Mechanizing the Metatheory of LF

Christian Urban
TU Munich

James Cheney
University of Edinburgh

Stefan Berghofer
TU Munich

Abstract

LF is a dependent type theory in which many other formal systems can be conveniently embedded. However, correct use of LF relies on nontrivial metatheoretic developments such as proofs of correctness of decision procedures for LF's judgments. Although detailed informal proofs of these properties have been published, they have not been formally verified in a theorem prover. We have formalized these properties within Isabelle/HOL using the Nominal Datatype Package, closely following a recent article by Harper and Pfenning. In the process, we identified and resolved a gap in one of the proofs and a small number of minor lacunae in others. Besides its intrinsic interest, our formalization provides a foundation for studying the adequacy of LF encodings, the correctness of Twelf-style metatheoretic reasoning, and the metatheory of extensions to LF.

1 Introduction

The (Edinburgh) Logical Framework (LF) was introduced by Harper, Honsell and Plotkin [6] as a framework for specifying and reasoning about formal systems. It has found many applications, such as proof-carrying code [11]. The Twelf system [14] has been used to mechanize reasoning about LF specifications.

The cornerstone of LF is the idea of encoding *judgments-as-types* and *proofs-as-terms* whereby judgments of a specified formal system are represented as LF-types and the LF-terms inhabiting these LF-types correspond to valid deductions for these judgments. Hence, the validity of a deduction in a specified system is equivalent to a type checking problem in LF. Therefore correct use of LF to encode other logics depends on the proofs of correctness of type checking algorithms for LF.

Type checking in LF is decidable, but proving decidability is nontrivial because typechecking depends on equality-tests for LF-terms and LF-types. Several algorithms for such equality-tests have been proposed in the literature [3, 5, 8]. Harper and Pfenning present in [8] a type-driven algorithm, which is practical and also has been extended to a variety of richer languages. The correctness of this algorithm is proved by establishing soundness and completeness with respect to the definitional equality rules of LF. These proofs are involved: Harper and Pfenning's detailed "pencil-and-paper" proof given in [8] spans more than 30 pages, yet still omits many cases and lemmas.

We present a formalization of the main results of Harper and Pfenning's article [8]. To our knowledge this is the first formalization of these results. We found a few of the proofs as presented in [8] do *not* go through as described, and there is a *gap* in the proof of soundness. Fixing the problem without changing the rules of the system was nontrivial. Our formalization was essential not only to find this gap in Harper and Pfenning's argument, but also to find and validate the possible repairs relatively quickly.

We used Isabelle/HOL [12] and the Nominal Datatype Package [18, 20] for our formalization. The latter provides an infrastructure for reasoning conveniently about datatypes with a built-in notion of alpha-equivalence: it allows to specify such datatypes, provides appropriate recursion combinators and derives strong induction principles that have the usual variable convention already built-in. The Nominal Datatype Package has already been used to formalize logical relation arguments similar to (but much simpler than) those in Harper and Pfenning's completeness proof [10]; logical relations proofs are currently not easy to formalize in Twelf, despite the recent breakthrough in [16].

Besides proving the correctness of their equivalence algorithm, Harper and Pfenning also sketched a proof of decidability. Unfortunately, since Isabelle/HOL is based on classical logic, proving decidability results of this kind is not straightforward. We have formalized the essential parts of the decidability proof by providing inductive definitions of the complements of the relations we wish to decide. It is clear by inspection that these relations define recursively enumerable sets, which implies decidability, but we have not formalized this part of the proof. A complete proof of decidability would require first developing a substantial amount of computability theory within Isabelle/HOL, a problem of independent interest we leave for future work.

Contributions: We present a formalization of the soundness and completeness of the equivalence algorithm and additional metatheoretic properties of LF presented in [8]. We discuss additional lemmas and other complications arising during the formalization, and explain the gap in the soundness proof and its solutions in detail. We also discuss our partial formalization of decidability. Due to space limitations, we omit detailed discussion of some of the material from sections 6 and 7 of [8]; the interested reader should consult our formalization and companion technical report [19].

2 Background

The logical framework LF [6] is a dependent type theory. We present it here following closely the article by Harper and Pfenning [8], to which we refer from now on as HP05. The syntax of LF includes *kinds*, *type families* and *objects* defined by the grammar:

$$\begin{array}{lrcl} \textit{Kinds} & K, L & ::= & \textit{type} \mid \Pi x{:}A.\ K \\ \textit{Type families} & A, B & ::= & a \mid \Pi x{:}A_1.\ A_2 \mid A\ M \\ \textit{Objects} & M, N & ::= & c \mid x \mid \lambda x{:}A.\ M \mid M_1\ M_2 \end{array}$$

where variables x and constants c and a are drawn from countably infinite, disjoint sets Var and Id of *variables* and *identifiers* respectively. We formalize the syntax of LF as nominal datatypes (i.e. α-equivalence classes) since the constructors λ and Π bind variables. Traditionally, LF has included λ-abstraction at the level of both types and objects. However, Geuvers and Barendsen [4] established that type-level λ-abstraction is superfluous in LF. Accordingly, HP05 omits type-level λ-abstraction, and so do we.

Substitutions are represented as lists of variable-term pairs and we define capture avoiding substitution in the standard way as

$$\begin{array}{rcll} x[\sigma] & = & \textit{lookup}\ \sigma\ x & \\ c[\sigma] & = & c & \\ (M\ N)[\sigma] & = & M[\sigma]\ N[\sigma] & \\ (\lambda x{:}A.\ M)[\sigma] & = & \lambda x{:}A[\sigma].\ M[\sigma] & \text{provided } x \# (\sigma, A) \\ a[\sigma] & = & a & \\ (A\ M)[\sigma] & = & A[\sigma]\ M[\sigma] & \\ (\Pi x{:}A.\ B)[\sigma] & = & \Pi x{:}A[\sigma].\ B[\sigma] & \text{provided } x \# (\sigma, A) \\ \textit{type}[\sigma] & = & \textit{type} & \\ (\Pi x{:}A.\ K)[\sigma] & = & \Pi x{:}A[\sigma].\ K[\sigma] & \text{provided } x \# (\sigma, A) \end{array}$$

where the variable case is defined in terms of the auxiliary function *lookup*:

$$\begin{array}{l} \textit{lookup}\ []\ x = x \\ \textit{lookup}\ ((y, M)::\sigma)\ x = (\textit{if } x = y \textit{ then } M \textit{ else lookup } \sigma\ x) \end{array}$$

The preconditions $x \# (\sigma, A)$ in the above definition are freshness constraints provided automatically by the Nominal Datatype Package and stand for x not occurring freely in the list σ and in type family A. Substitution for a single variable is defined as a special case: $(\_)[x{:=}M] \stackrel{\text{def}}{=} (\text{-})[(x,M)]$.

LF also includes *signatures* Σ and *contexts* Γ, both of which we represent as lists of pairs. The former consist of pairs of the form (c, A) or (a, K) associating the constant c with type A and the constant a with kind K respectively, and the latter consists of pairs (x, A) associating the variable x with type A. Accordingly, we write $(x, A)::\Gamma$ for list construction (rather than $\Gamma, x{:}A$), $\Gamma @ \Gamma'$ for list concatenation and $(x, A) \in \Gamma$ for list membership (similarly for Σ).

HP05 defines two judgments for identifying valid signatures and contexts, which we formalize as follows

$$\boxed{\vdash \Sigma\ sig} \qquad \frac{}{\vdash []\ sig} \qquad \frac{\vdash \Sigma\ sig \quad [] \vdash_\Sigma K : kind \quad a \# \Sigma}{\vdash (a, K)::\Sigma\ sig} \qquad \frac{\vdash \Sigma\ sig \quad [] \vdash_\Sigma A : type \quad c \# \Sigma}{\vdash (c, A)::\Sigma\ sig}$$

$$\boxed{\vdash_\Sigma \Gamma\ ctx} \qquad \frac{\vdash \Sigma\ sig}{\vdash_\Sigma []\ ctx} \qquad \frac{\vdash_\Sigma \Gamma\ ctx \quad \Gamma \vdash_\Sigma A : type \quad x \# \Gamma}{\vdash_\Sigma (x, A)::\Gamma\ ctx}$$

where $[]$ stands for the empty list. In contrast with HP05, we make explicit that the new bindings do not occur previously in Σ or Γ, using freshness constraints such as $x \# \Gamma$. We also leave explicit the dependence of all judgments on Σ.

Central in HP05 are the definitions of the validity and definitional equivalence judgments for LF, and of algorithmic judgments for checking equivalence. The validity and definitional equivalence rules are shown in Fig. 1. Note that there are three judgments for validity and three for equivalence, corresponding to objects, type families and kinds respectively. These six judgments are defined simultaneously with signature and context validity by induction. We added explicit validity hypotheses to some of the rules; these are left implicit in HP05. We also added some (redundant) freshness constraints to some rules in order to be able to use "strong" induction principles [18].

The rules for the equivalence checking algorithm are given in Fig. 2. There are five algorithmic judgments: algorithmic and structural object equivalence, algorithmic and structural type equivalence, and algorithmic kind equivalence. Note that the algorithmic rules are type- (or kind-) directed while the structural rules are syntax-directed. The algorithmic rules make use of several additional notations which we define next.

A crucial point of the algorithm in HP05 is that it does not analyze the precise types or kinds of objects or types respectively, rather it only uses approximate *simple types* τ and *simple kinds* κ defined as follows:

$$\tau ::= a^- \mid \tau \to \tau' \qquad \kappa ::= \textit{type}^- \mid \tau \to \kappa$$

This simplification is sufficient for obtaining a sound and complete equivalence checking algorithm, and crucially also simplifies the proof development in a number of places.

Similarly, *simple contexts* Δ consist of lists of pairs (x, τ) of variables and simple types. We write $\vdash \Delta\ sctx$ to indicate that Δ is valid, i.e. has no repeated variables, and write $\Delta \geq \Delta'$ to indicate that Δ' contains all of the bindings of Δ and Δ is a valid simple context.

The *erasure* function translates families and kinds to simple types and simple kinds:

$$(a)^- = a^- \qquad (type)^- = type^-$$
$$(A\ M)^- = A^- \qquad (\Pi x{:}A.\ K)^- = A^- \to K^-$$
$$(\Pi x{:}A_1.\ A_2)^- = A_1{}^- \to A_2{}^-$$

Similarly, we write Γ^- for the simple context resulting from replacing each binding (x, A) in Γ with (x, A^-).

The rules for the algorithm also employ a *weak head reduction* relation $\rightsquigarrow$ which performs beta-reductions only at the head of the top-level application of a term. It is defined as

$$\frac{x \# (A_1, M_1)}{(\lambda x{:}A_1.\ M_2)\ M_1 \rightsquigarrow M_2[x{:=}M_1]} \qquad \frac{M_1 \rightsquigarrow M_1{}'}{M_1\ M_2 \rightsquigarrow M_1{}'\ M_2}$$

The main results of HP05 are soundness and completeness of the algorithmic judgments relative to the equivalence judgments, namely

Theorem 1 (Completeness)**.**

1. *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\Gamma^- \vdash_\Sigma M \Leftrightarrow N : A^-$.
2. *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\Gamma^- \vdash_\Sigma A \Leftrightarrow B : K^-$.
3. *If* $\Gamma \vdash_\Sigma K = L : kind$ *then* $\Gamma^- \vdash_\Sigma K \Leftrightarrow L : kind^-$.

Theorem 2 (Soundness)**.**

1. *If* $\Gamma^- \vdash_\Sigma M \Leftrightarrow N : A^-$ *and* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma \vdash_\Sigma N : A$ *then* $\Gamma \vdash_\Sigma M = N : A$.
2. *If* $\Gamma^- \vdash_\Sigma A \Leftrightarrow B : K^-$ *and* $\Gamma \vdash_\Sigma A : K$ *and* $\Gamma \vdash_\Sigma B : K$ *then* $\Gamma \vdash_\Sigma A = B : K$.
3. *If* $\Gamma^- \vdash_\Sigma K \Leftrightarrow L : kind^-$ *and* $\Gamma \vdash_\Sigma K : kind$ *and* $\Gamma \vdash_\Sigma L : kind$ *then* $\Gamma \vdash_\Sigma K = L : kind$.

In what follows, we outline the proofs of these results and discuss how we have formalized them, paying particular attention to places where additional lemmas or different proof techniques were needed. We also discuss the gap in the soundness proof of HP05, along with several solutions.

3 The formalization

The proof in HP05 starts by establishing a number of useful metatheoretic properties for the validity and equality judgments (shown in Fig. 1), such as weakening, substitution, generalizations of the conversion rules and inversion principles. We needed two technical lemmas having to do with the implicit freshness and validity assumptions which must be explicitly handled in our formalization. Both are straightforward by induction, and both are needed frequently. For example for the validity judgments for objects we have:

Lemma 1 (Freshness)**.** *If* $x \# \Gamma$ *and* $\Gamma \vdash_\Sigma M : A$ *then* $x \# M$ *and* $x \# A$.

Lemma 2 (Implicit Validity)**.** *If* $\Gamma \vdash_\Sigma M : A$ *then* $\vdash \Sigma\ sig$ *and* $\vdash_\Sigma \Gamma\ ctx$.

Similarly for the other validity and definitional equivalence judgments.

All the metatheoretic properties can be proved as stated in the article (appealing to Lem. 1 and 2 as necessary); however, since all of the judgments of LF are interdependent, each inductive proof must consider all 35 cases, making each proof nontrivial as a practical matter (it is one of the biggest parts of our formalization).

HP05 organize the proofs of these metatheoretic properties very neatly. For example one can show that the validity judgment of terms implies the validity of the type, namely

Lemma 3 (Validity)**.** *Types and kinds appearing in derivable judgments are valid. For example, if* $\Gamma \vdash_\Sigma M : A$ *then* $\Gamma \vdash_\Sigma A : type$.

However, in order to establish this a number of auxiliary facts have to be proved first which depend on this property. In order to get the proof through, HP05 defined the rules given in Fig. 1 to explicitly check the validity of type and kind subterms $\Gamma \vdash_\Sigma A : type$ and $\Gamma \vdash_\Sigma K : kind$. In many cases, these checks are unnecessary once validity has been proved.

3.1. Algorithmic equivalence

The main metatheoretic properties of algorithmic equivalence are symmetry and transitivity. Several properties of weak head reduction and erasure needed later in HP05 are also proved. Most of the proofs were straightforward to formalize, given the details in HP05 (where provided). However, there were a few missing lemmas and other complications. The algorithmic system is less well-behaved than the definitional system because derivable judgments may have "ill-formed" arguments; for example, the judgment $[] \vdash_\Sigma (\lambda x{:}a.\ c)\ y \Leftrightarrow c : b^-$ is derivable provided $(c, b) \in \Sigma$ since $(\lambda x{:}a.\ c)\ y \rightsquigarrow c$. Thus, analogues of Lem. 1 and 2 do not hold for the algorithmic system, and in rules involving binding we need to impose additional freshness constraints. Moreover, proof search in the algorithmic system is not necessarily terminating because $\rightsquigarrow$ may diverge if called on ill-formed terms such as $(\lambda x{:}a.\ x\ x)\ (\lambda x{:}a.\ x\ x)$.

The erasure preservation lemma establishes basic properties of erasure which are frequently needed in HP05:

Lemma 4 (Erasure preservation)**.**

1. *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $A^- = B^-$.
2. *If* $\Gamma \vdash_\Sigma K = L : kind$ *then* $K^- = L^-$.
3. *If* $(x, A)::\Gamma \vdash_\Sigma B : type$ *then* $B^- = B[x{:=}M]^-$

$\boxed{\Gamma \vdash_\Sigma M : A}$

$$\frac{\vdash_\Sigma \Gamma\ ctx \quad (x, A) \in \Gamma}{\Gamma \vdash_\Sigma x : A} \qquad \frac{\vdash_\Sigma \Gamma\ ctx \quad (c, A) \in \Sigma}{\Gamma \vdash_\Sigma c : A} \qquad \frac{\Gamma \vdash_\Sigma M_1 : \Pi x{:}A_2.\, A_1 \quad \Gamma \vdash_\Sigma M_2 : A_2 \quad x \# \Gamma}{\Gamma \vdash_\Sigma M_1\ M_2 : A_1[x{:=}M_2]}$$

$$\frac{\Gamma \vdash_\Sigma A_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma M_2 : A_2 \quad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \lambda x{:}A_1.\, M_2 : \Pi x{:}A_1.\, A_2} \qquad \frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma A = B : type}{\Gamma \vdash_\Sigma M : B}$$

$\boxed{\Gamma \vdash_\Sigma A : K}$

$$\frac{\vdash_\Sigma \Gamma\ ctx \quad (a, K) \in \Sigma}{\Gamma \vdash_\Sigma a : K} \qquad \frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.\, K \quad \Gamma \vdash_\Sigma M : B \quad x \# \Gamma}{\Gamma \vdash_\Sigma A\ M : K[x{:=}M]}$$

$$\frac{\Gamma \vdash_\Sigma A_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma A_2 : type \quad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \Pi x{:}A_1.\, A_2 : type} \qquad \frac{\Gamma \vdash_\Sigma A : K \quad \Gamma \vdash_\Sigma K = L : kind}{\Gamma \vdash_\Sigma A : L}$$

$\boxed{\Gamma \vdash_\Sigma K : kind}$

$$\frac{\vdash_\Sigma \Gamma\ ctx}{\Gamma \vdash_\Sigma type : kind} \qquad \frac{\Gamma \vdash_\Sigma A : type \quad (x, A){::}\Gamma \vdash_\Sigma K : kind \quad x \# (\Gamma, A)}{\Gamma \vdash_\Sigma \Pi x{:}A.\, K : kind}$$

$\boxed{\Gamma \vdash_\Sigma M = N : A}$

$$\frac{\vdash_\Sigma \Gamma\ ctx \quad (x, A) \in \Gamma}{\Gamma \vdash_\Sigma x = x : A} \qquad \frac{\vdash_\Sigma \Gamma\ ctx \quad (c, A) \in \Sigma}{\Gamma \vdash_\Sigma c = c : A}$$

$$\frac{\Gamma \vdash_\Sigma M_1 = N_1 : \Pi x{:}A_2.\, A_1 \quad \Gamma \vdash_\Sigma M_2 = N_2 : A_2 \quad x \# \Gamma}{\Gamma \vdash_\Sigma M_1\ M_2 = N_1\ N_2 : A_1[x{:=}M_2]} \qquad \frac{\Gamma \vdash_\Sigma A_1{}' = A_1 : type \quad \Gamma \vdash_\Sigma A_1{}'' = A_1 : type \quad \Gamma \vdash_\Sigma A_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma M_2 = N_2 : A_2 \quad x \# \Gamma}{\Gamma \vdash_\Sigma \lambda x{:}A_1{}'.\, M_2 = \lambda x{:}A_1{}''.\, N_2 : \Pi x{:}A_1.\, A_2}$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\, A_2 \quad \Gamma \vdash_\Sigma N : \Pi x{:}A_1.\, A_2 \quad \Gamma \vdash_\Sigma A_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma M\, x = N\, x : A_2 \quad x \# \Gamma}{\Gamma \vdash_\Sigma M = N : \Pi x{:}A_1.\, A_2} \qquad \frac{\Gamma \vdash_\Sigma A_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma M_2 = N_2 : A_2 \quad \Gamma \vdash_\Sigma M_1 = N_1 : A_1 \quad x \# \Gamma}{\Gamma \vdash_\Sigma (\lambda x{:}A_1.\, M_2)\ M_1 = N_2[x{:=}N_1] : A_2[x{:=}M_1]}$$

$$\frac{\Gamma \vdash_\Sigma M = N : A}{\Gamma \vdash_\Sigma N = M : A} \qquad \frac{\Gamma \vdash_\Sigma M = N : A \quad \Gamma \vdash_\Sigma N = P : A}{\Gamma \vdash_\Sigma M = P : A} \qquad \frac{\Gamma \vdash_\Sigma M = N : A \quad \Gamma \vdash_\Sigma A = B : type}{\Gamma \vdash_\Sigma M = N : B}$$

$\boxed{\Gamma \vdash_\Sigma A = B : K}$

$$\frac{\vdash_\Sigma \Gamma\ ctx \quad (a, K) \in \Sigma}{\Gamma \vdash_\Sigma a = a : K} \qquad \frac{\Gamma \vdash_\Sigma A = B : \Pi x{:}C.\, K \quad \Gamma \vdash_\Sigma M = N : C \quad x \# \Gamma}{\Gamma \vdash_\Sigma A\ M = B\ N : K[x{:=}M]} \qquad \frac{\Gamma \vdash_\Sigma A_1 = B_1 : type \quad \Gamma \vdash_\Sigma A_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2 : type \quad x \# \Gamma}{\Gamma \vdash_\Sigma \Pi x{:}A_1.\, A_2 = \Pi x{:}B_1.\, B_2 : type}$$

$$\frac{\Gamma \vdash_\Sigma A = B : K}{\Gamma \vdash_\Sigma B = A : K} \qquad \frac{\Gamma \vdash_\Sigma A = B : K \quad \Gamma \vdash_\Sigma B = C : K}{\Gamma \vdash_\Sigma A = C : K} \qquad \frac{\Gamma \vdash_\Sigma A = B : K \quad \Gamma \vdash_\Sigma K = L : kind}{\Gamma \vdash_\Sigma A = B : L}$$

$\boxed{\Gamma \vdash_\Sigma K = L : kind}$

$$\frac{\vdash_\Sigma \Gamma\ ctx}{\Gamma \vdash_\Sigma type = type : kind} \qquad \frac{\Gamma \vdash_\Sigma A = B : type \quad \Gamma \vdash_\Sigma A : type \quad (x, A){::}\Gamma \vdash_\Sigma K = L : kind \quad x \# \Gamma}{\Gamma \vdash_\Sigma \Pi x{:}A.\, K = \Pi x{:}B.\, L : kind}$$

$$\frac{\Gamma \vdash_\Sigma K = L : kind}{\Gamma \vdash_\Sigma L = K : kind} \qquad \frac{\Gamma \vdash_\Sigma K = L : kind \quad \Gamma \vdash_\Sigma L = L' : kind}{\Gamma \vdash_\Sigma K = L' : kind}$$

Figure 1. Validity and definitional equivalence rules for kinds, type families and objects.

$\boxed{\Delta \vdash_\Sigma M \Leftrightarrow N : \tau}$

$$\frac{M \rightsquigarrow M' \quad \Delta \vdash_\Sigma M' \Leftrightarrow N : a^-}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^-} \qquad \frac{N \rightsquigarrow N' \quad \Delta \vdash_\Sigma M \Leftrightarrow N' : a^-}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^-}$$

$$\frac{\Delta \vdash_\Sigma M \leftrightarrow N : a^-}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^-} \qquad \frac{(x, \tau_1){::}\Delta \vdash_\Sigma M\, x \Leftrightarrow N\, x : \tau_2 \quad x \# (\Sigma, \Delta, M, N)}{\Delta \vdash_\Sigma M \Leftrightarrow N : \tau_1 \to \tau_2}$$

$\boxed{\Delta \vdash_\Sigma M \leftrightarrow N : \tau}$

$$\frac{(x, \tau) \in \Delta}{\Delta \vdash_\Sigma x \leftrightarrow x : \tau} \qquad \frac{(c, A) \in \Sigma}{\Delta \vdash_\Sigma c \leftrightarrow c : A^-} \qquad \frac{\Delta \vdash_\Sigma M_1 \leftrightarrow N_1 : \tau_2 \to \tau_1 \quad \Delta \vdash_\Sigma M_2 \Leftrightarrow N_2 : \tau_2}{\Delta \vdash_\Sigma M_1\ M_2 \leftrightarrow N_1\ N_2 : \tau_1}$$

$\boxed{\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa}$

$$\frac{\Delta \vdash_\Sigma A \leftrightarrow B : type^-}{\Delta \vdash_\Sigma A \Leftrightarrow B : type^-} \qquad \frac{(x, \tau){::}\Delta \vdash_\Sigma A\, x \Leftrightarrow B\, x : \kappa \quad x \# (\Sigma, \Delta, A, B)}{\Delta \vdash_\Sigma A \Leftrightarrow B : \tau \to \kappa}$$

$$\frac{\Delta \vdash_\Sigma A_1 \Leftrightarrow B_1 : type^- \quad (x, A_1{}^-){::}\Delta \vdash_\Sigma A_2 \Leftrightarrow B_2 : type^- \quad x \# (\Sigma, \Delta, A_1, B_1)}{\Delta \vdash_\Sigma \Pi x{:}A_1.\, A_2 \Leftrightarrow \Pi x{:}B_1.\, B_2 : type^-}$$

$\boxed{\Delta \vdash_\Sigma A \leftrightarrow B : \kappa}$

$$\frac{(a, K) \in \Sigma}{\Delta \vdash_\Sigma a \leftrightarrow a : K^-} \qquad \frac{\Delta \vdash_\Sigma A \leftrightarrow B : \tau \to \kappa \quad \Delta \vdash_\Sigma M \Leftrightarrow N : \tau}{\Delta \vdash_\Sigma A\, M \leftrightarrow B\, N : \kappa}$$

$\boxed{\Delta \vdash_\Sigma K \Leftrightarrow L : kind^-}$

$$\frac{}{\Delta \vdash_\Sigma type \Leftrightarrow type : kind^-} \qquad \frac{\Delta \vdash_\Sigma A \Leftrightarrow B : type^- \quad (x, A^-){::}\Delta \vdash_\Sigma K \Leftrightarrow L : kind^- \quad x \# (\Sigma, \Delta, A, B)}{\Delta \vdash_\Sigma \Pi x{:}A.\, K \Leftrightarrow \Pi x{:}B.\, L : kind^-}$$

Figure 2. Algorithmic equivalence rules

4. *If* $(x, A)::\Gamma \vdash_\Sigma K : \mathit{kind}$ *then* $K^- = K[x:=M]^-$

However, we found that the hypotheses of parts 3 and 4 are unnecessarily strong. Indeed, we can easily prove:

Lemma 5 (Erasure cancels substitution)**.**
1. $A[x:=N]^- = A^-$ *and* $A[\sigma]^- = A^-$
2. $K[x:=N]^- = K^-$ *and* $K[\sigma]^- = K^-$

We also needed the following algorithmic erasure preservation lemma (omitted from HP05):

Lemma 6 (Alg. erasure preservation)**.**
1. *If* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *then* $A^- = B^-$.
2. *If* $\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$ *then* $A^- = B^-$.
3. *If* $\Delta \vdash_\Sigma K \Leftrightarrow L : \mathit{kind}^-$ *then* $K^- = L^-$.

The determinacy lemma establishes several important properties of weak head reduction and algorithmic equivalence.

Lemma 7 (Determinacy)**.** *Suppose that* $\vdash \Sigma$ *sig and* $\vdash \Delta$ *sctx.*
1. *If* $M \rightsquigarrow M'$ *and* $M \rightsquigarrow M''$ *then* $M' = M''$.
2. *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\nexists M'.\ M \rightsquigarrow M'$.
3. *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\nexists N'.\ N \rightsquigarrow N'$.
4. *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *and* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau'$ *then* $\tau = \tau'$.
5. *If* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *and* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa'$ *then* $\kappa = \kappa'$.

However, we needed generalized forms of 4 and 5 in the proof of transitivity (Thm. 4). It is also later used in Thm. 12 in proving decidability of the algorithmic rules.

Lemma 8 (Generalised determinacy)**.** *Suppose that* $\vdash \Sigma$ *sig and* $\vdash \Delta$ *sctx.*
1. *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *and* $\Delta \vdash_\Sigma N \Leftrightarrow P : \tau'$ *then* $\tau = \tau'$.
2. *If* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *and* $\Delta \vdash_\Sigma B \Leftrightarrow C : \kappa'$ *then* $\kappa = \kappa'$.

Verifying symmetry of the algorithmic judgments is then straightforward, using properties established so far.

Theorem 3 (Symmetry of algorithmic equivalence)**.**
1. *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *then* $\Delta \vdash_\Sigma N \Leftrightarrow M : \tau$.
2. *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\Delta \vdash_\Sigma N \leftrightarrow M : \tau$.
3. *If* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *then* $\Delta \vdash_\Sigma B \Leftrightarrow A : \kappa$.
4. *If* $\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$ *then* $\Delta \vdash_\Sigma B \leftrightarrow A : \kappa$.
5. *If* $\Delta \vdash_\Sigma K \Leftrightarrow L : \mathit{kind}^-$ *then* $\Delta \vdash_\Sigma L \Leftrightarrow K : \mathit{kind}^-$.

However, verifying transitivity required more work.

Theorem 4 (Transitivity of algorithmic equivalence)**.** *Suppose that* $\vdash \Sigma$ *sig and* $\vdash \Delta$ *sctx.*
1. *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *and* $\Delta \vdash_\Sigma N \Leftrightarrow P : \tau$ *then* $\Delta \vdash_\Sigma M \Leftrightarrow P : \tau$.
2. *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *and* $\Delta \vdash_\Sigma N \leftrightarrow P : \tau$ *then* $\Delta \vdash_\Sigma M \leftrightarrow P : \tau$.
3. *If* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *and* $\Delta \vdash_\Sigma B \Leftrightarrow C : \kappa$ *then* $\Delta \vdash_\Sigma A \Leftrightarrow C : \kappa$.
4. *If* $\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$ *and* $\Delta \vdash_\Sigma B \leftrightarrow C : \kappa$ *then* $\Delta \vdash_\Sigma A \leftrightarrow C : \kappa$.
5. *If* $\Delta \vdash_\Sigma K \Leftrightarrow L : \mathit{kind}^-$ *and* $\Delta \vdash_\Sigma L \Leftrightarrow L' : \mathit{kind}^-$ *then* $\Delta \vdash_\Sigma K \Leftrightarrow L' : \mathit{kind}^-$.

Proof. As described in HP05, the proof is by simultaneous induction on the two derivations. For types and kinds, this simultaneous induction can be avoided by performing induction over one derivation and using inversion principles. For the object-level judgments (cases 1 and 2), we formalize this argument in Isabelle by defining object-level algorithmic judgments "instrumented" with a height argument, and prove parts 1 and 2 by well-founded induction on the sum of the heights of the derivations.

Because of the induction over the height, there are several cases where we need to perform some explicit α-conversion and renaming steps; these are places in an informal proof where one usually appeals to renaming principles "without loss of generality". The generalized determinacy property (Lem. 8) is needed here in the case of structural equivalence of applications. □

Strengthening At this point in the development, we can also prove that the algorithmic judgments satisfy *strengthening*; that is, unused variables can be removed from the context without harming derivability of a conclusion. Strengthening is not discussed in HP05 until later in the paper, but we found it necessary in repairing the proof of soundness. We first need an (easily established) freshness-preservation property of weak head reduction.

Lemma 9 (Weak head reduction preserves freshness)**.** *If* $M \rightsquigarrow N$ *and* $x \# M$ *then* $x \# N$.

Lemma 10 (Strengthening of algorithmic equivalence)**.**
1. *If* $\Delta'@[(x, \tau')]@\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *and* $x \# (\Delta', M, N)$ *then* $\Delta'@\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$.
2. *If* $\Delta'@[(x, \tau')]@\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *and* $x \# (\Delta', M, N)$ *then* $\Delta'@\Delta \vdash_\Sigma M \leftrightarrow N : \tau$.
3. *If* $\Delta'@[(x, \tau)]@\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *and* $x \# (\Delta', A, B)$ *then* $\Delta'@\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$.
4. *If* $\Delta'@[(x, \tau)]@\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$ *and* $x \# (\Delta', A, B)$ *then* $\Delta'@\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$.
5. *If* $\Delta'@[(x, \tau)]@\Delta \vdash_\Sigma K \Leftrightarrow L : \mathit{kind}^-$ *and* $x \# (\Delta', K, L)$ *then* $\Delta'@\Delta \vdash_\Sigma K \Leftrightarrow L : \mathit{kind}^-$.

Proof. Straightforward induction on derivations, using properties of freshness and Lem. 9. □

3.2. Completeness

The proof of completeness involves a Kripke-style logical relations argument. We can define the logical relation for objects, types, and substitutions, by induction on the structure of simple types τ and kinds κ and simple contexts Θ, respectively, as shown in Fig. 3.

$$
\begin{array}{rcl}
\Delta \vdash_\Sigma M = N \in [\![a^-]\!] & = & \Delta \vdash_\Sigma M \Leftrightarrow N : a^- \\
\Delta \vdash_\Sigma M = N \in [\![\tau \to \tau']\!] & = & \forall \Delta' \geq \Delta, M', N'.\ \Delta' \vdash_\Sigma M' \Leftrightarrow N' : \tau \text{ implies } \Delta' \vdash_\Sigma M\,M' \Leftrightarrow N\,N' : \tau' \\
\Delta \vdash_\Sigma A = B \in [\![type^-]\!] & = & \Delta \vdash_\Sigma A \Leftrightarrow B : type^- \\
\Delta \vdash_\Sigma A = B \in [\![\tau \to \kappa]\!] & = & \forall \Delta' \geq \Delta, M', N'.\ \Delta' \vdash_\Sigma M' \Leftrightarrow N' : \tau \text{ implies } \Delta' \vdash_\Sigma A\,M' \Leftrightarrow B\,N' : \tau' \\
\Delta \vdash_\Sigma K = L \in [\![kind^-]\!] & = & \Delta \vdash_\Sigma K \Leftrightarrow L : kind^- \\
\Delta \vdash_\Sigma [] = [] \in [\![[]]\!] & = & True \\
\Delta \vdash_\Sigma (x, M)::\sigma = (x, N)::\theta \in [\![(x, \tau)::\Theta]\!] & = & \Delta \vdash_\Sigma \sigma = \theta \in [\![\Theta]\!] \text{ and } x \# \Theta \text{ and } \Delta \vdash_\Sigma M = N \in [\![\tau]\!]
\end{array}
$$

Figure 3. Logical relation definition

The key steps in proving completeness are showing that logically related terms are algorithmically equivalent (Thm. 5) and that definitionally equivalent terms are logically related (Thm. 6). Many properties can be established by an induction on the structure of types, appealing to the properties of the algorithmic judgments established in section 3 of HP05 and the definition of the logical relation.

Lemma 11 (Log. rel. weakening)**.** *Suppose* $\Delta' \geq \Delta$.
1. *If* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$ *then* $\Delta' \vdash_\Sigma M = N \in [\![\tau]\!]$.
2. *If* $\Delta \vdash_\Sigma A = B \in [\![\kappa]\!]$ *then* $\Delta' \vdash_\Sigma A = B \in [\![\kappa]\!]$.
3. *If* $\Delta \vdash_\Sigma \sigma = \theta \in [\![\Theta]\!]$ *then* $\Delta' \vdash_\Sigma \sigma = \theta \in [\![\Theta]\!]$.

Theorem 5 (Log. rel. implies alg. equiv.)**.**
Suppose $\vdash \Delta$ *sctx.*
1. *If* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$.
2. *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$.
3. *If* $\Delta \vdash_\Sigma A = B \in [\![\kappa]\!]$ *then* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$.
4. *If* $\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$ *then* $\Delta \vdash_\Sigma A = B \in [\![\kappa]\!]$.

Lemma 12 (Closure under head expansion)**.**
Suppose $M \rightsquigarrow M'$ *and* $N \rightsquigarrow N'$.
1. *If* $\Delta \vdash_\Sigma M' = N \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$.
2. *If* $\Delta \vdash_\Sigma M = N' \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$.

Lemma 13 (Log. rel. symmetry)**.**
1. *If* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma N = M \in [\![\tau]\!]$.
2. *If* $\Delta \vdash_\Sigma A = B \in [\![\kappa]\!]$ *then* $\Delta \vdash_\Sigma B = A \in [\![\kappa]\!]$.
3. *If* $\Delta \vdash_\Sigma \sigma = \theta \in [\![\Theta]\!]$ *then* $\Delta \vdash_\Sigma \theta = \sigma \in [\![\Theta]\!]$.

Lemma 14 (Log. rel. transitivity)**.** *Suppose that* $\vdash \Sigma$ *sig and* $\vdash \Delta$ *sctx.*
1. *If* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$ *and* $\Delta \vdash_\Sigma N = P \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma M = P \in [\![\tau]\!]$.
2. *If* $\Delta \vdash_\Sigma A = B \in [\![\kappa]\!]$ *and* $\Delta \vdash_\Sigma B = C \in [\![\kappa]\!]$ *then* $\Delta \vdash_\Sigma A = C \in [\![\kappa]\!]$.
3. *If* $\Delta \vdash_\Sigma \sigma = \theta \in [\![\Theta]\!]$ *and* $\Delta \vdash_\Sigma \theta = \delta \in [\![\Theta]\!]$ *then* $\Delta \vdash_\Sigma \sigma = \delta \in [\![\Theta]\!]$.

The proof that definitionally equal terms are logically related required some care to formalize. The key step is showing that applying logically related substitutions to definitionally equal terms yields logically related terms.

Lemma 15. *Suppose* $\vdash \Delta$ *sctx and* $\Delta \vdash_\Sigma \sigma = \theta \in [\![\Gamma^-]\!]$.
1. *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\Delta \vdash_\Sigma M[\sigma] = N[\theta] \in [\![A^-]\!]$.
2. *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\Delta \vdash_\Sigma A[\sigma] = B[\theta] \in [\![K^-]\!]$.

The last step needed to establish completeness is to show that the identity substitution over a given context (written id_Γ) is related to itself:

Lemma 16. *If* $\vdash_\Sigma \Gamma$ *ctx then* $\Gamma^- \vdash_\Sigma id_\Gamma = id_\Gamma \in [\![\Gamma^-]\!]$.

Theorem 6 (Def. equal implies log. rel.)**.**
1. *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\Gamma^- \vdash_\Sigma M = N \in [\![A^-]\!]$.
2. *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\Gamma^- \vdash_\Sigma A = B \in [\![K^-]\!]$.

Corollary 1 (Completeness)**.**
1. *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\Gamma^- \vdash_\Sigma M \Leftrightarrow N : A^-$.
2. *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\Gamma^- \vdash_\Sigma A \Leftrightarrow B : K^-$.
3. *If* $\Gamma \vdash_\Sigma K = L : kind$ *then* $\Gamma^- \vdash_\Sigma K \Leftrightarrow L : kind^-$.

Note that part 3 of Cor. 1 was omitted from HP05, but it is straightforward to prove by induction given parts 1 and 2, and algorithmic transitivity and symmetry.

3.3. Soundness

Soundness of the algorithmic equivalence definition is proved under the assumption that the terms being compared are well-formed. This first requires showing that weak head reduction preserves well-formedness:

Lemma 17 (Subject reduction)**.** *Suppose* $M \rightsquigarrow M'$ *and* $\Gamma \vdash_\Sigma M : A$. *Then* $\Gamma \vdash_\Sigma M' : A$ *and* $\Gamma \vdash_\Sigma M = M' : A$.

The soundness theorem then states that if the arguments to a derivable algorithmic judgment are well-formed, then the corresponding definitional judgment holds; it however needs to be stated slightly more generally than Thm. 2. In contrast to completeness, the proof of soundness proceeds by entirely syntactic techniques, by induction over the structure of algorithmic and structural derivations. Our initial formalization attempt followed the proofs given by HP05. However, we encountered two difficulties which were not discussed in the article. Both difficulties arise in the algorithmic extensionality cases in parts 1 and 3 of Thm. 2.

First problem In proving the soundness of algorithmic extensionality for objects arising in part 1 of Thm. 2, recall that we have a derivation of the form:

$\boxed{\Delta \vdash_\Sigma A \rightleftharpoons B : \kappa}$

$$\frac{(a, K) \in \Sigma}{\Delta \vdash_\Sigma a \rightleftharpoons a : K^-} \qquad \frac{\Delta \vdash_\Sigma A \rightleftharpoons B : \tau \to \kappa \quad \Delta \vdash_\Sigma M \Leftrightarrow N : \tau}{\Delta \vdash_\Sigma A\, M \rightleftharpoons B\, N : \kappa}$$

$$\frac{\Delta \vdash_\Sigma A_1 \rightleftharpoons B_1 : \mathit{type}^- \quad (x, {A_1}^-)::\Delta \vdash_\Sigma A_2 \rightleftharpoons B_2 : \mathit{type}^- \quad x \# (\Sigma, \Delta, A_1, B_1)}{\Delta \vdash_\Sigma \Pi x{:}A_1.\, A_2 \rightleftharpoons \Pi x{:}B_1.\, B_2 : \mathit{type}^-}$$

Figure 4. Weak algorithmic type equivalence judgment

$$\frac{(x, \tau_1)::\Gamma^- \vdash_\Sigma M\, x \Leftrightarrow N\, x : \tau_2 \quad x \# (\Sigma, \Gamma^-, M, N)}{\Gamma^- \vdash_\Sigma M \Leftrightarrow N : \tau_1 \to \tau_2}$$

and we also know that $\Gamma \vdash_\Sigma M : A$ and $\Gamma \vdash_\Sigma N : A$ for some A with $A^- = \tau_1 \to \tau_2$. In order to apply the induction hypothesis, we need to know that $M\, x$ and $N\, x$ are well-formed in an extended context $(x, A_1)::\Gamma$. HP05's proof begins by assuming that $\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\, A_2$ and $\Gamma \vdash_\Sigma N : \Pi x{:}A_1.\, A_2$, and proceeding using inversion properties. However, it is *not* immediately clear that $A^- = \tau_1 \to \tau_2$ implies that $A = \Pi x{:}A_1.\, A_2$ for some A_1 and A_2; indeed, this can fail to be the case if A is not well-formed. Instead, we first need the following inversion principles for erasure:

Lemma 18 (Erasure inversion)**.**

1. *If* $\Gamma \vdash_\Sigma A : \Pi x{:}B.\, K$ *then* $\exists\, c.\, A^- = c^-$.
2. *If* $\tau_1 \to \tau_2 = A^-$ *and* $\Gamma \vdash_\Sigma A : \mathit{type}$ *and* $x \# A$ *then* $\exists\, A_1\, A_2.\, A = \Pi x{:}A_1.\, A_2$.
3. *If* $\tau \to \kappa = K^-$ *and* $x \# K$ *then* $\exists\, A\, L.\, K = \Pi x{:}A.\, L$.

Proof. Part 1 follows by induction on the derivation. Parts 2 and 3 follow by induction on the structure of A and K respectively. In the case for type applications $A\, M$, clearly A has a Π-kind, but by part 1, A erases to a constant, contradicting the assumption that $A^- = \tau_1 \to \tau_2$. So the case is vacuous. The remaining cases of part 2 are straightforward, as are the cases for part 3. □

Using Lem. 18, we can complete the proof of the first part of Thm. 2 as described in HP05:

Lemma 19 (Soundness of algorithmic object equivalence)**.**

1. *If* $\Gamma^- \vdash_\Sigma M \Leftrightarrow N : A^-$ *and* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma \vdash_\Sigma N : A$ *then* $\Gamma \vdash_\Sigma M = N : A$.
2. *If* $\Gamma^- \vdash_\Sigma M \leftrightarrow N : \tau$ *and* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma \vdash_\Sigma N : B$ *then* $\Gamma \vdash_\Sigma M = N : A \wedge \Gamma \vdash_\Sigma A = B : \mathit{type} \wedge A^- = \tau \wedge B^- = \tau$.

Second problem The second difficulty is more serious. The difficulty arises in the proof of soundness for the extensionality rule in the algorithmic type equivalence judgment (part 3 of Thm. 2). In this case, we have a derivation of the form:

$$\frac{(x, \tau)::\Gamma^- \vdash_\Sigma A\, x \Leftrightarrow B\, x : \kappa \quad x \# (\Sigma, \Gamma^-, A, B)}{\Gamma^- \vdash_\Sigma A \Leftrightarrow B : \tau \to \kappa}$$

We can easily show that the induction hypothesis applies, using the same technique as above, ultimately deriving $(x, A')::\Gamma \vdash_\Sigma A\, x = B\, x : K$ for some A' and K. However, we cannot complete the proof of this case in the same way as for object extensionality, because HP05's variant of LF does *not* include a type-level extensionality rule that permits us to conclude that $\Gamma \vdash_\Sigma A = B : \Pi x{:}A'.\, K$.

There appear to be several ways to fix this problem. One way is to just add the extensionality rule for types to the definitional system. Using our formalization, we were able to verify that this solves the problem and does not introduce any new complications (for this we had to make sure that every proof done earlier is either not affected by this additional rule or can be extended to include it).

A second solution, suggested by Harper (private communication), is to observe that the original algorithmic rules were unnecessarily general. In the absence of type-level λ-abstraction, the weaker, syntax-directed type equivalence rules shown in Fig. 4 suffice. In this solution, the type-level logical relation needs to be changed to

$$\Delta \vdash_\Sigma A = B \in [\![\kappa]\!] = \Delta \vdash_\Sigma A \Leftrightarrow B : \kappa\ .$$

The first two solutions however establish soundness only for a variant of the original definitions. Neither tells us whether the *original* equivalence algorithm is sound with respect to the *original* definitional system in HP05. We found a third solution for the problem, in which we were able to "patch" the proof of HP05 in order to establish soundness for the original definitions. This involves showing that the original type equivalence judgments imply weak type equivalence and that weak type equivalence is sound with respect to definitional equivalence.

To show that algorithmic and structural type equivalence imply weak type equivalence, we need to show that weak type equivalence admits extensionality (Lem. 24 below). This is nontrivial. To establish extensionality for weak type equivalence, we first need to develop some syntactic properties of algorithmic equivalence for objects, in particular that if $\Delta \vdash_\Sigma x \Leftrightarrow x : \tau$ then $(x, \tau) \in \Delta$. This requires several auxiliary definitions and lemmas. We say that an object M_0 is an *applied variable* if it is of the form

$$M_0 ::= x \mid M_0\, x$$

that is, it is a variable applied to a sequence of variables. Clearly, applied variables are weak head normal forms:

Lemma 20. *If M_0 is an applied variable then M_0 is in weak head normal form.*

We then introduce a weak well-formedness relation $\Delta \vdash_0 M_0 : \tau$ for applied variables, defined as follows:

$$\frac{(x, \tau) \in \Delta}{\Delta \vdash_0 x : \tau} \qquad \frac{\Delta \vdash_0 M_0 : \tau_1 \to \tau_2 \quad (y, \tau_1) \in \Delta}{\Delta \vdash_0 M_0\, y : \tau_2}$$

It is easy to show that that $\vdash_0$ satisfies strengthening:

Lemma 21. *If $(y, \tau')::\Delta \vdash_0 M_0 : \tau$ and $y \# M_0$ then $\Delta \vdash_0 M_0 : \tau$.*

Lemma 22. *Suppose M_0 is an applied variable and $\vdash \Delta$ sctx.*

1. *If $\Delta \vdash_\Sigma M_0 \Leftrightarrow M_0 : \tau$ then $\Delta \vdash_0 M_0 : \tau$.*
2. *If $\Delta \vdash_\Sigma M_0 \leftrightarrow M_0 : \tau$ then $\Delta \vdash_0 M_0 : \tau$.*

Proof. Induction on derivations. Lem. 20 is needed to show that the cases involving weak head reduction are vacuous. The only other interesting case is the case for an extensionality rule

$$\frac{(x, \tau_1)::\Delta \vdash_\Sigma M_0\, x \Leftrightarrow M_0\, x : \tau_2 \quad x \# (\Sigma, \Delta, M_0, M_0)}{\Delta \vdash_\Sigma M_0 \Leftrightarrow M_0 : \tau_1 \to \tau_2}$$

By induction, we have that $(x, \tau_1)::\Delta \vdash_0 M_0\, x : \tau_2$. By inversion, we can show that $(x, \tau_1)::\Delta \vdash_0 M_0 : \tau_1 \to \tau_2$. To complete the proof, we use Lem. 21 to show that $\Delta \vdash_0 M_0 : \tau_1 \to \tau_2$, which follows since $x \# M_0$. □

Corollary 2. *If $\Delta \vdash_\Sigma x \Leftrightarrow x : \tau$ and $\vdash \Delta$ sctx then $(x, \tau) \in \Delta$.*

We also need to establish strengthening for weak algorithmic type equivalence:

Lemma 23 (Strengthening of weak type equivalence)**.** *If $\Delta'@[(x, \tau)]@\Delta \vdash_\Sigma A \rightleftharpoons B : \kappa$ and $x \# (\Delta', A, B)$ then $\Delta'@\Delta \vdash_\Sigma A \rightleftharpoons B : \kappa$.*

Proof. Straightforward induction on derivations. Note that we need Lem. 10 here for strengthening of algorithmic object equivalence subderivations. □

We now establish the admissibility of extensionality for weak type equivalence:

Lemma 24 (Extensionality of weak type equivalence)**.** *If $(x, \tau)::\Delta \vdash_\Sigma A\, x \rightleftharpoons B\, x : \kappa$ and $x \# (\Sigma, \Delta, A, B)$ and $\vdash \Delta$ sctx then $\Delta \vdash_\Sigma A \rightleftharpoons B : \tau \to \kappa$.*

Proof. By inversion, we have subderivations $(x, \tau)::\Delta \vdash_\Sigma A \rightleftharpoons B : \tau' \to \kappa$ and $(x, \tau)::\Delta \vdash_\Sigma x \Leftrightarrow x : \tau'$ for some τ' Using Cor. 2 on the second subderivation we have that $(x, \tau') \in (x, \tau)::\Delta$ and using the validity of $(x, \tau)::\Delta$ we know that $\tau = \tau'$. Hence, $(x, \tau)::\Delta \vdash_\Sigma A \rightleftharpoons B : \tau \to \kappa$. Using Lem. 23 we conclude $\Delta \vdash_\Sigma A \rightleftharpoons B : \tau \to \kappa$. □

Lemma 25. *Suppose $\vdash \Delta$ sctx. Then*

1. *If $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ then $\Delta \vdash_\Sigma A \rightleftharpoons B : \kappa$.*
2. *If $\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$ then $\Delta \vdash_\Sigma A \rightleftharpoons B : \kappa$.*

Proof. By induction on the structure of derivations. The case for the algorithmic type extensionality rule requires Lem. 24. □

The proof of Thm. 2 is completed as follows.

Lemma 26 (Soundness of weak type equivalence)**.** *. If $\Gamma^- \vdash_\Sigma A \rightleftharpoons B : \kappa$ and $\Gamma \vdash_\Sigma A : K$ and $\Gamma \vdash_\Sigma B : L$ then $\Gamma \vdash_\Sigma A = B : K$, $\Gamma \vdash_\Sigma K = L : kind$, $K^- = \kappa$ and $L^- = \kappa$.*

Proof. Similar to the proof of soundness of algorithmic and structural type equivalence from HP05. Requires soundness of object equivalence (Lem. 19). □

Lemma 27 (Soundness of algorithmic type equivalence)**.**

1. *If $\Gamma^- \vdash_\Sigma A \Leftrightarrow B : K^-$ and $\Gamma \vdash_\Sigma A : K$ and $\Gamma \vdash_\Sigma B : K$ then $\Gamma \vdash_\Sigma A = B : K$.*
2. *If $\Gamma^- \vdash_\Sigma A \leftrightarrow B : \kappa$ and $\Gamma \vdash_\Sigma A : K$ and $\Gamma \vdash_\Sigma B : L$ then $\Gamma \vdash_\Sigma A = B : K$, $\Gamma \vdash_\Sigma K = L : kind$, $K^- = \kappa$ and $L^- = \kappa$.*

Proof. Immediate using Lem. 25 and 26. □

Lemma 28 (Soundness of algorithmic kind equivalence)**.** *If $\Gamma^- \vdash_\Sigma K \Leftrightarrow L : kind^-$ and $\Gamma \vdash_\Sigma K : kind$ and $\Gamma \vdash_\Sigma L : kind$ then $\Gamma \vdash_\Sigma K = L : kind$.*

Proof. As in HP05, using Lem. 27 as necessary. □

Thm. 2 follows immediately from Lem. 19, 27 and 28.

3.4. Additional properties

After the soundness and completeness proof, HP05 introduces an algorithmic version of the typechecking judgment, proves additional syntactic properties, sketches proofs of decidability, and discusses quasicanonical forms and adequacy of LF encodings of object languages.

Algorithmic typechecking The typechecking algorithm in HP05 omitted explicit definitions of algorithmic signature and context validity. In our formalization, we added these (obvious) rules, as shown in Fig. 5. The remaining rules are the same as in HP05 except for a trivial typographical error in the rule for type constants. Proving the soundness and completeness of algorithmic typechecking is a (mostly) straightforward induction:

Theorem 7 (Soundness of algorithmic typechecking)**.**

1. *If $\vdash \Sigma \Rightarrow sig$ then $\vdash \Sigma\ sig$.*
2. *If $\vdash_\Sigma \Gamma \Rightarrow ctx$ then $\vdash_\Sigma \Gamma\ ctx$.*
3. *If $\Gamma \vdash_\Sigma M \Rightarrow A$ then $\Gamma \vdash_\Sigma M : A$.*
4. *If $\Gamma \vdash_\Sigma A \Rightarrow K$ then $\Gamma \vdash_\Sigma A : K$.*

$\boxed{\vdash \Sigma \Rightarrow sig}$

$$\frac{}{\vdash [] \Rightarrow sig} \qquad \frac{\vdash \Sigma \Rightarrow sig \quad [] \vdash_\Sigma A \Rightarrow type \quad c \# \Sigma}{\vdash (c, A)::\Sigma \Rightarrow sig} \qquad \frac{\vdash \Sigma \Rightarrow sig \quad [] \vdash_\Sigma K \Rightarrow kind \quad a \# \Sigma}{\vdash (a, K)::\Sigma \Rightarrow sig}$$

$\boxed{\vdash_\Sigma \Gamma \Rightarrow ctx}$

$$\frac{\vdash \Sigma \Rightarrow sig}{\vdash_\Sigma [] \Rightarrow ctx} \qquad \frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad \Gamma \vdash_\Sigma A \Rightarrow type \quad x \# \Gamma}{\vdash_\Sigma (x, A)::\Gamma \Rightarrow ctx}$$

$\boxed{\Gamma \vdash_\Sigma M \Rightarrow A}$

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad (x, A) \in \Gamma}{\Gamma \vdash_\Sigma x \Rightarrow A} \qquad \frac{\Gamma \vdash_\Sigma A_1 \Rightarrow type \quad (x, A_1)::\Gamma \vdash_\Sigma M_2 \Rightarrow A_2 \quad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \lambda x{:}A_1.\, M_2 \Rightarrow \Pi x{:}A_1.\, A_2}$$

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad (c, A) \in \Sigma}{\Gamma \vdash_\Sigma c \Rightarrow A} \qquad \frac{\Gamma \vdash_\Sigma M_1 \Rightarrow \Pi x{:}A_2'.\, A_1 \quad \Gamma \vdash_\Sigma M_2 \Rightarrow A_2 \quad \Gamma^- \vdash_\Sigma A_2 \Leftrightarrow A_2' : type^- \quad x \# \Gamma}{\Gamma \vdash_\Sigma M_1\, M_2 \Rightarrow A_1[x{:=}M_2]}$$

$\boxed{\Gamma \vdash_\Sigma A \Rightarrow K}$

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad (a, K) \in \Sigma}{\Gamma \vdash_\Sigma a \Rightarrow K} \qquad \frac{\Gamma \vdash_\Sigma A \Rightarrow \Pi x{:}A_2'.\, K_1 \quad \Gamma \vdash_\Sigma M \Rightarrow A_2 \quad \Gamma^- \vdash_\Sigma A_2 \Leftrightarrow A_2' : type^- \quad x \# \Gamma}{\Gamma \vdash_\Sigma A\, M \Rightarrow K_1[x{:=}M]}$$

$$\frac{\Gamma \vdash_\Sigma A_1 \Rightarrow type \quad (x, A_1)::\Gamma \vdash_\Sigma A_2 \Rightarrow type \quad x \# (\Gamma, A_1)}{\Gamma \vdash_\Sigma \Pi x{:}A_1.\, A_2 \Rightarrow type}$$

$\boxed{\Gamma \vdash_\Sigma K \Rightarrow kind}$

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx}{\Gamma \vdash_\Sigma type \Rightarrow kind} \qquad \frac{\Gamma \vdash_\Sigma A \Rightarrow type \quad (x, A)::\Gamma \vdash_\Sigma K \Rightarrow kind \quad x \# (\Gamma, A)}{\Gamma \vdash_\Sigma \Pi x{:}A.\, K \Rightarrow kind}$$

Figure 5. Algorithmic typechecking rules

5. *If* $\Gamma \vdash_\Sigma K \Rightarrow kind$ *then* $\Gamma \vdash_\Sigma K : kind$.

Theorem 8 (Completeness of algorithmic typechecking)**.**
1. *If* $\vdash \Sigma\ sig$ *then* $\vdash \Sigma \Rightarrow sig$.
2. *If* $\vdash_\Sigma \Gamma\ ctx$ *then* $\vdash_\Sigma \Gamma \Rightarrow ctx$.
3. *If* $\Gamma \vdash_\Sigma M : A$ *then* $\exists A'.\ \Gamma \vdash_\Sigma M \Rightarrow A' \wedge \Gamma \vdash_\Sigma A = A' : type$.
4. *If* $\Gamma \vdash_\Sigma A : K$ *then* $\exists K'.\ \Gamma \vdash_\Sigma A \Rightarrow K' \wedge \Gamma \vdash_\Sigma K = K' : kind$.
5. *If* $\Gamma \vdash_\Sigma K : kind$ *then* $\Gamma \vdash_\Sigma K \Rightarrow kind$.

Strengthening and strong extensionality We omit detailed proofs and discussion of these results which can be found in [19]. The strengthening property states that all of the definitional judgments are preserved by removing an unused variable from the context.

Theorem 9 (Strengthening)**.** *Let* $\Gamma = \Gamma_2@[(x, B)]@\Gamma_1$.
1. *If* $\vdash_\Sigma \Gamma\ ctx$ *and* $x \# \Gamma_2$ *then* $\vdash_\Sigma \Gamma_2@\Gamma_1\ ctx$.
2. *If* $\Gamma \vdash_\Sigma M : A$ *and* $x \# (\Gamma_2, M, A)$ *then* $\Gamma_2@\Gamma_1 \vdash_\Sigma M : A$.
3. *If* $\Gamma \vdash_\Sigma M = N : A$ *and* $x \# (\Gamma_2, M, N, A)$ *then* $\Gamma_2@\Gamma_1 \vdash_\Sigma M = N : A$.

Similarly for kinds and type families.

HP05 also sketched a proof of a *strong* extensionality rule requiring fewer typechecking hypotheses. We were also able to verify this, but the proof is not as straightforward as in the article; the first line of the proof appeals to "without loss of generality" reasoning about inversion and renaming principles which was nontrivial to formalize.

Theorem 10 (Strong Extensionality)**.**
If $(x, A_1)::\Gamma \vdash_\Sigma M\, x = N\, x : A_2$ *and* $x \# (M, N)$ *then* $\Gamma \vdash_\Sigma M = N : \Pi x{:}A_1.\, A_2$.

Decidability HP05 also sketches proofs of the decidability of the algorithmic judgments (and hence also the definitional system). Reasoning about decidability within Isabelle/HOL is not straightforward because Isabelle/HOL is based on classical logic. Thus, unlike Coq or other constructive systems, we cannot prove decidability of P simply by constructively proving $P \vee \neg P$. Isabelle/HOL does have some support for extracting code from proofs that avoid non-constructive features, but code extraction currently does not work on proofs about nominal datatypes.

We write $M{\Downarrow}$ to indicate that M is strongly weak head normalizing. As a sanity check, we have shown that well-formed terms are strongly weak head normalizing.

Theorem 11. *If* $\Gamma \vdash_\Sigma M : A$ *then* $M{\Downarrow}$.

Proof. We first show the (standard) property that if $M\ N{\Downarrow}$ then $M{\Downarrow}$. We then show that if $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ then $M{\Downarrow}$ by induction on derivations. The main result follows by reflexivity and Thm. 1. □

We have also formalized what we believe is the essence of the decidability proof using the following methodology. For each property R we wish to prove decidable, possibly under conditions P:
1. Inductively define a "complement" relation R'.
2. Observe (informally) that R and R' are recursively enumerable.
3. Prove that $\neg\ (R \wedge R')$.
4. Prove that P implies $R \vee R'$.
5. Hence R is decidable (assuming P) since it is both r.e. and co-r.e.

We have introduced inductively defined complements for the algorithmic equivalence and typechecking judgments and verified parts 3 and 4 above for each of them. We have

not verified steps 2 or 5. We call a formula *quasidecidable* if both it and its negation are equivalent to an inductively defined relation, as described above. We were able to prove the following lemma analogous to HP05's Lemma 6.1:

Theorem 12 (Quasidecidability of alg. equiv.)**.**

1. *If* $\Delta \vdash_\Sigma M \Leftrightarrow M' : \tau$ *and* $\Delta \vdash_\Sigma N \Leftrightarrow N' : \tau$ *then* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *is quasidecidable.*
2. *If* $\Delta \vdash_\Sigma M \leftrightarrow M' : \tau_1$ *and* $\Delta \vdash_\Sigma N \leftrightarrow N' : \tau_2$ *then* $\exists\, \tau_3.\ \Delta \vdash_\Sigma M \leftrightarrow N : \tau_3$ *is quasidecidable.*
3. *If* $\Delta \vdash_\Sigma A \Leftrightarrow A' : \kappa$ *and* $\Delta \vdash_\Sigma B \Leftrightarrow B' : \kappa$ *then* $\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$ *is quasidecidable.*
4. *If* $\Delta \vdash_\Sigma A \leftrightarrow A' : \kappa_1$ *and* $\Delta \vdash_\Sigma B \leftrightarrow B' : \kappa_2$ *then* $\exists\, \kappa_3.\ \Delta \vdash_\Sigma A \leftrightarrow B : \kappa_3$ *is quasidecidable.*
5. *If* $\Delta \vdash_\Sigma K \Leftrightarrow K' : kind^-$ *and* $\Delta \vdash_\Sigma L \Leftrightarrow L' : kind^-$ *then* $\Delta \vdash_\Sigma K \Leftrightarrow L : kind^-$ *is quasidecidable.*

We were also able to prove that the algorithmic judgments are quasidecidable, which is the key step in HP05's Theorem 6.5. Proving exclusivity required establishing uniqueness of algorithmic typechecking.

Lemma 29 (Uniqueness of algorithmic typing)**.**

1. *If* $\Gamma \vdash_\Sigma M \Rightarrow A$ *and* $\Gamma \vdash_\Sigma M \Rightarrow A'$ *then* $A = A'$.
2. *If* $\Gamma \vdash_\Sigma A \Rightarrow K$ *and* $\Gamma \vdash_\Sigma A \Rightarrow K'$ *then* $K = K'$.

Theorem 13 (Quasidecidability of algorithmic typing)**.**

1. *For any* Σ, $\vdash \Sigma \Rightarrow$ *sig is quasidecidable.*
2. *For any* Σ,Γ, *if* $\vdash \Sigma \Rightarrow$ *sig holds then* $\vdash_\Sigma \Gamma \Rightarrow$ *ctx is quasidecidable.*
3. *For any* Σ,Γ,M, *if* $\vdash_\Sigma \Gamma \Rightarrow$ *ctx holds then* $\exists A.\ \Gamma \vdash_\Sigma M \Rightarrow A$ *is quasidecidable.*
4. *For any* Σ,Γ,A, *if* $\vdash_\Sigma \Gamma \Rightarrow$ *ctx holds then* $\exists K.\ \Gamma \vdash_\Sigma A \Rightarrow K$ *is quasidecidable.*
5. *For any* Σ,Γ, *if* $\vdash_\Sigma \Gamma \Rightarrow$ *ctx holds then* $\Gamma \vdash_\Sigma K \Rightarrow$ *kind is quasidecidable.*

Although we believe that this approach provides greater confidence in the decidability results compared to the proof sketches in HP05, these quasidecidability results leave room for improvement. However, reasoning about decidability in Isabelle/HOL seems to be an open problem in general, so we leave the question of fully formalizing the decidability of LF's algorithmic judgments to future work.

Quasicanonical forms The last section of HP05 discusses the existence and properties of *quasicanonical forms* which can be used to prove adequacy theorems about languages encoded in LF, as illustrated by a small example. We have also formalized and verified this section. The full details are omitted but can be found in [19].

4 Discussion

Methodological observations The formalization was performed by two of the authors; one is a developer of the Nominal Datatype Package and expert Isabelle/HOL user and the other had less than three months' prior experience with either. We estimate that the total effort involved was at most three person-months. Although there is still plenty of room for improvement in both Isabelle/HOL and the Nominal Datatype Package, our experience suggests that these tools can now be used to perform significant formalizations within reasonable time-frames, at least by expert users.

It took approximately six person-weeks to formalize everything up to the soundness proof (including pondering why the omitted case did not go through). However, once Harper and Pfenning confirmed that this case was indeed not handled correctly in their proof, one of the authors was able to check within 2 hours that adding a type-extensionality rule solves the problem. Re-checking the proof on paper would have meant reviewing approximately 31 pages of proofs. Since then we found another solution for the problem and checked the validity of a solution suggested by Harper. As a practical matter, the ability to rapidly evaluate the effects of changes to the system was essential for finding these solutions and evaluating other possibilities. In a similar formalization project, the first author showed that a central lemma in the informal proof in his PhD-thesis can be repaired [17].

Our formalization using nominal datatypes follows that given in HP05 very closely—more closely, we believe, than has been demonstrated by any other currently available technique. As illustration of this point, we have prepared this paper using Isabelle's documentation facilities [12]. Most lemmas, theorems, and definitions have been generated directly from the formalization (the main exceptions are the quasidecidability properties, which are paraphrased).

In Table 1, we report some simple metrics about our formalization such as the sizes, number of lines of code, and number of lemmas in each theory in the main formalization. As Table 1 shows, the core `LF` theory accounts for about 25% of the development. These syntactic properties are mostly straightforward, and their proofs merit only cursory discussion in HP05, but some lemmas have many cases which must all be handled. The `Decidability` theory accounts for another 25%; much of this is due to the quasidecidability proofs. The remaining theories account for at most 5–10% of the formalisation each; the `WeakAlgorithm` theory defines the weak algorithmic equivalence judgment and proves the properties needed for the third solution, and accounts for only around 2% of the total development.

The merit of metrics such as proof size or number of lemmas is debatable. We have not attempted to distinguish between "meaningful" lines of proof vs. blank or comment lines; nor have we distinguished between significant and trivial lemmas. Nevertheless, this information should at

Table 1. Summary of the formalization

Theory	Description	Size (bytes)	Lines	Lemmas
`LF`	Syntax and definitional judgments of LF	126,676	2,726	103
`Erasure`	Simple types and kinds, erasure	10,073	312	22
`PairOrdering`	Pair ordering used for transitivity	962	29	3
`Algorithm`	Algorithmic equivalence judgments and properties	45,116	956	42
`LogicalRelation`	Logical relation, completeness proof	50,781	770	21
`WeakAlgorithm`	Weak algorithmic typechecking	8,866	208	7
`Soundness`	Subject reduction, soundness proofs	30,551	564	8
`Decidability`	Algorithmic typechecking, strengthening, quasidecidability	143,673	2,896	69
`Canonical`	Quasicanonical forms, adequacy example	48,823	1,115	47
Total		465,521	9,576	322

least convey an idea of the *relative* effort involved in each part of the proof.

Correctness of the representation The facilities for defining and reasoning about languages with binding provided by the Nominal Datatype Package are convenient, but may appear strange to readers unfamiliar with nominal logic and abstract syntax. Thus, a skeptical reader might ask whether our definitions and reasoning principles are really *correct*; that is, whether they are equivalent to the definitions in HP05, as formalized using some conventional approach to binding syntax. For higher-order abstract syntax representations, this property is often called *adequacy*; this term appears to have been coined in the context of LF [6].

Adequacy is also important for nominal techniques and deserves further study. For the purposes of this paper, however, we view our formalization of LF's syntax and inference rules using nominal datatypes as an *axiomatization*, and leave the question of its correctness or adequacy for future work. Norrish and Vestergaard [13] have formalized isomorphisms between several variants of the λ-calculus, including a nominal representation. We believe extending their approach to the syntax of LF would be routine, if tedious.

5 Related and Future Work

McKinna and Pollack's LEGO formalization of Pure Type Systems [9] is probably the most extensive formalization of a dependent type theory in a theorem prover. Their formalization considered primarily syntactic properties of pure type systems with β-equivalence, including a proof of strengthening. Pollack [15] subsequently verified the partial correctness of typechecking algorithms for certain classes of Pure Type Systems including LF.

Aydemir et al. [2] have developed a methodology for formalizing metatheory in Coq based on using de Bruijn indices to manage binding, and using cofinite quantification to handle fresh names. Using their methodology to formalize the results in this paper would provide a useful comparison of these approaches, particularly concerning decidability.

Algorithms for equivalence and canonicalisation for dependent type theories have been studied by several authors. Prior work on equivalence checking for LF has focused on first checking well-formedness using erased types, then β- or $\beta\eta$-normalizing; these approaches are discussed in detail in [8]. Coquand's algorithm [3] is similar to Harper and Pfenning's but operates on untyped terms. Goguen's approach [5] involves first η-expanding and then β-normalizing, and relies on standard (nontrivial) properties such as strong normalization of β-reduction and strengthening. It may be interesting to verify these algorithms.

We chose to formalize Harper and Pfenning's article [8] because it is the most recent and most detailed development we could find. Moreover, their work seems more extensible. Another reason is that the quality standards in the LF-community are so high, that peer-reviewed work is generally trusted. Appel, for example, told us that he trusts the implementation of a type-checker for LF presented in [1], because first the code is very small and second the theoretical underpinnings have been studied thoroughly by Harper and Pfenning. For such "follow-up" work it is crucial that we were able to formalize the soundness and completeness results in HP05.

Our formalization provides a foundation for several possible future investigations. We are interested in extending our formalization to include verifying Twelf-style metareasoning about LF specifications, following Harper and Licata's detailed informal development [7]. Doing so could make it possible to extract Isabelle/HOL theorems from Twelf proofs. It would also be interesting to extend our formalization to accommodate extensions to LF involving (ordered) linear logic, concurrency, proof-irrelevance, or singleton kinds, as discussed in [8, Sec. 8]. We hope that anyone who proposes an extension to LF will be able to use our formalization as a starting point for verifying its metatheory.

6 Conclusions

LF is an extremely convenient tool for defining logics and other calculi. It has many compelling applications and underlies the system Twelf, which has a proven record in formalizing many programming language calculi. Hence, it is of intrinsic interest to verify key properties of LF's metatheory, such as the correctness and decidability of the typechecking algorithms. We have done so, using the Nominal Datatype Package for Isabelle/HOL. The infrastructure provided by this package allowed us to follow the proof of Harper and Pfenning closely.

For our formalization we had the advantage of working from Harper and Pfenning's carefully-written informal proof, which stood up to the rigors of mechanical formalization rather well. Still we found in this informal proof one gap and a few minor complications. We have shown that they can be repaired. We have also partially verified the decidability of the equivalence and typechecking algorithms, although some work remains to formally prove decidability per se. Formalizing decidability proofs of any kind in Isabelle/HOL appears to be an open problem, so we leave this for future work.

While verifying correctness of proofs is a central motivation for doing formalizations, it is *not* the only one. There is a second important benefit—they can be used to experiment with changes to the system. By "replaying" a modified formalization in a theorem prover one can immediately focus on places where the proof fails. This ability was essential in fixing the soundness proof.

Our formalization is not an end in itself but also provides a foundation for further study in several directions. Researchers using LF may find it useful for checking the adequacy of their LF encodings. Researchers developing extensions to LF based on Harper and Pfenning's algorithm may find our formalization useful as a starting point for verifying the metatheory of such extensions. More ambitiously, we contemplate formalizing the meaning and correctness of Twelf's metatheoretic reasoning facilities inside Isabelle/HOL, and extracting Isabelle/HOL theorems from Twelf proofs.

Acknowledgments: We are extremely grateful to Bob Harper for many discussions about LF. Benjamin Pierce and Stephanie Weirich have also made helpful comments on drafts of this paper.

References

[1] A. Appel, N. Michael, A. Stump, and R. Virga. A trustworthy proof checker. *J. Autom. Reasoning*, 31:231–260, 2003.

[2] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *POPL*, pages 3–15. ACM, 2008.

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

[4] H. Geuvers and E. Barendsen. Some logical and syntactical observations concerning the first-order dependent type system λP. *Mathematical Structures in Computer Science*, 9(4):335–359, 1999.

[5] H. Goguen. A syntactic approach to eta equality in type theory. In *POPL*, pages 75–84. ACM, 2005.

[6] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[7] R. Harper and D. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Programming*, 2007. To appear.

[8] R. Harper and F. Pfenning. On equivalence and canonical forms in the LF type theory. *ACM Transactions on Computational Logic*, 6(1):61–101, 2005.

[9] J. McKinna and R. Pollack. Some lambda calculus and type theory formalized. *J. Autom. Reasoning*, 23(3-4):373–409, 1999.

[10] J. Narboux and C. Urban. Formalising in Nominal Isabelle Crary's completeness proof for equivalence checking. In *LFMTP*, volume 196 of *ENTCS*, 2007.

[11] G. C. Necula. Proof-carrying code. In *POPL*, pages 106–119. ACM, 1997.

[12] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[13] M. Norrish and R. Vestergaard. Proof pearl: de bruijn terms really do work. In *TPHOLs*, volume 4732 of *LNCS*, pages 207–222. Springer, 2007.

[14] F. Pfenning and C. Schürmann. System description: Twelf–a meta-logical framework for deductive systems. In *CADE*, volume 1632 of *LNAI*, pages 202–206, 1999.

[15] R. Pollack. A verified typechecker. In M. Dezani-Ciancaglini and G. D. Plotkin, editors, *TLCA*, volume 902 of *LNCS*, pages 365–380. Springer, 1995.

[16] C. Schürmann and J. Sarnat. Structural logical relations. In LICS, 2008.

[17] C. Urban. Revisiting cut-elimination: One difficult proof is really a proof. In RTA, 2008.

[18] C. Urban, S. Berghofer, and M. Norrish. Barendregt's variable convention in rule inductions. In *CADE*, volume 4603 of *LNAI*, pages 35–50, 2007.

[19] C. Urban, J. Cheney, and S. Berghofer. Mechanising the metatheory of LF, 2008. Technical Report `arXiv:0804.1667`.

[20] C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In *CADE*, volume 3632 of *LNCS*, pages 38–53, 2005.