

The Type Theory of Lean

Mario Carneiro

March 17, 2018

Contents

1	Introduction	2
2	The axioms	2
2.1	Typing	2
2.2	Definitional equality	3
2.3	Reduction	4
2.4	let binders (ζ reduction)	4
2.5	Definitions (δ reduction)	5
2.6	Inductive types	5
2.6.1	Inductive specifications	5
2.6.2	Large elimination	7
2.6.3	The recursor	8
2.6.4	The computation rule (ι reduction)	8
2.7	Non-primitive axioms	9
2.7.1	Quotient types	9
2.7.2	Propositional extensionality	9
2.7.3	Axiom of choice	10
3	Properties of the type system	10
3.1	Undecidability of definitional equality	10
3.1.1	Algorithmic equality is not transitive	11
3.1.2	Failure of subject reduction	12
3.2	Regularity	12
4	Unique typing	14
4.1	The κ reduction	15
4.2	The Church-Rosser theorem	18
5	Reduction of inductive types to W-types	22

5.1	The menagerie	22
5.2	Translating type families	25
5.3	Translating subsingleton eliminators	26
5.4	The remainder	28
6	Soundness	28
6.1	Modeling Lean in ZFC	28
6.1.1	Definition of W-types in ZFC	30
6.1.2	Definition of acc in ZFC	31
6.2	Soundness	31
7	Strong normalization	36
7.1	Evaluation in the empty context	36

1 Introduction

In this paper, we develop the foundational theory used in the Lean theorem prover.

2 The axioms

2.1 Typing

The syntax of expressions is given by the following grammar:

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

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

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

$$\begin{array}{c}
\frac{\Gamma \vdash \alpha : \mathbf{U}_\ell \quad \Gamma \vdash e : \beta}{\Gamma, x : \alpha \vdash e : \beta} \quad \frac{\Gamma \vdash \alpha : \mathbf{U}_\ell}{\Gamma, x : \alpha \vdash x : \alpha} \quad \frac{}{\vdash \mathbf{U}_\ell : \mathbf{U}_{S\ell}} \\
\frac{\Gamma \vdash e_1 : \forall x : \alpha. \beta \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1 \, e_2 : \beta[e_2/x]} \quad \frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash \lambda x : \alpha. e : \forall x : \alpha. \beta} \\
\frac{\Gamma \vdash \alpha : \mathbf{U}_{\ell_1} \quad \Gamma, x : \alpha \vdash \beta : \mathbf{U}_{\ell_2}}{\Gamma \vdash \forall x : \alpha. \beta : \mathbf{U}_{\text{imax}(\ell_1, \ell_2)}} \quad \frac{\Gamma \vdash e : \alpha \quad \Gamma \vdash \alpha \equiv \beta}{\Gamma \vdash e : \beta}
\end{array}$$

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:

$$\boxed{\Gamma \vdash \alpha \text{ type}} \quad \frac{\Gamma \vdash \alpha : \mathbf{U}_\ell}{\Gamma \vdash \alpha \text{ type}} \quad \boxed{\vdash \Gamma \text{ ok}} \quad \frac{}{\vdash \cdot \text{ ok}} \quad \frac{\Gamma \vdash \alpha \text{ type}}{\vdash \Gamma, x : \alpha \text{ ok}}$$

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.

$$\begin{array}{c}
\boxed{\Gamma \vdash e \equiv e'} \\
\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash e \equiv e} \quad \frac{\Gamma \vdash e \equiv e'}{\Gamma \vdash e' \equiv e} \quad \frac{\Gamma \vdash e_1 \equiv e_2 \quad \Gamma \vdash e_2 \equiv e_3}{\Gamma \vdash e_1 \equiv e_3} \\
\frac{\ell \equiv \ell'}{\vdash U_\ell \equiv U_{\ell'}} \quad \frac{\Gamma \vdash e_1 \equiv e'_1 : \forall x : \alpha. \beta \quad \Gamma \vdash e_2 \equiv e'_2 : \alpha}{\Gamma \vdash e_1 e_2 \equiv e'_1 e'_2} \\
\frac{\Gamma \vdash \alpha \equiv \alpha' \quad \Gamma, x : \alpha \vdash e \equiv e'}{\Gamma \vdash \lambda x : \alpha. e \equiv \lambda x : \alpha'. e'} \quad \frac{\Gamma \vdash \alpha \equiv \alpha' \quad \Gamma, x : \alpha \vdash \beta \equiv \beta'}{\Gamma \vdash \forall x : \alpha. \beta \equiv \forall x : \alpha'. \beta'} \\
(\beta) \frac{\Gamma, x : \alpha \vdash e : \beta \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash (\lambda x : \alpha. e) e' \equiv e[e'/x]} \quad (\eta) \frac{\Gamma \vdash e : \forall y : \alpha. \beta}{\Gamma \vdash \lambda x : \alpha. e x \equiv e} \\
\frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p}{\Gamma \vdash h \equiv h'}
\end{array}$$

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 $\mathbb{P} := \mathbf{U}_0$) 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 $n = 0$):

$$\begin{array}{c}
\boxed{\ell \equiv \ell'} \quad \frac{\ell \leq \ell' \quad \ell' \leq \ell}{\ell \equiv \ell'} \\
\boxed{\ell \leq \ell' + n} \\
\frac{n \geq 0}{0 \leq \ell + n} \quad \frac{n \geq 0}{\ell \leq \ell + n} \\
\frac{\ell \leq \ell' + (n-1)}{S\ell \leq \ell' + n} \quad \frac{\ell \leq \ell' + (n+1)}{\ell \leq S\ell' + n} \\
\frac{\ell \leq \ell_1 + n}{\ell \leq \max(\ell_1, \ell_2) + n} \quad \frac{\ell \leq \ell_2 + n}{\ell \leq \max(\ell_1, \ell_2) + n} \quad \frac{\ell_1 \leq \ell + n \quad \ell_2 \leq \ell + n}{\max(\ell_1, \ell_2) \leq \ell + n} \\
\frac{0 \leq \ell + n}{\text{imax}(\ell_1, 0) \leq \ell + n} \quad \frac{\max(\ell_1, S\ell_2) \leq \ell + n}{\text{imax}(\ell_1, S\ell_2) \leq \ell + n} \\
\frac{\max(\text{imax}(\ell_1, \ell_3), \text{imax}(\ell_2, \ell_3)) \leq \ell + n}{\text{imax}(\ell_1, \text{imax}(\ell_2, \ell_3)) \leq \ell + n} \quad \frac{\ell \leq \max(\text{imax}(\ell_1, \ell_3), \text{imax}(\ell_2, \ell_3)) + n}{\ell \leq \text{imax}(\ell_1, \text{imax}(\ell_2, \ell_3)) + n} \\
\frac{\max(\text{imax}(\ell_1, \ell_2), \text{imax}(\ell_1, \ell_3)) \leq \ell + n}{\text{imax}(\ell_1, \max(\ell_2, \ell_3)) \leq \ell + n} \quad \frac{\ell \leq \max(\text{imax}(\ell_1, \ell_2), \text{imax}(\ell_1, \ell_3)) + n}{\ell \leq \text{imax}(\ell_1, \max(\ell_2, \ell_3)) + n} \\
\frac{\ell[0/u] \leq \ell'[0/u] + n \quad \ell[Su/u] \leq \ell'[Su/u] + n}{\ell \leq \ell' + n}
\end{array}$$

Although this definition looks complicated, it is most easily understood in terms of its semantics: A level takes values in \mathbb{N} , where $\llbracket 0 \rrbracket = 0$, $\llbracket S\ell \rrbracket = \llbracket \ell \rrbracket + 1$, $\llbracket \max(\ell_1, \ell_2) \rrbracket = \max(\llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)$ and $\llbracket \text{imax}(\ell_1, \ell_2) \rrbracket = \text{imax}(\llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)$, where $\text{imax}(m, n)$ is the function such that $\text{imax}(m, n+1) =$

$\max(m, n + 1)$ and $\text{imax}(m, 0) = 0$. Then a level inequality $\ell \leq \ell' + n$ holds if for all substitutions v of numerals for the variables in ℓ and ℓ' , $\llbracket \ell \rrbracket_v \leq \llbracket \ell' \rrbracket_v + n$. 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:

$$\begin{array}{c}
\boxed{\Gamma \vdash e \Leftrightarrow e'} \\
\frac{}{\Gamma \vdash e \Leftrightarrow e} \quad \frac{\Gamma \vdash e \Leftrightarrow e'}{\Gamma \vdash e' \Leftrightarrow e} \quad \frac{\ell \equiv \ell'}{\Gamma \vdash \mathbf{U}_\ell \Leftrightarrow \mathbf{U}'_\ell} \\
\frac{\Gamma \vdash \alpha \Leftrightarrow \alpha' \quad \Gamma, x : \alpha \vdash e \Leftrightarrow e'}{\Gamma \vdash \lambda x : \alpha. e \Leftrightarrow \lambda x : \alpha'. e'} \quad \frac{\Gamma \vdash \alpha \Leftrightarrow \alpha' \quad \Gamma, x : \alpha \vdash e \Leftrightarrow e'}{\Gamma \vdash \forall x : \alpha. e \Leftrightarrow \forall x : \alpha'. e'} \\
\frac{\Gamma \vdash e : \forall x : \alpha. \beta \quad \Gamma, x : \alpha \vdash e x \Leftrightarrow e' x}{\Gamma \vdash e \Leftrightarrow e'} \quad \frac{\Gamma \vdash p : \mathbb{P} \quad \Gamma \vdash h : p \quad \Gamma \vdash h' : p' \quad \Gamma \vdash p \Leftrightarrow p'}{\Gamma \vdash h \Leftrightarrow h'} \\
\frac{\Gamma \vdash e_1 \Leftrightarrow e'_1 \quad \Gamma \vdash e_2 \Leftrightarrow e'_2}{\Gamma \vdash e_1 e_2 \Leftrightarrow e'_1 e'_2} \quad \frac{e \rightsquigarrow k \quad \Gamma \vdash k \Leftrightarrow e'}{\Gamma \vdash e \Leftrightarrow e'}
\end{array}$$

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.

$$\boxed{e \rightsquigarrow e'} \quad \frac{e_1 \rightsquigarrow e'_1}{e_1 e_2 \rightsquigarrow e'_1 e_2} \quad \frac{}{(\lambda x : \alpha. e) e' \rightsquigarrow e[e'/x]}$$

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 $\text{let } x : \alpha := e' \text{ in } e$ 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.

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

$$\begin{array}{l}
e ::= \dots \mid \text{let } x : \alpha := e' \text{ in } e \\
\dots \quad \frac{\Gamma \vdash e' : \alpha \quad \Gamma \vdash e[e'/x] : \beta}{\Gamma \vdash \text{let } x : \alpha := e' \text{ in } e : \beta} \\
\dots \quad (\zeta) \quad \frac{\Gamma \vdash e' : \alpha \quad \Gamma \vdash e[e'/x] : \beta}{\Gamma \vdash \text{let } x : \alpha := e' \text{ in } e \equiv e[e'/x]}
\end{array}$$

$$\cdots \quad \overline{\text{let } x : \alpha := e' \text{ in } e \rightsquigarrow e[e'/x]}$$

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 `let $x : \alpha := e'$ 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.

For a constant definition `constant $c_{\bar{u}} : \alpha$` to be admissible, we require that $\vdash \alpha : \mathbf{U}_{\ell}$, where the universe variables in α and ℓ are contained in \bar{u} . Similarly, a definition is specified by a clause `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:

$$\begin{array}{c} e ::= \cdots \mid c_{\bar{u}} \\[10pt] \frac{}{\vdash c_{\bar{\ell}} : \tau_{\bar{\ell}}(c)} \quad \frac{\ell_1 \equiv \ell'_1 \ \dots \ \ell_n \equiv \ell'_n}{\vdash c_{\bar{\ell}} \equiv c_{\bar{\ell}'}} \quad \frac{\ell_1 \equiv \ell'_1 \ \dots \ \ell_n \equiv \ell'_n}{\Gamma \vdash c_{\bar{\ell}} \Leftrightarrow c_{\bar{\ell}'}} \end{array}$$

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

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

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 $f e_1 \dots e_n$.

A specification K is typechecked in a context of a variable $t : F$ where $F = \forall x :: \alpha. U_\ell$ 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:

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

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

$$\boxed{\Gamma; t : F \vdash \alpha \text{ ctor}} \quad \frac{\Gamma \vdash e :: \alpha}{\Gamma; t : \forall x :: \alpha. U_\ell \vdash t e \text{ ctor}}$$

$$\frac{\Gamma \vdash \beta : U_{\ell'} \quad \ell' \leq \ell \quad \Gamma, y : \beta; t : \forall x :: \alpha. U_\ell \vdash \tau \text{ ctor}}{\Gamma; t : \forall x :: \alpha. U_\ell \vdash \forall y : \beta. \tau \text{ ctor}}$$

$$\frac{\Gamma \vdash \gamma :: U_{\ell'} \quad \Gamma, z :: \gamma \vdash e :: \alpha \quad \text{imax}(\ell', \ell) \leq \ell \quad \Gamma; t : \forall x :: \alpha. U_\ell \vdash \tau \text{ ctor}}{\Gamma; t : \forall x :: \alpha. U_\ell \vdash (\forall z :: \gamma. t e) \rightarrow \tau \text{ ctor}}$$

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:

$$e ::= \dots \mid \mu x : e. K \mid c_{\mu x : e. K} \mid \text{rec}_{\mu x : e. K}$$

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

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 $\mathbb{N} := \mu N : U_1. (z : N) + (s : N \rightarrow N)$, yielding constructors $z_{\mathbb{N}} : \mathbb{N}$ (zero) and $s_{\mathbb{N}} : \mathbb{N} \rightarrow \mathbb{N}$ (successor). The eliminator for \mathbb{N} looks like this:

$$\text{rec}_{\mathbb{N}} : \forall (C : \mathbb{N} \rightarrow U_u). C z_{\mathbb{N}} \rightarrow (\forall x : \mathbb{N}. C x \rightarrow C (s_{\mathbb{N}} x)) \rightarrow \forall n : \mathbb{N}. C n$$

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 $\forall x : \mathbb{N}. C x \rightarrow C (s_{\mathbb{N}} x)$, 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.

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 \mathbf{U}_u .

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

1. The type family $t : \forall x :: \alpha. \mathbf{U}_\ell$ lives in a universe $1 \leq \ell$. (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:

$$\begin{array}{c}
\boxed{\Gamma; t : F \vdash K \text{ LE}} \\
\frac{1 \leq \ell}{\Gamma; t : \forall x :: \alpha. \mathbf{U}_\ell \vdash K \text{ LE}} \quad \frac{}{\Gamma; t : F \vdash 0 \text{ LE}} \quad \frac{\Gamma; t : F \vdash \alpha \text{ LE ctor}}{\Gamma; t : F \vdash (c : \alpha) \text{ LE}} \\
\boxed{\Gamma; t : F \vdash \alpha \text{ LE ctor}} \\
\frac{}{\Gamma; t : F \vdash t e \text{ LE ctor}} \quad \frac{\Gamma, t : F \vdash \alpha : \mathbb{P} \quad \Gamma, x : \alpha; t : F \vdash \beta \text{ LE ctor}}{\Gamma; t : F \vdash \forall x : \alpha. \beta \text{ LE ctor}} \\
\frac{\Gamma; t : F \vdash \beta \text{ LE ctor}}{\Gamma; t : F \vdash (\forall z :: \gamma. t e) \rightarrow \beta \text{ LE ctor}} \\
\frac{y \in e \quad \Gamma, y : \beta; t : \forall x :: \alpha. \mathbf{U}_\ell \vdash \forall z :: \gamma. t e \text{ LE ctor}}{\Gamma; t : \forall x :: \alpha. \mathbf{U}_\ell \vdash \forall y : \beta. \forall z :: \gamma. t e \text{ LE ctor}}
\end{array}$$

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:

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

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:

$$\begin{array}{c}
\alpha : \mathbf{U}_\ell, r : \alpha \rightarrow \alpha \rightarrow \mathbb{P} \vdash \text{acc}_r := \mu A : \alpha \rightarrow \mathbb{P}. \\
(\text{intro} : \forall x : \alpha. (\forall y : \alpha. r y x \rightarrow A y) \rightarrow A x)
\end{array}$$

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 $\mu t : F. K$.

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

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:

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

where:

- $\kappa = \forall a :: \alpha. C\ a \rightarrow \mathbf{U}_u$ where u is a fresh universe variable if $\Gamma; t : F \vdash K\ \text{LE}$, otherwise $\kappa = \forall a :: \alpha. C\ a \rightarrow \mathbb{P}$,
- 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:

$$\begin{aligned} \text{rec}_{\mathbb{N}}\ C\ a\ f\ z_{\mathbb{N}} &\equiv a \\ \text{rec}_{\mathbb{N}}\ C\ a\ f\ (s_{\mathbb{N}}\ n) &\equiv f\ n\ (\text{rec}_{\mathbb{N}}\ C\ a\ f\ n) \end{aligned}$$

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

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

where $v :: \delta$ is defined as $v_i = \lambda x :: \xi_i. \text{rec}_P\ C\ e\ \pi_i[b, x]\ (u_i\ x)$. (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:

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

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.

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

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:

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

Here $\text{rec}_{a=} C : a = b \rightarrow C a \rightarrow C b$ 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 $\alpha : \mathbb{U}_u$ and a relation $R : \alpha \rightarrow \alpha \rightarrow \mathbb{P}$, the quotient α/R represents the largest type with a surjection $\text{mk}_R : \alpha \rightarrow \alpha/R$ 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):

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

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 $\alpha/R := \alpha$, $\text{mk}_R a := a$, $\text{lift}_R f h := f$. As a terminological note, the rule $\text{lift}_R f h (\text{mk}_R a) \rightsquigarrow f a$ 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:

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

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 `nonempty` for this:

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

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

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](#):

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

(We are fixing a type α and a relation $< : \alpha \rightarrow \alpha \rightarrow \mathbb{P}$ 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 `acc x`:

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

Note that the output type of `invx` is the same as the argument to `intro x`. Thus, we have

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

by proof irrelevance.

Why does this matter? Normally, any proof of $\text{acc } x$ 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 $P : \mathbb{N} \rightarrow \mathbf{2}$ is a decidable predicate, such as $P \ n := \text{"Turing machine } M \text{ runs for at least } n \text{ steps without halting"}$, 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 $f : \forall n. \text{acc}_{>} n \rightarrow \mathbf{1}$ as follows:

$$f := \text{rec}_{\text{acc}} (\lambda _ . \mathbf{1}) (\lambda n _ . (g : \forall y. y > x \rightarrow \mathbf{1}). \\ \text{if } P \ n \text{ then } g \ (n + 1) \ (p \ n) \text{ else } ())$$

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

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

and the if statement evaluates to the left or right branch depending on whether $P \ n \rightsquigarrow^* \text{tt}$ or $P \ n \rightsquigarrow^* \text{ff}$. 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 $a : \text{acc}_{>} 0$:

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

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 $a : \text{acc}_{>} 0 \vdash f \ 0 \ a \equiv ()$ 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:

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

We can think of the middle step $f \ 0 \ (\text{intro}_{\text{acc}} \ 0 \ (\text{inv}_0 \ a))$ as a “creative” step, where we pick one of the many possible terms of type $\text{acc}_{>} 0$ 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 $a : \text{acc}_{<} 1$ 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 $p : \mathbb{P}$, $R : p \rightarrow p \rightarrow \mathbb{P}$, $\alpha : \mathbb{U}_1$, $f : p \rightarrow \alpha$, $H : \forall x y. r \ x \ y \rightarrow f \ x = f \ y$, $q : p/R$ and $h : p$, then:

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

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:

- $\Gamma \Vdash \text{id}_\beta \ e : \beta$ because the application forces checking $\Gamma \vdash \beta \Leftrightarrow \gamma$.
- $\Gamma \Vdash \text{id}_\alpha \ (\text{id}_\beta \ e) : \alpha$ since the application forces checking $\Gamma \vdash \alpha \Leftrightarrow \beta$.
- But $\Gamma \not\Vdash \text{id}_\alpha \ e : \alpha$ because this requires $\Gamma \vdash \alpha \Leftrightarrow \gamma$ which is false.

Since we obviously have $\text{id}_\beta \ e \rightsquigarrow e$ 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 $\vdash \Gamma \text{ ok}$.
- (2) If $\Gamma \vdash e : \alpha$, then $FV(e) \cup FV(\alpha) \subseteq \Gamma$.
- (3) If $\Gamma \vdash \alpha \text{ type}$, then $\Gamma \vdash \alpha : \mathbb{U}_\ell$ for some ℓ .
- (4) If $\Gamma \vdash e : \alpha$, then $\Gamma \vdash \alpha \text{ type}$.
- (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 $\Gamma; t : F \vdash K \text{ spec}$, then $\Gamma \vdash F \text{ type}$ (and more precisely, $F = \forall x :: \alpha. \mathbb{U}_\ell$ for some α, ℓ).
- (7) If $\Gamma; t : F \vdash K \text{ spec}$ and $(c : \alpha) \in K$, then $\Gamma; t : F \vdash \alpha \text{ ctor}$.
- (8) If $\Gamma; t : F \vdash \alpha \text{ ctor}$, then $\Gamma, t : F \vdash \alpha \text{ type}$.

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

Lemma 3.2 (Weakening).

- (1) If $\Gamma \vdash e : \alpha$ and $\vdash \Gamma, \Delta \text{ ok}$, then $\Gamma, \Delta \vdash e : \alpha$.
- (2) If $\Gamma \vdash e \equiv e'$ and $\vdash \Gamma, \Delta \text{ ok}$, 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

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

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' . \square

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 $\Gamma \vdash e_1[e_2/x] \equiv e'_1[e_2/x]$.

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

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. \square

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 $m \leq n$, $\Gamma \vdash_n e : \alpha$ 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 $m \leq n$.

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 $m \leq n$ then $\Gamma \vdash_m e : \alpha$ implies $\Gamma \vdash_n e : \alpha$.*
- (2) *If $m \leq n$ 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. \square

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

1. *If $\Gamma \vdash_n U_\ell \equiv U_{\ell'}$, 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. *$\Gamma \vdash_n U_\ell \not\equiv \forall x : \alpha. \beta$.*

(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 forall 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 $\Gamma \vdash_n \forall x : \alpha. \beta : \mathbf{U}_{\text{imax}(\ell_1, \ell_2)}, \mathbf{U}_{\text{imax}(\ell'_1, \ell'_2)}$ from $\Gamma \vdash \alpha : \mathbf{U}_{\ell_1}, \mathbf{U}_{\ell'_1}$ and $\Gamma \vdash \beta : \mathbf{U}_{\ell_2}, \mathbf{U}_{\ell'_2}$, and from the inductive hypothesis $\Gamma \vdash_n \mathbf{U}_{\ell_1} \equiv \mathbf{U}_{\ell'_1}$. From definitional inversion, $\ell_1 \equiv \ell'_1$ and $\ell_2 \equiv \ell'_2$, so $\Gamma \vdash_n \mathbf{U}_{\text{imax}(\ell_1, \ell_2)} \equiv \mathbf{U}_{\text{imax}(\ell'_1, \ell'_2)}$.
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 $\beta\eta\delta\zeta\iota$ -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 reduction relation 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 $\text{rec}_{a=} : \forall C. C a \rightarrow \forall b. a = b \rightarrow C b$:

$$\begin{aligned} \lambda h : a = a. \text{rec}_{a=} C e a h &\rightsquigarrow_{\eta} \text{rec}_{a=} C e a \\ \lambda h : a = a. \text{rec}_{a=} C e a h &\rightsquigarrow_{\iota} \lambda h : a = a. e \end{aligned}$$

To resolve this, we will require that rec and lift always have their required number of parameters. To accomplish this, we will slightly modify the language to remove rec_P the function and replace it with a fully applied form $\text{rec}_P(C, e, p, x)$. (We will use the notation $f(x)$ instead of $f x$ for these “atomic applications”.) The transformation is as follows:

- If e is a list of terms of length n and rec_P has $m \geq n$ arguments, then $\overline{\text{rec}_P e} = \lambda x :: \alpha. \text{rec}_P(e, x)$ 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 $\overline{\text{lift } e} = \lambda x :: \alpha. \text{lift}(e, x)$ 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.

The reverse translation maps $\overline{\text{rec}_P(C, e, p, x)} = \text{rec}_P C e p x$ and $\overline{\text{lift}(\alpha, R, \beta, f, h, q)} = \text{lift } \alpha R \beta f h q$ and leaves all other forms unchanged. The typing rules are extended in the obvious way to the new forms. The relevant properties of this transformation are:

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

- If $\Gamma \vdash e : \alpha$ in the modified language, then $\underline{\Gamma} \vdash \underline{e} : \underline{\alpha}$ in the original language and $\overline{(\underline{e})} = e$.
- If $\Gamma \vdash e : \alpha$ in the original language, then $\overline{\Gamma} \vdash \overline{e} : \overline{\alpha}$ and $\Gamma \vdash (\overline{\overline{e}}) \equiv e$. (In fact, the proof of equivalence uses only η .)
- If $\Gamma \vdash e \equiv e'$ in the modified language, then $\underline{\Gamma} \vdash \underline{e} \equiv \underline{e}'$.
- If $\Gamma \vdash e \equiv e'$ in the original language, then $\overline{\Gamma} \vdash \overline{e} \equiv \overline{e}'$.

With this new language, the problem above is rejected on syntax grounds:

$$\begin{aligned} \lambda h : a = a. \text{rec}_{a=}(C, e, a, h) &\rightsquigarrow_{\iota} \lambda h : a = a. e \\ \lambda h : a = a. \text{rec}_{a=}(C, e, a, h) &\not\rightsquigarrow_{\eta} \text{rec}_{a=}(C, e, a) \end{aligned}$$

where the second reduction doesn't work because $\text{rec}_{a=}(C, e, a, h)$ is not an application, so the η rule doesn't apply (and $\text{rec}_{a=}(C, e, a)$ is not syntactically correct because it has the wrong number of arguments).

Remark 1. *Rather than modifying the language, we could have instead modified the eta rule to not apply when the argument is a recursor with the correct number of arguments, but this approach of using application-like forms that are not applications seems clearer to the author.*

The κ reduction relation is defined on the modified language, with compatibility rules such as these for every syntax operator (including $\text{rec}_P(e)$ and $\text{lift}(e)$):

$$\begin{array}{c} \boxed{\Gamma \vdash e \rightsquigarrow_{\kappa} e'} \quad \frac{\Gamma \vdash e_1 \rightsquigarrow_{\kappa} e'_1}{\Gamma \vdash e_1 e_2 \rightsquigarrow_{\kappa} e'_1 e_2} \quad \frac{\Gamma \vdash e_2 \rightsquigarrow_{\kappa} e'_2}{\Gamma \vdash e_1 e_2 \rightsquigarrow_{\kappa} e_1 e'_2} \\[10pt] \frac{\Gamma \vdash \alpha \rightsquigarrow_{\kappa} \alpha'}{\Gamma \vdash \lambda x : \alpha. e \rightsquigarrow_{\kappa} \lambda x : \alpha'. e} \quad \frac{\Gamma, x : \alpha \vdash e \rightsquigarrow_{\kappa} e'}{\Gamma \vdash \lambda x : \alpha. e \rightsquigarrow_{\kappa} \lambda x : \alpha. e'} \quad \dots \end{array}$$

The substantive rules are:

$$\begin{array}{ll} (\beta) \frac{}{\Gamma \vdash (\lambda x : \alpha. e) e' \rightsquigarrow_{\kappa} e[e'/x]} & (\eta) \frac{x \notin FV(e) \quad \Gamma, x : \alpha \vdash h \equiv_p x}{\Gamma \vdash \lambda x : \alpha. e h \rightsquigarrow_{\kappa} e} \\[10pt] (\delta) \frac{\text{def } c : \alpha := e}{\Gamma \vdash c \rightsquigarrow_{\kappa} e} & (\zeta) \frac{}{\Gamma \vdash \text{let } x : \alpha := e' \text{ in } e \rightsquigarrow_{\kappa} e[e'/x]} \\[10pt] (\iota) \frac{P \text{ is non-SS inductive with ctor } c}{\Gamma \vdash \text{rec}_P(C, e, p, c b) \rightsquigarrow_{\kappa} e_c b v} & (\iota_q) \frac{}{\Gamma \vdash \text{lift}(R, f, h, \text{mk}_R a) \rightsquigarrow_{\kappa} f a} \end{array}$$

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

See [section 2.6](#) for the variable names and types used in the regular ι rule.

We have an alternate ι rule for SS inductives, where $\text{inv}[p, h]$ is a sequence of terms such that $\text{intro inv}[p, h] \equiv h$ (by proof irrelevance) and $\text{inv}_i[p, \text{intro } b] \equiv b_i$, which we call K^+ because it is a souped-up version of the K-like reduction rule in [section 2.6.4](#). 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 $\text{inv}_i[p, h] := p_j$ when the i th constructor argument is non-propositional and appears at position j in the output type, and $\text{inv}_i[p, h] = \text{inv}_i h$ for the propositions, where inv_i is an atomic projection function. These inv_i projection operators can be defined in the original language using the recursor, like we demonstrated for acc in [section 3.1](#), but we will treat them as constants of the modified language.

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.

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

$$\frac{\Gamma \vdash e : \alpha}{\Gamma \vdash e \equiv_p e} \quad \frac{\Gamma \vdash \alpha \equiv_p \alpha' \quad \Gamma, x : \alpha \vdash e \equiv_p e'}{\Gamma \vdash \forall x : \alpha. e \equiv_p \forall x : \alpha'. e'} \quad \frac{\Gamma \vdash e_1 \equiv_p e'_1 \quad \Gamma \vdash e_2 \equiv_p e'_2}{\Gamma \vdash e_1 e_2 \equiv_p e'_1 e'_2} \quad \dots$$

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

The ellipsis abbreviates all the compatibility rules, but note that the relation allows full definitional equality in the domain of a lambda, that is, it requires that $\Gamma \vdash \alpha \equiv \alpha'$ instead of $\Gamma \vdash \alpha \equiv_p \alpha'$ like the other rules. To give a hint on why this is relevant, consider the following reduction sequence:

$$\begin{aligned} \lambda x : \alpha. (\lambda y : \beta. e) x &\rightsquigarrow_\eta \lambda y : \beta. e \\ \lambda x : \alpha. (\lambda y : \beta. e) x &\rightsquigarrow_\beta \lambda x : \alpha. e[x/y] = \lambda y : \alpha. e \end{aligned}$$

(We view α -equivalent terms as syntactically equal.) By reducing the expression in two ways, we obtain the “same” expression $\lambda y. e$, but the type annotation is different, and although we know from the application rule that $\alpha \equiv \beta$, this equivalence may involve the full strength of the type system to prove. These two terms are, however, \equiv_p -equivalent, and this is close enough to equal for us to prove the main theorems.

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_κ .

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_{κ} by the following rules:

$$\begin{array}{c}
\frac{}{\Gamma \vdash x \gg_{\kappa} x} \quad \frac{\Gamma \vdash \alpha \gg_{\kappa} \alpha' \quad \Gamma, x : \alpha \vdash e \gg_{\kappa} e'}{\Gamma \vdash \lambda x : \alpha. e \gg_{\kappa} \lambda x : \alpha'. e'} \quad \frac{\Gamma \vdash e_1 \gg_{\kappa} e'_1 \quad \Gamma \vdash e_2 \gg_{\kappa} e'_2}{\Gamma \vdash e_1 e_2 \gg_{\kappa} e'_1 e'_2} \quad \dots \\
\\
\frac{\Gamma, x : \alpha \vdash e_1 \gg_{\kappa} e'_1 \quad \Gamma \vdash e_2 \gg_{\kappa} e'_2}{\Gamma \vdash (\lambda x : \alpha. e_1) e_2 \gg_{\kappa} e'_1[e'_2/x]} \quad \frac{x \notin FV(e) \quad \Gamma, x : \alpha \vdash h \equiv_p x \quad \Gamma \vdash e \gg_{\kappa} e'}{\Gamma \vdash \lambda x : \alpha. e h \gg_{\kappa} e'} \\
\\
\frac{\Gamma \vdash e_2[e_1/x] \gg_{\kappa} e'}{\Gamma \vdash \text{let } x : \alpha := e_1 \text{ in } e_2 \gg_{\kappa} e'} \quad \frac{\text{def } c : \alpha := e \quad \Gamma \vdash e \gg_{\kappa} e'}{\Gamma \vdash c \gg_{\kappa} e'} \\
\\
\frac{\Gamma \vdash f \gg_{\kappa} f' \quad \Gamma \vdash a \gg_{\kappa} a'}{\Gamma \vdash \text{lift}(R, f, h, \text{mk}_R a) \gg_{\kappa} f' a'} \quad \frac{P \text{ is non-SS inductive with ctor } c \quad \Gamma \vdash C, e, b, p \gg_{\kappa} C', e', b', p'}{\Gamma \vdash \text{rec}_P(C, e, p, c b) \gg_{\kappa} e' b' v'} \\
\\
\frac{P \text{ is SS inductive} \quad \Gamma \vdash C, e, p, h \gg_{\kappa} C', e', p', h'}{\Gamma \vdash \text{rec}_P(C, e, p, h) \gg_{\kappa} e'_c \text{inv}[p', h'] v'} \quad \frac{P \text{ is SS inductive} \quad \Gamma \vdash b_i \gg_{\kappa} b'_i}{\Gamma \vdash \text{inv}_i(\text{intro } b) \gg_{\kappa} b'_i}
\end{array}$$

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_{κ} 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_{κ} , except that the compatibility rules only apply if none of the substantive rules are applicable. This makes \ggg_{κ} 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_{κ}).

- (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_{κ} 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 $e_1 \gg_\kappa e_2 \equiv_p e_2$.
- 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 $e_1 \gg_\kappa e_2 \equiv_p e_2$. (We will thus assume that e_1 is not a proof in later cases.)
- If $\lambda x : \alpha_1. e_1 \equiv_p \lambda x : \alpha_3. e_3 \gg_\kappa \lambda x : \alpha_2. e_2$ by the lambda compatibility rule, then $\alpha_1 \equiv \alpha_3 \gg_\kappa \alpha_2$ and $e_1 \equiv_p e_3 \gg_\kappa e_2$, and by the IH we have $e_1 \gg_\kappa e_4 \equiv_p e_2$, so $\lambda x : \alpha_1. e_1 \gg_\kappa \lambda x : \alpha_3. e_4 \equiv_p \lambda x : \alpha_2. e_2$.
- 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 $e_1 \equiv_p e_2 \gg_\kappa e_3$, $e'_1 \equiv_p e'_2 \gg_\kappa e'_3$ 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_κ is a substantive rule.)
- If $\lambda x : \alpha_1. e_1 h_1 \equiv_p \lambda x : \alpha_3. e_3 h_3 \gg_\kappa e_2$ where $e_1 \equiv_p e_3 \gg_\kappa e_2$ and $h_1 \equiv_p h_3 \equiv_p x$, then by the IH $e_1 \gg_\kappa e_4 \equiv_p e_2$ so that $\lambda x : \alpha_1. e_1 h_1 \gg_\kappa e_4 \equiv_p e_2$.
- If $\text{lift}(R_1, \beta_1, f_1, h_1, q_1) \equiv_p \text{lift}(R_3, \beta_3, f_3, h_3, \text{mk}_R a_3)$ where $q_1 \equiv_p \text{mk}_R a_3$ 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 $\text{rec}_P(C_1, e_1, p_1, h_1) \equiv_p \text{rec}_P(C_3, e_3, p_3, c b_3) \gg_\kappa (e_2)_c b_2 v_2$ where P is non-SS inductive and $h_1 \equiv_p c b_3$ by proof irrelevance, it is a small eliminator, so $\text{rec}_P(C_1, e_1, p_1, h_1)$ is a proof.
- If $e_1 \equiv_p \text{inv}_i$ (intro b_3), then e_1 is a proof, since the inv_i function is only defined at propositional arguments.

□

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 $x \ll_\kappa x \gg_\kappa x$, then $x \gg_\kappa x \equiv_p x$.
- If $e \ggg_\kappa e^\bullet$ by the beta rule:
 - If $e_1^\bullet[e_2^\bullet/x] \ll_\kappa (\lambda x : \alpha. e_1) e_2 \gg_\kappa e_1^\bullet[e_2^\bullet/x]$ 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 $e_1^\bullet[e_2^\bullet/x] \ll_\kappa (\lambda x : \alpha. e_1) e_2 \gg_\kappa (\lambda x : \alpha. e_1') e_2'$ 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_κ and the IH.
 - If $e_1^\bullet h[e_2^\bullet/x] \ll_\kappa (\lambda x : \alpha. e_1 h) e_2 \gg_\kappa e_1' e_2'$ by the application rule and eta rule, then $e_1' e_2' \gg_\kappa e_1^\circ e_2^\circ \equiv_p e_1^\bullet h[e_2^\bullet/x]$ because $e_2^\circ \equiv_p e_2^\bullet = x[e_2^\bullet/x] \equiv_p h[e_2^\bullet/x]$.
- If $e \ggg_\kappa e^\bullet$ by the eta rule:
 - If $e^\bullet \ll_\kappa \lambda x : \alpha. e h \gg_\kappa e'$ by the eta rule, then $e' \gg_\kappa e^\circ \equiv_p e^\bullet$.
 - If $e^\bullet \ll_\kappa \lambda x : \alpha. e h \gg_\kappa \lambda x : \alpha'. e' h'$ by the lambda rule and application rule, then $\lambda x : \alpha'. e' h' \gg_\kappa e^\circ \equiv_p e^\bullet$.

- If $\lambda y : \beta^\bullet. e^\bullet \lll_\kappa \lambda x : \alpha. (\lambda y : \beta. e) h \ggg_\kappa \lambda x : \alpha'. e'[h/y]$ by the lambda rule and beta rule, then $\lambda x : \alpha'. e'[h/y] \ggg_\kappa \lambda x : \alpha^\circ. e^\circ[h/y] \equiv_p \lambda x : \beta^\bullet. e^\bullet[x/y] = \lambda y : \beta^\bullet. e^\bullet$. (Here we have used the λ rule for \equiv_p to equate $\alpha^\circ \equiv \beta^\bullet$, which are def.eq. because α and β are.)
- If $e^\bullet \lll_\kappa c \ggg_\kappa e'$ by the delta rule, then $e' \ggg_\kappa e^\circ \equiv_p e^\bullet$.
- If $e_1^\bullet[e_2^\bullet/x] \lll_\kappa \text{let } x : \alpha := e_1 \text{ in } e_2 \ggg_\kappa e_2'[e_1'/x]$ by the zeta rule, then $e_2'[e_1'/x] \ggg_\kappa e_1^\circ[e_2^\circ/x] \equiv_p e_1^\bullet[e_2^\bullet/x]$.
- If $e \ggg_\kappa e^\bullet$ by the non-SS inductive iota rule:
 - If $e_c^\bullet b^\bullet v^\bullet \lll_\kappa \text{rec}_P(C, e, p, c b) \ggg_\kappa e'_c b' v'$ by the iota rule, then $e'_c b' v' \ggg_\kappa e_c^\circ b^\circ v^\circ \equiv_p e_c^\bullet b^\bullet v^\bullet$.
 - If $e_c^\bullet b^\bullet v^\bullet \lll_\kappa \text{rec}_P(C, e, p, c b) \ggg_\kappa \text{rec}_P(C', e', p', c b')$ by the rec_P compatibility rule, then $\text{rec}_P(C', e', p', c b') \ggg_\kappa e_c^\circ b^\circ v^\circ \equiv_p e_c^\bullet b^\bullet v^\bullet$ by the iota rule.
- If $e \ggg_\kappa e^\bullet$ by the quotient iota rule:
 - If $f^\bullet a^\bullet \lll_\kappa \text{lift}(R, f, h, \text{mk}_R a) \ggg_\kappa f' a'$ by the iota rule, then $f' a' \ggg_\kappa f^\circ a^\circ \equiv_p f^\bullet a^\bullet$.
 - If $f^\bullet a^\bullet \lll_\kappa \text{lift}(R, f, h, \text{mk}_R a) \ggg_\kappa \text{lift}(R', f', h', \text{mk}_{R'} a')$ by the lift compatibility rule, then we have $q' \ggg_\kappa q^\circ \equiv_p q^\bullet$ for $q \in \{C, e, p, h\}$, and $\text{lift}(R', f', h', \text{mk}_{R'} a') \ggg_\kappa f^\circ a^\circ \equiv_p f^\bullet a^\bullet$.
- If $e \ggg_\kappa e^\bullet$ by the K^+ rule:
 - If $e_c^\bullet \text{inv}[p^\bullet, h^\bullet] v^\bullet \lll_\kappa \text{rec}_P(C, e, p, h) \ggg_\kappa e'_c \text{inv}[p', h'] v'$ by the iota rule, then $e'_c \text{inv}[p', h'] v' \ggg_\kappa e_c^\circ \text{inv}[p^\circ, h^\circ] v^\circ \equiv_p e_c^\bullet \text{inv}[p^\bullet, h^\bullet] v^\bullet$.
 - If $e_c^\bullet \text{inv}[p^\bullet, h^\bullet] v^\bullet \lll_\kappa \text{rec}_P(C, e, p, h) \ggg_\kappa \text{rec}_P(C', e', p', h')$ by the rec_P compatibility rule, then $\text{rec}_P(C', e', p', h') \ggg_\kappa e_c^\circ \text{inv}[p^\circ, h^\circ] v^\circ \equiv_p e_c^\bullet \text{inv}[p^\bullet, h^\bullet] v^\bullet$ by the iota rule.
- If $e \ggg_\kappa e^\bullet$ by the K-iota rule:
 - If $b_i^\bullet \lll_\kappa \text{inv}_i(c b) \ggg_\kappa b'_i$ by the iota rule, then $b'_i \ggg_\kappa b_i^\circ \equiv_p b_i^\bullet$ by the IH.
 - If $b_i^\bullet \lll_\kappa \text{inv}_i(c b) \ggg_\kappa \text{inv}_i(c b')$ by compatibility rules, then $\text{inv}_i(c b') \ggg_\kappa b_i^\circ \equiv_p b_i^\bullet$ by the K-iota rule.
- If $e \ggg_\kappa e^\bullet$ by a compatibility rule:
 - If $e_1^\bullet e_2^\bullet \lll_\kappa e_1 e_2 \ggg_\kappa e'_1 e'_2$ by the application rule, then $e'_1 e'_2 \ggg_\kappa e_1^\circ e_2^\circ \equiv_p e_1^\bullet e_2^\bullet$.
 - If $\forall x : \alpha^\bullet. e^\bullet \lll_\kappa \forall x : \alpha. e \ggg_\kappa \forall x : \alpha'. e'$ by the forall rule, then $\forall x : \alpha'. e' \ggg_\kappa \forall x : \alpha^\circ. e^\circ \equiv_p \forall x : \alpha^\bullet. e^\bullet$.
 - 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 $e_1 \lll_\kappa e \ggg_\kappa e_2$, then $\exists e'_1 e'_2. e_1 \ggg_\kappa e'_1 \equiv_p e'_2 \lll_\kappa e_2$.
2. If $\Gamma \vdash e : \alpha$, and $e_1 \lll_\kappa^* e \ggg_\kappa e_2$, then $\exists e'_1 e'_2. e_1 \ggg_\kappa e'_1 \equiv_p e'_2 \lll_\kappa^* e_2$.
3. If $\Gamma \vdash e : \alpha$, and $e_1 \lll_\kappa^* e \ggg_\kappa^* e_2$, then $\exists e'_1 e'_2. e_1 \ggg_\kappa^* e'_1 \equiv_p e'_2 \lll_\kappa^* e_2$.
4. If $\Gamma \vdash e : \alpha$, and $e_1 \lll_\kappa^* e \ggg_\kappa^* e_2$, then $\exists e'_1 e'_2. e_1 \ggg_\kappa^* e'_1 \equiv_p e'_2 \lll_\kappa^* e_2$.

(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 $e_3 \gg_\kappa e'_3 \equiv_p e'_1 \ll_\kappa e'_2$, 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_κ^* , 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 $\Gamma \vdash e_1 \rightsquigarrow_\kappa^* e'_1 \equiv_p e'_2 \rightsquigarrow_\kappa^* e_2$. 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 $\Gamma \vdash e \equiv_\kappa e'$ (in the modified language).

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_κ is an equivalence relation (by the Church-Rosser property).
- For the compatibility rules, since both \equiv_p and \rightsquigarrow_κ have compatibility rules, this property passes to \equiv_κ . 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_κ , and the proof irrelevance rule is in \equiv_p . All the other equivalence rules are also introduced in \rightsquigarrow_κ .
- For subsingleton eliminators, we must show $\text{rec}_P(C, e, p, \text{intro } b) \equiv_\kappa e \text{ } b \text{ } v$. From the K^+ rule we have $\text{rec}_P(C, e, p, \text{intro } b) \equiv_\kappa \text{inv}[p, c \text{ } b] \text{ } v$ so it suffices to show $\text{inv}_i[p, \text{intro } b] \equiv_\kappa b_i$ for each i . If b_i is propositional then this is by proof irrelevance, otherwise $\text{inv}_i[p, \text{intro } b] = p_j$, and the well-typedness of $\text{rec}_P(C, e, p, \text{intro } b)$ 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 $\Gamma \vdash_{n+1} U_\ell \equiv U_{\ell'}$, then $\ell \equiv \ell'$.

Again, there are no \rightsquigarrow_κ reductions from U_ℓ , so $\Gamma \vdash_{n+1} U_\ell \equiv_p U_{\ell'}$, and if the compatibility rule is used then $\ell \equiv \ell'$. If proof irrelevance is used, then $\Gamma \vdash_n U_\ell, U_{\ell'} : p$ for some $\Gamma \vdash_n p : \mathbb{P}$. Since $\Gamma \vdash_n U_\ell : U_{S\ell} : U_{SS\ell}$ as well, by unique typing at n , $\Gamma \vdash_n \mathbb{P} \equiv U_{SS\ell}$, 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_κ 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 $\Gamma \vdash_n \forall x : \alpha_1. \beta_1 : U_{\text{imax}(\ell_1, \ell_2)}$ for some ℓ_1, ℓ_2 since α_1 and β_1 are well-typed, so by unique typing at n , $p \equiv U_{\text{imax}(\ell_1, \ell_2)}$ and $0 \equiv S \text{imax}(\ell_1, \ell_2)$, a contradiction.

3. $\Gamma \vdash_n U_\ell \not\equiv \forall x : \alpha. \beta$.

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 $\Gamma \vdash_n U_\ell \equiv_p \forall x : \alpha'. \beta'$, and now there is no applicable rule except proof irrelevance, but this implies $U_\ell : p : \mathbb{P}$ 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 $\Gamma \vdash A : U_\ell$ and $\Gamma, x : A \vdash B : U_\ell$:

$$Wx : A. B := \mu w : U_\ell. (\text{sup} : \forall x : A. (B \rightarrow w) \rightarrow w)$$

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 $\text{ulift}_u^v : \mathbf{U}_u \rightarrow \mathbf{U}_v$, defined when $u \leq v$, as well as the **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:

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

$$\begin{aligned}
& \frac{}{\vdash \perp : \mathbb{P}} \quad \frac{1 \leq \ell \quad \Gamma \vdash C : \mathbf{U}_\ell}{\Gamma \vdash \text{rec}_\perp : \perp \rightarrow C} \\
& \frac{\Gamma \vdash \alpha : \mathbf{U}_\ell \quad \Gamma, x : \alpha \vdash \beta : \mathbf{U}_{\ell'}}{\Gamma \vdash \Sigma x : \alpha. \beta : \mathbf{U}_{\max(\ell, \ell', 1)}} \quad \frac{\Gamma \vdash \alpha : \mathbf{U}_\ell \quad \Gamma \vdash \beta : \mathbf{U}_{\ell'}}{\Gamma \vdash \alpha + \beta : \mathbf{U}_{\max(\ell, \ell', 1)}} \\
& \frac{\Gamma \vdash e_1 : \alpha \quad \Gamma \vdash e_2 : \beta}{\Gamma \vdash (e_1, e_2) : \Sigma x : \alpha. \beta} \quad \frac{\Gamma \vdash p : \Sigma x : \alpha. \beta}{\Gamma \vdash \pi_1 p : \alpha} \quad \frac{\Gamma \vdash p : \Sigma x : \alpha. \beta}{\Gamma \vdash \pi_2 p : \beta[\pi_1 p/x]} \\
& \frac{\Gamma \vdash \beta \text{ type} \quad \Gamma \vdash e : \alpha}{\Gamma \vdash \text{inl } e : \alpha + \beta} \quad \frac{\Gamma \vdash \alpha \text{ type} \quad \Gamma \vdash e : \beta}{\Gamma \vdash \text{inr } e : \alpha + \beta} \\
& \frac{1 \leq \ell \quad \Gamma \vdash C : \alpha + \beta \rightarrow \mathbf{U}_\ell \quad \Gamma \vdash a : \forall x : \alpha. C (\text{inl } x) \quad \Gamma \vdash b : \forall x : \beta. C (\text{inr } x)}{\Gamma \vdash \text{rec}_+ a b : \forall p : \alpha + \beta. C p} \\
& \frac{\Gamma \vdash \alpha : \mathbf{U}_\ell \quad \max(1, \ell) \leq \ell'}{\Gamma \vdash \text{ulift}_\ell^{\ell'} \alpha : \mathbf{U}_{\ell'}} \quad \frac{\Gamma \vdash \text{ulift}_\ell^{\ell'} \alpha : \mathbf{U}_{\ell'} \quad \Gamma \vdash e : \alpha}{\Gamma \vdash \uparrow e : \text{ulift}_\ell^{\ell'} \alpha} \quad \frac{\Gamma \vdash e : \text{ulift}_\ell^{\ell'} \alpha}{\Gamma \vdash \downarrow e : \alpha} \\
& \frac{\Gamma \vdash \alpha \text{ type}}{\Gamma \vdash \|\alpha\| : \mathbb{P}} \quad \frac{\Gamma \vdash e : \alpha}{\Gamma \vdash |e| : \|\alpha\|} \quad \frac{\Gamma \vdash C : \mathbb{P} \quad \Gamma \vdash f : \alpha \rightarrow C}{\Gamma \vdash \text{rec}_\parallel f : \|\alpha\| \rightarrow C} \\
& \frac{\Gamma \vdash \alpha : \mathbf{U}_\ell \quad \Gamma, x : \alpha \vdash \beta : \mathbf{U}_{\ell'}}{\Gamma \vdash \mathbf{W}x : \alpha. \beta : \mathbf{U}_{\max(\ell, \ell', 1)}} \quad \frac{\Gamma \vdash a : \alpha \quad \Gamma \vdash f : \beta[a/x] \rightarrow \mathbf{W}x : \alpha. \beta}{\Gamma \vdash \text{sup } a f : \mathbf{W}x : \alpha. \beta} \\
& \frac{1 \leq \ell \quad \Gamma \vdash C : (\mathbf{W}x : \alpha. \beta) \rightarrow \mathbf{U}_\ell \quad \Gamma \vdash e : \forall (a : \alpha) (f : \beta[a/x] \rightarrow \mathbf{W}x : \alpha. \beta). (\forall b : \beta[a/x]. C (f b)) \rightarrow C (\text{sup } a f)}{\Gamma \vdash \text{rec}_\mathbf{W} e : \forall w : (\mathbf{W}x : \alpha. \beta). C w} \\
& \frac{\Gamma \vdash a : \alpha \quad \Gamma \vdash b : \alpha}{\Gamma \vdash a = b : \mathbb{P}} \quad \frac{\Gamma \vdash a : \alpha}{\Gamma \vdash \text{refl } a : a = a} \\
& \frac{\Gamma \vdash a : \alpha \quad 1 \leq \ell \quad \Gamma \vdash C : \alpha \rightarrow \mathbf{U}_\ell \quad \Gamma \vdash e : C a}{\Gamma \vdash \text{rec}_= e : \forall b : \alpha. a = b \rightarrow C b} \\
& \frac{\Gamma \vdash r : \alpha \rightarrow \alpha \rightarrow \mathbb{P}}{\Gamma \vdash \text{acc}_r : \alpha \rightarrow \mathbb{P}} \quad \frac{\Gamma \vdash x : \alpha \quad \Gamma \vdash f : \forall y : \alpha. r y x \rightarrow \text{acc}_r y}{\Gamma \vdash \text{intro}_{\text{acc}} x f : \text{acc}_r x} \\
& \frac{1 \leq \ell \quad \Gamma \vdash C : \alpha \rightarrow \mathbf{U}_\ell \quad \Gamma \vdash e : \forall x : \alpha. (\forall y : \alpha. r y x \rightarrow \text{acc}_r y) \rightarrow (\forall y : \alpha. r y x \rightarrow C y) \rightarrow C x}{\Gamma \vdash \text{rec}_{\text{acc}} e : \forall x : \alpha. \text{acc}_r x \rightarrow C x}
\end{aligned}$$

All of these could have been defined as inductive types in the sense of [section 2.6](#):

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

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 $1 \leq \ell$. The general recursor can be constructed from this by using $C' := \lambda x : P. \text{ulift}_\ell^{\max(1, \ell)} (C x)$ (for each such inductive type P).

In a few of the constructors, additional parameters are elided, such as C in rec_\perp ; 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:

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

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:

$$\begin{aligned}
\mathbf{0}_\ell &:= \text{ulift}^\ell \perp & \top &:= \perp \rightarrow \perp & \mathbf{1}_\ell &:= \text{ulift}^\ell \top & \alpha \times \beta &:= \Sigma_- : \alpha. \beta \\
p \wedge q &:= \|p \times q\| & p \vee q &:= \|p + q\| \\
\{x : \alpha \mid p\} &:= \Sigma x : \alpha. p & \exists x : \alpha. p &:= \|\{x : \alpha \mid p\}\|
\end{aligned}$$

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 .)

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

The results of [section 4](#) apply straightforwardly to this setting, with these two rules added as \rightsquigarrow_κ 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 $1 \leq \ell$. 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 $\text{good} : S \rightarrow \forall x :: \alpha. \mathbb{P}$ that carves out the members that actually belong to index x . The final result will be the type $\lambda x :: \alpha. \{s : S \mid \text{good } s \ x\}$. For example, the type

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

has the indices erased to get

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

and then the predicate is defined by recursion on S :

$$\begin{aligned} \text{good one } m &:= m = 1 \\ \text{good (double } n \ x) \ m &:= m = 2n \wedge \text{good } x \ n \end{aligned}$$

Now S will also be reduced to the \mathbf{W} -type:

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

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

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

$$\begin{aligned} \text{where} \quad & \Gamma \vdash A : \mathbf{U}_\ell \\ & \Gamma \vdash B : A \rightarrow \mathbf{U}_\ell \\ & \Gamma \vdash G : \forall p : A. (B \ p \rightarrow \forall x :: \alpha. \mathbb{P}) \rightarrow \forall x :: \alpha. \mathbb{P}. \end{aligned}$$

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

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

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

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

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

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

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

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

In the final rule, the notation (z) where $z :: \gamma$ means the tuple of elements of z of type $\Sigma(z :: \gamma)$: explicitly, $(z_1, \dots, z_n) = (z_1, (z_2, \dots, (x_n, ()))) : \Sigma(z :: \gamma)$. 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 $x_1 = e_1 \wedge x_2 = e_2 \wedge \dots \wedge x_n = e_n$, or using the equality recursor $\exists(h_1 : x_1 = e_1) (h_2 : \text{rec}_= x_2 x_1 h_1 = e_2) \dots$. We will use $(x) = (e)$ since it is the least notationally burdensome of these options.

The final result is given by the following translation:

$$\frac{\Gamma; t : F \vdash K \text{ spec} \Rightarrow A; B; G}{\Gamma \vdash \llbracket \mu t : F. K \rrbracket = \lambda x :: \alpha. \{s : Wp : A. B p \mid \text{rec}_W (\lambda(p : A) \dots G p) s x\}}$$

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 it back to the original universe ℓ :

$$\frac{\Gamma; t : F \vdash K \text{ spec} \quad \neg(\Gamma; t : \forall x :: \alpha. \mathbf{U}_\ell \vdash K \text{ LE})}{\Gamma \vdash \llbracket \mu t : \forall x :: \alpha. \mathbf{U}_\ell. K \rrbracket = \lambda x :: \alpha. \text{ulift}^\ell \llbracket \mu t : \forall x :: \alpha. \mathbf{U}_{\ell'}. K \rrbracket x},$$

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:

$$\frac{\Gamma \vdash x :: \alpha}{\Gamma \vdash \llbracket \mu t : \forall x :: \alpha. \mathbf{U}_\ell. 0 \rrbracket = \lambda x :: \alpha. \mathbf{0}_\ell}$$

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

$$\alpha : \mathbf{U}_\ell, \varphi : \alpha \rightarrow \mathbb{P}, r : \alpha \rightarrow \alpha \rightarrow \mathbb{P} \vdash \text{acc}_r^\varphi = \mu t : \alpha \rightarrow \mathbb{P}. \\
(\text{intro} : \forall x : \alpha. \varphi x \rightarrow (\forall y : \alpha. r y x \rightarrow t y) \rightarrow t x)$$

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:

$$\text{acc}_r^\varphi x := \exists h : \varphi x. \text{acc}_{r'}(x, h)$$

$$\text{where } r' := \lambda x x' : \{x : \alpha \mid \varphi x\}. r (\pi_1 x) (\pi_1 x')$$

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:

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

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 ($\forall m. m < n \rightarrow t \ n \ m$).

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 non-recursive 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:

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

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 .

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

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

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

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

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:

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

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 $\perp, \Sigma, +, \text{ulift}, \|\cdot\|, \mathbf{W}, =, \text{acc}$ are indeed sufficient to cover all Lean-definable inductive types, and leave all this horrible induction behind.

6 Soundness

6.1 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 $\bullet = \emptyset$ (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 start with the levels, which have a simple interpretation. 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 $\llbracket \ell \rrbracket_v$ is defined when v is a function with domain containing $FV(\ell)$ and codomain \mathbb{N} , as follows:

$$\begin{aligned} \llbracket u \rrbracket_v &= v(u) \\ \llbracket 0 \rrbracket_v &= 0 \\ \llbracket S\ell \rrbracket_v &= \llbracket \ell \rrbracket_v + 1 \\ \llbracket \max(\ell, \ell') \rrbracket_v &= \max(\llbracket \ell \rrbracket_v, \llbracket \ell' \rrbracket_v) \\ \llbracket \text{imax}(\ell, \ell') \rrbracket_v &= \begin{cases} 0 & \text{if } \llbracket \ell' \rrbracket_v = 0 \\ \max(\llbracket \ell \rrbracket_v, \llbracket \ell' \rrbracket_v) & \text{if } \llbracket \ell' \rrbracket_v \neq 0 \end{cases} \end{aligned}$$

Since $\ell \equiv \ell'$ iff $\llbracket \ell \rrbracket_v = \llbracket \ell' \rrbracket_v$ for all v , from unique typing we have that given Γ and α , there is at most one n such that there exists ℓ with $\Gamma \vdash \alpha : \mathbf{U}_\ell$ and $\llbracket \ell \rrbracket_v = n$. We will denote this partial function, defined on expressions that are types, by $\text{lvl}_v(\Gamma \vdash \alpha)$. Fix a valuation v on the universe

variables. We define the expressions $\llbracket \Gamma \rrbracket_v$ when Γ is a well formed context, and $\llbracket \Gamma \vdash e \rrbracket_v$ 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:
 - $|\text{let } x : \alpha := e_1 \text{ in } e_2| = |e_2[e_1/x]| + 1$, and
 - $|c| = |e| + 1$ when $\text{def } c : \alpha := e$.
- 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 $\llbracket \Gamma \rrbracket$ will be a set of lists of types, and $\llbracket \Gamma \vdash e \rrbracket$ will be a (total) function on $\llbracket \Gamma \rrbracket$, if it is defined at all. We will denote the evaluation at $\gamma \in \llbracket \Gamma \rrbracket$ by $\llbracket \Gamma \vdash e \rrbracket_\gamma$ (or $\llbracket \Gamma \vdash e \rrbracket_{v,\gamma}$ when being explicit about the universe valuation as well).

Let $\prod_{x \in A}^n B = \begin{cases} \prod_{x \in A} B & \text{if } n \neq 0 \\ \{\bullet\} \cap \bigcap_{x \in A} B & \text{if } n = 0. \end{cases}$ This is the “modified” product space which depends

on the target universe ℓ . Also 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.

- $\llbracket \cdot \rrbracket = ()$, the empty list. (This can be encoded as \emptyset .)
- $\llbracket \Gamma, x : \alpha \rrbracket = \sum_{\gamma \in \llbracket \Gamma \rrbracket} \llbracket \Gamma \vdash \alpha \rrbracket_\gamma$, that is, the set of pairs (γ, x) such that $\gamma \in \llbracket \Gamma \rrbracket$ and $x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma$.
- $\llbracket \Gamma \vdash x \rrbracket_\gamma = \pi_i(\gamma)$, where x is the i th variable in the context.
- $\llbracket \Gamma \vdash U_\ell \rrbracket_\gamma = U_{\llbracket \ell \rrbracket}$
- If $\Gamma \vdash e_1 : \forall x : \alpha. \beta$ and $n := \text{lvl}(\Gamma, x : \alpha \vdash \beta)$, then

$$\llbracket \Gamma \vdash e_1 e_2 \rrbracket_\gamma = \begin{cases} \llbracket \Gamma \vdash e_1 \rrbracket_\gamma (\llbracket \Gamma \vdash e_2 \rrbracket_\gamma) & \text{if } n \neq 0 \\ \bullet & \text{if } n = 0. \end{cases}$$

- If $\Gamma, x : \alpha \vdash e : \beta : U_\ell$, then

$$\llbracket \Gamma \vdash \lambda x : \alpha. e \rrbracket_\gamma = \begin{cases} (x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma \mapsto \llbracket \Gamma, x : \alpha \vdash e \rrbracket_{(\gamma, x)}) & \text{if } \llbracket \ell \rrbracket \neq 0 \\ \bullet & \text{if } \llbracket \ell \rrbracket = 0. \end{cases}$$

- $\llbracket \Gamma \vdash \forall x : \alpha. \beta \rrbracket_\gamma = \prod_{x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma}^n \llbracket \Gamma, x : \alpha \vdash \beta \rrbracket_{(\gamma, x)}$ where $n := \text{lvl}(\Gamma, x : \alpha \vdash \beta)$.
- $\llbracket \Gamma \vdash \text{let } x : \alpha := e_1 \text{ in } e_2 \rrbracket_\gamma = \llbracket \Gamma \vdash e_2[e_1/x] \rrbracket_\gamma$
- $\llbracket \Gamma \vdash c \rrbracket_\gamma = \llbracket \Gamma \vdash e \rrbracket_\gamma$ when $\text{def } c : \alpha := e$
- $\llbracket \Gamma \vdash \perp \rrbracket_\gamma = \emptyset$
- $\llbracket \Gamma \vdash \text{rec}_\perp \rrbracket_\gamma = \emptyset$ (the empty function)
- $\llbracket \Gamma \vdash \Sigma x : \alpha. \beta \rrbracket_\gamma = \sum_{x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma} \llbracket \Gamma \vdash \beta \rrbracket_{(\gamma, x)}$
- $\llbracket \Gamma \vdash (e_1, e_2) \rrbracket_\gamma = (\llbracket \Gamma \vdash e_1 \rrbracket_\gamma, \llbracket \Gamma \vdash e_2 \rrbracket_\gamma)$
- $\llbracket \Gamma \vdash \pi_1 e \rrbracket_\gamma = \pi_1(\llbracket \Gamma \vdash e \rrbracket_\gamma)$
- $\llbracket \Gamma \vdash \pi_2 e \rrbracket_\gamma = \pi_2(\llbracket \Gamma \vdash e \rrbracket_\gamma)$
- $\llbracket \Gamma \vdash \alpha + \beta \rrbracket_\gamma = \llbracket \Gamma \vdash \alpha \rrbracket_\gamma \sqcup \llbracket \Gamma \vdash \beta \rrbracket_\gamma$
- $\llbracket \Gamma \vdash \text{inl } e \rrbracket_\gamma = \iota_1(\llbracket \Gamma \vdash \alpha \rrbracket_\gamma)$
- $\llbracket \Gamma \vdash \text{inr } e \rrbracket_\gamma = \iota_2(\llbracket \Gamma \vdash \beta \rrbracket_\gamma)$

- $\llbracket \Gamma \vdash \text{rec}_+ a b \rrbracket_\gamma$ is the function on $\llbracket \Gamma \vdash \alpha \rrbracket_\gamma \sqcup \llbracket \Gamma \vdash \beta \rrbracket_\gamma$ such that

$$\begin{aligned} \llbracket \Gamma \vdash \text{rec}_+ a b \rrbracket_\gamma(\iota_1(x)) &= \llbracket \Gamma \vdash a \rrbracket_\gamma(x) & \text{for } x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma \\ \llbracket \Gamma \vdash \text{rec}_+ a b \rrbracket_\gamma(\iota_2(y)) &= \llbracket \Gamma \vdash a \rrbracket_\gamma(y) & \text{for } y \in \llbracket \Gamma \vdash \beta \rrbracket_\gamma. \end{aligned}$$

- $\llbracket \Gamma \vdash \text{ulift}_\ell^{\ell'} \alpha \rrbracket_\gamma = \llbracket \Gamma \vdash \alpha \rrbracket_\gamma$
- $\llbracket \Gamma \vdash \uparrow e \rrbracket_\gamma = \llbracket \Gamma \vdash \downarrow e \rrbracket_\gamma = \llbracket \Gamma \vdash e \rrbracket_\gamma$
- $\llbracket \Gamma \vdash \|\alpha\| \rrbracket_\gamma = [\llbracket \Gamma \vdash \alpha \rrbracket_\gamma \neq \emptyset]$
- $\llbracket \Gamma \vdash |e| \rrbracket_\gamma = \bullet$
- $\llbracket \Gamma \vdash \text{rec}_\parallel^{C,\alpha} f \rrbracket_\gamma = \bullet$
- $\llbracket \Gamma \vdash Wx : \alpha. \beta \rrbracket_\gamma = W_{x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma} \llbracket \Gamma, x : \alpha \vdash \beta \rrbracket_{(\gamma, x)}$ (see below)
- $\llbracket \Gamma \vdash \sup a f \rrbracket_\gamma = (\llbracket \Gamma \vdash a \rrbracket_\gamma, \llbracket \Gamma \vdash f \rrbracket_\gamma)$
- $\llbracket \Gamma \vdash \text{rec}_W e \rrbracket_\gamma = \text{rec}_W(W_{x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma} \llbracket \Gamma, x : \alpha \vdash \beta \rrbracket_{(\gamma, x)}, \llbracket \Gamma \vdash e \rrbracket_\gamma)$ (see below)
- $\llbracket \Gamma \vdash a = b \rrbracket_\gamma = [\llbracket \Gamma \vdash a \rrbracket_\gamma = \llbracket \Gamma \vdash b \rrbracket_\gamma]$
- $\llbracket \Gamma \vdash \text{refl } a \rrbracket_\gamma = \bullet$
- $\llbracket \Gamma \vdash \text{rec}_\equiv^a e b h \rrbracket_\gamma = \llbracket \Gamma \vdash e \rrbracket_\gamma$
- $\llbracket \Gamma \vdash \text{acc}_{\alpha, r} x \rrbracket_\gamma = [x \in \text{acc}(\llbracket \Gamma \vdash \alpha \rrbracket_\gamma, \overline{\llbracket \Gamma \vdash r \rrbracket_\gamma})]$ (see below)
- $\llbracket \Gamma \vdash \text{intro}_{\text{acc}} x f \rrbracket_\gamma = \bullet$
- $\llbracket \Gamma \vdash \text{rec}_{\text{acc}}^r e \rrbracket_\gamma = \text{rec}_{\text{acc}}(\llbracket \Gamma \vdash \alpha \rrbracket_\gamma, \overline{\llbracket \Gamma \vdash r \rrbracket_\gamma}, \llbracket \Gamma \vdash e \rrbracket_\gamma)$ (see below)
- If $\Gamma \vdash \alpha : U_\ell$ where $\llbracket \ell \rrbracket = 0$, then
 - $\llbracket \Gamma \vdash \alpha/R \rrbracket_\gamma = \llbracket \Gamma \vdash \alpha \rrbracket_\gamma$
 - $\llbracket \Gamma \vdash \text{mk}_R x \rrbracket_\gamma = \bullet$
 - $\llbracket \Gamma \vdash \text{lift}_R^u \beta f h \rrbracket_\gamma = \llbracket \Gamma \vdash f \rrbracket_\gamma$

Otherwise, let \sim be the equivalence closure of $\overline{\llbracket \Gamma \vdash R \rrbracket_\gamma}$ in the following clauses:

- $\llbracket \Gamma \vdash \alpha/R \rrbracket_\gamma = \llbracket \Gamma \vdash \alpha \rrbracket_\gamma / \sim$
- $\llbracket \Gamma \vdash \text{mk}_R x \rrbracket_\gamma = [\llbracket \Gamma \vdash x \rrbracket_\gamma] \sim$
- $\llbracket \Gamma \vdash \text{lift}_R \beta f h \rrbracket_\gamma$ is the function such that $\llbracket \Gamma \vdash \text{lift}_R \beta f h \rrbracket_\gamma([x] \sim) = \llbracket \Gamma \vdash f \rrbracket_\gamma(x)$

- $\llbracket \Gamma \vdash \text{sound}_R \rrbracket_\gamma = \bullet$
- $\llbracket \Gamma \vdash \text{propext} \rrbracket_\gamma = \bullet$
- $\llbracket \Gamma \vdash \text{choice } \alpha h \rrbracket_\gamma = \varepsilon(\llbracket \Gamma \vdash \alpha \rrbracket_\gamma)$

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

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

6.1.2 Definition of acc in ZFC

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

Remark 2. If λ is an inaccessible cardinal, and $A \in V_\lambda$, $B : A \rightarrow V_\lambda$, 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.

6.2 Soundness

Fix a universe valuation v for this section.

Lemma 6.1 (Basics).

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

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

Theorem 6.2 (Soundness).

- If $\Gamma \vdash \alpha : \mathbb{U}_\ell$ where $\llbracket \ell \rrbracket = 0$, then $\llbracket \Gamma \vdash \alpha \rrbracket_\gamma \subseteq \{\bullet\}$.
- If $\Gamma \vdash e : \alpha$ where $\text{lvl}(\Gamma \vdash \alpha) = 0$, then $\llbracket \Gamma \vdash e \rrbracket_\gamma = \bullet$.
- If $\Gamma \vdash e : \alpha$, then there exists an k such that if the κ sequence is k -correct, then for all $\gamma \in \llbracket \Gamma \rrbracket$, $\llbracket \Gamma \vdash e \rrbracket_{v, \gamma} \in \llbracket \Gamma \vdash \alpha \rrbracket_{v, \gamma}$.
- If $\Gamma \vdash e \equiv e'$, then there exists an k such that if the κ sequence is k -correct, then for all $\gamma \in \llbracket \Gamma \rrbracket$, $\llbracket \Gamma \vdash e \rrbracket_{v, \gamma} = \llbracket \Gamma \vdash e' \rrbracket_{v, \gamma}$.

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 $\llbracket \alpha \rrbracket$ 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 $\llbracket \Gamma \vdash \alpha \rrbracket_\gamma$, $\bar{\beta}(x)$ means $\llbracket \Gamma, x : \alpha \vdash \beta \rrbracket_{(\gamma, x)}$, and so on, where Γ and γ are understood from context.

- **Weakening.** We have $\llbracket \Gamma \vdash e \rrbracket_\gamma \in \llbracket \Gamma \vdash \beta \rrbracket_\gamma$ by the IH, and $\llbracket \Gamma, x : \alpha \vdash e \rrbracket_{(\gamma, x)} = \llbracket \Gamma \vdash e \rrbracket_\gamma$ and $\llbracket \Gamma, x : \alpha \vdash \beta \rrbracket_{(\gamma, x)} = \llbracket \Gamma \vdash \beta \rrbracket_\gamma$ by the weakening lemma, so $\llbracket \Gamma, x : \alpha \vdash e \rrbracket_{(\gamma, x)} \in \llbracket \Gamma, x : \alpha \vdash \beta \rrbracket_{(\gamma, x)}$.
- **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 $\text{lvl}(\Gamma \vdash \alpha) = \text{lvl}(\Gamma \vdash \beta)$.
- **Variable.** $\llbracket \Gamma, x : \alpha \vdash x \rrbracket_{(\gamma, x)} = x$, so $(\gamma, x) \in \llbracket \Gamma, x : \alpha \rrbracket$ implies $x \in \llbracket \Gamma \vdash \alpha \rrbracket_\gamma = \llbracket \Gamma, x : \alpha \vdash \alpha \rrbracket_{(\gamma, x)}$ by the weakening lemma.
- **Universe.** $\llbracket \vdash U_\ell \rrbracket_{()} = U_{\llbracket \ell \rrbracket} \in U_{\llbracket \ell \rrbracket + 1} = \llbracket \vdash U_{S_\ell} \rrbracket_{()}$ since the U universes form a membership hierarchy. Parts 1 and 2 do not apply since $U_{S_\ell} \neq \mathbb{P}$.
- **Application.** Suppose $\Gamma \vdash e_1 : \forall x : \alpha. \beta$ and $\Gamma \vdash e_2 : \alpha$. Then $\Gamma, x : \alpha \vdash \beta$ **type**, so $n := \text{lvl}(\Gamma, x : \alpha \vdash \beta)$ is defined. By the IH, $\bar{e}_1 \in \prod_{x \in \bar{\alpha}}^n \bar{\beta}(x)$ and $\bar{e}_2 \in \bar{\alpha}$. The result type is $\text{lvl}(\Gamma \vdash \beta[e_2/x]) = n$, so part 2 only applies when $n = 0$.
 - If $n = 0$, then $\bar{e}_1 = \bullet \in \bigcap_{x \in \bar{\alpha}} \bar{\beta}(x)$, so in particular $\llbracket \Gamma \vdash e_1 e_2 \rrbracket_\gamma = \bullet \in \bar{\beta}(\bar{e}_2) = \llbracket \Gamma \vdash \beta[e_2/x] \rrbracket_\gamma$ by the substitution lemma.
 - If $n \neq 0$, then $\bar{e}_1 \in \prod_{x \in \bar{\alpha}} \bar{\beta}(x)$, so $\llbracket \Gamma \vdash e_1 e_2 \rrbracket_\gamma = \bar{e}_1(\bar{e}_2) \in \bar{\beta}(\bar{e}_2) = \llbracket \Gamma \vdash \beta[e_2/x] \rrbracket_\gamma$ by the substitution lemma.
- **Lambda.** Suppose $\Gamma, x : \alpha \vdash e : \beta$. Then $n := \text{lvl}(\Gamma, x : \alpha \vdash \beta)$ is defined. By the IH, $\bar{e}(x) \in \bar{\beta}(x)$ for all $x \in \bar{\alpha}$. The result type is $\text{lvl}(\Gamma \vdash \lambda x : \alpha. \beta) = \text{imax}(\text{lvl}(\Gamma \vdash \alpha), n)$ which is 0 iff n is, so part 2 only applies when $n = 0$.
 - If $n = 0$, then $\bar{e}(x) = \bullet \in \bar{\beta}(x)$ for all $x \in \bar{\alpha}$, so $\llbracket \Gamma \vdash \lambda x : \alpha. e \rrbracket_\gamma = \bullet \in \bigcap_{x \in \bar{\alpha}} \bar{\beta}(x)$.
 - If $n \neq 0$, then $\llbracket \Gamma \vdash \lambda x : \alpha. e \rrbracket_\gamma = (x \in \bar{\alpha} \mapsto \bar{e}(x)) \in \prod_{x \in \bar{\alpha}} \bar{\beta}(x)$.
- **Forall.** Suppose $\Gamma \vdash \alpha : U_{\ell_1}$ and $\Gamma, x : \alpha \vdash \beta : U_{\ell_2}$. Then $n := \text{lvl}(\Gamma, x : \alpha \vdash \beta) = \llbracket \ell_2 \rrbracket$. By the IH, $\bar{\alpha} \in U_{\llbracket \ell_1 \rrbracket}$ and $\bar{\beta}(x) \in U_n$ for all $x \in \bar{\alpha}$. The result type is a universe so part 2 does not apply, and part 1 applies only when $n = 0$.
 - If $n = 0$, then $\llbracket \Gamma \vdash \forall x : \alpha. \beta \rrbracket_\gamma = \{\bullet\} \cap \bigcap_{x \in \bar{\alpha}} \bar{\beta}(x) \subseteq \{\bullet\}$.
 - If $n \neq 0$, then $\llbracket \Gamma \vdash \forall x : \alpha. \beta \rrbracket_\gamma = \prod_{x \in \bar{\alpha}} \bar{\beta}(x) \in U_{\max(\llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)}$, provided that the κ sequence is $\max(\llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)$ -correct, because if $\kappa_{\max(\llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket) - 1}$ is inaccessible then $U_{\max(\llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)}$ is closed under dependent products.
- \perp : $\llbracket \vdash \perp \rrbracket_{()} = \emptyset \subseteq \{\bullet\}$.
- rec_\perp : $\llbracket \Gamma \vdash \text{rec}_\perp^C \rrbracket_\gamma = \emptyset \in \llbracket \Gamma \vdash \perp \rightarrow C \rrbracket_\gamma = \prod_{x \in \emptyset} \bar{C}$.
- Σ : Assuming the κ sequence is k -correct where $k = \max(1, \llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)$, if $\bar{\alpha} \in U_{\llbracket \ell_1 \rrbracket} \subseteq U_k$ and $\bar{\beta}(x) \in U_{\llbracket \ell_2 \rrbracket} \subseteq U_k$ 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 $\bar{e}_2 \in \llbracket \Gamma \vdash \beta[e_1/x] \rrbracket_\gamma = \bar{\beta}(\bar{e}_1)$, then $\llbracket \Gamma \vdash (e_1, e_2) \rrbracket_\gamma = (\bar{e}_1, \bar{e}_2) \in \sum_{x \in \bar{\alpha}} \bar{\beta}(x)$.
- π_1 : If $\bar{e} \in \sum_{x \in \bar{\alpha}} \bar{\beta}(x)$, then $\llbracket \Gamma \vdash \pi_1 e \rrbracket_\gamma = \pi_1(\bar{e}) \in \bar{\alpha}$.
- π_2 : If $\bar{e} \in \sum_{x \in \bar{\alpha}} \bar{\beta}(x)$, then $\llbracket \Gamma \vdash \pi_2 e \rrbracket_\gamma = \pi_2(\bar{e}) \in \bar{\beta}(\pi_1(\bar{e})) = \bar{\beta}(\llbracket \Gamma \vdash \pi_1 e \rrbracket_\gamma) = \llbracket \Gamma \vdash \beta[\pi_1 e/x] \rrbracket_\gamma$.
- $+$: If $k := \max(1, \llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)$, and $\bar{\alpha} \in U_{\llbracket \ell_1 \rrbracket} \subseteq U_k$ and $\bar{\beta} \in U_{\llbracket \ell_2 \rrbracket} \subseteq U_k$, then $\llbracket \Gamma \vdash \alpha + \beta \rrbracket_\gamma = \bar{\alpha} \sqcup \bar{\beta} \in U_k$, because $\text{rank}(\bar{\alpha} \sqcup \bar{\beta}) \leq \max(\text{rank } \bar{\alpha}, \text{rank } \bar{\beta}) + 2$ (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 $\llbracket \Gamma \vdash \text{inl } e \rrbracket_\gamma = \iota_1(\bar{e}) \in \bar{\alpha} \sqcup \bar{\beta} = \llbracket \Gamma \vdash \alpha + \beta \rrbracket_\gamma$. (We don't need the second IH.) **inr** is similar.

- rec_+ : By the IH, $\bar{C} : \bar{\alpha} \sqcup \bar{\beta} \rightarrow U_{[\ell]}$.
 $\llbracket \Gamma \vdash \text{rec}_\perp^C a \ b \rrbracket_\gamma \in \prod_{x \in \bar{\alpha} \sqcup \bar{\beta}} \bar{C}(x)$ because it was defined as a function such that $\llbracket \Gamma \vdash \text{rec}_\perp^C a \ b \rrbracket_\gamma(\iota_1(x)) = \bar{a}(x) \in \bar{C}(\iota_1(x))$, and $\llbracket \Gamma \vdash \text{rec}_\perp^C a \ b \rrbracket_\gamma(\iota_2(x)) = \bar{b}(x) \in \bar{C}(\iota_2(x))$.
- ulift : If $\bar{\alpha} \in U_{[\ell]}$ and $\ell \leq \ell'$, then $\llbracket \ell \rrbracket \leq \llbracket \ell' \rrbracket$ so $\llbracket \Gamma \vdash \text{ulift}_\ell^{\ell'} \alpha \rrbracket_\gamma = \bar{\alpha} \in U_{[\ell]} \subseteq U_{[\ell']}$.
- \uparrow and \downarrow are trivial from the IH.
- $\|\cdot\|$: $\llbracket \Gamma \vdash \|\alpha\| \rrbracket_\gamma = [\bar{\alpha} \neq \emptyset] \subseteq \{\bullet\}$ (we don't need the IH).
- $|\cdot|$: If $\bar{e} \in \bar{\alpha}$, then $\bar{\alpha} \neq \emptyset$ so $\llbracket \Gamma \vdash |e| \rrbracket_\gamma = \bullet \in [\bar{\alpha} \neq \emptyset] = \llbracket \Gamma \vdash \|\alpha\| \rrbracket_\gamma$.
- rec_\parallel : To show $\llbracket \Gamma \vdash \text{rec}_\parallel f \rrbracket_\gamma = \bullet \in \llbracket \Gamma \vdash \|\alpha\| \rightarrow C \rrbracket_\gamma$, 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 $k = \max(1, \llbracket \ell_1 \rrbracket, \llbracket \ell_2 \rrbracket)$, since we have already observed that V_λ where λ is inaccessible is closed under W -types.
- sup : $\llbracket \Gamma \vdash \text{sup } a \ f \rrbracket_\gamma = (\bar{a}, \bar{f}) \in W_{x \in \bar{\alpha}, \bar{\beta}}(x)$ since $\bar{a} \in \bar{\alpha}$ and $\bar{f} : \bar{\beta}(\bar{a}) \rightarrow W_{x \in \bar{\alpha}, \bar{\beta}}(x)$. (Application of the definition, IH, and substitution theorem.)
- rec_W : Let $W := \llbracket \Gamma \vdash Wx : \alpha. \beta \rrbracket_\gamma = W_{x \in \bar{\alpha}} \bar{\beta}(x)$. By the IH and applying the definitions,

$$\bar{e} := \bar{e} \in \prod_{a \in \bar{\alpha}} \prod_{f : \bar{\beta}(a) \rightarrow W} \left[\prod_{b \in \bar{\beta}(a)} \bar{C}(f(b)) \right] \rightarrow \bar{C}(a, f).$$

Now $\llbracket \Gamma \vdash \text{rec}_W e \rrbracket_\gamma = \text{rec}_W(W, \bar{e}) =: F$ 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 $f : \bar{\beta}(a) \rightarrow W$ 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: $\llbracket \Gamma \vdash a = b \rrbracket_\gamma = [\bar{a} = \bar{b}] \subseteq \{\bullet\}$ (we don't need the IH).
- refl : $\llbracket \Gamma \vdash \text{refl } a \rrbracket_\gamma = \bullet \in \llbracket \Gamma \vdash a = a \rrbracket_\gamma = [\bar{a} = \bar{a}]$ because $\bar{a} = \bar{a}$ (we don't need the IH).
- $\text{rec}_=$: We have $\bar{a} \in \bar{\alpha}$, $\bar{C} : \bar{\alpha} \rightarrow U_{[\ell]}$, and $\bar{e} \in \bar{C}(\bar{a})$ from the IH. To show $\llbracket \text{rec}_= e \rrbracket_\gamma \in \llbracket \forall b : \alpha. a = b \rightarrow C \ b \rrbracket_\gamma = \prod_{b \in \bar{\alpha}} \prod_{h \in [\bar{a} = b]} \bar{C}(b)$, suppose $b \in \bar{\alpha}$ and $h \in [\bar{a} = b]$. Then $\bar{a} = b$, so $\llbracket \text{rec}_= e \rrbracket_\gamma(b)(h) = \bar{e} \in \bar{C}(\bar{a}) = \bar{C}(b)$.
- acc : If $\bar{r} : \bar{\alpha} \rightarrow \bar{\alpha} \rightarrow U_0$ and $\bar{x} \in \bar{\alpha}$, then $\llbracket \Gamma \vdash \text{acc}_r x \rrbracket_\gamma = [x \in \text{acc}(\bar{\alpha}, \bar{r})] \subseteq \{\bullet\}$.
- $\text{intro}_{\text{acc}}$: If $\bar{x} : \bar{\alpha}$ and $\bar{f} = \bullet \in \bigcap_{y \in \bar{\alpha}} [\bullet \in (y, \bar{x}) \in \bar{r}] \rightarrow [y \in \text{acc}(\bar{\alpha}, \bar{r})]$ (applying the definitions and IH), then for all $y \in \bar{\alpha}$ with $(y, \bar{x}) \in \bar{r}$, $y \in \text{acc}(\bar{\alpha}, \bar{r})$, so \bar{x} is \bar{r} -accessible, i.e. $\bar{x} \in \text{acc}(\bar{\alpha}, \bar{r})$. Thus, $\llbracket \Gamma \vdash \text{intro}_{\text{acc}} f \ x \rrbracket_\gamma = \bullet \in \llbracket \Gamma \vdash \text{acc}_r x \rrbracket_\gamma = [\bar{x} \in \text{acc}(\bar{\alpha}, \bar{r})]$.
- rec_{acc} : We have $\bar{C} : \bar{\alpha} \rightarrow U_{[\ell]}$, and

$$\bar{e} \in \prod_{x \in \bar{\alpha}} [\forall y \in \bar{\alpha}, (y, x) \in \bar{r} \rightarrow y \in \text{acc}(\bar{\alpha}, \bar{r})] \rightarrow \left(\prod_{y \in \bar{\alpha}} [(y, x) \in \bar{r}] \rightarrow \bar{C}(y) \right) \rightarrow \bar{C}(x).$$

We want to show $\llbracket \text{rec}_{\text{acc}} e \rrbracket_\gamma \in \prod_{x \in \bar{\alpha}} [x \in \text{acc}(\bar{\alpha}, \bar{r})] \rightarrow \bar{C}(x)$. It was defined as a function on this domain, so let $x \in \bar{\alpha}$ and $h \in [x \in \text{acc}(\bar{\alpha}, \bar{r})]$; then $x \in \text{acc}(\bar{\alpha}, \bar{r})$ and $h = \bullet$, and $\llbracket \text{rec}_{\text{acc}} e \rrbracket_\gamma(x)(h) = F(x)$ for the function F in the definition of rec_{acc} . We prove that $F(x) \in \bar{C}(x)$ by induction on $x \in \text{acc}(\bar{\alpha}, \bar{r})$.

Suppose that for all $y \in \text{acc}(\bar{\alpha}, \bar{r})$, 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, $(y \in \bar{\alpha} \mapsto (h \in [(y, x) \in \bar{r}] \mapsto F(y))) \in \prod_{y \in \bar{\alpha}} [(y, x) \in \bar{r}] \rightarrow \bar{C}(y)$ 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 $\Gamma \vdash \alpha : \mathbf{U}_\ell$, $n := \llbracket \ell \rrbracket$, and $\Gamma \vdash r : \alpha \rightarrow \alpha \rightarrow \mathbb{P}$. Let \sim be the equivalence closure of \bar{r} . (We will assume this for the next few cases to do with quotients.)
 - If $n = 0$, then $\llbracket \Gamma \vdash \alpha/r \rrbracket_\gamma = \bar{\alpha} \subseteq \{\bullet\}$.
 - If $n \neq 0$, then $\llbracket \Gamma \vdash \alpha/r \rrbracket_\gamma = \bar{\alpha}/\sim \in U_n$ because $\bar{\alpha}/\sim$ is contained in the double powerset of $\bar{\alpha} \in \mathbf{U}_n$.
- **mk:** Suppose additionally $\Gamma \vdash x : \alpha$, so $\bar{x} \in \bar{\alpha}$ from the IH.
 - If $n = 0$, then $\llbracket \Gamma \vdash \mathbf{mk}_r x \rrbracket_\gamma = \bullet = \bar{x} \in \bar{\alpha} = \llbracket \Gamma \vdash \alpha/r \rrbracket_\gamma$.
 - If $n \neq 0$, then $\llbracket \Gamma \vdash \mathbf{mk}_r x \rrbracket_\gamma = [\bar{x}]_\sim \in \bar{\alpha}/\sim = \llbracket \Gamma \vdash \alpha/r \rrbracket_\gamma$.
- **lift:** Suppose $\Gamma \vdash \beta : \mathbf{U}_{\ell'}$ with $\llbracket \ell' \rrbracket \geq 1$, and $\Gamma \vdash f : \alpha \rightarrow \beta$, and $\Gamma \vdash h : \forall x y : \alpha. R x y \rightarrow f x = f y$.
 - If $n = 0$, then $\llbracket \Gamma \vdash \mathbf{lift}_r \beta f h \rrbracket_\gamma = \bar{f} \in \llbracket \Gamma \vdash \alpha/r \rightarrow \beta \rrbracket_\gamma = \prod_{x \in \bar{\alpha}} \bar{\beta}$ by the IH.
 - If $n \neq 0$, then from the IH $\bar{f} : \bar{\alpha} \rightarrow \bar{\beta}$ and

$$\bar{h} \in \bigcap_{x \in \bar{\alpha}} \bigcap_{y \in \bar{\alpha}} [(x, y) \in \bar{r} \rightarrow \bar{f}(x) = \bar{f}(y)].$$

Therefore $\forall x, y \in \bar{\alpha}. (x, y) \in \bar{r} \rightarrow \bar{f}(x) = \bar{f}(y)$, so since the property $\bar{f}(x) = \bar{f}(y)$ is an equivalence relation that contains \bar{r} , we have $x \sim y \rightarrow \bar{f}(x) = \bar{f}(y)$, so there is a well defined function $F : \bar{\alpha}/\sim \rightarrow \bar{\beta}$ such that $F([x]_\sim) = \bar{f}(x)$, and $\llbracket \Gamma \vdash \mathbf{lift}_r \beta f h \rrbracket_\gamma$ was defined to be this function. Thus $\llbracket \Gamma \vdash \mathbf{lift}_r \beta f h \rrbracket_\gamma \in \llbracket \Gamma \vdash \alpha/r \rightarrow \beta \rrbracket_\gamma$.

- **sound:** We want to verify that $\llbracket \Gamma \vdash \mathbf{sound}_r \rrbracket_\gamma = \bullet \in \llbracket \Gamma \vdash \forall x y : \alpha. r x y \rightarrow \mathbf{mk}_r x = \mathbf{mk}_r y \rrbracket_\gamma$, or after expansion,

$$\bullet \in \bigcap_{x \in \bar{\alpha}} \bigcap_{y \in \bar{\alpha}} [(x, y) \in \bar{r} \rightarrow [x]_\sim = [y]_\sim].$$

Let $x, y \in \bar{\alpha}$, and suppose $(x, y) \in \bar{r}$; then since \sim contains \bar{r} , $x \sim y$ and hence $[x]_\sim = [y]_\sim$.

- **propext:** We want to verify that $\llbracket \vdash \mathbf{propext} \rrbracket_{()} = \bullet \in \llbracket \vdash \forall p q : \mathbb{P}. (p \leftrightarrow q) \rightarrow p = q \rrbracket_{()}$. 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 $\Gamma \vdash \alpha : \mathbf{U}_\ell$ and $\Gamma \vdash h : \|\alpha\|$. Then $\bar{h} \in [\bar{\alpha} \neq \emptyset]$, so $\bar{\alpha} \neq \emptyset$, and $\bar{\alpha} \in U_{\llbracket \ell \rrbracket} \subseteq U_\omega$, so since ε is a choice function on U_ω , $\llbracket \Gamma \vdash \mathbf{choice} \alpha h \rrbracket_\gamma = \varepsilon(\bar{\alpha}) \in \bar{\alpha}$.

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 $\llbracket \Gamma \vdash \alpha + \beta \rrbracket_\gamma$ is defined only in terms of $\llbracket \Gamma \vdash \alpha \rrbracket_\gamma$ and $\llbracket \Gamma \vdash \beta \rrbracket_\gamma$. When a case split on $\llbracket \ell \rrbracket = 0$ is done, by unique typing it must be the same for both sides (since e and e' have the same 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 $\bar{e}' \in \bar{\alpha}$. Let $n = \text{lvl}(\Gamma, x : \alpha \vdash \beta)$.
 - If $n = 0$, then $\llbracket \Gamma \vdash (\lambda x : \alpha. e) e' \rrbracket_\gamma = \bullet = \llbracket \Gamma \vdash e[e'/x] \rrbracket_\gamma$ because $e[e'/x]$ is a proof (by part 2).
 - If $n \neq 0$, then $\llbracket \Gamma \vdash (\lambda x : \alpha. e) e' \rrbracket_\gamma = (x \in \bar{\alpha} \mapsto \bar{e}(x))(\bar{e}') = \bar{e}(\bar{e}') = \llbracket \Gamma \vdash e[e'/x] \rrbracket_\gamma$ by the substitution lemma.
- **Eta.** Suppose $\Gamma \vdash e : \forall y : \alpha. \beta$, so that by the inductive hypothesis $\bar{e} \in \prod_{y \in \bar{\alpha}} \bar{\beta}(y)$ where $n := \text{lvl}(\Gamma, x : \alpha \vdash \beta)$.

- If $n = 0$, then $\llbracket \Gamma \vdash \lambda x : \alpha. e \ x \rrbracket_\gamma = \bullet = \llbracket \Gamma \vdash e \rrbracket_\gamma$ because e is a proof.
- If $n \neq 0$, then $\llbracket \Gamma \vdash \lambda x : \alpha. e \ x \rrbracket_\gamma = (x \in \bar{\alpha} \mapsto \bar{e}(x)) = \bar{e}$ by function extensionality in ZFC.
- Proof irrelevance. If $\Gamma \vdash h, h' : p : \mathbb{P}$, then by part 2 of the theorem, $\llbracket \Gamma \vdash h \rrbracket_\gamma = \bullet = \llbracket \Gamma \vdash h' \rrbracket_\gamma$.
- Delta. If $\text{def } c : \alpha := e$, then $\llbracket \Gamma \vdash c \rrbracket_\gamma = \llbracket \Gamma \vdash e \rrbracket_\gamma$ by definition.
- Zeta. If $\text{def } c : \alpha := e$, then $\llbracket \Gamma \vdash \text{let } x : \alpha := e_1 \text{ in } e_2 \rrbracket_\gamma = \llbracket \Gamma \vdash e_2[e_1/x] \rrbracket_\gamma$ 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. Suppose

$$\begin{array}{ll}
\Gamma \vdash \alpha : \mathbf{U}_\ell & \bar{\alpha} \in U[\ell] \\
\Gamma \vdash r : \alpha \rightarrow \alpha \rightarrow \mathbb{P} & \bar{r} \subseteq \bar{\alpha} \times \bar{\alpha} \\
\Gamma \vdash \beta : \mathbf{U}_{\ell'} & \bar{\beta} \in U[\ell'] \\
\Gamma \vdash f : \alpha \rightarrow \beta & \bar{f} : \bar{\alpha} \rightarrow \bar{\beta} \\
\Gamma \vdash h : \forall x y : \alpha. r \ x \ y \rightarrow f \ x = f \ y & \bar{h} \in [\forall x, y \in \bar{\alpha}. (x, y) \in \bar{r} \rightarrow \bar{f}(x) = \bar{f}(y)] \\
\Gamma \vdash a : \alpha & \bar{a} \in \bar{\alpha}
\end{array}$$

so that the statements on the right are the IH. We want to show that $\llbracket \Gamma \vdash \text{lift}_r \beta \ f \ h \ (\text{mk}_r \ a) \rrbracket_\gamma = \llbracket \Gamma \vdash f \ a \rrbracket_\gamma$, or $\llbracket \Gamma \vdash \text{lift}_r \beta \ f \ h \rrbracket_\gamma([a]_\sim) = \bar{f}(a)$; but this is by definition (we showed it is well defined given the assumptions on α, r, β, f, h already).

- π_1 iota. $\llbracket \Gamma \vdash \pi_1(a, b) \rrbracket_\gamma = \pi_1(\bar{a}, \bar{b}) = \bar{a}$.
- π_2 iota. $\llbracket \Gamma \vdash \pi_2(a, b) \rrbracket_\gamma = \pi_2(\bar{a}, \bar{b}) = \bar{b}$.
- inl iota. $\llbracket \Gamma \vdash \text{rec}_+ a \ b \ (\text{inl } x) \rrbracket_\gamma = \llbracket \Gamma \vdash \text{rec}_+ a \ b \rrbracket_\gamma(\iota_1(\bar{x})) = \bar{a}(\bar{x}) = \llbracket \Gamma \vdash a \ x \rrbracket_\gamma$.
- inr iota. $\llbracket \Gamma \vdash \text{rec}_+ a \ b \ (\text{inr } x) \rrbracket_\gamma = \llbracket \Gamma \vdash \text{rec}_+ a \ b \rrbracket_\gamma(\iota_2(\bar{x})) = \bar{b}(\bar{x}) = \llbracket \Gamma \vdash b \ x \rrbracket_\gamma$.
- ulift iota. $\llbracket \Gamma \vdash \downarrow \uparrow x \rrbracket_\gamma = \llbracket \Gamma \vdash x \rrbracket_\gamma$ by definition.
- \mathbf{W} iota. Letting $F := \text{rec}_W(W_{x \in \bar{\alpha}} \bar{\beta}(x), \bar{e})$, we have $\llbracket \Gamma \vdash \text{rec}_W e \ (\sup a \ f) \rrbracket_\gamma = F(\bar{a}, \bar{f}) = \bar{e}(\bar{a})(\bar{f})(F \circ \bar{f})$ on the one hand, and $\llbracket \Gamma \vdash e \ a \ f \ (\lambda b : \beta[a/x]. \text{rec}_W e \ (f \ b)) \rrbracket_\gamma = \bar{e}(\bar{a})(\bar{f})(b \in \bar{\beta}(\bar{a}) \mapsto F(f(b)))$ 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. $\llbracket \Gamma \vdash \text{rec}_= e \ a \ h \rrbracket_\gamma = \bar{e}$ by definition.
- acc iota. If $F : \text{acc}(\bar{\alpha}, \bar{r}) \rightarrow V$ is the function defined in $\text{rec}_{\text{acc}}(\bar{\alpha}, \bar{r}, \bar{e})$, then we have

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

where $\bar{f} = \bullet$ because f is a proof.

- ulift eta. $\llbracket \Gamma \vdash \uparrow \downarrow x \rrbracket_\gamma = \llbracket \Gamma \vdash x \rrbracket_\gamma$ 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 $\llbracket \Gamma \vdash (\pi_1 \ p, \pi_2 \ p) \rrbracket_\gamma = (\pi_1(\bar{p}), \pi_2(\bar{p})) = (x, y) = \bar{p}$.

□

Corollary 6.3. *Lean is consistent if $\text{ZFC} + \{\text{there are } n \text{ inaccessible cardinals} \mid n \in \omega\}$ is. That is, there is no proof of \perp that is verified by the Lean kernel.*

Proof. Suppose $\Vdash e : \perp$ (the algorithmic typing judgment). Then $\vdash e : \perp$ since algorithmic equality implies definitional equality. Let v be the universe valuation that sets every variable to 0, 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 6.2](#). Then $\llbracket \vdash e \rrbracket_{()} \in \llbracket \vdash \perp \rrbracket_{()} = \emptyset$, a contradiction. \square

7 Strong normalization

7.1 Evaluation in the empty context

In [section 4.2](#), we set up a reduction relation \rightsquigarrow_κ which is plainly useless for a normalization algorithm, because of the following rule:

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

Recall that v is a sequence of lambdas $v_i = \lambda x :: \xi_i$. $\text{rec}_P(C, e, \pi_i[\text{inv}[p, h], x], u_i x)$. That is, we reduce any rec_P term, regardless of its arguments, to a longer expression that contains another rec_P term (assuming that the inductive type has a recursive argument). So this is guaranteed not to terminate, but it was sufficiently powerful to act as a semi-decision procedure for definitional equality where existing methods fail (by giving up early, they recover decidability at the cost of completeness).

But our examples thus far of undecidability of definitional equality all work in an inconsistent context. Is this necessary? What if the context is known to be inhabited by some terms? In this case we may as well substitute the terms in to reduce the problem to evaluation in the empty context.

That is, we wish to determine if $\vdash e \equiv e'$, hopefully by a method similar to the reduction \rightsquigarrow_κ , but without the self-destructive unfolding K rule.