Metatheory of LF Extended with Dependent Pair and Unit Types

Susmit Sarkar
2005
CMU-CS-05-179

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We study an extension of the well known LF type theory with dependent pair and unit types. The important metatheoretic properties proved in this study are decidability of type checking and the existence of canonical forms. The study follows standard techniques introduced by Harper and Pfenning [5] together with extensions studied by Vanderwaart and Crary [7].

This material is based on work supported by NSF grant CCR-0121633. Any opinions, findings, and conclusions or recommendations in this publication are those of the author and do not reflect the views of this agency.
Keywords: Logical Frameworks, Type Theory
1 Introduction

The dependently typed lambda calculus LF was introduced by Harper, Honsell and Plotkin [4] to serve as a logical framework. This language has since been used extensively to represent logics. This calculus can be described as a dependently-typed lambda calculus, with three levels: objects, type families and kinds. Both type families and kinds can depend on objects.

Our goal is to extend a functional language such as ML to be able to talk statically about logics represented in LF [6]. One key problem in this scenario is to deal with open terms. We have argued in earlier work [6] that we need only represent closed terms if we reify contexts within the language, and reify open terms as terms abstracted over the reified contexts. Our representation of contexts is as products of the types in the context, in a manner similar to the “telescopes” of deBruijn [3]. Since terms in the context can depend on terms abstracted over earlier in the context, we in fact require dependent pair types, also known as Σ-types. The null context is reified as the unit type, with a single canonical inhabitant.

In this paper, we will take a first step towards such a language. We will concentrate on the metatheory of the extended LF sketched out above. The current work studies LF extended with Σ types and unit, which we shall call LFΣ,

A crucial property required of a logical framework is the existence of canonical forms. The “adequacy theorem” which says that the representation is meaningful establishes a bijection between the propositions and proofs in the represented logic and canonical forms of certain types. Canonical forms are taken to be β-normal, η-long forms. A second important property we would like a logical framework to have is decidability of checking proofs. In LF, this reduces to being able to decide type checking, since proofs are represented as terms and judgments as types.

These properties have been studied before for related languages. We will summarize the previous contributions briefly. The first presentation of LF [4] relied on β conversion as the notion of definitional equality, leaving out the technically complex η conversion entirely. This posed a problem with the method sketched out above, since canonical forms are η-long forms. Subsequently, Harper and Pfenning [5] reformulated LF to make the definitional equality a typed judgment form instead of an untyped reduction scheme. This definitional equality compared objects at a specified type and in a specified context, and axiomatized a congruence relation generated by the βη convertibility of well typed terms. With this innovation, a new algorithm to decide type checking could be produced. The key component of a type-checking algorithm is an algorithm to decide definitional equality. They produced such an algorithm which is directed by the types at which objects are being compared. The algorithm works similarly to one by Coquand [1], except that comparisons is directed by types and not by the shape of terms. The novel type-directed algorithm was proved correct using the so-called method of logical relations. For technical reasons, Harper and Pfenning did not treat abstraction at the level of type families. Later, Vanderwaart and Crary [7] extended Harper and Pfenning’s method to the Linear Logical Framework LLF. They provided a separate argument to prove a necessary property of family level abstractions in this setting.

We will follow Harper and Pfenning [5] and Vanderwaart and Crary [7], and extend their equality algorithm to our language. As mentioned already, this algorithm is directed by the type at which objects are being compared. To avoid technical problems with proofs, it erases all dependencies from the type at which the comparison is done. The algorithm itself is simple. The main bulk of the work is to prove this algorithm sound and complete with respect to definitional equality. Completeness requires a complex proof based on a Kripke logical relation to make the inductive case go through. This method is described for a much simpler language by Crary [2].

Our study is a rather direct extension of Vanderwaart and Crary’s work, extended with a dependent pair and a unit type. For readers familiar with that work, our sequence of lemmas and theorems will be familiar. Our work also differs in the minor way that canonical forms is proved directly for terms in the language. Harper and Pfenning define a related but different notion called quasi-canonical forms, which do not syntactically belong to the language. Vanderwaart and Crary defer to their work, and do not prove the canonical forms property.


1.1 Overview

In section 2 we present the abstract syntax of our language. We define the typing and definitional equality judgments in section 3. Some elementary structural properties we require are proved in section 4. In particular, we will need regularity, i.e. terms appearing in typing judgments are well-formed. We will also need a property called functionality of substitutions, i.e. equal substitutions map equal arguments to equal arguments. Another property we need is called injectivity, which says that equal products and sums have equal components. This property is impossible to prove by simple induction on the equality judgment in the presence of the non-trivial equality induced by the presence of family-level abstractions. We follow Vanderwaart and Crary in proving this by means of a logical relation in section 5.

Next, we produce an algorithm in section 6 for deciding definitional equality. The completeness of this algorithm with respect to definitional equality is proved in section 7. We prove both soundness and existence of canonical forms together, in section 8. Finally in section 9 we extend the algorithm for equality to a sound and complete algorithm for typechecking. We prove decidability for all judgment forms.

2 Abstract Syntax

We start with presenting the abstract syntax of our language. This is generated by the following grammar.

\[
\begin{align*}
\text{Kinds} & \quad K ::= \text{type} \quad \text{kind of types} \\
 & \quad \quad \Pi \times A.K \quad \text{dependent product kind} \\
\text{Families} & \quad A ::= a \quad \text{family constants} \\
 & \quad \quad \lambda x:A_1.A_2 \quad \text{family level abstraction} \\
 & \quad \quad AM \quad \text{family application} \\
 & \quad \quad \Pi x:A_1.A_2 \quad \text{family of functions} \\
 & \quad \quad \Sigma x:A_1.A_2 \quad \text{family of products} \\
 & \quad \quad 1 \quad \text{unit type} \\
\text{Objects} & \quad M ::= c \quad \text{object constants} \\
 & \quad \quad x \quad \text{object variables} \\
 & \quad \quad \lambda x:A.M \quad \text{object functions} \\
 & \quad \quad M_1.M_2 \quad \text{object level application} \\
 & \quad \quad \langle M_1, M_2 \rangle^A \quad \text{pairs of objects} \\
 & \quad \quad \pi_i M \ (i = 1, 2) \quad \text{projections from pairs} \\
 & \quad \quad \langle \rangle \quad \text{unit object} \\
\text{Signatures} & \quad \Sigma ::= \cdot \quad \text{empty} \\
 & \quad \quad \Sigma, a:A \quad \text{extension by family level constant} \\
 & \quad \quad \Sigma, c:A \quad \text{extension by object level constant} \\
\text{Contexts} & \quad \Gamma ::= \cdot \quad \text{empty} \\
 & \quad \quad \Gamma, x:A \quad \text{context extension} \\
\end{align*}
\]

There are three levels of terms: objects, type families and kinds. Kinds classify families, and families belonging to kind type are called types. Types classify objects. The family level includes \( \Pi \) and \( \Sigma \) types, a unit type, as well as family-level abstractions and applications. We have object and family-level constants. A signature records the types and kinds assigned to these constants. We have object-level variables, and these are provided types by contexts. The other object level constructs are abstractions and applications, pairs and projections, and a canonical inhabitant of the unit type. Contexts, which assign types to variables, can be assumed to be ordered lists. Ordering is important because of dependencies. We assume that a variable is not bound more than once in contexts. This can always be assured in the standard way in all judgments.

We define a partial order on contexts syntactically. \( \Gamma_1 \leq \Gamma_2 \) holds if \( \Gamma_1(x) = \Gamma_2(x) \) for every \( x \in \text{Dom}(\Gamma_1) \). Thus if \( \Gamma_1 \leq \Gamma_2 \) then \( \text{Dom}(\Gamma_1) \subseteq \text{Dom}(\Gamma_2) \), and \( \Gamma_1 \) appears as a subsequence of \( \Gamma_2 \).

\[
\begin{align*}
\text{Substitutions} & \quad \sigma ::= \cdot \quad \text{empty} \\
 & \quad \quad \sigma, M/x \quad \text{cons}
\end{align*}
\]

2
Substitutions are finite maps from object variables to objects. They are defined as substituting for all variables in their domain simultaneously. We assume all object variables occurring in substitutions are distinct. We write id$_\Gamma$ for the substitution which is identity on all variables in the domain of $\Gamma$. The result of applying substitutions on objects, families and kinds is written as $M[\sigma]$, $A[\sigma]$ and $K[\sigma]$, and this notation is extended to all judgments $J$ of the theory.

3 Static Semantics

3.1 Judgment Forms

The static semantics is presented in the form of 8 mutually recursive judgments, whose meanings are explained below.

\[
\begin{align*}
\vdash \Sigma : \text{sig} & \quad \Sigma \text{ is a valid signature} \\
\vdash \Gamma : \text{ctx} & \quad \Gamma \text{ is a valid context} \\
\Gamma \vdash M : A & \quad M \text{ has type } A \\
\Gamma \vdash A : K & \quad A \text{ has kind } K \\
\Gamma \vdash K : \text{kind} & \quad K \text{ is a valid kind} \\
\Gamma \vdash M_1 = M_2 : A & \quad M_1 \text{ equals } M_2 \text{ at type } A \\
\Gamma \vdash A_1 = A_2 : K & \quad A_1 \text{ equals } A_2 \text{ at kind } K \\
\Gamma \vdash K_1 = K_2 : \text{kind} & \quad K_1 \text{ equals } K_2
\end{align*}
\]

The last three judgments define a typed definitional equality judgment. These equate terms at a particular type, families at particular kinds, and kinds.

3.2 Inference Rules

Recall that signatures assign types and kinds to constants. The first judgment ensures that such types and kinds are well-formed.

\[
\begin{align*}
\vdash \Sigma : \text{sig}
\end{align*}
\]

Empty

\[
\begin{align*}
\vdash \Sigma : \text{sig} \\
\vdash \Sigma, a : K : \text{kind}
\end{align*}
\]

Family Constant

\[
\begin{align*}
\vdash \Sigma : \text{sig} \\
\vdash \Sigma, a : A : \text{type}
\end{align*}
\]

Object Constant

From now on we assume fixed a valid signature $\Sigma$ and omit it from the judgments. All further judgments assume this signature to be fixed.

\[
\begin{align*}
\vdash \Gamma : \text{ctx}
\end{align*}
\]

Empty

\[
\begin{align*}
\vdash \Gamma : \text{ctx} \\
\vdash \Gamma, x : A : \text{ctx}
\end{align*}
\]

Cons
\[ \Gamma \vdash M : A \]

**Variables**

\[
\Gamma(x) = A \\
\Gamma \vdash x : A
\]

**Constants**

\[
\Sigma(c) = A \\
\Gamma \vdash c : A
\]

**Applications**

\[
\Gamma \vdash M_1 : \Pi x : A_2. A_1 \\
\Gamma \vdash M_2 : A_2 \\
\Gamma \vdash M_1 M_2 : A_1 [M_2/x]
\]

**Abstractions**

\[
\Gamma \vdash A_1 : \text{type} \\
\Gamma, x : A_1 \vdash M_2 : A_2 \\
\Gamma \vdash \lambda x : A_1. M_2 : \Pi x : A_1. A_2
\]

**Pairs**

\[
\Gamma \vdash \Sigma x : A_1. A_2 : \text{type} \\
\Gamma \vdash M_1 : A_1 \\
\Gamma \vdash M_2 : A_2 [M_1/x]
\]

\[
\Gamma \vdash (M_1, M_2)^\Sigma x : A_1. A_2 : \Sigma x : A_1. A_2
\]

**Projections**

\[
\Gamma \vdash M : \Sigma x : A_1. A_2 \\
\Gamma \vdash \pi_1 M : A_1 \\
\Gamma \vdash M : \Sigma x : A_1. A_2 \\
\Gamma \vdash \pi_2 M : A_2 [\pi_1 M/x]
\]

**Unit**

\[
\Gamma \vdash \langle \rangle : 1
\]

**Type Conversion**

\[
\Gamma \vdash M : A_1 \\
\Gamma \vdash A_1 = A_2 : \text{type} \\
\Gamma \vdash M : A_2
\]

\[ \Gamma \vdash A : K \]

**Constants**

\[
\Sigma(a) = K \\
\Gamma \vdash a : K
\]

**Abstractions**

\[
\Gamma \vdash A_1 : \text{type} \\
\Gamma, x : A_1 \vdash A_2 : K \\
\Gamma \vdash \lambda x : A_1. A_2 : \Pi x : A_1. K
\]

**Applications**

\[
\Gamma \vdash A_1 : \Pi x : A_2. K \\
\Gamma \vdash M : A_2 \\
\Gamma \vdash A_1 M : K [M/x]
\]

**Dependent Function Types**

\[
\Gamma \vdash A_1 : \text{type} \\
\Gamma, x : A_1 \vdash A_2 : \text{type} \\
\Gamma \vdash \Pi x : A_1. A_2 : \text{type}
\]

**Dependent Pair Types**

\[
\Gamma \vdash A_1 : \text{type} \\
\Gamma, x : A_1 \vdash A_2 : \text{type} \\
\Gamma \vdash \Sigma x : A_1. A_2 : \text{type}
\]
Unit
\[ \Gamma \vdash \text{1 : type} \]

Kind Conversion
\[ \begin{align*} \Gamma & \vdash A : K_1 \quad \Gamma \vdash K_1 = K_2 : \text{kind} \\ \Gamma & \vdash A : K_2 \end{align*} \]

\[ \Gamma \vdash K : \text{kind} \]

Type
\[ \Gamma \vdash \text{type : kind} \]

Products
\[ \begin{align*} \Gamma & \vdash A : \text{type} \quad \Gamma, x : A \vdash K : \text{kind} \\ \Gamma & \vdash \Pi x : A. K : \text{kind} \end{align*} \]

Definition equality is presented in the form of the three judgments detailed below. These axiomatize equality between objects, type families and kinds.

\[ \Gamma \vdash M_1 = M_2 : A \]

Variables
\[ \begin{align*} \Gamma & \vdash (x) = A \\ \Gamma & \vdash x = x : A \end{align*} \]

Constants
\[ \begin{align*} \Gamma & \vdash \Sigma(c) = A \\ \Gamma & \vdash c = c : A \end{align*} \]

Applications
\[ \begin{align*} \Gamma & \vdash M_{11} = M_{21} : \Pi x : A_2.A_1 \\ \Gamma & \vdash M_{12} = M_{22} : A_2 \\ \Gamma & \vdash M_{11} M_{12} = M_{21} M_{22} : A_1 [M_{12}/x] \end{align*} \]

Abstractions
\[ \begin{align*} \Gamma & \vdash A_{11} = A_1 : \text{type} \\ \Gamma & \vdash A_{12} = A_1 : \text{type} \\ \Gamma, x : A_1 \vdash M_1 = M_2 : A_2 \\ \Gamma & \vdash \lambda x : A_{11}. M_1 = \lambda x : A_{12}. M_2 : \Pi x : A_1.A_2 \end{align*} \]

Extensionality at \( \Pi \) type
\[ \begin{align*} \Gamma & \vdash A_1 : \text{type} \\ \Gamma & \vdash M_1 : \Pi x : A_1.A_2 \\ \Gamma & \vdash M_2 : \Pi x : A_1.A_2 \\ \Gamma, x : A_1 \vdash M_1 x = M_2 x : A_2 \\ \Gamma & \vdash M_1 = M_2 : \Pi x : A_1.A_2 \end{align*} \]

Parallel \( \beta \)-conversion for function application
\[ \begin{align*} \Gamma & \vdash A_1 : \text{type} \\ \Gamma, x : A_1 \vdash M_{12} = M_{22} : A_2 \\ \Gamma & \vdash M_{11} = M_{21} : A_1 \\ \Gamma & \vdash (\lambda x : A_1.M_{12}) M_{11} = M_{22} [M_{21}/x] : A_2 [M_{11}/x] \end{align*} \]

Pairs
\[ \begin{align*} \Gamma & \vdash \Sigma x : A_1.A_2 : \text{type} \\ \Gamma & \vdash M_{11} = M_{21} : A_1 \\ \Gamma & \vdash M_{12} = M_{22} : A_2 [M_{11}/x] \\ \Gamma & \vdash (M_{11}, M_{12}) \Sigma x : A_1.A_2 = (M_{21}, M_{22}) \Sigma x : A_1.A_2 : \Sigma x : A_1.A_2 \end{align*} \]

Projections
\[ \begin{align*} \Gamma & \vdash M_1 = M_2 : \Sigma x : A_1.A_2 \\ \Gamma & \vdash \pi_1 M_1 = \pi_1 M_2 : A_1 \\ \Gamma & \vdash M_1 = M_2 : \Sigma x : A_1.A_2 \\ \Gamma & \vdash \pi_2 M_1 = \pi_2 M_2 : A_2 [\pi_1 M_1/x] \end{align*} \]
Extensionality at Unit Type
\[ \frac{\Gamma \vdash M_1 : 1 \quad \Gamma \vdash M_2 : 1}{\Gamma \vdash M_1 = M_2 : 1} \]

Parallel $\beta$-Conversion for pair projection
\[ \frac{\Gamma \vdash M_1 = M_3 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \pi_1 (M_1, M_2)^A = M_3 : A_1} \]
\[ \frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \pi_2 (M_1, M_2)^A = M_3 : A_2} \]

Extensionality at $\Sigma$ type
\[ \frac{\Gamma \vdash M_1 : \Sigma x : A_1. A_2 \quad \Gamma \vdash M_2 : \Sigma x : A_1. A_2}{\Gamma \vdash \pi_1 M_1 = \pi_1 M_2 : A_1 \quad \Gamma \vdash \pi_2 M_1 = \pi_2 M_2 : A_2 [\pi_1 M_1/x]} \]

Symmetry
\[ \frac{\Gamma \vdash M_2 = M_1 : A}{\Gamma \vdash M_1 = M_2 : A} \]

Transitivity
\[ \frac{\Gamma \vdash M_1 = M_2 : A \quad \Gamma \vdash M_2 = M_3 : A}{\Gamma \vdash M_1 = M_3 : A} \]

Type Conversion
\[ \frac{\Gamma \vdash A_1 = A_2 : \text{type}}{\Gamma \vdash M_1 = M_2 : A_1 \quad \Gamma \vdash M_1 = M_2 : A_2} \]

\[ \frac{\Gamma \vdash A_1 = A_2 : K}{\Sigma(a) = K} \]
\[ \frac{\Gamma \vdash a = a : K}{\Gamma \vdash a = a : K} \]

Abstractions
\[ \frac{\Gamma \vdash A_{11} = A_1 : \text{type} \quad \Gamma \vdash A_{21} = A_1 : \text{type} \quad \Gamma, x : A_1 \vdash A_{12} = A_{22} : K}{\Gamma \vdash \lambda x : A_{11}, A_{12} = \lambda x : A_{21}, A_{22} : \Pi x : A_1. K} \]

Applications
\[ \frac{\Gamma \vdash A_1 = A_2 : \Pi x : A_3. K \quad \Gamma \vdash M_1 = M_2 : A_3}{\Gamma \vdash A_1 M_1 = A_2 M_2 : K [M_1/x]} \]

Extensionality at $\Pi$ kind
\[ \frac{\Gamma \vdash A : \text{type} \quad \Gamma \vdash A_1 : \Pi x : A. K \quad \Gamma \vdash A_2 : \Pi x : A. K \quad \Gamma, x : A \vdash A_1 x = A_2 x : K}{\Gamma \vdash A_1 = A_2 : \Pi x : A. K} \]

Parallel $\beta$-Conversion for function application
\[ \frac{\Gamma \vdash A : \text{type} \quad \Gamma, x : A \vdash A_1 = A_2 : K \quad \Gamma \vdash M_1 = M_2 : A}{\Gamma \vdash (\lambda x : A. A_1) M_1 = A_2 [M_2/x] : K [M_1/x]} \]
Dependent Function Types
\[ \Gamma \vdash A_{11} = A_{21} : \text{type} \quad \Gamma \vdash A_{12} : \text{type} \quad \Gamma, x : A_{11} \vdash A_{12} = A_{22} : \text{type} \]

\[ \Gamma \vdash \Pi x : A_{11}. A_{12} = \Pi x : A_{21}. A_{22} : \text{type} \]

Dependent Pair Types
\[ \Gamma \vdash A_{11} = A_{21} : \text{type} \quad \Gamma \vdash A_{12} : \text{type} \quad \Gamma, x : A_{11} \vdash A_{12} = A_{22} : \text{type} \]

\[ \Gamma \vdash \Sigma x : A_{11}. A_{12} = \Sigma x : A_{21}. A_{22} : \text{type} \]

Unit
\[ \Gamma \vdash 1 = 1 : \text{type} \]

Symmetry
\[ \Gamma \vdash A_{2} = A_{1} : K \quad \Gamma \vdash A_{1} = A_{2} : K \]

Transitivity
\[ \Gamma \vdash A_{1} = A_{2} : K \quad \Gamma \vdash A_{2} = A_{3} : K \]
\[ \Gamma \vdash A_{1} = A_{3} : K \]
\[ \Gamma \vdash A_{1} = A_{2} : K_{1} \quad \Gamma \vdash A_{1} = A_{2} : K_{2} \]
\[ \Gamma \vdash K_{1} = K_{2} : \text{kind} \]
\[ \Gamma \vdash K_{2} = K_{1} : \text{kind} \]
\[ \Gamma \vdash K_{1} = K_{3} : \text{kind} \quad \Gamma \vdash K_{2} = K_{3} : \text{kind} \]
\[ \Gamma \vdash K_{1} = K_{3} : \text{kind} \]

Kind Conversion
\[ \Gamma \vdash K_{1} = K_{2} : \text{kind} \]
\[ \Gamma \vdash K_{2} = K_{1} : \text{kind} \]
\[ \Gamma \vdash K_{1} = K_{1} : \text{kind} \]
\[ \Gamma \vdash K_{1} = K_{2} : \text{kind} \]
\[ \Gamma \vdash K_{2} = K_{3} : \text{kind} \]
\[ \Gamma \vdash K_{1} = K_{3} : \text{kind} \]

Type
\[ \Gamma \vdash \text{type} = \text{type} : \text{kind} \]

Dependent Function Kinds
\[ \Gamma \vdash A_{1} = A_{2} : \text{type} \quad \Gamma \vdash A_{1} : \text{type} \quad \Gamma, x : A_{1} \vdash K_{1} = K_{2} : \text{kind} \]
\[ \Gamma \vdash \Pi x : A_{1}. K_{1} = \Pi x : A_{2}. K_{2} : \text{kind} \]

Symmetry
\[ \Gamma \vdash K_{1} = K_{1} : \text{kind} \]
\[ \Gamma \vdash K_{2} = K_{2} : \text{kind} \]

Transitivity
\[ \Gamma \vdash K_{1} = K_{2} : \text{kind} \quad \Gamma \vdash K_{2} = K_{3} : \text{kind} \]
\[ \Gamma \vdash K_{1} = K_{3} : \text{kind} \]

Well Typed Substitutions
Finally, we introduce notation for typing substitutions below.

Definition 3.1 The judgment \( \Gamma_2 \vdash \sigma : \Gamma_1 \) holds iff \( \forall x \in \text{Dom}(\Gamma_1). \Gamma_2 \vdash \sigma(x) : \sigma(\Gamma_1(x)) \).

Definition 3.2 The judgment \( \Gamma_2 \vdash \sigma_1 = \sigma_2 : \Gamma_1 \) holds iff
\[ \cdot \Gamma_2 \vdash \sigma_1 : \Gamma_1, \]
\[ \cdot \Gamma_2 \vdash \sigma_2 : \Gamma_1, \text{ and} \]
\[ \cdot \forall x \in \text{Dom}(\Gamma_1). \Gamma_2 \vdash \sigma_1(x) = \sigma_2(x) : \sigma_1(\Gamma_1(x)). \]

4 Elementary Properties

We start with some basic properties of the system. The most important lemma in this section is the regularity lemma, which states that terms appearing in valid derivations are themselves well typed. To get there, we first prove some basic structural facts about the typing judgments can be proved by easy structural inductions.
Lemma 4.1 (Weakening) *If* $\Gamma_1 \vdash J$ *and* $\Gamma_1 \subseteq \Gamma_2$, *then* $\Gamma_2 \vdash J$.

**Proof**

By structural induction on the derivation of the first judgment.

---

Lemma 4.2 (Free Variable Containment) *If* $\vdash \Gamma : \text{ctx}$ *and* $\Gamma \vdash J$, *then* $\text{FV}(J) \in \text{Dom}(\Gamma)$.

**Proof**

By structural induction on the derivation of the second judgment.

---

Next we show that reflexivity is admissible for our definition of definitional equality. This lemma shows that definitional equality is an equivalence relation (it is already symmetric and transitive by rules).

**Lemma 4.3 (Reflexivity)**

1. If $\Gamma \vdash M : A$ then $\Gamma \vdash M = M : A$.
2. If $\Gamma \vdash A : K$ then $\Gamma \vdash A = A : K$.
3. If $\Gamma \vdash K : \text{kind}$ then $\Gamma \vdash K = K : \text{kind}$.

**Proof**

By structural induction on the derivation of the judgment.

---

One of the key properties of a declarative system is the admissibility of a substitution property for hypotheses. To prove this, we need some basic facts about substitutions, such as that we can always come up with an identity substitution for a well-typed context, and that substitutions can be extended to have no effect on terms not in the domain of the substitution.

**Lemma 4.4 (Identity Substitutions)** *If* $\vdash \Gamma : \text{ctx}$ *then* $\Gamma \vdash \text{id}_\Gamma : \Gamma$ and $\Gamma \vdash \text{id}_\Gamma = \text{id}_\Gamma : \Gamma$.

**Proof**

By induction on the construction of the context.

---

**Lemma 4.5 (Extending Substitutions)**

1. If $\Gamma_1 \vdash \sigma_1 : \Gamma$, $\Gamma \vdash A : \text{type}$ *and* $x \notin \text{Dom}(\Gamma) \cup \text{Dom}(\Gamma_1)$ *then* $\Gamma_1, x : A[\sigma_1] \vdash \sigma_1, x/x : \Gamma, x : A$.
2. If $\Gamma_1 \vdash \sigma_1 = \sigma_2 : \Gamma$, $\Gamma \vdash A : \text{type}$ *and* $x \notin \text{Dom}(\Gamma) \cup \text{Dom}(\Gamma_1)$ *then* $\Gamma_1, x : A[\sigma_1] \vdash \sigma_1, x/x = \sigma_2, x/x : \Gamma, x : A$.

**Proof**

**Case 1:**

Direct, by the definition of substitution typing, using Weakening.

**Case 2:**

Direct, by the definition of substitution equality, part 1 and Weakening.

---

Next, we show that the notion of substitutions can be lifted to entire judgments.

**Lemma 4.6 (Substitution)** *If* $\Gamma_1 \vdash J$ *and* $\Gamma_2 \vdash \sigma : \Gamma_1$, *then* $\Gamma_2 \vdash J[\sigma]$. 

---


Proof

By structural induction on the second judgment.

With the key substitution property already proved, an useful fact about contexts can now be proved. We often want to change declarations in the context to bind a definitionally equal type. This is now easy to prove.

Lemma 4.7 (Context Conversion) Assume \( \Gamma, x : A_1 \vdash \text{ctx}, \Gamma \vdash A_2 : \text{type} \), and \( \Gamma \vdash A_1 = A_2 : \text{type} \). If \( \Gamma, x : A_1 \vdash J \), then \( \Gamma, x : A_2 \vdash J \).

Proof

Direct, by weakening and substitution.

\[
\begin{align*}
\Gamma, x : A_2 \vdash x : A_2 & \quad \text{By rule (variable)} \\
\Gamma \vdash A_1 = A_2 : \text{type} & \quad \text{By assumption} \\
\Gamma \vdash A_2 = A_1 : \text{type} & \quad \text{By rule (symmetry)} \\
\Gamma, x : A_2 \vdash x : A_1 & \quad \text{By (type conversion)} \\
\Gamma, x : A_1 \vdash J & \quad \text{By assumption} \\
\Gamma, x_1 : A_1 \vdash J [x_1 / x] & \quad \text{By substitution} \\
\Gamma, x : A_2, x_1 : A_1 \vdash J [x_1 / x] & \quad \text{By weakening} \\
\Gamma, x : A_2 \vdash (J [x_1 / x])[x/x_1] & \quad \text{By substitution} \\
\Gamma, x : A_2 \vdash J & \quad \text{By definition of substitution}
\end{align*}
\]

A critical lemma for our purposes is the lemma that is usually called functionality. This says that equal substitutions are “functional”, i.e. given equal elements, they produce equal elements. Such a property is needed, for example, in our proofs of completeness of algorithmic equality with respect to the definitional equality given above. Our proof of this property needs regularity. However, our proof of regularity itself seems to need this property. Fortunately, the form of functionality needed in the regularity proof is less general. We need the fact that applying equal substitutions to the same element produces equal elements. This less general lemma can be proved directly, without an appeal to regularity. After we prove regularity, we can come back and prove the more general version we will need later.

Lemma 4.8 (Functionality for Typing) Assume \( \Gamma \vdash \sigma_1 : \Gamma, \Gamma \vdash \sigma_2 : \Gamma \), and \( \Gamma \vdash \sigma_1 = \sigma_2 : \Gamma \).

1. If \( \Gamma \vdash M : A \) then \( \Gamma \vdash M[\sigma_1] = M[\sigma_2] : A[\sigma_1] \).
2. If \( \Gamma \vdash A : K \) then \( \Gamma \vdash A[\sigma_1] = A[\sigma_2] : K[\sigma_1] \).
3. If \( \Gamma \vdash K : \text{kind} \) then \( \Gamma \vdash K[\sigma_1] = K[\sigma_2] : \text{kind} \).

Proof

By induction on the typing judgment. We show one representative cases for pairs.

Case 1:

\[
\begin{align*}
\Gamma \vdash \Sigma x : A_1, A_2 : \text{type} & \quad \Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2[M_1 / x] \\
\Gamma \vdash (M_1, M_2)_{\Sigma x : A_1, A_2} : \Sigma x : A_1, A_2
\end{align*}
\]

\[
\begin{align*}
\Gamma_1 \vdash (\Sigma x : A_1, A_2)[\sigma_1] : \text{type} & \quad \text{By Substitution} \\
\Gamma_1 \vdash \Sigma x : (A_1[\sigma_1]), (A_2[\sigma_1]) : \text{type} & \quad \text{By definition of substitution} \\
\Gamma_1 \vdash M_1[\sigma_1] = M_1[\sigma_2] : A_1[\sigma_1] & \quad \text{By induction} \\
\Gamma_1 \vdash M_2[\sigma_1] = M_2[\sigma_2] : (A_2[M_1 / x])[\sigma_1] & \quad \text{By induction}
\end{align*}
\]
Lemma 4.9 (Inversion on Products and Sums)

1. If $\Gamma \vdash \Pi x : A_1 . A_2 : K$ then $\Gamma \vdash A_1 : type$ and $\Gamma, x : A_1 \vdash A_2 : type$.
2. If $\Gamma \vdash \Sigma x : A_1 . A_2 : K$ then $\Gamma \vdash A_1 : type$ and $\Gamma, x : A_1 \vdash A_2 : type$.
3. If $\Gamma \vdash \Pi x : A . K : kind$ then $\Gamma \vdash A : type$ and $\Gamma, x : A \vdash K : kind$.

Proof
By induction on the first derivation.

We are now in a position to prove our key regularity property.

Lemma 4.10 (Regularity) Assume $\vdash \Gamma : ctx$.

1. If $\Gamma \vdash M : A$ then $\Gamma \vdash A : type$.
2. If $\Gamma \vdash M_1 = M_2 : A$ then $\Gamma \vdash M_1 : A$, $\Gamma \vdash M_2 : A$, and $\Gamma \vdash A : type$.
3. If $\Gamma \vdash A : K$ then $\Gamma \vdash K : kind$.
4. If $\Gamma \vdash A_1 = A_2 : K$ then $\Gamma \vdash A_1 : K$, $\Gamma \vdash A_2 : K$, and $\Gamma \vdash K : kind$.
5. If $\Gamma \vdash K_1 = K_2 : kind$ then $\Gamma \vdash K_1 : kind$, and $\Gamma \vdash K_2 : kind$.

Proof
By induction on the derivation of the judgment. We show some representative cases.

Case 1:

\[
\begin{array}{c}
\Gamma \vdash A_1 : type \\
\Gamma, x : A_1 \vdash M_2 : A_2 \\
\end{array}
\]

\[
\Gamma \vdash \lambda x : A_1 . M_2 : \Pi x : A_1 . A_2
\]

By induction

Case 2:

\[
\begin{array}{c}
\Gamma \vdash M : \Sigma x : A_1 . A_2 \\
\end{array}
\]

\[
\Gamma \vdash \pi_2 M : A_2 [\pi_1 M / x]
\]

By induction

\Gamma \term M : A_1 . A_2 \\
\Gamma \term A_1 : type \\
\Gamma, x : A_1 \term A_2 : type \\
\Gamma \term \pi_1 M : A_1 \\
\Gamma_2 \term A_2 [\pi_1 M / x] : type

By Inversion on Sums

By assumption

By rule (projection)

By Substitution
Case 3:

\[
\begin{align*}
\Gamma &\vdash A_{11} : \text{type} & \Gamma &\vdash A_{12} : \text{type} & \Gamma, x : A_{1} \vdash M_1 = M_2 : A_2 \\
\Gamma &\vdash \lambda x : A_{11} . M_1 = \lambda x : A_{12} . M_2 : \Pi x : A_{1} . A_{2}
\end{align*}
\]

\[\Gamma \vdash A_{11} : \text{type} \quad \text{By induction}\]
\[\Gamma \vdash A_{21} : \text{type} \quad \text{By induction}\]
\[\Gamma \vdash A_{1} : \text{type} \quad \text{By induction}\]
\[\Gamma, x : A_{1} \vdash M_{1} : A_{2}, \quad \text{By induction}\]
\[\Gamma, x : A_{1} \vdash A_{2} : A_{2}, \quad \text{By induction}\]
\[\Gamma, x : A_{1} \vdash A_{2} : \text{type} \quad \text{By induction}\]
\[\Gamma, x : A_{1} \vdash A_{2} = A_{2} : \text{type} \quad \text{By Reflexivity}\]
\[\Gamma \vdash \Pi x : A_{1} . A_{2} : \text{type} \quad \text{By rule}\]
\[\Gamma \vdash \Pi x : A_{1} . A_{2} = \Pi x : A_{11} . A_{2} : \text{type} \quad \text{By rule}\]
\[\Gamma, x : A_{21} \vdash M_{1} : A_{2} \quad \text{By rule}\]
\[\Gamma \vdash \lambda x : A_{11} . M_{1} : \Pi x : A_{11} . A_{2} \quad \text{By rule (abstraction)}\]
\[\Gamma \vdash \lambda x : A_{11} . M_{1} : \Pi x : A_{1} . A_{2} \quad \text{By rule (type conversion)}\]
\[\Gamma, x : A_{21} \vdash M_{2} : A_{2} \quad \text{By Context Conversion}\]
\[\Gamma \vdash \lambda x : A_{21} . M_{2} : \Pi x : A_{21} . A_{2} \quad \text{By rule (abstraction)}\]
\[\Gamma \vdash \lambda x : A_{21} . M_{2} : \Pi x : A_{1} . A_{2} \quad \text{By rule (type conversion)}\]

Case 4:

\[
\begin{align*}
\Gamma &\vdash A_{1} : \text{type} & \Gamma, x : A_{1} \vdash M_{12} = M_{22} : A_{2} & \Gamma \vdash M_{11} = M_{21} : A_{1} \\
\Gamma \vdash (\lambda x : A_{1} . M_{12}) M_{11} = M_{22} [M_{21}/x] : A_{2} [M_{11}/x]
\end{align*}
\]

\[\Gamma, x : A_{1} \vdash M_{12} : A_{2}, \quad \text{By induction}\]
\[\Gamma, x : A_{1} \vdash M_{22} : A_{2}, \quad \text{By induction}\]
\[\Gamma, x : A_{1} \vdash A_{2} : \text{type} \quad \text{By rule}\]
\[\Gamma \vdash M_{11} : A_{1}, \quad \text{By rule}\]
\[\Gamma \vdash M_{21} : A_{1}, \quad \text{By rule}\]
\[\Gamma \vdash A_{1} : \text{type} \quad \text{By induction}\]
\[\Gamma \vdash A_{2} [M_{11}/x] : \text{type} \quad \text{By Substitution}\]
\[\Gamma \vdash A_{2} [M_{11}/x] = A_{2} [M_{21}/x] : \text{type} \quad \text{By Functionality}\]
\[\Gamma \vdash M_{22} [M_{21}/x] : A_{2} [M_{21}/x] \quad \text{By Substitution}\]
\[\Gamma \vdash M_{22} [M_{21}/x] : A_{2} [M_{11}/x] \quad \text{By rule (type conversion)}\]
\[\Gamma, x : A_{2} . M_{11} = \Pi x : A_{1} . A_{2} \quad \text{By rule}\]
\[\Gamma \vdash (\lambda x : A_{1} . M_{12}) M_{11} : A_{2} [M_{11}/x] \quad \text{By rule}\]

Case 5:

\[
\begin{align*}
\Gamma &\vdash \Sigma x : A_{1}, A_{2} : \text{type} & \Gamma \vdash M_{11} = M_{21} : A_{1} & \Gamma \vdash M_{12} = M_{22} : A_{2} [M_{11}/x]
\end{align*}
\]

\[\Gamma \vdash (\Sigma x : A_{1}, A_{2}) \Sigma x : A_{1}, A_{2} = (\Sigma x : A_{1}, A_{2}) \Sigma x : A_{1}, A_{2} : \Sigma x : A_{1}, A_{2}\]

\[\Gamma \vdash A_{1} : \text{type}, \quad \text{By Inversion on Sums}\]
\[\Gamma, x : A_{1} \vdash A_{2} : \text{type} \quad \text{By function}\]
\[\Gamma \vdash M_{11} : A_{1}, \quad \text{By induction}\]
\[\Gamma \vdash M_{21} : A_{1}, \quad \text{By induction}\]
\[\Gamma \vdash A_{2} [M_{11}/x] = A_{2} [M_{21}/x] : \text{type} \quad \text{By rule (type conversion)}\]
\[\Gamma \vdash M_{12} : A_{2} [M_{11}/x], \quad \text{By assumption}\]
\[\Gamma \vdash M_{22} : A_{2} [M_{11}/x] \quad \text{By rule}\]
\[\Gamma \vdash \Sigma x : A_{1}, A_{2} : \text{type} \quad \text{By rule}\]
\[\Gamma \vdash (\Sigma x : A_{1}, A_{2}) \Sigma x : A_{1}, A_{2} : \Sigma x : A_{1}, A_{2} \quad \text{By rule}\]
\[\Gamma \vdash (\Sigma x : A_{1}, A_{2}) \Sigma x : A_{1}, A_{2} : \Sigma x : A_{1}, A_{2} \quad \text{By rule}\]
Lemma 4.11 (Functionality for Equality) Assume $\Gamma \vdash \sigma_1 = \sigma_2 : \Gamma$.

1. If $\Gamma \vdash M_1 = M_2 : A$ then $\Gamma \vdash M_1[\sigma_1] = M_2[\sigma_2] : A[\sigma_1]$.
2. If $\Gamma \vdash A_1 = A_2 : K$ then $\Gamma \vdash A_1[\sigma_1] = A_2[\sigma_2] : K[\sigma_1]$.
3. If $\Gamma \vdash K_1 = K_2 : \text{kind}$ then $\Gamma \vdash K_1[\sigma_1] = K_2[\sigma_2] : \text{kind}$.

Proof

Direct. We show one case.

1. $\Gamma \vdash \sigma_1 : \Gamma$,
2. $\Gamma \vdash \sigma_2 : \Gamma$,
3. $\Gamma \vdash M_1[\sigma_1] = M_2[\sigma_1] : A[\sigma_1]$,
4. $\Gamma \vdash M_2[\sigma_1] = M_2[\sigma_2] : A[\sigma_1]$,
5. $\Gamma \vdash M_1[\sigma_1] = M_2[\sigma_2] : A[\sigma_1]$,
6. By definition of substitution equality
7. By Substitution
8. By Regularity
9. By Functionalitv
10. By rule (transitivity)

Lemma 4.12 (Typing Inversion) Assume $\vdash \Gamma : \text{ctx}$.

1. If $\Gamma \vdash x : A_1$ then $\Gamma(x) = A_2$ and $\Gamma \vdash A_1 = A_2 : \text{type}$.
2. If $\Gamma \vdash c : A_1$ then $\Sigma(c) = A_2$ and $\Gamma \vdash A_1 = A_2 : \text{type}$.
3. If $\Gamma \vdash \lambda x : A_1, M_1 : A$ then $\Gamma \vdash A_1 : \text{type}$ and $\Gamma, x : A_1 \vdash M_1 : A_2$ and $\Gamma \vdash \Pi x : A_1. A_2 = A : \text{type}$.
4. If $\Gamma \vdash M_1 : A$ then $\Gamma \vdash M_1 : \Pi x : A_1. A_2$, $\Gamma \vdash M_2 : A_1$ and $\Gamma \vdash A_2[\sigma_1/x] = A : \text{type}$.
5. If $\Gamma \vdash (M_1, M_2) : \Sigma x : A_1. A_2$ then $\Gamma \vdash M_1 : A_1$, $\Gamma \vdash M_2 : A_2[\sigma_1/x]$ and $\Gamma \vdash \Sigma x : A_1. A_2 = A : \text{type}$.
6. If $\Gamma \vdash \pi_1 M_1 : A$ then $\Gamma \vdash M_1 : \Sigma x : A_1. A_2$ and $\Gamma \vdash A_1 = A : \text{type}$.
7. If \( \Gamma \vdash \pi_2 M_1 : A \) then \( \Gamma \vdash M_1 : \Sigma x : A_1.A_2 \) and \( \Gamma \vdash A_2 [\pi_1 M_1/x] = A : \text{type} \).

8. If \( \Gamma \vdash () : A \) then \( \Gamma \vdash 1 = A : \text{type} \).

9. If \( \Gamma \vdash a : K_1 \) then \( \Sigma(a) = K_2 \) and \( \Gamma \vdash K_1 = K_2 : \text{kind} \).

10. If \( \Gamma \vdash \lambda x : A_1. A_2 : K \) then \( \Gamma \vdash A_1 : \text{type} \) and \( \Gamma, x : A_1 \vdash A_2 : K_1 \) and \( \Gamma \vdash \Pi x : A_1.K_1 = K : \text{kind} \).

11. If \( \Gamma \vdash A : M : K \) then \( \Gamma \vdash A : \Pi x : A_1.K_1, \Gamma \vdash M : A_1 \) and \( \Gamma \vdash K_1 [M/x] = K : \text{kind} \).

12. If \( \Gamma \vdash \Pi x : A_1.A_2 : K \) then \( \Gamma \vdash A_1 : \text{type} \), \( \Gamma, x : A_1 \vdash A_2 : \text{type} \) and \( \Gamma \vdash \text{type} = K : \text{kind} \).

13. If \( \Gamma \vdash \Sigma x : A_1.A_2 : K \) then \( \Gamma \vdash A_1 : \text{type} \), \( \Gamma, x : A_1 \vdash A_2 : \text{type} \) and \( \Gamma \vdash \text{type} = K : \text{kind} \).

14. If \( \Gamma \vdash 1 : K \) then \( \Gamma \vdash \text{type} = K : \text{kind} \).

**Proof**

By induction on the derivation of the judgment. We show a few representative cases.

**Case 1:**

\[
\frac{\Gamma \vdash M : \Sigma x : A_1.A_2}{\Gamma \vdash \pi_2 M : A_2 [\pi_1 M/x]}
\]

\( \Gamma \vdash A_2 [\pi_1 M/x] = \text{type} \) \hspace{1cm} \text{By Regularity}

\( \Gamma \vdash A_2 [\pi_1 M/x] = A_2 [\pi_1 M/x] : \text{AAtype} \) \hspace{1cm} \text{By Reflexivity}

**Case 2:**

\[
\frac{\Gamma \vdash (M_1, M_2)^A : A_1 \quad \Gamma \vdash A_1 = A_2 : \text{type}}{\Gamma \vdash (M_1, M_2)^A : A_2}
\]

\( \Gamma \vdash M_1 : A_3 \), \hspace{1cm} \text{By induction}

\( \Gamma \vdash M_2 : A_4 [M_1/x] \), \hspace{1cm} \text{By rule (transitivity)}

\( \Gamma \vdash \Sigma x : A_4.A_4 = A_1 : \text{type} \)

\( \Gamma \vdash \Sigma x : A_4.A_4 = A_2 : \text{type} \)

\( \square \)

**Lemma 4.13 (Extending Equal Substitutions)**

If \( \Gamma \vdash \text{ctx}, \Gamma_1 \vdash \sigma_1 = \sigma_2 : \Gamma, \Gamma \vdash A : \text{type} \) and \( \Gamma \vdash M_1 = M_2 : A \) and \( x \notin \text{Dom}(\Gamma) \) then \( \Gamma_1 \vdash \sigma_1, M_1/x = \sigma_2, M_2/x : \Gamma, x : A \).

**Proof**

Direct, by Regularity and Functionality for Equality. \( \square \)

**Lemma 4.14 (Equality Inversion on Kinds)**

1. If \( \Gamma \vdash K = \text{kind} \) or \( \Gamma \vdash \text{type} = K : \text{kind} \) then \( K = \text{type} \).

2. If \( \Gamma \vdash K' = \Pi x : A.K : \text{kind} \) or \( \Gamma \vdash \Pi x : A.K = K' : \text{kind} \) then \( K' = \Pi x : A.K_1 \) with \( \Gamma \vdash A_1 = A : \text{type} \) and \( \Gamma \vdash K_1 = K : \text{kind} \).

**Proof**

By induction on derivations. \( \square \)
5 Injectivity

We need a property usually called injectivity. Briefly, this says that given equal dependent function types, the components of the product types are themselves equal (at the kind type). A similar property should hold for dependent pair types. In Harper and Pfenning [5], this follows by an easy induction analogous to the case for product kinds (Lemma 4.14). However, we now have a non-trivial equality relation at the level of types due to the presence of family-level abstractions and applications. The problem in an inductive proof comes with the use of the transitivity rule. While judging equality between two products (say), the mediating family need not be of the same form. To get around this, we follow Vanwaaert and Crary [7] and prove by the method of logical relations. In defining the logical relation we will need, we need the auxiliary notion of weak-head reduction. This notion is important in its own right, and will be reused in the definition of the equality algorithm.

5.1 Weak Head Reduction

By case analysis on the derivation of the first judgment.

Lemma 5.1 (Determinacy of Weak Head Reduction)

1. If \( M_1 \xrightarrow{\text{whr}} M_2 \) and \( M_1 \xrightarrow{\text{whr}} M_3 \) then \( M_2 = M_3 \).

2. If \( A_1 \xrightarrow{\text{whr}} A_2 \) and \( A_1 \xrightarrow{\text{whr}} A_3 \) then \( A_2 = A_3 \).

Proof

By case analysis on the derivation of the first judgment.

We write \( \xrightarrow{\text{whr}^*} \) for the reflexive transitive closure of the weak-head reduction relation \( \xrightarrow{\text{whr}} \). On occasion, we write \( M \xrightarrow{\text{whr}^*} \) to indicate that there exists no \( M_1 \) such that \( M \xrightarrow{\text{whr}} M_1 \) according to the rules set out here. An analogous notation \( A \xrightarrow{\text{whr}^*} \) is defined for families.

5.2 A Logical Relation

The logical relation we use is defined by induction on kinds. The notable feature is that at higher kinds, we require definitional equality instead of logical relatedness for the arguments. Since the structure of kinds (i.e. whether they are a dependent function kind or type) is not affected by substitution of object variables, the definition of the relation is well-founded.

- \( \Gamma \vdash A_1 \approx A_2 : \text{[type]} \) iff all these hold:
  - \( A_1 \xrightarrow{\text{whr}^*} \Pi x : A_{11}, A_{12} \) iff \( A_2 \xrightarrow{\text{whr}^*} \Pi x : A_{21}, A_{22} \).
  - \( A_1 \xrightarrow{\text{whr}^*} \Sigma x : A_{11}, A_{12} \) iff \( A_2 \xrightarrow{\text{whr}^*} \Sigma x : A_{21}, A_{22} \).
Proof

Lemma 5.4 (Equivalent Substitutions of a Valid Kind Related) If \( A_1 \xrightarrow{\text{whr}^*} \Pi x:A_{11}.A_{12} \) and \( A_2 \xrightarrow{\text{whr}^*} \Pi x:A_{21}.A_{22} \) then \( \Gamma \vdash A_{11} = A_{21} : \text{type} \) and \( \Gamma, x:A_{11} \vdash A_{12} = A_{22} : \text{type} \).

- If \( A_1 \xrightarrow{\text{whr}^*} \Sigma x:A_{11}.A_{12} \) and \( A_2 \xrightarrow{\text{whr}^*} \Sigma x:A_{21}.A_{22} \) then \( \Gamma \vdash A_{11} = A_{21} : \text{type} \) and \( \Gamma, x:A_{11} \vdash A_{12} = A_{22} : \text{type} \).

- If \( A_1 \approx_2 A_2 : [\Pi x:A.K] \) iff for every pair of objects \( M_1 \) and \( M_2 \) such that \( \Gamma \vdash M_1 = M_2 : A \) we have \( \Gamma \vdash A_1 M_1 \approx A_2 M_2 : [K[M_1/x]] \).

- If \( A_1 \approx_2 A_2 : [\text{kind}] \) iff for every pair of families \( A_1 \) and \( A_2 \), \( \Gamma \vdash A_1 \approx A_2 : [K_1] \) iff \( \Gamma \vdash A_1 \approx A_2 : [K_2] \).

The fundamental theorem of Logical Relations is that definitionaly equal type families and kinds are logically related under all substitutions. To handle the symmetry and transitivity cases we will need that the logical relation itself is symmetric and transitive. To handle the abstraction case, we will need that the logical relation is closed under weak head expansion. To handle the family-level constant case, we will need that paths are logically related to each other. We now prove these standard lemmas before setting out to prove the fundamental lemma.

Lemma 5.2 (Paths are Logically Related) If \( \Sigma(a) = \Pi x_1:A_1 \ldots \Pi x_k:A_k \cdot K \) (\( k \geq 0 \)) and \( \Gamma \vdash M_i = M'_i : A_i[M_1 \ldots M_{i-1}/x_1 \ldots x_{i-1}] \) for all \( i \), then \( \Gamma \vdash a_{M_1} \ldots M_i = a_{M'_1} \ldots M'_i : [K[M_1 \ldots M_i/x_1 \ldots x_i]] \).

Proof

By induction on the size of the kind \( K \).

Lemma 5.3 (Closure Under Weak Head Expansion) If \( \Gamma \vdash A_{11} \approx A_{21} : [K] \), \( \Gamma \vdash A_{12} : K \), \( \Gamma \vdash A_{22} : K \), \( A_{12} \xrightarrow{\text{whr}^*} A_{11} \) and \( A_{22} \xrightarrow{\text{whr}^*} A_{21} \) then \( \Gamma \vdash A_{12} \approx A_{22} : [K] \).

Proof

By induction on the size of \( K \).

We shall need a small fact about substituting logically related families into kinds for proving symmetry of the logical relation.

Lemma 5.4 (Equivalent Substitutions of a Valid Kind Related) If \( \vdash \Gamma : \text{ctx} \), \( \Gamma \vdash K : \text{kind} \), \( \Gamma_1 \vdash \sigma_1 = \sigma_2 : \Gamma \) and \( \Gamma_1 \vdash A_1 \approx A_2 : [K[\sigma_1]] \) then \( \Gamma_1 \vdash A_1 \approx A_2 : [K[\sigma_2]] \).

Proof

By induction on the size of \( K \).

We need symmetry and transitivity of the relation, for the main theorem.

Lemma 5.5 (Symmetry of Logical Relation) If \( \vdash \Gamma : \text{ctx} \), \( \Gamma \vdash K : \text{kind} \), and \( \Gamma \vdash A_1 \approx A_2 : [K] \) then \( \Gamma \vdash A_2 \approx A_1 : [K] \).

Proof

By induction on the size of \( K \). We show only the product kind case.

Case 1: \( K = \Pi x:A.K_1 \)
Lemma 5.6 (Transitivity of Logical Relation) If $\Gamma \vdash A_1 \approx A_2 : [K_1[M_2/x]]$ and $\Gamma \vdash A_2 \approx A_3 : [K_1[M_3/x]]$, then $\Gamma \vdash A_1 \approx A_3 : [K_1[M_3/x]]$.

Proof

By induction on the size of $K$.

Lemma 5.7 (Definitionally Equal Terms are Logically Related Under Substitutions)

1. If $\Gamma \vdash A_1 = A_2 : K$ and $\Gamma \vdash \sigma_1 = \sigma_2 : \Gamma$ and $\Gamma \vdash \Gamma : \text{ctx}$ then $\Gamma \vdash A_1[\sigma_1] \approx A_2[\sigma_2] : [K[\sigma_1]]$.
2. If $\Gamma \vdash A_1 = A_2 : \text{kind}$ and $\Gamma \vdash \sigma_1 = \sigma_2 : \Gamma$ and $\Gamma \vdash \Gamma : \text{ctx}$ then $\Gamma \vdash A_1[\sigma_1] \approx A_2[\sigma_2] : \text{[kind]}$.

Proof

By induction on derivations. We show some of the more interesting cases.

Case 1:

$\Sigma(a) = K$

$\vdash a = a : K$

1. $\vdash K : \text{kind}$

By definition of valid signature

$FV(K) = \cdot$

By Free Variable Containment

$\Gamma \vdash a \approx a : [K]$

By Logically Related Paths

$\Gamma \vdash a[\sigma_1] \approx a[\sigma_2] : [K[\sigma_1]]$

By definition of substitution

Case 2:

$\Gamma \vdash A_{11} = A_1 : \text{type}$  $\Gamma \vdash A_{21} = A_1 : \text{type}$  $\Gamma, x : A_1 \vdash A_{12} = A_2 : K$

$\Gamma \vdash \lambda x : A_{11}, A_{12} = \lambda x : A_{21}, A_{22} : \Pi x : A_1, K$

$\Gamma \vdash M_1 = M_2 : A_1$

$\Gamma \vdash M_2 = M_1 : A_1$

$\Gamma \vdash A_1 M_2 \approx A_2 M_1 : [K_1[M_2/x]]$

$\Gamma \vdash A_2 M_1 \approx A_1 M_2 : [K_1[M_2/x]]$

$\Gamma \vdash A : \text{type}$

$\vdash \Gamma, x : A : \text{ctx}$

$\Gamma \vdash \text{idr}_\Gamma = \text{idr}_\Gamma : \Gamma$

By reflexivity of substitution equality

$\Gamma \vdash \text{idr}_\Gamma, M_2/x = \text{idr}_\Gamma, M_1/x : \Gamma, x : A$

By Extending Substitutions by Equal Elements

$\Gamma \vdash A_1 M_2 \approx A_2 M_1 : [K_1[M_1/x]]$

$\Gamma \vdash A_2 \approx A_1 : [[\Pi x : A_1, K]]$

$\Gamma \vdash \lambda x : A_{11}[\sigma_1], A_{12}[\sigma_1] \approx \lambda x : A_{21}[\sigma_1], A_{22}[\sigma_2] : [K[\sigma_1][M_3/x][\sigma_1]]$

By definition of relation

$\Gamma \vdash \lambda x : A_{11}[\sigma_1], A_{12}[\sigma_1] \approx \lambda x : A_{21}[\sigma_1], A_{22}[\sigma_2] : [\Pi x : A_1, K]$

By Closure Under Expansion

$\Gamma \vdash \lambda x : A_{12}[\sigma_1] \approx \lambda x : A_{22}[\sigma_2] : [K[\sigma_1][M_3/x][\sigma_1]]$

From previous

$\Gamma \vdash \lambda x : A_{12}[\sigma_1] \approx \lambda x : A_{22}[\sigma_2] : [\Pi x : A_1, K]$

By definition of relation

$\Gamma \vdash \lambda x : A_{12}[\sigma_1] \approx \lambda x : A_{22}[\sigma_2] : [K[\sigma_1][M_3/x][\sigma_1]]$

By Closure Under Expansion

$\Gamma \vdash \lambda x : A_{12}[\sigma_1] \approx \lambda x : A_{22}[\sigma_2] : [\Pi x : A_1, K]$

By definition of relation

16
\[
\begin{align*}
\text{Case 5:} & \quad \Gamma \vdash A_1 = A_2 : \Pi x : A_3, K \quad \Gamma \vdash M_1 = M_2 : A_3 \\
\vdash \Gamma \vdash A_1 = A_2 : \Pi x : A_3, K & \quad \vdash \Gamma \vdash M_1 = M_2 : K[M_1/x] \\
\Gamma_1 \vdash A_1[\sigma_1] \Leftrightarrow A_2[\sigma_2] : [\Pi x : A_3[\sigma_1], K[\sigma_1]] & \quad \text{By induction} \\
\Gamma_1 \vdash M_1[\sigma_1] = M_2[\sigma_2] : A_3[\sigma_1] & \quad \text{By Functionality of Equality} \\
\Gamma_1 \vdash (A_1[\sigma_1] / (M_1[\sigma_1]) \Leftrightarrow (A_2[\sigma_2] / (M_2[\sigma_2]) : [(K[LFobj_1[\sigma_1]/x])[\sigma_1]) & \quad \text{By definition of relation} \\
\Gamma_1 \vdash (A_1 M_1)[\sigma_1] \Leftrightarrow (A_2 M_2)[\sigma_2] : [(K[LFobj_1[\sigma_1]/x])[\sigma_1]) & \quad \text{By definition of substitution} \\
\Gamma_1 \vdash A_1 = A_2 : \Pi x : A_3, K \quad \Gamma_1 \vdash M_1 = M_2 : A_3 & \quad \text{By definition of relation} \\
\end{align*}
\]
Theorem 5.8 (Injectivity of Products and Sums)
Proof
It is not obvious that the algorithm is symmetric, or transitive. Fortunately, the comparison depends only on the shape of the type or kind, which is what the erasure function gets at.

\[ \Gamma \vdash \sigma_2 = \sigma_1 : \Gamma \]
\[ \Gamma \vdash \sigma_1 = \sigma_1 : \Gamma \]
\[ \Gamma \vdash A_1[\sigma_1] \approx A_2[\sigma_1] : [K[\sigma_1]] \]
\[ \Gamma \vdash A_2[\sigma_2] \approx A_3[\sigma_2] : [K[\sigma_1]] \]
\[ \Gamma \vdash A_1[\sigma_1] \approx A_3[\sigma_2] : [K[\sigma_1]] \]

By symmetry of substitution equality
By transitivity of substitution equality
By induction
By induction
By Transitivity of Relation

<table>
<thead>
<tr>
<th>Case 8: [ \Gamma \vdash A_1 = A_2 : \text{type} \quad \Gamma \vdash A_1 : \text{type} \quad \Gamma, x:A_1 \vdash K_1 = K_2 : \text{kind} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash A_1 = A_2 : \text{type} ]</td>
</tr>
<tr>
<td>[ \Gamma \vdash A_1 : \text{type} ]</td>
</tr>
<tr>
<td>[ \Gamma, x:A_1 \vdash K_1 = K_2 : \text{kind} ]</td>
</tr>
<tr>
<td>[ \Gamma \vdash \Pi x:A_1.K_1 = \Pi x:A_2.K_2 : \text{kind} ]</td>
</tr>
</tbody>
</table>

The definition of relation at product kinds involves checking a bi-implication. We show one direction, the other is similar.

\[ \Gamma \vdash A_3 \approx A_4 : [\Pi x:A_2[\sigma_2].K_2[\sigma_2]] \]
\[ \Gamma \vdash M_1 = M_2 : A_1[\sigma_1] \]
\[ \Gamma \vdash A_1[\sigma_1] = A_2[\sigma_2] : \text{type} \]
\[ \Gamma \vdash M_1 = M_2 : A_2[\sigma_2] \]
\[ \Gamma \vdash A_3 M_1 \approx A_4 M_2 : [K_2[\sigma_2][M_1/x]] \]
\[ \Gamma \vdash \sigma_1, M_1/x = \sigma_2, M_1/x : \Gamma, x:A \]
\[ \Gamma \vdash K_1[\sigma_1, M_1/x] \approx K_2[\sigma_2, M_1/x] : [\text{kind}] \]
\[ \Gamma \vdash A_3 M_1 \approx A_4 M_2 : [K_1[\sigma_1][M_1/x]] \]
\[ \Gamma \vdash A_4 \approx A_4 : [\Pi x:A_1[\sigma_1].K_1[\sigma_1]] \]

By Definition of relation
By definition of relation
By Extending Substitution
By Induction
By Induction
By definition of relation
By definition of relation
New hypothesis
New hypothesis
By Functionality of Equality
By rule (type conversion)
By definition of relation
By definition of relation
By definition of relation

We can now prove the injectivity property we are interested in.

Theorem 5.8 (Injectivity of Products and Sums)

1. If \( \Gamma \vdash \Pi x:A_1.A_{12} = \Pi x:A_2.A_{22} : \text{type} \) then \( \Gamma \vdash A_{11} = A_{21} : \text{type} \) and \( \Gamma, x:A_{11} \vdash A_{12} = A_{22} : \text{type} \).

2. If \( \Gamma \vdash \Sigma x:A_1.A_{12} = \Sigma x:A_2.A_{22} : \text{type} \) then \( \Gamma \vdash A_{11} = A_{21} : \text{type} \) and \( \Gamma, x:A_{11} \vdash A_{12} = A_{22} : \text{type} \).

Proof

Direct, from the previous lemma and definition of the relation. We show one case, the other is analogous

Case 1: \( \Gamma \vdash \Pi x:A_1.A_{12} = \Pi x:A_2.A_{22} : \text{type} \)

\[ \Gamma \vdash \text{id}_{\Gamma} = \text{id}_{\Gamma} : \Gamma \]
\[ \Gamma \vdash (\Pi x:A_{11}.A_{12})[\text{id}_{\Gamma}] \approx (\Pi x:A_{21}.A_{22})[\text{id}_{\Gamma}] : [\text{type}] \]
\[ \Gamma \vdash \Pi x:A_{11}.A_{12} \approx \Pi x:A_{21}.A_{22} : [\text{type}] \]
\[ \Gamma \vdash A_{11} = A_{21} : \text{type} \]
\[ \Gamma, x:A_{11} \vdash A_{12} = A_{22} : \text{type} \]

By reflexivity of substitutions
By Definitionally Equal Terms Related
From previous
By definition of relation

\[ \square \]

6 Equality Algorithm

An algorithm for deciding equality is given following Harper and Pfenning [5]. This uses “erased” types and kinds to direct the comparison. The problem with having unerased types and kinds is that since we have dependent types, we have to pick some object to substitute in for the application or projection cases. Then it is not obvious that the algorithm is symmetric, or transitive. Fortunately, the comparison depends only on the shape of the type or kind, which is what the erasure function gets at.
6.1 Erasure

We erase all dependencies in families and kinds for the algorithm. This process identifies types that differ only in the terms appearing in them.

Simple Kinds \( \kappa ::= \text{type}^- \quad \text{simple kind of types} \)
\[ | \quad \tau \to \kappa \quad \text{simple arrow kind} \]

Simple Types \( \tau ::= \alpha \quad \text{simple type constants} \)
\[ | \quad \tau_1 \to \tau_2 \quad \text{simple arrow type} \]
\[ | \quad \tau_1 \times \tau_2 \quad \text{simple product type} \]

Simple Contexts \( \Delta ::= \cdot \quad \text{empty} \)
\[ | \quad \Delta, x: \tau \quad \text{context extension} \]

The erasure function is defined as follows:

\[
(a)^- = \alpha \\
(\lambda x:A_1.A_2)^- = A_2^- \\
(\Pi x:A.K)^- = A^- \to K^- \\
(\Sigma x:A_1.A_2)^- = A_1^- \times A_2^- \\
(\cdot)^- = \cdot \\
(\Gamma, x:A)^- = \Gamma^- \times x:A^- \\
\]

We will need some basic facts about erasure. We notice that erasure eliminates dependencies on terms, and further, for type-level abstractions and eliminations, erasure returns just the head family. Given this informal description, it is not surprising that erasure is invariant on substitution, and further, definitionally equal types and kinds erase to the same simple type and kind.

Lemma 6.1 (Erasure Preservation)
1. For any \( A, x \) and \( M \), \( (A[M/x])^- = A^- \).
2. For any \( K, x \) and \( M \), \( (K[M/x])^- = K^- \).

Proof
By induction on the structure of the type family or kind in the premise.

Lemma 6.2 (Erasure Preservation of Equality)
1. If \( \Gamma \vdash A_1 = A_2 : K \) then \( A_1^- = A_2^- \).
2. If \( \Gamma \vdash K_1 = K_2 : \text{kind} \) then \( K_1^- = K_2^- \).

Proof
By induction on the derivation of equality.

Since we will case analyze on erased types, we need to say something about what type families erase to a particular erased type. We notice that a variety of equal types can erase to the same erased type, but fortunately, all of these have the same weak head normal form.

Lemma 6.3
1. If \( \Gamma \vdash A : \text{type}, A^- = 1^- \) and \( A \xrightarrow{\text{whr}} \), then \( A = 1 \).
2. If \( \Gamma \vdash A : \text{type}, A^- = \tau_1 \to \tau_2 \) and \( A \xrightarrow{\text{whr}} \), then \( A = \Pi x:A_1.A_2 \).
3. If \( \Gamma \vdash A : \text{type}, A^- = \tau_1 \times \tau_2 \) and \( A \xrightarrow{\text{whr}} \), then \( A = \Sigma x:A_1.A_2 \).

Proof
By inspection of the construction of \( A \).
6.2 Algorithmic Equality

We are now in a position to give the algorithm. This is given by five mutually recursive judgments. The idea of the algorithm is that extensionality is used whenever terms are compared at non-atomic types, to drive the type at which comparison is done to an atomic type. Once at atomic type, the comparison is done by weak-head normalizing both sides and comparing structurally. The structural phase compares the primitive head of the term, and uses the type-directed phase for terms down the spine. An analogous process is carried out at the level of type families.

6.2.1 Judgements

\[ \Delta \vdash M_1 \iff M_2 : \tau \quad \text{Type Directed Object Equality} \]

\[ \Delta \vdash M_1 \iff M_2 : \tau \quad \text{Structural Object Equality} \]

\[ \Delta \vdash \tau_1 \iff \tau_2 : \kappa \quad \text{Kind Directed Type Equality} \]

\[ \Delta \vdash \tau_1 \iff \tau_2 : \kappa \quad \text{Structural Type Equality} \]

\[ \Delta \vdash \kappa_1 \iff \kappa_2 : \text{kind}^- \quad \text{Structural Kind Equality} \]

6.3 Inference Rules

\[ \Delta \vdash M_1 \iff M_2 : \tau \]

\[ \Delta, x : \tau_1 \vdash M_1 \iff M_2 : \tau_2 \]

\[ \Delta \vdash \pi_i M_1 \iff \pi_i M_2 : \tau_i \]

\[ \Delta \vdash \pi_{11} M_{11} \iff \pi_{12} M_{12} : \tau_2 \]

\[ \Delta \vdash \pi_{11} M_{11} \iff \pi_{12} M_{12} : \tau_1 \]

\[ \Delta \vdash \pi_{11} M_{11} \iff \pi_{12} M_{12} : \tau_1 \times \tau_2 \]

\[ \Delta \vdash \pi_i M_1 \iff \pi_i M_2 : \tau_i \]

\[ \Delta \vdash \pi_{i1} M_{i1} \iff \pi_{i2} M_{i2} : \tau_i \]

\[ \Delta \vdash A_1 \iff A_2 : \kappa \]

\[ \Delta \vdash A_1 \iff A_2 : \text{type}^- \]

\[ \Delta \vdash A_1 \iff A_2 : \text{type}^- \]

\[ \Delta \vdash A_1 \iff A_2 : \text{type}^- \]

\[ \Delta \vdash A_1 \iff A_2 : \text{type}^- \]

\[ \Delta \vdash A_1 \iff A_2 : \tau \iff \kappa \]
We will show this algorithm to be sound and complete for the equality judgments. We will need some basic facts about the algorithm, such as that weakening of the context is allowed, and the fact that the algorithm is deterministic.

**Lemma 6.4 (Weakening of Algorithmic Equality)** For all algorithmic judgments $\mathcal{J}$, if $\Delta \vdash \mathcal{J}$ and $\Delta \subseteq \Delta^+$, then $\Delta^+ \vdash \mathcal{J}$.

**Proof**

By induction on the first judgment.

**Lemma 6.5 (Determinacy of Algorithmic Equality)**

1. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau$ then $M_1 \xrightarrow{\text{whr}}$.
2. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau$ then $M_2 \xrightarrow{\text{whr}}$.
3. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau_1$ and $\Delta \vdash M_1 \leftrightarrow M_2 : \tau_2$ then $\tau_1 = \tau_2$.
4. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ then $A_1 \xrightarrow{\text{whr}}$.
5. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ then $A_2 \xrightarrow{\text{whr}}$.
6. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa_1$ and $\Delta \vdash A_1 \leftrightarrow A_3 : \kappa_2$ then $\kappa_1 = \kappa_2$.

**Proof**

By induction on the derivation of the premise.

**Lemma 6.6 (Erasure Preservation of Algorithmic Equality)**

1. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ then $A_1 \xrightarrow{\text{whr}} = A_2 \xrightarrow{\text{whr}}$.
2. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ then $A_1 \xrightarrow{\text{whr}} = A_2 \xrightarrow{\text{whr}}$.
3. If $A_1 \xrightarrow{\text{wfr}} A_2$ then $A_1^- = A_2^-$. 

**Proof**

By induction on the derivation. □

To prove that the algorithm is complete, in the cases for the use of symmetry and transitivity rules, we will need that the algorithm itself is symmetric and transitive.

**Lemma 6.7 (Symmetry of Algorithmic Equality)**

1. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau$ then $\Delta \vdash M_2 \leftrightarrow M_1 : \tau$.
2. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau$ then $\Delta \vdash M_2 \leftrightarrow M_1 : \tau$.
3. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ then $\Delta \vdash A_2 \leftrightarrow A_1 : \kappa$.
4. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ then $\Delta \vdash A_2 \leftrightarrow A_1 : \kappa$.
5. If $\Delta \vdash K_1 \leftrightarrow K_2 : \text{kind}^-$ then $\Delta \vdash K_2 \leftrightarrow K_1 : \text{kind}^-$. 

**Proof**

By induction on the derivation. □

**Lemma 6.8 (Transitivity of Algorithmic Equality)**

1. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau$ and $\Delta \vdash M_2 \leftrightarrow M_3 : \tau$, then $\Delta \vdash M_1 \leftrightarrow M_3 : \tau$.
2. If $\Delta \vdash M_1 \leftrightarrow M_2 : \tau$ and $\Delta \vdash M_2 \leftrightarrow M_3 : \tau$, then $\Delta \vdash M_1 \leftrightarrow M_3 : \tau$.
3. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ and $\Delta \vdash A_2 \leftrightarrow A_3 : \kappa$, then $\Delta \vdash A_1 \leftrightarrow A_3 : \kappa$.
4. If $\Delta \vdash A_1 \leftrightarrow A_2 : \kappa$ and $\Delta \vdash A_2 \leftrightarrow A_3 : \kappa$, then $\Delta \vdash A_1 \leftrightarrow A_3 : \kappa$.
5. If $\Delta \vdash K_1 \leftrightarrow K_2 : \text{kind}^-$ and $\Delta \vdash K_2 \leftrightarrow K_3 : \text{kind}^-$, then $\Delta \vdash K_1 \leftrightarrow K_3 : \text{kind}^-$. 

**Proof**

By induction on the two judgments in the premises. □

### 7 Completeness of the Algorithm

The completeness of the algorithm is proved by means of a Kripke logical relation.
7.1 A Kripke Logical Relation

The logical relation we use is defined by induction on the simple types and kinds.

- \( \Delta \vdash M_1 \text{ in } [\alpha] \) iff \( \Delta \vdash M_1 \leftrightarrow M_2 : \alpha \).
- \( \Delta \vdash M_1 \text{ in } [\tau_1 \rightarrow \tau_2] \) iff for every context \( \Delta_1 \) with \( \Delta \subseteq \Delta_1 \) and every pair of objects \( M_1' \) and \( M_2' \) such that \( \Delta_1 \vdash M_1' \text{ in } [\tau_1] \) we have \( \Delta_1 \vdash M_1 M_1' \text{ in } [\tau_2] \).
- \( \Delta \vdash M_1 \text{ in } [\tau_1 \times \tau_2] \) iff \( \Delta \vdash \pi_1 M_1 \text{ is } \pi_1 M_2 \text{ in } [\tau_1] \) and \( \Delta \vdash \pi_2 M_2 \text{ in } [\tau_2] \).
- \( \Delta \vdash M_1 \text{ in } [\mathbf{1}^-] \) always.
- \( \Delta \vdash A_1 \text{ in } [\text{type}^-] \) iff \( \Delta \vdash A_1 \leftrightarrow A_2 : \text{type}^- \).
- \( \Delta \vdash A_1 \text{ in } [\tau \rightarrow \kappa] \) iff for every context \( \Delta_1 \) with \( \Delta \subseteq \Delta_1 \) and every pair of objects \( M_1 \) and \( M_2 \) such that \( \Delta_1 \vdash M_1 \) is \( M_2 \) in \( [\tau] \) we have \( \Delta_1 \vdash A_1 M_1 \text{ is } A_2 M_2 \text{ in } [\kappa] \).
- \( \Delta \vdash \sigma_1 = \sigma_2 \text{ in } [-] \) iff \( \sigma_1 = \sigma_2 = . \).
- \( \Delta \vdash [\sigma_1 | M_1 / x] \text{ in } [\Delta, x : \tau] \) iff \( \Delta \vdash \sigma_1 = \sigma_2 = . \text{ in } [\Delta] \) and \( \Delta \vdash M_1 = M_2 \) in \( [\tau] \).

**Lemma 7.1 (Logically Related Terms are Algorithmically Equal [Fundamental Lemma])**

1. If \( \Delta \vdash M_1 \text{ in } [\tau] \) then \( \Delta \vdash M_1 \leftrightarrow M_2 : \tau \).
2. If \( \Delta \vdash A_1 \text{ in } [\kappa] \) then \( \Delta \vdash A_1 \leftrightarrow A_2 : \kappa \).
3. If \( \Delta \vdash M_1 \leftrightarrow M_2 : \tau \) then \( \Delta \vdash M_1 \text{ in } [\tau] \).
4. If \( \Delta \vdash A_1 \leftrightarrow A_2 : \kappa \) then \( \Delta \vdash A_1 \text{ in } [\kappa] \).

**Proof**

By simultaneous structural induction on the simple types and kinds in the premises. We will show parts 1 and 3, to do with terms. The argument for type families is similar.

**Case 1:** Part (1), \( \tau = \alpha \).

By definition of relation, \( \Delta \vdash M_1 \leftrightarrow M_2 : \alpha \).

**Case 2:** Part (1), \( \tau = \tau_1 \rightarrow \tau_2 \).

\[
\begin{align*}
\Delta \vdash M_1 \text{ in } [\tau_1 \rightarrow \tau_2] & \quad \text{By assumption} \\
\Delta, x : \tau_1 \vdash x \leftrightarrow x : \tau_1 & \quad \text{By rule (variable)} \\
\Delta, x : \tau_1 \vdash x = x : \tau_1 & \quad \text{By rule (products)} \\
\Delta \vdash \pi_1 M_1 \text{ in } [\tau_1] & \quad \text{By definition of relation} \\
\Delta \vdash \pi_2 M_2 \text{ in } [\tau_2] & \quad \text{By definition of relation} \\
\Delta \vdash \pi_1 M_1 \leftrightarrow \pi_1 M_2 : \tau_1 & \quad \text{By induction (part 1)} \\
\Delta \vdash \pi_2 M_1 \leftrightarrow \pi_2 M_2 : \tau_2 & \quad \text{By induction (part 1)} \\
\Delta \vdash M_1 \leftrightarrow M_2 : \tau_1 \rightarrow \tau_2 & \quad \text{By rule (functions)}
\end{align*}
\]

**Case 3:** Part (1), \( \tau = \tau_1 \times \tau_2 \).

\[
\begin{align*}
\Delta \vdash M_1 \text{ in } [\tau_1 \times \tau_2] & \quad \text{By assumption} \\
\Delta \vdash \pi_1 M_1 = \pi_1 M_2 \text{ in } [\tau_1] & \quad \text{By definition of relation} \\
\Delta \vdash \pi_2 M_1 = \pi_2 M_2 \text{ in } [\tau_2] & \quad \text{By definition of relation} \\
\Delta \vdash \pi_1 M_1 \leftrightarrow \pi_1 M_2 : \tau_1 & \quad \text{By induction (part 1)} \\
\Delta \vdash \pi_2 M_1 \leftrightarrow \pi_2 M_2 : \tau_2 & \quad \text{By induction (part 1)} \\
\Delta \vdash M_1 \leftrightarrow M_2 : \tau_1 \times \tau_2 & \quad \text{By rule (products)}
\end{align*}
\]
Case 4: Part (1), $\tau = 1^-$. 
\[ \Delta \vdash M_1 \iff M_2 : 1^- \] 
By rule (unit)

Case 5: Part (3), $\tau = \alpha$.
\[ \Delta \vdash M_1 \iff M_2 : \alpha \] 
By assumption
\[ \Delta \vdash M_1 \iff M_2 : \alpha \] 
By rule (constants)
\[ \Delta \vdash M_1 \text{ is } M_2 \text{ in } [0]\] 
By definition of relation

Case 6: Part (3), $\tau = \tau_1 \rightarrow \tau_2$.
\[ \Delta \vdash M_1 \iff M_2 : \tau_1 \rightarrow \tau_2 \] 
By assumption
\[ \Delta_1 \vdash M_{11} \text{ is } M_{21} \text{ in } [\tau_1] \text{ for an arbitrary } \Delta_1 \supseteq \Delta \] 
New hypothesis
\[ \Delta_1 \vdash M_{11} \iff M_{21} : \tau_1 \] 
By induction (part 1)
\[ \Delta \vdash M_1 \iff M_2 : \tau_1 \rightarrow \tau_2 \] 
By Weakening
\[ \Delta_1 \vdash M_1, M_{11} \iff M_2, M_{21} : \tau_2 \] 
By rule (application)
\[ \Delta_1 \vdash M_1, M_{11} \text{ is } M_2, M_{21} \text{ in } [\tau_2] \] 
By induction (part 3) on smaller type $\tau_2$
\[ \Delta \vdash M_1 \text{ is } M_2 \text{ in } [\tau_1 \rightarrow \tau_2] \] 
By definition of rule

Case 7: Part (3), $\tau = \tau_1 \times \tau_2$.
\[ \Delta \vdash M_1 \iff M_2 : \tau_1 \times \tau_2 \] 
By assumption
\[ \Delta \vdash \pi_i M_1 \iff \pi_i M_2 : \tau_i \] 
By rule (projection)
\[ \Delta \vdash \pi_i M_1 \text{ is } \pi_i M_2 \text{ in } [\tau_i] \] 
By induction (part 3)
\[ \Delta \vdash M_1 \text{ is } M_2 \text{ in } [\tau_1 \times \tau_2] \] 
By definition of relation

Case 8: Part (3), $\tau = 1^-$. 
\[ \Delta \vdash M_1 \text{ is } M_2 \text{ in } [1^-] \] 
By definition of relation

Similar to the simpler logical relations argument for proving injectivity, we will need closure under head expansion for proving completeness for the abstraction case. We will also need the symmetry and transitivity of the logical relations, which will use the corresponding properties at base types and lift to function and product types.

**Lemma 7.2 (Closure Under Head Expansion)**

1. If $M_{11} \text{ whr } M_{12}$ and $\Delta \vdash M_{12}$ is $M_2$ in $[\tau]$ then $\Delta \vdash M_{11}$ is $M_2$ in $[\tau]$.

2. If $A_{11} \text{ whr } A_{12}$ and $\Delta \vdash A_{12}$ is $A_2$ in $[\kappa]$ then $\Delta \vdash A_{11}$ is $A_2$ in $[\kappa]$.

3. If $M_{21} \text{ whr } M_{22}$ and $\Delta \vdash M_1$ is $M_2$ in $[\tau]$ then $\Delta \vdash M_1$ is $M_2$ in $[\tau]$.

4. If $A_{21} \text{ whr } A_{22}$ and $\Delta \vdash A_1$ is $A_2$ in $[\kappa]$ then $\Delta \vdash A_1$ is $A_2$ in $[\kappa]$.

**Proof**
By induction on the simple type or kind in the premises.

Lemma 7.3 (Weakening of Logical Relations)

1. If $\Delta \vdash M \in [\tau]$ and $\Delta \subseteq \Delta^+$, then $\Delta^+ \vdash M \in [\tau]$.
2. If $\Delta \vdash A \in [\kappa]$ and $\Delta \subseteq \Delta^+$, then $\Delta^+ \vdash A \in [\kappa]$.
3. If $\Delta \vdash \sigma_1 = \sigma_2 \in [\Delta_1]$ and $\Delta \subseteq \Delta^+$, then $\Delta^+ \vdash \sigma_1 = \sigma_2 \in [\Delta_1]$.

Proof

By induction on the simple type, kind or context in the premise.

Lemma 7.4 (Symmetry of Logical Relations)

1. If $\Delta \vdash M \in [\tau]$ then $\Delta \vdash M \in [\tau]$.
2. If $\Delta \vdash A \in [\kappa]$ then $\Delta \vdash A \in [\kappa]$.
3. If $\Delta \vdash \sigma_1 = \sigma_2 \in [\Delta_1]$ then $\Delta \vdash \sigma_1 = \sigma_2 \in [\Delta_1]$.

Proof

By induction on the simple type, kind or context in the premise. We will show the cases for objects. The other cases are similar.

Case 1: $\tau = \alpha$

$\Delta \vdash M_1 \in [\alpha]$

$\Delta \vdash M_1 \iff M_2 : \alpha$

$\Delta \vdash M_2 \iff M_1 : \alpha$

$\Delta \vdash M_2 \in [\alpha]$

By assumption

By definition of relation

By Symmetry of Algorithm

By definition of relation

Case 2: $\tau = \tau_1 \rightarrow \tau_2$

$\Delta \vdash M_1 \in [\tau_1 \rightarrow \tau_2]$

$\Delta^+ \vdash M_{11} \in [\tau_1]$ for $\Delta^+ \supseteq \Delta$

$\Delta^+ \vdash M_{11} \in [\tau_1]$

$\Delta^+ \vdash M_{21} \in [\tau_2]$

$\Delta^+ \vdash M_{21} \in [\tau_2]$

$\Delta \vdash M_2 \in [\tau_1 \rightarrow \tau_2]$

By assumption

New hypothesis

By definition of relation

By induction

By definition of relation

Case 3: $\tau = \tau_1 \times \tau_2$

$\Delta \vdash M_1 \in [\tau_1 \times \tau_2]$

$\Delta \vdash \pi_i M_1 \in [\tau_i]$

$\Delta \vdash \pi_i M_2 \in [\tau_i]$

$\Delta \vdash M_2 \in [\tau_1 \rightarrow \tau_2]$

By assumption

By definition of relation

By induction

By definition of relation

Case 4: $\tau = 1^-$

25
Lemma 7.5 (Transitivity of Logical Relations)

1. If \( \Delta \vdash M_1 \text{ is } M_2 \text{ in } [\tau] \) and \( \Delta \vdash M_2 \text{ is } M_3 \text{ in } [\tau] \), then \( \Delta \vdash M_1 \text{ is } M_3 \text{ in } [\tau] \).

2. If \( \Delta \vdash A_1 \text{ is } A_2 \text{ in } [\kappa] \) and \( \Delta \vdash A_2 \text{ is } A_3 \text{ in } [\kappa] \), then \( \Delta \vdash A_1 \text{ is } A_3 \text{ in } [\kappa] \).

3. If \( \Delta \vdash \sigma_1 \text{ is } \sigma_2 \text{ in } [\Delta_1] \) and \( \Delta \vdash \sigma_2 \text{ is } \sigma_3 \text{ in } [\Delta_1] \), then \( \Delta \vdash \sigma_1 \text{ is } \sigma_3 \text{ in } [\Delta_1] \).

Proof

By induction on the simple type, kind or context in the premise. We will show the cases for objects. The other cases are similar.

Case 1: \( \tau = \alpha \)

\[
\begin{align*}
\Delta \vdash M_1 \text{ is } M_2 \text{ in } [\alpha] & \quad \text{By assumption} \\
\Delta \vdash M_2 \text{ is } M_3 \text{ in } [\alpha] & \quad \text{By assumption} \\
\Delta \vdash M_1 \iff M_2 : \alpha & \quad \text{By definition of relation} \\
\Delta \vdash M_2 \iff M_3 : \alpha & \quad \text{By definition of relation} \\
\Delta \vdash M_1 \iff M_3 : \alpha & \quad \text{By Transitivity of Algorithm} \\
\Delta \vdash M_1 \text{ is } M_3 \text{ in } [\alpha] & \quad \text{By definition of relation}
\end{align*}
\]

Case 2: \( \tau = \tau_1 \rightarrow \tau_2 \)

\[
\begin{align*}
\Delta \vdash M_1 \text{ is } M_2 \text{ in } [\tau_1 \rightarrow \tau_2] & \quad \text{By assumption} \\
\Delta \vdash M_2 \text{ is } M_3 \text{ in } [\tau_1 \rightarrow \tau_2] & \quad \text{By assumption} \\
\Delta^+ \vdash M_{11} \text{ is } M_{21} \text{ in } [\tau_1] & \quad \text{for } \Delta^+ \supseteq \Delta \\
\Delta^+ \vdash M_{11} \text{ in } [\tau_1] & \quad \text{By Symmetry of Relation} \\
\Delta^+ \vdash M_{21} \text{ in } [\tau_1] & \quad \text{By induction} \\
\Delta^+ \vdash M_{11} \text{ is } M_{21} \text{ in } [\tau_2] & \quad \text{By definition of relation} \\
\Delta^+ \vdash M_{21} \text{ is } M_{31} \text{ in } [\tau_2] & \quad \text{By definition of relation} \\
\Delta^+ \vdash M_{11} \text{ is } M_{31} \text{ in } [\tau_2] & \quad \text{By induction} \\
\Delta^+ \vdash M_1 \text{ is } M_3 \text{ in } [\tau_1 \rightarrow \tau_2] & \quad \text{By definition of relation}
\end{align*}
\]

Case 3: \( \tau = \tau_1 \times \tau_2 \)

\[
\begin{align*}
\Delta \vdash M_1 \text{ is } M_2 \text{ in } [\tau_1 \times \tau_2] & \quad \text{By assumption} \\
\Delta \vdash M_2 \text{ is } M_3 \text{ in } [\tau_1 \times \tau_2] & \quad \text{By assumption} \\
\Delta \vdash \pi_i M_1 \text{ is } \pi_i M_2 \text{ in } [\tau_i] & \quad \text{By definition of relation} \\
\Delta \vdash \pi_i M_2 \text{ is } \pi_i M_3 \text{ in } [\tau_i] & \quad \text{By definition of relation} \\
\Delta \vdash \pi_i M_1 \text{ is } \pi_i M_3 \text{ in } [\tau_i] & \quad \text{By definition of relation} \\
\Delta \vdash M_1 \text{ is } M_3 \text{ in } [\tau_1 \rightarrow \tau_2] & \quad \text{By definition of relation}
\end{align*}
\]

Case 4: \( \tau = 1^- \)

\[
\begin{align*}
\Delta \vdash M_1 \text{ is } M_3 \text{ in } [1^-] & \quad \text{By definition of relation}
\end{align*}
\]
Lemma 7.6 (Definitionally Equal Terms are Logically Related Under Substitutions)

1. If $\Gamma \vdash M_1 = M_2 : A$ and $\Delta \vdash \sigma_1$ is $\sigma_2$ in $[\Gamma^-]$ then $\Delta \vdash M_1[\sigma_1]$ is $M_2[\sigma_2]$ in $[\Lambda^-]$.

2. If $\Gamma \vdash A_1 = A_2 : K$ and $\Delta \vdash \sigma_1$ is $\sigma_2$ in $[\Gamma^-]$ then $\Delta \vdash A_1[\sigma_1]$ is $A_2[\sigma_2]$ in $[K^-]$.

Proof

By induction on the judgment of definitional equality. We will show some representative cases.

Case 1: $\Gamma(x) = A$

$\Gamma \vdash x = x : A$

$\Delta \vdash \sigma_1$ is $\sigma_2$ in $[\Gamma^-]$  
By assumption

$\Delta \vdash M_1$ is $M_2$ in $[\Lambda^-]$  
for $[M_1/x] \in \sigma_1$, $[M_2/x] \in \sigma_2$  
By definition of relation

$\Delta \vdash x[\sigma_1]$ is $x[\sigma_2]$ in $[\Lambda^-]$  
By definition of substitution

Case 2: $\Sigma(c) = A$

$\Gamma \vdash c = c : A$

By rule (constant)

$\Delta \vdash c \leftrightarrow c : A^-$  
By Fundamental Theorem

$\Delta \vdash c \in [\Lambda^-]$  
By definition of substitution

$\Delta \vdash c[\sigma_1]$ is $c[\sigma_2]$ in $[\Lambda^-]$  
By definition of substitution

Case 3: $\Gamma \vdash M_{11} = M_{21} : \Pi x A_2, A_1  \quad \Gamma \vdash M_{12} = M_{22} : A_2$

$\Delta \vdash M_{11}[\sigma_1]$ is $M_{21}[\sigma_2]$ in $[A_2^- \rightarrow A_1^-]$  
By induction

$\Delta \vdash M_{12}[\sigma_1]$ is $M_{22}[\sigma_2]$ in $[A_2^-]$  
By induction

$\Delta \vdash (M_{11}[\sigma_1]) (M_{12}[\sigma_2])$ is $(M_{21}[\sigma_2]) (M_{22}[\sigma_2])$ in $[A_1^-]$  
By definition of relation

$\Delta \vdash (M_{11} M_{12}[\sigma_1])$ is $(M_{21} M_{22}[\sigma_2])$ in $[A_1^-]$  
By definition of substitution

Case 4: $\Gamma \vdash A_{11} = A_1 : \text{type}  \quad \Gamma \vdash A_{12} = A_1 : \text{type}  \quad \Gamma, x : A_1 \vdash M_1 = M_2 : A_2$

$\Gamma \vdash \lambda x : A_1, M_1 = \lambda x : A_1, M_2 : \Pi x A_1, A_2$

$\Delta^+ \vdash M_{11}$ is $M_{21}$ in $[A_1^-]$  
for $\Delta^+ \supseteq \Delta$  
New hypothesis

$\Delta^+ \vdash \sigma_1$ is $\sigma_2$ in $[\Gamma^-]$  
By weakening of relation

$\Delta^+ \vdash \sigma_1, M_{11}/x$ is $\sigma_2, M_{21}/x$ in $[\Gamma^-, x : A_1^-]$  
By definition of relation

$\Delta^+ \vdash M_{11}[\sigma_1, M_{11}/x]$ is $M_{21}[\sigma_2, M_{21}/x]$ in $[A_2^-]$  
By induction

$\Delta^+ \vdash (\lambda x : A_1, M_{11}[\sigma_1, M_{11}/x]) \downarrow M_{11}$ is $M_{21}[\sigma_2, M_{21}/x]$ in $[A_2^-]$  
By Closure Under Head Expansion

$\Delta^+ \vdash (\lambda x : A_1, M_{11}[\sigma_1, M_{11}/x]) M_{11}$ is $(\lambda x : A_1, M_1[\sigma_2]) M_{21}$ in $[A_2^-]$  
By Closure Under Head Expansion

$\Delta^+ \vdash ((\lambda x : A_1, M_{11}[\sigma_1]) M_{11})$ is $((\lambda x : A_1, M_2[\sigma_2]) M_{21}$ in $[A_2^-]$  
By definition of substitution

$\Delta^+ \vdash (\lambda x : A_1, M_1[\sigma_1]) M_1$ is $(\lambda x : A_2, M_2[\sigma_2]) M_2$ in $[A_2^-]$  
By definition of relation
Case 5:

\[\frac{\Gamma \vdash A_1 : \text{type} \quad \Gamma, x : A_1 \vdash M_1 x = M_2 x : A_2}{\Gamma \vdash M_1 = M_2 : \Pi x : A_1. A_2}\]

- \(\Delta \vdash M_1\) is \(M_2\) in \([A_1]^-\) for \(\Delta \supseteq \Delta\)
- \(\Delta \vdash \sigma_2\) in \([\Gamma^-]\)
- \(\Delta \vdash \sigma_1, M_{11/x} \rightarrow \sigma_2, M_{21/x} \in [\Gamma^- : x : A_1^-]\)
- \(\Delta \vdash (M_1 x)[\sigma_1, M_{11/x}] \rightarrow (M_2 x)[\sigma_2, M_{21/x}] \in [A_2^-]\)
- \(\Delta \vdash M_1[\sigma_1] \rightarrow M_2[\sigma_2] \in [A_1^- \rightarrow A_2^-]\)

New hypothesis
By weakening of relation
By definition of relation
By definition of relation
By definition of substitution
By induction

Case 6:

\[\text{\vdash \quad \Gamma, x : A_1 \vdash M_{12 : A_2} \quad \Gamma \vdash M_{11 : A_2}}\]

- \(\Delta \vdash \sigma_1\) in \([\Gamma^-]\)
- \(\Delta \vdash M_{11} \rightarrow M_{22} \in [A_1^-]\)
- \(\Delta \vdash M_{12[\sigma_1]} / \sigma_2, (M_{21[\sigma_2]}) / x \in [\Gamma^- : x : A_1^-]\)
- \(\Delta \vdash (M_{12[\sigma_1]}) / \sigma_2, (M_{21[\sigma_2]}) / x \in [A_2^-]\)
- \(\Delta \vdash (M_{12[\sigma_1], x/y})[(M_{11[\sigma_1]}) / x] \rightarrow (M_{22[\sigma_2], (M_{21[\sigma_2]}) / x} \in [A_2^-]\)
- \(\Delta \vdash (\lambda x : A_1. M_{12[\sigma_1, x/y]})(M_{11[\sigma_1]}) / x \rightarrow (M_{22[\sigma_2], (M_{21[\sigma_2]}) / x}) \in [A_2^-]\)
- \(\Delta \vdash (\lambda x : A_1. M_{12[\sigma_1]}) \rightarrow (M_{22[\sigma_2], (M_{21[\sigma_2]}) / x}) \in [A_2^-]\)

By assumption
By induction
By definition of relation
By definition of substitution
By definition of substitution
By definition of substitution
By Erasure Preservation

Case 7:

\[\text{\vdash \quad \Gamma \vdash \Sigma x : A_1. A_2 : \text{type}}\]

- \(\Delta \vdash M_{11} \rightarrow M_{22} \in [A_1^-]\)
- \(\Delta \vdash M_{12[\Sigma x : A_1. A_2]} \rightarrow M_{21[22]} \in [A_1^-]\)

By induction
By definition Under Head Expansion
By definition Under Head Expansion
By definition Under Head Expansion
By definition Under Head Expansion
By definition Under Head Expansion
By definition Under Head Expansion
By definition of substitution
By definition of relation

Case 8:

\[\text{\vdash \quad \Gamma \vdash M_1 \rightarrow M_2 : \Sigma x : A_1. A_2}\]

- \(\Delta \vdash M_1[\sigma_0] \rightarrow M_2[\sigma_2] \in [A_1^- \times A_2^-]\)
- \(\Delta \vdash \pi_1(M_1[\sigma_0]) \rightarrow \pi_1(M_2[\sigma_2]) \in [A_1^-]\)
- \(\Delta \vdash \pi_1(M_1[\sigma_0]) \rightarrow \pi_1(M_2[\sigma_2]) \in [A_1^-]\)

By induction
By definition of relation
By definition of substitution
Case 9: \[ \Gamma \vdash M_1 = M_2 : \Sigma x A_1. A_2 \]
\[ \Gamma \vdash \pi_2 M_1 = \pi_2 M_2 : A_2 [\pi_1 M_1 / x] \]
\[ \Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [A_1^- \times A_2^-] \]
\[ \Delta \vdash \pi_2(M_1[\sigma_1]) \text{ is } \pi_2(M_2[\sigma_2]) \text{ in } [A_2^-] \]
\[ \Delta \vdash (\pi_2 M_1)[\sigma_1] \text{ is } (\pi_2 M_2)[\sigma_2] \text{ in } [A_2^-] \]
By induction
By definition of relation
By definition of substitution

Case 10: \[ \Gamma \vdash M_1 : 1 \quad \Gamma \vdash M_2 : 1 \]
\[ \Gamma \vdash M_1 = M_2 : 1 \]
\[ \Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [1^-] \]
By definition of relation

Case 11: \[ \Gamma \vdash M_1 = M_3 : A_1 \quad \Gamma \vdash M_2 : A_2 \]
\[ \Gamma \vdash \pi_1(M_1, M_2)^A = M_3 : A_1 \]
\[ \Delta \vdash M_1[\sigma_1] \text{ is } M_3[\sigma_2] \text{ in } [A_1^-] \]
\[ \Delta \vdash \pi_1((M_1[\sigma_1]), (M_2[\sigma_1]))(\langle x \rangle) \text{ is } M_3[\sigma_2] \text{ in } [A_1^-] \]
\[ \Delta \vdash (\pi_1(M_1, M_2)^A)[\sigma_1] \text{ is } M_3[\sigma_2] \text{ in } [A_1^-] \]
By induction
By Closure Under Head Expansion
By definition of substitution

Case 12: \[ \Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 = M_3 : A_2 \]
\[ \Gamma \vdash \pi_2(M_1, M_2)^A = M_3 : A_2 \]
\[ \Delta \vdash M_2[\sigma_1] \text{ is } M_3[\sigma_2] \text{ in } [A_2^-] \]
\[ \Delta \vdash \pi_2((M_1[\sigma_1]), (M_2[\sigma_1]))(\langle x \rangle) \text{ is } M_3[\sigma_2] \text{ in } [A_2^-] \]
\[ \Delta \vdash (\pi_2(M_1, M_2)^A)[\sigma_1] \text{ is } M_3[\sigma_2] \text{ in } [A_2^-] \]
By induction
By Closure Under Head Expansion
By definition of substitution

Case 13: \[ \Gamma \vdash M_1 : \Sigma x A_1. A_2 \quad \Gamma \vdash M_2 : \Sigma x A_1. A_2 \quad \Gamma \vdash \pi_1 M_1 = \pi_1 M_2 : A_1 \quad \Gamma \vdash \pi_2 M_1 = \pi_2 M_2 : A_2 [\pi_1 M_1 / x] \]
\[ \Gamma \vdash M_1 = M_2 : \Sigma x A_1. A_2 \]
\[ \Delta \vdash (\pi_1 M_1)[\sigma_1] \text{ is } (\pi_1 M_2)[\sigma_2] \text{ in } [A_1^-] \]
\[ \Delta \vdash (\pi_2 M_1)[\sigma_1] \text{ is } (\pi_2 M_2)[\sigma_2] \text{ in } [A_2^-] \]
\[ \Delta \vdash \pi_1(M_1[\sigma_1]) \text{ is } \pi_1(M_2[\sigma_2]) \text{ in } [A_1^-] \]
\[ \Delta \vdash \pi_2(M_1[\sigma_1]) \text{ is } \pi_2(M_2[\sigma_2]) \text{ in } [A_2^-] \]
\[ \Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [A_1^- \times A_2^-] \]
By induction
By induction
By definition of substitution
By definition of substitution
By definition of substitution

Case 14: \[ \Gamma \vdash M_2 = M_1 : A \]
\[ \Gamma \vdash M_1 = M_2 : A \]
\[ \Delta \vdash \sigma_1 \text{ is } \sigma_2 \text{ in } [\Gamma^-] \]
\[ \Delta \vdash \sigma_1 \text{ is } \sigma_2 \text{ in } [\Gamma^-] \]
\[ \Delta \vdash M_2[\sigma_1] \text{ is } M_1[\sigma_1] \text{ in } [A^-] \]
\[ \Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [A^-] \]
By assumption
By Symmetry of Relation
By induction
By Symmetry of Relation
Case 15: \[
\Gamma \vdash M_1 = M_2 : A \quad \Gamma \vdash M_2 = M_3 : A
\]
\[
\Gamma \vdash M_1 = M_3 : A
\]
\[
\Delta \vdash \sigma_1 \text{ is } \sigma_2 \text{ in } [\Gamma^-]
\]
\[
\Delta \vdash \sigma_2 \text{ is } \sigma_1 \text{ in } [\Gamma^-]
\]
\[
\Delta \vdash \sigma_2 \text{ is } \sigma_2 \text{ in } [\Gamma^-]
\]
\[
\Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [A^-]
\]
\[
\Delta \vdash M_2[\sigma_2] \text{ is } M_3[\sigma_3] \text{ in } [A^-]
\]
\[
\Delta \vdash M_1[\sigma_1] \text{ is } M_3[\sigma_2] \text{ in } [A^-]
\]

By assumption

By Symmetry of Relation

By Transitivity of Relation

By induction

By induction

By Transitivity of Relation

Case 16: \[
\Gamma \vdash A_1 = A_2 : \text{type} \quad \Gamma \vdash M_2 = M_1 : A_2
\]
\[
\Lambda_1^- = A_2^-
\]
\[
\Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [A_2^-]
\]
\[
\Delta \vdash M_1[\sigma_1] \text{ is } M_2[\sigma_2] \text{ in } [A_1^-]
\]

By Erasure Preservation

By induction

From previous

Case 17: \[
\Gamma \vdash A_{11} = A_{21} : \text{type} \quad \Gamma \vdash A_{11} : \text{type} \quad \Gamma, x : A_{11} \vdash A_{12} = A_{22} : \text{type}
\]
\[
\Gamma \vdash \Sigma x : A_{11}, A_{12} = \Sigma x : A_{21}, A_{22} : \text{type}
\]
\[
\Delta \vdash A_{11}[\sigma_1] \text{ is } A_{21}[\sigma_2] \text{ in } [\text{type}^-]
\]
\[
\Delta \vdash A_{11}[\sigma_1] \leftrightarrow A_{21}[\sigma_2] : \text{type}^-
\]
\[
\Delta, x : A_{11} \vdash x \leftrightarrow x : A_{11}^-
\]
\[
\Delta, x : A_{11}^- \vdash x \text{ is } x \in [A_{11}^-]
\]
\[
\Delta, x : A_{11}^- \vdash x \text{ is } x \in [A_{11}^-]
\]
\[
\Delta, x : A_{11}^- \vdash x \text{ is } x \in [A_{11}^-]
\]
\[
\Delta, x : A_{11}^- \vdash A_{12}[\sigma_1, x/x] \text{ is } A_{22}[\sigma_2, x/x] \text{ in } [\text{type}^-]
\]
\[
\Delta, x : A_{11}^- \vdash A_{12}[\sigma_1, x/x] \leftrightarrow A_{22}[\sigma_2, x/x] : \text{type}^-
\]
\[
\Delta \vdash \Sigma x : A_{11}[\sigma_1], A_{12}[\sigma_1, x/x] \leftrightarrow \Sigma x : A_{21}[\sigma_2], A_{22}[\sigma_2, x/x] : \text{type}^-
\]
\[
\Delta \vdash \Sigma x : A_{11}[\sigma_1], A_{12}[\sigma_1, x/x] \leftrightarrow \Sigma x : A_{21}[\sigma_2], A_{22}[\sigma_2, x/x] \text{ in } [\text{type}^-]
\]
\[
\Delta \vdash (\Sigma x : A_{11}, A_{12})[\sigma_1] \text{ is } (\Sigma x : A_{21}, A_{22})[\sigma_2] \text{ in } [\text{type}^-]
\]

By induction

By definition of relation

By rule (variable)

By Fundamental Theorem

By definition of relation

By induction

By definition of relation

By rule (sums)

By definition of relation

By definition of substitution

Case 18: \[
\Gamma \vdash 1 = 1 : \text{type}^-
\]
\[
\Delta \vdash 1 \leftrightarrow 1 : \text{type}^-
\]
\[
\Delta \vdash 1 \text{ is } 1 \text{ in } [\text{type}^-]
\]
\[
\Delta \vdash 1[\sigma_1] \text{ is } 1[\sigma_2] \text{ in } [\text{type}^-]
\]

By rule (unit)

By definition of relation

By definition of substitution

We are almost done with proving completeness. We first need an easy lemma to show that identity substitutions are logically related to themselves.

**Lemma 7.7 (Identity Substitutions are Logically Related)** \(\Gamma^- \vdash \text{id}_\Gamma \text{ is } \text{id}_\Gamma \text{ in } [\Gamma^-].\)

**Proof**

By induction on the structure of \(\Gamma\).

□

30
Theorem 7.8 (Definitionally Equal Terms are Logically Related)

1. If $\Gamma \vdash M_1 = M_2 : A$ then $\Gamma^- \vdash M_1$ is $M_2$ in $[A^-]$.
2. If $\Gamma \vdash A_1 = A_2 : K$ then $\Gamma^- \vdash A_1$ is $A_2$ in $[K^-]$.

Proof
Direct, by Lemma 7.6 and Lemma 7.7. □

Theorem 7.9 (Completeness of Equality Algorithm)

1. If $\Gamma \vdash M_1 = M_2 : A$ then $\Gamma^- \vdash M_1 \iff M_2 : A$.
2. If $\Gamma \vdash A_1 = A_2 : K$ then $\Gamma^- \vdash A_1 \iff A_2 : K$.

Proof
Direct, by Lemma 7.8 and Lemma 7.1. □

8 Soundness of the Algorithm and Canonical Forms

We will need to prove a few simple lemmas before we prove our algorithm sound. The first one says that the weak-head reduction relation is sound. This is important because at base types the algorithm performs weak-head reductions.

Lemma 8.1 (Subject Reduction)

1. If $M_1 \xrightarrow{\text{whr}} M_2$ and $\Gamma \vdash M_1 : A$ then $\Gamma \vdash M_2 : A$ and $\Gamma \vdash M_1 = M_2 : A$.
2. If $A_1 \xrightarrow{\text{whr}} A_2$ and $\Gamma \vdash A_1 : K$ then $\Gamma \vdash A_2 : K$ and $\Gamma \vdash A_1 = A_2 : K$.

Proof
By induction on the definition of weak head reduction.

Lemma 8.2 (Inversion on Erasure)

1. If $\Gamma \vdash A : \text{type and } A^- = 1^-$ then $A \xrightarrow{\text{whr}^*} 1$ and $\Gamma \vdash A = 1 : \text{type}$.
2. If $\Gamma \vdash A : \text{type and } A^- = \tau_1 \rightarrow \tau_2$ then $A \xrightarrow{\text{whr}^*} \Pi x : A_1 . A_2$ and $\Gamma \vdash A = \Pi x : A_1 . A_2 : \text{type}$.
3. If $\Gamma \vdash A : \text{type and } A^- = \tau_1 \times \tau_2$ then $A \xrightarrow{\text{whr}^*} \Sigma x : A_1 . A_2$ and $\Gamma \vdash A = \Sigma x : A_1 . A_2 : \text{type}$.

Proof
By induction on the length of a weak head normal reduction starting from $A$.

Case 1: $\xrightarrow{\text{whr}}$
By Erasure Corresponds to Weak Head Normal Terms and Subject Reduction

Case 2: $\xrightarrow{\text{whr}} A'$
By induction and transitivity
8.1 Canonical Forms

We will actually prove soundness and canonical forms properties together for our language. We now define canonical and atomic objects, families and kinds, which are a syntactic subset of the corresponding terms, defined by the following grammar.

<table>
<thead>
<tr>
<th>Canonical Kinds</th>
<th>K :=</th>
<th>type</th>
<th>kind of types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Πx:A.K</td>
<td>dependent product kind</td>
</tr>
<tr>
<td>Atomic Families</td>
<td>A :=</td>
<td>a</td>
<td>family constants</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A M</td>
<td>family application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Πx:A_1.A_2</td>
<td>family of functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Σx:A_1.A_2</td>
<td>family of products</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>unit type</td>
</tr>
<tr>
<td>Canonical Families</td>
<td>A :=</td>
<td>A</td>
<td>atomic families</td>
</tr>
<tr>
<td></td>
<td></td>
<td>λx:A_1.Ā_2</td>
<td>family level abstraction</td>
</tr>
<tr>
<td>Atomic Objects</td>
<td>M :=</td>
<td>c</td>
<td>object constants</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
<td>object variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M_1 M_2</td>
<td>object level application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>π_i M (i = 1, 2)</td>
<td>projections from pairs</td>
</tr>
<tr>
<td>Canonical Objects</td>
<td>M :=</td>
<td>M</td>
<td>atomic objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⟨M_1, M_2⟩^A</td>
<td>pairs of objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⟨⟩</td>
<td>unit object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>λx:A M</td>
<td>object functions</td>
</tr>
</tbody>
</table>

Notice that these are different from the original notion of canonical forms. A better term might be quasi-canonical forms, or almost canonical forms, but that term is already used in Harper and Pfenning for something else. The difference from the original canonical forms is that type annotations of abstractions at both term and family levels need not be in canonical form. This is the same in spirit to the quasi-canonical form of Harper and Pfenning [5], but those forms elide type annotations on abstractions, so that the canonical forms do not belong syntactically to the language of LF.

We can now prove our main lemma, which implies both soundness of the algorithm for equality as well as existence of canonical forms.

**Lemma 8.3 (Algorithm produces Canonical Mediating Terms)**

1. If $\Gamma \vdash M_1 : A$, $\Gamma \vdash M_2 : A$ and $\Gamma \vdash M_1 \iff M_2 : A^-$, then there is a canonical object $\Bar{M}$ such that $\Gamma \vdash \Bar{M} : A$ and $\Gamma \vdash \Bar{M} = \Bar{M} : A$.

2. If $\Gamma \vdash M_1 : A_1$, $\Gamma \vdash M_2 : A_2$ and $\Gamma \vdash M_1 \iff M_2 : \tau$, then there is an atomic object $\Bar{M}$ such that $\Gamma \vdash \Bar{M} : A_1$, $\Gamma \vdash M_1 = \Bar{M} : A_1$ and $\Gamma \vdash M_2 = \Bar{M} : A_1$ and $\Gamma \vdash A_1 = A_2 : type$ and $A_1^- = A_2^- = \tau$.

3. If $\Gamma \vdash A_1 : K$, $\Gamma \vdash A_2 : K$ and $\Gamma \vdash A_1 \iff A_2 : K^-$, then there exists a canonical family $\Bar{A}$ such that $\Gamma \vdash \Bar{A} : K$, $\Gamma \vdash A_1 = \Bar{A} : K$ and $\Gamma \vdash A_2 = \Bar{A} : K$.

4. If $\Gamma \vdash A_1 : K_1$, $\Gamma \vdash A_2 : K_2$ and $\Gamma \vdash A_1 \iff A_2 : \kappa$, then there exists an atomic family $\Bar{A}$ such that $\Gamma \vdash \Bar{A} : K_1$, $\Gamma \vdash A_1 = \Bar{A} : K_1$ and $\Gamma \vdash A_2 = \Bar{A} : K_1$ and $\Gamma \vdash K_1 = K_2 : kind$ and $K_1^- = K_2^- = \kappa$.

5. If $\Gamma \vdash K_1 : kind$, $\Gamma \vdash K_2 : kind$ and $\Gamma \vdash K_1 \iff K_2 : kind^-$, then there exists a canonical kind $\Bar{K}$ such that $\Gamma \vdash \Bar{K} : kind$, $\Gamma \vdash K_1 = \Bar{K} : kind$ and $\Gamma \vdash K_2 = \Bar{K} : kind$.

**Proof**

By induction on the algorithmic judgment. We will show some representative cases.

**Case 1:**

$$\frac{M_1 \text{ whr } M_2, \Delta \vdash M_2 \iff M : \alpha}{\Delta \vdash M_1 \iff M : \alpha}$$
Case 4: \[ \Gamma \vdash M_1 = M_2 : A \]
\[ \Gamma \vdash M_2 : A \]
There exists a canonical object \( \bar{M} \) such that
\[ \Gamma \vdash \bar{M} : A \]
\[ \Gamma \vdash M_2 = \bar{M} : A \]
\[ \Gamma \vdash M = M : A \]
\[ \Gamma \vdash M_1 = M : A \]

By Subject Reduction
By Regularity

Case 2: \[ \Delta \vdash M_1 \iff M_2 : \alpha \]
\[ \Delta \vdash M_1 \iff M_2 : \alpha \]
There exists an atomic term \( \bar{M} \) such that
\[ \Gamma \vdash \bar{M} : A \]
\[ \Gamma \vdash M_1 = M : A \]
\[ \Gamma \vdash M_2 = \bar{M} : A \]

By induction

Case 3: \[ \Delta, x: \tau_1 \vdash M_1 x \iff M_2 x : \tau_2 \]
\[ \Delta \vdash M_1 \iff M_2 : \tau_1 \rightarrow \tau_2 \]

\[ \Gamma \vdash A = \Pi x: A_1.A_2 : \text{type}, \]
\[ A_1 \vdash \tau_1 \text{ and } A_2 \vdash \tau_2 \]
\[ \Gamma \vdash \Pi x: A_1.A_2 : \text{type} \]
\[ \Gamma \vdash A_1 : \text{type} \]
\[ \Gamma, x: A_1 \vdash A_2 : \text{type} \]
\[ \Gamma, x: A_1 \vdash M_1 x : A_2 \]
\[ \Gamma, x: A_1 \vdash M_2 x : A_2 \]
There exists a canonical object \( \bar{M} \) such that
\[ \Gamma, x: A_1 \vdash M : A_2 \]
\[ \Gamma, x: A_1 \vdash M_1 x = M : A_2 \]
\[ \Gamma, x: A_1 \vdash M_2 x = M : A_2 \]
\[ \Gamma \vdash \lambda x: A_1.\bar{M} : \Pi x: A_1.A_2 \]
\[ \Gamma \vdash \lambda x: A_1.\bar{M} : A \]
\[ \Gamma \vdash M_1 = \lambda x: A_1.\bar{M} : \Pi x: A_1.A_2 \]
\[ \Gamma \vdash M_2 = \lambda x: A_1.\bar{M} : \Pi x: A_1.A_2 \]
\[ \Gamma \vdash M_1 = \lambda x: A_1.\bar{M} : A \]
\[ \Gamma \vdash M_2 = \lambda x: A_1.\bar{M} : A \]

By Inversion on Erasure
By Regularity
By Inversion on Typing
By Inversion on Typing
By rule (application typing)
By rule (application typing)

By induction
By induction
By rule
By rule (type conversion)
By rule (extensionality)
By rule (extensionality)
By rule (type conversion)
By rule (type conversion)

Case 4: \[ \Delta \vdash \pi_1 M_1 \iff \pi_1 M_2 : \tau_1 \]
\[ \Delta \vdash \pi_2 M_1 \iff \pi_2 M_2 : \tau_2 \]
\[ \Delta \vdash M_1 \iff M_2 : \tau_1 \times \tau_2 \]

\[ \Gamma \vdash A = \Sigma x: A_1.A_2 : \text{type}, \]
\[ A_1 \vdash \tau_1 \text{ and } A_2 \vdash \tau_2 \]
\[ \Gamma \vdash \Sigma x: A_1.A_2 : \text{type} \]
\[ \Gamma \vdash A_1 : \text{type} \]
\[ \Gamma, x: A_1 \vdash A_2 : \text{type} \]
\[ \Gamma \vdash M_1 : \Sigma x: A_1.A_2 \]
\[ \Gamma \vdash M_2 : \Sigma x: A_1.A_2 \]

By Inversion on Erasure
By Regularity
By Inversion on Typing
By Inversion on Typing
By assumption
By assumption

33
There exists a canonical object $\bar{M}_1$ such that
\[
\Gamma \vdash M_1 : A_1,
\Gamma \vdash \pi_1 M_1 = M_1 : A_1,
\Gamma \vdash \pi_1 M_1 = \bar{M}_1 : A_1,
\Gamma \vdash A_2 \pi_1 M_1 / x = A_2 \pi_1 M_1 / x : \text{type}
\]
By induction
By rule (projection)
By rule (projection)
By rule (projection)

There exists a canonical object $\bar{M}_2$ such that
\[
\Gamma \vdash M_2 : A_2 \pi_1 M_1 / x,
\Gamma \vdash \pi_2 M_1 = M_2 : A_2 \pi_1 M_1 / x,
\Gamma \vdash \pi_2 M_2 = M_2 : A_2 \pi_1 M_1 / x,
\Gamma \vdash \langle M_1, M_2 \rangle \Sigma \times A_1, A_2 : \Sigma \times A_1, A_2
\]
By induction
By rule
By rule (type conversion)
By rule (type conversion)

Case 5: \[\Delta \vdash M_1 \iff M_2 : 1\]
\[
\Gamma \vdash \Delta \vdash A = 1 : \text{type}
\Gamma \vdash \Delta \vdash M_1 : 1
\]
By Inversion on Erasure
By rule (type conversion)

Case 6: \[\Delta(x) = \tau\]
\[
\Delta \vdash x \iff x : \tau
\]
By assumption
By assumption
By Inversion
By rule (variable equality)
By rule (type conversion)
By rules (symmetry, transitivity)
By Erasure Preservation

Case 7: \[\Delta \vdash M_{11} \iff M_{21} : \tau_2 \rightarrow \tau_1 \quad \Delta \vdash M_{12} \iff M_{22} : \tau_2 \]
\[
\Delta \vdash M_{11} M_{12} \iff M_{21} M_{22} : \tau_1
\]
\[\Gamma \vdash M_{12} : A_1\]
\[\Gamma \vdash M_{22} : A_2\]
\[\Gamma \vdash M_{11} : \Pi x : A_{11}.A_{12}\]
\[\Gamma \vdash A_{12}[M_{12}/x] = A_1 : \text{type}\]
\[\Gamma \vdash M_{21} : \Pi x : A_{21}.A_{22}\]
\[\Gamma \vdash A_{22}[M_{22}/x] = A_2 : \text{type}\]

There is an atomic object \(\tilde{M}\) such that
\[\Gamma \vdash M : \Pi x : A_{11}.A_{12}\]
\[\Gamma \vdash M_{11} = M : \Pi x : A_{11}.A_{12}\]
\[\Gamma \vdash M_{21} = M : \Pi x : A_{11}.A_{12}\]
\[\Pi x : A_{11}.A_{12} = \Pi x : A_{21}.A_{22} : \text{type}\]
\[(\Pi x : A_{11}.A_{12})^- = (\Pi x : A_{21}.A_{22})^- = \tau_2 \rightarrow \tau_1\]
\[\Gamma \vdash A_{11} = A_{21} : \text{type}\]
\[\Gamma, x : A_{11} \vdash A_{12} = A_{22} : \text{type}\]
\[A_{11}^- = A_{21}^- = \tau_2\]
\[A_{12}^- = A_{22}^- = \tau_1\]
\[\Gamma \vdash M_{22} : A_{11}\]

There exists a canonical object \(\tilde{M}\) such that
\[\Gamma \vdash M : A_{11}\]
\[\Gamma \vdash M_{12} = M : A_{11}\]
\[\Gamma \vdash M_{22} = M : A_{11}\]

\[\Gamma \vdash A_{12}[\tilde{M}/x] = A_{12}[M_{12}/x] : \text{type}\]
\[\Gamma \vdash M \tilde{M} : A_{12}[M_{12}/x]\]
\[\Gamma \vdash M \tilde{M} = A_{12}[M_{12}/x]\]
\[\Gamma \vdash M_{11} M_{12} = M M : A_{12}[M_{12}/x]\]
\[\Gamma \vdash M_{11} M_{12} = M M : A_1\]
\[\Gamma \vdash M_{21} M_{22} = M M : A_{12}[M_{12}/x]\]
\[\Gamma \vdash M_{21} M_{22} = M M : A_1\]
\[\Gamma \vdash A_{12}[M_{12}/x] = A_{22}[M_{22}/x] : \text{type}\]
\[\Gamma \vdash A_1 = A_2 : \text{type}\]
\[A_1^- = A_{12}^- = A_{22}^- = A_2^- = \tau_1\]

By induction

By rule (application)

By rule (type conversion)

By rule (type conversion)

By rule (application)

By rule (application)

By rules (symmetry, transitivity)

By Erasure Preservation

Case 8:
\[\Delta \vdash M_1 \iff M_2 : \tau_1 \times \tau_2\]
\[
\Delta \vdash \pi_1 M_1 \iff \pi_1 M_2 : \tau_1
\]

\[\Gamma \vdash \pi_1 M_1 : A_1\]
\[\Gamma \vdash \pi_1 M_2 : A_2\]
\[\Gamma \vdash M_1 : \Sigma x : A_1.A_{11}\]
\[\Gamma \vdash M_2 : \Sigma x : A_2.A_{21}\]

There exists an atomic object \(\tilde{M}\) such that
\[\Gamma \vdash M : \Sigma x : A_3.A_{31}\]
\[\Gamma \vdash M_1 = M : \Sigma x : A_3.A_{31}\]
\[\Gamma \vdash M_2 = M : \Sigma x : A_3.A_{31}\]
\[\Gamma \vdash \Sigma x : A_1.A_{11} = \Sigma x : A_3.A_{31} : \text{type}\]
\[\Gamma \vdash \Sigma x : A_2.A_{21} = \Sigma x : A_3.A_{31} : \text{type}\]
\[\Sigma x : A_1.A_{11}^- = \Sigma x : A_3.A_{31}^- = \Sigma x : A_3.A_{31} = \tau_1 \times \tau_2\]
\[\Gamma \vdash \pi_1 M_1 : A_3\]
\[\Gamma \vdash \pi_1 M_1 = \pi_1 \tilde{M} : A_3\]

By Inversion

By induction

By rule (projection)
Γ ⊢ π₁M₂ = π₁M : A₃
Γ ⊢ A₁ = A₃ : type
Γ ⊢ π₁M₁ = π₁M : A₁
Γ ⊢ π₁M₂ = π₁M : A₁
Γ ⊢ A₂ = A₃ : type
Γ ⊢ A₁ = A₂ : type
A₁⁻ = A₂⁻ = A₃⁻ = τ₁

By rule (projection)
By Injectivity
By rule (type conversion)
By rule (type conversion)
By Injectivity
By rules (symmetry, transitivity)
By Erasure Preservation

Case 9:
\[ \Delta ⊢ M₁ \iff M₂ : τ₁ × τ₂ \]
\[ \Delta ⊢ π₂M₁ \iff π₂M₂ : τ₂ \]

Γ ⊢ π₂M₁ : A₁
Γ ⊢ π₂M₂ : A₂
Γ ⊢ M₁ : Σx:A₁, A₁₂,
Γ ⊢ A₁₂ [π₁M₁/x] = A₁ : type
Γ ⊢ M₂ : Σx:A₂₁, A₂₂,
Γ ⊢ A₂₂ [π₁M₂/x] = A₂ : type

There exists an atomic object M such that
Γ ⊢ M : Σx:A₁₁, A₁₂,
Γ ⊢ M₁ = M : Σx:A₁, A₁₂,
Γ ⊢ M₂ = M : Σx:A₂₁, A₂₂,
Γ ⊢ Σx:A₁₁, A₁₂ = Σx:A₁, A₁₂ : type,
Γ ⊢ Σx:A₂₁, A₂₂ = Σx:A₂₁, A₂₂ : type,

(Σx:A₁₁, A₁₂)⁻ = (Σx:A₂₁, A₂₂)⁻ = (Σx:A₁, A₁₂)⁻ = τ₁ × τ₂

By assumption
By assumption
By Inversion
By Inversion

By induction
By rule (projection)
By rule (projection)
By rule (projection)
By rule (projection)
By Injectivity
By Functionality
By rules (symmetry, transitivity)
By Erasure Preservation

Case 10:
\[ \Delta ⊢ A_{11} \iff A_{21} : type^- \quad \Delta, x:A_{11}^- \vdash A_{12} \iff A_{22} : type^- \]
\[ \Delta ⊢ Πx:A_{11}. A_{12} \iff Πx:A_{21}. A_{22} : type^- \]

Γ ⊢ Πx:A₁₁, A₁₂ : K₁
Γ ⊢ Πx:A₂₁, A₂₂ : K₂
Γ ⊢ K₁ = type : kind,
Γ ⊢ K₂ = type : kind
Γ ⊢ A₁₁ : type,
Γ ⊢ A₂₁ : type

By assumption
By assumption
By Inversion
By Inversion

There is a canonical family ̃A₁ such that
Γ ⊢ ̃A₁ : type,
Γ ⊢ ̃A₁₁ = ̃A : type,
Γ ⊢ ̃A₂₁ = ̃A : type

By induction
By inversion

36
There is a canonical family $\bar{A}_2$ such that
\begin{align*}
\Gamma, x: A_1 \vdash A_2 : \text{type} \\
\Gamma', x: A_1 \vdash A_2 : \text{type} \\
\end{align*}

By Context Conversion

There is a canonical family $\bar{A}_2$ such that
\begin{align*}
\Gamma, x: A_1 \vdash A_{12} = A_2 : \text{type} \\
\Gamma, x: A_1 \vdash A_{22} = A_2 : \text{type} \\
\Gamma \vdash \Pi x: \bar{A}_1.A_2 : \text{type} \\
\Gamma \vdash \Pi x: A_{11}.A_{12} = \Pi x: \bar{A}_1.A_2 : \text{type} \\
\Gamma \vdash \Pi x: A_{21}.A_{22} = \Pi x: \bar{A}_1.A_2 : \text{type} \\
K_1^- = K_2^- = \text{type}^+ \\
\end{align*}

By rule

By rule (Product Equality)

By rule (Product Equality)

By Erasure Preservation

\begin{align*}
\Delta \vdash A_{11} & \iff A_{21} : \text{type}^- \\
\Delta, x: A_{11} \vdash A_{12} & \iff A_{22} : \text{type}^- \\
\Delta \vdash \Sigma x: A_{11}.A_{12} & \iff \Sigma x: A_{21}.A_{22} : \text{type}^- \\
\end{align*}

By assumption

By assumption

By Inversion

By Inversion

By Inversion

By induction

By inversion

By Context Conversion

By Context Conversion

There is a canonical family $\bar{A}_1$ such that
\begin{align*}
\Gamma \vdash \bar{A}_1 : \text{type}, \\
\Gamma \vdash A_{11} = \bar{A} : \text{type}, \\
\Gamma \vdash A_{21} = \bar{A} : \text{type} \\
\Gamma, x: A_{11} \vdash A_{12} : \text{type} \\
\Gamma, x: A_{21} \vdash A_{22} : \text{type} \\
\Gamma, x: A_1 \vdash A_{22} : \text{type} \\
\Gamma, x: A_1 \vdash A_{22} : \text{type} \\
\end{align*}

By induction

By Context Conversion

By product equality

By product equality

By Erasure Preservation

$\square$

\textbf{Theorem 8.4 (Soundness)}

1. If $\Gamma \vdash M_1 : A$, $\Gamma \vdash M_2 : A$ and $\Gamma^- \vdash M_1 \iff M_2 : A^-$, then $\Gamma \vdash M_1 = M_2 : A$.

2. If $\Gamma \vdash M_1 : A_1$, $\Gamma \vdash M_2 : A_2$ and $\Gamma^- \vdash M_1 \iff M_2 : \tau$, then $\Gamma \vdash M_1 = M_2 : A_1$, $\Gamma \vdash A_1 = A_2 : \text{type and} A_1^- = A_2^- = \tau$.

3. If $\Gamma \vdash A_1 : \mathcal{K}$, $\Gamma \vdash A_2 : \mathcal{K}$ and $\Gamma^- \vdash A_1 \iff A_2 : \mathcal{K}^-$, then $\Gamma \vdash A_1 = A_2 : \mathcal{K}$.

4. If $\Gamma \vdash A_1 : \mathcal{K}_1$, $\Gamma \vdash A_2 : \mathcal{K}_2$ and $\Gamma^- \vdash A_1 \iff A_2 : \kappa$, then $\Gamma \vdash A_1 = A_2 : \mathcal{K}_1$, $\Gamma \vdash K_1 = K_2 : \text{kind and} K_1^- = K_2^- = \kappa$.

5. If $\Gamma \vdash K_1 : \text{kind}$, $\Gamma \vdash K_2 : \text{kind and} \Gamma^- \vdash K_1 \iff K_2 : \text{kind}^-$, then $\Gamma \vdash K_1 = K_2 : \text{kind}$.
Proof
Directly, from previous lemma. We will show the cases for objects, the cases for families and kinds are similar.

Case 1:
\[ \Gamma \vdash M_1 = \bar{M} : A, \]
\[ \Gamma \vdash M_2 = \bar{M} : A \]
\[ \Gamma \vdash M = M_2 : A \]
\[ \Gamma \vdash M_1 = M_2 : A \]
By Canonical Mediating Terms for Algorithm
By rule (symmetry)
By rule (transitivity)

Case 2:
\[ \Gamma \vdash M_1 = M : A_1, \]
\[ \Gamma \vdash M_2 = M : A_1, \]
\[ \Gamma \vdash A_1 = A_2 : \text{type}, \]
\[ A_1 = A_2 = \tau \]
By Canonical Mediating Terms for Algorithm
By rule (symmetry)
By rule (transitivity)

We can now also prove canonical forms by appeal to lemma 8.3.

Theorem 8.5 (Canonical Forms)
1. If \( \Gamma \vdash M : A \) then there exists a canonical object \( \bar{M} \) such that \( \Gamma \vdash \bar{M} : A \) and \( \Gamma \vdash M = \bar{M} : A \).
2. If \( \Gamma \vdash A : K \) then there exists a canonical family \( \bar{A} \) such that \( \Gamma \vdash \bar{A} : K \) and \( \Gamma \vdash A = \bar{A} : K \).
3. If \( \Gamma \vdash K : \text{kind} \) then there exists a canonical kind \( \bar{K} \) such that \( \Gamma \vdash \bar{K} : \text{kind} \) and \( \Gamma \vdash K = \bar{K} : \text{kind} \).

Proof
Directly, from Canonical Mediating Terms for Equality Algorithm. We will show the case for objects.

\[ \Gamma \vdash M = M : A \]
\[ \Gamma \vdash M \iff M : A \]
By Reflexivity
By Completeness of Algorithm

There exists a canonical object \( \bar{M} \) such that \( \Gamma \vdash \bar{M} : A \),
\[ \Gamma \vdash M = \bar{M} : A \]
By Canonical Mediating Terms for Equality Algorithm

\[ \square \]

9 Decidability of Type Checking
Terms which are related by the algorithm for equality are called by Harper and Pfenning “normalizing”. This terminology is justified by our result on canonical forms, since these terms are provably equal to canonical forms. We first prove that equality between normalizing terms is decidable. This will imply decidability of equality for all well-typed terms, by the use of completeness of the algorithm for deciding equality.

Lemma 9.1 (Decidability of Equality for Normalizing Terms)
1. If $\Delta \vdash M_1 \iff M_2 : \tau$ and $\Delta \vdash M_3 \iff M_4 : \tau$ then it is decidable whether $\Delta \vdash M_1 \iff M_3 : \tau$.

2. If $\Delta \vdash M_1 \iff M_2 : \tau_1$ and $\Delta \vdash M_3 \iff M_4 : \tau_2$ then it is decidable whether $\Delta \vdash M_1 \iff M_3 : \tau_3$ for any $\tau_3$.

3. If $\Delta \vdash A_1 \iff A_2 : \kappa$ and $\Delta \vdash A_3 \iff A_4 : \kappa$ then it is decidable whether $\Delta \vdash A_1 \iff A_3 : \kappa$.

4. If $\Delta \vdash A_1 \iff A_2 : \kappa_1$ and $\Delta \vdash A_3 \iff A_4 : \kappa_2$ then it is decidable whether $\Delta \vdash A_1 \iff A_3 : \kappa_3$ for any $\kappa_3$.

5. If $\Delta \vdash K_1 \iff K_2 : \text{kind}^-$ and $\Delta \vdash K_3 \iff K_4 : \text{kind}^-$ then it is decidable whether $\Delta \vdash K_1 \iff K_3 : \text{kind}^-$.

Proof

By structural induction on the derivation of the judgment.

\[ \square \]

Theorem 9.2 (Decidability of Algorithmic Equality)

1. If $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$, then it is decidable whether $\Gamma^- \vdash M_1 \iff M_2 : A^-$. 

2. If $\Gamma \vdash A_1 : K$ and $\Gamma \vdash A_2 : K$, then it is decidable whether $\Gamma^- \vdash A_1 \iff A_2 : K^-$. 

3. If $\Gamma \vdash K_1 : \text{kind}$ and $\Gamma \vdash K_2 : \text{kind}$, then it is decidable whether $\Gamma^- \vdash K_1 \iff K_2 : \text{kind}^-$. 

Proof

Direct from decidability for normalizing terms, using reflexivity of equality. We show one case, others are analogous.

\[ \Gamma \vdash M_1 = M_1 : A \quad \text{By Reflexivity of Equality} \]
\[ \Gamma^- \vdash M_1 \iff M_1 : A^- \quad \text{By Completeness of Algorithmic Equality} \]
\[ \Gamma \vdash M_2 = M_2 : A \quad \text{By Reflexivity of Equality} \]
\[ \Gamma^- \vdash M_2 \iff M_2 : A^- \quad \text{By Completeness of Algorithmic Equality} \]

It is decidable whether $\Gamma^- \vdash M_1 \iff M_2 : A^-$.

\[ \text{By Decidability of Algorithmic Equality for Normalizing Terms} \]

\[ \square \]

Corollary 9.3 (Decidability of Definitional Equality)

1. If $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$, then it is decidable whether $\Gamma^- \vdash M_1 = M_2 : A^-$. 

2. If $\Gamma \vdash A_1 : K$ and $\Gamma \vdash A_2 : K$, then it is decidable whether $\Gamma^- \vdash A_1 = A_2 : K^-$. 

3. If $\Gamma \vdash K_1 : \text{kind}$ and $\Gamma \vdash K_2 : \text{kind}$, then it is decidable whether $\Gamma^- \vdash K_1 = K_2 : \text{kind}$. 

Proof

By soundness and completeness of algorithmic equality, it is enough to have decidability of algorithmic equality.

\[ \square \]

9.1 Algorithm for Type Checking

We now use the algorithm for equality as a subroutine to provide an algorithm for type checking.
Objects

\[ \Gamma(x) = A \quad \Sigma(c) = A \]
\[ \Gamma \vdash x : A \quad \Gamma \vdash c : A \]
\[ \Gamma \vdash M_1 \\Rightarrow A \quad A \xmapsto{\text{whr}^*} \Pi x : A_2, A_1 \quad \Gamma \vdash M_2 \\Rightarrow A_{22} \quad \Gamma \vdash A_{21} \iff A_{22} : \text{type}^\neg \]
\[ \Gamma \vdash M_1 M_2 \Rightarrow A_1 \quad [M_2/x] \]
\[ \Gamma \vdash A_1 \Rightarrow \text{type} \quad \Gamma, x : A_1 \vdash M_2 \Rightarrow A_2 \]
\[ \Gamma \vdash \lambda x : A_1, M_2 \Rightarrow \Pi x : A_1, A_2 \]
\[ \Gamma \vdash \Sigma x : A_1, A_2 \Rightarrow \text{type} \quad \Gamma \vdash M_1 \Rightarrow A_{11} \quad \Gamma \vdash M_2 \Rightarrow A_{21} \]
\[ \Gamma \vdash A \iff A_{22} [M_1/x] : \text{type}^\neg \]
\[ \Gamma \vdash (M_1, M_2) \Sigma x : A_1, A_2 \Rightarrow \Sigma x : A_1, A_2 \]
\[ \Gamma \vdash M \Rightarrow A \quad A \xmapsto{\text{whr}^*} \Sigma x : A_1, A_2 \]
\[ \Gamma \vdash \pi_1 M \Rightarrow A_1 \quad \Gamma \vdash \pi_2 M \Rightarrow A_2 [\pi_1 M/x] \]
\[ \Gamma \vdash \emptyset \Rightarrow 1 \]

Families

\[ \Sigma(a) = K \quad \Gamma \vdash A \Rightarrow \text{type} \quad \Gamma, x : A_1 \vdash A_2 \Rightarrow K \]
\[ \Gamma \vdash a \Rightarrow K \quad \Gamma \vdash \lambda x : A_1, A_2 \Rightarrow \Pi x : A_1, K \]
\[ \Gamma \vdash A_1 \Rightarrow \Pi x : A_2, K_1 \quad \Gamma \vdash M \Rightarrow A_{22} \quad \Gamma \vdash A_{21} \iff A_{22} : K_1^\neg \]
\[ \Gamma \vdash A_1 \Rightarrow K_1 [M/x] \]
\[ \Gamma \vdash A_1 \Rightarrow \text{type} \quad \Gamma, x : A_1 \vdash A_2 \Rightarrow \text{type} \quad \Gamma \vdash \Pi x : A_1, A_2 \Rightarrow \text{type} \]
\[ \Gamma \vdash \Sigma x : A_1, A_2 \Rightarrow \text{type} \]
\[ \Gamma \vdash \Pi x : A, K \Rightarrow \text{kind} \]
\[ \Gamma \vdash A \Rightarrow \text{type} \quad \Gamma, x : A_1 \Rightarrow K \Rightarrow \text{kind} \]

Kinds

\[ \Gamma \vdash \text{type} \Rightarrow \text{kind} \quad \Gamma \vdash A \Rightarrow \text{type} \quad \Gamma, x : A_1 \Rightarrow K \Rightarrow \text{kind} \]

It is easy to show the soundness and completeness of the algorithm for typechecking.

Lemma 9.4 (Soundness of Algorithmic Typechecking)

1. If \( \Gamma \vdash M \Rightarrow A \) then \( \Gamma \vdash M : A \).
2. If \( \Gamma \vdash A \Rightarrow K \) then \( \Gamma \vdash A : K \).
3. If \( \Gamma \vdash K \Rightarrow \text{kind} \) then \( \Gamma \vdash K : \text{kind} \).
Proof
By induction on the derivation.

To show completeness under the presence of a non-trivial equality at the type level (induced by the presence of type-level abstractions), we need the following technical lemma.

Lemma 9.5 (Inversion on Product and Sum Families)

1. If $\Gamma \vdash A = \Pi x:A_1.A_2 : type$ then $A \xrightarrow{\text{whr}} \Pi x:A_{11}.A_{21}$ such that $\Gamma \vdash A_1 = A_{11} : type$ and $\Gamma, x:A_1 \vdash A_2 = A_{21} : type$.

2. If $\Gamma \vdash A = \Sigma x:A_1.A_2 : type$ then $A \xrightarrow{\text{whr}} \Sigma x:A_{11}.A_{21}$ such that $\Gamma \vdash A_1 = A_{11} : type$ and $\Gamma, x:A_1 \vdash A_2 = A_{21} : type$.

Proof
By Erasure Preservation under Equality together with Inversion on Erasure, and using Injectivity.

Lemma 9.6 (Completeness of Algorithmic Typechecking)

1. If $\Gamma \vdash M : A$ then $\Gamma \vdash M \Rightarrow A'$ and $\Gamma \vdash A = A'$ : type.

2. If $\Gamma \vdash A : K$ then $\Gamma \vdash A \Rightarrow K'$ and $\Gamma \vdash K = K'$ : kind.

3. If $\Gamma \vdash K : \text{kind}$ then $\Gamma \vdash K \Rightarrow \text{kind}$.

Proof
By induction on the derivation. We show two significant cases.

Case 1:

\[
\frac{
\Gamma \vdash M_1 : \Pi x:A_2.A_1 \quad \Gamma \vdash M_2 : A_2
}{
\Gamma \vdash M_1 M_2 : A_1 [M_2/x]
}
\]

$\Gamma \vdash M_1 \Rightarrow A_{11}$,
$\Gamma \vdash A_{11} = \Pi x:A_2.A_1 : type$

$A_{11} \xrightarrow{\text{whr}} \Pi x:A_{11}' A_{21}'$ $\Gamma \vdash A_2 = A_{21}' : type,$

$\Gamma, x:A_2 \vdash A_1 = A_{11}' : type$
$\Gamma \vdash M_2 \Rightarrow A_{21},$

$\Gamma \vdash A_{21} = A_{21} : type$ By induction

$\Gamma \vdash A_{21} \iff A_{21}' : type$ By rule (transitivity)

$\Gamma \vdash M_1 M_2 \Rightarrow A_{11}' [M_2/x]$ By Completeness of Algorithmic Equality

$\Gamma \vdash A_{11}' [M_2/x] = A_1 [M_2/x] : type$ By rule

Case 2:

\[
\frac{
\Gamma \vdash M : \Sigma x:A_1.A_2
}{
\Gamma \vdash \pi_1 M : A_1
}
\]

$\Gamma \vdash M \Rightarrow A_1$

$\Gamma \vdash A = \Sigma x:A_1.A_2 : type$ By induction

$A \xrightarrow{\text{whr}} \Sigma x:A_{11}' A_{21}'$, By Inversion on Sum Families

$\Gamma \vdash A_1 = A_{11}' : type$,

$\Gamma, x:A_1 \vdash A_2 = A_{21}' : type$ By rule

$\Gamma \vdash \pi_1 M \Rightarrow A_{11}'$
10 Conclusion

We have presented a new type theory in this paper. This is the logical framework LF extended with dependent pair and unit types. To be used as a logical framework, the important property of canonical forms is essential. We have shown the existence of the canonical forms property. The other useful property of decidability of type checking and equivalence is also shown. The algorithm is an extension of previous algorithms for the restriction to LF. A careful proof of soundness and completeness has also been shown.

In future work, we intend to use this language as our representation language for a functional programming language that can express soundness of algorithms. The current results will be important in showing that such a language can be verified statically.

References


