 \qquad B_\Omega(1_3) = \mathsf{N}_1 \qquad B_\Omega(2_3) = \mathsf{N}$$
$$0 = \mathsf{sup}(0_3, \lambda\, x.\, \mathsf{case}_0(x)) \quad \mathsf{s}(n) = \mathsf{sup}(1_3, \lambda\, x.\, \mathsf{case}_1(x, n)) \quad \mathsf{lim}(f) = \mathsf{sup}(2_3, f)$$

■

These examples show how easily inductive types can be defined in terms of W-types. Indeed these are just two instances of a wide class of inductive types which can be captured by W-types. As shown by Dybjer [36], all strictly positive types – those types which are built up from set variables and constants using as operators $+$, $\times$ and $\to$, with the restriction that no set variable occurs to the left of an arrow – can be captured by the W-construction. Abbott et al. [2] have sharpened this result showing that W-types are sufficient to capture also nesting of inductive types.

Notice that some extensionality is needed for the above reduction to hold: the elimination and computation rules for the encoding of such inductive types as W-types can not be derived in intensional type theory. This is because of the functional component f in an element $\mathsf{sup}(a, f) : (\mathrm{W}\, x : A) B(x)$ which prevents us from deriving $\mathsf{sup}(a, f) = \mathsf{sup}(a, g)$ if f and g are not convertible to each other. Thus, in intensional type theory the W-types encoding an inductive type contains elements which do not represent any canonical element of the inductive type. This problem has been addressed also in the context of homotopy type theory: in [15] Awodey et al. have shown that the notion of weak W-type and a limited form of function extensionality suffice to encode simple inductive types in the usual way.

2.2.1 Initial algebra semantics

Initial algebra semantics [55] is one of the cornerstones in the categorical analysis of data types. To provide a backdrop for the following discussion, we give a brief summary of how initial algebra semantics can be used to provide a semantics to data types.
Within the paradigm of initial algebra semantics, each data type is regarded as the carrier of an initial algebra of a given functor; therefore we start recalling the fundamental notion of algebra of a functor.

Definition 2.20. If $\mathbb{C}$ is a category and $F\colon \mathbb{C} \to \mathbb{C}$ is an endofunctor on $\mathbb{C}$, then an F-algebra is a pair (X, h) where X is an object of $\mathbb{C}$ and $h\colon FX \to X$ a morphism. We call X the carrier of the algebra and h the structure map. ■

Intuitively, if we think of F as building structure on top of X, then the morphism h shows how to embed this new structure back into X. Thus we can think of X as having enough structure to model the new structure built by F. In other words, we can think of X as a *model* of the structure specified by F.
For any functor F, the collection of F-algebras itself forms a category $\mathbf{Alg}_F$ which we call the category of F-algebras. In $\mathbf{Alg}_F$, a morphism from (X, h) to (Y, g) is a map $f : X \to Y$ such that the following diagram commutes

$$\begin{array}{ccc} FX & \xrightarrow{h} & X \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle f} \\ FY & \xrightarrow{g} & Y \end{array}$$

If we think of X and Y as models of structure built by F, then an F-algebra morphisms $f : (X, h) \to (Y, g)$ is nothing but an homomorphism between such models, i.e. a structure preserving map. An *initial* or *free* F-algebra, which we indicate by $(\mu F, \mathsf{intro}_F)$, is an initial object in the category $\mathbf{Alg}_F$. As the adjective *free* above suggests, the idea is that the structure of μF has been freely generated so as to make it the *initial* F-model. Initial algebras

are of particular interest in the solution of recursive equations since they are fixed points (cf. e.g. [90]). Being a fixed point is expressed categorically by the following definition:

Definition 2.21. Given a functor $F : \mathbb{C} \to \mathbb{C}$, an object X is called a fixed point of F if there exists an isomorphism $FX \cong X$. ■

Any fixed point for a functor F is clearly an F-algebra. That the converse is true for initial algebras is an useful observation due to Lambek [68]:

Lemma 2.22. An initial F-algebra is a fixed point for F.

Proof. Let $(\mu F, \mathsf{intro}_F)$ be an initial algebra for F. Then $(F(\mu F), F(\mathsf{intro}_F))$ is an algebra for F, and by initiality of $(\mu F, \mathsf{intro}_F)$ we get a map $g : \mu F \to F(\mu F)$ such that $g \circ \mathsf{intro}_F = F(\mathsf{intro}_F) \circ F(g)$. Now, $\mathsf{intro}_F \circ g$ is an F-algebra morphism from $(\mu F, \mathsf{intro}_F)$ to itself and by initiality it has to be the identity, i.e. $\mathsf{intro}_F \circ g = \mathsf{id}$. But then we also have $\mathsf{id} = F(\mathsf{id}) = F(\mathsf{intro}_F \circ g) = F(\mathsf{intro}_F) \circ F(g) = g \circ \mathsf{intro}_F$. □

Initial algebras from below

The proof that initial algebras for a functor can be built *from below* can be traced back to famous fixed point theorems like those of Kleene and Knaster-Tarski. And, indeed, it is a simple and direct translation of the latter theorem in categorical terms which will enable us to build initial algebras for the functors we will consider throughout this thesis. Recall that for every functor F on a category $\mathbb{C}$ with an initial object $\bot$ and with filtered colimits we have an associated initial chain.

Definition 2.23 (Initial chain)**.** Given an endofunctor F on a category $\mathbb{C}$ with filtered colimits and an initial object $\bot$, define the initial chain of F by transfinite recursion as follows:

$$\begin{aligned} F^0 &= \bot \\ F^{\alpha+1} &= F(F^\alpha) \\ F^\alpha &= \bigvee_{\beta<\alpha} F^\beta \text{ for } \alpha \text{ limit.} \end{aligned}$$ ■

This definition actually defines a chain since we can define $k_{i,j} : F^i \to F^j$ for $i < j$ in the following way:

$$\begin{array}{lcl} k_{0,1} & = & \bot \to F(\bot) \\ k_{i+1,j+1} & = & F(k_{i,j}) \\ k_{i,j} & = & \text{the colimit cocone for } i \text{ limit.} \end{array}$$

Definition 2.24. We say that the initial chain for a functor F converges after λ steps if $k_{\lambda+1,\lambda}$ is an isomorphism. ■

Theorem 2.25 ([10], Theorem 3.1.4)**.** Let $\mathbb{C}$ be a category with an initial object $\bot$ and with colimits of chains. If the initial chain of a functor F converges in λ steps, then $\mu F = F^\lambda$ and $\mathsf{intro}_F = k^{-1}_{\lambda+1,\lambda}$.

Semantics for data types

Let us now come back to our starting motivation of using initial algebra semantics to model data types through the language of category theory: the object μF interprets the data type described by F, while the algebra intro_F interprets the constructors of the data type. Initiality of $(\mu_F, \mathsf{intro}_F)$ can be used to express a recursion principle for the data type. Given an F-algebra (X, h), by initiality of $(\mu_F, \mathsf{intro}_F)$ we get an F-algebra homomorphism, sometimes referred to as *catamorphism*, $\mathsf{fold}(h)$, from the initial algebra intro_F to the algebra h. We can express this recursion principle by means of an operator $\mathsf{fold} : (F(X) \to X) \to \mu F \to X$, together with equations $\mathsf{fold}(\_) \circ \mathsf{intro}_F = \_ \circ F(\mathsf{fold}(\_))$ expressing that the map $\mathsf{fold}(h)$ is an F-algebra morphism for a given F-algebra (X, h). Indeed, the actual computational interpretation of $\mathsf{fold}(h)$ for $h : F(X) \to X$ is given by the above equation $\mathsf{fold}(h) \circ \mathsf{intro}_F = h \circ F(\mathsf{fold}(h))$. It tell us that in order to define a function from the data type it suffices to recursively apply h along the constructors of the data type. Unicity of $\mathsf{fold}(h)$ translates directly to an η-rule for the data type. The fold operator represents a simple form of the kind of elimination rules we have already met. These are a priori more general since they allow the construction of dependent recursive functions out of a data type: we can eliminate on types which might depend on elements of our data type. However, by using Σ-types it can be shown that these two rules are

interderivable. We sketch how to derive the fold operator from the elimination rule and vice versa by means of examples. We start giving a closer look to the first implication (elimination implies recursion via a fold operator) by looking at the simple example of the type N. In Section 2.2.2 we show how to derive the elimination rule from the fold operator in the case of W-types. We refer the reader to [44, 51, 52] for an abstract framework where general induction schemes can be formulated and proved equivalent to the existence of initial algebras.

Natural numbers object

Let us illustrate how initial algebra semantics can be used to model inductive types through the analysis of the simple example of the type of N. We start by giving a purely categorical definition of the type N.

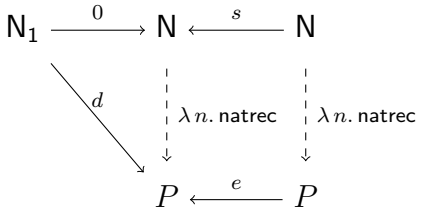
Example 2.26 (Example 1.1 continued)**.** A natural numbers object N in a category $\mathbb{C}$ with terminal object and binary coproduct is the carrier of an initial algebra for the endofunctor $F : \mathbb{C} \to \mathbb{C}$ whose action on an object X is given by $F(X) = 1 + X$. ■

As before, once we have defined N we can proceed further and define Ω.

Example 2.27 (Example 1.2 continued)**.** Given a natural number object N in $\mathbb{C}$ and assuming that it is an exponentiable object of $\mathbb{C}$ we define a Brouwer ordinals object as the initial algebra for the endofunctor $G : \mathbb{C} \to \mathbb{C}$ defined by $G(X) = 1 + X + X^{\mathsf{N}}$. ■

Let us now look at these definitions from the perspective of initial algebra semantics. In Set an initial algebra for the functor F is given by the pair $(\mathsf{N}, \mathsf{intro}_F)$ where the carrier N is given by the formation rule in 2.16 and $\mathsf{intro}_F : \mathsf{N}_1 + \mathsf{N} \to \mathsf{N}$ is the structure map defined as $\mathsf{intro}_F =_{\mathsf{def}} [\mathsf{0}, \mathsf{s}]$, where $\mathsf{0} =_{\mathsf{def}} \lambda x\,.\mathsf{0} : \mathsf{N}_1 \to \mathsf{N}$ and $\mathsf{s} =_{\mathsf{def}} \lambda n.\,\mathsf{s}(n) : \mathsf{N} \to \mathsf{N}$ are defined by the introduction rules in 2.16. Similarly, the structure map intro_G of an initial algebra $(\Omega, \mathsf{intro}_G)$ for the functor G defined in Example 2.27 is given by $[\mathsf{0}, \mathsf{s}, \mathsf{lim}] : \mathsf{N}_1 + \Omega + \Omega^{\mathsf{N}} \to \Omega$. Notice how formation and introduction rules for an inductive type can be used to build the corresponding algebra.

Weak initiality of $(\mathsf{N}, \mathsf{intro}_F)$ can be derived from the elimination rule in 2.16 by instantiating the latter for a non dependent type P: the premises of the rules asserts that P has an F-algebra structure given by $[\lambda\, \mathsf{0_1}.\, d, \lambda\, y.\, e]$ while the conclusion state the existence of a function $\lambda\, n.\, \mathsf{natrec}(a, d, e) : \mathsf{N} \to P$. The computation rule ensures that this map is indeed an F-algebra morphism. This is a direct translation of $(\mathsf{N}, \mathsf{intro}_F)$ being weakly initial in Set as the following diagram displays:



Using rules 2.14, 2.15 and 2.16 we can show that the morphism

$$\lambda\, n.\, \mathsf{natrec}(n, d, e) : \mathsf{N} \to P$$

is indeed unique. We can also show that the elimination and computation rules in 2.16 are derivable from the property of $(\mathsf{N}, \mathsf{intro}_F)$ being initial. But since this argument is completely analogous to the one we will use in Section 2.2.2 for W-types we do not give it here.

2.2.2 Initial algebra semantics of inductive types

A major step in the categorical understanding of inductive types has been the formulation of a categorical semantics for W-types. This has been a common theme for different group of researchers in recent years [36, 77, 46, 3]. In order to capture a notion of a W-type in a category endowed with sufficient structure, we turn to the language of initial algebra semantics. But first we need to identify the right class of functors to look at.

In this section we present different schemata which define – modulo minor differences – the same class of functors. Initial algebras for these functors are exactly the W-types. The three approaches we present are polynomials, containers and Inductive-Definitions. The main difference among the three

consists in the choice of the syntactic objects used to represent the functors we aim to capture: polynomials use *arrows*, containers use *families* while Inductive-Definitions use *codes*. Being aware that the presentation will end up being redundant we hope the reader will profit from this. In the rest of the thesis we use all of these three different schemes to present an inductive type depending on which aspects we want to highlight.

Containers

The notion of container and container functor has been extensively studied in Abbott's PhD thesis [1] and further refined and applied by Abbott and his coauthors in a series of papers [3, 2, 4]. Containers are first class objects of Martin-Löf type theory because they are nothing but families of sets. Thus they provide a rather direct way to reason about inductive types inside type theory.

Definition 2.28. A container is a pair (S, P) where $S : \mathsf{Set}$ and $P : S \to \mathsf{Set}$. ■

A metaphor informs the choice of the name container: each data type defined by a container consists of a set of shapes S and for each shape $s : S$ a set of positions $P(s)$ where data can be stored. A family of sets (S, P) can also be regarded as representing a (single sorted) signature: the set of shapes S represents the set of term-constructors while, for $s : S$, the set of its positions $P(s)$ represents the arity of the constructor represented by s. We will come back to this in the next section. Every container defines an associated functor on Set, called its extension.

Definition 2.29. Given a container (S, P), its extension is the functor $[\![S, P]\!]_{\mathsf{C}} : \mathsf{Set} \to \mathsf{Set}$ whose action on a set X is given by

$$[\![S, P]\!]_{\mathsf{C}}\, X =_{\mathsf{def}} (\Sigma\, s : S)\, (P(s) \to X)$$ ■

Abbott [1] develops the theory of containers in a fibrational setting. As explained in Section 2.18 in [47] the notion of container and its extension is modulo minor differences the same as that of polynomial and polynomial functor and most of the theory can be developed in a locally cartesian closed

category with a terminal object (see Section 1 in [47]). Therefore, we now introduce the category of containers and recall some of its rich structure which will be used in the rest of the thesis; more categorical analysis will come in the next section.

Definition 2.30. The category of containers Cont has as objects containers, and as morphisms $(S, P) \to (S', P')$ pairs (u, v) where $u : S \to S'$ and $v : (\Pi\, s : S) P'(u(s)) \to P(s)$. ■

To simplify the presentation and the categorical analysis we now introduce the category **Cont** of containers defined on **Set**. We will use the category **Cont** to show the rich categorical structure of the category of containers.

Definition 2.31. The category **Cont** of containers defined on **Set** has as objects pairs (S, P) where $S \in \mathbf{Set}$ and $P : S \to \mathbf{Set}$, and as morphisms $(S, P) \to (S', P')$ pairs (u, v) where $u : S \to S'$ and $v : (\Pi\, s : S) P'(u(s)) \to P(s)$.
Given a container $(S, P) : \mathbf{Cont}$, its extension is the functor $[\![S, P]\!]_{\mathsf{C}} : \mathbf{Set} \to \mathbf{Set}$ whose action on a set X is given by $[[\![S, P]\!]_{\mathsf{C}}\, X =_{\mathsf{def}} (\Sigma\, s : S)\, (P(s) \to X)$. ■

Remark 2.32. The category **Cont** can be easily seen to be nothing but the category $\mathbf{Fam}\,(\mathbf{Set}^{\mathsf{op}})$ (see Section 3.2.3).

Theorem 2.33 (full and faithfulness of $[\![\ \ ]\!]_{\mathsf{C}}$)**.** The extension function

$$[\![\ \ ]\!]_{\mathsf{C}} : \mathbf{Cont} \to [\mathbf{Set}, \mathbf{Set}]$$

mapping containers to functors can be extended to a full and faithful functor mapping container morphisms to natural transformations.

Proof. Define the component at X of $[\![(u, v)]\!]_{\mathsf{C}}$ as follows:

$$([\![u, v]\!]_{\mathsf{C}})_X : (\Sigma\, s : S)\, (P(s) \to X) \to (\Sigma\, s' : S')\, (P(s') \to X)$$
$$([\![u, v]\!]_{\mathsf{C}})_X (s, f) =_{\mathsf{def}} (u(s), f \circ (v(s)))$$

Since every container is a sum of representable functors full and faithfulness is just a consequence of full and faithfulness of the Yoneda embedding. □

Remark 2.34. Notice that in Theorem 2.33 we refer to natural transformation rather than *strong* natural transformation as in Theorem 2.47 below. This is because in **Set** every endofunctor can be endowed of a canonical strength compatible with naturality squares, making every natural transformation between endofunctor of **Set** automatically strong.

Lemma 2.35. Cont is a cocomplete cartesian closed category.

Proof. Since $\mathbf{Cont} = \mathbf{Fam}\,(\mathbf{Set}^{\mathsf{op}})$ and $\mathbf{Set}^{\mathsf{op}}$ has all small connected colimits **Cont** is cocomplete (cf. point 4 in Remark 3.10). The initial object is given by $(\mathsf{N}_0, !)$, the terminal object by $(\mathsf{N}_1, \lambda\, x.\, \mathsf{N}_1)$. The coproduct of two containers (S, P) and (S', P') is given by $(S \times S', \lambda\, x.\, (P(\pi_0(x)) + P(\pi_1(x))))$ while their product by $(S \times S', \lambda\, x.\, (P(\pi_0(x)) \times P(\pi_1(x))))$. This easily generalize to arbitrary families of containers. To define exponentiation in Cont Altenkirch et al. [12] noted that for a functor $F : \mathbf{Set} \to \mathbf{Set}$ one has $F^{[\![\mathsf{N}_1, A]\!]_{\mathsf{C}}} \cong F \circ (A +)$ where $(A +)$ is the functor $X \mapsto (A + X)$. It then follows $F^{[\![S,P]\!]_{\mathsf{C}}} \cong F^{(\Sigma\, s : S)[\![1, P(s)]\!]_{\mathsf{C}}} \cong (\Pi\, s : S)(F \circ (P(s) +))$ which is a container if F is, since **Cont** is closed under products and composition as shown below. □

Given containers (S, P) and (S', P'), their composite $(S, P) \cdot (S', P')$ is the container (R, Q), defined as follows:

$$\begin{aligned} R &=_{\mathsf{def}} (\Sigma\, s : S)(P(s) \to S') \\ Q((s, f)) &=_{\mathsf{def}} (\Sigma\, p : P(s))P'(f(p)). \end{aligned}$$

This composition is compatible with the extensions of the corresponding functors, i.e. we have $[\![(S, P) \cdot (S', P')]\!]_{\mathsf{C}} \cong [\![(S, P)]\!]_{\mathsf{C}} \circ [\![S', P']\!]_{\mathsf{C}}$. This composite operation can be extended to morphisms between containers and reflected in the semantics by Theorem 2.33. This operation gives also a bicategorical structure to **Cont** which we do not pursue here (cf. Section 4.4 in [1]).
We now state the main result which makes containers relevant for our discussion of inductive types.

Theorem 2.36 (Theorem 5.2.2 in [1])**.** W-types are initial algebras for containers, i.e. $\mu[\![S, P]\!]_{\mathsf{C}} = (\mathrm{W}\, S)P$ and $\mathsf{intro}_{[\![S,P]\!]_{\mathsf{C}}} = \mathsf{sup} : (\Sigma\, s : S)(P(s) \to (\mathrm{W}\, S)P) \to (\mathrm{W}\, S)P$.

We will come back on this result in the next section. Now we focus on some examples of containers representing simple inductive types in type theory.

Example 2.37 (Example 1.1 continued)**.** The container whose initial algebra is the set of natural numbers is $(S_{\mathsf{N}}, P_{\mathsf{N}})$ where $S_{\mathsf{N}} = \mathsf{N}_2$ and $P_{\mathsf{N}} : S_{\mathsf{N}} \to \mathsf{Set}$ is defined as $P(0_2) = \mathsf{N}_0$ and $P(1_2) = \mathsf{N}_1$. Notice that N has two constructors of arity $\mathsf{0}$ and $\mathsf{1}$ respectively. These correspond to the two shapes of S_{N} and their corresponding positions. It is immediate to check that the extension of $(S_{\mathsf{N}}, P_{\mathsf{N}})$ is the functor $F(X) = 1 + X$. ■

Example 2.38 (Example 1.2 continued)**.** The container representing the functor whose initial algebra is the set of Brouwer ordinals has shapes $S_\Omega = \mathsf{N}_3$ and corresponding positions $P_\Omega \colon S_\Omega \to \mathsf{Set}$ given by $P(0_3) = \mathsf{N}_0$, $P(1_3) = \mathsf{N}_1$, $P(2_3) = \mathsf{N}$. ■

Polynomial functors

The class of functors whose initial algebras are the categorical version of W-types has been identified by Moerdjik and Palmgren [77] as what they dubbed polynomial functors. The theory has been further developed by Gambino et al. [46, 47]. The starting point of their analysis is Seely's correspondence between Martin-Löf type theory and locally cartesian closed categories (lccc) (see e.g. [87, 27, 59, 30]). Therefore, to begin with, recall that in every lccc we have for every f in $\mathbb{C}^{\to}$ the following chain of adjunctions:

$$\Sigma_f \dashv \Delta_f \dashv \Pi_f$$

where Δ_f is the reindexing functor given by pulling back along f. In the internal logic of $\mathbb{C}$ a morphism $f : X \to A$ can be represented through its fibers as $(X_a)_{a:A}$, where each X_a is given by $f^{-1}(a) = (\Sigma\, x : X)(f(b) \equiv_A a)$. Given a morphism $f : B \to A$ in $\mathbb{C}$, the action of Δ_f and its adjoint can then be explicitly computed as follows:

$$\begin{aligned}
\Delta_f((X_a)_{a:A}) &= (X_{f(b)})_{b:B} \\
\Sigma_f((X_b)_{b:B}) &= ((\Sigma\, b : B_a)\, X_b)_{a:A} \\
\Pi_f((X_b)_{b:B}) &= ((\Pi\, b : B_a)\, X_b)_{a:A}
\end{aligned}$$

These functors allow us to define for each morphism $f : P \to S$ in $\mathbb{C}$ a specific endofunctor on $\mathbb{C}$, the polynomial functor $\mathcal{P}_f$ associated to f.

Definition 2.39. Given a map $f : P \to S$ the polynomial functor $\mathcal{P}_f : \mathbb{C} \to \mathbb{C}$ associated to f is defined as the composite

$$\mathbb{C} = \mathbb{C}/1 \xrightarrow{\Delta_{!B}} \mathbb{C}/P \xrightarrow{\Pi_f} \mathbb{C}/S \xrightarrow{\Sigma_{!A}} \mathbb{C}/1 = \mathbb{C}$$

■

Using the internal logic of $\mathbb{C}$, the action on an object $X \in \mathbb{C}$ is given explicitly by

$$\mathcal{P}_f(X) = (\Sigma\, s : S)(P_s \to X)$$

where we have used P_s to indicate the fiber of f above s. Without much surprise, this functor can be recognized to be the same as the extension of a container (see Definition 2.29). The name "polynomial" is justifyied by $\mathcal{P}_f$ being a sum of products.

We now reason in Set to convince the reader that the class of polynomial functors is the right one to look at in order to capture a categorical version of W-types. This will also lead to a proof of Theorem 2.36.

We start by giving some intuition on the structure of a W-type. Note, as we already done for *families*, that we can think of a *morphism* $f : P \to S$ in Set/S as specifying a signature: the codomain S of f represents the set of term-constructors, while the fibre P_s of f above an element $s : S$ represents the arity of the constructor represented by s. The W-type, $\mathrm{W}_f =_{\mathsf{def}} (\mathrm{W}\, x : S)P(x)$, associated to the signature specified by f, corresponds to its free term algebra. We can depict terms in a tree-like structure: each term corresponds to a tree where the nodes are the elements of S while the set of edges departing from a node $s : S$ are labelled by elements $p : P_s$. To build a new term of the set W_f, or, which is the same, a new well-founded tree of the right type, we choose an element $s : S$ and, to each edge $p : P_s$ coming out from that node we stick another term in W_f. This construction builds a new tree $\mathsf{sup}\,(s, f) : \mathrm{W}_f$ for every $s : S$ and for every function $g : P_s \to \mathrm{W}_f$, and it can be easily recognized to correspond to the constructor sup in Rule 2.18 for the W-types W_f.

Now let us come back to initial algebra semantics. As noted in Section 2.2.2, the constructors of an inductive type have their semantical counterpart in

the structure map of the initial algebra. An initial algebra for $\mathcal{P}_f$ in Set is given by the pair $(\mathrm{W}_f, \mathsf{sup} : \mathcal{P}_f(\mathrm{W}_f) \to \mathrm{W}_f)$ where $\mathcal{P}_f\,(\mathrm{W}_f) = (\Sigma\, s : S)(P_s \to \mathrm{W}_f)$. Deriving initiality from the elimination and computation rules in 2.18 is obtained by instantiating these rules in the simple case when the type $(w : \mathrm{W}_f \;\vdash\; Q(w) : \mathsf{Set})$ appearing there does not depend on $w : \mathrm{W}_f$ (we use Q here where we used P in the rules 2.18 in order to avoid confusion with the domain of f which we indicated with P to highlight the similiaries with containers). Since we spell out the details for the inductive type N in Section 2.2.1 we will not repeat the same argument here.

Let us see how to derive the elimination and computation rule from initiality. We express initiality of $(\mathrm{W}_f, \mathsf{sup} : \mathcal{P}_f(\mathrm{W}_f) \to \mathrm{W}_f)$ by means of the fold operator and its properties:

$$\begin{aligned} \mathsf{fold} \quad &: \quad (\mathcal{P}_f(X) \to X) \to W_f \to X \\ \mathsf{fold}(h, (\mathsf{sup}(s, g))) \quad &= \quad h(s, \lambda\, p : P_s.\, \mathsf{fold}(h, g(b))) \end{aligned}$$

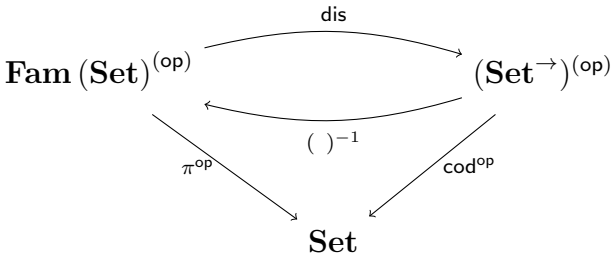
From the premises of the elimination rule we can form the set $(\Sigma\, w : \mathrm{W}_f)Q(w)$ and endow it with a $\mathcal{P}_f$-algebra structure: observe that, by the type-theoretic axiom of choice, we have

$$\begin{aligned} ac : (\Sigma\, s : S)(P_s \to (\Sigma\, w : \mathrm{W}_f)Q(w)) \cong \\ (\Sigma\, s : S)(\Sigma\, m : P_s \to \mathrm{W}_f)(\Pi\, p : P_s)Q(m\, p) \end{aligned}$$

and define the algebra $h' : \mathcal{P}_f((\Sigma\, w : \mathrm{W}_f)Q(w)) \to (\Sigma\, w : \mathrm{W}_f)Q(w)$ to be the composite $h' =_{\mathsf{def}} h \circ ac$, where h is as in the premisses of the elimination rule. By initiality of W_f we get a unique map $\mathsf{fold}(h') : \mathrm{W}_f \to (\Sigma\, w : \mathrm{W}_f)Q(w))$. Notice that $\pi_1 \circ \mathsf{fold}(h')$ has type $(\Pi\, x : \mathrm{W}_f)Q(\pi_0(\mathsf{fold}(h')(x)))$. This is the type of wrec as long as $\mathsf{fold}(h')$ is a section of π_0. This follows from π_0 being a $\mathcal{P}_f$-algebra morphism into the initial algebra sup so that $\pi_0 \circ \mathsf{fold}\, h' = \mathsf{id}_{\mathrm{W}_f}$. Finally, the computation rule is validated since the map $\mathsf{fold}(h')$ is a $\mathcal{P}_f$-algebra morphism from $(\mathrm{W}_f, \mathsf{sup})$.

As already done for **Cont** we could introduce morphisms between polynomials and show that the assignment $X \mapsto \mathcal{P}_f(X)$ extends to a full and faithful functor, but we will end up repeating ourselves. We defer the introduction of the category Poly till Section 2.2.3 where we will describe it as a special case of the category of dependent polynomials.

We conclude by noting that the correspondence between containers and polynomials can be linked to the correspondence between the representation of dependent types as morphisms or as families. This correspondence can be synthetically represented as a fibered equivalence between opposite fibrations, as pictured below



However, this short description relies on the not so short definition of an opposite fibration (see Definition 1.10.11 [64]). The two categories $(\mathbf{Fam}\,\mathbf{Set})^{\mathsf{op}}$ and $(\mathbf{Set}^{\to})^{\mathsf{op}}$ are (morally) obtained respectively from $\mathbf{Fam}\,(\mathbf{Set})$ and $\mathbf{Set}^{\to}$ by inverting vertical morphisms only; the two fibered functors dis and $(\ )^{-1}$ associate, respectively, to a family its display map, and to a map its inverse image:

$$\mathsf{dis}((S,P)) = (\Sigma\, s : S)P(s) \xrightarrow{\pi_0} S \tag{2.1}$$

$$(f : P \to S)^{-1} = (S, \lambda\, s.\, (\Sigma\, p : P) f(p) \equiv_S s) \tag{2.2}$$

Inductive Definitions

We finish this tour on initial algebra semantics of inductive types by presenting a third alternative. We introduce a third approach mainly for pedagogical reason: indeed, this third alternative complements the polynomial/container approach by getting closer to the syntax of induction-recursion we are going to introduce in the next chapter.

The main idea is to *internalize* the notion of inductive type by representing the syntax for describing inductive types itself as an inductive type: we first build inductively a (large) type, ID, of *codes* for inductive definitions. Semantics for these codes is then obtained as before associating to each code a functor. The construction of the type ID can be thought of as inspired

by the universe construction: we build a set of names to represent inductive types and then we map each name to the functor it actually denotes. We now give the formation and introduction rules to form the type ID. Elimination and computation rules do not add any insight and can be easily deduced, so we omit them.

Definition 2.40. The formation rule for ID is the following:

$$\frac{}{\mathsf{ID} : \mathsf{type}}$$

The introduction rules for ID-codes are the following:

(ι)
$$\frac{}{\iota : \mathsf{ID}}$$

(σ)
$$\frac{A : \mathsf{Set} \quad f : A \to \mathsf{ID}}{(\sigma\, A\, f) : \mathsf{ID}}$$

(δ)
$$\frac{B : \mathsf{Set} \quad \varphi : \mathsf{ID}}{(\delta\, B\, \varphi) : \mathsf{ID}}$$

■

Now to each term $\gamma : \mathsf{ID}$ we associate an endofunctor on Set by induction on the structure of the codes. We call a functor which interprets a code, an ID-functor.

Definition 2.41. For $\gamma : \mathsf{ID}$, define the ID-functor $[\![\gamma]\!]_{\mathsf{ID}} : \mathsf{Set} \to \mathsf{Set}$ by induction on the structure of the codes as follows:

(ι) if $\gamma = \iota$ then
$$[\![\iota]\!]_{\mathsf{ID}}\, X = \mathsf{N}_1$$

(σ) if $\gamma = \sigma\, A\, f$ for some $A : \mathsf{Set}$, $f : A \to \mathsf{ID}$ then
$$[\![\sigma\, A\, f]\!]_{\mathsf{ID}}\, X = (\Sigma\, a : A)[\![f(a)]\!]_{\mathsf{ID}}\, X$$

(δ) if $\gamma = \delta\, B\, \varphi$ for some $B : \mathsf{Set}$, $\varphi : \mathsf{ID}$ then

$$[\![\delta\, B\, \varphi]\!]_{\mathsf{ID}}\, X = (B \to X) \times [\![\varphi]\!]_{\mathsf{ID}}\, X$$ ■

By a simple induction on the structure of the code we can show that every ID-functor is an ID code is a container/polynomial functor. Vice versa, as a simple exercise we show that the code for the container represented by family (S, P) (or equivalently by a polynomial functor represented by a map $f : P \to S$) in Set is

$$\sigma\, S(\lambda\, a \mapsto \delta P(s)(\lambda\_ \mapsto \iota))$$

We compute the action on $X : \mathsf{Set}$ of the corresponding functor according to Definition 2.41 as follows:

$$\begin{aligned}
[\![\sigma\, S(\lambda\, s \mapsto \delta P(s)(\lambda_ \mapsto \iota))]\!]_{\mathsf{ID}}\, X &= (\Sigma\, s : S)[\![\delta P(s)(\lambda_ \mapsto \iota)]\!]_{\mathsf{ID}}\, X \\
&= (\Sigma\, s : S)(P(s) \to X) \times [\![\iota]\!]_{\mathsf{ID}}\, X \\
&= (\Sigma\, s : S)(P(s) \to X) \times \mathsf{N}_1 \\
&\cong [\![(S, P)]\!]_{\mathsf{C}}\, X
\end{aligned}$$

As the example above shows, induction on the structure of the code represents an easy and structural method to reason and prove properties on the universe of inductive types. The idea to represent inductive types internally through the introduction of the type ID has proved particularly successful in the context of generic functional programming [17] whose central idea is to define generic functions by induction on the definition of a data type in order to profit in terms of reusability and adaptiveness of the programs. An example of this elegant way to program is represented by the work of Dagand [31, 32] on ornaments which uses a close axiomatization to the one presented here.

2.2.3 Initial algebra semantics of indexed inductive types

As in the unindexed case we want to characterize the class of functors whose initial algebras are indexed inductive types. Since indices are an integral part of these types we are led to apply the initial algebra semantics machinery to more sophisticated categories where indices can be naturally taken into account. The natural way is to consider indices and indexed types as members

of the same category of types and therefore to consider the natural indexing which a category possess over itself by means of the slice construction.
To give a simple example of how initial algebra semantics can be used to model indexed inductive types in slice categories consider the type Lam and fin as seen in Example 1.4 and 1.3 respectively. We can represent the rule in a natural deduction style as follows:

Example 2.42 (Example 1.3 continued)**.** The type of finite sets is represented by a N-indexed data type, $\mathsf{Fin} : \mathsf{Nat} \to \mathsf{Set}$ with the following introduction rules

$$\frac{}{\mathsf{zero} : \mathsf{fin}(\mathsf{s}(n))} \qquad \frac{x : \mathsf{fin}(n)}{\mathsf{succ}(x) : \mathsf{fin}(\mathsf{s}(n))} \quad \blacksquare$$

Example 2.43 (Example 1.4 continued)**.** The data type $\mathsf{Lam} : \mathsf{Nat} \to \mathsf{Set}$ is a N-indexed data type with introduction rules:

$$\frac{x : \mathsf{fin}(n)}{\mathsf{Var}(x) : \mathsf{Lam}(n)} \qquad \frac{t : \mathsf{Lam}(n) \quad u : \mathsf{Lam}(n)}{\mathsf{App}(t, u) : \mathsf{Lam}(n)} \qquad \frac{t : \mathsf{Lam}(\mathsf{s}(n))}{\mathsf{Abs}(t) : \mathsf{Lam}(n)} \quad \blacksquare$$

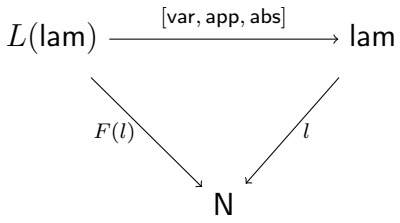
Note that in order to define $\mathsf{Lam}(n)$ we cannot use just those terms of $\mathsf{Lam}(n)$ that have already been constructed. Indeed, λ-abstraction make use of elements of type $\mathsf{Lam}(\mathsf{s}(n))$. Therefore we have to build the family of types Lam simultaneously. Since this is a N-indexed type the functor will have type $L : \mathsf{Set}/\mathsf{N} \to \mathsf{Set}/\mathsf{N}$. Each N-indexed type is interpreted as a morphism $f : X \to \mathsf{N}$ which we might conveniently represent fiberwise as $(X_n)_{n : \mathsf{N}}$. For example, letting $\mathsf{lam} =_{\mathsf{def}} (\Sigma\, n : \mathsf{N})\mathsf{Lam}(n)$ we can represent Lam by its first projection $l =_{\mathsf{def}} \pi_0 : \mathsf{lam} \to \mathsf{N}$ whose fibre at n, lam_n, is the set $\mathsf{Lam}(n)$ of λ-terms with n free variables.

Example 2.44 (Example 1.4 continued)**.** Given an object $X \xrightarrow{f} \mathsf{N}$ of $\mathbf{Set}/\mathsf{N}$, the map l representing Lam arises as the carrier of an initial algebra for the functor whose action on the fiber X_n is given by

$$L(X_n) = \mathsf{Fin}_n + X_n \times X_n + X_{n+1}$$

where $\mathsf{Fin}_n = (\Sigma\, x : (\Sigma\, m : \mathsf{N})(\mathsf{fin}(m))(\pi_0(x) \equiv_{\mathsf{N}} n)$. As before, note how the structure map of the initial algebra interprets the constructors of Lam:

an initial algebra for L is a pair $(l : \mathsf{lam} \to \mathsf{N}, \mathsf{intro}_L)$ where the structure map $\mathsf{intro}_L : L(l) \to l$ can be represented by a N-indexed family of functions $(\mathsf{intro}_L)_n : \mathsf{Fin}_n + \mathsf{lam}_n \times \mathsf{lam}_n + \mathsf{lam}_{n+1} \longrightarrow \mathsf{lam}_n$. Each of these maps is equivalent to a triple of maps $\mathsf{var}_n : \mathsf{Fin}_n \to \mathsf{lam}_n$, $\mathsf{app}_n : \mathsf{lam}_n \times \mathsf{lam}_n \to \mathsf{lam}_n$ and $\mathsf{abs}_n : \mathsf{lam}_{n+1} \to \mathsf{lam}_n$ which interpret respectively the three constructors Var, App and Abs of the indexed type Lam. The initial algebra (l, intro_L) can then be displayed as follows:



■

We now present a generalization of polynomials and their semantics which takes into account indices. Similarly we show how the family approach to dependent types informs the generalization from containers to indexed containers. We will see in Chapter 4 how to generalize the coding scheme for ID.

Dependent polynomial functors

Exactly as it happens in algebra with the polynomial ring $\mathcal{K}[X]$ over a field $\mathcal{K}$, once the notion of polynomial function in one variable has been grounded, one is naturally led to consider the polynomial ring $\mathcal{K}[X_1, X_2, \ldots, X_n]$ of polynomials functions in many variables. In the same vein dependent polynomials arise as a natural generalization of polynomials. In the context of category theory the step from one to many variables is performed by the operation of slicing. This means that we do not confine our attention to endofunctor defined on a lccc $\mathbb{C}$ but we allow for more general functors between slices of $\mathbb{C}$. This simple generalization allows us to take into account different form of indices. The more general definition of dependent polynomial albeit a simple generalization from one to many variables revealed itself as a natural notion, which have applications in different branches of mathematics [47].

Definition 2.45 (Dependent polynomial)**.** A dependent polynomial in a lccc $\mathbb{C}$ is given by a triple of maps $F = (r, t, q)$ in $\mathbb{C}$ which can be arranged in the following diagram

$$I \xleftarrow{\ r\ } P \xrightarrow{\ t\ } S \xrightarrow{\ q\ } O$$

To each of these triple $F = (r, t, q)$ we associate a functor, $\mathcal{P}_F$, called the extension of F or the dependent polynomial functor associated to F and defined as the composite

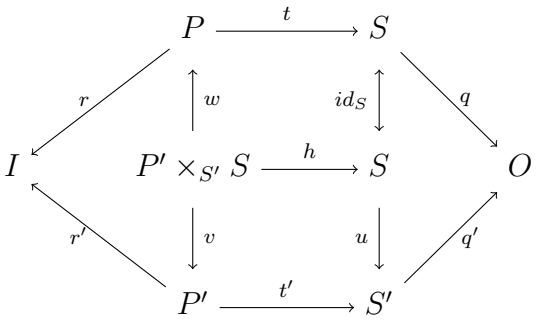
$$\mathbb{C}/I \xrightarrow{\ \Delta_r\ } \mathbb{C}/P \xrightarrow{\ \Pi_t\ } \mathbb{C}/S \xrightarrow{\ \Sigma_q\ } \mathbb{C}/O$$

In the internal logic of $\mathbb{C}$ the action of the functor $\mathcal{P}_F = \Sigma_q \circ \Pi_t \circ \Delta_r$ on an object $(X \xrightarrow{f} I) = (X_i)_{i:I}$ is given by

$$\big(\Sigma\, s : S_o)(\Pi\, p : P_s) X_{r(p)}\big)_{o:O}$$ ■

Dependent polynomials with fixed indices I and O form a category. Below, we recall the definition of the morphisms in this category. This gives the category $\mathsf{Poly}(I, O)$ of dependent polynomials and morphism between them.

Definition 2.46. A morphism between dependent polynomials $F = (r, t, q)$ and $G = (r', t', q')$ is given by a diagram of the form



where the bottom square is a pullback of u and t'. Thus a morphism $(r, t, q) \to (r', t', q')$ amounts to giving $u : S \to S'$ with $q = q' \circ u$ and $w : P' \times_{S'} S \to P$ with $r \circ w = r' \circ v$. ■

Since dependent polynomials are interpreted as functors we expect morphisms between dependent polynomials to be interpreted as natural transformation between the corresponding extensions. This is indeed the case:

Theorem 2.47 ([47] Theorem 2.12)**.** Given dependent polynomials $F = (r, t, q)$ and $G = (r', t', q')$, every strong natural transformation $\mathcal{P}_F \xrightarrow{\cdot} P_G$ is represented in an essentially unique way by a commuting diagram as pictured in definition 2.46.

This theorem ensures that the assignment $F \mapsto \mathcal{P}_F$ extends to a full and faithful functor which is also bijective on objects. If we indicate with $\mathsf{PolyFun}(I, O)$ the full subcategory of $[\mathbb{C}/I, \mathbb{C}/O]$ whose objects are dependent polynomial functors and whose morphisms are strong natural transformation between them, we obtain the following:

Corollary 2.48 (Representation)**.** Let I and O be objects of a lccc $\mathbb{C}$. Then, the categories $\mathsf{Poly}(I, O)$ and $\mathsf{PolyFun}(I, O)$ are equivalent.

The relevance of this theorem is twofold: it ensures that the syntactic representation given by diagrams is correct, and it enables the use of diagrammatic arguments when reasoning about polynomial functors.
The theory of polynomial and polynomials functors which we developed in Section 2.2.2 now reduce to the special case of the theory of dependent polynomials where the indices I and O are both chosen to be the terminal object of $\mathbb{C}$.
Dependent polynomials have several interesting closure properties. Indeed they are closed under sums, products, composition, parametrised initial algebras, and differentiation. In Chapter 4 we use binary products and sums of a family of dependent polynomials, therefore we now recall them. The sum of a K-indexed family of dependent polynomials $\{F_k = (r^k, t^k, q^k) \mid k \in K\}$, for an arbitrary set K, is the dependent polynomial $(\Sigma\, k : K)F_k$ given by the following diagram

$$I \xleftarrow{[r_k]_{k\in K}} (\Sigma\, k : K)P_k \xrightarrow{(\Sigma\, k : K)t_k} (\Sigma\, k : K)S_k \xrightarrow{[q_k]_{k\in K}} O$$

where $(\Sigma\, k : K)t_k = [\mathsf{in}_k \circ t^k]_{k \in K}$. From this definition it follows $\mathcal{P}_{(\Sigma\, k:K)F_k} \cong (\Sigma\, k : K)\mathcal{P}_{F_k}$.
Assuming that a lccc $\mathbb{C}$ has initial algebras for polynomial functor, Gambino and Hyland, generalizing a result in [77], have shown that every dependent polynomial in $\mathsf{Poly}(I, I)$ has an initial algebra (Theorem 14 [46]). Initial algebras for dependent polynomial functors are categorical counterpart of Peterson-Synek's general tree types (cf. Chapter 16 in [80]). If we think a W-type as the free term algebra for a single sort signature, then the general trees are a generalization of W-types to account for multi-sorted signature: S is still the set of constructors and P_s the arity of the constructors s, but now we have also the ability to distinguish between different input and output sorts thanks to the indices I and O. A term in the tree is built according to the following rule: nodes are labelled by elements $s : S$ and an index $q(s) : O$ representing the output sort; edges above a node $s : S$ are given by elements $p : P_s$. Such an element $p : P_s$ is likewise indexed by $r(p) : I$. This index represents an input sort for the constructror $s : S$.

Indexed containers

As dependent polynomials generalize the notion of polynomial, the notion of indexed container [79, 14] is the natural extension of containers into the realm of indexed data types. Again, the only difference with dependent polynomials consists in an explicit use of families rather than morphisms to model dependent types, thus avoiding the use of identity types which is intrinsic when reasoning fiberwise.

Definition 2.49 (Indexed container)**.** An indexed container in Set with input indices $I : \mathsf{Set}$ and output indices $O : \mathsf{Set}$ consists of a triple (S, P, n) where

$$\begin{aligned} &S : O \to \mathsf{Set} \\ &P : (\Pi\, o : O)(S(o) \to \mathsf{Set}) \\ &n : (\Pi\, o : O)(\Pi\, s : S(o))(P(o, s) \to I) \end{aligned}$$

■

The metaphor of shapes and positions still informs the idea of an indexed container. Note, however, that in this case, both shapes and positions come equipped with indices: each shape $s : S(o)$, is equipped with an extra index

$o : O$. And for such a shape $s : S(o)$ we have a corresponding set of positions $P(o, s)$ where storing data which, likewise, come equipped with its own index $i : I$ through the function n.
We are now led to recall the notion of morphism between indexed containers which corresponding to the type-theoretic data for a morphism in $\mathsf{Poly}(I, O)$.

Definition 2.50. Given $I, O : \mathsf{Set}$, a morphism between indexed containers from (S, P, n) to (S', P', n') is given by:

- a shape function : $u : (\Pi\, o : O)(S(o) \to S'(o))$;
- a position function: $w : (\Pi\, o : O)(\Pi\, s : S(o))(P'(o, (u(o, s))) \to P(o, s)$;

  such that a coherent condition on input holds: for every $o : O, s : S(o)$ and $p : P'(o, (u(o, s)))$ we have $n(o, s, (w(o, s, p))) = n'(o, (u(o, s)), p)$

■

Indexed containers of input indices I and output indices O form a category $\mathsf{IC}(I, O)$. We write with bold fonts $\mathbf{IC}(I, O)$ the corresponding notion of an indexed container defined in **Set**. As it happens for polynomials and dependent polynomials, the theory of containers presented in Section 2.2.2 is subsumed by that of indexed containers: containers are just those indexed containers where the indices I and O are the singleton set N_1. The category Cont introduced in Section 2.2.2 is then the category $\mathsf{IC}(\mathsf{N}_1, \mathsf{N}_1)$.
The extension of an indexed container is defined as that of a dependent polynomial, but it is more naturally given on indexed sets (or proof relevant predicates).

Definition 2.51. The extension of an indexed container (S, P, n) is a functor

$$[\![(S, P, n)]\!]_{\mathsf{IC}} : \mathsf{Set}^{\,I} \longrightarrow \mathsf{Set}^{\,O}$$

whose action on an I-indexed set $X : I \to \mathsf{Set}$ and output $o : O$ is given by the set

$$[\![(S, P, n)]\!]_{\mathsf{IC}}\, X\, o =_{\mathsf{def}} (\Sigma\, s : S(o))(\Pi\, p : P(o, s))X(n(o, s, p))$$ ■

Chapter 3

Induction-recursion

Abstract In this chapter we present the theory of induction-recursion as developed by Dybjer and Setzer in [38, 39, 40]. This theory extends the theory of inductive and indexed inductive definitions as seen in the previous chapter. We begin in Section 3.1 by looking at the notion of a universe as the paradigmatic example of an inductive-recursive type. We then give an axiomatic presentation of the theory of induction-recursion by building the syntax and the semantics for inductive-recursive types in Section 3.2. Finally we build a model within classical set theory augmented with a strong infinity axiom in Section 3.3.

3.1 Universes

The term *universe* is pervasive in mathematics since it is linked to the idea of *universe of discourse*. Boole [20] explains this use of the term in the following passage:

> Now, whatever may be the extent of the field within which all the objects of our discourse are found, that field may properly be termed the universe of discourse.

Informally universes are *universes of discourse*: collections of entities over which certain variables of interest in some formal system may range; turning this informal description – that can easily lead to circular arguments – into

a formal one is something that any foundational theory had to deal with. In this section we recall different, but closely related notions of universes, which arose in the area of set theory and category theory. A common theme underling these notions of universe is the distinction between *small* or *closed* entities and *large* or *open* ones. Without any ambition of exhaustiveness, we aim to give a gentle introduction to what can be considered the paradigmatic example of an inductive-recursive definition: that is the idea of a type theoretic universe.

3.1.1 Universes in set theory

In the theory of ZFC the set-theoretic universe, $\mathrm{V} =_{\mathsf{def}} \{x \mid x = x\}$ (compare Section 2.3.1 in [45] or Section I.9 in [67] for this notation), the *universe of discourse*, is described by axioms which are meant to characterize it. To give an example consider the foundation axiom: it expresses the requirement that every set in V is well-founded, i.e. no set has an infinite descending membership sequence[1]. Via the cumulative hierarchy we can build the universe of well-founded sets and express the axiom of foundation by requiring the two universes to coincide:

Definition 3.1 (cumulative hierarchy)**.** The cumulative hierarchy is defined by transfinite recursion as follows

$$\begin{aligned} &\mathrm{V}_0 = \emptyset \\ &\mathrm{V}_{\alpha+1} = \wp(\mathrm{V}_\alpha) \\ &\mathrm{V}_\lambda = \textstyle\bigcup_{\beta<\lambda} \mathrm{V}_\beta \quad \text{for } \lambda \text{ limit.} \end{aligned}$$ ■

By transfinite induction we can show that $\mathrm{V}_\alpha = \bigcup_{\beta<\alpha} \mathrm{V}_\beta$. The universe WF is defined as $\mathrm{WF} =_{\mathsf{def}} \bigcup_\alpha \mathrm{V}_\alpha$. The axiom of foundation is then equivalent to the statement $\mathrm{V} = \mathrm{WF}$ (cf. [67], Theorem 4.1). In a similar way we can define by transfinite recursion the universe of constructible sets L. Let $\mathrm{L}_0 = \emptyset$, $\mathrm{L}_{\alpha+1} = \mathrm{Def}(\mathrm{L}_\alpha)$ and $\mathrm{L}_\lambda = \bigcup_{\beta<\lambda} \mathrm{L}_\beta$ for λ a limit ordinal, where $\mathrm{Def}(X)$ denote the set of *definable* subset of X, i.e those subset of X definable by a property whose parameters and quantifiers range over X (for a definition

[1] For set theoretic universes with anti-foundation axioms see for example Aczel [8].

of *definability* see e.g. Chapter V in [67]). Then $\mathrm{L} =_{\mathsf{def}} \bigcup_\alpha \mathrm{L}_\alpha$ and the axiom of constructibility corresponds to the statement $\mathrm{V} = \mathrm{L}$.
However neither WF nor L can be asserted to be sets if we do not want to run into paradoxes. They are indeed proper *classes*. These universes are built up in stages indexed over the class ON of ordinal numbers and represent a view of the universe V as open-ended and under-determined by the set theoretic axioms. Already in Zermelo's work, initial segments of WF were proposed as models for the axioms of set theory. Zermelo [93] advocates a *dynamic* view of sets that posits an endless succession of models of set theory:

> "The two polar opposite tendencies of the thinking spirit, the idea of creative *advance* and that of collection and *completion*, ideas which also lie behind the Kantian 'antinomies', find their symbolic representation and symbolic reconciliation in the transfinite number series based on the concept of well-ordering. This series reaches no true completion in its unrestricted advance, but possesses only relative stopping-points, just those 'boundary numbers' which separate the higher model types from the lower."

The relative stopping-point Zermelo is referring to are the inaccessible cardinals (see Definition 3.29): indeed, if κ is an inaccessible cardinal, then V_κ is a model of ZFC ([66], Proposition 1.2). This view of regarding inaccessible cardinals as indices of the cumulative hierarchy corresponding to universes is intimately tied in with the notion of a *Grothendieck universe* —a transitive set U closed under pairs, powerset, and union of a family of elements of U indexed by an element of U. Indeed, if U is as above with the additional requirement $\omega \in U$ then the existence of a Grothendieck universe is equivalent to the existence of an inaccessible cardinal. Clearly the existence of an inaccessible cardinal cannot be proved inside ZFC otherwise ZFC would prove its own consistency contradicting Gödel's second incompleteness theorem. Thus, the strong axiom of infinity, asserting the existence of an inaccessible cardinal κ, is a genuine strengthening of ZFC, and can be considered the very first step into the *higher infinity*, represented by the realm of large cardinals. These axioms enable us to *reflect* properties of the *open* universe V into its initial segments, that are then regarded as *closed* universes. One of the pi-

oneers in this area, Paul Mahlo, investigated hierarchies of regular cardinals formulated in terms of higher fixed point phenomena. Its universes will play a pivotal role in the construction of the set theoretic model for the theory of IR.

3.1.2 Universes in category theory

In category theory sizes do matter: the distinction between small and large categories represents an important dichotomy raised at the very beginning of any introduction to the subject. It is discussed for example by Eilenberg and Mac Lane in their seminal work which dates the birth of the field ([41] page 246). Let **Set**[V] be the category whose objects and morphisms are the sets and the functions in V. The category **Set**[V] is inherently large, as any other category arising from classes. But the notion of *smallness* can also be regarded as a relative one: for example if we let **Set**[V_κ] to be the category whose objects and morphisms are sets and function in V_κ for κ an inaccessible cardinal, then we could call small any category whose collection of morphisms is an object in **Set**[V_κ]. This is similar to the approach followed by Bourbaki who introduced the notion of Grothendieck universe to accommodate larger and larger categories in a set-theoretical framework. From this idea stems the so called Grothendieck's axiom, asserting that each set is contained in a universe. Notice that, according to the axioms we want to validate, and the number of constructions we want our universe to be closed under, different levels of the cumulative hierarchy can be used: for example, V_ω models all the axioms of ZFC apart from infinity, and $V_{2\omega}$ models all the axioms of ZFC apart from replacement.

A well-studied and pivotal notion in category theory is that of a topos: it offers, among other perspectives, the idea of a generalized universe of sets. This way to regard a topos is attributed to Lawvere and it is validated by the internal language associated to a topos [19]. Joyal and Moerdjik [65] realized that an Heyting pretopos with a natural numbers object endowed with a situable class of small maps could serve the purpose of a categorical framework for both classical and intuitionistic set-theoretical universes. Their work initiated the field of *Algebraic Set Theory* (AST for short). Heyting preto-

poses and their associated internal language have been further investigated by Maietti [70, 71] as a possible candidate for the notion of a categorical universe. More recently Moerdijk and Palmgren [78] have tried to deepen the connection between AST and predicative systems by proposing ΠW-stratified pseudotoposes as categorical universes apt to model Martin-Löf type theory and CZF. Finally, by using a class of small maps analogous to that introduced in AST, Streicher suggested a notion corresponding to that of a Groethendick universe inside a topos [91].

3.1.3 Universes in type theory

We saw in Section 1.2 that inductive types can be thought of as built from below as the union of stages in a process determined by some specific set of rules (constructors). The idea behind a universe type is to *reflect* the construction of inductive types: it encodes the process of construction of inductive types by coding their rules (constructors). This meta-process allows us to construct a type of types closed under certain type constructors introduced at some earlier stage.

Universe types were introduced by Martin-Löf [74] to strengthen the language of intuitionistic type theory and they soon became integral part of it [73]. To understand how fundamental universes are in type theory it suffices to recall Smith's proof [89] that Martin-Löf type theory without universes cannot prove $\neg(a =_A b)$ for any type A. In particular, this leads to the unprovability of Peano's fourth axiom, asserting that for any natural number $n : \mathsf{N}$ we have $\neg(\mathsf{s}(n) = 0)$. Following the proposition-as-types paradigm, we can understand the notion of a universe both as a collection of types closed under certain operations, and as a set of constructively given infinitary formulas. This latter meaning of universes has been used for example by Palmgren [84] to give a modified realizability interpretation of infinitary constructive logic in Martin-Löf type theory. However, here we are mainly concerned with the former of these meanings, namely a universe as a type of types closed under certain type constructors.

Being a collection of types, a type universe can be presented essentially in two ways:

(i) à la Russell, where we do not distinguish between *denotans* and *denotatum*, i.e. we identify the name of a set with the collection of its elements. The formation rule for a universe U is:

$$\frac{}{\mathsf{U} : \mathsf{Set}} \qquad \frac{X : \mathsf{U}}{X : \mathsf{Set}}$$

(ii) à la Tarski, so called for the similarity with the definition of truth given by Tarski: a universe is represented by a *family* of sets and we clearly distinguish between the name of a set $u : \mathsf{U}$ and the actual collection $\mathsf{T}(u)$ of elements it denotes. We then have the following formation rule:

$$\frac{}{\mathsf{U} : \mathsf{Set}} \qquad \frac{x : \mathsf{U}}{\mathsf{T}(x) : \mathsf{Set}}$$

In what follows, we will always adopt formulations à la Tarski, regarding the former as an informal version of the latter. Universes can then be presented as families (U, T) where U is a set of *codes*, and $\mathsf{T} : \mathsf{U} \to \mathsf{Set}$ is a *decoding function*, that assigns its extension to every code; or, in other words, we can think of U as a set of names and T as a map assigning to every name $u : \mathsf{U}$ the set $\mathsf{T}(u)$ it denotes.

We specify that a universe contains certain ground types and reflects certain type constructors via its introduction rules: these describe how canonical elements of the universe are built. We introduce in U codes for ground types and codes reflecting types constructors. Then T decodes them accordingly: codes for ground types are mapped to their values and codes for type constructors applied to other codes are mapped to the value of the reflected constructors applied to the corresponding types. We illustrate this by means of a concrete example.

Example 3.2 (regular universes)**.** We spell out introduction rules for a universe $(\mathsf{U}_\Sigma(A, B), \mathsf{T}_\Sigma(A, B))$ containing codes for a specific collection of sets $B(a)$, for $a : A$ and closed under Σ-types. To increase readability in the following rules we refer to $(\mathsf{U}_\Sigma(A, B), \mathsf{T}_\Sigma(A, B))$ simpler as (U, T).

Rule 3.3 (Regular universe)**.**

$$\frac{a : A}{\widehat{B(a)} : \mathsf{U}}\ \widehat{B(a)}\,intro \qquad \frac{\hat{a} : \mathsf{U}}{\mathsf{T}(\widehat{B(a)}) = B(a) : \mathsf{Set}}\ \widehat{B(a)}\ reflection$$

$$\frac{a : \mathsf{U} \quad b : \mathsf{T}(a) \to \mathsf{U}}{\hat{\Sigma}\, a\, b : \mathsf{U}}\ \hat{\Sigma}\,intro \qquad \frac{a : \mathsf{U} \quad b : \mathsf{T}(a) \to \mathsf{U}}{\mathsf{T}\,(\hat{\Sigma}\, a\, b) = (\Sigma\, \mathsf{T}(a))(\mathsf{T} \circ b) : \mathsf{Set}}\ \hat{\Sigma}\,reflection$$

■

A concrete example is given by a universe containing a code for natural numbers and closed under Σ-types: we simply choose (A, B) to be $(\mathsf{N}_1, B_\mathsf{N})$ where $B_\mathsf{N} : \mathsf{N}_1 \to \mathsf{Set}$ is given by $B_\mathsf{N}(\mathsf{0}_1) = \mathsf{N}$. ■

Universes described in Example 3.2 have been successfully used by Palmgren [85] to define inductively generated formal topologies. They are called *regular universes* since they represent the categorical counterpart of the notion of regular cardinal in classical set theory and of regular set in constructive set theory.

A universe can be closed not only under Σ-types but possibly also under other type operators. Indeed, if we want to close our universe simultaneously under other type constructors like +, Π or W, we simply add introduction rules reflecting these constructors into U.

Rule 3.4 (Closure under type operators)**.**

$$\frac{a : \mathsf{U} \quad a' : \mathsf{U}}{a \mathbin{\hat{+}} a' : \mathsf{U}}\ \hat{+}\,intro \qquad \frac{a : \mathsf{U} \quad a' : \mathsf{U}}{\mathsf{T}\,(a \mathbin{\hat{+}} a') = \mathsf{T}(a) + \mathsf{T}(a') : \mathsf{Set}}\ \hat{+}\,reflection$$

$$\frac{a : \mathsf{U} \quad b : \mathsf{T}(a) \to \mathsf{U}}{\hat{\Pi}\, a\, b : \mathsf{U}}\ \hat{\Pi}\,intro \qquad \frac{a : \mathsf{U} \quad b : \mathsf{T}(a) \to \mathsf{U}}{\mathsf{T}\,(\hat{\Pi}\, a\, b) = (\Pi\, \mathsf{T}\, a)(\mathsf{T}\, b) : \mathsf{Set}}\ \hat{\Pi}\,reflection$$

$$\frac{a : \mathsf{U} \quad b : \mathsf{T}(a) \to \mathsf{U}}{\hat{\mathrm{W}}\, a\, b : \mathsf{U}}\ \hat{\mathrm{W}}\,intro \qquad \frac{a : \mathsf{U} \quad b : \mathsf{T}(a) \to \mathsf{U}}{\mathsf{T}\,(\hat{\mathrm{W}}\, a\, b) = (\mathrm{W}\ \mathsf{T}(a))\mathsf{T}(b) : \mathsf{Set}}\ \hat{\mathrm{W}}\,reflection$$

■

Notice the difference between the reflection rules for closing a universe under a dependent type operator like Σ, Π and W, and the reflection rules for

a non-dependent type operator like $+$: in the former we always have the recursive function T appearing in the premisses of the rules, while this is not the case for the latter.

Example 3.5 (finite universes)**.** We define the first three finite universes. Notice that in order to define these universes we need large elimination for the sets $\mathsf{N}_0, \mathsf{N}_1$ and N_2, since otherwise case_i would not allow us to define a recursive function with a large codomain.

- The empty universe $(\mathsf{U}_0, \mathsf{T}_0)$, where $U_0 = \mathsf{N}_0$ and $\mathsf{T}_0 \colon \mathsf{N}_0 \to \mathsf{Set}$ is the unique map given by $\lambda\, x.\, \mathsf{case}_0(x)$.
- The singleton universe $(\mathsf{U}_1, \mathsf{T}_1)$, where $U_1 = \mathsf{N}_1\ \mathsf{T}_{\mathsf{N}_1} : \mathsf{N}_1 \to \mathsf{Set}$ is given by $\lambda\, a.\, \mathsf{case}_1(a, \mathsf{N}_1)$
- The boolean universe $(\mathsf{U}_2, \mathsf{T}_2)$ where $\mathsf{T}_2 : \mathsf{N}_2 \to \mathsf{Set}$ is the map defined by $\lambda\, a.\, \mathsf{case}_2(a, \mathsf{N}_0, \mathsf{N}_1)$. This is the universe used by Smith [89] to prove independence of Peano's fourth axiom from ML.

We can build a *finitary* universe by choosing our set of codes U to contain codes for the sets $\mathsf{N}_0, \mathsf{N}_1$ and N_2 which decodes accordingly, and adding rules to close this universe under Σ and Π-types as in Rules 3.3 and 3.4. ■

A noteworthy example of the use of universes in constructive mathematics is represented by Aczel's interpretation of CZF in Martin-Löf type theory, dubbed as the *set-as-trees* interpretation (see e.g. [9]). The basic ingredient of this interpretation is the construction of a type of *iterative sets* V as defined in the following example.

Example 3.6 (The type of iterative sets)**.** Let (U, T) be a type universe containing codes for $\mathsf{0}$, N and closed under $+$, Σ, Π and identity types Id. The type of iterative sets V is then $V =_{\mathsf{def}} (\mathrm{W}\, u : \mathsf{U})\mathsf{T}(u)$. This definition captures the inductive rule generating a universe of iterative sets ([7], page 22): "If A is a *set of* iterative sets then A is an iterative set". V represents a type theoretic version of the cumulative hierarchy of sets; we can use V to encode in type theory membership diagrams as well-orderings which branch over sets in a universe. ■

A universe, as it has been described so far by these example, is the smallest set closed under certain specified set-forming operations. This suggests that universes are similar to inductive types and that it should be possible to capture the construction of the universes as a process itself and not just treat it as a meta-process.

Hierarchy of universes and Super universes

Once we have built a universe (U, T) reflecting certain ground types and type constructors, we can imagine to iterate this process: instead of building a single universe, we can imagine having a tower of them built one above the other. Thus, above a universe $(\mathsf{U}_0, \mathsf{T}_0)$, we build another universe $(\mathsf{U}_1, \mathsf{T}_1)$ closed under the same type constructors and whose elements are all the sets in $(\mathsf{U}_0, \mathsf{T}_0)$ but also $(\mathsf{U}_0, \mathsf{T}_0)$ itself; above $(\mathsf{U}_1, \mathsf{T}_1)$ we build another universe $(\mathsf{U}_2, \mathsf{T}_2)$ in the same way, and so on.
This process allows us to see universes as uniform constructions as explained by Palmgren [83]: assuming we have already built a universe $(\mathsf{U}_n, \mathsf{T}_n)$, then we add rules to build the next universe $(\mathsf{U}_{n+1}, \mathsf{T}_{n+1})$.

$$\frac{}{\mathsf{U}_{n+1} : \mathsf{Set}} \qquad \frac{x : \mathsf{U}_{n+1}}{\mathsf{T}_{n+1}(x) : \mathsf{Set}}$$

We specify in the introduction rules that $(\mathsf{U}_{n+1}, \mathsf{T}_{n+1})$ contains as elements all the sets in $(\mathsf{U}_n, \mathsf{T}_n)$ and also $(\mathsf{U}_n, \mathsf{T}_n)$ itself:

$$\frac{}{u_n : \mathsf{U}_{n+1}}\ \mathsf{U}_n\ intro \qquad \mathsf{T}_{n+1}(u_n) = \mathsf{U}_n\ \ \mathsf{U}_n\ reflection$$

$$\frac{a : \mathsf{U}_n}{t_n\, a : \mathsf{U}_{n+1}}\ \mathsf{U}_n{-}el\ intro \qquad \frac{a : \mathsf{U}_n}{\mathsf{T}_{n+1}(t_n\, a) = \mathsf{T}_n(a)}\ \mathsf{U}_n{-}el\ reflection$$

Then we add rules specifying that $(\mathsf{U}_{n+1}, \mathsf{T}_{n+1})$ is closed under the same type operators which were already reflected in $(\mathsf{U}_n, \mathsf{T}_n)$: these rules are formulated as in Rule 3.3 and 3.4.
The construction of $(\mathsf{U}_{n+1}, \mathsf{T}_{n+1})$ depends on the *family* $(\mathsf{U}_n, \mathsf{T}_n)$ only. To express that $(\mathsf{U}_{n+1}, \mathsf{T}_{n+1})$ is built above the previous universe we coded the family $(\mathsf{U}_n, \mathsf{T}_n)$ in $(\mathsf{U}_{n+1}, \mathsf{T}_{n+1})$: the process described is the same as in Example 3.2 where we built a universe closed by Σ and containing codes for a

specific collection of sets $B(a)$ where $a : A$. Notice, however, that this time we do want to encode the entire family $(\mathsf{U}_n, \mathsf{T}_n)$: consequently we have also added a code for the index set of the family, U_n.
This process can be seen in itself as an operator taking a family (A, B) as input and giving back the universe $(\mathbf{U}(A, B), \mathbf{T}(A, B))$ above it as output. The universe operator has formation rules given by

$$\frac{A : \mathsf{Set} \quad B : A \to \mathsf{Set}}{\mathbf{U}(A, B) : \mathsf{Set}} \qquad \frac{x : \mathbf{U}(A, B)}{\mathbf{T}(A, B, x) : \mathsf{Set}}$$

and introduction rules given by

$$\frac{}{\widehat{A} : \mathbf{U}(A, B)}\ \widehat{A}\ intro \qquad \mathbf{T}(A, B, \widehat{A}) = A\ \ \widehat{A}\ reflection$$

$$\frac{a : A}{t\,a : \mathbf{U}(A, B)}\ \widehat{B(a)}\ intro \qquad \frac{a : A}{\mathbf{T}(A, B, t\,a) = B(a)}\ \widehat{B(a)}\ reflection$$

In addition, we can add rules specifying that the universe $(\mathbf{U}(A, B), \mathbf{T}(A, B))$ is closed under certain type constructors. The hierarchy of universes described above can be recovered as a recursive definition using $(\mathsf{U}_0, \mathsf{T}_0)$ as initial family and letting $U_{n+1} =_{\mathsf{def}} \mathbf{U}(\mathsf{U}_n, \mathsf{T}_n)$ and $\mathsf{T}_{n+1} =_{\mathsf{def}} \mathbf{T}(\mathsf{U}_n, \mathsf{T}_n)$.

The super universe

The analysis of universes as a uniform construction and the formulation of a universe operator $(\mathbf{U}(\_, \_), \mathbf{T}(\_, \_))$ have set the ground to investigate a more powerful notion of universes. Indeed the universe operator is now just an operator acting on families: we can imagine to build a universe closed not only by the usual type constructors but also under the universe operator. Palmgren [83] formalizes this intuition in the definition of a super universe. Let (V, S) denotes the family of sets representing the super universe. We express the essential property of closure under the universe operator by adding codes for the universe operator. This operator does not take arbitrary families as input but *internal* families, i.e. families which are coded in the super universe. An internal family in (V, S) is a pair (v, s) where $v : \mathsf{V}$ and $s : \mathsf{S}(v) \to \mathsf{V}$. Notice that we have already used internal families when specifying introduction rules to reflect dependent type constructors like Σ, Π and W: in those

cases we considered families (a, b), with $a : \mathsf{U}$ and $b : \mathsf{T}(u) \to \mathsf{U}$, internal to the universe (U, T) we were defining.
The introduction rules to reflect the type universe operator $(\mathbf{U}(\_, \_), \mathbf{T}(\_, \_))$ in (V, S) are:

$$\frac{v : \mathsf{V} \quad s : \mathsf{S}(v) \to \mathsf{V}}{u(v, s) : \mathsf{V}}\ \widehat{\mathbf{U}(v,s)}\, intro$$

$$\frac{v : \mathsf{V} \quad s : \mathsf{S}(v) \to \mathsf{V}}{\mathsf{S}(u(v, s)) = \mathbf{U}(\mathsf{S}(v), \lambda\, x.\, \mathsf{S}(s(x)))}\ \widehat{\mathbf{U}(v,s)}\, reflection$$

$$\frac{v : \mathsf{V} \quad s : \mathsf{S}(v) \to \mathsf{V} \quad z : \mathsf{S}(u(v, s))}{\mathsf{el}(v, s, z) : \mathsf{V}}\ \widehat{\mathbf{U}(v,s)}{-}el\, intro$$

$$\frac{v : \mathsf{V} \quad s : \mathsf{S}(v) \to \mathsf{V} \quad z : \mathsf{S}(u(v, s))}{\mathsf{S}(\mathsf{el}(v, s, z)) = \mathbf{T}(\mathsf{S}(v), \lambda\, x.\, \mathsf{S}(s\, x), z)}\ \widehat{U(v,s)}{-}el\, reflection$$

These rules state that an element in V is either a code $u(v, s)$ for an internal family (v, s) or a code for an element $\mathsf{el}(v, s, z)$ in the universe built above the lifting of an internal family, where the lifting of an internal family (v, s) is the family $(\mathsf{S}(v), \mathsf{S} \circ s)$. Then S decodes these elements accordingly.

3.2 Inductive-recursive definitions

Some peculiarities in the universes' construction do not allow us to include universes among inductive or indexed inductive definitions: note indeed how, in the introduction rules given in Example 3.2 to define a regular universe, U and T depend on each other. T is recursively defined on U, and therefore it obviously depends on it, but also U depends on T since T appears, in a negative position, in the introduction rule for U. Therefore we have a *mutual dependency* of U and T: we *simultaneously* define the set U inductively and the function T recursively. This mutual dependency is even more evident if we express the definition of (U, T) in Example 3.2 by means of equations. Such a universe can be seen as the least family of sets (X, T) satisfying

$$\begin{array}{rcll} X & \cong & A + (\Sigma\, x : X)(T(x) \to X) & \\ T\,(\mathsf{inl}(a)) & = & B(a) & (3.1) \\ T\,(\mathsf{inr}\,(x, f)) & = & (\Sigma\, y : T(x))(T\,(f(y)))\ . & \end{array}$$

The analysis of more sophisticated types such as universe types, has led Dybjer to formulate a schema which generalizes the notion of inductive families (or indexed inductive type) [37]. The central insight of Dybjer was that in order to accomodate universes in the realm of inductive definitions, we need to enlarge this notion so as to capture definitions where we simultaneously define a set X and a recursive function $T : X \to D$ from X into another type D. This intuition led him to formulate the notion of an inductive-recursive definition. The mutual dependency of X and T informs the use of the adjective "simultaneous" used by Dybjer [37] to describe an inductive-recursive definition.
The definition of an inductive-recursive type goes very much like the one for an inductive types in ID (Definition 2.40). Indeed, the theory of induction-recursion, IR for short, consists of:

(i) an inductive definition of a syntax to represent IR-definitions of types;

(ii) a semantics mapping element of the syntax into functors.

Elements of the syntax are called IR-codes, while functors associated to IR-codes are called IR-functors. An inductive-recursive type is the initial algebra for an IR-functor.

3.2.1 Syntax of induction-recursion

The original presentation of induction-recursion given by Dybjer in [37] was as an external schema. Dybjer and Setzer developed further the theory to internalize the concept of inductive-recursive definition: in [38] a finite axiomatization of the theory was developed through the introduction of a special type of codes for inductive-recursive definitions. This inductive definition of inductive-recursive codes which we recall below is completely analogue to the axiomatization of inductive definitions, ID.

IR-codes

For any type I, O we can form the type $\mathsf{IR}(I, O)$, with formation rule:

$$\frac{I, O : \mathsf{type}}{\mathsf{IR}(I, O) : \mathsf{type}} \qquad (\mathsf{IR}\text{-formation})$$

Similarly to what happens for the type ID we have three constructors which build term (codes) for the type $\mathsf{IR}(I, O)$.

Definition 3.7 (IR-codes)**.** Let $I, O : \mathsf{type}$. The type of $\mathsf{IR}(I, O)$-codes has the following constructors

$$\frac{o : O}{\iota\, o : \mathsf{IR}(I, O)} \qquad (\mathsf{IR}\text{-}\iota \text{ intro})$$

$$\frac{A : \mathsf{Set} \quad f : A \to \mathsf{IR}(I, O)}{\sigma\, A\, f : \mathsf{IR}(I, O)} \qquad (\mathsf{IR}\text{-}\sigma \text{ intro})$$

$$\frac{A : \mathsf{Set} \quad F : (A \to I) \to \mathsf{IR}(I, O)}{\delta\, A\, F : \mathsf{IR}(I, O)} \qquad (\mathsf{IR}\text{-}\delta \text{ intro})$$

■

Remark 3.8. In this definition we have departed from the original presentation of Dybjer and Setzer [38, 39] in two minor ways:

1. Dybjer and Setzer consider inductive-recursive definitions for a unique type parameter D, while in the above definition we build a type $\mathsf{IR}(I, O)$ where positive and negative occurrences of the type D are separated.

2. We have used the type Set where Dybjer and Setzer use stype which is a type of small types, and it represents an intermediate level between type and Set. In their axiomatization of the Logical Framework, the type stype is a copy of Set, and it is introduced because they do not have rules to introduce elements of Set but leave the task of defining them to the theory of induction-recursion (see [37], page 4, and Section 3.2 in [39]).

We will come back to discuss this coding scheme in the next section, once we have recalled the functorial semantics for these codes. For completeness we now state the elimination an computation rules for the type $\mathsf{IR}(I, O)$.

Elimination for IR-codes

(Elimination rule):

$$\frac{\begin{array}{c}\gamma : \mathsf{IR}(I, O) \\ \gamma : \mathsf{IR}(I, O) \vdash P(\gamma) : \mathsf{type} \\ o : O \vdash a(o) : P(\iota\, o) \\ A : \mathsf{Set}, f : A \to \mathsf{IR}(I, O), g : (\Pi\, a : A) P(f(a)) \vdash b(A, f, g) : P(\sigma\, A\, f) \\ B : \mathsf{Set}, F : (B \to I) \to \mathsf{IR}(I, O), h : (\Pi\, g : B \to I) P(F(g)) \vdash c(B, F, h) : P(\delta\, B\, F)\end{array}}{\mathsf{elim}_{\mathsf{IR}(I,O)}(\gamma, a, b, c) : P(\gamma)}$$

(Computation rule). Under the same premises of the elimination rule the following equalities hold:

$$\mathsf{elim}_{\mathsf{IR}(I,O)}(\iota\, o, a, b, c) = a : P(\iota\, o)$$

$$\mathsf{elim}_{\mathsf{IR}(I,O)}(\sigma\, A\, f, a, b, c) = b(A, f, \lambda\, x.\, \mathsf{elim}_{\mathsf{IR}(I,O)}(f\, x, a, b, c)) : P(\sigma\, A\, f)$$

$$\mathsf{elim}_{\mathsf{IR}(I,O)}(\delta\, B\, F, a, b, c) = c(B, F, \lambda\, x.\, \mathsf{elim}_{\mathsf{IR}(I,O)}(F(x), a, b, c)) : P(\delta\, B\, F)$$

Every time we prove a property of $\mathsf{IR}(I, O)$-codes we are appealing to these rules, and, from now on, we will just write write "by induction on the structure of the codes..." for such an argument.

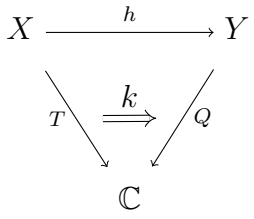
3.2.2 Semantics of induction-recursion

Inductive-recursive definitions generalize both inductive and indexed inductive definition. To capture the latter from a categorical perspective we used initial algebra semantics. Therefore we expect the same tools to adapt to inductive-recursive types. Below we define IR-functors by showing how to associate to each IR-code a functor whose initial algebra is an inductive recursive type. But first we need to understand what are the relevant categories where these functors actually live. In the next section we survey all relevant facts about the category of families which will be used for this purpose and throughout the rest of the thesis.

3.2.3 The Fam construction

Definition 3.9. Given a category $\mathbb{C}$ the category $\mathbf{Fam}\,(\mathbb{C})$ has objects pairs (X, T) where $X : \mathbf{Set}$ and $T : X \to \mathbb{C}$ is a functor which we can think as an

X-indexed family of elements of $\mathbb{C}$. A morphism between objects (X, T) and (Y, Q) is a pair (h, k) where $h : X \to Y$ is a function in **Set**, and $k : T \dot{\to} Q \circ h$ is a natural transformation, whose component at $x : X$ is a morphism $k_x : T(x) \to Q(h(x))$ in $\mathbb{C}$. We picture such a morphism as follows:



■

We recall some of the properties of **Fam** $(\mathbb{C})$ which will be used in the rest of this thesis:

Remark 3.10. For any category $\mathbb{C}$, the category **Fam** $(\mathbb{C})$ automatically has a rich structure:

1. **Fam** $(\mathbb{C})$ is fibred over Set via the functor $\pi : \mathbf{Fam}\,(\mathbb{C}) \to \mathsf{Set}$ defined by $\pi(X, T) = X$ and $\pi(h, k) = h$ (see Example 6.6 in Section 6.2).

2. **Fam** $(\mathbb{C})$ is the free Set-indexed coproduct completion of $\mathbb{C}$; that is, **Fam** $(\mathbb{C})$ has all set-indexed coproducts and there is a unit $\mathbb{C} \to \mathbf{Fam}\,(\mathbb{C})$ universal among functors $F : \mathbb{C} \to \mathbb{D}$ where $\mathbb{D}$ is a category with set indexed coproducts. Given an A-indexed collection of objects $(X_a, T_a)_{a:A}$ in **Fam** $(\mathbb{C})$, its A-indexed coproduct which we write $(\Sigma\, a : A)\,(X_a, T_a)$ is the family
$$((\Sigma\, a : A)X_a, [T_a]_{a:A})$$

3. If a category $\mathbb{C}$ has (chosen) coproducts then we can define a functor $\Sigma : \mathbf{Fam}\,(\mathbb{C}) \to \mathbb{C}$ which sends a family of objects to their set-indexed coproduct, and a family morphism (f, g) where $f : A \to A'$ and $g_x : B(x) \to B'(f(x))$ to the cotuple
$$\Sigma(f, g) = [\mathsf{in}_{f(x)} \circ g_x]_{x:A} : (\Sigma\, x : A)B \to (\Sigma\, x' : A')B'.$$
We call $\Sigma(f, g)$ the generalised sum of g over f. If $f = \mathsf{id}$, we simply write $(\Sigma\, a : A)g_a$ for the sum. The functor $\Sigma : \mathbf{Fam}\,(\mathbb{C}) \to \mathbb{C}$ is left

adjoint to the functor $\eta : \mathbb{C} \to \mathbf{Fam}\,(\mathbb{C})$, sending an object c to the family $(\mathsf{N}_1, \lambda\, x.\, c)$.

4. $\mathbf{Fam}\,(\mathbb{C})$ is cocomplete if and only if $\mathbb{C}$ has all small connected colimits (Carboni and Johnstone [24, dual of Prop. 2.1]).

5. $\mathbf{Fam}\,-$ is a functor $\mathsf{CAT} \to \mathsf{CAT}$; given $F : \mathbb{C} \to \mathbb{D}$, $\mathbf{Fam}\,F : \mathbf{Fam}\,(\mathbb{C}) \to \mathbf{Fam}\,(\mathbb{D})$ is given by composition, i.e. $\mathbf{Fam}\,F(X, T) = (X, F \circ T)$.

If we use constructive sets in Set in palce of elements of **Set** to index our families, we get a constructive version of the families' construction. This will be the basic category used to interpret inductive-recursive definitions.

Definition 3.11. Given a category $\mathbb{C}$ the category $\mathsf{Fam}\,(\mathbb{C})$ has (i) objects pairs (X, T) where $X : \mathsf{Set}$ and $T : X \to \mathbb{C}$ is a functor; (ii) morphisms between objects (X, T) and (Y, Q) are pairs (h, k) where $h : X \to Y$ is a function in **Set**, and $k : T \dot{\to} Q \circ h$ is a natural transformation ■

In several places we will make use of the following simple observation, which will turn out to be crucial for the interpretation of IR-codes as functors.

Observation 3.12. When $\mathbb{C}$ is a discrete category a morphism between families (X, T) and (Y, Q) consists of functions $h : X \to Y$ such that $T(x) = Q(h(x))$ for all x in X.

Remark 3.13. Given a type $D : \mathsf{type}$ we can think D as the discrete category $|D|$ whose objects are its terms $d : D$ and whose only morphism are the identities $\mathsf{refl} : d \equiv_D d$.

IR-functors

As already observed, a universe (U, T) is family of sets, i.e. $\mathsf{U} : \mathsf{Set}$ and $\mathsf{T} : \mathsf{U} \to \mathsf{Set}$. Thus on a categorical ground we are naturally led to consider it as an object of $\mathsf{Fam}\,(\mathsf{Set})$. This, however, is not quite true: as we are going to see, to define proper functors with an action on morphisms, as well as on objects, we have to restrict the action of these functors to the subcategory $\mathsf{Fam}|\mathsf{Set}|$ whose morphisms are morphisms (h, k) in $\mathsf{Fam}\,(\mathsf{Set})$ such that $k = \mathsf{id}$ (this observation can be traced back to Mendler [76] who

firstly gave a categorical account of type universes). More generally, since the recursive function T of an inductive-recursive type (U, T) can target an arbitrary type D we will consider functors on $\mathsf{Fam}|D|$ where $|D|$ is trivially the discrete category associated to a type D (cf. Remark 3.13). We refer to a morphism in $\mathsf{Fam}|D|$ simply as h instead of (h, id) (see Observation 3.12). In particular, the functor interpreting a code $\gamma : \mathsf{IR}(I, O)$ will have type

$$[\![\gamma]\!] : \mathsf{Fam}|I| \to \mathsf{Fam}|O|$$

Definition 3.14 (IR-functors)**.** Let $I, O : \mathsf{type}$, $\gamma : \mathsf{IR}(I, O)$ and (X, T) an object of $\mathsf{Fam}|I|$. Define $[\![\gamma]\!] : \mathsf{Fam}|I| \to \mathsf{Fam}|O|$ by induction on the structure of the code as follows:

(ι) if $\gamma = \iota\, o$ for some $o : O$, then

$$[\![\iota\, o]\!]\,(X, T) = (\mathsf{N}_1, \lambda\, x\,.\, o)$$

(σ) if $\gamma = \sigma\, A\, f$ for some $A : \mathsf{Set}$, $f : A \to \mathsf{IR}(I, O)$, then

$$[\![\sigma\, A\, f]\!]\,(X, T) = (\Sigma\, a : A)\, [\![f(a)]\!]\,(X, T)$$

(δ) if $\gamma = \delta\, B\, F$ for some $B : \mathsf{Set}$, $F : (B \to I) \to \mathsf{IR}(I, O)$, then

$$[\![\delta\, B\, F]\!]\,(X, T) = (\Sigma\, g : B \to X)\, [\![F\,(T \circ g)]\!]\,(X, T)$$ ■

Notice that the interpretation of both a σ-code and a δ-code make essential use of Set-indexed coproducts of families, as defined in Remark 3.10(2): in particular, the interpretation of a code $\delta\, B\, F$ is a sum over the function space $(B \to X)$ which is a set since both B and X are.

Remark 3.15. Dybjer and Setzer give the semantics in two steps by specifying the action of a functor $[\![\gamma]\!]$ on a family (X, T) in the two components[2] $([\![\gamma]\!]_0(X, T), [\![\gamma]\!]_1(X, T))$ where $[\![\gamma]\!]_0(X, T) : \mathsf{Set}$ and $[\![\gamma]\!]_1(X, T) : [\![\gamma]\!]_0(X, T) \to O$. The use of coproducts in $\mathsf{Fam}|I|$ enable us to give a more compact definition of both the action on objects and on morphisms.

[2]They use $(\mathsf{Arg}(\gamma, X, T),\ \mathsf{Fun}(\gamma, X, T))$ and $(\mathbb{F}^U_\gamma(X, T), \mathbb{F}^T_\gamma(X, T))$ as notation for the two components respectively in [38] and [39].

Remark 3.16. We interpret codes as functors on $\mathsf{Fam}|D|$ while Dybjer and Setzer [39] originally gave a semantics using functors on Type/D, where Type is the syntactic category arising from the judgments of the logical framework. In their axiomatization IR-functors are first introduced to act on families (X, T), where $X : \mathsf{Set}$, and then extended to act on large families (X, T), where $X : \mathsf{Type}$. However, since we want the initial algebra for an IR-functors to be a family with small domain, i.e. with $X : \mathsf{Set}$ we can avoid this roundtrip by considering functors on $\mathsf{Fam}|D|$ only.

We now give the action of IR functors on morphisms of families. Given $h : (X, T) \to (X', T')$ in $\mathsf{Fam}|I|$ we have: the action of the functor $[\![\iota\, o]\!]$ on h is nothing but the identity morphism on the family $(\mathsf{N}_1, \lambda\_ .\, o)$

$$[\![\iota\, o]\!]\, h = \mathsf{id}_{\mathsf{N}_1}$$

The action of the functor $[\![\sigma\, A\, f]\!]$ on h is given by the sum of the A-indexed family of morphisms $([\![f\ a]\!]\, h)_{a \in A}$ (cf. Remark 3.10(3))

$$[\![\sigma\, A\, f]\!]\, h = (\Sigma\, a : A)\, [\![f\ a]\!]\, h$$

The action of $[\![\delta\, B\, F]\!]$ on h is the generalized sum (cf. Remark 3.10(3)) of $[\![F(T \circ g)]\!]h$ over $\lambda\, g.\, (h \circ g)$ which we write as

$$[\![\delta\, B\, F]\!]\, h = \Sigma\, (\lambda\, g.\, h \circ g,\ \lambda\, g.\, [\![F(T \circ g)]\!]\, h) \tag{3.2}$$

We spell out the details of this concise definition. This will help us to point precisely where the restriction on discreteness arise, i.e. why we consider functors on $\mathsf{Fam}|\mathsf{Set}|$ in respect to $\mathsf{Fam}\,(\mathsf{Set})$ when, for example, I is Set. We want to exhibit a morphism

$$\big((\Sigma\, g : B \to X)[\![F\,(T \circ g)]\!](X, T)\big) \to \big((\Sigma\, g' : B \to X')[\![F\,(T' \circ g')]\!](X', T')\big)$$

Being a morphism from a coproduct we need a morphism for each component $[\![F\,(T \circ g)]\!]\,(X, T)$ of the sum. We can use post-composition with $h : X \to X'$ to get a function between the index sets of these sums, $(\lambda\, g.\, h \circ g) : (B \to X) \to (B \to X')$. Now we make essential use of the assumption that we are working in $\mathsf{Fam}|I|$ where the second component of a morphism is an identity:

since $T =_{X \to I} T' \circ h$ we can deduce $T \circ g =_{B \to I} T' \circ h \circ g$ and use the action of $[\![F(T \circ g)]\!] = [\![F(T' \circ h \circ g)]\!]$ on the morphism h given by the inductive hypothesis. Composing $[\![F(T' \circ h \circ g)]\!]\, h$ with the injection

$$\mathsf{in}_{h \circ g} \colon [\![F(T' \circ h \circ g)]\!](X', T') \to (\Sigma\, g' : B \to X')[\![F\,(T' \circ g')]\!]\,(X', T')$$

give us the desired map: indeed $[\mathsf{in}_{h \circ g} \circ [\![F(T \circ g)]\!]\, h]_{g\, B \to X}$ is the same as in equation (3.2) above, once we have expanded it with the definition of generalized sum.

The axiomatization of IR we gave with the three constructors ι, σ and δ stems from a careful analysis by Dybjer and Setzer [39] of the introduction rules of the usual set-constructors in Martin-Löf type theory. Dybjer and Setzer distinguish between *inductive* and *non-inductive* arguments in the premisses of a constructor. A non-inductive argument is an argument where elements of a previously constructed set occur. For example, the premises of the introduction rule for the type $(\Sigma\, a : A)B(a)$ consists of two non-inductive arguments: an element $a : A$ and an element $b : B(a)$, where the second argument depend on the first *directly*. An inductive argument is a premise where elements of the set we are currently building occurs. For example in the introduction rules for N the constructor $\mathsf{0}$ has no arguments and the constructor s has one inductive argument $n : \mathsf{N}$. Later premises cannot depend on an inductive argument directly (since we are currently building the elements of the set occurring in an inductive argument). But they can depend on it *indirectly*. Here, the recursive function T defined simultaneously with the inductive set U enter the scene. Consider the universe in Example 3.2: U has two constructors: the first has one non-inductive argument $a : A$, while the second has two inductive arguments $a : \mathsf{U}$ and $b : \mathsf{T}(u) \to \mathsf{U}$. The latter is an inductive argument which depends on a previous inductive argument indirectly via T. The three constructors ι, σ and δ corresponds to this analysis: the code ι represents a base case, i.e. it represents those constructors with no argument at all. The code σ represents constructors with non-inductive arguments and the code δ represents those with inductive arguments.

3.2.4 A coding scheme for IR based on Cont

Ghani and Hancock [50], following an observation of Altenkrich, give a different coding scheme for $\mathsf{IR}(I, O)$-codes based on containers. This also offers an alternative but equivalent presentation of the semantics. The idea is to keep the ι constructor and to merge the σ and the δ constructors together into a single constructor $\sigma\delta$ which now depends on a container (S, P). Similarly the function f and the (large) function F respectively in the premises of a σ-code and of a δ-code, are also merged into a single function G whose domain is the extension of the container (S, P). To avoid possible confusion between the semantics brackets for IR-functors and the container extensions we adopt the following alternative notation for the latter:

Notation 3.17. We indicate with $(S \lhd P)$ the extension of a container (S, P) (cf. Definition 2.29).

The introduction rule for the $\sigma\delta$ constructor is the following

$$\frac{(S, P) : \mathsf{Cont} \quad G : (S \lhd P)\, I \to \mathsf{IR}(I, O)}{\sigma\delta\,(S, P)\, G : \mathsf{IR}(I, O)}$$

We interpret a code $\sigma\delta\,(S, P)\, G : \mathsf{IR}(I, O)$ as the functor

$$[\![\sigma\delta\,(S, P)\; G]\!] : \mathsf{Fam}|I| \longrightarrow \mathsf{Fam}|O|$$

whose action on a family $(X, T) : \mathsf{Fam}|I|$ is

$$\begin{aligned}
[\![\sigma\delta(S, P)\, G]\!]\,(X, T) =& (\Sigma\, s : S)\,(\Sigma\, g : P\, s \to X)\,[\![G(s,\, T \circ g)]\!](X, T) \\
=& \Sigma\,(s, g) : (S \lhd P)\, X.\, [\![G(s,\, T \circ g)]\!](X, T) \\
=& \Sigma\,(s, g) : (S \lhd P)\, X.\, [\![G((S \lhd P)\, T(s, g))]\!](X, T)
\end{aligned}$$

where in the last line we used the action of $(S \lhd P)$ on morphisms, given by simple composition. We can now check that the two schemes actually define the same class of functors.

Lemma 3.18. The coding scheme given in Definition 3.7 and the coding scheme obtained by replacing the σ and δ constructors there by the single $\sigma\delta$ constructor as given above define the same class of functors.

Proof. By induction of the structure of the codes we build maps φ and ψ between the sets of codes obtained by the two schemes. In one direction φ maps a code $\sigma\delta\,(S, P)\,G$ to the code $\sigma S(s \mapsto \delta\,(P\,s)\,\varphi(G(s, \_)))$ which belong to the original coding scheme given in Definition 3.7. In the other direction ψ maps a code $\sigma\,A\,f$ to the code $\sigma\delta((A, \lambda\,a.\,\mathsf{N}_0))(x \mapsto \psi(f(\pi_0(a))))$, and a code $\delta\,B\,G$ to the code $\sigma\delta(\mathsf{N}_1, \lambda\,x.\,B)(y \mapsto \psi(G(\pi_1(y))))$. Checking that the semantics is preserved follows immediately from the definitions. □

3.2.5 Introduction and elimination rule for IR-types

We presented the theory of IR from the perspective of initial algebra semantics: an IR-code $\gamma : \mathsf{IR}(D, D)$ defines an IR-functors $[\![\gamma]\!] : \mathsf{Fam}|D| \to \mathsf{Fam}|D|$; the IR-type $(\mathsf{U}_\gamma, \mathsf{T}_\gamma) : \mathsf{Fam}|D|$ defined by γ is precisely the carrier of the initial algebras $(\mu_{[\![\gamma]\!]}, \mathsf{intro}_{[\![\gamma]\!]})$ for this functor. We can express this principle in a natural deduction style as follows (below we write an IR-functor as given by a pair $([\![\gamma]\!]_0, [\![\gamma]\!]_1)$ as in Remark 3.15):

$$\frac{a : [\![\gamma]\!]_0(\mathsf{U}_\gamma, \mathsf{T}_\gamma)}{\mathsf{intro}_{[\![\gamma]\!]}(a) : \mathsf{U}_\gamma} \qquad \frac{a : [\![\gamma]\!]_0(\mathsf{U}_\gamma, \mathsf{T}_\gamma)}{\mathsf{T}_\gamma(\mathsf{intro}_{[\![\gamma]\!]}(a)) = [\![\gamma]\!]_1(\mathsf{U}_\gamma, \mathsf{T}_\gamma)\,a}$$

Thus, it is natural for us to assume a simple elimination rule corresponding to the recursion principle given by the fold operator (see Section 2.2.1) associated to an IR-functor. However, it is also possible to formulate an induction principle for IR-types in terms of an elimination rule [39] analogous to those given for the other types in the theory.

To this end, for each code $\gamma : \mathsf{IR}(I, O)$, we define a set $\mathsf{IH}_\gamma((X, T), P, x)$ by induction on the structure of γ (see Section 3.5 in [39] for the details). This set collects the inductive hypothesis for a predicate $x : X \vdash P(x) : \mathsf{type}$ on a given family (X, T), and elements $x : [\![\gamma]\!]_0(X, T)$.

$$\frac{X : \mathsf{Set} \quad T : X \to \mathsf{Set} \quad y : X \vdash P(y) : \mathsf{type} \quad x : [\![\gamma]\!]_0(X, T)}{\mathsf{IH}_\gamma(X, T, P, x) : \mathsf{type}}$$

The induction principle for $(\mathsf{U}_\gamma, \mathsf{T}_\gamma)$ can now be stated as follows:

$$\frac{\begin{array}{c} u : \mathsf{U}_\gamma \\ x : \mathsf{U}_\gamma \vdash P(x) : \mathsf{type} \\ a : [\![\gamma]\!]_0(\mathsf{U}_\gamma, \mathsf{T}_\gamma), p : \mathsf{IH}_\gamma(\mathsf{U}_\gamma, \mathsf{T}_\gamma, P, a) \vdash g(\mathsf{intro}_{[\![\gamma]\!]}(a), p) : P(\mathsf{intro}_{[\![\gamma]\!]}(a)) \end{array}}{\mathsf{elim}_{[\![\gamma]\!]}(u, P, g) : P(u)} \quad ([\![\gamma]\!]\text{−elim})$$

To deal with the recursive calls in the computation rule we need to be able to define elements of $\mathsf{IH}_\gamma((X,T),P,x)$ from the values of a recursively defined function g on the inductive arguments $u : \mathsf{U}_\gamma$. For this purpose Dybjer and Setzer define a map $\mathsf{IH}_\gamma{}^{\mathsf{map}}$:

$$\frac{X : \mathsf{Set} \quad T : X \to \mathsf{Set} \quad x : X \vdash P(x) : \mathsf{type} \quad y : X \vdash g(y) : P(y)}{\mathsf{IH}_\gamma{}^{\mathsf{map}}(X, T, P, h) : [\![\gamma]\!]_0(X, T) \to \mathsf{IH}_\gamma(X, T, P, x)}$$

We can then state the computation rule as follows:

$$\frac{\begin{array}{c} x : \mathsf{U}_\gamma \vdash P(x) : \mathsf{type} \qquad u : [\![\gamma]\!](\mathsf{U}_\gamma, \mathsf{T}_\gamma) \\ a : [\![\gamma]\!]_0(\mathsf{U}_\gamma, \mathsf{T}_\gamma), p : \mathsf{IH}_\gamma(\mathsf{U}_\gamma, \mathsf{T}_\gamma, P, a) \vdash g(\mathsf{intro}_{[\![\gamma]\!]}(a), p) : P(\mathsf{intro}_{[\![\gamma]\!]}(a)) \end{array}}{\begin{array}{l} \mathsf{elim}_{[\![\gamma]\!]}(\mathsf{intro}_{[\![\gamma]\!]}(u), P, g) = \\ \qquad g(u, \mathsf{IH}_\gamma{}^{\mathsf{map}}((\mathsf{U}_\gamma, \mathsf{T}_\gamma), P, \lambda x.\mathsf{elim}_{[\![\gamma]\!]}(x, P, g))(u)) : P(\mathsf{intro}_{[\![\gamma]\!]}(u)) \end{array}} \quad ([\![\gamma]\!]\text{−comp})$$

As proved by Dybjer and Setzer ([39], Theorems 4.4.1 and 4.4.3), the elimination and computation rules stated above are equivalent to asserting that $(\mathsf{U}_\gamma, \mathsf{T}_\gamma)$ is an initial algebra for $[\![\gamma]\!]$. The proof of this fact follows a pattern similar to the argument we have used in 2.2.2.

Remark 3.19. Notice that, since we want the elimination principle for IR-types to account for recursive functions $T : X \to D$, where D can be a large type (as in the universe construction where $D = \mathsf{Set}$), we need to adopt the *large* elimination rule, where the predicate P in the rule is a type-valued predicate (see also [37] for further discussion on this), rather than a Set-valued predicate.

3.2.6 Indexed induction-recursion

Dybjer and Setzer did not stop just at inductive-recursive definitions but they went on to define the next level of data types: they introduced [40] the

theory of indexed induction-recursion, IIR for short. In a concise form: IIR-types are to IR-types as indexed inductive types are to inductive types. That is, within indexed inductive-recursive definitions we add indices on which an inductive-recursive type might depend. This can be useful if we want to define simultaneously a family of sets $X : I \to \mathsf{Set}$ and recursive functions $T : (\Pi\, i : I)(X(i) \to D(i))$ out of it, in such a way that elements in different sets of the family are mapped into different types. Noteworthy examples of indexed inductive-recursive types are: (i) the Tait-style computability predicates for dependent types used by Martin-Löf to prove normalization of his system [74], (ii) Palmgren's higher order universes [83] which generalize the super universe construction (cf. Section 3.1.3) and (iii) the Bove and Capretta's method to define nested general recursive functions in type theory [22]. Below, we recall the syntax of IIR:

Definition 3.20 (IIR)**.** Formation rule:

$$\frac{I, J : \mathsf{Set} \quad i : I \vdash D(i) : \mathsf{type} \quad j : J \vdash E(j) : \mathsf{type}}{\mathsf{IIR}(D, E) : \mathsf{type}}$$

Introduction rules:

$$\frac{j : J \quad e : E(j)}{\iota(j, e) : \mathsf{IIR}(D, E)} \qquad (\mathsf{IIR} - \iota - \text{ intro})$$

$$\frac{A : \mathsf{Set} \quad f : A \to \mathsf{IIR}(D, E)}{\sigma\, A\, f : \mathsf{IIR}(D, E)} \qquad (\mathsf{IIR} - \sigma - \text{ intro})$$

$$\frac{B : \mathsf{Set} \quad g : B \to I \quad F : ((\Pi\, b : B) D(g(b))) \to \mathsf{IIR}(D, E)}{\delta\, B\, g\, F : \mathsf{IIR}(D, E)}$$

$$(\mathsf{IIR} - \delta - \text{ intro})$$

■

Codes of type $\mathsf{IIR}(D, E)$ are interpreted as functors between I-fold product of categories of families defined in the following way:

Definition 3.21. Given $I : \mathsf{Set}$ and discrete categories $|D(i)|$ for $i : I$ the category $(\Pi\, i : I)\mathsf{Fam}|D(i)|$ has

- objects are pairs (X, T) where $X : I \to \mathsf{Set}$, $T : (\Pi\, i : I)(X(i) \to D(i))$,
- morphisms from (X, T) to (Y, Q) consists of a function $h : (\Pi\, i : I)(X(i) \to Y(i))$ such that $T\, i = (Q\, i) \circ h(i)$. ■

Dybjer and Setzer [40] note that, assuming extensional rules, the category $(\Pi\, i : I)\mathsf{Fam}|D(i)|$ can be presented as a category of families exactly like one where IR-functors are interpretated.

Lemma 3.22. Assuming rules 2.14, the category $(\Pi\, i : I)\mathsf{Fam}|D(i)|$ is equivalent to the category $\mathsf{Fam}\,(\Sigma\, i : I)|D(i)|$, where $(\Sigma\, i : I)|D(i)|$ is the I-indexed coproduct category of the discrete categories $|D(i)|$. The latter has objects pairs (U, T) where $U : \mathsf{Set}$ and $T : U \to (\Sigma\, i : I)D(i)$.

Proof. We give the two mappings on objects which are inverse to each other. The action on morphisms easily follows.

$$\begin{aligned}
&\varphi : (\Pi\, i : I)\mathsf{Fam}|D(i)| \to \mathsf{Fam}|(\Sigma\, i : I)D(i)| \\
&\varphi(X, T) = ((\Sigma\, i : I)X(i), (\Sigma\, i : I)(T(i, -))
\end{aligned}$$

$$\begin{aligned}
&\theta : \mathsf{Fam}|(\Sigma\, i : I)D(i)| \to (\Pi\, i : I)\mathsf{Fam}|D(i)| \\
&\theta(Y, Q) = ((\pi_0 \circ Q)^{-1}, \pi_1 \circ Q \circ \pi_0)
\end{aligned}$$

where we have used $(\Sigma\, i : I)(T(i, -)) : (\Sigma\, i : I)X(i) \to (\Sigma\, i : I)D(i)$ in the definition of φ to indicate the generalized sum of $T(i, -)$ as defined in Remark 3.10, and $\pi_1 \circ Q \circ \pi_0 : (\Pi\, i : I)((\pi_0 \circ Q)^{-1})(i) \to D(i)$ in the definition of θ to indicate the function $\lambda\, i : I.\lambda\, x : (\Sigma\, y : Y)(\pi_0(Q(y)) \equiv_I i).\, \pi_1(Q(\pi_0(x)))$ □

Dybjer and Setzer [40] uses these two equivalent categories to distinguish between a general and a restricted version of the theory of IIR, depending on the role the index I plays: in the restricted version an IIR-functor on $(\Pi\, i : I)\mathsf{Fam}|D(i)|$ is uniquely determined by its components, i.e. determined by its projection on $\mathsf{Fam}|D(i)|$ and therefore I acts parametrically. In the general version the indices $i : I$ for elements $u : U(i)$ are not fixed in advance,

but they are built on the way the elements u are introduced. The two corresponding theory are equivalent assuming the extensional rules 2.14. Therefore, we will not distinguish between restricted and general IIR, and we give here the restricted version. We interpret a code $\gamma : \mathsf{IIR}(D, E)$ as a functor

$$[\![\gamma]\!] : (\Pi\, i : I)\mathsf{Fam}|D(i)| \to (\Pi\, j : J)\mathsf{Fam}|E(j)|$$

Definition 3.23 (IIR-functors)**.** Let $i : I \vdash D(i) : \mathsf{type}$ and $j : J \vdash E(j) : \mathsf{type}$ be valid judgments, $\gamma : \mathsf{IIR}(D, E)$ and (X, T) an object of $(\Pi\, i : I)\mathsf{Fam}|D(i)|$. Define $[\![\gamma]\!] : (\Pi\, i : I)\mathsf{Fam}|D(i)| \to (\Pi\, j : J)\mathsf{Fam}|E(j)|$ by induction on the structure of the codes as follows:
if γ is $\iota\,(j, e)$ for $j : J$ and $e : E(j)$ then

$$[\![\iota\,(j, e)]\!]_{\mathsf{IIR}}(X, T) = \lambda\, j'.\, (j \equiv_J j', \lambda\_.\, e)$$

If $\gamma = \sigma\, A\, f$ for $A : \mathsf{Set}$ $f : A \to \mathsf{IIR}(D, E)$, then

$$[\![\sigma\, A\, f]\!]_{\mathsf{IIR}}(X, T) = (\Sigma\, a : A)[\![f\ a]\!]_{\mathsf{IIR}}(X, T)$$

If $\gamma = \delta\, B\, g\, F$ for $B : \mathsf{Set}$ $g : B \to I$, $F : ((\Pi\, b : B)D(g(b))) \to \mathsf{IIR}(D, E)$, then

$$[\![\delta\, B\, g\, F]\!]_{\mathsf{IIR}}(X, T) = (\Sigma\, f : (\Pi\, b : B)U(g(b)))[\![F(T(g(-), -) \circ \langle \mathsf{id}, f\rangle)]\!]_{\mathsf{IIR}}(X, T)$$

where the function $T(g(-), -) \circ \langle \mathsf{id}, f\rangle : (\Pi\, b : B)D(g(b))$ in the last equation is the function $\lambda\, b.\, T(g(b), (f(b)))$. ■

Remark 3.24. As already noted in Remark 3.15 the use of coproducts enable us to give a more compact definition for IR-functors. Here we have used set-indexed coproducts in $(\Pi\, j : J)\mathsf{Fam}|E(j)|$ to interpret both σ and δ-codes in Definition 3.23.

3.2.7 Examples

We give some examples of inductive-recursive types to show the theory at work. This selection of examples is not exhaustive in any possible way: the theory is so expressive that almost every existing type is representable within it (a remarkable exception is given by the internal Mahlo universe

constructed by Setzer [88]). We refer the reader to [39, 40] for a richer selection of examples.
To represent inductive-recursive types, we will simply exhibit the corresponding IR-codes. As already explained the intended type is given by taking initial algebra for the IR-functor which interprets the corresponding code. We begin with a technical remark which facilitates readability of the codes. We use the following convention:

Notation 3.25. Given codes $\gamma, \gamma' : \mathsf{IR}(I, O)$ let

$$\gamma + \gamma' =_{\mathsf{def}} \sigma\, \mathsf{N}_2(x \mapsto \mathsf{case}_2(x, \gamma, \gamma'))$$

It is immediate to check that $[\![\gamma + \gamma']\!] \cong [\![\gamma]\!] + [\![\gamma']\!]$

Inductive types as presented in Section 2.2 are all instances of the theory of IR. Indeed they are *degenerate* inductive-recursive definitions of a family (X, T), degenerate in the sense that the recursive function T does not contribute to the definition of the type; in particular ID-types correspond to $\mathsf{IR}(D, D)$ codes where $D = \mathsf{N}_1$. In this case T is the unique map from the inductively defined type X into N_1. The following two are examples of *degenerate* inductive-recursive definitions:

Natural numbers

The code for the set N of natural numbers is

$$\iota \mathsf{0}_1 + \delta\, \mathsf{N}_1(x \mapsto \iota\, \mathsf{0}_1) : \mathsf{IR}(\mathsf{N}_1, \mathsf{N}_1)$$

W-types

The code γ defining the W-type $(\mathrm{W}\, x : A)B(x)$ is

$$\gamma \;=_{\mathsf{def}} \sigma\, A\, (a \mapsto \delta\, B(a)\, (f \mapsto \iota\, \mathsf{0}_1)) : \mathsf{IR}(\mathsf{N}_1, \mathsf{N}_1)$$

Fresh lists

Let A be a set equipped with a relation $\not\approx : A \times A \to \mathsf{Set}$ expressing that two elements of A are different. We want to define a set L of lists of elements of

A in which each element has at most one occurrence, i.e. lists of elements of A where every element is different from every other in the list by the relation $\not\approx$. To this end, we simultaneously define a set of lists FrList and a function $\mathsf{Fresh} : \mathsf{FrList} \to A \to \mathsf{Set}$ which tells when an element $a : A$ is fresh for a list $l : \mathsf{FrList}$. FrList and Fresh are the least solution of the following equations:

$$\begin{array}{ll} X & \cong \mathsf{N}_1 + (\Sigma\,(xs, a){:}\, X \times A)\, T(l, a) \\ T\,(inl\,\mathsf{0}_1)\,a & \cong \lambda\, b.\, \mathsf{N}_1 \\ T\,(inr\,((xs, a), p)) & \cong \lambda\, b.\, (b \not\approx a) \times T(xs, b)\ . \end{array}$$

The code which defines the endofunctor whose fixed point is $(\mathsf{FrList}, \mathsf{Fresh})$ is given by the following $\mathsf{IR}(\mathsf{Set}^A, \mathsf{Set}^A)$-code:

$$\iota(\lambda\, a : A.\, \mathsf{N}_1) + \sigma\, A\, (a \mapsto \delta\, 1\, (f \mapsto \iota(\lambda\, b : A.(b \not\approx a) \times f(\mathsf{0}_1, b))))$$

Universes

We have already stressed that universes have been a primary source of examples of inductive recursive definitions. We now give IR-codes for defining universes closed by type operators.
The code γ for the universe described in Example 3.2 is

$$\begin{array}{ll} \gamma_{\mathsf{N},\Sigma} =_{\mathsf{def}} \sigma\, A(a \mapsto \iota B(a)) & \\ \qquad + \delta\, \mathsf{N}_1\, (X \mapsto \delta\, X(\mathsf{0}_1)(Y \mapsto \iota((\Sigma\, X(\mathsf{0}_1))\, Y))) : \mathsf{IR}(\mathsf{Set}, \mathsf{Set}) & (3.3) \end{array}$$

If the universe is closed under other type constructors like Π or W we simply add to $\gamma_{\mathsf{N},\Sigma}$ the following codes

$$\delta\, \mathsf{N}_1\, (X \mapsto \delta\, X(\mathsf{0}_1)(Y \mapsto \iota((\Pi\, X(\mathsf{0}_1))\, Y))) \quad (3.4)$$

$$\delta\, \mathsf{N}_1\, (X \mapsto \delta\, X(\mathsf{0}_1)(Y \mapsto \iota((\mathrm{W}\, X(\mathsf{0}_1))\, Y))) \quad (3.5)$$

Super universes

As explained in Section 3.1.3 the universe construction can be uniformly presented through a universe operator acting on families. Reflecting this operator gives us a family closed under the universe construction, i.e. a

super universe. Therefore a universe operator can be formally presented as an operator

$$(\mathbf{U}, \mathbf{T}) : \mathsf{Fam}|\mathsf{Set}| \to \mathsf{Fam}|\mathsf{Set}|$$

where

$$\begin{aligned}&\mathbf{U} : \mathsf{Fam}|\mathsf{Set}| \to \mathsf{Set}\\ &\mathbf{T} : ((A, B) : \mathsf{Fam}|\mathsf{Set}|) \to \mathbf{U}(A, B) \to \mathsf{Set}\end{aligned}$$

We can express that the super universe (S, V) is the least family containing a given family (A, B), and closed by the universe construction, by the following equations:

$$\begin{aligned}
\mathsf{V} &\cong \mathsf{N}_1 + A + (\Sigma\, v{:}\,\mathsf{V})(\mathsf{S}(v) \to \mathsf{V}) \\
&\qquad + (\Sigma v : \mathsf{V})(\Sigma\, s : \mathsf{S}(v) \to \mathsf{V})\, \mathbf{U}(\mathsf{S}(v), \lambda\, x.\, \mathsf{S}(s(x))) \\
\mathsf{S}\,(\mathsf{in}_1(\mathsf{0}_1)) &= A \\
\mathsf{S}\,(\mathsf{in}_2(a)) &= B(a) \\
\mathsf{S}\,(\mathsf{in}_3(v, f)) &= \mathbf{U}(\mathsf{S}\, v, \lambda\, x.\, \mathsf{S}(s(x))) \\
\mathsf{S}\,(\mathsf{in}_4(v, f, z)) &= \mathbf{T}(\mathbf{U}(\mathsf{S}\, v, \lambda\, x.\, \mathsf{S}(s(x))), z)
\end{aligned}$$

The $\mathsf{IR}(\mathsf{Set}, \mathsf{Set})$-code for a functor defining a super universe is therefore given by

$$\begin{aligned}
\gamma_{\mathsf{su}} =_{\mathsf{def}}\ & \iota\, A \\
& + \sigma\, A\, (a \mapsto \iota(B\, a)) \\
& + \delta\, \mathsf{1}(X \mapsto \delta X(\mathsf{0}_1)(Y \mapsto \iota(\mathbf{U}(X(\mathsf{0}_1), Y)))) \\
& + \delta\, \mathsf{1}(X \mapsto \delta X(\mathsf{0}_1)(Y \mapsto \sigma\, \mathbf{U}(X(\mathsf{0}_1), Y)(z \mapsto \iota(\mathbf{T}(\mathbf{U}(X(\mathsf{0}_1), Y), z)))))
\end{aligned}$$

An inaccessible (Peter Hancock)

In Example 1.5 we sketched how to built indexed families of number classes by fixing the index of the family in advance. We now use induction-recursion to build a family of number classes which are regular, and whose indices are number classes of the family which is currently being defined. This example has been shown to me by Peter Hancock. To this end we use containers to represent the functors whose initial algebras are the number classes; therefore the family (Ω, T) we will build will have type $\mathsf{Fam}|\mathsf{Cont}|$. We want this family

to satisfy the following equations:

$$\begin{array}{rcl} \Omega & \cong & \mathsf{N}_1 + \Omega + (\Sigma\,\alpha : \Omega)(\mu_{[\![\mathsf{T}(\alpha)]\!]_{\mathsf{C}}} \to \Omega) \\ \mathsf{T}(\mathsf{in}_1(\mathsf{0}_1)) & = & (S_{\mathsf{N}}, P_{\mathsf{N}}) \\ \mathsf{T}(\mathsf{in}_2(\alpha)) & = & \mathsf{T}(\alpha) + (\mathsf{N}_1, \mu_{[\![\mathsf{T}(\alpha)]\!]_{\mathsf{C}}}) \\ \mathsf{T}(\mathsf{in}_3(\alpha, f)) & = & (\Sigma\, t : \mu_{[\![\mathsf{T}(\alpha)]\!]_{\mathsf{C}}})\mathsf{T}(f(t)) \end{array}$$

Let us take a closer look at these equations: if $\alpha = \mathsf{in}_1(\mathsf{0}_1)$ then the corresponding number class $\mathsf{T}(\alpha)$ is given by the container $(S_{\mathsf{N}}, P_{\mathsf{N}})$, representing natural numbers (see Example2.37); indeed $\Omega(\mathsf{0}) =_{\mathsf{def}} \mathsf{N} = \mu\, X.(1 + X)$; if $\alpha = \mathsf{in}_2(\alpha')$ than we build the corresponding number class by extending the previously built number class $\mathsf{T}(\alpha')$ by the container with one shape and $\mu_{[\![\mathsf{T}(\alpha)]\!]_{\mathsf{C}}}$ positions; this represents a successor operator $\Omega(\alpha) \mapsto \Omega(\alpha + 1) = \mu\, X.\, (\Omega(\alpha) + X^{\Omega(\alpha)})$. Finally, if $\alpha = \mathsf{in}_3(\alpha', f)$ then $\mathsf{T}(\alpha)$ is the sum over $\mu_{[\![\mathsf{T}(\alpha)]\!]_{\mathsf{C}}}$ of the number classes in the image of f; that is, we use sums of containers to represent the limit operator $\Omega(\lambda) \mapsto (\Sigma\, \beta < \lambda)\Omega(\beta)$.
We can represent the family (Ω, T) by the following $\mathsf{IR}(\mathsf{Cont}, \mathsf{Cont})$-code:

$$\begin{array}{rl} \gamma_{(\Omega,\mathsf{T})} =_{\mathsf{def}} & \iota(S_{\mathsf{N}}, P_{\mathsf{N}}) \\ & + \delta\, \mathsf{N}_1(X \mapsto (\iota(X(\mathsf{0}_1) + (\mathsf{N}_1, \mu_{[\![X(\mathsf{0}_1)]\!]_{\mathsf{C}}})))) \\ & + \delta\, \mathsf{N}_1(X \mapsto \delta\, \mu_{[\![X(\mathsf{0}_1)]\!]_{\mathsf{C}}}(Y \mapsto \iota(\Sigma\, x : \mu_{[\![X(\mathsf{0}_1)]\!]_{\mathsf{C}}})(Y(x))) \end{array}$$

Peter Hancock dubbed the family (Ω, T) 'inaccessible": indeed, if we regard number classes, here represented as containers $\mathsf{T}(\alpha)$, for $\alpha : \Omega$ as enumerating regular ordinals, then (Ω, T) represents a family closed under the step to the next regular, and therefore in a sense that can be made precise, an inaccessible.

3.3 A set-theoretic model

In this section we recast the set-theoretic model built by Dybjer and Setzer [38, 40] in order to prove the consistency of the theory of IR. We adapt Dybjer and Setzer's original model to a more categorical setting. This recasting will pave the way to the initial algebras argument to come in Chapter 5 and 6. Dybjer and Setzer's model extends the type-as-(classical)-sets interpretation

(see e.g. [9, 34]) by accommodating inductive-recursive defined sets. Since IR allows one to build universes as fixed point, which in the type-as-sets interpretation correspond to inaccessible cardinals, one naturally expect the model to be strong enough to support the existence of inaccessible fixed points. Thus, in addition to the usual axioms of ZFC Dybjer and Setzer postulate the existence of one Mahlo cardinal to ensure that such a fixed point can actually be reached.
We divide this section into two: in Section 3.3.1 we interpret the Logical Framework extended with rules for IR; in Section 3.3.2 we adjust Dybjer and Setzer's original proof to a more categorical setting to actually prove existence of initial algebras for the interpretation of IR functors in this model.

3.3.1 The interpretation of the Logical Framework

When building set-theoretic models, initial segments of the cumulative hierarchy (see Definition 3.1) are crucial. On the one hand, as already discussed in Section 3.1.1, they can play the role of actual models depending on the properties we want to hold in the model. On the other hand, closure properties of a particular level V_α are uniquely determined by properties of the ordinal α. Therefore, before giving the actual interpretation, we recall the properties of some cardinals which play a relevant role when interpreting type theory inside set theory.

Definition 3.26 (cofinality)**.** A map $f : \alpha \to \beta$ is said to be cofinal if it is increasing and unbounded. The cofinality of β, $\mathsf{cf}(\beta)$, is the least α such that there exists a cofinal map $f : \alpha \to \beta$. ■

Definition 3.27 (regular)**.** An ordinal is regular if $\mathsf{cf}(\kappa) = \kappa$. ■

The cofinality of an ordinal is always a cardinal. Therefore regular ordinals are cardinals. These cardinals are closed under union of smaller sets indexed over smaller sets:

Lemma 3.28. A cardinal κ is regular if and only if it cannot be obtained as $\bigcup_{i<\beta} \alpha_i$ where $|\beta| < \kappa$ and $|\alpha_i| < \kappa$ for all $i < \beta$.

This property makes regular cardinal particularly relevant for fixed point arguments (cf. [6]). And it also explains why *regular universes*, i.e. universes closed under Σ-sets, as seen in Example 3.2, are the type-theoretic counterparts of regular cardinals.

Definition 3.29 (Inaccessible cardinal)**.** A cardinal κ is weakly inaccessible if it is regular and a limit cardinal, i.e. if $\alpha < \kappa$ then $\alpha^+ < \kappa$, where α^+ is the next cardinal after α. The cardinal κ is inaccessible (or strongly inaccessible) if it is regular and a strong limit, i.e. if $\lambda < \kappa$ then $2^\lambda < \kappa$. ■

We assume the generalized continuum hypothesis which states that $2^{\aleph_\alpha} = \aleph_{\alpha+1}$. Under this hypothesis the two notion of weakly inaccessible and inaccessible cardinal coincides.
In the types-as-sets interpretation, inaccessible cardinals can be used to interpret (a hierarchy of) type theoretic universes [34, 9].
As already mentioned, we will need a Mahlo cardinal to build the model. Therefore for the sake of completeness we recall its definition even if it will not be needed for the argument. Instead we will exploit a crucial property of Mahlo cardinals recalled below in lemma 3.32.

Definition 3.30. A set $C \subseteq \kappa$ is *closed* in κ if, for $\alpha < \kappa$, $\sup(C \cap \alpha) = \alpha \neq \emptyset$ then $\alpha \in C$. The set C is *unbounded* in κ if for every $\alpha < \kappa$ there exists $\beta \in C$ with $\alpha < \beta$. A set $C \subseteq \kappa$ is *stationary* in κ, for κ an uncountable cardinal, if C intersects every closed and unbounded set in κ. ■

Definition 3.31 (Mahlo cardinal)**.** A cardinal κ is Mahlo if the set $\{\alpha < \kappa \,|\, \alpha \text{ is inaccessible}\}$ is stationary in κ. ■

The crucial properties of Mahlo cardinal, used in the proof of Lemma 3.38 below, is the following:

Lemma 3.32. Every function $f : \mathsf{M} \to \mathsf{M}$ which is normal, i.e. increasing and continous at limit, has an inaccessible fixed point.

The types-as-sets interpretation of the Logical Framework

The basic idea of the types-as-sets interpretation is to associate to all type theoretic constructions their obvious set-theoretic counterparts in a strong

enough classical axiomatic set theory: each type is interpreted as a set, the judgement $a : A$ as $a \in A$, $a \equiv_A b$ as a truth value expressing that a and b are (or are not) equal elements of A. Dependent sums and dependent function types are interpreted as the set-theoretic cartesian coproduct and product respectively.

To be more precise we interpret types as objects of $\mathbf{Set}[\mathrm{V}_{\mathsf{M}^{+\mathsf{inacc}}}]$ where $\mathsf{M}^{+\mathsf{inacc}}$ is the next inaccessible after the Mahlo cardinal M. However, as noted by Dybjer and Setzer, all types could be interpreted inside $\mathbf{Set}[\mathrm{V}_\Lambda]$ where Λ is the first (non regular) fixed point after M of the function $\lambda\,\alpha.\,\aleph_\alpha$. For simplicity we will not do this here.

To simplify the presentation we use a partial interpretation function $[\![-]\!]$ but the interpretation of derivable judgement will be always defined (Theorem 3.33). Each expression A which contains free variables has an interpretation $[\![A]\!]_\rho$ depending on an *assignment* ρ, that is a function mapping the set of free variables to elements in $\mathrm{V}_{\mathsf{M}^{+\mathsf{inacc}}}$. An assignment, ρ, can be extended when needed: we write $\rho_{[x \mapsto a]}$ for the assignment ρ extended by mapping the variable x to the set a:

$$\rho_{[x\mapsto a]}(y) = \begin{cases} a & \text{if } y = x \\ \rho(y) & \text{otherwise} \end{cases}$$

For every expression A of the Logical Framework and every assignment ρ we will give an interpretation $[\![A]\!]_\rho$, regardless if $A : \mathsf{type}$ or $A : B$ is derivable or not. The interpretation function, however, might be undefined.

We write $[\![A]\!]_\rho \simeq [\![B]\!]_\rho$ for partial equality between interpreted expressions, i.e. $[\![A]\!]_\rho \simeq [\![B]\!]_\rho$ means $[\![A]\!]_\rho \downarrow \leftrightarrow [\![B]\!]_\rho \downarrow$ and, when both are defined, written $[\![A]\!]_\rho \downarrow, [\![B]\!]_\rho \downarrow$, we have $[\![A]\!]_\rho = [\![B]\!]_\rho$. Writing $[\![A]\!]_\rho :\simeq B$ we mean that the interpretation of A is defined to be B provided B is defined, and undefined

otherwise. The interpretation is defined in the following way:

$[\![\mathsf{type}]\!]_\rho =_{\mathsf{def}} \mathrm{V}_{\mathsf{M}^{+\mathsf{inacc}}}$	$[\![\mathsf{Set}]\!]_\rho =_{\mathsf{def}} \mathrm{V}_{\mathsf{M}}$
$[\![(\Pi\, x : A)B]\!]_\rho :\simeq \Pi\, y \in [\![A]\!]_\rho.\, [\![B]\!]_{\rho[x\mapsto y]}$	$[\![(\Sigma\, x : A)B]\!]_\rho :\simeq \Sigma\, y \in [\![A]\!]_\rho.\, [\![B]\!]_{\rho[x\mapsto y]}$
$[\![\lambda\, x : A.b]\!]_\rho :\simeq \lambda y.\, [\![A]\!]_\rho [\![b]\!]_{\rho[x\mapsto y]}$	$[\![(a, b)]\!]_\rho. \simeq ([\![a]\!]_\rho, [\![b]\!]_\rho)$
$[\![b(a)]\!]_\rho :\simeq [\![b]\!]_\rho([\![a]\!]_\rho)$	$[\![\pi_i(z)]\!]_\rho :\simeq \pi_i [\![z]\!]_\rho \quad (i = 0, 1)$
$[\![\mathsf{N}_0]\!]_\rho =_{\mathsf{def}} \emptyset$	$[\![\mathsf{N}_1]\!]_\rho. \simeq \{\emptyset\}$
$[\![\mathsf{N}_2]\!]_\rho =_{\mathsf{def}} \{\emptyset, \{\emptyset\}\}$	$[\![\mathsf{0}_1]\!]_\rho = [\![\mathsf{0}_2]\!]_\rho =_{\mathsf{def}} \emptyset$
$[\![\mathsf{1}_2]\!]_\rho =_{\mathsf{def}} \{\emptyset\}$	$[\![x]\!]_\rho :\simeq \rho(x)$

Context are interpreted as sets of assignments as follows:

$$[\![(\,)]\!] =_{\mathsf{def}} \emptyset \qquad [\![(\Gamma, x : A)]\!] :\simeq \{\rho[x \mapsto a] \mid \rho \in [\![\Gamma]\!] \wedge a \in [\![A]\!]_\rho\}$$

Given this interpretation, we can state the following soundness theorem:

Theorem 3.33 (Theorem 1 in Section 6.2 in [38])**.** The interpretation of valid judgments is always defined:

(a) If $\vdash \Gamma$ context, then $[\![\Gamma]\!]_\rho \downarrow$, and if $\vdash \Gamma = \Delta$ context, then $[\![\Gamma]\!]_\rho \downarrow$ and $[\![\Delta]\!]_\rho \downarrow$ and it holds $[\![\Gamma]\!]_\rho = [\![\Delta]\!]_\rho$.

(b) If $\Gamma \vdash A : \mathsf{type}$, then $[\![\Gamma]\!]_\rho \downarrow$, and $\forall\, \rho \in [\![\Gamma]\!]_\rho$ it holds $[\![A]\!]_\rho \in [\![\mathsf{type}]\!]_\rho$. If $\Gamma \vdash A = B : \mathsf{type}$, then $[\![\Gamma]\!]_\rho \downarrow$, and $\forall\, \rho \in [\![\Gamma]\!]_\rho$ it holds $[\![A]\!]_\rho = [\![B]\!]_\rho \in [\![\mathsf{type}]\!]_\rho$.

(c) If $\Gamma \vdash x : A$, then $[\![\Gamma]\!]_\rho \downarrow$ and $\forall\, \rho \in [\![\Gamma]\!]_\rho\ [\![x]\!]_\rho \in [\![A]\!]_\rho$. If $\Gamma \vdash a = b : A$, then $[\![\Gamma]\!]_\rho \downarrow$ and $\forall\, \rho \in [\![\Gamma]\!]_\rho$ it holds $[\![a]\!]_\rho = [\![b]\!]_\rho \in [\![A]\!]_\rho$

(d) falsehood is not derivable.

Interpretations of the theory of IR

To interpret terms which come from the extension of the Logical Framework with the theory of IR we proceed as follows: we define $[\![\mathsf{IR}(I, O)]\!]_\rho :\simeq [\![\mathsf{IR}]\!]_\rho([\![I]\!]_\rho, [\![O]\!]_\rho)$ where, given sets $I, O \in \mathrm{V}_{\mathsf{M}^{+\mathsf{inacc}}}$ the set $[\![\mathsf{IR}]\!]_\rho(I, O)$ is the least set such that

$$\begin{aligned} [\![\mathsf{IR}]\!]_\rho(I, O) = \;& O \\ & + \quad \Sigma\, A \in [\![\mathsf{Set}]\!]_\rho.(\, A \to [\![\mathsf{IR}]\!]_\rho(I, O)) \\ & + \quad \Sigma\, A : [\![\mathsf{Set}]\!]_\rho.(A \to I) \to [\![\mathsf{IR}]\!]_\rho(I, O) \end{aligned}$$

which we get by iterating κ times the appropriate functor corresponding to the inductive definitions of the type $\mathsf{IR}(I, O)$. Here κ is the least regular which bounds the cardinality of A and of $A \to I$ for all $A \in [\![\mathsf{Set}]\!]_\rho$. Given this set we define the interpretation of the constructors as follows

$$\begin{aligned} [\![\iota]\!]_\rho\, o &:\simeq \mathsf{in}_0\, o \\ [\![\sigma]\!]_\rho\, A\, f &:\simeq \mathsf{in}_1(A, f) \\ [\![\delta]\!]_\rho\, B\, F &:\simeq \mathsf{in}_2(B, F) \end{aligned}$$

Given an $\mathsf{IR}(I, O)$-code γ, the corresponding functor $[\![\gamma]\!]$ is interpreted accordingly as a functor $\mathbf{Fam}\,[\![I]\!]_\rho \to \mathbf{Fam}\,[\![O]\!]_\rho$ where $\mathbf{Fam}\,[\![I]\!]_\rho$ is the category whose objects are pairs (X, T) where $X \in [\![\mathsf{Set}]\!]_\rho$ and $T : X \to [\![I]\!]_\rho$, and whose morphisms are commuting triangles. For simplicity in the following we write $\mathbf{Fam}|D|$ in place of $\mathbf{Fam}\,[\![D]\!]_\rho$. In the next section we show that these functors $\mathbf{Fam}|I| \to \mathbf{Fam}|O|$ indeed have an initial algebra.

3.3.2 Existence of initial algebras

To build an initial algebra for an IR functor $[\![\gamma]\!]$, where $\gamma : \mathsf{IR}(D, D)$, we adopt the standard argument: we consider the initial sequence (cf. Definition 2.23) associated to $[\![\gamma]\!]$

$$(\emptyset, !) \to [\![\gamma]\!](\emptyset, !) \to [\![\gamma]\!]^2(\emptyset, !) \ldots$$

where $(\emptyset, !)$ is the initial object in $\mathbf{Fam}|D|$. We prove that $[\![\gamma]\!]$ is κ-continuous for an ordinal κ we determine, and we iterate this sequence up to κ. The initial algebra for the functor $[\![\gamma]\!]$ is then given by the directed colimit of this sequence.

Some technicalities in the above argument are introduced by the way IR-functors are defined. The action of $[\![\delta\, A\, F]\!]$ on an input family (X, T) is a coproduct whose index set is the function space $A \to X$ depending both on A and on X, the index set of the input family (X, T). Therefore, as we iterate the functor $[\![\delta\, A\, F]\!]$ on an input, the index set of this coproduct grows (exponentially). For this reason, to ensure that the iteration of the initial sequence will eventually reach a fixed point we first have to bound the size of the index sets which might depend on the input family. Following Dybjer and Setzer we recursively collect the index sets for a code γ and a

family (X, T) into a set $\mathsf{Aux}(\gamma, (X, T))$ (see Definition 3.34). We prove that the functor $[\![\gamma]\!]$ is κ-continuous (Lemma 3.37), under the assumption that an inaccessible bound κ for the cardinality of the index sets exists. We crucially use the assumption that a Mahlo cardinal exists and its property (Lemma 3.32) to discharge this assumption and find the sought inaccessible bound for the index sets (Lemma 3.38). Finally, we conclude the proof using the standard argument (Lemma 3.39).

Definition 3.34. Given an IR code γ and an object (X, T) of $\mathbf{Fam}|D|$, the set $\mathsf{Aux}(\gamma, (X, T))$ which collects all possible index sets for the code γ and the family (X, T) is defined by induction on the structure of γ:

$$\begin{aligned}
\mathsf{Aux}(\iota\, d, (X, T)) &= \emptyset \\
\mathsf{Aux}((\sigma\, A\, f), (X, T)) &= \bigcup_{a \in A} \mathsf{Aux}(f\, a, (X, T)) \\
\mathsf{Aux}((\delta\, A\, F), (X, T)) &= \{A\} \cup \bigcup_{f\colon A \to X} \mathsf{Aux}(F(T \circ f), (X, T))
\end{aligned}$$

■

Remark 3.35. Since all sets appearing in the IR codes live in $[\![\mathsf{Set}]\!]_\rho = \mathrm{V}_{\mathsf{M}}$ we have that $\mathsf{Aux}(\gamma, (X, T)) \subseteq \mathrm{V}_{\mathsf{M}}$.

We define Aux differently compared to Dybjer and Setzer [38]: we collect all the sets we are interested in, whereas they use products and coproducts to build one big set that contain all interesting sets.

Notation 3.36. Given an ordinal α, we indicate with $(X^\beta, T^\beta)_{\beta<\alpha}$ an α-sequence of objects of $\mathbf{Fam}|D|$, i.e. a sequence of objects of $\mathbf{Fam}|D|$ indexed over ordinals less than α. A *monotone* α-sequence is an α-sequence such that for each $\alpha < \beta$, $X^\alpha \subseteq X^\beta$ and $T_\beta \circ \mathsf{inj}_\alpha = T_\alpha$ where inj_α is the injection given by $X^\alpha \subseteq X^\beta$. We indicate with $\bigvee_{\beta<\alpha}(X^\beta, T^\beta)$ the directed colimit of the sequence $(X^\beta, T^\beta)_{\beta<\alpha}$, which is the family explicitly given by $(\bigcup_{\beta<\alpha} X^\beta, T^\alpha)$ where $T^\alpha(x) =_{\mathsf{def}} T^\beta(x)$ for $x \in X^\beta \subseteq \bigcup_{\beta<\alpha} X^\beta$.

Lemma 3.37. Let κ be inaccessible and $(X^\alpha, T^\alpha)_{\alpha<\kappa}$ be a monotone κ-sequence of objects of $\mathbf{Fam}|D|$. Assume for some $\alpha_0 < \kappa$ that

$$\mathsf{Aux}(\gamma, (X^\alpha, T^\alpha)) \subseteq \mathrm{V}_\kappa \tag{3.6}$$

for all $\alpha_0 \leq \alpha < \kappa$. Then $[\![\gamma]\!]$ is κ-continuous i.e.

$$[\![\gamma]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) \cong \bigvee_{\alpha<\kappa} [\![\gamma]\!](X^\alpha, T^\alpha) \ .$$

Proof. We prove the Lemma by induction over γ:

- $\gamma = \iota\, d$: we have

$$[\![\iota\ d]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) = 1 = \bigvee_{\alpha<\kappa} 1 = \bigvee_{\alpha<\kappa} [\![\iota\ d]\!](X^\alpha, T^\alpha) \ .$$

- $\gamma = \sigma\, A\, f$. By hypothesis 3.6 we know $\mathsf{Aux}(\sigma_A f, (X^\alpha, T^\alpha)) \subseteq \mathrm{V}_\kappa$. Therefore, by Definition 3.34 it follows $\mathsf{Aux}(f\, a, (X^\alpha, T^\alpha)) \subseteq \mathrm{V}_\kappa$ for all $\alpha_0 \leq \alpha < \kappa$ and $a \in A$. We can now compute as follows:

$$\begin{aligned}
[\![\sigma\, A f]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) &= (\Sigma\, a \in A)[\![f(a)]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) \\
&\overset{\text{I.H.}}{\cong} (\Sigma\, a \in A) \bigvee_{\alpha<\kappa} [\![f\ a]\!](X^\alpha, T^\alpha) \\
&\overset{(1)}{=} \bigvee_{\alpha<\kappa} (\Sigma\, a \in A)[\![f\ a]\!](X^\alpha, T^\alpha) \\
&= \bigvee_{\alpha<\kappa} [\![\sigma\, A\, f]\!](X^\alpha, T^\alpha)
\end{aligned}$$

  where (1) holds since Σ is a left adjoint and therefore preserves colimits (cf. Remark 3.10 item 3).

- $\gamma = \delta\, A\, F$. By hypothesis (3.6) we know $\mathsf{Aux}(\delta\, A\, F, (X^\alpha, T^\alpha)) \subseteq \mathrm{V}_\kappa$. By Definition 3.34 it follows $A \in \mathrm{V}_\kappa$ and $\mathsf{Aux}(F(T \circ g)(X^\alpha, T^\alpha)) \subseteq \mathrm{V}_\kappa$ for all $\alpha_0 \leq \alpha < \kappa$ and $g : A \to X^\alpha$. Since $A \in \mathrm{V}_\kappa$ then $|A| < \kappa$. Therefore, by regularity of κ we can factor the function $g : A \to \bigcup_{\alpha<\kappa} X^\alpha$ through one of the injections $\mathsf{inj}_\beta : X^\beta \to \bigcup_{\alpha<\kappa} X^\alpha$, i.e.

$$g = \mathsf{inj}_\beta \circ f \tag{3.7}$$

  for some $f : A \to X^\beta$ and $\beta < \kappa$. Without loss of generality we can assume $\alpha_0 \leq \beta$ (if not, choose $\beta' =_{\mathsf{def}} \alpha_0$; we still have $f : A \to X^{\beta'}$ since $\beta < \beta'$, hence $X^\beta \subseteq X^{\beta'}$).

$$
\begin{aligned}
[\![\delta\, A\, F]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) &\overset{(1)}{=} [\![\delta\, A\, F]\!](\bigcup_{\alpha<\kappa} X^\alpha, T^\kappa) \\
&= (\Sigma\, g : A \to \bigcup_{\alpha<\kappa} X^\alpha)[\![F(T^\kappa \circ g)]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) \\
&\overset{\text{I.H.}}{\cong} (\Sigma\, g : A \to \bigcup_{\alpha<\kappa} X^\alpha) \bigvee_{\alpha<\kappa} [\![F(T^\kappa \circ g)]\!](X^\alpha, T^\alpha) \\
&\overset{(2)}{\cong} (\Sigma\, f : A \to X^\alpha) \bigvee_{\alpha<\kappa} [\![F(T^\kappa \circ \mathsf{in}_\alpha \circ f)]\!](X^\alpha, T^\alpha) \\
&\overset{(3)}{\cong} (\Sigma\, f : A \to X^\alpha) \bigvee_{\alpha<\kappa} [\![F(T^\alpha \circ f)]\!](X^\alpha, T^\alpha) \\
&\overset{(4)}{\cong} \bigvee_{\alpha<\kappa} (\Sigma\, f : A \to X^\alpha)[\![F(T^\alpha \circ f)]\!](X^\alpha, T^\alpha) \\
&= \bigvee_{\alpha<\kappa} [\![\delta\, A\, F]\!](X^\alpha, T^\alpha)
\end{aligned}
$$

Where (1) holds by the definition of $\bigvee_{\alpha<\kappa}(X^\alpha, T^\alpha)$, (2) for equation (3.7), (3) by definition of T^κ and (4) since Σ is a left adjoint and therefore preserves colimits (cf. Remark 3.10 item 3). □

Lemma 3.37 tells us that we can prove $[\![\gamma]\!]$ to be κ-continuous if we can find a bound for the index sets appearing in a monotone κ-sequence. In the next lemma we prove that, under the assumption that a Mahlo cardinal M exists, such a bound can be found for the monotone sequence we are interested in, namely the initial sequence associated to $[\![\gamma]\!]$.

Lemma 3.38. Let γ be an IR code and $(X^\alpha, T^\alpha)_\alpha =_{\mathsf{def}} ([\![\gamma]\!]^\alpha(\emptyset, !))_\alpha$ the initial sequence of the associated functor. There exists an inaccessible κ such that, for all $\alpha < \kappa$:

$$\mathsf{Aux}(\gamma, X^\alpha, T^\alpha) \subseteq \mathrm{V}_\kappa$$

Proof. The strategy for the proof is as follows: we define an increasing function $f\colon \mathrm{ON} \to \mathrm{ON}$, which tells you how much further up the cumulative hierarchy one need to go to bound X^α after one iteration of $[\![\gamma]\!]$. The important property of f will be

$$\text{if } X^{\beta'} \subseteq \mathrm{V}_\beta \text{ then } X^{\beta'+1} \cup \mathsf{Aux}(\gamma, X^{\beta'}, T^{\beta'}) \subseteq \mathrm{V}_{f(\beta)} \tag{3.8}$$

for all $\beta' < \mathsf{M}$. We then show that M actually bounds f, i.e. $f : \mathsf{M} \to \mathsf{M}$ and use the property of M (cf. Lemma 3.32) to find a fixed point κ of f. Finally we show $\mathsf{Aux}(\gamma, X^\alpha, T^\alpha) \subseteq \mathrm{V}_\kappa$ by induction on α.
The function $f : \mathrm{ON} \to \mathrm{ON}$ is defined by transfinite recursion:

$$f(\beta) = \min\{\alpha \mid (\forall \beta' < \beta)\big(f(\beta') < \alpha\big) \wedge \\ (\forall \beta' < \mathsf{M})\big(X^{\beta'} \subseteq \mathrm{V}_\beta \implies X^{\beta'+1} \cup \mathsf{Aux}(\gamma, X^{\beta'}, T^{\beta'}) \subseteq \mathrm{V}_\alpha\big)\}$$

The first conjunct makes sure that f is increasing, and the second makes (3.8) true.
Notice that $f : \mathsf{M} \to \mathsf{M}$ follows from inaccessibility of M. Indeed, for $\beta < \mathsf{M}$ and for each β' validating (3.8), the set $B =_{\mathsf{def}} \{X^{\beta'} \subseteq \mathrm{V}_\beta \mid \beta' < \mathsf{M}\} \in \mathrm{V}_{\beta+1} \subseteq \mathrm{V}_\mathsf{M}$. Using Remark 3.35, we also have, for each $\beta' \in \mathsf{M}$, $X^{\beta'+1} \cup \mathsf{Aux}(\gamma, X^{\beta'}, T^{\beta'}) \subseteq \mathrm{V}_\mathsf{M}$ and hence $X^{\beta'+1} \cup \mathsf{Aux}(\gamma, X^{\beta'}, T^{\beta'}) \subseteq \mathrm{V}_{\alpha_{\beta'}}$ for some $\alpha_{\beta'} < \mathsf{M}$ since M is a limit. Thus $f(\beta) \leq \sup_{\beta'} \alpha_{\beta'} < \mathsf{M}$ since M is regular.
So f is an increasing function on M, however f need not be continuous at limits, hence not normal and the property of M stated in Lemma 3.32 might not apply. To fix this, we define a new function θ: if $\alpha < \mathsf{M}$ let $\theta(\alpha) = f^\alpha(0)$. We can now prove that $\theta : \mathsf{M} \to \mathsf{M}$, and θ is normal. We have $\theta(\alpha) < \mathsf{M}$ 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 continuous at limits by definition.
Hence by the Mahlo property, θ has an inaccessible fixed point $\kappa < \mathsf{M}$.
Finally, we can prove that κ bounds f, i.e. $f : \kappa \to \kappa$. Assume $\alpha < \kappa$. Since κ is a limit, $\alpha < \beta$ for some $\beta < \kappa$. Furthermore since θ is strictly increasing then $\beta \leq \theta(\beta)$. We can thus conclude noticing:

$$f(\alpha) < f(\beta) \leq f(\theta(\beta)) = \theta(\beta + 1) < \theta(\kappa) = \kappa$$

This, combined with (3.8) gives us, for all $\beta < \kappa$:

$$\text{if } X^{\beta'} \subseteq \mathrm{V}_\beta \text{ then } X^{\beta'+1} \cup \mathsf{Aux}(\gamma, X^{\beta'}, T^{\beta'}) \subseteq \mathrm{V}_\kappa \tag{3.9}$$

We are left to check that for any element X^α of the initial sequence $X^\alpha \subseteq \mathrm{V}_\kappa$ for all $\alpha < \kappa$; by (3.9), it then immediately follows the thesis, i.e.

$\mathsf{Aux}(\gamma, X^\alpha, T^\alpha) \subseteq \mathrm{V}_\kappa$. We prove $X^\alpha \subseteq \mathrm{V}_\kappa$, for all $\alpha < \kappa$, by induction on α:

- If $\alpha = 0$, then $X^0 = \emptyset \subseteq \mathrm{V}_\kappa$.
- If $\alpha = \beta + 1$, then $X^\beta \subseteq \mathrm{V}_\kappa$ by the induction hypothesis, and therefore $X^\beta \subseteq \mathrm{V}_\lambda$ for some $\lambda < \kappa$, since κ is a limit. By (3.9) we get $X^{\beta+1} \subseteq \mathrm{V}_\kappa$.
- If $\alpha = \lambda$ limit, then $X^\lambda = \bigcup_{\beta<\lambda} X^\beta$ and, by induction hypothesis $X^\beta \subseteq \mathrm{V}_\kappa$ for all $\beta < \lambda$. If $x \in \bigcup_{\beta<\lambda} X^\beta$ then $x \in X^\beta$ for some $\beta < \lambda$ and therefore $x \in \mathrm{V}_\kappa$ by the induction hypothesis. □

Theorem 3.39. Let γ be an IR code. The functor $[\![\gamma]\!]$ has an initial algebra.

Proof. By Lemma 3.37, we get a κ such that Lemma 3.38 holds, i.e. $[\![\gamma]\!]$ is κ continuous. Thus $\bigvee_{\alpha<\kappa}(X^\alpha, T^\alpha)$ is a fixed point for $[\![\gamma]\!]$:

$$\begin{aligned}
[\![\gamma]\!] \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) &= \bigvee_{\alpha<\kappa} [\![\gamma]\!](X^\alpha, T^\alpha) \\
&= \bigvee_{\alpha<\kappa} (X^{\alpha+1}, T^{\alpha+1}) \\
&= \bigvee_{\alpha<\kappa} (X^\alpha, T^\alpha) \ ,
\end{aligned}$$

and by Theorem 2.25 it is an initial algebra. □

Thus, we can now conclude that the model interpreting the Logical Framework suffices also to interpret inductive-recursive sets.

Corollary 3.40. The interpretation of the Logical Framework as given in Section 3.3.1 is sound with respect of the theory of IR, asserting existence of initial algebras for IR-functors.

3.4 Summary and discussion

In this section we recalled the basics of the theory of IR. We specified its syntax in terms of codes and its semantics in terms of functors between families of sets. We gave examples of inductive-recursive defined sets and we

recalled Dybjer and Setzer set-theoretic model of the theory, adjusting the proof to adapt to a more categorical framework.

As for possible future work there seems to be space to improve the set-theoretic model. Indeed the current model seems to require much more proof theoretic strength than is actually needed. Dybjer and Setzer [38] have suggested that a constructive model could be obtained via a realizability interpretation in Kripke-Platek set theory extended by a recursive Mahlo ordinal and ω admissibles above. Another possibility would be to extend Aczel's interpretation of $\mathsf{MLW}^{+}\mathsf{u}_{<\omega}$ in $\mathsf{CZF}^{+}\mathsf{u}_{<\omega}$ by a constructive version of Mahlo cardinals [88]. This setting would also open the possibility to investigate the *set-as-trees* (see e.g. [45]) interpretation of constructive set theories into a type theory extended with IR.

Chapter 4

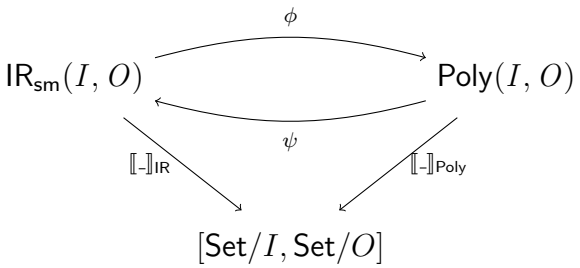
Small induction-recursion

Abstract In this chapter we introduce and investigate a sub-theory of IR which we dub *small induction-recursion* or Small IR for short. We introduce a coding scheme for the syntax and a functorial semantics for small inductive-recursive definitions. By introducing morphisms in the syntax of Small IR we are able to define a category and to prove an equivalence between the category of $\mathsf{IR}_{\mathsf{sm}}(I, O)$ and the category $\mathsf{Poly}(I, O)$ for any set I and O.

4.1 Introduction

In section 2.2.3 we showed on the one hand how dependent polynomials and indexed containers and their respective functorial interpretations represent neat and well-structured semantics for indexed inductive types. On the other hand, in Chapter 3 we saw how the theory of induction-recursion provides another extremely powerful tool to define even more sophisticated types, among which we remarkably find universes.
It is natural to ask what the relationship between these different theories of data types is. The equivalence between dependent polynomials and indexed containers is well established. But what is their relationship with induction recursion? Can we characterise those inductive-recursive definitions which correspond to dependent polynomials and indexed containers? The aim of this Chapter is to address precisely these questions. As we will see, dependent polynomials and indexed containers correspond exactly to inductive-recursive

definitions, where the smallness refers to the size of the target-type D of the recursively defined function $T : X \to D$ simultaneously defined with the type X. To appropriately state this correspondence in terms of an equivalence of categories we introduce in Section 4.2 the category $\mathsf{IR}_{\mathsf{sm}}(I, O)$ of small inductive-recursive codes with indices I and O or $\mathsf{IR}_{\mathsf{sm}}(I, O)$-codes for short, or simply $\mathsf{Small}\ \mathsf{IR}$-codes if the indices can be inferred from the context. One of the contribution of this section is then the definition of morphisms between $\mathsf{IR}_{\mathsf{sm}}(I, O)$-codes which smoothly scale also to the (large) setting of $\mathsf{IR}(I, O)$-codes. As it happens in the theories of dependent polynomials and indexed containers, we establish a tight correspondence between *syntactic* morphisms and *semantics* morphisms: indeed morphisms of $\mathsf{IR}_{\mathsf{sm}}(I, O)$-codes full and faithfully represent natural transformations between functors which interpret the corresponding codes. The main result of this chapter in Section 4.3 consists in establishing an equivalence between the category $\mathsf{Poly}(I, O)$ of dependent polynomials and the category $\mathsf{IR}_{\mathsf{sm}}(I, O)$ of small inductive-recursive codes respectively with same indices I and O as pictured in the following commuting diagram representing the statement of Theorem 4.17:



This result is not merely of theoretical importance: at a practical level, while systems such as Agda accept induction-recursion, some systems, e.g. Coq, do not. This result gives a simple way to add small induction-recursion to Coq by showing how to translate such definitions into indexed containers or dependent polynomials. Programmers are then free to convert definitions between the two forms, according to which works better for their own applications.

4.2 The category of Small IR codes

A category is always given by specifying its objects and its morphisms. Therefore we start our investigation of the category of Small IR with the definition of its objects, i.e. the $\mathsf{IR}_{\mathsf{sm}}(I, O)$-codes.

4.2.1 Small IR-codes

Definition 4.1. A Small IR-code is an $\mathsf{IR}(I, O)$ code where both the indices I and O are of type Set. To emphasize this we indicate with $\mathsf{IR}_{\mathsf{sm}}(I, O)$ the type $\mathsf{IR}(I, O)$ where $I, O : \mathsf{Set}$. ■

Small IR-functors

The restriction on the *size* of the indices of a Small IR-code allows us to regard the semantics introduced in Section 3.2.2 from another perspective. Indeed, when $I : \mathsf{Set}$, the category $\mathsf{Fam}|I|$ simply becomes the slice category Set/I. This simple observation lets us have more freedom in the semantics of Small IR: not only we can interpret these codes as functors $\mathsf{Set}/I \to \mathsf{Set}/O$, but thanks to the equivalence between slices and indexed sets we can also interpret Small IR codes as functors $\mathsf{Set}^I \to \mathsf{Set}^O$. Below we recall these two semantics.

Definition 4.2 (slice semantics)**.** Let $I, O : \mathsf{Set}$, $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$, and (X, f) an object of Set/I. The action of the functor

$$[\![\gamma]\!] : \mathsf{Set}/I \to \mathsf{Set}/O$$

on the object (X, f) is defined by induction on the structure of the code as follows:

$$\begin{aligned} [\![\iota\, o]\!]\,(X, f) &= (\mathsf{N}_1, \lambda\_ .\, o) \\ [\![\sigma\, A\, h]\!]\,(X, f) &= (\Sigma\, a : A)\, [\![h(a)]\!]\,(X, f) \\ [\![\delta\, B\, F]\!]\,(X, f) &= (\Sigma\, g : B \to X)\, [\![F\,(f \circ g)]\!]\,(X, f) \end{aligned}$$ ■

We now give the alternative presentation of the semantics of Small IR based on indexed sets: we interpret a code $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$ as a functor $(\!|\gamma|\!) : \mathsf{Set}^I \to$

Set^O. This interpretation is obtained by pre and post composing the functors in Definition 4.2 respectively with the two functors dis and $(\ )^{-1}$ as defined by equations (2.1) on page 35: we then obtain the semantics of Small IR which deploys indexed sets rather than morphisms in a slice.

Lemma 4.3 (indexed sets semantics)**.** Let $I, O : \mathsf{Set}$ and $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$. The action of the functor

$$(\!|\gamma|\!) : \mathsf{Set}^I \to \mathsf{Set}^O$$

defined by $(\!|\gamma|\!) =_{\mathsf{def}} (\ )^{-1} \circ [\![\gamma]\!] \circ \mathsf{dis}$ on the object $X : I \to \mathsf{Set}$ is given by induction on the structure of the code as follows:

$$\begin{array}{rcl} (\!|\iota\ o'|\!)\ X\ o & = & (o \equiv_O o') \\ (\!|\sigma\ A\ f|\!)\ X\ o & = & (\Sigma\, a : A)\ (\!|f(a)|\!)\ X\ o \\ (\!|\delta\ B\ F|\!)\ X\ o & = & (\Sigma\, g : B \to I)\ ((\Pi\, b : B)\ X\ (g(b))) \times (\!|F(g)|\!)\ X\ o \end{array}$$

Proof. Induction on $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$. □

4.2.2 Small IR-morphisms

We specified the objects of the category $\mathsf{IR}_{\mathsf{sm}}(I, O)$ in the previous section. Below we define morphisms between these objects, which we indicate as $\mathsf{IR}_{\mathsf{sm}}(I, O)$-morphisms or Small IR-morphisms for short. The key idea for their definition will be to mirror the categorical structure of the semantics given by functors and natural transformation back into the syntax. This approach ensures that every morphism give rise to a natural transformation between the corresponding $\mathsf{IR}_{\mathsf{sm}}(I, O)$-functors, and, in particular, pave the way to our representation theorem (Theorem 4.10).
A key observation for the appropriate definition of Small IR-morphisms is a categorical description for functors interpreting δ-codes. The following lemmas give this characterization.

Lemma 4.4. Given an object (X, f) in Set/I, there is a natural isomorphism

$$[\![\delta\ B\ F]\!]\ (X, f) \quad \cong \quad (\Sigma\, g : B \to I)\ \mathsf{Set}/I\big((B, g), (X, f)\big) \otimes [\![F(g)]\!]\ (X, f)$$

Here $\otimes$ indicates the tensor product: given a set A and an object (X, f) of Set/I the object $A \otimes (X, f)$ is the coproduct $(\Sigma\, a : A)\,(X, f)$, i.e the A-fold coproduct of the object (X, f).

Proof. We have a natural isomorphism

$$\begin{aligned}
&[\![\delta\, B\, F]\!]\,(X, f)\\
&= (\Sigma\, h : B \to X)\,[\![F(f \circ h)]\!]\,(X, f)\\
&\cong (\Sigma\, g : B \to I)\,(\Sigma\, h : B \to X)\,(g \equiv_{B\to I} f \circ h) \times [\![F(g)]\!]\,(X, f). \qquad (4.1)
\end{aligned}$$

Then observe that $\Sigma\, h : (B \to X)(g \equiv_{g:B\to I} f \circ h) \cong \mathsf{Set}/I\big((B, g), (X, f)\big)$. □

Notice how equation (4.1) uses an identity type to encode the action of $[\![\delta\, A\, F]\!]$ in the style of Henry Ford[1]: 'choose any g you want as long as it is $f \circ h$'. Lemma 4.4 lets us shed some light on the semantics of δ-codes. Indeed we can characterize the functor arising from a δ-code through a universal construction: that of left Kan extension. To this end recall that given functors $F : \mathbb{A} \to \mathbb{C}$ and $G : \mathbb{A} \to \mathbb{B}$ the left Kan extension of F along G is the functor $\mathsf{Lan}_G F : \mathbb{B} \to \mathbb{C}$ with the universal property that for every functor $H : \mathbb{B} \to \mathbb{C}$ there is a bijection $\mathbb{C}^{\mathbb{B}}(\mathsf{Lan}_G F, H) \cong \mathbb{C}^{\mathbb{A}}(F, H \circ G)$ natural in H. If (B, g) is an object in Set/I we use $((B, g) + \_)$, in the following lemma, to indicate the functor

$$\begin{array}{rccc}
((B, g) + _) : & \mathsf{Set}/I & \longrightarrow & \mathsf{Set}/I \\
& (X, f) & \longmapsto & (B + X, [g, f])
\end{array}$$

which gives, for every object (X, f), its coproduct with the object (B, g).

Theorem 4.5. There is a natural isomorphism

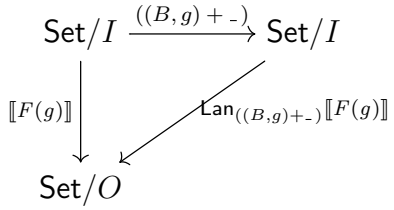
$$[\![\delta\, B\, F]\!] \;\cong\; (\Sigma\, g : B \to I)\mathsf{Lan}_{((B,g)+\_)}[\![F(g)]\!]$$

Proof. Recall that in the presence of enough categorical structure left Kan extension can be computed by means of coends: for $F : \mathbb{A} \to \mathbb{C}$, $G : \mathbb{A} \to \mathbb{B}$ we can compute $\mathsf{Lan}_G F$ as follows:

$$(\mathsf{Lan}_G F)\, b = \int^{a:\mathbb{A}} F(a) \times \mathbb{B}(G\, a, b) \qquad (4.2)$$

[1]The metaphor is due to Conor McBride (see e.g. [13]).

The particular case which interest us is represented in the following diagram



and we can compute $\mathsf{Lan}_{((B,g)+\_)}[\![F(g)]\!]$ as follows:

$$\mathsf{Lan}_{((B,g)+\_)}[\![F(g)]\!](X,f) = \\ (\Sigma\,(Y,l) : \mathsf{Set}/I)[\![F(g)]\!](Y,l) \times \mathsf{Set}/I\big((B+Y,[g,l]),(X,f)\big) \tag{4.3}$$

$$\begin{aligned}
&[\![\delta\ B\ F]\!]\,(X, f : X \to I) \\
&\overset{(1)}{\cong} (\Sigma\, g : B \to I)\mathsf{Set}/I\big((B,g),(X,f)\big) \otimes [\![F(g)]\!]\,(X,f) \\
&\overset{(2)}{\cong} (\Sigma\, g : B \to I)\big((\mathsf{Lan}_{\mathsf{Id}}[\![F(g)]\!])(X,F)\big) \times \mathsf{Set}/I\big((B,g),(X,f)\big) \\
&\overset{(3)}{\cong} \Sigma_{g:B\to I}\big(\Sigma_{(Y,l)\,:\mathsf{Set}/I}[\![F(g)]\!](Y,l) \times \mathsf{Set}/I\big((Y,l),(X,f)\big)\big) \times \mathsf{Set}/I\big((B,g),(X,f)\big) \\
&\overset{(4)}{\cong} \Sigma_{g:B\to I}\Sigma_{(Y,l)\,:\mathsf{Set}/I}\big([\![F(g)]\!]\,(Y,l) \times \mathsf{Set}/I\big((B+Y,[g,l]),(X,f)\big)\big) \\
&\overset{(5)}{=} \Sigma_{g:B\to I}\Sigma\,(Y,l) : \mathsf{Set}/I\big([\![F(g)]\!]\,(Y,l) \times \mathsf{Set}/I\big(((B,g)+_)(Y,l),(X,f)\big)\big) \\
&\overset{(6)}{\cong} (\Sigma\, g : B \to I)\big(\mathsf{Lan}_{((B,g)+_)}[\![F(g)]\!]\big)\,(X,f)
\end{aligned}$$

where (1) holds by Lemma 4.4, (2) since every functor is its own left Kan extension along the identity functor Id, (3) is given by equation (4.3), (4) by associativity of products and the universal property of coproduct, (5) by definition of $((B,g)+\_)$ and (6) by equation (4.3). □

This ensures that the definition of $\mathsf{IR}_{\mathsf{sm}}(I,O)$-morphisms we are going to give is well defined. Finally, in order to make sense of the definition of $\mathsf{IR}_{\mathsf{sm}}(I,O)$-morphisms, we need to check that $\mathsf{IR}_{\mathsf{sm}}(I,O)$-functors are closed by pre-composition with functors of the form $((B,g)+\_)$. We do this in the next lemma.

Lemma 4.6. Given a $\mathsf{IR}_{\mathsf{sm}}(I, O)$-code γ, and an object (B, g) of Set/I, there exists an $\mathsf{IR}_{\mathsf{sm}}(I, O)$-code, $\gamma \centerdot ((B, g) + \_)$, s.t.

$$[\![\gamma \centerdot ((B, g) + \_)]\!] \cong [\![\gamma]\!] \circ ((B, g) + \_)$$

Proof. We proceed by induction on the structure of the code $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$. The proof will follow the coding scheme based on containers introduced in Section 3.2.4 and its corresponding semantics. If γ is $\iota\, o$ for some $o : O$, then define $(\iota\, o) \centerdot ((B, g) + \_)$ simply as $\iota\, o$. For (X, f) in Set/I we have

$$\begin{aligned}\big([\![\iota\, o]\!] \circ ((B, g) + \_)\big)(X, f) &= [\![\iota\, o]\!](B + X, [g, f]) \\ &= (1, \lambda\_ .\, o) \\ &= [\![\iota\, o]\!](X, f) \\ &= [\![(\iota\, o) \centerdot ((B, g) + \_)]\!](X, f)\end{aligned}$$

Let γ be the code $\sigma\delta(S, P)G$ for $(S, P) : \mathsf{Cont}$ and $G : (S \lhd P)\, I \to \mathsf{IR}(I, O)$. We let $(\sigma\delta(S, P)G) \centerdot ((B, g) + \_) =_{\mathsf{def}} \sigma\delta(R, Q)J$ for the container (R, Q) and the map $J : (R \lhd Q)I \to \mathsf{IR}_{\mathsf{sm}}(I, O)$ given below. Since

$$\begin{aligned}&\big([\![\sigma\delta(S, P)G]\!] \circ ((B, g) + \_)\big)(X, f) = \\ &= (\Sigma\, x : (S \lhd P)(B + X))[\![G(((S \lhd P)[g, f])x)]\!](B + X, [g, f]),\end{aligned}$$

by the semantics of $\sigma\delta$-codes, we want $(R \lhd Q) \cong (S \lhd P)(B + X)$ which can be achieved by $(R, Q) =_{\mathsf{def}} (S, P) \cdot (B + \mathsf{N_1}, [\lambda\_ . \mathsf{N_0}, \mathsf{id}_{\mathsf{N_1}}])$, using container composition defined in Section 2.2.2. This definition gives us the following isomorphism:

$$\varphi : (S \lhd P)(B + \_) \overset{\cong}{\longleftrightarrow} (R \lhd Q) : \psi \tag{4.4}$$

Let $J : (R \lhd Q)(I) \to \mathsf{IR}(I, O)$ be the function

$$J(y) =_{\mathsf{def}} \big((G \circ [\![(S, P)]\!][g, \mathsf{id}_I] \circ \psi_I)\,(y)\big) \centerdot ((B, g) + \_)$$

constructed using the inductive hypothesis. We then have

$$
\begin{aligned}
&[\![\sigma\delta(S,P)\,G]\!] \circ ((B,g) + \_) \\
&= (\Sigma\, x : (S \lhd P)(B+X))[\![G\big(((S \lhd P)[g,f])\, x\big)]\!](B+X,[g,f]) \\
&\overset{(1)}{=} (\Sigma\, x : (S \lhd P)(B+X)) \\
&\qquad\qquad [\![G\big((S \lhd P)[g,\mathsf{id}_I](\mathsf{id}_B + f))\, x\big)]\!](B+X,[g,f]) \\
&\overset{(2)}{\cong} (\Sigma\, x : (S \lhd P)(B+X)) \\
&[\![G\big(\big((S \lhd P)[g,\mathsf{id}_I] \circ (S \lhd P)(\mathsf{id}_B + f)\big)\, x\big)]\!]_{\mathsf{C}}(B+X,[g,f]) \\
&\overset{(3)}{\cong} (\Sigma\, y : (R \lhd Q)\, X) \\
&\quad [\![G\big(\big((S \lhd P)[g,\mathsf{id}_I] \circ \psi_I \circ (R \lhd Q) f\big)\, y\big)]\!](B+X,[g,f]) \\
&\overset{(4)}{\cong} (\Sigma\, y : (R \lhd Q)\, X)[\![J\big(((R \lhd Q)\, f)\, y\big)]\!](X,f) \\
&= [\![\sigma\delta(R,Q)J]\!](X,f)
\end{aligned}
$$

where (1) holds by $[g,f] = ([g,\mathsf{id}_I] \circ (\mathsf{id}_B + f))$, (2) by the functoriality of $(S \lhd P)$, (3) by the isomorphism (4.4), and (4) by definition of J. □

We are now ready to define $\mathsf{IR}_{\mathsf{sm}}(I,O)$-morphisms by induction on the structure of the codes as follows.

Definition 4.7. Given codes $\gamma, \gamma' : \mathsf{IR}_{\mathsf{sm}}(I,O)$ the homset $\mathsf{IR}_{\mathsf{sm}}(I,O)(\gamma,\gamma')$ is defined as follows.
Morphisms from ι-codes:

1a.
$$\frac{p : o \equiv_O o'}{\Gamma_{\iota,\iota}(p) : \mathsf{IR}_{\mathsf{sm}}(I,O)(\iota\, o, \iota\, o')}$$

1b.
$$\frac{a : A \quad r : \mathsf{IR}_{\mathsf{sm}}(I,O)(\iota\, o, f(a))}{\Gamma_{\iota,\sigma}(a,r) : \mathsf{IR}_{\mathsf{sm}}(I,O)(\iota\, o, \sigma\, A\, f)}$$

1c.
$$\frac{g : B \to \mathsf{N}_0 \quad r : \mathsf{IR}_{\mathsf{sm}}(I,O)(\iota\, o, F\,(!_I \circ g))}{\Gamma_{\iota,\delta}(g,r) : \mathsf{IR}_{\mathsf{sm}}(I,O)(\iota\, o, \delta\, A\, F)}$$

where $!_I : \mathsf{N}_0 \to I$ is the unique morphism from the initial object N_0 to the set I.
Morphisms from σ-codes:

2.
$$\frac{\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O) \qquad r : (\Pi\, a : A)\mathsf{IR}_{\mathsf{sm}}(I, O)(f(a), \gamma)}{\Gamma_{\sigma,\gamma}(r) : \mathsf{IR}_{\mathsf{sm}}(I, O)(\sigma\, A\, f, \gamma)}$$

Morphisms from δ-codes:

3.
$$\frac{\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O) \qquad r : (\Pi\, g : B \to I)\mathsf{IR}_{\mathsf{sm}}(I, O)(F(g), \gamma \bullet ((B, g) + \_))}{\Gamma_{\delta,\gamma}(r) : \mathsf{IR}_{\mathsf{sm}}(I, O)(\delta\, B\, F, \gamma)}$$

■

We now explain how $\mathsf{IR}_{\mathsf{sm}}$-morphisms are interpreted as natural transformation between $\mathsf{IR}_{\mathsf{sm}}$-functors.

Lemma 4.8. The interpretation functions

$$\begin{aligned} [\![\ ]\!] &: \mathsf{IR}_{\mathsf{sm}}(I, O) \longrightarrow [\mathsf{Set}/I, \mathsf{Set}/O] \\ (\!|\ |\!) &: \mathsf{IR}_{\mathsf{sm}}(I, O) \longrightarrow [\mathsf{Set}^I, \mathsf{Set}^O] \end{aligned}$$

can be extended to $\mathsf{IR}_{\mathsf{sm}}(I, O)$-morphisms, i.e. every morphism $\tau : \mathsf{IR}_{\mathsf{sm}}(I, O)(\gamma, \gamma')$ defines a natural transformation $[\![\tau]\!] : [\![\gamma]\!] \stackrel{\cdot}{\longrightarrow} [\![\gamma']\!]$.

Proof. By induction on the structure of $\mathsf{IR}_{\mathsf{sm}}(I, O)$-morphisms we define the component of $[\![\tau]\!]$ at $(X, f)\, \mathsf{Set}/I$ as follows:

$$\begin{aligned} [\![\Gamma_{\iota,\iota}(p)]\!]_{(X,f)} &= \mathsf{id}_{\mathsf{N}_1} \\ [\![\Gamma_{\iota,\sigma}(a, r)]\!]_{(X,f)} &= \mathsf{in}_a \circ [\![r]\!]_{(X,f)} \\ [\![\Gamma_{\iota,\delta}(g, r)]\!]_{(X,f)} &= \mathsf{in}_{!_X \circ g} \circ [\![r]\!]_{(X,f)} \\ [\![\Gamma_{\sigma,\gamma}(r)]\!]_{(X,f)} &= [[\![r(a)]\!]_{(X,f)}]_{a:A} \\ [\![\Gamma_{\delta,\gamma}(r)]\!]_{(X,f)} &= [[\![r(g)]\!]_{(X,f)}]_{g:A\to I} \end{aligned}$$

Naturality of these transformation consists of a simple diagram chasing. Using the definition of $(\!|\gamma|\!)$ in Lemma 4.3 the assignment given above smoothly extends to the indexed set semantics. □

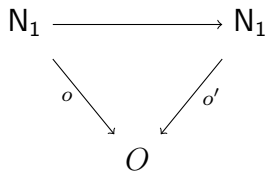
Remark 4.9. Notice that we work with simple natural transformations and not with strong natural transformation because the defnition of canonical strength for a functor in **Set** smoothly scales in the case of functors defined in Set and similarly for the respective slices categories.

The following theorem is the key ingredient to ensure that the structure of the codes given by the above definition is reflected in the functorial semantics given in Definition 4.2 and in Lemma 4.3.

Theorem 4.10. The interpretation functions $[\![\ ]\!]$ and $(\!|\ |\!)$ are injective and surjective on morphisms.

Proof. We prove the theorem by induction on the structure of $\mathsf{Small\ IR}$ morphisms using the interpretation $[\![\ ]\!]$. Since the functors dis and $(\ )^{-1}$ in equations (2.1) are full and faithful we also get injectivity and surjectivity of $(\!|\ |\!)$.

1a. The components of a natural transformation $\eta : [\![\iota\, o]\!] \dot{\to} [\![\iota\, o']\!]$ are all equal to the following morphism in Set/O



Such a morphism clearly exists if and only if $o \equiv_O o'$.

1b. Injectivity is immediate. We show that each natural transformation $\eta : [\![\iota\, o]\!] \dot{\to} [\![\sigma\, A\, f]\!]$ is uniquely determined by an element $a : A$ and a morphism $r : \mathsf{IR_{sm}}(I, O)\ (\iota\, o, f(a))$. Let η be such a natural transformation. By naturality of η we have

$$[[\![f(a)]\!]h]_{a:A} \circ \eta_{(X,k)} = \eta_{(Y,g)} \tag{4.5}$$

for every $h : (X, k) \to (Y, g)$ in Set/I. It then follows that for every (X, k) we have $\pi_0(\eta_{(X,k)}(\mathsf{0_1})) = a$ for some fixed $a : A$, in particular

define $a =_{\mathsf{def}} \pi_0(\eta_{(\mathsf{N}_0,!)}(\mathsf{0}_1))$. Composing both sides of (4.5) with the second projection we get equations

$$\pi_1 \circ [\![f(a)]\!]h]_{a:A} \circ \eta_{(X,k)} = \pi_1 \circ \eta_{(Y,g)}$$

asserting that $\pi_1 \circ \eta : [\![\iota\, o]\!] \dot{\to} [\![f(a)]\!]$ is a natural transformation. By the inductive hypothesis, $\pi_1 \circ \eta$ is equal to $[\![r]\!]$ for some $r : \mathsf{IR}_{\mathsf{sm}}(I, O)$ $(\iota\, o, f(a))$. Hence we have $\eta = [\![\Gamma_{\iota,\sigma}(a, r)]\!]$.

1c. Injectivity is immediate. We exploit the universal property of the initial object $(\mathsf{N}_0, ! : \mathsf{N}_0 \to I) : \mathsf{Set}/I$ to show that each natural transformation $\eta : [\![\iota\, o]\!] \dot{\to} [\![\delta\, B\, F]\!]$ is uniquely determined by a map $g : A \to \mathsf{N}_0$ and a map $r : \mathsf{IR}_{\mathsf{sm}}(I, O)$ $(\iota\, o, F\,(!_I \circ g))$. Given such a natural transformation η, its components are uniquely determined by $\eta_{\mathsf{N}_0,!}$, since, by naturality, we have

$$\eta_{(X,k)} = [\![\delta\, B\, F]\!]\, !_{(X,k)} \circ \eta_{(\mathsf{N}_0,!)}$$

Then observe that $\pi_0(\eta_{(\mathsf{N}_0,!)}(\mathsf{0}_1)) : A \to \mathsf{N}_0$ and

$$[\![F(!_I \circ \pi_0(\eta_{(\mathsf{N}_0,!)}(\mathsf{0}_1)))]\!]\, !_{(X,k)} \circ \pi_1 \circ \eta_{(\mathsf{N}_0,!)}$$

is the component at (X, k) of a natural transformation which by the inductive hypothesis is equal to $[\![r]\!]$ for some $r : \mathsf{IR}_{\mathsf{sm}}(I, O)(\iota\, o, F(!_I \circ \pi_0(\eta_{(\mathsf{N}_0,!)}(\mathsf{0}_1))))$. Therefore we have $[\![\eta]\!] = [\![\Gamma_{\iota,\delta}(g, r)]\!]$.

2. If we indicate by $[\mathsf{Set}/I, \mathsf{Set}/O]([\![\sigma\, A\, f]\!], [\![\gamma]\!])$ the set of natural transformation between the functors $[\![\sigma\, A\, f]\!]$ and $[\![\gamma]\!]$, we can compute as follows:

$$\begin{aligned}
[\mathsf{Set}/I, \mathsf{Set}/O]([\![\sigma\, A\, f]\!], [\![\gamma]\!]) &= [\mathsf{Set}/I, \mathsf{Set}/O](\Sigma\, a : A\, [\![f(a)]\!], [\![\gamma]\!]) \\
&= (\Pi\, a : A)\, [\mathsf{Set}/I, \mathsf{Set}/O]([\![f(a)]\!], [\![\gamma]\!]) \\
&\overset{(\star)}{=} (\Pi\, a : A)\, \mathsf{IR}_{\mathsf{sm}}(I, O)(f(a), \gamma) \\
&= \mathsf{IR}_{\mathsf{sm}}(I, O)(\sigma\, A\, f, \gamma)
\end{aligned}$$

where $(\star)$ comes from the inductive hypothesis.

3. Compute as follows:

$$
\begin{aligned}
&[\mathsf{Set}/I, \mathsf{Set}/O](\llbracket \delta\, B\, F \rrbracket, \llbracket \gamma \rrbracket) \\
&\overset{(1)}{\cong} [\mathsf{Set}/I, \mathsf{Set}/O](\Sigma\, g : B \to I.\, \mathsf{Lan}_{((B,g)+\_)} \llbracket F(g) \rrbracket, \llbracket \gamma \rrbracket) \\
&\overset{(2)}{\cong} (\Pi\, g : B \to I)[\mathsf{Set}/I, \mathsf{Set}/O](\llbracket F(g) \rrbracket, \llbracket \gamma \rrbracket \circ ((B, g) + \_)) \\
&\overset{(3)}{\cong} (\Pi\, g : B \to I)\, [\mathsf{Set}/I, \mathsf{Set}/O](\llbracket F(g) \rrbracket, \llbracket \gamma \centerdot ((B, g) + \_) \rrbracket) \\
&\overset{(4)}{\cong} (\Pi\, g : B \to I)\, \mathsf{IR}_{\mathsf{sm}}(I, O)(F(g), \gamma \centerdot ((B, g) + \_)) \\
&= \mathsf{IR}_{\mathsf{sm}}(I, O)(\delta\, B\, F, \gamma)
\end{aligned}
$$

where (1) holds by Theorem 4.5, (2) by the universal property of left Kan extensions, (3) by Lemma 4.6 and (4) by the inductive hypothesis. □

Full and faithfulness of the functions $\llbracket\ \rrbracket$ and $(\!|\ |\!)$ allows us to reflect composition of functors to composition of codes. Hence, as a corollary we have the following important result.

Corollary 4.11. $\mathsf{IR}_{\mathsf{sm}}(I, O)$-codes and their morphisms define a category. The interpretations $\llbracket\ \rrbracket$ and $(\!|\ |\!)$ define a full and faithful functor.

Remark 4.12. Notice that the definition of morphisms we gave in Definition 4.7 scales also to the *large* setting, i.e. it applies also to general $\mathsf{IR}(I, O)$-codes. The very same proof of Theorem 4.10 also goes through in the general case, i.e. we have a full and faithful functor

$$\llbracket\ \rrbracket : \mathsf{IR}(I, O) \longrightarrow [\mathsf{Fam}|I|, \mathsf{Fam}|O|]$$

Since we are not going to use general $\mathsf{IR}(I, O)$-morphisms in this thesis we preferred to introduce them here, instead of introducing them earlier and then deriving $\mathsf{IR}_{\mathsf{sm}}(I, O)$-morphisms as a special case.

4.3 The equivalence between Small IR and Poly

We divide this section into three: i) we first show how to translate dependent polynomials, and hence indexed containers, into IR-codes; and ii) we show

how every Small IR-code can be translated into a dependent polynomial. Crucially, we show that these translations preserve the functorial semantics of dependent polynomials and IR codes. Finally iii) we show how the isomorphism established in i) and ii) can be extended to a proper equivalence.

4.3.1 From Poly to Small IR

In Section 3.2.7 we saw that the $\mathsf{IR}(\mathsf{N}_1, \mathsf{N}_1)$-code

$$\sigma\, A\, (a \mapsto \delta\, B(a)\, (f \mapsto \iota\, \mathsf{0}_1))$$

represents the W-type $(\mathrm{W}\, x : A)B(x)$. Indeed the IR-functor corresponding to this code is nothing but the extension of the container (A, B). We now extend this correspondence to indexed containers and dependent polynomials, i.e. we exhibit codes representing arbitrary dependent polynomials and indexed containers. Crucially the IR-code representing a dependent polynomial with indices I and O is a Small IR-code with the same indices.

Lemma 4.13. Every dependent polynomial functor is a Small IR functor.

Proof. To prove the lemma we build a function $\psi : \mathsf{Poly}(I, O) \to \mathsf{IR}_{\mathsf{sm}}(I, O)$ which assigns to every dependent polynomial L, a code γ_L, whose interpretation, $[\![\gamma_L]\!]$, is isomorphic to the extension $\mathcal{P}_L$ of the dependent polynomial L. The code $\phi(L)$ representing $L = I \xleftarrow{r} P \xrightarrow{t} S \xrightarrow{q} O$ is:

$$\gamma_L =_{\mathsf{def}} \sigma S\, (s \mapsto \delta\, P_s\, (h \mapsto \sigma\, ((\Pi\, p : P_s)\, h(p) \equiv_I r(p))(\_ \mapsto \iota(q\, s)))))$$

To check that the corresponding Small IR-functor is isomorphic to $\mathcal{P}_L$, recall, from Remark 3.15, that, for a code $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$, the functor $[\![\gamma]\!] : \mathsf{Set}/I \longrightarrow \mathsf{Set}/O$ can be thought as a pair $([\![\gamma]\!]_0, [\![\gamma]\!]_1)$ where $[\![\gamma]\!]_0 \colon \mathsf{Set}/I \to \mathsf{Set}$ and $[\![\gamma]\!]_1 \colon (X, f) : \mathsf{Set}/I \to [\![\gamma]\!]_0(X, f) \to O$. The fiber of $[\![\gamma]\!](X, f)$ at $o : O$ is then given by the set

$$(\Sigma\, x : [\![\gamma]\!]_0(X, f))\, ((([\![\gamma]\!]_1(X, f))\, x) \equiv_O o)$$

In particular notice that the fiber at $o : O$ is given by those elements of $[\![\gamma]\!]_0$ which have accumulated an o in the ι-code at the end. We can then compute

the interpretation of the code γ_L using the semantics of Small IR given in definition 4.2 as follows:

$$
\begin{aligned}
&[\![\sigma S\,(s \mapsto \delta\, P_s\,(h \mapsto \sigma\,((\Pi\, p : P_s)\, h(p) \equiv_I r(p))(\_ \mapsto \iota(q(s)))))]\!]_0(X, f)\\
&= (\Sigma\, s : S)\,[\![\delta\, P_s\,(h \mapsto \sigma\,((\Pi\, p : P_s)\, h(p) \equiv_I r(p))(\_ \mapsto \iota(q(s))))]\!]_0(X, f)\\
&= (\Sigma\, s : S)\,(\Sigma\, g : P_s \to X)\,[\![\sigma\,((\Pi\, p : P_s)\, f(g(p)) \equiv_I r(p))(\_ \mapsto \iota(q(s)))]\!]_0(X, f)\\
&= (\Sigma\, s : S)\,(\Sigma\, g : P_s \to X)\,(\Sigma\, m : \Pi\ p : P_s.\ f(g(p)) \equiv_I r(p))\,[\![\iota(q(s))]\!]_0(X, f)\\
&\cong (\Sigma\, s : S)\,(\Sigma\, g : P_s \to X)\,(\Pi\ p : P_s)\, f(g(p)) \equiv_I r(p)\\
&\overset{(1)}{\cong} (\Sigma\, s : S)\,(\Pi\ p : P_s)(\Sigma\, x : X)\,(f(x) \equiv_I r(p))\\
&\overset{(2)}{=} (\Sigma\, s : S)\,(\Pi\ p : P_s)\, X_{r(p)}
\end{aligned}
$$

Where (1) holds by the axiom of choice and (2) by the definition of the fibre $\Delta_r f$ at p. Each element $(s, u) : [\![\gamma_L]\!]_0(X, f)$ is then mapped by $[\![\gamma_L]\!]_1$ to the element $q(s) : O$. Thus, the fiber at $o : O$ of $[\![\gamma_L]\!]$ is given by

$$
\begin{aligned}
&(\Sigma\, x : [\![\gamma_L]\!]_0(X, f))((([\![\gamma_L]\!]_1(X, f))\, x) \equiv_O o)\\
&= (\Sigma\, x : (\Sigma\, s : S)\,(\Pi\ p : P_s.\, X_r(p)))\,((([\![\gamma_L]\!]_1(X, f))\, x) \equiv_O o)\\
&\cong (\Sigma\, s : S)\,(\Pi\ p : P_s.\, X_r(p)) \times (q\, s \equiv_O o)\\
&\overset{(\star)}{=} (\Sigma\, s : S_o)\,(\Pi\ p : P_s)\, X_r(p)
\end{aligned}
$$

where $(\star)$ is by the definition of the fiber of q at $o : O$. □

Similarly, given an indexed container (S, P, n), the code $\gamma_{(S,P,n)} : \mathsf{IR}_{\mathsf{sm}}(I, O)$ representing it is the following

$$
\gamma_{(S,P,n)} =_{\mathsf{def}} \sigma\, O\ (o \mapsto \sigma\, S(o)\,(s \mapsto \delta\, P(o, s)\,(f \mapsto \sigma\,(f \equiv_{P \to I} n(o, s)\ \_ \mapsto \iota\, o))))
\tag{4.6}
$$

It can be easily checked that the interpretation of the code $\gamma_{(S,P,n)} : \mathsf{IR}_{\mathsf{sm}}(I, O)$ is isomorphic to the extension of the indexed container $[\![S, P, n]\!]_{\mathsf{IC}}$ by using the semantics of Small IR given in definition 4.3.

4.3.2 From Small IR to Poly

We now exhibit three indexed containers in $\mathsf{IC}(I, O)$ and the corresponding dependent polynomials in $\mathsf{Poly}(I, O)$ which simulate the three constructors

for a $\mathsf{IR}_{\mathsf{sm}}(I, O)$-code. In the following definition we indicate with a superscript the $\mathsf{IR}_{\mathsf{sm}}(I, O)$-code represented by the corresponding dependent polynomial.

Definition 4.14. We build a function $\phi : \mathsf{IR}_{\mathsf{sm}}(I, O) \to \mathsf{IC}(I, O)$ by recursion on the structure of the code which assigns to each code $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$, an indexed container $(S^\gamma, P^\gamma, n^\gamma)$ as follows:
if γ is $\iota\, o$ for some $o : O$ then $\phi(\iota\, o) =_{\mathsf{def}} (S^{\iota\, o}, P^{\iota\, o}, n^{\iota\, o})$ is given by :

$$S^{\iota\, o} : O \to \mathsf{Set} \qquad P^{\iota\, o} : (\Pi\, o' : O)(S^{\iota\, o}(o') \to \mathsf{Set})$$
$$S^{\iota\, o} o' =_{\mathsf{def}} (o' \equiv_O o) \qquad P^{\iota\, o}(o, s) =_{\mathsf{def}} \mathsf{N}_0;$$
$$n^{\iota\, o} : (\Pi\, o' : O)(\Pi\, s : S^{\iota\, o}(o'))(P^{\iota\, o}(o, s) \to I)$$
$$n^{\iota\, o}(o', s) =_{\mathsf{def}} !_I$$

As a dependent polynomial we can represent it as follows:

$$I \xleftarrow{!_I} \mathsf{N}_0 \xrightarrow{!_{\mathsf{N}_1}} \mathsf{N}_1 \xrightarrow{o} O$$

If γ is $\sigma A f$ for some $A : \mathsf{Set}$, $f : A \to \mathsf{IR}_{\mathsf{sm}}(I, O)$ then let $h =_{\mathsf{def}} \phi \circ f$ and define $\phi(\sigma A f) =_{\mathsf{def}} (S^{\sigma A h}, P^{\sigma A h}, n^{\sigma A h})$ as follows

$$S^{\sigma A h} : O \to \mathsf{Set} \qquad P^{\sigma A h} : (\Pi\, o : O)(S^{\sigma A h}(o) \to \mathsf{Set})$$
$$S^{\sigma A h}(o) =_{\mathsf{def}} (\Sigma a : A)\, S^{h(a)}(o) \qquad P^{\sigma A h}(o, a, s) =_{\mathsf{def}} P^{h(a)}(s);$$
$$n^{\sigma A h} : (\Pi\, o : O)(\Pi\, s : S^{\sigma A h}(o))(P^{\sigma A h}(o, s) \to I)$$
$$n^{\sigma A h}(o, s) =_{\mathsf{def}} [n^{h(a)}]_{a\,:\,A}$$

As a dependent polynomial we can represent it as follows:

$$I \xleftarrow{[r^{f(a)}]_{a:A}} (\Sigma\, a : A)\, P^{f(a)} \xrightarrow{t^{\sigma\, A\, f}} (\Sigma\, a : A)\, S^{f(a)} \xrightarrow{[q^{f(a)}]_{a:A}} O$$

where $t^{\sigma\, A\, f} =_{\mathsf{def}} (\Sigma\, a : A)\, t^{f(a)}$
If γ is $\delta A F$ for some $A : \mathsf{Set}$, $F : (A \to I) \to \mathsf{IR}_{\mathsf{sm}}(I, O)$ then let $H =_{\mathsf{def}} \phi \circ F$ and define $\phi(\delta A F) =_{\mathsf{def}} (S^{\delta A H}, P^{\delta A H}, n^{\delta A H})$ as follows:

$$S^{\delta A H} : O \to \mathsf{Set} \qquad P^{\delta A H} : (\Pi\, o : O)(S^{\delta A H}(o) \to \mathsf{Set})$$
$$S^{\delta A H} o =_{\mathsf{def}} (\Sigma\, g : A \to I)\, S^{H(g)}(o) \qquad P^{\delta A H}(g, s) =_{\mathsf{def}} A + P^{H(g)}(s);$$
$$n^{\delta A H} : (\Pi\, o : O)(\Pi\, s : S^{\delta A H}(o))(P^{\delta A H}(o, s) \to I)$$
$$n^{\delta A H}(o, s) =_{\mathsf{def}} [\,[g, n^{H(g)}]\,]_{g\,:A\to I}$$

As a dependent polynomial we can represent it as follows:

$$I \xleftarrow{r^{\delta\, B\, F}} (\Sigma\, g : B \to I)(B \times S^{F(g)}) + P^{F(g)} \xrightarrow{t^{\delta\, B\, F}} (\Sigma\, g : B \to I) S^{F(g)} \xrightarrow{q^{\delta\, B\, F}} O$$

where

$$r^{\delta\, B\, F} = [\,[g \circ \pi_0, r^{F(g)}]\,]_{g : I^B} \quad t^{\delta\, B\, F} = (\Sigma\, g : B \to I)\,[\pi_1, t^{F(g)}] \quad q^{\delta\, B\, F} = [q^{F(g)}]_{g : I^B}$$

■

The three indexed containers in Definition 4.14, represent the three basic cases to simulate an $\mathsf{IR}_{\mathsf{sm}}(I, O)$-functor.

Lemma 4.15. Every Small IR-functor is a dependent polynomial functor.

Proof. To simplify calculations we use indexed containers and the semantics of Small IR as given in Lemma 4.3. To prove the theorem we show that ϕ preserve the semantics, i.e. $(\!|\phi(\gamma)|\!) \cong [\![S^\gamma, p^\gamma, n^\gamma]\!]_{\mathsf{IC}}$:

$$\begin{aligned}
[\![\phi(\iota\, o)]\!]_{IC}\, X\, o' &= [\![(S^{\iota\, o}, P^{\iota\, o}, n^{\iota\, o})]\!]_{\mathsf{IC}}\, X\, o' \\
&= (\Sigma\, s : S^{\iota\, o}(o'))\, (\Pi\, p : P^{\iota\, o}(o', s))\, X\, (n^{\iota\, o}(o', s, p)) \\
&= (\Sigma\, m \colon (o' \equiv o))\, (\Pi\, p : \mathsf{N}_0)\, X\, (n^{\iota\, o}(o', s, p)) \\
&\overset{(1)}{\cong} (o' \equiv_O o) \\
&= (\!|\iota\, o|\!)\, X\, o'
\end{aligned}$$

where (1) holds since N_0 is an initial object.

$$\begin{aligned}
[\![\phi(\sigma\, A\, f)]\!]_{\mathsf{IC}}\, X\, o &= [\![(S^{\sigma Ah}, P^{\sigma Ah}, n^{\sigma Ah})]\!]_{\mathsf{IC}}\, X\, o \\
&= (\Sigma\, s : S^{\sigma Ah}(o))\, (\Pi\, p : P^{\sigma Ah}(o, s))\, X\, (n^{\sigma Ah}(o, s, p)) \\
&= (\Sigma\, a : A)\, (\Sigma\, s : S^{h(a)}(o))\, (\Pi\, p : P^{h(a)}(o, s))\, X\, (n^{h(a)}(o, s, p)) \\
&= (\Sigma\, a \colon A)\, [\![(S^{h(a)}, P^{h(a)}, n^{h(a)})]\!]_{\mathsf{IC}}\, X\, o \\
&= (\Sigma\, a : A)\, [\![\phi(f(a))]\!]_{\mathsf{IC}}\, X\, o \\
&\overset{(1)}{\cong} (\Sigma\, a \colon A)(\!|f(a)|\!)\, X\, o \\
&= (\!|\sigma\, A\, f|\!)\, X\, o
\end{aligned}$$

where (1) holds by the inductive hypothesis.

$$\begin{aligned}
[\![\phi(\delta\, A\, F)]\!]_{\mathsf{IC}}\, X\, o &= [\![(S^{\delta\, A\, H}, P^{\delta\, A\, H}, n^{\delta\, A\, H})]\!]_{\mathsf{IC}}\, X\, o \\
&= (\Sigma\, s : S^{\delta\, A\, H}(o))\, (\Pi\, p : P^{\delta\, A\, H}(o, s))\, X\, (n^{\delta\, A\, H}(o, s, p)) \\
&= (\Sigma\, g : A \to I)\, (\Sigma\, s : S^{H(g)}(o))\, (\Pi\, p : A + P^{H(g)}\, s) \\
&\qquad\qquad X[g, n^{H(g)}(o, s)](p) \\
&\overset{(1)}{\cong} (\Sigma\, g : A \to I)\, (\Sigma\, s : S^{H(g)}(o))\, ((\Pi\, a : A)\, X\, (g\, a)) \times \\
&\qquad\qquad (\Pi\, p : P^{H(g)}(o, s))\, X(n^{H(g)}(o, s, p))
\end{aligned}$$

$$\begin{aligned}
&\cong (\Sigma\, g : A \to I)\, ((\Pi\, a : A)\, X\, (g\, a)) \times \\
&\qquad (\Sigma\, s : S^{H(g)}(o))\, (\Pi\, p : P^{H(g)}(o, s))\, X\, (n^{H(g)}(o, s, p)) \\
&= (\Sigma\, g : A \to I)\, ((\Pi\, a : A)\, X\, (g(a))) \times \\
&\qquad\qquad [\![(S^{H(g)}, P^{H(g)}, n^{H(g)})]\!]_{\mathsf{IC}}\, X\, o \\
&= (\Sigma\, g : A \to I)((\Pi\, a : A)\, X\, (g(a))) \times [\![\phi(F(g))]\!]_{\mathsf{IC}}\, X\, o \\
&\overset{(2)}{\cong} (\Sigma\, g : A \to I)\, ((\Pi\, a : A)\, X\, (g(a))) \times (\!|F(g)|\!)\, X\, o \\
&= (\!|\delta\, A\, F|\!)\, X\, o
\end{aligned}$$

where (1) holds by the universal property of coproducts and (2) by the inductive hypothesis. □

4.3.3 Poly ≅ Small IR

In the previous sections we have seen that every Small IR-functor gives rise to an isomorphic dependent polynomial functor and vice versa. What can we say about natural transformations between these functors? In this section we show that extending the above isomorphism to an equivalence between the two categories $\mathsf{IR}_{\mathsf{sm}}(I, O)$ and $\mathsf{Poly}(I, O)$ is an immediate consequence of the categorical apparatus developed in Section 4.2.
We sum up the results of the previous sections in the following corollary.

Corollary 4.16. For every $\gamma\colon \mathsf{IR}_{\mathsf{sm}}(I, O)$ and, for every $(r, t, q)\colon \mathsf{Poly}(I, O)$ there exist functions ϕ and ψ such that:

(1) $[\![\psi(\phi(\gamma))]\!] \cong [\![\gamma]\!]$,

(2) $\mathcal{P}_{\phi(\psi(L))} \cong \mathcal{P}_L$

Proof. The results in the previous section allow us to establish an isomorphism between the classes of functors defined by $\mathsf{IR}_{\mathsf{sm}}(I, O)$ and $\mathsf{Poly}(I, O)$:

- in Section 4.3.1, we have defined a function $\psi : \mathsf{Poly}(I, O) \to \mathsf{IR}_{\mathsf{sm}}(I, O)$; such that $\mathcal{P}_{\_} \cong [\![\_]\!] \circ \psi$
- in Section 4.3.2 we show how to build ϕ: $\mathsf{IR}_{\mathsf{sm}}(I, O) \to \mathsf{Poly}(I, O)$, such that $[\![\_]\!] \cong \mathcal{P}_{\_} \circ \phi$. □

The equivalence of the two categories $\mathsf{IR}_{\mathsf{sm}}(I, O)$ and $\mathsf{Poly}(I, O)$ is an immediate consequence of the previous results combined with full and faithfulness of the respective interpretation functions:

Theorem 4.17. The two categories $\mathsf{IR}_{\mathsf{sm}}(I, O)$ and $\mathsf{Poly}(I, O)$ are equivalent.

Proof. Full and faithfulness of ϕ (or, equivalently of ψ) is immediate from full and faithfulness of the functorial semantics:

$$\begin{aligned}
\mathsf{IR}_{\mathsf{sm}}(I, O)(\gamma, \gamma') &\cong [\mathsf{Set}/I, \mathsf{Set}/O]([\![\gamma]\!], [\![\gamma']\!]) && \text{(theorem 4.11)} \\
&\cong [\mathsf{Set}/I, \mathsf{Set}/O](P_{\phi(\gamma)}, P_{\phi(\gamma')}) && \text{(lemma 4.15)} \\
&\cong \mathsf{Poly}(I, O)(\phi(\gamma), \phi(\gamma')) && \text{(corollary 2.48)}
\end{aligned}$$

Now, since a dependent polynomial, (r, t, q) is isomorphic to $\phi(\gamma)$ for some $\gamma : \mathsf{IR}_{\mathsf{sm}}(I, O)$ by (2) in Corollary 4.16, this is enough to conclude the stated equivalence (see e.g. Theorem 1, par. 4, ch. IV in [69]). □

4.3.4 Small indexed induction-recursion

In this section we show how the theory of small indexed inductive-recursive definitions, or Small IIR for short can be reduced to Small IR. The very same reduction applies also for the more general theory of IIR which can be reduced to the theory of IR as hinted by Dybjer and Setzer in [40] (end of Section 4.5). This simple result will automatically transfer the results of the previous

sections to Small IIR allowing to conclude a generalisation of the equivalence stated in theorem 4.17.
The *smallness* conditions for IR allows us to interpret Small IR-codes as functors in slice categories. The same observation applies here: a Small IIR-code $\gamma : \mathsf{IIR}_{\mathsf{sm}}(D, E)$, for $D : I \to \mathsf{Set}$ and $E : J \to \mathsf{Set}$ (cf. Definition 3.20) is interpreted as a functor

$$[\![\gamma]\!]_{\mathsf{IIR}} : (\Pi\, i : I)\, \mathsf{Set}/D(i) \longrightarrow (\Pi\, j : J)\, \mathsf{Set}/D(j)$$

Now, by exploiting the equivalence in Lemma 3.22 in this setting we get an equivalence between categories

$$\varphi : (\Pi\, i : I)\, \mathsf{Set}/D(i) \cong \mathsf{Set}/(\Sigma\, j : J)E(j) : \theta \tag{4.7}$$

Thus, a Small IIR-codes can be equivalently be interpreted as a functor

$$(\!|\gamma|\!)_{\mathsf{IIR}} \colon \mathsf{Set}/(\Sigma\, i : I)D(i) \to \mathsf{Set}/(\Sigma\, j : J)E(i) \tag{4.8}$$

by letting $(\!|\gamma|\!)_{\mathsf{IIR}} = \varphi \circ [\![\gamma]\!]_{\mathsf{IIR}} \circ \theta$, where φ and θ are the two components of the equivalence 4.7.
Now, observe that we can translate IIR-codes into plain IR-codes as follows:

Definition 4.18. For a code $\gamma : \mathsf{IIR}_{\mathsf{sm}}(D, E)$ define the corresponding code $\ulcorner \gamma \urcorner : \mathsf{IR}_{\mathsf{sm}}((\Sigma\, i : I)D(i), (\Sigma\, j : J)E(i))$ by induction on the structure of γ as follows:

$$\begin{aligned}
\ulcorner \iota(j, e) \urcorner &= \iota(j, e) \\
\ulcorner \sigma\, A\, f \urcorner &= \sigma\, A(a \mapsto \ulcorner f(a) \urcorner) \\
\ulcorner \delta\, B\, g\, F \urcorner &= \delta\, B(f \mapsto \sigma(g \equiv_{B \to I} \pi_0 \circ f)(p \mapsto \ulcorner F(\pi_1 \circ f) \urcorner))
\end{aligned}$$

■

The translation in Definition 4.18 enable us to state the reduction of Small IIR to IR:

Lemma 4.19. For every code $\gamma : \mathsf{IIR}_{\mathsf{sm}}(D, E)$ it holds $(\!|\gamma|\!)_{\mathsf{IIR}} \cong [\![\ulcorner \gamma \urcorner]\!]$.

Proof. The interesting case is $(\!|\delta\ B\ g\ F|\!)_{\mathsf{IIR}} \cong [\![\ulcorner \delta\, B\, g\, F \urcorner]\!]$. We use φ as defined in equation (4.7) to compare the functors:

$$\varphi \circ [\![\delta\ B\ g\ F]\!]_{\mathsf{IIR}},\ [\![\ulcorner \delta\, B\, g\, F \urcorner]\!] \circ \varphi : [(\Pi\, i : I)\mathsf{Set}/D(i), \mathsf{Set}/(\Sigma\, I : I)D(i)]$$

Given $(X,T) : (\Pi\, i : I)\mathsf{Set}/D(i)$ calculate as follows:

$$
\begin{aligned}
&[\![\ulcorner \delta\, B\, g\, F \urcorner]\!](\varphi(X,T)) \\
&= [\![\delta\, B(f \mapsto \sigma(g \equiv_{B\to I} \pi_0 \circ f)(p \mapsto \ulcorner F(\pi_1 \circ f)\urcorner))]\!](\varphi(X,T)) \\
&= (\Sigma\, f : B \to (\Sigma\, i : I)X(i))(\Sigma\, p : g \equiv_{B\to I} \pi_0 \circ (\Sigma\, i : I)T(i,-) \circ f) \\
&\qquad [\![\ulcorner F(\pi_1 \circ (\Sigma\, i : I)T(i,-) \circ f)\urcorner]\!]((\Sigma\, i : I)X(i), (\Sigma\, i : I)T(i,-)) \\
&\overset{(1)}{\cong} (\Sigma\, f' : B \to I)(\Sigma\, h : (\Pi\, b : B)X(f'(b)))(\Sigma\, p : g \equiv_{B\to I} f') \\
&\qquad [\![\ulcorner F(\pi_1 \circ (\Sigma\, i : I)T(i,-) \circ \langle f', h\rangle))\urcorner]\!]((\Sigma\, i : I)X(i), (\Sigma\, i : I)T(i,-)) \\
&\overset{(2)}{\cong} (\Sigma\, h : (\Pi\, b : B)X(g(b)))[\![\ulcorner F(T(-,-) \circ \langle g, h\rangle)\urcorner]\!]((\Sigma\, i : I)X(i), (\Sigma\, i : I)T(i,-)) \\
&\overset{(3)}{\cong} (\Sigma\, h : (\Pi\, b : B)X(g(b)))\varphi([\![F(\,T(g-,-) \circ \langle \mathsf{id}, h\rangle)]\!](X,T)) \\
&\overset{(4)}{\cong} \varphi((\Sigma\, h : (\Pi\, b : B)X(g(b)))[\![F(\,T(g-,-) \circ \langle \mathsf{id}, h\rangle)]\!](X,T)) = \varphi \circ [\![\delta\, B\, g\, F]\!]_{\mathsf{IIR}}
\end{aligned}
$$

where (1) holds by the axiom of choice and by $\pi_0 \circ (\Sigma\, i : I)T(i,-) \equiv_{((\Sigma\, i : I)X(i))\to I} \pi_0$, (2) by using $p : g \equiv_{B\to I} f'$ and by currying $\pi_1 \circ (\Sigma\, i : I)T(i,-)$, (3) by the inductive hypothesis and (4) since φ is part of an equivalence, hence a left adjoint and therefore preserves colimits. □

Given this embedding, we can endow $\mathsf{Small\ IIR}$ with the categorical machinery developed for $\mathsf{Small\ IR}$ in Section 4.2. We therefore can straightforwardly define a category of $\mathsf{IIR}_{\mathsf{sm}}(D,E)$-codes and their morphisms. Theorem 4.17 immediately give us the following corollary.

Corollary 4.20. The category $\mathsf{IIR}_{\mathsf{sm}}(D,E)$ and the category $\mathsf{Poly}((\Sigma\, i : I)D(i), (\Sigma\, j : J)E(j))$ are equivalent.

A remarkable example of the use of $\mathsf{Small\ IIR}$ is the Bove-Capretta method to model nested general recursion in type theory [21, 22]. The idea is to define a recursive function and its domain simultaneously. Thus, for a partial function $f : (\Pi\, i : I)D(i)$ we can build a code $\gamma_f : \mathsf{IIR}(D,D)$ where the domain of f is represented by $\mu[\![\gamma_f]\!]_0$ and the function f by $\mu[\![\gamma_f]\!]_1$. For those $i : I$ such that $\mu[\![\gamma_f]\!]_0$ is inhabited we can then compute the values of f by using $\mu[\![\gamma_f]\!]_1$.

Remark 4.21. Dybjer and Setzer [40] have used indexed induction-recursion to give a finite axiomatization of indexed inductive definitions. In particular,

they identifies the large type of indexed inductive definitions IID with the type $\mathsf{IIR}(\lambda\_.\mathsf{N}_1, \lambda\_.\mathsf{N}_1)$, i.e those for which $D(i) = \mathsf{N}_1$ for every $i : I$. Their axiomatization which uses IIR can be reconciled with ours, which uses $\mathsf{Small\ IR}$, by noting that codes of type $\mathsf{IIR}(\lambda\_.\mathsf{N}_1, \lambda\_.\mathsf{N}_1)$ correspond to $\mathsf{IR}_{\mathsf{sm}}(I, I)$-codes, where the set I is the index for the indexed inductive-recursive definitions in $\mathsf{IIR}(\lambda\_.\mathsf{N}_1, \lambda\_.\mathsf{N}_1)$. Indeed, using the semantics of $\mathsf{Small\ IR}$ given in Lemma 4.3, it is immediate to check that the two sets of codes represent the same classes of functors.

4.4 Internal IR

In the previous sections we saw that the theory of indexed inductive types sits within the theory of inductive-recursive type as a proper sub-theory. Notably, universes are inductive-recursive types which lie beyond indexed inductive types since they are defined as initial algebras of codes $\mathsf{IR}(\mathsf{Set}, \mathsf{Set})$ and thus not by a $\mathsf{Small\ IR}$-code.

In this section we sketch how it is possible to build *internal* universes by using $\mathsf{Small\ IR}$ if we assume that a universe built by IR is already at our disposal. This will also help to clarify why the reduction of $\mathsf{Small\ IR}$ to the theory of dependent polynomials presented in the previous sections does not go through for general IR-codes.

Let $(\mathsf{U}, \mathsf{T}) : \mathsf{Fam}|\mathsf{Set}|$ be a universe built using an $\mathsf{IR}(\mathsf{Set}, \mathsf{Set})$-code. To keep things simple, let say that U has a code $\widehat{\mathsf{N}} : \mathsf{U}$ for the set of natural numbers, i.e. $\mathsf{T}(\widehat{\mathsf{N}}) = \mathsf{N}$, and it is closed under Σ-types, i.e. for every $(u, g) : (\Sigma\, u : \mathsf{U})$ $(\mathsf{T}(u) \to \mathsf{U})$ there exists a code $\widehat{\Sigma}(u, g) : \mathsf{U}$ such that $\mathsf{T}(\widehat{\Sigma}(u, g)) = (\Sigma\, x{:}\, \mathsf{T}(u))$ $\mathsf{T}(g(x))$. By using $\mathsf{Small\ IR}$, we now build an internal universe which contains a code for $\widehat{\mathsf{N}}$ and which is closed under *internal* Σ-types, i.e. $\widehat{\Sigma}$-types in U. The code for this internal universe is a $\mathsf{Small\ IR}$-code, which uses the universe (U, T) for its definition:

$$\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}} =_{\mathsf{def}} \iota\, \widehat{\mathsf{N}} + \delta\, 1(u \mapsto \delta(\mathsf{T}(u))(g \mapsto \iota(\widehat{\Sigma}(u, g)))) : \mathsf{IR}_{\mathsf{sm}}(\mathsf{U}, \mathsf{U})$$

The initial algebra $(\mu_{[\![\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}]\!]}, \mathsf{intro}_{[\![\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}]\!]})$ for the corresponding functor $[\![\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}]\!]$:

$\mathsf{Set}/\mathsf{U} \to \mathsf{Set}/\mathsf{U}$ is the least solution of the following equations:

$$\begin{aligned} X &\cong \mathsf{N}_1 + (\Sigma\, x : X)(\mathsf{T}(f(x)) \to X) \\ f\,(\mathsf{inl}(0_1)) &\cong \widehat{\mathsf{N}} \\ f\,(\mathsf{inr}\,(x, g)) &\cong \widehat{\Sigma}(f(x)(f \circ g)). \end{aligned}$$

where (X, f) is an object of Set/U. Since $\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}$ is a $\mathsf{Small\ IR}$-code we can apply the equivalence between $\mathsf{IR}_{\mathsf{sm}}(\mathsf{U}, \mathsf{U})$ and $\mathsf{Poly}(\mathsf{U}, \mathsf{U})$ and build the corresponding dependent polynomial representing the universe $(\mu_{[\![\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}]\!]}, \mathsf{intro}_{[\![\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}]\!]})$. How is this possible? The idea is that we have already exploited the power of IR when building the universe (U, T): in particular this bought us a small set of codes $\mathsf{U} : \mathsf{Set}$ which we can then use as the target of a recursive function $f : X \to \mathsf{U}$ built with $\mathsf{Small\ IR}$. By computing $\phi(\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}})$ as in Definition 4.14, we obtain an indexed container whose set of shapes $S^{\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}}$ is the set $(\Sigma\, u : \mathsf{U})(\mathsf{T}(u) \to \mathsf{U})$. Since both U and $\mathsf{T}(u)$ are sets for every $u : \mathsf{U}$, then also $S^{\gamma_{\widehat{\Sigma},\widehat{\mathsf{N}}}}$ is. Notice that this would not be the case if we were starting the construction with the large universe $(\mathsf{Set}, \mathsf{El})$ because we would end up with a large quantification on Set.

4.5 Conclusion

In this section we have precisely characterized the sub-theory of IR corresponding to indexed inductive types by the equivalence between the categories $\mathsf{IR}_{\mathsf{sm}}(I, O)$ and $\mathsf{Poly}(I, O)$ (Theorem 4.17). By exploiting this equivalence we can transport the rich closure properties of dependent polynomials and indexed containers. In particular, we can transport composition and give a partial answer to the open question if IR-functors are closed under composition or not.

In recent work Capretta [23] has investigated small induction-recursive types from a coalgebraic perspective, dubbing them *wander types*. The results in this chapter ensure that final coalgebra for small inductive-recursive definition always exists since they are nothing but final coalgebra for dependent polynomial functors.

Chapter 5

Positive induction-recursion

Abstract In this chapter we introduce a generalization of IR, which we call *positive induction-recursion* which allows for the definition of data types as initial algebras of certain functors $\mathsf{Fam}\,(\mathbb{C}) \to \mathsf{Fam}\,(\mathbb{C})$, thus lifting the restriction in the theory of IR where $\mathbb{C}$ has to be discrete. This leads us to introduce simultaneously codes for such data types, and morphisms between them. We then give a functorial semantics and show application and concrete examples of the resulting theory. We justify the existence of these data types by adapting the set-theoretic model for IR to our setting.

5.1 Introduction

In Chapter 4 we studied a sub-theory of IR which precisely captures indexed inductive types. In this chapter we come back to the large class of inductive-recursive definitions (Definition 3.7), and again we use the categorical semantics to explore a generalization of both the syntax and the semantics of IR.

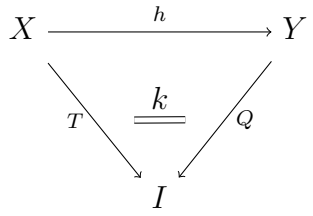
In Chapter 3 we saw that the families construction, which is pervasive in the fibrational semantics of dependent type theories [64], plays also a central role in the semantics of IR. As explained in Section 3.2.2, IR-functors are naturally defined on the category $\mathsf{Fam}|D|$ of families of elements of a (possibly large) type D. For example universes of sets closed under certain operations, such as dependent sums and products, arise as initial algebras

of endofunctors on $\mathsf{Fam}|\mathsf{Set}|$. Following the family approach we could try to extend this interpretation to obtain functors on the category $\mathsf{Fam}\,(\mathbb{C})$ of families of an arbitrary category $\mathbb{C}$. But, there are at least two complications which prevent us in doing this: on the one hand we saw in Section 3.2.2 page 60 that the definition of the action of IR-functors on morphisms already relies on the discreteness of the type D; on the other hand IR-functors such as those building universes closed under dependent products, contain a mixture of covariance and contravariance intrinsic in the Π operator which forces us to restrict to those morphisms of $\mathsf{Fam}\,(\mathbb{C})$ whose second component is an identity. In this chapter, we explore a generalization of IR by investigating the necessary changes of IR needed to provide a more general semantics of functors between arbitrary families. Thus, we consider a new variation of inductive-recursive definitions which we call *positive inductive-recursive definitions*, or IR^+ for short, which enables us to define data types as initial algebras of certain functors $\mathsf{Fam}\,(\mathbb{C}) \to \mathsf{Fam}\,(\mathbb{C})$. We organize this chapter as follows: (i) in Section 5.2 we introduce syntax and semantics of IR^+; (ii) in Section 5.3 we study the elimination principle for IR^+ by showing some of its applications; (iii) in Section 5.4 we use positive inductive-recursive definitions to shed new light on nested data types; (iv) in Section 5.5 we give a detailed comparison with the theory of IR. Finally, (iv) in Section 5.6 we adapt Dybjer and Setzer's model construction to our setting.

5.2 Positive induction-recursion

Given types I and O, the syntax of IR introduced in Section 3.2.1 allows us to build codes which are interpreted as functors $\mathsf{Fam}|I| \to \mathsf{Fam}|O|$. Thus, when interpreting IR-codes we perform the Fam construction on *discrete* categories only. But what happens, if I and O have a richer structure, as for example the type Set has, and we try to keep track of this structure? Why can't we consider endofunctor defined on $\mathsf{Fam}\,(\mathsf{Set})$ and not only on its subcategory $\mathsf{Fam}|\mathsf{Set}|$? As already seen in Section 3.2.2 to prove functoriality of the interpretation of a δ-code, this restriction is unavoidable, and there is no immediate generalization for our functors: indeed the action of a δ-code on a morphism (h, k): $(X, T) \to (Y, Q)$ in $\mathsf{Fam}|I|$ crucially depends on the second

component k being an identity as displayed in the following diagram:



But what happens if the diagram above does not commute on the nose and I is not simply a discrete category? Can we consider a more general natural transformation $k : T \Rightarrow Q \circ h$ which is not merely the identity?
We now investigate what are the necessary changes to the syntax of IR needed to use morphisms (h, k) in $\mathsf{Fam}\, I$, for I not necessarily a discrete category. The insight which guides us when introducing the syntax of IR^+ is to deploy a proper *functor* F, as opposed to a function, in the introduction rule for a δ-code. This enables to relax the restriction mentioned above. Recall that, when giving the action of $\delta\, B\, F : \mathsf{IR}(I, O)$ on morphisms we rely crucially on the equality $T \equiv_{X \to I} Q \circ h$ to use the inductive hypothesis on the code $F(T \circ g)$ where $g : B \to X$ is a given function. If $F : (B \to I) \to_{\mathsf{Cat}} \mathsf{IR}^+(\mathbb{C})$ is now a functor, and not just a function, we do not have to rely on the equality $T \circ g \equiv_{B \to I} Q \circ h \circ g$ between objects in $\mathbb{C}^B$ as before, but we can use the second component of a morphism (h, k) in $\mathsf{Fam}\,(\mathbb{C})$ to get a map $T \circ g \to Q \circ h \circ g$ in $\mathbb{C}^B$; by applying F to this map we get a morphism between codes $F(T \circ g) \to F(Q \circ h \circ g)$ and we can then use the inductive hypothesis on the code $F(Q \circ h \circ g)$.
For F to be a functor, $\mathsf{IR}^+(\mathbb{C})$ has to be a category. Thus, we are forced to introduce codes and morphisms between them simultaneously: to this purpose we use induction-induction [82] which we briefly recall below.

5.2.1 Inductive-inductive definitions

The theory of inductive-inductive type has been recently introduced by Nordvall Forsberg and Setzer [43]; a comprehensive study of this principle can be found in Nordvall Forsberg's thesis [81].
Inductive-inductive definitions are close relatives of inductive-recursive definitions, and share with them the mutual definition of a set X and of a family

$T : X \to \mathsf{Set}$. The distinctive feature of the inductive-inductive definition of a family (X, T), is that the family T is generated inductively and not recursively. That is, we build both X and T inductively using constructors $\mathsf{intro}_F : F(X, T) \to X$ and $\mathsf{intro}_G : (\Pi\, x : G(X, T, \mathsf{intro}_F))\, T(h(x))$ where F and G are strictly positive functors, and $h : G(X, T, \mathsf{intro}_F) \to X$ is a map which selects the index $h(\mathsf{intro}_G(x)) : X$ for the element $\mathsf{intro}_G(x)$.
Inductive-inductive definitions admit a finite axiomatization similar to the one given in Section 3.2 for IR-codes, but slightly complicated by the fact that one has to introduce two types of codes, one for the index set X, and one for the family $T : X \to \mathsf{Set}$.
The inductive definition of T allows for a more general elimination principle compared to that for inductive-recursive types. We saw in Section 3.2.5 that the elimination principle for an inductive-recursive type (X, T) is given for predicates $x : X \vdash P(x) : \mathsf{type}$. However, in an inductive-inductive type (X, T) both X and T are given by constructors, and therefore a predicate on (X, T) can now depend on both of them, i.e. the induction principle is given for pairs of predicates (P, Q) where $x : X \vdash P(x) : \mathsf{type}$ and $x : X, y : T(x), z : P(x) \vdash Q(x, y, z) : \mathsf{type}$[1]. Notice that Q can also depend on P. Nordvall Forsberg [81] (Section 3.2.5) suggests to summarize the situation by the slogan "the elimination principle for an inductive-inductive type is recursive-recursive". We will exploit this elimination principle when defining the semantics for IR^+.

5.2.2 A finite axiomatization of IR^+

We give an axiomatic presentation of IR^+ analogous to the one given in Definition 3.7 for the syntax of IR; however, we now have mutual introduction rules to build both the type of $\mathsf{IR}^+(\mathbb{C})$ codes and the type of $\mathsf{IR}^+(\mathbb{C})$ morphisms, for $\mathbb{C}$ a given category.

Definition 5.1. Given a category $\mathbb{C}$ we simultaneously define the type of positive inductive-recursive codes on $\mathbb{C}$, $\mathsf{IR}^+(\mathbb{C}) : \mathsf{type}$ and the type of morphisms between these codes $\mathsf{IR}^+(\mathbb{C})(\_, \_) : \mathsf{IR}^+(\mathbb{C}) \to \mathsf{IR}^+(\mathbb{C}) \to \mathsf{type}$ by the

[1] In general we do not need large elimination for induction-induction, but it will turn useful for our purposes.

following introduction rules:

- $\mathsf{IR}^+(\mathbb{C})$ codes:

$$\frac{c : \mathbb{C}}{\iota\, c : \mathsf{IR}^+(\mathbb{C})}$$

$$\frac{A : \mathsf{Set} \qquad f : A \to \mathsf{IR}^+(\mathbb{C})}{\sigma\, A\, f : \mathsf{IR}^+(\mathbb{C})}$$

$$\frac{A : \mathsf{Set} \qquad F : (B \to \mathbb{C}) \to_{\mathsf{Cat}} \mathsf{IR}^+(\mathbb{C})}{\delta\, B\, F : \mathsf{IR}^+(\mathbb{C})}$$

- $\mathsf{IR}^+(\mathbb{C})$ morphisms:

  identity morphisms

$$\frac{\gamma : \mathsf{IR}^+(\mathbb{C})}{\mathsf{id}_\gamma : \mathsf{IR}^+(\mathbb{C})(\gamma, \gamma)}$$

  morphisms from ιc:

$$\frac{f : \mathbb{C}(c, c')}{\Gamma_{\iota,\iota}(f) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \iota\, c')}$$

$$\frac{a : A \qquad r : \mathsf{IR}^+(\mathbb{C})(\iota\, c, f(a))}{\Gamma_{\iota,\sigma}(a, r) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \sigma\, A\, f)}$$

$$\frac{g : B \to \mathsf{N}_0 \qquad r : \mathsf{IR}^+(\mathbb{C})(\iota\, c, F(! \circ g))}{\Gamma_{\iota,\delta}(g, r) : \mathsf{IR}^+(\mathbb{C})(\iota\, c, \delta\, B\, F)}$$

  morphisms from $\sigma\, A\, f$:

$$\frac{\gamma : \mathsf{IR}^+(\mathbb{C}) \qquad r : (\Pi\, a : A)\mathsf{IR}^+(\mathbb{C})(f(a), \gamma)}{\Gamma_{\sigma,\gamma}(r) : \mathsf{IR}^+(\mathbb{C})(\sigma\, A\, f, \gamma)}$$

  morphisms from $\delta\, B\, F$

$$\frac{\gamma : \mathsf{IR}^+(\mathbb{C}) \quad \rho : \mathsf{Nat}(F, \mathsf{c}_\gamma)}{\Gamma_{\delta,\gamma}(\rho) : \mathsf{IR}^+(\mathbb{C})(\delta\, B\, F, \gamma)}$$

$$\frac{a : A \qquad \rho : \mathsf{Nat}(F, \mathsf{c}_{f(a)})}{\Gamma_{\delta,\sigma}(a,\rho) : \mathsf{IR}^+(\mathbb{C})(\delta\, B\, F, \sigma\, A\, f)}$$

$$\frac{f : B \to A \qquad \rho : \mathsf{Nat}(F, G(\_ \circ f))}{\Gamma_{\delta,\delta}(f,\rho) : \mathsf{IR}^+(\mathbb{C})(\delta\, A\, F, \delta\, B\, G)}$$

In the last three clauses we have indicated with $\mathsf{c}_\gamma \colon \mathbb{C}^A \to \mathsf{IR}^+(\mathbb{C})$ the constant functor with value γ. ■

We now need to make sure that 5.1 really defines a category, i.e. that composition of IR^+ morphisms can be defined, and that it is associative and has identities.

Lemma 5.2. Given a category $\mathbb{C}$, the set of codes and morphisms as given in Definition 5.1 makes $\mathsf{IR}^+(\mathbb{C})$ into a category.

Proof. We show how to define a composite operation $\_ \odot \_ \colon \mathsf{IR}^+(\mathbb{C})(\gamma', \gamma'') \times \mathsf{IR}^+(\mathbb{C})(\gamma, \gamma') \to \mathsf{IR}^+(\mathbb{C})(\gamma, \gamma'')$.

$$\begin{aligned}
\mathsf{id}_\gamma \odot \Gamma_{\gamma',\gamma} &= \Gamma_{\gamma',\gamma} \\
\Gamma_{\gamma,\gamma'} \odot \mathsf{id}_\gamma &= \Gamma_{\gamma,\gamma'} \\
\Gamma_{\iota,\iota}(g) \odot \Gamma_{\iota,\iota}(f) &= \Gamma_{\iota,\iota}(g \circ f) \\
\Gamma_{\gamma,\gamma'} \odot \Gamma_{\sigma,\gamma}(r) &= \Gamma_{\sigma,\gamma'}(\lambda\, a.(\Gamma_{\gamma,\gamma'} \odot f(a)) \\
\Gamma_{\gamma,\gamma'} \odot \Gamma_{\delta,\gamma}(\rho) &= \Gamma_{\delta,\gamma'}(\mathsf{c}_{\Gamma_{\gamma,\gamma'}} \cdot \rho) && \text{(a)} \\
\Gamma_{\iota,\iota}(g) \odot \Gamma_{\iota,\sigma}(a,r) &= \Gamma_{\iota,\sigma}(a, r \odot \Gamma_{\iota,\iota}(g)) \\
\Gamma_{\iota,\iota}(f) \odot \Gamma_{\iota,\delta}(g,r) &= \Gamma_{\iota,\delta}(g, r \odot \Gamma_{\iota,\iota}(f)) \\
\Gamma_{\sigma,\gamma}(s) \odot \Gamma_{\iota,\sigma}(a,r) &= r \odot s(a) \\
\Gamma_{\sigma,\gamma}(r) \odot \Gamma_{\delta,\sigma}(a,\rho) &= \Gamma_{\delta,\gamma}(\mathsf{c}_{r(a)} \cdot \rho) && \text{(b)} \\
\Gamma_{\delta,\gamma}(\rho) \odot \Gamma_{\iota,\delta}(g,r) &= \rho_{(!\circ g)} \odot r \\
\Gamma_{\delta,\gamma}(\tau) \odot \Gamma_{\delta,\delta}(f,\rho) &= \Gamma_{\delta,\gamma}(\tau(_ \circ f) \cdot \rho) && \text{(c)} \\
\Gamma_{\delta,\sigma}(a,\rho) \odot \Gamma_{\iota,\delta}(g,r) &= \Gamma_{\iota,\sigma}(a, \rho_{!\circ g} \odot r)(c) \\
\Gamma_{\delta,\sigma}(a,\tau) \odot \Gamma_{\delta,\delta}(f,\rho) &= \Gamma_{\delta,\sigma}(a, \tau(_ \circ f) \cdot \rho) && \text{(d)} \\
\Gamma_{\delta,\delta}(g,\rho) \odot \Gamma_{\iota,\delta}(f,r) &= \Gamma_{\iota,\delta}(f \circ g, \rho_{!\circ f} \odot r) \\
\Gamma_{\delta,\delta}(g,\tau) \odot \Gamma_{\delta,\delta}(f,\rho) &= \Gamma_{\delta,\delta}(f \circ g, \tau(_ \circ f) \cdot \rho) && \text{(e)}
\end{aligned}$$

In the clauses above we used $\_ \cdot \_$ for composition of natural transformation. Also, we used $\mathsf{c}_{\Gamma_{\gamma,\gamma'}} : \mathsf{c}_\gamma \dot{\to} \mathsf{c}_{\gamma'}$ in clause (a) for the natural transformation with constant components $\Gamma_{\gamma,\gamma'}$ and similarly for clause (b) while $\tau(\_ \circ f)$ in clause (c), (d) and (e) indicates the whiskering of $\tau : G \dot{\to} \mathsf{c}_\gamma$ and $(\_ \circ f) : \mathbb{C}^A \to \mathbb{C}^B$. Associativity is proved by induction and we omit the details. □

The careful reader might have noticed a possible redundancy in Definition 5.1: when defining morphisms out of δ codes we gave the introduction rule $\Gamma_{\delta,\gamma}(g)$ for morphisms with codomain an arbitrary code γ, but we also included $\Gamma_{\delta,\sigma}(b,\rho)$ and $\Gamma_{\delta,\delta}(g,\rho)$, representing morphisms from a δ code into a σ code and another δ code respectively. The reason is the following: it is necessary to have both these forms of morphisms out of a δ code when recursively defining composition of IR^+. Since more morphisms makes it easier to define codes, we are also interested in allowing as many morphisms as possible. The semantics we are going to give explains how elements of $\mathsf{IR}^+(\mathbb{C})$ decode as functors on $\mathsf{Fam}\,(\mathbb{C})$, while morphisms between such codes decode to natural transformations between the corresponding functors. The definition of morphisms above is rather different from the definition of morphisms between $\mathsf{Small}\ \mathsf{IR}$-codes given in Definition 4.7. Indeed, the latter crucially depends on the characterization of the interpretation of a δ-code as a sum of left Kan extensions (see Theorem 4.5): this characterization does not trivially extend to the more general setting of $\mathsf{Fam}\,(\mathbb{C})$, and, as a consequence, we loose the analogous result of full and faithfulness for the interpretation functor (Theorem 4.10). Indeed already the redundancy in the definition of morphisms out of a δ code mentioned above shows that it is not possible to expect the interpretation functor to be faithful.

The semantics of IR^+ closely follows the semantics of IR given in Section 3.2.2; as before we make essential use of coproducts in $\mathsf{Fam}\,(\mathbb{C})$ which are intrinsically inherited by the Fam construction (see Remark 3.10). A functor which is isomorphic to a functor induced by an IR^+-code will be called an IR^+-functor.

The crucial feature which separates the semantics of IR^+ from the semantics of IR is the following: when explaining the semantics of IR we can first interpret IR-codes as functors and only later we define morphisms between

codes which are interpreted as natural transformations between the corresponding functors. In IR^+ the type of of codes and the type of morphisms between codes are simultaneously defined in an inductive-inductive way, and therefore they are also decoded simultaneously as functors and natural transformations respectively. This is exactly what the elimination principle for an inductive-inductive definition gives us.

Theorem 5.3 (IR^+ functors)**.** Let $\mathbb{C}$ be a category.

(i) Every code $\gamma : \mathsf{IR}^+(\mathbb{C})$ induces a functor $[\![\gamma]\!] : \mathsf{Fam}\,(\mathbb{C}) \to \mathsf{Fam}\,(\mathbb{C})$.

(ii) Every morphism $r : \mathsf{IR}^+(\mathbb{C})(\gamma, \gamma')$ for codes $\gamma, \gamma' : \mathsf{IR}^+(\mathbb{C})$ gives rise to a natural transformation $[\![r]\!] : [\![\gamma]\!] \dot{\longrightarrow} [\![\gamma']\!]$.

Proof. While the action on objects is the same for both IR^+ and IR functors, the action on morphisms is different when interpreting a code of type $\delta\, B\, F$: in the semantics of IR^+ we exploit the fact that $F : (A \to \mathbb{C}) \to \mathsf{IR}^+(\mathbb{C})$ is now a functor by using its action on morphism (which we, for the sake of clarity, indicate with $F_\to$). We give the action of IR^+ functors on morphisms only, and refer to the semantics given in Section 3.2.2 for the action on objects of $\mathsf{Fam}\,(\mathbb{C})$.
The action on morphisms is given as follows. Let $(h, k) : (X, T) \to (Y, Q)$ be a morphism in $\mathsf{Fam}\,(\mathbb{C})$, we define $[\![\gamma]\!](h, k)$ by recursion on γ:

$$
\begin{aligned}
[\![\iota\, c]\!](h,k) &= (\mathsf{id}_{\mathsf{N}_1}, \mathsf{id}) \\
[\![\sigma\, A\, f]\!](h,k) &= (\Sigma\, a : A)([\![f(a)]\!](h,k) = [\mathsf{in}_a \circ [\![f(a)]\!](h,k)]_{a:A} \\
[\![\delta\, B\, F]\!](h,k) &= (\Sigma(\lambda\, g.\, h \circ g), \lambda\, g.[\![F(Q \circ h \circ g)]\!](h,k) \circ [\![F_\to(g^*(k))]\!]_{(X,T)}) \\
&= [\mathsf{in}_{h \circ g} \circ [\![F(Q \circ h \circ g)]\!](h,k) \circ [\![F_\to(g^*(k))]\!]_{(X,T)}]_{g:B \to X}
\end{aligned}
$$

In the last clause we use the notation for a generalized sum of morphisms (see Remark 3.10(3)) and we indicated with $g^*(k) : T \circ g \dot{\longrightarrow} Q \circ h \circ g$ the natural transformation with component $g^*(k)_b = k_{g(b)} : T\,(g(b)) \to Q\,(h\,(g(b)))$; notice that, from a fibrational perspective (cf. Section 6.2), such a natural transformation is nothing but a vertical morphism above B obtained by reindexing $(\mathsf{id}_X, k) : (X, T) \to (X, Q \circ h)$ along g in the families fibration $\pi : \mathsf{Fam}\,(\mathbb{C}) \to \mathsf{Set}$. To help the reader's intuition we draw the commuting diagram representing the inductive hypothesis of the last clause:

$$\begin{array}{ccc} [\![F(T \circ g)]\!](U,T) & \xrightarrow{[\![F_{\to}(g^* k)]\!]_{(X,T)}} & [\![F(Q \circ h \circ g)']\!](U,T) \\ \Big\downarrow{\scriptstyle [\![F(T \circ g)]\!](h,k)} & & \Big\downarrow{\scriptstyle [\![F(Q \circ h \circ g)]\!](h,k)} \\ [\![F(T \circ g)]\!](U',T') & \xrightarrow{[\![F_{\to}(g^* k)]\!]_{(Y,Q)}} & [\![F(Q \circ h \circ g)']\!](U',T') \end{array}$$

The definition of $[\![\delta\, B\, F]\!](h,k)$ above is given by the generalized sum of the family of morphisms obtained by composing two sides of the above diagram over the morphism $h \circ \_ : (B \to X) \to (B \to Y)$ mapping the index set of the source to the index set of the target.
We now explain how IR^+ morphisms are interpreted as natural transformations between IR^+ functors by specifying their component at $(X,T) : \mathsf{Fam}\,(\mathbb{C})$.

$$\begin{aligned} [\![\mathsf{id}_\gamma]\!]_{(X,T)} &= (\mathsf{id}_{[\![\gamma]\!]})_{(X,T)} \\ [\![\Gamma_{\iota,\iota}(f)]\!]_{(X,T)} &= (\mathsf{id}_{\mathsf{N}_1}, f) : (\mathsf{N}_1, \lambda\_.\, c) \to (\mathsf{N}_1, \lambda\_.\, c') \\ [\![\Gamma_{\iota,\sigma}(a,r)]\!]_{(X,T)} &= \mathsf{in}_a \circ [\![r]\!]_{(X,T)} \\ [\![\Gamma_{\iota,\delta}(g,r)]\!]_{(X,T)} &= \mathsf{in}_{!_X \circ g} \circ [\![r]\!]_{(X,T)} \\ [\![\Gamma_{\sigma,\gamma}(r)]\!]_{(X,T)} &= [[\![r(a)]\!]_{(X,T)}]_{a:A} \\ [\![\Gamma_{\delta,\gamma}(\rho)]\!]_{(X,T)} &= [[\![\rho_{T \circ g}]\!]_{(X,T)}]_{g:B \to X} \\ [\![\Gamma_{\delta,\sigma}(a,\rho)]\!]_{(X,T)} &= \Sigma(\mathsf{c}_a, [\![\rho_{T \circ g}]\!]_{(X,T)}) \\ &= \mathsf{in}_a \circ [[\![\rho_{T \circ g}]\!]_{(X,T)}]_{g:B \to X} \\ [\![\Gamma_{\delta,\delta}(f,\rho)]\!]_{(X,T)} &= \Sigma\,(\lambda\, g.\, g \circ f, \lambda\, g.\, [\![\rho_{T \circ g}]\!]_{(X,T)}) \\ &= [\mathsf{in}_{g \circ f} \circ [\![\rho_{T \circ g}]\!]_{(X,T)}]_{g:A \to X} \end{aligned}$$

where we have used $\mathsf{c}_a : (B \to X) \to A$ in the seventh equation for the constant function value $a : A$ Naturality of these transformations can be proved by a routine diagram chasing. □

Example 5.4 (A universe closed under dependent sums in $\mathsf{Fam}\,(\mathsf{Set}^{\mathsf{op}})$)**.** In Section 3.2.7 we defined in (3.3) an ordinary IR-code for a universe closed under sigma types. We can extend this code to an IR^+ code

$$\gamma_{\mathsf{N},\Sigma} = \iota\, \mathsf{N} +_{\mathsf{IR}} \delta\, \mathsf{N}_1\, (X \mapsto \delta\, X(\mathsf{0}_1)(Y \mapsto \iota((\Sigma\, x : X(\mathsf{0}_1))\, Y(x)))) : \mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$$

where now $G =_{\mathsf{def}} Y \mapsto \iota\,((\Sigma\, X(0_1))\, Y)$ and $F =_{\mathsf{def}} X \mapsto \delta\, X(0_1)\, G$ needs to be functors. From a morphism $f : Y \to Y'$ in $[X(0_1), \mathsf{Set}^{\mathsf{op}}]$, represented as an $X(0_1)$-indexed collection of morphisms $f_x : Y(x) \to Y'(x)$ in $\mathsf{Set}^{\mathsf{op}}$, we get a morphism $(\Sigma\, x : X(0_1)) f_x : (\Sigma\, X(0_1))\, Y \to (\Sigma\, X(0_1))\, Y'$ in $\mathsf{Set}^{\mathsf{op}}$. Therefore we can define

$$G(f) : \iota((\Sigma\, X(0_1))\, Y) \to \iota((\Sigma\, X(0_1))\, Y')$$

by $G(f) = \Gamma_{\iota,\iota}((\Sigma\, x : X(0_1)) f_x)$.
We also need F to be a functor. Given $f_{0_1} : X(0_1) \to X'(0_1)$ in $\mathsf{N}_1 \to \mathsf{Set}^{\mathsf{op}}$, we need to define $F(f) : \delta\, X(0_1)\, G \to \delta\, X'(0_1)\, G$. According to Definition 5.1, it is enough to give $f_{0_1} : X'(0_1) \to X(0_1)$ and a natural transformation ρ from G to $G(- \circ f_{0_1})$. We can choose ρ to be the natural transformation whose component at $Y : [X(0_1), \mathsf{Set}^{\mathsf{op}}]$ is given by $\rho_Y = \Gamma_{\iota,\iota}(\Sigma(f_{0_1}, \mathsf{id}))$, where $\Sigma(f_{0_1}, \mathsf{id}) =_{\mathsf{def}} [\mathsf{in}_{f_{0_1}(x)}]_{x\,:\,X'(0_1)} : (\Sigma X'(0_1))(Y \circ f_{0_1}) \to (\Sigma X(0_1))\, Y$. Notice that working in $\mathsf{Set}^{\mathsf{op}}$ made sure that f_{0_1} was going in the right direction. ■

Example 5.5 (A universe closed under Π-types in $\mathsf{Fam}\,(\mathsf{Set}^{\cong})$)**.** In Section 3.2.7 we saw how to extend the code for a universe closed under Σ-types to to a code for a universe which is also closed under Π-types in $\mathsf{Fam}|\mathsf{Set}|$. The code (3.4) we used was the following:

$$\gamma_{\mathsf{N},\Pi} = \delta\, \mathsf{N}_1(X \mapsto \delta\, X(0_1)(Y \mapsto \iota((\Pi\, X(0_1))\, Y))) : \mathsf{IR}(\mathsf{Set})$$

If we try to extend this to a IR^+ code in $\mathsf{Fam}\,(\mathsf{Set})$ or $\mathsf{Fam}\,(\mathsf{Set}^{\mathsf{op}})$, we run into problems. Basically, given a morphism $f : X' \to X$, we need to construct a morphism $(\Pi\, X')(Y \circ f) \to \Pi\, X\, Y$, which of course is impossible if e.g. $X' = \mathsf{N}_0$, $X = \mathsf{N}_1$, and $Y(0_1) = \mathsf{N}_0$.
Hence the inherent contravariance in the Π-type means that $\gamma_{\mathsf{N},\Pi}$ does not extend to a $\mathsf{IR}^+(\mathsf{Set})$ or $\mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$-code. However, if we move to the groupoid $\mathsf{Set}^{\cong}$, which is the subcategory of Set with only isomorphisms as morphisms, we do get an $\mathsf{IR}^+(\mathsf{Set}^{\cong})$-code describing the universe in question, which is still living in a category beyond the strict category $\mathsf{Fam}|\mathsf{Set}|$, namely $\mathsf{Fam}\,(\mathsf{Set}^{\cong})$. It would be interesting to understand the relevance of positive induction-recursion to homotopy type theory where groupoids and their higher order relatives play such a prominent role.

■

5.3 The elimination principle for IR^+

From Example 5.4 we know that the IR-code $\gamma_{\mathsf{N},\Sigma}$ defining a universe containing the set of natural numbers N and closed under Σ-type can be extended to a IR^+-code of type $\mathsf{IR}^+(\mathsf{Set}^{\cong})$ or $\mathsf{IR}^+(\mathsf{Set}^{\mathsf{op}})$. Thus, the code $\gamma_{\mathsf{N},\Sigma}$ can be interpreted as an endofunctor on $\mathsf{Fam}\,(\mathsf{Set}^{\cong})$ or on $\mathsf{Fam}\,(\mathsf{Set}^{\mathsf{op}})$ respectively. In this section we aim to explore by means of an example what the elimination principle for IR^+-codes can be used for: we show how the simple elaboration of the code $\gamma_{\mathsf{N},\Sigma}$ to a code of type $\mathsf{IR}^+(\mathsf{Set}^{\cong})$ offers us the possibility to implement a more sophisticated recursion principle on the universe we are currently building.
Indeed, by working in $\mathsf{Fam}\,(\mathbb{C})$ instead of $\mathsf{Fam}\,|\mathbb{C}|$, we are allowing many more algebras compared to ordinary inductive-recursive definitions, or put differently, we get a stronger elimination principle.

Example 5.6. To see why a stronger elimination principle is sometimes necessary, consider the initial algebra $((\mathsf{U}^*, \mathsf{T}^*), (\mathsf{intro}_0, \mathsf{intro}_1))$ for a code $\gamma_{\mathsf{N}_1,\mathsf{N},\Sigma} : \mathsf{IR}^+(\mathsf{Set}^{\cong})$ representing a universe containing the set N_1, the set N of natural numbers and moreover closed under Σ-types. The universe U^* contains many codes for "the same" set, up to isomorphism. For instance, it contains codes for each of the following isomorphic sets:

$$\begin{aligned} \mathsf{1} &\cong (\Sigma \mathsf{1})\mathsf{1} \cong (\Sigma \mathsf{1})(\Sigma \mathsf{1})\mathsf{1} \ \ldots \\ \mathsf{N} &\cong (\Sigma \mathsf{N})\mathsf{1} \cong (\Sigma \mathsf{1})\mathsf{N} \cong (\Sigma \mathsf{1})(\Sigma \mathsf{1})\mathsf{N} \ \ldots \end{aligned}$$

Moreover, for each Σ-type the following isomorphism holds:

$$(\Sigma\, c : (\Sigma\, a : A)B(a))C(z) \cong (\Sigma\, a : A)(\Sigma\, b : B(x))C(a, b) \tag{5.1}$$

Therefore, for each Σ set with at least two nested Σ's, the universe U^* contains a code for both these ways to parenthesize a Σ-type. It might be advantageous to instead keep a single representative for each isomorphism class. We might hope to do so using the initiality of $(\mathsf{U}^*, \mathsf{T}^*)$, and indeed, the elimination principle for positive inductive-recursive definitions allows us to do exactly that.
First of all we need to decide what normal forms for elements in the universe we want. We can specify this by defining a predicate $\mathsf{NF} : \mathsf{U}^* \to \mathsf{Set}$ on the

universe $(\mathsf{U}^*, \mathsf{T}^*)$, which decides if a set is in normal form: we decree that the codes for the sets N_1 and N are in normal form, and a code for $(\Sigma\, A)\, B$ is in normal form if A is in normal form, $B(a)$ is in normal form for for each $a : A$, A is not $\mathsf{1}$ and $B(a)$ is not $\mathsf{1}$ for all $a{:}\, A$, and finally it is of the form of the right hand side of (5.1) (the left hand side would work equally well). We now define a new family $(U_{\mathsf{NF}}, T_{\mathsf{NF}})$, containing sets in normal forms only, by letting

$$\begin{aligned} U_{\mathsf{NF}} &=_{\mathsf{def}} (\Sigma u{:}\, \mathsf{U}^*)\mathsf{NF}(u) \\ T_{\mathsf{NF}}(u, p) &=_{\mathsf{def}} \mathsf{T}^*(u) \end{aligned}$$

We can also define a morphism $(\phi, \eta) : [\![\gamma_{1,\mathsf{N},\Sigma}]\!](U_{\mathsf{NF}}, T_{\mathsf{NF}}) \to (U_{\mathsf{NF}}, T_{\mathsf{NF}})$ in $\mathsf{Fam}\,(\mathsf{Set}^{\cong})$ which endows $(U_{\mathsf{NF}}, T_{\mathsf{NF}})$ with an $[\![\gamma_{1,\mathsf{N},\Sigma}]\!]$-algebra structure. For this, it is crucial that we are working in $\mathsf{Fam}\,(\mathsf{Set}^{\cong})$ and not $\mathsf{Fam}\ |\mathsf{Set}|$, since we can only expect that a Σ-type of normal forms is isomorphic to a normal form, not equal to one; i.e. if A is in normal form, and $B(a)$ is in normal form for all $a : A$, then $(\Sigma\, A)\, B$ is not necessary normal (as e.g. $A = \mathsf{1}$ shows), but we can always find a normal form isomorphic to $(\Sigma\, A)\, B$. The function ϕ maps A and B to this normal form, and η is a proof that it is indeed isomorphic to $(\Sigma\, A)\, B$.
By initiality of $(\mathsf{U}^*, \mathsf{T}^*)$ we automatically get a morphism $(\mathsf{nf}, \mathsf{correct})$ making the following diagram commute:

$$\begin{array}{ccc} [\![\gamma_{1,\mathsf{N},\Sigma}]\!](\mathsf{U}^*, \mathsf{T}^*) & \xrightarrow{(\mathsf{intro}_0,\, \mathsf{intro}_1)} & (\mathsf{U}^*, \mathsf{T}^*) \\ \Big\downarrow{\scriptstyle [\![\gamma_{1,\mathsf{N},\Sigma}]\!](\mathsf{nf},\, \mathsf{correct})} & & \Big\downarrow{\scriptstyle (\mathsf{nf},\, \mathsf{correct})} \\ [\![\gamma_{1,\mathsf{N},\Sigma}]\!](U_{\mathsf{NF}}, T_{\mathsf{NF}}) & \xrightarrow{(\phi,\, \eta)} & (U_{\mathsf{NF}}, T_{\mathsf{NF}}) \end{array}$$

The map $(\mathsf{nf}, \mathsf{correct})$ recursively computes the normal form for each set in the universe $(\mathsf{U}^*, \mathsf{T}^*)$. Indeed, $\mathsf{nf}{:}\, \mathsf{U}^* \to U_{\mathsf{NF}}$ maps each name u of a set $\mathsf{T}^*(u)$ in the universe to the name of the corresponding set in normal form, while the natural transformation $\mathsf{correct}_u{:}\, \mathsf{T}^*(u) \cong T_{\mathsf{NF}}(\mathsf{nf}(u))$ ensures that the code actually denotes isomorphic sets.

■

Example 5.7. As another example of the use of elimination principles beyond ordinary inductive-recursive definitions, we can define functions between universes with different ground sets. Consider two universes U_1, U_2 closed under the same type-theoretic operations, but containing different ground sets B_1, B_2. Given a function $B_1 \to B_2$, we would like to be able to extend this function to a function $U_1 \to U_2$ between all of the two universes. For example, we could have a universe $(U_{\mathsf{N},\Sigma}, T_{\mathsf{N},\Sigma})$, closed under Σ-types and containing the natural numbers N, and another universe $(U_{\mathbb{Z},\Sigma}, T_{\mathbb{Z},\Sigma})$ also closed under Σ-types but instead containing the integers $\mathbb{Z}$ as ground set. Clearly these two universes are closely related and there ought to exist a function between them in $\mathsf{Fam}\,(\mathsf{Set}^{\mathsf{op}})$ (where the contravariance come from for the negative occurrence of U in the code for the sigma type). By the elimination principle for positive inductive-recursive definitions, it suffices to provide a function between the ground sets, i.e. a function from $\mathbb{Z}$ into N, and there are clearly plenty of such functions, for instance the absolute value function or the square function. In detail, every function $f : \mathbb{Z} \to \mathsf{N}$ induces a $\mathsf{Fam}\,(\mathsf{Set}^{\mathsf{op}})$-morphism

$$[\![\gamma_{\mathsf{N},\Sigma}]\!](U_{\mathbb{Z},\Sigma}, T_{\mathbb{Z},\Sigma}) \longrightarrow (U_{\mathbb{Z},\Sigma}, T_{\mathbb{Z},\Sigma})$$

showing that $(U_{\mathbb{Z},\Sigma}, T_{\mathbb{Z},\Sigma})$ has an $[\![\gamma_{\mathsf{N},\Sigma}]\!]$-algebra structure. Therefore, initiality of $(U_{\mathsf{N},\Sigma}, T_{\mathsf{N},\Sigma})$ gives us a map $(U_{\mathsf{N},\Sigma}, T_{\mathsf{N},\Sigma}) \to (U_{\mathbb{Z},\Sigma}, T_{\mathbb{Z},\Sigma})$ which uses f to recursively compute the embedding of $(U_{\mathsf{N},\Sigma}, T_{\mathsf{N},\Sigma})$ into $(U_{\mathbb{Z},\Sigma}, T_{\mathbb{Z},\Sigma})$. ■

5.4 Application: a container representation of nested types

Nested data types [5] have been used to implement a number of advanced data types in languages which support higher-kinded types, such as the widely-used functional programming language Haskell. Among these data types are those with constraints, such as perfect trees [58]; types with variable binding, such as untyped λ-terms [42]; cyclic data structures [49]. In Example 1.4 the type of untyped lambda terms have been introduced as a paradigmatic indexed inductive definition. There we used the type N as index-type

stratifying lambda terms according to the number of free variables occurring in a term. But, in a functional language like Haskell which does not support dependent types, the type `Lam` can instead be presented as a nested data type. In Haskell we can define $\texttt{Lam} : \texttt{Set} \to \texttt{Set}$ as follows:

```
data Lam A = Var A | App (Lam A) (Lam A) | Abs (Lam (Maybe A))
```

The type `Lam A` is the type of untyped λ-terms over variables of type `A` up to α-equivalence. Here, the constructor `Abs` models the bound variable in an abstraction of type `Lam A` by the `Nothing` constructor of type `Maybe A`, and any free variable `x` of type `A` in an abstraction of type `Lam A` by the term `Just x` of type `Maybe A`.

Note the important difference with the representation of `Lam` given in Example 1.4: there the domain of `Var` is `Fin n` for $\texttt{n} : \texttt{Nat}$, while here the domain of `Var` is a type variable `A`. However, both these presentations emphasize the intrinsic feature of the type `Lam`: elements of the type `Lam (Maybe A)` are needed to build elements of `Lam A` so that, in effect, the entire family of types determined by `Lam` has to be constructed simultaneously. Thus, rather than defining a `Nat`-indexed family of inductive types, the type constructor `Lam` defines an *type-indexed* inductive family of types.

This section uses IR^+ to show that nested data types like `Lam` are representable as containers (see Section 2.2.2). This fact allows one to apply the rich structure of containers to nested data types, e.g. one could classify the natural transformations between them and operate on them using, for example, the derivative. We sketch the overall development as follows:

- in Section 5.4.1 we define a basic grammar Nest for defining nested types and a decoding function $(\!|-|\!) \colon \mathsf{Nest} \to [\mathsf{Set}, \mathsf{Set}] \to [\mathsf{Set}, \mathsf{Set}]$.

- In Section 5.4.2 we show that $(\!|N|\!)$ restricts to an endofunctor $(\!|N|\!)_{\mathsf{Cont}}$ on the category Cont of containers. We then use IR^+ to define $(\!|N|\!)_{\mathsf{Cont}}$ and conclude that its initial algebra $\mu(\!|N|\!)_{\mathsf{Cont}}$ is a container whose extension is exactly $\mu(\!|N|\!)$.

5.4.1 A grammar for nested types

We now present a possible syntax for defining nested data types. It is not the most sophisticated grammar, since our point is not to push the theory of nested data types, but rather to illustrate an application of positive induction-recursion to nested data types. The grammar we use is

$$\mathcal{F} = \mathsf{Id} \mid K_{(S,P)} \mid \mathcal{F} + \mathcal{F} \mid \mathcal{F} \times \mathcal{F} \mid \mathcal{F} \circledast \mathcal{F}$$

where (S, P) is any container. The intention is that Id stands for the identity functor mapping a functor to itself, $K_{(S,P)}$ stands for the constant functor mapping any functor to the interpretation of the container (S, P), $+$ stands for the coproduct of functors, $\times$ for the product of functors and $\circledast$ for the pointwise composition of functors. These intentions are formalised by a semantics for the elements of our grammar given as follows

$$\begin{array}{lcl}
(\!|-|\!) & : & \mathsf{Nest} \to [\mathsf{Set}, \mathsf{Set}] \to [\mathsf{Set}, \mathsf{Set}] \\
(\!|\mathsf{Id}|\!)\ F & = & F \\
(\!|K_{(S,P)}|\!)\ F & = & [\![(S, P)]\!]_{\mathsf{C}} \\
(\!|\mathcal{F} + \mathcal{G}|\!)\ F & = & (\!|\mathcal{F}|\!)\ F + (\!|\mathcal{G}|\!)\ F \\
(\!|\mathcal{F} \times \mathcal{G}|\!)\ F & = & (\!|\mathcal{F}|\!)\ F \times (\!|\mathcal{G}|\!)\ F \\
(\!|\mathcal{F} \circledast \mathcal{G}|\!)\ F & = & (\!|\mathcal{F}|\!)\ F \circ (\!|\mathcal{G}|\!)\ F
\end{array}$$

For example, the functor

$$L\ F\ X \ = \ X \ + \ (FX \times FX) \ + \ F(X + 1)$$

whose initial algebra is the type `Lam` is of the form $(\!|N_L|\!)$ where

$$N_L = K_{(\mathsf{N}_1,\mathsf{N}_1)} \ + \ (K_{(\mathsf{N}_1,\mathsf{N}_2)}) \circledast \mathsf{Id} \ + \ \mathsf{Id} \circledast (K_{(S_{\mathsf{N}},P_{\mathsf{N}})})$$

where $(1, 1)$ is the container with one shape and one position which represents the identity functor on Set, $(1, 2)$ is the container with one shape and two positions which represents the functor mapping X to $X \times X$ and $(S_{\mathsf{N}}, P_{\mathsf{N}})$ is the container described in Example 2.37 representing the functor on Set mapping X to $X + 1$.

5.4.2 Representing nested types as containers

We start showing that every element N of Nest can be interpreted as an operator on containers $(\!|N|\!)_{\mathsf{Cont}} : \mathsf{Cont} \longrightarrow \mathsf{Cont}$ as exemplified by the following diagram

$$\begin{array}{ccc} \mathsf{Cont} & \xrightarrow{[\![-]\!]_{\mathsf{C}}} & [\mathsf{Set},\mathsf{Set}] \\ {\scriptstyle (\!|N|\!)_{\mathsf{Cont}}}\downarrow & & \downarrow{\scriptstyle (\!|N|\!)} \\ \mathsf{Cont} & \xrightarrow[[\![-]\!]_{\mathsf{C}}]{} & [\mathsf{Set},\mathsf{Set}] \end{array}$$

We can check that Cont is indeed closed under $(\!|N|\!)$, for N an element in Nest (by which we mean that the image of $(\!|N|\!)$ restricted to Cont is still an object of Cont), by induction on the structure of N by noting that containers are closed under coproduct, product and under composition. Thus, for example, we define $(S_L, P_L) = (\!|N_L|\!)_{\mathsf{Cont}}(S, P)$, where

$$\begin{array}{lcl} S_L & = & 1 + (S \times S) + (\Sigma s : S)\ (P(s) \to 2) \\ P_L\ (\mathsf{in}_1\, 0_1) & = & \mathsf{N}_1 \\ P_L\ (\mathsf{in}_2\, (s, s')) & = & P(s) + P(s') \\ P_L\ (\mathsf{in}_3\, (s, f)) & = & (\Sigma\, p : P(s))\ \mathsf{case}_2(f(p), \mathsf{N}_1, \mathsf{N}_0) \end{array}$$

Now, we want to show that for every code $N{:}\,\mathsf{Nest}$, the functor $(\!|N|\!)_{\mathsf{Cont}}$ is an IR^+ functor. The key idea is to exploit the identification of Cont with $\mathsf{Fam}\,(\mathsf{Set}^{\mathsf{op}})$ (see Remark 2.32). If we examine the constructions on families used to build $(\!|N|\!)_{\mathsf{Cont}}$ we note that the only delicate construction is the use of Σ-types to model the composition operator used in the definition of nested types and derived by container composition as shown, for example, in the definition of S_L and P_L. But, as we have seen in Example 5.4, families closed under Σ-types are canonical examples of a IR^+ construction. Thus, as shown in Section 5.6, we can define the initial algebra of the IR^+ functor $(\!|N|\!)_{\mathsf{Cont}}$ in the usual way as the colimit of the initial chain. We are left to show that the extension of the container $\mu(\!|N|\!)_{\mathsf{Cont}}$ built this way is the functor $\mu(\!|N|\!) : \mathsf{Set} \to \mathsf{Set}$. From Lemma 5.11 in Section 5.6 we know that the initial algebra chain of an IR^+ functor is made from cartesian morphisms only (see

Definition 6.1), and since $[\![-]\!]_{\mathsf{C}}$ preserves initial objects and filtered colimits of cartesian morphisms ([1] Propositions 4.5.1 and 4.6.7), we can indeed conclude that $[\![\mu(\!|N|\!)_{\mathsf{Cont}}]\!]_{\mathsf{C}} \cong \mu(\!|N|\!)$ showing that all nested types indeed are definable using containers.

5.5 Comparison to plain IR

We now investigate the relationship between IR^+ and IR. On the one hand we show in Proposition 5.8 how to embed Dybjer and Setzer's original coding scheme for IR into IR^+; this way we can see IR as a subsystem of IR^+. On the other hand we show in Proposition 5.9 that on discrete categories the two schemas agree having the same functorial interpretation.
Recall from Remark 3.13 that every type D can be regarded as a discrete category $|D|$. In the other direction, every category $\mathbb{C}$ gives rise to a type $|\mathbb{C}|$ whose elements are the objects of $\mathbb{C}$.

Proposition 5.8. There is a function $\varphi : \mathsf{IR}(D) \to \mathsf{IR}^+|D|$ s.t.

$$[\![\gamma]\!]_{\mathsf{IR}(D)} \cong [\![\varphi(\gamma)]\!]_{\mathsf{IR}^+|D|}$$

Proof. The only interesting case is when γ is a δ-code. Therefore, let γ be $\delta\, B\, F : \mathsf{IR}(D)$. Recursively applying ϕ we get a map $F : (B \to |D|) \to \mathsf{IR}^+(|D|)$. We need to ensure that F is indeed a functor, but since $|D|$ is a discrete category and B is a set, $(B \to |D|)$ is also discrete. Hence the mapping on objects $F : (B \to |D|) \to \mathsf{IR}^+(|D|)$ can trivially be extended to a functor $(B \to |D|) \to \mathsf{IR}^+(|D|)$. It is easy to see that the two semantics do agree: they have the same action on objects and on morphisms whose second component is an identity, which are the only morphisms of $\mathsf{Fam}|D|$ since $|D|$ is discrete. □

This proposition shows that the theory of IR can be embedded in the theory of IR^+. In the next proposition we make this result more precise: using the embedding $|-| : \mathsf{CAT} \to \mathsf{SET}$ which assigns to each category the collection of its objects and to each functor its action on objects only, we show that forgetting about the extra structure in IR^+ simply gets us back to plain IR.

Proposition 5.9. Let $|-|:\mathsf{CAT}\to\mathsf{SET}$ be the functor assigning to each category the collection of its objects and to each functor its action on objects only. There is a function $\psi:\mathsf{IR}^+\mathbb{C}\to\mathsf{IR}\,|\mathbb{C}|$ such that for all $\gamma:\mathsf{IR}^+\mathbb{C}$

$$[\![\psi(\gamma)]\!]_{\mathsf{IR}\,|\mathbb{C}|}\cong[\![\gamma]\!]_{\mathsf{IR}^+\mathbb{C}}\,|_{\mathsf{Fam}|\mathbb{C}|}$$

where $[\![\gamma]\!]_{\mathsf{IR}^+\mathbb{C}}\,|_{\mathsf{Fam}|\mathbb{C}|}$ indicates the restriction of $[\![\gamma]\!]_{\mathsf{IR}^+\mathbb{C}}$ to the subcategory of $\mathsf{Fam}\,(\mathbb{C})$ obtained by restricting morphisms to those whose second component is an identity. Furthermore, $\psi\circ\varphi=\mathsf{id}$.

The proof is again a simple induction on the structure of the codes.

5.6 Existence of initial algebras

We briefly revisit the initial algebra argument given in Section 3.3.2. Inspecting the proof, we see that it indeed is possible to adapt it also for the more general setting of positive inductive-recursive definitions by making the appropriate adjustments.

Call a morphism $(h,k):(X,T)\to(Y,Q)$ in $\mathbf{Fam}\,(\mathbb{C})$ *splitting* if $k=\mathsf{id}_T$, i.e. $Q\circ h=T$. As we will make it clear in Section 6.2, splitting morphisms are specific Cartesian morphisms for the family fibration $\pi:\mathbf{Fam}\,(\mathbb{C})\to\mathsf{Set}$. Therefore $\mathbf{Fam}|\mathbb{C}|$ is the subcategory of $\mathbf{Fam}\,(\mathbb{C})$ with the same objects, but with morphisms the splitting ones only (cf. Observation 3.12).

Inspecting the proofs in Section 3.3.2, we see that they crucially depend on morphisms being splitting in several places. And, indeed, the morphisms involved in the corresponding proofs for IR^+ actually are splitting.

As is well-known, a weaker condition than κ-continuity is actually sufficient: it is enough that the functor in question preserves the specific colimit of the initial κ-chain. We thus show that the initial chain of a IR^+ functor actually lives in $\mathbf{Fam}|\mathbb{C}|$, which will allow us to modify Dybjer and Setzer's proof accordingly.

Lemma 5.10. For every code $\gamma:\mathsf{IR}^+\,\mathbb{C}$ the induced functor $[\![\gamma]\!]:\mathbf{Fam}\,(\mathbb{C})\to\mathbf{Fam}\,(\mathbb{C})$ preserves splitting morphisms, i.e. if (f,g) is splitting, then so is $[\![\gamma]\!](f,g)$.

Proof. By induction on the structure of the code. The interesting case is $\gamma = \delta\, B\, F$. Let $(h, \mathsf{id}) : (X, P \circ h) \to (Y, P)$ be a splitting morphism. We have

$$\begin{aligned}[\![\delta\ B\ F]\!](h, \mathsf{id}) &= [\mathsf{in}_{h \circ g} \circ [\![F(P \circ h \circ g)]\!](h, \mathsf{id}) \circ [\![F_{\to}(g^*(\mathsf{id}))]\!]_{(X,T)}]_{g : B \to X} \\ &= [\mathsf{in}_{h \circ g} \circ [\![F(P \circ h \circ g)]\!](h, \mathsf{id})]_{g : B \to X}\end{aligned}$$

where $[\![F(g^*\mathsf{id})]\!]_{(X,T)} = \mathsf{id}$ since both g^*, F and $[\![\_]\!]$ are functors. By the inductive hypothesis, each $[\![F(P \circ h \circ g)]\!](h, \mathsf{id})$ is splitting. Furthermore injections are splitting in $\mathbf{Fam}\,(\mathbb{C})$. Since splitting morphisms are closed under composition and the cotuple of splitting morphisms is also a splitting in $\mathbf{Fam}\,(\mathbb{C})$ we conclude that $[\![\delta\ B\ F]\!](h, \mathsf{id})$ is a splitting morphism. □

Lemma 5.11. For each $\gamma : \mathsf{IR}^+\,\mathbb{C}$, the initial chain consists of splitting morphisms only.

Proof. Recall from Section 2.2.1 that the connecting morphisms $\omega_{j,k} \colon [\![\gamma]\!]^j(\bot) \to [\![\gamma]\!]^k(\bot)$ are uniquely determined as follows:

- $\omega_{0,1} = !_{[\![\gamma]\!](\bot)}$ is unique.
- $\omega_{j+1,k+1}$ is $[\![\gamma]\!](\omega_{j,k}) \colon [\![\gamma]\!]([\![\gamma]\!]^j(\bot)) \to [\![\gamma]\!]([\![\gamma]\!]^k(\bot))$.
- $\omega_{j,k}$ is the colimit cocone for j a limit ordinal.

We prove the statement by transfinite induction on j. For $\gamma = \bot$ the statement is trivially true. At successor stages, we can directly apply Lemma 5.10 and the inductive hypothesis. Finally, at limit stages, we use the fact that the colimit lives in $\mathbf{Fam}|\mathbb{C}|$, so that the colimit cocone is splitting. □

Inspecting the proofs that IR-functors have an initial algebra in Section 3.3.2, we see that it now goes through also for IR^+ if we insert appeals to Lemma 5.11 where necessary. To finish the proof, we also need to ensure that $\mathbf{Fam}\,(\mathbb{C})$ has κ-filtered colimits; this is automatically true if $\mathbb{C}$ has all small connected colimits (compare Remark 3.10), since $\mathbf{Fam}\,(\mathbb{C})$is then cocomplete. Note that discrete categories have all small connected colimits for trivial reasons.

Theorem 5.12. Assume that a Mahlo cardinal exists in the meta-theory. If $\mathbb{C}$ has connected colimits, then every functor $[\![\gamma]\!]$ for $\gamma\colon \mathsf{IR}^+\,\mathbb{C}$ has an initial algebra. □

Chapter 6

Fibred induction-recursion

Abstract In this chapter we introduce a theory of fibered data types which broadens the class of data types definable by the theory of IR. Fibered data types are inductive-recursive types regarded as objects of the total category of a fibration. We give a syntax which defines codes for fibered data types and a corresponding semantics which interprets these codes as functors on the total category of a fibration. We present examples of fibered data types and we justify their existence by adapting the original initial algebra argument for inductive-recursive types.

6.1 Introduction

In this chapter we introduce fibrations as they offer a conceptual cleaner treatment of induction-recursion which, simultaneously, significantly broadens the class of data types which can be defined by induction-recursion.
Fibrations proved to be extremely successful in modeling type theories (see for example Jacobs' monograph [64]). In particular, when discussing dependent types it becomes natural to distinguish between a base category of indices and a total category of indexed objects. This perspective offers a clean categorical infrastructure to capture indexed data types and, as we argue in this chapter, also inductive-recursive types.
The characteristic feature of an inductive recursive type, as already remarked in several places, is the simultaneous definition of a type X and of a recursive

function $T : X \to D$. With respect to this, the advantage of using fibrations to organize these data types is then twofold: on the one hand we get a more abstract and versatile setting by regarding X as an index, and therefore as an object of the base category of a fibration, and $T : X \to D$ as an indexed object, i.e. an object of the total category of a fibration- "above" X. On the other hand, we can reflect the interplay arising from the mutual definition of X and T by the use of objects in the base category to define the action of IR functors on objects of the total category.
A more concrete motivation for the use of fibrations to model induction-recursion arises when we ask what happens if we wish to define some other form of indexed structure where the indices are generated at the same time as the data so indexed. For example, we may wish to define in an inductive-recursive way any of the following structures:

- An extensional family $T : X \to \mathsf{Setoid}$ where X is a setoid and T preserves the setoid structure of X.
- A presheaf $T : \mathbb{C} \to \mathsf{Set}$ where $\mathbb{C}$ is a category and T is a functor.
- A category with families which is a functor $T : \mathbb{C}^{\mathsf{op}} \to \mathsf{Fam}\,(\mathsf{Set})$ where $\mathbb{C}$ is a category thought of as a category of contexts.
- An indexed category/split fibration $T : \mathbb{C}^{\mathsf{op}} \to \mathsf{Cat}$.

These structures are not covered by any of the schemas seen in the previous chapters. Therefore, after a brief recap on fibrations, we start looking for a generalization of the original IR constructors suitable to be interpreted in the setting of a fibration. This analysis will also suggest the necessary structure a fibration needs to possess in order to get a sound interpretation of these constructors as functors between the total category of the fibration. At that point we will see the above mentioned structures as instances of our schema and precisely as initial algebra of functors defined by our new constructors.

6.2 Fibrations in a nutshell

The aim of this chapter is to obtain inductive-recursive definitions for different structures by appropriately instantiating a single, generic theory of

induction-recursion. To pursue such a uniform approach we turn to the language of fibrations [64]. In this section we collect some standard material about fibrations which will be used in the rest of this chapter.

Definition 6.1. Let $p : \mathbb{E} \to \mathbb{B}$ be a functor. A morphism $g : Q \to P$ in $\mathbb{E}$ is said Cartesian over a morphism $f : X \to Y$ in $\mathbb{B}$ if $p(g) = f$, and for every $g' : Q' \to P$ in $\mathbb{E}$ for which $p(g') = f \circ v$ for some $v : p(Q') \to X$ there exists a unique $h : Q' \to Q$ in $\mathbb{E}$ such that $p(h) = v$ and $g \circ h = g'$. ■

We indicate with $f^{\S}_P$ a Cartesian morphism over a morphism f with codomain $p(P)$. Cartesian morphisms can be easily recognized to be unique up to isomorphism. If P is an object of $\mathbb{E}$, then we write f^*P for the domain of $f^{\S}_P$. Cartesian morphisms are the essence of fibrations, as the following definition shows.

Definition 6.2. Let $p : \mathbb{E} \to \mathbb{B}$ be a functor. Then p is a fibration if for every object P of $\mathbb{E}$ and every morphism $f : X \to p(P)$ in $\mathbb{B}$ there is a Cartesian morphism $f^{\S}_P : Q \to P$ in $\mathbb{E}$ such that $p(f^{\S}_P) = f$. ■

If $p : \mathbb{E} \to \mathbb{B}$ is a fibration, we call $\mathbb{B}$ the *base category* of p and $\mathbb{E}$ the *total category* of p. In the rest of the chapter we assume the base category is locally small; this will enable us later to take coproducts indexed by the morphisms of specific hom-sets (cf. Lemma 6.9). Objects of the total category $\mathbb{E}$ can be thought of as indexed entities while objects of the base category $\mathbb{B}$ can be thought of as indices. Therefore the functor p can be thought of as mapping each indexed structure P in $\mathbb{E}$ to the index $p(P)$ of P. We say that an object P in $\mathbb{E}$ is *above* its image $p(P)$ under p, and similarly for morphisms. For any object X of $\mathbb{B}$, we write $\mathbb{E}_X$ for the *fibre above* X, i.e., for the subcategory of $\mathbb{E}$ consisting of objects above X and morphisms above id_X.
A fibration is called *cloven* if it comes with a choice of Cartesian liftings, and *split* if this choice is further done functorially, i.e. $\mathsf{id}^* = \mathsf{id}$ and $(v \circ u)^* = u^*\, v^*$. If the fibration $p : \mathbb{E} \to \mathbb{B}$ is cloven, and $f : X \to Y$ is a morphism in $\mathbb{B}$, then the function mapping each object P of $\mathbb{E}$ to f^*P extends to a functor $f^* : \mathbb{E}_Y \to \mathbb{E}_X$. We call the functor f^* the *reindexing functor induced by* f.
In general, given an arbitrary fibration $p : \mathbb{E} \to \mathbb{B}$ we can always obtain a new fibration, called the groupoid fibration associated to p, by restricting

the morphisms of $\mathbb{E}$ to be just the Cartesian morphisms of p. Indeed, from $\mathbb{E}$ we obtain a category $\mathbb{E}^{\mathsf{c}}$ whose objects are the same as those of $\mathbb{E}$ and whose morphisms are the Cartesian morphisms of p.

Lemma 6.3. The functor $p^{\mathsf{c}} : \mathbb{E}^{\mathsf{c}} \hookrightarrow \mathbb{E} \to \mathbb{B}$ obtained by restricting p to the category $\mathbb{E}^{\mathsf{c}}$ is a fibration.

Proof. $\mathbb{E}^{\mathsf{c}}$ is a subcategory of $\mathbb{E}$ since all isomorphisms are Cartesian and Cartesian morphisms are closed by composition. The fact that $\mathbb{E}^{\mathsf{c}} \to \mathbb{B}$ is a fibration is just a consequence of an abstract formulation of the pullback lemma in terms of fibrations stating that given composable morphisms g and f, if both g and $g \circ f$ are cartesian, then also f is cartesian. □

When p is split, we can further consider the subcategory $\mathbb{E}^{\mathsf{sp}}$ of $\mathbb{E}^{\mathsf{c}}$ consisting of Cartesian morphisms coming from the splitting only. In this case we restrict the morphisms of $\mathbb{E}^{\mathsf{c}}$ to a choice of Cartesian morphisms such that $\mathsf{id}^* = \mathsf{id}$ and $(v \circ u)^* = u^*\, v^*$ for every morphism u and v in $\mathbb{E}^{\mathsf{sp}}$. Morphisms in $\mathbb{E}^{\mathsf{sp}}$ are called *splitting morphisms*. As is common when modelling type theories, we will only be concerned with split fibrations. This is not a restriction, since every fibration is equivalent to a split one (Jacobs [64] Corollary 5.2.5). Of course, Lemma 6.3 applies to $p^{\mathsf{sp}} : \mathbb{E}^{\mathsf{sp}} \hookrightarrow \mathbb{E} \to \mathbb{B}$ as well. We collect two immediate but useful facts about $\mathbb{E}^{\mathsf{sp}}$ in the following lemma:

Lemma 6.4. Let $K : \mathbb{E} \to \mathbb{B}$ be a split fibration.

1. If f is a splitting morphism, then $(pf)^{\S} = f$.
2. Each fibre $\mathbb{E}^{\mathsf{sp}}_A$ is discrete.

Proof. The first item is obvious since a splitting is nothing but a choice of Cartesian morphisms. The second is just a consequence of the fact that the fibre of $\mathbb{E}^{\mathsf{c}}$ are groupoids since vertical cartesian morphisms are always isomorphisms. □

That $\mathsf{Fam}|D|$ is fibred over Set, as noted in Remark 3.10, is the crucial observation which allowed us to realise that induction-recursion can be recast in a fibred setting. But before we get to that, we now give some examples of fibrations which will be used later in this chapter.

Example 6.5 (The domain fibration)**.** Let $\mathbb{B}$ be a category. The *arrow category* of $\mathbb{B}$, denoted $\mathbb{B}^{\to}$, has the morphisms of $\mathbb{B}$ as its objects. A morphism in $\mathbb{B}^{\to}$ from $f : X \to Y$ to $f' : X' \to Y'$ is a pair (α_1, α_2) of morphisms in $\mathbb{B}$ such that $f' \circ \alpha_1 = \alpha_2 \circ f$. The domain functor dom maps an object $f : X \to Y$ of $\mathbb{B}^{\to}$ to the object X of $\mathbb{B}$. This functor is always a fibration called the *domain fibration*; the reindexing of an object $f : X \to Y$ above X by a morphism $g : X' \to X$ in $\mathbb{B}$ being simply the composite $f \circ g : X' \to Y$. For each object A of $\mathbb{B}$ there is an obvious domain fibration $\mathsf{dom}_A : \mathbb{B}/A \to \mathbb{B}$ obtained by the restriction of dom to the fiber above A. ■

Example 6.6 (The families fibration)**.** In Definition 3.9 we defined the category $\mathsf{Fam}\,(\mathbb{C})$ for any category $\mathbb{C}$. The functor $\pi : \mathsf{Fam}\,(\mathbb{C}) \to \mathsf{Set}$ mapping (X, T) to X is a split fibration called the *families fibration*. Given a family $T : X \to \mathbb{C}$ above X and a morphism $f : X' \to X$ in Set, the reindexing of T by f is the family $T \circ f : X' \to \mathbb{C}$ with the splitting arrow $(f, \mathsf{id}) : T \circ f \to T$ having as first component f and as second component the X'-indexed collection of identity morphisms. ■

The following definition introduce a generalization of the families fibration which will turn useful in Section 6.5 when we verify that our examples meet the required conditions to interpret the functors we are interested in.

Definition 6.7. Let $\mathbb{A}$ be a category, $F : \mathbb{A} \to \mathsf{CAT}$ a functor and $\mathbb{D}$ a category. The lax comma category $F \downarrow_{\text{lax}} \mathbb{D}$ has objects pairs (X, T) where X is an object in $\mathbb{A}$ and T is a functor $F(X) \to \mathbb{D}$. A morphism from (X, T) to (X', T') is a pair (f, g) where $f : X \to X'$ and $g : T \xrightarrow{\cdot} T' \circ F(f)$. ■

Proposition 6.8. The category $\mathbb{E} =_{\mathsf{def}} F \downarrow_{\text{lax}} \mathbb{D}$ is fibred over $\mathbb{A}$ via a split fibration $p : \mathbb{E} \to \mathbb{A}$. The subfibration $p^{\mathsf{sp}} : \mathbb{E}^{\mathsf{sp}} \to \mathbb{B}$ is given by the comma category $F \downarrow \mathbb{D}$. If $\mathbb{A}$ has coproducts and F preserves them, then also $\mathbb{E}^{\mathsf{sp}}$ has coproducts.

Proof. The fibration $p : \mathbb{E} \to \mathbb{A}$ maps an object (X, T) to X and a morphism (f, g) to f. Given a morphism $f : Y \to p(X, T)$ in $\mathbb{A}$, we choose $(f, \mathsf{id}) : (Y, T \circ F(f)) \to (X, T)$ to be the Cartesian morphism above f. This choice is clearly functorial in f, and thus the splitting morphisms are exactly those with second component identities, or equivalently, morphisms in $F \downarrow \mathbb{D}$.

Now assume $\mathbb{A}$ has coproducts and that F preserves them. Then, coproducts in $F \downarrow \mathbb{D}$ are given by $(X,T) + (X',T') =_{\mathsf{def}} (X + X', [T,T'] \circ \phi)$ where $\phi : F(X + X') \to FX + FX'$ witnesses that F preserves $+$. Injections are given by $\mathsf{inl} = (\mathsf{inl}_{X+X'}, \mathsf{id})$, and $\mathsf{inr} = (\mathsf{inr}_{X+X'}, \mathsf{id})$. □

6.3 Fibred IR-codes

We look once more at the three constructors for IR-codes given in Definition 3.7, and contemplate how we can generalise them to an arbitrary fibration. For simplicity we look at $\mathsf{IR}(D,D)$-codes, i.e. IR-codes with same input and output parameter D. The key idea which guides this abstraction is to regard the original schema given in Definition 3.7 from the perspective of the family fibration $\pi : \mathsf{Fam}\,(D) \to \mathsf{Set}$. We abstract on this fibrational structure stepwise considering one constructor at a time.

(ι) The first of the three IR constructors is ι whose effect is to define a constant functor returning a given family. Because families are objects of the total category of the families fibration, we can generalise the ι constructor to an arbitrary fibration $p : \mathbb{E} \to \mathbb{B}$ by the following rule

$$\frac{P : \mathbb{E}}{\iota\, P : \mathsf{IR}\,(p)}$$

whose intent is that ιP is a code for the constantly P valued functor on the total category of the fibration p.

(σ) The second IR constructor is σ whose effect is to take set-indexed coproducts of functors given by codes. Such an A-indexed collection of codes can be given by a function $f : A \to \mathsf{IR}\,(p)$. Hence we can generalise the σ constructor to an arbitrary fibration $p : \mathbb{E} \to \mathbb{B}$ by the following rule

$$\frac{A : \mathsf{Set} \quad f : A \to \mathsf{IR}\,(p)}{\sigma\, A\, f : \mathsf{IR}\,(p)}$$

(δ) The third of the three IR constructors is δ. The premise of this constructor is a function $(B \to D) \to \mathsf{IR}(D,D)$ for a fixed index B. Notice that

$(B \to D)$ is just the fibre above B in the families fibration $\mathsf{Fam}\,(D)$. Thus we may abstract δ to an arbitrary fibration $p : \mathbb{E} \to \mathbb{B}$ as follows:

$$\frac{B : \mathbb{B} \quad F : |\mathbb{E}_B| \to \mathsf{IR}\,(p)}{\delta\, B\, F : \mathsf{IR}\,(p)}$$

We can already appreciate the fibrational framework as paying dividends in terms of cleaning up the syntax of IR-codes.

6.4 Fibred IR-functors

Now we turn to the semantic content of our fibred IR-codes. Since our syntax has been shaped abstracting on the fibrational content revealed in the original scheme for IR, the natural setting for the interpretation of fibered IR-codes should be a fibration $p : \mathbb{E} \to \mathbb{B}$. However, recall that an $\mathsf{IR}(D, D)$-code, as defined in Definition 3.7, is always interpreted as an endofunctor on $\mathsf{Fam}|D|$, where D is regarded as a discrete category. As seen in Chapter 5, to find a semantics which interprets codes as endofunctors on $\mathsf{Fam}\,(\mathbb{C})$, for not necessarily discrete categories $\mathbb{C}$, we need to modify also the syntax of IR. Therefore, to match the original semantics for IR-codes, we expect to interpret a code $\gamma : \mathsf{IR}\,(p)$ not as a functor on the total category $\mathbb{E}$ of the fibration p, but as a functor $[\![\gamma]\!]$ on a subcategory of $\mathbb{E}$. To highlight the structure of this subcategory we separate the analysis of the action of a fibered IR-functor $[\![\gamma]\!]$ on objects and on morphisms of $\mathbb{E}$.
In our abstraction, both σ and δ build Set-indexed coproducts of functors; in particular, as we are going to show in the next lemma, the interpretation of a δ codes use hom-sets in $\mathbb{B}$ as index-sets of these coproducts. Therefore we require $\mathbb{E}$ to have set-indexed coproducts and $\mathbb{B}$ to be locally small.

Lemma 6.9. Let $p : \mathbb{E} \to \mathbb{B}$ be a split fibration. Assume $\mathbb{E}$ has Set-indexed coproducts and $\mathbb{B}$ is locally small. Every fibred IR-code $\gamma : \mathsf{IR}\,(p)$ induces a mapping $[\![\gamma]\!]$ on the objects of $\mathbb{E}$.

Proof. We go through each of the constructors in turn and we show its action on an object Q in $\mathbb{E}$.

- If P is an object of $\mathbb{E}$, then the code $\iota\ P$ defines the constantly P valued map on objects of $\mathbb{E}$. Formally,

$$[\![\iota\ P]\!]\, Q = P$$

- Let A be a set and $f : A \to \mathsf{IR}\,(p)$ a function assigning to each element $a : A$ a fibred IR-code $f(a) : \mathsf{IR}\,(p)$. By the inductive hypothesis, for every $a : A$ we have an object $[\![f(a)]\!]\, Q$ of $\mathbb{E}$. Since $\mathbb{E}$ has set-indexed coproducts, we can take the coproduct of these objects to define $[\![\sigma\ A\, f]\!]\ Q$. Formally,

$$[\![\sigma\ A\, f]\!]\, Q = (\Sigma a : A)\, [\![f(a)]\!]\, Q$$

- Let B an object of $\mathbb{B}$ and $F : |\mathbb{E}_B| \to \mathsf{IR}\,(p)$ a map assigning to each object P in the fibre above B a code $F(P) : \mathsf{IR}\,(p)$. Using p we can map Q to the object $p(Q)$ in the base and we can consider the maps $g : B \to p(Q)$ in $\mathbb{B}$. For each such g, we can reindex Q along g to get an object g^*Q in the fibre above B. We can then apply F to this object to get a code $F(g^*Q) : \mathsf{IR}\,(p)$ and inductively compute the action of this code on Q. Notice that since $\mathbb{B}$ is locally small, the collection of morphisms $\mathbb{B}(B, p(Q))$ forms a set and so we can take the set-indexed coproduct over the choice of g and thus obtain the action of $[\![\delta\ B\ F]\!]$ on Q. Formally,

$$[\![\delta\ B\ F]\!]\, Q = (\Sigma g : B \to p(Q))\, [\![F(g^*\, Q)]\!]\, Q \qquad \square$$

Now we are left to show the action on morphisms of the interpretation of a code $\gamma : \mathsf{IR}\,(p)$. We already observed in Section 3.2.2 that universes arise as initial algebras for functors defined not in $\mathsf{Fam}\,(\mathsf{Set})$, but on its subcategory $\mathsf{Fam}|\mathsf{Set}|$. The same issue arises here, and we can then wonder if this condition can be lift at the level of the fibrational infrastructure. And indeed the answer is quite straightforward: the morphisms IR-functors act on are the splitting morphisms of the families fibration. So, the natural generalisation of this condition on morphisms between families that works in an arbitrary fibration is simply that $[\![\gamma]\!]$ acts only on the splitting morphisms of the fibration p. Our goal then, is to show that for a split fibration $p : \mathbb{E} \to \mathbb{B}$, each code $\gamma : \mathsf{IR}\,(p)$

gives rise to a functor $[\![\gamma]\!] : \mathbb{E}^{\mathsf{sp}} \to \mathbb{E}^{\mathsf{sp}}$. Since coproducts play an important role in the interpretation we gave on objects, we need for them to interact nicely with the splitting morphisms. Recall from Remark 3.10 that (chosen) coproducts in a category $\mathbb{E}$ give rise to a functor $\Sigma : \mathsf{Fam}\,(\mathbb{E}) \to \mathbb{E}$ which sends a family morphism (f, g) where $f : A \to A'$ and $g_x : B(x) \to B'(f(x))$ to the cotuple

$$\Sigma(f, g) = [\mathsf{in}_{f(x)} \circ g_x]_{x:A} : \Sigma(A, B) \to \Sigma(A', B').$$

which we called the generalised sum of g over f.

Lemma 6.10. Let $p : \mathbb{E} \to \mathbb{B}$ be a split fibration. If $\mathbb{E}$ has set-indexed coproducts, and if a generalised sum of splitting morphisms is splitting, then every IR-code $\gamma : \mathsf{IR}\,(p)$ induces a functor $[\![\gamma]\!] : \mathbb{E}^{\mathsf{sp}} \to \mathbb{E}^{\mathsf{sp}}$.

Proof. In Lemma 6.9, we saw how $[\![\gamma]\!]$ maps objects of $\mathbb{E}$ to objects of $\mathbb{E}$. As the objects of $\mathbb{E}$ and $\mathbb{E}^{\mathsf{sp}}$ are the same, this defines the action of $[\![\gamma]\!]$ on the objects of $\mathbb{E}^{\mathsf{sp}}$. So now let us define the action of $[\![\gamma]\!]$ on a splitting morphism $h : Q \to Q'$ by looking at the three constructors for fibred IR-codes in turn.

- If $c = \iota\, P$, then $[\![\iota\, P]\!]$ was defined to be the constantly P valued map on objects of $\mathbb{E}$. Therefore we can define

$$[\![\iota\ P]\!]\ h = \mathsf{id}_P$$

- If $c = \sigma\, A\, f$, then we defined $[\![\sigma_A f]\!]\, Q = (\Sigma a : A)\, [\![f(a)]\!]\, Q$ For the inductive hypothesis we have for every $a : A$ a splitting morphism

$$[\![f(a)]\!]\, h : [\![f(a)]\!]\, Q \to [\![f(a)]\!]\ Q'$$

  By assumption, the sum of all these morphisms is splitting and hence we can define

$$[\![\sigma\, A\, f]\!]\, h = (\Sigma a : A)\, [\![f(a)]\!]\, h$$

- If $c = \delta\, B\, F$, then we defined $[\![\delta_B F]\!]\, Q = (\Sigma\, g : B \to p(Q))\, [\![F(g^* Q)]\!]\, Q$. Now we have to exhibit a map

$$\big((\Sigma\, g : B \to p(Q))\, [\![F(g^* Q)]\!]\, Q\big) \to \big((\Sigma\, i : B \to p(Q'))\, [\![F(i^* Q')]\!]\, Q'\big)$$

Post-composition with $p(h)$ induces a function $(p(h))\circ - : \mathbb{B}(B, p(Q)) \to \mathbb{B}(B, p(Q'))$ between the hom-sets which index these coproducts. By the induction hypothesis, we have a family of splitting morphisms $[\![F(g^*Q)]\!]\, h : [\![F(g^*Q)]\!]\, Q \to [\![F(g^*Q)]\!]\, Q'$ for $g : \mathbb{B}(B, p(Q))$.

From Lemma 6.4, we know that $Q = (p(h))^*Q'$ (since $h = (K(h))^{\S}$), hence

$$g^*Q = g^*((p(h))^*Q') = ((p(h)) \circ g)^*Q'$$

where the second equalities comes from $\mathbb{E}^{\mathsf{sp}}$ being split. Thus, we can rewrite the codomain of $[\![F(g^*Q)]\!]\, h$ obtaining

$$[\![F(g^*Q)]\!]\, h : [\![F(g^*Q)]\!]\, Q \to [\![F(((p(h)) \circ g)^*Q')]\!]\, Q'$$

This $\mathbb{B}(B, p(Q))$-indexed family of maps in $\mathbb{E}^{\mathsf{sp}}$ is exactly what is needed to form the generalised sum

$$\Sigma\big((p(h)) \circ -, [\![F(-^*Q)]\!]\, h\big)$$

which we take as definition of $[\![\delta\, B\, F]\!]\, h$. □

Remark 6.11. Inspecting the proof, we see that it would go through with any functor $S : \mathsf{Fam}\,(\mathbb{E}^{\mathsf{sp}}) \to \mathbb{E}^{\mathsf{sp}}$ instead of Σ, but we expect coproducts to most closely model the data types we are interested in. Note that if the subcategory $\mathbb{E}^{\mathsf{sp}}$ is closed under coproducts, then a generalized sum of splitting morphisms is automatically splitting.

We can also offer the following alternative proof, based on an "impredicative" decoding of a δ-code[1] similar to the decoding given in Lemma 4.4. We make use of the following lemma, which allows us to switch between global structure in the total category and local structure in the fibre.

Lemma 6.12 ([64], 1.4.10)**.** Let $p : \mathbb{E} \to \mathbb{B}$ be a cloven fibration. There is a natural isomorphism

$$\mathbb{E}(X, Y) \cong (\Sigma\, g : p\, X \to p\, Y)\, \mathbb{E}_{p(X)}(X, g^*(Y))$$

[1] The adjective impredicative comes from the reformulation of the interpretation of a δ-code based on the quantification over the a priori "large" collection $|\mathbb{E}_B|$.

If the fibration p is split, by Lemma 6.4 all fibres of p^{sp} are discrete and we can rephrase the isomorphism from Lemma 6.12 for $p^{\mathsf{sp}} \colon \mathbb{E}^{\mathsf{sp}} \to \mathbb{B}$ as

$$\mathbb{E}^{\mathsf{sp}}(X, Y) \cong (\Sigma\, g : p\, X \to p\, Y)\, (X \equiv_{\mathbb{E}_{p(X)}} g^* Y) \tag{6.1}$$

Using (6.1), we can reformulate the decoding of a δ code as:

$$\begin{aligned} [\![\delta_B F]\!]\, Q &= (\Sigma g \colon B \to p\, Q)\, [\![F(g^* Q)]\!]\, Q \\ &\cong (\Sigma\, X : |\mathbb{E}_B|)\, (\Sigma g : p(X) \to p(Q))\, (X \equiv_{\mathbb{E}_B} g^* Q) \times [\![F(X)]\!] Q \\ &\cong (\Sigma\, X : |\mathbb{E}_B|)\, \mathbb{E}^{\mathsf{sp}}(X, Q) \times [\![F(X)]\!] Q \end{aligned} \tag{6.2}$$

Here, we have used an identity type in (6.2) to encode a constraint in the style of Henry Ford (as already seen in equation (4.1)): 'choose any X you like as long as it is g^*Q'. The isomorphism (6.2) can also be used to recast the action of $[\![\delta\, B\, F]\!]$ on a (splitting) morphism $h : Q \to Q'$: It is a sum of pairs of actions, where we can go from $\mathbb{E}^{\mathsf{sp}}(X, Q)$ to $\mathbb{E}^{\mathsf{sp}}(X, Q')$ by composing with h, and from $[\![F(X)]\!]\, Q$ to $[\![F(X)]\!]\, Q'$ by the induction hypothesis. If h was an arbitrary morphism, there would be no guarantee that the composite would still be splitting.
We call fibered data type the initial algebra of $[\![\gamma]\!]$ for a fibered IR-code γ. We will show in Section 5.6 that this exists, under certain conditions on the fibration $p : \mathbb{E} \to \mathbb{B}$. But first, let us look at some examples of fibred inductive-recursive definitions.

6.5 Examples

In this section we present some examples of data type definable within the theory of Fibred IR. We will see how the theory of fibered induction-recursion covers all data types we have seen so far (with the exception of those data types defined by positive induction-recursion as seen in Chapter 5), and how it allows us to define more complex structures not previously included within the schemes in the previous chapters. We postpone to the end of this section (see Lemma 6.20 and Lemma 6.21) the necessary checking that all fibrations involved in these examples satisfy the conditions of Lemma 6.10.

We start with the families fibration (Example 6.6) because, when we specialise fibred induction-recursion to this fibration, we get a system of codes with exactly the same expressive power as Djbyer and Setzer's IR.

Theorem 6.13 (IR and the families fibration)**.** Let $\pi : \mathsf{Fam}|D| \to \mathsf{Set}$ the families fibration described in Example 6.6. The scheme for $\mathsf{IR}\,(\pi)$-codes defines a class of functor equivalent to the scheme defined by the coding scheme for $\mathsf{IR}(D, D)$-codes given in Definition 3.7.

Proof. Firstly, we start comparing the syntax of the $\mathsf{IR}(D, D)$-codes as defined in Section 3.2.1 and the syntax $\mathsf{IR}\,(\pi)$-codes. The main difference between the two coding schemas is in the ι constructors. Indeed, the ι-codes in Definition 3.7 produce families with exactly one index while our fibred IR-codes a priori are more general as they produce families with arbitrary indices. However a family $T : X \to \mathsf{Set}$ is isomorphic to the coproduct of the X-indexed set of families whose x'th family is given by $(\mathsf{N}_1, \lambda\_.\, T(x))$, that is $(X, T) \cong (\Sigma x : X)\,(1, \lambda\_.T\,u)$. Hence we can simulate a code $\iota\,(X, P) : \mathsf{IR}\,(\pi)$ within the scheme given in Definition 3.7 by a σ-code built on top of the ι-code, precisely we represent $\iota\,(X, P) : \mathsf{IR}\,(\pi)$ by the code $\sigma\, X(x \mapsto \iota\,(P(x))) : \mathsf{IR}(D, D)$. Our σ-codes are exactly the same, as our δ-codes: indeed, a function $F : |\mathbb{E}_A| \to \mathsf{IR}\,(p)$ is exactly, in the families fibration, a function $F : (A \to D) \to \mathsf{IR}\,(\pi)$, since the fibre above A in the fibration π is nothing but the set $(A \to D)$.
Next, we look at the semantics of IR-codes, it is clear that both sets of codes interpret ι as constant functors and σ as coproducts. As for the δ-constructor, these are seen to define the same functors once we realise that, in the families fibration, if we reindex a family $P : X \to \mathsf{Set}$ by a function $g : Y \to X$, we have $g^*(X, P) = (Y, P \circ g)$. □

In chapter 4 we saw that both simple and indexed inductive types can be obtained as specific sub-theories of IR. Here we want to show that they are instances of fibred induction-recursion: to this purpose we use containers and indexed containers.

Theorem 6.14. Containers are fibred inductive-recursive codes for the identity fibration $\mathsf{Id} : \mathsf{Set} \to \mathsf{Set}$ while indexed containers coincide with the class of fibred IR-codes for domain fibrations of the form $\mathsf{dom}_I : \mathsf{Set}/I \to \mathsf{Set}$.

Proof. Consider first containers: they are fibred inductive-recursive codes for the identity fibration $\mathsf{Id} : \mathsf{Set} \to \mathsf{Set}$ because in this fibration the fibre $\mathbb{E}_A$ is a singleton $\{A\}$ for each A, so that the dependency in the δ codes trivialises and thus every code can be reduced to a "container normal form" $\sigma\, S(s \mapsto \delta\, P(s)(\_ \mapsto \iota\, \mathsf{N_1})$.
An indexed container $(S, P, n) : \mathsf{IC}(I, I)$ can be represented by a fibred IR-code in $\mathsf{IR}(\mathsf{dom}_I)$ where dom_I is defined as in Example 6.5. We see again that indeed $\mathsf{N_1}$-indexed containers and ordinary containers coincide, since $\mathsf{dom}_{\mathsf{N_1}}$ is equivalent the identity functor on $\mathsf{Set}/\mathsf{N_1} \cong \mathsf{Set}$. □

We can now look back at the results in Chapter 4 from a fibrational perspective. The scheme defined there has been dubbed *small* where the "smallness" was referring to the type-theoretic size of the parameter type D. Here we can show that the "smallness" can also refer to a property of the underlying fibration. To this end recall the definition of a split generic object in a fibration:

Definition 6.15. Given a split fibration $p : \mathbb{E} \to \mathbb{B}$, we say that an object $\Omega \in \mathbb{B}$ is a split generic object if for any X in $\mathbb{B}$ there are isomorphisms

$$\theta_X : \mathbb{B}(X, \Omega) \cong |\mathbb{E}_X|$$

natural in X, i.e. $\theta_X(f \circ g) = g^*(\theta_Y(f))$ for $g : X \to Y$ and $f : Y \to \Omega$. ■

Generic objects can be used to characterize small fibrations as the following lemma shows.

Lemma 6.16 ([64], Corollary 9.5.6)**.** A fibration is small if and only if it has a generic object and it is locally small.

A domain fibration dom_I always yields a split generic object, namely the object I. In addition, domain fibrations are always locally small. These two features together make domain fibrations *small*. We can now appreciate from a different perspective why the scheme defined in Chapter 4 has been dubbed *small*: when describing inductive and indexed inductive types from the perspective of fibered IR, the fibrations involved are exactly small fibrations.

Example 6.17 (A universe of Setoids)**.** A setoid $(|A|, \simeq)$ in type theory is a type $|A|$ together with an equivalence relation $\simeq$ on $|A|$ (that is, a relation $\simeq$ together with proofs of reflexivity, transitivity and symmetry). Setoids are often used when developing mathematics in type theory [16], as they can simulate both quotient types and function extensionality. Naturally, we would like to consider universes also in this setting. By instantiating fibred induction-recursion in a fibration of families of setoids, we can get such universes without any more effort than if we were working with mere sets.

Following Palmgren [86], a family of setoids $B : A \to \mathsf{Setoid}$ consists of an index setoid $A = (|A|, \simeq)$ together with an $|A|$-indexed family of setoids $B(a) : \mathsf{Setoid}$, for $a : |A|$, such that if p is a proof that $x \simeq y$, then $B(x)$ and $B(y)$ are "the same", i.e. there is a "reindexing" bijection $\phi_p : B(x) \to B(y)$ (in a coherent way). Setoids naturally form a category Setoid, with a morphism $f : (|A|, \simeq_A) \to (|B|, \simeq_B)$ being a function $|f| : |A| \to |B|$ which respects the equivalence relations. We can form a category $\mathsf{Fam}(\mathsf{Setoid})$ of families of setoids which is fibred over Setoid in the same way $\mathsf{Fam}\,(\mathsf{Set})$ is fibred over Set. In particular, the objects in the fibre $\mathbb{E}_A$ are families of setoids $B : A \to \mathsf{Setoid}$ with index setoid A.

Using fibred induction-recursion instantiated to this fibration, we can now write down a code $\gamma_{B,\Sigma}$ for a universe of setoids containing codes for the setoids $B(a)$, for $a : |A|$, and closed under Σ-setoids. The code $\gamma_{B,\Sigma}$ is the following

$$\sigma\,|A|\,(a \mapsto \iota(\mathsf{1}, \lambda\_.B(a))) + \delta\,\mathsf{N_1}(A \mapsto \delta\,A(\mathsf{0_1})(B \mapsto \iota(\mathsf{1}, \lambda\_.\Sigma_{\mathsf{Setoid}}\,(A(\mathsf{0_1}))\,B)))$$

where $\mathsf{1}$ is the unit setoid with the obvious equivalence relation. For $A : \mathsf{Setoid}$ and $B : A \to \mathsf{Setoid}$, the sigma setoid is defined by $\Sigma_{\mathsf{Setoid}} A\,B = (\Sigma\,|A|\,|B|, \simeq_\Sigma)$ where $(x, y) \simeq_\Sigma (x', y')$ if $(\exists p{:}\, x \simeq_A x')(\phi_p(y) \simeq_{B_{x'}} y')$.

The code $\gamma_{B,\Sigma}$ describes a universe (U, T) which contains all the setoids $B(a)$ and is closed under sigma setoids; the underlying family of sets $(|\mathsf{U}|, |\mathsf{T}|)$ satisfies equations (3.1) but all operations now automatically preserve the equivalence relations. ■

Example 6.18 (The Category with Families Fibration)**.** Categories with families [61] were introduced by Dybjer as a variant of Cartmell's *categories*

with attributes: their aim is to model type theory from a closer perspective to its syntax. A category with families consists of a small category $\mathbb{C}$ and a functor $F = (\mathsf{Ty}, \mathsf{Te}) : \mathbb{C}^{\mathsf{op}} \to \mathsf{Fam}\,(\mathsf{Set})$ with some extra structure -a comprehension- modeling extension of a context by types in that context. We think of $\mathbb{C}$ as a category of contexts, and of $\mathsf{Te}(\Gamma) : \mathsf{Ty}(\Gamma) \to \mathsf{Set}$ as a family of terms, indexed by the type $\mathsf{Ty}(\Gamma)$, all in the context Γ.
The (large) category CwF of categories with families and structure's preserving morphisms between them forms a fibration with base category Cat, the category of all small categories. The fibre above $\mathbb{C}$ consists of the functors' category $[\mathbb{C}^{\mathsf{op}}, \mathsf{Fam}\,(\mathsf{Set})]$ with their extra structure and the reindexing of $(\mathbb{D}, G)$ along $f : \mathbb{C} \to \mathbb{D}$ is given by $(\mathbb{C}, G \circ f^{\mathsf{op}})$.
An important category with families for a given type theory is the one formed from the syntax of the theory itself, i.e. its *term model*. The construction thereof for dependent type theories is famously induction-recursion like in nature; types are indexed over contexts, but contexts can only be extended by well-formed types. McBride [75] implements the syntax of dependent type theory this way in Agda using induction-recursion, while Danielsson [33] and Chapman [25], using induction-induction (see Section 5.2.1).
An example similar to that of CwF is given by the presheaves fibration: here again the base category is given by Cat and reindexing of $X : \mathbb{C}^{\mathsf{op}} \to \mathsf{Set}$ along $f : \mathbb{C} \to \mathbb{D}$ is still given by precompostion with f^{op}. By using fibered induction-recursion in the presheaf fibration we can define universes that automatically come equipped with a notion of substitution: indeed, as shown by Fiore et al. [42], we can automatically take into account the substitution structure by turning indices and contexts into the structure of presheaves. ■

Example 6.19 (Fibred IIR)**.** Recall from Section 3.2.6 that within the framework of indexed induction-recursion we add another layer of indices by letting the parameter D to be indexed by a set I.
Now, let I be a set of indices for types $D(i)$. Dybjer and Setzer's scheme for indexed induction-recursion with same input and output parameter D, $\mathsf{IIR}(D, D)$, gives rise to endofunctors of type

$$(\Pi\, i :\, I)\, \mathsf{Fam}|D(i)| \longrightarrow (\Pi\, i :\, I)\, \mathsf{Fam}|D(i)|$$

The scheme for $\mathsf{IIR}(D, D)$-code arises as an example of fibred induction-

recursion for the fibration $\pi^I : (\Pi\, i :\, I)\, \mathsf{Fam}|D\ i| \to [I, \mathsf{Set}]$ where the base category consists of I-indexed set $X : I \to \mathsf{Set}$, and the fibre above X is the (discrete) category whose objects are pairs (X, T) for $T : (\Pi\, i : I)(X\, i \to D\, i)$. ■

Lemma 6.20. The families fibration (Example 6.13), the domain fibration (Example 6.5), the families of setoids fibration (Example 6.17) and the categories with families fibration (Example 6.18) are split, and the subcategories of splitting morphisms in the total category have coproducts.

Proof. All fibrations mentioned are instances of the situation in Proposition 6.8:

- The families fibration $\mathsf{Fam}\,(\mathbb{C}) \to \mathsf{Set}$ arises as $\mathbb{A} = \mathsf{Set}$, $F(X) =$ discrete category on X and $\mathbb{D} = \mathbb{C}$.
- The families of setoids fibration arises as $\mathbb{A} = \mathsf{Setoid}$, $F(X)$ the category with objects the elements in the carrier of X and morphisms proofs of relatedness and $\mathbb{D} = \mathsf{Setoid}$.
- The categories with families fibration arises as $\mathbb{A} = \mathsf{Cat}$, $F(X) = X^{\mathsf{op}}$ and $\mathbb{D} = \mathsf{Fam}\,(\mathsf{Set})$.
- The domain fibration $\mathsf{dom}_I : \mathsf{Set}/I \to \mathsf{Set}$ arises as $\mathbb{A} = \mathsf{Set}$, $F(X) =$ discrete category on X and $\mathbb{D} =$ discrete category on I. When I is a singleton the domain fibration is equivalent to the identity fibration.

In all cases $\mathbb{A}$ has coproducts and F preserves them. □

Lemma 6.21. The fibration $p : (\Pi\, i : I)\mathsf{Fam}|D_i| \to [I, \mathsf{Set}]$ from Example 6.19 is split and it has set-indexed coproduct.

Proof. The fibration $p : (\Pi\, i : I)\mathsf{Fam}|D_i| \to [I, \mathsf{Set}]$ can be regarded as the product of the I-indexed family of fibrations $\pi_i : \mathsf{Fam}|D_i| \to \mathsf{Set}$. It is split since the product of split fibrations is split. The fibration$p$ has also set-indexed coproducts given pointwise: if (X, T) and (Y, Q) are objects of $(\Pi\, i : I)\mathsf{Fam}|D_i|$ their coproduct, $(X + Y, [T, Q])$, is given by $(X + Y)(i) = X(i) + Y(i)$ and $[T, Q](i, z) = [T(i), Q(i)](z)$. □

6.6 Existence of initial algebras

We now want to show that fibered IR-functors indeed have initial algebras, under some conditions on the fibration $p : \mathbb{E} \to \mathbb{B}$. We do this by adapting the proof in Section 3.3 to the fibrational setting.
An important point for both their and our proof is keeping track of the "size" of the index objects appearing in the codes. In particular, the size of the index object B in a code $\delta\, B\, F$ may depend on the input object Q we pass to the functor, as it can be observed, for example, when defining a universe closed under certain type constructors (see Section 3.2.7). Therefore we follow Definition 3.34 and collect all the index objects in a class $\mathsf{Aux}(c, Q)$.

Definition 6.22. For a fibred IR-code γ and object Q in $\mathbb{E}$, define the collection $\mathsf{Aux}(\gamma, Q) \subseteq |\mathbb{B}|$ by induction over γ:

$$\begin{aligned}
\mathsf{Aux}((\iota\, P), Q) &= \emptyset \\
\mathsf{Aux}((\sigma\, A\, f), Q) &= \bigcup_{a \in A} \mathsf{Aux}(f a, Q) \\
\mathsf{Aux}((\delta\, A\, F), Q) &= \{A\} \cup \bigcup_{g\colon\, A \to p(Q)} \mathsf{Aux}(F(g^* Q), Q)
\end{aligned}$$

■

We now observe that if for certain Q all $A \in \mathsf{Aux}(c, Q)$ are "small" in a suitable sense then $[\![\gamma]\!]$ is κ-continuous, i.e. preserves κ-directed colimits.
In Section 3.3, to express that $A \in \mathsf{Aux}(c, Q)$ is small, we used $A \in \mathrm{V}_\kappa$, from which it follows $|A| < \kappa$. Here we replace this condition by the categorical notion of smallness given by κ-presentability. We recall its definition below.

Definition 6.23. An object A of $\mathbb{B}$ is κ-presentable if $\mathbb{B}(A, -) : \mathbb{B} \to \mathbf{Set}$ preserves κ-directed colimits. ■

Explicitly, the object A is κ-presentable if and only if for each colimit cocone $\bigvee_{i:I} Q_i$, where I a κ-directed poset, every map $f : A \to \bigvee_{i:I} Q_i$ factors uniquely as $f = \mathsf{in}_i \circ g$ for some $g : A \to Q_i$. When $\mathbb{B}$ is $\mathbf{Set}$, an object A is κ-presentable if and only if it has cardinality $|A| < \kappa$.
We can now reformulate Lemma 3.37 as follows:

Lemma 6.24. Let $\bigvee_{i:I} Q_i$ be a κ-directed colimit for a diagram $Q : I \to \mathbb{E}^{\mathsf{sp}}$ with all $A \in \mathsf{Aux}(\gamma, Q_i)$ κ-presentable. If $p^{\mathsf{sp}} : \mathbb{E}^{\mathsf{sp}} \to \mathbb{B}$ preserves $\bigvee_{i:I} Q_i$, then so does $[\![\gamma]\!] : \mathbb{E}^{\mathsf{sp}} \to \mathbb{E}^{\mathsf{sp}}$.

Proof. The proof of the lemma is entirely similar to the proof of Lemma 3.37. We give details for $\gamma = \delta\, A\, F$.

$$
\begin{aligned}
[\![\delta_A F]\!](\bigvee_{i:I} Q_i) &= (\Sigma\, f : A \to p(\bigvee_{i:I} Q_i))[\![F(f^* \bigvee_{i:I} Q_i)]\!](\bigvee_{i:I} Q_i) \\
&\overset{(1)}{\cong} (\Sigma\, g : A \to \bigvee_{i:I} p(Q_i))[\![F(([p(\mathsf{in}_i)]_{i:I} \circ g)^* \bigvee_{i:I} Q_i)]\!](\bigvee_{i:I} Q_i) \\
&\overset{(2)}{\cong} (\Sigma\, h : A \to p(Q_j))[\![F(([p(\mathsf{in}_i)]_{i:I} \circ in_j \circ h)^* \bigvee_{i:I} Q_i)]\!](\bigvee_{i:I} Q_i) \\
&\overset{(3)}{\cong} (\Sigma\, h : A \to p(Q_j))[\![F(((p(\mathsf{in}_j)) \circ h)^* \bigvee_{i:I} Q_i)]\!](\bigvee_{i:I} Q_i) \\
&\overset{(4)}{\cong} (\Sigma\, h : A \to p(Q_j))[\![F(h^* Q_j)]\!](\bigvee_{i:I} Q_i) \\
&\overset{\text{I.H.}}{\cong} \bigvee_j (\Sigma\, h : A \to p(Q_j)) \bigvee_{i:I} [\![F(h^* Q_j)]\!](Q_i) \\
&\overset{(5)}{\cong} \bigvee_{i:I} (\Sigma\, h : A \to p(Q_j))[\![F(h^* Q_j)]\!](Q_i) = \bigvee_{i:I} [\![\delta_A F]\!](Q_i)
\end{aligned}
$$

where (1) holds since p preserves directed colimits, and (2) since $A \in \mathsf{Aux}(c, Q)$ is κ-presentable, so that g factors through one of the Q_j. Equation (3) holds because again p preserves filters colimits and therefore $[p(\mathsf{in}_i)]_{i:I} \circ \mathsf{in}_j = K(in_j)$; equation (4) holds by Lemma 6.4. Finally, (5) holds since Σ is a left adjoint as noticed in Remark 3.10. □

In order to ensure that the first hypothesis of the previous lemma holds for the colimit we need, i.e. that all $A \in \mathsf{Aux}(c, Q_i)$ are κ-presentable, for κ the length of the initial sequence, we need the same meta theoretical assumption used in Section 3.3: we assume that there exists a Mahlo cardinal M. We also assume that all indexing sets in the coproducts used in the decoding have cardinality at most M. But before proving that we can actually find a κ satisfying the hypothesis of lemma 6.24 we need two other lemmas ensuring that we use just M-presentable objects as indices of our coproducts. The following two lemmas take care of this.

Lemma 6.25 (Adámek and Rosicky [11], 1.16)**.** A colimit of a λ-small diagram of λ-presentable objects is λ-presentable.

Lemma 6.26. Let $p : \mathbb{E} \to \mathbb{B}$ be a split fibration with coproducts in $\mathbb{E}^{\mathsf{sp}}$. Assume that p^{sp} preserves them, and that $p(P)$ is M-presentable for every P occurring in a ι code in γ. Then $p(\llbracket\gamma\rrbracket(Q))$ is M-presentable for all Q in $\mathbb{E}^{\mathsf{sp}}$.

Proof. If $\gamma = \iota\, P$, then $p(\llbracket\iota\ P\rrbracket(Q)) = p(P)$ is M-presentable by assumption. If $\gamma = \sigma\, A\, f$ or $\gamma = \delta\, A\, F$, then $p(\llbracket\gamma\rrbracket(Q))$ is a coproduct of objects which are M-presentable by the induction hypothesis, hence M-presentable by Lemma 6.25. □

We can now formulate the analogue to Lemma 3.38.

Lemma 6.27. Let $\gamma{:}\,\mathsf{IR}\,(p)$ and (Q_α) the initial sequence of the induced functor $\llbracket\gamma\rrbracket$. Assume that if Q_α is M-presentable, then all objects in $\mathsf{Aux}(\gamma, Q_\alpha)$ are M-presentable. Then there exists an inaccessible κ such that all $A \in \mathsf{Aux}(\gamma, Q_\alpha)$ are κ presentable for all $\alpha < \kappa$.

Proof. The proof goes as in that of Lemma 3.38: we define an increasing function $f : \mathrm{ON} \to \mathrm{ON}$, which tells us the smallest cardinal κ one needs to go up to still be κ-presentable after one iteration of $\llbracket\gamma\rrbracket$. We modify the definition of f accordingly. The important property of f will be

$$\begin{array}{l}\text{if } p(Q_{\beta'}) \text{ is } \beta\text{-presentable then}\\ \quad \text{all } A \in \{p(Q_{\beta'+1})\} \cup \mathsf{Aux}(\gamma, Q_{\beta'}) \text{ are } f(\beta)\text{-presentable}\end{array} \tag{6.3}$$

for all $\beta' < \mathsf{M}$. We define $f : \mathrm{ON} \to \mathrm{ON}$ by transfinite recursion as follows:

$$\begin{aligned} f(\beta) = \min\{\alpha \mid &(\forall \beta' < \beta)\big(f(\beta') < \alpha\big) \wedge \\ &(\forall \beta' < \mathsf{M})\big(p(Q_{\beta'}) \text{ is } \aleph_\beta\text{-presentable} \implies \\ &\text{all } A \in \{p(Q_{\beta'+1})\} \cup \mathsf{Aux}(c, Q_{\beta'}) \\ &\qquad\qquad \text{are } \aleph_\alpha\text{-presentable.}\big)\} \end{aligned}$$

The first conjunct makes sure that f is increasing, and the second makes (6.3) true.
We then proceed as in the proof of Lemma 3.38 by showing that M actually bounds f i.e. $f : \mathsf{M} \to \mathsf{M}$ and use the Mahlo property to find a fixed point κ of f. We can therefore obtain the following reformulation of (6.3)

$$\begin{array}{l}\text{if } p(Q_{\beta'}) \text{ is } \beta\text{-presentable then} \\ \quad \text{all } A \in \{p(Q_{\beta'+1})\} \cup \mathsf{Aux}(c, Q_{\beta'}) \text{ are } \kappa\text{-presentable}\end{array} \tag{6.3$'$}$$

for all $\beta < \kappa$ (since $f(\beta) < \kappa$).
Finally, we prove that $p(Q_\alpha)$ is κ-presentable for all $\alpha < \kappa$ by induction on α.

- If $\alpha = 0$, then $p(Q_0) = p(\bot) = \bot$ which is κ-presentable for all κ.
- If $\alpha = \beta + 1$, then Q_β is κ-presentable by the induction hypothesis, and we are done by (6.3$'$).
- If $\alpha = \lambda$ limit, then $Q_\lambda = \bigvee_{\beta<\lambda} Q_\beta$ is κ-presentable by the induction hypothesis and Lemma 6.25.

By (6.3$'$), it immediately follows that all $A \in \mathsf{Aux}(c, Q_\alpha)$ are κ-presentable. □

Theorem 6.28. Under the assumptions of Lemma 6.27, if $\mathbb{E}^{\mathsf{sp}}$ has κ-directed colimits for κ as in Lemma 6.27, and p^{sp} preserves them, then the functor $[\![\gamma]\!] \colon \mathbb{E}^{\mathsf{sp}} \to \mathbb{E}^{\mathsf{sp}}$ has an initial algebra.

Proof. Lemma 6.27 ensures that we can find an inaccessible κ such that the hypothesis of Lemma 6.24 are validated. Therefore, $[\![\gamma]\!]$ is κ-continuous and we can build the initial algebra of $[\![\gamma]\!]$ as the colimit of the initial sequence. □

Chapter 7

Conclusion

Abstract In this chapter we sum up the results obtained in this thesis and discuss possible lines of research for the future.

7.1 Summary

Induction-recursion represents an extremely powerful which sits on top of a hierarchy of more and more sophisticated inductive types.
In this thesis we have provided support for this claim and broadened our understanding of the theory of IR: i) we have shown how the theories of inductive and indexed inductive types can be seen as sub-theories of induction-recursion. This analysis have revealed the importance played by a notion of of size within the theory of IR. ii) We have expanded the expressive power of IR, showing how to extend the theory of induction-recursion in two orthogonal ways: in one direction we explored the changes needed to obtain a more expressive semantics which gives rise to a more comprehensive elimination principle for inductive-recursive types. In another direction we have revealed a fibrational structure of the theory which suggested a generalization of induction-recursion to a fibrational setting. In both cases we: (a) gave a finite axiomatization of the theories introduced; (b) brought evidence of novel applications of these theories not covered by the already expressive theory of induction-recursion, and finally (c) justified the existence of data types built within these theories.

7.2 Future work

We discuss some other topics which might set for a better understanding of the theory of induction-recursion.

Composition and normal forms

The theory of (indexed) inductive types benefits, among many others, of a very natural property, namely composition: if we have dependent polynomials $F : \mathsf{Poly}(I, M)$ and $G : \mathsf{Poly}(M, O)$ we can build a dependent polynomial $G \cdot F : \mathsf{Poly}(I, O)$ such that $\mathcal{P}_{G \cdot F} \cong \mathcal{P}_G \circ \mathcal{P}_F$. Thus, it is natural to ask if a similar property holds for general inductive-recursive types, i.e, given codes $\gamma : \mathsf{IR}(I, M)$ and $\gamma' : \mathsf{IR}(M, O)$ does there exist a code $\gamma'' : \mathsf{IR}(I, O)$ such that $[\![\gamma'']\!] \cong [\![\gamma']\!] \circ [\![\gamma]\!]$?
We gave a partial answer to this question in Chapter 4 by exploiting the closure property of dependent polynomials. But, the answer to the same question for general IR-code still remains open. This answer would shed light on a normal form for IR-codes. The question is so simple to state as the details to answer it are intricate. Some calculations seem to suggest that it is possible to give a positive answer for a subset of the codes. Another approach that one might consider is by modifying the syntax of IR. Conor McBride (private communication) have explored other systems of codes containing the IR-codes as a proper sub-system and for which proving composition is easier. However, this does not settle an answer to the above question since closure under composition of an host system does not imply that one of its subsystem shares this property.

An alternative approach to fibered induction-recursion

In the abstraction of IR to a fibrational setting given in Chapter 6 we chose as starting point the family fibration $\pi : \mathsf{Fam}|D| \to \mathsf{Set}$. This approach was naturally suggested by the axiomatization of the syntax and the semantics given in Chapter 3. However, the original semantics of IR given by Dybjer and Setzer in [39] interprets a code $\gamma : \mathsf{IR}(I, O)$ as a functor $[\![\gamma]\!] : \mathsf{Type}/I \to \mathsf{Type}/O$ where Type is the syntactic category arising from the judgements of the Log-

ical Framework. We have already discussed some drawbacks of moving into the larger category of types in Remark 3.16. Here we want to sketch how to combine these approaches to get the best of both worlds. Indeed, it is immediate to see that the embedding $\mathsf{Fam}|D| \hookrightarrow \mathsf{Type}/D$ define a full and faithful subcategory of Type/D of objects with a *small* domain. Moreover the host category of Type/D sits inside $\mathsf{Type}^{\to}$ as the fiber above D in the codomain fibration. In this setting we can get back the interplay between indexed objects in the fiber and indices in the base category as seen in Chapter 6 via the domain fibration. The interplay between the domain and the codomain fibration can then be used to abstract IR in the setting of comprehension categories (with unit) which are well studied models of dependent type theory. Another motivation to develop this framework is given by the possibility to compare it with that of *Algebraic Set Theory* [65] where the distinction between small and large objects, which seems central to IR, is taken as a fundamental one: indeed the large category Type more naturally matches a category of classes where the class of small maps can be singled out via the generic object given by the the map $\mathsf{dis}(\mathsf{El})\colon (\Sigma\, A : \mathsf{Set})\mathsf{El}(A) \xrightarrow{\pi_0} \mathsf{Set}$.

Final coalgebras

In this thesis we have been concerned with initial algebras for IR-functors and variants thereof. What about their dual, namely final coalgebras? We mentioned in Chapter 4 that for Small IR-functors these objects have already been investigated by Capretta under the name of *wander types* [23] and they capture previously known-constructions such as mixed induction-coinduction and continuous stream processors. A general theory of final coalgebras for IR-functors still needs to be developed.

Bibliography

[1] Michael Abbott. *Category of Containers*. PhD thesis, University of Leicester, 2003.

[2] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Representing nested inductive types using w-types. In *ICALP*, pages 59–71, 2004.

[3] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers: Constructing strictly positive types. *Theoretical Computer Science*, 342(1):3 – 27, 2005.

[4] Michael Abbott, Thorsten Altenkirch, Conor McBride, and Neil Ghani. ∂ for data: Differentiating data structures. *Fundam. Inform.*, 65(1-2):1–28, 2005.

[5] Andreas Abel, Ralph Matthes, and Tarmo Uustalu. Iteration and coiteration schemes for higher-order and nested datatypes. *Theoretical Computer Science*, 333(1-2):3–66, 2005.

[6] Peter Aczel. An introduction to inductive definitions. In Barwise Jon, editor, *Handbook of Mathematical Logic*. Elsevier, 1977.

[7] Peter Aczel. The type theoretic interpretation of constructive set theory: Inductive definitions. *Logic, Methodology and Philosophy of Science*, 7:17–49, 1986.

[8] Peter Aczel. *Non-well-founded-sets*, volume 14 of *CSLI Lecture Notes*. CSLI Pubblications, 1988.

[9] Peter Aczel. On relating type theories and set theories. In *TYPES*, pages 1–18, 1998.

[10] Jiri Adámek, Stefan Milius, and Lawrence Moss. Initial algebras and terminal coalgebras: a survey. Draft, June 29 2010.

[11] Jiri Adámek and Jiri Rosicky. *Locally Presentable and Accessible Categories*. Prentice-Hall, 1994.

[12] Thorsten Altenkirch, Paul Levy, and Sam Statton. Higher order containers. In *CiE*, 2010.

[13] Thorsten Altenkirch and Conor McBride. Towards observational type theory. Manuscript, available online, February 2006.

[14] Thorsten Altenkirch and Peter Morris. Indexed containers. In *LICS*, pages 277 –285, 2009.

[15] Steven Awodey, Nicola Gambino, and Kristina Sojakova. Inductive types in homotopy type theory. In *LICS*, pages 95–104, 2012.

[16] Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *Journal of Functional Programming*, 13(2):261–293, 2003.

[17] Marcin Benke, Peter Dybjer, and Patrik Jansson. Universes for generic programs and proofs in dependent type theory. *Nord. J. Comput.*, 10(4):265–289, 2003.

[18] Richard Bird and Oege de Moor. *Algebra of programming*. Prentice-Hall, 1997.

[19] Andre Boileau and André Joyal. La logique des topos. *The Journal of Symbolic Logic*, 46(1):6–16, 1981.

[20] George Boole. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Dover Pubblications, 1958.

[21] Ana Bove and Venanzio Capretta. Nested general recursion and partiality in type theory. In *TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 121–135, 2001.

[22] Ana Bove and Venanzio Capretta. Modelling general recursion in type theory. *Mathematical Structures in Computer Science*, 15(4):671–708, 2005.

[23] Venanzio Capretta. Wander types: A formalization of coinduction-recursion. *Progress in Informatics*, (10):47–64, 2013.

[24] Aurelio Carboni and Peter Johnstone. Connected limits, familial representability and Artin glueing. *Mathematical Structures in Computer Science*, 5(04):441–459, 1995.

[25] James Chapman. Type theory should eat itself. *Electronic Notes in Theoretical Computer Science*, 228:21–36, 2009.

[26] Adam Chlipala. Modular development of certified program verifiers with a proof assistant,. *J. Funct. Program.*, 18(5-6):599–647, 2008.

[27] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. In *TLCA*, pages 91–106, 2011.

[28] Thierry Coquand and Peter Dybjer. Inductive definitions and type theory: an introduction. In *FSTTCS*, pages 60–76, 1994. (Preliminary Version).

[29] Thierry Coquand and Christine Paulin. Inductively defined types. In *Conference on Computer Logic*, pages 50–66, 1988.

[30] Pier-Louis Curien, Richard Garner, and Martin Hofmann. Revisiting the categorical interpretation of dependent type theory. To appear, 2013.

[31] Pierre-Évariste Dagand and Conor McBride. A categorical treatment of ornaments. In *LICS*, pages 530–539, 2013.

[32] Pierre-Évariste Dagand and Conor McBride. Transporting functions across ornaments. *J. Funct. Program.*, 24(2-3):316–383, 2014.

[33] Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. *LNCS*, 4502:93–109, 2007.

[34] Peter Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In Huet Gerard and Gordon Plotkin, editors, *Logical Framework*, pages 280–306. Prentice Hall, 1991.

[35] Peter Dybjer. Inductive families. *Formal aspects of computing*, 6(4):440–465, 1994.

[36] Peter Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science*, 176(1-2):329–335, 1997.

[37] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2):525–549, 2000.

[38] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed lambda calculi and applications: 4th international conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999: proceedings*, pages 129–146. Springer Verlag, 1999.

[39] Peter Dybjer and Anton Setzer. Induction–recursion and initial algebras. *Annals of Pure and Applied Logic*, 124(1-3):1–47, 2003.

[40] Peter Dybjer and Anton Setzer. Indexed induction–recursion. *Journal of logic and algebraic programming*, 66(1):1–49, 2006.

[41] Samuel Eilenberg and Saunders Mac Lane. A general theory of natural equivilances. *Transaction of the American Mathematical Society*, 58(2):231–294, 1945.

[42] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proc. Logic in Computer Science*, pages 193–202, 1999.

[43] Fredrik Nordvall Forsberg and Anton Setzer. Inductive-inductive definitions. In *CSL*, pages 454–468, 2010.

[44] Clément Fumex. *Induction and conduction schemes in category theory.* PhD thesis, University of Strathclyde, 2012.

[45] Nicola Gambino. *Sheaf Interpretations for Generalised Predicative Intuitionistic Systems*. PhD thesis, University of Manchester, 2002.

[46] Nicola Gambino and Martin Hyland. Wellfounded trees and dependent polynomial functors. In *Types for Proofs and Programs*, volume 3085 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2004.

[47] Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 154, pages 153–192. Cambridge University Press, 2013.

[48] Richard Garner. On the strength of dependent products in the type theory of Martin-Löf. *Ann. Pure Appl. Logic*, 160(1):1–12, 2009.

[49] Neil Ghani, Makoto Hamana, Tarmo Uustalu, and Varmo Vene. Representing cyclic structures as nested types. Presented art Trends in Functional Programming, 2006.

[50] Neil Ghani and Peter Hancock. An algebraic implementation of induction-recursion and indexed induction-recursion. *Submitted to Mathematical Structures in Computer Science*, 2011.

[51] Neil Ghani, Patricia Johann, and Clément Fumex. Generic fibrational induction. *Logical Methods in Computer Science*, 8(2), 2012.

[52] Neil Ghani, Patricia Johann, and Clément Fumex. Indexed induction and coinduction, fibrationally. *Logical Methods in Computer Science*, 9(3), 2013.

[53] Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg. Positive inductive-recursive definitions. In Stefan Heckel, Reiko Milius, editor, *Algebra and Coalgebra in Computer Science, CALCO 2013*, volume 8089 of *Lectures Notes in Computer Science*, pages 19–33, 2013.

[54] Neil Ghani, Lorenzo Malatesta, Fredrik Nordvall Forsberg, and Anton Setzer. Fibred data types. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 243–252, 2013.

[55] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, 1977.

[56] Georges Gonthier. The four colour theorem: Engineering of a formal proof. In *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*, page 333, 2007.

[57] Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small induction recursion. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Application, TLCA 2013*, volume 7941 of *Lectures Notes in Computer Science*, pages 156–172, 2013.

[58] Ralf Hinze. Functional pearl: Perfect trees and bit-reversal permutation. *Journal of Functional Programming*, 10(3):305–317, 2000.

[59] Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *CSL*, pages 427–441, 1994.

[60] Martin Hofmann. *Extensional concept in intensional type theory.* PhD thesis, University of Edinburgh, 1995.

[61] Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79 – 130. Cambridge University Press, 1997.

[62] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In Giovanni Sambin and Jan M. Smith, editors, *Twenty-Five Years of Constructive Type Theory*, volume 36 of *Oxford Logic Guides*, pages 83–111, Oxford, 1998. Clarendon Press.

[63] Gérard P. Huet and Amokrane Saïbi. Constructive category theory. In *Proof, Language, and Interaction*, pages 239–276, 2000.

[64] Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. North Holland, Elsevier, 1999.

[65] André Joyal and Ieke Moerdijk. *Algebraic Set Theory*, volume 220 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1995.

[66] Akihiro Kanamori. *The Higher Infinity*. Springer Monographs in Mathematics. Springer Verlag, 2003.

[67] Kenneth Kunen. *Set Theory: An Introduction to Independent Proofs*. North Holland, 1980.

[68] Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103(2):151–161, 1968.

[69] Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[70] Maria Emilia Maietti. *The Type Theory of Categorical Universes*. PhD thesis, University of Padova, 1998.

[71] Maria Emilia Maietti. Modular correspondence between dependent type theories and categories including pretopoi and topoi. *Mathematical Structures in Computer Science*, 15(6):1089–1149, 2005.

[72] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216. North-Holland, 1971.

[73] Per Martin-Löf. *Intuitionistic type theory*. Bibliopolis, Naples, 1984.

[74] Per Martin-Löf. An intuitionistic theory of types. In Giovanni Sambin and Jan M. Smith, editors, *Twenty-Five Years of Constructive Type Theory*, volume 36 of *Oxford Logic Guides*, pages 127–172, Oxford, 1998. Clarendon Press. (Preprint from 1972).

[75] Conor McBride. Outrageous but meaningful coincidences: dependent type-safe syntax and evaluation. In *ICFP-WGP*, pages 1–12, 2010.

[76] Nax Paul Mendler. Predicative type universes and primitive recursion. In *LICS*, pages 173–184, 1991.

[77] Ieke Moerdijk and Erik Palmgren. Wellfounded trees in categories. *Ann. Pure Appl. Logic*, 104(1-3):189–218, 2000.

[78] Ieke Moerdijk and Erik Palmgren. Type theories, toposes and constructive set theory: predicative aspects of ast. *Annals of Pure and Applied Logic*, 114:155–201, 2002.

[79] Peter Morris, Thorsten Altenkirch, and Neil Ghani. Constructing strictly positive families. In *CATS*, pages 111–121, 2007.

[80] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's type theory: an introduction*. Clarendon Press, 1990.

[81] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, University of Swansea, 2013.

[82] Fredrik Nordvall Forsberg and Anton Setzer. A finite axiomatisation of inductive-inductive definitions. In *Logic, Construction, Computation*, volume 3, pages 259 – 287. 2012.

[83] Erik Palmgren. On universes in type theory. In Giovanni Sambin and Jan Smith, editors, *Twenty five years of constructive type theory*, pages 191 – 204. Oxford University Press, 1998.

[84] Erik Palmgren. Internalising modified realisability in constructive type theory. *Logical Methods in Computer Science*, 1:1–7, 2005.

[85] Erik Palmgren. Regular universes and formal spaces. *Ann. Pure Appl. Logic*, 137(1-3):299–316, 2006.

[86] Erik Palmgren. Proof-relevance of families of setoids and identity in type theory. *Archive for Mathematical Logic*, 51:35 – 47, 2012.

[87] Robert Seely. Locally cartesian closed categories and type theory. In *Mathematical proceedings of the Cambridge philosophical society*, volume 95, pages 33–48, 1984.

[88] Anton Setzer. Extending Martin-Löf type theory by one Mahlo-universe. *Archive for Mathematical Logic*, 39(3):155–181, 2000.

[89] Jan M. Smith. The independence of Peano's fourth axiom from Martin-Löf type theory without universes. *The Journal of Symbolic Logic*, 53(3):840–845, 1988.

[90] Michael B. Smyth and Gordon D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Comput.*, 11(4):761–783, 1982.

[91] Thomas Streicher. Universes in toposes. In Laura Crosilla and Peter Schuster, editors, *From Sets and Types to Topology and Analysis*. Oxford University Press, 2005.

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

[93] Ernst Zermelo. On boundary numbers and domains of sets: New investigation in the foundation of set theory. In Ewald William, editor, *From Kant to Hilbert: A Source Book in The Foundation of Mathematics*, volume 2, pages 1219 – 33. Oxford University Press, 1996.

Acknowledgments

This thesis (or the idea of it) has accompanied me for the last three years. It has been the obsession of this journey called PhD.
In Scotland I met several people who shared with me part of this trip. They all contributed to this thesis at different levels. I am grateful to those who were always by my side throughout the uncertainties that traveling to new lands always brings.
I have to thanks my advisor Neil Ghani. He suggested me the topic of the thesis, and showed me some good math. His expertise in academic life helped me to survive this PhD.
I have greatly benefit as a researcher and as a human being from my (retired) second supervisor Peter Hancock. I am grateful for his guidance in the research. He shared with me not only some good science but also some good philosophy.
I would like to thanks my examiners Nicola Gambino and Ross Duncan for their feedback on this thesis and the interest they showed when I discussed this thesis during the viva.
I have further benefited from discussion about the topics of this thesis with several people: Thorsten Altenkirch, Marcelo Fiore, Conor McBride, Anton Setzer and Thomas Streicher.
I am tremendously grateful to Fredrik Nordvall Forsberg: he supported me as a friend, as a colleague and as a (unofficial) supervisor. I learnt the art of doing research mostly by sharing ideas with him: his Scandivian enthusiasm and the gentle skepticism he uses to consider new ideas have helped me to keep my feet on the ground and produce actual research. Grazie!
The Mathematically Structured Programming group in Strathclyde has been my second house for (at least) a couple of years. I would like to thanks all

of its members for the math I have been exposed, the recursion on the tale, the beers and the friendship. I am grateful to Bob Atkey, Pierre-Evariste Dagand, Adam Gundry, and Clemens Kupke. I am especially grateful to Clèment Fumex, Stevan Andjelkovic Guillame Allais, Stuart Hannah and Christopher Gibbson for their friendship.

Luigia and Pietro patiently waited for me during the drafting process and always supported me during this trip. This thesis would not exists without them.