Bootstrapping Extensionality

University of Nottingham

Filippo Sestini

August 7, 2023

Abstract

Intuitionistic type theory is a formal system designed by Per Martin-Löf to be a full-fledged foundation in which to develop constructive mathematics. One particular variant, *intensional type theory* (ITT), features nice computational properties like decidable type-checking, making it especially suitable for computer implementation. However, as traditionally defined, ITT lacks many vital extensionality principles, such as function extensionality. We would like to extend ITT with the desired extensionality principles while retaining its convenient computational behaviour. To do so, we must first understand the extent of its expressive power, from its strengths to its limitations.

The contents of this thesis are an investigation into intensional type theory, and in particular into its power to express extensional concepts. We begin, in the first part, by developing an extension to the strict setoid model of type theory with a universe of setoids. The model construction is carried out in a minimal intensional type theoretic metatheory, thus providing a way to *bootstrap extensionality* by "compiling" it down to a few building blocks such as inductive families and proof-irrelevance.

In the second part of the thesis we explore inductive-inductive types (ITTs) and their relation to simpler forms of induction in an intensional setting. We develop a general method to reduce a subclass of infinitary IITs to inductive families, via an encoding that can be expressed in ITT without any extensionality besides proof-irrelevance. Our results contribute to further understand IITs and the expressive power of intensional type theory, and can be of practical use when formalizing mathematics in proof assistants that do not natively support induction-induction.

Acknowledgements

I would like to extend my sincere thanks to the many people that have contributed to make my PhD studies and my stay in Nottingham the pleasant experience that it has been.

I am first of all very grateful to my supervisor, Thorsten Altenkirch, for giving me freedom to explore my research interests while always being there to give helpful advice and discuss new ideas. I am also thankful to the many people in the type theory community that have helped me grow, and especially Ambrus Kaposi, with whom I have collaborated on a paper and had many insightful conversations.

I would like to express my sincere gratitude to all members of the Functional Programming Lab, particularly the past and present postdocs and PhD students, for making it an enjoyable and friendly environment where to learn a lot and have fun.

Finally, special thanks to the Nottingham Japanese Society, for being a source of many good friends and a weekly distraction from the inevitable stress of research.

Contents

Part I

Introduction

Chapter 1

Overview

Intuitionistic type theory is a formal system designed by Per Martin-Löf to be a full-fledged foundation in which to develop constructive mathematics [Mar75, Mar84]. A central aspect of type theory is the coexistence of two notions of equality. On the one hand definitional equality, the computational equality that is built into the formalism. On the other hand "propositional" equality, the internal notion of equality that is actually used to state and prove equational theorems within the system. The precise balance between these two notions is at the center of type theory research; however, it is generally understood that to properly support formalization of mathematics, one should aim for a notion of propositional equality that is as *extensional* as possible.

Two extensionality principles seem particularly desirable, since they arguably constitute the bare minimum for type theory to be comparable to set theory as a foundational system for set-level mathematics, in terms of power and ergonomics. One is function extensionality (or *funext*), according to which functions are equal if point-wise equal. Another is propositional extensionality (or *propext*), that equates all propositions that are logically equivalent.

Type theory with equality reflection, also known as *extensional type theory* (ETT) does support extensional reasoning to some degree, but unfortunately equality reflection makes the problem of type-checking ETT terms computationally unfeasible: it is undecidable.

On the other hand, *intensional type theory* (ITT) has nice computational properties like decidable type checking that can make it more suitable for computer implementation, but as usually defined (for example, in [Mar75]) it severely lacks extensionality. It is known from model constructions that extensional principles like *funext* are consistent with ITT. Moreover, ITT extended with the principle of *uniqueness of identity proofs* (UIP) and *funext* is known to be as powerful as ETT [Hof96]. We could recover the expressive power of ETT by adding these principles to ITT as axioms, however destroying some computational properties like canonicity.

What we would like instead is a formulation of ITT that supports extensionality, while retaining its convenient computational behaviour. Unfortunately, this vision does not seem attainable if we want our theory to enjoy canonicity while also implementing equality as the standard Martin-Löf's identity type inductively defined from reflexivity: in that setting, canonicity implies that if two terms are propositionally equal in the empty context, then they are also definitionally equal. This is incompatible with function extensionality, since that would allow, for example, to equate the otherwise intensionally distinct functions $\lambda x \,.\, x+0$ and $\lambda x \,.\, 0+x$. One first step towards a solution is to give up the idea of propositional equality as a single inductive definition given generically for arbitrary types. Instead, equality should be *specific* to each type former in the type theory, or in other words, every type former should be introduced alongside an explanation of what counts as equality for its elements.

This idea of pairing types together with their own equality relation goes back to the notion of *setoid* or *Bishop set*. Setoids provide a quite natural and useful semantic domain in which to interpret type theory. An early version of the setoid model was proposed by Hofmann to justify function extensionality without relying on *funext* in the metatheory [Hof95], however the model posed issues when attempting to introduce universes and large elimination.

An alternative variant of the setoid model was published by Altenkirch in [Alt99]. The construction is carried out in a type-theoretic metatheory with a universe of definitionally proof-irrelevant propositions, which results in a model that validates all the necessary equations *strictly*[1], and that supports universes/large elimination. This model, which we refer to as *the strict setoid model*, was later shown in [ABKT19] to validate a universe of strict, definitionally-proof irrelevant propositions internalizing the metatheoretic one.

The strict setoid model satisfies all the extensionality principles that we would like to have in a set-level type theory[2], but it is not a syntactic theory and therefore inadequate for mathematical reasoning as-is. The question is thus whether we can formulate a version of intensional type theory inspired by this model, that supports setoid reasoning and consequently the forms of extensionality enabled by it.

This question was revisited and answered in Altenkirch et al. [ABKT19]. In this paper, the authors define Setoid Type Theory (SeTT), an extension of intensional Martin-Löf type theory with constructs for setoid reasoning, where *funext* and *propext* hold by definition. SeTT is based on the strict setoid model, the *strictness* of which makes it possible to show consistency via a syntactic translation. This is in contrast with other type theories based on the setoid model, like Observational

[1]A *strict* model is one where every equation holds definitionally.

[2]In the sense of HoTT we mean a type theory limited to h-sets.

Type Theory (OTT) [AMS07], which instead rely on ETT for their justification. A major property of SeTT is thus to illustrate how to bootstrap extensionality, by translation into a minimal intensional core. In the same lineage of SeTT is XTT [SAG19], an version of intensional MLTT presenting a cubical reconstruction of OTT. With similar goals and scope, XTT provides an alternative to SeTT as a syntactic theory of setoids, although with quite different syntactic style and semantic justification.

SeTT as defined in [ABKT19] is already a rich theory, but its universes are limited to propositions. We would like to extend the theory with a universe of setoids, so as to properly internalise the notion of type. This goal brings up several questions, one of which has to do with the notion of equality with which the universe should come equipped: the universe of setoids is itself a setoid (since any type is) so it certainly cannot be univalent, as setoids lack the necessary structure. Another issue is the way such universe can be justified by the setoid model, and in particular what principles are needed in the metatheory to do so. The appeal of the original strict setoid model [Alt99] is that it explains extensionality by reducing it to a core intensional theory, so it seems desirable to implement any extension to it in a way that would preserve this property.

We thus have this tension between wanting our metatheory to be expressive, while at the same time keeping the set of core primitives as minimal as possible. A way to achieve this is to *encode* seemingly complex constructs in terms of simpler ones. A typical example is the reduction of inductive families to (indexed) W-types, a corollary of which is that merely extending a type theory with a single type constructor (that is, W-types) equips it with the full expressive power of arbitrary inductive definitions [Dyb97, Hug21].

These encoding/reduction methods are often of interest in the metatheoretic study of type theory. One reason is that they further our understanding of the concept being reduced, by explaining it in terms of well-understood building blocks. Moreover, they strengthen metatheorems, by showing that some assumptions can either be simplified or eliminated.

There are also practical reasons, like simplified exposition: these reductions allow us to replace a type theory with another that is equivalent in expressive power, but syntactically simpler. This is certainly desirable in metatheoretic studies, where type theories are the objects of our statements and proofs. For example, when studying the metatheory of type theory with inductive types, it is much more convenient to just assume the presence of W types rather than to work with general definition schemas.

Another practical benefit of these encodings is that they inspire ways to extend the capabilities of existing type-theoretic proof-assistants beyond their original design. Inductive-inductive types (IITs) [NF13] are a particularly evident example, since several well-known and widely-used proof assistants like Coq or Lean do not directly support them. Some forms of induction-induction are known to be reducible to plain inductive families [KKL20,vR19], which means that these proof-assistants could be made to support them (for example, via metaprogramming) without the need to change the core foundational theory [vR22].

The theme of *reducing* and *explaining* complex structures in terms of simpler building blocks is central to this thesis's work. In a modest attempt to pursue this general idea, we contribute a few novel results targeting two distinct but related topics.

We begin by presenting work towards equipping SeTT with a universe of setoids. Specifically, we show how to extend the strict setoid model with semantics for such universe. In line with the design choices of previous formulations of the model [Alt99,ABKT19], we try to keep the metatheory as minimal and intensional as possible. Accordingly, we construct the setoid universe in multiple stages, first by relying on complex forms of induction-recursion and infinitary induction-induction which are gradually reduced to simpler forms of induction. The final result is a setoid universe that can be encoded via plain inductive families, in a modest extension of intensional type theory. The universe is shown to validate *judgmental closure* under type formers, in addition to universe induction/typecase principle with propositional β-laws. We accompany our work with a complete Agda formalization.

The aforementioned universe construction process goes through the reduction of an infinitary inductive-inductive type to its encoding as plain inductive families. Despite the relative complexity of the "source" IIT we were able to carry out our encoding with only modest extensions to the metatheory, which notably does *not* include function-extensionality nor propositional extensionality, let alone equality reflection. A question that naturally arises is thus whether the same encoding can be applied to other infinitary IITs in addition to the one used for the setoid universe. Our second major contribution is work towards answering this question. We present and formalize a general method to reduce a subclass of infinitary IITs to inductive families, and encode them in an intensional type theory without *funext.* The construction draws from a known method to reduce *finitary* IITs to inductive families [vR19], adapting it to apply to the *infinitary* case.

Chapter 2

Background

2.1 Type Theory

Intuitionistic Type Theory is a formal system designed by Per Martin-Löf to provide a precise and rigorous framework for developing constructive mathematics [Mar75, Mar84]. The main constructions in type theory are *types* and *elements* of types. Its inference rules specify how to derive *judgments*, which are used to assert that something is a type, a term of a certain type, and that two types or terms are equal *by definition*. A fundamental aspect of type theory is that types constitute the primitive structure, on top of which logic is derived. The implementation of logic in type theory is based on the propositions-as-types principle, where propositions are identified with (certain) types, and proofs of a proposition with elements of the representing type.

Traditionally, type theory has often been presented as a form of typed λ-calculus, usually in a set-theoretic metatheory. These systems are defined by a set of grammar rules for untyped terms giving the "raw" syntax, and a set of "typing" inference rules that assign typing to the raw terms. For example, the simply-typed λ-calculus can be defined as the following set of untyped terms Tm and typing rules:

$$\mathsf{Tm} \ni t, s ::= x \mid \lambda x.t \mid t\ s$$

$$\frac{}{\Gamma, x : A, \Delta \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash t\ s : B}$$

This style of presentation is also called *extrinsic*, since the notion of type is applied to the syntax *post-facto*. We will instead present type theory in an *intrinsic style*, where the syntactic terms are defined together with their typing. This approach is equivalent to defining type theory as a finitely-presented generalized algebraic theory [Car86]. We have *sorts* for contexts, types, and terms, as well as corresponding judgments about objects of these sorts:

- A sort Con of contexts; the judgment $\Gamma : \mathsf{Con}$ states that Γ is a context;
- A sort Sub of substitutions, for any contexts $\Gamma, \Delta : \mathsf{Con}$; the judgment $\sigma : \mathsf{Sub}\ \Gamma\ \Delta$ states that σ is a substitution between contexts Γ, Δ;
- A sort $\mathsf{Ty}\ \Gamma$ of types, for any context $\Gamma : \mathsf{Con}$; the judgment $A : \mathsf{Ty}\ \Gamma$ states that A is a type in context Γ;
- A sort $\mathsf{Tm}\ \Gamma\ A$ of terms, for any context $\Gamma : \mathsf{Con}$ and type $A : \mathsf{Ty}\ \Gamma$; the judgment $t : \mathsf{Tm}\ \Gamma\ A$ states that t is a term in context Γ, of type A;
- Given types A, B of equal sort, or terms t, s of equal sort, the judgments $A = B$ and $t = s$ state definitional equality between the types and terms respectively.

Remark 2.1.1. If the specification of sorts and judgments shown above looks like the specification of an inductive type, that is because it precisely is one [AK16]. More specifically, the syntax of type theory can be alternatively presented as the signature of a Quotient Inductive-Inductive Type (QIIT), with the sorts $\mathsf{Con}, \mathsf{Ty}, \mathsf{Tm}, \mathsf{Sub}$ given as type constructors of the QIIT, and the definitional equalities given as path constructors. ■

In addition to these forms of judgment, we also have inference rules specifying how to construct evidence for these judgments. For example, we have context formation rules explaining how to derive valid contexts, as well as substitution rules governing how substitutions are formed, applied, and computed. In addition, for every type former T to be included in the theory, we have

- Formation rules, explaining how to show that T is a well-formed type;
- Introduction rules, explaining how to *construct* well-typed elements of T called *canonical* forms;
- Elimination rules, explaining how to *use* elements of T to define new expressions of other types with *elimination* forms;
- Equations, stipulating when a certain configuration of terms should be considered definitionally equal to another.

We now give some examples of rules for each of the categories mentioned above.

Rules for context formation Well-formed contexts are either the empty context or a context obtained by extension.

$$\frac{}{\bullet : \mathsf{Con}} \qquad \frac{\Gamma : \mathsf{Con} \qquad A : \mathsf{Ty}\ \Gamma}{\Gamma \triangleright A : \mathsf{Con}}$$

Rules for substitutions Substitution rules both specify how substitutions are built, as well as how they interact with the terms of the type theory. This treatment of substitutions is close to what is known as *explicit substitutions* [ACCL89] in the context of λ-calculi.

We have rules for applying substitutions to types and terms:

$$\frac{A : \mathsf{Ty}\ \Gamma \qquad \sigma : \mathsf{Sub}\ \Delta\ \Gamma}{A[\sigma] : \mathsf{Ty}\ \Delta} \qquad \frac{t : \mathsf{Tm}\ \Gamma\ A \qquad \sigma : \mathsf{Sub}\ \Delta\ \Gamma}{t[\sigma] : \mathsf{Tm}\ \Delta\ (A[\sigma])}$$

Then we have introduction rules for substitutions.

$$\frac{\Gamma : \mathsf{Con}}{\mathsf{id} : \mathsf{Sub}\ \Gamma\ \Gamma} \qquad \frac{\Gamma : \mathsf{Con}}{\epsilon : \mathsf{Sub}\ \Gamma\ \bullet} \qquad \frac{\sigma : \mathsf{Sub}\ \Gamma\ \Delta \qquad t : \mathsf{Tm}\ \Delta\ A[\sigma]}{\sigma, t : \mathsf{Sub}\ \Gamma\ (\Delta \triangleright A)}$$

$$\frac{\sigma : \mathsf{Sub}\ \Gamma\ \Delta \qquad \tau : \mathsf{Sub}\ \Omega\ \Gamma}{\tau \circ \sigma : \mathsf{Sub}\ \Omega\ \Delta} \qquad \frac{\sigma : \mathsf{Sub}\ \Gamma\ (\Delta, A)}{\pi_1\ \sigma : \mathsf{Sub}\ \Gamma\ \Delta} \qquad \frac{\sigma : \mathsf{Sub}\ \Gamma\ (\Delta, A)}{\pi_2\ \sigma : \mathsf{Tm}\ \Gamma\ (A[\pi_1\ \sigma])}$$

We employ a nameless syntax where explicit variable names are replaced by De Bruijn indices [de 72], built up from the free variable vz and weakening wk:

$$\begin{aligned} &\mathsf{wk} : \mathsf{Sub}\ (\Gamma, A)\ \Gamma && \mathsf{vz} : \mathsf{Tm}\ (\Gamma \triangleright A)\ (A[\mathsf{wk}]) \\ &\mathsf{wk} :\equiv \pi_1\ \mathsf{id} && \mathsf{vz} :\equiv \pi_2\ \mathsf{id} \end{aligned}$$

Thus for instance we can index the free variable of type A in a context $\Gamma \triangleright A \triangleright B \triangleright C$ as:

$$\mathsf{vz}[\mathsf{wk} \circ \mathsf{wk}] : \mathsf{Tm}\ (\Gamma \triangleright A \triangleright B \triangleright C)\ (A[\mathsf{wk} \circ \mathsf{wk} \circ \mathsf{wk}])$$

We can and will occasionally write explicitly named variables as a matter of convenience, taking it as syntactic sugar for the equivalent nameless notation. We may also drop weakening substitutions if they can be inferred from context. As an illustrative example, we can compactly rewrite the judgment above as:

$$a : \mathsf{Tm}\ (\Gamma \triangleright a : A \triangleright b : B \triangleright c : C)\ A$$

We have equality judgments explaining how to compute with substitutions

$$\begin{aligned} &\mathsf{id} \circ \sigma = \sigma && (\sigma \circ \tau) \circ \phi = \sigma \circ (\tau \circ \phi) \\ &\sigma \circ \mathsf{id} = \sigma && (\delta, t) \circ \sigma = (\delta \circ \sigma), t[\sigma] \\ &\pi_1\ (\delta, t) = \delta && (\pi_1\ \delta, \pi_2\ \delta) = \delta \\ &\sigma = \epsilon\ (\text{for } \sigma : \mathsf{Sub}\ \Gamma\ \bullet) \end{aligned}$$

and how substitutions interact with types and terms

$$\begin{array}{ll} A[\mathsf{id}] = A & A[\delta][\sigma] = A[\delta \circ \sigma] \\ t[\mathsf{id}] = t & t[\delta][\sigma] = t[\delta \circ \sigma] \\ \pi_2\ (\delta, a) = a & \end{array}$$

The components given so far constitute a standard presentation of a *substitution calculus* for type theory. Its structure directly follows from our understanding of models of type theory as certain kinds of categories (such as CwFs). In those settings we require, for example, contexts to form a category with a terminal object, types Ty to be a presheaf on contexts, and terms Tm to be a presheaf on $\int \mathsf{Ty}$. A simple unfolding of these constraints gives rise to the constructors and equations shown above.

Rules for type formers For each type former that we want to include in the theory, we provide rules of formation, introduction, and elimination, in addition to computation equations.

As an example, we now give rules for Π-types and Σ-types. Π-types, or dependent function types, are a dependently typed generalization of function types, in which the type of the codomain can depend on the *value* of the argument to the function.

$$\frac{A : \mathsf{Ty}\ \Gamma \qquad B : \mathsf{Ty}\ (\Gamma \rhd A)}{\Pi\ A\ B : \mathsf{Ty}\ \Gamma} \qquad \frac{t : \mathsf{Tm}\ (\Gamma \rhd A)\ B}{\lambda\ t : \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B)} \qquad \frac{t : \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B)}{\mathsf{app}\ t : \mathsf{Tm}\ (\Gamma \rhd A)\ B}$$

$$\begin{array}{ll} \mathsf{app}\ (\lambda\ t) = t & (\beta) \\ \lambda\ (\mathsf{app}\ t) = t & (\eta) \end{array}$$

Note that we define application as the inverse of λ-abstraction. We can derive the more familiar application operator from the current one:

$$\frac{t : \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B) \qquad u : \mathsf{Tm}\ \Gamma\ A}{t \cdot u :\equiv (\mathsf{app}\ t)[\mathsf{id}, u] : \mathsf{Tm}\ \Gamma\ (B[\mathsf{id}, u])}$$

Σ-types, or dependent pairs, are a dependently typed generalization of the cartesian product, where the type of the second component of the pair can depend on the *value* of the first component of the pair. We overload the projection symbols π_1, π_2.

$$\frac{A : \mathsf{Ty}\,\Gamma \qquad B : \mathsf{Ty}\,(\Gamma \triangleright A)}{\Sigma\ A\ B : \mathsf{Ty}\,\Gamma} \qquad \frac{a : \mathsf{Tm}\,\Gamma\ A \qquad b : \mathsf{Tm}\,\Gamma\ B[\mathsf{id}, a]}{\langle a, b\rangle : \mathsf{Tm}\,\Gamma\ (\Sigma\ A\ B)}$$

$$\frac{p : \mathsf{Tm}\,\Gamma\ (\Sigma\ A\ B)}{\pi_1\ p : \mathsf{Tm}\,\Gamma\ A} \qquad \frac{p : \mathsf{Tm}\,\Gamma\ (\Sigma\ A\ B)}{\pi_2\ p : \mathsf{Tm}\,\Gamma\ B[\mathsf{Id}, \pi_1\ p]}$$

$$\begin{aligned}
&\pi_1\,\langle a, b\rangle = a && (\beta_1)\\
&\pi_2\,\langle a, b\rangle = b && (\beta_2)\\
&\langle \pi_1\,p, \pi_2\,p\rangle = p && (\eta)
\end{aligned}$$

In addition to these rules, we also have rules explaining how substitutions interact with the new type and term formers. For example:

$$\begin{aligned}
&(\Pi\ A\ B)[\sigma] = \Pi\ (A[\sigma])(B[\sigma \uparrow A]) && (\Sigma\ A\ B)[\sigma] = \Sigma\ (A[\sigma])(B[\sigma \uparrow A])\\
&\langle a, b\rangle[\sigma] = \langle p[\sigma], q[\sigma]\rangle && (\mathsf{lam}\ t)[\sigma] = \mathsf{lam}\ (t[\sigma \uparrow A])
\end{aligned}$$

where $\sigma \uparrow A = (\sigma \circ \pi_1\ \mathsf{id}), \pi_2\ \mathsf{id}$.

Rules for the universe With the rules seen so far we can construct types and their elements, but we cannot formulate and prove statements *about* types themselves within the system. We thus introduce a *universe* $\mathcal{U}$ of types, whose elements are *codes* standing for types of the theory. We turn codes into the corresponding types by an operation $\mathcal{E}l$, together with a rule stating that if $A : \mathcal{U}$ is a well-formed code for a type, then $\mathcal{E}l\,A$ is a well-formed type. Types in the universe are also called *small* types, to distinguish them from *large* types like $\mathcal{U}$ itself which must not be contained in $\mathcal{U}$ to avoid paradoxes.

$$\frac{\Gamma : \mathsf{Con}}{\mathcal{U} : \mathsf{Ty}\,\Gamma} \qquad \frac{A : \mathsf{Tm}\,\Gamma\,\mathcal{U}}{\mathcal{E}l\,A : \mathsf{Ty}\,\Gamma}$$

This presentation of universes is called *á la Tarski.* A universe *á la Russell* is one where universe codes and the corresponding types are syntactically identified.

We usually consider universes that are closed under the same type-forming operations of the Ty judgment. For example, to close $\mathcal{U}$ under Π-types we can add a π constructor with a rule

$$\frac{A : \mathsf{Tm}\,\Gamma\,\mathcal{U} \qquad B : \mathsf{Tm}\,(\Gamma \triangleright \mathcal{E}l\,A)\,\mathcal{U}}{\pi\ A\ B : \mathsf{Tm}\,\Gamma\,\mathcal{U}}$$

We then usually require that $\mathcal{E}l$ commutes with the constructors of $\mathcal{U}$. For example, in the case of Π-types:

$$\mathcal{E}l(\pi\ A\ B) = \Pi\ (\mathcal{E}l\ A)\ (\mathcal{E}l\ B)$$

This is sometimes called *judgmental closure* under type formers.

2.1.1 Identity types

A crucial aspect of Martin-Löf Type Theory is the *identity type*, the elements of which are proofs that two terms of a given type are equal.

$$\frac{A : \mathsf{Ty}\,\Gamma \quad a : \mathsf{Tm}\,\Gamma\,A \quad b : \mathsf{Tm}\,\Gamma\,A}{\mathsf{Id}_A\,a\,b : \mathsf{Ty}\,\Gamma} \qquad \frac{a : \mathsf{Tm}\,\Gamma\,A}{\mathsf{refl}\,a : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}_A\,a\,a)}$$

Given terms a, b of some type A, one can form a new type $\mathsf{Id}_A\,a\,b$ of *proofs* that a and b are equal. The only constructor is $\mathsf{refl}\,a : \mathsf{Id}_A\,a\,a$, witnessing reflexivity.

The J eliminator gives the induction principle of identity types, stating that to define a term depending on a proof of equality, it is sufficient to consider the reflexivity case.

$$\frac{\begin{array}{c} M : \mathsf{Ty}\,(\Gamma \triangleright x : A \triangleright p : \mathsf{Id}_A\,a\,x) \\ m : \mathsf{Tm}\,\Gamma\,M[a, \mathsf{refl}\,a] \\ a : \mathsf{Tm}\,\Gamma\,A \qquad b : \mathsf{Tm}\,\Gamma\,A \qquad p : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}_A\,a\,b) \end{array}}{\mathsf{J}_M\,m\,p : \mathsf{Tm}\,\Gamma\,M[\mathsf{id}, b, p]}$$

There are different, essentially equivalent ways to formulate the J eliminator. The version shown above is due to Paulin-Mohring [PM93], and fixes one of the two endpoints of the identity proof as a parameter, while leaving the other free as an index. Another version has both endpoints ranging as indices.

The computation rule expresses that J applied to refl is exactly what we have specified it to be for the reflexivity case.

$$\mathsf{J}_M\,m\,(\mathsf{refl}\,a) = m$$

Some formulations of type theory [Mar84] include, in addition to identity types, a so-called *reflection rule* which stipulates that any propositional equality can be considered definitional.

$$\frac{p : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}_A\,a\,b)}{a = b}$$

Adding a reflection rule to a type theory makes its definitional equality undecidable, as equality becomes contingent on proofs of inhabitation of identity types, which is generally an undecidable problem.

Systems of type theory including the reflection rule are usually referred to as *Extensional Type Theory* (ETT). In contrast, we talk about *Intensional Type Theory* when the rule is not included.

2.1.2 Strict propositions and h-propositions

Under a literal interpretation of the propositions-as-types paradigm, the notion of proposition isn't a primitive aspect of the theory: the types are primitive, and we take some of them to represent propositions, with their terms and proofs.

Nevertheless, not all types are morally propositions, and in a type theory with identity types we can internalize the notion of proposition as a type with at most one element, or equivalently, one where all its elements are equal.

$$\mathsf{isProp}\ A :\equiv \Pi\ (x : A)\ \Pi\ (y : A)\ (\mathsf{Id}_A\ x\ y)$$

By this definition, being a proposition is an emergent property of types that requires proof. This kind of propositions are also known as *h-propositions* [The13].

An alternative approach is to have a judgment stipulating how to derive propositions. In this setting, being a proposition is a static property evident by construction rather than proof. We call propositions of this kind *strict propositions*

A way to implement this idea is to have a *universe* of strict propositions

$$\frac{\Gamma : \mathsf{Con}}{\mathcal{P} : \mathsf{Ty}\ \Gamma} \qquad \frac{P : \mathsf{Tm}\ \Gamma\ \mathcal{P}}{\mathcal{E}l_P\ P : \mathsf{Ty}\ \Gamma}$$

so that a term P is a strict proposition whenever P : Tm Γ $\mathcal{P}$ is derivable. Strict propositions are derived similarly to types, with formation, introduction, elimination, and computation rules. For example:

$$\frac{P : \mathsf{Tm}\ \Gamma\ \mathcal{P} \qquad Q : \mathsf{Tm}\ \Gamma\ \mathcal{P}}{P \wedge Q : \mathsf{Tm}\ \Gamma\ \mathcal{P}} \quad \frac{p : \mathsf{Tm}\ \Gamma\ (\mathcal{E}l_P\ P) \qquad q : \mathsf{Tm}\ \Gamma\ (\mathcal{E}l_P\ Q)}{\langle p, q\rangle : \mathsf{Tm}\ \Gamma\ (\mathcal{E}l_P\ (P \wedge Q))}$$

The static nature of strict propositions makes it easier to imbue them with additional features like *definitional proof-irrelevance*, an extensionality principle which stipulates that any two elements of a strict proposition are equal by definition.

$$\frac{p_1 : \mathsf{Tm}\ \Gamma\ (\mathcal{E}l_P\ P) \qquad p_2 : \mathsf{Tm}\ \Gamma\ (\mathcal{E}l_P\ P)}{p_1 = p_2}$$

Strict propositions are less flexible than h-propositions: we cannot dynamically turn a proof-relevant type into a strict proposition by proving its propositionality, nor we can eliminate strict propositions in a proof-relevant context. However, they have the advantage of allowing additional computation via definitional proof-irrelevance, by which any two proofs of the same proposition can be automatically replaced with one another as part of the type-checking phase.

Strict propositions are featured in the Lean theorem prover [dMKA+15], and have recently found their way into Coq and Agda [GCST19].

2.1.3 Uniqueness of identity proofs

Uniqueness of identity proofs (UIP) is the statement that any two proof terms of the same equality type are identified. In classical mathematics this principle wouldn't even deserve a name, since equality is always a proposition. In type theory, however, equality is a type, and its propositionality must be established by proof unless baked into the system itself. UIP is not directly provable in type theory, as shown by Hofmann and Streicher with a counterexample in groupoids [HS98]. It can, however, be consistently added to some forms of type theory, and it may be desirable to do so in some scenarios. For instance, ETT is conservative over intensional type theory with function extensionality and UIP [Hof96].

2.2 Models of Type Theory

We will use category theory as a framework to define abstract notions of semantics of dependent type theory. The advantage of this kind of abstraction is that it makes easier to show that a given mathematical structure models type theory, by showing that it is an instance of the abstract framework [Hof97].

We will consider one notion of model of type theory due to Dybjer, namely *categories with families* (CwFs) [Dyb95].

We use Extensional Type Theory (ETT) as the metalanguage; we write $x : A$ for a term x of type A; we write $(x : A) \to B$ for the type of dependent functions, and $\Sigma(x : A)\ B$ for the type of dependent pairs; we write $=$ for equality. We construct functions via abstraction $\lambda x \,.\, t$ and pairs by pairing x, y. As part of the tools of ETT, we recall function extensionality and uniqueness of identity proofs (UIP), to which we add schemas for inductive definitions and a cumulative hierarchy of universes $\mathbf{Type}_0, \mathbf{Type}_1, ...$ closed under the standard type formers.

We assume some basic background of category theory. For a category $\mathcal{C}$, we write $|\mathcal{C}|$ or simply $\mathcal{C}$ for the type[1] of objects, and $\mathcal{C}(c, c')$ for the set of morphisms between objects c and c' in $\mathcal{C}$, and sometimes just $c \to c'$ when $\mathcal{C}$ is clear from the context. Given a functor F, object c and morphism f of the appropriate types, we write $F\ c$ and $F\ f$ for the functorial action. We write *Sets* for the category

[1]Note that our types here are all "sets" in the sense of HoTT.

of types and functions between them; we abuse notation and implicitly take *Sets* to be parametric over universe levels, thus allowing it to be used for "sets" larger than those in $\mathbf{Type}_0$.

Given a presheaf $F : \mathcal{C}^{op} \longrightarrow \mathit{Sets}$, we define the *category of elements* of F as $|\int F| :\equiv \Sigma(c : \mathcal{C})(F\ c)$ and $(\int F)((c, x), (c', x')) :\equiv \Sigma(f : \mathcal{C}(c, c'))(F\ f\ x' = x)$. When applying a functor of the form $H : \int F \longrightarrow \mathcal{D}$, we often omit the equality proofs and write $H\ f$ instead of $H\ (f, p)$.

2.2.1 Categories with Families

A *category with families* (CwF) is comprised of the following:

- A category $\mathcal{C}$ of semantic contexts and context morphisms (substitutions), with a terminal object to model empty contexts;
- A presheaf $\mathsf{Ty} : \mathcal{C}^{op} \longrightarrow \mathit{Sets}$ mapping semantic contexts to a collection of types; for $A : \mathsf{Ty}\ \Gamma$ and $\sigma : \mathcal{C}(\Delta, \Gamma)$, we write $A[\sigma] \equiv \mathsf{Ty}\ \sigma\ A$;
- A presheaf $\mathsf{Tm} : (\int \mathsf{Ty})^{op} \longrightarrow \mathit{Sets}$ on the category of elements of Ty, mapping semantic types in a context to a collection of terms of that type; for $t : \mathsf{Tm}\ (\Gamma, A)$ and $\sigma : \mathcal{C}(\Delta, \Gamma)$, we write $t[\sigma] \equiv \mathsf{Tm}\ \sigma\ t$
- For Γ and $A : \mathsf{Ty}\ \Gamma$, a *context extension* $\Gamma.A : \mathcal{C}$ together with projections $\mathsf{p}_A : \mathcal{C}(\Gamma.A, \Gamma)$ and $\mathsf{v}_A : \mathsf{Tm}(\Gamma.A, A[\mathsf{p}_A])$ such that for each $\sigma : \mathcal{C}(\Delta, \Gamma)$ and $M : \mathsf{Tm}(\Delta, A[\sigma])$, there exists a unique morphism $\langle \sigma, M \rangle_A : \mathcal{C}(\Delta, \Gamma.A)$ satisfying $\mathsf{p}_A \circ \langle \sigma, M \rangle_A = \sigma$ and $\mathsf{v}_A[\langle \sigma, M \rangle_A] = M$.

We will often write $\sigma : \Gamma \longrightarrow \Delta$ in place of $\sigma : \mathcal{C}(\Gamma, \Delta)$ for morphisms, when the category $\mathcal{C}$ is clear from the context.

Weakening Given $\sigma : \Delta \longrightarrow \Gamma$ and $A : \mathsf{Ty}\ \Gamma$, we can define the weakening of σ by A, $\mathsf{q}(\sigma, A) : \Delta.A[\sigma] \longrightarrow \Gamma.A$, as follows

$$\mathsf{q}(\sigma, A) :\equiv \langle \sigma \circ \mathsf{p}_{A[\sigma]}, \mathsf{v}_{A[\sigma]} \rangle_A$$

A *weakening map* is a morphism of the form $\mathsf{p}_A : \Gamma.A \longrightarrow \Gamma$, or of the form $\mathsf{q}(w, A)$ where w is a weakening map. Syntactically, it corresponds to the usual weakening of a context of assumptions.

We sometimes write A^+ and M^+ for $A[w]$ and $M[w]$ if w is a weakening map that is clear from the context, and σ^+ for $\mathsf{q}(\sigma, A)$.

Semantic type formers

The definition of a CwF only considers the rules that are common to all systems of dependent types, in particular those about typing judgments and substitutions. In order to interpret type formers, we must give specifications for semantic type formers that a CwF must be equipped with in order to faithfully match the syntax. We give such specification for Π-types and identity types, referring the reader to [Hof97] for the full details of these and other type formers.

Π-types A CwF supports Π-types if for any $\Gamma : \mathcal{C}$ and semantic types $A : \mathsf{Ty}\,\Gamma$ and $B : \mathsf{Ty}\ (\Gamma.A)$, there exists a semantic type $\Pi\,A\,B : \mathsf{Ty}\,\Gamma$, a semantic abstraction operation $\mathsf{lam} : \mathsf{Tm}(\Gamma.A, B) \to \mathsf{Tm}(\Gamma, \Pi\,A\,B)$, and a semantic application operation $\mathsf{app} : \mathsf{Tm}(\Gamma, \Pi\,A\,B) \to \mathsf{Tm}(\Gamma.A, B)$, such that the following equations hold

$$\begin{array}{ll} \mathsf{app}(\mathsf{lam}\ M) = M & (\mathsf{lam}\ M)[\sigma] = \mathsf{lam}(M[\mathsf{q}(\sigma, A)]) \\ \mathsf{lam}(\mathsf{app}\ M) = M & (\mathsf{app}\ M)[\sigma] = \mathsf{app}(M[\mathsf{p} \circ \sigma]) \\ (\Pi\,A\,B)[\sigma] \;\;= \Pi\,(A[\sigma])\,(B[\mathsf{q}(\sigma, A)]) & \end{array}$$

We can recover the familiar binary application operator $\_\cdot\_$ from the available app: for $t : \mathsf{Tm}\ (\Gamma, \Pi\ A\ B)$ and $u : \mathsf{Tm}\ (\Gamma, A)$, we define $t \cdot u :\equiv (\mathsf{app}\ t)\langle \mathsf{id}, u\rangle_A$.

2.2.2 Presheaf model of Type Theory

Given a category $\mathcal{C}$, the category $\hat{\mathcal{C}}$ of presheaves on $\mathcal{C}$ admits a CwF structure and is therefore a model of type theory. We now sketch the construction using [Hof97, Lao17] as a reference, where more details can be found.

We model contexts as presheaves, and substitutions as natural transformations. Given a context Γ, we define $\mathsf{Ty}\ \Gamma$ as the type of presheaves on $\int \Gamma$. In both cases, we define presheaves to be ranging over some sort/universe **Type** in the metatheory[2]. We have a context extension operation defined as follows

$$\begin{aligned} &(\Gamma.A)\ c :\equiv \Sigma\ (\gamma : \Gamma\ c)(A\ (c, \gamma)) \\ &(\Gamma.A)\ f\ (\gamma, a) :\equiv \Gamma\ f\ \gamma, A\ f\ a \end{aligned}$$

We define a term $a : \mathsf{Tm}\ \Gamma\ A$ to be a function $a : (c : \mathcal{C})(\gamma : \Gamma\ c) \to A\ (c, \gamma)$ such that for all $i, j : \mathcal{C}$, $f : \mathcal{C}(j, i)$, and $\gamma : \Gamma\ i$, the equation $A\ f\ (a\ i\ \gamma) = a\ j\ (\Gamma\ f\ \gamma)$ holds.

[2]We avoid discussing size and universe levels in too much detail, however we do point out that the size of Ty as a presheaf is clearly larger than any presheaf $A : \mathsf{Ty}\,\Gamma$, as the type of small presheaves is not small.

Given $\sigma : \Gamma \longrightarrow \Delta$, we obtain $\sigma' : (\int \Gamma)^{op} \longrightarrow (\int \Delta)^{op}$ in the obvious way. We thus define substitution as $A[\sigma] :\equiv A \circ \sigma'$. For terms, we define $(t[\sigma])\ c :\equiv t\ c \circ (\sigma\ c)$.

We can also model all the usual type formers, like Π and Σ-types. For example, given $\Gamma : \mathsf{Con}, A : \mathsf{Ty}\ \Gamma, B : \mathsf{Ty}\ (\Gamma.A)$, we define:

$$(\Pi\ A\ B)\ (c, \gamma) :\equiv \forall\ c'\ (h : \mathcal{C}(c', c))(a : A\ (c', \Gamma\ h\ \gamma)) \to B\ (c', \Gamma\ h\ \gamma, a)$$

We can also model extensional (in the sense of the reflection rule) identity types:

$$(\mathsf{Id}_A\ x\ y)\ (c, \gamma) :\equiv x\ c\ \gamma = y\ c\ \gamma$$

Universes

If the metatheory is equipped with a cumulative hierarchy of universes [Mar75, Pal98] $\mathbf{Type}_0 : \mathbf{Type}_1 : ... : \mathbf{Type}_n : ... : \mathbf{Type}$, these can be lifted to universes in a presheaf category [HS97]. In particular, we obtain a hierarchy of presheaves Ty_i for all i. Similarly to Ty, we define $\mathsf{Ty}_i\ \Gamma$ as the type of $\mathbf{Type}_i$-small presheaves on $\int \Gamma$, i.e. presheaves H such that $H\,x$ is a type in $\mathbf{Type}_i$ for all x.

We thus extend the CwF structure of $\hat{\mathcal{C}}$ with types $\mathcal{U}_i : \mathsf{Ty}\,\Gamma$ and families $\mathcal{E}_i : \mathsf{Tm}\,\Gamma\,\mathcal{U}_i \to \mathsf{Ty}\,\Gamma$ for all i, such that $\mathcal{U}_i$ is closed under the same type formers as $\mathbf{Type}_i$.

Recall the Yoneda embedding $\mathsf{y} : \mathcal{C} \longrightarrow \hat{\mathcal{C}}$, mapping $c : \mathcal{C}$ to the presheaf $\mathsf{y}c$ such that $\mathsf{y}c\ x :\equiv \mathcal{C}(x, c)$. We define a universe presheaf $\overline{\mathcal{U}}_i$ as $\overline{\mathcal{U}}_i\ x :\equiv \mathsf{Ty}_i\ (\mathsf{y}\ x)$. Note that from any presheaf H, we can obtain a constant type $\mathsf{K}\ H : \mathsf{Ty}\ \Gamma$ in the obvious way. Hence we define $\mathcal{U}_i :\equiv \mathsf{K}\,\overline{\mathcal{U}}_i$, and $\mathcal{E}_i\,a\,c :\equiv a\,c\,\mathsf{id}_c$.

We have the following *representation* isomorphism $\mathsf{Ty}_i\ \Gamma \cong \mathsf{Tm}\ \Gamma\ \mathcal{U}_i$, or equivalently, since $\mathcal{U}_i$ is constant, $\mathsf{Ty}_i\ \Gamma \cong \hat{\mathcal{C}}(\Gamma, \mathcal{U})$, both natural in Γ [Lao17].

2.2.3 Models of type theory within presheaves

As shown in the previous paragraphs, for any category $\mathcal{C}$ the presheaf category $\hat{\mathcal{C}}$ is a model of extensional type theory. This allows us to use type-theoretic syntax as the *internal language* of $\hat{\mathcal{C}}$ to express constructions in it.

When $\mathcal{C}$ is equipped with a CwF structure, we can use the internal language of $\hat{\mathcal{C}}$ to talk about constructions in the CwF $\mathcal{C}$ in a succinct way.

For example, we can reflect the semantic types $\mathsf{Ty} : \mathsf{PSh}(\mathcal{C})$ and terms $\mathsf{Tm} : \mathsf{PSh}(\int \mathsf{Ty})$ of $\mathcal{C}$ as the following (internal) terms:

$$\overline{\mathsf{Ty}} : \mathcal{U}$$
$$\overline{\mathsf{Tm}} : \overline{\mathsf{Ty}} \to \mathcal{U}$$

Viewed externally, this requires us to define natural transformations $\overline{\mathsf{Ty}} : \hat{\mathcal{C}}(\mathbb{1}, \mathcal{U})$ and $\overline{\mathsf{Tm}} : \hat{\mathcal{C}}(\mathbb{1}, \overline{\mathsf{Ty}} \to \mathcal{U})$. By $\mathcal{U}$'s representation equivalence, $\hat{\mathcal{C}}(\mathbb{1}, \mathcal{U}) \cong \mathsf{PSh}(\int \mathbb{1}) \cong \mathsf{PSh}(\mathcal{C})$, hence we have

$$\begin{aligned}&\overline{\mathsf{Ty}} : \mathsf{PSh}(\mathcal{C})\\&\overline{\mathsf{Ty}} :\equiv \mathsf{Ty}\end{aligned}$$

Similarly, by $\mathcal{U}$'s representation equivalence and cartesian closure, $\hat{\mathcal{C}}(\mathbb{1}, \overline{\mathsf{Ty}} \to \mathcal{U}) \cong \hat{\mathcal{C}}(\overline{\mathsf{Ty}}, \mathcal{U}) \cong \mathsf{PSh}(\int \overline{\mathsf{Ty}}) = \mathsf{PSh}(\int \mathsf{Ty})$, hence we have:

$$\begin{aligned}&\overline{\mathsf{Tm}} : \mathsf{PSh}(\textstyle\int \mathsf{Ty})\\&\overline{\mathsf{Tm}} :\equiv \mathsf{Tm}\end{aligned}$$

We can similarly reflect the type formers of $\mathcal{C}$'s CwF structure in $\hat{\mathcal{C}}$. For instance, we can represent $\Pi : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ internally as follows:

$$\overline{\Pi} : (A : \overline{\mathsf{Ty}}) \to (\overline{\mathsf{Tm}}\,A \to \overline{\mathsf{Ty}}) \to \overline{\mathsf{Ty}}$$

Note that the reflected terms do not make any mention of contexts and substitutions in $\mathcal{C}$; these have been hidden under the presheaf structure of $\hat{\mathcal{C}}$, which ensures that any construction we carry out automatically respects the CwF structure of $\mathcal{C}$ and its laws. Moreover, as shown in $\overline{\Pi}$, we express type dependency by exploiting the dependent function type of the internal language, in a way that is reminiscent of *higher-order abstract syntax* [PE88].

This idea of abstracting CwF structures via presheaves goes both ways: instead of using presheaves to *describe* and talk about a specific existing model, we can use it to *specify* type theories and their notion of models *ex novo*, in a way that is equivalent to CwFs but that saves us from mentioning contexts and substitutions at all. We start by specifying the sorts, type, and term formers of our type theory using the internal language of presheaf categories. For example, the following list specifies a type theory with (extensional) Π-types.

$$\begin{aligned}&\mathsf{Ty} : \mathcal{U}\\&\mathsf{Tm} : \mathsf{Ty} \to \mathcal{U}\\&\Pi : (A : \mathsf{Ty}) \to (\mathsf{Tm}\ A \to \mathsf{Ty}) \to \mathsf{Ty}\\&\mathsf{app} : \forall\, A\, B,\ \mathsf{Tm}\,(\Pi\, A\, B) \simeq ((a : \mathsf{Tm}\, A) \to \mathsf{Tm}\,(B\, a))\end{aligned}$$

where $\simeq$ is isomorphism. A model of this theory is then given by a category $\mathcal{C}$ with a terminal object, and an interpretation of the terms above in $\hat{\mathcal{C}}$.

If we unfold this notion of model internal to $\hat{\mathcal{C}}$, what we obtain externally is the same as the structure of a CwF on top of $\mathcal{C}$ specifying the type theory above[3], or equivalently an algebra for the signature of a quotient inductive-inductive type (QIIT) describing the theory. Both the CwF structure and the QIIT are significantly more verbose since they include sorts and constructors for contexts and substitutions, as well as equations governing them, all of which are implicit in the internal model presentation [BKS21]. This internal formulation is therefore very convenient when one wants to prove meta-theorems about a type theory by reasoning about its models: by working in the internal language of presheaves categories, we can define and reason about such models in a succinct way, without having to keep track of contexts and stability under substitutions.

[3]Actually, for this formulation of models to be equivalent to the full structure of a CwF, and in particular to obtain the property of context extension such that $\mathcal{C}(\Gamma, \Delta \triangleright A) \simeq \Sigma(\gamma : \mathcal{C}(\Gamma, \Delta))(\mathsf{Tm}\,\Gamma\, A[\sigma])$, we need to define Tm as a family of *locally-representable* presheaves. This can be done by defining a suitable sub-universe $\mathcal{U}^*$ and asking for a $\mathcal{U}^*$-valued family Tm [BKS21].

Chapter 3

Induction-induction

Induction-induction is a schema in type theory that allows one to *inductively* define a type $A : \mathbf{Type}$ mutually with a family $B : A \to \mathbf{Type}$ over A [NF13]. Inductive-inductive types (IITs) come equipped with elimination principles induced by their inductive structure. Consider an arbitrary IIT $(A : \mathbf{Type}, B : A \to \mathbf{Type})$; the most general form of (dependent) eliminator is given by functions

$$\begin{aligned} &\mathsf{elim}_A : (a : A) \to F\ a \\ &\mathsf{elim}_B : (a : A) \to (b : B\ a) \to G\ a\ b\ (\mathsf{elim}_A\ a) \end{aligned}$$

for each pair of *motives*[1] $F : A \to \mathbf{Type}$ and $G : (a : A) \to (b : B\ a) \to F\ a \to \mathbf{Type}$.

Note that the form of type dependency in the sorts of the IIT is naturally reflected in the type of its eliminators: that is, one of the functions above, elim_A, appears in the type of the other, elim_B. We say that the two functions are *recursive-recursive*, after [NF13].

A simpler notion of eliminator does not require the second component of the motive to mention the first: for motives $F : A \to \mathbf{Type}$ and $G : (a : A) \to B\ a \to \mathbf{Type}$, we have functions

$$\begin{aligned} &\mathsf{elim}'_A : (a : A) \to F\ a \\ &\mathsf{elim}'_B : (a : A)(b : B\ a) \to G\ a\ b \end{aligned}$$

Note that this elimination principle—that we call *simple elimination* after [NF13]—is no longer recursive-recursive, since elim_A is not mentioned in the type of elim_B.

Induction-induction allows for an arbitrary number of sorts to be defined simultaneously. An example of multi-sorted IIT is given by the intrinsic definition

[1]We refer to the target types and terms of the induction as *motives* and *methods*, respectively.

of a dependent type theory in type theory, where each sort implements a different form of judgment [Cha09, AK16]. We illustrate this with a minimal syntax with contexts Con, types Ty, and terms Tm (and no equations):

$$\begin{aligned}
&\mathsf{Con} : \mathbf{Type}\\
&\mathsf{Ty} : \mathsf{Con} \to \mathbf{Type}\\
&\mathsf{Tm} : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\ \Gamma) \to \mathbf{Type}\\
\\
&\mathsf{nil} : \mathsf{Con}\\
&\mathsf{ext} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Con}\\
&\mathsf{iota} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma\\
&\mathsf{pi} : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\mathsf{ext}\ \Gamma\ A) \to \mathsf{Ty}\ \Gamma\\
&\mathsf{lam} : \forall\{\Gamma\ A\ B\} \to \mathsf{Tm}\ (\mathsf{ext}\ \Gamma\ A)\ B \to \mathsf{Tm}\ \Gamma\ (\mathsf{pi}\ \Gamma\ A\ B)
\end{aligned}$$

What characterizes IITs in contrast to other forms of induction is that constructors can refer to other constructors that are defined within the same mutual definition. For example, in the signature shown above the type constructor pi refers to context extension ext, whereas lam refers to both ext and pi.

Recursion-recursion In dependent type theory, recursive definitions are usually expressed as instantiations of the elimination principle of some inductive type. The elimination principle for inductive families is strong enough to express a wide variety of recursive definitions, including many forms of mutual recursion.

A mutual recursion schema that does not seem to arise from inductive families is recursion-recursion. Informally, we say that a group of mutually-defined functions is recursive-recursive when some of those functions appear in the type of other functions in the same group. We have seen this pattern of dependency just now, in the general eliminators for inductive-inductive types: recall that, for example, given an IIT $A : \mathbf{Type}, B : A \to \mathbf{Type}$ and *motives* $F : A \to \mathbf{Type}$ and $G : (a : A) \to (b : B\ a) \to F\ a \to \mathbf{Type}$, the corresponding general eliminators can be expressed as the following pair of functions

$$\begin{aligned}
&\mathsf{elim}_A : (a : A) \to F\ a\\
&\mathsf{elim}_B : (a : A) \to (b : B\ a) \to G\ a\ b\ (\mathsf{elim}_A\ a)
\end{aligned}$$

Thus, one way to make the concept of recursion-recursion precise is to frame it as the form of recursion induced by the induction principle of IITs [NF13]. From this point of view, "recursion-recursion" is simply an informal label for particular instantiations of IIT eliminators. We are not aware of alternative treatments of recursion-recursion outside the context of induction-induction; therefore, in this thesis we will largely use recursion-recursion in conjunction with IITs, in the sense just described. An exception to this is Section 6.2, where we give a first approximation of a universe construction in the strict setoid model of type theory as an inductive-recursive-recursive (IRR) type; given the difficultly to express IRR definitions in terms of well-understood forms of induction, we will quickly proceed, in Section 6.4, to reduce this first universe construction to one that only relies on induction-induction and its induction principles.

3.1 Reducing multi-sorted IITs

Many instances of multi-sorted IITs can be reduced to equivalent two-sorted IITs, via a systematic reduction method originally observed by Zongpu (Szumi) Xie [Kap19]. We are not aware of a formal proof of this construction for arbitrary IITs, but we conjecture that it does apply to all instances of induction-induction and consequently that it shows two-sorted IITs are enough to represent any specifiable IIT.

We illustrate the idea of this reduction with an example. Let us consider the small type-theoretic syntax $\mathsf{Con}, \mathsf{Ty}, \mathsf{Tm}$ defined a few paragraphs above. We define a two-sorted IIT $\mathsf{V} : \mathbf{Type}, \mathsf{F} : \mathsf{V} \to \mathbf{Type}$, with the idea to use V to encode the sorts of the multi-sorted IIT, and F to encode its constructors:

$$
\begin{aligned}
&\mathsf{Con}_\mathsf{V} : \mathsf{V}\\
&\mathsf{Ty}_\mathsf{V} : \mathsf{F}\ \mathsf{Con}_\mathsf{V} \to \mathsf{V}\\
&\mathsf{Tm}_\mathsf{V} : (\Gamma : \mathsf{F}\ \mathsf{Con}_\mathsf{V}) \to \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ \Gamma) \to \mathsf{V}\\
\\
&\mathsf{nil}_\mathsf{F} : \mathsf{F}\ \mathsf{Con}_\mathsf{V}\\
&\mathsf{ext}_\mathsf{F} : (\Gamma : \mathsf{F}\ \mathsf{Con}_\mathsf{V}) \to \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ \Gamma) \to \mathsf{F}\ \mathsf{Con}_\mathsf{V}\\
&\mathsf{iota}_\mathsf{F} : (\Gamma : \mathsf{F}\ \mathsf{Con}_\mathsf{V}) \to \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ \Gamma)\\
&\mathsf{pi}_\mathsf{F} : (\Gamma : \mathsf{F}\ \mathsf{Con}_\mathsf{V})(A : \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ \Gamma)) \to \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ (\mathsf{ext}_\mathsf{F}\ \Gamma\ A)) \to \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ \Gamma)\\
&\mathsf{lam}_\mathsf{F} : \forall\{\Gamma\ A\ B\} \to \mathsf{F}\ (\mathsf{Tm}_\mathsf{V}\ (\mathsf{ext}_\mathsf{F}\ \Gamma\ A)\ B) \to \mathsf{F}\ (\mathsf{Tm}_\mathsf{V}\ \Gamma\ (\mathsf{pi}_\mathsf{F}\ \Gamma\ A\ B))
\end{aligned}
$$

We can easily model the types of the original IIT by defining

$$\mathsf{Con} :\equiv \mathsf{F}\ \mathsf{Con}_\mathsf{V} \qquad \mathsf{Ty}\ \Gamma :\equiv \mathsf{F}\ (\mathsf{Ty}_\mathsf{V}\ \Gamma) \qquad \mathsf{Tm}\ \Gamma\ A :\equiv \mathsf{F}\ (\mathsf{Tm}_\mathsf{V}\ \Gamma\ A)$$

Modeling the constructors is straightforward, as they are just exactly their V/F counterparts: $\mathsf{nil} :\equiv \mathsf{nil}_\mathsf{F}, \mathsf{ext} :\equiv \mathsf{ext}_\mathsf{F}, \mathsf{etc.}$

The eliminators follow just as easily from the induction principle of V/F. For example, given motives $\mathsf{Con}^D, \mathsf{Ty}^D, \mathsf{Tm}^D$ and methods $\mathsf{nil}^D, \mathsf{ext}^D, \mathsf{iota}^D, ...$, we can define the eliminators for $\mathsf{Con}/\mathsf{Ty}/\mathsf{Tm}$

$$
\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}} : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma \\
&\mathsf{elim}_{\mathsf{Ty}} : (A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma) \\
&\mathsf{elim}_{\mathsf{Tm}} : (t : \mathsf{Tm}\ \Gamma\ A) \to \mathsf{Tm}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\mathsf{elim}_{\mathsf{Ty}}\ A)
\end{aligned}
$$

by mutual recursion-recursion:

$$
\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}}\ \mathsf{nil}_{\mathsf{F}} &&:\equiv \mathsf{nil}^D \\
&\mathsf{elim}_{\mathsf{Con}}\ (\mathsf{ext}_{\mathsf{F}}\ \Gamma\ A) &&:\equiv \mathsf{ext}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\mathsf{elim}_{\mathsf{Ty}}\ A) \\
&\mathsf{elim}_{\mathsf{Ty}}\ (\mathsf{iota}_{\mathsf{F}}\ \Gamma) &&:\equiv \mathsf{iota}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma) \\
&\ldots
\end{aligned}
$$

3.2 Reducing finitary induction-induction

Induction-induction is as powerful as it is complex, and both the syntax and semantics of IITs is topic for current research. Because of this, a question that naturally arises is whether we can reduce some forms of induction-induction to simpler objects, like inductive families. There are several reasons for doing this, ranging from pure theoretical interest to mere practical needs. On the practical side, it should be noted that some of the more popular proof-assistants based on type theory, like Coq and Lean, do not currently support inductive-inductive definitions out of the box. In those systems, encoding IITs as inductive types is the only way to work with induction-induction.

It is known that finitary inductive-inductive definitions can be reduced to inductive families [AKKvR19, AKKvR18, KKL20]. To illustrate the process, let us consider a well-known example of a finitary inductive-inductive type, the by-now familiar intrinsic encoding of type theory in type theory. We consider a minimal version with just contexts $\mathsf{Con} : \mathbf{Type}$ and types $\mathsf{Ty} : \mathsf{Con} \to \mathbf{Type}$, as those alone are already enough to require induction-induction.

The reduction method showcased in this section is essentially the one described in [vR19]. We assume to be working in a sufficiently strong version of intensional type theory with inductive types, equality, and UIP.

Remark 3.2.1. We will use Agda-style notation and write $(x : A) \to B\ x$ for dependent function types, using curly brackets $\{a : A\} \to B\ x$ for implicit quantification. We write $\lambda x\,.\,t$ to construct functions, and $\_,\_$ as infix constructor for dependent pairs. We also make extensive use of (dependent) pattern-matching definitions. We sometimes explicitly name patterns, writing $x@(p\,...)$ to give the pattern $(p\,...)$ a scoped name x. We also write $f\,\{x\}\,y\,z...$ in function definitions to bring an implicit parameter x in scope. We use underscores notation _ like in Agda, i.e. as stand-ins for unnamed parameters to function definitions, as well as arguments to function applications when their value can be inferred from context/unification. We do however depart from Agda notation in writing $\equiv$ and $:\equiv$ for definitional equality, and $=$ for propositional equality. ■

Contexts in Con are formed out of empty contexts $\bullet$ and context extension $\_\triangleright\_$. Types in Ty are either the base type ι or dependent function types $\bar{\pi}$.

$$\begin{aligned}
&\bullet \quad : \mathsf{Con} && \iota : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma\\
&\_\triangleright\_ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Con} && \bar{\pi} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma \triangleright A) \to \mathsf{Ty}\ \Gamma
\end{aligned}$$

The general method to eliminate induction-induction is to split the original IIT into a type of codes— we call them *erased types*— and associated well-formedness predicates. In our Con/Ty example, we have erased types $\mathsf{Con}_0, \mathsf{Ty}_0 : \mathbf{Type}$ and predicates $\mathsf{Con}_1 : \mathsf{Con}_0 \to \mathbf{Type}, \mathsf{Ty}_1 : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathbf{Type}$.

The definition of the erased and predicate types follows that of the original inductive-inductive type, and can be derived systematically from it. More importantly, they can be defined without induction-induction.

$$\begin{aligned}
&\bullet_0 \quad : \mathsf{Con}_0\\
&\_\triangleright_0\_ : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Con}_0\\
&\iota_0 \quad : \mathsf{Con}_0 \to \mathsf{Ty}_0\\
&\bar{\pi}_0 \quad : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0
\end{aligned}$$

$$\begin{aligned}
&\bullet_1 \quad : \mathsf{Con}_1\ \bullet_0\\
&\_\triangleright_1\_ : \forall\{\Gamma_0\ A_0\} \to \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ A_0\\
&\qquad \to \mathsf{Con}_1\ (\Gamma_0 \triangleright_0 A_0)\\
&\iota_1 \quad : \forall\{\Gamma_0\} \to \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0)\\
&\bar{\pi}_1 \quad : \forall\{\Gamma_0\ A_0\ B_0\} \to \mathsf{Con}_1\ \Gamma_0\\
&\qquad \to \mathsf{Ty}_1\ \Gamma_0\ A_0 \to \mathsf{Ty}_1\ (\Gamma_0 \triangleright_0 A_0)\ B_0\\
&\qquad \to \mathsf{Ty}_1\ \Gamma_0\ (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0)
\end{aligned}$$

Because erasing the indices loses information, there are generally more erased expressions of type $\mathsf{Con}_0, \mathsf{Ty}_0$ than there are of the original Con/Ty IIT. The well-formedness predicates serve the purpose to constrain the value of erased elements that they predicate over, so that we only accept as well-formed those erased expressions that could have been equivalently obtained via the constructors of the original IIT.

By induction on well-formedness proofs we can expose these constraints, and in particular we have the following equational inversion principles:

$$\mathsf{inv}\text{-}\iota_1 : \forall\{\Gamma_0\ \Gamma_0'\} \to \mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0') \to \Gamma_0 = \Gamma_0'$$

$$\mathsf{inv}\text{-}\bar{\pi}_1 : \forall\{\Gamma_0\ \Gamma_0'\ A_0\ B_0\} \to \mathsf{Ty}_1\ \Gamma_0\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0) \to \Gamma_0 = \Gamma_0'$$

Moreover, we can prove that the well-formedness predicates are h-propositions, by straightforward induction/pattern-matching. Propositionality of the predicates will play a crucial role later, when we derive the eliminators for the encoded IIT.

We can recover the original inductive-inductive type as $\mathsf{Con} :\equiv \Sigma\ (\Gamma_0 : \mathsf{Con}_0)\ (\mathsf{Con}_1\ \Gamma_0)$ and $\mathsf{Ty}\ \Gamma :\equiv \Sigma\ (A_0 : \mathsf{Ty}_0)\ (\mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0)$. Recovering the constructors is straightforward:

$$\begin{aligned}
&\bullet && :\equiv (\bullet_0, \bullet_1)\\
&(\Gamma_0,\Gamma_1) \triangleright (A_0, A_1) && :\equiv ((\Gamma_0 \triangleright_0 A_0), (\Gamma_1 \triangleright_1 A_1))\\
&\iota\ (\Gamma_0,\Gamma_1) && :\equiv (\iota_0\ \Gamma_0, \iota_1\ \Gamma_1)\\
&\bar{\pi}\ \{\Gamma_0,\Gamma_1\}(A_0,A_1)(B_0,B_1) && :\equiv (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0, \bar{\pi}_1\ \Gamma_1\ A_1\ B_1)
\end{aligned}$$

If we were to define the type above as an IIT, this would come equipped with an inductive principle, i.e. *eliminators*, reflecting the inductive structure of the type [NF13]. In order to write down the eliminators for this IIT, we look at *displayed algebras* over Con and Ty, which are given by indexed types:

$$\mathsf{Con}^D : \mathsf{Con} \to \mathbf{Type} \qquad \mathsf{Ty}^D : \{\Gamma : \mathsf{Con}\} \to \mathsf{Con}^D\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathbf{Type}$$

and functions corresponding to each constructor:

$$\begin{aligned}
&\bullet^D && : \mathsf{Con}^D\ \bullet\\
&\_ \triangleright^D \_ && : \forall\{\Gamma\ A\}(\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ A \to \mathsf{Con}^D\ (\Gamma \triangleright A)\\
&\iota^D && : \forall\{\Gamma\}(\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ (\iota\ \Gamma)\\
&\bar{\pi}^D && : \forall\{\Gamma\ A\ B\}(\Gamma^D : \mathsf{Con}^D\ \Gamma)(A^D : \mathsf{Ty}^D\ \Gamma^D\ A)(B^D : \mathsf{Ty}^D\ (\Gamma^D \triangleright^D A)\ B)\\
& && \to \mathsf{Ty}^D\ \Gamma^D\ (\bar{\pi}\ \Gamma\ A\ B)
\end{aligned}$$

The general eliminators have the following signatures:

$$\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}} : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma\\
&\mathsf{elim}_{\mathsf{Ty}} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ A
\end{aligned}$$

In addition, we have β-equations that explain the computational behaviour of the eliminators on constructors:

$$\mathsf{elim}_{\mathsf{Con}}\ \bullet \qquad \equiv \bullet^D$$

$$\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}}\ (\Gamma \rhd A) &&\equiv \mathsf{elim}_{\mathsf{Con}}\ \Gamma \rhd^D \mathsf{elim}_{\mathsf{Ty}}\ A\\
&\mathsf{elim}_{\mathsf{Ty}}\ (\iota\ \Gamma) &&\equiv \iota^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\\
&\mathsf{elim}_{\mathsf{Ty}}(\bar{\pi}\ \Gamma\ A\ B) &&\equiv \bar{\pi}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\mathsf{elim}_{\mathsf{Ty}}\ A)\ (\mathsf{elim}_{\mathsf{Ty}}\ B)
\end{aligned}$$

The general eliminators can be derived from our encoding of Con and Ty via untyped codes and well-typing predicates. Unfortunately it is not possible to define them directly, as the induction principle of the erased and predicate types cannot express the recursive-recursive structure of the eliminators. Instead, we first define the graph of the eliminators in the form of inductively-generated relations:

$$\begin{aligned}
&\mathsf{Con}^R : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma \to \mathbf{Type}\\
&\mathsf{Ty}^R\ \ : \{\Gamma : \mathsf{Con}\}(\Gamma^D : \mathsf{Con}^D\ \Gamma)(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ \Gamma\ A\ \Gamma^D \to \mathbf{Type}\\
&\bullet^R\ \ \ \ : \mathsf{Con}^R\ \bullet\ \bullet^D\\
&\_\rhd^R\_ : \forall\{\Gamma\ \Gamma^D\ A\ A^D\} \to \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ A\ A^D\\
&\qquad \to \mathsf{Con}^R\ (\Gamma \rhd A)\ (\Gamma^D \rhd^D A^D)\\
&\iota^R\ \ \ \ : \forall\{\Gamma\ \Gamma^D\} \to \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ (\iota\ \Gamma)\ (\iota^D\ \Gamma^D)\\
&\bar{\pi}^R\ \ \ \ : \forall\{\Gamma\ \Gamma^D\ A\ A^D\ B\ B^D\} \to \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ A\ A^D\\
&\qquad\ \ \to \mathsf{Ty}^R\ (\Gamma^D \rhd^D A^D)\ B\ B^D\\
&\qquad\ \ \to \mathsf{Ty}^R\ \Gamma^D\ (\bar{\pi}\ \Gamma\ A\ B)\ (\bar{\pi}^D\ \Gamma^D\ A^D\ B^D)
\end{aligned}$$

The next step is to prove that these relations are functional, by induction on the untyped codes Con_0 and Ty_0. We split functionality into mutual proofs $\mathsf{Con}^\exists$, $\mathsf{Ty}^\exists$ of left-totality, and proofs $\mathsf{Con}^!$, $\mathsf{Ty}^!$ of right-uniqueness.

$$\begin{aligned}
&\mathsf{Con}^\exists : \forall\ \Gamma \to \Sigma(\mathsf{Con}^D\ \Gamma)(\mathsf{Con}^R\ \Gamma)\\
&\mathsf{Ty}^\exists : \forall\{\Gamma\,\Gamma^D\}(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \Sigma(\mathsf{Ty}^D\,\Gamma\ A\ \Gamma^D)(\mathsf{Ty}^R\ A\ \Gamma^D)\\
&\mathsf{Con}^! : \forall\,\{\Gamma\,\Gamma^D\,\Gamma^{D'}\} \to \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Con}^R\ \Gamma\ \Gamma^{D'} \to \Gamma^D = \Gamma^{D'}\\
&\mathsf{Ty}^! : \forall\,\{\Gamma\,\Gamma^D\,A\,A^D\,A^{D'}\} \to \mathsf{Ty}^R\,A\,\Gamma^D\,A^D \to \mathsf{Ty}^R\,A\,\Gamma^D\,A^{D'} \to A^D = A^{D'}
\end{aligned}$$

We will not go through the entire definition of these terms. Instead, let us consider a couple of illustrative examples, beginning with the proof of left-totality in the case of the ι constructor. Pattern-matching on the first component of the pair A given as input to $\mathsf{Ty}^\exists$ leads to the following:

$$\mathsf{Ty}^{\exists}\ \{\Gamma\ \Gamma^D\}\ A@(\iota_0\ \Gamma_0, A_1)\ r :\equiv\ ?$$

We would like to conclude the proof with the pair $(\iota^D\ \Gamma^D, \iota^R\ r)$. Alas, the type of this expression does not match the goal type. We cannot proceed in any way unless we first establish that the input term A is equal to the expected canonical form $\iota\ \Gamma$. We prove this equation component-wise, beginning with $\pi_1\ (\iota\ \Gamma) = \pi_1\ A$. By injectivity of ι_0, this reduces to $\pi_1\ \Gamma \equiv \Gamma_0$, which we can prove by inversion on A_1 (using $\mathsf{inv}\text{-}\iota_1$), or alternatively by dependent pattern-matching. We now need to prove $\pi_2\ (\iota\ \Gamma) = \pi_2\ A$, that is, $\pi_2\ \Gamma = \Gamma_1$. This equation follows from propositionality of the well-formedness predicates, which we have established previously.

We can thus informally rewrite the incomplete definition as:

$$\mathsf{Ty}^{\exists}\ \{\Gamma\ \Gamma^D\}\ (\iota\ \Gamma)\ r :\equiv\ ?$$

We conclude the proof with $(\iota^D\ \Gamma^D, \iota^R\ r)$.

Let us now consider the case of the $\bar{\pi}$ constructor. Again, we pattern-match of the first component of the Ty argument to $\mathsf{Ty}^{\exists}$, which yields the following

$$\mathsf{Ty}^{\exists}\ \{\Gamma\ \Gamma^D\}\ A@(\bar{\pi}_0\ \Gamma_0\ X_0\ Y_0, A_1)\ r :\equiv\ ?$$

As before, just pattern matching on the erased component of A is not enough, as we cannot proceed any further with the proof unless we first prove that the input term A is equal to a canonical expression formed by the IIT constructor $\bar{\pi}$, under context Γ. Like in the previous case, we prove $\Gamma = (\Gamma_0, \Gamma_1)$ by inversion and propositionality, where the well-formedness proof $\Gamma_1 : \mathsf{Con}_1\ \Gamma_0$ is obtained by inversion on A_1, along with $X_1 : \mathsf{Ty}_1\ \Gamma_0\ X_0$ and $Y_1 : \mathsf{Ty}_1\ (\Gamma_0 \triangleright_0 X_0)\ Y_0$. We set $X :\equiv (X_0, X_1)$ and $Y :\equiv (Y_0, Y_1)$.

By induction, we obtain displayed terms and relatedness proofs:

$$\begin{aligned} &X^D : \mathsf{Ty}^D\ \Gamma\ X\ \Gamma^D && X^R : \mathsf{Ty}^R\ X\ \_\ X^D \\ &Y^D : \mathsf{Ty}^D\ (\Gamma \triangleright X)\ Y\ (\Gamma^D \triangleright^D X^D) && Y^R : \mathsf{Ty}^R\ Y\ \_\ Y^D \end{aligned}$$

with which we conclude the left-totality proof

$$\mathsf{Ty}^{\exists}\ \{\Gamma\ \Gamma^D\}\ (\bar{\pi}\ \Gamma\ X\ Y)\ r :\equiv (\bar{\pi}^D\ \Gamma^D\ X^D\ Y^D, \bar{\pi}^R\ r\ X^R\ Y^R)$$

Defining the eliminators is immediate from functionality of the relations:

$$\begin{aligned} &\mathsf{elim}_{\mathsf{Con}}\ \Gamma :\equiv \pi_1\ (\mathsf{Con}^{\exists}\ \Gamma) \\ &\mathsf{elim}_{\mathsf{Ty}}\ \{\Gamma\}\ A :\equiv \pi_1\ (\mathsf{Ty}^{\exists}\ A\ (\pi_2\ (\mathsf{Con}^{\exists}\ \Gamma))) \end{aligned}$$

These eliminators also admit the β-equations discussed above. However, if the ambient theory is intensional as in the current case, then the β-equations only hold up to propositional equality, due to the transports involved in the proofs of functionality of the relations. We observe, however, that some of these transports can in fact be reduced away by employing an equality type with a strong notion of UIP, and in particular equipped with a transport operation that computes to the identity function on arbitrary reflexive proofs $p : x = x$. We will explore this further in Chapter 8, where we consider a reduction method relying on a definitionally proof-irrelevant identity type.

Note, moreover, that we did not need to invoke right-uniqueness of the eliminator relations to prove left-totality in this example. This is due to the "shape" of the type Con/Ty and will not be the case for every IIT, and in particular for the so called *non-linear* IITs as discussed further down in Section 9.1. Proving right-uniqueness would thus appear to be necessary in the general case.

Chapter 4

This thesis

4.1 Contributions

This thesis contains the following main contributions:

- We extend the strict setoid model of intensional type theory with a universe of setoids. We define the universe in multiple stages, first as an inductive-recursive definition, which is then translated to an inductive-inductive definition and finally to an inductive family. Notably, the first definition of the universe uses a unique combination of large induction-recursion, recursion-recursion, and proof-irrelevance. While useful to give an intuitive presentation of the universe, this complex form of induction-recursion-recursion is not currently properly understood. A core contribution of this thesis is to show how to define a setoid universe without relying on any form of induction-recursion or recursion-recursion. We present this work in Chapter 6.

  The material covered in Chapter 6 up to Section 6.5 is joint work with Altenkirch, Boulier, Kaposi, and Sattler, and has been published in the paper at [ABK$^+$21]. While the paper does present a definition of the setoid universe in terms of an infinitary IIT encoded via inductive families, it notably does not derive the general eliminators for it. We improve on the results of the paper and construct full dependent eliminators for the aforementioned universe IIT (Section 6.6). We then show that the setoid universe defined via this IIT supports universe induction (Section 6.6.1.)

- We contribute a method to systematically reduce a wide class of infinitary IITs to inductive families, using an encoding that does not rely on function extensionality in the encoding theory. We first demonstrate the reduction with two concrete examples (Chapter 8). The method we employ is a modified version of the reduction method for *finitary* IITs [vR19], with two main novelties that make it applicable to infinitary types:

  – careful control over the proof of left-totality of the eliminator relations, so that proving right-uniqueness becomes unnecessary;
  – use of a universe of definitionally proof-irrelevant propositions and a proof irrelevant identity type with a strong transport rule targeting proof-relevant types. These allow to define predicates that are propositional by definition, and to achieve definitional β-rules for the encoded eliminators even in an intensional setting.

  We identify a subclass of all infinitary IIT specifications, and generalize the reduction method demonstrated in Chapter 8 to arbitrary specifications in this class (Chapter 9). After making formal how to specify the IITs we are targeting, we show how to encode the types, constructors, and eliminators of any specifiable IIT, in any model of intensional type theory supporting inductive families and a universe of definitionally proof-irrelevant propositions (Chapter 11, Chapter 12).

- We provide Agda formalizations [Ses23] for the entire mathematical content of Part II (first contribution point), and for most of the mathematical content of Part III (second and third contribution points.)

4.2 Structure

The rest of the thesis is divided into two parts, Part II and Part III, with Part I concluding with this section. Part II will cover our work on the setoid model and its extension with a universe of setoids, while Part III will cover our work on reducing infinitary IITs to inductive families.

We begin Part II in Chapter 5 with an introductory discussion of setoid models of type theory, with a focus on the strict setoid model [Alt99] and its syntactic manifestation SeTT [ABKT19]. We then extend the strict setoid model with a universe of setoids in Chapter 6. We wrap up our work on the setoid model and Part II with Chapter 7.

We move on to Part III with Chapter 8, which presents some examples of encoding infinitary IITs as inductive families in an intensional ambient theory. Chapter 9 sets the stage for the generalization of such encoding method from concrete examples to arbitrary IIT specifications. We frame the general reduction method as a construction on algebras: we begin by defining algebras of IITs (Chapter 10), then show that for any specifiable IIT, we can construct a corresponding algebra (Chapter 11) that enjoys the expected induction principle (Chapter 12). We finally wrap Part III and the thesis with Chapter 13.

Part II

Setoids and extensionality

Chapter 5

The setoid model of type theory

Extensionality is the ability to treat mathematical objects that behave the same way in every situation as equal. Extensional reasoning is not concerned with details of encoding and implementation, but only considers the qualities exposed by objects to the outside. This is in contrast with *intensionality*, where we do pay attention to *how* objects are constructed, and accordingly distinguish between them.

Extensional reasoning is ubiquitous in mathematical practice. In fact, mathematicians reason extensionally all the time, when they identify point-wise equal functions, sets with the same elements, or isomorphic algebraic structures. Extensional reasoning is greatly facilitated by the ability of the formal system to enforce layers of abstraction: replacing one object with another with the same external interface but different implementation is only safe if the external "clients" are not allowed to peek at the implementation. When these interface boundaries are not enforced by the formal system, extensional reasoning becomes an unsafe operation, and must be justified on a case-by-case basis by ensuring that no parts of the construction depend on the implementation details. An example of this scenario is reasoning about isomorphic sets in the context of set theory: while it is often desirable to consider isomorphic sets as extensionally equal and therefore identical and replaceable with one another, there is nothing in the formal rules of set theory preventing the objects relying on these sets to look at their implementation, and potentially distinguishing between two isomorphic ones.

Arguably, the point of *types* is to establish and enforce contracts and layers of abstraction. It wouldn't be surprising then to expect dependent type theory to support various forms of extensional reasoning, since the infrastructure is already there. As it turns out, type theory does in fact have the *potential* for extensional reasoning, as it is not possible to construct type-theoretic terms that distinguish between extensionally equivalent structures like point-wise equal functions or isomorphic sets and groups. Alas, "vanilla" type theory does not allow us to internally prove two equivalent structures equal; we thus have the potential for extensional reasoning, but no way to make actual use of it. Historically, this aspect has been a hindrance to the use of type theory for formalizing mathematics: even basic extensionality principles like function extensionality had to be added as postulates, or via alternative means.

Setoids, i.e. sets (or types) equipped with an equivalence relation, have found widespread application in type theory precisely as a tool to overcome its limitations with respect to extensional reasoning. Setoids form a model of type theory, where contexts/closed types are interpreted as setoids, and dependent types are interpreted as dependent/indexed setoids. A first setoid model for intensional type theory was presented by M. Hofmann [Hof95], with the goal to provide a semantics for extensionality principles such as function extensionality and propositional extensionality. Alas, that version of the setoid model did not allow for an interpretation of universes.

In this thesis we will be focusing on a setoid model construction due to Altenkirch [Alt99], which we refer to as *the strict setoid model.* The metatheory used is an extension of a very minimal intensional Martin-Löf Type Theory with a definitionally proof-irrelevant universe of propositions, as well as η-rules for Π-types and Σ-types. Notably, we do not require inductive types nor identity types. The resulting setoid model validates all the expected type formers and equalities, in addition to function extensionality, a universe of propositions with propositional extensionality, and quotient types. Moreover, it is a *strict* model, in the sense that all the model equations hold by definition in the metatheory.

The strict setoid model provides a way to bootstrap and "explain" extensionality, since the construction effectively gives an implementation of various extensionality principles in terms of a core intensional theory. For the same reason, it also appears to be an ideal starting point to design a type theory inspired by it — a sort of synthetic theory of setoids — where principles like *funext* are valid and computational. Setoid Type Theory (SeTT) is a recently developed formal system derived from this model construction [ABKT19]. Observational Type Theory (OTT) [AMS07] is a syntax for the strict setoid model differing from SeTT in the use of a different notion of heterogeneous equality. Moreover, the consistency proof for OTT relies on Extensional Type Theory, whereas for SeTT it is obtained via a syntactic translation. XTT [SAG19] is a cubical variant of OTT where the equality type is defined using an interval pretype.

One important component that is missing from SeTT is a universe of setoids, allowing for a proper internalization of the notion of type (i.e. setoid). In this part of the thesis, we will present work towards extending SeTT with a universe of setoids. More specifically we will tackle the first necessary step, which is to extend the semantic foundation of SeTT, i.e. the strict setoid model, with such a universe construction. After describing the metatheory that we will use throughout this part of the thesis (Section 5.1), we briefly revisit Altenkirch's strict setoid model [ABKT19] in the framework of Categories with Families (Section 5.2). We then quickly discuss the rules of Setoid Type Theory (Section 5.3). The chapter that follows will describe our novel extension of the strict setoid model with a universe of setoids.

5.1 MLTT$^{\mathbf{Prop}}$

This section describes $\mathsf{MLTT}^{\mathbf{Prop}}$, the ambient metatheory where the model construction will take place. We employ Agda notation to write down $\mathsf{MLTT}^{\mathbf{Prop}}$ terms throughout this part of the thesis.

One of the main appeals of Altenkirch's setoid model is that it can justify several useful extensionality principles while being defined in a minimal intensional metatheory. We tried to stay true to this idea when figuring out the necessary metatheoretical tools for the universe construction in this thesis. In particular, we wanted to avoid having to assume strong definition schemas that go beyond inductive families. $\mathsf{MLTT}^{\mathbf{Prop}}$ is thus a type theory in the style of intensional Martin-Löf type theory with some extra components.

We have sorts $\mathbf{Type}_i$ of types and $\mathbf{Prop}_i$ of strict propositions for $i \in \{0,1\}$. Here, $i = 0$ means "small" (and we will omit the subscript) and $i = 1$ means "large". We have implicit lifting from $i = 0$ to $i = 1$, but do not assume type formers are preserved. $\mathbf{Type}_1$ has universes for $\mathbf{Type}$ and $\mathbf{Prop}$. We do not distinguish notationally between universes and sorts. We continue to describe only the case $i = 0$; everything introduced has an analogue at level $i = 1$. Propositions lift to types via $\mathsf{Lift} : \mathbf{Prop} \to \mathbf{Type}$, with constructor $\mathsf{lift} : \{P : \mathbf{Prop}\} \to P \to \mathsf{Lift}\ P$ and destructor $\mathsf{unlift} : \{P : \mathbf{Prop}\} \to \mathsf{Lift}\ P \to P$.

We have standard type formers $\Pi, \Sigma, \mathsf{Bool}, \mathbf{0}, \mathbf{1}$ in Type. Σ-types are defined negatively by pairing $\_,\_$ and projections π_1, π_2. We have definitional η-rules for Π-, Σ-, $\mathbf{1}$-types. We also require indexed W-types, both in $\mathbf{Type}$ and $\mathbf{Prop}$: $W_\square : (S : I \to \mathbf{Type}) \to ((i : I) \to S\ i \to I \to \mathbf{Type}) \to I \to \square$ where $\square \in \{\mathbf{Type}, \mathbf{Prop}\}$. The elimination principle of $W_{\mathbf{Prop}}$ only allows defining functions into elements of $\mathbf{Prop}$. From $W_{\mathbf{Prop}}$ we can define a strict truncation operator (also known as *squash*) $\|\_\| : \mathbf{Type} \to \mathbf{Prop}$, with constructor $|\_| : \{A : \mathbf{Type}\} \to A \to \|A\|$ and eliminator $\mathsf{elim}_{\|\_\|} : \{P : \mathbf{Prop}\} \to (A \to P) \to \|A\| \to P$.

In addition to type formers in $\mathbf{Type}$, we will need the propositional versions of $\mathbf{0}$, $\mathbf{1}$, Π, and Σ. The latter three can be defined from their $\mathbf{Type}$ counterparts via truncation. That is, given $P : \mathbf{Prop}$ and $Q : P \to \mathbf{Prop}$:

$$\mathbf{1}_{\mathbf{Prop}} :\equiv \|\mathbf{1}\|$$

$$\begin{aligned}
&\Pi_{\mathbf{Prop}}\ P\ Q :\equiv \|\Pi\ (\mathsf{Lift}\ P)\ (\mathsf{Lift} \circ Q \circ \mathsf{unlift})\| \\
&\Sigma_{\mathbf{Prop}}\ P\ Q :\equiv \|\Sigma\ (\mathsf{Lift}\ P)\ (\mathsf{Lift} \circ Q \circ \mathsf{unlift})\|
\end{aligned}$$

We assume that we have $\mathbf{0}_{\mathbf{Prop}} : \mathbf{Prop}$ together with $\mathsf{exfalso}_{\mathbf{Prop}} : \{A : \mathbf{Type}\} \to \mathbf{0}_{\mathbf{Prop}} \to A$.

We use Agda-style notation, with the notational conventions described in Section 3.2, Remark 3.2.1. We will also often omit implicit quantifications, thus for instance write $F\ x \to G\ x$ instead of $\forall\{x\} \to F\ x \to G\ x$, whenever the nature of x as implicitly quantified is clear from context.

5.2 Strict setoid model

The strict setoid model can be framed categorically as a CwF with extra structure for the various type and term formers. Recall the core structure of a CwF:

$$\begin{aligned}
&\mathsf{Con} : \mathbf{Type} \\
&\mathsf{Ty}\ \ : (\Gamma : \mathsf{Con}) \to \mathbf{Type} \\
&\mathsf{Sub} : (\Gamma\ \Delta : \mathsf{Con}) \to \mathbf{Type} \\
&\mathsf{Tm}\ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathbf{Type}
\end{aligned}$$

Our contexts are setoids, that is, types together with an equivalence relation. A key point of the model is that the equivalence relation is valued in **Prop** and is thus definitionally proof-irrelevant. We define a semantic context/setoid $\Gamma : \mathsf{Con}$ as the following record type:

$$\begin{aligned}
&|\Gamma| && : \mathbf{Type} \\
&\Gamma^{\sim} && : |\Gamma| \to |\Gamma| \to \mathbf{Prop} \\
&\mathsf{refl}\ \Gamma && : (\gamma : |\Gamma|) \to \Gamma^{\sim}\ \gamma\ \gamma \\
&\mathsf{sym}\ \Gamma && : \forall\{\gamma_0\ \gamma_1\} \to \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to \Gamma^{\sim}\ \gamma_1\ \gamma_0 \\
&\mathsf{trans}\ \Gamma && : \forall\{\gamma_0\ \gamma_1\ \gamma_2\} \to \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to \Gamma^{\sim}\ \gamma_1\ \gamma_2 \to \Gamma^{\sim}\ \gamma_0\ \gamma_2
\end{aligned}$$

Types in a context Γ are given by displayed setoids over Γ with a fibration condition represented by coe and coh. Given $\Gamma : \mathsf{Con}$, we define a semantic type/indexed setoid $A : \mathsf{Ty}\ \Gamma$ as the following record type:

$$\begin{aligned}
&|A| && : |\Gamma| \to \mathbf{Type} \\
&A^{\sim} && : \forall\ \{\gamma_0\ \gamma_1\} \to \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to |A|\gamma_0 \to |A|\gamma_1 \to \mathbf{Prop} \\
&\mathsf{refl}^* && : (a : |A|\ \gamma) \to A^{\sim}\ (\mathsf{refl}\ \Gamma\ \gamma)\ a\ a \\
&\mathsf{sym}^* && : \{p : \Gamma^{\sim}\ \gamma_0\ \gamma_1\} \to A^{\sim}\ p\ a_0\ a_1 \to A^{\sim}\ (\mathsf{sym}\ \Gamma\ p)\ a_1\ a_0
\end{aligned}$$

$$\begin{aligned}
&\mathsf{trans^*} : A^{\sim}\ p_0\ a_0\ a_1 \to A^{\sim}\ p_1\ a_1\ a_2 \to A^{\sim}\ (\mathsf{trans}\,\Gamma\,p_0\,p_1)\ a_0\ a_2\\
&\mathsf{coe}\quad\ : \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to |A|\gamma_0 \to |A|\gamma_1\\
&\mathsf{coh}\quad\ : (p : \Gamma^{\sim}\ \gamma_0\ \gamma_1)(a : |A|\gamma_0) \to A^{\sim}\,p\,a\,(\mathsf{coe}\,A\,p\,a)
\end{aligned}$$

This definition of types in the setoid model is different from the one in [Alt99], but it is equivalent to it [Bou18, Section 1.6.1]. The main difference here is in the use of a heterogeneous equivalence relation $A^{\sim}$ in the definition of types.

Substitutions are interpreted as functors between the corresponding setoids, whereas terms of type A in context Γ are sections of the type seen as a setoid fibration $\Gamma.A \to \Gamma$. Note that we only need to include components for the functorial action on objects and morphisms, since the functor laws hold by definition from proof-irrelevance in the metatheory.

We interpret substitutions $\sigma : \mathsf{Sub}\ \Gamma\ \Delta$ and terms $t : \mathsf{Tm}\ \Gamma\ A$ as the following record types:

$$\begin{aligned}
&|\sigma| : |\Gamma| \to |\Delta| && |t| : (\gamma : |\Gamma|) \to |A|\,\gamma\\
&\sigma^{\sim} : \Gamma^{\sim}\ \rho_0\,\rho_1 \to \Delta^{\sim}\ (|\sigma|\rho_0)\ (|\sigma|\rho_1) && t^{\sim} : (p : \Gamma^{\sim}\ \gamma_0\ \gamma_1) \to A^{\sim}\ p\ (|t|\gamma_0)\ (|t|\gamma_1)
\end{aligned}$$

5.3 Setoid Type Theory

The setoid model presented in the previous section is *strict*, that is, every equation of the CwF structure holds by definition. One advantage of strict models is that they can be turned into *syntactic translations*, in which the syntactic objects being modeled are interpreted as their counterparts in another *target* theory. In the case of the setoid model, this gives rise to a *setoid translation*, where source contexts are interpreted as target contexts together with a target type representing the equivalence relation, and so on.

A setoid translation is used in [ABKT19] to justify Setoid Type Theory (SeTT), an extension of intensional MLTT with equality types for contexts and dependent types that reflect the setoid equality of the model.

We recall the rules of SeTT that extend regular MLTT below.

We have a universe of propositions Prop defined as follows:

$$\frac{\Gamma : \mathsf{Con}}{\mathsf{Prop} : \mathsf{Ty}\ \Gamma} \qquad \frac{P : \mathsf{Tm}\ \Gamma\ \mathsf{Prop}}{\underline{P} : \mathsf{Ty}\ \Gamma} \qquad \frac{u : \mathsf{Tm}\ \Gamma\ \underline{P} \qquad v : \mathsf{Tm}\ \Gamma\ \underline{P}}{u = v}$$

Equality type constructors for contexts and dependent types internalize the idea that every context and type comes equipped with a setoid equivalence relation.

As in the model, equality for dependent types is indexed over context equality.

$$\frac{\Gamma : \mathsf{Con} \qquad \rho_0, \rho_1 : \mathrm{Sub}\ \Delta\ \Gamma}{\Gamma^{\sim}\ \rho_0\ \rho_1 : \mathsf{Tm}\ \Delta\ \mathsf{Prop}} \qquad \frac{\begin{array}{c} A : \mathsf{Ty}\ \Gamma \qquad \rho_{01} : \mathsf{Tm}\ \Delta\ \underline{\Gamma^{\sim}\ \rho_0\ \rho_1} \\ a_0 : \mathsf{Tm}\ \Delta\ A[\rho_0] \qquad a_1 : \mathsf{Tm}\ \Delta\ A[\rho_1] \end{array}}{A^{\sim}\ \rho_{01}\ a_0\ a_1 : \mathsf{Tm}\ \Delta\ \mathsf{Prop}}$$

We have rules witnessing that these are indeed equivalence relations. We only recall reflexivity:

$$\frac{\rho : \mathsf{Sub}\ \Delta\ \Gamma}{\mathsf{R}\ \rho : \mathsf{Tm}\ \Delta\ \underline{\Gamma^{\sim}\ \rho\ \rho}} \qquad \frac{A : \mathsf{Ty}\ \Gamma \qquad \rho : \mathsf{Sub}\ \Delta\ \Gamma \qquad a : \mathsf{Tm}\ \Delta\ A[\rho]}{\mathsf{R}\ a : \mathsf{Tm}\ \Gamma\ \underline{A^{\sim}\ (\mathsf{R}\ \rho)\ a\ a}}$$

We also have rules representing the fact that every construction in SeTT respects setoid equality, so that we can transport along any such equality:

$$\frac{A : \mathsf{Ty}\ \Gamma \qquad \rho_0, \rho_1 : \mathrm{Sub}\ \Delta\ \Gamma \qquad p : \mathsf{Tm}\ \Delta\ \underline{\Gamma^{\sim}\ \rho_0\ \rho_1} \qquad a : \mathsf{Tm}\ \Delta\ A[\rho_0]}{\begin{array}{c} \mathsf{coe}_A\ p\ a : \mathsf{Tm}\ \Delta\ A[\rho_1] \\ \mathsf{coh}_A\ p\ a : \mathsf{Tm}\ \Delta\ \underline{A^{\sim}\ p\ a\ (\mathsf{coe}_A\ p\ a)} \end{array}}$$

Notably, equality types in SeTT compute definitionally on concrete type formers. In particular, they compute to their obvious intended meaning, so that an equality of pairs is a pair of equalities, an equality of functions is a map of equalities, and so on. From this, we get definitional versions of function and propositional extensionality:

$$\begin{aligned} &(\Pi(a : A)\ B)^{\sim}\ p\ f_0\ f_1 = \Pi(a_0\, a_1 : A)\Pi(q : A^{\sim}\ p\ a_0\ a_1)(B^{\sim}\ (p, q)\ (f_0\ a_0)\ (f_1\ a_1)) \\ &(\Sigma\ A\ B)^{\sim}\ p\ (a_0, b_0)\ (a_1, b_1) = \Sigma\ (q : A^{\sim}\ p\ a_0\ a_1)(B^{\sim}\ (p, q)\ b_0\ b_1) \\ &\mathsf{Prop}^{\sim}\ p\ P\ Q = (P \Rightarrow Q) \times (Q \Rightarrow P) \end{aligned}$$

where we write $\Rightarrow$ and $\times$ for the non-dependent version of Π and Σ in SeTT.

We can easily recover the usual Martin-Löf identity type from setoid equality, with transport implemented via coercion.

$$\frac{A : \mathsf{Ty}\ \Gamma \qquad a_0, a_1 : \mathsf{Tm}\ \Gamma\ A}{\mathsf{Id}_A\ a_0\ a_1 :\equiv A^{\sim}\ (\mathsf{R}\ \Gamma)\ a_0\ a_1 : \mathsf{Tm}\ \Gamma\ \mathsf{Prop}}$$

$$\frac{P : \mathsf{Ty}\ (\Gamma.A) \qquad p : \mathsf{Tm}\ \Gamma\ \underline{\mathsf{Id}_A\ a_0\ a_1} \qquad t : \mathsf{Tm}\ \Gamma\ P[a_0]}{\mathsf{transp}\ P\ p\ t :\equiv \mathsf{coe}\ P\ (\mathsf{R}\ \mathsf{id}, p)\ t : \mathsf{Tm}\ \Gamma\ P[a_1]}$$

We can also derive Martin-Löf's J eliminator for this homogeneous identity type. The only caveat is that transp and the J eliminator do not compute definitionally on reflexivity, although the rules can be consistently added as shown in [ABKT19].

Chapter 6

The setoid universe

Setoid Type Theory as presented in [ABKT19] and summarized in Section 5.3 is limited by the lack of universes internalizing the notion of setoid. In this chapter we will work towards extending SeTT with a universe of setoids; since SeTT is a direct syntactic reflection of the setoid model, this extension requires first and foremost to show that we can extend the setoid model with a semantics for a universe of setoids with the desired structure.

We begin our discussion with various design choices related to the setoid universe (Section 6.1). We then recall inductive-recursive (IR) universes, and the way they can be equivalently defined as a plain inductive definition (Section 6.2). We provide a first complete definition of the setoid universe using a combination of induction-recursion and recursion-recursion (Section 6.2) that we may call induction-recursion-recursion (IRR) as its recursive components are mutually and recursively-recursively defined. While recursion-recursion can be understood as the induction principle of IITs (see Chapter 3), this form of IRR clearly goes beyond standard induction-induction, and does not seem to be an instance of known axiomatizations of IR (such as [Dyb03]); moreover, we are not aware of a general method to reduce it to plain inductive types, like it is the case for small IR. Therefore, while recognizing its quality as a pedagogical device, we will limit our use of IRR to Section 6.2, and move away from induction-recursion and recursion-recursion in Section 6.4 with an alternative definition of the universe that relies instead on infinitary induction-induction. This inductive-inductive encoding of the universe is obtained from the IRR one, following a transformation that is similar to what described in Section 6.2.

We continue the series of universe constructions with a purely inductive definition of the setoid universe. This version of the universe does not rely on IRR or II, and is obtained from the universe IIT from Section 6.5 via a novel application of a known encoding method of finitary IITs in terms of inductive families. To successfully apply this method, we rely on an extension of the metatheory with a definitionally proof-irrelevant identity type with a strong transport rule that allows to eliminate into arbitrary proof-relevant families.

The encoded IIT is powerful enough to support the definition of a setoid universe, even its simple elimination principle alone [ABK$^+$21]. Moreover, it can be equipped with general eliminators, as discussed in Section 6.6, which makes it an equivalent, full encoding of the IIT from Section 6.4. We use the full induction principle to justify universe induction/typecase (Section 6.6.1.)

6.1 Design choices

Extending the strict setoid model with a universe of setoids opens several questions and possible design choices, starting with the very definition of the universe itself: as any type in the setoid model, this universe must be a setoid and thus come equipped with an equivalence relation. However, unlike the universe of propositions, a universe of setoids cannot be univalent, since this would force it to be a groupoid. The obvious choice is therefore to have a non-univalent universe, and instead define the universe's relation so that it reflects a simple syntactic equality of codes rather than setoid equivalence.

Another question has to do with the metatheoretic tools required to carry out the construction of the universe. In fact, one of the main aspects of the strict setoid model is that it can be carried out in a very minimal type theoretic metatheory, thus providing a way to reduce extensionality to a minimal intensional core. We would like to stay faithful to this ideal when constructing this setoid universe.

A known and established method for defining universes in type theory relies on induction-recursion (IR), a definition schema developed by Dybjer [Dyb03, DS99]. Inductive-recursive definitions can be found throughout the literature, from the already mentioned type theoretic universes, including the original formulation à la Tarski by Martin-Löf [Mar84], to metamathematical tools like computability predicates.

Although universe constructions in type theory—including our own setoid universe—are naturally presented as inductive-recursive definitions, they may not necessarily require a metatheory with a general definition schema for IR. In fact, it is possible to reduce some instances of IR to plain induction (more specifically, inductive families), including some universe definitions. We recall this reduction in Section 6.2.

Other design choices on the setoid universe are less essential, but still require careful consideration. For instance, one question is whether the setoid universe should be equipped with universe induction, exposing the inductive structure of the codes. Universe induction [NPS90], aka *type-case*, is known to clash with extensionality principles like univalence and parametricity. Nevertheless, we would like to have it as an option, especially in the context of SeTT where univalence is already precluded in any case. Although the published version of the setoid universe did not come with universe induction [ABK+21], in this thesis we include an alternative formulation that does (Section 6.6.1.)

Another design choice has to do with how the setoid universe relates to the other universes. One could provide a code for Prop in the setoid universe. Moreover, the setoid universes could form a hierarchy, possibly cumulative.

Yet another choice is whether to have two separate sorts, one for propositions and one for sets (with propositions convertible to sets) or a single sort of types (sets), with propositions given by elements of a universe of propositions, which is a (large) type. We have chosen to present the second option to fit with the standard notion of (unisorted) CwF. However, this has downsides: to even talk about propositions, we need to have a notion of large types. The first option is more symmetric: we can have parallel hierarchies for propositions and sets.

6.2 Inductive-recursive universes

An inductive-recursive universe is given by a type of codes $\mathsf{U} : \mathbf{Type}$, and a family $\mathsf{El} : \mathsf{U} \to \mathbf{Type}$ that assigns, to each code corresponding to some type, the metatheoretic type of its elements. The resulting definition is inductive-recursive because the inductive type of codes is defined simultaneously with the recursive function El.

An example is the following definition of a small universe with booleans and dependent function types:

$$\begin{array}{ll}
\mathsf{data}\ \mathsf{U}\ : \mathbf{Type} & \mathsf{El}\ : \mathsf{U} \to \mathbf{Type} \\
\quad \mathsf{bool} : \mathsf{U} & \mathsf{El}\ \mathsf{bool} :\equiv \mathbb{2} \\
\quad \mathsf{pi} : (A : \mathsf{U}) \to (\mathsf{El}\ A \to \mathsf{U}) \to \mathsf{U} & \mathsf{El}\ (\mathsf{pi}\ A\ B) :\equiv (a : \mathsf{El}\ A) \to \mathsf{El}\ (B\ a)
\end{array}$$

Induction-recursion is arguably a natural way to define internal universes in type theory, althought it may not always be required. We can translate basic instances of IR into inductive families using the equivalence of I-indexed families of types and types over I (that is, $A : \mathbf{Type}$ with $A \to I$) [MAG+13]. However, this equivalence is not exact, and only applies when the result of the construction A is allowed to be larger than the index I, as also noted in [Cap04].

In the case of our simple example, we can encode U as an inductive type inU that *carves out* all types in $\mathbf{Type}$ that are in the image of El. In other words, inU is a predicate that holds for any type that would have been obtained via El in the inductive-recursive definition. As El is indexed by the type of codes, the definition of inU reflects the inductive structure of codes.

$$\begin{aligned}
&\mathsf{data\ inU} : \mathbf{Type} \to \mathbf{Type}_1\\
&\quad \mathsf{inBool} : \mathsf{inU}\ \mathbb{2}\\
&\quad \mathsf{inPi} \quad : \mathsf{inU}\ A \to ((a : A) \to \mathsf{inU}\ (B\ a)) \to \mathsf{inU}\ ((a : A) \to (B\ a))
\end{aligned}$$

U and El can be given by $\mathsf{U} :\equiv \Sigma\ (A : \mathbf{Type})\ (\mathsf{in\text{-}U}\ A)$ and $\mathsf{El} :\equiv \pi_1$.

Note that this construction gives rise to a universe in $\mathbf{Type}_1$, rather than $\mathbf{Type}$, since the definition of U quantifies over all possible types in $\mathbf{Type}$. It follows that this kind of construction requires a metatheory with at least one universe.

6.3 Inductive-recursive setoid universe

In this section we give a first definition of the setoid universe, as a direct generalization of the simple IR definition just shown. We only consider a very small universe with bool type $\mathbb{2}$ and Π for simplicity; a more realistic universe that includes more type formers can be found in the Agda formalization [Ses23].

To construct the universe of setoids in the setoid model, we first of all need to define a type $\mathbb{U} : \mathsf{Ty}\ \Gamma$ for every $\Gamma : \mathsf{Con}$, and for every $A : \mathsf{Tm}\ \Gamma\ \mathbb{U}$ a type $\mathbb{E}l\ A : \mathsf{Ty}\ \Gamma$. Recalling Section 5.2, these are essentially record types made of several components. Since $\mathbb{U}$ is a closed type, it requires the same data of a setoid; in particular, we need a type of codes together with an equivalence relation reflecting equality of codes, in addition to proofs that these are indeed equivalence relations:

$$\begin{aligned}
&\mathsf{data}\ \mathcal{U} && : \mathbf{Type}_1 && \mathsf{refl}_{\mathcal{U}} && : (A : \mathcal{U}) \to A \sim_{\mathcal{U}} A\\
& && && \mathsf{sym}_{\mathcal{U}} && : A \sim_{\mathcal{U}} B \to B \sim_{\mathcal{U}} A\\
&\_ \sim_{\mathcal{U}} \_ && : \mathcal{U} \to \mathcal{U} \to \mathbf{Prop}_1 && \mathsf{trans}_{\mathcal{U}} && : A \sim_{\mathcal{U}} B \to B \sim_{\mathcal{U}} C \to A \sim_{\mathcal{U}} C
\end{aligned}$$

$\mathbb{E}l$ is given by a family of setoids indexed over the universe, that is, a way to assign to each code in the universe a carrier set and an equivalence relation.

$$\begin{aligned}
&\mathcal{E}l && : \mathcal{U} \to \mathbf{Type}\\
&\_ \vdash \_ \sim_{\mathcal{E}l} \_ && : \{a\ a' : \mathcal{U}\} \to a \sim_{\mathcal{U}} a' \to \mathcal{E}l\ a \to \mathcal{E}l\ a' \to \mathbf{Prop}
\end{aligned}$$

Note that $\_ \vdash \_ \sim_{\mathcal{E}l} \_$ is indexed over equality on the universe, because $\mathcal{E}l$ is a displayed setoid over $\mathcal{U}$, hence in particular it must respect the setoid equality of $\mathcal{U}$. We also require data and proofs that make sure we get setoids out of $\mathcal{E}l$:

$$\begin{aligned}
&\mathsf{refl}_{\mathcal{E}l} && : (A : \mathcal{U})(x : \mathcal{E}l\ A) \to \mathsf{refl}_{\mathcal{U}}\ A \vdash x \sim_{\mathcal{E}l} x\\
&\mathsf{sym}_{\mathcal{E}l} && : p \vdash x \sim_{\mathcal{E}l} x' \to \mathsf{sym}_{\mathcal{U}}\ p \vdash x' \sim_{\mathcal{E}l} x\\
&\mathsf{trans}_{\mathcal{E}l} && : p \vdash x \sim_{\mathcal{E}l} x' \to q \vdash x' \sim_{\mathcal{E}l} x'' \to \mathsf{trans}_{\mathcal{U}}\ p\ q \vdash x \sim_{\mathcal{E}l} x''
\end{aligned}$$

$$\begin{array}{l}
\mathsf{data}\ \mathcal{U} : \mathbf{Type}_1 \\
\quad \mathsf{bool} : \mathcal{U} \\
\quad \mathsf{pi} : (A : \mathcal{U})(B : \mathcal{E}l\ A \to \mathcal{U}) \\
\qquad \to (\{x\ x' : \mathcal{E}l\ A\} \to \mathsf{refl}_{\mathcal{U}}\ A \vdash x \sim_{\mathcal{E}l} x' \\
\qquad\quad \to B\ x \sim_{\mathcal{U}} B\ x') \to \mathcal{U}
\end{array}$$

$$\begin{array}{l}
\mathsf{bool} \sim_{\mathcal{U}} \mathsf{bool} :\equiv \top \\
\mathsf{pi}\, A\, B\, p \sim_{\mathcal{U}} \mathsf{pi}\, A'\, B'\, p' :\equiv \\
\quad \Sigma(a^{\sim} : A \sim_{\mathcal{U}} A') \\
\qquad (\forall\, x\, x', a^{\sim} \vdash x \sim_{\mathcal{E}l} x' \to B\, x \sim_{\mathcal{U}} B'\, x')
\end{array}$$

$$\begin{array}{l}
\mathsf{refl}_{\mathcal{U}}\, \mathsf{bool} :\equiv \mathsf{tt} \\
\mathsf{refl}_{\mathcal{U}}\, (\mathsf{pi}\, A\, B\, p^{\sim}) :\equiv \mathsf{refl}_{\mathcal{U}}\, A, \lambda\, \_\, \_\, \_\, .\, p^{\sim}
\end{array}$$

$$\begin{array}{l}
\mathcal{E}l\ \mathsf{bool} :\equiv \mathbb{2} \\
\mathcal{E}l\ (\mathsf{pi}\ A\ B\ h) :\equiv \\
\quad \Sigma\ (f : (a : \mathcal{E}l\ A) \to \mathcal{E}l\ (B\ a)) \\
\qquad (\forall \{x\ x'\}(p : \mathsf{refl}_{\mathcal{U}}\ A \vdash x \sim_{\mathcal{E}l} x') \\
\qquad\ \to h\ p \vdash f\ x \sim_{\mathcal{E}l} f\ x')
\end{array}$$

$$\begin{array}{ll}
a^{\sim} \vdash x_0 \sim_{\mathcal{E}l} x_0 :\equiv x_0 \overset{?}{=}_{\mathbb{2}} x_1 & (\mathsf{bool}\ \text{case}) \\
(a^{\sim}, b^{\sim}) \vdash (f_0, \_) \sim_{\mathcal{E}l} (f_1, \_) :\equiv & (\mathsf{pi}\ \text{case}) \\
\quad \forall\, x_0\, x_1\, (x^{\sim} : a^{\sim} \vdash x_0 \sim_{\mathcal{E}l} x_1), & \\
\qquad b^{\sim}\, \_\, \_\, x^{\sim} \vdash f_0\, x_0 \sim_{\mathcal{E}l} f_1\, x_1 &
\end{array}$$

Figure 6.1: Core definitions for the inductive-recursive setoid universe

$$\begin{array}{ll}
\mathsf{coe}_{\mathcal{E}l} & : A \sim_{\mathcal{U}} B \to \mathcal{E}l\ A \to \mathcal{E}l\ B \\
\mathsf{coh}_{\mathcal{E}l} & : (p : A \sim_{\mathcal{U}} A')\ (x : \mathcal{E}l\ A) \to p \vdash x \sim_{\mathcal{E}l} \mathsf{coe}_{\mathcal{E}l}\ p\ x
\end{array}$$

We give an inductive definition of $\mathcal{U}$, mutually with a recursive definition of the 4 functions $\_ \sim_{\mathcal{U}} \_$, $\mathsf{refl}_{\mathcal{U}}$, $\mathcal{E}l$ and $\_ \vdash \_ \sim_{\mathcal{E}l} \_$ (Figure 6.1). Note that $\sim_{\mathcal{U}}$ appears in the type of $\mathsf{refl}_{\mathcal{U}}$, thus making this definition *inductive-recursive-recursive.* We write $\_ \overset{?}{=}_{\mathbb{2}} \_$ for the **Prop**-valued predicate of decidable equality between booleans.

The remaining functions are then recursively defined: $\mathsf{refl}_{\mathcal{E}l}$ by itself, then $\mathsf{sym}_{\mathcal{U}}$ and $\mathsf{sym}_{\mathcal{E}l}$ by mutual recursion-recursion, and finally $\mathsf{trans}_{\mathcal{U}}$, $\mathsf{trans}_{\mathcal{E}l}$, $\mathsf{coe}_{\mathcal{E}l}$ and $\mathsf{coh}_{\mathcal{E}l}$ again by mutual recursion-recursion. While $\mathsf{refl}_{\mathcal{E}l}$, $\mathsf{sym}_{\mathcal{U}}$ and $\mathsf{sym}_{\mathcal{E}l}$ are fairly straightforward, the other definitions are quite technical. We point the reader to the Agda formalization [Ses23] for the gritty details.

Note that in the definition of $\mathcal{U}$ we require that the family $B : \mathcal{E}l\ A \to \mathcal{U}$ be a setoid morphism, respecting the setoid equalities involved. This choice is crucial for the definition of $\mathcal{E}l$ to go through, in particular since we eliminate the code for Π-types into the setoid of functions that map equal elements to equal results. To state this mapping property we need to compare elements in different types, coming from applying f to different arguments x and x'. We know that x and x' are equal, but to conclude $B\ x \sim_{\mathcal{U}} B\ x'$ we need to know that B respects setoid equality. This is exactly what we get from our definition of $\mathcal{U}$.

We give a full definition of the setoid universe, and of $\mathbb{E}l\ A$ for any $A : \mathsf{Tm}\ \Gamma\ \mathbb{U}$,

$$\begin{array}{llll}
|\mathbb{U}| & :\equiv \lambda\,\gamma.\,\mathcal{U} & |\mathbb{E}l\ A| & :\equiv \lambda\,\gamma.\,\mathcal{E}l\ (|A|\,\gamma) \\
\mathbb{U}^{\sim} & :\equiv \lambda\,p\,x\,y.\,x \sim_{\mathcal{U}} y & (\mathbb{E}l\ A)^{\sim} & :\equiv \lambda\,p\,x\,y.\,A^{\sim}\,p \vdash x \sim_{\mathcal{E}l} y \\
\mathsf{refl}\ \mathbb{U} & :\equiv \mathsf{refl}_{\mathcal{U}} & \mathsf{refl}\ (\mathbb{E}l\,A) & :\equiv \mathsf{refl}_{\mathcal{E}l} \\
\mathsf{sym}\ \mathbb{U} & :\equiv \mathsf{sym}_{\mathcal{U}} & \mathsf{sym}\ (\mathbb{E}l\,A)\,\{\gamma_{01}\} & :\equiv \mathsf{sym}_{\mathcal{E}l}\{p = t^{\sim}\,a\,\gamma_{01}\} \\
\mathsf{trans}\ \mathbb{U} & :\equiv \mathsf{trans}_{\mathcal{U}} & \mathsf{trans}\ (\mathbb{E}l\,A)\,\{\gamma_{01}\,\gamma_{12}\} & :\equiv \mathsf{trans}_{\mathcal{E}l}\,\{p = t^{\sim}\,a\,\gamma_{01}\}\{t^{\sim}\,a\,\gamma_{12}\} \\
\mathsf{coe}\ \mathbb{U} & :\equiv \lambda\,p\,a.\,a & \mathsf{coe}\ (\mathbb{E}l\ A) & :\equiv \lambda\,p.\,\mathsf{coe}_{\mathcal{E}l}\,(A^{\sim}\,p) \\
\mathsf{coh}\ \mathbb{U} & :\equiv \lambda\,p.\,\mathsf{refl}_{\mathcal{U}} & \mathsf{coh}\ (\mathbb{E}l\ A) & :\equiv \lambda\,p.\,\mathsf{coh}_{\mathcal{E}l}\,(A^{\sim}\,p)
\end{array}$$

Figure 6.2: Inductive-recursive setoid universe

in Figure 6.2.

We can show that $\mathbb{U}$ is *judgmentally* closed under dependent function types and booleans. The universe can be closed under more constructions if more codes are added to $\mathcal{U}$. This gives a complete definition of a universe of setoids; however, the full definition is inductive-recursive, and some of its recursive components depend on each other in a recursive-recursive way. This combination of IR and recursion-recursion is not obviously reducible to well-understood axiomatizations of IR like the one described in [DS99]. In any case, we would like to avoid extending the metatheory with general definition schemas for induction-recursion-recursion (IRR), in order to keep the metatheory as minimal as possible.

In the next section we will transform our current universe definition to one that does not use IR of any form. The reduction is inspired by the well-known trick to eliminate IR described in Section 6.2, but modified in a novel way to account for the presence of **Prop**-valued types and recursion-recursion. To our knowledge, this is the first time this reduction method is applied to an IRR type of this kind.

6.4 Inductive-inductive setoid universe

We follow the method outlined in Section 6.2, and apply it to our current IRR definition of the setoid universe components, i.e. $\mathcal{U}, \mathcal{E}l$, etc. As the reduced IRR type includes recursion-recursion, the result of this reduction is an *inductive-inductive type*. Specifically, in addition to inU for defining $\mathcal{U}$, we also have a family $\mathsf{inU}\!\sim$ of binary relations between types in the universe, from which we then define $\_ \sim_{\mathcal{U}} \_$. The full definition of this universe IIT is given in Figure 6.3.

Just as the role of inU is, as before, to classify all types that are image of $\mathcal{E}l$, in the same way $\mathsf{inU}\!\sim\ a\ a'$ classifies all relations of type $A \to A' \to \mathbf{Prop}$ that are image of $\_ \vdash \_ \sim_{\mathcal{E}l} \_$, given proofs $a : \mathsf{inU}\ A, a' : \mathsf{inU}\ A'$. In particular, this definition of $\mathsf{inU}\!\sim$ states that the appropriate equivalence for boolean elements is the obvious

```
data inU : Type → Type₁
    bool : inU 𝟚
    pi : (a : inU A) → inU∼ a a A₀₁ → (b : (x : A) → inU (B x))
      → (∀{x₀ x₁}(x₀₁ : A₀₁ x₀ x₁) → inU∼ (b x₀) (b x₁) (B₀₁ x₀₁))
      → inU (Σ (f : (x : A) → B x)
                ((x₀ x₁ : A)(x₀₁ : A₀₁ x₀ x₁) → B₀₁ x₀₁ (f x₀) (f x₁)))

data inU∼ : {A A′ : Type} → inU A → inU A′ → (A → A′ → Prop) → Type₁
    bool∼ : inU∼ bool bool (_ =?₂ _)
    pi∼ :  {b₀ : (x₀ : A₀) → inU (B₀ x₀)}{b₁ : (x₁ : A₁) → inU (B₁ x₁)}
           {a₀₀ : inU∼ a₀ a₀ A₀₀}{a₁₁ : inU∼ a₁ a₁ A₁₁}
           {b₀₀ : ∀{x₀ x₁}(x₀₁ : A₀₀ x₀ x₁) → inU∼ (b₀ x₀) (b₀ x₁) (B₀₀ x₀₁)}
           {b₁₁ : ∀{x₀ x₁}(x₀₁ : A₁₁ x₀ x₁) → inU∼ (b₁ x₀) (b₁ x₁) (B₁₁ x₀₁)}
       → inU∼ a₀ a₁ A₀₁
       → (∀{x₀ x₁}(x₀₁ : A₀₁ x₀ x₁) → inU∼ (b₀ x₀) (b₁ x₁) (B₀₁ x₀₁))
       → inU∼ (pi a₀ a₀₀ b₀ b₀₀) (pi a₁ a₁₁ b₁ b₁₁)
                (λf₀ f₁ . ∀(x₀ x₁) → A₀₁ x₀ x₁ → B₀₁ x₀₁ (π₁ f₀ x₀) (π₁ f₁ x₁))
```

Figure 6.3: Inductive-inductive translation of the setoid universe

syntactic equality $\_ \overset{?}{=}_{2} \_$, whereas functions are to be compared point-wise. Note that inU appears in the sort of inU∼. Since these types are mutually defined, they form an instance of *induction-induction.*

As in the universe example in Section 6.2, we now define $\mathcal{U}$ as a Σ-type, and $\mathcal{El}$ as the corresponding first projection.

$$\begin{aligned} &\mathcal{U} : \mathbf{Type}_1 && \mathcal{El} : \mathcal{U} \to \mathbf{Type} \\ &\mathcal{U} :\equiv \Sigma\ (X : \mathbf{Type})\ (\mathsf{inU}\ X) && \mathcal{El} :\equiv \pi_1 \end{aligned}$$

What is left now is to define the setoid equality relation on the universe, as well as the setoid equality relation on $\mathcal{El}\ A$ for any A in $\mathcal{U}$. Two codes A, B in the universe $\mathcal{U}$ are equal when there exists a setoid equivalence relation on their respective sets $\mathcal{El}\ A$ and $\mathcal{El}\ B$. Intuitively, since elements of a setoid are only ever compared to elements of the same setoid, this should only be possible if A and B are codes for the same setoid, that is, if $A \sim_{\mathcal{U}} B$. Existence and well-formedness of such relations is expressed via the type inU∼ just defined, hence we would expect

$A \sim_{\mathcal{U}} B$ to be defined as follows:

$$(A, a) \sim_{\mathcal{U}} (B, b) :\equiv \Sigma\ (R : A \to B \to \mathbf{Prop})\ (\mathsf{inU}{\sim}\ a\ b\ R)$$

Unfortunately this definition only manages to capture the idea, but does not actually typecheck. In fact, $\_ \sim_{\mathcal{U}} \_$ should be a $\mathbf{Prop}_1$-valued relation, so $A \sim_{\mathcal{U}} B$ should be a proposition. However, the Σ-type shown above clearly is not, since it quantifies over a type of relations, which is not a proposition. One possible solution is actually quite simple, and it just involves truncating the Σ-type above to force it to be in $\mathbf{Prop}_1$.

$$\_ \sim_{\mathcal{U}} \_ : \mathcal{U} \to \mathcal{U} \to \mathbf{Prop}_1$$
$$(A, a) \sim_{\mathcal{U}} (B, b) :\equiv \|\Sigma\ (R : A \to B \to \mathbf{Prop})\ (\mathsf{inU}{\sim}\ a\ b\ R)\|$$

We are now left to define the indexed equivalence relation on $\mathcal{E}l$:

$$\_ \vdash \_ \sim_{\mathcal{E}l} \_ : \{A\ B : \mathcal{U}\} \to A\ \ \sim_{\mathcal{U}}\ \ B \to \mathcal{E}l\ A \to \mathcal{E}l\ B \to \mathbf{Prop}$$
$$p \vdash x \sim_{\mathcal{E}l} y :\equiv\ ?$$

In the definition above, p has type $\|\Sigma\ (R : \mathcal{E}l\, A \to \mathcal{E}l\, B \to \mathbf{Prop})\ (\ldots)\|$. If the type were not propositionally truncated, we could define $p \vdash x \sim_{\mathcal{E}l} y$ by extracting the relation out of the first component of p, and apply it to x, y. That is, $p \vdash x \sim_{\mathcal{E}l} y :\equiv \pi_1\, p\, x\, y$. This would make the definition of $\_ \sim_{\mathcal{U}} \_$ and $\_ \vdash \_ \sim_{\mathcal{E}l} \_$ in line with how we defined $\mathcal{U}$ and $\mathcal{E}l$.

Alas this does not work: since the type of p *is* propositionally truncated, it cannot be eliminated to construct a proof-relevant object. Fortunately, we can work around this limitation by defining $p \vdash x \sim_{\mathcal{E}l} y$ by induction on the codes $A\ B : \mathcal{U}$, in a way that ends up being logically equivalent to the proposition we would have obtained by $\pi_1\ p\ x\ y$ if there were no truncation. More precisely, we need to construct proofs that for any concrete R and inR, the types $|(R, \mathsf{inR})| \vdash x \sim_{\mathcal{E}l} y$ and $R\ x\ y$ are logically equivalent.

$$\mathsf{out} : (a_{01} : \mathsf{inU}{\sim}\ a_0\ a_1\ A_{01}) \to |(A_{01}, a_{01})| \vdash x_0 \sim_{\mathcal{E}l} x_1 \to A_{01}\ x_0\ x_1$$
$$\mathsf{in}\ \ : (a_{01} : \mathsf{inU}{\sim}\ a_0\ a_1\ A_{01}) \to A_{01}\ x_0\ x_1 \to |(A_{01}, a_{01})| \vdash x_0 \sim_{\mathcal{E}l} x_1$$

These in turn need to be defined mutually with $\_ \vdash \_ \sim_{\mathcal{E}l} \_$.

$$\mathsf{out}\ \mathsf{bool}{\sim}\ x_{01} :\equiv x_{01}$$
$$\mathsf{out}\ (\mathsf{pi}{\sim}\ a_{01}\ b_{01})\ f_{01}\ \_\ \_\ x_{01} :\equiv$$
$$\quad \mathsf{out}\ (b_{01}\ x_{01})\ (f_{01}\ \_\ \_\ (\mathsf{in}\ a_{01}\ x_{01}))$$

$$\mathsf{in}\ \mathsf{bool}{\sim}\ x_{01} :\equiv x_{01}$$
$$\mathsf{in}\ (\mathsf{pi}{\sim}\ a_{01}\ b_{01})\ f_{01}\ \_\ \_\ x_{01} :\equiv$$
$$\quad \mathsf{in}\ (b_{01}\ (\mathsf{out}\ a_{01}\ x_{01}))(f_{01}\ \_\ \_\ (\mathsf{out}\ a_{01}\ x_{01}))$$

$$\_ \vdash x_0 \sim_{\mathcal{E}l} x_1 :\equiv \mathsf{if}\ x_0\ \mathsf{then}\ (\mathsf{if}\ x_1\ \mathsf{then}\ \top\ \mathsf{else}\ \bot)\ \mathsf{else}\ (\mathsf{if}\ x_1\ \mathsf{then}\ \bot\ \mathsf{else}\ \top) \quad \text{(bool case)}$$

$$(\_, \mathsf{pi}{\sim}\ a_{01}\ b_{01}) \vdash (f_0, \_) \sim_{\mathcal{E}l} (f_1, \_) :\equiv \qquad \text{(pi case)}$$
$$\forall\, x_0\, x_1\, (x_{01} : |\_, a_{01}| \vdash x_0 \sim_{\mathcal{E}l} x_1)\, .\ |\_, b_{01}\ (\mathsf{out}\ a_{01}\ x_{01})| \vdash (f_0\ x_0) \sim_{\mathcal{E}l} (\ f_1\ x_1)$$

We can prove that any relation constructed via $\sim_{\mathcal{E}l}$ is "valid" as established by inU$\sim$:

$$\begin{aligned}
&\mathsf{inj} : (w : (A_0, a_0) \sim_{\mathcal{U}} (A_1, a_1)) \to \mathsf{inU}{\sim}\ a_0\ a_1\ (\lambda\, x\, y\, .\, w \vdash x \sim_{\mathcal{E}l} y)\\
&\mathsf{inj\ bool\ bool}\ \_ :\equiv \mathsf{bool}{\sim}\\
&\mathsf{inj}\ (\mathsf{pi}\ a_0\ a_{00}\ b_0\ b_{00})\ (\mathsf{pi}\ a_1\ a_{11}\ b_1\ b_{11})\ |\_, \mathsf{pi}{\sim}\ a_{01}\ b_{01}| :\equiv\\
&\qquad \mathsf{pi}{\sim}\ (\mathsf{inj}\ a_0\ a_1\ |\_, a_{01}|)\ (\lambda\, x_{01}\, .\, \mathsf{inj}\ \_\ \_\ |\_, b_{01}\ (\mathsf{out}\ a_{01}\ x_{01})|)
\end{aligned}$$

Remark 6.4.1. We abuse notation and pattern-match on the proof of $\_ \sim_{\mathcal{U}} \_$ in the second clause of $\sim_{\mathcal{E}l}$ and inj; this does not violate consistency as we only use the terms exposed by the pattern to construct other proof-irrelevant terms. ■

The full definition of the universe is concluded with the remaining functions, like $\mathsf{refl}_{\mathcal{U}}$, $\mathsf{refl}_{\mathcal{E}l}$, etc., which can be adapted from their IR counterparts more or less straightforwardly. For example,

$$\begin{aligned}
&\mathsf{refl}_{\mathcal{U}} : (A : \mathcal{U}) \to A \sim_{\mathcal{U}} A\\
&\mathsf{refl}_{\mathcal{U}}\ (\_, \mathsf{bool}) :\equiv |\_, \mathsf{bool}{\sim}|\\
&\mathsf{refl}_{\mathcal{U}}\ (\_, \mathsf{pi}\, a\, a_{01}\, b\, b_{01}) :\equiv |\_, \mathsf{pi}{\sim}\, a_{01}\, b_{01}|
\end{aligned}$$

As before, we can close the universe under dependent functions and booleans:

$$\begin{aligned}
&\mathbb{2}_{\mathbb{U}} : \mathsf{Tm}\ \Gamma\ \mathbb{U}\\
&|\mathbb{2}_{\mathbb{U}}|\ \_ :\equiv \_\ , \mathsf{bool}\\
&\mathbb{2}_{\mathbb{U}}^{\sim}\ \_ :\equiv |\_\ , \mathsf{bool}{\sim}|
\end{aligned}$$

$$\begin{aligned}
&\Pi_{\mathbb{U}} : (A : \mathsf{Tm}\ \Gamma\ \mathbb{U}) \to \mathsf{Tm}\ (\Gamma \triangleright \mathbb{E}l\ A)\ \mathbb{U} \to \mathsf{Tm}\ \Gamma\ \mathbb{U}\\
&|\Pi_{\mathbb{U}}\ A\ B|\gamma :\equiv \_\ ,\ \mathsf{pi}\ (\pi_2(|A|\gamma))\ (\mathsf{inj}\ \_\ \_\ (\mathsf{refl}_{\mathcal{U}}\ (|A|\gamma)))\\
&\qquad\qquad\qquad (\lambda\, x\, .\, \pi_2\, (|B|\ (\gamma, x)))\\
&\qquad\qquad\qquad (\lambda\, x_{01}\, .\, \mathsf{inj}\ \_\ \_\ (B^{\sim}\ (\mathsf{refl}\ \Gamma\ \gamma, x_{01})))\\
&(\Pi_{\mathbb{U}}\ A\ B)^{\sim}\ \gamma_{01} :\equiv |\_, \mathsf{pi}{\sim}\ (\mathsf{inj}\ \_\ \_\ (A^{\sim}\ \gamma_{01}))\ (\lambda\, x_{01}\, .\, \mathsf{inj}\ \_\ \_\ (B^{\sim}\ (\gamma_{01}, x_{01})))|
\end{aligned}$$

We direct the reader to the Agda formalization [Ses23] for the full details of these definitions, as they are quite technical. The final result does not use induction-recursion, but it is nevertheless an instance of infinitary induction-induction. The ability to define arbitrary, infinitary inductive-inductive types clashes, again, with our objective of keeping the core metatheory as minimal as possible. The next step is therefore to reduce this inductive-inductive universe to one that does not require any form of induction-induction.

6.5 Inductive setoid universe

The goal of this section is to construct the inductive-inductive universe of setoids from the previous section without assuming arbitrary inductive-inductive definitions. Practically, this entails reducing the universe IIT to simpler principles, like inductive families, that are available in the metatheory.

Finitary IITs are known to be reducible to inductive families, via a systematic method illustrated in Section 3.2. It is not *yet* clear, however, whether this method can be applied to *infinitary* types like our universe IIT. Of course, the absence of a general reduction method does not mean that we cannot reduce particular concrete instances of infinitary induction-induction, which is exactly what we hope for our universe construction.

The obvious challenge in successfully completing this reduction is to avoid the need for extensionality in the metatheory. In fact, consider the simple infinitary inductive-inductive type obtained from the previous Con/Ty example by replacing the finitary constructor Π with an infinitary one: $\Pi^\infty : \{\Gamma : \mathsf{Con}\} \to (\mathbb{N} \to \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ \Gamma$. Already with this simple example, we run into problems as soon as we try to define the eliminator. One issue is that the definition of the eliminators relies on proving that the well-typing predicates are propositional, that is, any two of their elements are equal. We have seen an example of this phenomenon in Section 3.2, where propositionality of the predicates was a critical requirement even for the simpler cases of the left-totality proofs. Without further assumptions on the nature of the predicates, proving propositionality must be done by induction; alas, this process requires function extensionality in our case, since the predicates for the universe IIT are infinitary and thus include higher-order constructors.

One way to get around this is to define the well-typing predicates as **Prop**-valued families, rather than in **Type**:

$$
\begin{aligned}
&\mathsf{data}\ \mathsf{inU}_0 &&: \mathbf{Type} \to \mathbf{Type}_1\\
&\mathsf{data}\ \mathsf{inU}{\sim}_0 &&: \{A\ A' : \mathbf{Type}\} \to (A \to A' \to \mathbf{Prop}) \to \mathbf{Type}_1\\
&\mathsf{data}\ \mathsf{inU}_1 &&: (A : \mathbf{Type}) \to \mathsf{inU}_0\ A \to \mathbf{Prop}_1\\
&\mathsf{data}\ \mathsf{inU}{\sim}_1 &&: \{A\ A' : \mathbf{Type}\}\{R : A \to A' \to \mathbf{Prop}\}\\
& &&\quad \to \mathsf{inU}_0\ A \to \mathsf{inU}_0\ A' \to \mathsf{inU}{\sim}_0\ R \to \mathbf{Prop}_1
\end{aligned}
$$

The constructors of these types are technically verbose but conceptually straightforward, as they are just systematically derived from the original definition of inU and $\mathsf{inU}{\sim}$ via the same process of erasure described in Section 3.2 for finitary IITs. We only give the boolean cases below:

$$\begin{array}{ll} \mathsf{bool}_0 : \mathsf{inU}_0\ \mathbb{2} & \mathsf{bool}{\sim}_0 : \mathsf{inU}{\sim}_0\ (\_ \overset{?}{=}_{\mathbb{2}} \_) \\ \mathsf{bool}_1 : \mathsf{inU}_1\ \mathsf{bool}_0 & \mathsf{bool}{\sim}_1 : \mathsf{inU}{\sim}_1\ \mathsf{bool}_0\ \mathsf{bool}_0\ \mathsf{bool}{\sim}_0 \end{array}$$

Using **Prop** avoids the issue of proving propositionality altogether, since the predicates are now propositional by definition. However, it introduces a different issue. In order to define the eliminators for inU and $\mathsf{inU}{\sim}$, we need to be able to define inversion principles on inU_1 and $\mathsf{inU}{\sim}_1$ expressing equational constraints on their indices, as well as transport along those equations. Again, we have already seen examples of this in Section 3.2, where the inversion principles arising from the well-formedness proofs were crucial in defining left-totality of the eliminator relations[1].

For example, when defining the eliminator for $\mathsf{inU}{\sim}$ in the case of $\mathsf{bool}_{\sim}$, we need an inversion principle on $\mathsf{inU}{\sim}_1$ stating that the $\mathsf{bool}_{\sim 0}$ constructor is well-formed if and only if the inU_0 indices are themselves constructed via bool_0:

$$\mathsf{inv\text{-}bool}{\sim}_1 : \mathsf{inU}{\sim}_1\ x\ y\ \mathsf{bool}{\sim}_0 \to x = \mathsf{bool}_0 \times y = \mathsf{bool}_0$$

In order to even be able to express these inversion principles, we need to extend the metatheory with some kind of identity type. However, this type cannot live in **Type**, since proving the inversions would require arbitrary elimination of proof-irrelevant predicates into a proof-relevant identity type, which is not possible for consistency reasons.

Instead, we extend the metatheory with the following definitionally proof-irrelevant identity type

$$\begin{array}{ll} \mathsf{Id} & : \{A : \mathbf{Type}\} \to A \to A \to \mathbf{Prop} \\ \mathsf{refl} & : \{A : \mathbf{Type}\}(a : A) \to \mathsf{Id}\ a\ a \end{array}$$

[1]Actually, in the example of Section 3.2 we have used dependent pattern matching instead of explicit inversion principles, as the two were completely equivalent in that case. In the current case, however, proof-irrelevance prevents us from being able to pattern match of well-formedness proofs.

$$\mathsf{transp} : \{A : \mathbf{Type}\}(C : A \to \mathbf{Type})\{a_0\ a_1 : A\} \to \mathsf{Id}\ a_0\ a_1 \to C\ a_0 \to C\ a_1$$

with the computation rule $\mathsf{transp}\ C\ e\ u \equiv u$ whenever e is a reflexive equation.

With Id we can now state and prove all the necessary inversion principles, as well as transport over them. We are now left to define the actual eliminators for the encoded IIT. Perhaps surprisingly, the *simple* (i.e. non-recursive-recursive) elimination principle is sufficient for our purposes: all functions described in Section 6.4 can be defined just using the simple eliminator without recursion-recursion. The simple eliminator itself can be defined by pattern matching on the untyped codes, and does not require extensionality or any extra principles beyond **Prop** and strong transport. Once the inductive encoding of the inductive-inductive universe is done, the setoid universe can be defined just as in Section 6.4.

6.6 Inductive setoid universe with general eliminators

The inductive setoid universe thus described is proposed in our published work [ABK^{+}21], on which we based this chapter until now. The paper version of the universe IIT does not include an encoding of the fully general eliminators, but instead constructs the setoid universe using the simple eliminators. The reason for this omission was a suspicion that deriving the general eliminators would require function extensionality.

Being able to define the setoid universe with just the simple elimination principle is certainly a remarkable result in its own right; however, because we were forced to work with a less powerful elimination principle, the definitions themselves ended up being quite technical and arguably not very easy on the reader. Partially for this reason, we won't discuss its details here and instead refer the reader to the Agda formalization [Ses23].

Having said that, it is natural to ask ourselves if deriving the general eliminators for the universe IIT is truly out of the question. As it turns out, despite our fears we can in fact derive them. While a "naive" application of the *finitary* reduction method to the universe IIT would fail without *funext* in the encoding metatheory, we realize that there are ways to circumvent the difficulties that arise. The two major stumbling blocks when encoding infinitary IITs in an intensional setting are the proof of propositionality of the well-formedness predicates and the proof of right-uniqueness of the eliminator relations, both of which require *funext*. While we have established that we can avoid proving propositionality of the predicates by defining them in **Prop**, it is not yet clear how to get away from proving right-uniqueness. We will see in the following sections that many infinitary IITs can be encoded via eliminator relations without having to prove right-uniqueness, by careful use of the induction hypotheses and the inputs to the proof term of left-totality. Fortunately, the universe IIT $\mathsf{inU}/\mathsf{inU}{\sim}$ happens to be one of these types. We defer discussing the details of the encoding until Section 8.2. In the meantime, we will assume to have the general induction principle for $\mathsf{inU}/\mathsf{inU}{\sim}$ at our disposal, and take a look at some useful applications of it, namely the derivation of universe induction, in the next section.

6.6.1 Universe induction

Universes have introduction rules expressing how types belonging to that universe can be formed, however there usually is no formal enforcement that types can *only* be constructed from that specific set of constructors. When this is the case, we say that the universe is *open*.

An alternative to open universes is to equip them with an induction principle expressing that all types belonging to that universe are a result of repeated application of the introduction rules [NPS90]. We then say that the universe is *closed*.

For example, given a universe of small sets $\mathsf{S} : \mathbf{Type}, \mathsf{T} : S \to \mathbf{Type}$ with codes $\mathsf{bool}, \mathsf{prod}, \mathsf{fun}$ for booleans, products, and functions, an eliminator for S is a term:

$$\frac{\begin{array}{c} C : \mathsf{S} \to \mathbf{Type} \\ c_1 : C\,\mathsf{bool} \\ c_2 : \forall\{a\,b\}\,,\, C\,a \to C\,b \to C\,(\mathsf{prod}\,a\,b) \\ c_3 : \forall\{a\,b\}\,,\, C\,a \to ((x : \mathsf{T}\,a) \to C\,(b\,x)) \to C\,(\mathsf{pi}\,a\,b) \end{array}}{f \equiv \mathsf{elim}\,C\,c_1\,c_2\,c_3 : (s : \mathsf{S}) \to C\,s}$$

such that $f\,\mathsf{bool} \equiv c_1$, $f\,(\mathsf{prod}\,a\,b) \equiv c_2\,(f\,a)\,(f\,b)$, and $f\,(\mathsf{pi}\,a\,b) \equiv c_2\,(f\,a)\,(\lambda\,x\,.\,f\,(b\,x))$.

Whether to adopt *closed* or *open* universes is a matter of trade off, as both approaches have advantages and disadvantages. An advantage and common use case of closed inductive universes is *generic programming*. For example, we can take our universe S and write a generic program that *curries* any term:

$\mathsf{curry\text{-}fun\text{-}ty} : \mathsf{S} \to \mathsf{S} \to \mathsf{S}$

$\mathsf{curry\text{-}fun\text{-}ty} :\equiv \mathsf{elim}\,(\lambda\,\_\,.\,\mathsf{S} \to \mathsf{S})\,(\mathsf{pi}\,\mathsf{bool})\,(\lambda\,h_a\,h_b\,.\,h_a \circ h_b)\,(\lambda\,\{a\,b\}\,\_\,\_\,.\,\mathsf{pi}\,(\mathsf{pi}\,a\,b))$

$\mathsf{curry\text{-}ty} :\equiv \mathsf{elim}\,\_\,(\lambda x.x)\,(\lambda\,\{a\,b\}\,\_\,\_\,.\,\mathsf{curry\text{-}fun\text{-}ty}\,a\,b) : \mathsf{S} \to \mathsf{S}$

$\mathsf{curry\text{-}fun} : (a\ b : \mathsf{S}) \to \mathsf{T}\,(\mathsf{fun}\,a\,b) \to \mathsf{T}\,(\mathsf{curry\text{-}fun\text{-}ty}\,a\ b)$

$\mathsf{curry\text{-}fun} :\equiv \mathsf{elim}\,\_\,(\lambda\,\_\,f\,.\,f)\,(\lambda\,h_a\,h_b\,\_\,f\,.\,h_a\,\_\,(\lambda x\,.\,h_b\,\_\,(\lambda y\,.\,f\,(x, y))))\,(\lambda\,\_\,\_\,\_\,f\,.\,f)$

$\mathsf{curry} : (s : \mathsf{S}) \to \mathsf{T}\,s \to \mathsf{T}\,(\mathsf{curry\text{-}ty}\,s)$

$\mathsf{curry} :\equiv \mathsf{elim}\,\_\,(\lambda x\,.\,x)\,(\lambda\,\_\,\_\,x\,.\,x)\,(\lambda\,\{a\,b\}\,\_\,\_\,.\,\mathsf{curry\text{-}fun}\,a\,b)$

A drawback of universe induction, aka *type-case*, is that it clashes with other principles like *univalence* and *parametricity*. These principles are strongly tied to the notion of types as black boxes representing and enforcing layers of abstraction, which is in contradiction with the ability to inspect and perform case analysis on types.

Universe induction is employed in systems like OTT [AMS07] and XTT [SAG19] to generically compute the structure of equality types for any given type in the universe. Both these systems are based on a setoid model, thus we wonder if the same principle can be added to our own setoid universe in the context of SeTT. As it turns out, the stronger elimination principle that we have implemented for the universe IIT $\mathsf{inU}/\mathsf{inU}{\sim}$ now allows us to model universe induction for our setoid universe $\mathbb{U}$.

Given a family $F : \mathsf{Ty}\,(\Gamma \rhd \mathbb{U})$ over the universe, and *methods*[2]:

$$\begin{aligned}
&\mathit{fb} : \mathsf{Tm}\ \Gamma\ F[2_{\mathbb{U}}]\\
&\mathit{fp} : \mathsf{Tm}\ (\Gamma \rhd A : \mathbb{U} \rhd B : \Pi\,(a : \mathbb{E}l\ A)\,\mathbb{U} \rhd F[A] \rhd \Pi\,(a : \mathbb{E}l\ A)\,F[B \cdot a])\ F[\Pi_{\mathbb{U}}\ A\, B]
\end{aligned}$$

We define a dependent function

$$\mathsf{Elim} : \mathsf{Tm}\ \Gamma\ (\Pi\ \mathbb{U}\ F)$$

by induction on its argument. We sketch the core part of the definition, i.e. the component $|\mathsf{Elim}| : (\gamma : |\Gamma|)(x : |\mathbb{U}|\gamma) \to |F|(\gamma, x)$.

$$\begin{aligned}
&|\mathsf{Elim}|\,\gamma\,(\_, \mathsf{bool}) = |\mathit{fb}|\gamma\\
&|\mathsf{Elim}|\,\gamma\,(\_, \mathsf{pi}\,\{A\}\,\{B\}\,\{A_{01}\}\,\{B_{01}\}\,a\ a_{01}\ b\ b_{01}) = \,?
\end{aligned}$$

The boolean case is immediate. On the function case, we construct the following by induction hypothesis

$$\begin{aligned}
&h_a :\equiv |\mathsf{Elim}|\,\gamma\,(\_, a)\\
&h_b :\equiv (\lambda x\,.\,|\mathsf{Elim}|\,\gamma\,(\_, b\,a)), (\lambda x_{01}\,.\,\mathsf{Elim}^{\sim}\,(\mathsf{refl}\,\Gamma\,\gamma)\,|\_, b_{01}\,(\mathsf{out}\,a_{01}\,x_{01})|)\\
&h :\equiv |\mathit{fp}|\ (\_, h_a, h_b)
\end{aligned}$$

The term h is the result we want, but its type is not quite right as some of its indices are only extensionally equal (i.e., up to setoid equality) to those of the goal

[2]Recall the definition of $\_ \cdot \_$ on CwFs from Section 2.2.1. As we are presenting the setoid model as a CwF with extra structure, we assume here to have a corresponding term $\_ \cdot \_$.

type. Let $\mathsf{tm}_A : \mathsf{Tm}\,\Gamma\,\mathbb{U}$ and $\mathsf{tm}_B : \mathsf{Tm}\,(\Gamma \triangleright \mathbb{E}l\,\mathsf{tm}_A)\,\mathbb{U}$ be setoid terms internalizing the codes a and b as elements of $\mathbb{U}$. The type of h results in

$$|F|(\gamma, |\Pi_{\mathbb{U}}\,\mathsf{tm}_A\,\mathsf{tm}_B|\gamma)$$

whereas the goal type is just

$$|F|(\gamma, (\_, \mathsf{pi}\,a\,a_{01}\,b\,b_{01}))$$

Unfortunately these two types are not definitionally equal, as a consequence of our use of the out and in operators to define the universe, and in particular the $\Pi_{\mathbb{U}}$ constructor. While these operators establish equivalences between different relations, they certainly need not establish definitional equalities.

Nevertheless, we can prove a setoid equality between the two codes, as witnessed by the following proof:

$$e : \mathsf{inU}{\sim}\ (\mathsf{pi}\,a\,a_{01}\,b\,b_{01})\ (\pi_2\,(|\Pi\,\mathsf{tm}_A\,\mathsf{tm}_B|\gamma))$$
$$e :\equiv \mathsf{pi}{\sim}\ (\mathsf{inj}\,_{\_\,\_}|\_, a_{01}|)\ (\lambda x_{01}\,.\,\mathsf{inj}\,_{\_\,\_}|\_, b_{01}\,(\mathsf{out}\,a_{01}\,x_{01})|)$$

We thus coerce along e, and conclude with $\mathsf{coe}\ F\ (\mathsf{refl}\ \Gamma\ \gamma, |\_, e|)\ h$.

The second and final ingredient to define Elim is a proof $\mathsf{Elim}^{\sim}$ that $|\mathsf{Elim}|$ respects setoid equality:

$$\mathsf{Elim}^{\sim} : (p : \Gamma^{\sim}\,\gamma_0\,\gamma_1) \to (\Pi\ \mathbb{U}\ F)^{\sim}\ p\ (|\mathsf{Elim}|\,\gamma_0)\ (|\mathsf{Elim}|\,\gamma_1)$$

We refer the reader to the formalization [Ses23] for its definition. We do note, however, that $\mathsf{Elim}^{\sim}$ is defined by induction on $\mathsf{inU}{\sim}$, and mutually with $|\mathsf{Elim}|$ which also appears in its type. This recursive-recursive definition of Elim thus crucially relies on the *general* induction principle of $\mathsf{inU}/\mathsf{inU}{\sim}$.

We can prove the relevant β-equations for the eliminator thus defined. We have a definitional equation for the boolean case:

$$\mathsf{Elim} \cdot \mathbb{2}_{\mathbb{U}} \equiv fb$$

However, because of the extra coercion in the definition of Elim for the dependent function case, we only obtain a computation rule up to internal propositional equality:

$$\mathsf{Tm}\ \Gamma\ (\mathsf{Id}\ _{\_}\ (\mathsf{Elim} \cdot \Pi_{\mathbb{U}}\,A\,B)$$
$$\qquad (fp[A, B, \mathsf{Elim} \cdot A, \mathsf{lam}\,(\mathsf{Elim} \cdot B[\mathsf{vz}])]))$$

Chapter 7

Conclusion of Part II

In this part of the thesis we have described the construction of a universe of setoids in the strict setoid model of type theory; the universe is derived in several steps, first as an inductive-recursive definition, then as an inductive-inductive definition, and finally as an inductive type. Every encoding is obtained from the previous by adapting known datatype transformation methods in a novel way that accounts for the peculiarities of our construction. This part of the thesis, therefore, not only contributes a setoid universe construction, but also demonstrates how our initial inductive-recursive-recursive definition of the universe can be encoded as a purely inductive construction that does not rely on induction-recursion or recursion-recursion, as well as function extensionality.

Part of the content presented here has already been published as joint work in [ABK$^+$21]. We have improved upon, and added to it with (1) an implementation of general eliminators with definitional β-equations for the universe IIT encoded via inductive families, as well as (2) an implementation of universe induction/typecase for the setoid universe defined in terms of such IIT.

7.1 Formalization

All the mathematical content of Chapter 6 has been formalized in the Agda proof assistant. The files can be found in the permanent repository [Ses23], inside the directory `setoid-univ`.

The portion of formalization covering the material up to Section 6.5 has been previously published as part of [ABK$^+$21], and can be found unchanged in [Ses23]. In addition to that, we include a complete formalization of the content of Section 6.6, Section 6.6.1, and Section 8.2. We refer the interested reader to the file `Readme.agda` for a guide on the contents of the formalization. We emphasize in particular the Agda files relevant to the two major contributions original to this part of the thesis, namely

- the encoding of the types and term constructors of the universe IIT in terms of inductive families, in the module `Setoit.Sets.lib`;
- the implementation of the general eliminators for the aforementioned encoded IIT (Section 8.2), in the module `Setoid.Sets.gen-elim`;
- the implementation of the universe elimination principle/typecase (Section 6.6.1) for the setoid universe constructed via the universe IIT (as presented in Section 6.4), in the module `Setoid.UnivElim-SetsII`.

7.2 Future work

The work presented in this part of the thesis offers several opportunities for improvement and future work, which we summarize below:

- The obvious next step is to extend the rules of SeTT as written in [ABKT19] with rules for a universe reflecting the semantics presented here.
- In Section 6.6.1 we have demonstrated that the IIT formulation of the setoid universe can support universe induction; however, we only managed to prove the β-rule for the code of function types up to setoid equality, i.e. SeTT's internal propositional equality. We would like to investigate whether this equation can be made definitional, perhaps via a different formulation of the setoid universe.
- The inductive version of the universe IIT $\mathsf{inU}/\mathsf{inU}{\sim}$ relies on the presence of a proof-irrelevant identity type with a strong transport rule. While universes of definitionally proof-irrelevant propositions are a relatively common and well-understood tool supported by proof-assistants like Agda and Coq [GCST19], proof-irrelevant identity types have received less attention. We would like to study the metatheory of type theories with this kind of identity type, since previous work on the topic seems to suggest that is represents a non-trivial addition [AC19]. We think consistency can be established by modeling **Prop** as h-propositions, however canonicity and normalization would require further work.
- Both SeTT [ABKT19] and XTT [SAG19] are syntactic presentations of the setoid model with similar design choices, like definitional proof-irrelevance and heterogeneous equality types. Moreover, as we have shown in this thesis, SeTT universes can support universe induction, which is a crucial ingredient

of XTT. We would like to know whether their respective notions of models are equivalent, that is, if we can obtain an XTT model from a SeTT model, and vice versa.

7.3 Related work

There are several alternative approaches to constructing universes of setoids in the literature. We recall OTT and XTT, which were previously mentioned in Chapter 6. In [AMS07], the authors define an inductive-recursive universe for OTT, and rely on ETT for its justification; on the other hand, we construct our inductive-recursive-recursive universe by reducing it to an indexed inductive definition that can be fully expressed in intensional MLTT extended with strict propositions and proof-irrelevant identity types.

In addition to OTT's universes and our own, XTT [SAG19] also features inductive-recursive universes supporting a form of type-case. It should be noted, however, that while completely optional in our setting, type-case in OTT and XTT seems to be necessary to make equality proofs and coercions compute as intended. Similarly to our universe, XTT universes are presented in an inductive-recursive style, but are actually defined using simpler building blocks such as W types and Σ-types.

In [AR14a], the authors give a formalized account of Allen's PER semantics [All87] for a hierarchy of type universes for Nuprl, carried out in the Coq proof assistant (see Section 4.2 of the technical report [AR14b] for a detailed account). In this semantics, typehood (which terms are types) and member equality (which terms are equal elements of some type) would have to be given mutually, giving rise to an inductive-recursive definition. In the attempt to avoid induction-recursion (IR), the authors follow Allen's approach and encode the mutual definition as a purely inductive construction, relegating IR — which is, in any case, not currently supported by Coq — to a purely explanatory role. Their encoding can be understood as an instance of Capretta's method to reduce (small) IR to induction in the Calculus of Inductive Constructions [Cap04], and undoubtedly bears many similarities with our work in its general approach; however, our novel combination of large IR with constructs such as strict propositions and recursion-recursion arguably brings on a unique set of challenges and related solutions.

On the broader topic of bootstrapping extensionality from an intensional base, we must mention the Minimalist Foundation by Maietti and Sambin [MS05,Mai09]. This system is implemented in two levels: an extensional level emTT—a variation of ETT with quotients, proof irrelevance, and a notion of collection — and an intensional level — a variation of intensional MLTT with a notion of collection, and a weaker notion of conversion that does not include the ξ rule for λ-terms. The extensional level is built on top of the intensional one via a setoid model. One of the core design principles of the Minimalist Foundation, shared by SeTT, is to alleviate the burden of working with extensional concepts in intensional type theory, by offering instead a more ergonomic system that hides the complex setoid machinery under the hood, while also guaranteeing soundness with respect to a particular setoid model construction. One crucial difference between SeTT and emTT is that emTT is a version of ETT and thus has undecidable type-checking, whereas SeTT is an attempt to add extensionality to intensional type theory without sacrificing decidability.

Part III

Inductive-inductive types

Chapter 8

Infinitary induction-induction

Induction-induction [NF13] is a powerful form of induction that can be used to define complex structures like the syntax of dependent type theory [AK16]. In a type theory that does not support induction-induction as a primitive notion, like the one underlying the proof-assistants Coq and Lean, it is not possible to just *define* IITs directly as a single inductive definition. Still, if the theory is rich enough it might be possible to instead *encode* the components that constitute the IIT by explicitly defining its types, constructors, and eliminators using the building blocks available. One example of this process of *reduction* is when we encode arbitrary inductive types as a particular instantiation of W-types [Dyb97].

It is known that *finitary* inductive-inductive definitions can be reduced to inductive families [AKKvR19, AKKvR18, KKL20]. We have seen an instance of this construction in Section 3.2, where we applied the reduction method described in [vR19] to a minimal type-theoretic syntax with two sorts $\mathsf{Con}, \mathsf{Ty}$.

On the other hand, the problem of reducing *infinitary* IITs to simpler forms of induction has received comparatively less attention, and most works that do touch on the topic tend, perhaps unsurprisingly, to presuppose an extensional setting (see discussion in Section 13.4). We are interested in the reduction of infinitary IITs withing ambient encoding theories that are intensional, and that in particular *do not* admit function extensionality.

We have ventured into the infinitary territory in Section 6.5, where we have managed to apply the finitary encoding method as per Section 3.2 outside of its intended scope. Specifically, we targeted the universe IIT $\mathsf{inU}/\mathsf{inU}{\sim}$, a fairly complex infinitary IIT, and carried out the encoding in an intensional metatheory without *funext*. Although we did succeed in our goal it wasn't all smooth sailing, as we had to tweak the encoding method (and the metatheory) by adding definitional proof-irrelevance and a proof-irrelevant identity type.

Our endeavours to reduce the universe IIT to inductive families raised some questions:

- is it possible to encode the general eliminators for the universe IIT $\mathsf{inU}/\mathsf{inU}{\sim}$?
- is it possible to generalize the encoding method we used on the universe to a wider class of infinitary IITs?

The chapters that follow serve as our answers. An answer to the first question was already hinted at in Section 6.6, where we deferred discussing the details of the encoding until this chapter. We finally do so in Section 8.2. We complement that in Section 8.1 with an in-depth illustrative encoding of a simple but realistic infinitary IIT: a version of the Con/Ty IIT with an additional infinitary constructor for dependent function types quantifying over the natural numbers. We will use this running example as a reference in the later chapters, when generalizing the encoding method to arbitrary IIT specifications.

Chapters 9 to 12 will cover the main contribution of Part III, and tackle the second of the questions above. By looking at our encoding examples from Section 8.1 and 8.2 we identify some sufficient conditions for an infinitary IIT to be reducible to inductive families without having to assume funext in the encoding metatheory (Section 9.1). Accordingly, we fix a suitable "target" subclass of all IITs (Section 9.2) and formally define how to specify IITs in such class (Section 9.4). The remaining chapters will go into the details of the generalized encoding. We first identify what constitutes the IIT corresponding to given a specification, by defining a notion of IIT algebra (Chapter 10). We then show that for any specifiable IIT we can encode a corresponding algebra, i.e. its types and constructors (Chapter 11). We conclude by showing that the algebras thus constructed can always be equipped with general eliminators with definitional β-rules (Chapter 12).

8.1 Example: contexts and types

In this section we revisit the reduction example from Section 3.2 in an infinitary setting. The idea is to test how much of the construction showcased in Section 3.2 can be applied to infinitary IITs, and what alterations to the encoding theory are required to make the method work.

We start from the same metatheory/encoding theory as in Section 3.2: this is intensional Martin-Löf Type Theory with universes $\mathbf{Type}_i$ (at least two) equipped with (mutual) inductive families and Π, Σ types. Learning our lesson from the universe IIT encoding attempts in Section 6.5, we further extend the metatheory with two components: (1) a universe **Prop** of definitionally proof-irrelevant propositions, containing $\top, \Pi, \Sigma$; (2) a proof-irrelevant identity type with strong transport rule

$$\mathsf{Id} : (A : \mathbf{Type}) \to A \to A \to \mathbf{Prop}$$

$$\begin{aligned}
&\mathsf{refl} : \{A : \mathbf{Type}\}(x : A) \to \mathsf{Id}\ A\ a\ a \\
&\mathsf{transp} : \{A : \mathbf{Type}\}(B : A \to \mathbf{Type})\{x\ y : A\} \to \mathsf{Id}\ A\ x\ y \to B\ x \to B\ y \\
&\mathsf{transp}\ B\ (\mathsf{refl}\ x)\ b \equiv b
\end{aligned}$$

We also have a lifting operator $\mathsf{Lift} : \mathbf{Prop} \to \mathbf{Type}$ with constructor $\mathsf{lift} : \{P : \mathbf{Prop}\} \to P \to \mathsf{Lift}\ P$ and destructor $\mathsf{unlift} : \{P : \mathbf{Prop}\} \to \mathsf{Lift}\ P \to P$, which we use to implicitly coerce propositions into types as convenient.

We take the already familiar Con/Ty IIT and add a higher-order constructor $\hat{\pi}$ that makes it *infinitary*.

$$\begin{aligned}
&\bullet && : \mathsf{Con} \\
&\_ \triangleright \_ && : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Con} \\
&\iota && : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \\
&\bar{\pi} && : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\ \Gamma)(B : \mathsf{Ty}\ (\Gamma \triangleright A)) \to \mathsf{Ty}\ \Gamma \\
&\hat{\pi} && : (\Gamma : \mathsf{Con}) \to (\mathbb{N} \to \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ \Gamma
\end{aligned}$$

A real-world example of such an infinitary constructor can be found in the specification syntax for QIITs proposed by Kaposi et al. [KKA19]: they define a type theory of signatures as a QIIT, which includes sorts Con and Ty, and a constructor $\hat{\Pi} : (T : \mathbf{Type}) \to (T \to \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ \Gamma$ to express non-inductive parameters. Our constructor $\hat{\pi}$ is just $\hat{\Pi}$ instantiated to $T :\equiv \mathbb{N}$.

As an IIT, we would expect Con/Ty to be equipped with *eliminators*. To state them, we consider a *displayed algebra* over Con and Ty, which is given by indexed types

$$\begin{aligned}
&\mathsf{Con}^D && : \mathsf{Con} \to \mathbf{Type} \\
&\mathsf{Ty}^D && : \{\Gamma : \mathsf{Con}\} \to \mathsf{Con}^D\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathbf{Type}
\end{aligned}$$

and functions corresponding to each constructor:

$$\begin{aligned}
&\bullet^D && : \mathsf{Con}^D\ \bullet \\
&\_ \triangleright^D \_ && : \forall\{\Gamma\ A\}(\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ A \to \mathsf{Con}^D\ (\Gamma \triangleright A) \\
&\iota^D && : \{\Gamma : \mathsf{Con}\}(\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ (\iota\ \Gamma) \\
&\bar{\pi}^D && : \{\Gamma\ A\ B\}(\Gamma^D : \mathsf{Con}^D\ \Gamma)(A^D : \mathsf{Ty}^D\ \Gamma^D\ A)(B^D : \mathsf{Ty}^D\ (\Gamma^D \triangleright^D A)\ B) \\
& && \to \mathsf{Ty}^D\ \Gamma^D\ (\bar{\pi}\ \Gamma\ A\ B) \\
&\hat{\pi}^D && : \{\Gamma : \mathsf{Con}\}\{F : \mathbb{N} \to \mathsf{Ty}\ \Gamma\}(\Gamma^D : \mathsf{Con}^D\ \Gamma) \to ((n : \mathbb{N}) \to \mathsf{Ty}^D\ \Gamma^D\ (F\ n)) \\
& && \to \mathsf{Ty}^D\ \Gamma^D\ (\hat{\pi}\ \Gamma\ F)
\end{aligned}$$

The general eliminators have the following signatures:

$$
\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}} : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma\\
&\mathsf{elim}_{\mathsf{Ty}} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ A
\end{aligned}
$$

In addition, we have computation rules that explain the computational behaviour of the eliminators on constructors:

$$
\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}}\ \bullet \equiv \bullet^D\\
&\mathsf{elim}_{\mathsf{Con}}\ (\Gamma \rhd A) \equiv \mathsf{elim}_{\mathsf{Con}}\ \Gamma \rhd^D \mathsf{elim}_{\mathsf{Ty}}\ A\\
&\mathsf{elim}_{\mathsf{Ty}}\ (\iota\ \Gamma) \equiv \iota^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\\
&\mathsf{elim}_{\mathsf{Ty}}(\bar{\pi}\ \Gamma\ A\ B) \equiv \bar{\pi}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\mathsf{elim}_{\mathsf{Ty}}\ A)\ (\mathsf{elim}_{\mathsf{Ty}}\ B)\\
&\mathsf{elim}_{\mathsf{Ty}}(\hat{\pi}\ \Gamma\ F) \equiv \pi^D_\infty\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\lambda n\ .\ \mathsf{elim}_{\mathsf{Ty}}\ (F\,n))
\end{aligned}
$$

Following what we did in Section 3.2, we now split the IIT specification into *erased types* $\mathsf{Con}_0, \mathsf{Ty}_0$: **Type** and *well-formedness predicates* $\mathsf{Con}_1 : \mathsf{Con}_0 \to$ **Prop**, $\mathsf{Ty}_1 : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to$ **Prop**, with the important difference that now $\mathsf{Con}_1, \mathsf{Ty}_1$ are defined in **Prop**.

$$
\begin{aligned}
\bullet_0 &: \mathsf{Con}_0\\
\_\rhd_0\_ &: \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Con}_0\\
\iota_0 &: \mathsf{Con}_0 \to \mathsf{Ty}_0\\
\bar{\pi}_0 &: \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0\\
\hat{\pi}_0 &: \mathsf{Con}_0 \to (\mathbb{N} \to \mathsf{Ty}_0) \to \mathsf{Ty}_0
\end{aligned}
$$

$$
\begin{aligned}
\bullet_1 &: \mathsf{Con}_1\ \bullet_0\\
\_\rhd_1\_ &: \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ A_0 \to \mathsf{Con}_1\ (\Gamma_0 \rhd_0 A_0)\\
\iota_1 &: \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0)\\
\bar{\pi}_1 &: \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ A_0 \to \mathsf{Ty}_1\ (\Gamma_0 \rhd_0 A_0)\ B_0\\
&\quad \to \mathsf{Ty}_1\ \Gamma_0\ (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0)\\
\hat{\pi}_1 &: \mathsf{Con}_1\ \Gamma_0 \to (\forall n.\mathsf{Ty}_1\ \Gamma_0\ (F_0\,n))\\
&\quad \to \mathsf{Ty}_1\ \Gamma_0\ (\hat{\pi}_0\ \Gamma_0\ F_0)
\end{aligned}
$$

The predicate types can be defined as a mutual inductive definition, or alternatively by (large) recursion over the erased types. More details about this construction later in Section 8.1.1.

We have inversion principles expressing that well-formedness of a compound term implies well-formedness of its components:

$$
\begin{aligned}
&\mathsf{inv}\text{-}\rhd_1\text{-}\mathsf{Con} : \forall\{\Gamma_0\ A_0\} \to \mathsf{Con}_1\ (\Gamma_0 \rhd_0 A_0) \to \mathsf{Con}_1\ \Gamma_0\\
&\mathsf{inv}\text{-}\rhd_1\text{-}\mathsf{Ty} : \forall\{\Gamma_0\ A_0\} \to \mathsf{Con}_1\ (\Gamma_0 \rhd_0 A_0) \to \mathsf{Ty}_1\ \Gamma_0\ A_0\\
&\mathsf{inv}\text{-}\hat{\pi}\text{-}\mathsf{Con} : \forall\{\Gamma_0\ A_0\} \to \mathsf{Ty}_1\ \Gamma_0\ (\hat{\pi}_0\ \Gamma_0\ A_0) \to \mathsf{Con}_1\ \Gamma_0\\
&\mathsf{inv}\text{-}\hat{\pi}\text{-}\mathsf{Ty} : \forall\{\Gamma_0\ A_0\} \to \mathsf{Ty}_1\ \Gamma_0\ (\hat{\pi}_0\ \Gamma_0\ A_0) \to \mathsf{Ty}_1\ \Gamma_0\ A_0\\
&\mathsf{inv}\text{-}\bar{\pi}_1\text{-}\mathsf{Dom} : \forall\{\Gamma_0\,A_0\,B_0\} \to \mathsf{Ty}_1\,\Gamma_0\,(\bar{\pi}_0\,\Gamma_0\,A_0\,B_0) \to \mathsf{Ty}_1\,\Gamma_0\,A_0
\end{aligned}
$$

$$\mathsf{inv}\text{-}\bar{\pi}_1\text{-}\mathsf{Cod} : \forall\{\Gamma_0\, A_0\, B_0\} \to \mathsf{Ty}_1\,\Gamma_0\,(\bar{\pi}_0\,\Gamma_0\,A_0\,B_0) \to \mathsf{Ty}_1\,(\Gamma_0 \rhd_0 A_0)\,B_0$$

As with the finitary Con/Ty, we can recover the original inductive-inductive type as $\mathsf{Con} :\equiv \Sigma\ (\Gamma_0 : \mathsf{Con}_0)\ (\mathsf{Con}_1\ \Gamma_0)$ and $\mathsf{Ty}\ \Gamma :\equiv \Sigma\ (A_0 : \mathsf{Ty}_0)\ (\mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0)$. We only show the new constructor:

$$\hat{\pi}\ (\Gamma_0, \Gamma_1)\, F :\equiv (\hat{\pi}_0\ \Gamma_0\ (\lambda n.\pi_1(F\,n)), \hat{\pi}_1\ \Gamma_1\ (\lambda n.\pi_2(F\,n)))$$

Next, we give the graph of the eliminators as a pair of mutually-defined relations. Except $\hat{\pi}^R$, the rest of the definition is just like the finitary case.

$$\begin{aligned}
&\mathsf{Con}^R : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma \to \mathbf{Type}\\
&\mathsf{Ty}^R\ \ : \{\Gamma : \mathsf{Con}\}(\Gamma^D : \mathsf{Con}^D\ \Gamma)(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ \Gamma\ A\ \Gamma^D \to \mathbf{Type}\\
&\bullet^R\ \ \ : \mathsf{Con}^R\ \bullet\ \bullet^D\\
&\_\rhd^R\_ : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ A\ A^D \to \mathsf{Con}^R\ (\Gamma \rhd A)\ (\Gamma^D \rhd^D A^D)\\
&\iota^R\ \ \ \ : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ (\iota\ \Gamma)\ (\iota^D\ \Gamma^D)\\
&\bar{\pi}^R\ \ \ : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ A\ A^D \to \mathsf{Ty}^R\ (\Gamma^D \rhd^D A^D)\ B\ B^D\\
&\qquad \to \mathsf{Ty}^R\ \Gamma^D\ (\pi\ \Gamma\ A\ B)\ (\pi^D\ \Gamma^D\ A^D\ B^D)\\
&\hat{\pi}^R\ \ \ : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to ((n : \mathbb{N}) \to \mathsf{Ty}^R\ \Gamma^D\ (F\ n)\ (F^D\ n))\\
&\qquad \to \mathsf{Ty}^R\ \Gamma^D\ (\hat{\pi}\ \Gamma\ F)\ (\hat{\pi}^D\ \Gamma^D\ F^D)
\end{aligned}$$

Like the predicate types, these relations can be defined inductively, or by recursion over the erased types. A recursive definition is given later in Section 8.1.1.

We now prove that these relations are left-total. This is done by mutual induction on the erased components of the Con and Ty inputs, respectively:

$$\begin{aligned}
&\mathsf{Con}^\exists : \forall\ \Gamma_0\ \Gamma_1 \to \Sigma\ (\Gamma^D : \mathsf{Con}^D\,(\Gamma_0, \Gamma_1))(\mathsf{Con}^R\ (\Gamma_0, \Gamma_1)\ \Gamma^D)\\
&\mathsf{Ty}^\exists : \forall\{\Gamma\,\Gamma^D\}(A_0 : \mathsf{Ty}_0)(A_1 : \mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0) \to \mathsf{Con}^R\ \Gamma\ \Gamma^D\\
&\qquad\qquad \to \Sigma(A^D : \mathsf{Ty}^D\,\Gamma\ (A_0, A_1)\ \Gamma^D)(\mathsf{Ty}^R\ (A_0, A_1)\ \Gamma^D\ A^D)
\end{aligned}$$

Proving $\mathsf{Con}^\exists$ is fairly straightforward:

$$\begin{aligned}
&\mathsf{Con}^\exists\ \ \bullet_0\ \ p :\equiv (\bullet^D, \bullet^R)\\
&\mathsf{Con}^\exists\ (\Gamma_0 \rhd_0 A_0)\ p :\equiv (\pi_1\ h \rhd^D \pi_1\ h'), (\pi_2\ h \rhd^R \pi_2\ h')
\end{aligned}$$

where in the second clause we obtain h and h' by induction hypothesis:

$$h :\equiv \mathsf{Con}^\exists\ \Gamma_0\ (\mathsf{inv}\text{-}\rhd_1\text{-}\mathsf{Con}\ p)$$

$$h' :\equiv \mathsf{Ty}^{\exists}\ A_0\ (\mathsf{inv}\text{-}\triangleright_1\text{-}\mathsf{Ty}\ p)\ (\pi_2\ h)$$

Proving $\mathsf{Ty}^{\exists}$ isn't conceptually different, however we now have to pay attention to the fact that Γ both appears as a standalone input as well as data to construct $A : \mathsf{Ty}\ \Gamma$. For example:

$$\mathsf{Ty}^{\exists}\ \{\Gamma_0, \Gamma_1\}\ \{\Gamma^D\}\ (\iota_0\ \Gamma_0')\ p\ r :\equiv\ ?$$

At this point, both Γ_0 and Γ_0' appear in the goal type, therefore it's impossible for us to conclude the proof via ι^D and ι-R unless we equate these two contexts.

Recall that we encountered the same stumbling block in Section 3.2 for the finitary version of Con/Ty. Like in that case, we should be able to conclude that Γ_0 equals Γ_0' from inversion on p. Here is where the proof-irrelevant identity type with strong transport rule becomes necessary: because Ty_1 is a strict proposition, any inversion principle from a proof of type Ty_1 can only ever provide us with an equality in **Prop**. In our case, we express this equality via the Id type, which we also write as $=$.

We have the following equational inversion principles on predicate types:

$$\begin{aligned}
&\mathsf{inv}\text{-}\iota_1 : \forall\{\Gamma_0\ \Gamma_0'\} \to \mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0') \to \Gamma_0 = \Gamma_0'\\
&\mathsf{inv}\text{-}\hat{\pi}_1 : \forall\{\Gamma_0\ \Gamma_0'\ F_0\} \to \mathsf{Ty}_1\ \Gamma_0\ (\hat{\pi}_0\ \Gamma_0'\ F_0) \to \Gamma_0 = \Gamma_0'\\
&\mathsf{inv}\text{-}\bar{\pi}_1 : \forall\{\Gamma_0\ \Gamma_0'\ A_0\ B_0\} \to \mathsf{Ty}_1\ \Gamma_0\ (\pi_0\ \Gamma_0'\ A_0\ B_0) \to \Gamma_0 = \Gamma_0'
\end{aligned}$$

After transporting via $\mathsf{inv}\text{-}\iota_1$, the proof of left-totality for the ι case concludes like in the finitary example of Section 3.2, with a pair $(\iota^D\,\Gamma^D, \iota^R\,r)$. Actually, unlike in the finitary example we do not need to further transport along proofs of propositionality of the predicate types, since definitional proof-irrelevance makes it trivial.

We quickly go through the remaining cases of $\mathsf{Ty}^{\exists}$, which are all fairly similar in execution.

$$\mathsf{Ty}^{\exists}\ \{\Gamma\}\ (\hat{\pi}_0\ \Gamma_0'\ F_0)\ p\ r :\equiv\ ?$$

After transporting along $\mathsf{inv}\text{-}\hat{\pi}_1\ p$ to establish $\pi_1\ \Gamma = \Gamma_0'$, the goal type becomes

$$\Sigma(A^D : \mathsf{Ty}^D\ \Gamma^D\ (\hat{\pi}\ \Gamma\ F))(\mathsf{Ty}^R\ \_\ A^D)$$

where $F_1 :\equiv \mathsf{inv}\text{-}\hat{\pi}\text{-}\mathsf{Ty}\ p$ and $F :\equiv \lambda\, n\,.\,(F_0\, n, F_1\, n)$. By inductive hypothesis, we have $h :\equiv \lambda n\,.\,\mathsf{Ty}^{\exists}\ (F_0\ n)\ (F_1\ n)\ r$. We conclude with

$$(\hat{\pi}^D\ \Gamma^D\ (\pi_1 \circ h), \hat{\pi}^R\ r\ (\pi_2 \circ h))$$

The $\bar{\pi}$ case is similar:

$$\mathsf{Ty}^{\exists}\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0)\ p :\equiv\ ?$$

After transporting along $\mathsf{inv}\text{-}\bar{\pi}_1\ p$, the goal type becomes

$$\Sigma(\mathsf{Ty}^D\ \Gamma^D\ (\bar{\pi}\ \Gamma\ A\ B))(\mathsf{Ty}^R\ \_)$$

where A_1, B_1 are obtained by inversion on p, and $A :\equiv (A_0, A_1), B :\equiv (B_0, B_1)$. By inductive hypothesis, we have

$$\begin{aligned}&h_A :\equiv \mathsf{Ty}^\exists\ A_0\ A_1\ r\\&h_B :\equiv \mathsf{Ty}^\exists\ B_0\ B_1\ (r \triangleright^R \pi_2\ h_A)\end{aligned}$$

from which we easily conclude

$$(\bar{\pi}^D\ \Gamma^D\ (\pi_1\ h_A)\ (\pi_1\ h_B)\ ,\ \bar{\pi}^R\ r\ (\pi_2\ h_A)\ (\pi_2\ h_B))$$

The detailed definition of $\mathsf{Ty}^\exists$, with explicit transports, in shown in Figure 8.1.

Having proved left-totality of the eliminator relations, we can finally define the eliminators as simple projections:

$$\begin{aligned}&\mathsf{elim}_{\mathsf{Con}}\ \Gamma :\equiv \pi_1\ (\mathsf{Con}^\exists\ \Gamma)\\&\mathsf{elim}_{\mathsf{Ty}}\ \{\Gamma\}\ A :\equiv \pi_1\ (\mathsf{Ty}^\exists\ A\ (\pi_2\ (\mathsf{Con}^\exists\ \Gamma)))\end{aligned}$$

Thanks to our use of **Prop** to define the predicates, all the expected β-equations on the eliminators as defined above hold by definition.

8.1.1 Recursive predicates and relations

The well-formedness predicates and eliminator relations that we used to encode the Con/Ty IIT can be defined by (large) elimination on erased types, without the need for **Prop** to be closed under arbitrary inductive definitions.

We start with the well-formedness predicate types.

$$\begin{aligned}&\mathsf{Con}_1 : \mathsf{Con}_0 \to \mathbf{Prop}\\&\mathsf{Ty}_1 : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathbf{Prop}\end{aligned}$$

Defining Con_1 is pretty straightforward: for any erased context, the predicate should hold if it also holds for all the sub-components:

$$\begin{aligned}&\mathsf{Con}_1\ \bullet_0 :\equiv \top\\&\mathsf{Con}_1\ (\Gamma_0 \triangleright_0 A_0) :\equiv \mathsf{Con}_1\ \Gamma_0 \times \mathsf{Ty}_1\ \Gamma_0\ A_0\end{aligned}$$

```
$T : \mathsf{Ty}_0 \to \mathsf{Con}_0 \to \mathbf{Type}$
$T\ A_0\ \Gamma_0 :\equiv (\Gamma_1 : \mathsf{Con}_1\ \Gamma_0)(A_1 : \mathsf{Ty}_1\ \Gamma_0\ A_0)(\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1))$
          $\to \mathsf{Con}^R\ \_\ \Gamma^D \to \Sigma(A^D : \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1))(\mathsf{Ty}^R\ \_\ A^D)$


$\mathsf{Ty}^\exists\ \{\Gamma_0, \Gamma_1\}\ (\iota_0\ \Gamma_0')\ p :\equiv \mathsf{transp}\ (T\ y_0)\ e\ h\ \Gamma_1\ p$
  **where**
    $h : T\ (\iota_0\ \Gamma_0')\ \Gamma_0'$
    $h :\equiv \lambda\ \Gamma_1'\ \Gamma'^D\ r\ \to\ \iota^D\ \Gamma^D, \iota^R\ r$
    $e : \Gamma_0' = \Gamma_0$
    $e :\equiv \mathsf{sym}\ (\mathsf{inv}\text{-}\iota_1\ p)$
$\mathsf{Ty}^\exists\ (\hat{\pi}_0\ \Gamma_0'\ F_0)\ p :\equiv \mathsf{transp}\ (T\ (\hat{\pi}_0\ \Gamma_0'\ F_0))\ e\ h\ \_\ p$
  **where**
    $h : T\ (\hat{\pi}_0\ \Gamma_0'\ F_0)\ \Gamma_0'$
    $h :\equiv \lambda\ \Gamma_1'\ p\ \Gamma'^D\ r\ \to\ \hat{\pi}^D\ \Gamma^D\ (\lambda n.\pi_1\ (h\, n)), \hat{\pi}^R\ r\ (\lambda n.\pi_2\ (h\, n))$
      **where**
        $ih :\equiv \lambda n.\mathsf{Ty}^\exists\ (F_0\ n)\ (\mathsf{inv}\text{-}\pi\text{-}\mathsf{Ty}\ p\ n)\ r$
$\mathsf{Ty}^\exists\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0)\ p :\equiv \mathsf{transp}\ (T\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0))\ e\ h\ \_\ p$
  **where**
    $h : T\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0)\ \Gamma_0'$
    $h :\equiv \lambda\ \Gamma_1'\ p\ \Gamma'^D\ r\ \to\ \bar{\pi}^D\ \Gamma^D\ (\pi_1\ \mathsf{ih})\ (\pi_1\ \mathsf{ih}')\ , \bar{\pi}^R\ r\ (\pi_2\ \mathsf{ih})\ (\pi_2\ \mathsf{ih}')$
      **where**
        $ih :\equiv \mathsf{Ty}^\exists\ A_0\ (\mathsf{inv}\text{-}\bar{\pi}\text{-}\mathsf{Dom}\ p)\ r$
        $ih' :\equiv \mathsf{Ty}^\exists\ B_0\ (\mathsf{inv}\text{-}\bar{\pi}\text{-}\mathsf{Cod}\ p)\ (r \rhd^R \pi_2\ h)$
```

Figure 8.1: Detailed definition of $\mathsf{Ty}^\exists$

The definition of $\mathsf{Ty}_1\ \Gamma_0\ A_0$ is almost the same, with the addition of equational constraints between Γ_0 and A_0 which express when Γ_0 is a valid index for A_0:

$$\mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0') \qquad :\equiv \mathsf{Con}_1\ \Gamma_0' \times (\Gamma_0 = \Gamma_0')$$

$$\begin{aligned}
\mathsf{Ty}_1\ \Gamma_0\ (\hat{\pi}_0\ \Gamma_0'\ F_0) &:\equiv \mathsf{Con}_1\ \Gamma_0' \times ((n : \mathbb{N}) \to \mathsf{Ty}_1\ \Gamma_0'\ (F_0\ n)) \times (\Gamma_0 = \Gamma_0') \\
\mathsf{Ty}_1\ \Gamma_0\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0) &:\equiv \mathsf{Con}_1\ \Gamma_0' \times \mathsf{Ty}_1\ \Gamma_0'\ A_0 \times \mathsf{Ty}_1\ (\Gamma' \triangleright_0 A_0)\ B_0 \times (\Gamma_0 = \Gamma_0')
\end{aligned}$$

We can easily define all the expected constructors:

$$\begin{aligned}
&\bullet_1 :\equiv \mathsf{tt} && \hat{\pi}_1\ \Gamma_1\ F_1 :\equiv \Gamma_1, F_1, \mathsf{refl} \\
&\Gamma_1 \triangleright_1 A_1 :\equiv \Gamma_1, A_1 && \bar{\pi}_1\ \Gamma_1\ A_1\ B_1 :\equiv \Gamma_1, A_1, B_1, \mathsf{refl} \\
&\iota_1\ \Gamma_1 :\equiv \Gamma_1, \mathsf{refl}
\end{aligned}$$

We can also define all the inversion principles used in the encoding, by simple projection. We omit the details.

Eliminator relations are a bit more involved, but follow the same line of reasoning. We first recursively define the following intermediate relations:

$$\begin{aligned}
\mathsf{Con}^R_{\mathsf{rec}} &: (\Gamma_0 : \mathsf{Con}_0)(\Gamma_1 : \mathsf{Con}_1) \to \mathsf{Con}^D\ (\Gamma_0, \Gamma_1) \to \mathbf{Type} \\
\mathsf{Ty}^R_{\mathsf{rec}} &: \{\Gamma : \mathsf{Con}\}(A_0 : \mathsf{Ty}_0)(A_1 : \mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0)(\Gamma^D : \mathsf{Con}^D\ \Gamma) \\
&\quad \to \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1) \to \mathbf{Type}
\end{aligned}$$

Defining $\mathsf{Con}^R_{\mathsf{rec}}$ is relatively straightforward: we relate a context Γ and a displayed context Γ^D whenever there are displayed terms for each sub-component of Γ which are also related. Moreover, if Γ is obtained from some constructor, then Γ^D should be also obtained via the function in the displayed algebra corresponding to that constructor. For example, the context $(\Gamma \triangleright A)$ and some displayed object x^D are related whenever x^D is actually equal to $\Gamma^D \triangleright^D A^D$ for some Γ^D, A^D, such that Γ is related to Γ^D and A is related to A^D. The notion of relatedness for each of the sub-components is available via inductive hypothesis, since the statements pertain recursively smaller objects.

On the other hand, when defining $\mathsf{Ty}^R_{\mathsf{rec}}\,\{\Gamma\}\,A_0\,A_1\,\Gamma^D\,A^D$, we encounter the same issue that we had when defining $\mathsf{Ty}^{\exists}$, in that when doing induction on the input erased type A_0, this exposes a discrepancy between the context Γ obtained as input to the function itself, and the context exposed when pattern-matching on A_0.

For example, suppose we are given contexts $\Gamma : \mathsf{Con}, \Gamma^D : \mathsf{Con}^D\ \Gamma$, and types $(A_0, A_1) : \mathsf{Ty}\ \Gamma, A^D : \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1)$, and that by case analysis we discover that $A_0 \equiv \iota_0\ \Gamma_0'$. The natural way to define this case of the relation is proceed

```
$T :\equiv \lambda\, A_0\, \Gamma_0\, .\, \forall\, \Gamma_1\, A_1\, (\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1)) \to \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1) \to \mathbf{Type}$
$x \overset{q}{=}\ y :\equiv \mathsf{transp}\ (\lambda\, z\, .\, \mathsf{Ty}^D\ z\ \_)\ q\ x = y$

$\mathsf{Ty}^R_{\mathsf{rec}}\ (\iota_0\ \Gamma_0)\ p :\equiv$
  $\mathsf{transp}\ (T\ (\iota_0\ \Gamma_0))\ e\ h\ \_\ p$
  where
    $e :\equiv \mathsf{sym}\ (\mathsf{inv}\text{-}\iota_1\ p)$
    $h : T\ (\iota_0\ \Gamma_0)\ \Gamma_0$
    $h\ \Gamma_1\ p\ x^D\ y^D :\equiv$
      $(\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1))\times$
      $\mathsf{Con}^R_{\mathsf{rec}}\ \Gamma_0\ \Gamma_1\ \Gamma^D\times$
      $(q : \Gamma^D = x^D) \times \iota^D\ \Gamma^D \overset{q}{=}\ y^D$

$\mathsf{Ty}^R_{\mathsf{rec}}\ (\hat{\pi}_0\ \Gamma_0\ F_0)\ p :\equiv$
  $\mathsf{transp}\ (T\ (\hat{\pi}_0\ \Gamma_0\ F_0))\ e\ h\ \_\ p$
  where
    $e :\equiv \mathsf{sym}\ (\mathsf{inv}\text{-}\hat{\pi}_1\ p)$
    $h : T\ (\hat{\pi}_0\ \Gamma_0\ F_0)\ \Gamma_0$
    $h\ \Gamma_1\ p\ x^D\ y^D :\equiv$
      $(\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1))\times$
      $(F^D : \forall n.\mathsf{Ty}^D\ \Gamma^D\ (F_0\, n, F_1\, n))\times$
      $\mathsf{Con}^R_{\mathsf{rec}}\ \Gamma_0\ \Gamma_1\ \Gamma^D\times$
      $(\forall n.\mathsf{Ty}^R_{\mathsf{rec}}\ (F_0\, n)\ (F_1\, n)\ \Gamma^D\ (F^D\, n))\times$
      $(q : \Gamma^D = x^D) \times \hat{\pi}^D\ \Gamma^D\ F^D \overset{q}{=}\ y^D$
      where $F_1 :\equiv \mathsf{inv}\text{-}\hat{\pi}_1\text{-}\mathsf{Ty}\ p$

$\mathsf{Ty}^R_{\mathsf{rec}}\ (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0)\ p :\equiv$
  $\mathsf{transp}\ (T\ (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0))\ e\ h\ \_\ p$
  where
    $e :\equiv \mathsf{sym}\ (\mathsf{inv}\text{-}\bar{\pi}_1\ p)$
    $h : T\ (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0)\ \Gamma_0$
    $h\ \Gamma_1\ p\ x^D\ y^D :\equiv$
      $(\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1))\times$
      $(A^D : \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1))\times$
      $(B^D : \mathsf{Ty}^D\ (\Gamma^D \triangleright^D A^D)\ (B_0, B_1))\times$
      $\mathsf{Con}^R_{\mathsf{rec}}\ \Gamma_0\ \Gamma_1\ \Gamma^D\times$
      $\mathsf{Ty}^R_{\mathsf{rec}}\ A_0\ A_1\ \Gamma^D\ A^D\times$
      $\mathsf{Ty}^R_{\mathsf{rec}}\ B_0\ B_1\ (\Gamma^D \triangleright^D A^D)\ B^D\times$
      $(q : \Gamma^D = x^D) \times \bar{\pi}^D\ \Gamma^D\ A^D\ B^D \overset{q}{=}\ y^D$
      where
        $A_1 :\equiv \mathsf{inv}\text{-}\bar{\pi}_1\text{-}\mathsf{Dom}\ p$
        $B_1 :\equiv \mathsf{inv}\text{-}\bar{\pi}_1\text{-}\mathsf{Cod}\ p$
```

Figure 8.2: Recursive eliminator relations for Con/Ty

recursively, adding a constraint $A^D = \iota^D\ \Gamma^D$, and requiring that (Γ'_0, Γ'_1) be related to Γ^D.

$$\mathsf{Ty}^R_{\mathsf{rec}}\ \{\Gamma\}\ (\iota_0\ \Gamma'_0)\ A_1\ \Gamma^D\ A^D :\equiv \mathsf{Con}^R_{\mathsf{rec}}\ (\Gamma'_0, \Gamma'_1)\ \Gamma^D \times A^D = \iota^D\ \Gamma^D$$

However, for this to typecheck we need to establish that $\pi_1\ \Gamma = \Gamma'_0$. Luckily this equation follows by inversion on A_1; we transport along it to change the goal type to one where these two contexts actually match.

Figure 8.2 shows the gory details of these two definitions. Note that we pull out the type of $\mathsf{Ty}^R_{\mathsf{rec}}$ and explicitly define it as T, so that we can easily transport over it. The definition of $\mathsf{Ty}^R_{\mathsf{rec}}$ thus looks very similar to that of $\mathsf{Con}^R_{\mathsf{rec}}$, with the exception of the extra step of transporting along the appropriate equality between erased contexts. Moreover, since $\mathsf{Ty}^R_{\mathsf{rec}}$ relates both Con^D and Ty^D variables, we have one more equational constraint compared to $\mathsf{Con}^R_{\mathsf{rec}}$.

We can now recover the expected types and constructors of the eliminator relations from the recursively defined ones.

$$\begin{aligned}
&\mathsf{Con}^R\ (\Gamma_0, \Gamma_1)\ \Gamma^D &&:\equiv \mathsf{Con}^R_{\mathsf{rec}}\ \Gamma_0\ \Gamma_1\ \Gamma^D \\
&\mathsf{Ty}^R\ \Gamma^D\ (A_0, A_1)\ A^D &&:\equiv \mathsf{Ty}^R_{\mathsf{rec}}\ A_0\ A_1\ \Gamma^D\ A^D \\
&\bullet^R &&:\equiv \mathsf{refl} \\
&r \rhd^R r' &&:\equiv \_, \_, r, r', \mathsf{refl} \\
&\iota^R\ r &&:\equiv \_, r, \mathsf{refl}, \mathsf{refl} \\
&\bar{\pi}^R\ r_\Gamma\ r_A\ r_B &&:\equiv \_, \_, \_, r_\Gamma, r_A, r_B, \mathsf{refl}, \mathsf{refl} \\
&\hat{\pi}^R\ r\ r' &&:\equiv \_, \_, r, r', \mathsf{refl}, \mathsf{refl}
\end{aligned}$$

We have shown the recursive definitions of Con^R and Ty^R in painstaking detail, which makes them look quite technical and perhaps more complicated than they actually, conceptually are. In fact, they look closer to what a *code generator* would produce. This is on purpose, and in anticipation of Chapter 12 where we will indeed define algorithmic procedures generalizing this recursive definition of predicate and relation types to arbitrary IIT specifications. The way we presented this concrete example here will hopefully contribute to clarify how the general procedure directly relates to, and generalizes it.

8.2 Example: the setoid universe IIT

Despite its extreme simplicity, the Con/Ty example allows us to talk about non-trivial nuances of the reduction that realistically apply to many examples of induction-induction. Nevertheless, it is still fair to expect "real-world" instances of induction-induction to be generally more complex than that; we might then wonder whether this reduction still works for more complex use cases, and moreover if there are any interesting examples of infinitary IITs taking place in a metatheory without extensionality, which would therefore benefit from the reduction method presented here.

The setoid universe IIT from Section 6.6 provides an anecdotal answer to both questions. The context of this construction is the strict setoid model of type theory, originally presented in [Alt99] as a way to justify extensionality principles in intensional type theory, and later revisited as a type-theoretic syntax in [ABKT19]. One of the appeals of the strict setoid model is that it can be defined in an intensional type theoretic metatheory without function extensionality, thus effectively representing a way to "compile" extensionality into a small computational core.

Chapter 6 tackled the issue of extending the setoid model, and thus SeTT, with a universe of setoids, in a way that does not sacrifice the philosophical appeal of a minimal, intensional, core metatheory. Particularly, we did not want to extend the metatheory with inductive definitions schemas beyond inductive families, let alone *funext*.

The answer we reached was to define the universe in terms of a certain infinitary inductive-inductive type which we encoded in the ambient theory using inductive families.

$$\begin{aligned}&\mathsf{data}\ \mathsf{inU} : \mathbf{Type} \to \mathbf{Type}_1\\&\mathsf{data}\ \mathsf{inU}{\sim} : \{A\ A' : \mathbf{Type}\} \to \mathsf{inU}\ A \to \mathsf{inU}\ A' \to (A \to A' \to \mathbf{Prop}) \to \mathbf{Type}_1\end{aligned}$$

We now equip these types with the expected elimination principle, by encoding its general eliminators.

We begin with presenting the erased types $\mathsf{inU}_p/\mathsf{inU}{\sim}_p$ and well-formedness predicates $\mathsf{inU}_t/\mathsf{inU}{\sim}_t$, which are systematically generated from the signature of $\mathsf{inU}/\mathsf{inU}{\sim}$[1].

$$\begin{aligned}&\mathsf{data}\ \mathsf{inU}_p : \mathbf{Type} \to \mathbf{Type}_1\\&\mathsf{data}\ \mathsf{inU}{\sim}_p : (A_0\ A_1 : \mathbf{Type})(A_{01} : A_0 \to A_1 \to \mathbf{Prop}) \to \mathbf{Type}_1\end{aligned}$$

$$\begin{aligned}&\mathsf{data}\ \mathsf{inU}_t : \mathsf{inU}_p\ A \to \mathbf{Prop}_1\\&\mathsf{data}\ \mathsf{inU}{\sim}_t : (a_0 : \mathsf{inU}_p\ A_0)(a_1 : \mathsf{inU}_p\ A_1) \to \mathsf{inU}{\sim}_p\ A_0\ A_1\ A_{01} \to \mathbf{Prop}_1\end{aligned}$$

Figure 8.3 shows their term constructors.

The types of the target IIT are recovered via Σ-construction:

$$\mathsf{inU}\ X :\equiv \Sigma\,(x_0 : \mathsf{inU}_p\ X)\ (\mathsf{inU}_t\ x_0)$$

[1]We use subscript p and t, instead of *0* and *1*, to mark erased and well-formedness objects respectively, to avoid clashing with other uses of subscript *0* and *1*.

$$\mathsf{inU}{\sim}\ a_0\ a_1\ A_{01} :\equiv \Sigma\,(a_{01} : \mathsf{inU}{\sim}_p\ A_0\ A_1\ A_{01})\ (\mathsf{inU}{\sim}_t\ (\pi_1\ a_0)\ (\pi_1\ a_1)\ a_{01})$$

Their constructors are also easily recovered from the erased/well-formedness components:

$$\begin{aligned}
&\mathsf{bool} :\equiv \mathsf{bool}_p\ ,\ \mathsf{bool}_t\\
&\mathsf{pi}\ a\ a_{01}\ b\ b_{01} :\equiv \_\ ,\ \mathsf{pi}_t\ (\pi_2\ a)\ (\pi_2\ a_{01})\ (\pi_2 \circ b)\ (\pi_2 \circ b_{01})\\
&\mathsf{bool}{\sim} :\equiv \mathsf{bool}{\sim}_p, \mathsf{bool}{\sim}_t\\
&\mathsf{pi}{\sim}\ \{a_0\,a_{00}\,a_1\,a_{11}\,b_0\,b_{00}\,b_1\,b_{11}\}\ a_{01}\,b_{01} :\equiv\\
&\quad \_\ ,\ \mathsf{pi}{\sim}_t\,(\pi_2\,a_0)\,(\pi_2\,a_{00})\,(\pi_2\,a_1)\,(\pi_2\,a_{11})\,(\pi_2\,a_{01})\\
&\qquad\qquad (\pi_2 \circ b_0)\,(\pi_2 \circ b_{00})\,(\pi_2 \circ b_1)\,(\pi_2 \circ b_{11})\,(\pi_2 \circ b_{01})
\end{aligned}$$

We now show that $\mathsf{inU}/\mathsf{inU}{\sim}$ thus encoded supports the expected *general eliminators*. To do this, we define a notion of *displayed algebra* over $\mathsf{inU}/\mathsf{inU}{\sim}$, as well as eliminator relations between $\mathsf{inU}/\mathsf{inU}{\sim}$ and an arbitrary displayed algebra. Figure 8.4 provides the signatures of these components.

Assuming we are given an arbitrary displayed algebra $\mathsf{inU}^D, \mathsf{inU}{\sim}^D$, and defined eliminator relations $\mathsf{inU}^R, \mathsf{inU}{\sim}^R$ as per Figure 8.4, our goal is now to prove that the eliminator relations have the left-totality property, that is, for any element on the left of the relation there exists a related element on the right:

$$\begin{aligned}
&\mathsf{inU}^{\exists}\ \ : (a : \mathsf{inU}\,A) \to \Sigma(\mathsf{inU}^D\,a)(\mathsf{inU}^R\,a)\\
&\mathsf{inU}{\sim}^{\exists} : (r_0 : \mathsf{inU}^R\,x_0\,x_0^D)(r_1 : \mathsf{inU}^R\,x_1\,x_1^D)(x_{01} : \mathsf{inU}{\sim}\,x_0\,x_1\,X_{01})\\
&\qquad \to \Sigma(\mathsf{inU}{\sim}^D\,x_0^D\,x_1^D\,x_{01})(\mathsf{inU}{\sim}^R\,x_{01})
\end{aligned}$$

Let us take a look at the proof of $\mathsf{inU}^{\exists}$. The boolean case is immediate.

$$\mathsf{inU}^{\exists}\ (\mathsf{bool}_p\ ,\ q) :\equiv \_\ ,\ \mathsf{bool}^R$$

For the function case, we define

$$\mathsf{inU}^{\exists}\ (\mathsf{pi}_p\,a_p\,a_{01p}\,b_p\,b_{01p}\ ,\ q) :\equiv \_\ ,\ \mathsf{pi}^R\ (\pi_2\ h_a)\ (\pi_2\ h_{a_{01}})\ (\pi_2 \circ h_b)\ (\pi_2 \circ h_{b_{01}})$$

by first deriving the well-formedness proofs $a_t, a_{01t}, b_t, b_{01t}$ by inversion on q, and subsequently applying the inductive hypothesis:

$$\begin{aligned}
&h_a :\equiv \mathsf{inU}^{\exists}\ (a_p\ ,\ a_t)\\
&h_{a_{01}} :\equiv \mathsf{inU}{\sim}^{\exists}\ (\pi_2\ h_a)\ (\pi_2\ h_a)\ (a_{01p}\ ,\ a_{01t})
\end{aligned}$$

$$h_b :\equiv \lambda\, x \,.\, \mathsf{inU}^{\exists}\, (b_p\, x \ ,\ b_t\, x)$$
$$h_{b_{01}} :\equiv \lambda\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}^{\exists}\, (\pi_2\, (h_b\, x_0))\, (\pi_2\, (h_b\, x_1))\, (b_{01p}\, x_{01} \ ,\ b_{01t}\, x_{01})$$

The boolean case of $\mathsf{inU}{\sim}^{\exists}$ is, again, straightforward:

$$\mathsf{inU}{\sim}^{\exists}\ \mathsf{bool}^R\ \mathsf{bool}^R\ (\mathsf{bool}{\sim}_p\ ,\ q) :\equiv \_\ ,\ \mathsf{bool}{\sim}^R$$

Note that we pattern match on the input proofs of relatedness r_0, r_1 to expose them as build our of the constructor bool^R.

The clause for $\mathsf{pi}{\sim}$ requires some more work.

$$\mathsf{inU}{\sim}^{\exists}\ \{x_0\ x_1\ x_0^D\ x_1^D\}\ r_0\ r_1\ (\mathsf{pi}{\sim}_p\ a_{0p}\, a_{00p}\, a_{1p}\, a_{11p}\, a_{01p}\, b_{0p}\, b_{00p}\, b_{1p}\, b_{11p}\, b_{01p}\ ,\ q) :\equiv\ ?$$

Let

$$\begin{aligned} a_0 &:\equiv (a_{0p}, a_{0t}) & b_0 &:\equiv \lambda\, x \,.\, (b_{0p}\, x\ ,\ b_{0t}\, x) \\ a_{00} &:\equiv (a_{00p}, a_{00t}) & b_{00} &:\equiv \lambda\, \{x_0\, x_1\}\, x_{01} \,.\, (b_{00p}\, x_{01}\ ,\ b_{00t}\, x_{01}) \\ a_1 &:\equiv (a_{1p}, a_{1t}) & b_1 &:\equiv \lambda\, x \,.\, (b_{1p}\, x\ ,\ b_{1t}\, x) \\ a_{11} &:\equiv (a_{11p}, a_{11t}) & b_{11} &:\equiv \lambda\, \{x_0\, x_1\}\, x_{01} \,.\, (b_{11p}\, x_{01}\ ,\ b_{11t}\, x_{01}) \\ a_{01} &:\equiv (a_{01p}, a_{01t}) & b_{01} &:\equiv \lambda\, \{x_0\, x_1\}\, x_{01} \,.\, (b_{01p}\, x_{01}\ ,\ b_{01t}\, x_{01}) \end{aligned}$$

where the well-formedness components $a_{0t}, a_{00t}, a_{1t}, a_{11t}, b_{0t}, b_{00t}, b_{1t}, b_{11t}, a_{01t}, b_{01t}$ are obtained from q by inversion.

We would like to inhabit the goal type via $\mathsf{pi}{\sim}^D$ and $\mathsf{pi}{\sim}^R$, however this would not type-check unless we can prove that the fixed variables x_0, x_1, x_0^D, x_1^D are themselves expressions obtained from the constructors pi and pi^D.

By inversion on q, we prove the equations

$$\pi_1\ x_0 = \mathsf{pi}_p\ a_{0p}\ a_{00p}\ b_{0p}\ b_{00p}$$
$$\pi_1\ x_1 = \mathsf{pi}_p\ a_{1p}\ a_{11p}\ b_{1p}\ b_{11p}$$

hence by proof-irrelevance, we have $a_0 = \mathsf{pi}\ a_0\ a_{00}\ b_0\ b_{00}$ and $a_1 = \mathsf{pi}\ a_1\ a_{11}\ b_1\ b_{11}$. Transporting r_0, r_1 along these equations allows us to do inversion on them, which exposes the following displayed terms and relatedness proofs

$$\begin{aligned} a_0^R &: \mathsf{inU}^R\, a_0\, a_0^D & b_0^R &: \forall\, x \,.\, \mathsf{inU}^R\, (b_0\, x)\, (b_0^D\, x) \\ a_{00}^R &: \mathsf{inU}{\sim}^R\, a_{00}\, a_{00}^D & b_{00}^R &: \forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}^R\, (b_{00}\, x_{01})\, (b_{00}^D\, x_{01}) \\ a_1^R &: \mathsf{inU}^R\, a_1\, a_1^D & b_1^R &: \forall\, x \,.\, \mathsf{inU}^R\, (b_1\, x)\, (b_1^D\, x) \end{aligned}$$

$$a^R_{11} : \mathsf{inU}{\sim}^R\, a_{11}\, a^D_{11} \qquad b^R_{11} \forall \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}^R\, (b_{11}\, x_{01})\, (b^D_{11}\, x_{01})$$

as well as the equations $x^D_0 = \mathsf{pi}^D\, a^D_0\, a^D_{00}\, b^D_0\, b^D_{00}$ and $x^D_1 = \mathsf{pi}^D\, a^D_1\, a^D_{11}\, b^D_1\, b^D_{11}$.

We have the inductive hypotheses

$$h_a :\equiv \mathsf{inU}{\sim}^\exists\, a^R_0\, a^R_1\, a_{01}$$
$$h_b :\equiv \lambda\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}^\exists\, (b^D_0\, x_0)\, (b^D_1\, x_1)\, (b_{01}\, x_{01})$$

from which we derive

$$a^D_{01} :\equiv \pi_1\, h_a \qquad b^D_{01} :\equiv \pi_1 \circ h_b$$
$$a^R_{01} :\equiv \pi_2\, h_b \qquad b^R_{01} :\equiv \pi_2 \circ h_b$$

We finally inhabit the goal type with the following pair

$$\mathsf{pi}{\sim}^D\, a^D_0\, a^D_{00}\, a^D_1\, a^D_{11}\, a^D_{01}\, b^D_0\, b^D_{00}\, b^D_1\, b^D_{11}\, b^D_{01}\ ,\ \mathsf{pi}{\sim}^R\, a^R_0\, a^R_{00}\, a^R_1\, a^R_{11}\, a^R_{01}\, b^R_0\, b^R_{00}\, b^R_1\, b^R_{11}\, b^R_{01}$$

The eliminators are, as usual, simple projections out of the proofs of left-totality. It is immediate to verify that the expected β-equalities hold by definition.

$$\mathsf{elim}_{\mathsf{inU}} : (a : \mathsf{inU}\, A) \to \mathsf{inU}^D\, a$$
$$\mathsf{elim}_{\mathsf{inU}}\, a :\equiv \pi_1\, (\mathsf{inU}^\exists\, a)$$
$$\mathsf{elim}_{\mathsf{inU}\sim} : (a_{01} : \mathsf{inU}{\sim}\, a_0\, a_1\, A_{01}) \to \mathsf{inU}{\sim}^D\, (\mathsf{elim}_{\mathsf{inU}}\, a_0)\, (\mathsf{elim}_{\mathsf{inU}}\, a_1)\, a_{01}$$
$$\mathsf{elim}_{\mathsf{inU}\sim}\, \{a_0\, a_1\}\, a_{01} :\equiv \pi_1\, (\mathsf{inU}{\sim}^\exists\, (\pi_2\, (\mathsf{inU}^\exists\, a_0))\, (\pi_2\, (\mathsf{inU}^\exists\, a_1))\, a_{01})$$

$$\begin{array}{ll}
\mathsf{bool}_p & : \mathsf{inU}_p\ \mathbb{2} \\
\mathsf{pi}_p & : \mathsf{inU}_p\ A \to \mathsf{inU}{\sim}_p\ A\ A\ A_{01} \to (\forall\, x \,.\, \mathsf{inU}_p\ (B\, x)) \\
& \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_p\, (B\, x_0)\, (B\, x_1)\, (B_{01}\, x_{01})) \\
& \to \mathsf{inU}_p\, (\mathsf{Fun}\, A\, B\, A_{01}\, B_{01}) \\
\mathsf{bool}{\sim}_p & : \mathsf{inU}{\sim}_p\ \mathbb{2}\ \mathbb{2}\ (\_ \overset{?}{=}_{\mathbb{2}} \_) \\
\mathsf{pi}{\sim}_p & : \mathsf{inU}_p\ A_0 \to \mathsf{inU}{\sim}_p\ A_0\ A_0\ A_{00} \to \mathsf{inU}_p\ A_1 \to \mathsf{inU}{\sim}_p\ A_1\ A_1\ A_{11} \\
& \to \mathsf{inU}{\sim}_p\ A_0\ A_1\ A_{01} \\
& \to (\forall\, x \,.\, \mathsf{inU}_p\ (B_0\, x)) \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_p\, (B_0\, x_0)\, (B_0\, x_1)\, (B_{00}\, x_{01})) \\
& \to (\forall\, x \,.\, \mathsf{inU}_p\ (B_1\, x)) \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_p\, (B_1\, x_0)\, (B_1\, x_1)\, (B_{11}\, x_{01})) \\
& \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_p\, (B_0\, x_0)\, (B_1\, x_1)\, (B_{01}\, x_{01})) \\
& \to \mathsf{inU}{\sim}_p\ (\mathsf{Fun}\, A_0\, B_0\, A_{00}\, B_{00})\ (\mathsf{Fun}\, A_1\, B_1\, A_{11}\, B_{11}) \\
& \qquad\qquad (\lambda f_0\ f_1 \,.\, \forall (x_0\ x_1) \to A_{01}\ x_0\ x_1 \to B_{01}\ x_{01}\ (\pi_1\ f_0\ x_0)\ (\pi_1\ f_1\ x_1)) \\
\mathsf{bool}_t & : \mathsf{inU}_t\ \mathsf{bool}_p \\
\mathsf{pi}_t & : \mathsf{inU}_t\ a \to \mathsf{inU}{\sim}_t\ a\ a\ a_{01} \to (\forall\, x \,.\, \mathsf{inU}_t\, (b\, x)) \\
& \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_t\, (b\, x_0)\, (b\, x_1)\, (b_{01}\, x_{01})) \\
& \to \mathsf{inU}_t\, (\mathsf{pi}_p\ a\ a_{01}\ b\ b_{01}) \\
\mathsf{bool}{\sim}_t & : \mathsf{inU}{\sim}_t\ \mathsf{bool}_p\ \mathsf{bool}_p\ \mathsf{bool}{\sim}_p \\
\mathsf{pi}{\sim}_t & : \mathsf{inU}_t\ a_0 \to \mathsf{inU}{\sim}_t\ a_0\ a_0\ a_{00} \to \mathsf{inU}_t\ a_1 \to \mathsf{inU}{\sim}_t\ a_1\ a_1\ a_{11} \\
& \to \mathsf{inU}{\sim}_t\ a_0\ a_1\ a_{01} \\
& \to (\forall\, x \,.\, \mathsf{inU}_t\, (b_0\, x)) \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_t\, (b_0\, x_0)\, (b_0\, x_1)\, (b_{00}\, x_{01})) \\
& \to (\forall\, x \,.\, \mathsf{inU}_t\, (b_1\, x)) \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_t\, (b_1\, x_0)\, (b_1\, x_1)\, (b_{11}\, x_{01})) \\
& \to (\forall\, \{x_0\, x_1\}\, x_{01} \,.\, \mathsf{inU}{\sim}_t\, (b_0\, x_0)\, (b_1\, x_1)\, (b_{01}\, x_{01})) \\
& \to \mathsf{inU}{\sim}_t\ (\mathsf{pi}_p\ a_0\ a_{00}\ b_0\ b_{00})\ (\mathsf{pi}_p\ a_1\ a_{11}\ b_1\ b_{11}) \\
& \qquad\qquad (\mathsf{pi}{\sim}_p\ a_0\ a_{00}\ a_1\ a_{11}\ a_{01}\ b_0\ b_{00}\ b_1\ b_{11}\ b_{01})
\end{array}$$

where $\mathsf{Fun}\, A\, B\, A_{01}\, B_{01} :\equiv \Sigma\ (f : (x : A) \to B\ x)(\forall \{x_0\ x_1\}\, x_{01} \,.\, B_{01}\ x_{01}\ (f\ x_0)\ (f\ x_1))$

Figure 8.3: Constructors of the erased and predicate types generated by $\mathsf{inU}, \mathsf{inU}{\sim}$

$$
\begin{array}{ll}
\mathsf{inU}^D & : \mathsf{inU}\,A \to \mathbf{Type} \\
\mathsf{inU}{\sim}^D & : \mathsf{inU}^D\,a_0 \to \mathsf{inU}^D\,a_1 \to \mathsf{inU}{\sim}\,a_0\,a_1\,A_{01} \to \mathbf{Type} \\
\mathsf{bool}^D & : \mathsf{inU}^D\,\mathsf{bool} \\
\mathsf{pi}^D & : (a^D : \mathsf{inU}^D\;a) \to \mathsf{inU}{\sim}^D\,a^D\,a^D\,a_{01} \to (b^D : \forall\,x\,.\,\mathsf{inU}^D\,(b\,x)) \\
& \to (\forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^D\,(b^D\,x_0)\,(b^D\,x_1)\,(b_{01}\,x_{01})) \\
& \to \mathsf{inU}^D\,(\mathsf{pi}\;a\;a_{01}\;b\;b_{01}) \\
\mathsf{bool}{\sim}^D & : \mathsf{inU}{\sim}^D\,\mathsf{bool}^D\,\mathsf{bool}^D\,\mathsf{bool}{\sim} \\
\mathsf{pi}{\sim}^D & : (a_0^D : \mathsf{inU}^D\;a_0)(a_{00}^D : \mathsf{inU}{\sim}^D\,a_0^D\,a_0^D\,a_{00})(a_1^D : \mathsf{inU}^D\;a_1)(a_{11}^D : \mathsf{inU}{\sim}^D\,a_1^D\,a_1^D\,a_{11}) \\
& \quad (a_{01}^D : \mathsf{inU}{\sim}^D\,a_0^D\,a_1^D\,a_{01}) \\
& \quad (b_0^D : \forall\,x\,.\,\mathsf{inU}^D\,(b_0\,x))\,(b_{00}^D : \forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^D\,(b_0^D\,x_0)\,(b_0^D\,x_1)\,(b_{00}\,x_{01})) \\
& \quad (b_1^D : \forall\,x\,.\,\mathsf{inU}^D\,(b_1\,x))\,(b_{11}^D : \forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^D\,(b_1^D\,x_0)\,(b_1^D\,x_1)\,(b_{11}\,x_{01})) \\
& \quad (b_{01}^D : \forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^D\,(b_0^D\,x_0)\,(b_1^D\,x_1)\,(b_{01}\,x_{01})) \\
& \to \mathsf{inU}{\sim}^D\;(\mathsf{pi}^D\,a_0^D\,a_{00}^D\,b_0^D\,b_{00}^D)\,(\mathsf{pi}^D\,a_1^D\,a_{11}^D\,b_1^D\,b_{11}^D)\,(\mathsf{pi}{\sim}\,a_{01}\,b_{01})
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{inU}^R & : (a : \mathsf{inU}\,A) \to \mathsf{inU}^D\,a \to \mathbf{Type} \\
\mathsf{inU}{\sim}^R & : \forall\{a_0^D\;a_1^D\}(a_{01} : \mathsf{inU}{\sim}\,a_0\,a_1\,A_{01}) \to \mathsf{inU}{\sim}^D\,a_0^D\,a_1^D\,a_{01} \to \mathbf{Type} \\
\mathsf{bool}^R & : \mathsf{inU}^R\,\mathsf{bool}\,\mathsf{bool}^D \\
\mathsf{pi}^R & : \mathsf{inU}^R\;a\;a^D \to \mathsf{inU}{\sim}^R\;a_{01}\;a_{01}^D \\
& \to (\forall\,x\,.\,\mathsf{inU}^R\;(b\,x)\,(b^D\,x)) \to (\forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^R\,(b_{01}\,x_{01})\,(b_{01}^D\,x_{01})) \\
& \to \mathsf{inU}^R\;(\mathsf{pi}\,a\,a_{01}\,b\,b_{01})\,(\mathsf{pi}^D\;a^D\;a_{01}^D\;b^D\;b_{01}^D) \\
\mathsf{bool}{\sim}^R & : \mathsf{inU}{\sim}^R\,\mathsf{bool}^D\,\mathsf{bool}^D\,\mathsf{bool}{\sim}\,\mathsf{bool}{\sim}^D \\
\mathsf{pi}{\sim}^R & : \mathsf{inU}^R\,a_0\,a_0^D \to \mathsf{inU}{\sim}^R\,a_{00}\,a_{00}^D \to \mathsf{inU}^R\,a_1\,a_1^D \to \mathsf{inU}{\sim}^R\,a_{11}\,a_{11}^D \to \mathsf{inU}{\sim}^R\,a_{01}\,a_{01}^D \\
& \to (\forall\,x\,.\,\mathsf{inU}^R\,(b_0\,x)\,(b_0^D\,x)) \to (\forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^R\,(b_{00}\,x_{01})\,(b_{00}^D\,x_{01})) \\
& \to (\forall\,x\,.\,\mathsf{inU}^R\,(b_1\,x)\,(b_1^D\,x)) \to (\forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^R\,(b_{11}\,x_{01})\,(b_{11}^D\,x_{01})) \\
& \to (\forall\,\{x_0\,x_1\}\,x_{01}\,.\,\mathsf{inU}{\sim}^R\,(b_{01}\,x_{01})\,(b_{01}^D\,x_{01})) \\
& \to \mathsf{inU}{\sim}^R\,(\mathsf{pi}{\sim}\,a_{01}\,b_{01})\,(\mathsf{pi}{\sim}^D\;a_0\,a_{00}^D\,a_1^D\;a_{11}^D\,a_{01}^D\,b_0^D\,b_{00}^D\,b_1^D\,b_{11}^D\,b_{01}^D)
\end{array}
$$

Figure 8.4: Specification of displayed algebras and eliminator relations

Chapter 9

Generalizing the encoding

Chapter 8 demonstrated a method of encoding infinitary IITs in a suitable extension of intensional MLTT, by way of two concrete examples. We begun in Section 8.1 with an infinitary version of the Con/Ty IIT, which served as an in-depth example of the encoding method targeting a simple yet non-trivial infinitary IIT. We then looked at the detailed encoding of the universe IIT first defined in Section 6.6, as an example of a complex, "real-world" case.

It is natural to ask ourselves whether the same construction can be carried out for different IITs. We would like to identify a sub-class of infinitary IITs that are amenable to encoding via our method, and give a *general* constructive proof that this reduction can be applied systematically to arbitrary types in that sub-class. We will do so in the chapters that follow.

We begin by pointing out the kind of IITs that we may not be able to encode with our method. We then lay out some sufficient (although not necessary) conditions for our method to work (Section 9.1), and define the target class of IITs accordingly (Section 9.2).

A general reduction from a class of type to another is a meta-theorem about a particular theory — the *target* or *encoding* theory where the encoding takes place — and as such it usually cannot be stated nor proved *inside* that theory itself; instead, one can work in a *metatheory* where the target theory exists as a first-class entity and can become the subject of meta-theorems of interest, which are stated externally.

We discuss the distinction between the so-called *meta-level* and *target-level* in Section 9.3. We define the "ambient" metatheory where all our reasoning takes place. Inside the metatheory, we define a datatype of *specifications*, which formalizes the class of IITs that we intend to target in the general reduction proof (Section 9.4). We then define the target theory, or rather its models. The idea is to prove the reduction as a property of target-theory models: for any model and IIT specification, there exist semantic types and terms corresponding to the specified IIT.

The chapters that follow will make precise what we mean by IIT in the general case. In the examples considered so far the target IIT was fixed, so it was clear what types, constructors, and eliminators we needed to encode. When generalizing the reduction, however, the target IIT is represented by some abstract specification, i.e. an element of the aforementioned specification datatype. Before even beginning to talk about encoding such IIT in the target theory, we need to define what it means for this IIT to exist, i.e. what are its types, constructors, and eliminators. In the tradition of theoretical studies of data types, we do this by assigning a notion of *algebra* indexed by IIT specifications: for any specification we obtain a type of algebras, and take the specified IIT to be one particular instance satisfying a suitable property, which in our case will turn out to be *section induction* (Chapter 10.)

Having made clear what constitutes a target-level IIT for any given specification, we then focus on the building blocks of the encoding: these are the erased types and the well-formedness predicates that we have familiarized ourselves with in the previous chapters. The idea of the reduction was as follows: given suitable erased and predicate types, we defined the types and constructors of the target IIT; moreover, from the induction principle of the erased types and the inversion principles of the predicates we derived eliminators for the target IIT.

We generalize the concept of erased type and well-formedness predicate similarly to how we generalized the notion of IITs, that is, we define the notion of *erased algebras* and *predicate algebras* indexed by specifications. The general encoding method thus becomes a construction on algebras: given a pair of an erased algebra and a predicate algebra for some specification, we construct an IIT algebra (Chapter 11); moreover, we prove that if the input algebras are equipped with suitable induction and inversion principles, then we can derive an induction principle for the IIT algebra so constructed (Chapter 12).

9.1 Scope of the encoding

We take a closer look at what made the reduction in Section 8.1 work, in an attempt to identify some sufficient conditions, on both the encoding theory and the subclass of infinitary IITs that we are going to target, that ensure the applicability of the encoding method for arbitrary IITs.

Propositionality of the predicates In the finitary case, propositionality of the well-formedness predicates can be proved *a posteriori*, by induction/pattern matching on the erased types. Our work, however, targets potentially infinitary types, for which such a proof requires function extensionality in general. In order to escape *funext* and thus fulfill our requirements, we instead defined the predicates inside **Prop**, thus making propositionality hold by definition.

This convenience doesn't come for free: we now cannot use inversion or pattern matching on proof terms of the predicate types in proof-relevant contexts, in particular when constructing the proofs of left-totality of the eliminator relations. As we have seen in the examples, this limitation can be circumvented by equipping the encoding type theory with a strictly propositional identity type with a strong transport rule allowing elimination into proof-relevant types.

A pleasant side-effect of using **Prop**-valued well-formedness predicates is that the β-laws of the encoded IIT eliminators hold *definitionally*. This does not seem to be the case for (finitary) encodings where propositionality of the predicates is proved a posteriori up to propositional equality, like in [vR19].

A possible alternative solution to the issue could have been to define the well-formedness predicates as higher inductive types (HITs), with propositional truncation as a path constructor, however this would still lead to weak β-laws. We have not investigated this option enough to comment on its viability.

Right-uniqueness and non-linearity The reduction method discussed in [vR19] comprises the important step of proving functionality of the eliminator relations, i.e. left-totality and right-uniqueness. On the other hand, in our examples we never prove nor need right-uniqueness of the eliminator relations to prove their left-totality, and therefore to construct the eliminators. This aspect is due to the particular IIT specifications considered in those examples, and our reduction method crucially relies on it, since proving right-uniqueness requires function extensionality in the infinitary case.

The fact that we were able to make do without proving right-uniqueness of the eliminator relations in all our examples suggests that there exists a subclass of infinitary IITs for which this proof is simply not needed. Characterizing this subclass is vital for of our work, as any IIT outside of it would inevitably require *funext* to be encoded, and therefore fall outside the scope of our method. Determining the precise boundaries of this subclass might be challenging, but we attempt at a partial answer to the question by identifying a property of IIT specifications that seems to necessitate right-uniqueness when encoding them. This property is the presence of constructors with a non-linear occurrence of variables in the codomain. The epitomical example is the reflexivity constructor of the inductively-defined identity type.

$$\mathsf{refl} : \{x : A\} \to x = x$$

In the type of refl, the variable x appears non-linearly in the codomain $x = x$, thus making the constructor non-linear in our sense.

To illustrate the issue with non-linear constructors, let us consider the previous Con/Ty type extended with a bogus constructors that make the definition non-linear:

$$\begin{aligned}&\mathsf{pair} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Con}\\ &\mathsf{non\text{-}lin} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ (\mathsf{pair}\ \Gamma\ \Gamma)\end{aligned}$$

Assume we are given a displayed algebra $\mathsf{Con}^D, \mathsf{Ty}^D$, which therefore comes with a term $\mathsf{non\text{-}lin}^D : (\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ (\mathsf{pair}^D\ \Gamma^D\ \Gamma^D)\ (\mathsf{non\text{-}lin}\ \Gamma)$. When proving left-totality of the eliminator relation Ty^R in the case of the $(\mathsf{non\text{-}lin}\,\Gamma)$ constructor, we are given as input a proof $r : \mathsf{Con}^R\ (\mathsf{pair}\ \Gamma\ \Gamma)\ \nabla^D$ for some $\nabla^D : \mathsf{Con}^D\ (\mathsf{pair}\ \Gamma\ \Gamma)$, and we need to show the existence of some $y^D : \mathsf{Ty}^D\ \nabla^D\ (\mathsf{non\text{-}lin}\,\Gamma)$ which is Ty^R-related to $\mathsf{non\text{-}lin}\,\Gamma$. We might attempt this by constructing $\Gamma^D : \mathsf{Con}^D\,\Gamma$ via inductive hypothesis on Γ and defining $y^D :\equiv \mathsf{non\text{-}lin}^D\,\Gamma^D$, however this does not typecheck since the Con^D-valued index of this expression, $\mathsf{pair}^D\,\Gamma^D\,\Gamma^D$, is not the same as the expected ∇^D.

Alternatively, we might try to proceed by inversion/pattern matching on r, from which we get two objects Γ_1^D, Γ_2^D and proofs $r_1 : \mathsf{Con}^R\ \Gamma\ \Gamma_1^D$ and $r_2 : \mathsf{Con}^R\ \Gamma\ \Gamma_2^D$, as well as $\nabla^D \equiv \mathsf{pair}^D\,\Gamma_1^D\,\Gamma_2^D$. Although now we know that ∇^D is a term constructed via displayed pairing pair^D, unfortunately this is now enough, as we cannot complete the proof without establishing that $\Gamma^D, \Gamma_1^D, \Gamma_2^D$ are all equal to each other. While this would certainly be possible to prove via right-uniqueness of Con^R, we do not see a way to prove it without.

Because of these issues with non-linear constructors, here we will focus on infinitary IITs that are linear. We leave open the question of whether a different reduction method can be applied to non-linear IITs.

Remark 9.1.1. We have defined the notion of linear constructor in a very syntactic way, in terms of variable occurrence in the codomain type. This definition is informal and might not fully capture what we mean. For example, take the constructor

$$\mathsf{non\text{-}lin\text{-}eq} : (\Gamma_0\ \Gamma_1 : \mathsf{Con}) \to \Gamma_0 = \Gamma_1 \to \mathsf{Ty}\ (\mathsf{pair}\ \Gamma_0\ \Gamma_1)$$

The constructor $\mathsf{non\text{-}lin\text{-}eq}$ should be considered non-linear, even though the free variables in its codomain are syntactically distinct. ■

In addition to linearity, another aspect of our reduction method that allows to prove left-totality of the relations absent of right-uniqueness is the *controlled* and *careful* use of the inductive hypothesis. To illustrate this, consider again the Con/Ty IIT with its constructor:

$\iota : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma$

To prove left-totality of Ty^R in the case of ι, we need to produce evidence of $\mathsf{Ty}^R\,(\iota\,\Gamma)\,(\iota^D\,\Gamma^D)$ for some Γ^D such that $r : \mathsf{Con}^R\ \Gamma\ \Gamma^D$. We are given two alternatives to construct the proof r: (1) use the term Γ^D and proof r that are already given to $\mathsf{Ty}^\exists$ as input, or (2) construct different $\Gamma^{D'}$ and $r' : \mathsf{Con}^R\ \Gamma\ \Gamma^{D'}$ by induction on Γ. Because Γ^D is fixed and appears as an index in the goal type, option 2 is only viable if we can prove that $\Gamma^D = \Gamma^{D'}$, which again needs right-uniqueness of Con^R.

We distill this example into a general rule that we shall follow when encoding the eliminators of an IIT of the form $A : \mathbf{Type}, B : A \to \mathbf{Type}$, for some displayed algebra A^D, B^D and eliminator relations A^R, B^R. The rule specifically applies when proving left-totality of B^R for the case of a constructor c applied to arguments $x_1, ..., x_n$: in this scenario, we are given as input a term $y^D : A^D\ y$ and a proof r relating $y :\equiv c\,x_1 \ldots x_n$ to y^D; to prove the goal, we need to construct displayed terms x_i^D and corresponding proofs r_i relating each x_i to x_i^D for all i. The rule is then the following:

> If x_i appears as an index in the type of y then the displayed algebra element x_i^D and the proof r_i that x_i and x_i^D are related must be obtained from the input relation r, either by taking r as-is or by inversion on it. Otherwise, x_i^D and r_i must be obtained by inductive hypothesis.

Adding this condition to the reduction method inevitably complicates its generalization to arbitrary IITs, since it means we have to introduce some form of conditional branching in the algorithm, which now has to make an informed decision based on the particular shape of the IIT constructors involved.

9.2 Target inductive-inductive types

In light of the previous paragraphs we now identify a suitable class of infinitary IITs to be targeted by our general reduction proof. This is a subclass of all infinitary IITs obtained by imposing certain restrictions, some of which we believe are necessary for the reduction method to work, while others are purely for the sake of simplifying the proofs and the exposition.

We consider (infinitary) IITs that are:

1. two-sorted, i.e. of the form $A : \mathbf{Type}, B : A \to \mathbf{Type}$, with no additional indices or parameters;

2. *linear* (see Definition 9.2.1);
3. small (i.e. with no universe quantification), and with limited access to external types

We define a *linear IIT* as follows:

Definition 9.2.1. A specification of a two-sorted IIT $A : \mathbf{Type}, B : A \to \mathbf{Type}$ is *linear* if the index of the conclusion type of all constructors of B is either a variable or a constructor of A applied to a linear list of variables, i.e. one where no variable appears more than once. ■

Condition (2) seems necessary for the reduction method to work, for the reasons discussed in the previous section. Note that the definition of linearity in Definition 9.2.1 is a bit more restrictive than what one would normally consider linear. In particular, we disallow nested constructors in addition to non-linear occurrences of variables, even though it would be possible to nest constructors and still result in a linear expression. The reason for this limitation is purely technical: this restricted formulation of linearity is much simpler to formalize, especially in a computer-assisted setting.

On the other hand, we believe that limiting ourselves to two sorts (as per Condition (1)) does not impact the generality of the reduction too much, as we can reduce any multi-sorted IIT to an equivalent two-sorted IIT via Szumi's reduction (see Section 3.1.) We do realize, however, that by Condition (2) many two-sorted IITs obtained via Szumi's reduction may fall out of the scope of our current implementation of the encoding method. Allowing for a more liberal notion of linearity that can cover those cases is certainly an important point for future work.

A consequence of Condition (1) is that we also do not allow sorts to be parameterized/indexed by external types. We believe the reduction method described in this thesis can be extended to various externally indexed IITs — the universe IIT encoding showcased in Section 8.2 is such an instance — although we leave this to future work.

By Condition (3), we restrict ourselves to small types, and only allow constant small types to be used in a constructor's list of parameters. For example, we don't allow a constructor for A of the form $c : (n : \mathbb{N}) \to \mathsf{Fin}\ n \to A$, although we allow the equivalent $c : \Sigma(n : \mathbb{N})(\mathsf{Fin}\ n) \to A$. We do not allow type-level applications for external types, so in particular constructors taking proofs of equality as arguments are not allowed. Note that this condition excludes, as expected, non-linear constructors like $\mathsf{non\text{-}lin\text{-}eq}$ shown above, which Definition 9.2.1 alone would fail to exclude.

Condition (3) is also included to simplify presentation, and again we believe extending our construction to more refined IIT specifications would be a technical challenge more than a conceptual one. One piece of anecdotal evidence for our claims about Conditions (1) and (3) is provided by the example from Section 8.2, where we successful reduce an IIT violating both conditions.

From now on, when we talk about inductive-inductive types, it will be implied we are referring to IITs in the specific subclass just outlined, which is the intended target of our reduction. The description above is informal and not sufficient to grasp all the details of the kind of IITs that we will be targeting. Things shall be clearer when we give a fully formal definition of IIT specifications later in Section 9.4.2.

9.3 Metatheory and target theory

Our aim for this part of the thesis is to show that we can encode the required types, constructors, and eliminators of any specifiable infinitary IIT inside any sufficiently equipped intensional type theory.

For individual concrete instances, like in the examples we have seen so far, it was enough to work directly in the intensional theory and simply exhibit the encoding of the IIT, whose specification was fixed and informally understood. In other words, we only ever had to deal with a single type theory, the intensional type theory where the encoding took place.

Such a limited point of view may work for a few concrete examples, but does not scale to our goal, which is to give a rigorous demonstration of this encoding that applies generically and parametrically for arbitrary IIT specifications. To give a generic account of this reduction, we need at the very least a notion of IIT specification over which to quantify, and a way to reason about the computational (that is, judgmental) behaviour of the encoded IITs. While the encoding of any concrete IIT is an internal construction in the intensional type theory of choice, the general reduction method and its proof of correctness is an external, metatheoretic statement *about* the target theory. Consequently, we shall carry out our results over two different levels:

- The *target theory* (also called *object theory* or *encoding theory*): this is the theory were the IIT encodings will take place. Concretely, it is an extension of intensional MLTT with universes for types and definitionally proof-irrelevant propositions, inductive families, and an proof-irrelevant identity type with a strong transport rule as seen in Section 8.1;

- The *metatheory*: this is our ambient language in which we state and prove our results about the target theory and the IIT encodings it supports. It is a rich, extensional language corresponding to the usual ground-level mathematics.

Concretely, we will take it to be some version of extensional type theory with quotient inductive-inductive types (QIITs).

We will perform all our constructions in the metatheory, and define the target theory as a first class entity internal to the metalanguage. We will achieve so by defining a notion of model of the target theory as a metatheoretic structure (Section 9.3.2): our results can therefore be equivalently understood as a construction on models of the target theory.

This separation in two levels not only allows us to be precise about where the encoding takes place, as opposed to the reasoning *about* the encoding, but also allows the ambient metalanguage to be as expressive and extensional as we need, without fear of this extensionality *leaking* into the target theory and destroying its intensional nature, thus jeopardizing the entire premise of this whole endevour.

The next two sections will give a detailed description of the metatheory and target theory, with their respective notational conventions.

9.3.1 Metatheory

While the goal of our reduction is to show how to encode infinitary IITs in a restricted intensional type theory, the metalanguage in which we reason about these encodings need not be restricted at all. We will thus rely on a rich metatheory, concretely taken to be a version of extensional type theory equipped with an hierarchy of universes and QIITs, in addition with the usual tools of MLTT. An example of such a theory is Setoid Type Theory [ABKT19] extended with QIITs [KX21], which is as expressive as ETT by Hofmann's conservativity result [Hof96]. Another example is given by the theory arising as the internal language of presheaf categories (see Section 2.2.2).

We use pseudo-Agda syntax for metatheoretic constructions, with all the notational conventions listed in Remark 3.2.1. In addition, for dependent sum types we sometimes use the infix notation $(x : A) \times (B x)$ instead of $\Sigma\ (x : A)(B x)$ when it benefits formatting. We write $\mathbb{1}$ and $\star$, respectively, for the metatheoretic unit type and its constructor. We have η-equality for unit types, as well as dependent pairs and functions. We also have sum types $A + B$ with constructors left and right. We sometimes refer to h-propositions, i.e. types $A : \mathbf{Type}$ for which $\forall\, x\, y \,.\, x = y$ holds, as *weak propositions*.

9.3.2 Target theory

Since we want to express our encoding method as a metatheorem *about* the target theory, it follows that we need a way to define and manipulate the target theory as an internal, first class entity in the metalanguage. Rather than fixing the target theory directly as a syntactic object, we will instead define a notion of model of the target theory, and state our theorems for an arbitrary such model.

We specify the models of the target theory using a form of higher-order abstract syntax, relying on the meta-theoretic function space to express binding (Section 2.2.3.) This can be justified by imagining to work in the internal language of the category $\hat{\mathcal{C}}$ of presheaves over some arbitrary fixed category $\mathcal{C}$. $\hat{\mathcal{C}}$ is a model of type theory (Section 2.2.2), and in particular it enjoys the same properties of the ambient metatheory (where **Type** lives). We abuse notation, and treat the internal language of $\hat{\mathcal{C}}$ as being itself our metatheory. In particular, we shall from now on consider the notation regarding the metatheory to be describing constructions in the language of $\hat{\mathcal{C}}$.

We can now define what constitutes a model of the target theory in a completely context-free way (as per Section 2.2.3): an (external) interpretation of the (internal) signature outlined below. We begin with sorts Type of (large) types, and universes of small types and propositions:

$$\begin{aligned}
&\mathsf{Type} : \mathbf{Type}\\
&\mathsf{Term} : \mathsf{Type} \to \mathbf{Type}\\
&\mathcal{U}, \mathcal{P} : \mathsf{Type}\\
&\mathsf{El}_{\mathcal{U}} : \mathcal{U} \to \mathsf{Type}\\
&\mathsf{El}_{\mathcal{P}} : \mathcal{P} \to \mathsf{Type}
\end{aligned}$$

We abbreviate $\mathsf{SType} :\equiv \mathsf{Term}\ \mathcal{U}$, $\mathsf{SProp} :\equiv \mathsf{Term}\ \mathcal{P}$, $\mathsf{STerm}\ A :\equiv \mathsf{Term}\ (\mathsf{El}_{\mathcal{U}}\ A)$, $\mathsf{PTerm}\ P :\equiv \mathsf{Term}\ (\mathsf{El}_{\mathcal{P}}\ P)$. We will often just write Term instead of $\mathsf{STerm}, \mathsf{PTerm}$ when there is no ambiguity, as well as just El instead of $\mathsf{El}_{\mathcal{U}}$ and $\mathsf{El}_{\mathcal{P}}$. Moreover, we will write $\ulcorner A \urcorner$ as compact notation for $\mathsf{Term}\, A$.

We allow to lift propositions into small types, with functions going back and forth:

$$\begin{aligned}
&\mathsf{Lift} : \mathsf{SProp} \to \mathsf{SType}\\
&\mathsf{lift} : \{A : \mathsf{SProp}\} \to \mathsf{Term}\, A \to \mathsf{Term}\, (\mathsf{Lift}\, A)\\
&\mathsf{unlift} : \{A : \mathsf{SProp}\} \to \mathsf{Term}\, (\mathsf{Lift}\, A) \to \mathsf{Term}\, A\\
&\mathsf{lift}\ (\mathsf{unlift}\ x) = x
\end{aligned}$$

We require dependent function types between large types, as well as function and pair types between small types:

$$\mathbf{\Pi}_T : (A : \mathsf{Type}) \to (\mathsf{Term}\ A \to \mathsf{Type}) \to \mathsf{Type}$$

$$\begin{aligned}
&\mathbf{\Pi}_{u,u'} : (A : \mathsf{Term}\ u) \to (\mathsf{Term}\ (\mathsf{El}\ u) \to \mathsf{Term}\ u') \to \mathsf{Term}\ u' \\
&\mathbf{\Sigma}_{u,u'} : (A : \mathsf{Term}\ u) \to (\mathsf{Term}\ (\mathsf{El}\ u) \to \mathsf{Term}\ u') \to \mathsf{Term}\ u
\end{aligned}$$

where $u, u' \in \{\mathcal{U}, \mathcal{P}\}$. We will write $\mathbf{\Pi}$ for both $\mathbf{\Pi}_T$ and $\mathbf{\Pi}_{u,u'}$, and $\mathbf{\Sigma}$ for $\mathbf{\Sigma}_{u,u'}$, whenever the subscripts can be inferred from context. We will frequently write $\mathbf{\Pi}\,(x : A)\,(B\,x)$ for $\mathbf{\Pi}\,A\,B$, and similarly $\mathbf{\Sigma}\,(x : A)\,(B\,x)$ for $\mathbf{\Sigma}\,A\,B$.

We have the following constructors for dependent functions and pairs:

$$\begin{aligned}
&\boldsymbol{\lambda} : ((a : \mathsf{Term}\ A) \to \mathsf{Term}\ (B\ a)) \to \mathsf{Term}\ (\mathbf{\Pi}\ A\ B) \\
&\mathsf{app} : \mathsf{Term}\ (\mathbf{\Pi}\ A\ B) \to (a : \mathsf{Term}\ A) \to \mathsf{Term}\ (B\ a) \\
&\langle \_, \_ \rangle : (a : \mathsf{Term}\ A)(b : \mathsf{Term}\ (B\ a)) \to \mathsf{Term}\ (\mathbf{\Sigma}\ A\ B) \\
&\mathsf{fst} : \mathsf{Term}\ (\mathbf{\Sigma}\ A\ B) \to \mathsf{Term}\ A \\
&\mathsf{snd} : (p : \mathsf{Term}\ (\mathbf{\Sigma}\ A\ B)) \to \mathsf{Term}\ (B\ (\mathsf{fst}\ p))
\end{aligned}$$

with β/η equations

$$\begin{aligned}
&\mathsf{app}\ (\boldsymbol{\lambda}\ f) = f \\
&\boldsymbol{\lambda}\ (\mathsf{app}\ t) = t \\
&\mathsf{fst}\ \langle a, b \rangle = a \\
&\mathsf{snd}\ \langle a, b \rangle = b \\
&\langle \mathsf{fst}\ p, \mathsf{snd}\ p \rangle = p
\end{aligned}$$

We write $\boldsymbol{\lambda}\,x\,.\,t$ for $\boldsymbol{\lambda}\,(\lambda\,x\,.\,t)$, and $f \cdot x$ for $\mathsf{app}\ f\ x$. We use a bold cross symbol $A \times B$ for the target-level non-dependent product.

We also include unit types $\mathbf{1} : \mathsf{SType}$ and $\top : \mathsf{SProp}$, with constructors $* : \mathsf{Term}\ \mathbf{1}$, $\mathsf{truth} : \mathsf{Term}\ \top$, and η-equation stating $t = *$ for all $t : \mathsf{Term}\ \mathbf{1}$.

We assume the existence of arbitrary indexed inductive definitions in both $\mathcal{U}$ and Type. Rather than equipping the model with W-types or a definition schema right now, once and for all, we will instead extend it with concrete inductive definitions as we go. We do however assume right away to have natural numbers in $\mathcal{U}$, which we write as Nat.

Finally, we require a propositional identity type with a strong transport rule

$$\begin{aligned}
&\mathsf{Id} : (A : \mathsf{SType}) \to \mathsf{Term}\ A \to \mathsf{Term}\ A \to \mathsf{SProp} \\
&\mathsf{Refl} : \{A : \mathsf{SType}\}(a : \mathsf{Term}\ A) \to \mathsf{Id}\ A\ a\ a \\
&\mathsf{Transp} : \{A : \mathsf{SType}\}(B : \mathsf{Term}\ A \to \mathsf{Type})
\end{aligned}$$

$$\{x\ y : \mathsf{Term}\ A\} \to \mathsf{Term}\ (\mathsf{Id}\ A\ x\ y) \to \mathsf{Term}\ (B\ x) \to \mathsf{Term}\ (B\ y)$$

that computes on any reflexive equation:

$$\mathsf{Transp}\ B\ (\mathsf{Refl}\ x)\ u = u$$

Remark 9.3.1. Note that the target theory (or rather, its notion of model) is given as an internal construction in the metatheory, and hence it is clearly separate from it. As mentioned in Section 9.3, this allows us to make sure that the extensionality of the metalanguage does not impact the intensional nature of the target language. In particular, we have no way to construct a proof term of *funext* for the target identity type Id, as Id is essentially akin to an abstract data type which inhabitants can only be constructed from the tools we have carefully selected. On the other hand target-level definitional equality is, as expected, fully extensional, since it is expressed in terms of the meta-level identity type. ■

We will often write Id omitting the type and using infix notation, as $x \approx y$.

From transport we derive proof terms for symmetry Sym and transitivity Trans in the obvious way. With the lifting operator Lift we can extend Transp to a term $\mathsf{TranspP}$ acting on families of propositions.

Thanks to η-equality in the target theory, we can arbitrarily replace variables of product types with explicit tuples. For example, we can and will write tuple patterns in target-level function definitions like $\boldsymbol{\lambda}\,\langle x, y\rangle\,.\,t$, and bind patterns to variables à la Agda like $\boldsymbol{\lambda}\,p@\langle x, y\rangle\,.\,t$.

The general reduction we seek to prove is a metatheorem about models of the target theory. We thus now assume we are given one such model; that is, we assume we have terms $\mathsf{Type}, \mathsf{Term}, \boldsymbol{\Pi}$, etc., satisfying the necessary equations, and state our metatheorems over that model. Because the model we fixed is completely arbitrary, it follows that the metatheorems hold for all models.

9.4 Specifying Inductive-Inductive Types

In this section we discuss formal ways to specify IITs, including the formal specification of linear IITs that we will be using for our developments. The idea is to define a datatype of specifications, with its elements giving a syntactic description of the types and constructors of the specified IIT.

9.4.1 Type Theory as a datatype of specifications

Inductive-inductive types and their constructors exhibit a complex dependency structure that closely resembles dependent types and type-theoretic contexts. It is not surprising, then, that type theory itself would serve as a good tool to specify IITs.

The idea to use small-scale type theories as domain specific languages to specify IITs has been proposed and refined in several works that take a syntactic approach to IIT specifications (see Section 13.4 for a literature review.) The advantage of this approach is in its expressive power, as it can be used to specify many forms of induction-induction, from regular IITs [AKKvR18, AKKvR19, vR19] to QIITs [KKA19] and higher IITs [KK18].

We now take a brief look at how these specification type theories look like, using [vR19] as the reference of choice. The datatype of specifications is presented as a small type-theoretic syntax, with contexts, types, terms, and substitutions, including the following constructors:

$$
\begin{aligned}
&\mathsf{Con} : \mathbf{Type}\\
&\mathsf{Ty} : \mathsf{Con} \to \mathbf{Type}\\
&\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathbf{Type}\\
&\ldots\\
&\bullet : \mathsf{Con}\\
&\_ \rhd \_ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Con}\\
&\mathsf{U} : \mathsf{Ty}\ \Gamma\\
&\mathsf{El} : \mathsf{Tm}\ \Gamma\ \mathsf{U} \to \mathsf{Ty}\ \Gamma\\
&\pi : (A : \mathsf{Tm}\ \Gamma\ \mathsf{U}) \to \mathsf{Ty}\ (\Gamma, \mathsf{El}\ A) \to \mathsf{Ty}\ \Gamma\\
&\mathsf{ap} : \mathsf{Tm}\ \Gamma\ (\pi\ A\ B) \to \mathsf{Tm}\ (\Gamma, A)\ B\\
&\ldots
\end{aligned}
$$

A specification then corresponds to a well-formed context in such a syntax. For example, we can specify natural numbers as the following context: [1]

$$\bullet, N : \mathsf{U}, z : \mathsf{El}\ N, s : \pi\ N\ (\mathsf{El}\ N)$$

The universe U of *small types* is used to list the *sorts* of the specified IIT. It also serves to enforce strict positivity, as the type of π prevents us from declaring constructor parameters ranging over the IIT being defined.

This presentation of specifications as type-theoretic contexts plays really well with inductive-inductive types, because it allows to represent the full spectrum of dependency between sorts and constructors that characterizes induction-induction. For example, we can specify the Con/Ty IIT from Section 3.2 as follows (we write $\odot$ for the usual binary application operator, which can easily be defined from ap):

[1]Here we make use of named variables, and ignore explicit weakenings for the sake of legibility.

$$\begin{aligned}&\bullet, \mathit{Con} : \mathsf{U}, \mathit{Ty} : \pi\ \mathit{Con}\ \mathsf{U}, \mathit{nil} : \mathsf{El}\ \mathit{Con}, \mathit{ext} : \pi\ \mathit{Con}\ (\pi\ \mathit{Ty}\ (\mathsf{El}\ \mathit{Con})),\\ &\quad \mathit{iota} : \pi\ \mathit{Con}\ (\mathsf{El}\ (\mathsf{ap}\ \mathit{Ty})),\\ &\quad \mathit{pi} : \pi(\Gamma : \mathit{Con})(\pi(A : \mathit{Ty} \odot \Gamma)(\pi\ (\mathit{Ty} \odot (\mathit{ext} \odot \Gamma \odot A))(\mathit{Ty} \odot \Gamma)))\end{aligned}$$

Figure 9.1

This kind of type-theoretic specification syntax comes in different incarnations (see Section 13.4), and although each of them features slight differences depending on the class of IITs that is being targeted, they all share the same core CwF structure. We will refer to this style of IIT specification as "*Kaposi-Kovács-style syntax*".

9.4.2 Linear infinitary IITs

We now define a QIIT that serves as the datatype of specifications for the IITs targeted by our general reduction method. Figure 9.2 and Figure 9.3 give the complete list of all the constructors for this QIIT, including type constructors, term constructors, and equality constructors. A few symbols in these definitions are overloaded, and rely on metavariable names to disambiguate. Name conventions for metavariables are illustrated later on.

Our specifications datatype is inspired by the Kaposi-Kovács-style syntax, but differs from it in a few crucial ways. The first difference is the absence of a universe U in our specifications. In the Kaposi-Kovács syntax the universe is used to define the sorts of the specified IIT, as well as to subsequently refer to them in the constructors. Since we only consider two-sorted IITs, there is no use for this universe in our specifications. The two sorts are "hard coded" in our specification datatype as constructors of the type Ty of "inner" types, which specifies all the types that can appear within IIT constructors. We have two constructors $\mathsf{T}^1, \mathsf{T}^2$, corresponding, respectively, to the "non-indexed" and the "indexed" type of the IIT being specified. We refer to these two types as the *base types*.

The second notable aspect of our QIIT concerns the implementation of *contexts*. In our specifications, we distinguish between "outer" (or "global") contexts that accumulate constructors of the IIT being specified, and "inner" (or "local") contexts that accumulate parameters local to a single constructor. To see what we mean, consider the *pi* constructor from in the example specification from Figure 9.1: here, under our terminology, the *constructor terms nil*, *ext*, *iota* would be part of the "outer" context, whereas parameters like Γ and A, which are only in scope within *pi*, would be part of the "inner" context of *pi*.

$\mathsf{Spec} : \mathbf{Type}$
$\mathsf{Wk} : \mathsf{Spec} \to \mathsf{Spec} \to \mathbf{Type}$
$\mathsf{Ctor} : \mathsf{Spec} \to \mathbf{Type}$
$\mathsf{Params} : \mathsf{Spec} \to \mathbf{Type}$
$\mathsf{Base} : (\Gamma : \mathsf{Spec}) \to \mathsf{Params}\ \Gamma \to \mathbf{Type}$

$\mathsf{Ty} : (\Gamma : \mathsf{Spec}) \to \mathsf{Params}\ \Gamma \to \mathbf{Type}$
$\mathsf{Tm} : \forall \Gamma\ \Delta \to \mathsf{Ty}\ \Gamma\ \Delta \to \mathbf{Type}$
$\mathsf{CTm} : \forall \Gamma \to \mathsf{Ctor}\ \Gamma \to \mathbf{Type}$
$\mathsf{Sub} : \forall \{\Gamma\} \to \mathsf{Params}\ \Gamma \to \mathsf{Params}\ \Gamma \to \mathbf{Type}$
$\mathsf{isBase} : \mathsf{Ty}\ \Gamma\ \Delta \to \mathbf{Type}$

$\mathsf{data\ Spec\ where}$
  $\diamond : \mathsf{Spec}$
  $\_ \rhd \_ : \forall \Gamma \to \mathsf{Ctor}\ \Gamma \to \mathsf{Spec}$
$\mathsf{data\ Params\ where}$
  $\bullet : \mathsf{Params}\ \Gamma$
  $\_ \rhd\!\rhd \_ : \forall \Delta \to \mathsf{Ty}\ \Gamma\ \Delta \to \mathsf{Params}\ \Gamma$
  $\_[\_] : \mathsf{Params}\ \Omega \to \mathsf{Wk}\ \Gamma\ \Omega \to \mathsf{Params}\ \Gamma$
$\mathsf{data\ Ty\ where}$
  $\mathsf{ext} : \mathsf{SType} \to \mathsf{Ty}\ \Gamma\ \Delta$
  $\pi : \forall A \to \mathsf{Base}\ \Gamma\ (\Delta \rhd\!\rhd \mathsf{ext}\ A) \to \mathsf{Ty}\ \Gamma\ \Delta$
  $\mathsf{T}^1 : \mathsf{Ty}\ \Gamma\ \Delta$
  $\mathsf{T}^2 : \mathsf{Tm}\ \Gamma\ \Delta\ \mathsf{T}^1 \to \mathsf{Ty}\ \Gamma\ \Delta$
  $\_[\_] : \mathsf{Ty}\ \Gamma \Delta \to \mathsf{Sub}\ \nabla\ \Delta \to \mathsf{Ty}\ \Gamma\ \Delta$
  $\_[\_] : \mathsf{Ty}\ \Gamma \Delta \to \mathsf{Wk}\ \Omega\ \Gamma \to \mathsf{Ty}\ \Omega\ (\Delta[w])$
$\mathsf{data\ Wk\ where}$
  $\mathsf{id} : \mathsf{Wk}\ \Gamma\ \Gamma$
  $\mathsf{drop} : \mathsf{Wk}\ (\Gamma \rhd C)\ \Gamma$
  $\_ \circ \_ : \mathsf{Wk}\ \Gamma\ \Omega \to \mathsf{Wk}\ \Omega\ \Theta \to \mathsf{Wk}\ \Gamma\ \Theta$
$\mathsf{data\ Sub\ where}$
  $\mathsf{id} : \mathsf{Sub}\ \Delta\ \Delta$
  $\mathsf{ext} : (\sigma : \mathsf{Sub}\ \Delta\ \nabla) \to \mathsf{Tm}\ \Gamma\ \Delta\ (A[\sigma])$
    $\to \mathsf{Sub}\ \Delta\ (\nabla \rhd\!\rhd A)$
  $\mathsf{drop} : \mathsf{Sub}\ (\Delta \rhd\!\rhd A)\ \Delta$
  $\_ \circ \_ : \mathsf{Sub}\ \Gamma\ \Delta \to \mathsf{Sub}\ \Delta\ \nabla \to \mathsf{Sub}\ \Gamma\ \nabla$
  $\_[\_] : \mathsf{Sub}\ \nabla\ \Delta \to (w : \mathsf{Wk}\ \Gamma\ \Omega)$
    $\to \mathsf{Sub}\ (\nabla[w])\ (\Delta[w])$

$\mathsf{isBase}\, \{\Gamma\, \Delta\}\, A :\equiv (A = \mathsf{T}^1)\ +\ \Sigma\, (t : \mathsf{Tm}\, \Gamma\, \Delta\, \mathsf{T}^1)\, (A = \mathsf{T}^2\, t)$
$\mathsf{Base}\ \Gamma\ \Delta :\equiv \Sigma(\mathsf{Ty}\ \Gamma\ \Delta)\ \mathsf{isBase}$
$\mathsf{Ctor}\ \Gamma :\equiv (\Delta : \mathsf{Params}\ \Gamma) \times (\mathsf{Base}\ \Gamma\ \Delta)$
$\mathsf{ctor} : (\Delta : \mathsf{Params}\ \Gamma) \to \mathsf{Base}\ \Gamma\ \Delta \to \mathsf{Ctor}\ \Gamma$
$\mathsf{ctor}\ \Delta\ X :\equiv (\Delta, X)$

$\mathsf{data\ Tm\ where}$
  $\mathsf{vz} : \mathsf{Tm}\ \Gamma\ (\Delta \rhd\!\rhd A)\ (A[\mathsf{drop}])$
  $\mathsf{vs} : \mathsf{Tm}\ \Gamma\ \Delta\ A$
    $\to \mathsf{Tm}\ \Gamma\ (\Delta \rhd\!\rhd B)\ (A[\mathsf{drop}])$
  $\mathsf{ext} : \ulcorner X \urcorner \to \mathsf{Tm}\ \Gamma\ \Delta\ (\mathsf{ext}\ X)$
  $\mathsf{capp} : \mathsf{CTm}\ \Gamma\ (\mathsf{ctor}\ \Delta\ A) \to (\sigma : \mathsf{Sub}\ \nabla\ \Delta)$
    $\to \mathsf{Tm}\ \Gamma\ \nabla\ (A[\sigma])$
  $\mathsf{ap} : \mathsf{Tm}\ \Gamma\ \Delta\ (\pi\ A\ B) \to \mathsf{Tm}\ (\Delta \rhd\!\rhd \mathsf{ext}\ A)\ B$
  $\mathsf{lm} : \mathsf{Tm}\ (\Delta \rhd\!\rhd \mathsf{ext}\ A)\ B \to \mathsf{Tm}\ \Gamma\ \Delta\ (\pi\ A\ B)$
  $\_[\_] : \mathsf{Tm}\ \Gamma\ \Delta\ A \to (\sigma : \mathsf{Sub}\ \nabla\ \Delta)$
    $\to \mathsf{Tm}\ \Gamma\ \nabla\ (A[\sigma])$
  $\_[\_] : \mathsf{Tm}\ \Gamma\ \Delta\ A \to (w : \mathsf{Wk}\ \Omega\ \Gamma)$
    $\to \mathsf{Tm}\ \Omega\ (\Delta[w])\ (A[w])$
$\mathsf{data\ CTm\ where}$
  $\mathsf{cvz} : \mathsf{CTm}\ (\Gamma \rhd C)(C[\mathsf{drop}])$
  $\mathsf{cvs} : \forall \{C'\}\,.\, \mathsf{CTm}\ \Gamma\ C \to \mathsf{CTm}\ (\Gamma \rhd C')(C[\mathsf{drop}])$
  $\_[\_] : \mathsf{CTm}\ \Gamma\ C \to (w : \mathsf{Wk}\ \Omega\ \Gamma) \to \mathsf{CTm}\ \Omega\ (C[w])$

Figure 9.2: Type and term constructors of the QIIT specification type

In contrast, global and local contexts are unified in the Kaposi-Kovács-style syntax: for example, the codomain of *iota* is "typechecked" in context ($nil \rhd ext \rhd \Gamma$), which includes both "global" constructors and "local" parameters. Our decision to implement two notions of contexts is driven by convenience, as more fine-grained control over contexts makes it significantly easier to formalize the notion of *linear* constructor specifications (see Section 9.4.3.)

We thus define the type Spec of "outer" contexts (which correspond to specification contexts Con in Kaposi-Kovács syntax), and the type Params : Spec → **Type**

$\bullet\,[w] = \bullet$

$(\Delta \rhd\!\rhd T)[w] = \Delta[w] \rhd\!\rhd T[w]$

$\Delta[w_1][w_2] = \Delta[w_2 \circ w_1]$

$\Delta[\mathsf{id}] = \Delta$

$(\mathsf{ext}\ X)[\sigma] = \mathsf{ext}\ X$

$(\mathsf{ext}\ X)[w] = \mathsf{ext}\ X$

$(\pi\, A\ B)[\sigma] = \pi\, A\, (B[\mathsf{ext}\, (\mathsf{drop} \circ \sigma)\, \mathsf{vz}])$

$(\pi\, A\ B)[w] = \pi\, A\ (B[w])$

$T[\sigma][\tau] = T[\tau \circ \sigma]$

$T[w_1][w_2] = T[w_2 \circ w_1]$

$T[\sigma][w] = T[w][\sigma[w]]$

$T[\mathsf{id}] = T$

$\mathsf{T}^1[\sigma] :\equiv \mathsf{T}^1$

$\mathsf{T}^1[w] :\equiv \mathsf{T}^1$

$(\mathsf{T}^2\ t)[\sigma] :\equiv \mathsf{T}^2\ (t[\sigma])$

$(\mathsf{T}^2\ t)[w] :\equiv \mathsf{T}^2\ (t[w])$

$\mathsf{id} \circ w = w$

$w \circ \mathsf{id} = w$

$(w_1 \circ w_2) \circ w_3 = w_1 \circ (w_2 \circ w_3)$

$\tau \circ \mathsf{ext}\ \sigma\ t = \mathsf{ext}\ (\tau \circ \sigma)\ (t[\tau])$

$\mathsf{ext}\ \sigma\ t \circ \mathsf{drop} = \sigma$

$\mathsf{id}[w] = \mathsf{id}$

$\mathsf{drop}[w] = \mathsf{drop}$

$(\mathsf{ext}\ \sigma\ t)[w] = \mathsf{ext}\ (\sigma[w])\ (t[w])$

$\sigma[w_1][w_2] = \sigma[w_2 \circ w_1]$

$\sigma[\mathsf{id}] = \sigma$

$(\mathsf{ctor}\ \Delta\ B)[w] :\equiv \mathsf{ctor}(\Delta[w])(B[w])$

$t[\mathsf{id}] = t$

$(\mathsf{ext}\ X\ x)[\sigma] = \mathsf{ext}\ X\ x$

$(\mathsf{ext}\ X\ x)[w] = \mathsf{ext}\ X\ x$

$(\mathsf{capp}\ c\ \sigma)[\tau] = \mathsf{capp}\ c\ (\tau \circ \sigma)$

$(\mathsf{capp}\ c\ \sigma)[w] = \mathsf{capp}\ (c[w])(\sigma[w])$

$(\mathsf{ap}\ t)[\mathsf{ext}\ (\mathsf{drop} \circ \sigma)\ \mathsf{vz}] = \mathsf{ap}(t[\sigma])$

$(\mathsf{ap}\ t)\ [w] = \mathsf{ap}(t[w])$

$(\mathsf{lm}\ t)[\sigma] = \mathsf{lm}\ (t[\mathsf{ext}\ (\mathsf{drop} \circ \sigma)\ \mathsf{vz}])$

$(\mathsf{lm}\ t)[w] = \mathsf{lm}\ (t[w])$

$\mathsf{ap}\ (\mathsf{lm}\ t) = t$

$\mathsf{lm}\ (\mathsf{ap}\ t) = t$

$t[\sigma][\tau] = t[\tau \circ \sigma]$

$t[w_1][w_2] = t[w_2 \circ w_1]$

$t[\sigma][w] = t[w][\sigma[w]]$

$\mathsf{vz}[\mathsf{ext}\ \sigma\ t] = t$

$(\mathsf{vs}\ t)[\mathsf{ext}\ \sigma\ s] = t[\sigma]$

$c[\mathsf{id}] = c$

$c[\mathsf{drop} \circ w] = \mathsf{vs}\ (c[w])$

$c[w_1][w_2] = c[w_2 \circ w_1]$

Figure 9.3: Equality constructors of the IIT of specifications

of inner contexts (or, *parameters*). Note that parameters are indexed by an outer context, which reflects the fact that each constructor (and therefore its list of parameters) is allowed to depend on previously specified constructors within the same IIT specification.

Two kinds of contexts naturally call for two kinds of "types" that make up these contexts: we extend outer contexts with constructors of type $\mathsf{Ctor} : \mathsf{Spec} \to \mathbf{Type}$, and extend parameters with "local types" of type $\mathsf{Ty} : (\Gamma : \mathsf{Spec}) \to \mathsf{Params}\,\Gamma \to \mathbf{Type}$. We render these two notions of context extension as $\triangleright$ and $\triangleright\!\triangleright$, respectively. The *locality* of $\triangleright\!\triangleright$ is confirmed by its type, which does not alter the global context. Note that types of parameters local to a specific constructor depend on all constructors specified up to that point, in addition to all the parameters specified up to that point *locally to that constructor*; hence the double index on Ty. This structure of parameters and local types resembles type-theoretic telescopes.

A constructor in outer context Γ is specified by giving a list of parameters, i.e. a telescope of inner types Δ, and a base type depending on these parameters, in addition to all previously defined constructors: $\mathsf{ctor} : (\Delta : \mathsf{Params}\ \Gamma) \to \mathsf{Base}\ \Gamma\ \Delta \to \mathsf{Ctor}\ \Gamma$. Here we define Base as a subtype of Ty, via the predicate isBase, and Ctor as a simple dependent pair:

$$\mathsf{Base}\,\Gamma\ \Delta :\equiv \Sigma(\mathsf{Ty}\ \Gamma\ \Delta)\ \mathsf{isBase}$$
$$\mathsf{Ctor}\ \Gamma :\equiv \Sigma(\Delta : \mathsf{Params}\ \Gamma)(\mathsf{Base}\ \Gamma\ \Delta)$$

We then define ctor to be just pairing.

This use of Base in the specification of constructors reflects the idea that the codomain of any constructor of the IIT being specified is necessarily one of the two types that constitute it.

Remark 9.4.1. We can prove that isBase is a proposition, by induction on Ty and injectivity of its constructors. We will thus treat Base as a subtype of Ty and implicitly coerce between the two whenever the operation can be justified, without explicitly writing down proofs of isBase. For example, we will write T^1 and T^2 as if they were Base constructors. Moreover, we can justify weakening and substitution operations on base types, as there are proofs $\mathsf{isBase}\,T \to \mathsf{isBase}\,(T[w])$ and $\mathsf{isBase}\,T \to \mathsf{isBase}\,(T[\sigma])$ for any T, w, σ, both constructed by case analysis on the disjunctive proof, and application of the equality constructors of the QIIT. ■

The *iota* constructor from Figure 9.1 would be specified, in our own syntax, as follows:

$$\mathit{iota} : \mathsf{Ctor}\ (\diamond \triangleright \mathit{nil} \triangleright \mathit{ext})$$
$$\mathit{iota} :\equiv \mathsf{ctor}\ (\bullet \triangleright\!\triangleright \mathsf{T}^1)\ (\mathsf{T}^2\ \mathsf{vz})$$

Note that the codomain $\mathsf{T}^2\ \mathsf{vz}$ of *iota* is typed as

$$\mathsf{Base}\ (\diamond \triangleright \mathit{nil} \triangleright \mathit{ext})\ (\bullet \triangleright\!\triangleright \mathsf{T}^1)$$

where we have a clear distinction between the "outer" context of constructors, and the "inner" context of parameters local to the constructor itself.

Inner types Ty specify what types can be assigned to individual parameters of a IIT constructor. These can be an external type in SType, a function space out of an external type (to account for infinitary IITs), or a *base type.*

Functional inner types are formed with the π constructor. The domain of such functions is always an external type, ensuring strict positivity. We can form terms of an external type by indexing into an existing parameter (using vz and vs), or by giving it as a literal constant (using ext).

Remark 9.4.2. The scope of expressions of external types that we support is quite limited. For example, we can specify a constructor like $c : (f : \mathbb{N} \to A)(n : \mathbb{N}) \to f\,n \to A$, but not $c : (f : \mathbb{N} \to A)(n : \mathbb{N}) \to f\,(n+5) \to A$. We could have fixed this by adding more constructors for external term formation, however we decided to opt for simplicity as we believe the gained expressiveness would have no impact on whether the specified IIT is reducible to inductive families. In other words, allowing more expressive external terms wouldn't make the problem more "interesting". ■

Our specification syntax uses explicit substitutions, and we distinguish two kinds of substitutions that apply to the two kinds of contexts. As it turns out, we only need full-fledged substitutions for parameters contexts, whereas weakenings are sufficient for outer contexts. We have constructors $\_[\_]$ for applying specification terms to substitutions and weakenings, plus equations governing how these terms compute and propagate on canonical forms. We overload most of the symbols involved, and use $\_[\_]$ as well as $\mathsf{id}, \mathsf{drop}$ for both weakenings and substitutions, relying on metavariable names to disambiguate. Some metavariable names we use consistently are w for weakenings, σ, τ for substitutions, Δ, ∇ for parameter telescopes, Γ, Ω for specification contexts, T and B for inner types and base types, and t and c for terms and constructor terms.

We extend weakenings to ctor is the obvious way: $(\mathsf{ctor}\ \Delta\ T)[w] :\equiv \mathsf{ctor}\ \Delta[w]\ T[w]$. All the expected laws, like $C[w_1][w_2] = C[w_2 \circ w_1]$ for any $C : \mathsf{Ctor}\ \Gamma$, easily follow by congruence.

We define two kinds of terms indexing into the two kinds of contexts: we have *constructor terms* $\mathsf{CTm} : (\Gamma : \mathsf{Spec}) \to \mathsf{Ctor}\ \Gamma \to \mathbf{Type}$, that index constructors out of a given outer context, and *local terms* (or just *terms*) $\mathsf{Tm} : (\Gamma : \mathsf{Spec})(\Delta : \mathsf{Params}\ \Gamma) \to \mathsf{Ty}\ \Gamma\ \Delta \to \mathbf{Type}$, that instead index arguments within a given local context. In addition to indexing (vz, vs) and external terms (ext), we can form local terms by abstraction/application (ap, lm), substitution/weakening, or by applying a constructor term c to a list of arguments σ (capp).

Remark 9.4.3. The datatype of specifications Spec is a QIIT, hence its elimination principle would require us to check that every function defined by induction on it maps equal inputs to equal outputs. We will omit these equality checks in the text, and only write down the data components of the induction. Although we expect most, if not all, of these checks to be easily verifiable, we haven't actually verified all of them. See Section 13.3 for further discussion on this. ∎

9.4.3 Specifying linearity

We call a term Γ of type Spec a *specification*. As an example, we can now give the specification of our toy infinitary Con/Ty IIT from Section 8.1 as a term $\Theta : \mathsf{Spec}$ defined as follows[2]:

$$\begin{aligned}\Theta :\equiv \diamond &\rhd \mathsf{ctor}\ \bullet\ \mathsf{T}^1\\ &\rhd \mathsf{ctor}\ (\bullet \rhd\!\rhd \mathsf{T}^1 \rhd\!\rhd \mathsf{T}^2\ \mathsf{vz})\ \mathsf{T}^1\\ &\rhd \mathsf{ctor}\ (\bullet \rhd\!\rhd \mathsf{T}^1)\ (\mathsf{T}^2\ \mathsf{vz})\\ &\rhd \mathsf{ctor}\ (\bullet \rhd\!\rhd \mathsf{T}^1 \rhd\!\rhd (\pi\ \mathsf{Nat}\ (\mathsf{T}^2\ \mathsf{vz})))\ (\mathsf{T}^2\ (\mathsf{vs}\ \mathsf{vz}))\\ &\rhd \mathsf{ctor}\ (\bullet \rhd\!\rhd \mathsf{T}^1 \rhd\!\rhd \mathsf{T}^2\ \mathsf{vz} \rhd\!\rhd \mathsf{T}^2\ (\mathsf{capp}\ (\mathsf{cvs}\ \mathsf{cvs}\ \mathsf{cvz})\ \mathsf{id}))\\ &\qquad\quad (\mathsf{T}^2\ (\mathsf{vs}\ \mathsf{vz}))\end{aligned}$$

As discussed in Section 9.2, our work focuses on linear IITs as the target for our reduction. However, nothing in our datatype Spec of specifications mentions any linearity conditions, and in particular it allows to specify non-linear IITs.

We now formalize linear specifications by defining a suitable predicate on Spec that implements Definition 9.2.1. We rely on the notion of *telescope-prefix*, which is a relation between parameter telescopes where one can be seen as the prefix on the other. We first define a predicate of *weakening* substitutions:

$$\frac{\Gamma : \mathsf{Spec} \qquad \Delta\ \nabla : \mathsf{Params}\ \Gamma \qquad \sigma : \mathsf{Sub}\ \Delta\ \nabla}{\mathsf{PWk}\ \sigma : \mathbf{Type}} \qquad \frac{}{\mathsf{PWk}\ \mathsf{id}} \qquad \frac{\mathsf{PWk}\ \sigma}{\mathsf{PWk}\ (\mathsf{drop} \circ \sigma)}$$

A proof of $\mathsf{PWk}\ \sigma$ for some $\sigma : \mathsf{Sub}\ \Delta\ \nabla$ states that Δ is obtained from ∇ by repeated extension. Thus, ∇ can be viewed as a sub-telescope of Δ and also its prefix.

We define *weakening substitutions* as the subtype $\mathsf{WkSub}\ \Delta\ \nabla :\equiv \Sigma(\mathsf{Sub}\ \Delta\ \nabla)\ \mathsf{PWk}$. We lift the constructors $\mathsf{id}, \mathsf{drop}$ to WkSub in the obvious way, and implicitly coerce between WkSub and Sub when convenient.

[2]As usual, we rely on equality reflection in the metatheory and omit writing most of the transports that would otherwise be necessary.

Weakening substitutions provide an easy way to state linearity: as an example, let $A : \mathbf{Type}, B : A \to \mathbf{Type}$ be an IIT with a A-constructor c_A and a B-constructor $c_B : (\overrightarrow{x} : \Delta) \to B\ (c_A\ \overrightarrow{y})$, where $\overrightarrow{y}$ is a sequence of variables, and $(\overrightarrow{x} : \Delta)$ is short for a telescope of assumptions $(x_1 : X_1) \ldots (x_n : X_n)$ where $\Delta :\equiv X_1, ..., X_n$. Then, c_B is linear (in the sense of Definition 9.2.1) if and only if $\overrightarrow{y}$ is a (not necessarily contiguous) substring of $\overrightarrow{x}$, that is, $\overrightarrow{y}$ can be obtained from $\overrightarrow{x}$ by dropping some elements. However, if this is the case then we can always reshuffle $\overrightarrow{x}$ into some $\overrightarrow{z}$ so that $\overrightarrow{y}$ appears as a contiguous prefix of $\overrightarrow{z}$. We can therefore state, without loss of generality, that constructors of the form of c_B are linear if and only if $\overrightarrow{y}$ appears as a contiguous prefix of $\overrightarrow{x}$.

When the constructor c_B is specified as $\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ (\mathsf{capp}\ c_A\ \sigma))$ for some $c_A : \mathsf{CTm}\ \Gamma\ (\mathsf{ctor}\ \nabla\ \mathsf{T}^1)$, linearity of c_B can thus be stated as ∇ being a prefix of Δ, or equivalently as the existence of a weakening substitution $\mathsf{WkSub}\ \Delta\ \nabla$.

We finally get to the following general definition: a *linear constructor* with a telescope of parameters Δ is one where the codomain is either T^1, or $\mathsf{T}^2\ (\mathsf{capp}\ c\ \sigma)$ where c is a T^1-valued constructor term and σ is a linear selection of variables from Δ, i.e. a weakening substitution.

$$\frac{\Gamma : \mathsf{Spec} \qquad C : \mathsf{Ctor}\ \Gamma}{\mathsf{LinearCtor}\ C : \mathbf{Type}} \qquad \frac{}{\mathsf{LinearCtor}\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1)}$$

$$\frac{c : \mathsf{CTm}\ \Gamma\ (\mathsf{ctor}\ \nabla\ \mathsf{T}^1) \qquad \sigma : \mathsf{WkSub}\ \Delta\ \nabla}{\mathsf{LinearCtor}\ (\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ (\mathsf{capp}\ c\ \sigma)))} \qquad \frac{}{\mathsf{LinearCtor}\ (\mathsf{ctor}\ (\Delta \rhd\!\rhd \mathsf{T}^1)\ (\mathsf{T}^2\ \mathsf{vz}))}$$

A *linear specification* is one that is only made of linear constructors.

$$\frac{\Gamma : \mathsf{Spec}}{\mathsf{LinearSpec}\ \Gamma : \mathbf{Type}} \qquad \frac{}{\mathsf{LinearSpec}\ \diamond} \qquad \frac{\mathsf{LinearSpec}\ \Gamma \qquad \mathsf{LinearCtor}\ C}{\mathsf{LinearSpec}\ (\Gamma \rhd C)}$$

It follows that indexing from a linear specification yields a linear constructor:

Lemma 9.4.1. For any $\Gamma : \mathsf{Spec}, C : \mathsf{Ctor}\ \Gamma, c : \mathsf{CTm}\ \Gamma\ C$, if $\mathsf{LinearSpec}\ \Gamma$ holds then $\mathsf{LinearCtor}\ C$ holds. ■

Proof. By induction on Γ and c. □

Most of the definitions and proofs throughout the paper are orthogonal to the linearity issue, and will apply to specifications $\Gamma : \mathsf{Spec}$ with no imposed linearity condition. Linearity will play an important role when we discuss the general encoding of IIT eliminators in Chapter 12.

From now on, we will refer to *specifications* as terms Γ of the Spec datatype, thus potentially including non-linearity. We will then call *linear specifications* those specifications Γ for which the predicate $\mathsf{LinearSpec}\,\Gamma$ holds.

Chapter 10

Algebras of IITs

In the previous chapter we have discussed how to formally specify IITs within the subclass of interest. Since our goal is to demonstrate how to encode any IIT as a target-level object, we now have to actually define what it means for an IIT of a given specification to exist in the target theory. In other words, we need a way to map an IIT specification to signatures of target-level types and terms that constitute the intended IIT's type formers, constructors, and eliminators.

A standard approach in the theoretical study of datatypes is to characterize inductive types as certain kinds of *algebras*. For simple inductive types, one can consider algebras of certain endofunctors. For instance, the functor $F : \mathbf{Type} \to \mathbf{Type}$ defined as

$$F :\equiv \lambda X.1 + X$$

specifies the natural numbers as inductively generated by either 0 of successor. This functorial specification gives rise to a notion of algebra of natural numbers, i.e. pairs of a type X together with a function $FX \to X$, as well as a notion of *morphism* between algebras. F-algebras and morphisms between them form a category; the inductive type *being specified*, i.e. the natural numbers, is then taken to be one of such algebras, the *initial* algebra, which turns out to be unique up to isomorphism.

We take a similar approach and characterize IITs as certain kinds of algebras. Rather than using functor algebras, we will define a mapping from IIT specifications to the corresponding type of algebras by recursion on specifications; we thus have a type family Alg indexed over specifications, where $\mathsf{Alg}\,\Gamma$ is the type of algebras of the IIT specified by Γ, and the elements of $\mathsf{Alg}\,\Gamma$ are the algebras of such IIT.

Similarly to how we characterize simple inductive types as *initial* algebras, we will characterize the IIT specified by some $\Gamma : \mathsf{Spec}$ as an algebra $X : \mathsf{Alg}\,\Gamma$ with a suitable *section induction* property [ACD+18] that ensures the algebra is equipped with the induction principle expected of an IIT. We call such algebras *inductive* or *section-inductive.*

In Section 10.1 below we will lay out a systematic way to derive a notion of algebra for a given IIT specification. This will be followed by Section 10.2.2, where we define *displayed IIT algebras* and *sections* of algebras, leading to the notion of section-induction. Section 10.2.2 also discusses the difference between section-induction and initiality, and the reason why we use the former to characterize IITs.

10.1 Algebras

Before defining IIT algebras in the general case, we want to first present concrete instances of algebras for our running example Con/Ty, both as metatheoretic constructions and as target-theory constructions. These will serve as running examples for the rest of the Section, where we generalize these examples into a general notion of IIT algebra parametric over arbitrary specifications.

Example 10.1.1 (metatheory)**.** As a meta-level construction, an algebra of the IIT Con/Ty is given by a pair of types $C : \mathbf{Type}, T : C \to \mathbf{Type}$, and terms giving the algebra operations corresponding to each constructor of the IIT:

$$\begin{aligned}
&\bullet && : C \\
&\_ \rhd \_ && : (c : C) \to T\ c \to C \\
&\iota && : (c : C) \to T\ c \\
&\hat{\pi} && : (c : C) \to (\mathbb{N} \to T\ c) \to T\ c \\
&\bar{\pi} && : (c : C)(a : T\ c) \to T\ (c \rhd a) \to T\ c
\end{aligned}$$

■

Example 10.1.2 (target theory)**.** An algebra of the IIT Con/Ty in the target theory is given by (small) types $C : \mathsf{SType}, T : \ulcorner C \urcorner \to \mathsf{SType}$, and terms:

$$\begin{aligned}
&\bullet : \ulcorner C \urcorner \\
&\rhd : \ulcorner \mathbf{\Pi}\,(c : C)\,(T\,c \Rightarrow C) \urcorner \\
&\iota : \ulcorner \mathbf{\Pi}\, C\, T \urcorner \\
&\hat{\pi} : \ulcorner \mathbf{\Pi}\,(c : C)\,((\mathsf{Nat} \Rightarrow T\ c) \Rightarrow T\,c) \urcorner
\end{aligned}$$

$\bar{\pi} : \ulcorner \mathbf{\Pi}\,(c : C)\,\mathbf{\Pi}\,(a : T\ c)\,(T\,(\triangleright \cdot c \cdot a) \Rightarrow T\ c)\urcorner$

We can see that the structure shown above is a direct, 1-to-1 target-theory translation of the metatheoretic algebra structure in Example 10.1.1.

Note that we use Nat for the type of natural numbers in the target theory. ■

We now assign a notion of algebra to arbitrary IIT specifications. For $\Gamma : \mathsf{Spec}$ we want to calculate the type of algebras for the IIT specified by Γ, including its carrier types and the signature of algebra operations.

All specifiable IITs are made of two sorts; the type of carriers $\mathsf{Ca} : \mathbf{Type}$ of an IIT algebra is then easily defined once and for all, independent of the specification: $\mathsf{Ca} :\equiv \Sigma\,(A : \mathsf{SType})\,(\ulcorner A \urcorner \to \mathsf{SType})$. Given a specification Γ and carriers $F : \mathsf{Ca}$, we define a type $\Gamma^{\mathsf{A}}_{F} : \mathbf{Type}$ of algebra operations induced by the constructors in Γ. This structure is essentially an iterated product of target-theory types, one for each constructor.

We define Γ^{A}_{F} mutually with auxiliary algebra operators for each sort of the specification QIIT. Each of these operators define a family of (target-level) types, indexed by algebra structures of the corresponding indices.

$$\begin{array}{lll}
\Gamma : \mathsf{Spec} & \vdash & \Gamma^{\mathsf{A}}_{F}\ : \mathbf{Type} \\
C : \mathsf{Ctor}\ \Gamma & \vdash & C^{\mathsf{A}}_{F} : \Gamma^{\mathsf{A}}_{F} \to \mathsf{SType} \\
\Delta : \mathsf{Params}\ \Gamma & \vdash & \Delta^{\mathsf{A}}_{F} : \Gamma^{\mathsf{A}}_{F} \to \mathsf{SType} \\
T : \mathsf{Ty}\ \Gamma\ \Delta & \vdash & T^{\mathsf{A}}_{F}\ : (\gamma : \Gamma^{\mathsf{A}}_{F}) \to \ulcorner \Delta^{\mathsf{A}}_{F}\ \gamma \urcorner \to \mathsf{SType}
\end{array}$$

We also define algebra operators lifting the action of weakenings/substitutions and (constructor) terms to algebra structures:

$$\begin{array}{lll}
w : \mathsf{Wk}\ \Omega\ \Gamma & \vdash & w^{\mathsf{A}}_{F} : \Omega^{\mathsf{A}}_{F} \to \Gamma^{\mathsf{A}}_{F} \\
\sigma : \mathsf{Sub}\ \Delta\ \nabla & \vdash & \sigma^{\mathsf{A}}_{F}\ : (\gamma : \Gamma^{\mathsf{A}}_{F}) \to \ulcorner \Delta^{\mathsf{A}}_{F}\ \gamma \urcorner \to \ulcorner \nabla^{\mathsf{A}}_{F}\ \gamma \urcorner \\
t : \mathsf{Tm}\ \Gamma\ \Delta\ T & \vdash & t^{\mathsf{A}}_{F}\ : (\gamma : \Gamma^{\mathsf{A}}_{F})\ (\delta : \ulcorner \Delta^{\mathsf{A}}_{F}\ \gamma \urcorner) \to \ulcorner T^{\mathsf{A}}_{F}\ \gamma\ \delta \urcorner \\
c : \mathsf{CTm}\ \Gamma\ C & \vdash & c^{\mathsf{A}}_{F}\ : (\gamma : \Gamma^{\mathsf{A}}_{F}) \to \ulcorner C^{\mathsf{A}}_{F}\ \gamma \urcorner
\end{array}$$

Remark 10.1.1. We overload the symbol $\_^{\mathsf{A}}$, and rely on metavariables for disambiguation. We will often drop the subscript $\__{F}$ when clear from context. ■

We define Γ^{A}_{F} by induction on Γ, building up the iterated pair type via a recursive process:

$$\begin{aligned}
\diamond^{\mathsf{A}}_{F} &:\equiv \mathbb{1} \\
(\Gamma \rhd C)^{\mathsf{A}}_{F} &:\equiv \Sigma\ (\gamma : \Gamma^{\mathsf{A}}_{F})(\ulcorner C^{\mathsf{A}}\ \gamma \urcorner)
\end{aligned}$$

Here $C^{\mathsf{A}}\ \gamma$ takes the specification of a constructor C, and produces the type of algebra operations corresponding to C. As we are dealing with algebra structures in the target theory, these operations are target-level terms belonging to some target-level types, hence why the operator C^{A} produces a STerm. Constructor algebras are always functions from algebras of their parameters, to algebras of their conclusion type. Specifically, they are a **Π**-type from a **Σ**-type of parameters as domain, and one of the two carrier types as codomain:

$$(\mathsf{ctor}\ \Delta\ T)^{\mathsf{A}} :\equiv \mathbf{\Pi}\ (\delta : \Delta^{\mathsf{A}}\ \gamma)(T^{\mathsf{A}}\ \gamma\ \delta)$$

Here, $\Delta^{\mathsf{A}}\ \gamma$ recursively builds up the target-level product of parameters, according to their specification:

$$\begin{aligned}
\bullet^{\mathsf{A}}\ \gamma &:\equiv \mathbf{1} \\
(\Delta \rhd\!\rhd T)^{\mathsf{A}}\ \gamma &:\equiv \mathbf{\Sigma}\ (\delta : \Delta^{\mathsf{A}}\ \gamma)(T^{\mathsf{A}}\ \gamma\ \delta) \\
(\Delta[w])^{\mathsf{A}} &:\equiv \Delta^{\mathsf{A}}\ (w^{\mathsf{A}}\ \gamma)
\end{aligned}$$

We do similarly for parameter types and base types. Note that base types in the algebra structure are always, as expected, one of the two carrier types.

$$\begin{aligned}
(\pi\ A\ B)^{\mathsf{A}}\ \gamma\ \delta &:\equiv \mathbf{\Pi}\ (a : A)\ (B^{\mathsf{A}}\ \gamma\ \langle \delta, a \rangle) & \qquad (\mathsf{ext}\ X)^{\mathsf{A}}\ \_\ \_ &:\equiv X \\
(X[w])^{\mathsf{A}}\ \gamma\ \delta &:\equiv X^{\mathsf{A}}\ (w^{\mathsf{A}}\ \gamma)\ \delta & (\mathsf{T}^1)^{\mathsf{A}}_{F}\ \gamma\ \delta &:\equiv \pi_1\ F \\
(X[\sigma])^{\mathsf{A}}\ \gamma\ \delta &:\equiv X^{\mathsf{A}}\ \gamma\ (\sigma^{\mathsf{A}}\ \gamma\ \delta) & (\mathsf{T}^2\ t)^{\mathsf{A}}_{F}\ \gamma\ \delta &:\equiv \pi_2\ F\ (t^{\mathsf{A}}\ \gamma\ \delta)
\end{aligned}$$

In the case of a base type $\mathsf{T}^2\ t$, we apply the second carrier type to the algebra of t.

The algebra operators on terms act as indexing functions, taking the algebra of a context as input, and returning its contents at the specified index.

$$
\begin{aligned}
\mathsf{vz}^{\mathsf{A}}\ \gamma\ \delta &:\equiv \mathsf{snd}\ \delta \\
(\mathsf{vs}\ t)^{\mathsf{A}}\ \gamma\ \delta &:\equiv t^{\mathsf{A}}\ \gamma\ (\mathsf{fst}\ \delta) \\
(\mathsf{capp}\ c\ \sigma)^{\mathsf{A}}\ \gamma\ \delta &:\equiv c^{\mathsf{A}}\ \gamma \cdot \delta \\
(\mathsf{ext}\ x)^{\mathsf{A}}\ \gamma\ \delta &:\equiv x \\
(\mathsf{lm}\ f)^{\mathsf{A}}\ \gamma\ \delta &:\equiv \boldsymbol{\lambda} x\,.\,t^{\mathsf{A}}\ \gamma\ \langle \delta, x\rangle \\
(\mathsf{ap}\ f)^{\mathsf{A}}\ \gamma\ \langle \delta, x\rangle &:\equiv t^{\mathsf{A}}\ \gamma\ \delta \cdot x \\
(t[w])^{\mathsf{A}}\ \gamma\ \delta &:\equiv t^{\mathsf{A}}\ (w^{\mathsf{A}}\ \gamma)\ \delta \\
(t[\sigma])^{\mathsf{A}}\ \gamma\ \delta &:\equiv t^{\mathsf{A}}\ \gamma\ (\sigma^{\mathsf{A}}\ \gamma\ \delta)
\end{aligned}
\qquad
\begin{aligned}
\mathsf{cvz}^{\mathsf{A}}\ \gamma &:\equiv \pi_2\ \gamma \\
(\mathsf{cvs}\ c)^{\mathsf{A}}\ \gamma &:\equiv c^{\mathsf{A}}\ (\pi_1\ \gamma) \\
(c[w])^{\mathsf{A}}\ \gamma &:\equiv c^{\mathsf{A}}\ (w^{\mathsf{A}}\ \gamma)
\end{aligned}
$$

Finally, the operators on weakenings and substitutions transform algebras in the same way weakenings and substitutions transform contexts.

$$
\begin{aligned}
\mathsf{id}^{\mathsf{A}}\ \gamma\ \delta &:\equiv \delta \\
\mathsf{drop}^{\mathsf{A}}\ \gamma\ \delta &:\equiv \mathsf{fst}\ \delta \\
(\mathsf{ext}\ \sigma\ t)^{\mathsf{A}}\ \gamma\ \delta &:\equiv \langle \sigma^{\mathsf{A}}\ \gamma\ \delta, t^{\mathsf{A}}\ \gamma\ \delta\rangle \\
(\sigma \circ \tau)^{\mathsf{A}}\ \gamma\ \delta &:\equiv \tau^{\mathsf{A}}\ \gamma\ (\sigma^{\mathsf{A}}\ \gamma\ \delta) \\
(\sigma[w])^{\mathsf{A}}\ \gamma\ \delta &:\equiv \sigma^{\mathsf{A}}\ (w^{\mathsf{A}}\ \gamma)\ \delta
\end{aligned}
\qquad
\begin{aligned}
\mathsf{id}^{\mathsf{A}}\ \gamma &:\equiv \gamma \\
\mathsf{drop}^{\mathsf{A}}\ \gamma &:\equiv \pi_1\ \gamma \\
(w_1 \circ w_2)^{\mathsf{A}}\ \gamma &:\equiv {w_2}^{\mathsf{A}}\ ({w_1}^{\mathsf{A}}\ \gamma)
\end{aligned}
$$

Example 10.1.3. Consider the specification $\Theta : \mathsf{Spec}$ of Con/Ty as defined in Section 9.4.3. Given carrier types $C : \mathsf{SType}, T : \ulcorner C \urcorner \to \mathsf{SType}$, the expression $\Theta^{\mathsf{A}}_{(C,T)}$ computes to the following type

$$
\begin{aligned}
&\mathbb{1}\times \\
&(\bullet : \ulcorner \mathbf{1} \Rightarrow C \urcorner)\times \\
&(\triangleright : \ulcorner (\mathbf{1} \times \boldsymbol{\Sigma}\ C\ T) \Rightarrow C \urcorner)\times \\
&(\iota : \ulcorner \boldsymbol{\Pi}\ (\delta : \mathbf{1} \times C)\ (T\ (\mathsf{snd}\ \delta)) \urcorner)\times \\
&(\hat{\pi} : \ulcorner \boldsymbol{\Pi}\ (\delta : \mathbf{1} \times \boldsymbol{\Sigma}\ (c : C)\ (\mathsf{Nat} \Rightarrow T\ c))\ (T\ (\mathsf{fst}\ (\mathsf{snd}\ \delta))) \urcorner)\times \\
&(\bar{\pi} : \ulcorner \boldsymbol{\Pi}\ (\delta : \mathbf{1} \times \boldsymbol{\Sigma}\ (c : C)\ \boldsymbol{\Sigma}(a : T\ c)(T\ (\triangleright \cdot \langle c, a\rangle)))\ (T\ (\mathsf{fst}\ (\mathsf{snd}\ \delta))) \urcorner)
\end{aligned}
$$

Being the result of what is essentially a simple code generation algorithm, this type isn't particularly readable and contains many spurious elements, like products with the unit type. After some simplification, we obtain the following equivalent type

$$
\begin{aligned}
&(\bullet : \ulcorner C \urcorner)\times\\
&(\triangleright : \ulcorner \mathbf{\Pi}\ (\mathbf{\Sigma}\ C\ T)\ C \urcorner)\times\\
&(\iota : \ulcorner \mathbf{\Pi}\ C\ T \urcorner)\times\\
&(\hat{\pi} : \ulcorner \mathbf{\Pi}\ (p : \mathbf{\Sigma}\ (c : C)\ (\mathsf{Nat} \Rightarrow T\ c))\ (T\ (\mathsf{fst}\ p)) \urcorner)\times\\
&(\bar{\pi} : \ulcorner \mathbf{\Pi}\ (p : \mathbf{\Sigma}\ (c : C)\ \mathbf{\Sigma}(a : T\ c)(T\ (\triangleright \cdot \langle c, a\rangle)))\ (T\ (\mathsf{fst}\ p)) \urcorner)
\end{aligned}
$$

It is easy to see that this structure is just an equivalent, *uncurried* version of that presented in Example 10.1.2. ■

Remark 10.1.2. As observed in Example 10.1.3, the structures that are "code-generated" by the algebra operators may appear syntactically different from what we would expect when specifying algebras by hand within concrete examples. The difference is, however, semantically meaningless, as we can find straightforward definitional equivalences going back and forth between the different representations. From now on we will regard these different forms as completely equivalent representations for the same object, choosing to use one over the other for pure convenience of presentation. ■

We wrap up the section with a formal definition of IIT algebras.

Definition 10.1.1. For a specification Γ : Spec, an algebra $\mathsf{Alg}\ \Gamma$ of the IIT specified by Γ is a dependent pair, given by carriers F : Ca and a structure γ : Γ^{A}_{F}. ■

10.2 Morphisms, sections, and induction

Defining the algebras is only one step in characterizing what counts as an IIT. Any inductive type has, generally, many algebras, not all of which correspond to the type itself. Consider, for example, the whole collection of algebras of the natural numbers, which includes many types (like the reals) much "larger" than them.

What we are looking for are algebras that are the "smallest" in a suitable sense. This property can be characterized as an *induction principle*. When formulating the rules of type theory, this induction principle is usually expressed in the form of *elimination rules* and *eliminator functions*, together with *computation rules* expressing some definitional equations that the eliminators satisfy. As exemplified in Section 8.1, eliminators take input data in the form of types (aka *motives*) and functions (aka *methods*) describing the target and computational behaviour of the induction. These data closely match the structure of the IIT and are indexed by it.

Induction can be framed categorically by organizing algebras into a category: given the category of algebras of some inductive type, the inductive type itself is identified with the *initial* algebra, with the unique morphism out of it representing the non-dependent induction principle. Initiality implies uniqueness (up to isomorphism), hence we can refer to it as *the* initial algebra.

Initiality implements a non-dependent elimination principle, which is provably equivalent to the dependent one in an extensional setting. Unfortunately *funext* is not available in our setting, so we cannot define IITs to be initial objects in their respective category of algebras, as this would not be sufficient to equip them with the dependent eliminators.

We instead rely on the notion of *section*, which we define as a mapping between an algebra and *displayed algebras* over it. *Displayed algebras* are essentially indexed versions of regular algebras, and as shown in Section 8.1 they can be used to exactly provide the data of motives and methods of eliminators. Sections can be though of as a straightforward dependent generalization of an algebra morphism, as the only essential difference is that the codomain algebra of the mapping "depends" on the domain algebra. In fact, we can define algebra morphisms as sections targeting constant displayed algebras.

Sections precisely capture the notion of dependent eliminators, hence allowing us to specify what we mean by "inductive-inductive type":

Definition 10.2.1. Given a specification Γ, an algebra *alg* : $\mathsf{Alg}\ \Gamma$ is *section-inductive* if for any displayed algebra over *alg* there exists a section between them. ■

Definition 10.2.2. Given a specification Γ, an IIT specified by Γ is a section-inductive algebra in *alg* : $\mathsf{Alg}\ \Gamma$. ■

Note that we refer to section-inductive algebras as "an IIT" rather than "the IIT" fitting a given specification. This is because section-induction, unlike initiality, does not necessarily imply uniqueness (see Remark 10.2.2.)

In Section 10.2.1 down below we formally define displayed algebras generally for any IIT specification, similarly to what we did in Section 10.1 for regular algebras. Section 10.2.2 then defines sections between regular and displayed IIT algebras.

10.2.1 Displayed algebras

Displayed algebras can be understood as algebras indexed over regular algebras. We take a look at a couple of concrete examples before tackling the general case.

Example 10.2.1 (metatheory)**.** Section 8.1 already provided the specification of displayed algebras over a Con/Ty algebra $(\mathsf{Con}, \mathsf{Ty}, \bullet, \_ \rhd \_, \iota, \bar{\pi}, \hat{\pi})$ as the following collection of types and operations:

$$\begin{aligned}
&\mathsf{Con}^D &&: \mathsf{Con} \to \mathbf{Type}\\
&\mathsf{Ty}^D &&: \mathsf{Con}^D\ \Gamma \to \mathsf{Ty}\ \Gamma \to \mathbf{Type}\\
&\bullet^D &&: \mathsf{Con}^D\ \bullet\\
&\_ \rhd^D \_ &&: (\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ A \to \mathsf{Con}^D\ (\Gamma \rhd A)\\
&\iota^D &&: (\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ (\iota\ \Gamma)\\
&\bar{\pi}^D &&: (\Gamma^D : \mathsf{Con}^D\ \Gamma)(A^D : \mathsf{Ty}^D\ \Gamma^D\ A)(B^D : \mathsf{Ty}^D\ (\Gamma^D \rhd^D A)\ B)\\
& &&\to \mathsf{Ty}^D\ \Gamma^D\ (\bar{\pi}\ \Gamma\ A\ B)\\
&\hat{\pi}^D &&: (\Gamma^D : \mathsf{Con}^D\ \Gamma) \to ((n : \mathbb{N}) \to \mathsf{Ty}^D\ \Gamma^D\ (F\ n)) \to \mathsf{Ty}^D\ \Gamma^D\ (\hat{\pi}\ \Gamma\ F)
\end{aligned}$$

■

Example 10.2.2 (target theory)**.** Let us consider a target-level Con/Ty-algebra made of carriers C, T, and algebra structure $\bullet, \rhd, \iota, \hat{\pi}, \bar{\pi}$, as shown in Example 10.1.2. A target-level Con/Ty displayed algebra indexed over it then consists of a pair of families

$$\begin{aligned}
&C^D : \ulcorner C \urcorner \to \mathsf{SType}\\
&T^D : (c : \ulcorner C \urcorner)\ (c^D : \ulcorner C^D\ c \urcorner) \to \ulcorner T\ c \urcorner \to \mathsf{SType}
\end{aligned}$$

and displayed algebra operations:

$$\begin{aligned}
&\bullet^D : \ulcorner C^D\ \bullet \urcorner\\
&\rhd^D : \ulcorner \mathbf{\Pi}(c : C)(t : T\ c)(c^D : C^D\ c)(T^D\ c\ c^D\ t \Rightarrow C^D\ (\rhd \cdot c \cdot t)) \urcorner\\
&\iota^D : \ulcorner \mathbf{\Pi}\ (c : C)(c^D : C^D\ c)(T^D\ c\ c^D\ (\iota \cdot c)) \urcorner\\
&\hat{\pi}^D : \ulcorner \mathbf{\Pi}\ (c : C)(f : \mathsf{Nat} \Rightarrow T\ c)(c^D : C^D\ c)(f^D : \mathbf{\Pi}(n : \mathsf{Nat})(T^D\ c\ c^D\ (f \cdot n)))\\
&\qquad (T^D\ c\ c^D\ (\hat{\pi} \cdot c \cdot f)) \urcorner\\
&\bar{\pi}^D : \ulcorner \mathbf{\Pi}\ (c : C)(a : T\ c)(b : T\ (\rhd \cdot c \cdot a))\\
&\qquad (c^D : C^D\ c)(a^D : T^D\ c\ c^D\ a)(b^D : T^D\ (\rhd \cdot c \cdot a)\ (\rhd^D \cdot c \cdot a \cdot c^D \cdot a^D)\ b)\\
&\qquad (T^D\ c\ c^D\ (\bar{\pi} \cdot c \cdot a \cdot b)) \urcorner
\end{aligned}$$

The structure above is a direct target-level translation of the structure in Example 10.2.1, modulo the absence of implicit parameters which are spelled out explicitly here.

■

We now want to define displayed algebras in the general case, for arbitrary IIT specifications. We define operators that, given $\Gamma : \mathsf{Spec}$ and an object in $\mathsf{Alg}\ \Gamma$, calculate the type of displayed algebras over it.

Carriers of displayed algebras are easily calculated as families indexed over regular carriers:

$$\begin{aligned}
&\mathsf{Ca}^{\mathsf{D}} : \mathsf{Ca} \to \mathbf{Type}\\
&\mathsf{Ca}^{\mathsf{D}}\ (A, B) :\equiv \Sigma\ (A^D : \ulcorner A \urcorner \to \mathsf{SType})\\
&\qquad\qquad\qquad ((a : \ulcorner A \urcorner) \to \ulcorner A^D\ a \urcorner \to \ulcorner B\ a \urcorner \to \mathsf{SType})
\end{aligned}$$

We define a displayed algebra operator $\_^{\mathsf{D}}$ on specifications such that if $(F, \gamma) : \mathsf{Alg}\ \Gamma$ is an IIT algebra and $D : \mathsf{Ca}^{\mathsf{D}}\ F$ are displayed carriers, then $\Gamma^{\mathsf{D}}_{D}\ \gamma : \mathbf{Type}$ is the signature of the displayed algebra structure on D, i.e. an iterated product of the types of algebra operations. Example 10.2.2 provides an example of this iterated product for the Con/Ty case.

Similarly to the algebra operators for regular IIT algebras (Section 10.1), we define a displayed algebra operator for each sort of the QIIT of specifications. For any $F : \mathsf{Ca}, D : \mathsf{Ca}^{\mathsf{D}}\ F$, we define:

$$\begin{array}{ll}
\Gamma : \mathsf{Spec} & \vdash \Gamma^{\mathsf{D}}_{D} : \Gamma^{\mathsf{A}}_{F} \to \mathbf{Type}\\
C : \mathsf{Ctor}\ \Gamma & \vdash C^{\mathsf{D}}_{D} : \Gamma^{\mathsf{D}}_{D}\ \gamma \to \ulcorner C^{\mathsf{A}}_{F}\ \gamma \urcorner \to \mathsf{SType}\\
\Delta : \mathsf{Params}\ \Gamma & \vdash \Delta^{\mathsf{D}}_{D} : \Gamma^{\mathsf{D}}_{D}\ \gamma \to \ulcorner \Delta^{\mathsf{A}}_{F}\ \gamma \urcorner \to \mathsf{SType}\\
T : \mathsf{Ty}\ \Gamma\ \Delta & \vdash T^{\mathsf{D}}_{D} : (\gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma)(\delta^{\mathsf{D}} : \ulcorner \Delta^{\mathsf{D}}_{D}\ \gamma^D\ \delta \urcorner) \to \ulcorner T^{\mathsf{A}}_{F}\ \gamma\ \delta \urcorner \to \mathsf{SType}
\end{array}$$

These are defined together with operators lifting the action of weakenings, substitutions, and (constructor) terms to displayed algebras:

$$\begin{array}{ll}
t : \mathsf{Tm}\ \Gamma\ \Delta\ T & \vdash t^{\mathsf{D}}_{D} : \{\gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma\}(\delta^D : \ulcorner \Delta^{\mathsf{D}}_{D}\ \gamma^D\ \delta \urcorner) \to \ulcorner T^{\mathsf{D}}_{D}\ \gamma^D\ \delta^D\ (t^{\mathsf{A}}_{F}\ \gamma\ \delta) \urcorner\\
c : \mathsf{CTm}\ \Gamma\ C & \vdash c^{\mathsf{D}}_{D} : (\gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma) \to \ulcorner C^{\mathsf{D}}_{D}\ \gamma^D\ (c^{\mathsf{A}}_{F}\ \gamma) \urcorner\\
w : \mathsf{Wk}\ \Gamma\ \Omega & \vdash w^{\mathsf{D}}_{D} : \Gamma^{\mathsf{D}}_{D}\ \gamma \to \Omega^{\mathsf{D}}_{D}\ (w^{\mathsf{A}}_{F}\ \gamma)\\
\sigma : \mathsf{Sub}\ \Delta\ \nabla & \vdash \sigma^{\mathsf{D}}_{D} : \{\gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma\} \to \ulcorner \Delta^{\mathsf{D}}_{D}\ \gamma^D\ \delta \urcorner \to \ulcorner \nabla^{\mathsf{D}}_{D}\ \gamma^D\ (\sigma^{\mathsf{A}}_{F}\ \gamma\ \delta) \urcorner
\end{array}$$

In line with the idea of displayed algebras as indexed over regular algebras, the displayed algebra operators are themselves an indexed version of the regular algebra operators defined in the previous section. We summarize their definitions

$$\diamond^{\mathsf{D}}_{D}\ \_ :\equiv \mathbf{1}$$
$$(\Gamma \triangleright C)^{\mathsf{D}}_{D}\ (\gamma, c) :\equiv \Sigma\ (\gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma)\ \ulcorner C^{\mathsf{D}}\ \gamma^D\ c \urcorner$$

$$(\mathsf{ctor}\ \Delta\ X)^{\mathsf{D}}\ \gamma^D\ c :\equiv$$
$$\quad \mathbf{\Pi}(\delta : \Delta^A\ \_)(\Delta^{\mathsf{D}}\ \gamma^D\ \delta \Rightarrow X^{\mathsf{D}}\ \gamma^D\ \delta^D\ (c \cdot \delta))$$

$$\bullet^{\mathsf{D}}\ \_\ \_ :\equiv \mathbf{1}$$
$$(\Delta \triangleright\!\triangleright T)^{\mathsf{D}}\ \gamma^D\ \langle \delta, x \rangle :\equiv$$
$$\quad \mathbf{\Sigma}(\delta^D : \Delta^{\mathsf{D}}\ \gamma^D\ \delta)(T^{\mathsf{D}}\ \gamma^D\ \delta^D\ x)$$
$$(\Delta[w])^{\mathsf{D}}\ \gamma^D\ \delta :\equiv \Delta^{\mathsf{D}}\ (w^{\mathsf{D}}\ \gamma^D)\ \delta$$

$$(\mathsf{ext}\ A)^{\mathsf{D}}\ \gamma^D\ \delta^D\ a :\equiv \mathbf{1}$$
$$(\pi\ A\ B)^{\mathsf{D}}\ \gamma^D\ \delta^D\ f :\equiv$$
$$\quad \mathbf{\Pi}(x : A)(B^{\mathsf{D}}\ \gamma^D\ \langle \delta^D, * \rangle\ (f \cdot x))$$
$$(A[\sigma])^{\mathsf{D}}\ \gamma^D\ \delta^D\ a :\equiv A^{\mathsf{D}}\ \gamma^D\ (\sigma^{\mathsf{D}}\ \delta^D)\ a$$
$$(A[w])^{\mathsf{D}}\ \gamma^D\ \delta^D\ a :\equiv A^{\mathsf{D}}\ (w^{\mathsf{D}}\ \gamma^D)\ \delta^D\ a$$
$$(\mathsf{T}^1)^{\mathsf{D}}_{D}\ \gamma^D\ \delta^D\ x :\equiv \pi_1\ D\ x$$
$$(\mathsf{T}^2\ t)^{\mathsf{D}}_{D}\ \{\gamma\}\ \gamma^D\ \{p\}\ \delta^D\ y :\equiv$$
$$\quad \pi_2\ D\ (t^A\ \gamma\ p)\ (t^{\mathsf{D}}\ \gamma^D\ \delta^D)\ y$$

$$\mathsf{cvz}^{\mathsf{D}}\ \gamma^D :\equiv \pi_2\ \gamma^D$$
$$(\mathsf{cvs}\ c)^{\mathsf{D}}\ \gamma^D :\equiv c^{\mathsf{D}}\ (\pi_1\ \gamma^D)$$
$$(c[w])^{\mathsf{D}}\ \gamma^D :\equiv c^{\mathsf{D}}\ (w^{\mathsf{D}}\ \gamma^D)$$

$$\mathsf{id}^{\mathsf{D}}\ \delta^D :\equiv \delta^D$$
$$\mathsf{drop}^{\mathsf{D}}\ \delta^D :\equiv \mathsf{fst}\ \delta^D$$
$$(\mathsf{ext}\ \sigma\ t)^{\mathsf{D}}\ \delta^D :\equiv \langle \sigma^{\mathsf{D}}\ \delta^D, t^{\mathsf{D}}\ \delta^D \rangle$$
$$(\sigma \circ \tau)^{\mathsf{D}}\ \delta^D :\equiv \tau^{\mathsf{D}}\ (\sigma^{\mathsf{D}}\ \delta^D)$$
$$(\sigma[w])^{\mathsf{D}}\ \{\gamma^D\}\ \delta^D :\equiv \sigma^{\mathsf{D}}\ \{w^{\mathsf{D}}\ \gamma^D\}\ \delta^D$$

$$\mathsf{id}^{\mathsf{D}}\ \gamma^D :\equiv \gamma^D$$
$$\mathsf{drop}^{\mathsf{D}}\ \gamma^D :\equiv \pi_1\ \gamma^D$$
$$(w_1 \circ w_2)^{\mathsf{D}}\ \gamma^D :\equiv w_2{}^{\mathsf{D}}\ (w_1{}^{\mathsf{D}}\ \gamma^D)$$

$$\mathsf{vz}^{\mathsf{D}}\ \delta^D :\equiv \mathsf{snd}\ \delta^D$$
$$(\mathsf{vs}\ t)^{\mathsf{D}}\ \delta^D :\equiv t^{\mathsf{D}}\ (\mathsf{fst}\ \delta^D)$$
$$(\mathsf{ext}\ A\ x)^{\mathsf{D}}\ \delta^D :\equiv *$$
$$(\mathsf{capp}\ c\ \sigma)^{\mathsf{D}}\ \{\gamma^D\}\ \delta^D :\equiv c^{\mathsf{D}}\ \gamma^D \cdot \_ \cdot \sigma^{\mathsf{D}}\ \delta^D$$
$$(\mathsf{ap}\ t)^{\mathsf{D}}\ \{\delta\}\ \delta^D :\equiv t^{\mathsf{D}}\ (\mathsf{fst}\ \delta^D) \cdot \mathsf{fst}\ \delta$$
$$(\mathsf{lm}\ t)^{\mathsf{D}}\ \{\delta\}\ \delta^D :\equiv \boldsymbol{\lambda} x\,.\, t^{\mathsf{D}}\ \{\langle \delta, x \rangle\} \langle \delta^D, * \rangle$$
$$(t[\sigma])^{\mathsf{D}}\ \delta^D :\equiv t^{\mathsf{D}}\ (\sigma^{\mathsf{D}}\ \gamma^D\ \delta^D)$$
$$(t[w])^{\mathsf{D}}\ \{\gamma^D\}\ \delta^D :\equiv t^{\mathsf{D}}\ \{w^{\mathsf{D}}\ \gamma^D\}\ \delta^D$$

Figure 10.1: Displayed IIT algebra operators

in Figure 10.1. Note that the displayed algebra of an external type ($\mathsf{ext}\ A$) is trivial, as displayed algebras can only index over the regular carriers in F. The other definitions are straightforward generalizations of the operators from Section 10.1.

Example 10.2.3. Let $((C, T), \theta) : \mathsf{Alg}\,\Theta$ be a Con/Ty algebra[1]. Given displayed carriers C^D, T^D, the displayed algebra structure computed by $\Theta^{\mathsf{D}}_{(C^D, T^D)}\ \theta$ is precisely the structure listed in Example 10.2.2, modulo the usual curried/uncurried functions and spurious multiplications by the unit type.

[1]Recall that Θ is the formal specification of Con/Ty as a term of Spec

■

We conclude with a formal definition of IIT displayed algebras.

Definition 10.2.3. For a specification $\Gamma : \mathsf{Spec}$, and an algebra $(F, \gamma) : \mathsf{Alg}\ \Gamma$, a displayed algebra over (F, γ) is a dependent pair given by carrier types $D : \mathsf{Ca}^{\mathsf{D}}\ F$ and displayed algebra structure $\gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma$. ■

The next section will relate IIT algebras and their corresponding displayed algebras via the notion of section. We will see how displayed algebras can be used to specify the data of eliminators for IITs, and how sections can be used to specify the existence of eliminators given that data.

10.2.2 Sections

The data of an algebra morphism between two algebras comprises maps between the respective carriers, together with proofs that these maps "respect" the algebra operations. The data of a section are a generalized, indexed version of that of an algebra morphism. We illustrate this with a couple of examples.

Example 10.2.4 (metatheory)**.** We have already encountered an instance of a section in the context of the Con/Ty example of Section 8.1, where we specified the elimination principle for Con/Ty: for any displayed algebra $(C^D, T^D, \bullet^D, \triangleright^D, \iota^D, \bar{\pi}^D, \hat{\pi}^D)$ over the IIT, we had a pair of functions into the carriers of the displayed algebra:

$$\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}} : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma \\
&\mathsf{elim}_{\mathsf{Ty}} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ \Gamma\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ A
\end{aligned}$$

such that these functions commute with the IIT constructors and the corresponding displayed functions:

$$\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}}\ \bullet \equiv \bullet^D \\
&\mathsf{elim}_{\mathsf{Con}}\ (\Gamma \triangleright A) \equiv \mathsf{elim}_{\mathsf{Con}}\ \Gamma \triangleright^D \mathsf{elim}_{\mathsf{Ty}}\ A \\
&\mathsf{elim}_{\mathsf{Ty}}\ (\iota\ \Gamma) \equiv \iota^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma) \\
&\mathsf{elim}_{\mathsf{Ty}}(\hat{\pi}\ \Gamma\ F) \equiv \hat{\pi}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\lambda n.\mathsf{elim}_{\mathsf{Ty}}\ (F\, n)) \\
&\mathsf{elim}_{\mathsf{Ty}}(\bar{\pi}\ \Gamma\ A\ B) \equiv \bar{\pi}^D\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)\ (\mathsf{elim}_{\mathsf{Ty}}\ A)\ (\mathsf{elim}_{\mathsf{Ty}}\ B)
\end{aligned}$$

The structure above specifies none other than a section between the "initial" algebra Con/Ty, and the assumed displayed algebra over it.

■

Example 10.2.5 (target theory)**.** Take a Con/Ty algebra $(C, T, \bullet, \triangleright, \iota, \hat{\pi}, \bar{\pi})$ and displayed algebra $(C^D, T^D, \bullet^D, \triangleright^D, \iota^D, \hat{\pi}^D, \bar{\pi}^D)$ over it, instances of which are shown in Example 10.1.2 and Example 10.2.2. A section between these algebras is given by the data of a pair of functions between the respective carriers:

$$
\begin{aligned}
&f_C : \ulcorner \mathbf{\Pi}\; C\; C^D \urcorner \\
&f_T : \ulcorner \mathbf{\Pi}\; (c : C)\; \mathbf{\Pi}\; (t : T\; c)(T^D\; c\; (f_C \cdot c)\; t) \urcorner
\end{aligned}
$$

such that these functions "respect" the algebra structure arising from the constructors. In other words, we require the functions f_C, f_T to satisfy the following meta-level equations:

$$
\begin{aligned}
f_C \cdot \bullet &= \bullet^D \\
f_C \cdot (\triangleright \cdot c \cdot t) &= \triangleright^D \cdot c \cdot t \cdot (f_C \cdot c) \cdot (f_T \cdot c \cdot t) \\
f_T \cdot c \cdot (\iota \cdot c) &= \iota^D \cdot c \cdot (f_C \cdot c) \\
f_T \cdot c \cdot (\hat{\pi} \cdot c \cdot f) &= \hat{\pi}^D \cdot c \cdot f \cdot (f_C \cdot c) \cdot (\boldsymbol{\lambda} n \,.\, f_T \cdot c \cdot (f \cdot n)) \\
f_T \cdot c \cdot (\bar{\pi} \cdot c \cdot a \cdot b) &= \bar{\pi}^D \cdot c \cdot a \cdot b \cdot (f_C \cdot c) \cdot (f_T \cdot c \cdot a)(f_T \cdot (\triangleright \cdot c \cdot a) \cdot b)
\end{aligned}
$$

Note that the equations above are meta-theoretic identifications between target-level terms. From the point of view of the target theory, this means that section maps respect the algebra structure *strictly*, i.e. up to target-level *definitional equality*.

■

We now want to calculate the structure of a section, with its functions and equations, for arbitrary IIT specifications $\Gamma : \mathsf{Spec}$ and pairs of algebras $(F, \gamma) : \mathsf{Alg}\;\Gamma$ and $(D, \gamma^D) : \mathsf{Alg}^\mathsf{D}\;\Gamma\;(F, \gamma)$.

The functional component is straightforward: we define a structure $\mathsf{Map}\,F\,D : \mathbf{Type}$ between $F : \mathsf{Ca}$ and $D : \mathsf{Ca}^\mathsf{D}\;F$ as a pair of target-level function terms:

$$
\begin{aligned}
&\mathsf{Map} : (F : \mathsf{Ca}) \to \mathsf{Ca}^\mathsf{D}\; F \to \mathbf{Type} \\
&\mathsf{Map}\; (A, B)\; (A^D, B^D) :\equiv \\
&\qquad \Sigma\; (f_A : \ulcorner \mathbf{\Pi}\; A\; A^D \urcorner)\; (\ulcorner \mathbf{\Pi}\; (a : A)\; \mathbf{\Pi}\; (b : B\; a)(B^D\; a\; (f_A \cdot a)\; b) \urcorner)
\end{aligned}
$$

To determine the equational part of the section structure we define section operators, one for each sort of the datatype of specifications. Each of these operators compute the effect of applying the section maps to IIT algebras. Reflecting the nature of sections as relations, the types of section operators are indexed by pairs of algebras and displayed algebras on some carriers F, D, as well as section maps $f : \mathsf{Map}\ F\ D$:

$$\begin{array}{ll}
\Gamma : \mathsf{Spec} & \vdash \Gamma^{\mathsf{S}}_f \ : (\gamma : \Gamma^{\mathsf{A}}_F) \to \Gamma^{\mathsf{D}}_D\ \gamma \to \mathbf{Type} \\
C : \mathsf{Ctor}\ \Gamma & \vdash C^{\mathsf{S}}_f \ : (s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D)(c : \ulcorner C^{\mathsf{A}}_F\ \gamma \urcorner) \to C^{\mathsf{D}}_D\ \gamma^D\ c \to \mathbf{Type} \\
\Delta : \mathsf{Params}\ \Gamma & \vdash \Delta^{\mathsf{S}}_f : (s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D)(\delta : \ulcorner \Delta^{\mathsf{A}}_F\ \gamma \urcorner) \to \ulcorner \Delta^{\mathsf{D}}_D\ \gamma^D\ \delta \urcorner \\
T : \mathsf{Ty}\ \Gamma\ \Delta & \vdash T^{\mathsf{S}}_f \ : (s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D)(x : \ulcorner T^{\mathsf{A}}_F\ \gamma\ \delta \urcorner) \to \ulcorner T^{\mathsf{D}}_D\ \gamma^D\ (\Delta^{\mathsf{S}}_f\ s\ \delta)\ x \urcorner \\
w : \mathsf{Wk}\ \Gamma\ \Omega & \vdash w^{\mathsf{S}}_f \ : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D \to \Omega^{\mathsf{S}}_f\ (w^{\mathsf{A}}_F\ \gamma)\ (w^{\mathsf{D}}_D\ \gamma^D) \\
c : \mathsf{CTm}\ \Gamma\ C & \vdash c^{\mathsf{S}}_f \ : (s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D) \to C^{\mathsf{S}}_f\ s\ (c^{\mathsf{A}}_F\ \gamma)\ (c^{\mathsf{D}}_D\ \gamma^D)
\end{array}$$

These operators are defined by mutual induction on the QIIT of specifications, together with equational properties about them that we compile as Lemma 10.2.1 and Lemma 10.2.2 below.

On a specification $\Gamma : \mathsf{Spec}$, the expression Γ^{S}_f computes the iterated product of meta-theoretic equations on f that are required by the section structure. We define it by induction on specification contexts.

$$\begin{array}{ll}
\diamond^{\mathsf{S}}_f\ \gamma\ \gamma^D & :\equiv \mathbb{1} \\
(\Gamma \rhd C)^{\mathsf{S}}_f\ (\gamma, c)\ (\gamma^D, c^D) & :\equiv \Sigma(s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D)(C^{\mathsf{S}}_f\ s\ c\ c^D)
\end{array}$$

For a constructor $C : \mathsf{Ctor}\ \Gamma$ and pair of algebras c, c^D, the equation generated by $C^S\ s\ c\ c^D$ is the statement that applying the functions in f to $c \cdot \delta$ — that is, the constructor algebra c applied to some arguments — is the same as applying f to each of the arguments δ first, and then applying the displayed constructor algebra c^D to the result. In other words, the section functions f commute with the constructor algebras:

$$(\mathsf{ctor}\ \Delta\ T)^{\mathsf{S}}_f\ s\ c\ c^D :\equiv \forall\ \delta \to T^{\mathsf{S}}_f\ s\ (c \cdot \delta) = c^D \cdot \delta \cdot \Delta^{\mathsf{S}}_f\ s\ \delta$$

Note that whether we need to apply $\pi_1\ f$ or $\pi_2\ f$ to $c \cdot \delta$ depends on the base type T, hence the helper function $T^{\mathsf{S}}_f\ s$ which just selects the correct component of f to apply according to T:

$$\begin{array}{ll}
(\mathsf{T}^1)^{\mathsf{S}}_f\ s\ x & :\equiv \pi_1\ f \cdot x \\
(\mathsf{T}^2\ t)^{\mathsf{S}}_f\ \{\delta\}\ s\ x & :\equiv \pi_2\ f \cdot t^{\mathsf{A}}\ \delta \cdot x
\end{array}$$

Moreover, in order to apply f to a list δ of arguments to the constructor, we define $\Delta^{\mathsf{S}}_f\ s$ which just recursively applies the function f to δ in the obvious recursive way.

$$\diamond^{\mathsf{S}}_f\ \gamma\ \gamma^D :\equiv \mathbb{1}$$
$$(\Gamma \triangleright C)^{\mathsf{S}}_f\ (\gamma, c)\ (\gamma^D, c^D) :\equiv$$
$$\quad \Sigma(s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D)(C^{\mathsf{S}}_f\ s\ c\ c^D)$$

$$(\mathsf{ctor}\ \Delta\ T)^{\mathsf{S}}_f\ s\ c\ c^D :\equiv$$
$$\quad \forall\,\delta\,.\,T^{\mathsf{S}}_f\ s\ (c \cdot \delta) = c^D \cdot \delta \cdot \Delta^{\mathsf{S}}_f\ s\ \delta$$

$$(\mathsf{ext}\ X)^{\mathsf{S}}_f\ \_\ \_ :\equiv *$$
$$(\pi\ A\ B)^{\mathsf{S}}_f\ s\ f :\equiv \boldsymbol{\lambda}\ (x : A)(B^{\mathsf{S}}_f\ s\ (f \cdot x))$$
$$(T[\sigma])^{\mathsf{S}}_f\ s\ x :\equiv T^{\mathsf{S}}_f\ s\ x$$
$$(T[w])^{\mathsf{S}}_f\ s\ x :\equiv T^{\mathsf{S}}_f\ (w^{\mathsf{S}}_f\ s)\ x$$
$$(\mathsf{T}^1)^{\mathsf{S}}_f\ s\ x :\equiv \pi_1\ f \cdot x$$
$$(\mathsf{T}^2\ t)^{\mathsf{S}}_f\ \{\delta\}\ s\ x :\equiv \pi_2\ f \cdot t^{\mathsf{A}}\,\delta \cdot x$$

$$\bullet^{\mathsf{S}}_f\ \_\ \_ :\equiv \mathsf{tt}$$
$$(\Delta \mathbin{\triangleright\!\triangleright} T)^{\mathsf{S}}_f\ s\ \langle \delta, x\rangle :\equiv \langle \Delta^{\mathsf{S}}_f\ s\ \delta, T^{\mathsf{S}}_f\ s\ x\rangle$$
$$(\Delta[w])^{\mathsf{S}}_f\ s :\equiv \Delta^{\mathsf{S}}_f\ (w^{\mathsf{S}}_f\ s)$$

$$\mathsf{id}^{\mathsf{S}}_f\ s :\equiv s$$
$$\mathsf{drop}^{\mathsf{S}}_f\ s :\equiv \pi_1\ s$$
$$(w_1 \circ w_2)^{\mathsf{S}}_f\ s :\equiv w_2{}^{\mathsf{S}}_f\ (w_1{}^{\mathsf{S}}_f\ s)$$

$$\mathsf{cvz}^{\mathsf{S}}_f\ s :\equiv \pi_2\ s$$
$$(\mathsf{cvs}\ c)^{\mathsf{S}}_f\ s :\equiv c^{\mathsf{S}}_f\ (\pi_1\ s)$$
$$(c[w])^{\mathsf{S}}_f\ s :\equiv c^{\mathsf{S}}_f\ (w^{\mathsf{S}}_f\ s)$$

Figure 10.2: IIT section operators

$$\begin{aligned}
\bullet^{\mathsf{S}}_f\ \_\ \_ &:\equiv \mathsf{tt} \\
(\Delta \mathbin{\triangleright\!\triangleright} T)^{\mathsf{S}}_f\ s\ \langle \delta, x\rangle &:\equiv \langle \Delta^{\mathsf{S}}_f\ s\ \delta, T^{\mathsf{S}}_f\ s\ x\rangle \\
(\Delta[w])^{\mathsf{S}}_f\ s &:\equiv \Delta^{\mathsf{S}}_f\ (w^{\mathsf{S}}_f\ s)
\end{aligned}$$

Note that to type-check the definition of $(\mathsf{T}^2\ t)^{\mathsf{S}}_f$ we rely on Lemma 10.2.1 proved later in this section.

On the remaining types $T : \mathsf{Ty}\ \Gamma\ \Delta$, the operator T^{S} just implements the action of the section functions on Ty-algebras. Again, to type-check the definition of $(T[\sigma])^{\mathsf{S}}$ we rely on Lemma 10.2.2 given below. We summarize all section operators in Figure 10.2.

All these section structure functions are defined simultaneously, mutually, by induction on the IIT of specifications, together with the following equational properties about the section operators themselves:

Lemma 10.2.1. For any $t : \mathsf{Tm}\,\Gamma\,\Delta\,X$, $f : \mathsf{Map}\ F\ D$, $s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D$ and $\delta : \ulcorner\Delta^{\mathsf{A}}_F\ \gamma\urcorner$,

$$X^{\mathsf{S}}_f\ s\ (t^{\mathsf{A}}_F\ \delta) = t^{\mathsf{D}}_D\ (\Delta^{\mathsf{S}}_f\ s\ \delta) \qquad \blacksquare$$

Proof. By induction on t. □

Lemma 10.2.2. For any $\sigma : \mathsf{Sub}\ \Delta\ \nabla$, $f : \mathsf{Map}\ F\ D$, $s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D$ and $\delta : \ulcorner \Delta^{\mathsf{A}}_F\ \gamma \urcorner$,

$$\nabla^{\mathsf{S}}_f\ s\ (\sigma^{\mathsf{A}}_F\ \delta) = \sigma^{\mathsf{D}}_D\ (\Delta^{\mathsf{S}}_f\ s\ \delta)$$ ■

Proof. By induction on σ. □

Remark 10.2.1. Since the section structure operator computes a product of identity types by recursion over specifications, it is straightforward to show that Γ^{S} constitutes a family of (weak) propositions. ■

Example 10.2.6. Let C, T, C^D, T^D carriers, and Con/Ty algebra structures $\theta : \Theta^{\mathsf{A}}_{(C,T)}$ and $\theta^D : \Theta^{\mathsf{D}}_{(C^D,T^D)}\ \theta$. Given functions $f : \mathsf{Map}\,(C,T)\,(C^D,T^D)$, we can easily verify that the expression $\Theta^{\mathsf{S}}_f\ \theta\ \theta^D$ computes to (a straightforwardly equivalent version of) the section structure shown in Example 10.2.5.

■

Definition 10.2.4. Given an IIT specification $\Gamma : \mathsf{Spec}$, a regular algebra $(F,\gamma) : \mathsf{Alg}\ \Gamma$, and a displayed algebra $(D,\gamma^D) : \mathsf{Alg}^D\ \Gamma\ (F,\gamma)$, a section between (F,γ) and (D,γ^D) is a pair of functions $f : \mathsf{Map}\ F\ D$, and a proof of $\Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D$. ■

Having defined what sections are, we can finally formalize what it means for an IIT algebra to support the intended induction principle.

Definition 10.2.5. Given an IIT specification $\Gamma : \mathsf{Spec}$, we say than an algebra $a : \mathsf{Alg}\ \Gamma$ is *inductive*, or *section inductive*, when for any displayed algebra $d : \mathsf{Alg}^D\ \Gamma\ a$ there exists a section between them. ■

Remark 10.2.2 (uniqueness)**.** *Section induction* is also considered in [ACD+18] in the context of QIITs. There, the authors establish an equivalence between section induction and initiality, thus showing that inductive algebras, similarly to initial algebras, can be characterized as being unique (up to isomorphism). They work in an extensional setting, in particular assuming function extensionality.

We will not concern ourselves with uniqueness properties of inductive IIT algebras in this work, as our goal is to establish the existence of *some* inductive IIT algebra for any given specification. Nevertheless, the question is interesting and certainly worth future investigation. In particular, we wonder if we can establish some uniqueness property of our notion of inductive algebras similarly to [ACD+18].

When discussing uniqueness of algebras, the intensional nature of our setting poses an extra challenge, particularly in deciding what we mean by equality and at what "level" (meta vs target) we mean it. On the one hand metatheoretic equality would be too strict, failing to relate anything beyond definitional identity. On the other hand the intensional target-level identity type is too weak to properly compare infinitary objects. We believe the appropriate way to relate IIT algebras would be via their embedding into the setoid model/SeTT, which provides a fitting, extensional form of equality.

■

We have formally defined what counts as an IIT in the target theory, for any given IIT specification. In the chapters that follow we will proceed to our main result, which is not only to establish that all specifiable IITs always exist — i.e. there always exists a corresponding inductive algebra — but that we have a systematic way to construct it.

Our aims for this part of the thesis can be summarized as the following theorem:

Theorem 10.2.1. Given any IIT specification $\Gamma : \mathsf{Spec}$, there exists an inductive algebra for it. That is,

1. There exists an algebra $a : \mathsf{Alg}\,\Gamma$
2. such that for any displayed algebra $d : \mathsf{Alg}^D\ \Gamma\ a$, there exists a section $(f, s) : \mathsf{Sect}\,\Gamma\,a\,d$. ■

We will tackle both points of this theorem in the sections that follow. Specifically, we will address the first point in Chapter 11, where we show how to construct an IIT algebra given any IIT specification, and the second point in Chapter 12, where we show that the algebra constructed in Chapter 11 is actually inductive.

Chapter 11

Constructing IIT algebras

In Chapter 10 we have formalized how to specify IITs, and defined what constitutes an IIT for a given specification. In this Chapter we tackle the next step, which is to show that we can in fact always define target-level types and terms encoding any specifiable IIT. This corresponds to the first point of Theorem 10.2.1.

In Section 8.1 we encoded the type formers and constructors of our running example, Con/Ty, in terms of type formers and constructors of so-called erased types and a well-formedness predicates. Technically, the encoding was as simple as pairing together erased constructors and their corresponding well-formedness proofs. We referred to this pairing as Σ-*construction*, after [vR19].

Although the Con/Ty example deals with a single concrete IIT, the process of deriving the erased types and well-formedness predicates, as well as their combination via the Σ-construction, wasn't specific to that example and can be systematically applied to any IIT specification. The aim of this chapter is demonstrate this claim. That is, to give a general account of erased types, well-formedness predicates, and the Σ-construction, that applies in general to arbitrary specifications.

Similarly to our general account of IITs, we tackle erased and predicate types by framing them as certain kinds of algebras. We define families of *erased algebras* and *predicate algebras* indexed over IIT specifications, then restrict our attention to those particular algebras that exhibit some property of interest, like section induction or the presence of suitable inversion principles.

We then define the Σ-construction as a generic *operation on algebras*; that is, for any specification we show how to construct an IIT algebra given any pair of an erased algebra and predicate algebra. A direct consequence of the Σ-construction is that if erased and predicate algebras always exist, then IIT algebras do as well.

We first deal with erased types and their algebras in Section 11.1, followed by an account of well-formedness predicates in Section 11.2. Both sections conclude with arguments for the existence of erased and predicate algebras, respectively, for any given specification. We finally wrap things up with the Σ-construction in

Section 11.3.1.

11.1 Erased types

Every (two-sorted) IIT specification automatically induces the specification of two mutually-defined inductive types. This process of *erasure* works by simply stripping the type and term constructors of the original IIT of all its indices, to produce what we call *erased types*.

Erasure is a systematic process that can be carried out irrespective of the particular IIT. We now give a general account of erasure for arbitrary specifications. We first assign a notion of *erased algebra* to IIT specifications, and complement it with displayed algebras and sections. We then define "erased types" to be the section-inductive algebras.

11.1.1 Algebras of erased types

To get an idea of the structure of erased algebras, let us revisit our running example, the Con/Ty IIT, both as the familiar metatheoretic construction and as a target level one:

Example 11.1.1 (metatheory). Recall the constructors for the erased types $\mathsf{Con}_0, \mathsf{Ty}_0$ shown in the example of Section 8.1:

$$
\begin{aligned}
&\bullet_0 &&: \mathsf{Con}_0 \\
&\_ \triangleright_0 \_ &&: \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Con}_0 \\
&\iota_0 &&: \mathsf{Con}_0 \to \mathsf{Ty}_0 \\
&\hat{\pi}_0 &&: \mathsf{Con}_0 \to (\mathbb{N} \to \mathsf{Ty}_0) \to \mathsf{Ty}_0 \\
&\bar{\pi}_0 &&: \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0
\end{aligned}
$$

As a metatheoretic construction, an algebra of $\mathsf{Con}_0, \mathsf{Ty}_0$ is essentially like the structure above, with $\mathsf{Con}_0, \mathsf{Ty}_0$ replaced by arbitrary carrier types. In other words, algebras are a pair of types $C_0 : \mathbf{Type}, T_0 : \mathbf{Type}$ plus an element of the following product

$$
\begin{aligned}
&C_0 \times (C_0 \to T_0 \to C_0) \times (C_0 \to T_0) \\
&\quad \times (C_0 \to (\mathbb{N} \to T_0) \to T_0) \times (C_0 \to T_0 \to T_0 \to T_0)
\end{aligned}
$$

■

Example 11.1.2 (target theory)**.** As a target-level structure, an algebra of $\mathsf{Con}_0/\mathsf{Ty}_0$ is similarly a pair of small types $C_0 : \mathsf{SType}, T_0 : \mathsf{SType}$ plus a term of the product type

$$\begin{aligned}&\ulcorner C_0 \urcorner \times \ulcorner C_0 \Rightarrow T_0 \Rightarrow C_0 \urcorner \times \ulcorner C_0 \Rightarrow T_0 \urcorner \\ &\quad \times \ulcorner C_0 \Rightarrow (\mathsf{Nat} \Rightarrow T_0) \Rightarrow T_0 \urcorner \times \ulcorner C_0 \Rightarrow T_0 \Rightarrow T_0 \Rightarrow T_0 \urcorner\end{aligned}$$

It is immediate to see that the structure above is a direct target-level translation of the one shown in Example 11.1.1. ■

We now define how to calculate the structure of erased algebras for an arbitrary specification. Since we are only considering IITs of fixed sorts, the type of carriers of an erased algebra is always just a pair of target-level small types $\mathsf{Ca}_0 :\equiv \mathsf{SType} \times \mathsf{SType}$. Given $\Gamma : \mathsf{Spec}$ and carriers $F_0 : \mathsf{Ca}_0$, we define the corresponding erased algebra structure $\Gamma^{\mathsf{E}}_{F_0}$, together with dedicated erased algebra operators for each specification type.

$$\begin{array}{lll}\Gamma : \mathsf{Spec} & \vdash & \Gamma^{\mathsf{E}}_{F_0} : \mathbf{Type} \\ C : \mathsf{Ctor}\,\Gamma & \vdash & C^{\mathsf{E}}_{F_0} : \mathsf{SType} \\ \Delta : \mathsf{Params}\,\Gamma & \vdash & \Delta^{\mathsf{E}}_{F_0} : \mathsf{SType} \\ T : \mathsf{Ty}\,\Gamma\,\Delta & \vdash & T^{\mathsf{E}}_{F_0} : \mathsf{SType}\end{array}$$

Remark 11.1.1. The erased algebra operators refer to each other in a non-circular way in their definition, and not at all in their types. They are, therefore, not mutually defined, unlike the IIT algebra operators of the previous Chapter. ■

$\Gamma^{\mathsf{E}}_{F_0}$ should be an iterated product of algebra operations similar to Figure 11.1.2, but specific to Γ and F_0. Just like the operators Γ^{A}_{F} and Γ^{D}_{D}, we define $\Gamma^{\mathsf{E}}_{F_0}$ by recursion on specifications:

$$\begin{array}{ll}\diamond^{\mathsf{E}}_{F_0} & :\equiv \mathbb{1} \\ (\Gamma \rhd C)^{E}_{F_0} & :\equiv \Gamma^{\mathsf{E}}_{F_0} \times \ulcorner C^{\mathsf{E}}_{F_0} \urcorner\end{array}$$

The erased algebra for a constructor C is a function type from a tuple of parameters to one of the two erased carrier types:

$$(\mathsf{ctor}\,\Delta\,T)^{\mathsf{E}}_{F_0} :\equiv \Delta^{\mathsf{E}}_{F_0} \Rightarrow T^{\mathsf{E}}_{F_0}$$

Erased parameter algebras are given by a straightforward recursion:

$$\begin{aligned}
\bullet^{\mathsf{E}}_{F_0} &:\equiv \mathbf{1}\\
(\Delta \rhd\!\rhd T)^{\mathsf{E}}_{F_0} &:\equiv \Delta^{\mathsf{E}}_{F_0} \times T^{\mathsf{E}}_{F_0}\\
(\Delta[w])^{\mathsf{E}}_{F_0} &:\equiv \Delta^{\mathsf{E}}_{F_0}
\end{aligned}$$

Note that $(\Delta[w])^{\mathsf{E}}$ simply ignores the weakening, reflecting the fact that erased types (and therefore their algebras) arise from forgetting all type dependency information. The same holds for inner type and base type algebras:

$$\begin{aligned}
(\mathsf{ext}\ X)^{\mathsf{E}}_{F_0} &:\equiv X & (\mathsf{T}^1)^{\mathsf{E}}_{F_0} &:\equiv \pi_1\ F_0\\
(\pi\ A\ B)^{\mathsf{E}}_{F_0} &:\equiv A \Rightarrow B^{\mathsf{E}}_{F_0} & (\mathsf{T}^2\ t)^{\mathsf{E}}_{F_0} &:\equiv \pi_2\ F_0\\
(T[w])^{\mathsf{E}}_{F_0} &:\equiv T^{\mathsf{E}}_{F_0}\\
(T[\sigma])^{\mathsf{E}}_{F_0} &:\equiv T^{\mathsf{E}}_{F_0}
\end{aligned}$$

We have additional algebra operators for (constructor) terms and weakenings/substitutions:

$$\begin{aligned}
t : \mathsf{Tm}\ \Gamma\ \Delta\ T &\vdash t^{\mathsf{E}}_{F_0} : \Gamma^{\mathsf{E}}_{F_0} \to \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner \to \ulcorner T^{\mathsf{E}}_{F_0} \urcorner\\
c : \mathsf{CTm}\ \Gamma\ C &\vdash c^{\mathsf{E}}_{F_0} : \Gamma^{\mathsf{E}}_{F_0} \to \ulcorner C^{\mathsf{E}}_{F_0} \urcorner\\
w : \mathsf{Wk}\ \Gamma\ \Omega &\vdash w^{\mathsf{E}}_{F_0} : \Gamma^{\mathsf{E}}_{F_0} \to \Omega^{\mathsf{E}}_{F_0}\\
\sigma : \mathsf{Sub}\ \Delta\ \nabla &\vdash \sigma^{\mathsf{E}}_{F_0} : \Gamma^{\mathsf{E}}_{F_0} \to \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner \to \ulcorner \nabla^{\mathsf{E}}_{F_0} \urcorner
\end{aligned}$$

As usual, the operators on weakenings and substitutions lift the action of these operations to context and parameter algebras.

$$\begin{aligned}
\mathsf{id}^{\mathsf{E}}\ \gamma_0\ \delta_0 &:\equiv \delta_0 & \mathsf{id}^{\mathsf{E}}\ \gamma_0 &:\equiv \gamma_0\\
(\mathsf{ext}\ \sigma\ t)^{\mathsf{E}}\ \gamma_0\ \delta_0 &:\equiv \langle \sigma^{\mathsf{E}}\ \gamma_0\ \delta_0, t^{\mathsf{E}}\ \gamma_0\ \delta_0 \rangle & \mathsf{drop}^{\mathsf{E}}\ \gamma_0 &:\equiv \pi_1\ \gamma_0\\
\mathsf{drop}^{\mathsf{E}}\ \gamma_0\ \delta_0 &:\equiv \mathsf{fst}\ \delta_0 & (w_1 \circ w_2)^{\mathsf{E}} \gamma_0 &:\equiv w_2{}^{\mathsf{E}}\ (w_1{}^{\mathsf{E}}\ \gamma_0)\\
(\sigma \circ \tau)^{\mathsf{E}}\ \gamma_0\ \delta_0 &:\equiv \tau^{\mathsf{E}}\ \gamma_0\ (\sigma^{\mathsf{E}}\ \gamma_0\ \delta_0)\\
(\sigma[w])^{\mathsf{E}}\ \gamma_0\ \delta_0 &:\equiv \sigma^{\mathsf{E}}\ (w^{\mathsf{E}} \gamma_0)\ \delta_0
\end{aligned}$$

Moreover, the algebra operators on terms express indexing into algebras of contexts.

$$
\begin{array}{ll}
\mathsf{vz}^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv \mathsf{snd}\ \delta_0 \\
(\mathsf{vs}\ t)^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv t^{\mathsf{E}}\gamma_0\ (\mathsf{fst}\ \delta_0) \\
(\mathsf{ext}\ A\ x)^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv x \\
(\mathsf{capp}\ c\ \sigma)^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv c^{\mathsf{E}}\ \gamma_0 \cdot \sigma^{\mathsf{E}}\ \gamma_0\ \delta_0 \\
(\mathsf{ap}\ t)^{\mathsf{E}}\ \gamma_0\ \langle \delta_0, x\rangle & :\equiv t^{\mathsf{E}}\ \gamma_0\ \delta_0 \cdot x \\
(\mathsf{lm}\ t)^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv \boldsymbol{\lambda} x\,.\,t^{\mathsf{E}}\ \gamma_0\ \langle \delta_0, x\rangle \\
(t[\sigma])^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv t^{\mathsf{E}}\ \gamma_0\ (\sigma^{\mathsf{E}}\ \gamma_0\ \delta_0) \\
(t[w])^{\mathsf{E}}\ \gamma_0\ \delta_0 & :\equiv t^{\mathsf{E}}\ (w^{\mathsf{E}}\ \gamma_0)\ \delta_0
\end{array}
\qquad
\begin{array}{ll}
\mathsf{cvz}^{\mathsf{E}}\ \gamma_0 & :\equiv \pi_2\ \gamma_0 \\
(\mathsf{cvs}\ c)^{\mathsf{E}}\ \gamma_0 & :\equiv c^{\mathsf{E}}\ (\pi_1\ \gamma_0) \\
(c[w])^{\mathsf{E}}\ \gamma_0 & :\equiv c^{\mathsf{E}}\ (w^{\mathsf{E}}\ \gamma_0)
\end{array}
$$

Example 11.1.3. Given carriers C_0, T_0 from Example 11.1.2, we can easily verify that the expression $\Theta^{\mathsf{E}}_{(C_0,T_0)}$ computes to the erased algebra structure defined in that example. ■

We wrap things up with a definition of erased algebras.

Definition 11.1.1. Given an IIT specification $\Gamma : \mathsf{Spec}$, an erased algebra $\mathsf{Alg}_0\ \Gamma$ is a pair of carrier types $F_0 : \mathsf{Ca}_0$, and a term $\gamma_0 : \Gamma^{\mathsf{E}}_{F_0}$. ■

Erased types, and therefore erased algebras, constitute the building blocks of our encoding of IITs in terms of inductive families. However, not all erased algebras are suitable for this role, as their induction principle might not be strong enough. We are specifically interested in those algebras that are *section-inductive*.

In accordance to this requirement, the next sections will tackle the definition of *displayed erased algebras* and *erased sections*, from which we derive the notion of section-inductive erased algebras.

11.1.2 Displayed algebras of erased types

Example 11.1.4 (metatheory)**.** Recall the $\mathsf{Con}_0/\mathsf{Ty}_0$ erased algebra from Example 11.1.1, given by types C_0, T_0 : **Type** and terms $\bullet_0, \triangleright_0, \iota_0, \hat{\pi}_0, \bar{\pi}_0$. A metatheoretic displayed algebra over it is a pair of indexed types, and terms indexed over the erased constructors:

$$
\begin{array}{ll}
C_0^D & : C_0 \to \mathbf{Type} \\
T_0^D & : T_0 \to \mathbf{Type} \\
\bullet_0^D & : C_0^D\ \bullet_0
\end{array}
$$

$$\begin{array}{ll}
\_ \triangleright_0^D \_ & : C_0^D\ \Gamma_0 \to T_0\ A_0 \to C_0^D\ (\Gamma_0 \triangleright_0 A_0) \\
\iota_0^D & : C_0^D\ \Gamma_0 \to T_0^D\ (\iota_0\ \Gamma_0) \\
\hat{\pi}_0^D & : C_0^D\ \Gamma_0 \to ((n : \mathbb{N}) \to T_0^D\ (F_0\ n)) \to T_0^D\ (\hat{\pi}_0\ \Gamma_0\ F_0) \\
\bar{\pi}_0^D & : C_0^D\ \Gamma_0 \to T_0^D\ A_0 \to T_0^D\ B_0 \to T_0^D\ (\bar{\pi}_0\ \Gamma_0\ A_0\ B_0)
\end{array}$$

■

Example 11.1.5 (target theory)**.** Consider an target-level Con/Ty erased algebra, as shown in Example 11.1.2: this is given by types $C_0 : \mathsf{SType}, T_0 : \mathsf{SType}$, and terms $\bullet_0, \triangleright_0, \iota_0, \hat{\pi}_0, \bar{\pi}_0$. A *displayed erased algebra* over this algebra is then given by indexed carrier types

$$C_0^D : \ulcorner C_0 \urcorner \to \mathsf{Type}, \qquad T_0^D : \ulcorner T_0 \urcorner \to \mathsf{Type}$$

and terms for each constructor:

$$\begin{array}{l}
\bullet_0^D : \ulcorner C_0^D\ \bullet_0 \urcorner \\
\triangleright_0^D : \ulcorner \mathbf{\Pi}\ (\delta_0 : C_0)(t_0 : T_0)(C_0^D\ c_0 \times T_0^D\ t_0 \Rightarrow C_0^D\ (\triangleright_0 \cdot \delta_0)) \urcorner \\
\iota_0^D : \ulcorner \mathbf{\Pi}(c_0 : C_0)(C_0^D\ c_0 \Rightarrow T_0^D\ (\iota_0 \cdot c_0)) \urcorner \\
\hat{\pi}_0^D : \ulcorner \mathbf{\Pi}\ (c_0 : C_0)(f_0 : \mathsf{Nat} \Rightarrow T_0) \\
\qquad\qquad (C_0^D\ c_0 \Rightarrow \mathbf{\Pi}\ (n : \mathsf{Nat})\ (T_0^D\ (f_0 \cdot n)) \Rightarrow T_0^D\ (\hat{\pi}_0 \cdot c_0 \cdot f_0)) \urcorner \\
\bar{\pi}_0^D : \ulcorner \mathbf{\Pi}\ (c_0 : C_0)(a_0 : T_0)(b_0 : T_0) \\
\qquad\qquad (C_0^D\ c_0 \Rightarrow T_0^D\ a_0 \Rightarrow T_0^D\ b_0 \Rightarrow T_0^D\ (\bar{\pi}_0 \cdot c_0 \cdot a_0 \cdot b_0)) \urcorner
\end{array}$$

One can see that the structure above is a direct target-level translation of the structure in Example 11.1.4. ■

We now want to assign a notion of displayed algebra to any specification. We thus define algebra operators that calculate the carrier types and the term structure of a displayed algebra given an arbitrary IIT specification and erased algebra for it.

We begin by defining displayed carriers as a type indexed by regular erased carriers:

$$\begin{array}{l}
\mathsf{Ca}_0^D : \mathsf{Ca}_0 \to \mathbf{Type} \\
\mathsf{Ca}_0^D\ (A_0, B_0) :\equiv (\ulcorner A_0 \urcorner \to \mathsf{Type}) \times (\ulcorner B_0 \urcorner \to \mathsf{Type})
\end{array}$$

Note that we allow the displayed carriers to be families of large types. This is because we will need to do large elimination our of erased algebras later in our development.

We now define displayed algebra operators $\_^{\mathsf{ED}}$ on specifications, so that $\Gamma^{\mathsf{ED}}_{D_0}\ \gamma_0$ calculates the displayed algebra structure over the carriers, as an iterated product of algebra operations like in Figure 11.1.5. As with all the previous algebra definitions, we define an algebra operator for each sort of the specification QIIT. The type of each of these operators is indexed by the carrier types, as well as a regular erased algebra of the corresponding specification sort:

$$\begin{array}{lll}
\Gamma : \mathsf{Spec} & \vdash \Gamma^{\mathsf{ED}}_{D_0} & : \Gamma^{\mathsf{E}}_{F_0} \to \mathbf{Type} \\
C : \mathsf{Ctor}\ \Gamma & \vdash C^{\mathsf{ED}}_{D_0} & : \ulcorner C^{\mathsf{E}}_{F_0} \urcorner \to \mathsf{Type} \\
\Delta : \mathsf{Params}\ \Gamma & \vdash \Delta^{\mathsf{ED}}_{D_0} & : \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner \to \mathsf{Type} \\
T : \mathsf{Ty}\ \Gamma\ \Delta & \vdash X^{\mathsf{ED}}_{D_0} & : \ulcorner T^{\mathsf{E}}_{F_0} \urcorner \to \mathsf{Type}
\end{array}$$

In addition, we have operators for (constructor) terms and weakenings/substitutions:

$$\begin{array}{lll}
c : \mathsf{CTm}\ \Gamma\ C & \vdash c^{\mathsf{ED}}_{D_0} & : \Gamma^{\mathsf{ED}}_{D_0}\ \gamma_0 \to \ulcorner C^{\mathsf{ED}}_{D_0} \urcorner \\
t : \mathsf{Tm}\ \Gamma\ \Delta\ T & \vdash t^{\mathsf{ED}}_{D_0} & : \Gamma^{\mathsf{ED}}_{D_0}\ \gamma_0 \to \ulcorner \Delta^{\mathsf{ED}}_{D_0}\ \delta_0 \urcorner \to \ulcorner T^{\mathsf{ED}}_{D_0}\ (t^{\mathsf{E}} \gamma_0\ \delta_0) \urcorner \\
w : \mathsf{Wk}\ \Omega\ \Gamma & \vdash w^{\mathsf{ED}}_{D_0} & : \Omega^{\mathsf{ED}}_{D_0}\ \gamma_0 \to \Gamma^{\mathsf{ED}}_{D_0}\ (w^{\mathsf{E}} \gamma_0) \\
\sigma : \mathsf{Sub}\ \Delta\ \nabla & \vdash \sigma^{\mathsf{ED}}_{D_0} & :\to \Omega^{\mathsf{ED}}_{D_0}\ \gamma_0 \to \ulcorner \Delta^{\mathsf{ED}}_{D_0}\ \delta_0 \urcorner \to \ulcorner \nabla^{\mathsf{ED}}_{D_0}\ (\sigma^{\mathsf{E}} \gamma_0\ \delta_0) \urcorner
\end{array}$$

Similarly to displayed algebras for IITs (Section 10.2.1), displayed erased algebras are essentially an indexed generalization of regular erased algebras, or equivalently, an erased version of displayed IIT algebras. Most of these definitions require little discussion. One notable difference with the previous treatment of algebra operators is that for specification contexts $\Gamma : \mathsf{Spec}$, we define the type Γ^{ED} without recursion on the specification, but instead as the following equivalent type:

$$\Gamma^{\mathsf{ED}}_{D_0}\ \gamma_0 :\equiv \forall \{C\} (c : \mathsf{CTm}\ \Gamma\ C) \to \ulcorner C^{\mathsf{ED}}_{D_0}\ (c^{\mathsf{E}} \gamma_0) \urcorner$$

This is similar to defining the type of tuples of natural numbers of size n as $\mathsf{Fin}\ n \to \mathbb{N}$, instead of the equivalent $\mathsf{Vec}\ \mathbb{N}\ n$, using $\mathsf{Fin}\ n$ as an index. Here we are using $\mathsf{CTm}\,\Gamma\,C$ as an index into Γ. This direct, non-recursive definition of Γ^{ED} will make a few constructions easier later on. Still, even though the elements of Γ^{ED} are functions, we will often write them in the more convenient tuple notation $(c_0, c_1, ..., c_n)$.

Remark 11.1.2. Such compact definition of Γ^{ED} is possible because the erased algebra operators are not recursive-recursive, unlike IIT algebra operators. An equivalent expression for IIT algebras would have been:

$$\Gamma^{\mathsf{ED}}_{D_0}\ \gamma_0 :\equiv \forall\{C\}(c : \mathsf{CTm}\ \Gamma\ C) \to \ulcorner C^{\mathsf{ED}}_{D_0}\ (c^{\mathsf{E}} \gamma_0)\urcorner$$
$$(\mathsf{ctor}\ \Delta\ T)^{\mathsf{ED}}\ c_0 :\equiv \mathbf{\Pi}(\delta_0 : \Delta^{\mathsf{E}})\ (\Delta^{\mathsf{ED}}\ \delta_0 \Rightarrow T^{\mathsf{ED}}\ (c_0 \cdot \delta_0))$$

$$\begin{aligned}
&\bullet^{\mathsf{ED}}\ \delta_0 :\equiv \mathbf{1}\\
&(\Delta \rhd\!\rhd T)^{\mathsf{ED}} \langle \delta_0, x_0\rangle :\equiv \Delta^{\mathsf{ED}}\ \delta_0 \times T^{\mathsf{ED}}\ x_0\\
&(\Delta[w])^{\mathsf{ED}} \delta_0 :\equiv \Delta^{\mathsf{ED}} \delta_0
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{ext}\ \_)^{\mathsf{ED}}_{\_} :\equiv \mathbf{1}\\
&(\pi\ A\ B)^{\mathsf{ED}} f_0 :\equiv \mathbf{\Pi}(a : A)(B^{\mathsf{ED}}(f_0 \cdot a))\\
&(T[\sigma])^{\mathsf{ED}} x_0 :\equiv T^{\mathsf{ED}} x_0\\
&(T[w])^{\mathsf{ED}} x_0 :\equiv T^{\mathsf{ED}} x_0\\
&(\mathsf{T}^1)^{\mathsf{ED}}_{D_0}\ a_0 :\equiv \pi_1\ D_0\ a_0\\
&(\mathsf{T}^2\ \_)^{\mathsf{ED}}_{D_0}\ b_0 :\equiv \pi_2\ D_0\ b_0
\end{aligned}$$

$$\begin{aligned}
&\mathsf{id}^{\mathsf{ED}} \gamma_0^D :\equiv \gamma_0^D\\
&\mathsf{drop}^{\mathsf{ED}} \gamma_0^D :\equiv \pi_1\ (\gamma_0^D)\\
&(w_1 \circ w_2)^{\mathsf{ED}} \gamma_0^D :\equiv w_2^{\mathsf{ED}}(w_1^{\mathsf{ED}} \gamma_0^D)
\end{aligned}$$

$$c^{\mathsf{ED}} \gamma_0^D :\equiv \gamma_0^D\ c$$

$$\begin{aligned}
&\mathsf{id}^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv \delta_0\\
&(\mathsf{ext}\ \sigma\ t)^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv\\
&\quad \langle \sigma^{\mathsf{ED}} \gamma_0^D\ \delta_0^D, t^{\mathsf{ED}} \gamma_0^D\ \delta_0^D\rangle\\
&\mathsf{drop}^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv \mathsf{fst}\ \delta_0^D\\
&(\sigma \circ \tau)^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv \tau^{\mathsf{ED}} \gamma_0^D\ (\sigma^{\mathsf{ED}} \gamma_0^D\ \delta_0^D)\\
&(\sigma[w])^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv \sigma^{\mathsf{ED}}(w^{\mathsf{ED}} \gamma_0^D)\ \delta_0^D
\end{aligned}$$

$$\begin{aligned}
&\mathsf{vz}^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv \mathsf{snd}\ \delta_0^D\\
&(\mathsf{vs}\ t)^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv t^{\mathsf{ED}} \gamma_0^D\ (\mathsf{fst}\ \delta_0^D)\\
&(\mathsf{ext}\ A\ x)^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv x\\
&(\mathsf{capp}\ c\ \sigma)^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv c^{\mathsf{ED}} \gamma_0^D \cdot {}_{\_} \cdot \sigma^{\mathsf{ED}} \gamma_0^D\ \delta_0^D\\
&(\mathsf{ap}\ t)^{\mathsf{ED}} \gamma_0^D\ \{\delta_0\}\ \delta_0^D :\equiv\\
&\quad (t^{\mathsf{ED}} \gamma_0^D\ (\mathsf{fst}\ \delta_0^D)) \cdot (\mathsf{snd}\ \delta_0)\\
&(\mathsf{lm}\ t)^{\mathsf{ED}} \gamma_0^D\ \{\delta_0\}\ \delta_0^D :\equiv\\
&\quad \boldsymbol{\lambda} x.t^{\mathsf{ED}} \gamma_0^D\ \{\langle \delta_0, x\rangle\}\ \langle \delta_0^D, *\rangle\\
&(t[\sigma])^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv t^{\mathsf{ED}} \gamma_0^D\ (\sigma^{\mathsf{ED}} \gamma_0^D\ \delta_0^D)\\
&(t[w])^{\mathsf{ED}} \gamma_0^D\ \delta_0^D :\equiv t^{\mathsf{ED}}(w^{\mathsf{ED}} \gamma_0^D)\ \delta_0^D
\end{aligned}$$

Figure 11.1: Displayed erased algebra operators

$$\Gamma^{\mathsf{A}}_F :\equiv \forall\{C\}(c : \mathsf{CTm}\ \Gamma\ C) \to \ulcorner C^{\mathsf{A}}_F\ (c^{\mathsf{A}}\ \gamma)\urcorner$$

This would not work, because c^{A} takes a structure $\gamma : \Gamma^{\mathsf{A}}_F$ as argument, but Γ^{A}_F is precisely what is being defined. ■

Because of how Γ^{ED} is defined, constructor term algebra operators c^{ED} on a context term $c : \mathsf{CTm}\ \Gamma\ C$ simply reduce to function application:

$$c^{\mathsf{ED}} \gamma_0^D :\equiv \gamma_0^D\ c$$

We summarize all the displayed erased algebra operators in Figure 11.1.

Example 11.1.6. Let $(C_0, T_0, \theta_0) : \mathsf{Alg}_0\, \Theta$ be a $\mathsf{Con}_0/\mathsf{Ty}_0$ erased algebra, and let $(C_0^D, T_0^D) : \mathsf{Ca}_0^D\, (C_0, T_0)$ be displayed carriers. We can easily verify that the type of $\theta_0^D : \Theta^{\mathsf{ED}}_{(C_0^D, T_0^D)}\, \theta_0$ computes to (an equivalent version of) the structure shown in Example 11.1.5. ■

We finally wrap up with a formal definition:

Definition 11.1.2. Given $\Gamma : \mathsf{Spec}$ and $(F_0, \gamma_0) : \mathsf{Alg}_0\ \Gamma$, a displayed algebra $\mathsf{Alg}_0^D\ \Gamma\ (F_0, \gamma_0)$ over (F_0, γ) is a pair of carrier types $D_0 : \mathsf{Ca}_0^D\ F_0$, and a term of type $\Gamma^{\mathsf{ED}}_{D_0}\ \gamma_0$. ■

11.1.3 Sections of erased algebras

We now relate erased algebras and their displayed version by the notion of *erased sections*. Like sections between IIT algebras, a section between erased algebras is a dependent mapping between the carrier types of the algebras that respects the algebra structure arising from the constructors. We will use the notion of erased section to imbue erased algebras with the necessary induction principles that are required by the general reduction proof.

Example 11.1.7 (metatheory)**.** Consider once again the erased types $\mathsf{Con}_0/\mathsf{Ty}_0$. Given a metatheoretical erased algebra $(C_0, T_0, \bullet_0, \rhd_0, \iota_0, \bar{\pi}_0, \hat{\pi}_0)$ and displayed algebra $(C_0, T_0, \bullet_0^D, \rhd_0^D, \iota_0^D, \bar{\pi}_0^D, \hat{\pi}_0^D)$ over it, a section between the two is a pair of functions

$$\begin{aligned}
&f_C : (c_0 : C_0) \to C_0^D\ c_0\\
&f_T : (t_0 : T_0) \to T_0^D\ t_0
\end{aligned}$$

respecting the algebraic structure of the constructors, as witnessed by the following equations

$$\begin{aligned}
&f_C\ \bullet_0 &&= \bullet_0^D\\
&f_C\ (c_0 \rhd_0 t_0) &&= (f_C\ c_0) \rhd_0^D (f_T\ t_0)\\
&f_T\ (\iota_0\ c_0) &&= \iota_0^D\ (f_C\ c_0)\\
&f_T\ (\hat{\pi}_0\ c_0\ f_0) &&= \hat{\pi}_0^D\ (f_C\ c_0)\ (\lambda n. f_T\ (f_0\ n))\\
&f_T\ (\bar{\pi}_0\ c_0\ a_0\ b_0) &&= \bar{\pi}_0^D\ (f_C\ c_0)\ (f_T\ a_0)\ (f_T\ b_0)
\end{aligned}$$

■

Example 11.1.8 (target theory)**.** Consider now the same erased types $\mathsf{Con}_0/\mathsf{Ty}_0$ in the setting of the target theory. Let $((C_0, T_0), (\bullet_0, \rhd_0, \iota_0, \hat{\pi}_0, \bar{\pi}_0))$ be an erased algebra and $((C_0^D, T_0^D), (\bullet_0^D, \triangleright_0^D, \iota_0^D, \hat{\pi}_0^D, \bar{\pi}_0^D))$ an erased displayed algebra over it. A section between them is a pair of function terms

$$\begin{aligned} &f_C : \ulcorner \mathbf{\Pi}\ C_0\ C_0^D \urcorner \\ &f_T : \ulcorner \mathbf{\Pi}\ T_0\ T_0^D \urcorner \end{aligned}$$

Such that the following equations hold:

$$\begin{aligned} &f_C \cdot \bullet_0 = \bullet_0^D \\ &f_C \cdot (\triangleright_0 \cdot c_0 \cdot t_0) = \triangleright_0^D \cdot (f_C \cdot c_0) \cdot (f_T \cdot t_0) \\ &f_T \cdot (\iota_0 \cdot c_0) = \iota_0^D \cdot (f_C \cdot c_0) \\ &f_T\ (\hat{\pi}_0 \cdot c_0 \cdot f_0) = \hat{\pi}_0^D \cdot (f_C \cdot c_0) \cdot (\boldsymbol{\lambda} n \,.\, f_T \cdot (f_0 \cdot n)) \\ &f_T\ (\bar{\pi}_0 \cdot c_0 \cdot a_0 \cdot b_0) = \bar{\pi}_0^D \cdot (f_C \cdot c_0) \cdot (f_T \cdot a_0) \cdot (f_T \cdot b_0) \end{aligned}$$

■

We shall now define, given an arbitrary IIT specification Γ and erased algebras $a : \mathsf{Alg}_0\,\Gamma, d : \mathsf{Alg}_0^D\,\Gamma\,a$, how to calculate the structure of a section between them. This will consist of the type of maps between the respective carriers, as well as an iterated product of equations that these maps must satisfy.

The functional part of a section is given by the operator Map_0, which is a type family indexed over the carriers of the algebras related by the section:

$$\begin{aligned} &\mathsf{Map}_0 : (F_0 : \mathsf{Ca}_0) \to \mathsf{Ca}_0^D\ F_0 \to \mathbf{Type} \\ &\mathsf{Map}_0(A_0, B_0)(A_0^D, B_0^D) :\equiv \ulcorner \mathbf{\Pi}\ A\ A_0^D \urcorner \times \ulcorner \mathbf{\Pi}\ B_0\ B_0^D \urcorner \end{aligned}$$

We then define erased section operators that calculate the structure of a section for arbitrary specifications. As usual, we define a section operator for each sort of the specification datatype.

For carriers F_0, D_0 and $f_0 : \mathsf{Map}_0\ F_0\ D_0$, we define:

$$\begin{aligned} &\Gamma : \mathsf{Spec} &&\vdash \Gamma_{f_0}^{\mathsf{ES}} : (\gamma_0 : \Gamma_{F_0}^{\mathsf{E}}) \to \ulcorner \Gamma_{D_0}^{\mathsf{ED}}\ \gamma_0 \urcorner \to \mathbf{Type} \\ &C : \mathsf{Ctor}\,\Gamma &&\vdash C_{f_0}^{\mathsf{ES}} : (c_0 : C_{F_0}^{\mathsf{E}}) \to \ulcorner C_{D_0}^{\mathsf{ED}}\ c_0 \urcorner \to \mathbf{Type} \\ &\Delta : \mathsf{Params}\,\Gamma &&\vdash \Delta_{f_0}^{\mathsf{ES}} : (\delta_0 : \Delta_{F_0}^{\mathsf{E}}) \to \ulcorner \Delta_{D_0}^{\mathsf{ED}}\ \delta_0 \urcorner \\ &T : \mathsf{Ty}\,\Gamma\,\Delta &&\vdash T_{f_0}^{\mathsf{ES}} : (x_0 : T_{F_0}^{\mathsf{E}}) \to \ulcorner T_{D_0}^{\mathsf{ED}}\ x_0 \urcorner \\ &w : \mathsf{Wk}\,\Gamma\,\Omega &&\vdash w_{f_0}^{\mathsf{ES}} : \Gamma_{f_0}^{\mathsf{ES}}\ \gamma_0\ \gamma_0^D \to \Omega_{f_0}^{\mathsf{ES}}\ (w_{F_0}^{\mathsf{E}}\ \gamma_0)\ (w_{D_0}^{\mathsf{ED}}\ \gamma_0^D) \end{aligned}$$

$$\Gamma^{\mathsf{ES}}_{f_0}\ \gamma_0\ \gamma_0^D :\equiv \forall\{C\}(c : \mathsf{CTm}\ \Gamma\ C) \to C^{\mathsf{ES}}_{f_0}\ (C^{\mathsf{E}}\gamma_0)\ (C^{\mathsf{ED}}\gamma_0^D)$$
$$(\mathsf{ctor}\ \Delta\ B)^{\mathsf{ES}}_{f_0}\ c_0\ c_0^D :\equiv \forall\ \delta_0 \to B^{\mathsf{ES}}_{f_0}\ (c_0 \cdot \delta_0) = c_0^D \cdot \delta_0 \cdot (\Delta^{\mathsf{ES}}_{f_0}\ \delta_0)$$

$$\begin{array}{llll}
\bullet^{\mathsf{ES}}\ \delta_0 & :\equiv \mathbf{1} & (\mathsf{ext}\ A)^{\mathsf{ES}} a_0 & :\equiv * \\
(\Delta \rhd\!\rhd T)^{\mathsf{ES}}\langle \delta_0, x_0\rangle & :\equiv \langle \Delta^{\mathsf{ES}}\delta_0, T^{\mathsf{ES}}x_0\rangle & (\pi\ A\ B)^{\mathsf{ES}} f_0 & :\equiv \boldsymbol{\lambda}\ (\lambda x. B^{\mathsf{ES}}(f_0 \cdot x)) \\
(\Delta[w])^{\mathsf{ES}}\delta_0 & :\equiv \Delta^{\mathsf{ES}}\delta_0 & (T[\sigma])^{\mathsf{ES}} x_0 & :\equiv T^{\mathsf{ES}} x_0 \\
 & & (T[w])^{\mathsf{ES}} x_0 & :\equiv T^{\mathsf{ES}} x_0 \\
(\mathsf{T}^1)^{\mathsf{ES}}_{f_0}\ x_0 & :\equiv \pi_1\ f_0 \cdot x_0 & & \\
(\mathsf{T}^2\ t)^{\mathsf{ES}}_{f_0}\ x_0 & :\equiv \pi_2\ f_0 \cdot x_0 & &
\end{array}$$

Figure 11.2: Erased section operators

We could alternatively see the section operators defined above as a non-indexed counterpart of those in Section 10.2.2.

Given constructor algebras c_0, c_0^D, the type $C^{\mathsf{ED}}_{f_0}\ c_0\ c_0^D$ is the section equation relative to the constructor specified by C, stating that the functions in f_0 "respect" the algebra structure when applied to c_0, c_0^D and their arguments. The type $\Gamma^{\mathsf{ES}}_{m_0}$ is then the product of all these equations, one for each constructor in the specification; it would thus be natural to define $\Gamma^{\mathsf{ES}}_{f_0}$ by recursion on specifications, like we did for Γ^{S}. Instead, we use a simpler, equivalent definition that quantifies over constructor terms.

$$\Gamma^{\mathsf{ES}}_{f_0}\ \gamma_0\ \gamma_0^D :\equiv \forall\{C\}(c : \mathsf{CTm}\ \Gamma\ C) \to C^{\mathsf{ES}}_{f_0}\ (C^{\mathsf{E}}\gamma_0)\ (C^{\mathsf{ED}}\gamma_0^D)$$

Thus, we can just define $w^{\mathsf{ES}} s_0 :\equiv \lambda c. s_0\ (c[w])$. The remaining operators are just an erased version of IIT section operators. We summarize all of them in Figure 11.2.

We prove the following lemmata about erased sections, which are mutually stated and proved:

Lemma 11.1.1. For any $\sigma : \mathsf{Sub}\ \Delta\ \nabla$, $f_0 : \mathsf{Map}_0\ F_0\ D_0$, algebras $\gamma_0, \gamma_0^D, \delta_0$, and proof $s_0 : \Gamma^{\mathsf{ES}}_{f_0}\ \gamma_0\ \gamma_0^D$,

$$\nabla^{\mathsf{ES}}_{f_0}\ (\sigma^{\mathsf{E}}\gamma_0\ \delta_0) = \sigma^{\mathsf{ED}}\gamma_0^D\ (\Delta^{\mathsf{ES}}_{f_0}\ \delta_0) \qquad \blacksquare$$

Proof. By induction on σ, and Lemma 11.1.2. □

Lemma 11.1.2. For any $t : \mathsf{Tm}\ \Gamma\ \Delta\ T$, $f_0 : \mathsf{Map}_0\ F_0\ D_0$, algebras $\gamma_0, \gamma_0^D, \delta_0$, and proof $s_0 : \Gamma^{\mathsf{ES}}_{f_0}\ \gamma_0\ \gamma_0^D$,

$$T^{\mathsf{ES}}_{f_0}\ (t^{\mathsf{E}}\gamma_0\ \delta_0) = t^{\mathsf{ED}}\gamma_0^D\ (\Delta^{\mathsf{ES}}_{f_0}\ \delta_0)$$ ■

Proof. By induction on t, and Lemma 11.1.1. □

Remark 11.1.3. For any $\Gamma, f_0, \gamma_0, \gamma_0^D$, the type $\Gamma^{\mathsf{ES}}_{f_0}\,\gamma_0\,\gamma_0^D$ is an iterated product of metatheoretical identity types, and therefore a (weak) proposition. ■

Definition 11.1.3. For any $\Gamma : \mathsf{Spec}$, erased algebra $(F_0, \gamma_0) : \mathsf{Alg}_0\ \Gamma$, and erased displayed algebra $(D_0, \gamma_0^D) : \mathsf{Alg}_0^D\ \Gamma\ (F_0, \gamma_0)$, the type $\mathsf{Sect}_0\ \Gamma\ (F_0, \gamma_0)\ (D_0, \gamma_0^D)$ of sections between them is given by pairs of functions $f_0 : \mathsf{Map}_0\ F_0\ D_0$ and a proof $s_0 : \Gamma^{\mathsf{ES}}_{f_0}\ \gamma_0\ \gamma_0^D$. ■

Erased sections allow us to talk about erased algebras that are *inductive*, in the sense that they support the induction principle expected of erased *inductive types*.

Definition 11.1.4. Given an IIT specification $\Gamma : \mathsf{Spec}$, an erased algebra $alg_0 : \mathsf{Alg}_0\ \Gamma$ is *inductive*, or *section inductive*, when for any erased displayed algebra $alg_0^D : \mathsf{Alg}_0^D\ \Gamma\ alg_0$ there exists a section $\mathsf{Sect}_0\ \Gamma\ alg_0\ alg_0^D$. ■

11.1.4 Existence of erased types

In the previous sections we have generalized the idea of erased types, by defining a general notion of erased algebra indexed over IIT specifications. Still, just defining the algebras is not enough for our objective, which is not only to show that IIT algebras can be reduced to pairs of erased and predicate algebras, but also that such IIT algebras always exist for any specification. To show that this is the case, we must first establish that erased algebras always exist, as they constitute the building blocks for all our encoded IIT algebras. This existence result for erased algebras is the topic of this section. In particular, we will show that the type $\mathsf{Alg}_0\ \Gamma$ is inhabited for any $\Gamma : \mathsf{Spec}$, and that the inhabitant is *section inductive*.

We proceed by internalizing the specification syntax and the algebra operators as target-level inductive types.

$$\mathsf{Spec}_0, \mathsf{Params}_0, \mathsf{Ty}_0, \mathsf{Wk}_0, \mathsf{Ctor}_0 : \mathsf{Type}$$
$$\mathsf{CTm}_0 : \ulcorner\mathsf{Spec}_0\urcorner \to \ulcorner\mathsf{Ctor}_0\urcorner \to \mathsf{Type}$$

$$
\begin{array}{ll}
\diamond_0 & : \ulcorner \mathsf{Spec}_0 \urcorner \\
\_ \rhd_0 \_ & : \ulcorner \mathsf{Spec}_0 \urcorner \to \ulcorner \mathsf{Ctor}_0 \urcorner \to \ulcorner \mathsf{Spec}_0 \urcorner \\
\bullet_0 & : \ulcorner \mathsf{Params}_0 \urcorner \\
\_ \rhd\!\!\rhd_0 \_ & : \ulcorner \mathsf{Params}_0 \urcorner \to \ulcorner \mathsf{Ty}_0 \urcorner \to \ulcorner \mathsf{Params}_0 \urcorner \\
\mathsf{T}^1_0 & : \ulcorner \mathsf{Ty}_0 \urcorner \\
\mathsf{T}^2_0 & : \ulcorner \mathsf{Ty}_0 \urcorner \\
\mathsf{ext}_0 & : \mathsf{SType} \to \ulcorner \mathsf{Ty}_0 \urcorner \\
\pi_0 & : \mathsf{SType} \to \ulcorner \mathsf{Ty}_0 \urcorner \to \ulcorner \mathsf{Ty}_0 \urcorner
\end{array}
\qquad
\begin{array}{ll}
\mathsf{ctor}^1_0 & : \ulcorner \mathsf{Params}_0 \urcorner \to \ulcorner \mathsf{Ctor}_0 \urcorner \\
\mathsf{ctor}^2_0 & : \ulcorner \mathsf{Params}_0 \urcorner \to \ulcorner \mathsf{Ctor}_0 \urcorner \\
\mathsf{id}_0 & : \ulcorner \mathsf{Wk}_0 \urcorner \\
\mathsf{drop}_0 & : \ulcorner \mathsf{Wk}_0 \urcorner \\
\_ \circ_0 \_ & : \ulcorner \mathsf{Wk}_0 \urcorner \to \ulcorner \mathsf{Wk}_0 \urcorner \to \ulcorner \mathsf{Wk}_0 \urcorner \\
\mathsf{cvz}_0 & : \ulcorner \mathsf{CTm}_0 \; (\Gamma_0 \rhd_0 C_0) \; C_0 \urcorner \\
\mathsf{cvs}_0 & : \ulcorner \mathsf{CTm}_0 \; \Gamma_0 \; C_0 \urcorner \\
 & \to \ulcorner \mathsf{CTm}_0 \; (\Gamma_0 \rhd_0 C'_0) \; C_0 \urcorner
\end{array}
$$

These types are clearly definable as inductive families; we thus assume their existence in the target theory. Being inductive, these types come equipped with induction principles. We will slightly abuse notation and define *internal, target-level* functions by induction on these types with the same notation as meta-level pattern matching, with the implicit understanding that such definitions can easily be translated to completely equivalent uses of the internal pattern matching/eliminators in the target theory.

As an example of function definition by internal pattern matching, we define weakening of internal constructor terms:

$$
\begin{array}{ll}
\_[\_]_0 & : \ulcorner \mathsf{CTm}_0 \; \Gamma_0 \; C_0 \urcorner \to \ulcorner \mathsf{Wk}_0 \; \Omega_0 \; \Gamma_0 \urcorner \to \ulcorner \mathsf{CTm}_0 \; \Omega_0 \; C_0 \urcorner \\
c[\mathsf{id}_0]_0 & := c \\
c[\mathsf{drop}_0]_0 & := \mathsf{cvs}_0 \; c \\
c[w_1 \circ_0 w_2]_0 & := (c[w_2]_0)[w_1]_0
\end{array}
$$

We define internal versions of the erased algebra operators, by induction on erased specifications (Figure 11.3). We define regular and displayed versions of the operators, and like their external counterparts we parameterize them by erased carriers $F_0 : \mathsf{Ca}_0$ and $D : \mathsf{Ca}^D_0 \; F_0$.

Given any erased specification $\Omega_0 : \ulcorner \mathsf{Spec}_0 \urcorner$, there exist the following target-level types $\mathsf{Erased} : \mathsf{Ca}_0$ generated by constructors $\mathsf{erased}^1, \mathsf{erased}^2$:

$$
\mathsf{Erased}^1 : \mathsf{SType}, \quad \mathsf{Erased}^2 : \mathsf{SType}, \quad \mathsf{Er} :\equiv (\mathsf{Erased}^1, \mathsf{Erased}^2)
$$
$$
\mathsf{erased}^1 : (\Delta_0 : \ulcorner \mathsf{Params}_0 \urcorner) \to \ulcorner \mathsf{CTm}_0 \; \Omega_0 \; (\mathsf{ctor}^1_0 \; \Delta_0) \urcorner \to \ulcorner \Delta_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner \to \ulcorner \mathsf{Erased}^1 \urcorner
$$
$$
\mathsf{erased}^2 : (\Delta_0 : \ulcorner \mathsf{Params}_0 \urcorner) \to \ulcorner \mathsf{CTm}_0 \; \Omega_0 \; (\mathsf{ctor}^2_0 \; \Delta_0) \urcorner \to \ulcorner \Delta_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner \to \ulcorner \mathsf{Erased}^2 \urcorner
$$

$$\Gamma_0 : \ulcorner\mathsf{Spec}_0\urcorner \;\vdash\; {\Gamma_0}^{\overline{\mathsf{E}}}_{F_0} : \mathsf{Type} \qquad \Delta_0 : \ulcorner\mathsf{Params}_0\urcorner \;\vdash\; {\Delta_0}^{\overline{\mathsf{E}}}_{F_0} : \mathsf{SType}$$

$$C_0 : \ulcorner\mathsf{Ctor}_0\urcorner \;\vdash\; {C_0}^{\overline{\mathsf{E}}}_{F_0} : \mathsf{SType} \qquad T_0 : \ulcorner\mathsf{Ty}_0\urcorner \;\vdash\; {T_0}^{\overline{\mathsf{E}}}_{F_0} : \mathsf{SType}$$

$$w_0 : \ulcorner\mathsf{Wk}_0\;\Omega_0\;\Gamma_0\urcorner \;\vdash\; {w_0}^{\overline{\mathsf{E}}}_{F_0} : \ulcorner{\Omega_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner \to \ulcorner{\Gamma_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner$$

$$\begin{aligned}
{\Gamma_0}^{\overline{\mathsf{E}}} &:= \mathbf{\Pi}\,(C_0 : \mathsf{Ctor}_0)(\mathsf{CTm}_0\,\Gamma_0\,C_0 \Rightarrow {C_0}^{\overline{\mathsf{E}}})\\
(\mathsf{ctor}^1_0\;\Delta_0)^{\overline{\mathsf{E}}} &:= {\Delta_0}^{\overline{\mathsf{E}}}_{F_0} \Rightarrow \pi_1\;F_0\\
(\mathsf{ctor}^2_0\;\Delta_0)^{\overline{\mathsf{E}}} &:= {\Delta_0}^{\overline{\mathsf{E}}}_{F_0} \Rightarrow \pi_2\;F_0\\
{\mathsf{id}_0}^{\overline{\mathsf{E}}}\;\gamma_0 &:= \gamma_0\\
{\mathsf{drop}_0}^{\overline{\mathsf{E}}}\;\gamma_0 &:= \boldsymbol{\lambda}\,\_\;c\,.\,\gamma_0\cdot\_\cdot\mathsf{cvs}_0\;c\\
(w_1 \circ_0 w_2)^{\overline{\mathsf{E}}}\;\gamma_0 &:= {w_1}^{\overline{\mathsf{E}}}\;({w_2}^{\overline{\mathsf{E}}}\;\gamma_0)
\end{aligned}
\qquad
\begin{aligned}
{\bullet_0}^{\overline{\mathsf{E}}} &:= \mathbf{1}\\
(\Delta \rhd\!\!\rhd_0 A_0)^{\overline{\mathsf{E}}} &:= {\Delta_0}^{\overline{\mathsf{E}}} \times {A_0}^{\overline{\mathsf{E}}}\\
(\mathsf{ext}_0\;X)^{\overline{\mathsf{E}}} &:= X\\
(\pi_0\;A_0\;B_0)^{\overline{\mathsf{E}}} &:= A_0 \Rightarrow {B_0}^{\overline{\mathsf{E}}}\\
{\mathsf{T}^1_0}^{\overline{\mathsf{E}}} &:= \pi_1\;F_0\\
{\mathsf{T}^2_0}^{\overline{\mathsf{E}}} &:= \pi_2\;F_0
\end{aligned}$$

$$\Gamma_0 : \ulcorner\mathsf{Spec}_0\urcorner \;\vdash {\Gamma_0}^{\overline{\mathsf{ED}}}_{D_0} : \ulcorner{\Gamma_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner \to \mathsf{Type} \qquad C_0 : \ulcorner\mathsf{Ctor}_0\urcorner \vdash {C_0}^{\overline{\mathsf{ED}}}_{D_0} : \ulcorner{C_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner \to \mathsf{Type}$$

$$\Delta_0 : \ulcorner\mathsf{Params}_0\urcorner \vdash {\Delta_0}^{\overline{\mathsf{ED}}}_{D_0} : \ulcorner{\Delta_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner \to \mathsf{Type} \qquad T_0 : \ulcorner\mathsf{Ty}_0\urcorner \;\vdash {T_0}^{\overline{\mathsf{ED}}}_{D_0} : \ulcorner{T_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner \to \mathsf{Type}$$

$$c_0 : \mathsf{CTm}_0\;\Gamma_0\;C_0 \;\vdash {c_0}^{\overline{\mathsf{ED}}}_{D_0} : (\gamma_0 : \ulcorner{\Gamma_0}^{\overline{\mathsf{E}}}_{F_0}\urcorner)\ulcorner{\Gamma_0}^{\overline{\mathsf{ED}}}_{D_0}\;\gamma_0\urcorner \to \ulcorner{C_0}^{\overline{\mathsf{ED}}}_{D_0}\urcorner$$

$$\begin{aligned}
&{\diamond_0}^{\overline{\mathsf{ED}}}\;\_ := \mathbf{1}\\
&(\Gamma_0 \rhd_0 C_0)^{\overline{\mathsf{ED}}}\;\gamma_0 :=\\
&\qquad {\Gamma_0}^{\overline{\mathsf{ED}}}\;(\boldsymbol{\lambda}\,c\,.\,\gamma_0\cdot\mathsf{cvs}_0\;c) \times {C_0}^{\overline{\mathsf{ED}}}\;(\gamma_0\cdot\mathsf{cvz}_0)\\
&(\mathsf{ctor}^1_0\;\Delta_0)^{\overline{\mathsf{ED}}}\;c_0 :=\\
&\qquad \mathbf{\Pi}(\delta_0 : {\Delta_0}^{\overline{\mathsf{E}}})\;({\Delta_0}^{\overline{\mathsf{ED}}}\;\delta_0 \Rightarrow \pi_1\,D_0\,(c_0\cdot\delta_0))\\
&(\mathsf{ctor}^2_0\;\Delta_0)^{\overline{\mathsf{ED}}}\;c_0 :=\\
&\qquad \mathbf{\Pi}(\delta_0 : {\Delta_0}^{\overline{\mathsf{E}}})\;({\Delta_0}^{\overline{\mathsf{ED}}}\;\delta_0 \Rightarrow \pi_2\,D_0\,(c_0\cdot\delta_0))\\
&{\bullet_0}^{\overline{\mathsf{ED}}}\delta_0 := \mathbf{1}\\
&(\Delta_0 \rhd\!\!\rhd_0 T_0)^{\overline{\mathsf{ED}}}\langle\delta_0, x_0\rangle := {\Delta_0}^{\overline{\mathsf{ED}}}\;\delta_0 \times {T_0}^{\overline{\mathsf{ED}}}\;x_0
\end{aligned}
\qquad
\begin{aligned}
&(\mathsf{ext}_0\;\_)^{\overline{\mathsf{ED}}}\_ := \mathbf{1}\\
&(\pi_0\;A_0\;B_0)^{\overline{\mathsf{ED}}} f_0 :=\\
&\qquad \mathbf{\Pi}\,(a : A_0)({B_0}^{\overline{\mathsf{ED}}}(f_0\cdot a))\\
&{\mathsf{T}^1_0}^{\overline{\mathsf{ED}}}\;x_0 := \pi_1\;D_0\;x_0\\
&{\mathsf{T}^2_0}^{\overline{\mathsf{ED}}}\;x_0 := \pi_2\;D_0\;x_0\\
&{\mathsf{cvz}_0}^{\overline{\mathsf{ED}}}\,\_\;\gamma^D_0 := \mathsf{snd}\;\gamma^D_0\\
&(\mathsf{cvs}_0\;c)^{\overline{\mathsf{ED}}}\,\_\;\gamma^D_0 := c^{\overline{\mathsf{ED}}}\;\_\;(\mathsf{fst}\;\gamma^D_0)
\end{aligned}$$

Figure 11.3: Internal erased algebra operators

The list of terms above specifies a simple mutual inductive definition, therefore we can assume the existence of these terms in the target theory. Note that the self-reference in the expression $\Delta_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}}$ does not pose problems to this definition, since $\lambda\, X \,.\, \Delta_0{}^{\overline{\mathsf{E}}}_{X}$ is strictly positive for any Δ_0.

We can generalize the constructors as the following recursive function:

$$\begin{aligned}
&\mathsf{erased} : (C_0 : \ulcorner \mathsf{Ctor}_0 \urcorner)(c : \ulcorner \mathsf{CTm}_0\ \Omega_0\ C_0 \urcorner) \to \ulcorner C_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner \\
&\mathsf{erased}\ (\mathsf{ctor}^1_0\ \Delta_0)\ c :\equiv \boldsymbol{\lambda}\,(\mathsf{erased}^1\ \Delta_0\ c) \\
&\mathsf{erased}\ (\mathsf{ctor}^2_0\ \Delta_0)\ c :\equiv \boldsymbol{\lambda}\,(\mathsf{erased}^2\ \Delta_0\ c)
\end{aligned}$$

Definition 11.1.5. There is an internal erased algebra $\overline{\omega}_0 : \ulcorner \Omega_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner$ on the pair Er:

$$\overline{\omega}_0 :\equiv \boldsymbol{\lambda}\, C_0 \,.\, \boldsymbol{\lambda}\, c_0 \,.\, \mathsf{erased}\ C_0\ c_0$$

■

Definition 11.1.6. For any internal displayed algebra with carriers $D_0 : \mathsf{Ca}^D_0\ \mathsf{Er}$ and structure $\overline{\omega}^D_0 : \ulcorner \Omega_0{}^{\overline{\mathsf{ED}}}_{D_0} \urcorner$, there are functions f^1_0, f^2_0 defined recursively and mutually with the following:

$$\begin{aligned}
&f^1_0 : (x : \ulcorner \mathsf{Erased}^1 \urcorner) \to \ulcorner \pi_1\, D_0\, x \urcorner &\qquad& T_0{}^{\overline{\mathsf{ES}}} : (x_0 : \ulcorner T_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner) \to \ulcorner T_0{}^{\overline{\mathsf{ED}}}_{D_0}\, x_0 \urcorner \\
&f^2_0 : (x : \ulcorner \mathsf{Erased}^2 \urcorner) \to \ulcorner \pi_2\, D_0\, x \urcorner && \Delta_0{}^{\overline{\mathsf{ES}}} : (\delta_0 : \ulcorner \Delta_0{}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner) \to \ulcorner \Delta_0{}^{\overline{\mathsf{ED}}}_{D_0}\, \delta_0 \urcorner
\end{aligned}$$

$$\begin{aligned}
&f^1_0\,(\mathsf{erased}^1\ \Delta_0\ c\ \delta_0) && :\equiv c^{\overline{\mathsf{ED}}}_{D_0}\, \overline{\omega}^D_0 \cdot \delta_0 \cdot \Delta_0{}^{\overline{\mathsf{ES}}}_{D_0}\, \delta_0 &\qquad& (\mathsf{ext}_0\ X)^{\overline{\mathsf{ES}}}\ \_ && :\equiv * \\
&f^2_0\,(\mathsf{erased}^2\ \Delta_0\ c\ \delta_0) && :\equiv c^{\overline{\mathsf{ED}}}_{D_0}\, \overline{\omega}^D_0 \cdot \delta_0 \cdot \Delta_0{}^{\overline{\mathsf{ES}}}_{D_0}\, \delta_0 && (\pi_0\ \_\ B_0)^{\overline{\mathsf{ES}}}\ f_0 && :\equiv \boldsymbol{\lambda}\, x \,.\, B_0{}^{\overline{\mathsf{ES}}}\,(f_0 \cdot x) \\
& && && \mathsf{T}^1_0{}^{\overline{\mathsf{ES}}}\ x_0 && :\equiv f^1_0\ x_0 \\
&\bullet_0{}^{\overline{\mathsf{ES}}}\ \_ && :\equiv * && \mathsf{T}^2_0{}^{\overline{\mathsf{ES}}}\ x_0 && :\equiv f^2_0\ x_0 \\
&(\Delta_0 \rhd\!\rhd_0 T_0)^{\overline{\mathsf{ES}}}\ \langle \delta_0, x_0 \rangle && :\equiv \langle \Delta_0{}^{\overline{\mathsf{ES}}}\ \delta_0, T_0{}^{\overline{\mathsf{ES}}}\ x_0 \rangle
\end{aligned}$$

We thus obtain maps $f_0 : \mathsf{Map}_0\ \mathsf{Er}\ D_0$ as $f_0 :\equiv (\boldsymbol{\lambda}\, f^1_0, \boldsymbol{\lambda}\, f^2_0)$. ■

We now relate external and internal algebra operators and specifications, starting with the erasure operators in Figure 11.4.

We can relate internal and external algebra operators directly, when the internal algebra is calculated on erased IIT specifications:

Lemma 11.1.3. Given $F_0 : \mathsf{Ca}_0, D_0 : \mathsf{Ca}^D_0\ F_0$, the following holds:

$$\begin{aligned}
&\Delta^{\overline{\mathsf{E}}}_{F_0} = (\Delta^{\downarrow})^{\overline{\mathsf{E}}}_{F_0}, &\quad& \Delta^{\overline{\mathsf{ED}}}_{D_0}\, \delta_0 = (\Delta^{\downarrow})^{\overline{\mathsf{ED}}}_{D_0}\, \delta_0, &\quad& \text{for all } \Delta : \mathsf{Params}\ \Gamma, \delta_0 \\
&T^{\overline{\mathsf{E}}}_{F_0} = (T^{\downarrow})^{\overline{\mathsf{E}}}_{F_0}, && T^{\overline{\mathsf{ED}}}_{D_0}\, x_0 = (T^{\downarrow})^{\overline{\mathsf{ED}}}_{D_0}\, x_0, && \text{for all } T : \mathsf{Ty}\ \Gamma\ \Delta, x_0
\end{aligned}$$

■

$$\begin{array}{llll}
\Gamma : \mathsf{Spec} & \vdash \Gamma^{\downarrow} : \ulcorner \mathsf{Spec}_0 \urcorner & T : \mathsf{Ty}\,\Gamma\,\Delta & \vdash T^{\downarrow} : \ulcorner \mathsf{Ty}_0 \urcorner \\
C : \mathsf{Ctor}\,\Gamma & \vdash C^{\downarrow} : \ulcorner \mathsf{Ctor}_0 \urcorner & w : \mathsf{Wk}\,\Gamma\,\Omega & \vdash w^{\downarrow} : \ulcorner \mathsf{Wk}_0\,\Gamma^{\downarrow}\,\Omega^{\downarrow} \urcorner \\
\Delta : \mathsf{Params}\,\Gamma & \vdash \Delta^{\downarrow} : \ulcorner \mathsf{Params}_0 \urcorner & c : \mathsf{Ctor}\,\Gamma\,C & \vdash c^{\downarrow} : \ulcorner \mathsf{CTm}_0\,\Gamma^{\downarrow}\,C^{\downarrow} \urcorner
\end{array}$$

$$\begin{array}{llll}
\diamond^{\downarrow} & :\equiv \diamond_0 & T[w]^{\downarrow} & :\equiv T^{\downarrow} \\
(\Gamma \rhd C)^{\downarrow} & :\equiv \Gamma^{\downarrow} \rhd_0 C^{\downarrow} & T[\sigma]^{\downarrow} & :\equiv T^{\downarrow} \\
\bullet^{\downarrow} & :\equiv \bullet_0 & \mathsf{cvz}^{\downarrow} & :\equiv \mathsf{cvz}_0 \\
(\Delta \rhd\!\rhd T)^{\downarrow} & :\equiv \Delta^{\downarrow} \rhd\!\rhd_0 T^{\downarrow} & (\mathsf{cvs}\,c)^{\downarrow} & :\equiv \mathsf{cvs}_0\,c^{\downarrow} \\
\Delta[w]^{\downarrow} & :\equiv \Delta^{\downarrow} & c[w]^{\downarrow} & :\equiv c^{\downarrow}[w^{\downarrow}]_0 \\
(\mathsf{ext}\,X)^{\downarrow} & :\equiv \mathsf{ext}_0\,X & \mathsf{id}^{\downarrow} & :\equiv \mathsf{id}_0 \\
(\pi\,A\,B)^{\downarrow} & :\equiv \pi_0\,A\,B^{\downarrow} & \mathsf{drop}^{\downarrow} & :\equiv \mathsf{drop}_0 \\
\mathsf{T}^{1\downarrow} & :\equiv \mathsf{T}^1_0 & (w_1 \circ w_2)^{\downarrow} & :\equiv w_1^{\downarrow} \circ_0 w_2^{\downarrow} \\
(\mathsf{T}^2\,t)^{\downarrow} & :\equiv \mathsf{T}^2_0 & &
\end{array}$$

Figure 11.4: Specification of erasure operators

Proof. By straightforward induction on Δ, T. □

As a consequence of Lemma 11.1.3, we also prove the following similar results for section maps:

Lemma 11.1.4. Let $f_0 :\equiv (\boldsymbol{\lambda}\, f_0^1, \boldsymbol{\lambda}\, f_0^2)$ be the maps structure defined as per Definition 11.1.6. Then the following holds

$$\begin{array}{ll}
\Delta^{\mathsf{ES}}\delta_0\,f_0 = (\Delta^{\downarrow})^{\overline{\mathsf{ES}}}\delta_0 & \text{for } \Delta : \mathsf{Params}\,\Gamma, \delta_0 : \ulcorner \Delta^{\mathsf{E}}_{\mathsf{Er}} \urcorner \\
T^{\mathsf{ES}}x_0\,f_0 = (T^{\downarrow})^{\overline{\mathsf{ES}}}x_0 & \text{for } T : \mathsf{Ty}\,\Gamma\,\Delta, x_0 : \ulcorner T^{\mathsf{E}}_{\mathsf{Er}} \urcorner
\end{array}$$

■

Proof. By straightforward induction on Δ, T. □

We have functions relating whole algebra structures on the external and internal levels, for any F_0, D_0:

$$\Gamma : \mathsf{Spec}, \gamma : \ulcorner (\Gamma^{\downarrow})^{\mathsf{E}}_{F_0} \urcorner \qquad \vdash \ \gamma^{\uparrow} : \Gamma^{\mathsf{E}}_{F_0}$$

$$\Gamma : \mathsf{Spec}, \gamma : \ulcorner (\Gamma^{\downarrow})^{\overline{\mathsf{E}}}_{F_0} \urcorner, \gamma^{D} : \Gamma^{\mathsf{ED}}_{D_0}\,(\gamma^{\uparrow}) \;\vdash\; \gamma^{D\downarrow} : \ulcorner (\Gamma^{\downarrow})^{\overline{\mathsf{ED}}}_{D_0}\,\gamma \urcorner$$

We define both by induction on external specification contexts $\Gamma : \mathsf{Spec}$:

$$\begin{aligned}
\gamma^{\uparrow} &:\equiv \star && \text{for } \Gamma \equiv \diamond \\
\gamma^{\uparrow} &:\equiv (\mathsf{drop}_0{}^{\overline{\mathsf{E}}}\gamma)^{\uparrow}, \gamma \cdot \_ \cdot \mathsf{cvz}_0 && \text{for } \Gamma \equiv \Gamma' \rhd C
\end{aligned}$$

$$\begin{aligned}
\gamma^{D\downarrow} &:\equiv * && \text{for } \Gamma \equiv \diamond \\
\gamma^{D\downarrow} &:\equiv \langle (\mathsf{drop}^{\mathsf{ED}}\gamma^{D})^{\downarrow}, \gamma^{D} \cdot \mathsf{cvz} \rangle && \text{for } \Gamma \equiv \Gamma' \rhd C
\end{aligned}$$

Lemma 11.1.5. Let $\Gamma : \mathsf{Spec}$, and algebras $\gamma_0 : (\Gamma^{\downarrow})^{\overline{\mathsf{E}}}_{F_0}$ and $\gamma_0^{D} : \Gamma^{\mathsf{ED}}_{D_0}\,({\gamma_0}^{\uparrow})$. The following holds for all $c : \mathsf{CTm}\,\Gamma\,C$:

$$\gamma_0 \cdot C^{\downarrow} \cdot c^{\downarrow} = c^{\mathsf{E}}_{F_0}\,({\gamma_0}^{\uparrow}), \qquad\qquad c^{\downarrow}{}^{\overline{\mathsf{ED}}}_{D_0}\,\gamma_0\,({\gamma_0^{D}}^{\downarrow}) = \gamma_0^{D}\,c \qquad ■$$

Proof. By induction on c. □

We finally wrap everything up with the main result of this section:

Theorem 11.1.1. Let $\Omega : \mathsf{Spec}$ be a specification. There exists a section inductive erased algebra $alg_0 : \mathsf{Alg}_0\,\Omega$. ■

Proof. Let $\Omega_0 :\equiv \Omega^{\downarrow}$. By Definition 11.1.5, there are target-level types $\mathsf{Er} : \mathsf{Ca}_0$ and an internal structure $\overline{\omega}_0 : \ulcorner {\Omega_0}^{\overline{\mathsf{E}}}_{\mathsf{Er}} \urcorner$. Thus, we obtain an external algebra structure $\omega_0 : \Omega^{\mathsf{E}}_{\mathsf{Er}}$ as $\omega_0 :\equiv {\overline{\omega}_0}^{\uparrow}$. We now show that the algebra (Er, ω_0) is section inductive. Assume we are given a displayed algebra $(D_0, \omega_0^{D}) : \mathsf{Alg}_0^{D}\,\Omega\,(\mathsf{Er}, \omega_0)$. Via erasure, we obtain an internal displayed algebra $\overline{\omega}_0^{D} : \ulcorner {\Omega_0}^{\overline{\mathsf{ED}}} \urcorner\,\overline{\omega}_0$, therefore by Definition 11.1.6 we have maps $f_0 : \mathsf{Map}_0\,\mathsf{Er}\,D_0$. We are left to prove that f_0 constitutes a section, which amounts to show the following equation for any $B : \mathsf{Base}\,\Omega\,\Delta$, $c : \mathsf{CTm}\,\Omega\,(\mathsf{ctor}\,\Delta\,B)$, $\delta_0 : \ulcorner \Delta^{\mathsf{E}}_{\mathsf{Er}} \urcorner$:

$$B^{\mathsf{ES}}_{f_0}\,(c^{\mathsf{E}}\,\omega_0 \cdot \delta_0) = \omega_0^{D}\;c \cdot \delta_0 \cdot \Delta^{\mathsf{ES}}_{f_0}\,\delta_0$$

We prove it by case analysis on B, starting with $B :\equiv \mathsf{T}^1$:

$$(\mathsf{T}^1)^{\mathsf{ES}}_{f_0}\,(c^{\mathsf{E}}\,\omega_0 \cdot \delta_0)$$

$$\begin{array}{ll}
= & \quad \{ \text{ Lemma 11.1.3, Lemma 11.1.5 } \} \\
& f_0^1\ (\mathsf{erased}^1\ c^{\downarrow}\ \delta_0) \\
\equiv & \\
& c^{\downarrow\,\overline{\mathsf{ED}}}\ \overline{\omega}_0\ \overline{\omega}_0^D \cdot \delta_0 \cdot (\Delta^{\downarrow})^{\overline{\mathsf{ES}}}\ \delta_0 \\
= & \quad \{ \text{ Lemma 11.1.4 } \} \\
& c^{\downarrow\,\overline{\mathsf{ED}}}\ \overline{\omega}_0\ \overline{\omega}_0^D \cdot \delta_0 \cdot \Delta^{\mathsf{ES}}\ \delta_0 \\
= & \quad \{ \text{ Lemma 11.1.5 } \} \\
& \omega_0^D\ c \cdot \delta_0 \cdot \Delta^{\mathsf{ES}}\ \delta_0
\end{array}$$

The other case of B is identical, except we have f_0^2 instead of f_0^1. □

11.2 Well-formedness predicates

We now give a general account of well-formedness predicates. As we did for IITs and erased types, we will begin by defining *predicate algebras*, before pinpointing what particular properties we require of those algebras to be useful for our IIT encoding.

11.2.1 Algebras of well-formedness predicates

As usual, let us revisit our running example, Con/Ty, and generalize from there.

Example 11.2.1 (metatheory)**.** The predicate types from Section 8.1 were defined as follows:

$$\begin{array}{ll}
\mathsf{Con}_1 & : \mathsf{Con}_0 \to \mathbf{Prop} \\
\mathsf{Ty}_1 & : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathbf{Prop} \\
\\
\bullet_1 & : \mathsf{Con}_1\ \bullet_0 \\
\_ \rhd_1 \_ & : \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ A_0 \to \mathsf{Con}_1\ (\Gamma_0 \rhd_0 A_0) \\
\iota_1 & : \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0) \\
\hat{\pi}_1 & : \mathsf{Con}_1\ \Gamma_0 \to (\forall n. \mathsf{Ty}_1\ \Gamma_0\ (F_0\, n)) \to \mathsf{Ty}_1\ \Gamma_0\ (\hat{\pi}_0\ \Gamma_0\ F_0) \\
\bar{\pi}_1 & : \mathsf{Con}_1\, \Gamma_0 \to \mathsf{Ty}_1\, \Gamma_0\, A_0 \to \mathsf{Ty}_1\, (\Gamma_0 \rhd_0 A_0)\, B_0 \to \mathsf{Ty}_1\, \Gamma_0\, (\bar{\pi}_0\, \Gamma_0\, A_0\, B_0)
\end{array}$$

We can read the list above as simply a description of an algebra of the predicate types specified by Con/Ty. Just like predicates are indexed by erased types, this algebra is indexed by the erased algebra $(\mathsf{Con}_0, \mathsf{Ty}_0, \bullet_0, \ldots)$. ■

Example 11.2.2 (target theory)**.** Given $((C_0, T_0), (\bullet_0, \triangleright_0, \iota_0, \hat{\pi}_0, \bar{\pi}_0)) : \mathsf{Alg}_0\,\Theta$, a target-level predicate algebra indexed by it consists of carriers

$$\begin{aligned}
&C_1 : \ulcorner C_0 \urcorner \to \mathsf{SProp}\\
&T_1 : \ulcorner C_0 \urcorner \to \ulcorner T_0 \urcorner \to \mathsf{SProp}
\end{aligned}$$

and a term of the iterated product type:

$$\begin{aligned}
\bullet_1 &: \ulcorner C_1\ \bullet_0 \urcorner\\
\triangleright_1 &: \ulcorner \mathbf{\Pi}\ (c_0 : C_0)(t_0 : T_0)(C_1\ c_0 \Rightarrow T_1\ c_0\ t_0 \Rightarrow C_1(\triangleright_0 \cdot c_0 \cdot t_0)) \urcorner\\
\iota_1 &: \ulcorner \mathbf{\Pi}(c_0 : C_0)(C_1\ c_0 \Rightarrow T_1\ (\iota_0 \cdot c_0)) \urcorner\\
\hat{\pi}_1 &: \ulcorner \mathbf{\Pi}(c_0 : C_0)(f_0 : \mathsf{Nat} \Rightarrow T_0)\\
&\qquad (C_1\ c_0 \Rightarrow \mathbf{\Pi}(n : \mathsf{Nat})(T_1\ c_0\ (f_0 \cdot n)) \Rightarrow T_1\ (\hat{\pi}_0 \cdot c_0 \cdot f_0)) \urcorner\\
\bar{\pi}_1 &: \ulcorner \mathbf{\Pi}\ (c_0 : C_0)(a_0 : T_0)(b_0 : T_0)\\
&\qquad\quad (C_1\ c_0 \Rightarrow T_1\ c_0\ a_0 \Rightarrow T_1\ (\triangleright_0 \cdot c_0 \cdot a_0)\ b_0 \Rightarrow T_1\ c_0\ (\bar{\pi}_0 \cdot c_0 \cdot a_0 \cdot b_0)) \urcorner
\end{aligned}$$

■

We now want to generalize predicate algebras to arbitrary IIT specifications. We first define an operator Ca_1 that calculates the carrier types of the predicate algebra given carrier types of the erased algebra.

$$\begin{aligned}
&\mathsf{Ca}_1 : \mathsf{Ca}_0 \to \mathbf{Type}\\
&\mathsf{Ca}_1(A_0, B_0) :\equiv (\ulcorner A_0 \urcorner \to \mathsf{SProp}) \times (\ulcorner A_0 \urcorner \to \ulcorner B_0 \urcorner \to \mathsf{SProp})
\end{aligned}$$

We then define an operator $\_^{\mathsf{W}}$ that calculates the predicate algebra structure of terms. For a specification Γ and erased algebra (F_0, γ_0), and predicate carriers $F_1 : \mathsf{Ca}_1\ F_0$, the type $\Gamma^{\mathsf{W}}_{F_1}\,\gamma_0$ is a product of target-level types, one for each term of the predicate algebra. We define Γ^{W} along with auxiliary operators that compute the algebra types for each component of an IIT specification.

$$\begin{aligned}
&\Gamma : \mathsf{Spec} && \vdash \Gamma^{\mathsf{W}}_{F_1} : \Gamma^{\mathsf{E}}_{F_0} \to \mathbf{Type}\\
&C : \mathsf{Ctor}\ \Gamma && \vdash C^{\mathsf{W}}_{F_1} : \Gamma^{\mathsf{E}}_{F_0} \to \ulcorner C^{\mathsf{E}}_{F_0} \urcorner \to \mathsf{SProp}\\
&\Delta : \mathsf{Params}\ \Gamma && \vdash \Delta^{\mathsf{W}}_{F_1} : \Gamma^{\mathsf{E}}_{F_0} \to \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner \to \mathsf{SProp}\\
&T : \mathsf{Ty}\ \Gamma\ \Delta && \vdash T^{\mathsf{W}}_{F_1} : \Gamma^{\mathsf{E}}_{F_0} \to \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner \to \ulcorner T^{\mathsf{E}}_{F_0} \urcorner \to \mathsf{SProp}
\end{aligned}$$

We also have operators that lift the action of terms and weakenings/substitutions to algebras:

$$\begin{array}{lll}
\sigma : \mathsf{Sub}\ \Delta\ \nabla & \vdash \sigma^{\mathsf{W}} & : \Gamma^{\mathsf{W}}\gamma_0 \to \ulcorner \Delta^{\mathsf{W}}\gamma_0\ \delta_0 \urcorner \to \ulcorner \nabla^{\mathsf{W}}\gamma_0\ (\sigma^{\mathsf{E}}\gamma_0\ \delta_0)\urcorner \\
w : \mathsf{Wk}\ \Omega\ \Gamma & \vdash w^{\mathsf{W}} & : \Omega^{\mathsf{W}}\gamma_0 \to \Gamma^{\mathsf{W}}(w^{\mathsf{E}}\gamma_0) \\
t : \mathsf{Tm}\ \Gamma\ \Delta\ A & \vdash t^{\mathsf{W}} & : \Gamma^{\mathsf{W}}\gamma_0 \to \ulcorner \Delta^{\mathsf{W}}\gamma_0\ \delta_0 \urcorner \to \ulcorner A^{\mathsf{W}}\gamma_0\ \delta_0\ (t^{\mathsf{E}}\gamma_0\ \delta_0)\urcorner \\
c : \mathsf{CTm}\ \Gamma\ C & \vdash c^{\mathsf{W}} & : \Gamma^{\mathsf{W}}\gamma_0 \to \ulcorner C^{\mathsf{W}}\ \gamma_0\ (c^{\mathsf{E}}\gamma_0)\urcorner
\end{array}$$

Most of these functions are just a slight variation of their erased algebras counterparts, to account for the additional algebra components in the indices. One important difference with erased algebras is that the predicate algebra component for external types is trivial. This is expected, as there is no need to enforce any kind of well-formedness on external types:

$$(\mathsf{ext}\ \_)^{\mathsf{W}}\_\ \_\ \_ :\equiv \top$$

Algebras for base types just correspond to the respective carrier types applied to suitable erased terms. In the case of a base type of the form $\mathsf{T}^2\ t$, the first index depends on t and thus is calculated from it.

$$(\mathsf{T}^2\ t)^{\mathsf{W}}_{F_1}\ \gamma_0\ \delta_0\ x_0 :\equiv \pi_2\ F_2\ (t^{\mathsf{E}}\gamma_0\ \delta_0)\ x_0$$

The remaining operators are a straightforward generalization of the erased ones to account for additional indexing. We summarize all of them in Figure 11.5.

Example 11.2.3. To check our work, we can easily verify that given $\theta_0 : \Theta^{\mathsf{E}}_{(C_0,T_0)}$ such that $\theta_0 \equiv (\bullet_0, \triangleright_0, \iota_0, \hat{\pi}_0, \bar{\pi}_0)$, the type of $\theta_1 : \Theta^{\mathsf{W}}_{(C_1,T_1)}\ \theta_0$ computes to the one in Example 11.2.2[1]. ■

Remark 11.2.1. Note that for any Γ, F_1, γ_0, the type $\Gamma^{\mathsf{W}}_{F_1}\ \gamma_0$ is an iterated product of target-level propositional terms. By definition $\mathsf{Term}\ P$ is a meta-level (weak) proposition for any target-level proposition P, thus we can easily show that $\Gamma^{\mathsf{W}}_{F_1}\ \gamma_0$ is a meta-level (weak) proposition. ■

We would like to have a way to construct predicate algebras Γ^{W} in a more direct way that doesn't rely on induction on specifications. We thus prove the following:

[1]As usual, up to straightforward equivalences.

$$\begin{aligned}
\diamond^{\mathsf{W}}\ \gamma_0 &:\equiv \mathbb{1}\\
(\Gamma \rhd C)^{\mathsf{W}}(\gamma_0, c_0) &:\equiv \Gamma^{\mathsf{W}}\gamma_0 \times \ulcorner C^{\mathsf{W}}\gamma_0\ c_0 \urcorner
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{ctor}\ \Delta\ T)^{\mathsf{W}}\gamma_0\ f_0 :\equiv\\
&\quad \boldsymbol{\Pi}(\delta_0 : \_)\ (\Delta^{\mathsf{W}}\gamma_0\ \delta_0 \Rightarrow T^{\mathsf{W}}\gamma_0\ \delta_0\ (f_0 \cdot \delta_0))
\end{aligned}$$

$$\begin{aligned}
&\bullet^{\mathsf{W}}\ \gamma_0\ \delta_0 :\equiv \top\\
&(\Delta \rhd\!\rhd T)^{\mathsf{W}}\gamma_0\ \langle \delta_0, x_0 \rangle :\equiv\\
&\quad \Delta^{\mathsf{W}}\gamma_0\ \delta_0 \times T^{\mathsf{W}}\gamma_0\ \delta_0\ x_0\\
&(\Delta[w])^{\mathsf{W}}\gamma_0\ \delta_0 :\equiv \Delta^{\mathsf{W}}(w^{\mathsf{E}}\gamma_0)\ \delta_0
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{ext}\ \_)^{\mathsf{W}}\gamma_0\ \delta_0\ a_0 :\equiv \top\\
&(\pi\ A\ B)^{\mathsf{W}}\gamma_0\ \delta_0\ f_0 :\equiv\\
&\quad \boldsymbol{\Pi}(x : A)(B^{\mathsf{W}}\gamma_0\ \langle \delta_0, x \rangle\ (f_0 \cdot x))\\
&(T[\sigma])^{\mathsf{W}}\gamma_0\ \delta_0\ x_0 :\equiv T^{\mathsf{W}}\gamma_0\ (\sigma^{\mathsf{E}}\ \gamma_0\ \delta_0)\ x_0\\
&(T[w])^{\mathsf{W}}\gamma_0\ \delta_0\ x_0 :\equiv T^{\mathsf{W}}\ (w^{\mathsf{E}}\ \gamma_0)\ \delta_0\ x_0
\end{aligned}$$

$$\begin{aligned}
\mathsf{id}^{\mathsf{W}}\gamma_1 &:\equiv \gamma_1\\
\mathsf{drop}^{\mathsf{W}}\gamma_1 &:\equiv \pi_1\ \gamma_1\\
(w_1 \circ w_2)^{\mathsf{W}}\gamma_1 &:\equiv {w_2}^{\mathsf{W}}({w_1}^{\mathsf{W}}\gamma_1)
\end{aligned}$$

$$\begin{aligned}
(\mathsf{T}^1)^{\mathsf{W}}_{F_1}\ \gamma_0\ \delta_0\ x_0 &:\equiv \pi_1\ F_1\ x_0\\
(\mathsf{T}^2\ t)^{\mathsf{W}}_{F_1}\ \gamma_0\ \delta_0\ x_0 &:\equiv \pi_2\ F_2\ (t^{\mathsf{E}}\ \gamma_0\ \delta_0)\ x_0
\end{aligned}$$

$$\begin{aligned}
\mathsf{id}^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \delta_1\\
(\mathsf{ext}\ \sigma\ t)^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \langle \sigma^{\mathsf{W}}\gamma_1\ \delta_1, t^{\mathsf{W}}\gamma_1\ \delta_1 \rangle\\
\mathsf{drop}^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \mathsf{fst}\ \delta_1\\
(\sigma \circ \tau)^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \tau^{\mathsf{W}}\gamma_1\ (\sigma^{\mathsf{W}}\gamma_1\ \delta_1)\\
(\sigma[w])^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \sigma^{\mathsf{W}}(w^{\mathsf{W}}\gamma_1)\ \delta_1
\end{aligned}$$

$$\begin{aligned}
\mathsf{vz}^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \mathsf{snd}\ \delta_1\\
(\mathsf{vs}\ t)^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv t^{\mathsf{W}}\gamma_1\ (\mathsf{fst}\ \delta_1)\\
(\mathsf{ext}\ A\ x)^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv \mathsf{truth}\\
(\mathsf{capp}\ c\ \sigma)^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv c^{\mathsf{W}}\gamma_1 \cdot \_ \cdot \sigma^{\mathsf{W}}\gamma_1\ \delta_1\\
(\mathsf{ap}\ t)^{\mathsf{W}}\gamma_1\ \{\delta_0\}\ \delta_1 &:\equiv t^{\mathsf{W}}\gamma_1\ (\mathsf{fst}\ \delta_1) \cdot \mathsf{snd}\ \delta_0\\
(\mathsf{lm}\ t)^{\mathsf{W}}\gamma_1\ \{\delta_0\}\ \delta_1 &:\equiv \boldsymbol{\lambda} x.t^{\mathsf{W}}\gamma_1\ \langle \delta_0, x \rangle\ \langle \delta_1, \_ \rangle\\
(t[\sigma])^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv t^{\mathsf{W}}\gamma_1\ (\sigma^{\mathsf{W}}\gamma_1\ \delta_1)\\
(t[w])^{\mathsf{W}}\gamma_1\ \delta_1 &:\equiv t^{\mathsf{W}}(w^{\mathsf{W}}\gamma_1)\ \delta_1
\end{aligned}$$

$$\begin{aligned}
\mathsf{cvz}^{\mathsf{W}}\gamma_1 &:\equiv \pi_2\ \gamma_1\\
(\mathsf{cvs}\ c)^{\mathsf{W}}\gamma_1 &:\equiv c^{\mathsf{W}}(\pi_1\ \gamma_1)\\
(c[w])^{\mathsf{W}}\gamma_1 &:\equiv c^{\mathsf{W}}(w^{\mathsf{W}}\gamma_1)
\end{aligned}$$

Figure 11.5: Well-formedness predicate algebra operators

Lemma 11.2.1. Let $F_0 : \mathsf{Ca}_0$ and $F_1 : \mathsf{Ca}_1\ F_0$. For any $\Gamma : \mathsf{Spec}$, we have the following propositional equivalence:

$$[\forall\ (C : \mathsf{Ctor}\ \Gamma)\ (c : \mathsf{CTm}\ \Gamma\ C) \to \ulcorner C^{\mathsf{W}}_{F_1}\ \gamma_0\ (c^{\mathsf{E}}\gamma_0)\urcorner] \quad \leftrightarrow \quad \Gamma^{\mathsf{W}}_{F_1}\ \gamma_0 \qquad \blacksquare$$

Proof. By induction on Γ. □

Definition 11.2.1. Given a specification Γ and erased algebra (F_0, γ_0), a predicate algebra $\mathsf{Alg}_1\ \Gamma\ (F_0, \gamma_0)$ is a pair of predicates $F_1 : \mathsf{Ca}_1\ F_0$ and an element of $\Gamma^{\mathsf{W}}_{F_1}\ \gamma_0$. ■

11.2.2 Inversion principles

We have generalized well-formedness predicates to arbitrary IITs, by defining a notion of predicate algebra indexed over specifications.

As we will see in Section 11.3.1, a pair of an erased algebra and a predicate algebra is enough to construct an IIT algebra via the so-called Σ-construction. However, it will become clear in Chapter 12 that a mere algebra is not enough for the purpose of our encoding. In particular, given a pair of erased and predicate algebras, the IIT algebra Σ-constructed from them is not necessarily section inductive, unless the source algebras come with additional structure.

In the case of erased algebras, the extra structure we require is section induction. To get an idea of what this could be for predicate algebras, let us look back at the Con/Ty example of Section 8.1. Unlike erased types, we did not require the predicate types to be inductively defined, however we did ask for some inversion principles. These can be summarized into two kinds, one relating well-formedness of a compound and well-formedness of its components, and another establishing an equational constraint on the erased indices of the predicate type.

We now state the inversion principles in general.

Definition 11.2.2. Let $(F_0, \gamma_0) : \mathsf{Alg}_0\,\Gamma$, and $(F_1, \gamma_1) : \mathsf{Alg}_1\,\Gamma\,(F_0, \gamma_0)$. We say that (F_1, γ_1) supports inversions if there exist functions of the following types:

$$\begin{aligned}
\mathsf{pred\text{-}inv} \;&: (c : \mathsf{CTm}\;\Gamma\;(\mathsf{ctor}\;\Delta\;T))\,(\delta_0 : \ulcorner\Delta^{\mathsf{E}}_{F_0}\urcorner)\\
&\to \ulcorner T^{\mathsf{W}}_{F_1}\;\gamma_0\;\delta_0\;(c^{\mathsf{E}}\gamma_0\cdot\;\delta_0)\urcorner\\
&\to \ulcorner\Delta^{\mathsf{W}}_{F_1}\;\gamma_0\;\delta_0\urcorner
\end{aligned}$$

$$\begin{aligned}
\mathsf{pred\text{-}ix\text{-}inv} : (c : \mathsf{CTm}\;\Gamma\;(\mathsf{ctor}\;\Delta\;(\mathsf{T}^2\;t)))\,(\delta_0 : \ulcorner\Delta^{\mathsf{E}}_{F_0}\urcorner)\,(a_0 : \ulcorner\pi_1\;F_0\urcorner)\\
\to \ulcorner\pi_2\;F_1\;a_0\;(c^{\mathsf{E}}\gamma_0\cdot\delta_0)\urcorner\\
\to \ulcorner a_0 \approx t^{\mathsf{E}}\gamma_0\;\delta_0\urcorner
\end{aligned}$$

■

The first inversion principle states that given any proof of predicate type applied to an erased term obtained by applying a constructor c to some arguments δ_0, then we can extract a proof that the predicate holds for all the arguments δ_0.

The second inversion principle states that for predicates A_1, B_1, if we have a proof of $B_1\;a_0\;b_0$, then a_0 is completely determined by b_0.

11.2.3 Existence of predicate types

In order to reduce IIT algebras to their erased and predicate components, we need to argue for the existence of erased and predicate algebras. We have discussed the existence of erased algebras in Section 11.1.4. We now tackle predicate algebras, showing that the type $\mathsf{Alg}_1\,\Gamma\,\mathit{alg}_0$ is inhabited for any $\Gamma : \mathsf{Spec}$ and *section inductive* $\mathit{alg}_0 : \mathsf{Alg}_0\,\Gamma$, and that the inhabitant has IIT inversions as prescribed in Definition 11.2.2.

We fix an arbitrary specification $\Omega : \mathsf{Spec}$, and inductive erased algebra $(F_0, \omega_0) : \mathsf{Alg}_0\ \Omega$. The goal is to define a pair of predicates $\mathsf{P}_1 : \mathsf{Ca}_1\ F_0$, as well as a proof term $\omega_1 : \Omega^{\mathsf{W}}_{\mathsf{P}_1}\ \omega_0$, such that the predicate algebra (P_1, ω_1) has inversion principles.

We define the predicate types P_1 by recursion on their erased indices F_0, generalizing the construction from Section 8.1.1. As per definition of inductive erased algebras, induction on F_0 amounts to defining a suitable displayed erased algebra $(\mathsf{P}^D_0, \omega^D_0)$ giving the motives and methods of the induction. We define the motives as the following pair P^D_0:

$$\begin{aligned}
&\mathsf{P}^D_0 : \mathsf{Ca}^D_0\ F_0\\
&\mathsf{P}^D_0 :\equiv ((\lambda\_.\mathcal{P}), (\lambda\_ \to \mathsf{El}_{\mathcal{U}}\ (\pi_1\ F_0) \Rightarrow \mathcal{P}))
\end{aligned}$$

The methods are given as a displayed erased algebra structure ω^D_0:

$$\omega^D_0 : \forall\{C : \mathsf{Ctor}\ \Omega\}(c : \mathsf{CTm}\ \Omega\ C) \to \ulcorner C^{\mathsf{ED}}_{\mathsf{P}^D_0}\ \omega_0 \urcorner$$

We define ω^D_0 by case analysis on C, starting from $C \equiv \mathsf{ctor}\ \Delta\ \mathsf{T}^1$ for some Δ:

$$\omega^D_0\ \{\mathsf{ctor}\ \Delta\ \mathsf{T}^1\}\ c :\equiv \boldsymbol{\lambda}\,\delta_0.\boldsymbol{\lambda}\,\delta^D_0.\ ?$$

Here we are asked to define well-formedness of the canonical expression given by applying the erased constructor algebra of c to arguments δ_0; that is, $\pi_1\ \mathsf{P}_1\ (c^{\mathsf{E}}_{F_0}\ c \cdot \delta_0)$. Well-formedness on canonical forms should be defined recursively in terms of well-formedness of the subcomponents. We thus seek a way to recurse on δ_0 and apply the inductively generated predicates to each of its elements. Recall that this is exactly how we defined Con_1 recursively in Section 8.1.1. For instance:

$$\mathsf{Con}_1\ (\Gamma_0 \rhd_0 A_0) :\equiv \mathsf{Con}_1\ \Gamma_0 \times \mathsf{Ty}_1\ \Gamma_0\ A_0$$

We implement this recursive traversal via the auxiliary operator $\Delta^{\mathsf{ih}}\ \delta_0\ \delta^D_0$, which takes a list $\delta^D_0 : \Delta^{\mathsf{ED}}\ \delta_0$ and produces a target-level proposition. Conceptually, δ^D_0 is a list of recursively generated well-formedness predicates, one for each type in Δ, applied to their respective arguments in δ_0. $\Delta^{\mathsf{ih}}\ \delta_0\ \delta^D_0$ thus recursively traverses this list, fully applies those predicates that are only partially applied, and combines them into an iterated logical conjunction to form a single proposition. The $\omega_0^D\{\mathsf{ctor}\ \Delta\ \mathsf{T}^1\}\ c$ clause is thus given as

$$\omega_0^D\ \{\mathsf{ctor}\ \Delta\ \mathsf{T}^1\}\ c :\equiv \boldsymbol{\lambda}\,\delta_0.\boldsymbol{\lambda}\,\delta_0^D.\Delta^{\mathsf{ih}}\ \delta_0\ \delta_0^D$$

We define Δ^{ih} in Figure 11.6, mutually with similar operators for local type and base type specifications (with overloaded names). Their definition is relatively straightforward: we do recursion on parameter telescopes and local types until we reach a base type, at which point we end up with the “current” inductive hypothesis already applied to its designated arguments. In the T^1 case, this is exactly the proposition we need, and we return it as is; in the T^2 case the inductive hypothesis is a binary relation, thus we apply it further to the appropriate term algebra. Since we cannot do recursion over the fixed Ω, all operators are parameterized by a weakening $w : \mathsf{Wk}\ \Omega\ \Gamma$, which we omit writing when the identity.

The other case $C \equiv \mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t)$ is similar, with the addition of an equational constraint.

$$\begin{aligned}&\omega_0^D\ \{\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t)\}\ c :\equiv\\ &\quad \boldsymbol{\lambda}\,\delta_0.\boldsymbol{\lambda}\,\delta_0^D.\boldsymbol{\lambda}\,a_0.\ \Delta^{\mathsf{ih}}\ \delta_0\ \delta_0^D \times a_0 \approx t^{\mathsf{E}}\ \gamma_0\ \delta_0\end{aligned}$$

Here, we use the erased algebra operator t^{E} to compute the constraint on the index a_0 according to the specified term t. Again, this is not unlike the definition of Ty_1 in Section 8.1.1. For instance:

$$\mathsf{Ty}_1\ \Gamma_0\ (\bar{\pi}_0\ \Gamma_0'\ A_0\ B_0) :\equiv \mathsf{Con}_1\ \Gamma_0' \times \mathsf{Ty}_1\ \Gamma_0'\ A_0 \times \mathsf{Ty}_1\ (\Gamma' \triangleright_0 A_0)\ B_0 \times \Gamma_0 =_P \Gamma_0'$$

In addition to recursive statements of well-formedness, we constrained the index Γ_0 obtained as input to the predicate (corresponding to a_0 above) to be equal to the context Γ_0' argument to the constructor $\bar{\pi}_0$ (corresponding to $t^{\mathsf{E}}\ \gamma_0 \cdot \delta_0$ above.)

This concludes the definition of ω_0^D. Thus, by induction (Definition 10.2.5), we have a pair of functions $f_0 : \mathsf{Map}_0\ F_0\ \mathsf{P}_0^D$ with a proof $s_0 : \Gamma_{f_0}^{\mathsf{ES}}\ \gamma_0\ \gamma_0^D$. From these functions we immediately obtain a pair of predicates $\mathsf{P}_1 : \mathsf{Ca}_1\ F_0$ on the erased types F_0:

$$\begin{aligned}&\mathsf{P}_1 : \mathsf{Ca}_1\ F_0\\ &\mathsf{P}_1 :\equiv (\lambda a_0.\pi_1\ f_0 \cdot a_0\ ,\ \lambda\ a_0\ b_0\ .\ (\pi_2\ f_0) \cdot b_0 \cdot a_0)\end{aligned}$$

We now need to confirm that P_1 constitutes a predicate algebra, i.e., that there exists an algebra structure $\omega_1 : \Omega_{\mathsf{P}_1}^{\mathsf{W}}\ \omega_0$. We begin with a lemma stating that the action of the inductive hypotheses operators is actually the same as predicate algebras operators on the carriers P_1.

Lemma 11.2.2. For all $w : \mathsf{Wk}\ \Omega\ \Gamma, \Delta : \mathsf{Params}\ \Gamma$ and $\delta_0 : \Delta_{F_0}^{\mathsf{E}}$:

$$\begin{aligned}
\Delta : \mathsf{Params}\ \Gamma &\vdash \Delta^{\mathsf{ih}}_w : (\delta_0 : \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner) \to \ulcorner \Delta^{\mathsf{ED}}_{\mathsf{P}^D_0}\ \delta_0 \urcorner \to \mathsf{SProp} \\
T : \mathsf{Ty}\ \Gamma\ \Delta &\vdash T^{\mathsf{ih}}_w : (\delta_0 : \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner)\{x_0 : \ulcorner T^{\mathsf{E}}_{F_0} \urcorner\} \to \ulcorner T^{\mathsf{ED}}_{\mathsf{P}^D_0}\ x_0 \urcorner \to \mathsf{SProp}
\end{aligned}$$

$$\begin{aligned}
\bullet^{\mathsf{ih}}_w\ \_\ \delta^D_0 &:\equiv \top \\
(\Delta \rhd\!\rhd T)^{\mathsf{ih}}_w\ \langle \delta_0, x_0 \rangle\ \langle \delta^D_0, x^D_0 \rangle &:\equiv \Delta^{\mathsf{ih}}_w\ \delta_0\ \delta^D_0 \times T^{\mathsf{ih}}_w\ \delta_0\ x^D_0 \\
(\Delta[w_2])^{\mathsf{ih}}_{w_1}\ \delta_0\ \delta^D_0 &:\equiv \Delta^{\mathsf{ih}}_{(w_1 \circ w_2)}\ \delta_0\ \delta^D_0
\end{aligned}$$

$$\begin{aligned}
(\mathsf{ext}\ \_)^{\mathsf{ih}}_w\ \delta_0\ a^D_0 &:\equiv \top \\
(\pi\ A\ B)^{\mathsf{ih}}_w\ \delta_0\ f^D_0 &:\equiv \mathbf{\Pi}\ (x : A)(B^{\mathsf{ih}}_w\ \langle \delta_0, x \rangle\ (f^D_0 \cdot x)) \\
(T[\sigma])^{\mathsf{ih}}_w\ \delta_0\ x^D_0 &:\equiv T^{\mathsf{ih}}_w\ (\sigma^{\mathsf{E}}(w^{\mathsf{E}}\gamma_0)\ \delta_0)\ x^D_0 \\
(T[w_2])^{\mathsf{ih}}_{w_1}\ \delta_0\ x^D_0 &:\equiv T^{\mathsf{ih}}_{(w_1 \circ w_2)}\ \delta_0\ x^D_0 \\
(\mathsf{T}^1)^{\mathsf{ih}}_w\ \delta_0\ x_0 &:\equiv x_0 \\
(\mathsf{T}^2\ t)^{\mathsf{ih}}_w\ \delta_0\ x_0 &:\equiv x_0 \cdot t^{\mathsf{E}}(w^{\mathsf{E}}\gamma_0)\ \delta_0
\end{aligned}$$

Figure 11.6: Inductive hypothesis operators for recursive predicates

- $\Delta^{\mathsf{ih}}_w\ (\Delta^{\mathsf{ES}}_{f_0}\ \delta_0) = \Delta^{\mathsf{W}}_{\mathsf{P}_1}\ (w^{\mathsf{E}}\gamma_0)\ \delta_0$
- $T^{\mathsf{ih}}_w\ \delta_0\ (T^{\mathsf{ES}}_{f_0}\ x_0) = T^{\mathsf{W}}_{\mathsf{P}_1}\ (w^{\mathsf{E}}\gamma_0)\ \delta_0\ x_0,$ for $T : \mathsf{Ty}\ \Gamma\ \Delta,\ x_0 : T^{\mathsf{E}}_{F_0}$ ■

Proof. By straightforward mutual induction and reflexivity. □

We finally establish the existence of a suitable algebra structure.

Lemma 11.2.3. There exists a proof of $\Omega^{\mathsf{W}}_{\mathsf{P}_1}\ \omega_0$. ■

Proof. We apply Lemma 11.2.1 and instead prove the following statement:

$$\forall\ (C : \mathsf{Ctor}\ \Omega)(c : \mathsf{CTm}\ \Omega\ C) \to \ulcorner C^{\mathsf{W}}_{\mathsf{P}_1}\ \omega_0\ (c^{\mathsf{E}}_{F_0}\ \omega_0) \urcorner$$

We proceed by case analysis on C. Both cases are proved immediately after rewriting via Lemma 11.2.2. □

We now show that the algebra just defined, which we temporarily label ω_1, has the necessary inversion principles as stated in Definition 11.2.2:

Lemma 11.2.4. The algebra (P_1, ω_1) has IIT-inversions. ■

Proof. From our construction, we have a section structure $s_0 : \Gamma^{\mathsf{ES}}_{f_0}\ \omega_0\ \omega_0^D$. From s_0 and by the way we defined P_1, we know that for all $c : \mathsf{CTm}\ \Omega\ (\mathsf{ctor}\ \Delta\ T)$ and $\delta_0 : \Delta^{\mathsf{E}}_{F_0}$, the following holds

$$T^{\mathsf{W}}_{\mathsf{P}_1}\ \omega_0\ \delta_0\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_0) = \omega_0^D\ c \cdot \delta_0 \cdot \Delta^{\mathsf{ES}}_{f_0}\ \delta_0$$

We substitute this equation in the statement of the inversion principles, and further rewrite via Lemma 11.2.2. Defining the inversions is then a straightforward application of projections. □

We can finally put everything together, and state the main result of this last part.

Theorem 11.2.1. Let Γ : Spec, and alg_0 : $\mathsf{Alg}_0\ \Gamma$ an inductive erased algebra. Then, there exists a predicate algebra P : $\mathsf{Alg}_1\ \Gamma\ alg_0$ that has inversions. ■

Proof. This whole section is a detailed description of how to construct such predicate algebra from a fixed specification and associated erased algebra. Since the fixed components are totally arbitrary, it follows that the construction works for any choice of these components. □

11.3 IIT Σ-algebra

In this section we prove the first point of Theorem 10.2.1, and one of the fundamental results of this part of the thesis. That is, we show that for any IIT specification we can construct a corresponding IIT algebra, obtained by packing together suitable erased terms and well-formedness proofs. This process is a generalization of the Σ-construction employed in Section 8.1.

The beauty of the Σ-construction is its generality, in that it can be defined purely as an operation on algebras: given an IIT specification and any arbitrary erased algebra and predicate algebra over it, we construct an IIT algebra. In particular, we do not require any extra structure or property, like induction or inversion principles, on the input algebras. A corollary of the Σ-construction is that if erased and predicate algebras always exist for any specification, then IIT algebras also exist.

We explain the general Σ-construction in Section 11.3.1 that follows. We make use of it in Section 11.3.2 to show that IIT algebras always exist for any given specification.

11.3.1 Σ-construction

The idea of the Σ-construction is that any inductive-inductive definition can be split into a definition of some erased types representing the untyped core structure of the IIT, and a definition of predicates on the erased types that carve out all the combinations of untyped terms that correspond to the original inductive-inductive definition. The original IIT, with its type formers and constructors, can then be recovered by pairing the erased and predicate types.

We take a look at two examples, where we apply the Σ-construction to the Con/Ty IIT both as a metatheoretic construction and as a target theory construction, before moving on to its generalization.

Example 11.3.1 (metatheory)**.** Recall how in Section 8.1 we were able to recover $\mathsf{Con}, \mathsf{Ty}$ by encoding them as dependent pairs of erased types and well-formedness predicates:

$$\begin{aligned}&\mathsf{Con} :\equiv \Sigma(\Gamma_0 : \mathsf{Con}_0)(\mathsf{Con}_1\ \Gamma_0)\\&\mathsf{Ty}\ \Gamma :\equiv \Sigma(A_0 : \mathsf{Ty}_0)(\mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0)\end{aligned}$$

Similarly, we were able to encode the constructors as pairs of the corresponding constructor in the erased types and predicates. For example:

$$\hat{\pi}\ (\Gamma_0, \Gamma_1)\ F :\equiv (\hat{\pi}_0\ \Gamma_0\ (\lambda n.\pi_1(F\, n)), \hat{\pi}_1\ \Gamma_1\ (\lambda n.\pi_2(F\, n)))$$ ■

Example 11.3.2 (target theory)**.** We can replicate the Σ-construction of Con/Ty in the target theory. Suppose we have erased carriers $(C_0, T_0) : \mathsf{Ca}_0$ and predicates $(C_1, T_1) : \mathsf{Ca}_1\ (C_0, T_0)$. We can then form the following family:

$$\begin{aligned}&C : \mathsf{SType} && T : \ulcorner C \urcorner \to \mathsf{SType}\\&C :\equiv \boldsymbol{\Sigma}\ C_0\ C_1 && T :\equiv \lambda\ \Gamma\ .\ \boldsymbol{\Sigma}\ (A_0 : T_0)\ (T_1\ (\mathsf{fst}\ \Gamma)\ A_0)\end{aligned}$$

Then, given an erased algebra structure $(\bullet_0, \triangleright_0, \iota_0, \hat{\pi}_0, \bar{\pi}_0)$ and a predicate algebra structure $(\bullet_1, \triangleright_1, \iota_1, \hat{\pi}_1, \bar{\pi}_1)$ over it, we can construct an IIT algebra structure $(\bullet, \triangleright, \iota, \hat{\pi}, \bar{\pi}) : \Theta^{\mathsf{A}}_{(C,T)}$ via the obvious pairing. For example:

$$\begin{aligned}&\triangleright : \ulcorner \boldsymbol{\Pi}(c : C)(T\, c \Rightarrow C) \urcorner\\&\triangleright :\equiv \boldsymbol{\lambda}\ \langle c_0, c_1\rangle \langle t_0, t_1\rangle\ .\ \langle \triangleright_0 \cdot c_0 \cdot t_0, \triangleright_1 \cdot \_ \cdot c_1 \cdot t_1\rangle\end{aligned}$$

$$\begin{aligned}&\hat{\pi} : \ulcorner \boldsymbol{\Pi}\ (\Gamma : C)((\mathsf{Nat} \Rightarrow T\ \Gamma) \Rightarrow T\,\Gamma) \urcorner\\&\hat{\pi} :\equiv \boldsymbol{\lambda}\ \langle c_0, c_1\rangle\ f\ .\ \langle \hat{\pi}_0 \cdot c_0 \cdot (\boldsymbol{\lambda}\ n\ .\ \mathsf{fst}\ (f \cdot n)), \hat{\pi}_1 \cdot \_ \cdot \_ \cdot c_1 \cdot (\boldsymbol{\lambda}\ n\ .\ \mathsf{snd}\ (f \cdot n))\rangle\end{aligned}$$

■

We now generalize the Σ-construction to arbitrary specifications and algebras, so that for any $\Gamma : \mathsf{Spec}$ and erased and predicate algebras $alg_0 : \mathsf{Alg}_0\,\Gamma$, $alg_1 : \mathsf{Alg}_1\,\Gamma\,alg_0$, we know how to construct an IIT algebra $alg : \mathsf{Alg}\,\Gamma$, which we will call Σ-*algebra*. We first deal with the carriers, and define a family Ca_Σ of IIT carriers indexed by a pair of erased and predicate carriers:

$$\begin{aligned}&\mathsf{Ca}_\Sigma : (F_0 : \mathsf{Ca}_0) \to \mathsf{Ca}_1\ F_0 \to \mathsf{Ca}\\ &\mathsf{Ca}_\Sigma\ (A_0, B_0)\ (A_1, B_1) :\equiv \mathbf{\Sigma}\ A_0\ A_1\ ,\ \lambda a.\mathbf{\Sigma}\ (b_0 : B_0)(B_1\ (\mathsf{fst}\ a_0)\ b_0)\end{aligned}$$

The idea is that if (F_0, γ_0) and (F_1, γ_1) are erased and predicate algebras, then $\mathsf{Ca}_\Sigma\,F_0\,F_1$ are the carriers of the induced Σ-algebra.

Next, we define a Σ-construction operator that assembles an IIT algebra structure from suitable erased and predicate algebra structures. Its pairing function is easily deduced from its type:

$$\Gamma^{\Sigma} : (\gamma_0 : \Gamma^{\mathsf{E}}_{F_0})(\gamma_1 : \Gamma^{\mathsf{W}}_{F_1}\ \gamma_0) \to \Gamma^{\mathsf{A}}_{(\mathsf{Ca}_\Sigma\ F_0\ F_1)}$$

We define this mutually with auxiliary operators that implement the same pairing for algebra structures specific to each subcomponent of an IIT specification:

$$\begin{aligned}&C : \mathsf{Ctor}\,\Gamma &&\vdash\ C^{\Sigma} : (c_0 : \ulcorner C^{\mathsf{E}}\urcorner)(c_1 : \ulcorner C^{\mathsf{W}}\ \gamma_0\ c_0\urcorner) \to \ulcorner C^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)\urcorner\\ &\Delta : \mathsf{Params}\,\Gamma &&\vdash\ \Delta^{\Sigma} : (\delta_0 : \ulcorner \Delta^{\mathsf{E}}\urcorner)(\delta_1 : \ulcorner \Delta^{\mathsf{W}}\ \gamma_0\ \delta_0\urcorner) \to \ulcorner \Delta^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)\urcorner\\ &T : \mathsf{Ty}\,\Gamma\,\Delta &&\vdash\ T^{\Sigma} : (x_0 : \ulcorner T^{\mathsf{E}}\urcorner)(x_1 : \ulcorner T^{\mathsf{W}}\ \gamma_0\ \delta_0\ x_0\urcorner) \to \ulcorner T^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)(\Delta^{\Sigma}\ \delta_0\ \delta_1)\urcorner\end{aligned}$$

We also define projection operators $\_^{\Sigma}_1$ and $\_^{\Sigma}_2$ out of Σ-algebra structures, which reflect the idea that these operators really just implement a pairing of algebras.

$$\begin{aligned}&\Delta : \mathsf{Params}\,\Gamma &&\vdash \Delta^{\Sigma}_1 : \ulcorner \Delta^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)\urcorner \to \ulcorner \Delta^{\mathsf{E}}_{F_0}\urcorner\\ & &&\quad\ \Delta^{\Sigma}_2 : (\delta : \ulcorner \Delta^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)\urcorner) \to \ulcorner \Delta^{\mathsf{W}}\ \gamma_0\ (\Delta^{\Sigma}_1\ \delta)\urcorner\\ &T : \mathsf{Ty}\,\Gamma\,\Delta &&\vdash T^{\Sigma}_1 : \ulcorner T^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)\ \delta\urcorner \to \ulcorner T^{\mathsf{E}}_{F_0}\urcorner\\ & &&\quad\ T^{\Sigma}_2 : (x : \ulcorner T^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)\ \delta\urcorner) \to \ulcorner T^{\mathsf{W}}_{F_1}\ \gamma_0\ (\Delta^{\Sigma}_1\ \delta)\ (T^{\Sigma}_1\ x)\urcorner\end{aligned}$$

These operators are all defined by a fairly straightforward induction on specifications. We give their full definition in Figure 11.7.

We prove some β and η laws for the projection operators.

Lemma 11.3.1. The following β-equations hold:

$$\begin{aligned}
&\bullet^{\Sigma}\ \_\ \_ :\equiv *\\
&(\Delta \rhd\!\rhd T)^{\Sigma}\ \langle \delta_0, x_0\rangle\ \langle \delta_1, x_1\rangle :\equiv\\
&\quad \langle \Delta^{\Sigma}\ \delta_0\ \delta_1, T^{\Sigma}\ x_0\ x_1\rangle\\
&(\Delta[w])^{\Sigma}\ \delta_0\ \delta_1 :\equiv \Delta^{\Sigma}\ \delta_0\ \delta_1\\
&\\
&\bullet^{\Sigma}_1\ \_ :\equiv *\\
&(\Delta \rhd\!\rhd T)^{\Sigma}_1\ \langle p, x\rangle :\equiv \langle \Delta^{\Sigma}_1\ p, T^{\Sigma}_1\ x\rangle\\
&(\Delta[w])^{\Sigma}_1\ p :\equiv \Delta^{\Sigma}_1\ p\\
&\\
&\bullet^{\Sigma}_2\ p :\equiv \mathsf{truth}\\
&(\Delta \rhd\!\rhd T)^{\Sigma}_2\ \langle p, x\rangle :\equiv \langle \Delta^{\Sigma}_2\ p, T^{\Sigma}_2\ x\rangle\\
&(\Delta[w])^{\Sigma}_2\ p :\equiv \Delta^{\Sigma}_2\ p\\
&\\
&(\mathsf{ext}\ X)^{\Sigma}\ x_0\ x_1 :\equiv a_0\\
&(\pi\ A\ B)^{\Sigma}\ f_0\ f_1 :\equiv \boldsymbol{\lambda}\, x\,.\, B^{\Sigma}\ (f_0 \cdot x)\ (f_1 \cdot x)\\
&(T[\sigma])^{\Sigma}\ x_0\ x_1 :\equiv T^{\Sigma}\ x_0\ x_1\\
&(T[w])^{\Sigma}\ x_0\ x_1 :\equiv T^{\Sigma}\ x_0\ x_1\\
&(\mathsf{T}^1)^{\Sigma}\ x_0\ x_1 :\equiv \langle x_0, x_1\rangle\\
&(\mathsf{T}^2\ t)^{\Sigma}\ x_0\ x_1 :\equiv \langle x_0, x_1\rangle
\end{aligned}$$

$$\begin{aligned}
&\diamond^{\Sigma}\ \gamma_0\ \gamma_1 :\equiv \star\\
&(\Gamma \rhd C)^{\Sigma}\ (\gamma_0, c_0)\ (\gamma_1, c_1) :\equiv\\
&\quad \Gamma^{\Sigma}\ \gamma_0\ \gamma_1, C^{\Sigma}\ c_0\ c_1\\
&\\
&(\mathsf{ext}\ X)^{\Sigma}_1\ x :\equiv x\\
&(\pi\ A\ B)^{\Sigma}_1\ f :\equiv \boldsymbol{\lambda}\, x\,.\, B^{\Sigma}_1\ (f \cdot x)\\
&(A[\sigma])^{\Sigma}_1\ x :\equiv A^{\Sigma}_1\ x\\
&(A[w])^{\Sigma}_1\ x :\equiv A^{\Sigma}_1\ x\\
&(\mathsf{T}^1)^{\Sigma}_1\ x :\equiv \mathsf{fst}\ x\\
&(\mathsf{T}^2\ t)^{\Sigma}_1\ x :\equiv \mathsf{fst}\ x\\
&\\
&(\mathsf{ext}\ X)^{\Sigma}_2\ x :\equiv \mathsf{truth}\\
&(\pi\ A\ B)^{\Sigma}_2\ f :\equiv \boldsymbol{\lambda}\, x\,.\, B^{\Sigma}_2\ (f \cdot x)\\
&(A[\sigma])^{\Sigma}_2\ x :\equiv A^{\Sigma}_2\ x\\
&(A[w])^{\Sigma}_2\ x :\equiv A^{\Sigma}_2\ x\\
&(\mathsf{T}^1)^{\Sigma}_2\ x :\equiv \mathsf{snd}\ x\\
&(\mathsf{T}^2\ t)^{\Sigma}_2\ x :\equiv \mathsf{snd}\ x
\end{aligned}$$

Figure 11.7: Σ-construction operators on algebras

1. $\Delta^{\Sigma}_1\ (\Delta^{\Sigma}\ \delta_0\ \delta_1) = \delta_0$, for $\Delta : \mathsf{Params}\ \Gamma$
2. $T^{\Sigma}_1\ (T^{\Sigma}\ x_0\ x_1) = x_0$, for $T : \mathsf{Ty}\ \Gamma\ \Delta$ ■

Proof. Mutual induction on specifications. □

Note that the β-equations regarding the second projection hold by the fact that they involve proofs of propositional types.

Lemma 11.3.2. The following η-equations hold:

1. $\Delta^{\Sigma}\ (\Delta^{\Sigma}_1\ \delta)(\Delta^{\Sigma}_2\ \delta) = \delta$, for $\Delta : \mathsf{Params}\ \Gamma$
2. $T^{\Sigma}\ (T^{\Sigma}_1\ x)(T^{\Sigma}_2\ x) = x$, for $T : \mathsf{Ty}\ \Gamma\ \Delta$ ■

Proof. Mutual induction on specifications. □

We have the following lemmata showing that algebra operators and Σ-construction operators for weakenings and substitutions commute with each other.

Lemma 11.3.3. The following holds:

1. $w^{\mathsf{A}}\ (\Omega^{\Sigma}\ \gamma_0\ \gamma_1) = \Gamma^{\Sigma}\ (w^{\mathsf{E}}\ \gamma_0)(w^{\mathsf{W}}\ \gamma_1), \quad \text{for } w : \mathsf{Wk}\ \Omega\ \Gamma$
2. $\sigma^{\mathsf{A}}\ (\nabla^{\Sigma}\ \delta_0\ \delta_1) = \Delta^{\Sigma}\ (\sigma^{\mathsf{E}}\ \gamma_0\ \delta_0)\ (\sigma^{\mathsf{W}}\ \gamma_1\ \delta_1), \quad \text{for } \sigma : \mathsf{Sub}\ \nabla\ \Delta$ ■

Proof. By induction on Γ, σ, and Lemma 11.3.4.2. □

We also have a similar result for (constructor) terms.

Lemma 11.3.4. The following holds:

1. $c^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1) = C^{\Sigma}\ (c^{\mathsf{E}}\ \gamma_0)(c^{\mathsf{W}}\ \gamma_1), \quad \text{for } C : \mathsf{Ctor}\ \Gamma, c : \mathsf{CTm}\ \Gamma\ C$
2. $t^{\mathsf{A}}\ (\Gamma^{\Sigma}\ \gamma_0\ \gamma_1)(\Delta^{\Sigma}\ \delta_0\ \delta_1) = T^{\Sigma}\ (t^{\mathsf{E}}\ \gamma_0\ \delta_0)(t^{\mathsf{W}}\ \gamma_1\ \delta_1), \quad \text{for } t : \mathsf{Tm}\ \Gamma\ \Delta\ T$ ■

Proof. By mutual induction on c, t, and Lemma 11.3.3.2. □

The result established by Lemma 11.3.4 is precisely what we would expect: (constructor) term algebras of an IIT Σ-algebra are pairs of erased and predicate (constructor) term algebras.

All the lemmata above are defined mutually with the Σ-construction operators, and are necessary for typechecking most of the operators themselves, although we do not explicitly show their application due to extensionality in the metatheory.

The Σ-construction we have just defined establishes a functional relation: any pair of erased and predicate algebras induce a corresponding IIT algebra. That is, for any $\Gamma : \mathsf{Spec}$, we have a function

$$\begin{aligned}
&\mathsf{sigma} : \Sigma\,(\mathit{alg}_0 : \mathsf{Alg}_0\,\Gamma)(\mathsf{Alg}_1\,\Gamma\,\mathit{alg}_0) \to \mathsf{Alg}\,\Gamma\\
&\mathsf{sigma}\ ((F_0, \gamma_0), (F_1, \gamma_1)) :\equiv (\mathsf{Ca}_{\Sigma}\,F_0\,F_1, \Gamma^{\Sigma}\,\gamma_0\,\gamma_1)
\end{aligned}$$

We refer to an IIT-algebra obtained via this function as a Σ-*algebra.*

Remark 11.3.1. Due to the nature of Σ-algebras as pairings of erased and predicate components, we will often use abbreviation symbols reminiscent of pair types and projections when manipulating elements of Σ-algebra structures, whenever the implicitly omitted terms are clear from context.

For $\Delta : \mathsf{Params}\ \Gamma, T : \mathsf{Ty}\ \Gamma\ \Delta$, we abbreviate

$$\begin{aligned}
\langle \delta_0, \delta_1 \rangle &:\equiv \Delta^{\Sigma}\ \{\gamma_1\}\ \delta_0\ \delta_1\\
\langle x_0, x_1 \rangle &:\equiv T^{\Sigma}\ \{\gamma_1\ \delta_1\}\ x_0\ x_1
\end{aligned}$$

Moreover, we abbreviate projections $\__1^{\Sigma}, \__2^{\Sigma}$ as $\__0, \__1$, when applied to algebras of parameters and inner types. ■

11.3.2 Existence of IIT algebras

We finally prove the first of the two main results of this work, i.e. the first point of Theorem 10.2.1.

Theorem 11.3.1. For any $\Gamma : \mathsf{Spec}$, there exists an IIT algebra $alg : \mathsf{Alg}\ \Gamma$. ■

Proof. Assume Γ. By Theorem 11.1.1, there exists an erased algebra $alg_0 : \mathsf{Alg}_0\ \Gamma$. By Theorem 11.2.1, there exists a predicate algebra $alg_1 : \mathsf{Alg}_1\ \Gamma\ alg_0$. Via the Σ-construction, there obtain $alg :\equiv \mathsf{sigma}\ alg_0\ alg_1 : \mathsf{Alg}\ \Gamma$. □

It is now left to show that the IIT algebra defined as per Theorem 11.3.1 is an IIT, i.e. that it is section inductive. This will be the topic of Chapter 12.

Chapter 12

Defining the IIT eliminators

A consequence of the Σ-construction and our developments in the previous chapter is that any IIT specification always admits a corresponding IIT-algebra. It hasn't been established yet, however, that such constitutes the inductive-inductive type we are looking for. In other words, whether this algebra is equipped with the necessary induction principles that we would expect from an IIT.

Section 8.1 gave an answer to these questions for the concrete case of the Con/Ty IIT. In that setting, we showed that the types obtained by Σ-construction on the erased types and predicates do support the expected induction principle, provided the erased types are themselves inductive, and the well-formedness predicates support certain inversion principles. The type thus constructed was shown to be inductive-inductive by defining its eliminators and proving the associated equations. To do so, we relied on eliminator relations representing the graph of the eliminator functions, and a proof of left-totality of these relations.

The plan for this chapter is to replicate the same steps for arbitrary IIT specifications, in the setting of the target theory. We first generalize eliminator relations (Section 12.1), by defining a notion of relation algebra indexed over regular and displayed IIT algebras. We then identify the subset of all relation algebras that are useful for the purpose of constructing IIT eliminators: these will turn out to be those algebras equipped with certain inversion principles. Finally, we argue that relation algebras with the necessary inversions always exist under certain conditions (Section 12.1.3), and are left-total (Section 12.2.) We use these relation algebras and their left-totality property to define eliminators for any suitable IIT Σ-algebra (Section 12.3).

12.1 Eliminator relations

This section generalizes the eliminator relations seen in previous examples, by reframing them as algebras with suitable inversion properties. We begin by defining relation algebras by induction on the QIIT of specifications.

12.1.1 Relation algebras

To understand the structure of relation algebras, we take the usual look at a couple of examples, both from the point of view of the metatheory as well as the target theory.

Example 12.1.1 (metatheory)**.** Recall the definition of eliminator relations from Section 8.1. Given Con/Ty and displayed algebra $(\mathsf{Con}^D, \mathsf{Ty}^D, \bullet^D, \_ \rhd^D \_, \iota^D, \hat{\pi}^D, \bar{\pi}^D)$ over it, we specified the eliminator relations with the following list of types and constructors:

$$
\begin{aligned}
&\mathsf{Con}^R : (\Gamma : \mathsf{Con}) \to \mathsf{Con}^D\ \Gamma \to \mathbf{Type}\\
&\mathsf{Ty}^R \quad : (\Gamma^D : \mathsf{Con}^D\ \Gamma)(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}^D\ \Gamma\ A\ \Gamma^D \to \mathbf{Type}\\
&\bullet^R \quad\ : \mathsf{Con}^R\ \bullet\ \bullet^D\\
&\_ \rhd^R \_ : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ A\ A^D \to \mathsf{Con}^R\ (\Gamma \rhd A)\ (\Gamma^D \rhd^D A^D)\\
&\iota^R \quad\ : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ (\iota\ \Gamma)\ (\iota^D\ \Gamma^D)\\
&\bar{\pi}^R \quad : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to \mathsf{Ty}^R\ \Gamma^D\ A\ A^D \to \mathsf{Ty}^R\ (\Gamma^D \rhd^D A^D)\ B\ B^D\\
&\qquad\ \to \mathsf{Ty}^R\ \Gamma^D\ (\pi\ \Gamma\ A\ B)\ (\pi^D\ \Gamma^D\ A^D\ B^D)\\
&\hat{\pi}^R \quad : \mathsf{Con}^R\ \Gamma\ \Gamma^D \to ((n : \mathbb{N}) \to \mathsf{Ty}^R\ \Gamma^D\ (F\ n)\ (F^D\ n))\\
&\qquad\ \to \mathsf{Ty}^R\ \Gamma^D\ (\hat{\pi}\ \Gamma\ F)\ (\hat{\pi}^D\ \Gamma^D\ F^D)
\end{aligned}
$$

If we take $(\mathsf{Con}, \mathsf{Ty}, \bullet, \_ \rhd \_, \iota, \hat{\pi}, \bar{\pi})$ to be an arbitrary algebra, then the list of terms above is a specification of (metalevel) algebras of the eliminator relations induced by Con/Ty. ■

Example 12.1.2 (target theory)**.** Given a Con/Ty IIT algebra $((C, T), \theta) : \mathsf{Alg}\,\Theta$ and displayed algebra $((C^D, T^D), \theta^D) : \mathsf{Alg}^D\,\Theta\,((C, T), \theta)$, where $\theta \equiv (\bullet, \rhd, \iota, \hat{\pi}, \bar{\pi})$ and $\theta^D \equiv (\bullet^D, \rhd^D, \iota^D, \hat{\pi}^D, \bar{\pi}^D)$, a relation algebra induced by Θ, between $((C, T), \theta)$ and $((C^D, T^D), \theta^D)$, is given by families

$$
\begin{aligned}
&C^R : (\Gamma : \ulcorner C \urcorner) \to \ulcorner C^D\ \Gamma \urcorner \to \mathsf{SType}\\
&T^R : (\Gamma : \ulcorner C \urcorner)(\Gamma^D : \ulcorner C^D\ \Gamma \urcorner)(A : \ulcorner T\ \Gamma \urcorner) \to \ulcorner T^D\ \Gamma\ A\ \Gamma^D \urcorner \to \mathsf{SType}
\end{aligned}
$$

and an algebra structure given by the following terms:

$$\begin{aligned}
&\bullet^R : \ulcorner C^R \;\bullet\; \bullet^D \urcorner\\
&\triangleright^R : \ulcorner \mathbf{\Pi}\ (c : C)(t : T\,c)(c^D : C^D\ c)(t^D : T^D\ c^D\ t)\\
&\qquad\qquad (C^R\ c\,c^D \Rightarrow T^R\ c\,c^D\ t\,t^D \Rightarrow C^R\ \_\ (\triangleright^D \cdot c \cdot t \cdot c^D \cdot t^D))\urcorner\\
&\iota^R\ : \ulcorner \mathbf{\Pi}(c : C)(c^D : C^D c)(C^R\ c\,c^D \Rightarrow T^R\ c\,c^D\ (\iota \cdot c)\ (\iota^D \cdot c \cdot c^D))\urcorner\\
&\hat{\pi}^R : \ulcorner \mathbf{\Pi}\ (c : C)\ (f : \mathsf{Nat} \Rightarrow T\ c)(c^D : C^D\ c)(f^D : \mathbf{\Pi}(n : \mathsf{Nat})(T^D\ c^D\ (f \cdot n)))\\
&\qquad\qquad (C^R\ c\,c^D \Rightarrow \mathbf{\Pi}(n : \mathsf{Nat})(T^R\ c\,c^D\ (f \cdot n)\ (f^D \cdot n)) \Rightarrow\\
&\qquad\qquad\quad T^R\ c\,c^D\ (\hat{\pi} \cdot c \cdot f)\ (\hat{\pi}^D \cdot c \cdot f \cdot c^D \cdot f^D))\urcorner\\
&\bar{\pi}^R : \ulcorner \mathbf{\Pi}\ (c : C)(a : T\ c)(b : T\ (\triangleright \cdot\ \ c \cdot a))\\
&\qquad\qquad (c^D : C^D\ c)(a^D : T^D\ c^D\ a)(b^D : T^D\ (\triangleright^D \cdot c \cdot a \cdot c^D \cdot a^D)\ b)\\
&\qquad\qquad\quad (C^R\ c\,c^D \Rightarrow T^R\ c\,c^D\ a\,a^D \Rightarrow T^R\ \_\ \_\ b\,b^D \Rightarrow\\
&\qquad\qquad\qquad T^R\ c\,c^D\ (\bar{\pi} \cdot c \cdot a \cdot b)\ (\bar{\pi}^D \cdot c \cdot a \cdot b \cdot c^D \cdot a^D \cdot b^D))\urcorner
\end{aligned}$$

We can see that the structure above is a direct target-level translation of the metalevel structure shown in Example 12.1.1. ■

We now generalize the target-level construction shown in Example 12.1.2 to arbitrary IIT specifications. We define operators that take a specification, an IIT algebra, and a displayed algebra, and calculate the constituting components of an algebra of the corresponding eliminator relation.

The carrier types are easily calculated from the respective carriers of the regular and displayed algebra:

$$\begin{aligned}
&\mathsf{Ca}^R : (F : \mathsf{Ca}) \to \mathsf{Ca}^{\mathsf{D}}\ F \to \mathbf{Type}\\
&\mathsf{Ca}^R\ (C, T)(C^D, T^D) :\equiv\\
&\qquad ((c : \ulcorner C \urcorner) \to \ulcorner C^D\ c \urcorner \to \mathsf{SType}) \times\\
&\qquad ((c : \ulcorner C \urcorner)(c^D : \ulcorner C^D\ c \urcorner)(t : \ulcorner T\ c \urcorner) \to \ulcorner T^D\ c\ c^D\ t \urcorner \to \mathsf{SType})
\end{aligned}$$

As displayed in Figure 12.1.2, the structure of operations on these carriers can be expressed as an iterated product type. We define an operator $\_^{\mathsf{R}}$ that computes this structure by induction on specifications.

For $F : \mathsf{Ca}, D : \mathsf{Ca}^{\mathsf{D}}\ \Gamma, R : \mathsf{Ca}^R\ F\ D$, we define

$$\Gamma : \mathsf{Spec} \qquad\quad \vdash \Gamma^{\mathsf{R}}_{R} : (\gamma : \Gamma^{\mathsf{A}}_{F}) \to \Gamma^{\mathsf{D}}_{D}\ \gamma \to \mathbf{Type}$$

$$\begin{array}{ll}
C : \mathsf{Ctor}\ \Gamma & \vdash C^{\mathsf{R}}_{R} : (c : \ulcorner C^{\mathsf{A}}_{F}\ \gamma \urcorner) \to \ulcorner C^{\mathsf{D}}_{D}\ \gamma^{D}\ c \urcorner \to \mathsf{SType} \\
\Delta : \mathsf{Params}\ \Gamma & \vdash \Delta^{\mathsf{R}} : (\delta : \ulcorner \Delta^{\mathsf{A}}_{F}\ \gamma \urcorner) \to \ulcorner \Delta^{\mathsf{D}}_{D}\ \gamma^{D}\ \delta \urcorner \to \mathsf{SType} \\
T : \mathsf{Ty}\ \Gamma\ \Delta & \vdash T^{\mathsf{R}}\ : (x : \ulcorner T^{\mathsf{A}}_{F}\ \gamma\ \delta \urcorner) \to \ulcorner T^{\mathsf{D}}_{D}\ \gamma^{D}\ \delta^{D}\ x \urcorner \to \mathsf{SType}
\end{array}$$

We also lift the action of weakenings/substitutions and (constructor) terms to algebras, thus defining the following operators:

$$\begin{array}{ll}
w : \mathsf{Wk}\ \Gamma\ \Omega & \vdash w^{\mathsf{R}}_{R} : \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D} \to \Omega^{\mathsf{R}}_{R}\ (w^{\mathsf{A}}_{F}\ \gamma)(w^{\mathsf{D}}_{D}\ \gamma^{D}) \\
\sigma : \mathsf{Sub}\ \{\Gamma\}\ \Delta\ \nabla & \vdash \sigma^{\mathsf{R}}_{R}\ : \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D} \to \ulcorner \Delta^{\mathsf{R}}_{R}\ \delta\ \delta^{D} \urcorner \to \ulcorner \nabla^{\mathsf{R}}_{R}\ (\sigma^{\mathsf{A}}_{F}\ \delta)(\sigma^{\mathsf{D}}_{D}\ \delta^{D}) \urcorner \\
t : \mathsf{Tm}\ \Gamma\ \Delta\ T & \vdash t^{\mathsf{R}}_{R}\ \ : \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D} \to \ulcorner \Delta^{\mathsf{R}}_{R}\ \delta\ \delta^{D} \urcorner \to \ulcorner T^{\mathsf{R}}_{R}\ (t^{\mathsf{A}}_{F}\ \gamma\ \delta)(t^{\mathsf{D}}_{D}\ \gamma^{D}\ \delta^{D}) \urcorner \\
c : \mathsf{CTm}\ \Gamma\ C & \vdash c^{\mathsf{R}}_{R}\ \ : \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D} \to \ulcorner C^{\mathsf{R}}_{R}\ (c^{\mathsf{A}}_{F}\ \gamma)\ (c^{\mathsf{D}}_{D}\ \gamma^{D}) \urcorner
\end{array}$$

The full definition of these algebra operators can be found in Figure 12.1.

Similarly to Lemma 11.2.1, we can show that there is a more direct way to construct relation algebras by quantifying over constructor terms, rather than by induction on the specification.

Lemma 12.1.1. For any $\Gamma, R, \gamma, \gamma^{D}$, we have the following map:

$$[\forall\ \{C\}\ (c : \mathsf{CTm}\ \Gamma\ C) \to \ulcorner C^{\mathsf{R}}_{R}\ (c^{\mathsf{A}}\ \gamma)\ (c^{\mathsf{D}}\ \gamma^{D}) \urcorner] \quad \to \quad \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D} \qquad \blacksquare$$

Proof. By induction on Γ. □

The action of constructor term algebra operators on structures obtained via this mapping is the same as the function used to construct the structure:

Lemma 12.1.2. Let $\Gamma, R, \gamma, \gamma^{D}$, and

$$f : \forall\ \{C : \mathsf{Ctor}\ \Gamma\}\ (c : \mathsf{CTm}\ \Gamma\ C) \to \ulcorner C^{\mathsf{R}}_{R}\ (c^{\mathsf{A}}\ \gamma)\ (c^{\mathsf{D}}\ \gamma^{D}) \urcorner$$

If $\gamma^{R} : \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D}$ is obtained from f as per Lemma 12.1.1, then $c^{\mathsf{R}}\ \gamma^{R} = f\ c$. ■

Proof. By induction on c. □

Example 12.1.3. To check our work, we can verify that the expression $\Theta^{\mathsf{R}}\ \theta\ \theta^{D}\ (C_{R}, T_{R})$ computes to the structure shown in Example 12.1.2. ■

Definition 12.1.1. Given a specification $\Gamma : \mathsf{Spec}$, and algebras $(F, \gamma) : \mathsf{Alg}\ \Gamma$ and $(D, \gamma^{D}) : \mathsf{Alg}^{D}\ \Gamma\ (F, \gamma)$, a relation algebra $\mathsf{Alg}^{R}\ \Gamma\ (F, \gamma)\ (D, \gamma^{D})$ is a pair of types $R : \mathsf{Ca}^{R}\ F\ D$ and a term $\gamma^{R} : \Gamma^{\mathsf{R}}_{R}\ \gamma\ \gamma^{D}$. ■

$$\begin{aligned}
&\diamond^{\mathsf{R}}_R\ \_\ \_\ \_ :\equiv \mathbf{1}\\
&(\Gamma \rhd C)^{\mathsf{R}}_R\ (\gamma, c)\ (\gamma^D, c^D) :\equiv \Gamma^{\mathsf{R}}_R\ \gamma\ \gamma^D \times \ulcorner C^{\mathsf{R}}_R\ c\ c^D \urcorner
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{ctor}\ \Delta\ T)^{\mathsf{R}}_R\ \{\gamma\ \gamma^D\}\ c\ c^D :\equiv\\
&\quad \mathbf{\Pi}\ (\delta : \Delta^{\mathsf{A}}_F\ \gamma)\ \mathbf{\Pi}\ (\delta^D : \Delta^{\mathsf{D}}_D\ \gamma^D\ \delta)\\
&\qquad (\Delta^{\mathsf{R}}\ \delta\ \delta^D \Rightarrow T^{\mathsf{R}}\ (c \cdot \delta)\ (c \cdot \delta \cdot \delta^D))
\end{aligned}$$

$$\begin{aligned}
&\bullet^{\mathsf{R}}\ \delta\ \delta^D :\equiv \mathbf{1}\\
&(\Delta \rhd\!\rhd T)^{\mathsf{R}}\ \_\ \langle \delta^D, x^D \rangle :\equiv \Delta^{\mathsf{R}}\ \_\ \delta^D \times T^{\mathsf{R}}\ \_\ x^D\\
&(\Delta[w])^{\mathsf{R}}\ \{\gamma^D\} :\equiv \Delta^{\mathsf{R}}\ \{w^{\mathsf{D}}\ \gamma^D\}
\end{aligned}$$

$$\begin{aligned}
&(\mathsf{ext}\ A)^{\mathsf{R}}\ a\ a^D :\equiv \mathbf{1}\\
&(\pi\ A\ B)^{\mathsf{R}}\ f\ f^D :\equiv \mathbf{\Pi}(x : A)(B^{\mathsf{R}}\ (f \cdot x)\ (f^D \cdot x))\\
&(T[\sigma])^{\mathsf{R}}\ \{\delta^D\} :\equiv T^{\mathsf{R}}\ \{\sigma^{\mathsf{D}}\ \_\ \delta^D\}\\
&(T[w])^{\mathsf{R}}\ \{\gamma^D\} :\equiv T^{\mathsf{R}}\ \{w^{\mathsf{D}}\ \gamma^D\}\\
&(\mathsf{T}^1)^{\mathsf{R}}_R\ x\ x^D :\equiv \pi_1\ R\ x\ x^D\\
&(\mathsf{T}^2\ t)^{\mathsf{R}}_R\ \{\gamma\ \gamma^D\ \delta\ \delta^D\}\ x\ x^D :\equiv\\
&\quad \pi_2\ R\ (t^{\mathsf{A}}\ \gamma\ \delta)(t^{\mathsf{D}}\ \gamma^D\ \delta^D)\ x\ x^D
\end{aligned}$$

$$\begin{aligned}
&\mathsf{id}^{\mathsf{R}}\ \gamma^R :\equiv \gamma^R\\
&\mathsf{drop}^{\mathsf{R}}\ \gamma^R :\equiv \pi_1\ \gamma^R\\
&(w_1 \circ w2)^{\mathsf{R}}\ \gamma^R :\equiv {w_2}^{\mathsf{R}}\ ({w_1}^{\mathsf{R}}\ \gamma^R)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{id}^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv \delta^R\\
&\mathsf{drop}^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv \mathsf{fst}\ \delta^R\\
&(\mathsf{ext}\ \sigma\ t)^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv\\
&\quad \langle \sigma^{\mathsf{R}}\ \gamma^R\ \delta^R, t^{\mathsf{R}}\ \gamma^R\ \delta^R \rangle\\
&(\sigma \circ \tau)^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv \tau^{\mathsf{R}}\ \gamma^R\ (\sigma^{\mathsf{R}}\ \gamma^R\ \delta^R)\\
&(\sigma[w])^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv \sigma^{\mathsf{R}}\ (w^{\mathsf{R}}\ \gamma^R)\ \delta^R
\end{aligned}$$

$$\begin{aligned}
&\mathsf{vz}^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv \mathsf{snd}\ \delta^R\\
&(\mathsf{vs}\ t)^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv t^{\mathsf{R}}\ \gamma^R\ (\mathsf{fst}\ \delta^R)\\
&(\mathsf{ext}\ A\ x)^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv *\\
&(\mathsf{capp}\ c\ \sigma)^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv\\
&\quad c^{\mathsf{R}}\ \gamma^R \cdot \_ \cdot \_ \cdot \sigma^{\mathsf{R}}\ \gamma^R\ \delta^R\\
&(\mathsf{ap}\ t)^{\mathsf{R}}\ \{p\}\ \gamma^R\ \delta^R :\equiv\\
&\quad t^{\mathsf{R}}\ \gamma^R\ (\mathsf{fst}\ \delta^R) \cdot \mathsf{snd}\ \delta\\
&(\mathsf{lm}\ t)^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv \boldsymbol{\lambda} x.t^{\mathsf{R}}\ \gamma^R\ \langle \delta^R, * \rangle\\
&(t[\sigma])^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv t^{\mathsf{R}}\ \gamma^R\ (\sigma^{\mathsf{R}}\ \gamma^R\ \delta^R)\\
&(t[w])^{\mathsf{R}}\ \gamma^R\ \delta^R :\equiv t^{\mathsf{R}}\ (w^{\mathsf{R}}\ \gamma^R)\ \delta^R
\end{aligned}$$

$$\begin{aligned}
&\mathsf{cvz}^{\mathsf{R}}\ \gamma^R :\equiv \pi_2\ \gamma^R\\
&(\mathsf{cvs}\ c)^{\mathsf{R}}\ \gamma^R :\equiv c^{\mathsf{R}}\ (\pi_1\ \gamma^R)\\
&(c[w])^{\mathsf{R}}\ \gamma^R :\equiv c^{\mathsf{R}}\ (w^{\mathsf{R}}\ \gamma^R)
\end{aligned}$$

Figure 12.1: Relation algebra operators

12.1.2 Inversion principles

Relation algebras will be instrumental in defining eliminators for the IITs obtained via Σ-construction. However, not all relation algebras are suitable for this purpose. This was already hinted at in Section 8.1, where we required the eliminator relations for the Con/Ty IIT to support certain inversion principles. Predicate types and predicate algebras are subject to similar conditions, as already discussed in Section 11.2.2.

The required inversions on relation algebras are a direct generalization of those seen in the example of Section 8.1. These are of two kinds: one showing that relatedness of a compound implies relatedness of the components, and the another establishing an equational constraint on the displayed algebra indices.

Definition 12.1.2. Given a specification $\Gamma : \mathsf{Spec}$, and algebras $(F, \gamma) : \mathsf{Alg}\ \Gamma$ and $(D, \gamma^D) : \mathsf{Alg}^D\ \Gamma\ (F, \gamma)$, a relation algebra $(R, \gamma^R) : \mathsf{Alg}^R\ \Gamma\ (F, \gamma)\ (D, \gamma^D)$ supports IIT-inversions if there exist functions of the following types, where we name $(A, B) :\equiv F, (A^D, B^D) :\equiv D, (A^R, B^R) :\equiv R$:

$$\begin{aligned}
\textsf{rel-inv}\ &: (c : \mathsf{CTm}\ \Gamma\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1))(\delta : \ulcorner \Delta^{\mathsf{A}}_F\ \gamma \urcorner)\{a^D : \ulcorner A^D\ (c^{\mathsf{A}}_F\ \gamma \cdot \delta)\urcorner\} \\
&\to \ulcorner A^R\ (c^{\mathsf{A}}_F\ \gamma \cdot \delta)\ a^D \urcorner \ \to\ \ulcorner \boldsymbol{\Sigma}\ (\delta^D : \Delta^{\mathsf{D}}_D\ \gamma^D\ \delta)(\Delta^{\mathsf{R}}_R\ \delta\ \delta^D)\urcorner \\
\textsf{rel-ix-inv} &: (c : \mathsf{CTm}\ \Gamma\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1))(\delta : \ulcorner \Delta^{\mathsf{A}}_F\ \gamma \urcorner)\{a^D : \ulcorner A^D\ (c^{\mathsf{A}}_F\ \gamma \cdot \delta)\urcorner\} \\
&\quad (r : \ulcorner A^R\ (c^{\mathsf{A}}_F\ \gamma \cdot \delta)\ a^D \urcorner)\ \to\ \ulcorner a^D \approx c^{\mathsf{D}}_D\ \gamma^D \cdot \delta \cdot \mathsf{fst}\ (\textsf{rel-inv}\ c\ \delta\ r)\urcorner
\end{aligned}$$

such that rel-inv is well-behaved on constructor terms, that is, for any $c, \delta, \delta^D, \delta^R$,

$$\textsf{rel-inv}\ c\ \delta\ (c^{\mathsf{R}}_R\ \gamma^R \cdot \delta \cdot \delta^D \cdot \delta^R) = \langle \delta^D, \delta^R \rangle \qquad \blacksquare$$

In other words, a consequence of rel-inv is that for any proof of the relation $A^R\ (c^{\mathsf{A}}_F\ \gamma \cdot \delta)\ a^D$ involving a element of A originating from a constructor c applied to some arguments δ, we obtain that the arguments δ are also related to some displayed counterparts δ^D, i.e. we have $\Delta^{\mathsf{R}}_R\ \delta\ \delta^D$. Moreover, by rel-ix-inv we know the related element a^D must also arise from the same constructor, applied to the displayed arguments δ^D calculated by rel-inv.

Note that these inversion principles only pertain to constructors targeting the first of the two possible sorts of the specified IIT. Although it is possible to state similar inversions for constructors targeting the second sort, those are not needed for our developments.

12.1.3 Existence of the relations

In this section we show that the type $\mathsf{Alg}^R\ \Gamma\ \gamma\ \gamma^D$ is inhabited for any Γ, γ, γ^D, whenever γ is a Σ-algebra obtained from a section inductive erased algebra. Moreover, we show that the witnessing algebra has inversions as per Definition 12.1.2.

The proof uses section induction on the erased algebra to define the relations by recursion/large elimination. This construction generalizes what previously demonstrated in Section 8.1.1 to arbitrary specifications.

Let us fix a specification $\Omega : \mathsf{Spec}$, an assume an inductive erased algebra $(F_0, \omega_0) : \mathsf{Alg}_0\, \Omega$ and a predicate algebra $(F_1, \omega_1) : \mathsf{Alg}_1\, \Omega\, (F_0, \omega_0)$ with inversions. We thus have a Σ-algebra $(F, \omega) : \mathsf{Alg}\ \Omega$.

Suppose we are given a displayed algebra $(F^D, \omega^D) : \mathsf{Alg}^D\ \Omega\ \omega$. We want to define a relation algebra $(R, \omega^R) : \mathsf{Alg}^R\ \Omega\ \omega\ \omega^D$, and subsequently show that it enjoys the necessary inversion principles. The first step is to define two auxiliary functions by induction on the erased algebra. Recall Section 8.1.1, where such functions had the following signature:

$$\begin{aligned}
&\mathsf{Con}^R_{\mathsf{rec}} : (\Gamma_0 : \mathsf{Con}_0)(\Gamma_1 : \mathsf{Con}_1) \to \mathsf{Con}^D\ (\Gamma_0, \Gamma_1) \to \mathbf{Type}\\
&\mathsf{Ty}^R_{\mathsf{rec}}\ \ : (A_0 : \mathsf{Ty}_0)(A_1 : \mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0)(\Gamma^D : \mathsf{Con}^D\ \Gamma) \to \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1) \to \mathbf{Type}
\end{aligned}$$

We want to define equivalent target-level functions relative to (F_0, ω_0) and (F_1, ω_1). These should have the following signature, where $(A_1, B_1) :\equiv F_1$, $(A^D, B^D) :\equiv F^D$, and $(A, B) :\equiv F$:

$$\begin{aligned}
&A^R_{\mathsf{rec}} : (a_0 : \ulcorner A_0 \urcorner)(a_1 : \ulcorner A_1 \urcorner) \to \ulcorner A^D\ \langle a_0, a_1 \rangle \urcorner \to \mathsf{SType}\\
&B^R_{\mathsf{rec}} : (a : \ulcorner A \urcorner)(b_0 : \ulcorner B_0 \urcorner)(b_1 : \ulcorner B_1\ (\mathsf{fst}\ a)\ b_0 \urcorner)(a^D : \ulcorner A^D\ a \urcorner)\\
&\qquad\quad \to \ulcorner B^D\ a^D\ \langle b_0, b_1 \rangle \urcorner \to \mathsf{SType}
\end{aligned}$$

We can express the types of these functions as displayed erased carriers $\mathsf{R}_{\mathsf{rec}}$ over F_0:

$$\begin{aligned}
&\mathsf{R}_{\mathsf{rec}} : \mathsf{Ca}^D_0\ F_0\\
&\mathsf{R}_{\mathsf{rec}} :\equiv\\
&\quad (\ \lambda a_0.\mathbf{\Pi}(a_1 : A_1\ a_0)(A^D\ \langle a_0, a_1 \rangle \Rightarrow \mathcal{U})\\
&\quad ,\ \lambda b_0.\mathbf{\Pi}(a : A)\mathbf{\Pi}(b_1 : B_1\ (\mathsf{fst}\ a)\ b_0)\mathbf{\Pi}(a^D : A^D\ a)(B^D\ a\ a^D\ \langle b_0, b_1 \rangle \Rightarrow \mathcal{U})\ )
\end{aligned}$$

so that defining A^R_{rec} and B^R_{rec} is equivalent to defining a structure $f_0 : \mathsf{Map}_0\ F_0\ \mathsf{R}_{\mathsf{rec}}$. We construct such f_0 by induction on the erased types A_0, B_0. That is, we define a term ω^D_0 so that the pair $(\mathsf{R}_{\mathsf{rec}}, \omega^D_0) : \mathsf{Alg}^D_0\ \Omega\ (F_0, \omega_0)$ is a displayed erased algebra providing the motives and methods of the induction. We then take f_0 to be the induced section map.

Expanding the type $\Omega^{\mathsf{ED}}_{\mathsf{R}_{\mathsf{rec}}}\ \omega_0$ of our goal ω^D_0, we get

$$\omega^D_0 : \forall\{C\}(c : \mathsf{CTm}\ \Omega\ C) \to \ulcorner C^{\mathsf{ED}}_{\mathsf{R}_{\mathsf{rec}}}\ (c^{\mathsf{E}}_{F_0}\ \omega_0) \urcorner$$

The constructor C is of the form $C \equiv \mathsf{ctor}\ \Delta\ B$ for some Δ, B. We proceed by case analysis on B. Both cases are a straightforward generalization of the concrete example shown in Section 8.1.1. Recall the definition of $\mathsf{Con}^R_{\mathsf{rec}}$ on $\triangleright_0$:

$$\begin{aligned}
&\mathsf{Con}^R_{\mathsf{rec}}\ (\Gamma_0 \rhd_0 A_0)\ p\ x^D :\equiv \\
&\quad (\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1)) \times (A^D : \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1)) \times \\
&\quad \mathsf{Con}^R_{\mathsf{rec}}\ \Gamma_0\ \Gamma_1\ \Gamma^D \times \mathsf{Ty}^R_{\mathsf{rec}}\ A_0\ A_1\ \Gamma^D\ A^D \times \\
&\quad x^D = \Gamma^D \rhd^D A^D
\end{aligned}$$

where $\Gamma_1 :\equiv \mathsf{inv}\text{-}\rhd\text{-}\mathsf{Con}\ p$ and $A_1 :\equiv \mathsf{inv}\text{-}\rhd\text{-}\mathsf{Ty}\ p$ are obtained by inversion. For each argument Γ_0, A_0 of the constructor we required a corresponding displayed term Γ^D_0, A^D_0 and a proof of relatedness, as well as an equation stating that the displayed index is obtained from these components via the displayed equivalent of the $\rhd$ constructor. The first case of ω^D_0 is analogous:

$$\omega^D_0\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1)\ c :\equiv \boldsymbol{\lambda}\, \delta_0 \,.\, \boldsymbol{\lambda}\, \delta^D_0 \,.\, \boldsymbol{\lambda}\, a_1 \,.\, \boldsymbol{\lambda}\, a^D \,.\, ?$$

We are given a constructor term $c : \mathsf{CTm}\ \Omega\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1)$ and arguments δ_0, as well as an algebra δ^D_0 encapsulating terms of the inductive hypothesis for all arguments in δ_0. The goal is to construct a small type, that should correspond to $A^R_{\mathsf{rec}}\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_0)\ a_1\ a^D$. By inversion on a_1 we obtain $\delta_1 : \Delta^{\mathsf{W}}\ \omega_0\ \delta_0$, so the goal can essentially be rephrased as defining the relation between $c^{\mathsf{A}}_F\, \omega \cdot \delta$ and a^D, where $\delta :\equiv \langle \delta_0, \delta_1 \rangle$. Following $\mathsf{Con}^R_{\mathsf{rec}}$, the result expression should be a dependent product of three components: (1) a tuple δ^D of displayed algebras for each parameter of c, (2) an iterated product eliminator relations obtained by inductive hypothesis and applied pair-wise to δ and δ^D, and (3) an equation stating that a^D arises as a constructor term algebra of c from δ and δ^D.

We represent the tuple of displayed algebras as a single parameter algebra $\delta^D : \Delta^{\mathsf{D}}_D\ \omega^D\ \delta$, and state the constraint on a^D as an equation with the displayed constructor term algebra:

$$\begin{aligned}
&\omega^D_0\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1)\ c :\equiv \boldsymbol{\lambda}\, \delta_0 \,.\, \boldsymbol{\lambda}\, \delta^D_0 \,.\, \boldsymbol{\lambda}\, a_1 \,.\, \boldsymbol{\lambda}\, a^D \,. \\
&\quad \boldsymbol{\Sigma}\ (\delta^D : \Delta^{\mathsf{D}}_D\ \omega^D\ \delta)(\ ?\ \times a^D \approx c^{\mathsf{D}}_D\ \omega^D \cdot \delta \cdot \delta^D)
\end{aligned}$$

We are left to state the proofs of relatedness, one for each type in Δ. The inductive hypotheses are already provided via the algebra δ^D_0, which can be thought of as a list of relations constructed by inductive-hypothesis, and already partially applied to a component of the list δ. We rely on an auxiliary *inductive hypotheses operator* $\_^{\mathsf{ih}}$, which function is similar to the homonymous operator defined in Section 11.2.3. This operator takes a list δ^D_0 of partially-applied inductive hypotheses, i.e. type families, and applies each of them to the corresponding arguments in δ^D. The result of this operation is a small type: the iterated product of all these type families applied to their corresponding argument in δ^D.

$$(\mathsf{T}^1)^{\mathsf{ih}}_w\; x^D\; x^D_0 :\equiv x^D_0 \cdot \_ \cdot x^D$$
$$(\mathsf{T}^2\; t)^{\mathsf{ih}}_w\; \{\Delta\; \delta\; \delta^D\; x\}\; x^D\; x^D_0 :\equiv x^D_0 \cdot (t^{\mathsf{A}}\; (w^{\mathsf{A}}\; \omega)\; \delta) \cdot \mathsf{snd}\, x \cdot (t^{\mathsf{D}}\; (w^{\mathsf{D}}\; \gamma^D)\; \delta^D) \cdot x^D$$
$$(\mathsf{ext}\; A)^{\mathsf{ih}}_w\; a^D\; a^D_0 :\equiv \mathbf{1}$$
$$(\pi\; A\; B)^{\mathsf{ih}}_w\; f^D\; f^D_0 :\equiv \mathbf{\Pi}\; (a : A)\; (B^{\mathsf{ih}}_w\; (f^D \cdot a)\; (f^D_0 \cdot a))$$
$$(A[\sigma])^{\mathsf{ih}}_w :\equiv A^{\mathsf{ih}}_w$$
$$(A[w_2])^{\mathsf{ih}}_{w_1} :\equiv A^{\mathsf{ih}}_{(w_1\, \circ\, w_2)}$$

$$\bullet^{\mathsf{ih}}_w\;\; \delta^D\; \delta^D_0 :\equiv \mathbf{1}$$
$$(\Delta \rhd\!\rhd A)^{\mathsf{ih}}_w\; \langle \delta^D, a^D \rangle\; \langle \delta^D_0, a^D_0 \rangle :\equiv \Delta^{\mathsf{ih}}_w\; \delta^D\; \delta^D_0 \times A^{\mathsf{ih}}_w\; a^D\; a^D_0$$
$$(\Delta[w_2])^{\mathsf{ih}}_{w_1} :\equiv \Delta^{\mathsf{ih}}_{(w_1\, \circ\, w_2)}$$

Figure 12.2: Inductive hypotheses operators for eliminator relations

$$\begin{array}{ll} \Delta : \mathsf{Params}\;\Gamma & \vdash \Delta^{\mathsf{ih}}_w : \ulcorner \Delta^{\mathsf{D}} (w^{\mathsf{D}}\, \omega^D)\; \delta \urcorner \to \ulcorner \Delta^{\boxplus\mathsf{D}}\; \delta_{\underline{0}} \urcorner \to \mathsf{SType} \\ A : \mathsf{Ty}\;\Gamma\;\Delta & \vdash A^{\mathsf{ih}}_w : \ulcorner A^{\mathsf{D}}\; (w^{\mathsf{D}}\, \omega^D)\; \delta^D\; a \urcorner \to \ulcorner A^{\boxplus\mathsf{D}}\; a_{\underline{0}} \urcorner \to \mathsf{SType} \end{array}$$

We define $\_^{\mathsf{ih}}$ on parameter and type specifications by mutual induction (Figure 12.2.) Note that we parameterize each of them by a weakening $w : \mathsf{Wk}\,\Omega\,\Gamma$, for variable Γ, which we omit mentioning when the identity. Moreover, we rely on Lemma 11.3.4.2 to typecheck $\mathsf{snd}\, x$ as an argument of x^D_0 in the $(\mathsf{T}^2\; t)^{\mathsf{ih}}_w$ clause.

We finally conclude the first clause of ω^D_0:

$$\begin{array}{l} \omega^D_0\; (\mathsf{ctor}\; \Delta\; \mathsf{T}^1)\; c :\equiv \boldsymbol{\lambda}\, \delta_0\, .\, \boldsymbol{\lambda}\, \delta^D_0\, .\, \boldsymbol{\lambda}\, a_1\, .\, \boldsymbol{\lambda}\, a^D\, . \\ \quad \mathbf{\Sigma}\; (\delta^D : \Delta^{\mathsf{D}}_D\; \omega^D\; \delta)(\Delta^{\mathsf{ih}}\; \delta^D\; \delta^D_0 \times a^D \approx c^{\mathsf{D}}_D\; \omega^D \cdot \_ \cdot \delta^D) \end{array}$$

To understand the second case of ω^D_0, recall how we defined $\mathsf{Ty}^R_{\mathsf{rec}}$ on the constructor ι_0:

$$\begin{array}{l} \mathsf{Ty}^R_{\mathsf{rec}}\; (\iota_0\; \Gamma_0)\; p :\equiv \mathsf{transp}\; (T\; (\iota_0\; \Gamma_0))\; (\mathsf{sym}\; (\mathsf{inv}\text{-}\iota_1\; p))\; h\; \_\; p \\ \quad \mathbf{where} \\ \qquad h : T\; (\iota_0\; \Gamma_0)\; \Gamma_0 \end{array}$$

```
$\omega_0^D\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1)\ c :\equiv$
  $\boldsymbol{\lambda}\,\delta_0\,.\,\boldsymbol{\lambda}\,\delta_0^D\,.\,\boldsymbol{\lambda}\,a_1\,.\,\boldsymbol{\lambda}\,a^D\,.$
    $\mathbf{let}\ \delta :\equiv \langle \delta_0, \mathsf{pred\text{-}inv}\ c\ \delta_0\ a_1\rangle$
    $\mathbf{in}\ \boldsymbol{\Sigma}\ (\delta^D : \Delta^{\mathsf{D}}_D\ \omega^D\ \delta)$
        $(\Delta^{\mathsf{ih}}\ \delta^D\ \delta_0^D$
        $\times\ \mathsf{Id}\ \_\ a^D\ (c^{\mathsf{D}}_D\ \omega^D \cdot \delta \cdot \delta^D))$

$T : \ulcorner B_0 \urcorner \to \ulcorner A_0 \urcorner \to \mathsf{Type}$
$T\ b_0\ a_0 :\equiv$
  $\boldsymbol{\Pi}(a_1 : A_1\ a_0)\ \boldsymbol{\Pi}(b_1 : B_1\ a_0\ b_0)$
    $\boldsymbol{\Pi}(a^D : A^D\ \langle a_0, a_1\rangle)$
      $(B^D\ \_\ a^D\ \langle b_0, b_1\rangle \Rightarrow \mathcal{U})$

$\omega_0^D\ (\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t))\ c :\equiv \boldsymbol{\lambda}\,\delta_0\,.\,\boldsymbol{\lambda}\,\delta_0^D\,.\,\boldsymbol{\lambda} a\,.\,\boldsymbol{\lambda} b_1\,.$
  $\mathbf{let}$
    $e :\equiv \mathsf{pred\text{-}ix\text{-}inv}\ c\ \delta_0\ (\mathsf{fst}\ a)\ b_1$
    $h : T\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_0)\ (t^{\mathsf{E}}\ \omega\ \delta_0)$
    $h :\equiv \boldsymbol{\lambda}\,a_1\,.\,\boldsymbol{\lambda}\,b_1\,.\,\boldsymbol{\lambda}\,a^D\,.\,\boldsymbol{\lambda}\,b^D\,.$
      $\mathbf{let}\ \delta_1 :\equiv \mathsf{pred\text{-}inv}\ c\ \delta_0\ b_1$
      $\mathbf{in}\ \boldsymbol{\Sigma}\ (\delta^D : \Delta^{\mathsf{D}}\ \omega^D\ \langle \delta_0, \delta_1\rangle)$
        $(\Delta^{\mathsf{ih}}\ \delta^D\ \delta_0^D\ \times$
          $\boldsymbol{\Sigma}\ (q : \mathsf{Id}\ \_\ a^D\ (t^{\mathsf{D}}\ \omega^D\ \delta^D))$
            $(\mathsf{Id}\ \_\ (\mathsf{Transp}\ (\lambda z.\mathsf{El}_{\mathcal{U}}\ (B^D\ \_\ z\ \_))$
                                  $q\ b^D)$
                          $(c^{\mathsf{D}}\ \omega^D \cdot \_ \cdot \delta^D)))$
  $\mathbf{in}\ \mathsf{Transp}\ (T\ \_)\ (\mathsf{Sym}\ e)\ h \cdot \mathsf{snd}\ a \cdot b_1$
```

Figure 12.3: Erased displayed algebra for recursively-defined eliminator relations

$$h\ \Gamma_1\ p\ x^D\ y^D :\equiv (\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1)) \times \mathsf{Con}^R_{\mathsf{rec}}\ \Gamma_0\ \Gamma_1\ \Gamma^D \times$$
$$(q : \Gamma^D = x^D) \times \iota^D\ \Gamma^D \overset{q}{=}\ y^D$$

where $T\,A_0\,\Gamma_0 :\equiv \forall\,\Gamma_1\,A_1\,(\Gamma^D : \mathsf{Con}^D\ (\Gamma_0, \Gamma_1)) \to \mathsf{Ty}^D\ \Gamma^D\ (A_0, A_1) \to \mathbf{Type}$. The main difference with $\mathsf{Con}^R_{\mathsf{rec}}$ was that we needed to transport along the equational constraint obtained from the well-formedness predicate in order to be able to type-check what is otherwise just like one of the $\mathsf{Con}^R_{\mathsf{rec}}$ cases.

The corresponding case for $\omega_0^D\ (\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t))$ follows the same structure.

$$\omega_0^D\ (\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t))\ c :\equiv \boldsymbol{\lambda}\,\delta_0\,.\,\boldsymbol{\lambda}\,\delta_0^D\,.\,\boldsymbol{\lambda} a\,.\,\boldsymbol{\lambda} b_1\,.\,\boldsymbol{\lambda}\,a^D\,.\,\boldsymbol{\lambda}\,b^D\,.\,?$$

The goal is again a small type in $\mathcal{U}$, this time corresponding to

$$B^R_{\mathsf{rec}}\ a\ (c^{\mathsf{E}}\ \omega_0\ \delta_0)\ b_1\ a^D\ b^D$$

Before defining this, we want to expose the fact that a arises from term algebras of t. By inversion on b_1 and proof-irrelevance, we obtain $\mathsf{fst}\ a \approx t^{\mathsf{E}}\ \omega_0\ \delta_0$ and transport over it.

```
$\omega_0^D\ (\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t))\ c :\equiv \boldsymbol{\lambda}\,\delta_0\,.\,\boldsymbol{\lambda}\,\delta_0^D\,.\,\boldsymbol{\lambda} a\,.\,\boldsymbol{\lambda} b_1\,.$
    $\mathbf{let}\ h : T\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_0)\ (t^{\mathsf{E}}\ \omega\ \delta_0)$
```

$$h :\equiv \boldsymbol{\lambda}\, a_1 \,.\, \boldsymbol{\lambda}\, b_1 \,.\, \boldsymbol{\lambda}\, a^D \,.\, \boldsymbol{\lambda}\, b^D \,.\ ?$$
$$\textbf{in}\ \mathsf{Transp}\ (T\ \_)\ (\mathsf{Sym}\ (\mathsf{pred\text{-}ix\text{-}inv}\ c\ \delta_0\ (\mathsf{fst}\ a)\ b_1))\ h \cdot \mathsf{snd}\ a \cdot b_1$$

The rest of the type expression is like the the first clause of ω_0^D, with an additional equational constraint on b^D. We give the full, precise definition of ω_0^D in Figure 12.3.

Because (F_0, ω_0) is an inductive algebra, we obtain functions $f_0 : \mathsf{Map}_0\, F_0\, \mathsf{R}_{\mathsf{rec}}$ and section structure $s_0 : \Omega^{\mathsf{ES}}_{f_0}\ \omega_0\ \omega_0^D$. We easily define the auxiliary relations, as well as a pair $\mathsf{R} : \mathsf{Ca}^R\ (A, B)\ F^D$.

$$A^R_{\mathsf{rec}} :\equiv \lambda a_0\ a_1\ a^D . \pi_1\ f_0 \cdot a_0 \cdot a_1 \cdot a^D$$
$$B^R_{\mathsf{rec}} :\equiv \lambda a\ a^D\ b_0\ b_1\ b^D . \pi_2\ f_0 \cdot b_0 \cdot a \cdot b_1 \cdot a^D \cdot b^D$$
$$\pi_1\ \mathsf{R} :\equiv \lambda a\ a^D\ .\ A^R_{\mathsf{rec}}\ (\mathsf{fst}\ a)\ (\mathsf{snd}\ a)\ a^D$$
$$\pi_2\ \mathsf{R} :\equiv \lambda a\ a^D\ b\ b^D\ .\ B^R_{\mathsf{rec}}\ a\ a^D\ (\mathsf{fst}\ b)\ (\mathsf{snd}\ b)\ b^D$$

We now show that the relations R just defined constitute a relation algebra. We begin with some equational lemmas relating the auxiliary inductive hypotheses operators to the section we just defined.

Lemma 12.1.3. For all $\Gamma : \mathsf{Spec}, w : \mathsf{Wk}\ \Omega\ \Gamma, \Delta : \mathsf{Params}\ \Gamma$, and algebras $\delta : \ulcorner\Delta^{\mathsf{A}}\ (w^{\mathsf{A}}\ \omega)\urcorner$, $\delta^D : \ulcorner\Delta^{\mathsf{D}}\ (w^{\mathsf{D}}\ \omega^D)\ \delta\urcorner$, the following hold

1. $\Delta^{\mathsf{ih}}_w\ \delta^D\ (\Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}}) = \Delta^{\mathsf{R}}_{\mathsf{R}}\ \delta\ \delta^D$
2. $T^{\mathsf{ih}}_w\ x^D\ (T^{\mathsf{ES}}_{f_0}\ x_{\underline{0}}) = T^{\mathsf{R}}_{\mathsf{R}}\ x\ x^D$, for $T : \mathsf{Ty}\ \Gamma\ \Delta$, $x : \ulcorner T^{\mathsf{A}}\ \delta\urcorner$, $x^D : \ulcorner T^{\mathsf{D}}\ \delta^D\ x\urcorner$

■

Proof. Both points proved by mutual induction on Δ, T respectively. □

Showing that R is a relation algebra amounts to constructing a term ω^R of type $\Omega^{\mathsf{R}}_{\mathsf{R}}\ \omega\ \omega^D$. We apply Lemma 12.1.1 and reduce this to

$$\omega^R : \forall\{C\}(c : \mathsf{CTm}\ \Omega\ C) \to \ulcorner C^{\mathsf{R}}_{\mathsf{R}}\ (c^{\mathsf{A}}\ \omega)\ (c^{\mathsf{D}}\ \omega^D)\urcorner$$

Assuming $C \equiv \mathsf{ctor}\ \Delta\ B$, we proceed by case analysis on B.

1. Case $B \equiv \mathsf{T}^1$. The goal is a function from $\delta : \ulcorner\Delta^{\mathsf{A}}\ \omega\urcorner, \delta^D : \ulcorner\Delta^{\mathsf{D}}\ \omega^D\ \delta\urcorner, \delta^R : \ulcorner\Delta^{\mathsf{R}}_{\mathsf{R}}\ \delta\ \delta^D\urcorner$, to

$$\ulcorner A^R_{\mathsf{rec}}\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_{\underline{0}})(c^{\mathsf{W}}\ \omega_1 \cdot \delta_{\underline{0}} \cdot \delta_{\underline{1}})(c^{\mathsf{D}}\ \omega^D \cdot \delta \cdot \delta^D)\urcorner$$

By Lemma 11.1.2, we turn this into

$$\ulcorner \omega_0^D\ c \cdot \delta_{\underline{0}} \cdot \Delta_{f_0}^{\mathsf{ES}}\ \delta_{\underline{0}} \cdot (c^{\mathsf{W}}\ \omega_1 \cdot \delta_{\underline{0}} \cdot \delta_{\underline{1}}) \cdot (c^{\mathsf{D}}\ \omega^D \cdot \delta \cdot \delta^D)\urcorner$$

By Lemma 12.1.3.1, this type reduces to a target-level product type of a displayed parameter algebra, a relation parameter algebra, and a reflexive equation, that we solve by $\langle \delta^D, \langle \delta^R, \mathsf{Refl}\ \_\rangle\rangle$.

2. Case $B \equiv \mathsf{T}^2\ t$ for some $t : \mathsf{Tm}\ \Omega\ \Delta\ \mathsf{T}^1$. This case is just like the previous, except we end up with two reflexive equations in the goal type, which we solve with two uses of Refl.

This concludes the construction of a relation algebra $\mathsf{Alg}^R\ \Omega\ (F, \omega)\ (D, \omega^D)$. We now finally show that this algebra enjoys IIT-inversion as stated in Definition 12.1.2.

Lemma 12.1.4. The relation algebra (R, ω^R) has IIT-inversions. ■

Proof. By applying Lemma 11.1.2 and Lemma 12.1.3 to the involved goal types, defining the inversions becomes a matter of projecting out the relevant components from the proof of relatedness itself. In particular:

$$\begin{aligned}&\mathsf{rel\text{-}ix} :\equiv \lambda c\ p\ r\ .\ \langle \pi_1\ r, \pi_1\ (\pi_2\ r)\rangle\\&\mathsf{rel\text{-}ix\text{-}inv} :\equiv \lambda c\ p\ r\ .\ \pi_2\ (\pi_2\ r)\end{aligned}$$ □

We wrap everything up by noting that the IIT specification and algebras used throughout this section were fixed but completely arbitrary, thus the whole construction can be read as a demonstration of how to define a relation algebra with inversions given *any* specification and corresponding pair of an IIT Σ-algebra and displayed algebra.

Theorem 12.1.1. Given a specification Γ, inductive erased algebra alg_0, predicate algebra $alg_1 : \mathsf{Alg}_1\ \Gamma\ alg_0$ with IIT-inversions, and a displayed algebra $alg^D : \mathsf{Alg}^D\ \Gamma\ alg$ of the Σ-algebra $alg :\equiv \mathsf{sigma}\ alg_0\ alg_1$, there exists a relation algebra $alg^R : \mathsf{Alg}^R\ \Gamma\ alg\ alg^D$ which enjoys IIT-inversions. ■

Proof. This whole section provides an extended proof. □

Auxiliary algebra operators

Displayed algebras and relation algebras will show up together in most of our developments from now on. We therefore introduce some abbreviations and notational conventions aimed at improving the ergonomics when handling these objects.

We introduce families of types $\_^{\mathsf{L}}$ merely standing for pairs of a displayed algebra structure and a relation algebra structure over it. Given $F : \mathsf{Ca}, D : \mathsf{Ca}^{\mathsf{D}}\ F, R : \mathsf{Ca}^{R}\ F\ D$,

$$\begin{array}{ll}
\Gamma : \mathsf{Spec} & \vdash \Gamma^{\mathsf{L}}_{D,R} : \Gamma^{\mathsf{A}}_{F} \to \mathbf{Type} \\
\Delta : \mathsf{Params}\ \Gamma & \vdash \Delta^{\mathsf{L}}_{D,R} : \Gamma^{\mathsf{D}}_{D}\ \gamma \to \ulcorner \Delta^{\mathsf{A}}_{F}\ \gamma \urcorner \to \mathsf{SType} \\
T : \mathsf{Ty}\ \Gamma\ \Delta & \vdash T^{\mathsf{L}}_{D,R} : (\gamma^{D} : \Gamma^{\mathsf{D}}_{D}\ \gamma) \to \ulcorner \Delta^{\mathsf{D}}_{D}\ \gamma^{D}\ \delta \urcorner \to \ulcorner T^{\mathsf{A}}_{F}\ \delta \urcorner \to \mathsf{SType} \\
C : \mathsf{Ctor}\ \Gamma & \vdash C^{\mathsf{L}}_{D,R} : \Gamma^{\mathsf{D}}_{D}\ \gamma \to \ulcorner C^{\mathsf{A}}_{F}\ \gamma \urcorner \to \mathsf{SType}
\end{array}$$

All these families are defined as simple pairings.

$$\begin{array}{ll}
\Gamma^{\mathsf{L}}_{D,R}\ \gamma & :\equiv \Sigma\,(\Gamma^{\mathsf{D}}_{D}\ \gamma)\ \Gamma^{\mathsf{R}}_{R} \\
\Delta^{\mathsf{L}}_{D,R}\ \gamma^{D}\ \delta & :\equiv \mathbf{\Sigma}\,(\Delta^{\mathsf{D}}_{D}\ \gamma^{D}\ \delta)(\Gamma^{\mathsf{R}}_{R}\ \delta) \\
A^{\mathsf{L}}_{D,R}\ \gamma^{D}\ \delta^{D}\ a & :\equiv \mathbf{\Sigma}(A^{\mathsf{D}}_{D}\ \gamma^{D}\ \delta^{D}\ a)(A^{\mathsf{R}}_{R}\ a) \\
C^{\mathsf{L}}_{D,R}\ \gamma^{D}\ c & :\equiv \mathbf{\Sigma}(C^{\mathsf{D}}_{D}\ \gamma^{D})(C^{\mathsf{R}}_{R}\ c)
\end{array}$$

Given $\delta^{\mathrm{L}} : \ulcorner \Delta^{\mathsf{L}}\ \gamma^{D}\ \delta \urcorner$, we write $\delta^{\underline{D}}, \delta^{\underline{R}}$ for the first and second projection of δ^{L} respectively. We do the same for algebras of inner types and specification contexts.

We can easily lift the algebra operators induced by weakenings, substitutions, and (constructor) terms:

$$\begin{array}{ll}
c : \mathsf{CTm}\ \Gamma\ C & \vdash c^{\mathsf{L}}_{D,R} : (\gamma^{\mathrm{L}} : \Gamma^{\mathsf{L}}_{D,R}\ \gamma) \to \ulcorner C^{\mathsf{L}}_{D,R} \urcorner \\
\sigma : \mathsf{Sub}\ \Delta\ \nabla & \vdash \sigma^{\mathsf{L}}_{D,R} : (\gamma^{\mathrm{L}} : \Gamma^{\mathsf{L}}_{D,R}\ \gamma) \to \ulcorner \Delta^{\mathsf{L}}_{D,R}\ (\gamma^{\underline{R}})\ \delta \urcorner \\
 & \qquad \to \ulcorner \nabla^{\mathsf{L}}_{D,R}\ (\gamma^{\underline{R}})\ (\sigma^{\mathsf{A}}\ \delta) \urcorner \\
t : \mathsf{Tm}\ \Gamma\ \Delta\ A & \vdash t^{\mathsf{L}}_{D,R} : (\gamma^{\mathrm{L}} : \Gamma^{\mathsf{L}}_{D,R}\ \gamma)(\delta^{\mathrm{L}} : \ulcorner \Delta^{\mathsf{L}}_{D,R}\ (\gamma^{\underline{R}})\ \delta \urcorner) \\
 & \qquad \to \ulcorner A^{\mathsf{L}}_{D,R}\ (\gamma^{\underline{R}})\ (\delta^{\underline{R}})\ (t^{\mathsf{A}}\ \delta) \urcorner \\
w : \mathsf{Wk}\ \Gamma\ \Omega & \vdash w^{\mathsf{L}}_{D,R} : \Gamma^{\mathsf{L}}_{D,R}\ \gamma \to \Omega^{\mathsf{L}}_{D,R}\ (w^{\mathsf{A}}\ \gamma)
\end{array}$$

$$\begin{array}{l}
c^{\mathsf{L}}\ (\gamma^{D}, \gamma^{R}) :\equiv \langle c^{\mathsf{D}}\ \gamma^{D}, c^{\mathsf{R}}\ \gamma^{R} \rangle \\
\sigma^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ \langle \delta^{D}, \delta^{R} \rangle :\equiv \langle \sigma^{\mathsf{D}}\ \delta^{D}, \sigma^{\mathsf{R}}\ \gamma^{\underline{R}}\ \delta^{R} \rangle \\
t^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ \langle \delta^{D}, \delta^{R} \rangle :\equiv \langle t^{\mathsf{D}}\ \delta^{D}, t^{\mathsf{R}}\ \gamma^{\underline{R}}\ \delta^{R} \rangle \\
w^{\mathsf{L}}\ \gamma^{\mathrm{L}} :\equiv w^{\mathsf{D}}\ \gamma^{\underline{D}}, w^{\mathsf{R}}\ \gamma^{\underline{R}}
\end{array}$$

There is an isomorphism

$$\ulcorner(\Delta \rhd\!\rhd A)^{\mathrm{L}}\ \gamma^{D}\ \langle \delta, a\rangle\urcorner \cong \ulcorner\mathbf{\Sigma}\ (\delta^{\mathrm{L}} : \Delta^{\mathrm{L}}\ \gamma^{D}\ \delta)\ (A^{\mathrm{L}}\ \gamma^{D}\ \delta^{\underline{D}}\ a)\urcorner$$

and a map

$$\ulcorner(\mathsf{ctor}\ \Delta\ B)^{\mathrm{L}}\ \gamma^{D}\ c\urcorner \to \ulcorner\mathbf{\Pi}\ (\delta^{\mathrm{L}} : \Delta^{\mathrm{L}}\ \gamma^{D}\ \delta)(B^{\mathrm{L}}\ \gamma^{D}\ (c \cdot \delta))\urcorner$$

defined in the obvious way. We will implicitly apply both of them as coercions throughout the rest of the chapter.

12.2 Left-totality of the relations

Left-totality of the eliminator relations is a crucial component of our reduction method. As we plan to use the eliminator relations constructed in Section 12.1.3 to encode the IIT eliminators, it follows that we need to first establish their left-totality. We will do so in the sections that follow.

12.2.1 Specifying left-totality

Let us first figure out how to specify left-totality for target-level eliminator algebras. As a reference, we go back once again to Section 8.1, where we gave proofs of left-totality of the eliminator relations as terms of the following types, constructed by mutual induction on the erased types $\mathsf{Con}_0, \mathsf{Ty}_0$.

$$\begin{aligned}
&\mathsf{Con}^{\exists} : \forall\ \Gamma_0\ \Gamma_1 \to \Sigma\ (\Gamma^{D} : \mathsf{Con}^{D}\ (\Gamma_0, \Gamma_1))(\mathsf{Con}^{R}\ (\Gamma_0, \Gamma_1)\ \Gamma^{D})\\
&\mathsf{Ty}^{\exists} : \forall\{\Gamma\ \Gamma^{D}\}(A_0 : \mathsf{Ty}_0)(A_1 : \mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0) \to \mathsf{Con}^{R}\ \Gamma\ \Gamma^{D}\\
&\qquad\quad \to \Sigma(A^{D} : \mathsf{Ty}^{D}\ \Gamma\ (A_0, A_1)\ \Gamma^{D})(\mathsf{Ty}^{R}\ (A_0, A_1)\ \Gamma^{D}\ A^{D})
\end{aligned}$$

We can try to replicate these statements as the target-level signatures. Let us fix some arbitrary collection of erased types, predicates, displayed families and eliminator relations:

$$F_0 : \mathsf{Ca}_0, \quad F_1 : \mathsf{Ca}_1\ F_0, \quad D : \mathsf{Ca}^{\mathsf{D}}\ F, \quad R : \mathsf{Ca}^{R}\ F\ D$$

where we name $(A_0, B_0) :\equiv F_0$, $(A_1, B_1) :\equiv F_1$, $(A, B) \equiv F :\equiv (\mathsf{Ca}_{\Sigma}\ F_0\ F_1)$, $(A^{D}, B^{D}) :\equiv D$, $(A^{R}, B^{R}) :\equiv R$. We abbreviate $A^{\mathrm{L}}\, a :\equiv \mathbf{\Sigma}(a^{D} : A^{D})(A^{R}\, a)$, and $B^{\mathrm{L}}\, a\, a^{D}\, b :\equiv \mathbf{\Sigma}(b^{D} : B^{D}\, a\, a^{D})(B^{R}\, a\, a^{D}\, b)$.

To say that A^{R}, B^{R} have the left-totality property is to have two terms of the following types:

$$A^{\exists} : \ulcorner\mathbf{\Pi}(a : A)\ (A^{\mathrm{L}}\, a)\urcorner$$

$$B^{\exists} : \ulcorner \mathbf{\Pi}(a : A)\, \mathbf{\Pi}(a^{\mathrm{L}} : A^{\mathrm{L}}\, a)\, \mathbf{\Pi}(b : B\, a)(B^{\mathrm{L}}\, a\, a^{\underline{D}}\, b) \urcorner$$

We can express the types of these functions as displayed erased carriers Ex_0^D over the erased types F_0:

Definition 12.2.1.

$$\begin{aligned}
&\mathsf{Ex}_0^D \quad : \mathsf{Ca}_0^D\ F_0 \\
&\mathsf{Ex}_0^D :\equiv (\ \lambda a_0.\mathbf{\Pi}(a_1 : A_1\ a_0)(A^{\mathrm{L}}\,\langle a_0, a_1\rangle) \\
&\qquad\quad ,\ \lambda b_0.\mathbf{\Pi}(a : A)\mathbf{\Pi}(a^{\mathrm{L}} : A^{\mathrm{L}}\, a)\mathbf{\Pi}(b_1 : B_1\ b_0)(B^{\mathrm{L}}\, a\, a^{\underline{D}}\,\langle b_0, b_1\rangle)\ )
\end{aligned}$$

■

Constructing functions $A^{\exists}, B^{\exists}$ is then equivalent to constructing some maps $\mathsf{exists}_0 : \mathsf{Map}_0\ F_0\ \mathsf{Ex}_0^D$:

$$\begin{aligned}
A^{\exists} &:\equiv \boldsymbol{\lambda}\langle a_0, a_1\rangle\,.\,\pi_1\ \mathsf{exists}_0 \cdot a_0 \cdot a_1 \\
B^{\exists} &:\equiv \boldsymbol{\lambda} a\,.\,\boldsymbol{\lambda} a^{\mathrm{L}}\,.\,\boldsymbol{\lambda}\langle b_0, b_1\rangle\,.\,\pi_2\ \mathsf{exists}_0 \cdot b_0 \cdot a \cdot a^{\mathrm{L}} \cdot \mathsf{snd}\ b_1
\end{aligned}$$

In essence, maps of type $\mathsf{Map}_0\ F_0\ \mathsf{Ex}_0^D$ take pairs of erased and predicate terms as inputs, and return pairs of displayed terms and relational proofs as output. $A^{\exists}$ and $B^{\exists}$ express this mapping for each of the two base types respectively. We can generalize this to all inner types, as well as parameter telescopes, via a simple recursive traversal of these structures, and an application of $A^{\exists}, B^{\exists}$ whenever we reach a base type. We thus obtain functions mapping pairs of erased and predicate *algebras* to pairs of displayed and relation *algebras*.

We define the following *generalized left-totality operators*, for any $\Delta : \mathsf{Params}\ \Gamma$, $T : \mathsf{Ty}\ \Gamma\ \Delta$:

$$\begin{aligned}
&T^{\exists} : (\delta^{\mathrm{L}} : \ulcorner \Delta^{\mathsf{L}}\ \gamma^D\ \delta \urcorner)\ (x : \ulcorner T^{\mathsf{A}}\ \gamma\ \delta \urcorner)(x_0^D : \ulcorner T^{\mathsf{ED}}\ x_{\underline{0}} \urcorner) \to \ulcorner T^{\mathsf{L}}\ \gamma^D\ \delta^{\underline{D}}\ x \urcorner \\
&\Delta^{\exists} : (\delta : \Delta^{\mathsf{A}}\ \gamma)(\delta_0^D : \ulcorner \Delta^{\mathsf{ED}}\ \delta_{\underline{0}} \urcorner) \to \ulcorner \Delta^{\mathsf{L}}\ \gamma^D\ \delta \urcorner
\end{aligned}$$

where we implicitly quantify over algebra structures of the following types

$$\gamma_0 : \Gamma^{\mathsf{E}}_{F_0}, \quad \gamma_1 : \Gamma^{\mathsf{W}}_{F_1}\ \gamma_0, \quad \gamma^D : \Gamma^{\mathsf{D}}_{D}\ \gamma$$

where $\gamma :\equiv \Gamma^{\Sigma}\ \gamma_0\ \gamma_1$.

We will mainly use $\Delta^{\exists}$ for parameter telescopes Δ. Its action can alternatively be understood as follows: given a list δ of parameters which types are described by Δ, and δ_0^D is a list of "inductive hypotheses", i.e. a list of left-totality functions that we are in the process of defining, one for each type in Δ. The action of $\_^{\exists}$ is thus to just apply these inductive hypotheses to the parameters $\delta_{\underline{0}}$ recursively.

$$(\mathsf{T}^1)^{\exists}\ \delta^{\mathrm{L}}\ x\ x_0^D :\equiv x_0^D \cdot x_{\underline{1}}$$

$$(\mathsf{T}^2\ t)^{\exists}\ \{\gamma\ \delta\}\ \delta^{\mathrm{L}}\ x\ x_0^D :\equiv$$
$$\quad x_0^D \cdot t^{\mathsf{A}}\ \gamma\ \delta \cdot t^{\mathsf{D}}\ \gamma^D\ \delta^{\underline{D}} \cdot t^{\mathsf{R}}\ \gamma^D\ \delta^{\underline{R}} \cdot x_{\underline{1}}$$

$$(\mathsf{ext}\ \_)^{\exists}\ \_\ x\ x_0^D :\equiv *$$

$$(\pi\ A\ B)^{\exists}\ \_\ f\ f_0^D :\equiv$$
$$\quad \langle \boldsymbol{\lambda} x\,.\,\mathsf{fst}\ (h\,n), \boldsymbol{\lambda} x\,.\,\mathsf{snd}\ (h\,n)\rangle$$
$$\quad \textbf{where } h :\equiv$$
$$\qquad \lambda x\,.\,B^{\exists}\ \_\ (f_{\underline{0}} \cdot x)\ (f_{\underline{1}} \cdot x)\ (f_0^D \cdot x)$$

$$T[\sigma]^{\exists}\ \delta^{\mathrm{L}} :\equiv T^{\exists}\ (\sigma^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ \delta^{\mathrm{L}})$$

$$T[w]^{\exists} :\equiv T^{\exists}\{\gamma^{\mathrm{L}} = w^{\mathsf{L}}\ \gamma^{\mathrm{L}}\}$$

$$\bullet^{\exists}\ \_\ \_ :\equiv *$$

$$(\Delta \rhd\!\rhd T)^{\exists}\ \langle \delta, x\rangle\ \langle p_0^D, x_0^D\rangle :\equiv \langle \delta^{\mathrm{L}}, x^{\mathrm{L}}\rangle$$
$$\quad \textbf{where } \delta^{\mathrm{L}} :\equiv \Delta^{\exists}\ \delta\ p_0^D$$
$$\qquad\qquad x^{\mathrm{L}} :\equiv T^{\exists}\ \delta^{\mathrm{L}}\ x\ x_0^D$$

$$(\Delta[w])^{\exists} :\equiv \Delta^{\exists}\ \{\gamma^{\mathrm{L}} = w^{\mathsf{L}}\ \gamma^{\mathrm{L}}\}$$

$$\mathsf{id}^{\overrightarrow{\exists}}\ \_\ \_\ \delta^{\mathrm{L}} :\equiv \delta^{\mathrm{L}}$$

$$(\mathsf{drop}\ \{T\}\ \sigma)^{\overrightarrow{\exists}}\ \langle \delta, x\rangle\ \langle \delta_0^D, x_0^D\rangle\ p^{\mathrm{L}} :\equiv \langle \delta^{\mathrm{L}}, x^{\mathrm{L}}\rangle$$
$$\quad \textbf{where } \delta^{\mathrm{L}} :\equiv \sigma^{\overrightarrow{\exists}}\ \delta\ \delta_0^D\ p^{\mathrm{L}}$$
$$\qquad\qquad x^{\mathrm{L}} :\equiv T^{\exists}\ x\ x_0^D\ \delta^{\mathrm{L}}$$

Figure 12.4: Generalized left-totality operators

This is not unlike the auxiliary operators $\_^{\mathsf{ih}}$ that we defined in Section 11.2.3 and Section 12.1.3 to apply lists of inductive hypotheses to lists of arguments.

We also define an alternative version of $\Delta^{\exists}$ that only acts on a suffix of the input telescope. This is necessary because sometimes we need to focus on a specific portion of a constructor's list of arguments, for reasons that we pointed out at the very end Section 9.1. We can express that ∇ is a sub-telescope of Δ as the existence of a weakening substitution $w : \mathsf{WkSub}\ \Delta\ \nabla$. We thus define the following function $\_^{\overrightarrow{\exists}}$, again for any Γ and $\gamma_0, \gamma_1, \gamma^D$, by induction on $w : \mathsf{WkSub}\ \Delta\ \nabla$:

$$w^{\overrightarrow{\exists}} : (\delta : \Delta^{\mathsf{A}}\ \gamma)\ (\delta_0^D : \ulcorner \Delta^{\mathsf{ED}}\ \delta_{\underline{0}} \urcorner)(p^{\mathrm{L}} : \ulcorner \nabla^{\mathsf{L}}\ \gamma^D\ (\sigma^{\mathsf{A}}\ \gamma\ \delta)\urcorner) \to \ulcorner \Delta^{\mathsf{L}}\ \gamma^D\ \delta \urcorner$$

Just like $\_^{\exists}$, $\_^{\overrightarrow{\exists}}$ is given an erased displayed algebra for the full telescope Δ, in addition to displayed and relation parameter algebras p^{L} for the prefix sub-telescope ∇. It thus constructs algebras for the full telescope Δ by taking the algebras for the prefix ∇ and extending them with the components of Δ that are not in ∇. These extra components are obtained just like in $\Delta^{\exists}$, i.e. via an application of $A^{\exists}$ for each "extra" local type A.

Expanding displayed parameter algebras via $\Delta^{\overrightarrow{\exists}}$ and then restricting the result again via σ^{D} along the same weakening substitution σ yields the identity:

Lemma 12.2.1. For all $w : \mathsf{WkSub}\ \Delta\ \nabla$ and parameter algebras $\delta, \delta_0^D, p^{\mathrm{L}}$, we have that

$$\sigma^{\mathsf{D}}\ (w^{\vec{\exists}}\ \delta\ \delta_0^D\ p^{\mathrm{L}})^{\underline{D}} = p^{\underline{D}} \qquad \blacksquare$$

Proof. By straightforward induction on w. $\square$

We also have two additional lemmas about the interaction between $\_^{\exists}$, $\_^{\vec{\exists}}$, and weakening substitutions.

Lemma 12.2.2. Let $\Gamma, \gamma_0, \gamma_1,\ \gamma \equiv \Gamma^{\Sigma}\ \gamma_0\ \gamma_1$, as well as $\gamma_0^D : \Gamma^{\mathsf{ED}}\ \gamma_0$, and $w : \mathsf{WkSub}\,\{\Gamma\}\,\Delta\,\nabla$, and parameter algebras δ, δ_0^D. Then,

$$\nabla^{\exists}\ (\sigma^{\mathsf{A}}\ \gamma\ \delta)\ (\sigma^{\mathsf{ED}}\ \gamma_0^D\ \delta_0^D) = \sigma^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ (\Delta^{\exists}\ \delta\ \delta_0^D) \qquad \blacksquare$$

Proof. Straightforward induction on w. $\square$

Lemma 12.2.3. Let $\Gamma, \Delta, \nabla, \sigma, w, \gamma_0, \gamma_1, \gamma^D, \gamma_0^D, \delta, \delta_0^D$ like in Lemma 12.2.2. Let $\gamma :\equiv \Gamma^{\Sigma}\ \gamma_0\ \gamma_1$. Then,

$$w^{\vec{\exists}}\ \delta\ \delta_0^D\ (\nabla^{\exists}\ (\sigma^{\mathsf{A}}\ \gamma\ \delta)\ (\sigma^{\mathsf{ED}}\ \gamma_0^D\ \delta_0^D)) = \Delta^{\exists}\ \delta_0\ \delta_1\ \delta_0^D \qquad \blacksquare$$

Proof. Straightforward induction on w. $\square$

12.2.2 Proving left-totality

The take-away from Section 12.2.1 can be summarized as follows: given carriers $F_0 : \mathsf{Ca}_0, F_1 : \mathsf{Ca}_1\ F_0, D : \mathsf{Ca}^{\mathsf{D}}\ F, R : \mathsf{Ca}^{R}\ F\ D$, where $F :\equiv \mathsf{Ca}_{\Sigma}\ F_0\ F_1$, we can succinctly state left-totality of the relations R as having a structure $f_0 : \mathsf{Map}_0\ F_0\ \mathsf{Ex}_0^D$, for families Ex_0^D as given in Definition 12.2.1. If the types F_0 are equipped with a section inductive erased algebra structure, then we can define f_0 inductively, provided we construct a suitable erased displayed algebra giving the methods of the induction. This section shows how to do so.

Let us fix a *linear* specification Ω, and assume some arbitrary algebra structures on the carriers F_0, F_1, D, R

$$\omega_0 : \Omega^{\mathsf{E}}_{F_0}, \quad \omega_1 : \Omega^{\mathsf{W}}_{F_1}\ \omega_0, \quad \omega^D : \Omega^{\mathsf{D}}_{D}\ \omega, \quad \omega^R : \Omega^{\mathsf{R}}_{R}\ \omega\ \omega^D$$

where $\omega \equiv: \Omega^{\Sigma}\ \omega_0\ \omega_1$. We additionally require that ω_0 is section inductive and that ω_1, ω^R have IIT-inversions. We define $\omega^{\mathrm{L}} : \Omega^{\mathsf{L}}\ \omega$ as $\omega^{\mathrm{L}} :\equiv (\omega^D, \omega^R)$.

We want to construct left-totality proofs, that is, a term $f_0 : \mathsf{Map}_0\ F_0\ \mathsf{Ex}_0^D$, by induction on the erased types A_0, B_0 (where $(A_0, B_0) :\equiv F_0$). This reduces to defining an erased displayed algebra structure $\omega_0^D : \Omega^{\mathsf{ED}}_{\mathsf{Ex}_0^D}\ \omega_0$. Unfolding the types, we have

$$\omega_0^D : \forall\{C\}(c : \mathsf{CTm}\ \Omega\ C) \to \ulcorner C^{\mathsf{ED}}_{\mathsf{Ex}_0^D}\ (c^{\mathsf{E}}_{F_0}\ \omega_0)\urcorner$$

Assuming $C \equiv \mathsf{ctor}\ \Delta\ B$, we proceed by cases on B.

- Case $B \equiv \mathsf{T}^1$. Then, $(\omega_0^D\ c)$ is a target-level function from inputs

  $$\delta_0 : \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner, \quad \delta_0^D : \ulcorner \Delta^{\mathsf{ED}}\ \delta_0 \urcorner, \quad a_1 : \ulcorner A_1\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_0)\urcorner$$

  to a result of type $A^{\mathrm{L}}\ \langle c^{\mathsf{E}}\ \omega_0 \cdot \delta_0, a_1\rangle$.

  Assuming $\delta_0, \delta_0^D, a_1$, we define $\delta_1 : \ulcorner \Delta^{\mathsf{W}}\ \delta_0 \urcorner$ by inversion on a_1, so that $\delta :\equiv \langle \delta_0, \delta_1\rangle$. By proof irrelevance of a_1, we can rewrite the goal into $A^{\mathrm{L}}\, a$, where $a :\equiv c^{\mathsf{A}}\ \omega\ \delta$. Since $\Delta^{\exists}\ \delta\ \delta_0^D : \ulcorner \Delta^{\mathsf{L}}\ \gamma^D\ \delta \urcorner$, we inhabit the goal with $c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot (\Delta^{\exists}\ \delta\ \delta_0^D)$.

- Case $B \equiv \mathsf{T}^2\ t$. By linearity of Ω, we further distinguish two cases:

  – Case $t \equiv \mathsf{capp}\ c'\ \sigma$ with $\sigma : \mathsf{Sub}\ \Delta\ \nabla$ and $w : \mathsf{PWk}\ \sigma$.
  The type of $(\omega_0^D\ c)$ becomes a function from inputs

  $$\begin{aligned}&\delta_0 : \ulcorner \Delta^{\mathsf{E}}_{F_0} \urcorner, \quad \delta_0^D : \ulcorner \Delta^{\mathsf{ED}}\ \delta_0 \urcorner,\\ &a : \ulcorner A \urcorner, \quad a^{\mathrm{L}} : \ulcorner A^{\mathrm{L}}\, a \urcorner,\\ &b_1 : \ulcorner B_1\ (\mathsf{fst}\ a)\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_0)\urcorner\end{aligned}$$

  to a conclusion of type $B^{\mathrm{L}}\, a\, a^{\underline{D}}\, \langle c^{\mathsf{E}}\ \omega_0 \cdot \delta_0, b_1\rangle$
  Unlike in the previous case, we now cannot just conclude with an application of c^{L}: even assuming we had some suitable $\delta^{\mathrm{L}} : \Delta^{\mathsf{L}}\ \omega^D\ \delta$, such term would have the following type

  $$c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot \delta^{\mathrm{L}} : \ulcorner B^{\mathrm{L}}\ (c'^{\mathsf{A}}\ \omega \cdot \sigma^{\mathsf{A}}\ \delta)\ (c'^{\mathsf{D}}\ \omega^D \cdot \_ \cdot \sigma^{\mathsf{D}}\ \delta^{\underline{D}})\ (c^{\mathsf{A}}\ \omega \cdot \delta)\urcorner$$

  However, the first two indices $a, a^{\underline{D}}$ of B^{L} in the current goal type are just fixed variables, and we have to so some work to prove that they are in fact equal to $c'^{\mathsf{A}}\ \omega \cdot \sigma^{\mathsf{A}}\ \delta$ and $c'^{\mathsf{D}}\ \omega^D \cdot \_ \cdot \sigma^{\mathsf{D}}\ \delta^{\underline{D}}$ respectively. We encountered a similar issue in the example of Section 8.1 when proving the left-totality property for Ty^R; and again just like in our previous example, we fix this issue by transporting the goal type along suitable equations obtained by inversion on the proof of well-formedness and relatedness.

Let us assume $\delta_0, \delta_0^D, a, a^{\mathrm{L}}, b_1$. By inversion on b_1, we get a target-level proof of the equation $\mathsf{fst}\ a \approx c'^{\mathsf{E}}\ \omega_0\ (\sigma^{\mathsf{E}}\ \omega_0\ \delta_0)$.

Transporting along this equation, we get the following updated types for our assumptions:

$$\begin{aligned}&b_1 : \ulcorner B_1\ (c'^{\mathsf{E}}\ \omega_0\ (\sigma^{\mathsf{E}}\ \omega_0\ \delta_0))\ (c^{\mathsf{E}}\ \omega_0\ \delta_0)\urcorner\\ &a^{\mathrm{L}} : \ulcorner A^{\mathrm{L}}\ (c'^{\mathsf{A}}\ \omega \cdot (\sigma^{\mathsf{A}}\ \delta))\urcorner\end{aligned}$$

where $\delta_1 : \ulcorner \Delta^{\mathsf{W}}\ \delta_0 \urcorner$ is obtained by inversion on b_1, and $\delta \equiv \langle \delta_0, \delta_1\rangle$. Thus, by proof-irrelevance we get $a = c'^{\mathsf{A}}\ \omega\ (\sigma^{\mathsf{A}}\ \delta)$ and $\langle c^{\mathsf{E}}\ \omega_0 \cdot \delta_0, b_1\rangle = c^{\mathsf{A}}\ \omega \cdot \delta$.

By inversion on r, we get parameter algebras $q^{\mathrm{L}} : \ulcorner \nabla^{\mathsf{L}}\ \omega^D\ (\sigma^{\mathsf{A}}\ \delta)\urcorner$.

as well as a proof of the equation $a^D \approx c'^{\mathsf{D}}\ \omega^D \cdot {}_{-} \cdot q^{\underline{D}}$.

Transporting once again the goal type along this equation, we get the following

$$B^{\mathrm{L}}\ (c'^{\mathsf{A}}\ \omega \cdot \sigma^{\mathsf{A}}\ \delta)\ (c'^{\mathsf{D}}\ \omega^D \cdot {}_{-} \cdot q^{\underline{D}})\ (c^{\mathsf{A}}\ \omega \cdot \delta)$$

From $w^{\overrightarrow{\exists}}\ \delta\ \delta_0^D\ q^{\mathrm{L}}$, we obtain algebras $\delta^{\mathrm{L}} : \ulcorner \Delta^{\mathsf{L}}\ \omega^D\ \delta\urcorner$. Moreover, by Lemma 12.2.1 we have $q^{\underline{D}} = \sigma^{\mathsf{D}}\ \omega^D\ \delta^{\underline{D}}$, hence rewrite to the goal type

$$B^{\mathrm{L}}\ (c'^{\mathsf{A}}\ \omega \cdot \sigma^{\mathsf{A}}\ \delta)\ (c'^{\mathsf{D}}\ \omega^D \cdot {}_{-} \cdot \sigma^{\mathsf{D}}\ \omega^D\ \delta^{\underline{D}})\ (c^{\mathsf{A}}\ \omega \cdot \delta)$$

which we inhabit by $c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot \delta^{\mathrm{L}}$.

– Case $t \equiv \mathsf{vz}$. This case is handled in the same way as the previous, with the exception that we don't need to transport along, or even prove, any equation involving the displayed index $a^{\underline{D}}$.

By the section induction property of (F_0, ω_0) and the structure ω_0^D, it follows that there exists a pair of terms $f_0 : \mathsf{Map}_0\ F_0\ \mathsf{Ex}_0^D$ with a section structure $s_0 : \Omega^{\mathsf{ES}}_{f_0}\ \omega_0\ \omega_0^D$.

From f_0, we obtain proofs $A^{\exists}, B^{\exists}$ of left-totality of A^R and B^R by projecting out the first and second component respectively.

$$\begin{aligned}&A^{\exists} :\equiv \boldsymbol{\lambda}\langle a_0, a_1\rangle \,.\, \pi_1\ f_0 \cdot a_0 \cdot a_1\\ &B^{\exists} :\equiv \boldsymbol{\lambda} a \,.\, \boldsymbol{\lambda} a^{\mathrm{L}} \,.\, \boldsymbol{\lambda}\langle b_0, b_1\rangle \,.\, \pi_2\ f_0 \cdot b_0 \cdot a \cdot a^{\mathrm{L}} \cdot \mathsf{snd}\ b_1\end{aligned}$$

However, this is not sufficient to convince ourselves that these left-totality proofs have the intended behaviour, i.e. that they respect the constructors specified by Ω. More specifically, we want to prove the following two lemmata:

Lemma 12.2.4. For all $\Delta : \mathsf{Params}\ \Omega$, $c : \mathsf{CTm}\ \Omega\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1)$, and parameter algebra δ,

$$A^{\exists} \cdot (c^{\mathsf{A}}\ \omega \cdot \delta) = c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot (\Delta^{\exists}\ \delta\ (\Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}})) \qquad \blacksquare$$

Proof. From the section structure on f_0 and Lemma 11.1.2, we know that

$$A^{\exists} \cdot (c^{\mathsf{A}}\ \omega \cdot \delta) \equiv \pi_1\ f_0\ (c^{\mathsf{E}}\ \omega_0 \cdot \delta_{\underline{0}}) \cdot \_ = \omega_0^D\ c \cdot \delta_{\underline{0}} \cdot (\Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}}) \cdot \_$$

The rest follows immediately by definition of ω_0^D. $\square$

Lemma 12.2.5. For all $\Delta : \mathsf{Params}\ \Omega$, $t : \mathsf{Tm}\ \Omega\ \Delta\ \mathsf{T}^1$, $c : \mathsf{CTm}\ \Omega\ (\mathsf{ctor}\ \Delta\ (\mathsf{T}^2\ t))$, and parameter algebra δ. Then,

$$B^{\exists} \cdot (t^{\mathsf{A}}\ \omega\ \delta) \cdot (t^{\mathsf{L}}\ \omega^{\mathrm{L}}\ \delta^{\mathrm{L}}) \cdot (c^{\mathsf{A}}\ \omega \cdot \delta) = c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot \delta^{\mathrm{L}}$$

where $\delta^{\mathrm{L}} :\equiv \Delta^{\exists}\ \delta\ (\Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}})$. $\blacksquare$

Proof. By Lemma 11.1.2, the goal equation reduces to

$$\omega_0^D\ c \cdot (\Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}}) \cdot (t^{\mathsf{A}}\ \omega\ \delta) \cdot (t^{\mathsf{L}}\ \omega^{\mathrm{L}}\ \delta^{\mathrm{L}}) \cdot (c^{\mathsf{W}}\ \omega_1 \cdot \delta_{\underline{0}} \cdot \delta_{\underline{1}}) = c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot \delta^{\mathrm{L}}$$

By linearity of Ω, we distinguish two cases:

- Case $t \equiv \mathsf{capp}\ c'\ \sigma$ where $c' \equiv \mathsf{CTm}\ \Omega\ (\mathsf{ctor}\ \nabla\ \mathsf{T}^1)$, and $\sigma : \mathsf{Sub}\ \Delta\ \nabla$ such that $w : \mathsf{PWk}\ \sigma$.

  Let

  $$\delta_0^D :\equiv \Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}}, \quad q^{\mathrm{L}} :\equiv \Delta^{\exists}\ \delta\ \delta_0^D, \quad p^{\mathrm{L}} :\equiv w^{\overrightarrow{\exists}}\ p_0^D\ (\sigma^{\mathsf{L}}\ \omega^{\mathrm{L}}\ q^{\mathrm{L}})$$

  By definition of ω_0^D, the goal equation rewrites to

  $$c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot p^{\mathrm{L}} = c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot q^{\mathrm{L}}$$

  This follows by congruence and $p^{\mathrm{L}} = q^{\mathrm{L}}$, which in turn follows from Lemma 12.2.2, Lemma 12.2.3, and transitivity.

- Case $t \equiv \mathsf{vz}$. By reflexivity.

$\square$

The lemmata above establish that $A^{\exists}$ and $B^{\exists}$ respect constructor term algebras, and their action can be reduced to the action of displayed and relation algebra operators. We can generalize this result to the generalized existence operators $\_^{\exists}$ for parameter telescopes, inner types, and base types. More specifically, we have the following lemma:

Lemma 12.2.6. Given $\Delta : \mathsf{Params}\ \Omega$ and $\delta : \ulcorner \Delta^{\mathsf{A}}\ \omega \urcorner$, let

$$\delta_0^D :\equiv \Delta_{f0}^{\mathsf{ES}}\ \delta_{\underline{0}}, \quad \delta^{\mathrm{L}} :\equiv \Delta^{\exists}\ \delta\ \delta_0^D$$

Then,

1. $\nabla^{\exists}\ (\sigma^{\mathsf{A}}\ \delta)\ (\sigma^{\mathsf{ED}}\ \omega_0^D\ \delta_0^D) = \sigma^{\mathsf{L}}\ \omega^{\mathrm{L}}\ \delta^{\mathrm{L}}, \qquad$ for $\sigma : \mathsf{Sub}\ \Delta\ \nabla$
2. $T^{\exists}\ \delta^{\mathrm{L}}\ (t^{\mathsf{A}}\ \omega\ \delta)\ (t^{\mathsf{ED}}\ \omega_0^D\ \delta_0^D) = t^{\mathsf{L}}\ \omega^{\mathrm{L}}\ \delta^{\mathrm{L}}, \qquad$ for $t : \mathsf{Tm}\ \Omega\ \Delta\ T$
3. $T^{\exists}\ \delta^{\mathrm{L}}\ (c^{\mathsf{A}}\ \omega \cdot \delta)\ (c^{\mathsf{ED}}\ \omega_0^D \cdot \delta_0^D) = c^{\mathsf{L}}\ \omega^{\mathrm{L}} \cdot \delta^{\mathrm{L}}, \qquad$ for $c : \mathsf{CTm}\ \Omega\ (\mathsf{ctor}\ \Delta\ T)$ ■

Proof. By mutual induction

1. on σ
2. on t
3. on T, as well as appropriate use of Lemma 12.2.4 and Lemma 12.2.5 depending on the shape of T. □

A consequence of Lemma 12.2.6.2 is that we can actually generalize Lemma 12.2.4 to show that $A^{\exists}$ respects *any* term algebra in addition to just *constructor* term algebras.

Lemma 12.2.7. For all $\Gamma : \mathsf{Spec}, w : \mathsf{Wk}\ \Omega\ \Gamma$, $\Delta : \mathsf{Params}\ \Gamma$, $t : \mathsf{Tm}\ \Gamma\ \Delta\ \mathsf{T}^1$, and parameter algebra δ,

$$A^{\exists} \cdot (t^{\mathsf{A}}\ \gamma\ \delta) = t^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ (\Delta^{\exists}\ \delta\ (\Delta_{f0}^{\mathsf{ES}}\ \delta_{\underline{0}}))$$

where $\gamma :\equiv w^{\mathsf{A}}\ \omega, \gamma^{\mathrm{L}} :\equiv w^{\mathsf{L}}\ \omega^{\mathrm{L}}$. ■

Proof. By Lemma 11.1.2 and Lemma 12.2.6.2. □

12.3 Constructing the sections

We finally reach the last step in proving section induction for IIT algebras. We continue our discussion from the previous Section 12.2, where we demonstrated how to construct target-level proofs of left-totality for eliminator relations arising from suitable Σ-algebras. The next step is to use those proof terms to define eliminator functions, and prove that they indeed constitute a section.

Defining the functions themselves is straightforward, and not unlike the several examples we have already seen. Recall the metatheoretic definition of the eliminators from the Con/Ty IIT from Section 8.1:

$$\begin{aligned}
&\mathsf{elim}_{\mathsf{Con}}\ \Gamma :\equiv \pi_1\ (\mathsf{Con}^{\exists}\ \Gamma)\\
&\mathsf{elim}_{\mathsf{Ty}}\ \{\Gamma\}\ A :\equiv \pi_1\ (\mathsf{Ty}^{\exists}\ A\ (\pi_2\ (\mathsf{Con}^{\exists}\ \Gamma)))
\end{aligned}$$

Similarly, keeping the target-level carrier types A, B, A^D, B^D and left-totality proofs $A^{\exists}, B^{\exists}$ from the previous section, we define:

$$\begin{aligned}
&\mathsf{elim}_A : \boldsymbol{\Pi}(a : A)(A^D\ a) && \mathsf{elim}_B : \boldsymbol{\Pi}(a : A)\boldsymbol{\Pi}(b : B\ a)\ (B^D\ (\mathsf{elim}_A \cdot a)\ b)\\
&\mathsf{elim}_A\ :\equiv \boldsymbol{\lambda} a\,.\,\mathsf{fst}\ (A^{\exists} \cdot a) && \mathsf{elim}_B :\equiv \boldsymbol{\lambda} a\,.\,\boldsymbol{\lambda} b\,.\,\mathsf{fst}(B^{\exists} \cdot a \cdot (A^{\exists} \cdot a) \cdot b)
\end{aligned}$$

These terms form a section map $f : \mathsf{Map}\ (\mathsf{Ca}_{\Sigma}\ F_0\ F_1)\ F^D$ as $f :\equiv (\mathsf{elim}_A, \mathsf{elim}_B)$. We are now left to show that these functions admit a section structure. We shall first establish a few lemmas relating erased sections on the left-totality proofs $f_0 : \mathsf{Map}_0\ F_0\ \mathsf{Ex}_0^D$ as defined in the previous section, as well as IIT sections on the eliminators f.

Lemma 12.3.1. Let $w : \mathsf{Wk}\ \Omega\ \Gamma$ for arbitrary Γ, and let $(\gamma_0, \gamma_1, \gamma, \gamma^D, \gamma^R) :\equiv (w^{\mathsf{E}}\ \omega_0, ..., w^{\mathsf{R}}\ \omega^R)$.

Moreover, assume $s : \Gamma_f^{\mathsf{S}}\ \gamma\ \gamma^D$, $\Delta : \mathsf{Params}\ \Gamma$, $\delta : \ulcorner \Delta_F^{\mathsf{A}}\ \gamma \urcorner$, and let $\delta_0^D :\equiv \Delta_{f_0}^{\mathsf{ES}}\ \delta_{\underline{0}}$. Then

1. $(\Delta^{\exists}\ \delta\ \delta_0^D)^{\underline{D}} = \Delta_f^{\mathsf{S}}\ s\ \delta$

2. $(T^{\exists}\ \delta^{\mathrm{L}}\ x\ (T_{f_0}^{\mathsf{ES}}\ x_{\underline{0}}))^{\underline{D}} = T_f^{\mathsf{S}}\ s\ x,\qquad$ for $T : \mathsf{Ty}\ \Gamma\ \Delta$, $\delta^{\mathrm{L}} :\equiv \Delta^{\exists}\ \delta\ \delta_0^D$ ∎

Proof. By mutual induction

1. on Δ

2. on T. When $T :\equiv T'[\sigma]$ for some T', σ, we make use of Lemma 12.2.6 (1) and Lemma 11.1.1. The case $T :\equiv \mathsf{T}^1$ is immediate by point 1 of this lemma, which holds by inductive hypothesis. The case $T :\equiv \mathsf{T}^2\ t$ for some

$t : \mathsf{Tm}\ \Gamma\ \Delta\ \mathsf{T}^1$ is a bit more involved. If we unfold the definitions on both sides of the equation, we obtain

$$B^{\exists} \cdot (t^{\mathsf{A}}\ \gamma\ \delta) \cdot (t^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ \delta^{\mathrm{L}}) \cdot b = B^{\exists} \cdot (t^{\mathsf{A}}\ \gamma\ \delta) \cdot (A^{\exists} \cdot (t^{\mathsf{A}}\ \gamma\ \delta)) \cdot b$$

This reduces to proving $A^{\exists} \cdot (t^{\mathsf{A}}\ \gamma\ \delta) = t^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ \delta^{\mathrm{L}}$, which follows immediately from Lemma 12.2.7.

The remaining cases are straightforward. $\square$

Lemma 12.3.2. Let $\Gamma : \mathsf{Spec}, w : \mathsf{Wk}\ \Omega\ \Gamma$, and $(\gamma_1, \gamma, \gamma^D) :\equiv (w^{\mathsf{W}}\ \omega_1, w^{\mathsf{A}}\ \omega, w^{\mathsf{D}}\ \omega^D)$. For all $C : \mathsf{Ctor}\ \Gamma, c : \mathsf{CTm}\ \Omega\ (C[w])$ and $s : \Gamma^{\mathsf{S}}_f\ \gamma\ \gamma^D$, we have a term of type

$$C^{\mathsf{S}}_f\ s\ (C^{\Sigma}\ (c^{\mathsf{E}}\ \omega_0)\ (c^{\mathsf{W}}\ \omega_1))\ (c^{\mathsf{D}}\ \omega^D) \qquad \blacksquare$$

Proof. Assume c, and suppose $C \equiv \mathsf{ctor}\ \Delta\ X$ for some $\Delta : \mathsf{Params}\ \Gamma, X : \mathsf{Base}\ \Gamma\ \Delta$. Then, the goal type reduces to asserting that the following holds

$$X^{\mathsf{S}}_f\ s\ (c^{\mathsf{A}}\ \omega \cdot \delta) = c^{\mathsf{D}}\ \omega^D \cdot \delta \cdot (\Delta^{\mathsf{S}}_f\ s\ \delta)$$

for all $\delta : \ulcorner \Delta^{\mathsf{A}}\ \gamma \urcorner$. To show that this is the case, let $\delta^D_0 :\equiv \Delta^{\mathsf{ES}}_{f_0}\ \delta_{\underline{0}}$ and $\delta^D :\equiv (\Delta^{\exists}\ \delta\ \delta^D_0)^{\underline{D}}$.

We proceed by case analysis on X.

- Case $X \equiv \mathsf{T}^1$. The goal equation and its solution is as follows:

$$\begin{aligned}
& (\mathsf{T}^1)^{\mathsf{S}}_f\ s\ (c^{\mathsf{A}}\ \omega \cdot \delta) \\
\equiv\ & \\
& \pi_1\ f \cdot (c^{\mathsf{A}}\ \omega \cdot \delta) \\
=\ & \quad \{\ s_0\ c\ \delta_{\underline{0}}\ \} \\
& c^{\mathsf{D}}\ \omega^D \cdot \delta \cdot \delta^D \\
=\ & \quad \{\ \text{Lemma 12.3.1.1}\ \} \\
& c^{\mathsf{D}}\ \omega^D \cdot \delta \cdot (\Delta^{\mathsf{S}}_f\ s\ \delta)
\end{aligned}$$

- Case $X :\equiv \mathsf{T}^2\ t$ for some $t : \mathsf{Tm}\ \Gamma\ \Delta\ \mathsf{T}^1$. Let

$$a :\equiv t^{\mathsf{A}}\ \gamma\ \delta, \qquad b :\equiv c^{\mathsf{A}}\ \omega \cdot \delta, \qquad a^{\mathrm{L}} :\equiv A^{\exists} \cdot a, \qquad \delta^{\mathrm{L}} :\equiv \Delta^{\exists}\ \delta\ \delta^D_0$$

Then

$$(\mathsf{T}^2\ t)^{\mathsf{S}}_f\ s\ b$$

$$
\begin{aligned}
= &\quad \{ \text{ Lemma 11.1.2 } \} \\
& (B^{\exists} \cdot a \cdot a^{\mathrm{L}} \cdot b)^{\underline{D}} \\
= &\quad \{ \text{ Lemma 12.2.7 } \} \\
& (B^{\exists} \cdot a \cdot (t^{\mathsf{L}}\ \gamma^{\mathrm{L}}\ \delta^{\mathrm{L}}) \cdot b)^{\underline{D}} \\
= &\quad \{ \text{ Lemma 12.2.5 } \} \\
& c^{\mathsf{D}}\ \omega^{D} \cdot \delta \cdot \delta^{D} \\
= &\quad \{ \text{ Lemma 12.3.1.1 } \} \\
& c^{\mathsf{D}}\ \omega^{D} \cdot \delta \cdot (\Delta^{\mathsf{S}}_{f}\ s\ \delta)
\end{aligned}
$$
□

Lemma 12.3.3. For all $\Gamma : \mathsf{Spec}$ and $w : \mathsf{Wk}\ \Omega\ \Gamma$, the following holds

$$\Gamma^{\mathsf{S}}_{f}\ (w^{\mathsf{A}}\ \omega)\ (w^{\mathsf{D}}\ \omega^{D})$$
■

Proof. Let us abbreviate $(\gamma_0, \gamma_1, \gamma, \gamma^D) :\equiv (w^{\mathsf{E}}\ \omega_0, ..., w^{\mathsf{D}}\ \omega^D)$

We proceed by induction on Γ. The case where $\Gamma \equiv \diamond$ is trivial.

When $\Gamma \equiv \Phi \triangleright C$ for some $\Phi : \mathsf{Spec}, C : \mathsf{Ctor}\ \Phi$, then the goal unfolds to the following record:

$$
\begin{aligned}
& s : \Phi^{\mathsf{S}}_{f}\ (\Phi^{\Sigma}\ (\pi_1\ \gamma_0)\ (\pi_1\ \gamma_1))\ (\pi_1\ \gamma^D) \\
& h : C^{\mathsf{S}}_{f}\ s\ (C^{\Sigma}\ (\pi_2\ \gamma_0)\ (\pi_2\ \gamma_1))\ (\pi_2\ \gamma^D)
\end{aligned}
$$

We obtain s by inductive hypothesis, given the weakening $w \circ \mathsf{drop} : \mathsf{Wk}\ \Omega\ \Phi$, whereas h follows from Lemma 12.3.2, given again the weakening $w \circ \mathsf{drop}$, and constructor term $\mathsf{cvz}[w] : \mathsf{CTm}\ \Omega\ (C[w \circ \mathsf{drop}])$. □

Theorem 12.3.1. The pair of eliminators $f :\equiv (\mathsf{elim}_A, \mathsf{elim}_B)$ is a section, that is, $\Omega^{\mathsf{S}}_{f}\ \omega\ \omega^D$ holds. ■

Proof. Direct consequence of Lemma 12.3.3 on the identity weakening. □

We finally reached the culmination of this chapter, through which we have defined a pair of eliminator functions $(\mathsf{elim}_A, \mathsf{elim}_B)$ for the fixed Σ-algebra ω and displayed algebra ω^D, and showed that these functions constitute a section. The linear specification and the algebras fixed at the beginning were completely arbitrary, therefore our entire construction can be carried out for any choice of specification and algebra that fits the necessary requirements.

Theorem 12.3.2. Let $\Gamma : \mathsf{Spec}$, and inductive erased algebra $alg_0 : \mathsf{Alg}_0\ \Gamma$ and predicate algebra with inversions $alg_1 : \mathsf{Alg}_0\ \Gamma\ alg_0$. Then, the Σ-algebra $alg :\equiv \mathsf{sigma}\ alg_0\ alg_1$ is *section inductive.* ■

Proof. Suppose we have a displayed algebra $alg^D : \mathsf{Alg}^D\ \Gamma\ alg$. This whole chapter provides instructions for constructing a section $(f, s) : \mathsf{Sect}\ \Gamma\ alg\ alg^D$. □

We conclude the chapter with a proof of the second and last half of the main theorem of this part of the thesis.

Theorem 12.3.3. For any $\Gamma : \mathsf{Spec}$, the IIT-algebra defined as per Theorem 11.3.1 is section inductive. ■

Proof. The IIT-algebra defined in Theorem 11.3.1 is a Σ-algebra obtained from an inductive erased algebra (Theorem 11.1.1) and a predicate algebra with inversions (Theorem 11.2.1). Therefore, Theorem 12.3.2 applies. □

Chapter 13

Conclusion of Part III

This part of the thesis has presented a reduction from a subclass of all infinitary IITs to inductive families. A notable aspect of the encoding is that it does not rely on function extensionality in the encoding type theory. We have drawn from a known method to reduce finitary IITs to inductive families [AKKvR19, AKKvR18, vR19], and adapted it to make it applicable to infinitary IITs in an intensional setting. The main obstacles to intensionally encoding infinitary types arise from the proofs of propositionality of the well-formedness predicates, as well as right-uniqueness of the eliminator relations, both of which appear tricky to establish in the absence of *funext*. Our first modest contribution was to point out that fortunately, for a wide class of infinitary IITs, one can in fact make do without proving those properties at all, as long as the metatheory is extended with a few extra tools. One such tool is a universe of definitionally proof-irrelevant propositions, which allows to define well-formedness predicates that are propositional by definition. As a nice byproduct we get that the IIT eliminators encoded in this fashion exhibit definitional β-rules.

Of course the work presented here is just the beginning, as we focused on a proper subclass of all infinitary IITs. In the following sections we want to lay out possible ways in which to improve and expand upon our work.

This chapter and all the previous ones contain many hints for future work, which we summarize in Section 13.3. Finally, having made our own contributions clear, we conclude by presenting some related work, with a focus on discussing the similarities and differences of our work with the existing literature.

13.1 Beyond linear and infinitary IITs

In this thesis we have considered the problem of encoding linear infinitary IITs in terms of inductive families. We now briefly touch upon some closely related problems, and in particular consider what our study of linear infinitary IITs can tell us regarding IITs that are not linear or infinitary.

None of the ideas discussed below have been proved with any degree of rigor, and are only substantiated by examples and informal reasoning. A proper proof (or disproof) of these ideas is outside the scope of the thesis.

13.1.1 Finitary IITs

Being tailored specifically to infinitary IITs, one might wonder if our work can tell us anything new about the problem of reducing finitary types. We believe the answer is positive, since our novel use of definitionally proof-irrelevant propositions to define well-formedness predicates is universally applicable to both the finitary and infinitary case. It is immediate to adapt the known reduction method for finitary IITs (Section 3.2) to employ these tools. If the encoding theory is sufficiently equipped, this then gives a way to encode all finitary IITs with definitional β-rules for the encoded IIT eliminators. Previous works on the topic do not consider universes of strict propositions, and only consider β-rules up to propositional equality [AKKvR19, AKKvR18, vR19].

13.1.2 Linearization of non-linear IITs

Any non-linear IIT induces a linear IIT via a process of "linearization" that we now illustrate. The idea is that any instance of non-linearity can be reframed in terms of equality types.

Take the IIT $A : \mathbf{Type}, B : A \to \mathbf{Type}$, with constructors

$$\begin{aligned}
&\bullet : A \\
&\mathsf{ext} : (a : A) \to B\ a \to A \\
&\mathsf{pair} : (a\ a' : A) \to A \\
&\mathsf{nonlin} : (a : A) \to B\ (\mathsf{pair}\ a\ a)
\end{aligned}$$

The non-linear constructor nonlin can be equivalently rewritten as

$$\mathsf{nonlin} : (x\ y : A) \to x = y \to B\ (\mathsf{pair}\ x\ y)$$

where we make explicit the use of the identity type to express non-linearity. While not as syntactically evident, the updated nonlin constructor is still essentially non-linear, and as such it would fail to be encodable with our method for the reasons already discussed in Section 9.1. The issue here isn't with the statement of equality, but with the fact that the identity type itself, which we take here to be the standard Martin-Löf identity type inductively generated by reflexivity, is defined non-linearly.

We can make A, B linear by extending its definition with two additional types $A^\sim, B^\sim$ representing an inductive version of equality on A and B respectively.

$$\begin{aligned}
&A : \mathbf{Type}\\
&B : \mathbf{Type}\\
&A^\sim : A \to A \to \mathbf{Type}\\
&B^\sim : \forall\{a\ a'\} \to B\ a \to B\ a' \to \mathbf{Type}
\end{aligned}$$

We then keep all the constructors above, except nonlin which we now write as follows to make use of the newly-introduced equality types replacing the identity type

$$\mathsf{lin} : (x\ y : A) \to A^\sim\ x\ y \to B\ (\mathsf{pair}\ x\ y)$$

We also add constructors for $A^\sim$ and $B^\sim$ that state all the possible ways to inductively prove equality for A and B:

$$\begin{aligned}
&\bullet^\sim : A^\sim\ \bullet\ \bullet\\
&\mathsf{ext}^\sim : A^\sim\ a\ a' \to B^\sim\ b\ b' \to A^\sim\ (\mathsf{ext}\ a\ b)\ (\mathsf{ext}\ a'\ b')\\
&\mathsf{pair}^\sim : A^\sim\ x\ x' \to A^\sim\ y\ y' \to A^\sim\ (\mathsf{pair}\ x\ y)\ (\mathsf{pair}\ x'\ y')\\
&\mathsf{lin}^\sim : A^\sim\ x\ x' \to B^\sim\ (\mathsf{lin}\ x\ y\ p)\ (\mathsf{lin}\ x'\ y'\ p')
\end{aligned}$$

The relations $A^\sim, B^\sim$ play the same role of the identity type, but their definition is *linear*. As a result, the IIT composed of $A, B, A^\sim, B^\sim$ is now properly linear. We can prove that $A^\sim, B^\sim$ are equivalence relations, hence we have a term $\mathsf{refl}_{A^\sim}$. We then encode the constructors of the previous IIT, including nonlin:

$$\begin{aligned}
&\mathsf{nonlin} : (a : A) \to B\ (\mathsf{pair}\ a\ a)\\
&\mathsf{nonlin}\ a :\equiv \mathsf{lin}\ a\ a\ (\mathsf{refl}_{A^\sim}\ a)
\end{aligned}$$

We believe that this process of linearization can be applied systematically to any IIT. The question that naturally arises is then: what kind of relation can we establish between the original non-linear IIT and its linearized version? In the finitary case, the inductively-defined equivalence relations are provably equivalent to the identity type, thus making the two IITs equivalent. This seems to suggest that the classes of linear and non-linear IIT are, in the finitary case, equivalent. In the infinitary case, proving the equivalence between the identity type and the inductively-generated equivalence relations seems to require function extensionality. We therefore conjecture that the two types are equivalent when embedded in a theory supporting extensional reasoning, like SeTT.

13.2 Formalization

We accompany the mathematical content of this part with a formalization in the Agda proof assistant. Due to time and technical constraints the formalization is largely but not fully complete, and partially deviates from the proofs on paper. We provide it as non-definitive proof of correctness of our work, with the following caveats:

- In the formalization, we do not explicitly distinguish between the metalevel and the target level. This was initially done to facilitate exploration and prototyping, with the idea of later introducing a proper separation. Alas, formalizing with two levels proved to be more challenging than expected: to make the formalization process bearable we wanted to embed the target level so that the model equalities would hold definitionally. To achieve this we tried two approaches: a shallow embedding similar to [KKK19], and a deep embedding relying on postulates and rewriting rules. The former approach achieved good typechecking performance, but turned out to be difficult to use due to Agda's somewhat limited facilities for fine-grained control over goal simplification and unfolding, which led to goal types that were very difficult to read. The latter approach fared better on goal readability, but lost on the performance side due to the computational demands of Agda's still experimental rewriting facility.

  In light of these considerations, we decided to keep working on the first, "one-level" formalization as the "official" and more extensive version, while simultaneously developing a more faithful but incomplete "two-level"[1] version for additional sanity-check. The two-level version uses postulates and rewriting rules to implement the target level, as this approach seemed to give better ergonomics overall compared to a proper shallow embedding, despite the performance issues. Note that the single-level formalization is not free from performance issues, as it also makes heavy use of rewriting to simulate, to some extent, equality reflection and alleviate *transport hell*. The Agda typechecker was especially put to the test in the construction of the IIT eliminators, where we had to comment out entire parts of some proofs, despite their apparent correctness, because the rewriting rules would not kick in as expected.

[1]We use "two-level" to describe the embedding of the target theory/models within the metatheory via HOAS, as discussed in Section 9.3. In particular, it is not a reference to the system of two-level type theory (2LTT) proposed in [ACKS23]. While there may be points of affinity and potential applications of 2LTT to our case, we have not looked into the matter nearly enough to comment on it.

We provide both formalizations, with the two-level version essentially covering most of the material as the other one, with some differences explained below.

- Due to Agda's lack of support for QIITs, the formalization of the specifications datatype Spec relies on postulates to introduce equality constructors. As a consequence, all constructions involving Spec are an approximation of the intended ones. In particular, definitions by induction on Spec disregard the clauses regarding equality constructors, which from Agda's point of view do not exist, and must therefore be checked by hand. We expect most of these clauses to hold by reflexivity, although we have only formally checked a handful of them.

- The formalized version of Spec contain some auxiliary constructors that are meant to improve its computational behaviour, and that are not present in the on-paper definition. For example, we use a dedicated constructor $\mathsf{capp}_1 : \mathsf{CTm}\ \Gamma\ (\mathsf{ctor}\ \Delta\ \mathsf{T}^1) \to \mathsf{Sub}\ \nabla\ \Delta \to \mathsf{Tm}\ \Gamma\ \nabla\ \mathsf{T}^1$ for constructor terms targeting the first base type T^1, with an equation $\mathsf{capp}\ c\ \sigma = \mathsf{capp}_1\ c\ t$. This is to avoid having to construct terms $\mathsf{capp}\ c\ \sigma : \mathsf{Tm}\,\Gamma\,\nabla\,(\mathsf{T}^1[\sigma])$ where the substitution $\mathsf{T}^1[\sigma]$ in the type is known to be reducible away.

- We make extensive use of Agda's rewriting facility to simulate equality reflection in the metatheory.

The files can be found in [Ses23], under the directories `iit-reduction` and `iit-reduction-twolevel`. The directory `iit-reduction` contains the single-level formalization, and covers all the mathematical material of Part III, with the exception of the concrete encodings of erased types, predicates, and eliminator relations from Section 11.1.4, Section 11.2.3, and Section 12.1.3 respectively.

These encodings are instead fully covered by the two-level formalization, which can be found under `iit-reduction-twolevel`. The two-level formalization does not include the construction of the eliminators, however it does include the Σ-construction.

Both formalizations include a `README.agda` file pointing to the relevant modules, with a brief description of their contents.

13.3 Future work

The work described in this part of the thesis offers many opportunities for improvement and further investigation. We summarize some of them below:

- We would like to extend our reduction to all linear IITs beyond the notion of linearity covered by Definition 9.2.1. We could not find a satisfactory way to formalize linear constructors in full generality, so we contented ourselves with a limited form. Still, we believe that the encoding method applies to any linear IIT, and our tests on concrete examples have supported this claim.

- We would like to extend our reduction to IITs with more complex sorts, like those indexed by external types. We do not expect to have trouble adapting the reduction method to *closed* IITs with multiple sorts, given how these can always be reduced to equivalent two-sorted IITs (Section 3.1). Things might be more complex, however, when we switch to *open* IITs and introduce external indices in their sorts. As we have seen in Section 6.6, not all open IITs are amenable to be encoded with our method, as some choices of external types in the sorts can prevent the definition of inversion principles that are crucial for the encoding. The precise conditions that need to be imposed on the external indices of reducible IITs is an open question;

- We have conjectured linearity of the constructors to be a sufficient condition for the applicability of our reduction method, however we do not know if it is a necessary condition. In other words, we would like to investigate whether there are classes of infinitary IITs beyond the linear ones that can similarly be reduced to inductive families, perhaps via a different encoding method;

- Most of the theorems involving the general reduction are given by induction on the QIIT of specifications Spec. Because of the quotient aspect of it, the elimination principle of Spec requires to check that related constructors are mapped to related methods of the elimination. We have not actually verified that this is the case except for a handful of equations, which hold trivially by reflexivity. We leave the verification of the rest of them for future work.

- Section induction and initiality for QIIT algebras can be shown to be equivalent in an extensional setting [ACD$^+$18]. While relying on the notion of section induction in our work, we do not consider initiality nor we relate it to section induction. We would like to work on this aspect in the future.

- Another point for future work is to fix the issues with the formalization, which we discussed in Section 13.2. This could mean improve the existing formalization, or rewrite it with tools that would be better suited for the project, like a proof-assistant with native support for extensional (setoid) reasoning and QIITs.

13.4 Related work

Although we claim novelty in our contributions, we can find several entries in the existing literature that tackle similar problems. In this Section we want to point out related work that is particularly affine to our own, and discuss the similarities and differences.

Kaposi and Kovács [KK18] propose a type-theoretic syntax to specify higher inductive-inductive types (HIITs). The specification syntax is expressive, and allows infinitary (higher) constructors while preventing non-strictly positive ones. Compared to their syntax of codes, our specification data type Spec bears many similarities, as well as some important differences, as already extensively discussed in Section 9.4. Overall, their syntax is much more general and expressive, and is meant to cover a class of types that is as wide as possible, whereas we have tailored ours for a specific sub-class. Ultimately, the goals of our works different quite a bit, as they do not consider any form of reduction from HIITs to simpler forms of induction.

The reduction method employed in this thesis, based on the definition of erased types, well-formed predicates, and eliminator relations, is directly inspired by Altenkirch, Kaposi, Kovács, and Von Raumer's treatment of finitary IITs [AKKvR18, AKKvR19], as summarized in Von Raumer's PhD thesis [vR19]. He describes a general method for reducing (finitary) IITs to inductive families; he gives a syntax IIT specifications, and defines a notion of algebra of IITs by induction on it; he also defines a syntax of specifications for general inductive families (IF), as well as algebras. The Σ-*construction* is then implemented by defining maps from IIT specifications to pairs of IF specifications giving the codes of the erased types and well-formedness predicates respectively. Similarly, he also defines mappings from IIT specifications to IF specifications for the eliminator relations.

His proofs of the existence of erased and predicate types, as well as eliminator relations, is more thorough and systematic, as he provides a general specification syntax for IFs as well as an associated notion of algebra, and then proves that any specifiable IF exists via a term model. On the other hand we define erased, predicate, and relation algebras directly by induction on IIT specifications, thus skipping IF specifications entirely. We then prove the existence of erased types in an ad-hoc way, by assuming the existence of a few concrete inductive families. Well-formedness predicates and eliminator relations are then constructed by large elimination over erased types.

Von Raumer's proof of a general reduction method stops at the Σ-construction. He does define a general mapping from IIT codes to IF codes of their corresponding eliminator relations, however he does not prove functionality of the relations thus obtained, and therefore does not define the eliminators. On the other hand we provide a complete proof of left-totality for the eliminator relations we construct, from which we define the eliminators and prove their β-equalities.

Although the distinction between ambient theory/metatheory and target theory is hinted at in [vR19], our work is more explicit in the distinction between these two levels. This is a consequence of our requirements, which demand a clear separation between the *meta* level of the metatheory, where the theorems and proofs live and where extensional reasoning is not only allowed but often necessary, and the *encoding* level of the target theory, where extensionality is strictly controlled. This clear separation between the two levels is to prevent extensionality accidentally "leaking" into target-level constructions.

In [KKA19], Kaposi et al. describe a syntax for finitary QIITs. They show that the existence of the specification datatype, which is itself a QIIT, implies the existence of all finitary QIITs. The specification syntax is in the by now familiar type-theoretic style. The class of IITs considered is significantly different from ours: in addition to only targeting finitary constructors, they also consider quotients. Note that [KK18] also deals with equality constructors, however it does not consider reductions/encodings of any kind.

[KKA19] proves a reduction from all finitary QIITs to a single, "universal" QIIT, however one still needs some form of induction-induction to justify the existence of such QIIT. On the other hand the kind of reduction showcased in [vR19] and this thesis seeks to eliminate induction-induction completely, by reducing it to plain inductive types. This difference is perhaps not surprising, as the presence of quotients is a non-trivial addition to the already complex induction-induction; it therefore may not even be possible to completely reduce QIITs to inductive types/W-types without some primitive form of quotient in the encoding/target theory.

Bibliography

[ABK$^+$21] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, Christian Sattler, and Filippo Sestini. Constructing a universe for the setoid model. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures*, pages 1–21, Cham, 2021. Springer International Publishing.

[ABKT19] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. Setoid type theory—a syntactic translation. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 155–196, Cham, 2019. Springer International Publishing.

[AC19] Andreas Abel and Thierry Coquand. Failure of normalization in impredicative type theory with proof-irrelevant propositional equality. *Log. Methods Comput. Sci.*, 16, 2019.

[ACCL89] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit substitutions. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, page 31–46, New York, NY, USA, 1989. Association for Computing Machinery.

[ACD$^+$18] Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 293–310, Cham, 2018. Springer International Publishing.

[ACKS23] Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *Mathematical Structures in Computer Science*, page 1–56, 2023.

[AK16] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. *SIGPLAN Not.*, 51(1):18–29, January 2016.

[AKKvR18] Thorsten Altenkirch, Ambrus Kaposi, András Kovács, and Jakob von Raumer. Reducing inductive-inductive types to indexed inductive types. In José Espírito Santo and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs, TYPES 2018*. University of Minho, 2018.

[AKKvR19] Thorsten Altenkirch, Ambrus Kaposi, András Kovács, and Jakob von Raumer. Constructing inductive-inductive types via type erasure. In Marc Bezem, editor, *25th International Conference on Types for Proofs and Programs, TYPES 2019*. Centre for Advanced Study at the Norwegian Academy of Science and Letters, 2019.

[All87] Stuart Allen. *A Non-Type-Theoretic Semantics For Type-Theoretic Language*. PhD thesis, Cornell University, 1987.

[Alt99] Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, page 412, USA, 1999. IEEE Computer Society.

[AMS07] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *PLPV '07: Proceedings of the 2007 workshop on Programming languages meets program verification*, pages 57–68, New York, NY, USA, 2007. ACM.

[AR14a] Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving*, pages 27–44, Cham, 2014. Springer International Publishing.

[AR14b] Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. Technical report. `http://www.nuprl.org/html/Nuprl2Coq/`, 2014.

[BKS21] Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. Relative induction principles for type theories. `https://doi.org/10.48550/arXiv.2102.11649`, 2021.

[Bou18] Simon Boulier. *Extending Type Theory with Syntactical Models*. PhD thesis, IMT Atlantique, 2018.

[Cap04] Venanzio Capretta. A polymorphic representation of induction-recursion. `http://www.cs.nott.ac.uk/~pszvc/publications/induction_recursion.pdf`, 2004.

[Car86] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[Cha09] James Chapman. Type theory should eat itself. *Electron. Notes Theor. Comput. Sci.*, 228:21–36, jan 2009.

[de 72] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

[dMKA$^+$15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *2015 Conference on Automated Deduction*, pages 378–388. Springer, Cham, July 2015.

[DS99] Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 129–146, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Dyb95] Peter Dybjer. Internal type theory. In *International Workshop on Types for Proofs and Programs*, pages 120–134. Springer, 1995.

[Dyb97] Peter Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science*, 176(1):329–335, 1997.

[Dyb03] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65, 06 2003.

[GCST19] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proceedings of the ACM on Programming Languages*, pages 1–28, January 2019.

[Hof95] Martin Hofmann. *Extensional concepts in intensional type theory.* PhD thesis, University of Edinburgh, 1995.

[Hof96] Martin Hofmann. Conservativity of equality reflection over intensional type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 153–164, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Hof97] Martin Hofmann. *Syntax and Semantics of Dependent Types*, page 79–130. Publications of the Newton Institute. Cambridge University Press, 1997.

[HS97] Martin Hofmann and Thomas Streicher. Lifting Grothendieck universes. `https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf`, 1997.

[HS98] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.

[Hug21] Jasper Hugunin. Why Not W? In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs (TYPES 2020)*, volume 188 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:9, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Kap19] Ambrus Kaposi. Message to the Agda mailing list. `https://lists.chalmers.se/pipermail/agda/2019/011176.html`, 2019.

[KK18] Ambrus Kaposi and András Kovács. A Syntax for Higher Inductive-Inductive Types. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[KKA19] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019.

[KKK19] Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow embedding of type theory is morally correct. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 329–365, Cham, 2019. Springer International Publishing.

[KKL20] Ambrus Kaposi, András Kovács, and Ambroise Lafont. For finitary induction-induction, induction is enough. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for*

Proofs and Programs (TYPES 2019), volume 175 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:30, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[KX21] Ambrus Kaposi and Zongpu Xie. Quotient inductive-inductive types in the setoid model. In *27th International Conference on Types for Proofs and Programs, TYPES 2021*, 2021.

[Lao17] Alexis Laouar. A presheaf model of dependent type theory. Internal report. `https://perso.crans.org/alaouar/rapportm1.pdf`, 2016/2017.

[MAG$^+$13] Lorenzo Malatesta, Thorsten Altenkirch, Neil Ghani, Peter Hancock, and Conor McBride. Small induction recursion, indexed containers and dependent polynomials are equivalent, 2013. 11th International Conference on Typed Lambda Calculi and Applications, TLCA 2013.

[Mai09] Maria Emilia Maietti. A minimalist two-level foundation for constructive mathematics. *Annals of Pure and Applied Logic*, 160(3):319–354, 2009. Computation and Logic in the Real World: CiE 2007.

[Mar75] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73 – 118. Elsevier, 1975.

[Mar84] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in proof theory*. Bibliopolis, 1984.

[MS05] Maria Emilia Maietti and Giovanni Sambin. Toward a minimalist foundation for constructive mathematics. In *From Sets and Types to Topology and Analysis: Towards practicable foundations for constructive mathematics*. Oxford University Press, 10 2005.

[NF13] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.

[NPS90] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. Clarendon Press, USA, 1990.

[Pal98] Erik Palmgren. On universes in type theory. In *Twenty Five Years of Constructive Type Theory*. Oxford University Press, 10 1998.

[PE88] F. Pfenning and C. Elliott. Higher-order abstract syntax. *SIGPLAN Not.*, 23(7):199–208, jun 1988.

[PM93] Christine Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, TLCA '93, page 328–345, Berlin, Heidelberg, 1993. Springer-Verlag.

[SAG19] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. Cubical syntax for reflection-free extensional equality. In Herman Geuvers, editor, *Proceedings of the 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:25. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[Ses23] Filippo Sestini. Accompanying agda formalization - permanent repository. `https://doi.org/10.17639/nott.7291`, 2023.

[The13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[vR19] Jakob von Raumer. *Higher Inductive Types, Inductive Families, and Inductive-Inductive Types*. PhD thesis, University of Nottingham, 2019.

[vR22] Jakob von Raumer. Lean 4 IITs. `https://github.com/javra/iit`, 2022.