The Type Theory of Lean

Mario Carneiro

April 16, 2019

Abstract

This thesis is a presentation of dependent type theory with inductive types, a hierarchy of universes, with an impredicative universe of propositions, proof irrelevance, and subsingleton elimination, along with axioms for propositional extensionality, quotient types, and the axiom of choice. This theory is notable for being the axiomatic framework of the Lean theorem prover. The axiom system is given here in complete detail, including "optional" features of the type system such as let binders and definitions. We provide a reduction of the theory to a finitely axiomatized fragment utilizing a fixed set of inductive types (the W-type plus a few others), to ease the study of this framework.

The metatheory of this theory (which we will call Lean) is studied. In particular, we prove unique typing of the definitional equality, and use this to construct the expected set-theoretic model, from which we derive consistency of Lean relative to ZFC + {there are n inaccessible cardinals | $n < \omega$} (a relatively weak large cardinal assumption). As Lean supports models of ZFC with n inaccessible cardinals, this is optimal.

We also show a number of negative results, where the theory is less nice than we would like. In particular, type checking is undecidable, and the type checking as implemented by the Lean theorem prover is a decidable non-transitive underapproximation of the typing judgment. Non-transitivity also leads to lack of subject reduction, and the reduction relation does not satisfy the Church-Rosser property, so reduction to a normal form does not produce a decision procedure for definitional equality. However, a modified reduction relation allows us to restore the Church-Rosser property at the expense of guaranteed termination, so that unique typing is shown to hold.

Contents

1 Introduction

1.1 Type theory in programming languages

The history of types in mathematical logic dates back to Frege's *Begriffsschrift* [10], which establishes a notation system for what amounts to second-order logic with equality. Bertrand Russell discovered a paradox in Frege's system: The predicate $P(A) := \neg A(A)$ leads to a contradiction (or in set-theoretic notation, the set $S = \{x \mid x \notin x\}$ cannot be a set). In reaction, Ernst Zermelo resolved the contradiction by imposing a "size restriction" on sets, leading to Zermelo set theory and eventually to Zermelo-Fraenkel set theory (ZFC), which has become the gold standard for axiomatization in modern mathematics. This yields an untyped but stratified view of the universe of mathematical concepts.

Russell's own reaction to Russell's paradox was instead to impose a stratification on the language itself, rejecting the expressions $A(A)$ or $x \in x$ as "ill-typed". This line of reasoning says that A is not an object that predicates on objects of the same type as itself, so the notion is prima facie ill-formed. This idea is developed in *Principia Mathematica* [24] and Quine's New Foundations [21], but the most relevant application was to the simply typed λ-calculus [5] by Church (1940).

Somewhat independently, programming languages rediscovered the idea of a type [16]. Early programming languages had no explicit notion of type. Lisp used an evaluation model closely related to the untyped λ-calculus. FORTRAN (1956) had "modes" of expressions, either fixed or floating point. Algol 60 (1960) developed expressions and variables of type (**integer**, **real**, **Boolean**), and the extension Algol W by Wirth and Hoare (1966) developed a generative syntax for types including record types and typed references.

The logical and programming traditions are finally explicitly connected in the Curry-Howard isomorphism [11], which observed the connection between logical derivations (in the sequent calculus) and lambda terms in the simply typed λ-calculus. (In the same correspondence, Howard also discusses extensions to first order logic, with lambdas ranging over "number variables" $(\lambda x.\, F^{\beta})^{\forall x\, \beta}$ separate from typed lambdas $(\lambda X^{\alpha}.\, F^{\beta})^{\alpha \supset \beta}$.) But dependent type theory really begins in earnest with Per Martin-Löf [14], who set the foundations for Brouwer's intuitionistic type theory as an outgrowth of the simply typed λ-calculus with dependent types.

Martin-Löf describes how constructive type theory can be used in programming languages:

> By choosing to program in a formal language for constructive mathematics, like the theory of types, one gets access to the whole conceptual apparatus of pure mathematics, neglecting those parts that depend critically on the law of excluded middle, whereas even the best high level programming languages so far designed are wholly inadequate as mathematical languages (and, of course, nobody has claimed them to be so). In fact, I do not think that the search for logically ever more satisfactory high level programming languages can stop short of anything but a language in which (constructive) mathematics can be adequately expressed. [15]

This dream was converted to action by Coquand and Huet, who introduced the Calculus of Constructions (CoC) [6] and developed it into an interactive proof assistant Coq [4]. This type theory was extended with inductive types [8] to form the Calculus of Inductive Constructions (CIC) [19].

Lean [7] is a theorem prover based on CIC as well, with some subtle but important differences. The goal of this paper is to demonstrate the consequences of these differences, all taken together.

While CIC itself is well-studied [1, 2, 3], most papers study subsystems of the actual axiomatic system implemented in Coq, which might be called CIC^+ for its many small extensions added over the years. While we will not analyze CIC^+ in this paper, we will be able to analyze all the extensions that are in Lean CIC, so our proof of consistency is directly applicable to the full Lean kernel. (See section 2.8 for the possible issues that can come up in trying to extend this analysis to CIC^+.)

1.2 Set theoretic models of type theory

Martin-Löf type theory has a relatively obvious model, in which types are interpreted as sets, and terms of that type are interpreted as elements of the corresponding sets. Essentially, this amounts to treating "function types" like $A \to B$ as literal sets of ZFC-encoded functions (sets of ordered pairs). Since most of the type theoretic intuition and terminology is inherited from this context, from the point of view of standard mathematics it is reasonable to expect that this should work as a model, and conversely we can use this model to guide our expectations for the reasonableness of variations on the rules of type theory.

In this model the easiest way to accomodate propositions is to have a two element universe $\mathsf{Prop} = \{0, 1\}$, where the two elements are propositions $0 = \emptyset$ and $1 = \{\bullet\}$, where $\bullet$ is the "trivial proof of true". This makes the model *proof-irrelevant*, in the sense that proofs of a proposition are not distinguished in the model. This is great for type theories that have proof irrelevance in some form (also known as axiom K or uniqueness of identity proofs (UIP)), but contradicts homotopy type theory (HoTT) [22], since the axiom of univalence is inconsistent with UIP. (Since definitional UIP is an axiom of Lean, we will not pursue models of HoTT further in this paper.)

This model is also tailored for *impredicativity* of the universe of propositions. That is, the type theory allows propositions to quantify over elements in "large universes". To a mathematician accustomed to set theory, this may seem a non-issue, but impredicativity is quite axiomatically strong and corresponds to having "full powersets" in the ZFC sense. For predicative type theories, such as the axiom system used by Agda [18], the impredicative model will work but is in some sense "overkill"; here the preference is for models based on partial equivalence relations on terms, which avoids large cardinals.

Prop: Type
Type: Type1
Type1: Type2
Type2: Type3

Inductive types can be defined in all kind of universes

Universes of CIC are closed under function types and inductive types, which when translated to the set theoretic language implies that they are Grothendieck universes. If we limit our attention to levels of the cumulative hierarchy V_λ, this amounts to a requirement that λ be an inaccessible cardinal. Since Lean has an infinite sequence of universes this translates to having ω many inaccessible cardinals in the ZFC universe from which to build the model, and this is the main large cardinal axiom we require. It is not difficult to see that this assumption (or something with similar consistency strength) is necessary, because with a suitably defined inductive type we can construct a model of ZFC in each universe above the first one, and moreover we can define particular inaccessible cardinals in the larger models, so that we can have models of ZFC with n inaccessible cardinals.

Con -> consistency

In "Sets in Types, Types in Sets" [23], Werner demonstrates the equiconsistency of ZFC_ω and CIC_ω by showing that $\mathsf{ZFC}_n \vdash \mathrm{Con}(\mathsf{CIC}_{n+1})$ and $\mathsf{CIC}_{n+2} \vdash \mathrm{Con}(\mathsf{ZFC}_n)$, where ZFC_n is ZFC with n inaccessible cardinals and CIC_n is CIC with n universes, and ZFC_ω and CIC_ω are the unions of these theories over $n < \omega$. Werner's construction of a model of set theory in CIC (in Coq) has been replicated in Lean with only minor modifications, so we can also claim $\mathsf{Lean}_{n+2} \vdash \mathrm{Con}(\mathsf{ZFC}_n)$ for essentially the same reasons.

The present work establishes the reverse direction $\mathsf{ZFC}_{n+1} \vdash \mathrm{Con}(\mathsf{Lean}_{n+1})$, so that Lean_ω (or

ZFC_0 can construct arbitarily large(but not inaccessible) cardinals, CIC_1 has type0 can model all of these

just Lean) is also equiconsistent with ZFC_ω and CIC_ω. We don't attempt to be precise with the universe bounds, but if we wanted to get a result like Werner's $\mathsf{ZFC}_n \vdash \mathrm{Con}(\mathsf{CIC}_{n+1})$, we would have to assume an axiom of global choice in ZFC (i.e. there is a proper class choice function on the universe V) to interpret Lean's choice axiom.

To some degree one can view this work as merely an elaboration of Werner's work in the context of Lean in place of Coq. However, we believe that inductive types in CIC, and Lean, are more complicated than they appear from simple worked examples, and we wanted to ensure that we correctly model the entire language, including all the edge case features that interact in unusual ways. In fact, as we shall see, a combination of subsingleton eliminating inductive types and definitional proof irrelevance breaks the decidability of Lean's type system, making a number of desirable properties fail to hold. In the light of this, as well as some historical soundness bugs in Coq as a result of unusual features in inductive specifications or pattern matching [9], we felt it important to write down the complete axiomatic basis for Lean's type system, and work from there. See section 2 for the specification.

In "The not so simple proof irrelevant model of CC" [17], Miquel and Werner detail an issue that arises in proof irrelevant models such as the one described here. In short, without knowing the universe in which an expression or type lives, it becomes difficult to translate the Pi type over propositions differently than the Pi type over other universes, as one must, in order to ensure that $U_0 = \{\emptyset, \{\bullet\}\}$ can serve as the boolean universe of propositions. This issue arises here as well, and the key step in overcoming it is the unique typing property. While this is mostly trivial in the context of PTSs for [17], in Lean this is a tricky syntactic argument, proven in section 4. While it is inspired by the Tait–Martin-Löf proof of the Church–Rosser theorem [20], definitional proof irrelevance causes many new difficulties, and the proof is novel to our knowledge.

In [1], Barras uses a simple and ingenious trick to uniformize the treatment of the proof irrelevant universe of propositions with other universes - to use Aczel's encoding of functions, $f := \{(x, y) \mid x \in \mathrm{dom}(f) \wedge y \in f(x)\}$, which has the property that $(x \in A \mapsto \bullet) = \bullet$ if we interpret $\bullet$ as the empty set. This simple property means that we don't need to determine the sorts of types and elements in the construction, and so we can avoid the dependency on unique typing in the proof of soundness. So if our only goal was proving soundness we could skip section 4 entirely. Nevertheless, it is a useful property to have, and with it we can use the straightforward ZFC encoding for functions.

The remainder of the paper is organized as follows. Section 2 details the type system of Lean in formal notation. Section 3 does some basic metatheory of the type system, and in particular shows a number of negative results stemming from lack of decidability of the type system. Section 4 is the proof of unique typing of the type system (even including the undecidable bits). Section 5 shows how all inductive types can be reduced to a finite basis of 8 particular, basic inductive types. Section 6 is the soundness theorem, which constructs the aforementioned set theoretic model for the W basis in detail.

2 The axioms

2.1 Typing

The syntax of expressions is given by the following grammar:

$$\begin{aligned}\ell &::= u \mid 0 \mid S\ell \mid \max(\ell,\ell) \mid \operatorname{imax}(\ell,\ell)\\ e &::= x \mid \mathsf{U}_\ell \mid e\ e \mid \lambda x : e.\ e \mid \forall x : e.\ e\\ \Gamma &::= \cdot \mid \Gamma, x : e\end{aligned}$$

Here u is a universe variable, and x is an expression variable. The typing judgment is defined by the rules:

$$\boxed{\Gamma \vdash e : \alpha}$$

introduce variable, if alpha can be shown living in l-level universe, then in the extended context, variable x has type alpha

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_\ell \quad \Gamma \vdash e : \beta}{\Gamma, x : \alpha \vdash e : \beta} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_\ell}{\Gamma, x : \alpha \vdash x : \alpha} \qquad \frac{}{\vdash \mathsf{U}_\ell : \mathsf{U}_{S\ell}}$$

$$\frac{\Gamma \vdash e_1 : \forall x : \alpha.\ \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1\ e_2 : \beta[e_2/x]} \qquad \frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash \lambda x : \alpha.\ e : \forall x : \alpha.\ \beta}$$

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_{\ell_1} \quad \Gamma, x : \alpha \vdash \beta : \mathsf{U}_{\ell_2}}{\Gamma \vdash \forall x : \alpha.\ \beta : \mathsf{U}_{\operatorname{imax}(\ell_1,\ell_2)}} \qquad \frac{\Gamma \vdash e : \alpha \quad \Gamma \vdash \alpha \equiv \beta}{\Gamma \vdash e : \beta}$$

Each constant has a list of universe variables $\bar{u}$ that may appear in its type; these are substituted for given universe level expressions in $\tau_{\bar{u}}(c)[\bar{\ell}/\bar{u}]$.

For convenience, we will also define the following simple judgments:

$$\boxed{\Gamma \vdash \alpha\ \mathsf{type}} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_\ell}{\Gamma \vdash \alpha\ \mathsf{type}} \qquad \boxed{\vdash \Gamma\ \mathsf{ok}} \qquad \frac{}{\vdash \cdot\ \mathsf{ok}} \qquad \frac{\Gamma \vdash \alpha\ \mathsf{type}}{\vdash \Gamma, x : \alpha\ \mathsf{ok}}$$

2.2 Definitional equality

We will distinguish two notions of definitional equality: the "ideal" definitional equality, denoted $\alpha \equiv \beta$, and "algorithmic" definitional equality, denoted $\alpha \Leftrightarrow \beta$, which will imply $\alpha \equiv \beta$ and is what is actually checked by Lean.

$$\boxed{\Gamma \vdash e \equiv e'}$$

$$\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash e \equiv e} \qquad \frac{\Gamma \vdash e \equiv e'}{\Gamma \vdash e' \equiv e} \qquad \frac{\Gamma \vdash e_1 \equiv e_2 \quad \Gamma \vdash e_2 \equiv e_3}{\Gamma \vdash e_1 \equiv e_3}$$

$$\frac{\ell \equiv \ell'}{\vdash \mathsf{U}_\ell \equiv \mathsf{U}_{\ell'}} \qquad \frac{\Gamma \vdash e_1 \equiv e_1' : \forall x : \alpha.\ \beta \quad \Gamma \vdash e_2 \equiv e_2' : \alpha}{\Gamma \vdash e_1\ e_2 \equiv e_1'\ e_2'}$$

$$\frac{\Gamma \vdash \alpha \equiv \alpha' \quad \Gamma, x : \alpha \vdash e \equiv e'}{\Gamma \vdash \lambda x : \alpha.\ e \equiv \lambda x : \alpha'.\ e'} \qquad \frac{\Gamma \vdash \alpha \equiv \alpha' \quad \Gamma, x : \alpha \vdash \beta \equiv \beta'}{\Gamma \vdash \forall x : \alpha.\ \beta \equiv \forall x : \alpha'.\ \beta'}$$

$$(\beta)\ \frac{\Gamma, x : \alpha \vdash e : \beta \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash (\lambda x : \alpha.\ e)\ e' \equiv e[e'/x]} \qquad (\eta)\ \frac{\Gamma \vdash e : \forall y : \alpha.\ \beta}{\Gamma \vdash \lambda x : \alpha.\ e\ x \equiv e}$$

$$\frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p}{\Gamma \vdash h \equiv h'}$$

The notation $\Gamma \vdash e \equiv e' : \alpha$ in the application rule abbreviates $\Gamma \vdash e \equiv e' \wedge \Gamma \vdash e : \alpha \wedge \Gamma \vdash e' : \alpha$. The last rule is called proof irrelevance, which states that any two proofs of a proposition (a type in $\mathbb{P} := \mathsf{U}_0$) are equal. Equality of levels is defined in terms of an algorithmic inequality judgement $\ell \leq \ell' + n$ where $n \in \mathbb{Z}$ (abbreviated to $\ell \leq \ell'$ when $n = 0$):

$$\boxed{\ell \equiv \ell'} \qquad \frac{\ell \leq \ell' \quad \ell' \leq \ell}{\ell \equiv \ell'}$$

$$\boxed{\ell \leq \ell' + n}$$

$$\frac{n \geq 0}{0 \leq \ell + n} \qquad \frac{n \geq 0}{\ell \leq \ell + n}$$

$$\frac{\ell \leq \ell' + (n-1)}{S\ell \leq \ell' + n} \qquad \frac{\ell \leq \ell' + (n+1)}{\ell \leq S\ell' + n}$$

$$\frac{\ell \leq \ell_1 + n}{\ell \leq \max(\ell_1, \ell_2) + n} \qquad \frac{\ell \leq \ell_2 + n}{\ell \leq \max(\ell_1, \ell_2) + n} \qquad \frac{\ell_1 \leq \ell + n \quad \ell_2 \leq \ell + n}{\max(\ell_1, \ell_2) \leq \ell + n}$$

$$\frac{0 \leq \ell + n}{\mathrm{imax}(\ell_1, 0) \leq \ell + n} \qquad \frac{\max(\ell_1, S\ell_2) \leq \ell + n}{\mathrm{imax}(\ell_1, S\ell_2) \leq \ell + n}$$

$$\frac{\max(\mathrm{imax}(\ell_1, \ell_3), \mathrm{imax}(\ell_2, \ell_3)) \leq \ell + n}{\mathrm{imax}(\ell_1, \mathrm{imax}(\ell_2, \ell_3)) \leq \ell + n} \qquad \frac{\ell \leq \max(\mathrm{imax}(\ell_1, \ell_3), \mathrm{imax}(\ell_2, \ell_3)) + n}{\ell \leq \mathrm{imax}(\ell_1, \mathrm{imax}(\ell_2, \ell_3)) + n}$$

$$\frac{\max(\mathrm{imax}(\ell_1, \ell_2), \mathrm{imax}(\ell_1, \ell_3)) \leq \ell + n}{\mathrm{imax}(\ell_1, \max(\ell_2, \ell_3)) \leq \ell + n} \qquad \frac{\ell \leq \max(\mathrm{imax}(\ell_1, \ell_2), \mathrm{imax}(\ell_1, \ell_3)) + n}{\ell \leq \mathrm{imax}(\ell_1, \max(\ell_2, \ell_3)) + n}$$

$$\frac{\ell[0/u] \leq \ell'[0/u] + n \quad \ell[Su/u] \leq \ell'[Su/u] + n}{\ell \leq \ell' + n}$$

Although this definition looks complicated, it is most easily understood in terms of its semantics: A level takes values in $\mathbb{N}$, where $[\![0]\!] = 0$, $[\![S\ell]\!] = [\![\ell]\!] + 1$, $[\![\max(\ell_1, \ell_2)]\!] = \max([\![\ell_1]\!], [\![\ell_2]\!])$ and $[\![\mathrm{imax}(\ell_1, \ell_2)]\!] = \mathrm{imax}([\![\ell_1]\!], [\![\ell_2]\!])$, where $\mathrm{imax}(m, n)$ is the function such that $\mathrm{imax}(m, n+1) = \max(m, n+1)$ and $\mathrm{imax}(m, 0) = 0$. Then a level inequality $\ell \leq \ell + n$ holds if for all substitutions v of numerals for the variables in ℓ and ℓ', $[\![\ell]\!]_v \leq [\![\ell']\!]_v + n$. We will return to this in detail in section 6.

2.3 Reduction

The algorithmic definitional equivalence relation is defined in terms of a reduction operation on terms:

$$\boxed{\Gamma \vdash e \Leftrightarrow e'}$$

does not contain transitivity

$$\frac{}{\Gamma \vdash e \Leftrightarrow e} \qquad \frac{\Gamma \vdash e \Leftrightarrow e'}{\Gamma \vdash e' \Leftrightarrow e} \qquad \frac{\ell \equiv \ell'}{\Gamma \vdash \mathsf{U}_\ell \Leftrightarrow \mathsf{U}'_\ell}$$

$$\frac{\Gamma \vdash \alpha \Leftrightarrow \alpha' \quad \Gamma, x : \alpha \vdash e \Leftrightarrow e'}{\Gamma \vdash \lambda x : \alpha.\ e \Leftrightarrow \lambda x : \alpha'.\ e'} \qquad \frac{\Gamma \vdash \alpha \Leftrightarrow \alpha' \quad \Gamma, x : \alpha \vdash e \Leftrightarrow e'}{\Gamma \vdash \forall x : \alpha.\ e \Leftrightarrow \forall x : \alpha'.\ e'}$$

$$\frac{\Gamma \vdash e : \forall x : \alpha.\ \beta \quad \Gamma, x : \alpha \vdash e\ x \Leftrightarrow e'\ x}{\Gamma \vdash e \Leftrightarrow e'} \qquad \frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p' \quad \Gamma \vdash p \Leftrightarrow p'}{\Gamma \vdash h \Leftrightarrow h'}$$

proof irrelevance rule

e has dependent function type

we do not care the content of proof, but only consider about their types equality

$$\frac{\Gamma \vdash e_1 \Leftrightarrow e_1' \quad \Gamma \vdash e_2 \Leftrightarrow e_2'}{\Gamma \vdash e_1\ e_2 \Leftrightarrow e_1'\ e_2'} \qquad \frac{e \rightsquigarrow k \quad \Gamma \vdash k \Leftrightarrow e'}{\Gamma \vdash e \Leftrightarrow e'}$$

wavy arrow means expression can be reduced one step further

$\forall x : \alpha.\ \beta$ is a type, where beta(as a type) depends on variable x whose type is alpha

In this judgment the transitivity rule is notably absent. Most of the congruence rules remain except for the β rule, and these constitute all the "easy" cases of definitional equality. The η rule is replaced with an extensionality principle. (This is justified because if $e\ x \equiv e'\ x$ then $\lambda x : \alpha.\ e\ x \equiv \lambda x : \alpha.\ e'\ x$, so $e \equiv e'$ by the η rule.) When the other rules fail to make progress, we use the head reduction relation $e \rightsquigarrow^* k$ to apply the β rule as well as the δ, ι, ζ rules which are discussed in their own section.

$$\boxed{e \rightsquigarrow e'} \qquad \frac{e_1 \rightsquigarrow e_1'}{e_1\ e_2 \rightsquigarrow e_1'\ e_2} \qquad \frac{}{(\lambda x : \alpha.\ e)\ e' \rightsquigarrow e[e'/x]}$$

We will add more rules to this list as we introduce new constructs, but this completes the description of the base dependent type theory foundation for Lean.

2.4 let binders (ζ reduction)

The first and simplest extension to this language is to add support for a let binder. We will define the expression $\mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e$ to be equivalent to $e[e'/x]$. (The rule asserting this equality is called ζ-reduction.) This differs from the expression $(\lambda x : \alpha.\ e)\ e'$ in that this expression requires $\lambda x : \alpha.\ e$ to be well-typed, while in the let binder we will be able to make use of a definitional equality $x \equiv e'$ while type-checking the body of e. One could imagine extending the context with such definitional equalities, but Lean takes a simpler approach and simply zeta expands these binders when necessary for checking. can introduce x == e' predicate

Adding let binders to the language entails adding the following clauses to the judgments of the previous sections:

$$e ::= \cdots \mid \mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e$$

$$\cdots \qquad \frac{\Gamma \vdash e' : \alpha \quad \Gamma \vdash e[e'/x] : \beta}{\Gamma \vdash \mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e : \beta}$$

$$\cdots \qquad (\zeta)\ \frac{\Gamma \vdash e' : \alpha \quad \Gamma \vdash e[e'/x] : \beta}{\Gamma \vdash \mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e \equiv e[e'/x]}$$

$$\cdots \qquad \frac{}{\mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e \rightsquigarrow e[e'/x]}$$

Note that we don't have any congruence rules for let. We simply unfold it whenever we need to check anything about it. It is easy to see that this is a conservative extension, because we can replace $\mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e$ with $e[e'/x]$ and remove any ζ-reduction steps in a whnf derivation to recover the original system.

2.5 Definitions (δ reduction)

There are two kinds of constants in lean: those with definitions and primitive constants. Both have the form $c_{\bar{u}}$ where c is a new name and $\bar{u}$ is a list of universe variables, but a definition will also have a reduction step (called δ reduction) associated with it. constants can be used in any level of universe

For a constant definition $\mathsf{constant}\ c_{\bar{u}} : \alpha$ to be admissible, we require that $\vdash \alpha : \mathsf{U}_\ell$, where the universe variables in α and ℓ are contained in $\bar{u}$. Similarly, a definition is specified by a clause $\mathsf{def}\ c_{\bar{u}} : \alpha := e$, which is admissible when $\vdash e : \alpha$ and the universe variables in e and α are contained in $\bar{u}$. Let $\tau_{\bar{\ell}}(c) = \alpha[\bar{\ell}/\bar{u}]$ and $v_{\bar{\ell}}(c) = e[\bar{\ell}/\bar{u}]$ denote the type and value of the definition after substitution for the universe variables. We add the following rules to the system:

$$e ::= \cdots \mid c_{\bar{u}}$$

constant declare: constant List_u : Type_u → Type_u

definition delcare: def id_u : ∀ (A : Type_u), A → A := λ A. λ x. x

$$\frac{}{\vdash c_{\bar{\ell}} : \tau_{\bar{\ell}}(c)} \qquad \frac{\ell_1 \equiv \ell'_1 \ \dots \ \ell_n \equiv \ell'_n}{\vdash c_{\bar{\ell}} \equiv c_{\bar{\ell}'}} \qquad \frac{\ell_1 \equiv \ell'_1 \ \dots \ \ell_n \equiv \ell'_n}{\Gamma \vdash c_{\bar{\ell}} \Leftrightarrow c_{\bar{\ell}'}}$$

Furthermore, for definitions, we add the following additional rules:

$$(\delta)\ \frac{}{\vdash c_{\bar{\ell}} \equiv v_{\bar{\ell}}(c)} \qquad \frac{}{c_{\bar{\ell}} \rightsquigarrow v_{\bar{\ell}}(c)}$$

It is similarly easy to see that a definition is a conservative extension, because we can replace $c_{\bar{\ell}}$ with $v_{\bar{\ell}}(c)$ everywhere and remove any δ-reduction steps to get a derivation which doesn't use the definition. This argument of course does not extend to constant, which has no reduction rules and so is simply an axiomatic extension of the system. We will discuss various consistent and conservative extensions by constants, when definitions will not suffice for technical reasons.

2.6 Inductive types

2.6.1 Inductive specifications

Inductive types are by far the most complex feature of Lean's axiomatic system, and moreover are very tricky to prove properties about due to their notational complexity. We will define a syntax for defining inductive types, and judgments for showing that they are admissible.

$$K ::= 0 \mid (c : e) + K$$

This is the type of an inductive specification, which is a list of introduction forms with name c and type e. We will write $(c : \alpha)$ for the single constructor form $(c : \alpha) + 0$, and abbreviate the whole sequence as $\sum_i (c_i : \alpha_i)$.

Let the notation $(x :: \alpha)$, called a "telescope", denote a dependent sequence of binders $x_1 : \alpha_1, x_2 : \alpha_2, \dots, x_n : \alpha_n$. This will be used in contexts, on the left $(\Gamma, x :: \alpha \vdash e : \beta)$ as well as on the right $(\Gamma \vdash x :: \alpha)$; this latter expression means that $\Gamma \vdash x_1 : \alpha_1$, and $\Gamma, x_1 : \alpha_1 \vdash x_2 : \alpha_2$, and so on up to
$\Gamma, x_1 : \alpha_1, \dots, x_{n-1} : \alpha_{n-1} \vdash x_n : \alpha_n$. It will also be used to abbreviate sequences of λ and $\forall$ as in $\lambda x :: \alpha.\ \beta = \lambda x_1 : \alpha_1 \dots \lambda x_n : \alpha_n.\ \beta$. If $e :: \alpha$ and $f : \forall x :: \alpha.\ \beta$, then $f\ e : \beta[e/x]$ denotes the sequence of applications $f\ e_1 \dots e_n$.

F is the type of type constructors that take parameters x :: α and produce types in universe U_ ℓ

A specification K is typechecked in a context of a variable $t : F$ where $F = \forall x :: \alpha.\ \mathsf{U}_\ell$ is a family of sorts (so t is a family of types). The result will be the recursive type $\mu t : F.\ K$, which roughly satisfies the equivalence $\mu t : F.\ K \simeq K[\mu t : F.\ K/t]$. A specification is a sequence of constructors:

$$\boxed{\Gamma; t : F \vdash K\ \mathsf{spec}} \qquad \frac{\Gamma \vdash x :: \alpha \quad \Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \beta_i\ \mathsf{ctor}}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \sum_i (c_i : \beta_i)\ \mathsf{spec}}$$

A constructor is a sequence of arguments ending in an application with head t:

$$\boxed{\Gamma; t : F \vdash \alpha\ \mathsf{ctor}} \qquad \frac{\Gamma \vdash e :: \alpha}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash t\ e\ \mathsf{ctor}}$$ base case

$$\frac{\Gamma \vdash \beta : \mathsf{U}_{\ell'} \quad \ell' \le \ell \quad \Gamma, y : \beta; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \tau\ \mathsf{ctor}}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \forall y : \beta.\ \tau\ \mathsf{ctor}}$$ non-recursive inductive case

$$\frac{\Gamma \vdash \gamma :: \mathsf{U}_{\ell'} \quad \Gamma, z :: \gamma \vdash e :: \alpha \quad \mathrm{imax}(\ell', \ell) \le \ell \quad \Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \tau\ \mathsf{ctor}}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash (\forall z :: \gamma.\ t\ e) \to \tau\ \mathsf{ctor}}$$ recursive inductive case

There are two kinds of arguments, represented by the two inductive cases here. The first kind is a nonrecursive argument. The type of this argument must not mention t, but it can be used in the types of later arguments. A recursive argument has the type $\forall z :: \gamma.\ t\ e$, and cannot be referenced in later arguments.

With the definition of spec in hand, we can finally define the type constructor and introduction operator:

$$e ::= \cdots \mid \mu x : e.\ K \mid c_{\mu x:e.K} \mid \mathsf{rec}_{\mu x:e.K}$$

$$\frac{\Gamma; t : F \vdash K\ \mathsf{spec}}{\Gamma \vdash \mu t : F.\ K : F} \qquad \frac{\Gamma; t : F \vdash K\ \mathsf{spec} \quad (c : \alpha) \in K}{\Gamma \vdash c_{\mu t:F.K} : \alpha[\mu t : F.\ K/t]}$$

In Lean, $\mu t : F.\ K$ and $c_{\mu t:F.K}$ are implemented as additional axiomatic constant symbols (with no free variables, by abstracting over the variables in Γ). Having them as binders here makes the substitution story more complicated, so we will treat $\mu t : F.\ K$ as simply a nice syntax for $(\lambda x :: \Gamma.\ \mu t : F.\ K)\ x$, so that substitutions do not affect F and K.

Before we get to the general definition of the eliminator, let us review an example: the natural numbers. The natural numbers are defined in the above format as $\mathbb{N} := \mu N : \mathsf{U}_1.\ (z : N) + (s : N \to N)$, yielding constructors $z_{\mathbb{N}} : \mathbb{N}$ (zero) and $s_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ (successor). The eliminator for $\mathbb{N}$ looks like this:

$$\mathsf{rec}_{\mathbb{N}} : \forall (C : \mathbb{N} \to \mathsf{U}_u).\ C\ z_{\mathbb{N}} \to (\forall x : \mathbb{N}.\ C\ x \to C\ (s_{\mathbb{N}}\ x)) \to \forall n : \mathbb{N}.\ C\ n$$

There are three components to this definition: the "motive" C, which will be a type family over the inductive type family just constructed, the "minor premises" $C\ z_{\mathbb{N}}$ and $\forall x : \mathbb{N}.\ C\ x \to C\ (s_{\mathbb{N}}\ x)$, which asserts that C preserves each constructor, and the "major premise" $n : \mathbb{N}$ which then produces an element of the type family $C\ n$. We want to generalize each of these pieces.

eliminator enforces structural recursion, which means we can only make recursive calls on structurally smaller arguments

2.6.2 Large elimination

One additional point requires noting in the previous example: The type family C ranges over an arbitrary universe u. This is called *large elimination* because it means that one can use recursion over natural numbers to produce functions in large universes. By contrast, the existential quantifier (defined as an inductive predicate) does not have large elimination, meaning that the motive only ranges over $\mathbb{P}$ instead of U_u.

There are two reasons an inductive type can be large eliminating:

1. The type family $t : \forall x :: \alpha.\ \mathsf{U}_\ell$ lives in a universe $1 \le \ell$. (This means that ℓ is not zero for any values of the parameters.) $\mathbb{N}$ falls into this category.
2. The type family has at most one constructor, and all the non-recursive arguments to the constructor are either propositions or directly appear in the output type. This is called *subsingleton (SS) elimination*, and is relevant for the definition of equality as a large eliminating proposition.

Here it is again with an explicit judgment:

$$\boxed{\Gamma; t : F \vdash K\ \mathsf{LE}}$$

$$\frac{1 \le \ell}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash K\ \mathsf{LE}} \qquad \frac{}{\Gamma; t : F \vdash 0\ \mathsf{LE}} \qquad \frac{\Gamma; t : F \vdash \alpha\ \mathsf{LE\ ctor}}{\Gamma; t : F \vdash (c : \alpha)\ \mathsf{LE}}$$

$$\boxed{\Gamma; t : F \vdash \alpha\ \mathsf{LE\ ctor}}$$

$$\frac{}{\Gamma; t : F \vdash t\ e\ \mathsf{LE\ ctor}} \qquad \frac{\Gamma, t : F \vdash \alpha : \mathbb{P} \quad \Gamma, x : \alpha; t : F \vdash \beta\ \mathsf{LE\ ctor}}{\Gamma; t : F \vdash \forall x : \alpha.\ \beta\ \mathsf{LE\ ctor}}$$

$$\frac{\Gamma; t : F \vdash \beta\ \mathsf{LE\ ctor}}{\Gamma; t : F \vdash (\forall z :: \gamma.\ t\ e) \to \beta\ \mathsf{LE\ ctor}}$$

$$\frac{y \in e \quad \Gamma, y : \beta; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \forall z :: \gamma.\ t\ e\ \mathsf{LE\ ctor}}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \forall y : \beta.\ \forall z :: \gamma.\ t\ e\ \mathsf{LE\ ctor}}$$

In the final rule, $y \in e$ means that y is one of the elements of the sequence $e :: \alpha$. Intuitively, you should think of these rules as ensuring that the inductive type contains at most one element: With multiple constructors or a non-propositional argument, you could inhabit the type with more than one element, unless the argument to the constructor is also a parameter to the type family, in which case each distinct element of the argument type maps to a different member of the inductive type family. The equality type is defined with the following signature:

$$\alpha : \mathsf{U}_\ell, a : \alpha \vdash \mathsf{eq}_a := \mu t : \alpha \to \mathbb{P}.\ (\mathsf{refl} : t\ a)$$

and although it is a type family over $\mathbb{P}$ (so it fails the first reason to be large eliminating), it has exactly one constructor, with no arguments, so it is large eliminating. Another important large eliminating type is the accessibility relation, which is the source of proof by well-founded recursion:

$$\begin{aligned}\alpha : \mathsf{U}_\ell, r : \alpha \to \alpha \to \mathbb{P} \vdash \mathsf{acc}_r := \mu A : \alpha \to \mathbb{P}.\\ (\mathsf{intro} : \forall x : \alpha.\ (\forall y : \alpha.\ r\ y\ x \to A\ y) \to A\ x)\end{aligned}$$

Here we have subsingleton elimination because the nonrecursive argument $x : \alpha$ appears in the target type $A\ x$.

2.6.3 The recursor

To give a uniform description of the recursor and operations on it, let us label all the parts of an inductive definition $\mu t : F.\ K$.

$$\begin{aligned}F &= \forall a :: \alpha.\ \mathsf{U}_\ell\\ P &= \mu t : F.\ K\\ K &= \textstyle\sum_c (c : \forall b :: \beta.\ t\ p[b])\\ u :: \gamma \subseteq b :: \beta &\text{ is the subsequence of recursive arguments}\\ \text{with } \gamma_i &= \forall x :: \xi_i.\ P\ \pi_i[b, x].\end{aligned}$$

Here $\Gamma, b :: \beta \vdash p[b] :: \alpha$ is a sequence of terms depending on the nonrecursive arguments in $b :: \beta$, and $\Gamma, b :: \beta, x :: \xi_i \vdash \pi_i[b, x] :: \alpha$ is also a sequence of terms. Now the type of the recursor is:

$$\frac{\Gamma, t : F \vdash K\ \mathsf{spec}}{\Gamma \vdash \mathsf{rec}_P : \forall C : \kappa.\ \forall e :: \varepsilon.\ \forall a :: \alpha.\ \forall z : P\ a.\ C\ a\ z}$$

where:

- $\kappa = \forall a :: \alpha.\ P\ a \to \mathsf{U}_u$ where u is a fresh universe variable if $\Gamma; t : F \vdash K\ \mathsf{LE}$, otherwise $\kappa = \forall a :: \alpha.\ P\ a \to \mathbb{P}$,
- ε is a sequence of the same length as K, where $\varepsilon_c = \forall b :: \beta.\ \forall v :: \delta.\ C\ p[b]\ (c\ b)$,
- δ is a sequence of the same length as γ, where $\delta_i = \forall x :: \xi_i.\ C\ \pi_i[b, x]\ (u_i\ x)$.

2.6.4 The computation rule (ι reduction)

There is one more part to the definition of an inductive type: the so called ι rule. This states that a recursor evaluated on a constructor gives the corresponding case. For example, for $\mathbb{N}$ we have the rules:

$$\mathsf{rec}_{\mathbb{N}}\ C\ a\ f\ z_{\mathbb{N}} \equiv a$$
$$\mathsf{rec}_{\mathbb{N}}\ C\ a\ f\ (s_{\mathbb{N}}\ n) \equiv f\ n\ (\mathsf{rec}_{\mathbb{N}}\ C\ a\ f\ n)$$

In general, using the same names as in the previous section, we have the following computational rule corresponding to $(c : \forall b :: \beta.\ t\ p[b])$:

$$\frac{\Gamma, t : F \vdash K\ \mathsf{spec}}{\Gamma, C : \kappa, e :: \varepsilon, b :: \beta \vdash \mathsf{rec}_P\ C\ e\ p[b]\ (c\ b) \equiv e_c\ b\ v}$$

where $v :: \delta$ is defined as $v_i = \lambda x :: \xi_i.\ \mathsf{rec}_P\ C\ e\ \pi_i[b, x]\ (u_i\ x)$. (Technically, the reduction rule is all substitution instances of this rule for all the variables left of the turnstile.) This is also implemented as a reduction rule:

$$\frac{}{\mathsf{rec}_P\ C\ e\ p[b]\ (c\ b) \rightsquigarrow e_c\ b\ v}$$

This rule suffices for the theoretical presentation, but there is a second reduction rule called "K-like reduction" used for subsingleton eliminators. It can be thought of as a combination of proof irrelevance to change the major premise into a constructor followed by the iota rule.

$$\frac{F = \forall a :: \alpha.\ \mathbb{P}}{\mathsf{rec}_P\ C\ e\ p[b]\ h \rightsquigarrow e_c\ b\ v}$$

This rule only applies when all the variables in b are actually on the LHS, which is the reason for the peculiar requirements on subsingleton eliminators. If b_i appears in the parameters for its type, that means that $p_j[b] = b_i$ for some j, and so b_i is on the LHS.

The foremost example of this is known in the literature as axiom K, which is the reason for the name "K-like reduction", which is this principle applied to the equality type:

$$\mathsf{rec}_{a=}\ C\ x\ a\ h \equiv x$$

Here $\mathsf{rec}_{a=}\ C : a = b \to C\ a \to C\ b$ is the substitution principle of equality (suppressing the dependence of C on the proof argument), and the computation rule says that "casting" $x : C\ a$ over an equality $h : a = a$ produces x again.

2.7 Non-primitive axioms

All the axioms mentioned thus far are built into Lean so that they are valid even before the first line of code. There are three more axioms that are defined later:

2.7.1 Quotient types

Given a type $\alpha : \mathsf{U}_u$ and a relation $R : \alpha \to \alpha \to \mathbb{P}$, the quotient α/R represents the largest type with a surjection $\mathsf{mk}_R : \alpha \to \alpha/R$ such that two elements which are R-related are identified in the quotient. Formally, we have the following constants (all of which have two extra arguments for α and R):

$$\begin{aligned}
\alpha/R &: \mathsf{U}_u \\
\mathsf{mk}_R &: \alpha \to \alpha/R \\
\mathsf{sound}_R &: \forall x\ y : \alpha.\ R\ x\ y \to \mathsf{mk}_R\ x = \mathsf{mk}_R\ y \\
\mathsf{lift}_R &: \forall \beta : \mathsf{U}_v.\ \forall f : \alpha \to \beta.\ (\forall x\ y : \alpha.\ R\ x\ y \to f\ x = f\ y) \to \alpha/R \to \beta \\
\mathsf{lift}_R\ \beta\ f\ h\ (\mathsf{mk}_R\ a) &\rightsquigarrow f\ a
\end{aligned}$$

Because the last rule is a computational rule, not a constant, and Lean does not support adding computational rules to the kernel, this is a "semi-builtin" axiom; one has the option to disable quotient types, or to enable them and get the computational rule. Also, only sound_R is considered an axiom here, even though all four are undefined constants, because the other constants and the computational rule would all be satisfied with the definitions $\alpha/R := \alpha$, $\mathsf{mk}_R\ a := a$, $\mathsf{lift}_R\ f\ h := f$. As a terminological note, the rule $\mathsf{lift}_R\ f\ h\ (\mathsf{mk}_R\ a) \rightsquigarrow f\ a$ is also referred to as an ι reduction rule.

2.7.2 Propositional extensionality

The axiomatics of the Calculus of Inductive Constructions (CIC) in general leave equality of types in a universe almost completely unspecified, so that most of these statements are left undecided. For example, the notation $\mu t : F.\ K$ defined here for inductive types seems to suggest that the type is determined by F and K, but in fact in Lean you can write exactly the same inductive definition twice and get two possibly distinct (but isomorphic) types. (We could repair our construction here by marking a recursive type with an arbitrary name or number $\mu_i t : F.\ K$ so that we can make such "mirror copy" types.)

However this sort of agnosticism is quite annoying to work with in practice when dealing with propositions, for which we would like to use the substitution axiom of equality to substitute equivalent propositions. To that end, the propositional extensionality axiom says that propositions that imply each other are equal:

$$\mathsf{propext} : \forall p\ q : \mathbb{P}.\ (p \leftrightarrow q) \to p = q$$

2.7.3 Axiom of choice

The axiom of choice in Lean is expressed as a global choice function, and is simply stated by saying that there is a function from proofs that α is nonempty to α itself. We need the definition of $\mathsf{nonempty}$ for this:

$$\begin{aligned}
\mathsf{nonempty} &:= \lambda \alpha : \mathsf{U}_u.\ \mu t : \mathbb{P}.\ (\mathsf{intro} : \alpha \to t) \\
\mathsf{choice} &: \forall \alpha : \mathsf{U}_u.\ \mathsf{nonempty}\ \alpha \to \alpha
\end{aligned}$$

From the axiom of choice, the law of excluded middle is derived (it is not stated as a separate axiom).

2.8 Differences from Coq

As mentioned in the introduction, Coq is a theorem prover also based on the Calculus of Constructions with inductive types (CIC), and it is quite old and well studied [1, 2, 3, 6, 12, 13, 17]. So a natural question is to what degree Lean and CIC are similar, and whether proofs that apply to one system generalize, straightforwardly or otherwise, to the other. See [12] for a concise description of the proof theory of CIC. The following is a summary of differences with Lean's axiomatization, and their effects on the theorems here:

1. Coq has universe cumulativity. That is, the definitional equality relation is replaced by a cumulativity relation $\preceq$ that is roughly the same, except that $\Gamma \vdash \mathsf{U}_i \preceq \mathsf{U}_j$ when $i \leq j$. This breaks the unique typing theorem theorem 4.1, and it is not clear whether there is an adequate replacement in conjunction with all the other axioms of Lean. Luo [13] shows that a large subset of CIC including cumulative universes retains good type theoretic properties, including strong normalization, from which an analogue of unique typing can be derived.
2. Gallina, the underlying core syntax of Coq, uses primitives fix and match to implement inductive types, rather than rec as is done here, and this is difference usually reflected in theoretical presentations as well. The difference is that while rec performs structural recursion over an inductive type, fix performs unbounded recursion, while match does (primitive) pattern matching over inductive types. In order to prevent infinite recursion and inconsistency as a result, the body of a fix must be typechecked with a modified typing judgment to ensure that all recursive calls are to elements generated by a match on the input.
   While in theory these approaches are equivalent, the fix/match approach is more expressive, and the equivalence is sensitive to the exact rules available in both systems. Lean addresses this mismatch by allowing definitions using (effectively) fix and match at the user level, and compiling these away to recursors in the kernel language.
3. Definitions in Lean are universe polymorphic, in the sense that they may contain free universe variables that are implicitly universally quantified at the point of definition, and applications of the constants include substitutions for all the universe variables involved in the definition. Coq definitions live in "indefinite universes" – that is, each constant lives in a concrete universe but the level of this universe is held variable globally over the whole database, and using constants together generates level inequalities as side conditions that are maintained as a partial order. Coq reports an error if this order becomes inconsistent, i.e. there is no assignment of natural numbers to these variables that respects all the side conditions.
   There are Lean terms that cannot be checked in Coq with this approach, because Lean can reuse the same constant at two different levels while Coq has to resolve both instances of the constant to the same level. But this does not affect the set of provable theorems, since "universe polymorphism is a luxury"; for a concrete theorem at a fixed universe level we may make duplicates of Coq constants as necessary to represent different instantiations of Lean constants.
4. Coq inductive types allow "non-uniform parameters". These are parameters that vary subject to the restriction that they appear as is in each constructor's target type. These can be encoded using regular inductive types.
5. Coq also supports mutual inductives, nested inductive types, and coinductive types. These can all be encoded using regular inductives, although some definitional equalities may fail to hold in the encodings.

6. On the other hand, Lean supports definitional proof irrelevance, while Coq merely has an axiom that asserts this as a propositional equality. This is a major departure for the theory, and the reason why the counterexamples in section 3.1 don't work in Coq.
7. Lean supports quotient types with a definitional reduction rule, but Coq doesn't. The Coq ecosystem has compensated for this by using *setoids* in place of types in many places, which are types with a designated equivalence relation that plays the role of equality. Although we have not investigated this, it should be possible to eliminate quotients from Lean entirely by using setoids instead. (There are good ergonomic reasons to have quotient types though, lest we end up in "setoid hell".)
8. Lean offers (and de facto uses) three axioms, for propositional extensionality, quotient types and the axiom of choice. Coq has a comparatively large list of common axioms:
   - Proof irrelevance and axiom K are propositional versions of Lean's definitional proof irrelevance. They hold in Lean "with no axioms".
   - Propositional extensionality is the same in Coq and Lean.
   - Functional extensionality is proven in Lean as a consequence of propositional extensionality and quotient types.
   - Coq has many variations on the law of excluded middle – $P \vee \neg P$, $P = \mathsf{true} \vee P = \mathsf{false}$, and $P + \neg P$ (using a sum type). The first is excluded middle, the second is propositional degeneracy, which follows from excluded middle and propositional extensionality, and the third follows from excluded middle and the axiom of choice. In Lean all of these are proven using the axiom of choice.
   - The axiom of choice can be stated as $(\forall x, \exists y, R(x,y)) \to (\exists f, \forall x, R(x, f(x)))$ or $\exists f, \forall x, (\exists y, R(x,y)) \to R(x, f(x))$. These assert the existence of choice functions over limited domains, which is of course implied by a global choice function as with Lean's $\mathsf{choice} : \mathsf{nonempty}\ \alpha \to \alpha$.
   - Indefinite description, $(\exists x, P(x)) \to \Sigma x, P(x)$, is equivalent to Lean's choice.
   - Hilbert's epsilon, $\epsilon : (\alpha \to \mathsf{Prop}) \to \alpha$ such that $(\exists x, P(x)) \to P(\epsilon(P))$, is also equivalent to choice.

   So all of Coq's axioms taken together are implied by Lean's axioms, and the converse is true except for definitional proof irrelevance and a computation rule for quotient types. (One can build set-quotients in Coq as well as Lean, but they lack the computation rule.)

3 Properties of the type system

A theorem we would like to have of Lean's type system is that it is consistent, and sound with respect to some semantics in a well understood axiom system such as ZFC. Moreover, we want to relate this to Lean's actual typechecker, in the sense that anything Lean verifies as type-correct will be derivable in this axiom system and hence Lean will not certify a contradiction. But first we must understand some aspects of the type system itself, before relating it to other systems.

It is important to note that *Lean's typechecker is not complete.* Obviously Lean can fail on correct theorems due to, say, running out of resources, but the "algorithmic equality" relation does not validate all definitional equalities. In fact, we can show that definitional equality as defined here is undecidable.

3.1 Undecidability of definitional equality

Recall the type acc from section 2.6.2:

$$\mathsf{acc}_< := \mu A : \alpha \to \mathbb{P}.\ (\mathsf{intro} : \forall x : \alpha.\ (\forall y : \alpha.\ y < x \to A\ y) \to A\ x)$$

(We are fixing a type α and a relation $< : \alpha \to \alpha \to \mathbb{P}$ here.) Informally, we would read this as: "x is $<$-accessible if for all $y < x$, y is $<$-accessible". Accessibility is then inductively generated by this clause. If every $x : \alpha$ is accessible, then $<$ is a well-founded relation. One interesting fact about acc is that we can project out the argument given a proof of $\mathsf{acc}\ x$:

$$\begin{aligned}
&\mathsf{inv}_x : \mathsf{acc}\ x \to \forall y : \alpha.\ y < x \to \mathsf{acc}\ y \\
&\mathsf{inv}_x := \lambda a : \mathsf{acc}\ x.\ \lambda y : \alpha.\ \mathsf{rec}_{\mathsf{acc}}\ (\lambda z.\ y < z \to \mathsf{acc}\ y) \\
&\qquad (\lambda z.\ \lambda h : (\forall w.\ w < z \to \mathsf{acc}\ w).\ \lambda_.\ h\ y)\ x\ a
\end{aligned}$$

Note that the output type of inv_x is the same as the argument to $\mathsf{intro}\ x$. Thus, we have

$$a \equiv \mathsf{intro}_{\mathsf{acc}}\ x\ (\mathsf{inv}_x\ a)$$

by proof irrelevance.

Why does this matter? Normally, any proof of $\mathsf{acc}\ x$ could only be unfolded finitely many times by the very nature of inductive proofs, but if we are in an inconsistent context, it is possible to get a proof of wellfoundedness which isn't actually wellfounded, and we can end up unfolding it forever.

To show how to get undecidability from this, suppose $P : \mathbb{N} \to \mathbf{2}$ is a decidable predicate, such as $P\ n :=$ "Turing machine M runs for at least n steps without halting", for which $P\ n$ is decidable but $\forall n.\ P\ n$ is not. Let $>$ be the standard greater-than function on $\mathbb{N}$ (which is not well-founded). We define a function $f : \forall n.\ \mathsf{acc}_>\ n \to \mathbf{1}$ as follows:

$$\begin{aligned}
f := \mathsf{rec}_{\mathsf{acc}}\ (\lambda_.\ \mathbf{1})\ (\lambda n\ _\ (g : \forall y.\ y > x \to \mathbf{1}).& \\
\mathsf{if}\ P\ n\ \mathsf{then}\ g\ (n+1)\ (p\ n)\ \mathsf{else}\ ()&
\end{aligned}$$

where $p\ n$ is a proof of $n < n + 1$. Of course this whole function is trivial since the precondition $\mathsf{acc}_> n$ is impossible, but definitional equality works in all contexts, including inconsistent ones. This function evaluates as:

$$f\ n\ (\mathsf{intro}_{\mathsf{acc}}\ n\ h) \rightsquigarrow^* \mathsf{if}\ P\ n\ \mathsf{then}\ f\ (n+1)\ (h\ (n+1)\ (p\ n))\ \mathsf{else}\ ()$$

and the if statement evaluates to the left or right branch depending on whether $P\ n \rightsquigarrow^* \mathsf{tt}$ or $P\ n \rightsquigarrow^* \mathsf{ff}$. Now, this is all true of the reduction relation $\rightsquigarrow$, but if we bring in the full power of definitional equivalence we have the ability to work up from a single proof $a : \mathsf{acc}_> 0$:

$$\begin{aligned}
f\ 0\ a &\equiv f\ 0\ (\mathsf{intro}_{\mathsf{acc}}\ 0\ (\mathsf{inv}_0\ a)) \\
&\equiv f\ 1\ (\mathsf{inv}_0\ a\ 1\ (p\ 0)) \\
&\equiv f\ 1\ (\mathsf{intro}_{\mathsf{acc}}\ 1\ (\mathsf{inv}_1\ (\mathsf{inv}_0\ a\ 1\ (p\ 0)))) \\
&\equiv f\ 2\ (\mathsf{inv}_1\ (\mathsf{inv}_0\ a\ 1\ (p\ 0))\ 2\ (p\ 1)) \\
&\equiv \ldots
\end{aligned}$$

where we have shown the case where $P\ 0$ and $P\ 1$ both evaluate to true. If any $P\ n$ evaluates to false, then we will eventually get an equivalence to $()$, but if $P\ n$ is always true, then f will never reduce to $()$ – every term definitionally equal to $f\ 0\ a$ will contain a subterm def.eq. to f. So $a : \mathsf{acc}_> 0 \vdash f\ 0\ a \equiv ()$ holds if and only if $\forall n.\ P\ n$, and hence $\equiv$ is undecidable.

3.1.1 Algorithmic equality is not transitive

From the results of the previous section, given that algorithmic equality is implemented by Lean, and hence is obviously decidable, they cannot be equal as relations, so there is some rule of definitional equality that is not respected by algorithmic equality. In the above example, we can typecheck the various parts of the equality chain to see that $\Leftrightarrow$ is not transitive:

$$\begin{aligned} f\ 0\ a &\Leftrightarrow f\ 0\ (\mathsf{intro}_{\mathsf{acc}}\ 0\ (\mathsf{inv}_0\ a)) \\ &\Leftrightarrow f\ 1\ (\mathsf{inv}_0\ a\ 1\ (p\ 0)) \\ &\text{but} \\ f\ 0\ a &\not\Leftrightarrow f\ 1\ (\mathsf{inv}_0\ a\ 1\ (p\ 0)). \end{aligned}$$

We can think of the middle step $f\ 0\ (\mathsf{intro}_{\mathsf{acc}}\ 0\ (\mathsf{inv}_0\ a))$ as a "creative" step, where we pick one of the many possible terms of type $\mathsf{acc}_>\ 0$ which happens to reduce in the right way. But since the expression $f\ 0\ a$ is a normal form, we don't attempt to reduce it, and indeed if we did we would have nontermination problems (since reduction here only makes the term larger).

Note that the fact that we are in an inconsistent context doesn't matter for this: we could have used $a : \mathsf{acc}_<\ 1$ with the same result.

This instance of non-transitivity can be traced back to the usage of a subsingleton eliminator via acc. There is another, less known source of non-transitivity: quotients of propositions. While this is not a particularly useful operation, since any proposition is already a subsingleton, so a quotient will not do anything, they can technically be formed, and lift acts like a subsingleton eliminator in this case. So for example, if $p : \mathbb{P}$, $R : p \to p \to \mathbb{P}$, $\alpha : \mathsf{U}_1$, $f : p \to \alpha$, $H : \forall x\ y.\ r\ x\ y \to f\ x = f\ y$, $q : p/R$ and $h : p$, then:

$$\begin{aligned} &\mathsf{lift}_R\ \alpha\ f\ H\ q \Leftrightarrow \mathsf{lift}_R\ \alpha\ f\ H\ (\mathsf{mk}_R\ h) \Leftrightarrow f\ h \\ &\qquad\qquad \text{but} \\ &\mathsf{lift}_R\ \alpha\ f\ H\ q \not\Leftrightarrow f\ h. \end{aligned}$$

3.1.2 Failure of subject reduction

While the type system given here actually satisfies subject reduction (which is to say, if $\Gamma \vdash e : \alpha$ and $e \rightsquigarrow e'$ (or $\Gamma \vdash e \Leftrightarrow e'$, or $\Gamma \vdash e \equiv e'$), then $\Gamma \vdash e' : \alpha$), this is because we use the $\equiv$ relation in the conversion rule $\Gamma \vdash e : \alpha$, $\Gamma \vdash \alpha \equiv \beta$ implies $\Gamma \vdash e : \beta$. If we used algorithmic equality instead, to get a variant typing judgment $\Gamma \Vdash e : \alpha$ closer to what one would expect of the Lean typechecker, we find failure of subject reduction, directly from failure of transitivity. If $\Gamma \vdash \alpha \Leftrightarrow \beta$, $\Gamma \vdash \beta \Leftrightarrow \gamma$, $\Gamma \vdash \alpha \not\Leftrightarrow \gamma$, and $\Gamma \Vdash e : \gamma$, then:

- $\Gamma \Vdash \mathrm{id}_\beta\ e : \beta$ because the application forces checking $\Gamma \vdash \beta \Leftrightarrow \gamma$.
- $\Gamma \Vdash \mathrm{id}_\alpha\ (\mathrm{id}_\beta\ e) : \alpha$ since the application forces checking $\Gamma \vdash \alpha \Leftrightarrow \beta$.
- But $\Gamma \nVdash \mathrm{id}_\alpha\ e : \alpha$ because this requires $\Gamma \vdash \alpha \Leftrightarrow \gamma$ which is false.

Since we obviously have $\mathrm{id}_\beta\ e \rightsquigarrow e$ by the β and δ rules, this is a counterexample to subject reduction.

3.2 Regularity

These lemmas are essentially trivial inductions and are true by virtue of the way we set up the type system, so they are recorded here simply to keep track of the invariants.

Lemma 3.1 (Regularity)**.**

(1) If $\Gamma \vdash e : \alpha$*, then* $\vdash \Gamma$ ok*.*

(2) If $\Gamma \vdash e : \alpha$*, then* $FV(e) \cup FV(\alpha) \subseteq \Gamma$*.*

(3) If $\Gamma \vdash \alpha$ type*, then* $\Gamma \vdash \alpha : \mathsf{U}_\ell$ *for some* ℓ*.*

(4) If $\Gamma \vdash e : \alpha$*, then* $\Gamma \vdash \alpha$ type*.*

(5) If $\Gamma \vdash e \equiv e'$*, then there exists* α, α' *such that* $\Gamma \vdash e : \alpha$ *and* $\Gamma \vdash e' : \alpha'$*.*

(6) If $\Gamma; t : F \vdash K$ spec*, then* $\Gamma \vdash F$ type *(and more precisely,* $F = \forall x :: \alpha.\ \mathsf{U}_\ell$ *for some* α, ℓ*).*

(7) If $\Gamma; t : F \vdash K$ spec *and* $(c : \alpha) \in K$*, then* $\Gamma; t : F \vdash \alpha$ ctor*.*

(8) If $\Gamma; t : F \vdash \alpha$ ctor*, then* $\Gamma, t : F \vdash \alpha$ type*.*

Proof. By induction on the respective judgments (all of the parts may be proven separately). □

Lemma 3.2 (Weakening)**.**

(1) If $\Gamma \vdash e : \alpha$ *and* $\vdash \Gamma, \Delta$ ok*, then* $\Gamma, \Delta \vdash e : \alpha$*.*

(2) If $\Gamma \vdash e \equiv e'$ *and* $\vdash \Gamma, \Delta$ ok*, then* $\Gamma, \Delta \vdash e \equiv e'$*.*

(3) If $\Gamma, \Delta \vdash e : \alpha$ *and* $FV(e) \subseteq \Gamma$*, then* $\Gamma \vdash e : \alpha$*.*

(4) If $\Gamma, \Delta \vdash e \equiv e'$ *and* $FV(e) \cup FV(e') \subseteq \Gamma$*, then* $\Gamma \vdash e \equiv e'$*.*

(5) $\Gamma \vdash e : \alpha$ *implies* $\Gamma \vdash' e : \alpha$*, and* $\Gamma \vdash e \equiv e'$ *implies* $\Gamma \vdash' e \equiv e'$*, where the modified judgment* $\vdash'$ *eliminates the weakening rules and replaces the variable and universe rules with*

$$\frac{(x : \alpha) \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{}{\Gamma \vdash \mathsf{U}_\ell : \mathsf{U}_{S\ell}} \qquad \frac{\ell \equiv \ell'}{\Gamma \vdash \mathsf{U}_\ell \equiv \mathsf{U}_{\ell'}}$$

Proof. (1,2) and (3,4) are each proven by mutual induction on the first hypothesis. For (5), since weakening is provable for the judgment $\vdash'$ it follows that all rules of $\vdash$ are provable in $\vdash'$. □

Lemma 3.3 (Properties of substitution)**.**

(1) If $\Gamma, x : \alpha, \Delta \vdash e_1 \equiv e_1'$ *and* $\Gamma \vdash e_2 : \alpha$*, then* $\Gamma, \Delta[e_2/x] \vdash e_1[e_2/x] \equiv e_1'[e_2/x]$*.*

(2) If $\Gamma, x : \alpha, \Delta \vdash e_1 : \beta$ *and* $\Gamma \vdash e_2 : \alpha$*, then* $\Gamma, \Delta[e_2/x] \vdash e_1[e_2/x] : \beta[e_2/x]$*.*

(3) If $\Gamma, x : \alpha \vdash e_1 : \beta$ *and* $\Gamma \vdash e_2 \equiv e_2' : \alpha$*, then* $\Gamma \vdash e_1[e_2/x] \equiv e_1[e_2'/x]$*.*

Proof. (1) and (2) must be proven simultaneously by induction on the first hypotheses. All cases are straightforward. In the proof irrelevance case, we know $\Gamma, x : \alpha \vdash e_1 : p$ and $\Gamma, x : \alpha \vdash e_1' : p$ for some p with $\Gamma, x : \alpha \vdash p : \mathbb{P}$. By the induction hypothesis, $\Gamma \vdash e_1[e_2/x] : p[e_2/x]$ and $\Gamma \vdash e_1'[e_2/x] : p[e_2/x]$ and $\Gamma \vdash p[e_2/x] : \mathbb{P}[e_2/x]$; but $\mathbb{P}[e_2/x] = \mathbb{P}$ so proof irrelevance applies to show $\Gamma \vdash e_1[e_2/x] = e_1'[e_2/x]$.

(3) is proven by induction on the structure of e_1 and applying compatibility lemmas in each case. □

With this theorem we can upgrade lemma 3.1.(5) to:

Lemma 3.4 (Regularity continued)**.**

(1) If $\Gamma \vdash e \equiv e'$*, then there exists* α *such that* $\Gamma \vdash e \equiv e' : \alpha$*.*

(2) If $\Gamma \vdash e : \alpha$ *and* $e \rightsquigarrow e'$*, then* $\Gamma \vdash e \equiv e' : \alpha$*.*

(3) If $\Gamma \vdash e : \alpha$ *and* $\Gamma \vdash e' : \alpha$*, and* $\Gamma \vdash e \Leftrightarrow e'$*, then* $\Gamma \vdash e \equiv e'$*.*

Proof. Straightforward induction on the derivation of $\Gamma \vdash e \equiv e'$. We need lemma 3.3.(2) to typecheck both sides of the β rule. Note that the induction hypothesis is not strong enough for the application rule, except that we explicitly require that both sides have agreeing types in this case. □

Lemma 3.4.(2) implies subject reduction for $\rightsquigarrow$, and lemma 3.4.(3) is the main reason we are interested in algorithmic equality, since it is a thing we can check which implies "true" well-typedness. It is this that will allow us to conclude that Lean is consistent given that the ideal typing judgment we are developing here is consistent.

4 Unique typing

There are a large number of "natural" properties about the typing and definitional equality judgments we will want to be true in order to reason that certain judgments are not derivable for "obvious" reasons, for example that it is not possible to prove $\vdash \mathbb{P} : \mathbb{P}$ (which is a necessary condition for soundness).

Theorem 4.1 (Unique typing)**.** *If* $\Gamma \vdash e : \alpha$ *and* $\Gamma \vdash e : \beta$*, then* $\Gamma \vdash \alpha \equiv \beta$*.*

Unfortunately, we cannot yet prove this theorem. The critical step is the Church-Rosser theorem, which we will develop in the next section. However, we can set up the induction, which is necessary now since the Church-Rosser theorem will require that this theorem is true, and we will be caught in a circularity unless we are careful about the claims.

We will prove this theorem by induction on the number of alternations between the judgments $\Gamma \vdash e : \alpha$ and $\Gamma \vdash \alpha \equiv \beta$ (which are mutually recursive). Define $\Gamma \vdash_n e : \alpha$ and $\Gamma \vdash_n \alpha \equiv \beta$ by induction on $n \in \mathbb{N}$ as follows:

- $\Gamma \vdash_0 \alpha \equiv \beta$ iff $\alpha = \beta$.
- $\Gamma \vdash_{n+1} \alpha \equiv \beta$ iff there is a proof of $\Gamma \vdash \alpha \equiv \beta$ using only $\Gamma \vdash_n e : \alpha$ typing judgments.
- Assuming $\Gamma \vdash_m \alpha \equiv \beta$ is defined for $m \le n$, $\Gamma \vdash_n e : \alpha$ means that there is a proof of $\Gamma \vdash e : \alpha$ in which all appeals to the conversion rule use $\Gamma \vdash_m \alpha \equiv \beta$ for $m \le n$.

So if $\Gamma \vdash_0 e : \alpha$, then there is a proof that does not use the conversion rule at all; if $\Gamma \vdash_1 \alpha \equiv \beta$ then there is a proof whose typing judgments do not use the conversion rule; if $\Gamma \vdash_1 e : \alpha$ then there is a proof using only the 1-provable conversion rule; and so on. We will prove theorem 4.1 by induction on this n.

Lemma 4.2 (n-provability basics)**.**

(1) If $m \le n$ *then* $\Gamma \vdash_m e : \alpha$ *implies* $\Gamma \vdash_n e : \alpha$*.*

(2) If $m \le n$ *then* $\Gamma \vdash_m \alpha \equiv \beta$ *implies* $\Gamma \vdash_n \alpha \equiv \beta$*.*

(3) If $\Gamma \vdash e : \alpha$ *then* $\Gamma \vdash_n e : \alpha$ *for some* $n \in \mathbb{N}$*.*

(4) If $\Gamma \vdash \alpha \equiv \beta$ *then* $\Gamma \vdash_n \alpha \equiv \beta$ *for some* $n \in \mathbb{N}$*.*

Proof. (1) is immediate from the definition, (2) follows from (1). (3,4) are proven by a mutual induction on the typing judgment. □

Definition 1. *Say that $\vdash_n$ has* definitional inversion *if the following properties hold:*

1. *If $\Gamma \vdash_n \mathsf{U}_\ell \equiv \mathsf{U}_{\ell'}$, then $\ell \equiv \ell'$.*
2. *If $\Gamma \vdash_n \forall x : \alpha.\ \beta \equiv \forall x : \alpha'.\ \beta'$, then $\Gamma \vdash_n \alpha \equiv \alpha'$ and $\Gamma, x : \alpha \vdash_n \beta \equiv \beta'$.*
3. *$\Gamma \vdash_n \mathsf{U}_\ell \not\equiv \forall x : \alpha.\ \beta$.*

(We will also use the term unique typing *for this property given theorem 4.3.)*

There are other inversions along these lines, but distinguishing universes and foralls is the most important part and it is what we need for the induction.

Theorem 4.3 (Unique typing)**.** *If $\vdash_n$ has definitional inversion, and $\Gamma \vdash_n e : \alpha$ and $\Gamma \vdash_n e : \beta$, then $\Gamma \vdash_n \alpha \equiv \beta$.*

Proof. By the weakening lemma, we can use instead the judgment $\vdash'_n$ which has no weakening rule.

By induction on the proof of $\Gamma \vdash'_n e : \alpha$ with a secondary induction on $\Gamma \vdash'_n e : \beta$.

1. If $\Gamma \vdash'_n e : \alpha$ from the conversion rule on $\Gamma \vdash_n \alpha' \equiv \alpha$, $\Gamma \vdash_n e : \alpha'$, then $\Gamma \vdash_n \alpha' \equiv \beta$ by the IH, so $\Gamma \vdash_n \alpha \equiv \beta$ by transitivity. (Similarly if the conversion rule applies on $\Gamma \vdash'_n e : \beta$.)
2. Otherwise, the same typing rule applies in both derivations. The variable, universe, lambda, let, and constant cases are trivial.
3. In the forall case, we have $\Gamma \vdash_n \forall x : \alpha.\ \beta : \mathsf{U}_{\mathrm{imax}(\ell_1,\ell_2)}, \mathsf{U}_{\mathrm{imax}(\ell'_1,\ell'_2)}$ from $\Gamma \vdash \alpha : \mathsf{U}_{\ell_1}, \mathsf{U}_{\ell'_1}$ and $\Gamma \vdash \beta : \mathsf{U}_{\ell_2}, \mathsf{U}_{\ell'_2}$, and from the inductive hypothesis $\Gamma \vdash_n \mathsf{U}_{\ell_1} \equiv \mathsf{U}_{\ell'_1}$. From definitional inversion, $\ell_1 \equiv \ell'_1$ and $\ell_2 \equiv \ell'_2$, so $\Gamma \vdash_n \mathsf{U}_{\mathrm{imax}(\ell_1,\ell_2)} \equiv \mathsf{U}_{\mathrm{imax}(\ell'_1,\ell'_2)}$.
4. In the application case, we have $\Gamma \vdash_n e_1\ e_2 : \beta[e_2/x], \beta'[e_2/x]$ from $\Gamma \vdash_n e_1 : \forall x : \alpha.\ \beta, \forall x : \alpha'.\ \beta'$ and $\Gamma \vdash_n e_2 : \alpha, \alpha'$, and from the inductive hypothesis $\Gamma \vdash_n \forall x : \alpha.\ \beta \equiv \forall x : \alpha'.\ \beta'$. From definitional inversion, $\Gamma \vdash_n \alpha \equiv \alpha'$ and $\Gamma, x : \alpha \vdash_n \beta \equiv \beta'$, so $\Gamma \vdash_n \beta[e_2/x] \equiv \beta'[e_2/x]$.

□

Thus, it suffices to prove that $\vdash_n$ has definitional inversion for every n to establish theorem 4.1. We can show the base case:

Lemma 4.4. *$\vdash_0$ has definitional inversion.*

Proof. Since $\Gamma \vdash_0 e \equiv e'$ means $e = e'$, all cases are trivial by inversion on the construction of the term. □

4.1 The κ reduction

Note: In this section, we will omit the indices from the provability relation, but we will focus on characterizing the $\equiv$ relation at a particular level. So read $\Gamma \vdash \alpha \equiv \beta$ as $\Gamma \vdash_{n+1} \alpha \equiv \beta$, and $\Gamma \vdash e : \alpha$ as $\Gamma \vdash_n e : \alpha$. Also (and importantly) we will assume that $\vdash_n$ has unique typing, which will prevent the appearance of certain pathologies.

The standard formulation of the Church-Rosser theorem, when applied to the $\rightsquigarrow$ reduction relation, is not true; under reasonable definitions of reduction, Lean will not have unique normal

forms, because of proof irrelevance. (We already saw how this plays out in section 3.1). All other substantive reduction rules act on terms the same way regardless of their types. To analyze this, we will split the definitional equality judgment into two parts: A $\beta\delta\zeta\iota$-reduction relation (henceforth abbreviated κ reduction), and a relation that does proof irrelevance. The idea is that κ reduction satisfies a modified version of the Church-Rosser theorem, while proof irrelevance picks up the pieces, quantifying exactly how non-unique the normal form is.

The η rule can sometimes fight against the ι reduction in the sense that it is possible for a subsingleton eliminator to reduce in two ways, where the η reduced form cannot reduce, for example with the following reductions, using $\mathsf{rec}_{a=} : \forall C.\ C\ a \to \forall b.\ a = b \to C\ b$:

$$\lambda h : a = a.\ \mathsf{rec}_{a=}\ C\ e\ a\ h \rightsquigarrow_\eta \mathsf{rec}_{a=}\ C\ e\ a$$
$$\lambda h : a = a.\ \mathsf{rec}_{a=}\ C\ e\ a\ h \rightsquigarrow_\iota \lambda h : a = a.\ e$$

To resolve this, we will require that rec and lift always have their required number of parameters. To accomplish this, we define an η-expansion map as a preprocessing stage on terms before reduction. The transformation is as follows:

- If e is a list of terms of length n and rec_P has $m \geq n$ arguments, then $\overline{\mathsf{rec}_P\ e} = \lambda x :: \alpha.\ \mathsf{rec}_P\ e\ x$ where x is the remaining $n - m$ arguments, with type α according to the specification of P.
- If e is a list of terms of length $n \leq 6$ (note that lift has 6 arguments), then $\overline{\mathsf{lift}\ e} = \lambda x :: \alpha.\ \mathsf{lift}\ e\ x$ where x is the remaining $6 - n$ arguments.
- Otherwise, the transformation is recursive in subterms: $\overline{x} = x$, $\overline{\lambda x : \alpha.\ e} = \lambda x : \overline{\alpha}.\ \overline{e}$, etc.

A term is said to be in rec-normal form if every rec_P and lift subterm is followed by a sequence of applications of the appropriate length.

Lemma 4.5 (Properties of the rec-normal form)**.**

- *A term e is in rec-normal form iff $\bar{e} = e$.*
- *$\bar{e}$ is always in rec-normal form.*
- *If $\Gamma \vdash e : \alpha$, then $\Gamma \vdash \bar{e} \equiv e$. (In fact, the proof of equivalence uses only η.)*
- *If e_1, e_2 are in rec-normal form, then so is $e_1[e_2/x]$.*

The κ reduction relation is defined on terms in rec-normal form, with compatibility rules such as these for every syntax operator (including $\mathsf{rec}_P\ e$ and $\mathsf{lift}\ e$):

$$\boxed{\Gamma \vdash e \rightsquigarrow_\kappa e'} \qquad \frac{\Gamma \vdash e_1 \rightsquigarrow_\kappa e_1'}{\Gamma \vdash e_1\ e_2 \rightsquigarrow_\kappa e_1'\ e_2} \qquad \frac{\Gamma \vdash e_2 \rightsquigarrow_\kappa e_2'}{\Gamma \vdash e_1\ e_2 \rightsquigarrow_\kappa e_1\ e_2'}$$

$$\frac{\Gamma \vdash \alpha \rightsquigarrow_\kappa \alpha'}{\Gamma \vdash \lambda x : \alpha.\ e \rightsquigarrow_\kappa \lambda x : \alpha'.\ e} \qquad \frac{\Gamma, x : \alpha \vdash e \rightsquigarrow_\kappa e'}{\Gamma \vdash \lambda x : \alpha.\ e \rightsquigarrow_\kappa \lambda x : \alpha.\ e'} \qquad \cdots$$

The substantive rules are:

$$(\beta)\ \frac{}{\Gamma \vdash (\lambda x : \alpha.\ e)\ e' \rightsquigarrow_\kappa e[e'/x]} \qquad (\delta)\ \frac{\mathsf{def}\ c : \alpha := e}{\Gamma \vdash c \rightsquigarrow_\kappa e} \qquad (\zeta)\ \frac{}{\Gamma \vdash \mathsf{let}\ x : \alpha := e'\ \mathsf{in}\ e \rightsquigarrow_\kappa e[e'/x]}$$

$$(\iota)\ \frac{P \text{ is non-SS inductive with ctor } c}{\Gamma \vdash \mathsf{rec}_P\ C\ e\ p\ (c\ b) \rightsquigarrow_\kappa e_c\ b\ v} \qquad (\iota_q)\ \frac{}{\Gamma \vdash \mathsf{lift}\ R\ f\ h\ (\mathsf{mk}_R\ a) \rightsquigarrow_\kappa f\ a}$$

$$(K^+)\ \frac{P \text{ is SS inductive} \quad \Gamma \vdash \mathsf{intro}\ \mathsf{inv}[p, h] : \alpha}{\Gamma \vdash \mathsf{rec}_P\ C\ e\ p\ h \rightsquigarrow_\kappa e\ \mathsf{inv}[p, h]\ v}$$

See section 2.6 for the variable names and types used in the ι rules; recall in particular that v in the RHS of the rule is a sequence of lambdas $v_i = \lambda x :: \xi_i.\ \mathsf{rec}_P\ C\ e\ \pi_i[b, x]\ (u_i\ x)$ dictated by the definition of the inductive type.

We have an alternate ι rule for SS inductives, where $\mathsf{inv}[p, h]$ is a sequence of terms such that $\mathsf{intro}\ \mathsf{inv}[p, h] \equiv h$ (by proof irrelevance) and $\mathsf{inv}_i[p, \mathsf{intro}\ b] \equiv b_i$, which we call K^+ because it is a souped-up version of the K-like reduction rule in section 2.6.4. It applies only when $\mathsf{intro}\ \mathsf{inv}[p, h]$ is well-typed (and is the reason why $\rightsquigarrow_\kappa$ needs a context), which can also be written as a collection of $\equiv$ judgments at $\vdash_n$.

By the definition of a subsingleton inductive, every argument to the intro constructor is either propositional, or appears as one of the parameters p_i to the inductive family. We define $\mathsf{inv}_i[p, h] := p_j$ when the ith constructor argument is non-propositional and appears at position j in the output type, and $\mathsf{inv}_i[p, h] = \mathsf{inv}_i\ h$ for the propositions, where inv_i is an atomic projection function. These inv_i projection operators can be defined using the recursor, like we demonstrated for acc in section 3.1. It doesn't really matter if these terms reduce or not (i.e. they could be constants or defined via the recursor), since they are proofs and are thus going to be pushed into the proof irrelevance relation.

The proof irrelevance relation deals with all the ways that normal forms can fail to be unique. Specifically, this relation is responsible for changing universe levels and changing proofs, as well as the η rule.

$$\boxed{\Gamma \vdash e \equiv_p e'}$$

$$\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash e \equiv_p e} \qquad \frac{\Gamma \vdash \alpha \equiv_p \alpha' \quad \Gamma, x : \alpha \vdash e \equiv_p e'}{\Gamma \vdash \lambda x : \alpha.\ e \equiv_p \lambda x : \alpha'.\ e'} \qquad \frac{\Gamma \vdash e_1 \equiv_p e_1' \quad \Gamma \vdash e_2 \equiv_p e_2'}{\Gamma \vdash e_1\ e_2 \equiv_p e_1'\ e_2'} \qquad \dots$$

$$\frac{\Gamma, x : \alpha \vdash e \equiv_p e'\ x}{\Gamma \vdash \lambda x : \alpha.\ e \equiv_p e'} \qquad \frac{\Gamma, x : \alpha \vdash e\ x \equiv_p e'}{\Gamma \vdash e \equiv_p \lambda x : \alpha.\ e'} \qquad \frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p}{\Gamma \vdash h \equiv_p h'}$$

The ellipsis abbreviates all the compatibility rules. The η rule here is split in two because $\equiv_p$ lacks a symmetry rule, but it is also tightly syntax-constrained – it is essentially only useful for proving $\lambda x :: \alpha.\ e\ x \equiv_p e$. In particular it does not apply at all on proving that two variables of function type are equivalent, or proving that two universes or non-function things are equivalent.

Lemma 4.6 (Regularity of reductions)**.**

(1) If $\Gamma \vdash e : \alpha$ *and* $\Gamma \vdash e \rightsquigarrow_\kappa e'$*, then* $\Gamma \vdash e \equiv e' : \alpha$*.*

(2) $\equiv_p$ *is an equivalence relation.*

(3) If $\Gamma \vdash e \equiv_p e'$*, then* $\Gamma \vdash e \equiv e'$*.*

(4) If $\Gamma, x : \alpha \vdash e_1 \equiv_p e_1'$ *and* $\Gamma \vdash e_2 \equiv_p e_2'$ *then* $\Gamma \vdash e_1[e_2/x] \equiv_p e_1'[e_2'/x]$*.*

Proof. All parts are easy inductions. □

Note that the first part implies subject reduction for $\rightsquigarrow_\kappa$.

4.2 The Church-Rosser theorem

Theorem 4.7 (Church-Rosser property)**.** *If* $\Gamma \vdash e : \alpha$*, and* $e \rightsquigarrow_\kappa^* e_1$ *and* $e \rightsquigarrow_\kappa^* e_2$*, then there exists* e_1' *and* e_2' *such that* $\Gamma \vdash e_1' \equiv_p e_2'$*, and* $e_1 \rightsquigarrow_\kappa^* e_1'$ *and* $e_2 \rightsquigarrow_\kappa^* e_2'$*.*

The proof follows the Tait–Martin-Löf method, extended to all the κ rules. Define the parallel reduction $\gg_\kappa$ by the following rules:

$$\frac{}{\Gamma \vdash x \gg_\kappa x} \qquad \frac{\Gamma \vdash \alpha \gg_\kappa \alpha' \quad \Gamma, x : \alpha \vdash e \gg_\kappa e'}{\Gamma \vdash \lambda x : \alpha.\ e \gg_\kappa \lambda x : \alpha'.\ e'} \qquad \frac{\Gamma \vdash e_1 \gg_\kappa e_1' \quad \Gamma \vdash e_2 \gg_\kappa e_2'}{\Gamma \vdash e_1\ e_2 \gg_\kappa e_1'\ e_2'} \qquad \cdots$$

$$\frac{\Gamma, x : \alpha \vdash e_1 \gg_\kappa e_1' \quad \Gamma \vdash e_2 \gg_\kappa e_2'}{\Gamma \vdash (\lambda x : \alpha.\ e_1)\ e_2 \gg_\kappa e_1'[e_2'/x]}$$

$$\frac{\Gamma \vdash e_2[e_1/x] \gg_\kappa e'}{\Gamma \vdash \mathsf{let}\ x : \alpha := e_1\ \mathsf{in}\ e_2 \gg_\kappa e'} \qquad \frac{\mathsf{def}\ c : \alpha := e \quad \Gamma \vdash e \gg_\kappa e'}{\Gamma \vdash c \gg_\kappa e'}$$

$$\frac{\Gamma \vdash f \gg_\kappa f' \quad \Gamma \vdash a \gg_\kappa a'}{\Gamma \vdash \mathsf{lift}\ R\ f\ h\ (\mathsf{mk}_R\ a) \gg_\kappa f'\ a'} \qquad \frac{\begin{array}{c}P \text{ is non-SS inductive with ctor } c \\ \Gamma \vdash C, e, b, p \gg_\kappa C', e', b', p'\end{array}}{\Gamma \vdash \mathsf{rec}_P\ C\ e\ p\ (c\ b) \gg_\kappa e'\ b'\ v'}$$

$$\frac{P \text{ is SS inductive} \quad \Gamma \vdash \mathsf{intro}\ \mathsf{inv}[p, h] : \alpha \quad \Gamma \vdash C, e, p, h \gg_\kappa C', e', p', h'}{\Gamma \vdash \mathsf{rec}_P\ C\ e\ p\ h \gg_\kappa e_c'\ \mathsf{inv}[p', h']\ v'}$$

The ellipsis on the first line abbreviates compatibility rules for all the term constructors, recursing into all subterms like in the examples for lambda and application. All the substantive rules also follow a similar pattern: for each substantive rule in $\rightsquigarrow_\kappa$, there is a corresponding rule where after applying the $\rightsquigarrow_\kappa$ rule all variables on the RHS are $\gg_\kappa$ evaluated to the primed versions, and these are what end up in the target expression. (Note that in the ι rule, v is a term that mentions e and p; these are replaced by the primed versions in v'.)

In addition, we define the following "complete reduction" $\Gamma \vdash e \ggg_\kappa e'$ by exactly the same rules as $\gg_\kappa$, except that the compatibility rules only apply if none of the substantive rules are applicable. This makes $\ggg_\kappa$ almost deterministic (producing a unique e' given e), except that the $\equiv_p$ hypothesis in the ι rule allows some freedom of choice of the parameters b.

It is easy to prove the following properties by induction:

Lemma 4.8 (Properties of $\gg_\kappa$)**.**

(1) If $\Gamma \vdash e : \alpha$, then $\Gamma \vdash e \gg_\kappa e$.

(2) If $\Gamma \vdash e \rightsquigarrow_\kappa e'$ then $\Gamma \vdash e \gg_\kappa e'$.

(3) If $\Gamma \vdash e \gg_\kappa e'$ then $\Gamma \vdash e \rightsquigarrow_\kappa^* e'$.

(4) If $\Gamma, x : \alpha \vdash e_1 \gg_\kappa e_1'$ and $\Gamma \vdash e_2 \gg_\kappa e_2'$ (where $\Gamma \vdash e_2 : \alpha$) then $\Gamma \vdash e_1[e_2/x] \gg_\kappa e_1'[e_2'/x]$.

(5) If $\Gamma \vdash e \ggg_\kappa e'$, then $\Gamma \vdash e \gg_\kappa e'$.

(6) If $\Gamma \vdash e : \alpha$, then $\Gamma \vdash e \ggg_\kappa e'$ for some e'.

Lemma 4.9 (Compatibility of $\gg_\kappa$ with $\equiv_p$)**.** *If $\Gamma \vdash e_1 \equiv_p e_3 \gg_\kappa e_2$, then there exists e_4 such that $\Gamma \vdash e_1 \gg_\kappa e_4 \equiv_p e_2$.*

Proof. By induction on $e_1 \equiv_p e_3$ and inversion on $e_3 \gg_\kappa e_2$. (We will omit the contexts from the relations.)

- If $e_1 \equiv_p e_3 = e_1$ by the reflexivity rule, then $e_1 \gg_\kappa e_2 \equiv_p e_2$.
- If $e_1 \equiv_p e_3$ by the proof irrelevance rule, then $e_3 : p : \mathbb{P}$, so $e_2 : p : \mathbb{P}$ as well and hence $e_1 \gg_\kappa e_1 \equiv_p e_2$.

- If $e_1 \equiv_p e_3$ and $e_3 \gg_\kappa e_2$ both use the same compatibility rule, then it is immediate from the induction hypothesis.
- If $e_1 : p : \mathbb{P}$ is a proof, then $e_1 \gg_\kappa e_1 \equiv_p e_2$. (We will thus assume that e_1 is not a proof in later cases.)
- If $(\lambda x : \alpha_1.\ e_1)\ e_1' \equiv_p (\lambda x : \alpha_3.\ e_3)\ e_3' \gg_\kappa e_2[e_2'/x]$ where $e_1 \equiv_p e_3 \gg_\kappa e_2$, $e_1' \equiv_p e_3' \gg_\kappa e_2'$ and $\alpha_1 \equiv \alpha_3$, then $(\lambda x : \alpha_1.\ e_1)\ e_1' \gg_\kappa e_1[e_1'/x] \equiv_p e_2[e_2'/x]$. (Other cases are similar, when the $\equiv_p$ is proven by compatibility rules and the $\gg_\kappa$ is a substantive rule.)
- If $e_1\ e_1' \equiv_p (\lambda x : \alpha_3.\ e_3)\ e_3' \gg_\kappa e_2[e_2'/x]$ where $e_1\ x \equiv_p e_2 \gg_\kappa e_3$ and $e_1' \equiv_p e_2' \gg_\kappa e_3'$, then $e_1\ e_1' = (e_1\ x)[e_1'/x] \equiv_p e_2[e_2'/x]$.
- If $\mathsf{lift}\ R_1\ \beta_1\ f_1\ h_1\ q_1 \equiv_p \mathsf{lift}\ R_3\ \beta_3\ f_3\ h_3\ (\mathsf{mk}_R\ a_3)$ where $q_1 \equiv_p \mathsf{mk}_R\ a_3$ by proof irrelevance, then $\beta : \mathbb{P}$ so $e_1 : \beta$ is a proof. (Note: we are using that $\vdash_n$ has unique typing here.)
- If $\mathsf{rec}_P\ C_1\ e_1\ p_1\ h_1 \equiv_p \mathsf{rec}_P\ C_3\ e_3\ p_3\ (c\ b_3) \gg_\kappa (e_2)_c\ b_2\ v_2$ where P is non-SS inductive and $h_1 \equiv_p c\ b_3$ by proof irrelevance, it is a small eliminator, so $\mathsf{rec}_P\ C_1\ e_1\ p_1\ h_1$ is a proof.

□

Lemma 4.10 (Triangle lemma)**.** *If $\Gamma \vdash e : \alpha$, $e \gg_\kappa e'$, and $e \ggg_\kappa e^\bullet$, then there exists e° such that $\Gamma \vdash e' \gg_\kappa e^\circ \equiv_p e^\bullet$.*

Proof. By induction on $e \ggg_\kappa e^\bullet$ and inversion on $e \gg_\kappa e'$.

- If $x \lll_\kappa x \gg_\kappa x$, then $x \gg_\kappa x \equiv_p x$.
- If $e \ggg_\kappa e^\bullet$ by the beta rule:
  - If $e_1^\bullet[e_2^\bullet/x] \lll_\kappa (\lambda x : \alpha.\ e_1)\ e_2 \gg_\kappa e_1'[e_2'/x]$ by the beta rule, then $e_1' \gg_\kappa e_1^\circ$ and $e_2' \gg_\kappa e_2^\circ$ by the inductive hypothesis, so $e_1'[e_2'/x] \gg_\kappa e_1^\circ[e_2^\circ/x] \equiv_p e_1^\bullet[e_2^\bullet/x]$ by the substitution property.
  - If $e_1^\bullet[e_2^\bullet/x] \lll_\kappa (\lambda x : \alpha.\ e_1)\ e_2 \gg_\kappa (\lambda x : \alpha.\ e_1')\ e_2'$ by the application rule and lambda rule, then $(\lambda x : \alpha.\ e_1')\ e_2' \gg_\kappa e_1^\circ[e_2^\circ/x] \equiv_p e_1^\bullet[e_2^\bullet/x]$ by the beta rule for $\gg_\kappa$ and the IH.
- If $e^\bullet \lll_\kappa c \gg_\kappa e'$ by the delta rule, then $e' \gg_\kappa e^\circ \equiv_p e^\bullet$.
- If $e_1^\bullet[e_2^\bullet/x] \lll_\kappa \mathsf{let}\ x : \alpha := e_1\ \mathsf{in}\ e_2 \gg_\kappa e_2'[e_1'/x]$ by the zeta rule, then $e_1'[e_2'/x] \gg_\kappa e_1^\circ[e_2^\circ/x] \equiv_p e_1^\bullet[e_2^\bullet/x]$.
- If $e \ggg_\kappa e^\bullet$ by the non-SS inductive iota rule:
  - If $e_c^\bullet\ b^\bullet\ v^\bullet \lll_\kappa \mathsf{rec}_P\ C\ e\ p\ (c\ b) \gg_\kappa e_c'\ b'\ v'$ by the iota rule, then $e_c'\ b'\ v' \gg_\kappa e_c^\circ\ b^\circ\ v^\circ \equiv_p e_c^\bullet\ b^\bullet\ v^\bullet$.
  - If $e_c^\bullet\ b^\bullet\ v^\bullet \lll_\kappa \mathsf{rec}_P\ C\ e\ p\ (c\ b) \gg_\kappa \mathsf{rec}_P\ C'\ e'\ p'\ (c\ b')$ by the rec_P compatibility rule, then $\mathsf{rec}_P\ C'\ e'\ p'\ (c\ b') \gg_\kappa e_c^\circ\ b^\circ\ v^\circ \equiv_p e_c^\bullet\ b^\bullet\ v^\bullet$ by the iota rule.
- If $e \ggg_\kappa e^\bullet$ by the quotient iota rule:
  - If $f^\bullet\ a^\bullet \lll_\kappa \mathsf{lift}\ R\ f\ h\ (\mathsf{mk}_R\ a) \gg_\kappa f'\ a'$ by the iota rule, then $f'\ a' \gg_\kappa f^\circ\ a^\circ \equiv_p f^\bullet\ a^\bullet$.
  - If $f^\bullet\ a^\bullet \lll_\kappa \mathsf{lift}\ R\ f\ h\ (\mathsf{mk}_R\ a) \gg_\kappa \mathsf{lift}\ R'\ f'\ h'\ (\mathsf{mk}_{R'}\ a')$ by the lift compatibility rule, then we have $q' \gg_\kappa q^\circ \equiv_p q^\bullet$ for $q \in \{C, e, p, h\}$, and $\mathsf{lift}\ R'\ f'\ h'\ (\mathsf{mk}_{R'}\ a') \gg_\kappa f^\circ\ a^\circ \equiv_p f^\bullet\ a^\bullet$.
- If $e \ggg_\kappa e^\bullet$ by the K^+ rule:
  - If $e_c^\bullet\ \mathsf{inv}[p^\bullet, h^\bullet]\ v^\bullet \lll_\kappa \mathsf{rec}_P\ C\ e\ p\ h \gg_\kappa e_c'\ \mathsf{inv}[p', h']\ v'$ by the iota rule, then $e_c'\ \mathsf{inv}[p', h']\ v' \gg_\kappa e_c^\circ\ \mathsf{inv}[p^\circ, h^\circ]\ v^\circ \equiv_p e_c^\bullet\ \mathsf{inv}[p^\bullet, h^\bullet]\ v^\bullet$.

  - If $e_c^\bullet\ \mathsf{inv}[p^\bullet, h^\bullet]\ v^\bullet \lll_\kappa \mathsf{rec}_P\ C\ e\ p\ h \gg_\kappa \mathsf{rec}_P\ C'\ e'\ p'\ h'$ by the rec_P compatibility rule, then $\mathsf{rec}_P\ C'\ e'\ p'\ h' \gg_\kappa e_c^\circ\ \mathsf{inv}[p^\circ, h^\circ]\ v^\circ \equiv_p e_c^\bullet\ \mathsf{inv}[p^\bullet, h^\bullet]\ v^\bullet$ by the iota rule.
- If $e \ggg_\kappa e^\bullet$ by a compatibility rule:
  - If $e_1^\bullet\ e_2^\bullet \lll_\kappa e_1\ e_2 \gg_\kappa e_1'\ e_2'$ by the application rule, then $e_1'\ e_2' \gg_\kappa e_1^\circ\ e_2^\circ \equiv_p e_1^\bullet\ e_2^\bullet$.
  - If $\forall x : \alpha^\bullet.\ e^\bullet \lll_\kappa \forall x : \alpha.\ e \gg_\kappa \forall x : \alpha'.\ e'$ by the forall rule, then $\forall x : \alpha'.\ e' \gg_\kappa \forall x : \alpha^\circ.\ e^\circ \equiv_p \forall x : \alpha^\bullet.\ e^\bullet$.
  - Other compatibility rules follow the same pattern. □

The main proof of Church-Rosser is a corollary of lemma 4.10, and does not differ substantially from the usual proof putting diamonds together, because the additional complication of having $\equiv_p$ at the bottom of the diamond commutes with all the other reductions.

Proof of theorem 4.7. We prove in succession the following theorems:

1. If $\Gamma \vdash e : \alpha$, and $e_1 \ll_\kappa e \gg_\kappa e_2$, then $\exists e_1'\ e_2'.\ e_1 \gg_\kappa e_1' \equiv_p e_2' \ll_\kappa e_2$.
2. If $\Gamma \vdash e : \alpha$, and $e_1 \ll_\kappa^* e \gg_\kappa e_2$, then $\exists e_1'\ e_2'.\ e_1 \gg_\kappa e_1' \equiv_p e_2' \ll_\kappa^* e_2$.
3. If $\Gamma \vdash e : \alpha$, and $e_1 \ll_\kappa^* e \gg_\kappa^* e_2$, then $\exists e_1'\ e_2'.\ e_1 \gg_\kappa^* e_1' \equiv_p e_2' \ll_\kappa^* e_2$.
4. If $\Gamma \vdash e : \alpha$, and $e_1 \leftsquigarrow_\kappa^* e \rightsquigarrow_\kappa^* e_2$, then $\exists e_1'\ e_2'.\ e_1 \rightsquigarrow_\kappa^* e_1' \equiv_p e_2' \leftsquigarrow_\kappa^* e_2$.

(4) is the theorem we want.

1. If $e \gg_\kappa e_1, e_2$, then by lemma 4.10 there exists e_1', e_2' such that $e_i \gg_\kappa e_i'$ and $e_1' \equiv_p e^\bullet \equiv_p e_2'$. But then $e_1' \equiv_p e_2'$ are as desired.
2. By induction on $e \gg_\kappa^* e_1$. If $e \gg_\kappa^* e_1 \gg_\kappa e_3$ and we have inductively that $e_1 \gg_\kappa e_1' \equiv_p e_2' \ll_\kappa^* e_2$, then by applying (1) to $e_3 \ll_\kappa e_1 \gg_\kappa e_1'$ we obtain $e_3 \gg_\kappa e_3' \equiv_p e_1'' \ll_\kappa e_1'$, and by lemma 4.9 applied to $e_1'' \ll_\kappa e_1' \equiv_p e_2'$ we obtain $e_1'' \equiv_p e_2'' \ll_\kappa e_2'$ so that $e_3 \gg_\kappa e_3' \equiv_p e_1'' \equiv_p e_2'' \ll_\kappa e_2' \ll_\kappa^* e_2$.
3. By induction on $e \gg_\kappa^* e_2$. The proof is the same as (2), replacing lemma 4.9 for the analogous statement for $\gg_\kappa^*$, i.e. if $e_1 \gg_\kappa^* e_2$ and $e_1 \equiv_p e_1'$ then there exists e_2' such that $e_1' \gg_\kappa^* e_2' \equiv_p e_1'$. This follows by induction on lemma 4.9.
4. The equivalence of (3) and (4) comes from properties 4.8.(2) and 4.8.(3).

□

Now say that $\Gamma \vdash e_1 \equiv_\kappa e_2$ if $\Gamma \vdash e_1, e_2 : \alpha$ for some α, and there exists e_1', e_2' such that $\Gamma \vdash e_1 \rightsquigarrow_\kappa^* e_1' \equiv_p e_2' \leftsquigarrow_\kappa^* e_2$. This relation is obviously reflexive and symmetric and implies $\Gamma \vdash e_1 \equiv e_2$, and the Church-Rosser property implies it is also transitive.

Theorem 4.11 (Completeness of the κ reduction)**.** $\Gamma \vdash e \equiv e'$ *if and only if* $\Gamma \vdash e \equiv_\kappa e'$.

Proof. The reverse direction follows from regularity lemmas observed above. The forward direction is by induction on $\equiv$.

- The equivalence relation rules are immediate since $\equiv_\kappa$ is an equivalence relation (by the Church-Rosser property).
- For the compatibility rules, since both $\equiv_p$ and $\rightsquigarrow_\kappa$ have compatibility rules, this property passes to $\equiv_\kappa$. Thus, for example in the lambda case, we have $\Gamma \vdash \lambda x : \alpha.\ e \equiv_\kappa \lambda x : \alpha.\ e'$ since $\Gamma, x : \alpha \vdash e \equiv_\kappa e'$ from the IH, and similarly $\Gamma \vdash \lambda x : \alpha.\ e' \equiv_\kappa \lambda x : \alpha'.\ e'$, so by transitivity $\Gamma \vdash \lambda x : \alpha.\ e \equiv_\kappa \lambda x : \alpha'.\ e'$.

- The universe changing rules (for constants and U_ℓ) are in $\equiv_p$.
- The β and η rules are in $\rightsquigarrow_\kappa$, and the proof irrelevance rule is in $\equiv_p$. All the other equivalence rules are also introduced in $\rightsquigarrow_\kappa$.
- For subsingleton eliminators, we must show $\mathsf{rec}_P(C, e, p, \mathsf{intro}\ b) \equiv_\kappa e\ b\ v$. From the K^+ rule we have $\mathsf{rec}_P(C, e, p, \mathsf{intro}\ b) \equiv_\kappa e\ \mathsf{inv}[p, c\ b]\ v$ so it suffices to show $\mathsf{inv}_i[p, \mathsf{intro}\ b] \equiv_\kappa b_i$ for each i. If b_i is propositional then this is by proof irrelevance, otherwise $\mathsf{inv}_i[p, \mathsf{intro}\ b] = p_j$, and the well-typedness of $\mathsf{rec}_P(C, e, p, \mathsf{intro}\ b)$ implies that $\Gamma \vdash_n b_i \equiv p_j$. Thus by completeness of the κ reduction at $\vdash_n$, $\Gamma \vdash_n b_i \equiv_\kappa p_j$ and hence $\Gamma \vdash_{n+1} b_i \equiv_\kappa p_j$.

□

Now we can finally finish the inductive step of the proof of theorem 4.1:

Theorem 4.12 (Definitional inversion)**.** $\vdash_{n+1}$ *has definitional inversion.*

Proof. In each case we apply theorem 4.11 on the assumptions.

1. *If* $\Gamma \vdash_{n+1} \mathsf{U}_\ell \equiv \mathsf{U}_{\ell'}$*, then* $\ell \equiv \ell'$.
   Again, there are no $\rightsquigarrow_\kappa$ reductions from U_ℓ, so $\Gamma \vdash_{n+1} \mathsf{U}_\ell \equiv_p \mathsf{U}_{\ell'}$, and if the compatibility rule is used then $\ell \equiv \ell'$. If proof irrelevance is used, then $\Gamma \vdash_n \mathsf{U}_\ell, \mathsf{U}_{\ell'} : p$ for some $\Gamma \vdash_n p : \mathbb{P}$. Since $\Gamma \vdash_n \mathsf{U}_\ell : \mathsf{U}_{S\ell} : \mathsf{U}_{SS\ell}$ as well, by unique typing at n, $\Gamma \vdash_n \mathbb{P} \equiv \mathsf{U}_{SS\ell}$, so by definitional inversion $0 \equiv SS\ell$, a contradiction.
2. *If* $\Gamma \vdash_{n+1} \forall x : \alpha.\ \beta \equiv \forall x : \alpha'.\ \beta'$*, then* $\Gamma \vdash_{n+1} \alpha \equiv \alpha'$ *and* $\Gamma, x : \alpha \vdash_{n+1} \beta \equiv \beta'$.
   In this case, there are no $\rightsquigarrow_\kappa$ reductions except the compatibility rules, so $\forall x : \alpha.\ \beta \rightsquigarrow_\kappa^* \forall x : \alpha_1.\ \beta_1$ for some $\alpha \rightsquigarrow_\kappa^* \alpha_1$ and $\beta \rightsquigarrow_\kappa^* \beta_1$, and similarly $\alpha' \rightsquigarrow_\kappa^* \alpha_1'$ and $\beta' \rightsquigarrow_\kappa^* \beta_1'$, and if these are $\equiv_p$ equivalent using the compatibility rule then we are done.
   If $\Gamma \vdash_{n+1} \forall x : \alpha_1.\ \beta_1 \equiv_p \forall x : \alpha_1'.\ \beta_1'$ by proof irrelevance, then $\Gamma \vdash_n \forall x : \alpha_1.\ \beta_1, \forall x : \alpha_1'.\ \beta_1' : p : \mathbb{P}$. But $\Gamma \vdash_n \forall x : \alpha_1.\ \beta_1 : \mathsf{U}_{\mathrm{imax}(\ell_1,\ell_2)}$ for some ℓ_1, ℓ_2 since α_1 and β_1 are well-typed, so by unique typing at n, $p \equiv \mathsf{U}_{\mathrm{imax}(\ell_1,\ell_2)}$ and $0 \equiv S\,\mathrm{imax}(\ell_1, \ell_2)$, a contradiction.
3. $\Gamma \vdash_n \mathsf{U}_\ell \not\equiv \forall x : \alpha.\ \beta$.
   Suppose not. Similarly to previous parts, as there are no reductions from U_ℓ and no reductions except the compatibility rule for $\forall$, we obtain $\Gamma \vdash_n \mathsf{U}_\ell \equiv_p \forall x : \alpha'.\ \beta'$, and now there is no applicable rule except proof irrelevance, but this implies $\mathsf{U}_\ell : p : \mathbb{P}$ and hence $0 \equiv SS\ell$, a contradiction.

□

We've already described the structure of this theorem in earlier parts, but now we are finally ready to put all the parts together:

Proof of theorem 4.1. We prove by induction on n that $\vdash_n$ has definitional inversion (and hence unique typing, by theorem 4.3), and also that it satisfies the conclusion of theorem 4.11.

- For $n = 0$, $\vdash_0$ has definitional inversion by lemma 4.4, and theorem 4.11 is trivial (where both $\Gamma \vdash e \equiv_\kappa e'$ and $\Gamma \vdash e \equiv e'$ mean $e = e'$).
- For $n + 1$, suppose $\vdash_n$ has definitional inversion and satisfies theorem 4.11. Then all the results of section 4.2 follow, including theorem 4.11. Then definitional inversion at $n + 1$ is theorem 4.12.

□

5 Reduction of inductive types to W-types

Given the complicated structure involved in simply stating the axioms of inductive types, one may wonder if there is an easier way. In fact there is; we can replace the whole structure of inductive types with a few simple inductive type constructors.

5.1 The menagerie

The most well known general form of our kind of inductive type is the W-type, defined when $\Gamma \vdash A : \mathsf{U}_\ell$ and $\Gamma, x : A \vdash B : \mathsf{U}_\ell$:

$$\mathsf{W}x : A.\ B := \mu w : \mathsf{U}_\ell.\ (\mathsf{sup} : \forall x : A.\ (B \to w) \to w)$$

This carries most of the "power" of inductive types, but we still need some glue to be able to reduce everything else to this. First, note that most of the telescopes $x :: \alpha$ in an inductive type can be replaced by $\Sigma(x :: \alpha)$, where $\Sigma() := \mathbf{1}$ and $\Sigma(x : \alpha, y :: \beta) := \Sigma x : \alpha, \Sigma(y :: \beta)$. This just packs up all the types in the telescope into one dependent tuple. Similarly, we want the types $\mathbf{0}$ and $\alpha + \beta$ to pack up all the constructors into one.

To localize the universe management we will have a "universe lift" function $\mathsf{ulift}_u^v : \mathsf{U}_u \to \mathsf{U}_v$, defined when $u \le v$, as well as the $\mathsf{nonempty}$ operation (also known as the propositional truncation $\|\alpha\|$) to construct small eliminators. All the other type operators above will have the smallest possible universe level.

Finally, to handle inductive families and subsingleton eliminators, we will need the equality and acc types discussed previously. Here are the rules for these types:

$$\begin{aligned} e ::= \ldots \mid{}& \bot \mid \Sigma x : e.\ e \mid e + e \mid \mathsf{ulift}_\ell^\ell\ e \mid \|e\| \mid \mathsf{W}x : e.\ e \mid e = e \mid \mathsf{acc}_e \\ \mid{}& \mathsf{rec}_\bot \mid (e, e) \mid \pi_1\ e \mid \pi_2\ e \mid \mathsf{inl}\ e \mid \mathsf{inr}\ e \mid \mathsf{rec}_+\ e\ e \mid \uparrow e \mid \downarrow e \\ \mid{}& |e| \mid \mathsf{rec}_{||}\ e \mid \mathsf{sup}\ e\ e \mid \mathsf{rec}_\mathsf{W}\ e \mid \mathsf{refl}\ e \mid \mathsf{rec}_=\ e\ e \mid \mathsf{intro}_\mathsf{acc}\ e\ e \mid \mathsf{rec}_\mathsf{acc}\ e \end{aligned}$$

$$\frac{}{\vdash \bot : \mathbb{P}} \qquad \frac{1 \le \ell \quad \Gamma \vdash C : \mathsf{U}_\ell}{\Gamma \vdash \mathsf{rec}_\bot : \bot \to C}$$

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_\ell \quad \Gamma, x : \alpha \vdash \beta : \mathsf{U}_{\ell'}}{\Gamma \vdash \Sigma x : \alpha.\ \beta : \mathsf{U}_{\max(\ell,\ell',1)}} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_\ell \quad \Gamma \vdash \beta : \mathsf{U}_{\ell'}}{\Gamma \vdash \alpha + \beta : \mathsf{U}_{\max(\ell,\ell',1)}}$$

$$\frac{\Gamma \vdash e_1 : \alpha \quad \Gamma \vdash e_2 : \beta\ e_1}{\Gamma \vdash (e_1, e_2) : \Sigma x : \alpha.\ \beta} \qquad \frac{\Gamma \vdash p : \Sigma x : \alpha.\ \beta}{\Gamma \vdash \pi_1\ p : \alpha} \qquad \frac{\Gamma \vdash p : \Sigma x : \alpha.\ \beta}{\Gamma \vdash \pi_2\ p : \beta[\pi_1\ p/x]}$$

$$\frac{\Gamma \vdash \beta\ \mathsf{type} \quad \Gamma \vdash e : \alpha}{\Gamma \vdash \mathsf{inl}\ e : \alpha + \beta} \qquad \frac{\Gamma \vdash \alpha\ \mathsf{type} \quad \Gamma \vdash e : \beta}{\Gamma \vdash \mathsf{inr}\ e : \alpha + \beta}$$

$$\frac{1 \le \ell \quad \Gamma \vdash C : \alpha + \beta \to \mathsf{U}_\ell \quad \Gamma \vdash a : \forall x : \alpha.\ C\ (\mathsf{inl}\ x) \quad \Gamma \vdash b : \forall x : \beta.\ C\ (\mathsf{inr}\ x)}{\Gamma \vdash \mathsf{rec}_+\ a\ b : \forall p : \alpha + \beta.\ C\ p}$$

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_\ell \quad \max(1, \ell) \le \ell'}{\Gamma \vdash \mathsf{ulift}_\ell^{\ell'}\ \alpha : \mathsf{U}_{\ell'}} \qquad \frac{\Gamma \vdash \mathsf{ulift}_\ell^{\ell'}\ \alpha : \mathsf{U}_{\ell'} \quad \Gamma \vdash e : \alpha}{\Gamma \vdash \uparrow e : \mathsf{ulift}_\ell^{\ell'}\ \alpha} \qquad \frac{\Gamma \vdash e : \mathsf{ulift}_\ell^{\ell'}\ \alpha}{\Gamma \vdash \downarrow e : \alpha}$$

$$\frac{\Gamma \vdash \alpha\ \mathsf{type}}{\Gamma \vdash \|\alpha\| : \mathbb{P}} \qquad \frac{\Gamma \vdash e : \alpha}{\Gamma \vdash |e| : \|\alpha\|} \qquad \frac{\Gamma \vdash C : \mathbb{P} \quad \Gamma \vdash f : \alpha \to C}{\Gamma \vdash \mathsf{rec}_{||}\ f : \|\alpha\| \to C}$$

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_\ell \quad \Gamma, x : \alpha \vdash \beta : \mathsf{U}_{\ell'}}{\Gamma \vdash \mathsf{W}x : \alpha.\ \beta : \mathsf{U}_{\max(\ell,\ell',1)}} \qquad \frac{\Gamma \vdash a : \alpha \quad \Gamma \vdash f : \beta[a/x] \to \mathsf{W}x : \alpha.\ \beta}{\Gamma \vdash \mathsf{sup}\ a\ f : \mathsf{W}x : \alpha.\ \beta}$$

$$\frac{1 \le \ell \quad \Gamma \vdash C : (\mathsf{W}x : \alpha.\ \beta) \to \mathsf{U}_\ell \quad \Gamma \vdash e : \forall (a : \alpha)\ (f : \beta[a/x] \to \mathsf{W}x : \alpha.\ \beta).\ (\forall b : \beta[a/x].\ C\ (f\ b)) \to C\ (\mathsf{sup}\ a\ f)}{\Gamma \vdash \mathsf{rec_W}\ e : \forall w : (\mathsf{W}x : \alpha.\ \beta).\ C\ w}$$

$$\frac{\Gamma \vdash a : \alpha \quad \Gamma \vdash b : \alpha}{\Gamma \vdash a = b : \mathbb{P}} \qquad \frac{\Gamma \vdash a : \alpha}{\Gamma \vdash \mathsf{refl}\ a : a = a}$$

$$\frac{\Gamma \vdash a : \alpha \quad 1 \le \ell \quad \Gamma \vdash C : \alpha \to \mathsf{U}_\ell \quad \Gamma \vdash e : C\ a}{\Gamma \vdash \mathsf{rec_=}\ e : \forall b : \alpha.\ a = b \to C\ b}$$

$$\frac{\Gamma \vdash r : \alpha \to \alpha \to \mathbb{P}}{\Gamma \vdash \mathsf{acc}_r : \alpha \to \mathbb{P}} \qquad \frac{\Gamma \vdash x : \alpha \quad \Gamma \vdash f : \forall y : \alpha.\ r\ y\ x \to \mathsf{acc}_r\ y}{\Gamma \vdash \mathsf{intro_{acc}}\ x\ f : \mathsf{acc}_r\ x}$$

$$\frac{1 \le \ell \quad \Gamma \vdash C : \alpha \to \mathsf{U}_\ell \quad \Gamma \vdash e : \forall x : \alpha.\ (\forall y : \alpha.\ r\ y\ x \to \mathsf{acc}_r\ y) \to (\forall y : \alpha.\ r\ y\ x \to C\ y) \to C\ x}{\Gamma \vdash \mathsf{rec_{acc}}\ e : \forall x : \alpha.\ \mathsf{acc}_r\ x \to C\ x}$$

All of these could have been defined as inductive types in the sense of section 2.6:

$$\begin{aligned}
\bot &:= \mu t : \mathbb{P}.\ 0 \\
\Sigma x : \alpha.\ \beta &:= \mu t : \mathsf{U}_{\max(\ell,\ell',1)}.\ (\mathsf{pair} : \forall x : \alpha.\ \beta \to t) \\
\alpha + \beta &:= \mu t : \mathsf{U}_{\max(\ell,\ell',1)}.\ (\mathsf{inl} : \alpha \to t) + (\mathsf{inr} : \beta \to t) \\
\mathsf{ulift}_\ell^{\ell'}\ \alpha &:= \mu t : \mathsf{U}_{\ell'}.\ (\mathsf{up} : \alpha \to t) \\
\|\alpha\| &:= \mu t : \mathbb{P}.\ (\mathsf{intro} : \alpha \to t) \\
\mathsf{W}x : \alpha.\ \beta &:= \mu t : \mathsf{U}_{\max(\ell,\ell',1)}.\ (\mathsf{sup} : \forall x : \alpha.\ (\beta \to t) \to t) \\
a = b &:= (\mu t : \alpha \to \mathbb{P}.\ (\mathsf{refl} : t\ a))\ b \\
\mathsf{acc}_r &:= \mu t : \alpha \to \mathbb{P}.\ (\mathsf{intro} : \forall x : \alpha.\ (\forall y : \alpha.\ r\ y\ x \to t\ y) \to t\ x)
\end{aligned}$$

However, we are interested in taking them as primitive in this section and deriving general inductive types. All of the new operators have compatibility rules for $\equiv$ and $\Leftrightarrow$; we will not belabor this as they all look roughly the same: when all the parts are equivalent, so is the whole. For example:

$$\frac{\Gamma \vdash \alpha \equiv \alpha' \quad \Gamma, x : \alpha \vdash \beta \equiv \beta'}{\Gamma \vdash \Sigma x : \alpha.\ \beta \equiv \Sigma x : \alpha'.\ \beta'}$$

Since we will need to handle $\mathbb{P}$ specially in the proof of soundness, we have simplified all the large eliminating recursors to require $1 \le \ell$. The general recursor can be constructed from this by using $C' := \lambda x : P.\ \mathsf{ulift}_\ell^{\max(1,\ell)}\ (C\ x)$ (for each such inductive type P).

In a few of the constructors, additional parameters are elided, such as C in $\mathsf{rec}_\bot$; one should imagine that each constructor is sufficiently annotated to ensure unique typing. Following their interpretation as inductive types, they also come with the following ι rules:

$$\begin{aligned}
\pi_1(a, b) &\equiv a \\
\pi_2(a, b) &\equiv b \\
\mathsf{rec}_+\ a\ b\ (\mathsf{inl}\ x) &\equiv a\ x \\
\mathsf{rec}_+\ a\ b\ (\mathsf{inr}\ x) &\equiv b\ x \\
\downarrow\uparrow x &\equiv x \\
\mathsf{rec_W}\ e\ (\mathsf{sup}\ a\ f) &\equiv e\ a\ f\ (\lambda b : \beta[a/x].\ \mathsf{rec_W}\ e\ (f\ b)) \\
\mathsf{rec_=}\ e\ a\ h &\equiv e \\
\mathsf{rec_{acc}}\ e\ x\ (\mathsf{intro_{acc}}\ x\ f) &\equiv e\ x\ f\ (\lambda (y : \alpha)\ (h : r\ y\ x).\ \mathsf{rec_{acc}}\ e\ y\ (f\ y\ h))
\end{aligned}$$

which are valid in any context that typechecks everything on the LHS.

Here are a few additional type operators that can be defined from the ones given:

$$\mathbf{0}_\ell := \mathsf{ulift}^\ell\ \bot \qquad \top := \bot \to \bot \qquad \mathbf{1}_\ell := \mathsf{ulift}^\ell\ \top \qquad \alpha \times \beta := \Sigma\_ : \alpha.\ \beta$$

$$p \wedge q := \|p \times q\| \qquad p \vee q := \|p + q\|$$

$$\{x : \alpha \mid p\} := \Sigma x : \alpha.\ p \qquad \exists x : \alpha.\ p := \|\{x : \alpha \mid p\}\|$$

The following additional "η rules" are needed for the reduction, which are provable but not definitional equalities in Lean. Since we are going for soundness only, we will help ourselves to this modest strengthening of the system; moreover this is only for convenience – without such η rules we would only be able to go as far as indexed W-types, which are more complex. (These rules are also required for this axiomatization since we've omitted the recursors in favor of projections for Σ and ulift.)

$$\uparrow\downarrow x \equiv x \qquad (\pi_1\ x, \pi_2\ x) \equiv x$$

The results of section 4 apply straightforwardly to this setting, with these two rules added as $\rightsquigarrow_\kappa$ reduction rules along with all the ι rules mentioned above.

5.2 Translating type families

Let us first suppose that the inductive family lives in a universe $1 \le \ell$. In this case we don't have to worry about $\mathbb{P}$ and small elimination. The idea is to eliminate families by first erasing the indices to get a "skeleton" type S that mixes all the different members of the family together, and then separately define a predicate $\mathsf{good} : S \to \forall x :: \alpha.\mathbb{P}$ that carves out the members that actually belong to index x. The final result will be the type $\lambda x :: \alpha.\ \{s : S \mid \mathsf{good}\ s\ x\}$. For example, the type

$$X = \mu t : \mathbb{N} \to \mathsf{U}_1.\ (\mathsf{one} : t\ 1) + (\mathsf{double} : \forall n : \mathbb{N}.\ t\ n \to t\ (2n))$$

has the indices erased to get

$$S = \mu t : \mathsf{U}_1.\ (\mathsf{one} : t) + (\mathsf{double} : \forall n : \mathbb{N}.\ t \to t),$$

and then the predicate is defined by recursion on S:

$$\mathsf{good}\ \mathsf{one}\ m := m = 1$$
$$\mathsf{good}\ (\mathsf{double}\ n\ x)\ m := m = 2n \wedge \mathsf{good}\ x\ n$$

Now S will also be reduced to the W-type:

$$S' = \mathsf{W}x : \mathbf{1} + \mathbb{N}.\ \mathsf{rec}_+\ (\lambda\_.\ \mathbf{0})\ (\lambda n.\ \mathbf{1})$$

because there are two branches, one with no non-recursive arguments and one with a non-recursive argument of type $\mathbb{N}$ (hence $\mathbf{1} + \mathbb{N}$), and first branch has no recursive arguments and the second has one.

So the general translation will take the form

$$P\ x \simeq \{s : \mathsf{W}p : A.\ B\ p \mid \mathsf{rec}_\mathsf{W}\ (\lambda(p : A)\ \_.\ G\ p)\ s\ x\},$$

$$\begin{array}{ll}\text{where} & \Gamma \vdash A : \mathsf{U}_\ell \\ & \Gamma \vdash B : A \to \mathsf{U}_\ell \\ & \Gamma \vdash G : \forall p : A.\ (B\ p \to \forall x :: \alpha.\ \mathbb{P}) \to \forall x :: \alpha.\ \mathbb{P}.\end{array}$$

We will construct these three terms recursively based on the derivation of the spec judgment.

$$\boxed{\Gamma; t : F \vdash K\ \mathsf{spec} \Rightarrow A; B; G}$$

$$\frac{1 \le \ell \quad \Gamma \vdash x :: \alpha}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash 0\ \mathsf{spec} \Rightarrow \mathbf{0}; \mathsf{rec}_0; \mathsf{rec}_0}$$

$$\frac{\Gamma; t : F \vdash \beta\ \mathsf{ctor} \Rightarrow A_1; p.B_1; pgx.G_1 \quad \Gamma; t : F \vdash K\ \mathsf{spec} \Rightarrow A; B; G}{\Gamma; t : F \vdash (c : \beta) + K\ \mathsf{spec} \Rightarrow A_1 + A; \mathsf{rec}_+\ (\lambda p.B_1)\ B; \mathsf{rec}_+\ (\lambda pg\ (x :: \alpha).\ G_1)\ G}$$

$$\boxed{\Gamma; t : F \vdash \beta\ \mathsf{ctor} \Rightarrow A; p.B; pgx.G}$$

$$\frac{\Gamma \vdash e :: \alpha}{\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash t\ e\ \mathsf{ctor} \Rightarrow \mathbf{1}_\ell;\ p.\ \mathbf{0}_\ell;\ pgx.\ x = e}$$

$$\frac{\Gamma \vdash \beta : \mathsf{U}_{\ell'} \quad \ell' \le \ell \quad \Gamma, y : \beta; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \tau\ \mathsf{ctor} \Rightarrow A; p.B; pgx.G}{\begin{array}{c}\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \forall y : \beta.\ \tau\ \mathsf{ctor} \Rightarrow \Sigma y' : \beta.\ A[y'/y]; \\ p'.B[\pi_1\ p'/y][\pi_2\ p'/p];\ p'gx.G[\pi_1\ p'/y][\pi_2\ p'/p]\end{array}}$$

$$\frac{\begin{array}{c}\Gamma \vdash \gamma :: \mathsf{U}_{\ell'} \quad \Gamma, z :: \gamma \vdash e :: \alpha \quad \ell'_i \le \ell \\ \Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \tau\ \mathsf{ctor} \Rightarrow A; p.B; pg'x.G\end{array}}{\begin{array}{c}\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash (\forall z :: \gamma.\ t\ e) \to \tau\ \mathsf{ctor} \Rightarrow A;\ p.\ \Sigma(z :: \gamma) + B; \\ pgx.\ G[\lambda b.\ g\ (\mathsf{inr}\ b)/g'] \wedge \forall z :: \gamma.\ g\ (\mathsf{inl}\ (z))\ e\end{array}}$$

In the final rule, the notation (z) where $z :: \gamma$ means the tuple of elements of z of type $\Sigma(z :: \gamma)$: explicitly, $(z_1, \ldots, z_n) = (z_1, (z_2, \ldots, (x_n, ())))) : \Sigma(z :: \gamma)$. Note that in the base case of ctor, we have $x = e$ where x and e are telescopes; this can be defined as $(x) = (e)$, or using heterogeneous equality $x_1 = e_1 \wedge x_2 == e_2 \wedge \cdots \wedge x_n == e_n$, or using the equality recursor $\exists(h_1 : x_1 = e_1)\ (h_2 : \mathsf{rec}_=\ x_2\ x_1\ h_1 = e_2) \ldots$. We will use $(x) = (e)$ since it is the least notationally burdensome of these options.

The final result is given by the following translation:

$$\frac{\Gamma; t : F \vdash K\ \mathsf{spec} \Rightarrow A; B; G}{\Gamma \vdash [\![\mu t : F.\ K]\!] = \lambda x :: \alpha.\ \{s : \mathsf{W} p : A.\ B\ p \mid \mathsf{rec}_\mathsf{W}\ (\lambda(p : A)\ \_.\ G\ p)\ s\ x\}}$$

In the case of a small eliminator, we just artificially lift the target universe above 1, translate it, and then propositionally truncate the resulting type and lift if back to the original universe ℓ:

$$\frac{\Gamma; t : F \vdash K\ \mathsf{spec} \quad \neg(\Gamma; t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash K\ \mathsf{LE})}{\Gamma \vdash [\![\mu t : \forall x :: \alpha.\ \mathsf{U}_\ell.\ K]\!] = \lambda x :: \alpha.\ \mathsf{ulift}^\ell \| [\![\mu t : \forall x :: \alpha.\ \mathsf{U}_{\ell'}.\ K]\!]\ x \|},$$

where ℓ' is the maximum of 1 and all the constructor arguments. The idea here is that since we have a small eliminator, it's impossible to tell that members of the inductive type are distinct, so we lose nothing in the propositional truncation.

5.3 Translating subsingleton eliminators

The hard case is when we have a subsingleton eliminator. In this case we must abandon W-types entirely, since we have to produce a subsingleton family from the start – propositional truncation will destroy the large elimination property, so we have to use acc instead. The zero case is easy:

$$\frac{\Gamma \vdash x :: \alpha}{\Gamma \vdash [\![\mu t : \forall x :: \alpha.\ \mathsf{U}_\ell.\ 0]\!] = \lambda x :: \alpha.\ \mathbf{0}_\ell}$$

For our purposes it will be easier to work with the following variant on acc:

$$\begin{aligned}\alpha : \mathsf{U}_\ell, \varphi : \alpha \to \mathbb{P}, r : \alpha \to \alpha \to \mathbb{P} \vdash \mathsf{acc}_r^\varphi = \mu t : \alpha \to \mathbb{P}.\\ (\mathsf{intro} : \forall x : \alpha.\ \varphi\ x \to (\forall y : \alpha.\ r\ y\ x \to t\ y) \to t\ x)\end{aligned}$$

This is just the same as acc_r but for the additional parameter φ that restricts the satisfying instances. This can be built from plain acc in our existing axiomatization as follows:

$$\begin{aligned}\mathsf{acc}_r^\varphi\ x &:= \exists h : \varphi\ x.\ \mathsf{acc}_{r'}\ (x, h)\\ \text{where} \qquad r' &:= \lambda x\ x' : \{x : \alpha \mid \varphi\ x\}.\ r\ (\pi_1\ x)\ (\pi_1\ x')\end{aligned}$$

Large elimination for acc_r^φ is derivable because $\exists h : p.\ q$ has projections when p is a proposition.

In the translation, we must pack up the family into a single type and then use acc for the recursive instances. Let us run an example first:

$$P = \mu t : \mathbb{N} \to \mathbb{N} \to \mathbb{P}.\ (\mathsf{intro} : \forall n : \mathbb{N}.\ n > 2 \to (\forall m.\ m < n \to t\ n\ m) \to t\ 0\ n)$$

This is a large eliminating type because of the constructor's three arguments, one appears in the result $(t\ 0\ n)$, one is a proposition $(n > 2)$, and one is recursive $(\forall m.\ m < n \to t\ n\ m)$.

First we pack the domain into a sigma type, in this case $\mathbb{N} \times \mathbb{N}$, and the propositional constraints go into φ. The recursive arguments become the edge relation for acc. Here, (a, b) is accessible when there exists an n such that $(a, b) = (0, n)$, $n > 2$ and for all $m < n$, (n, m) is accessible, so we translate this to $\varphi(a, b)$ iff there exists n such that $(a, b) = (0, n)$ and $n > 2$, and $r\ (a', b')\ (a, b)$ iff there exists m, n such that $(a, b) = (0, n)$ and $m < n$ and $(a', b') = (n, m)$.

In both clauses we introduce a variable n equal to b or b', and this variable can be eliminated. This is true generally because of the restriction on large eliminators: every non-propositional nonrecursive argument, like n here, must appear in the output type, yielding a variable-variable equality $n = b$ which can be used to eliminate n. However, due to potential dependencies on earlier arguments, we will delay this elimination to the recursor. So in this translation we have:

$$\begin{aligned}P\ x \simeq \mathsf{acc}_r^\varphi\ (x) \qquad \text{where} \qquad \Gamma \vdash \varphi &:= \lambda p : \Sigma(x :: \alpha).\ B[p/(x)]\\ \Gamma \vdash r &:= \lambda p\ q : \Sigma(x :: \alpha).\ R[p/(x')][q/(x)]\\ \Gamma, x :: \alpha \vdash B &: \mathbb{P}\\ \Gamma, x' :: \alpha, x :: \alpha \vdash R &: \mathbb{P}\end{aligned}$$

where we must specify the definition of B and R inductively with the displayed free variables. Here the notation $B[p/(x)]$ means to replace each x_i with the appropriate projection $\pi_1(\pi_2^i\ p)$ in B. We will also accumulate an auxiliary $\Gamma, x' :: \alpha, x :: \alpha \vdash S : \mathbb{P}$ for constructing the disjunctions in R.

$$\boxed{\Gamma; t : F \vdash \tau \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.B; x'x.[S; R]}$$

$$\frac{}{\Gamma; t : F \vdash t\ e \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.\ x = e;\ \ x'x.\ [x = e; \bot]}$$

$$\frac{\Gamma, t : F \vdash \beta : \mathsf{U}_\ell \quad \Gamma, y : \beta; t : F \vdash \tau \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.B; x'x.[S; R]}{\Gamma; t : F \vdash \forall y : \beta.\ \tau \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.\exists y : \beta.\ B; x'x.[\exists y : \beta.\ S; \exists y : \beta.\ R]}$$

$$\frac{\Gamma; t : F \vdash \beta \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.B; x'x.[S; R]}{\Gamma; t : F \vdash (\forall z :: \gamma.\ t\ e) \to \beta \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.B; x'x.[S; (S \wedge \exists z :: \gamma.\ x' = e) \vee R]}$$

Intuitively, S collects the facts that are true about the main instance argument x, so that in each recursive constructor we push a conjunction of S with the fact $\exists z :: \gamma.\ x' = e$ we need to hold for x'. Since we do the same thing for propositional and index arguments (just existentially generalize everything), we have collapsed both into one rule. Once we have constructed the term, we have the following rule:

$$\frac{\Gamma, t : \forall x :: \alpha.\ \mathsf{U}_\ell \vdash \beta \ \mathsf{LE}\ \mathsf{ctor} \Rightarrow x.B; x'x.[S; R]}{\Gamma \vdash [\![\mu t : \forall x :: \alpha.\ \mathsf{U}_\ell.\ (c : \beta)]\!] = \lambda x :: \alpha.\ \mathsf{ulift}^\ell(\mathsf{acc}_r^\varphi(x))}$$

where, as before, $\varphi := \lambda p : \Sigma(x :: \alpha).\ B[p/(x)]$
and $r := \lambda p\ q : \Sigma(x :: \alpha).\ R[p/(x')][q/(x)]$.

5.4 The remainder

We have described the translation of a recursive type in great detail, but it still remains to define the introduction rules and the recursor, and show that the iota rule holds definitionally with these definitions. As these are more or less uniquely determined by the translated type of the inductive type itself, and it is yet more cumbersome than what has been thus far written, this will be left as future work, probably as part of a formalization of all of this. For now, we will proceed with the understanding that the eight inductive types $\bot, \Sigma, +, \mathsf{ulift}, \|\cdot\|, \mathsf{W}, =, \mathsf{acc}$ are indeed sufficient to cover all Lean-definable inductive types, and leave all this horrible induction behind.

6 Soundness

6.1 Proof splitting

The first step in our proof of soundness will be to translate the entire language into one in which the propositional forall and the non-propositional Pi type are syntactically separate, so that we can translate them straightforwardly.

Most of the type rules are the same, with all references to levels ℓ replaced by natural numbers n. The lambda and forall rules are split as follows:

$$e ::= \cdots \mid \forall x : e.\ e \mid \Pi x : e.\ e \mid \lambda x : e.\ e \mid \Lambda x : e.\ e \mid e\ e \mid e \cdot e$$

$$\frac{\Gamma \vdash e_1 : \Pi x : \alpha.\ \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1 \cdot e_2 : \beta[e_2/x]} \qquad \frac{\Gamma \vdash e_1 : \forall x : \alpha.\ \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1\ e_2 : \beta[e_2/x]}$$

$$\frac{\Gamma, x : \alpha \vdash e : \beta : \mathsf{U}_n \quad 1 \le n}{\Gamma \vdash \Lambda x : \alpha.\ e : \Pi x : \alpha.\ \beta} \qquad \frac{\Gamma, x : \alpha \vdash e : \beta : \mathbb{P}}{\Gamma \vdash \lambda x : \alpha.\ e : \forall x : \alpha.\ \beta}$$

$$\frac{\Gamma \vdash \alpha : \mathsf{U}_{n_1} \quad \Gamma, x : \alpha \vdash \beta : \mathsf{U}_{n_2} \quad 1 \le n_2}{\Gamma \vdash \Pi x : \alpha.\ \beta : \mathsf{U}_{\max(n_1, n_2)}} \qquad \frac{\Gamma \vdash \alpha : \mathsf{U}_n \quad \Gamma, x : \alpha \vdash \beta : \mathbb{P}}{\Gamma \vdash \forall x : \alpha.\ \beta : \mathbb{P}}$$

$$\frac{\Gamma, x : \alpha \vdash e : \beta : \mathsf{U}_n \quad 1 \le n \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash (\Lambda x : \alpha.\ e) \cdot e' \equiv e[e'/x]} \qquad \frac{\Gamma, x : \alpha \vdash e : \beta : \mathbb{P} \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash (\lambda x : \alpha.\ e)\ e' \equiv e[e'/x]}$$

$$\frac{\Gamma \vdash e : \Pi y : \alpha.\ \beta}{\Gamma \vdash \Lambda x : \alpha.\ e \cdot x \equiv e}$$

The translation process fixes a universe valuation v to interpret all the level expressions. Let $UV(\ell)$ denote the set of free universe variables in the level expression ℓ, and similarly with $UV(e)$. (There are no universe binding operations, so all variables are free.) The expression $[\![\ell]\!]_v$ is defined when v is a function with domain containing $UV(\ell)$ and codomain $\mathbb{N}$, as follows:

$$\begin{aligned}
[\![u]\!]_v &= v(u) \\
[\![0]\!]_v &= 0 \\
[\![S\ell]\!]_v &= [\![\ell]\!]_v + 1 \\
[\![\max(\ell, \ell')]\!]_v &= \max([\![\ell]\!]_v, [\![\ell']\!]_v) \\
[\![\mathrm{imax}(\ell, \ell')]\!]_v &= \begin{cases} 0 & \text{if } [\![\ell']\!]_v = 0 \\ \max([\![\ell]\!]_v, [\![\ell']\!]_v) & \text{if } [\![\ell']\!]_v \neq 0 \end{cases}
\end{aligned}$$

An important consequence of unique typing is the lvl and sort functions on well typed types and terms, respectively:

Lemma 6.1 (The lvl and sort functions)**.**

1. *There exists a function* $\mathrm{lvl}_v(\Gamma \vdash \alpha)$ *defined on types such that* $\Gamma \vdash \alpha : \mathsf{U}_\ell$ *implies* $[\![\ell]\!]_v = \mathrm{lvl}_v(\Gamma \vdash \alpha)$.
2. *If* $\Gamma \vdash \alpha \equiv \beta$, *then* $\mathrm{lvl}(\Gamma \vdash \alpha) = \mathrm{lvl}(\Gamma \vdash \beta)$ *if either is defined.*
3. *There exists a function* $\mathrm{sort}_v(\Gamma \vdash e)$ *on well typed terms such that if* $\Gamma \vdash e : \alpha$, *then* $\mathrm{sort}(\Gamma \vdash e) = \mathrm{lvl}(\Gamma \vdash \alpha)$.

Proof.

1. By unique typing, if $\Gamma \vdash \alpha : \mathsf{U}_\ell$ and $\Gamma \vdash \alpha : \mathsf{U}_{\ell'}$, then $\ell \equiv \ell'$, so $[\![\ell]\!]_v = [\![\ell']\!]_v$. Therefore $\mathrm{lvl}_v(\Gamma \vdash \alpha)$ is unique, and exists by definition.
2. If $\Gamma \vdash \alpha : \mathsf{U}_\ell$ and $\Gamma \vdash \alpha \equiv \beta$, then $\Gamma \vdash \beta : \mathsf{U}_\ell$ as well, so $\mathrm{lvl}(\Gamma \vdash \alpha) = \mathrm{lvl}(\Gamma \vdash \beta)$.
3. If $\Gamma \vdash e : \alpha$ and $\Gamma \vdash e : \beta$, then by unique typing $\Gamma \vdash \alpha \equiv \beta$, so $\mathrm{lvl}(\Gamma \vdash \alpha) = \mathrm{lvl}(\Gamma \vdash \beta)$ by the previous part. Thus $\mathrm{sort}(\Gamma \vdash e)$ is well defined. □

Well typed terms are translated in a context. (The universe valuation v is suppressed in the rules.)

- $\langle x \rangle_\Gamma = x$

- $\langle \mathsf{U}_\ell \rangle_\Gamma = \mathsf{U}_{[\![\ell]\!]}$
- $\langle e_1\ e_2 \rangle_\Gamma = \begin{cases} \langle e_1 \rangle_\Gamma\ \langle e_2 \rangle_\Gamma & \text{if } \mathrm{sort}(\Gamma \vdash e_1) = 0 \\ \langle e_1 \rangle_\Gamma \cdot \langle e_2 \rangle_\Gamma & \text{if } \mathrm{sort}(\Gamma \vdash e_1) \geq 1 \end{cases}$
- $\langle \lambda x : \alpha.\ e \rangle_\Gamma = \begin{cases} \lambda x : \langle \alpha \rangle_\Gamma.\ \langle e \rangle_{\Gamma, x:\alpha} & \text{if } \mathrm{sort}(\Gamma \vdash e) = 0 \\ \Lambda x : \langle \alpha \rangle_\Gamma.\ \langle e \rangle_{\Gamma, x:\alpha} & \text{if } \mathrm{sort}(\Gamma \vdash e) \geq 1 \end{cases}$
- $\langle \forall x : \alpha.\ \beta \rangle_\Gamma = \begin{cases} \forall x : \langle \alpha \rangle_\Gamma.\ \langle \beta \rangle_{\Gamma, x:\alpha} & \text{if } \mathrm{lvl}(\Gamma \vdash \beta) = 0 \\ \Pi x : \langle \alpha \rangle_\Gamma.\ \langle e \rangle_{\Gamma, x:\alpha} & \text{if } \mathrm{lvl}(\Gamma \vdash \beta) \geq 1 \end{cases}$
- Other terms are translated simply by translating their parts.

Theorem 6.2 (Translation of terms)**.** *If $\Gamma \vdash e : \alpha$, then $\langle e \rangle_\Gamma$ is defined.*

Proof. By induction, using the assumption to show that the sort and lvl functions are only applied to well typed terms. □

We can translate whole contexts by the rule $\langle \cdot \rangle = \cdot$, $\langle \Gamma, x : \alpha \rangle = \langle \Gamma \rangle, x : \langle \alpha \rangle_\Gamma$.

Theorem 6.3 (Type preservation of the translation)**.**

1. *If $\Gamma \vdash e : \alpha$, then $\langle \Gamma \rangle \vdash \langle e \rangle_\Gamma : \langle \alpha \rangle_\Gamma$.*
2. *If $\Gamma \vdash e \equiv e'$, then $\langle \Gamma \rangle \vdash \langle e \rangle_\Gamma : \langle e' \rangle_\Gamma$*

Proof. The proof is straightforward by induction on $\Gamma \vdash e : \alpha$ and $\Gamma \vdash e \equiv e'$. □

The reverse translation is even easier to describe, and does not need a context:

- $\overline{\lambda x : \alpha.\ e} = \overline{\Lambda x : \alpha.\ e} = \lambda x : \overline{\alpha}.\ \overline{e}$
- $\overline{\forall x : \alpha.\ e} = \overline{\Pi x : \alpha.\ e} = \forall x : \overline{\alpha}.\ \overline{e}$
- $\overline{e_1\ e_2} = \overline{e_1 \cdot e_2} = \overline{e_1}\ \overline{e_2}$
- $\overline{\mathsf{U}_n} = \mathsf{U}_{\overline{n}}$, where $\overline{n}$ is the level expression corresponding to n, i.e. $SS \ldots S0$ with n S-applications.
- Otherwise, the translation is recursive in subterms.

We have type preservation in this direction as well:

Lemma 6.4 (Type preservation of reverse translation)**.**

1. *If $\Gamma \vdash e : \alpha$ then $\overline{\Gamma} \vdash \overline{e} : \overline{\alpha}$.*
2. *If $\Gamma \vdash e_1 \equiv e_2$ then $\overline{\Gamma} \vdash \overline{e_1} \equiv \overline{e_2}$.*

Lemma 6.5 (Bijection of reverse translation)**.**

1. *If $\Gamma \vdash e : \alpha$ then $\overline{\langle e \rangle_\Gamma}$ and e are equal except at level arguments, with the levels being equivalent after substitution of $\overline{v(u)}$ for each universe variable u.*
2. *If $\Gamma \vdash e : \alpha$ in the proof split language then $\langle \overline{e} \rangle_{\overline{\Gamma}} = e$.*
3. *$\overline{e}$ and $\overline{\Gamma}$ have no universe variables.*

Proof. Straightforward by induction. □

The existence of the reverse translation implies unique typing for the proof split language, so the lvl and sort functions are also well defined in this language and have the same values as their translations do.

Although this type theory is less expressive than the original due to the lack of universe parametricity, it is sufficient to capture situations where the universes have been fixed, in particular in evaluation and in proofs of contradiction, which can have all universe variables set to zero while preserving the proof. This is why we will use it as the source language for the ZFC translation.

6.2 Modeling Lean in ZFC

Fix an increasing sequence $(\kappa_n)_{n\in\mathbb{N}}$ of strong limit cardinals. We will say that the sequence is n-correct if $\kappa_0, \dots, \kappa_{n-1}$ are all inaccessible cardinals (that is, the cofinality of κ_k is κ_k for all $k < n$). The sequence is ω-correct if it is n-correct for all n.

Note that a sequence satisfying the given properties can be proven to exist in ZFC, and a sequence which is n-correct can be proven to exist in ZFC + "there are at least n inaccessible cardinals", since for any cardinal μ, $\beth_\omega(\mu)$ is a strong limit cardinal greater than μ.

Now let $U_0 = \{\emptyset, \{\bullet\}\}$ where $\bullet = \emptyset$ is the "proof object" (the exact identity of $\bullet$ is not important), and let $U_{n+1} = V_{\kappa_n}$.

Let $\kappa_\omega = \sup_{n<\omega} \kappa_n$, and $U_\omega = V_{\kappa_\omega} = \bigcup_{n<\omega} U_n$. Fix a choice function ε on U_ω, that is, a function such that $\varepsilon(x) \in x$ for all $x \in U_\omega$.

We will use the proof-split language for the interpretation map, so we do not need to worry about translating level expressions, which have already been removed from the terms. We define the expressions $[\![\Gamma]\!]$ when Γ is a well formed context, and $[\![\Gamma \vdash e]\!]$ when $\Gamma \vdash e : \alpha$ for some α, by mutual recursion on the following measure:

- The size of an expression $|e|$ is the sum of all its immediate subterms plus 1, except:
  - $|\mathsf{let}\ x : \alpha := e_1\ \mathsf{in}\ e_2| = |e_2[e_1/x]| + 1$, and
  - $|c| = |e| + 1$ when $\mathsf{def}\ c : \alpha := e$.
- The size of a context is $|\cdot| = 1/2$, $|\Gamma, x : \alpha| = |\Gamma| + |\alpha|$.
- The size of an expression in context is $|\Gamma \vdash e| = |\Gamma| + |e| - 1/2$.

Note that $|\Gamma| < |\Gamma \vdash e|$ and $|\Gamma \vdash \alpha| < |\Gamma, x : \alpha|$. Here $[\![\Gamma]\!]$ will be a set of lists of types, and $[\![\Gamma \vdash e]\!]$ will be a (total) function on $[\![\Gamma]\!]$, if it is defined at all. We will denote the evaluation at $\gamma \in [\![\Gamma]\!]$ by $[\![\Gamma \vdash e]\!]_\gamma$.

Let $[p] = \{x \in \{\bullet\} \mid p\}$, and $\bar{R} = \{(a, b) \mid \bullet \in R(a)(b)\}$, which translates DTT predicates and relations into their ZFC counterparts.

- $[\![\cdot]\!] = \{()\}$, the singleton of the empty list. (This can be encoded as $\emptyset$.)
- $[\![\Gamma, x : \alpha]\!] = \sum_{\gamma\in[\![\Gamma]\!]} [\![\Gamma \vdash \alpha]\!]_\gamma$, that is, the set of pairs (γ, x) such that $\gamma \in [\![\Gamma]\!]$ and $x \in [\![\Gamma \vdash \alpha]\!]_\gamma$.
- $[\![\Gamma \vdash x]\!]_\gamma = \pi_i(\gamma)$, where x is the ith variable in the context.
- $[\![\Gamma \vdash \mathsf{U}_n]\!]_\gamma = U_n$
- $[\![\Gamma \vdash e_1\ e_2]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash e_1 \cdot e_2]\!]_\gamma = [\![\Gamma \vdash e_1]\!]_\gamma([\![\Gamma \vdash e_2]\!]_\gamma)$
- $[\![\Gamma \vdash \lambda x : \alpha.\ e]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \Lambda x : \alpha.\ e]\!]_\gamma = (x \in [\![\Gamma \vdash \alpha]\!]_\gamma \mapsto [\![\Gamma, x : \alpha \vdash e]\!]_{(\gamma,x)})$

- $[\![\Gamma \vdash \forall x : \alpha.\ \beta]\!]_\gamma = \{\bullet\} \cap \bigcap_{x \in [\![\Gamma \vdash \alpha]\!]_\gamma} [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)} = [\forall x \in [\![\Gamma \vdash \alpha]\!]_\gamma, \bullet \in [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)}]$
- $[\![\Gamma \vdash \Pi x : \alpha.\ \beta]\!]_\gamma = \prod_{x \in [\![\Gamma \vdash \alpha]\!]_\gamma} [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)}$
- $[\![\Gamma \vdash \mathsf{let}\ x : \alpha := e_1\ \mathsf{in}\ e_2]\!]_\gamma = [\![\Gamma \vdash e_2[e_1/x]]\!]_\gamma$
- $[\![\Gamma \vdash c]\!]_\gamma = [\![\Gamma \vdash e]\!]_\gamma$ when $\mathsf{def}\ c : \alpha := e$
- $[\![\Gamma \vdash \bot]\!]_\gamma = \emptyset$
- $[\![\Gamma \vdash \mathsf{rec}_\bot]\!]_\gamma = \emptyset$ (the empty function)
- $[\![\Gamma \vdash \Sigma x : \alpha.\ \beta]\!]_\gamma = \sum_{x \in [\![\Gamma \vdash \alpha]\!]_\gamma} [\![\Gamma \vdash \beta]\!]_{(\gamma,x)}$
- $[\![\Gamma \vdash (e_1, e_2)]\!]_\gamma = ([\![\Gamma \vdash e_1]\!]_\gamma, [\![\Gamma \vdash e_2]\!]_\gamma)$
- $[\![\Gamma \vdash \pi_1\ e]\!]_\gamma = \pi_1([\![\Gamma \vdash e]\!]_\gamma)$
- $[\![\Gamma \vdash \pi_2\ e]\!]_\gamma = \pi_2([\![\Gamma \vdash e]\!]_\gamma)$
- $[\![\Gamma \vdash \alpha + \beta]\!]_\gamma = [\![\Gamma \vdash \alpha]\!]_\gamma \sqcup [\![\Gamma \vdash \beta]\!]_\gamma$
- $[\![\Gamma \vdash \mathsf{inl}\ e]\!]_\gamma = \iota_1([\![\Gamma \vdash \alpha]\!]_\gamma)$
- $[\![\Gamma \vdash \mathsf{inr}\ e]\!]_\gamma = \iota_2([\![\Gamma \vdash \beta]\!]_\gamma)$
- $[\![\Gamma \vdash \mathsf{rec}_+\ a\ b]\!]_\gamma$ is the function on $[\![\Gamma \vdash \alpha]\!]_\gamma \sqcup [\![\Gamma \vdash \beta]\!]_\gamma$ such that

$$[\![\Gamma \vdash \mathsf{rec}_+\ a\ b]\!]_\gamma(\iota_1(x)) = [\![\Gamma \vdash a]\!]_\gamma(x) \qquad \text{for } x \in [\![\Gamma \vdash \alpha]\!]_\gamma$$
$$[\![\Gamma \vdash \mathsf{rec}_+\ a\ b]\!]_\gamma(\iota_2(y)) = [\![\Gamma \vdash a]\!]_\gamma(y) \qquad \text{for } y \in [\![\Gamma \vdash \beta]\!]_\gamma.$$

- $[\![\Gamma \vdash \mathsf{ulift}_n^{n'} \alpha]\!]_\gamma = [\![\Gamma \vdash \alpha]\!]_\gamma$
- $[\![\Gamma \vdash \uparrow e]\!]_\gamma = [\![\Gamma \vdash \downarrow e]\!]_\gamma = [\![\Gamma \vdash e]\!]_\gamma$
- $[\![\Gamma \vdash \|\alpha\|]\!]_\gamma = [[\![\Gamma \vdash \alpha]\!]_\gamma \neq \emptyset]$
- $[\![\Gamma \vdash |e|]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \mathsf{rec}_{||}^{C,\alpha}\ f]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \mathsf{W}x : \alpha.\ \beta]\!]_\gamma = \mathsf{W}_{x \in [\![\Gamma \vdash \alpha]\!]_\gamma} [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)}$ (see below)
- $[\![\Gamma \vdash \mathsf{sup}\ a\ f]\!]_\gamma = ([\![\Gamma \vdash a]\!]_\gamma, [\![\Gamma \vdash f]\!]_\gamma)$
- $[\![\Gamma \vdash \mathsf{rec}_\mathsf{W}\ e]\!]_\gamma = \mathsf{rec}_\mathsf{W}(\mathsf{W}_{x \in [\![\Gamma \vdash \alpha]\!]_\gamma} [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)}, [\![\Gamma \vdash e]\!]_\gamma)$ (see below)
- $[\![\Gamma \vdash a = b]\!]_\gamma = [[\![\Gamma \vdash a]\!]_\gamma = [\![\Gamma \vdash b]\!]_\gamma]$
- $[\![\Gamma \vdash \mathsf{refl}\ a]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \mathsf{rec}_=^a\ e\ b\ h]\!]_\gamma = [\![\Gamma \vdash e]\!]_\gamma$
- $[\![\Gamma \vdash \mathsf{acc}_{\alpha,r}\ x]\!]_\gamma = [x \in \mathsf{acc}([\![\Gamma \vdash \alpha]\!]_\gamma, \overline{[\![\Gamma \vdash r]\!]_\gamma})]$ (see below)
- $[\![\Gamma \vdash \mathsf{intro}_\mathsf{acc}\ x\ f]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \mathsf{rec}_\mathsf{acc}^r\ e]\!]_\gamma = \mathsf{rec}_\mathsf{acc}([\![\Gamma \vdash \alpha]\!]_\gamma, \overline{[\![\Gamma \vdash r]\!]_\gamma}, [\![\Gamma \vdash e]\!]_\gamma)$ (see below)
- Let $\sim$ be the equivalence closure of $\overline{[\![\Gamma \vdash R]\!]_\gamma}$ in the following clauses:
  - $[\![\Gamma \vdash \alpha/R]\!]_\gamma = [\![\Gamma \vdash \alpha]\!]_\gamma/\sim$
  - $[\![\Gamma \vdash \mathsf{mk}_R\ x]\!]_\gamma = [[\![\Gamma \vdash x]\!]_\gamma]_\sim$
  - $[\![\Gamma \vdash \mathsf{lift}_R\ \beta\ f\ h]\!]_\gamma$ is the function such that $[\![\Gamma \vdash \mathsf{lift}_R\ \beta\ f\ h]\!]_\gamma([x]_\sim) = [\![\Gamma \vdash f]\!]_\gamma(x)$
- $[\![\Gamma \vdash \mathsf{sound}_R]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \mathsf{propext}]\!]_\gamma = \bullet$
- $[\![\Gamma \vdash \mathsf{choice}\ \alpha\ h]\!]_\gamma = \varepsilon([\![\Gamma \vdash \alpha]\!]_\gamma)$

6.2.1 Definition of W-types in ZFC

If A is a set and $B(x)$ is a family of sets indexed by $x \in A$, then $\mathsf{W}_{x\in A}B(x)$ is a set, defined as the intersection of all sets W such that $(a, f) \in W$ whenever $a \in A$ and $f : B(a) \to W$. If $\mathrm{cf}(\lambda) > \sup_{x\in A}|B(x)|$, then V_λ is an upper bound for W, since $\mathrm{rank} \circ f$ is a sequence of ordinals of length $B(a) < \mathrm{cf}(\lambda)$. Thus, U_{n+1} is closed under W-types if the κ sequence is $(n + 1)$-correct.

The recursor $F := \mathsf{rec}_{\mathsf{W}}(W, e) : W \to V$ is defined by transfinite recursion on $x \in W$: Assuming that $F(y)$ is defined for all y with $\mathrm{rank}\, y < \mathrm{rank}\, x$, we let $F(a, f) = e(a)(f)(F \circ f)$ when $x = (a, f)$ is a pair. Note that $\mathrm{rank}\, f(y) < \mathrm{rank}\, f < \mathrm{rank}\, x$ for all $y \in \mathrm{dom}\, f$, so the function is well-defined.

6.2.2 Definition of acc in ZFC

If $R \subseteq A \times A$ is a relation, $\mathsf{acc}(A, R) \subseteq A$ is the set of all elements $x \in A$ that are R-accessible, or equivalently, R restricted to the set $\{y \mid (y, x) \in R^*\}$ is well-founded. It is the smallest set C such that if $y \in C$ for all y such that $(y, x) \in R$, then $x \in C$.

Note that $\mathsf{acc}(A, R)$ is itself well-founded by R, because any descending sequence in $\mathsf{acc}(A, R)$ must begin at some point of it, and can only continue finitely long from there. We define the recursor as $\mathsf{rec}_{\mathsf{acc}}(A, R, e) = (x \in A \mapsto (h \in [x \in \mathsf{acc}(A, R)] \mapsto F(x))$, where $F : \mathsf{acc}(A, R) \to V$ is defined by recursion on $\mathsf{acc}(A, R)$ along R, such that $F(x) = e(x)(\bullet)(y \in A \mapsto (h \in [(y, x) \in R] \mapsto F(y)))$.

Remark 1. *If λ is an inaccessible cardinal, and $A \in V_\lambda$, $B : A \to V_\lambda$, then $\prod_{x\in A} B(x) \in V_\lambda$, because every element of V_λ has cardinality $< \lambda$, so B is bounded in rank (by some $\mu < \lambda$), and then the rank of $\prod_{x\in A} B(x)$ is at most $\mu + 4$ or so.*

6.3 Soundness

Lemma 6.6 (Basics)**.**

- *(Weakening) If $\Gamma \vdash e : \alpha$ and $\vdash \Gamma, \Delta\ \mathsf{ok}$, and $(\gamma, \delta) \in [\![\Gamma, \Delta]\!]$, then $[\![\Gamma, \Delta \vdash e]\!]_{\gamma,\delta} = [\![\Gamma \vdash e]\!]_\gamma$.*
- *(Substitution) If $\Gamma, x : \alpha \vdash e_1 : \beta$, $\Gamma \vdash e_2 : \alpha$, $\gamma \in [\![\Gamma]\!]$, and $z := [\![\Gamma \vdash e_2]\!]_\gamma \in [\![\Gamma \vdash \alpha]\!]_\gamma$, then $[\![\Gamma \vdash e_1[e_2/x]]\!]_\gamma = [\![\Gamma, x : \alpha \vdash e_1]\!]_{(\gamma,z)}$.*

Proof. Straightforward. (In the substitution lemma, we are assuming soundness for e_2, because we haven't proven it yet.) □

Theorem 6.7 (Soundness)**.**

- *If $\Gamma \vdash \alpha : \mathbb{P}$, then $[\![\Gamma \vdash \alpha]\!]_\gamma \subseteq \{\bullet\}$.*
- *If $\Gamma \vdash e : \alpha$ and $\mathrm{lvl}(\Gamma \vdash \alpha) = 0$, then $[\![\Gamma \vdash e]\!]_\gamma = \bullet$.*
- *If $\Gamma \vdash e : \alpha$, then there exists an k such that if the κ sequence is k-correct, then for all $\gamma \in [\![\Gamma]\!]$, $[\![\Gamma \vdash e]\!]_\gamma \in [\![\Gamma \vdash \alpha]\!]_\gamma$.*
- *If $\Gamma \vdash e \equiv e'$, then there exists an k such that if the κ sequence is k-correct, then for all $\gamma \in [\![\Gamma]\!]$, $[\![\Gamma \vdash e]\!]_\gamma = [\![\Gamma \vdash e']\!]_\gamma$.*

Proof. The proof is constructive for the value of k; it is essentially just the max of all universe numbers that appear in the course of the proof. We will not spend much time discussing it, but it is worth noting that we may have $\Gamma \vdash e : \alpha$ where $[\![\alpha]\!]$ is not a member of the expected universe, without assuming a higher value of k than the one that appears in the proof.

(There is nothing surprising in this proof, except perhaps the fact that I took the trouble to write it down.)

Part 1 is a special case of part 3, but does not require the k assumption. We will prove it in parallel with the other parts.

For brevity of notation, we will adopt the convention that $\bar{\alpha}$ means $[\![\Gamma \vdash \alpha]\!]_\gamma$, $\bar{\beta}(x)$ means $[\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)}$, and so on, where Γ and γ are understood from context.

- Weakening. We have $[\![\Gamma \vdash e]\!]_\gamma \in [\![\Gamma \vdash \beta]\!]_\gamma$ by the IH, and $[\![\Gamma, x : \alpha \vdash e]\!]_{(\gamma,x)} = [\![\Gamma \vdash e]\!]_\gamma$ and $[\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)} = [\![\Gamma \vdash \beta]\!]_\gamma$ by the weakening lemma, so $[\![\Gamma, x : \alpha \vdash e]\!]_{(\gamma,x)} \in [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma,x)}$.
- Conversion. $\Gamma \vdash e : \alpha$ and $\Gamma \vdash \alpha \equiv \beta$. Then by the IH $\bar{e} \in \bar{\alpha} = \bar{\beta}$. Parts 1 and 2 follow from the first IH, since $\mathrm{lvl}(\Gamma \vdash \alpha) = \mathrm{lvl}(\Gamma \vdash \beta)$.
- Variable. $[\![\Gamma, x : \alpha \vdash x]\!]_{(\gamma,x)} = x$, so $(\gamma, x) \in [\![\Gamma, x : \alpha]\!]$ implies $x \in [\![\Gamma \vdash \alpha]\!]_\gamma = [\![\Gamma, x : \alpha \vdash \alpha]\!]_{(\gamma,x)}$ by the weakening lemma.
- Universe. $[\![\vdash \mathsf{U}_n]\!]_() = U_n \in U_{n+1} = [\![\vdash \mathsf{U}_{n+1}]\!]_()$ since the U universes form a membership hierarchy. Parts 1 and 2 do not apply since $\mathsf{U}_{n+1} \not\equiv \mathbb{P}$.
- Proof application. Suppose $\Gamma \vdash e_1 : \forall x : \alpha.\ \beta$ and $\Gamma \vdash e_2 : \alpha$. By the IH, $\bar{e_1} = \bullet \in \bigcap_{x \in \bar{\alpha}} \bar{\beta}(x)$ and $\bar{e_2}_\gamma \in \bar{\alpha}$, so in particular $[\![\Gamma \vdash e_1\ e_2]\!]_\gamma = \bullet \in \bar{\beta}(\bar{e_2}) = [\![\Gamma \vdash \beta[e_2/x]]\!]_\gamma$ by the substitution lemma.
- Type application. Suppose $\Gamma \vdash e_1 : \Pi x : \alpha.\ \beta$ and $\Gamma \vdash e_2 : \alpha$. By the IH, $\bar{e_1} \in \prod_{x \in \bar{\alpha}} \bar{\beta}(x)$ and $\bar{e_2}_\gamma \in \bar{\alpha}$, so $[\![\Gamma \vdash e_1 \cdot e_2]\!]_\gamma = \bar{e_1}(\bar{e_2}) \in \bar{\beta}(\bar{e_2}) = [\![\Gamma \vdash \beta[e_2/x]]\!]_\gamma$ by the substitution lemma.
- Proof lambda. Suppose $\Gamma, x : \alpha \vdash e : \beta$. By the IH, $\bar{e}(x) = \bullet \in \bar{\beta}(x)$ for all $x \in \bar{\alpha}$, so $[\![\Gamma \vdash \lambda x : \alpha.\ e]\!]_\gamma = \bullet \in \bigcap_{x \in \bar{\alpha}} \bar{\beta}(x)$.
- Type lambda. Suppose $\Gamma, x : \alpha \vdash e : \beta$. By the IH, $\bar{e}(x) \in \bar{\beta}(x)$ for all $x \in \bar{\alpha}$. Thus $[\![\Gamma \vdash \Lambda x : \alpha.\ e]\!]_\gamma = (x \in \bar{\alpha} \mapsto \bar{e}(x)) \in \prod_{x \in \bar{\alpha}} \bar{\beta}(x)$.
- Forall. $[\![\Gamma \vdash \forall x : \alpha.\ \beta]\!]_\gamma = \{\bullet\} \cap \bigcap_{x \in \bar{\alpha}} \bar{\beta}(x) \subseteq \{\bullet\}$.
- Pi. Suppose $\Gamma \vdash \alpha : \mathsf{U}_{n_1}$ and $\Gamma, x : \alpha \vdash \beta : \mathsf{U}_{n_2}$. By the IH, $\bar{\alpha} \in U_{n_1} \subseteq U_k$ and $\bar{\beta}(x) \in U_{n_2} \subseteq U_k$ for all $x \in \bar{\alpha}$, where $k = \max(n_1, n_2)$. Therefore $[\![\Gamma \vdash \Pi x : \alpha.\ \beta]\!]_\gamma = \prod_{x \in \bar{\alpha}} \bar{\beta}(x) \in U_k$, provided that the κ sequence is k-correct, because if κ_{k-1} is inaccessible then U_k is closed under dependent products.
- $\bot$: $[\![\vdash \bot]\!]_() = \emptyset \subseteq \{\bullet\}$.
- $\mathsf{rec}_\bot$: $[\![\Gamma \vdash \mathsf{rec}_\bot^C]\!]_\gamma = \emptyset \in [\![\Gamma \vdash \bot \to C]\!]_\gamma = \prod_{x \in \emptyset} \bar{C}$.
- Σ: Assuming the κ sequence is k-correct where $k = \max(1, n_1, n_2)$, if $\bar{\alpha} \in U_{n_1} \subseteq U_k$ and $\bar{\beta}(x) \in U_{n_2} \subseteq U_k$ for all $x \in \bar{\alpha}$, the family is bounded, so $\sum_{x \in \bar{\alpha}} \bar{\beta}(x) \in U_k$.
- Pair: If $\bar{e_1} \in \bar{\alpha}$ and $\bar{e_2} \in [\![\Gamma \vdash \beta[e_1/x]]\!]_\gamma = \bar{\beta}(\bar{e_1})$, then $[\![\Gamma \vdash (e_1, e_2)]\!]_\gamma = (\bar{e_1}, \bar{e_2}) \in \sum_{x \in \bar{\alpha}} \bar{\beta}(x)$.
- π_1: If $\bar{e} \in \sum_{x \in \bar{\alpha}} \bar{\beta}(x)$, then $[\![\Gamma \vdash \pi_1\ e]\!]_\gamma = \pi_1(\bar{e}) \in \bar{\alpha}$.
- π_2: If $\bar{e} \in \sum_{x \in \bar{\alpha}} \bar{\beta}(x)$, then $[\![\Gamma \vdash \pi_2\ e]\!]_\gamma = \pi_2(\bar{e}) \in \bar{\beta}(\pi_1(\bar{e})) = \bar{\beta}([\![\Gamma \vdash \pi_1\ e]\!]_\gamma) = [\![\Gamma \vdash \beta[\pi_1\ e/x]]\!]_\gamma$.
- $+$: If $k := \max(1, n_1, n_2)$, and $\bar{\alpha} \in U_{n_1} \subseteq U_k$ and $\bar{\beta} \in U_{n_2} \subseteq U_k$, then $[\![\Gamma \vdash \alpha + \beta]\!]_\gamma = \bar{\alpha} \sqcup \bar{\beta} \in U_k$, because $\mathrm{rank}(\bar{\alpha} \sqcup \bar{\beta}) \le \max(\mathrm{rank}\,\bar{\alpha}, \mathrm{rank}\,\bar{\beta}) + 2$ (when encoded as marked pairs), so V_λ is closed under disjoint unions whenever λ is a limit ordinal.
- inl: If $\bar{e} \in \bar{\alpha}$, then $[\![\Gamma \vdash \mathsf{inl}\ e]\!]_\gamma = \iota_1(\bar{e}) \in \bar{\alpha} \sqcup \bar{\beta} = [\![\Gamma \vdash \alpha + \beta]\!]_\gamma$. (We don't need the second IH.) inr is similar.

- rec_+: By the IH, $\bar{C} : \bar{\alpha} \sqcup \bar{\beta} \to U_n$.
  $[\![\Gamma \vdash \mathsf{rec}_\perp^C\ a\ b]\!]_\gamma \in \prod_{x \in \bar{\alpha} \sqcup \bar{\beta}} \bar{C}(x)$ because it was defined as a function such that $[\![\Gamma \vdash \mathsf{rec}_\perp^C\ a\ b]\!]_\gamma(\iota_1(x)) = \bar{a}(x) \in \bar{C}(\iota_1(x))$, and $[\![\Gamma \vdash \mathsf{rec}_\perp^C\ a\ b]\!]_\gamma(\iota_2(x)) = \bar{b}(x) \in \bar{C}(\iota_2(x))$.
- ulift: If $\bar{\alpha} \in U_n$ and $n \le n'$, then $[\![\Gamma \vdash \mathsf{ulift}_n^{n'}\ \alpha]\!]_\gamma = \bar{\alpha} \in U_n \subseteq U_{n'}$.
- $\uparrow$ and $\downarrow$ are trivial from the IH.
- $\|\cdot\|$: $[\![\Gamma \vdash \|\alpha\|]\!]_\gamma = [\bar{\alpha} \neq \emptyset] \subseteq \{\bullet\}$ (we don't need the IH).
- $|\cdot|$: If $\bar{e} \in \bar{\alpha}$, then $\bar{\alpha} \neq \emptyset$ so $[\![\Gamma \vdash |e|]\!]_\gamma = \bullet \in [\bar{\alpha} \neq \emptyset] = [\![\Gamma \vdash \|\alpha\|]\!]_\gamma$.
- $\mathsf{rec}_{||}$: To show $[\![\Gamma \vdash \mathsf{rec}_{||}\ f]\!]_\gamma = \bullet \in [\![\Gamma \vdash \|\alpha\| \to C]\!]_\gamma$, it suffices to show that if $x \in [\bar{\alpha} \neq \emptyset]$ (i.e. $\bar{\alpha} \neq \emptyset$), then $\bullet \in \bar{C}$. Let $y \in \bar{\alpha}$. Then $\bar{f} = \bullet \in \bigcap_{x \in \bar{\alpha}} \bar{C}$, so $\bullet \in \bar{C}$, using $x := y$.
- W: Similar to the Σ case, assuming the κ sequence is k-correct where $k = \max(1, n_1, n_2)$, since we have already observed that V_λ where λ is inaccessible is closed under W-types.
- sup : $[\![\Gamma \vdash \mathsf{sup}\ a\ f]\!]_\gamma = (\bar{a}, \bar{f}) \in \mathsf{W}_{x \in \bar{\alpha}} \bar{\beta}(x)$ since $\bar{a} \in \bar{\alpha}$ and $\bar{f} : \bar{\beta}(\bar{a}) \to \mathsf{W}_{x \in \bar{\alpha}} \bar{\beta}(x)$. (Application of the definition, IH, and substitution theorem.)
- rec_W: Let $W := [\![\Gamma \vdash \mathsf{W}x : \alpha.\ \beta]\!]_\gamma = \mathsf{W}_{x \in \bar{\alpha}} \bar{\beta}(x)$. By the IH and applying the definitions,
  $$\bar{e} := \bar{e} \in \prod_{a \in \bar{\alpha}} \prod_{f : \bar{\beta}(a) \to W} \Big[\prod_{b \in \bar{\beta}(a)} \bar{C}(f(b)) \Big] \to \bar{C}(a, f).$$
  Now $[\![\Gamma \vdash \mathsf{rec}_\mathsf{W}\ e]\!]_\gamma = \mathsf{rec}_\mathsf{W}(W, \bar{e}) =: F$ is defined as a function on W, so it suffices to check that $F(w) \in \bar{C}(w)$ for all $w \in W$. By induction on $w \in W$, it suffices to check that $F(a, f) \in \bar{C}(a, f)$ when $a \in \bar{\alpha}$ and $f : \bar{\beta}(a) \to W$ satisfies $F(f(b)) \in \bar{C}(f(b))$ for all $b \in \bar{\beta}(a)$. Since $F(a, f) = \bar{e}(a)(f)(F \circ f)$, and $F \circ f \in \prod_{b \in \bar{\beta}(a)} \bar{C}(f(b))$ because $F(f(b)) \in \bar{C}(f(b))$ by assumption, we have $\bar{e}(a)(f)(F \circ f) \in \bar{C}(a, f)$ as desired.
- Equality: $[\![\Gamma \vdash a = b]\!]_\gamma = [\bar{a} = \bar{b}] \subseteq \{\bullet\}$ (we don't need the IH).
- refl: $[\![\Gamma \vdash \mathsf{refl}\ a]\!]_\gamma = \bullet \in [\![\Gamma \vdash a = a]\!]_\gamma = [\bar{a} = \bar{a}]$ because $\bar{a} = \bar{a}$ (we don't need the IH).
- $\mathsf{rec}_=$: We have $\bar{a} \in \bar{\alpha}$, $\bar{C} : \bar{\alpha} \to U_n$, and $\bar{e} \in \bar{C}(\bar{a})$ from the IH. To show $[\![\mathsf{rec}_=\ e]\!]_\gamma \in [\![\forall b : \alpha.\ a = b \to C\ b]\!]_\gamma = \prod_{b \in \bar{\alpha}} \prod_{h \in [\bar{a} = b]} \bar{C}(b)$, suppose $b \in \bar{\alpha}$ and $h \in [\bar{a} = b]$. Then $\bar{a} = b$, so $[\![\mathsf{rec}_=\ e]\!]_\gamma(b)(h) = \bar{e} \in \bar{C}(\bar{a}) = \bar{C}(b)$.
- acc: If $\bar{r} : \bar{\alpha} \to \bar{\alpha} \to U_0$ and $\bar{x} \in \bar{\alpha}$, then $[\![\Gamma \vdash \mathsf{acc}_r\ x]\!]_\gamma = [x \in \mathsf{acc}(\bar{\alpha}, \bar{r})] \subseteq \{\bullet\}$.
- $\mathsf{intro}_\mathsf{acc}$: If $\bar{x} : \bar{\alpha}$ and $\bar{f} = \bullet \in \bigcap_{y \in \bar{\alpha}} [\bullet \in (y, \bar{x}) \in \bar{r}] \to [y \in \mathsf{acc}(\bar{\alpha}, \bar{r})]$ (applying the definitions and IH), then for all $y \in \bar{\alpha}$ with $(y, \bar{x}) \in \bar{r}$, $y \in \mathsf{acc}(\bar{\alpha}, \bar{r})$, so $\bar{x}$ is $\bar{r}$-accessible, i.e. $\bar{x} \in \mathsf{acc}(\bar{\alpha}, \bar{r})$. Thus, $[\![\Gamma \vdash \mathsf{intro}_\mathsf{acc}\ f\ x]\!]_\gamma = \bullet \in [\![\Gamma \vdash \mathsf{acc}_r\ x]\!]_\gamma = [\bar{x} \in \mathsf{acc}(\bar{\alpha}, \bar{r})]$.
- $\mathsf{rec}_\mathsf{acc}$: We have $\bar{C} : \bar{\alpha} \to U_n$, and
  $$\bar{e} \in \prod_{x \in \bar{\alpha}} [\forall y \in \bar{\alpha}, (y, x) \in \bar{r} \to y \in \mathsf{acc}(\bar{\alpha}, \bar{r})] \to \Big(\prod_{y \in \bar{\alpha}} [(y, x) \in \bar{r}] \to \bar{C}(y) \Big) \to \bar{C}(x).$$
  We want to show $[\![\mathsf{rec}_\mathsf{acc}\ e]\!]_\gamma \in \prod_{x \in \bar{\alpha}} [x \in \mathsf{acc}(\bar{\alpha}, \bar{r})] \to \bar{C}(x)$. It was defined as a function on this domain, so let $x \in \bar{\alpha}$ and $h \in [x \in \mathsf{acc}(\bar{\alpha}, \bar{r})]$; then $x \in \mathsf{acc}(\bar{\alpha}, \bar{r})$ and $h = \bullet$, and $[\![\mathsf{rec}_\mathsf{acc}\ e]\!]_\gamma(x)(h) = F(x)$ for the function F in the definition of $\mathsf{rec}_\mathsf{acc}$. We prove that $F(x) \in \bar{C}(x)$ by induction on $x \in \mathsf{acc}(\bar{\alpha}, \bar{r})$.
  Suppose that for all $y \in \mathsf{acc}(\bar{\alpha}, \bar{r})$, if $(y, x) \in \bar{r}$ then $F(y) \in \bar{C}(y)$. Then $F(x) = \bar{e}(x)(\bullet)(y \in \bar{\alpha} \mapsto (h \in [(y, x) \in \bar{r}] \mapsto F(y)))$. Clearly $x \in \bar{\alpha}$, and $\bullet \in [(y, x) \in \bar{r}]$. Also, $(y \in \bar{\alpha} \mapsto (h \in [(y, x) \in \bar{r}] \mapsto F(y))) \in \prod_{y \in \bar{\alpha}} [(y, x) \in \bar{r}] \to \bar{C}(y)$ because if $y \in \bar{\alpha}$ and $h \in [(y, x) \in \bar{r}]$, then $(y, x) \in \bar{r}$ so $F(y) \in \bar{C}(y)$ by the IH. Thus $F(x) \in \bar{C}(x)$.

- Quotients. Suppose $\Gamma \vdash \alpha : \mathsf{U}_n$ with $n \geq 1$, and $\Gamma \vdash r : \alpha \to \alpha \to \mathbb{P}$. Let $\sim$ be the equivalence closure of $\bar{\bar{r}}$. (We will assume this for the next few cases to do with quotients.) Then $[\![\Gamma \vdash \alpha/r]\!]_\gamma = \bar{\alpha}/{\sim} \in U_n$ because $\bar{\alpha}/{\sim}$ is contained in the double powerset of $\bar{\alpha} \in \mathsf{U}_n$.
- mk: Suppose additionally $\Gamma \vdash x : \alpha$, so $\bar{x} \in \bar{\alpha}$ from the IH. Then $[\![\Gamma \vdash \mathsf{mk}_r\ x]\!]_\gamma = [\bar{x}]_\sim \in \bar{\alpha}/{\sim} = [\![\Gamma \vdash \alpha/r]\!]_\gamma$.
- lift: Suppose $\Gamma \vdash \beta : \mathsf{U}_{n'}$ with $n' \geq 1$, and $\Gamma \vdash f : \alpha \to \beta$, and $\Gamma \vdash h : \forall x\ y : \alpha.\ R\ x\ y \to f\ x = f\ y$. From the IH, $\bar{f} : \bar{\alpha} \to \bar{\beta}$ and
$$\bar{h} \in \bigcap_{x \in \bar{\alpha}} \bigcap_{y \in \bar{\alpha}} [(x, y) \in \bar{\bar{r}} \to \bar{f}(x) = \bar{f}(y)].$$
Therefore $\forall x, y \in \bar{\alpha}.\ (x, y) \in \bar{\bar{r}} \to \bar{f}(x) = \bar{f}(y)$, so since the property $\bar{f}(x) = \bar{f}(y)$ is an equivalence relation that contains $\bar{\bar{r}}$, we have $x \sim y \to \bar{f}(x) = \bar{f}(y)$, so there is a well defined function $F : \bar{\alpha}/{\sim} \to \bar{\beta}$ such that $F([x]_\sim) = \bar{f}(x)$, and $[\![\Gamma \vdash \mathsf{lift}_r\ \beta\ f\ h]\!]_\gamma$ was defined to be this function. Thus $[\![\Gamma \vdash \mathsf{lift}_r\ \beta\ f\ h]\!]_\gamma \in [\![\Gamma \vdash \alpha/r \to \beta]\!]_\gamma$.
- sound: We want to verify that $[\![\Gamma \vdash \mathsf{sound}_r]\!]_\gamma = \bullet \in [\![\Gamma \vdash \forall x\ y : \alpha.\ r\ x\ y \to \mathsf{mk}_r\ x = \mathsf{mk}_r\ y]\!]_\gamma$, or after expansion,
$$\bullet \in \bigcap_{x \in \bar{\alpha}} \bigcap_{y \in \bar{\alpha}} [(x, y) \in \bar{\bar{r}} \to [x]_\sim = [y]_\sim].$$
Let $x, y \in \bar{\alpha}$, and suppose $(x, y) \in \bar{\bar{r}}$; then since $\sim$ contains $\bar{\bar{r}}$, $x \sim y$ and hence $[x]_\sim = [y]_\sim$.
- $\mathsf{propext}$: We want to verify that $[\![\vdash \mathsf{propext}]\!]_() = \bullet \in [\![\vdash \forall p\ q : \mathbb{P}.\ (p \leftrightarrow q) \to p = q]\!]_()$. Suppose $p, q \in U_0$. Then $p, q \subseteq \{\bullet\}$. If $\bullet \in p \leftrightarrow \bullet \in q$, then either $\bullet \in p$ and $\bullet \in q$, so $p = \{\bullet\} = q$, or $\bullet \notin p, q$, so $p = \emptyset = q$.
- choice: Let $\Gamma \vdash \alpha : \mathsf{U}_n$ and $\Gamma \vdash h : \|\alpha\|$. Then $\bar{h} \in [\bar{\alpha} \neq \emptyset]$, so $\bar{\alpha} \neq \emptyset$, and $\bar{\alpha} \in U_n \subseteq U_\omega$, so since ε is a choice function on U_ω, $[\![\Gamma \vdash \mathsf{choice}\ \alpha\ h]\!]_\gamma = \varepsilon(\bar{\alpha}) \in \bar{\alpha}$.

This completes the proof of parts 1-3; now we consider the equivalence rules, which only involves part 4.

- Reflexivity, symmetry and transitivity follow since $\bar{e} = \bar{e}'$ is an equivalence relation.
- Compatibility. This expresses the fact that each syntax constructor such as $[\![\Gamma \vdash \alpha + \beta]\!]_\gamma$ is defined only in terms of $[\![\Gamma \vdash \alpha]\!]_\gamma$ and $[\![\Gamma \vdash \beta]\!]_\gamma$. When a case split on $[\![\ell]\!] = 0$ is done, by unique typing it must be the same for both sides (since e and e' have the same type).
- Proof beta. Suppose $\Gamma, x : \alpha \vdash e : \beta$ and $\Gamma \vdash e' : \alpha$, so that by the inductive hypothesis $\bar{e}(x) \in \bar{\beta}(x)$ for all $x \in \bar{\alpha}$, and $\bar{e}' \in \bar{\alpha}$. Then $[\![\Gamma \vdash (\lambda x : \alpha.\ e)\ e']\!]_\gamma = \bullet = [\![\Gamma \vdash e[e'/x]]\!]_\gamma$ because $e[e'/x]$ is a proof (by part 2).
- Type beta. Suppose $\Gamma, x : \alpha \vdash e : \beta$ and $\Gamma \vdash e' : \alpha$, so that by the inductive hypothesis $\bar{e}(x) \in \bar{\beta}(x)$ for all $x \in \bar{\alpha}$, and $\bar{e}' \in \bar{\alpha}$. Then $[\![\Gamma \vdash (\Lambda x : \alpha.\ e) \cdot e']\!]_\gamma = (x \in \bar{\alpha} \mapsto \bar{e}(x))(\bar{e}') = \bar{e}(\bar{e}') = [\![\Gamma \vdash e[e'/x]]\!]_\gamma$ by the substitution lemma.
- Eta. Suppose $\Gamma \vdash e : \Pi y : \alpha.\ \beta$, so that by the inductive hypothesis $\bar{e} \in \prod_{y \in \bar{\alpha}} \bar{\beta}(y)$. Then $[\![\Gamma \vdash \Lambda x : \alpha.\ e \cdot x]\!]_\gamma = (x \in \bar{\alpha} \mapsto \bar{e}(x)) = \bar{e}$ by function extensionality in ZFC.
- Proof irrelevance. If $\Gamma \vdash h, h' : p : \mathbb{P}$, then by part 2 of the theorem, $[\![\Gamma \vdash h]\!]_\gamma = \bullet = [\![\Gamma \vdash h']\!]_\gamma$.
- Delta. If $\mathsf{def}\ c : \alpha := e$, then $[\![\Gamma \vdash c]\!]_\gamma = [\![\Gamma \vdash e]\!]_\gamma$ by definition.
- Zeta. If $\mathsf{def}\ c : \alpha := e$, then $[\![\Gamma \vdash \mathsf{let}\ x : \alpha := e_1\ \mathsf{in}\ e_2]\!]_\gamma = [\![\Gamma \vdash e_2[e_1/x]]\!]_\gamma$ by definition. (We don't use the substitution lemma here because it is not necessarily true that $\Gamma, x : \alpha \vdash e_2$ is well typed.)

- Quotient iota. $[\![\Gamma \vdash \mathsf{lift}_r\ \beta\ f\ h\ (\mathsf{mk}_r\ a)]\!]_\gamma = [\![\Gamma \vdash \mathsf{lift}_r\ \beta\ f\ h]\!]_\gamma([\bar{a}]_\sim) = \bar{f}(\bar{a})$ by definition (we showed it is well defined given the assumptions on α, r, β, f, h already).
- π_1 iota. $[\![\Gamma \vdash \pi_1(a,b)]\!]_\gamma = \pi_1(\bar{a},\bar{b}) = \bar{a}$.
- π_2 iota. $[\![\Gamma \vdash \pi_2(a,b)]\!]_\gamma = \pi_2(\bar{a},\bar{b}) = \bar{b}$.
- inl iota. $[\![\Gamma \vdash \mathsf{rec}_+\ a\ b\ (\mathsf{inl}\ x)]\!]_\gamma = [\![\Gamma \vdash \mathsf{rec}_+\ a\ b]\!]_\gamma(\iota_1(\bar{x})) = \bar{a}(\bar{x}) = [\![\Gamma \vdash a\ x]\!]_\gamma$.
- inr iota. $[\![\Gamma \vdash \mathsf{rec}_+\ a\ b\ (\mathsf{inr}\ x)]\!]_\gamma = [\![\Gamma \vdash \mathsf{rec}_+\ a\ b]\!]_\gamma(\iota_2(\bar{x})) = \bar{b}(\bar{x}) = [\![\Gamma \vdash b\ x]\!]_\gamma$.
- ulift iota. $[\![\Gamma \vdash \downarrow\uparrow x]\!]_\gamma = [\![\Gamma \vdash x]\!]_\gamma$ by definition.
- W iota. Letting $F := \mathsf{rec}_\mathsf{W}(\mathsf{W}_{x\in\bar{a}}\bar{\beta}(x), \bar{e})$, we have $[\![\Gamma \vdash \mathsf{rec}_\mathsf{W}\ e\ (\mathsf{sup}\ a\ f)]\!]_\gamma = F(\bar{a},\bar{f}) = \bar{e}(\bar{a})(\bar{f})(F\circ\bar{f})$ on the one hand, and $[\![\Gamma \vdash e\ a\ f\ (\lambda b : \beta[a/x].\ \mathsf{rec}_\mathsf{W}\ e\ (f\ b))]\!]_\gamma = \bar{e}(\bar{a})(\bar{f})(b \in \bar{\beta}(\bar{a}) \mapsto F(f(b)))$ on the other; and $F\circ\bar{f} = (b \in \bar{\beta}(\bar{a}) \mapsto F(f(b)))$ because $\bar{\beta}(\bar{a})$ is the domain of f.
- $=$ iota. $[\![\Gamma \vdash \mathsf{rec}_=\ e\ a\ h]\!]_\gamma = \bar{e}$ by definition.
- acc iota. If $F : \mathsf{acc}(\bar{\alpha}, \overline{\bar{r}}) \to V$ is the function defined in $\mathsf{rec}_\mathsf{acc}(\bar{\alpha}, \overline{\bar{r}}, \bar{e})$, then we have

$$\begin{aligned}[\![\Gamma \vdash \mathsf{rec}_\mathsf{acc}\ e\ x\ (\mathsf{intro}_\mathsf{acc}\ x\ f)]\!]_\gamma &= \mathsf{rec}_\mathsf{acc}(\bar{\alpha}, \overline{\bar{r}}, \bar{e})(x)(\bullet) = F(x)\\ &= e(\bar{x})(\bullet)(y \in \bar{\alpha} \mapsto (h \in [(y,x) \in \overline{\bar{r}}] \mapsto F(y))\\ &= e(\bar{x})(\bar{f})(y \in \bar{\alpha} \mapsto (h \in [(y,x) \in \overline{\bar{r}}] \mapsto F(y))\\ &= [\![\Gamma \vdash e\ x\ f\ (\lambda(y:\alpha)\ (h : r\ y\ x).\ \mathsf{rec}_\mathsf{acc}\ e\ y\ (f\ y\ h))]\!]_\gamma\end{aligned}$$

  where $\bar{f} = \bullet$ because f is a proof.
- ulift eta. $[\![\Gamma \vdash \uparrow\downarrow x]\!]_\gamma = [\![\Gamma \vdash x]\!]_\gamma$ by definition.
- Σ eta. If $\Gamma \vdash p : \Sigma x : \alpha.\ \beta$, then $\bar{p} \in \sum_{x\in\bar{\alpha}} \bar{\beta}(x)$, so $\bar{p} = (x,y)$ is a pair, so $[\![\Gamma \vdash (\pi_1\ p, \pi_2\ p)]\!]_\gamma = (\pi_1(\bar{p}), \pi_2(\bar{p})) = (x,y) = \bar{p}$.

□

Corollary 6.8. *Lean is consistent if* ZFC + {*there are n inaccessible cardinals* | $n \in \omega$} *is. That is, there is no proof of $\bot$ that is verified by the Lean kernel.*

Proof. Suppose $\Vdash e : \bot$ (the algorithmic typing judgment). Then $\vdash e : \bot$ since algorithmic equality implies definitional equality. Let v be the universe valuation that sets every variable to 0, so $\vdash \langle e\rangle_{v,\cdot} : \bot$ and let $(\kappa_i)_{i\in\omega}$ be a cardinal sequence which is n-correct with n sufficiently large to satisfy the assumption of theorem 6.7. Then $[\![\vdash \langle e\rangle]\!]_() \in [\![\vdash \bot]\!]_() = \emptyset$, a contradiction. □

6.4 Type injectivity

The semantics we have given for types collapses many types into the same ZFC set, somewhat by accident in the sense that ZFC does not track the complete construction of a type, so that types that the type theory thinks are not definitionally equal become equal in the ZFC interpretation. To resolve this, we will define a special set of "tagged" types, which will respect equality of types but are otherwise freely generated. This will allow us to have a ZFC analogue of unique typing.

We will define a sequence of sets $T_n \subseteq U_n$ and simultaneously define a function $(\!|t|\!) \in U_n$ for $t \in T_n$, inductively generated by the clauses below. At $n = 0$, we have $T_0 = U_0$ and $(\!|p|\!) = p$ for $p \in T_0$. For $n > 0$:

- $(\mathsf{U}, n-1) \in T_n$, and $(\!|(\mathsf{U}, n-1)|\!) = T_{n-1}$ (where U is a literal character encoded in ZFC).

- If $A \in T_n$, $B : (\!|A|\!) \to T_n$, and $B \in U_n$, $\prod_{x \in (\!|A|\!)} B(x) \in U_n$, then $(\Pi, A, B) \in T_n$ and $(\!|(\Pi, A, B)|\!) = \prod_{x \in (\!|A|\!)} (\!|B(x)|\!)$.
- If $A \in T_n$, $B : (\!|A|\!) \to T_n$, and $B \in U_n$, $\sum_{x \in (\!|A|\!)} B(x) \in U_n$, then $(\Sigma, A, B) \in T_n$ and $(\!|(\Sigma, A, B)|\!) = \sum_{x \in (\!|A|\!)} (\!|B(x)|\!)$.
- If $A \in T_n$, $B : (\!|A|\!) \to T_n$, and $B \in U_n$, $\mathsf{W}_{x \in (\!|A|\!)} B(x) \in U_n$, then $(\mathsf{W}, A, B) \in T_n$ and $(\!|(\mathsf{W}, A, B)|\!) = \mathsf{W}_{x \in (\!|A|\!)} (\!|B(x)|\!)$.
- If $A, B \in T_n$, then $(+, A, B) \in T_n$ and $(\!|(+, A, B)|\!) = (\!|A|\!) \sqcup (\!|B|\!)$.
- If $A \in T_m$ and $m \le n$, then $(\mathsf{ulift}, m, A) \in T_n$ and $(\!|(\mathsf{ulift}, m, A)|\!) = (\!|A|\!)$.

It is an easy induction to show that $T_n \subseteq U_n$ and $(\!|t|\!) \in U_n$ if $t \in U_n$.

Now we change the interpretation of types to elements of T_n, and use $x \in (\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!)$ in place of $x \in [\![\Gamma \vdash \alpha]\!]_\gamma$ to get the ZFC-elements of a type.

- $[\![\Gamma, x : \alpha]\!] = \sum_{\gamma \in [\![\Gamma]\!]} (\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!)$
- $[\![\Gamma \vdash \mathsf{U}_n]\!]_\gamma = (\mathsf{U}, n)$
- $[\![\Gamma \vdash \Pi x : \alpha.\ \beta]\!]_\gamma = (\Pi, [\![\Gamma \vdash \alpha]\!]_\gamma, (x \in (\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!) \mapsto [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma, x)}))$
- $[\![\Gamma \vdash \Sigma x : \alpha.\ \beta]\!]_\gamma = (\Sigma, [\![\Gamma \vdash \alpha]\!]_\gamma, (x \in (\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!) \mapsto [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma, x)}))$
- $[\![\Gamma \vdash \mathsf{W} x : \alpha.\ \beta]\!]_\gamma = (\mathsf{W}, [\![\Gamma \vdash \alpha]\!]_\gamma, (x \in (\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!) \mapsto [\![\Gamma, x : \alpha \vdash \beta]\!]_{(\gamma, x)}))$
- $[\![\Gamma \vdash \mathsf{ulift}_m^n \alpha]\!]_\gamma = (\mathsf{ulift}, m, [\![\Gamma \vdash \alpha]\!]_\gamma)$
- Other cases are the same as before, with $x \in (\!|t|\!)$ in place of $x \in t$ when getting the elements of a type.

Now the main part of the soundness theorem states:

Theorem 6.9 (Soundness)**.**

- *If $\Gamma \vdash e : \alpha$, then there exists an k such that if the κ sequence is k-correct, then for all $\gamma \in [\![\Gamma]\!]$, $[\![\Gamma \vdash e]\!]_\gamma \in (\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!)$.*
- *If $\Gamma \vdash e \equiv e'$, then there exists an k such that if the κ sequence is k-correct, then for all $\gamma \in [\![\Gamma]\!]$, $[\![\Gamma \vdash e]\!]_\gamma = [\![\Gamma \vdash e']\!]_\gamma$.*

Proof. The proof is virtually unchanged from theorem 6.7, since $(\!|[\![\Gamma \vdash \alpha]\!]_\gamma|\!)$ has the same meaning as $[\![\Gamma \vdash \alpha]\!]_\gamma$ in the original proof – none of the tags affect any of the reasoning. □

We can recover some of unique typing as a consequence of this theorem, but not all of it. So for example, if $\Gamma \vdash \mathsf{U}_n \equiv \Pi x : \alpha.\ \beta$, and Γ is an inhabited context, say $\gamma \in [\![\Gamma]\!]$, then $(\mathsf{U}, n) = [\![\Gamma \vdash \mathsf{U}_n]\!]_\gamma = [\![\Gamma \vdash \Pi x : \alpha.\ \beta]\!]_\gamma = (\Pi, \dots)$ which implies $\mathsf{U} = \Pi$, which is false (here U and Π are distinct elements of a small alphabet). So $\Gamma \vdash \mathsf{U}_n \not\equiv \Pi x : \alpha.\ \beta$. Compare this with the definitional inversion property Definition 1, proven in theorem 4.12, which does not require that Γ be inhabited. We also get weakened versions of the U-U and Π-Π clauses, where we learn that the arguments are only equal in the model, rather than definitionally equal.

References

[1] Bruno Barras. Sets in Coq, Coq in Sets. *Journal of Formalized Reasoning*, 3(1):29–48, 2010.

[2] Bruno Barras and Benjamin Grégoire. On the Role of Type Decorations in the Calculus of Inductive Constructions. In *International Workshop on Computer Science Logic*, pages 151–166. Springer, 2005.

[3] Bruno Barras and Benjamin Werner. Coq in Coq. *Available on the WWW*, 1997.

[4] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer Science & Business Media, 2013.

[5] Alonzo Church. A Formulation of the Simple Theory of Types. *The journal of symbolic logic*, 5(2):56–68, 1940.

[6] Thierry Coquand and Gérard Huet. *The Calculus of Constructions*. PhD thesis, INRIA, 1986.

[7] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The Lean Theorem Prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[8] Peter Dybjer. Inductive Families. *Formal aspects of computing*, 6(4):440–465, 1994.

[9] Maxime Dénès. Propositional extensionality is inconsistent in Coq, Dec 2013.

[10] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought. *From Frege to Gödel: A source book in mathematical logic*, 1931:1–82, 1879.

[11] William A Howard. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980.

[12] Gyesik Lee and Benjamin Werner. Proof-irrelevant model of cc with predicative induction and judgmental equality. *arXiv preprint arXiv:1111.0123*, 2011.

[13] Zhaohui Luo. Ecc, an extended calculus of constructions. In *Logic in Computer Science, 1989. LICS'89, Proceedings., Fourth Annual Symposium on*, pages 386–395. IEEE, 1989.

[14] Per Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 73–118. Elsevier, 1975.

[15] Per Martin-Löf. Constructive Mathematics and Computer Programming. In *Studies in Logic and the Foundations of Mathematics*, volume 104, pages 153–175. Elsevier, 1982.

[16] Simone Martini. Several types of types in programming languages. In *International Conference on History and Philosophy of Computing*, pages 216–227. Springer, 2015.

[17] Alexandre Miquel and Benjamin Werner. The not so simple proof-irrelevant model of cc. In *International Workshop on Types for Proofs and Programs*, pages 240–258. Springer, 2002.

[18] Ulf Norell. Dependently typed programming in Agda. In *International School on Advanced Functional Programming*, pages 230–266. Springer, 2008.

[19] Christine Paulin-Mohring. Introduction to the Calculus of Inductive Constructions, 2015.

[20] Robert Pollack. Polishing up the tait-martin-löf proof of the church-rosser theorem. 1995.

[21] Willard V Quine. New Foundations for Mathematical Logic. *The American mathematical monthly*, 44(2):70–80, 1937.

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

[23] Benjamin Werner. Sets in types, types in sets. In *International Symposium on Theoretical Aspects of Computer Software*, pages 530–546. Springer, 1997.

[24] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume 2. University Press, 1912.