A Simplified Account of the Metatheory of Linear LF [1]

Joseph C. Vanderwaart and Karl Crary

School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Abstract

We present a variant of the linear logical framework LLF that avoids the restriction that well-typed terms be in pre-canonical form and adds λ-abstraction at the level of families. We abandon the use of β-conversion as definitional equality in favor of a set of typed definitional equality judgments that include rules for parallel conversion and extensionality. We show type-checking is decidable by giving an algorithm to decide definitional equality for well-typed terms and showing the algorithm is sound and complete. The algorithm and the proof of its correctness are simplified by the fact that they apply only to well-typed terms and may therefore ignore the distinction between intuitionistic and linear hypotheses.

1 Introduction

Decidability of type-checking and the existence of canonical forms for well-typed terms are arguably the two most important metatheoretic results for a logical framework such as LF [5]. Type-checking is essential because the checking of proofs reduces to type-checking of the terms that represent them; canonical forms are crucial because it is the canonical terms (of certain types) that may be proven via an "adequacy" theorem to be in a meaningful correspondence with propositions and proofs in a logic.

Canonical forms in the LF type theory are β-normal, η-long forms. It therefore seems reasonable to take definitional equality to be $\beta\eta$-conversion, and decide whether two terms are equal by reducing them to $\beta\eta$-normal form and comparing; unfortunately, η-reduction is not as well behaved as β-reduction

[1] This material is based on work supported in part by NSF grants CCR-9984812 and CCR-0121633. Any opinions, findings, and conclusions or recommendations in this publication are those of the authors and do not reflect the views of this agency.

and so this approach encounters significant problems. The original presentation of LF by Harper, Honsell and Plotkin [5] (hereafter "HHP") avoided the difficulties of η-reduction by using β-conversion as definitional equality even though this destroyed the property that every term is equal to some canonical form.

Felty's Canonical LF [4] is a version of LF where all well-typed objects and families are in canonical form, avoiding all issues of definitional equality. Felty showed that Canonical LF is essentially the same as full LF if typing derivations are restricted to pre-canonical terms (those whose β-normal forms are canonical) with β-conversion as definitional equality. A similar approach was taken by Cervesato and Pfenning for their presentation of the linear logical framework LLF [1] (hereafter "CP"). The typing rules of LLF forced well-formed terms into η-long form, making all well-typed β-normal forms canonical and rendering any η rules for definitional equality unnecessary.

Subsequent to the original definition of LLF, Harper and Pfenning [6,7] (hereafter, "HP") gave an alternate formulation of ordinary LF that allowed a clean treatment of definitional equality without having to restrict terms to pre-canonical form. Their approach was based on the use of a typed definitional equality judgment as opposed to an untyped reduction relation—that is, they focused on comparing two objects *at a certain type* and *in a certain typing context*, rather than $\beta\eta$-normalizing them in isolation. The decidability of type-checking and the existence of canonical forms for well-typed terms were then established by giving a set of algorithmic judgments that are sound and complete with respect to definitional equality and can be instrumented to extract canonical forms from the terms they compare. The algorithm is similar to one introduced by Coquand [2], except that Coquand's algorithm performs η-expansion based on the shapes of terms, while HP's is directed by types and kinds. The type-directed nature of the algorithm has the advantage of making it scalable to theories such as LLF that contain unit types. Apart from the typed, declarative formulation of definitional equality, the LF type theory considered by HP was essentially the same as that of HHP, except that the λ-abstraction construct at the level of type families had to be removed for technical reasons. This was not considered a problem, as experience with LF had shown that this form of abstraction was not needed in practice.

In this paper, we present a variant of LLF that employs a typed formulation of definitional equality in the style of HP, along with a sound and complete equality algorithm. We also resolve the technical issues that necessitated the removal of the family-level λ-abstraction, and so we are able to give a variant of LLF that includes abstraction at the family level even though CP's LLF did not have this feature. This result has been applied by the authors to establish the decidability of typing in the LTT type theory for certified code [3], and has been extended by Polakow for the Ordered Linear Logical Framework [9].

1.1 Overview

The structure of this paper is as follows. In Section 2 we present our variant of the Linear LF type theory, which will essentially be an extension of HP's formulation of LF with linear implication, additive conjunction and additive truth. We will also add a family-level λ-abstraction construct similar to the one that appeared in HHP. Our presentation of the theory will include the typed definitional equality rules we are using to replace the untyped conversion relation of CP. The remainder of the paper focuses on proving that definitional equality is decidable. Section 3 defines a set of "algorithmic equality" judgments, in the style of HP, that are directed by the shapes of the types of the objects being compared. Some elementary properties of the algorithmic judgments are also proven in that section. The next two sections prove that the new judgments are correct—that is, sound and complete with respect to definitional equality—for well-typed terms; soundness is established in Section 4 and completeness in Section 5. Both of these proofs more or less follow the method of HP, although the linear features of Linear LF make the soundness proof somewhat more complicated. Finally in Section 6 we verify that the "algorithmic" judgments we introduced in Section 3 do in fact define an algorithm—that is, that the search for an algorithmic equality derivation between two well-typed terms will always terminate. This result is equivalent to the decidability of definitional equality and thus implies the decidability of typing in Linear LF.

2 A Variant of Linear LF

In this section we will present our variant of Linear LF type theory. This version of Linear LF has essentially the same syntax as that presented by CP, with the addition of λ-abstraction at the level of families. The typing rules we give here, however, differ from CP's in that we do not restrict terms to precanonical form. Another minor difference is that we use two separate contexts in our typing judgments, one for intuitionistic and one for linear assumptions, rather than combining the two. Since none of the variables declared in the linear context will appear in any types, we may identify linear contexts that differ only in the order of declarations and thus we do not need a formal context-splitting judgment for the linear application rules. Our variant of Linear LF can also be thought of as a linear version of the LF type theory as presented by HP.

The syntax of Linear LF terms is given by the grammar in Figure 1. There are three classes, or levels, of terms: objects, families and kinds. Kinds classify families: families of kind Type are called *types*, and classify objects, while $\Pi u{:}A.K$ is the (possibly dependent) kind of a function from objects of family A to families of kind K. At the family level we have constants (a), function ab-

Kinds	K	$::=$	$\mathsf{Type} \mid \Pi u{:}A.K$
Families	A	$::=$	$a \mid \lambda u{:}A_1.A_2 \mid A\,M \mid$
			$\Pi u{:}A_1.A_2 \mid$
			$A_1 \multimap A_2 \mid A_1 \,\&\, A_2 \mid \top$
Objects	M	$::=$	$u \mid c \mid$
			$\lambda u{:}A.M \mid M_1\,M_2 \mid$
			$\hat{\lambda} u{:}A.M \mid M_1 \hat{}\, M_2 \mid$
			$\langle M_1, M_2 \rangle \mid \pi_i M \mid \langle\rangle$
Contexts	Γ	$::=$	$\epsilon \mid \Gamma, a{:}K \mid \Gamma, u{:}A$
Linear Contexts	Δ	$::=$	$\epsilon \mid \Delta, u\hat{:}A$

Fig. 1. Syntax of Linear LF

Judgment	Meaning
$\vdash S\ \mathsf{sig}$	S is a valid signature
$\vdash \Gamma\ \mathsf{context}$	Γ is a valid context
$\Gamma \vdash \Delta\ \mathsf{context}$	Δ is a valid linear context
$\Gamma \vdash K : \mathsf{kind}$	K is a valid kind
$\Gamma \vdash A : K$	A is a family of kind K
$\Gamma; \Delta \vdash M : A$	M is an object of type A
$\Gamma \vdash K = L : \mathsf{kind}$	K and L are equal kinds
$\Gamma \vdash A = B : K$	A and B are equal families
$\Gamma; \Delta \vdash M = N : A$	M and N are equal objects

Fig. 2. Typing Judgment Forms

stractions and applications as well as (intuitionistic) dependent function types, linear implication ($\multimap$), additive conjunction ($\&$), and additive truth ($\top$). The object level contains variables (u), constants (c), intuitionistic λ-abstraction and application ($M_1\,M_2$), linear $\hat{\lambda}$-abstraction and application ($M_1 \hat{}\, M_2$), pairs ($\langle M_1, M_2, \rangle$), projections ($\pi_i\,M$) and the object $\langle\rangle$ that inhabits $\top$.

Contexts, both intuitionistic (Γ) and linear (Δ), will be used in the typing and definitional equality rules. Note that the order of assumptions in an intuitionistic context may be significant, since term variables may appear in types. The typing rules will prevent types from depending on any linear assumptions, which justifies our grouping all the linear assumptions into a separate linear context. We regard linear contexts that differ only in the ordering of assumptions as identical.

If the variable u does not appear in the family B, then we may write $A \to B$ for $\Pi u{:}A.B$. Similarly, $A \to K$ will mean $\Pi u{:}A.K$ if u does not appear in K. As usual, we denote by $E[E'_1, \ldots, E'_n/u_1, \ldots, u_n]$ the simultaneous capture-avoiding substitution of the terms E'_1 through E'_n for the corresponding variables u_1 through u_n in E.

$$\frac{}{\vdash \epsilon\ \mathsf{sig}} \qquad \frac{\vdash S\ \mathsf{sig} \quad \vdash_S K : \mathsf{kind}}{\vdash S, a{:}K\ \mathsf{sig}}\ (a \notin \mathrm{Dom}(S))$$

$$\frac{\vdash S\ \mathsf{sig} \quad \vdash_S A : \mathsf{Type}}{\vdash S, c{:}A\ \mathsf{sig}}\ (c \notin \mathrm{Dom}S)$$

$$\frac{}{\vdash \epsilon\ \mathsf{context}} \qquad \frac{\vdash \Gamma\ \mathsf{context} \quad \Gamma \vdash A : \mathsf{Type}}{\vdash \Gamma, u{:}A\ \mathsf{context}}\ (u \notin \mathrm{Dom}(\Gamma))$$

$$\frac{}{\Gamma \vdash \epsilon\ \mathsf{context}} \qquad \frac{\Gamma \vdash \Delta\ \mathsf{context} \quad \Gamma \vdash A : \mathsf{Type}}{\Gamma \vdash \Delta, u\hat{:}A\ \mathsf{context}}\ (u \notin \mathrm{Dom}(\Gamma), \mathrm{Dom}(\Delta))$$

Fig. 3. Rules for Well-Formed Contexts

$$\frac{}{\Gamma \vdash \mathsf{Type} : \mathsf{kind}} \qquad \frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, u{:}A \vdash K : \mathsf{kind}}{\Gamma \vdash \Pi u{:}A.K : \mathsf{kind}} \qquad \frac{(S(a) = K)}{\Gamma \vdash a : K}$$

$$\frac{\Gamma \vdash A : K' \quad \Gamma \vdash K' = K : \mathsf{kind}}{\Gamma \vdash A : K} \qquad \frac{\Gamma \vdash A_1 : \mathsf{Type} \quad \Gamma, u{:}A_1 \vdash A_2 : K}{\Gamma \vdash \lambda u{:}A_1.A_2 : \Pi u{:}A_1.K}$$

$$\frac{\Gamma \vdash A : \Pi u{:}A_1.K \quad \Gamma; \epsilon \vdash M : A_1}{\Gamma \vdash A\, M : K[M/u]} \qquad \frac{\Gamma \vdash A_1 : \mathsf{Type} \quad \Gamma \vdash A_2 : \mathsf{Type}}{\Gamma \vdash A_1 \multimap A_2 : \mathsf{Type}}$$

$$\frac{\Gamma \vdash A_1 : \mathsf{Type} \quad \Gamma, u{:}A_1 \vdash A_2 : \mathsf{Type}}{\Gamma \vdash \Pi u{:}A_1.A_2 : \mathsf{Type}} \qquad \frac{\Gamma \vdash A_1 : \mathsf{Type} \quad \Gamma \vdash A_2 : \mathsf{Type}}{\Gamma \vdash A_1 \,\&\, A_2 : \mathsf{Type}}$$

$$\frac{}{\Gamma \vdash \top : \mathsf{Type}} \qquad \frac{(S(c) = A)}{\Gamma; \epsilon \vdash c : A} \qquad \frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma; \Delta, u\hat{:}A \vdash M : B}{\Gamma; \Delta \vdash \hat{\lambda} u{:}A.M : A \multimap B}$$

$$\frac{(\Gamma(u) = A)}{\Gamma; \epsilon \vdash u : A} \qquad \frac{\Gamma; \Delta \vdash M_1 : A_1 \quad \Gamma; \Delta \vdash M_2 : A_2}{\Gamma; \Delta \vdash \langle M_1, M_2 \rangle : A_1 \& A_2} \qquad \frac{}{\Gamma; \Delta \vdash \langle\rangle : \top}$$

$$\frac{}{\Gamma; u\hat{:}A \vdash u : A} \qquad \frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, u{:}A; \Delta \vdash M : B}{\Gamma; \Delta \vdash \lambda u{:}A.M : \Pi u{:}A.B}$$

$$\frac{\Gamma; \Delta \vdash M : A_1 \& A_2}{\Gamma; \Delta \vdash \pi_i M : A_i}\ (i = 1, 2) \qquad \frac{\Gamma; \Delta_1 \vdash M_1 : A \multimap B \quad \Gamma; \Delta_2 \vdash M_2 : A}{\Gamma; \Delta_1, \Delta_2 \vdash M_1 \hat{}\, M_2 : B}$$

$$\frac{\Gamma; \Delta \vdash M_1 : \Pi u{:}A.B \quad \Gamma; \epsilon \vdash M_2 : A}{\Gamma; \Delta \vdash M_1\, M_2 : B[M_2/u]}$$

$$\frac{\Gamma; \Delta \vdash M : A' \quad \Gamma \vdash A' = A : \mathsf{Type}}{\Gamma; \Delta \vdash M : A}$$

Fig. 4. Typing Rules

2.1 Typing and Definitional Equality Rules

In LF and LLF, the types and kinds of constants are given by a *signature*. It is by the choice of signature that the logical framework is instantiated to represent a particular logic. For simplicity, in the bulk of this paper we will tacitly assume a fixed signature S; the exception is in the well-formedness

rules for signatures, where we use a subscripted turnstile ($\vdash_S$) to indicate that certain premises are to be understood with respect to the signature mentioned in the conclusion.

The judgment forms of the LLF type theory are shown in Figure 2; the typing rules are shown in Figure 4, and the definitional equality rules are shown in Figures 5 and 7. The validity judgment for linear contexts specifies an intuitionistic context giving types to all the free variables that may occur in the linear assumptions. The restriction that linear assumptions not appear in types or kinds is enforced by using only an intuitionistic context for kind and family judgments, and forcing the linear context in which an object is typed for dependent application to be empty.

The definitional equality rules include compatibility rules for all the syntactic constructs of the calculus, as well as parallel conversion rules and extensionality rules for each of the type and kind constructors of the theory. Symmetry and transitivity of equality are explicitly included as rules; reflexivity is shown to be admissible.

2.2 Injectivity

In the proof of soundness for the algorithm, it will be important that the type constructors of Linear LF are *injective* up to definitional equality; that is, if $A \multimap B = C \multimap D$, then $A = C$ and $B = D$. This property is formalized in the following lemma.

Lemma 2.1 (Injectivity)

(i) *If* $\Gamma \vdash \Pi u{:}A_1.A_2 = \Pi u{:}B_1.B_2 : \mathsf{Type}$ *and* $\vdash \Gamma$ $\mathsf{context}$ *then* $\Gamma \vdash A_1 = B_1 : \mathsf{Type}$ *and* $\Gamma, u{:}A_1 \vdash A_2 = B_2 : \mathsf{Type}$.

(ii) *If* $\Gamma \vdash A_1 \multimap A_2 = B_1 \multimap B_2 : \mathsf{Type}$*and* $\vdash \Gamma$ $\mathsf{context}$ *then* $\Gamma \vdash A_1 = B_1 : \mathsf{Type}$ *and* $\Gamma \vdash A_2 = B_2 : \mathsf{Type}$.

(iii) *If* $\Gamma \vdash A_1 \mathbin{\&} A_2 = B_1 \mathbin{\&} B_2 : \mathsf{Type}$ *and* $\vdash \Gamma$ $\mathsf{context}$ *then* $\Gamma \vdash A_1 = B_1 : \mathsf{Type}$ *and* $\Gamma \vdash A_2 = B_2 : \mathsf{Type}$.

In the presence of λ-abstraction and parallel conversion rules at the family level, the proof of this lemma is nontrivial. We have estabilshed the injectivity property for our variant of Linear LF using a logical relations argument; the details are interesting, but space considerations prevent our including them here. The proof may be found in the companion technical report [10].

3 Equality Algorithm

3.1 Erasure

To avoid serious difficulties with dependencies on terms, the algorithm and Kripke logical relation presented by HP use *simple types* and *simple kinds* in

Compatibility

$$\frac{(\Gamma(u) = A)}{\Gamma; \epsilon \vdash u = u : A} \qquad \frac{(S(c) = A)}{\Gamma; \epsilon \vdash c = c : A} \qquad \frac{}{\Gamma; u\hat{:}A \vdash u = u : A}$$

$$\frac{\Gamma; \Delta \vdash M_1 = M_2 : \Pi u{:}A.B \quad \Gamma; \epsilon \vdash N_1 = N_2 : A}{\Gamma; \Delta \vdash M_1\, N_1 = M_2\, N_2 : B[N_1/u]}$$

$$\frac{\Gamma; \Delta_1 \vdash M_1 = M_2 : A \multimap B \quad \Gamma; \Delta_2 \vdash N_1 = N_2 : A}{\Gamma; \Delta_1, \Delta_2 \vdash M_1 \hat{} N_1 = M_2 \hat{} N_2 : B}$$

$$\frac{\Gamma; \Delta \vdash M_1 = M_2 : A_1 \quad \Gamma; \Delta \vdash N_1 = N_2 : A_2}{\Gamma; \Delta \vdash \langle M_1, N_1 \rangle = \langle M_2, N_2 \rangle : A_1 \,\&\, A_2} \qquad \frac{(i = 1, 2) \quad \Gamma; \Delta \vdash M_1 = M_2 : A_1 \,\&\, A_2}{\Gamma; \Delta \vdash \pi_i M_1 = \pi_i M_2 : A_i}$$

$$\frac{\Gamma \vdash A_1 = A : \mathsf{Type} \quad \Gamma \vdash A_2 = A : \mathsf{Type} \quad \Gamma, u{:}A; \Delta \vdash M_1 = M_2 : B}{\Gamma; \Delta \vdash \lambda u{:}A_1.M_1 = \lambda u{:}A_2.M_2 : \Pi u{:}A.B}$$

$$\frac{\Gamma \vdash A_1 = A : \mathsf{Type} \quad \Gamma \vdash A_2 = A : \mathsf{Type} \quad \Gamma; \Delta, u\hat{:}A \vdash M_1 = M_2 : B}{\Gamma; \Delta \vdash \hat{\lambda} u{:}A_1.M_1 = \hat{\lambda} u{:}A_2.M_2 : A \multimap B}$$

Type Conversion

$$\frac{\Gamma; \Delta \vdash M_1 = M_2 : A' \quad \Gamma \vdash A' = A : \mathsf{Type}}{\Gamma; \Delta \vdash M_1 = M_2 : A}$$

Equivalence

$$\frac{\Gamma; \Delta \vdash M_2 = M_1 : A}{\Gamma; \Delta \vdash M_1 = M_2 : A} \qquad \frac{\Gamma; \Delta \vdash M_1 = M_3 : A \quad \Gamma; \Delta \vdash M_3 = M_2 : A}{\Gamma; \Delta \vdash M_1 = M_2 : A}$$

Parallel Conversion

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, u{:}A; \Delta \vdash M_1 = M_2 : B \quad \Gamma; \epsilon \vdash N_1 = N_2 : A}{\Gamma; \Delta \vdash (\lambda u{:}A.M_1)\, N_1 = M_2[N_2/u] : B[N_1/u]}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma; \Delta_1, u\hat{:}A \vdash M_1 = M_2 : B \quad \Gamma; \Delta_2 \vdash N_1 = N_2 : A}{\Gamma; \Delta_1, \Delta_2 \vdash (\hat{\lambda} u{:}A.M_1) \hat{} N_1 = M_2[N_2/u] : B}$$

$$\frac{\Gamma; \Delta \vdash M_1 = N_1 : A_1 \quad \Gamma; \Delta \vdash M_2 = N_2 : A_2}{\Gamma; \Delta \vdash \pi_i \langle M_1, M_2 \rangle = N_i : A_i}\ (i = 1, 2)$$

Fig. 5. Definitional Equality Rules: Objects (Except Extensionality)

place of ordinary families and kinds. Not only are the type-directed phase of HP's algorithm directed by simple types and the Kripke logical relation indexed by simple types and kinds, but the contexts (or "worlds") in both the algorithmic judgments and the logical relation give only simple types to variables. The process of erasing ordinary families and kinds into simple ones effectively identifies types that differ only in the terms that appear in them.

We adopt this practice of erasure as well, extending it to erase the distinction between intuitionistic and linear assumptions in a context. The need for this arises because of the splitting of the linear context that occurs in the

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma;\Delta \vdash M_1 : \Pi u{:}A.B \quad \Gamma;\Delta \vdash M_2 : \Pi u{:}A.B \quad \Gamma, u{:}A;\Delta \vdash M_1\, u = M_2\, u : B}{\Gamma;\Delta \vdash M_1 = M_2 : \Pi u{:}A.B}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma;\Delta \vdash M_1 : A \multimap B \quad \Gamma;\Delta \vdash M_2 : A \multimap B \quad \Gamma;\Delta, u\hat{:}A \vdash M_1\hat{}u = M_2\hat{}u : B}{\Gamma;\Delta \vdash M_1 = M_2 : A \multimap B}$$

$$\frac{\Gamma;\Delta \vdash M_1 : A_1 \,\&\, A_2 \quad \Gamma;\Delta \vdash M_2 : A_1 \,\&\, A_2 \quad \Gamma;\Delta \vdash \pi_i M_1 = \pi_i M_2 : A_i \text{ for } i = 1,2}{\Gamma;\Delta \vdash M_1 = M_2 : A_1 \,\&\, A_2} \qquad \frac{\Gamma;\Delta \vdash M : \top \quad \Gamma;\Delta \vdash N : \top}{\Gamma;\Delta \vdash M = N : \top}$$

Fig. 6. Definitional Equality Rules: Object Extensionality

$$\frac{}{\Gamma \vdash \mathsf{Type} = \mathsf{Type} : \mathsf{kind}} \qquad \frac{\Gamma \vdash K_2 = K_1 : \mathsf{kind}}{\Gamma \vdash K_1 = K_2 : \mathsf{kind}}$$

$$\frac{\Gamma \vdash K_1 = K_3 : \mathsf{kind} \quad \Gamma \vdash K_3 = K_2 : \mathsf{kind}}{\Gamma \vdash K_1 = K_2 : \mathsf{kind}}$$

$$\frac{\Gamma \vdash A_1 : \mathsf{Type} \quad \Gamma \vdash A_1 = A_2 : \mathsf{Type} \quad \Gamma, u{:}A_1 \vdash K_1 = K_2 : \mathsf{kind}}{\Gamma \vdash \Pi u{:}A_1.K_1 = \Pi u{:}A_2 s.K_2 : \mathsf{kind}}$$

$$\frac{\Gamma \vdash A_1 = A : \mathsf{Type} \quad \Gamma \vdash A_2 = A : \mathsf{Type} \quad \Gamma, u{:}A \vdash B_1 = B_2 : K}{\Gamma \vdash \lambda u{:}A_1.B_1 = \lambda u{:}A_2.B_2 : \Pi u{:}A.K}$$

$$\frac{\Gamma \vdash B : \mathsf{Type} \quad \Gamma, u{:}B \vdash A_1 = A_2 : K \quad \Gamma;\epsilon \vdash M_1 = M_2 : B}{\Gamma \vdash (\lambda u{:}B.A_1)\, M_1 = A_2[M_2/u] : K[M_1/u]}$$

$$\frac{\Gamma \vdash B : \mathsf{Type} \quad \Gamma \vdash A_1 : \Pi u{:}B.K \quad \Gamma \vdash A_2 : \Pi u{:}B.K \quad \Gamma, u{:}B \vdash A_1\, u = A_2\, u : K}{\Gamma \vdash A_1 = A_2 : \Pi u{:}B.K} \qquad \frac{(S(a) = K)}{\Gamma \vdash a = a : K}$$

$$\frac{\Gamma \vdash A_1 = A_2 : \mathsf{Type} \quad \Gamma \vdash B_1 = B_2 : \mathsf{Type}}{\Gamma \vdash A_1 \multimap B_1 = A_2 \multimap B_2 : \mathsf{Type}} \qquad \frac{}{\Gamma \vdash \top = \top : \mathsf{Type}}$$

$$\frac{\Gamma \vdash A_1 = A_3 : K \quad \Gamma \vdash A_3 = A_2 : K}{\Gamma \vdash A_1 = A_2 : K} \qquad \frac{\Gamma \vdash A_2 = A_1 : K}{\Gamma \vdash A_1 = A_2 : K}$$

$$\frac{\Gamma \vdash A_1 = A_2 : \mathsf{Type} \quad \Gamma \vdash B_1 = B_2 : \mathsf{Type}}{\Gamma \vdash A_1 \& B_1 = A_2 \& B_2 : \mathsf{Type}} \qquad \frac{\Gamma \vdash A_1 = A_2 : \Pi u{:}B.K \quad \Gamma;\epsilon \vdash M_1 = M_2 : B}{\Gamma \vdash A_1\, M_1 = A_2\, M_2 : K[M_1/u]}$$

$$\frac{\Gamma \vdash A_1 = A_2 : \mathsf{Type} \quad \Gamma, u{:}A_1 \vdash B_1 = B_2 : \mathsf{Type}}{\Gamma \vdash \Pi u{:}A_1.B_1 = \Pi u{:}A_2.B_2 : \mathsf{Type}} \qquad \frac{\Gamma \vdash A_1 = A_2 : K' \quad \Gamma \vdash K' = K : \mathsf{kind}}{\Gamma \vdash A_1 = A_2 : K}$$

Fig. 7. Definitional Equality Rules: Kinds and Families

$$
\begin{array}{lrcl}
\text{Simple Kinds} & \kappa & ::= & \mathsf{t}^- \mid \tau \to \kappa \\
\text{Simple Types} & \tau & ::= & \alpha \mid \tau_1 \to \tau_2 \mid \tau_1 \multimap \tau_2 \mid \tau_1 \,\&\, \tau_2 \mid \top \\
\text{Simple Contexts} & \Sigma & ::= & \epsilon \mid \Sigma, u{:}\tau
\end{array}
$$

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
(a)^- & = & \alpha & (\mathsf{Type})^- & = & \mathsf{t}^- \\
(\lambda u{:}A_1.A_2)^- & = & A_2^- & (\Pi u{:}A.K)^- & = & A^- \to K^- \\
(A\,M)^- & = & A^- & & & \\
(\Pi u{:}A_1.A_2)^- & = & A_1^- \to A_2^- & (\epsilon)^- & = & \epsilon \\
(A_1 \multimap A_2)^- & = & A_1^- \multimap A_2^- & (\Gamma, u{:}A)^- & = & \Gamma^-, u{:}A^- \\
(A_1 \,\&\, A_2)^- & = & A_1^- \,\&\, A_2^- & (\Delta, u\hat{:}A)^- & = & \Delta^-, u{:}A^- \\
(\top)^- & = & \top & & &
\end{array}
$$

Fig. 8. Simple Types and Erasure

typing and definitional equality rules for linear function applications. If this context-splitting were to be enforced in the algorithmic judgments, then the transitivity proof for those judgments would be upset by the possibility that different derivations mentioning the same linear application term might split the context differently.

Essentially, we want to avoid this problem by not requiring the algorithmic equality rule for linear applications to split the linear context. Such a change by itself would destroy the property that every linear assumption in a judgment must be used, so we also have to remove the restriction on linear variable use. However, this leaves us with two separate contexts that are treated in exactly the same way, so we go a step further and combine the intuitionistic and linear contexts into one. Consequently there is no distinction between intuitionistic and linear assumptions in the algorithm or logical relation. This does not affect soundness, since we only wish to prove the algorithm sound for well-typed terms, which must respect linearity.

Our grammar for families and kinds with no term dependencies, and contexts that combine intuitionistic and linear assumptions, is given in Figure 8. The erasure function $(\cdot)^-$ given in the figure maps ordinary families, kinds and contexts to simple ones.

To validate our intuition that erasure should remove all dependencies on terms, we prove the following lemma which states that substitution does not affect erasure and that definitionally equal terms have the same erasure.

Lemma 3.1 (Erasure Preservation)

(i) *For any family A, variable u and object M, $(A[M/u])^- = (A)^-$.*

(ii) *If $\Gamma \vdash A = B : K$, then $A^- = B^-$.*

(iii) *If $\Gamma \vdash K = L : \mathsf{kind}$, then $K^- = L^-$.*

$\Sigma \vdash M_1 \Longleftrightarrow M_2 : \tau$	Type-Directed Object Equality
$\Sigma \vdash M_1 \longleftrightarrow M_2 : \tau$	Structural Object Equality
$\Sigma \vdash A_1 \Longleftrightarrow A_2 : \kappa$	Kind-Directed Family Equality
$\Sigma \vdash A_1 \longleftrightarrow A_2 : \kappa$	Structural Family Equality
$\Sigma \vdash K_1 \longleftrightarrow K_2 : \mathsf{kind}^-$	Algorithmic Kind Equality

Fig. 9. Algorithmic Equality Judgment Forms

$$\frac{M \xrightarrow{\text{wh}} M' \quad \Sigma \vdash M' \Longleftrightarrow N : \alpha}{\Sigma \vdash M \Longleftrightarrow N : \alpha} \qquad \frac{N \xrightarrow{\text{wh}} N' \quad \Sigma \vdash M \Longleftrightarrow N' : \alpha}{\Sigma \vdash M \Longleftrightarrow N : \alpha}$$

$$\frac{\Sigma \vdash M \longleftrightarrow N : \alpha}{\Sigma \vdash M \Longleftrightarrow N : \alpha} \qquad \frac{}{\Sigma \vdash M \Longleftrightarrow N : \top}$$

$$\frac{\Sigma, u{:}\tau_1 \vdash M\,u \Longleftrightarrow N\,u : \tau_2}{\Sigma \vdash M \Longleftrightarrow N : \tau_1 \to \tau_2} \qquad \frac{\Sigma, u{:}\tau_1 \vdash M\hat{}u \Longleftrightarrow N\hat{}u : \tau_2}{\Sigma \vdash M \Longleftrightarrow N : \tau_1 \multimap \tau_2}$$

$$\frac{\Sigma \vdash \pi_1 M \Longleftrightarrow \pi_1 N : \tau_1 \quad \Sigma \vdash \pi_2 M \Longleftrightarrow \pi_2 N : \tau_2}{\Sigma \vdash M \Longleftrightarrow N : \tau_1 \,\&\, \tau_2}$$

$$\frac{(\Sigma(u) = \tau)}{\Sigma \vdash u \longleftrightarrow u : \tau} \qquad \frac{(S(c) = A)}{\Sigma \vdash c \longleftrightarrow c : A^-} \qquad \frac{\Sigma \vdash M_1 \longleftrightarrow M_2 : \tau_1 \,\&\, \tau_2}{\Sigma \vdash \pi_i M_1 \longleftrightarrow \pi_i M_2 : \tau_i}$$

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

$$\frac{\Sigma \vdash M_1 \longleftrightarrow M_2 : \tau_2 \multimap \tau_1 \quad \Sigma \vdash N_1 \Longleftrightarrow N_2 : \tau_2}{\Sigma \vdash M_1\hat{}N_1 \longleftrightarrow M_2\hat{}N_2 : \tau_1}$$

$$\frac{A \xrightarrow{\text{wh}} A' \quad \Sigma \vdash A' \Longleftrightarrow B : \mathsf{t}^-}{\Sigma \vdash A \Longleftrightarrow B : \mathsf{t}^-} \qquad \frac{B \xrightarrow{\text{wh}} B' \quad \Sigma \vdash A \Longleftrightarrow B' : \mathsf{t}^-}{\Sigma \vdash A \Longleftrightarrow B : \mathsf{t}^-}$$

$$\frac{\Sigma \vdash A \longleftrightarrow B : \mathsf{t}^-}{\Sigma \vdash A \Longleftrightarrow B : \mathsf{t}^-} \qquad \frac{\Sigma, u{:}\tau \vdash A\,u \Longleftrightarrow B\,u : \kappa}{\Sigma \vdash A \Longleftrightarrow B : \tau \to \kappa}$$

$$\frac{(S(a) = K)}{\Sigma \vdash a \longleftrightarrow a : K^-} \qquad \frac{\Sigma \vdash A_1 \Longleftrightarrow A_2 : \mathsf{t}^- \quad \Sigma \vdash B_1 \Longleftrightarrow B_2 : \mathsf{t}^-}{\Sigma \vdash A_1 \multimap B_1 \longleftrightarrow A_2 \multimap B_2 : \mathsf{t}^-}$$

$$\frac{\Sigma \vdash A_1 \longleftrightarrow A_2 : \tau \to \kappa \quad \Sigma \vdash M_1 \Longleftrightarrow M_2 : \tau}{\Sigma \vdash A_1\,M_1 \longleftrightarrow A_2\,M_2 : \kappa}$$

$$\frac{\Sigma \vdash A_1 \Longleftrightarrow A_2 : \mathsf{t}^- \quad \Sigma, u{:}A_1^- \vdash B_1 \Longleftrightarrow B_2 : \mathsf{t}^-}{\Sigma \vdash \Pi u{:}A_1.B_1 \longleftrightarrow \Pi u{:}A_2.B_2 : \mathsf{t}^-} \qquad \frac{}{\Sigma \vdash \top \longleftrightarrow \top : \mathsf{t}^-}$$

$$\frac{\Sigma \vdash A_1 \Longleftrightarrow A_2 : \mathsf{t}^- \quad \Sigma \vdash B_1 \Longleftrightarrow B_2 : \mathsf{t}^-}{\Sigma \vdash A_1 \,\&\, B_1 \longleftrightarrow A_2 \,\&\, B_2 : \mathsf{t}^-} \qquad \frac{}{\Sigma \vdash \mathsf{t}^- \longleftrightarrow \mathsf{t}^- : \mathsf{kind}^-}$$

$$\frac{\Sigma \vdash A_1 \Longleftrightarrow A_2 : \mathsf{t}^- \quad \Sigma, u{:}A_1^- \vdash K_1 \longleftrightarrow K_2 : \mathsf{kind}^-}{\Sigma \vdash \Pi u{:}A_1.K_1 \longleftrightarrow \Pi u{:}A_2.K_2 : \mathsf{kind}^-}$$

Fig. 10. Algorithmic Equality Rules

3.2 The Equality Algorithm

Our algorithmic equality judgment forms are shown in Figure 9. The rules are shown in Figure 10. For objects and families we give both classifier-directed rules (that is, type-directed rules for comparing objects and kind-directed rules for comparing families) and structural rules. The classifier-directed rules apply extensionality until a base classifier is reached, then reduce to weak head normal form and compare structurally. The structural rules compare the constant, variable or primitive head and revert to the classifier-directed phase of the algorithm for any other subterms. Since there are no classifiers for kinds (or, put another way, every kind is of the same sort), we only need structural rules to compare kinds.

Intuitively, the classifier-directed portion of the algorithm takes a context, two terms, and a classifier and attempts to derive the corresponding algorithmic equality judgment, returning either success or failure; the structural portion takes a context and two terms in weak-head normal form and attempts to synthesize a simple type or kind for which the structural equality judgment is derivable, returning that classifier if it exists.

Notice that the algorithm ignores all issues of linearity. There is no distinction between intuitionistic and linear assumptions—the algorithm does not enforce any restrictions on the number of times something may be used—and the structural rule for linear applications does not split the context. Later, we will see that in the soundness proof, all the necessary information about allocation of linear assumptions is extracted from the typing derivations rather than the derivations of algorithmic equality.

4 Soundness of Algorithmic Equality

In this section we will prove the soundness result for our algorithm; essentially, we want to show that if two terms are algorithmically equal then they are definitionally equal. It is clear, however, that this can only be true for well-typed terms. (Consider the type-directed rule at type $\top$!) Our soundness theorem will therefore have to require that typing derivations exist for the terms being compared. Since our algorithm does not enforce the linearity restrictions present in the definitional equality rules, the proof must also rely on the typing derivations to determine how linear contexts should be split among premises when dealing with linear function applications.

This can pose some difficulty if the two typing derivations disagree on how the context should be split. This can't be avoided, as equal terms may sometimes use their resources differently if unit expressions are involved. For example, in the context $\Delta = u{:}\top, v{:}\top, w{:}\top \multimap \top \multimap A$, the terms $(w\hat{}\langle\rangle)\hat{}u$ and $(w\hat{}\langle\rangle)\hat{}v$ are equal, but there is no linear context in which u and v may be simultaneously well-typed, let alone equal.

To solve this problem, we follow a suggestion proposed by Pfenning [8]. The key is to observe that the way to prove those two problematic applications equal is to use the fact that any variable of type $\top$ is equal to $\langle\rangle$. Using this extensionality rule and congruence rules, we prove that both of the above terms are equal to $(w\hat{}\langle\rangle)\hat{}\langle\rangle$; thus by transitivity they are equal to each other. But changing an expression of type $\top$ into $\langle\rangle$ is just η-expansion, and HP showed that the type-directed algorithm can be instrumented to find η-long forms. Therefore, the soundness proof should, rather than directly proving the algorithmically equal terms to be definitionally equal, extract a mediating term and prove that it is definitionally equal to both. Comparison with HP's discussion of pseudo-canonical forms strongly suggests that in the classifier-directed cases of the proof, this mediating term will be canonical except for the type labels on λ-abstractions, but we will not prove this.

Using this insight, we can prove the main lemma of this section, which will imply soundness of the algorithm. Given two terms that are well-formed and algorithmically equal, the proof constructs a term that is definitionally equal to each of them. Decisions about how the linear context should be split in the definitional equality derivations are made based on the given typing derivations, and since two separate equality derivations are being constructed, there is no need to attempt to resolve differences between the two typing derivations. Soundness of algorithmic equality follows directly from this lemma, using transitivity.

Lemma 4.1 *Assume* $\vdash \Gamma$ $\mathsf{context}$ *and, where applicable,* $\Gamma \vdash \Delta_i$ $\mathsf{context}$. *Assume further that if* $(u\hat{:}C) \in \Delta_1$ *and* $(u\hat{:}C') \in \Delta_2$), *then* $C = C'$.

(i) If $\Gamma; \Delta_1 \vdash M : A$ *and* $\Gamma; \Delta_2 \vdash N : A$ *and* $\Sigma \vdash M \Longleftrightarrow N : A^-$ *and* $\Gamma^-, \Delta_i^- \subseteq \Sigma$ *for each* i, *then there is some* P *such that* $\Gamma; \Delta_1 \vdash P = M : A$ *and* $\Gamma; \Delta_2 \vdash P = N : A$.

(ii) If $\Gamma; \Delta_1 \vdash M : A$ *and* $\Gamma; \Delta_2 \vdash N : B$ *and* $\Sigma \vdash M \longleftrightarrow N : \tau$ *and* $\Gamma^-, \Delta_i^- \subseteq \Sigma$ *for each* i, *then* $\Gamma \vdash A = B : \mathsf{Type}$ *and there is some* P *such that* $\Gamma; \Delta_1 \vdash P = M : A$ *and* $\Gamma; \Delta_2 \vdash P = N : A$ *and* $A^- = B^- = \tau$.

(iii) If $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *and* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$, *then* $\Gamma \vdash A = B : K$.

(iv) If $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : L$ *and* $\Gamma \vdash A \longleftrightarrow B : \kappa$, *then* $\Gamma \vdash A = B : K$ *and* $\Gamma \vdash K = L : \mathsf{kind}$ *and* $K^- = L^- = \kappa$.

(v) If $\Gamma \vdash K : \mathsf{kind}$ *and* $\Gamma \vdash L : \mathsf{kind}$ *and* $\Gamma^- \vdash K \longleftrightarrow L : \mathsf{kind}^-$ *then* $\Gamma \vdash K = L : \mathsf{kind}$.

Proof. By induction on the algorithmic equality derivation. We will only show two cases.

Case:

$$\frac{\Sigma, u\hat{:}\tau_1 \vdash M_1\,\hat{}\,u \Longleftrightarrow M_2\,\hat{}\,u : \tau_2}{\Sigma \vdash M_1 \Longleftrightarrow M_2 : \tau_1 \multimap \tau_2}$$

Note that $A = A_1 \multimap A_2$ where $A_1^- = \tau_1$ and $A_2^- = \tau_2$.
Using Regularity and inversion lemmas, $\Gamma \vdash A_1 : \mathsf{Type}$ and so
$\Gamma \vdash \Delta_i, u\hat{:}A_1$ $\mathsf{context}$ for each i.
By rules, $\Gamma; \Delta_i, u\hat{:}A_1 \vdash M_i\,\hat{}\,u : \tau_2$ for each i.
Also, note that $(\Delta_i, u\hat{:}A_1)^- \subseteq (\Sigma, u{:}\tau_1)$.
By the i.h., $\Gamma; \Delta_i, u\hat{:}A_1 \vdash P = M_i\,\hat{}\,u : A_2$ for each i.
By equivalence rules, $\Gamma; \Delta_i, u\hat{:}A_1 \vdash P = P : A_2$.
By rule, $\Gamma; u\hat{:}A_1 \vdash u = u : A_1$.
By parallel conversion, $\Gamma; \Delta_i, u\hat{:}A_1 \vdash (\hat{\lambda}u{:}A_1.P)\,\hat{}\,u = P : A_2$.
By transitivity, $\Gamma; \Delta_i, u\hat{:}A_1 \vdash (\hat{\lambda}u{:}A_1.P)\,\hat{}\,u = M_i\,\hat{}\,u : A_2$ for each i.
By extensionality, $\Gamma; \Delta_i \vdash \hat{\lambda}u{:}A_1.P = M_i : A_1 \multimap A_2$.

Case:

$$\frac{\Sigma \vdash M_1 \longleftrightarrow M_2 : \tau_2 \multimap \tau_1 \quad \Sigma \vdash N_1 \Longleftrightarrow N_2 : \tau_2}{\Sigma \vdash M_1\,\hat{}\,N_1 \longleftrightarrow M_2\,\hat{}\,N_2 : \tau_1}$$

By inversion, $\Delta_1 = (\Delta_1', \Delta_1'')$ and $\Gamma; \Delta_1' \vdash M_1 : A_2 \multimap A_1$ and $\Gamma; \Delta_1'' \vdash N_1 : A_2$ and $\Gamma \vdash A_1 = A : \mathsf{Type}$.
Similarly, $\Delta_2 = (\Delta_2', \Delta_2'')$ and $\Gamma; \Delta_2' \vdash M_2 : B_2 \multimap B_1$ and $\Gamma; \Delta_2'' \vdash N_1 : B_2$ and $\Gamma \vdash B_1 = B : \mathsf{Type}$.
Observe that $\Gamma \vdash \Delta_i'$ $\mathsf{context}$ and $\Gamma \vdash \Delta_i''$ $\mathsf{context}$.
By the i.h., $\Gamma \vdash C = A_2 \multimap A_1 : \mathsf{Type}$ and $\Gamma \vdash C = B_2 \multimap B_1 : \mathsf{Type}$ and $\Gamma; \Delta_i \vdash P = M_i : C$.
Also note that $A_1^- = B_1^- = \tau_1$ and $A_2^- = B_2^- = \tau_2$.
By rules, $\Gamma \vdash A_2 \multimap A_1 = B_2 \multimap B_1 : \mathsf{Type}$.
By injectivity, $\Gamma \vdash A_j = B_j : \mathsf{Type}$ for $j = 1, 2$.
By symmetry and transitivity, $\Gamma \vdash A = B : \mathsf{Type}$.
By type conversion, $\Gamma; \Delta_2'' \vdash N_2 : A_2$ and $\Gamma; \Delta_i' \vdash P = M_i : A_2 \multimap A_1$.
By the i.h., $\Gamma; \Delta_i'' \vdash Q = N_i : A_2$.
By a congruence rule, $\Gamma; \Delta_i \vdash P\,\hat{}\,Q = M_i\,\hat{}\,N_i : A_1$.
By type conversion, $\Gamma; \Delta_i \vdash P\,\hat{}\,Q = M_i\,\hat{}\,N_i : A$.
Using erasure preservation, $A^- = B^- = \tau_1$. □

Theorem 4.2 (Soundness of Algorithmic Equality) *Assume* $\vdash \Gamma$ context *and* $\Gamma \vdash \Delta$ context.

(i) *If* $\Gamma; \Delta \vdash M : A$ *and* $\Gamma; \Delta \vdash N : A$ *and* $\Gamma^-, \Delta^- \vdash M \Longleftrightarrow N : A^-$, *then* $\Gamma; \Delta \vdash M = N : A$.

(ii) *If* $\Gamma \vdash A : K$, $\Gamma \vdash B : K$ *and* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$, *then* $\Gamma \vdash A = B : K$.

(iii) *If* $\Gamma \vdash K : \mathsf{kind}$, $\Gamma \vdash L : \mathsf{kind}$ *and* $\Gamma^- \vdash K \longleftrightarrow L : \mathsf{kind}^-$, *then* $\Gamma \vdash K = L : \mathsf{kind}$.

Proof. By Lemma 4.1, symmetry and transitivity. □

5 Completeness of Algorithmic Equality

In this section we prove that algorithmic equality is complete — that is, that any two terms that are definitionally equal are also algorithmically equal. To do this, we define a Kripke logical relation in the style of HP such that logically related terms are algorithmically equal. We then prove that definitional equality implies the logical relation, thereby establishing completeness.

5.1 A Kripke Logical Relation

Our Kripke logical relation is defined inductively over the same simple types and kinds as were used in the algorithm, and is extended to include substitutions, where it is defined inductively over simple contexts. The worlds are simple contexts, ordered by inclusion. More formally, we will say that a context Σ' extends Σ, written $\Sigma' \succeq \Sigma$, if Σ' contains all the declarations in Σ and possibly more. Our logical relations on terms, families and substitutions are defined in Figure 11.

Lemma 5.1 (Monotonicity of Logical Relations) *Let* R *be any logical relation. If* $(\Sigma, \Sigma') \vdash R$ *then* $(\Sigma, u{:}\tau, \Sigma') \vdash R$.

Proof. By induction on the type or kind, using the weakening property of algorithmic equality. □

Our next lemma states that logically related terms are algorithmically equal, and that structurally equal terms are logically related.

Lemma 5.2

(i) *If* $\Sigma \vdash M = N \in [\![\tau]\!]$ *then* $\Sigma \vdash M \Longleftrightarrow N : \tau$.

(ii) *If* $\Sigma \vdash A = B \in [\![\kappa]\!]$ *then* $\Sigma \vdash A \Longleftrightarrow B : \kappa$.

(iii) *If* $\Sigma \vdash M \longleftrightarrow N : \tau$ *then* $\Sigma \vdash M = N \in [\![\tau]\!]$.

(iv) *If* $\Sigma \vdash A \longleftrightarrow B : \kappa$ *then* $\Sigma \vdash A = B \in [\![\kappa]\!]$.

Proof. By simultaneous structural induction over simple types and kinds. □

(i) $\Sigma \vdash M = N \in [\![\alpha]\!]$ iff $\Sigma \vdash M \Longleftrightarrow N : \alpha$.

(ii) $\Sigma \vdash M = N \in [\![\tau_1 \to \tau_2]\!]$ iff $\forall \Sigma' \succeq \Sigma$, $\Sigma' \vdash M_1 = N_1 \in [\![\tau_1]\!]$ implies $\Sigma' \vdash M\, M_1 = N\, N_1 \in [\![\tau_2]\!]$.

(iii) $\Sigma \vdash M = N \in [\![\tau_1 \multimap \tau_2]\!]$ iff $\forall \Sigma' \succeq \Sigma$, $\Sigma' \vdash M_1 = N_1 \in [\![\tau_1]\!]$ implies $\Sigma' \vdash M \hat{} M_1 = N \hat{} N_1 \in [\![\tau_2]\!]$.

(iv) $\Sigma \vdash M = N \in [\![\tau_1 \;\&\; \tau_2]\!]$ iff $\Sigma \vdash \pi_1 M = \pi_1 N \in [\![\tau_1]\!]$ and $\Sigma \vdash \pi_2 M = \pi_2 N \in [\![\tau_2]\!]$.

(v) $\Sigma \vdash M = N \in [\![\top]\!]$, always.

(vi) $\Sigma \vdash A = B \in [\![\mathsf{t}^-]\!]$ iff $\Sigma \vdash A \Longleftrightarrow B : \mathsf{t}^-$.

(vii) $\Sigma \vdash A = B \in [\![\tau \to \kappa]\!]$ iff $\forall \Sigma' \succeq \Sigma$, $\Sigma' \vdash M = N \in [\![\tau]\!]$ implies $\Sigma' \vdash A\, M = B\, N \in [\![\kappa]\!]$.

(viii) $\Sigma \vdash \sigma_1 = \sigma_2 \in [\![\epsilon]\!]$ iff $\sigma_1 = \sigma_2 = \cdot$, the empty substitution.

(ix) $\Sigma' \vdash \sigma_1[u{\mapsto}M_1] = \sigma_2[u{\mapsto}M_2] \in [\![(\Sigma, u{:}\tau)]\!]$ iff $\Sigma' \vdash \sigma_1 = \sigma_2 \in [\![\Sigma]\!]$ and $\Sigma' \vdash M_1 = M_2 \in [\![\tau]\!]$.

Fig. 11. Definition of our Kripke Logical Relation

5.2 Definitionally Equal Terms are Logically Related

It takes a little more work to prove the next part of completeness, namely that any two terms that are definitionally equal will be related by our Kripke logical relation. We must prove symmetry, transitivity and closure under head expansion before tackling the proof by induction on definitional equality derivations; due to space considerations we omit those lemmas here.

Now we can prove the main lemma of this section, namely that logically related substitutions map definitionally equal terms to logically related terms. Once we have done this we can use the fact that identity substitutions are logically related to establish completeness of the algorithm. Because the definitional equality judgments enforce linearity but the logical relations do not, the statement of the main lemma must allow the domain of the substitutions to contain variables that are not mentioned in the equality judgment.

Lemma 5.3

(i) If $\Gamma; \Delta \vdash M_1 = M_2 : A$ *and* $\Sigma \vdash \sigma_1 = \sigma_2 \in [\![\Gamma^-, \Delta^-, \Theta]\!]$ *then* $\Sigma \vdash \sigma_1 M_1 = \sigma_2 M_2 \in [\![A^-]\!]$.

(ii) If $\Gamma \vdash A_1 = A_2 : K$ *and* $\Sigma \vdash \sigma_1 = \sigma_2 \in [\![\Gamma^-, \Theta]\!]$ *then* $\Sigma \vdash \sigma_1 A_1 = \sigma_2 A_2 \in [\![K^-]\!]$.

Proof. By induction on derivations. We will show only a few of the interesting cases.

Case:

$$\frac{\Gamma;\Delta_1 \vdash M_1 = M_2 : A \multimap B \quad \Gamma;\Delta_2 \vdash N_1 = N_2 : A}{\Gamma;\Delta_1,\Delta_2 \vdash M_1\hat{}N_1 = M_2\hat{}N_2 : B}$$

First, note that $(A \multimap B)^- = A^- \multimap B^-$.
Let $\Theta_1 = \Delta_2^-, \Theta$ and $\Theta_2 = \Delta_1^-, \Theta$. Then for each i, $\Sigma \vdash \sigma_1 = \sigma_2 \in [\![\Gamma^-, \Delta_i^-, \Theta_i]\!]$.
By the i.h., $\Sigma \vdash \sigma_1 M_1 = \sigma_2 M_2 \in [\![A^- \multimap B^-]\!]$ and $\Sigma \vdash \sigma_1 N_1 = \sigma_2 N_2 \in [\![A^-]\!]$.
By definition of $[\![A^- \multimap B^-]\!]$, $\Sigma \vdash (\sigma_1 M_1)\hat{}(\sigma_1 N_1) = (\sigma_2 M_2)\hat{}(\sigma_2 N_2) \in [\![B^-]\!]$.
That is, $\Sigma \vdash \sigma_1(M_1\hat{}N_1) = \sigma_2(M_2\hat{}N_2) \in [\![B^-]\!]$.

Case:

$$\frac{\begin{array}{c}\Gamma;\Delta, u\hat{:}A \vdash M_1 = M_2 : B \\ \Gamma \vdash A_1 = A : \mathsf{Type} \quad \Gamma \vdash A_2 = A : \mathsf{Type}\end{array}}{\Gamma;\Delta \vdash \hat{\lambda}u{:}A_1.M_1 = \hat{\lambda}u{:}A_2 M_2 : A \multimap B}$$

WLOG, we may assume $u \notin \mathrm{Dom}(\Theta)$.
Note that $(A \multimap B)^- = A^- \multimap B^-$.
So, suppose $\Sigma' \succeq \Sigma$ and $\Sigma' \vdash N_1 = N_2 \in [\![A^-]\!]$.
By Lemma 5.1, $\Sigma' \vdash \sigma_1 = \sigma_2 \in [\![\Gamma^-, \Delta^-, \Theta]\!]$.
By definition of the logical relation,
$\Sigma' \vdash \sigma_1[u{\mapsto}N_1] = \sigma_2[u{\mapsto}N_2] \in [\![\Gamma^-, \Delta^-, u{:}A^-, \Theta]\!]$.
By the i.h., $\Sigma' \vdash \sigma_1[u{\mapsto}N_1]M_1 = \sigma_2[u{\mapsto}N_2]M_2 \in [\![B^-]\!]$.
That is, $\Sigma' \vdash (\sigma_1 M_1)[N_1/u] = (\sigma_2 M_2)[N_2/u] \in [\![B^-]\!]$.
By closure under head expansion,
$\Sigma' \vdash (\hat{\lambda}u{:}\sigma_1 A_1.\sigma_1 M_1)\hat{}N_1 = (\hat{\lambda}u{:}\sigma_2 A_2.\sigma_2 M_2)\hat{}N_2 \in [\![B^-]\!]$.
Thus by definition of $[\![A^- \multimap B^-]\!]$,
$\Sigma \vdash \hat{\lambda}u{:}\sigma_1 A_1.\sigma_1 M_1 = \hat{\lambda}u{:}\sigma_2 A_2.\sigma_2 M_2 \in [\![A^- \multimap B^-]\!]$.
That is, $\Sigma \vdash \sigma_1(\hat{\lambda}u{:}A_1.M_1) = \sigma_2(\hat{\lambda}u{:}A_2.M_2) \in [\![A^- \multimap B^-]\!]$. □

Lemma 5.4 $\Sigma \vdash \mathrm{id}_\Sigma = \mathrm{id}_\Sigma \in [\![\Sigma]\!]$.

Proof. By Lemma 5.2 and the definition of the logical relation for substitutions.

Theorem 5.5 (Completeness)

(i) If $\Gamma;\Delta \vdash M = N : A$ *then* $\Gamma^-, \Delta^- \vdash M \Longleftrightarrow N : A^-$.

(ii) If $\Gamma \vdash A = B : K$ *then* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$

Proof. By Lemmas 5.2, 5.3 and 5.4. □

6 Decidability of Equality

Having established soundness and completeness for our algorithmic equality judgments, we may prove that equality is decidable — in effect, that the

algorithmic rules do in fact define an algorithm . The proof is split into two parts. First, we prove that algorithmic equality is decidable when each of the terms being compared is algorithmically equal to some other term. Then, we use this fact to prove that definitional equality is decidable for all well-typed terms by noting that any well-typed term is algorithmically equal to itself.

Lemma 6.1

(i) *If* $\Sigma \vdash M \Longleftrightarrow M' : \tau$ *and* $\Sigma \vdash N \Longleftrightarrow N' : \tau$ *then it is decidable whether* $\Sigma \vdash M \Longleftrightarrow N : \tau$.

(ii) *If* $\Sigma \vdash M \longleftrightarrow M' : \tau_1$ *and* $\Sigma \vdash N \longleftrightarrow N' : \tau_2$ *then it is decidable whether* $\Sigma \vdash M \longleftrightarrow N : \tau_3$ *for some* τ_3.

(iii) *If* $\Sigma \vdash A \Longleftrightarrow A' : \kappa$ *and* $\Sigma \vdash B \Longleftrightarrow B' : \kappa$ *then it is decidable whether* $\Sigma \vdash A \Longleftrightarrow B : \kappa$.

(iv) *If* $\Sigma \vdash A \longleftrightarrow A' : \kappa_1$ *and* $\Sigma \vdash B \longleftrightarrow B' : \kappa_2$ *then it is decidable whether* $\Sigma \vdash A \longleftrightarrow B : \kappa_3$ *for some* κ_3.

(v) *If* $\Sigma \vdash K \longleftrightarrow K' : \mathsf{kind}^-$ *and* $\Sigma \vdash L \longleftrightarrow L' : \mathsf{kind}^-$ *then it is decidable whether* $\Sigma \vdash K \longleftrightarrow L : \mathsf{kind}^-$.

Proof. By induction on derivations. □

Theorem 6.2 (Decidability for Well-Formed Terms)

(i) *If* $\Gamma; \Delta \vdash M : A$ *and* $\Gamma; \Delta \vdash N : A$ *then it is decidable whether* $\Gamma; \Delta \vdash M = N : A$.

(ii) *If* $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *then it is decidable whether* $\Gamma \vdash A = B : K$.

(iii) *If* $\Gamma \vdash K : \mathsf{kind}$ *and* $\Gamma \vdash L : \mathsf{kind}$ *then it is decidable whether* $\Gamma \vdash K = L : \mathsf{kind}$.

Proof. Because algorithmic equality is sound (Theorem 4.2) and complete (Theorem 5.5), it suffices to check algorithmic equality in each case. Furthermore, by reflexivity and completeness, each term is algorithmically equal to itself. Thus by Lemma 6.1, algorithmic equality of the two terms is decidable. □

7 Conclusion

We have presented a variant of the LLF type theory in which terms need not be in pre-canonical form in order to be well-typed. Our variant differs from the original presentation of LLF by Cervesato and Pfenning by employing a set of typed definitional equality judgments rather than taking definitional equality to be untyped β- or $\beta\eta$-conversion. We have proved that this notion of definitional equality for well-typed terms is decidable by giving a type-directed algorithm and proving it sound and complete. The algorithm is simplified by

the identification of intuitionistic and linear assumptions, relying on the well-formedness of the terms being compared to ensure linearity is respected.

We have not addressed the problem of finding canonical forms for terms, or proving that they exist. However, we believe that our algorithm, like that of Harper and Pfenning on which it is based, can be instrumented to extract canonical forms for the terms it compares. In fact, a trick very similar to this instrumentation is performed implicitly in our soundness proof, where algorithmically equal terms are proved definitionally equal by extracting a mediating term. It appears that this mediating term is canonical except for the type labels on λ- (and $\hat{\lambda}$-) abstractions.

References

[1] Iliano Cervesato and Frank Pfenning. A linear logical framework. In *Eleventh IEEE Symposium on Logic in Computer Science*, pages 264–275, July 1996.

[2] 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.

[3] Karl Crary and Joseph C. Vanderwaart. An expressive, scalable type theory for certified code. Technical Report CMU-CS-01-113, Carnegie Mellon University, May 2001.

[4] Amy Felty. Encoding dependent types in intuitionistic logic. In Gérard Huet and Gordon D. Plotkin, editors, *Logical Frameworks*, pages 214–251. Cambridge University Press, 1991.

[5] 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.

[6] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. Technical Report CMU-CS-99-159, Carnegie Mellon University, September 1999.

[7] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. Technical Report CMU-CS-00-148, Carnegie Mellon University, July 2000.

[8] Frank Pfenning. Personal communication.

[9] Jeff Polakow. *Ordered Linear Logic and Applications.* PhD thesis, Carnegie Mellon University, August 2001. Available as Technical Report CMU-CS-01-152.

[10] Joseph C. Vanderwaart and Karl Crary. A simplified account of the metatheory of Linear LF. Technical report, Carnegie Mellon University, 2002.