# On Equivalence and Canonical Forms in the LF Type Theory (Extended Abstract)*

Robert Harper and Frank Pfenning
Department of Computer Science
Carnegie Mellon University

August 30, 1999

**Abstract**

Decidability of definitional equality and conversion of terms into canonical form play a central role in the meta-theory of a type-theoretic logical framework. Most studies of definitional equality are based on a confluent, strongly-normalizing notion of reduction. Coquand has considered a different approach, directly proving the correctness of a practical equivalence algorithm based on the shape of terms. Neither approach appears to scale well to richer languages with unit types or subtyping, and neither directly addresses the problem of conversion to canonical form.

In this paper we present a new, type-directed equivalence algorithm for the LF type theory that overcomes the weaknesses of previous approaches. The algorithm is practical, scales to richer languages, and yields a new notion of canonical form sufficient for adequate encodings of logical systems. The algorithm is proved complete by a Kripke-style logical relations argument similar to that suggested by Coquand. Crucially, both the algorithm itself and the logical relations rely only on the shapes of types, ignoring dependencies on terms.

## 1 Introduction

At present the mechanization of constructive reasoning relies almost entirely on type theories of various forms. The principal reason is that the computational meaning of constructive proofs is an integral part of the type theory itself. The main computational mechanism in such type theories is reduction, which has therefore been studied extensively.

For logical frameworks the case for type theoretic meta-languages is also compelling, since they allow us to internalize deductions as objects. The validity of a deduction is then verified by type-checking in the meta-language. To ensure that proof checking remains decidable under this representation, the type checking problem for the meta-language must also be decidable. To support deductive systems of practical interest, the type theory must support *dependent types*, that is, types that depend on objects.

The correctness of the representation of a logic in type theory is given by an *adequacy theorem* that correlates the syntax and deductions of the logic with *canonical forms* of suitable type. To establish a precise correspondence, canonical forms are taken to be $\beta$-normal, $\eta$-long forms. In

*This work was sponsored NSF Grant CCR-9619584.
To appear at the Workshop on Logical Frameworks and Meta-Languages, Paris, France, September 1999

particular, it is important that canonical forms enjoy the property that constants and variables of higher type are "fully applied" — that is, each occurrence is applied to enough arguments to reach a base type.

Thus we see that the methodology of logical frameworks relies on two fundamental meta-theoretic results: the decidability of type checking, and the existence of canonical forms. For many type theories the decidability of type checking is easily seen to reduce to the decidability of definitional equality of types and terms. Therefore we focus attention on the decision problem for definitional equality and on the conversion of terms to canonical form.

Traditionally, both problems have been treated by considering normal forms for $\beta$, and possibly $\eta$, reduction. If we take definitional equality to be conversion, then its decidability follows from confluence and strong normalization for the corresponding notion of reduction. In the case of $\beta$-reduction this approach to deciding definitional equality works well, but for $\beta\eta$-reduction the situation is much more complex. In particular, $\beta\eta$-reduction is confluent only for well-typed terms, and subject reduction depends on strengthening, which is difficult to prove directly.

These technical problems with $\beta\eta$-reduction have been addressed in work by Salvesen [Sal90], Geuvers [Geu92] and later with a different method by Goguen [Gog99], but nevertheless several problems remain. First, canonical forms are not $\beta\eta$-normal forms and so conversion to canonical form must be handled separately. Second, the algorithms implicit in the reduction-based accounts are not practical; if two terms are *not* definitionally equal, we can hope to discover this without reducing both to normal form. Third, the approach does not appear to scale to richer theories such as those including unit types or subtyping.

These problems were side-stepped in the original paper on the LF logical framework [HHP93] by restricting attention to $\beta$-conversion for definitional equality, which is sufficient if we also restrict attention to $\eta$-long forms [FM90, Cer96]. This restriction is somewhat unsatisfactory, especially in linear variants of LF [CP98].

More recently, $\eta$-expansion has been studied in its own right, using modification of standard techniques from rewriting theory to overcome the lack of strong normalization when expansion is not restricted [JG95, Gha97]. In the dependently typed case, even the definition of long normal form is not obvious [DHW93] and the technical development is fraught with difficulties. We have not been able to reconstruct the proofs in [Gha97] and the development in [Vir99] relies on a complex intermediate system with annotated terms.

To address the problems of practicality, Coquand suggested abandoning reduction-based treatments of definitional equality in favor of a direct presentation of a practical equivalence algorithm [Coq91]. Coquand's approach is based on analyzing the "shapes" of terms, building in the principle of extensionality instead of relying on $\eta$-reduction or expansion. This algorithm improves on reduction-based approaches by avoiding explicit computation of normal forms, and allowing for early termination in the case that two terms are determined to be inequivalent. However, Coquand's approach does not address the problem of computing canonical forms, nor can it be easily extended to richer type theories such as those with unit types. In both cases the problem can be traced to the reliance on the shape of terms, rather than on their classifying types, to guide the algorithm. For example, if $x$ and $y$ are two variables of unit type, they are definitionally equal, but are structurally distinct; moreover, their canonical forms would be the sole element of unit type.

In this paper we present a new type-directed algorithm for testing equality for a dependent type theory in the presence of $\beta$ and $\eta$-conversion, that generalizes the algorithm for the simply-typed case given by Pfenning [Pfe92]. We prove its correctness directly via logical relations. The essential idea is that we can erase dependencies when defining the logical relation, even though the domain of the relation contains dependently typed terms. This makes the definition obviously well-founded.

Moreover, it means that the type-directed equality-testing algorithm on dependently typed terms requires only simple types. Consequently, transitivity of the algorithm is easy to show, a crucial property that we were unable to obtain without this simplifying step. Soundness and completeness of the equality-testing algorithm yields the decidability of the type theory rather directly.

Another advantage of our approach is that it can be easily adapted to support adequacy proofs using a new notion of *quasi-canonical forms*, which are canonical forms without type labels on $\lambda$-abstractions. We show that quasi-canonical forms of a given type are sufficient to determine the meaning of an object, since the type labels can be reconstructed (up to definitional equality) from the classifying type. Interestingly, recent research on dependently typed rewriting [Vir99] has also isolated equivalence classes of terms modulo conversion of the type labels as a critical concept.

While it is beyond the scope of this paper, we believe our construction is robust with respect to extension of the type theory with products, unit, linearity, subtyping and similar complicating factors. The reason is the flexibility of type-directed equality in the simply-typed case and the harmony between the definition of the logical relation and the algorithm, both of which are based on the erased types.

Throughout this extended abstract, we omit formal statements of lemmas and their proofs when they follow standard techniques. For the sake of brevity we concentrate in the lemmas and theorems on objects and may omit analogous statements for the levels of families and kinds. Full details can be found in a forthcoming technical report [HP99].

## 2 A Variant of the LF Type Theory

Syntactically, our formulation of the LF type theory follows the original proposal by Harper, Honsell and Plotkin [HHP93], except that we omit type-level $\lambda$-abstraction. This simplifies the proof of the soundness theorem considerably, since we can prove the injectivity of products (Lemma 6) at an early stage. In practice, this restriction has no impact since types in normal form never contain type-level $\lambda$-abstractions. In compensation for the omission of type-level $\lambda$-abstractions, we augment the type theory with a *functionality* rule stating that families respect equality in their free variables. The functionality rule for families is derivable in the presence of family-level abstractions, but appears to be essential in their absence.

$$
\begin{array}{rrcl}
\text{Kinds} & K & ::= & \text{type} \mid \Pi x{:}A.\ K \\
\text{Families} & A & ::= & a \mid A\ M \mid \Pi x{:}A_1.\ A_2 \\
\text{Objects} & M & ::= & c \mid x \mid \lambda x{:}A.\ M \mid M_1\ M_2 \\
\text{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}A
\end{array}
$$

We use $K$ for kinds, $A, B, C$ for type families, $M, N, O$ for objects, $\Gamma, \Psi$ for contexts and $\Sigma$ for signatures. We also use the symbol "kind" to classify the valid kinds. We consider terms that differ only in the names of their bound variables as identical. We write $[N/x]M$, $[N/x]A$ and $[N/x]K$ for capture-avoiding substitution. Signatures and contexts may declare each constant and variable at most once.

Our formulation of the typing rules is similar to the second version given in [HHP93]. In preparation for the various algorithms we presuppose and inductively preserve the validity of contexts involved in the judgments, instead of checking these properties at the leaves. This is a matter of expediency rather than necessity.

**Signatures**

$$\frac{}{\vdash \cdot \text{ sig}} \qquad \frac{\vdash \Sigma \text{ sig} \qquad \cdot \vdash_\Sigma K : \text{kind}}{\vdash \Sigma, a{:}K \text{ sig}} \qquad \frac{\vdash \Sigma \text{ sig} \qquad \cdot \vdash_\Sigma A : \text{type}}{\vdash \Sigma, c{:}A \text{ sig}}$$

From now on we fix a valid signature $\Sigma$ and omit it from the judgments, since it does not change in derivations.

**Contexts**

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

From now on we presuppose that all contexts in judgments are valid, instead of checking their validity explicitly. This means, for example, that we have to verify the validity of the type labels in $\lambda$-abstractions before adding them to the context.

**Objects**

$$\frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x : A} \qquad \frac{c{:}A \text{ in } \Sigma}{\Gamma \vdash c : A}$$

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

$$\frac{\Gamma \vdash A_1 : \text{type} \qquad \Gamma, x{:}A_1 \vdash M_2 : A_2}{\Gamma \vdash \lambda x{:}A_1.\ M_2 : \Pi x{:}A_1.\ A_2}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A = B : \text{type}}{\Gamma \vdash M : B}$$

**Families**

$$\frac{a{:}K \text{ in } \Sigma}{\Gamma \vdash a : K} \qquad \frac{\Gamma \vdash A : \Pi x{:}B.\ K \qquad \Gamma \vdash M : B}{\Gamma \vdash A\ M : [M/x]K}$$

$$\frac{\Gamma \vdash A_1 : \text{type} \qquad \Gamma, x{:}A_1 \vdash A_2 : \text{type}}{\Gamma \vdash \Pi x{:}A_1.\ A_2 : \text{type}}$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash K = L : \text{kind}}{\Gamma \vdash A : L}$$

**Kinds**

$$\frac{}{\Gamma \vdash \text{type} : \text{kind}} \qquad \frac{\Gamma \vdash A : \text{type} \qquad \Gamma, x{:}A \vdash K : \text{kind}}{\Gamma \vdash \Pi x{:}A.\ K : \text{kind}}$$

The rules for definitional equality are also written with the presupposition that a valid signature $\Sigma$ is fixed and that all contexts $\Gamma$ are valid. The intent is that equality implies validity of the objects, families, or kinds involved, a property formalized in Lemma 2. In contrast to the original

formulation in [HHP93], equality is based on a notion of parallel conversion plus extensionality, rather then $\beta\eta$-conversion. We believe this is a robust foundation, easily transferred to richer and more complicated type theories.

Characteristically for parallel conversion, reflexivity is admissible. This significantly simplifies the completeness proof for the equality checking algorithm. Similarly, the somewhat unwieldy congruence rule for $\lambda$-abstraction can be derived from the general extensionality principle. We enclose the admissible rules in [brackets]. Also, for technical reasons some rules have typing premises that we can later prove to be redundant. Such redundant premises are enclosed in {braces}.

**Simultaneous Congruence**

$$\frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x = x : A} \qquad \frac{c{:}A \text{ in } \Sigma}{\Gamma \vdash c = c : A}$$

$$\frac{\Gamma \vdash M_1 = N_1 : \Pi x{:}A_2.\ A_1 \qquad \Gamma \vdash M_2 = N_2 : A_2}{\Gamma \vdash M_1\, M_2 = N_1\, N_2 : [M_2/x]A_1}$$

$$\left[ \frac{\Gamma \vdash A_1' = A_1 : \text{type} \qquad \Gamma \vdash A_1'' = A_1 : \text{type} \qquad \Gamma, x{:}A_1 \vdash M_2 = N_2 : A_2}{\Gamma \vdash \lambda x{:}A_1'.\ M_2 = \lambda x{:}A_1''.\ N_2 : \Pi x{:}A_1.\ A_2} \right]$$

**Extensionality**

$$\frac{\Gamma \vdash A_1 : \text{type} \qquad \{\Gamma \vdash M : \Pi x{:}A_1.\ A_2\} \qquad \{\Gamma \vdash N : \Pi x{:}A_1.\ A_2\} \qquad \Gamma, x{:}A_1 \vdash M\, x = N\, x : A_2}{\Gamma \vdash M = N : \Pi x{:}A_1.\ A_2}$$

**Parallel Conversion**

$$\frac{\{\Gamma \vdash A_1 : \text{type}\} \qquad \Gamma, x{:}A_1 \vdash M_2 = N_2 : A_2 \qquad \Gamma \vdash M_1 = N_1 : A_1}{\Gamma \vdash (\lambda x{:}A_1.\ M_2)\, M_1 = [N_1/x]N_2 : [M_1/x]A_2}$$

**Equivalence**

$$\frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A} \qquad \frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash N = O : A}{\Gamma \vdash M = O : A}$$

$$\left[ \frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A} \right]$$

**Type Conversion**

$$\frac{\Gamma \vdash M = N : A \qquad \Gamma \vdash A = B : \text{type}}{\Gamma \vdash M = N : B}$$

**Family Congruence**

$$\frac{a{:}K \text{ in } \Sigma}{\Gamma \vdash a = a : K}$$

$$\frac{\Gamma \vdash A = B : \Pi x{:}C.\ K \qquad \Gamma \vdash M = N : C}{\Gamma \vdash A\,M = B\,N : [M/x]K}$$

$$\frac{\Gamma \vdash A_1 = B_1 : \text{type} \qquad \{\Gamma \vdash A_1 : \text{type}\} \qquad \Gamma, x{:}A_1 \vdash A_2 = B_2 : \text{type}}{\Gamma \vdash \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : \text{type}}$$

**Family Equivalence**

$$\frac{\Gamma \vdash A = B : K}{\Gamma \vdash B = A : K} \qquad \frac{\Gamma \vdash A = B : K \qquad \Gamma \vdash B = C : K}{\Gamma \vdash A = C : K}$$

$$\left[\ \frac{\Gamma \vdash A : K}{\Gamma \vdash A = A : K}\ \right]$$

**Family Functionality**

$$\frac{\Gamma \vdash M = N : A \qquad \vdash \Gamma, x{:}A, \Gamma' \text{ ctx} \qquad \Gamma, x{:}A, \Gamma' \vdash B = C : K}{\Gamma, [M/x]\Gamma' \vdash [M/x]B = [N/x]C : [M/x]K}$$

**Kind Conversion**

$$\frac{\Gamma \vdash A = B : K \qquad \Gamma \vdash K = L : \text{kind}}{\Gamma \vdash A = B : L}$$

**Kind Congruence**

$$\frac{}{\Gamma \vdash \text{type} = \text{type} : \text{kind}}$$

$$\frac{\Gamma \vdash A = B : \text{type} \qquad \{\Gamma \vdash A : \text{type}\} \qquad \Gamma, x{:}A \vdash K = L : \text{kind}}{\Gamma \vdash \Pi x{:}A.\ K = \Pi x{:}B.\ L : \text{kind}}$$

**Kind Equivalence**

$$\frac{\Gamma \vdash K = L : \text{kind}}{\Gamma \vdash L = K : \text{kind}} \qquad \frac{\Gamma \vdash K = L : \text{kind} \qquad \Gamma \vdash L = L' : \text{kind}}{\Gamma \vdash K = L' : \text{kind}}$$

$$\left[\ \frac{\Gamma \vdash K : \text{kind}}{\Gamma \vdash K = K : \text{kind}}\ \right]$$

In the logical relations argument we require a notion of simultaneous substitution of terms for free variables. Substitutions are defined as follows:

$$\text{Substitutions} \quad \sigma \quad ::= \quad \cdot \mid \sigma, M/x$$

We assume that no variable is defined more than once in any substitution, which can be achieved by appropriate renaming where necessary.

We write $\mathrm{id}_\Gamma$ for the identity substitution on the context $\Gamma$. We use the notation $M[\sigma]$, $A[\sigma]$ and $K[\sigma]$ for the *simultaneous* substitution by $\sigma$ into an object, family, or kind defined in the usual way.

If $J$ is a typing or equality judgement, then we write $J[\sigma]$ for the obvious substitution of $J$ by $\sigma$. For example, if $J$ is $M : A$, then $J[\sigma]$ stands for the judgment $M[\sigma] : A[\sigma]$.

**Lemma 1 (Basic Properties)**

1. *(Weakening) If* $\Gamma, \Gamma' \vdash J$ *then* $\Gamma, x{:}A, \Gamma' \vdash J$.
2. *(Reflexivity) If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash M = M : A$ *and similarly at the level of families and kinds.*
3. *(Substitution) If* $\Gamma, x{:}A, \Gamma'$ *is valid,* $\Gamma \vdash M : A$, *and* $\Gamma, x{:}A, \Gamma' \vdash J$ *then* $\Gamma, [M/x]\Gamma' \vdash [M/x]J$.

**Proof:** All three properties follow by straightforward inductions on the structure of the given derivations, except that substitution at the level of families is handled directly by the functionality rule. □

The property of validity is needed for the soundness proof of the algorithm (although not for the completeness argument). It shows that given a valid context, the components of a derivable judgment are also valid in an appropriate sense.

**Lemma 2 (Validity)** *Assume* $\Gamma$ *is a valid context.*

1. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A$ : type.
2. *If* $\Gamma \vdash M = N : A$, *then* $\Gamma \vdash M : A$, $\Gamma \vdash N : A$, *and* $\Gamma \vdash A$ : type.

*Analogous properties hold at the level of families and kinds.*

**Proof:** By a straightforward simultaneous induction on the given derivations. The functionality rule is required to handle the case of applications. The typing premises on the rule of extensionality ensure that strengthening is not required at this point. □

Now we can easily prove that the typing premises enclosed in {braces} in the equality rules are redundant, with the exception of the extensionality rule, which requires strengthening to prove their eliminability. This will only available after soundness and completeness of our algorithm for testing equality has been established.

# 3 Algorithmic Equality

The algorithm for deciding equality can be summarized as follows:

1. When comparing objects at function type, apply extensionality.
2. When comparing objects at base type, reduce both sides to weak head-normal form and then compare heads directly and, if they are equal, each corresponding pair of arguments according to their type.

Since this algorithm is type-directed in case (1) we need to carry types. Unfortunately, this makes it difficult to prove correctness of the algorithm in the presence of dependent types, because transitivity is not an obvious property. The informal description above already contains a clue to the solution: we do not need to know the precise type of the objects we are comparing, as long as we know that they are functions.

We therefore define a calculus of simple types and an erasure function $(-)^-$ that eliminates dependencies for the purpose of this algorithm. The same idea is used later in the definition of the logical relations used to prove completeness of the algorithm.

We write $\alpha$ to stands for simple base types and we have two special type constants, $\text{type}^-$ and $\text{kind}^-$, for the equality judgments at the level of types and kinds.

$$\begin{array}{rrcl} \text{Simple Kinds} & \kappa & ::= & \text{type}^- \mid \tau \to \kappa \\ \text{Simple Types} & \tau & ::= & \alpha \mid \tau_1 \to \tau_2 \\ \text{Simple Contexts} & \Delta & ::= & \cdot \mid \Delta, x{:}\tau \end{array}$$

We use $\tau, \theta, \delta$ for simple types and $\Delta, \Theta$ for contexts declaring simple types for variables. We also use $\text{kind}^-$ in a similar role to kind in the LF type theory.

We write $A^-$ for the simple type that results from erasing dependencies in $A$, and similarly $K^-$. We translate each constant type family $a$ to a base type $a^-$ and extend this to all type families. We extend it further to contexts by applying it to each declaration. It is immediate that the erasure of equal types are equal and the erasure does not change under object substitution.

We now present the algorithm in the form of three judgments.

$M \xrightarrow{\text{whr}} M'$ *(M weak head reduces to M′)* Algorithmically, we assume $M$ is given and compute $M'$ (if $M$ is head reducible) or fail.

$\Delta \vdash M \Longleftrightarrow N : \tau$ *(M is equal to N at simple type τ)* Algorithmically, we assume $\Delta$, $M$, $N$, and $\tau$ are given and we simply succeed or fail. We only apply this judgment if $M$ and $N$ have the same type $A$ and $\tau = A^-$.

$\Delta \vdash M \longleftrightarrow N : \tau$ *(M is structurally equal to N)* Algorithmically, we assume that $\Delta$, $M$ and $N$ are given and we compute $\tau$ or fail. If successful, $\tau$ will be the approximate type of $M$ and $N$.

Note that the structural and type-directed equality are mutually recursive, while weak head reduction does not depend on the other two judgments.

**Weak Head Reduction**

$$\frac{}{(\lambda x{:}A_1.\ M_2)\, M_1 \xrightarrow{\text{whr}} [M_1/x]M_2} \qquad \frac{M_1 \xrightarrow{\text{whr}} M_1'}{M_1\, M_2 \xrightarrow{\text{whr}} M_1'\, M_2}$$

**Type-Directed Object Equality**

$$\frac{M \xrightarrow{\text{whr}} M' \qquad \Delta \vdash M' \Longleftrightarrow N : \alpha}{\Delta \vdash M \Longleftrightarrow N : \alpha} \qquad \frac{N \xrightarrow{\text{whr}} N' \qquad \Delta \vdash M \Longleftrightarrow N' : \alpha}{\Delta \vdash M \Longleftrightarrow N : \alpha}$$

$$\frac{M \longleftrightarrow N : \alpha}{M \Longleftrightarrow N : \alpha} \qquad \frac{\Delta, x{:}\tau_1 \vdash M\, x \Longleftrightarrow N\, x : \tau_2}{\Delta \vdash M \Longleftrightarrow N : \tau_1 \to \tau_2}$$

**Structural Object Equality**

$$\frac{x{:}\tau \text{ in } \Delta}{\Delta \vdash x \longleftrightarrow x : \tau} \qquad \frac{c{:}A \text{ in } \Sigma}{\Delta \vdash c \longleftrightarrow c : A^-}$$

$$\frac{\Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \to \tau_1 \qquad \Delta \vdash M_2 \Longleftrightarrow N_2 : \tau_2}{\Delta \vdash M_1\, M_2 \longleftrightarrow N_1\, N_2 : \tau_1}$$

Analogous structural rules for families and kinds are omitted here.

The algorithmic equality satisfies some simple structural properties. Weakening is required in the proof of its correctness. It does not appear that exchange, contraction, or strengthening are needed in our particular argument, but these properties can all be easily proved. Note that versions of the logical relations proofs nonetheless apply in the linear, strict, and affine $\lambda$-calculi.

The algorithm is essentially deterministic in the sense that when comparing terms at base type we have to weakly head-normalize both sides and compare the results structurally. This is because terms that are weakly head reducible will never be considered structurally equal. Again, we elide a more formal statement of this property.

The completeness proof requires symmetry and transitivity of the algorithm, which follow by simple structural inductions. This would introduce some difficulty if the algorithm employed precise instead of approximate types. This is one reason why both the algorithm and later the logical relation are defined using approximate types only.

# 4 Completeness of Algorithmic Equality

In this section we develop the completeness theorem for the type-directed equality algorithm. That is, if two terms are definitionally equal, the algorithm will succeed. The goal is to present a flexible and modular technique that can be adapted easily to related type theories, such as the one underlying the linear logical framework [CP98], one based on non-commutative linear logic [PP99], or one including subtyping [Pfe93]. Other techniques presented in the literature, particularly those based on a notion of $\eta$-reduction, do not seem to adapt well to these richer theories.

The central idea is to proceed by an argument via logical relations defined inductively on the approximate type of an object, where the approximate type arises from erasing all dependencies in an LF type.

The completeness direction of the correctness proof for type-directed equality states:

$$\text{If } \Gamma \vdash M = N : A \text{ then } \Gamma^- \vdash M \Longleftrightarrow N : A^-.$$

One would like to prove this by induction on the structure of the derivation for the given equality. However, such a proof attempt fails at the case for application. Instead we define a logical relation $\Delta \vdash M = N \in [\![\tau]\!]$ that provides a stronger induction hypothesis so that both

1. if $\Gamma \vdash M = N : A$ then $\Gamma^- \vdash M = N \in [\![A^-]\!]$, and
2. if $\Gamma^- \vdash M = N \in [\![A^-]\!]$ then $\Gamma^- \vdash M \Longleftrightarrow N \in A^-$,

can be proved.

We define a Kripke logical relation inductively on simple types. At base type we require the property we eventually would like to prove. At higher types we reduce the property to those for simpler types. We also extend it further to include substitutions, where it is defined by induction over the structure of the matching context.

1. $\Delta \vdash M = N \in [\![\alpha]\!]$ iff $\Delta \vdash M \Longleftrightarrow N : \alpha$.

2. $\Delta \vdash M = N \in [\![\tau_1 \to \tau_2]\!]$ iff for every $\Delta'$ extending $\Delta$ and for all $M_1$ and $N_1$ such that $\Delta' \vdash M_1 = N_1 \in [\![\tau_1]\!]$ we have $\Delta' \vdash M\, M_1 = N\, N_1 \in [\![\tau_2]\!]$.

3. $\Delta \vdash A = B \in [\![\mathrm{type}^-]\!]$ iff $\Delta \vdash A \longleftrightarrow B : \mathrm{type}^-$.

4. $\Delta \vdash A = B \in [\![\tau \to \kappa]\!]$ iff for every $\Delta'$ extending $\Delta$ and for all $M$ and $N$ such that $\Delta' \vdash M = N \in [\![\tau]\!]$ we have $\Delta' \vdash A\, M = B\, N \in [\![\kappa]\!]$.

5. $\Delta \vdash \sigma = \theta \in [\![\cdot]\!]$ iff $\sigma = \cdot$ and $\theta = \cdot$.

6. $\Delta \vdash \sigma = \theta \in [\![\Theta, x{:}\tau]\!]$ iff $\sigma = (\sigma', M/x)$ and $\theta = (\theta', N/x)$ where $\Delta \vdash \sigma' = \theta' \in [\![\Theta]\!]$ and $\Delta \vdash M = N \in [\![\tau]\!]$.

Three general structural properties of the logical relations that we can show directly by induction are exchange, weakening, contraction, and strengthening. We will use only weakening.

It is straightforward to show that logically related terms are considered identical by the algorithm. This proof always proceeds by induction on the structure of the type. A small insight may be required to arrive at the necessary generalization of the induction hypothesis. Here, this involves the statement that structurally equal terms are logically related. This has an important consequence we will need later on, namely that variables and constants are logically related to themselves.

**Theorem 3 (Logically Related Terms are Algorithmically Equal)**

1. *If* $\Delta \vdash M = N \in [\![\tau]\!]$ *then* $\Delta \vdash M \Longleftrightarrow N : \tau$.

2. *If* $\Delta \vdash A = B \in [\![\kappa]\!]$, *then* $\Delta \vdash A \longleftrightarrow B : \kappa$.

3. *If* $\Delta \vdash M \longleftrightarrow N : \tau$ *then* $\Delta \vdash M = N \in [\![\tau]\!]$.

4. *If* $\Delta \vdash A \longleftrightarrow B : \kappa$ *then* $\Delta \vdash A = B \in [\![\kappa]\!]$.

**Proof:** By simultaneous induction on the structure of $\tau$. □

The other part of the logical relations argument states that two equal terms are logically related. This requires a sequence of lemmas regarding algorithmic equality and the logical relation.

We summarize the first three properties by simply stating that all logical relations we have defined are symmetric and transitive and relations on objects are closed under head expansion. These are proved by induction over the structure of the given simple type, kind, or context.

The critical lemma for completeness shows that definitionally equal terms are logically related under related substitutions.

**Lemma 4 (Definitionally Equal Terms are Logically Related under Substitutions)**

1. *If* $\Gamma \vdash M = N : A$ *and* $\Delta \vdash \sigma = \theta \in [\![\Gamma^-]\!]$ *then* $\Delta \vdash M[\sigma] = N[\theta] \in [\![A^-]\!]$.

2. *If* $\Gamma \vdash A = B : K$ *and* $\Delta \vdash \sigma = \theta \in [\![\Gamma^-]\!]$ *then* $\Delta \vdash A[\sigma] = B[\theta] \in [\![K^-]\!]$.

**Proof:** By induction on the derivation of definitional equality, using the prior lemmas in this section. □

Since it is easy to see that the identity substitutions are logically related, the above lemma directly implies that definitionally equal terms are logically related. The completeness of algorithmic equality is then a simple corollary.

**Corollary 5 (Completeness of Algorithmic Equality)**

1. *If* $\Gamma \vdash M = N : A$ *then* $\Gamma^- \vdash M \Longleftrightarrow N : A^-$.
2. *If* $\Gamma \vdash A = B : K$ *then* $\Gamma^- \vdash A \longleftrightarrow B : K^-$.

# 5 Soundness of Algorithmic Equality

In general, the algorithm for type-directed equality is not sound. However, when applied to valid objects of the same type, it is sound and relates only equal terms. This direction requires a number of lemmas regarding the nature of typing and equality, but it otherwise mostly straightforward.

The first properties we need are some obvious inversion principles, which determine the type or kind of a term or family up to definitional equality. We elide the statement of these properties here. A second crucial property is the injectivity of product types. The proof is direct, essentially because we have omitted family-level $\lambda$-abstraction. However, the proof requires the strong form of family functionality that we have built into our variant of the LF type theory. Note that in this section we always assume that the contexts in the given judgment are valid.

**Lemma 6 (Injectivity of Products)** *Assume* $\Gamma$ *is a valid context.*
*If* $\Gamma \vdash \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : \text{type}$ *then* $\Gamma \vdash A_1 = B_1 : \text{type}$ *and* $\Gamma, x{:}A_1 \vdash A_2 = B_2 : \text{type}$.

**Proof:** This follows directly from a slightly stronger statement, namely that $\Gamma \vdash A = \Pi x{:}B_1.\ B_2$ or $\Gamma \vdash \Pi x{:}B_1.\ B_2 = A$ imply that $A = \Pi x{:}A_1.\ A_2$ where $A_1$ and and $A_2$ satisfy the properties postulated above. The strong form of family functionality is required to make the induction go through. A similar property at the level of kinds is also necessary and follows analogously. □

To show the soundness of the algorithm on well-typed terms we next need subject reduction.

**Lemma 7 (Subject Reduction)** *Assume* $\Gamma$ *is a valid context.*
*If* $M \xrightarrow{\text{whr}} M'$ *and* $\Gamma \vdash M : A$ *then* $\Gamma \vdash M' : A$ *and* $\Gamma \vdash M = M' : A$.

**Proof:** By induction on the definition of weak head reduction, making use of the inversion and substitution lemmas. □

The soundness property of algorithmic equality now follows from subject reduction and validity (Lemma 2).

**Theorem 8 (Soundness of Algorithmic Equality)** *Assume* $\Gamma$ *is a valid context.*

1. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$ *and* $\Gamma^- \vdash M \Longleftrightarrow N : A^-$, *then* $\Gamma \vdash M = N : A$.
2. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : B$ *and* $\Gamma^- \vdash M \longleftrightarrow N : \tau$, *then* $\Gamma \vdash M = N : A$, $\Gamma \vdash A = B : \text{type}$ *and* $A^- = B^- = \tau$.

*Analogous results hold for families and kinds.*

**Proof:** By induction on the structure of the given derivations for algorithmic equality, using inversion on the typing derivations. □

**Corollary 9 (Logically Related Terms are Definitionally Equivalent)** *Assume* $\Gamma$ *is valid. If* $\Gamma \vdash M : A$, $\Gamma \vdash N : A$, *and* $\Gamma^- \vdash M = N \in [\![A^-]\!]$, *then* $\Gamma \vdash M = N : A$.

Now we have all the information in place to prove strengthening and show that the typing premises for $M$ and $N$ in the extensionality rule are redundant. We omit the formal statement of these properties.

# 6 Decidability of Definitional Equality and Type-Checking

In this section we show that the judgment for algorithmic equality constitutes a decision procedure on valid terms of the same type. This result is then lifted to yield decidability of all judgments in the LF type theory.

The first step is to show that equality is decidable for terms that are algorithmically equal to themselves. Note that this property does not depend on the soundness of completeness of algorithmic equality—it is a purely syntactic result. The second step uses completeness of algorithmic equality and reflexivity to show that every well-typed term is algorithmically equal to itself. These two observations, together with soundness and completeness of algorithmic equality, yield the decidability of definitional equality for well-typed terms.

We say a well-formed object is *normalizing* iff it is related to itself by the type-directed equivalence algorithm. More precisely, if $\Gamma \vdash M : A$, then $M$ is *normalizing* iff $\Gamma^- \vdash M \Longleftrightarrow M : A^-$. A well-formed term is *structurally normalizing* iff it is related to itself by the structural equivalence algorithm. That is, if $\Gamma \vdash M : A$, then $M$ is *structurally normalizing* iff $\Gamma^- \vdash M \longleftrightarrow M : A^-$, and similarly for families and kinds.

We elide the formal statement, but it is clear that the algorithmic equality judgments are decidable on normalizing terms. By soundness and completeness, the decidability of definitional equality is a simple corollary. From this, the decidability of a syntax-directed type-checking algorithm follows by standard methods as in [HHP93]. We therefore omit the development here.

# 7 Quasi-Canonical Forms

The representation techniques of LF mostly rely on compositional bijections between the expressions (including terms, formulas, deductions, *etc.*) of the object language and *canonical forms* in a meta-language, where canonical forms are $\eta$-long and $\beta$-normal forms. So if we are presented with an LF object $M$ of a given type $A$ and we want to know which object-language expression $M$ represents, we convert it to canonical form and apply the inverse of the representation function.

This leads to the question on how to compute the canonical form of a well-typed object $M$ of type $A$ in an appropriate context $\Gamma$. Generally, we would like to extract this information from a derivation that witnesses that $M$ is normalizing, that is, a derivation that shows that $M$ is algorithmically equal to itself. This idea cannot be applied directly in our situation, since a derivation $\Gamma^- \vdash M \Longleftrightarrow M : A^-$ yields no information on the type labels of the $\lambda$-abstractions in $M$. Fortunately, these turn out to be irrelevant: if we have an object $M$ of a given type $A$ that is in canonical form, possibly with the exception of some type labels, then the type labels are actually uniquely determined up to definitional equality.

We formalize this intuition by defining quasi-canonical forms (and the auxiliary notion of quasi-atomic forms) in which type labels have been deleted. A quasi-canonical form can easily be extracted from a derivation that shows that a term is normalizing. Quasi-canonical forms are sufficient to prove adequacy theorems for the representation, since the global type of a quasi-canonical form determines a corresponding LF object up to definitional equality applied to type labels. The set of *quasi-canonical (QC)* and *quasi-atomic (QA)* terms are defined by the following grammar:

$$\begin{array}{lrcl} \textit{Quasi-canonical objects} & \bar{\bar{M}} & ::= & \bar{M} \mid \lambda x.\, \bar{\bar{M}} \\ \textit{Quasi-atomic objects} & \bar{M} & ::= & x \mid c \mid \bar{M}\, \bar{\bar{M}} \end{array}$$

It is a simple matter to instrument the algorithmic equality relations to extra a common quasi-canonical or quasi-atomic form for the terms being compared. Note that only one quasi-canonical form need be extracted since two terms are algorithmically equivalent iff they have the same quasi-canonical form. The instrumented rules are as follows:

**Instrumented Type-Directed Object Equality**

$$\frac{M \xrightarrow{\text{whr}} M' \qquad \Delta \vdash M' \Longleftrightarrow N : \alpha \Uparrow \bar{\bar{O}}}{\Delta \vdash M \Longleftrightarrow N : \alpha \Uparrow \bar{\bar{O}}} \qquad \frac{N \xrightarrow{\text{whr}} N' \qquad \Delta \vdash M \Longleftrightarrow N' : \alpha \Uparrow \bar{\bar{O}}}{\Delta \vdash M \Longleftrightarrow N : \alpha \Uparrow \bar{\bar{O}}}$$

$$\frac{M \longleftrightarrow N : \alpha \downarrow \bar{O}}{M \Longleftrightarrow N : \alpha \Uparrow \bar{\bar{O}}} \qquad \frac{\Delta, x{:}\tau_1 \vdash M\, x \Longleftrightarrow N\, x : \tau_2 \Uparrow \bar{\bar{O}}}{\Delta \vdash M \Longleftrightarrow N : \tau_1 \to \tau_2 \Uparrow \lambda x.\, \bar{\bar{O}}}$$

**Instrumented Structural Object Equality**

$$\frac{x{:}\tau \text{ in } \Delta}{\Delta \vdash x \longleftrightarrow x : \tau \downarrow x} \qquad \frac{c{:}A \text{ in } \Sigma}{\Delta \vdash c \longleftrightarrow c : A^- \downarrow c}$$

$$\frac{\Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \to \tau_1 \downarrow \bar{O}_1 \qquad \Delta \vdash M_2 \Longleftrightarrow N_2 : \tau_2 \Uparrow \bar{\bar{O}}_2}{\Delta \vdash M_1\, M_2 \longleftrightarrow N_1\, N_2 : \tau_1 \downarrow \bar{O}_1\, \bar{\bar{O}}_2}$$

It follows from the foregoing development that every well-formed term has a unique quasi-canonical form. We now have the following theorem relating quasi-canonical forms to the usual development of the LF type theory. We write $|M|$ for the result of erasing the type labels from an object $M$.

**Theorem 10 (Quasi-Canonical and Quasi-Atomic Forms)** *Assume $\Gamma$ is valid.*

1. *If $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$ and $\Gamma^- \vdash M_1 \Longleftrightarrow M_2 : A^- \Uparrow \bar{\bar{O}}$, then there is an $N$ such that $|N| = \bar{\bar{O}}$, $\Gamma \vdash N : A$, $\Gamma \vdash M_1 = N : A$ and $\Gamma \vdash M_2 = N : A$ and this $N$ is unique up to definitional equality.*

2. *If $\Gamma \vdash M_1 : A_1$ and $\Gamma \vdash M_2 : A_2$ and $\Gamma^- \vdash M_1 \longleftrightarrow M_1 : \tau \downarrow \bar{O}$ then $\Gamma \vdash A_1 = A_2 :$ type, $A^- = B^- = \tau$ and there exists an $N$ such that $|N| = \bar{O}$, $\Gamma \vdash N : A$, $\Gamma \vdash M_1 = N : A$ and $\Gamma \vdash M_2 = N : A$ and this $N$ is unique up to definitional equality.*

**Proof:** By simultaneous induction on the instrumented equality derivations. It is critical that we have the types of the objects that are compared (and not just the approximate type) to that we can use this information to fill in the missing $\lambda$-labels. □

Note that the uniqueness of $N$ up to definitional equality affects only the type labels, since $O$ determines $N$ in all other respects. This result shows that all adequacy proofs for LF representation on canonical forms still hold. In fact, they can be carried out directly on quasi-canonical forms. We elide the simple example that shows how to accomplish this—the recipe is simply that on quasi-canonical forms the type labels on $\lambda$-abstractions can be determined from context.

# 8 Conclusions

We have presented a new, type-directed algorithm for definitional equality in the LF type theory. This algorithm improves on previous accounts by avoiding consideration of reduction and its associated meta-theory and by providing a practical method for testing definitional equality in an implementation. The algorithm also yields a notion of canonical form, which we call quasi-canonical, that is suitable for proving the adequacy of encodings in a logical framework. The omission of type labels presents no difficulties for the methodology of LF, essentially because abstractions arise only in contexts where the domain type is known. The formulation of the algorithm and its proof of correctness relies on the "shapes" of types, from which dependencies on terms have been eliminated.

Surprisingly, it was the soundness proof for the algorithm, and not its completeness proof, which presented some technical difficulties. In particular, we have eliminated family-level $\lambda$-abstractions from our formulation of the type theory in order to prove injectivity of products syntactically. However, this means that it is no longer possible to $\beta$-expand a type, convert the argument object of the resulting application, and substitute the converted object back into the type. As a result we had to add a primitive family-level rule to simulate the effect of this operation. It may be possible to adapt the technique of our completeness theorem to other formulations of the type theory and establish soundness by other means.

The type-directed approach for equality checking scales to richer languages such as those with unit types, precisely because it makes use of type information during comparison. For example, one expects that any two variables of unit type are equal, even though they are structurally distinct head normal forms. A similar approach is used by Stone and Harper [SH99] to study a dependent type theory with singleton kinds and subkinding. There it is impossible to eliminate dependencies, resulting in a substantially more complex correctness proof, largely because of the loss of symmetry in the presence of dependencies. Nevertheless, the fundamental method is the same, and results in a practical approach to checking definitional equality for a rich type theory.

# References


[Cer96] Iliano Cervesato. *A Linear Logical Framework*. PhD thesis, Dipartimento di Informatica, Università di Torino, February 1996.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[CP98] Iliano Cervesato and Frank Pfenning. A linear logical framework. *Information and Computation*, 1998. To appear in a special issue with invited papers from LICS'96, E. Clarke, editor.

[DHW93] Gilles Dowek, Gérard Huet, and Benjamin Werner. On the definition of the eta-long normal form in type systems of the cube. In Herman Geuvers, editor, *Informal Proceedings of the Workshop on Types for Proofs and Programs*, Nijmegen, The Netherlands, May 1993.

[FM90] Amy Felty and Dale Miller. Encoding a dependent-type $\lambda$-calculus in a logic programming language. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 221–235, Kaiserslautern, Germany, July 1990. Springer-Verlag LNCS 449.

[Geu92] Herman Geuvers. The Church-Rosser property for $\beta\eta$-reduction in typed $\lambda$-calculi. In A. Scedrov, editor, *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 453–460, Santa Cruz, California, June 1992.

[Gha97] Neil Ghani. Eta-expansions in dependent type theory — the calculus of constructions. In P. de Groote and J.R. Hindley, editors, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 164–180, Nancy, France, April 1997. Springer-Verlag LNCS 1210.

[Gog99] Healfdene Goguen. Soundness of the logical framework for its typed operational semantics. In Jean-Yves Girard, editor, *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA'99)*, pages 177–197, L'Aquila, Italy, April 1999. Springer-Verlag LNCS 1581.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[HP99] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. Technical Report CMU-CS-99-159, Department of Computer Science, Carnegie Mellon University, 1999. Forthcoming.

[JG95] C. B. Jay and N. Ghani. The virtues of $\eta$-expansion. *Journal of Functional Programming*, 5(2):135–154, 1995.

[Pfe92] Frank Pfenning. Computation and deduction. Unpublished lecture notes, 277 pp. Revised May 1994, April 1996, May 1992.

[Pfe93] Frank Pfenning. Refinement types for logical frameworks. In Herman Geuvers, editor, *Informal Proceedings of the Workshop on Types for Proofs and Programs*, pages 285–299, Nijmegen, The Netherlands, May 1993.

[PP99] Jeff Polakow and Frank Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In J.-Y. Girard, editor, *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA'99)*, pages 295–309, L'Aquila, Italy, April 1999. Springer-Verlag LNCS 1581.

[Sal90] Anne Salvesen. The Church-Rosser theorem for LF with $\beta\eta$-reduction. Unpublished notes to a talk given at the First Workshop on Logical Frameworks in Antibes, France, May 1990.

[SH99] Chris Stone and Robert Harper. Decidable equivalence for singleton kinds. Submitted for publication., July 1999.

[Vir99] Roberto Virga. *Higher-Order Rewriting with Dependent Types*. PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 1999. Forthcoming.