Mechanizing the Metatheory of LF

CHRISTIAN URBAN
TU Munich
and
JAMES CHENEY
University of Edinburgh
and
STEFAN BERGHOFER
TU Munich

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. We also formally derive a version of the type checking algorithm from which Isabelle/HOL can generate executable code. 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.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Language**]: Mathematical Logic—*Lambda calculus and related systems*

General Terms: Languages, theorem provers

Additional Key Words and Phrases: Logical frameworks, Nominal Isabelle

1. INTRODUCTION

The (Edinburgh) Logical Framework (LF) is a dependent type theory introduced by Harper, Honsell and Plotkin [1993] as a framework for specifying and reasoning about formal systems. It has found many applications, such as proof-carrying code [Necula 1997]. The Twelf system [Pfenning and Schürmann 1999] 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 types may contain expressions with computational behavior. This means that typechecking depends on equality-tests for LF-terms and LF-types. Several algorithms for such equality-tests have been proposed in the literature [Coquand 1991; Goguen 2005b; Harper and Pfenning 2005]. Harper and Pfenning [2005] present 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 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. To our knowledge this is the first formalization of these or comparable results. While most of the formal proofs go through as described by Harper and Pfenning [2005], we found a few do *not* go through as described, and there is a *gap* in the proof of soundness. Although the problem can be avoided easily by adding to or changing the rules of Harper and Pfenning [2005], we found that it was still possible to prove the original results, though the argument 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 [Nipkow et al. 2002] and the Nominal Datatype Package [Urban et al. 2007; Urban and Tasson 2005; Urban 2008] 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 [Narboux and Urban 2007]; it is worth noting that logical relations proofs are currently not easy to formalize in Twelf itslef, despite the recent breakthrough by Schürmann and Sarnat [2008].

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.

We were able to follow the arguments in Harper and Pfenning's article very closely by using the Nominal Datatype Package for our formalisation, but the current system does not allow us to generate executable code directly from definitions involving nominal datatypes. We therefore also implemented a type-checking algorithm based on the locally nameless approach for representing binders [McKinna and Pollack 1999; Aydemir et al. 2008]. We proved that the nominal datatype formalization of Harper and Pfenning's algorithm is equivalent to the locally nameless formulation. Moreover, by making the choice of fresh names explicit, we can generate a working ML implementation directly from the verified formalization.

Contributions:. We present a formalization of the soundness and completeness of the equivalence algorithm presented by Harper and Pfenning [2005]. We discuss additional lemmas and other complications arising during the formalization, and discuss the gap in the soundness proof and its solutions in detail. We also discuss our partial formalization of decidability and other results from [Harper and Pfenning 2005] which were omitted in the conference version of this paper [Urban et al. 2008], including proving the admissibility of strengthening and strong extensionality rules for LF, proving the existence and uniqueness of quasicanonical forms, and reasoning about the adequacy of encodings of object languages in LF. Finally, we derive an equivalent version of the type checking algorithm from which Isabelle/HOL can generate executable code.

2. BACKGROUND

We used the Nominal Datatype Package in Isabelle/HOL [Urban et al. 2007; Urban and Tasson 2005; Urban 2008] to formalize the syntax and judgments of LF. The key features we rely upon are

(1) support for *nominal datatypes* with a built-in notion of binding (i.e. α-equivalence classes),

(2) facilities for defining functions over nominal datatypes (such as substitution) by *(nominal) primitive recursion*, and

(3) *strong induction principles* for datatypes and inductive definitions that build in Barendregt-style renaming conventions.

Together, these features make it possible to formalize most of the definitions and proofs following their paper versions closely. We will not review the features of this system in this article, but will discuss details of the formalization only when they intrude. The interested reader is referred to previous work on nominal techniques and the Nominal Datatype Package for further details [Gabbay and Pitts 2002; Pitts 2006; Urban et al. 2007; Urban and Tasson 2005; Urban 2008].

2.1 Syntax of LF

The logical framework LF [Harper et al. 1993] is a dependent type theory. We present it here following closely the article by Harper and Pfenning [2005], 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. Traditionally, LF has included λ-abstraction at the level of both types and objects. However, Geuvers and Barendsen [1999] established that type-level λ-abstraction is superfluous in LF. Accordingly, HP05 omits type-level λ-abstraction, and so do we.

We formalize the syntax of LF as nominal datatypes since the constructors λ and Π bind variables. Substitutions are represented as lists of variable-term pairs and we define capture avoiding substitution in the standard way as

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

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

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

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

LF also includes *signatures* Σ and *contexts* Γ, both of which we represent as lists of pairs. We use ML-like notation $[]$ for the empty list and $x :: L$ for list construction. 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). List inclusion for contexts is defined as follows:

$$\Gamma_1 \subseteq \Gamma_2 \stackrel{\text{def}}{=} \forall\, x\ A.\ (x,\ A) \in \Gamma_1 \text{ implies } (x,\ A) \in \Gamma_2$$

2.2 Validity and Definitional Equivalence

HP05 defines two judgments for identifying valid signatures and contexts, which we formalize in Fig. 1. 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 make the dependence of all judgments on Σ explicit.

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. 2 and 3. There are three judgments for validity and three for equivalence corresponding to objects, type families and kinds respectively:

	Objects	Type families	Kinds
Validity	$\Gamma \vdash_\Sigma M : A$	$\Gamma \vdash_\Sigma A : K$	$\Gamma \vdash_\Sigma K : \mathit{kind}$
Equivalence	$\Gamma \vdash_\Sigma M = N : A$	$\Gamma \vdash_\Sigma A = B : K$	$\Gamma \vdash_\Sigma K = L : \mathit{kind}$

These six judgments are defined simultaneously with signature validity ($\vdash \Sigma\ \mathit{sig}$) and context validity ($\vdash_\Sigma \Gamma\ \mathit{ctx}$) by induction. We added explicit validity hypotheses

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

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

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

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

Fig. 1. Validity rules for signatures and contexts

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

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

$$\frac{\Gamma \vdash_\Sigma M_1 : \Pi x{:}A_2.\ A_1 \quad \Gamma \vdash_\Sigma M_2 : A_2 \quad x \# \Gamma}{\Gamma \vdash_\Sigma M_1\ M_2 : A_1[x{:=}M_2]}$$

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

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

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

$$\frac{\vdash_\Sigma \Gamma\ ctx \quad (a,\ K) \in \Sigma}{\Gamma \vdash_\Sigma a : K}$$

$$\frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.\ K \quad \Gamma \vdash_\Sigma M : B \quad x \# \Gamma}{\Gamma \vdash_\Sigma A\ M : K[x{:=}M]}$$

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

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

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

$$\frac{\vdash_\Sigma \Gamma\ ctx}{\Gamma \vdash_\Sigma type : kind}$$

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

Fig. 2. Validity rules for kinds, type families and objects.

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 [Urban et al. 2007].

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

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

$$\frac{\Gamma \vdash_\Sigma M_1 = N_1 : \Pi x{:}A_2.\ A_1 \quad \Gamma \vdash_\Sigma M_2 = N_2 : A_2 \quad x \# \Gamma}{\Gamma \vdash_\Sigma M_1\ M_2 = N_1\ N_2 : A_1[x{:=}M_2]}$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma {A_1}' = A_1 : type \\ \Gamma \vdash_\Sigma {A_1}'' = A_1 : type \quad \Gamma \vdash_\Sigma A_1 : type \quad (x,\, A_1){::}\Gamma \vdash_\Sigma M_2 = N_2 : A_2 \quad x \# \Gamma\end{array}}{\Gamma \vdash_\Sigma \lambda x{:}{A_1}'.\ M_2 = \lambda x{:}{A_1}''.\ N_2 : \Pi x{:}A_1.\ A_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\ A_2 \\ \Gamma \vdash_\Sigma N : \Pi x{:}A_1.\ A_2 \quad \Gamma \vdash_\Sigma A_1 : type \quad (x,\, A_1){::}\Gamma \vdash_\Sigma M\ x = N\ x : A_2 \quad x \# \Gamma\end{array}}{\Gamma \vdash_\Sigma M = N : \Pi x{:}A_1.\ A_2}$$

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

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

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

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

$$\frac{\vdash_\Sigma \Gamma\ ctx \quad (a,\, K) \in \Sigma}{\Gamma \vdash_\Sigma a = a : K}$$

$$\frac{\Gamma \vdash_\Sigma A = B : \Pi x{:}C.\ K \quad \Gamma \vdash_\Sigma M = N : C \quad x \# \Gamma}{\Gamma \vdash_\Sigma A\ M = B\ N : K[x{:=}M]}$$

$$\frac{\Gamma \vdash_\Sigma A_1 = B_1 : type \quad \Gamma \vdash_\Sigma A_1 : type \quad (x,\, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2 : type \quad x \# \Gamma}{\Gamma \vdash_\Sigma \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : type}$$

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

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

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

$$\frac{\vdash_\Sigma \Gamma\ ctx}{\Gamma \vdash_\Sigma type = type : kind}$$

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

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

Fig. 3. Definitional equivalence rules for kinds, type families and objects.

2.3 Algorithmic Equivalence

The definitional equivalence judgment captures equivalence between LF terms, types and kinds declaratively, but it is highly nondeterministic due tothe symmetry, transitivity and conversion rules. Accordingly, HP05 introduces algorithmic equivalence judgments that are type- and syntax-directed, and the main contribution of that article is the proof that the algorithmic and declarative systems coincide.

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:

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

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

Similarly, *simple contexts* Δ, Θ consist of lists of pairs $(x,\ \tau)$ of variables and simple types. We write $\vdash \Delta\ \mathit{sctx}$ 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.

Finally, we also introduce *simple signatures*, also written Σ, consisting of lists of pairs $(c,\ \tau)$ or $(a,\ \kappa)$ of constants and simple kinds or types. We write $\vdash \Sigma\ \mathit{ssig}$ to indicate that Σ is a well-formed simple signature with no repeated type or kind assignments.

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

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

Similarly, we write Γ^- for the simple context resulting from replacing each binding $(x,\ A)$ in Γ with $(x,\ A^-)$. Likewise, we extend the erasure function to map signatures Σ^- to simple signatures Σ in the natural way.

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

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

The rules for the equivalence checking algorithm are given in Fig. 4. There are five algorithmic equivalence judgments:

	Objects	Type families	Kinds
Algorithmic	$\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$	$\Delta \vdash_\Sigma A \Leftrightarrow B : \kappa$	$\Delta \vdash_\Sigma K \Leftrightarrow L : \mathit{kind}^-$
Structural	$\Delta \vdash_\Sigma M \leftrightarrow N : \tau$	$\Delta \vdash_\Sigma A \leftrightarrow B : \kappa$	

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.

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

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

$$\frac{M \stackrel{\text{whr}}{\longrightarrow} M' \quad \Delta \vdash_\Sigma M' \Leftrightarrow N : a^-}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^-}$$

$$\frac{N \stackrel{\text{whr}}{\longrightarrow} N' \quad \Delta \vdash_\Sigma M \Leftrightarrow N' : a^-}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^-}$$

$$\frac{\Delta \vdash_\Sigma M \leftrightarrow N : a^-}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^-}$$

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

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

$$\frac{(x,\, \tau) \in \Delta \quad \vdash \Delta\ \mathit{sctx} \quad \vdash \Sigma\ \mathit{ssig}}{\Delta \vdash_\Sigma x \leftrightarrow x : \tau}$$

$$\frac{(c,\, \tau) \in \Sigma \quad \vdash \Delta\ \mathit{sctx} \quad \vdash \Sigma\ \mathit{ssig}}{\Delta \vdash_\Sigma c \leftrightarrow c : \tau}$$

$$\frac{\Delta \vdash_\Sigma M_1 \leftrightarrow N_1 : \tau_2 \to \tau_1 \quad \Delta \vdash_\Sigma M_2 \Leftrightarrow N_2 : \tau_2}{\Delta \vdash_\Sigma M_1\ M_2 \leftrightarrow N_1\ N_2 : \tau_1}$$

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

$$\frac{\Delta \vdash_\Sigma A \leftrightarrow B : \mathit{type}^-}{\Delta \vdash_\Sigma A \Leftrightarrow B : \mathit{type}^-}$$

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

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

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

$$\frac{(a,\, \kappa) \in \Sigma \quad \vdash \Delta\ \mathit{sctx} \quad \vdash \Sigma\ \mathit{ssig}}{\Delta \vdash_\Sigma a \leftrightarrow a : \kappa}$$

$$\frac{\Delta \vdash_\Sigma A \leftrightarrow B : \tau \to \kappa \quad \Delta \vdash_\Sigma M \Leftrightarrow N : \tau}{\Delta \vdash_\Sigma A\ M \leftrightarrow B\ N : \kappa}$$

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

$$\frac{\vdash \Delta\ \mathit{sctx} \quad \vdash \Sigma\ \mathit{ssig}}{\Delta \vdash_\Sigma \mathit{type} \Leftrightarrow \mathit{type} : \mathit{kind}^-}$$

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

Fig. 4. Algorithmic equivalence rules

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

3.1 Syntactic properties

The proof in HP05 starts by developing of a number of useful metatheoretic properties for the validity and equality judgments (shown in Fig. 2), such as weakening, substitution, generalizations of the conversion rules and inversion principles. Most of these properties have multiple parts corresponding to the eight different judgments in the definitional theory of LF. We will list the main properties; however, to aid readability we will only show the statements of most of these properties for the object-level judgments, and we omit symmetric cases. The full formal statements of the syntactic properties can be found in the Appendix.

To prove the main syntactic properties we needed two technical lemmas having to do with the implicit freshness and validity assumptions that must be handled explicitly in our formalization. Both are straightforward by induction, and both are needed frequently.

LEMMA 1 (FRESHNESS). *If* $x \# \Gamma$ *and* $\Gamma \vdash_\Sigma M : A$ *then* $x \# M$ *and* $x \# A$. *Similarly, if* $x \# \Gamma$ *and* $\Gamma \vdash_\Sigma M = N : A$ *then* $x \# M$ *and* $x \# N$ *and* $x \# A$.

LEMMA 2 (IMPLICIT VALIDITY). *If* $\Gamma \vdash_\Sigma M : A$ *or* $\Gamma \vdash_\Sigma M = N : A$ *then* $\vdash \Sigma$ *sig and* $\vdash_\Sigma \Gamma$ *ctx.*

LEMMA 3 (WEAKENING). *Suppose* $\vdash_\Sigma \Gamma_2$ *ctx and* $\Gamma_1 \subseteq \Gamma_2$.
(*1*) *If* $\Gamma_1 \vdash_\Sigma M : A$ *then* $\Gamma_2 \vdash_\Sigma M : A$.
(*2*) *If* $\Gamma_1 \vdash_\Sigma M = N : A$ *then* $\Gamma_2 \vdash_\Sigma M = N : A$.

LEMMA 4 (SUBSTITUTION). *Suppose* $\Gamma_2 \vdash_\Sigma P : C$ *and let* $\Gamma = \Gamma_1 @ [(y,\ C)] @ \Gamma_2$.
(*1*) *If* $\vdash_\Sigma \Gamma$ *ctx then* $\vdash_\Sigma \Gamma_1[y{:=}P] @ \Gamma_2$ *ctx.*
(*2*) *If* $\Gamma \vdash_\Sigma M : B$ *then* $\Gamma_1[y{:=}P] @ \Gamma_2 \vdash_\Sigma M[y{:=}P] : B[y{:=}P]$.
(*3*) *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\Gamma_1[y{:=}P] @ \Gamma_2 \vdash_\Sigma M[y{:=}P] = N[y{:=}P] : A[y{:=}P]$.

LEMMA 5 (CONTEXT CONVERSION). *Assume that* $\Gamma \vdash_\Sigma B : type$ *and* $\Gamma \vdash_\Sigma A = B : type$. *Then:*

(*1*) *If* $(x,\, A)::\Gamma \vdash_\Sigma M : C$ *then* $(x,\, B)::\Gamma \vdash_\Sigma M : C$
(*2*) *If* $(x,\, A)::\Gamma \vdash_\Sigma C : K$ *then* $(x,\, B)::\Gamma \vdash_\Sigma C : K$

LEMMA 6 (FUNCTIONALITY FOR TYPING). *Assume that* $\Gamma \vdash_\Sigma M : C$ *and* $\Gamma \vdash_\Sigma N : C$ *and* $\Gamma \vdash_\Sigma M = N : C$. *Then if* $\Gamma' \,@\,[(y,\, C)] \,@\, \Gamma \vdash_\Sigma P : B$ *then* $\Gamma'[y{:=}M] \,@\, \Gamma \vdash_\Sigma P[y{:=}M] = P[y{:=}N] : B[y{:=}M]$.

Since our judgements contain explicit validity statements for contexts, the proof of Lem. 6 relies on the fact that functionality holds also for contexts, namely

LEMMA 7 (FUNCTIONALITY FOR CONTEXTS). *If* $\vdash_\Sigma \Gamma' \,@\,[(x,\, C)] \,@\, \Gamma$ *ctx* *and* $\Gamma \vdash_\Sigma M : C$ *then* $\vdash_\Sigma \Gamma'[x{:=}M] \,@\, \Gamma$ *ctx*.

This fact can be established by induction on Γ'.

LEMMA 8 (VALIDITY). *Objects, types and kinds appearing in derivable judgments are valid, that is*
(*1*) *If* $\Gamma \vdash_\Sigma M : A$ *then* $\Gamma \vdash_\Sigma A : type$.
(*2*) *If* $\Gamma \vdash_\Sigma M = N : B$ *then* $\Gamma \vdash_\Sigma M : B$ *and* $\Gamma \vdash_\Sigma N : B$ *and* $\Gamma \vdash_\Sigma B : type$.

LEMMA 9 (TYPING INVERSION). *The validity rules are invertible, up to conversion of types and kinds.*
(*1*) *If* $\Gamma \vdash_\Sigma x : A$ *then* $\exists B.\ (x,\, B) \in \Gamma$ *and* $\Gamma \vdash_\Sigma A = B : type$.
(*2*) *If* $\Gamma \vdash_\Sigma c : A$ *then* $\exists B.\ (c,\, B) \in \Sigma$ *and* $\Gamma \vdash_\Sigma A = B : type$.
(*3*) *If* $\Gamma \vdash_\Sigma M_1\ M_2 : A$ *then* $\exists x\ A_1\ A_2.\ \Gamma \vdash_\Sigma M_1 : \Pi x{:}A_2.\ A_1$ *and* $\Gamma \vdash_\Sigma M_2 : A_2$ *and* $\Gamma \vdash_\Sigma A = A_1[x{:=}M_2] : type$.
(*4*) *If* $\Gamma \vdash_\Sigma \lambda x{:}A.\ M : B$ *and* $x \# \Gamma$ *then* $\exists A'.\ \Gamma \vdash_\Sigma B = \Pi x{:}A.\ A' : type$ *and* $\Gamma \vdash_\Sigma A : type$ *and* $(x,\, A)::\Gamma \vdash_\Sigma M : A'$.

Next HP05 established some inversion and invertibility properties for definitional equality:

LEMMA 10 (EQUALITY INVERSION).
(*1*) *If* $\Gamma \vdash_\Sigma type = L : kind$ *then* $L = type$.
(*2*) *If* $\Gamma \vdash_\Sigma A = \Pi x{:}B_1.\ B_2 : type$ *and* $x \# \Gamma$ *then* $\exists A_1\ A_2.\ A = \Pi x{:}A_1.\ A_2$ *and* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x,\, A_1)::\Gamma \vdash_\Sigma A_2 = B_2 : type$.
(*3*) *If* $\Gamma \vdash_\Sigma K = \Pi x{:}B_1.\ L_2 : kind$ *and* $x \# \Gamma$ *then* $\exists A_1\ K_2.\ K = \Pi x{:}A_1.\ K_2$ *and* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x,\, A_1)::\Gamma \vdash_\Sigma K_2 = L_2 : kind$.

Finally, we can prove that the product type constructor is invertible, which is needed for soundness:

LEMMA 11 (PRODUCT INJECTIVITY). *Suppose* $x \# \Gamma$.
(*1*) *If* $\Gamma \vdash_\Sigma \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : type$ *then* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x,\, A_1)::\Gamma \vdash_\Sigma A_2 = B_2 : type$.
(*2*) *If* $\Gamma \vdash_\Sigma \Pi x{:}A.\ K = \Pi x{:}B.\ L : kind$ *then* $\Gamma \vdash_\Sigma A = B : type$ *and* $(x,\, A)::\Gamma \vdash_\Sigma K = L : kind$.

All the metatheoretic properties given above can be proved as stated in HP05 (appealing to Lem. 1 and 2 as necessary); however, since all of the definitional 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 as shown in Lem. 8 the validity judgment of terms implies the validity of the 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, some of HP05's rules given in Fig. 2 are formulated to explicitly include validity constraints such as $\Gamma \vdash_\Sigma A : \mathit{type}$ and $\Gamma \vdash_\Sigma K : \mathit{kind}$. After proving the above properties, however, we can show that these extra hypotheses are not needed, by establishing stronger forms of the rules:

Lemma 12 (Strong versions of rules). *The following rules are admissible:*

(1) $\dfrac{\Gamma \vdash_\Sigma M_1 : \Pi x{:}A_2.\ A_1 \quad \Gamma \vdash_\Sigma M_2 : A_2}{\Gamma \vdash_\Sigma M_1\ M_2 : A_1[x{:=}M_2]}$

(2) $\dfrac{\Gamma \vdash_\Sigma A : \Pi x{:}B.\ K \quad \Gamma \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma A\ M : K[x{:=}M]}$

(3) $\dfrac{(x,\ A_1){::}\Gamma \vdash_\Sigma M_2 = N_2 : A_2 \quad \Gamma \vdash_\Sigma M_1 = N_1 : A_1 \quad x \# \Gamma}{\Gamma \vdash_\Sigma (\lambda x{:}A_1.\ M_2)\ M_1 = N_2[x{:=}N_1] : A_2[x{:=}M_1]}$

3.2 Algorithmic equivalence

The main metatheoretic properties of algorithmic equivalence proved in Sec. 3 of HP05 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, for any object term y, provided that $(c,\ b) \in \Sigma$ since $(\lambda x{:}a.\ c)\ y \xrightarrow{\text{whr}} c$. 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 $(-) \xrightarrow{\text{whr}} (-)$ 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 13 (Erasure preservation).
(1) *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $A^- = B^-$.
(2) *If* $\Gamma \vdash_\Sigma K = L : \mathit{kind}$ *then* $K^- = L^-$.
(3) *If* $(x,\ A){::}\Gamma \vdash_\Sigma B : \mathit{type}$ *then* $B^- = B[x{:=}M]^-$
(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 14 (Erasure cancels substitution). *For any type family A, kind K, and substitution σ, we have*
(1) $A[\sigma]^- = A^-$
(2) $K[\sigma]^- = K^-$

In the proofs of symmetry and transitivity of the algorithmic judgments (Thm. 3 and Thm. 4), we also needed the following algorithmic erasure preservation lemma (it is omitted from HP05, but straightforward by induction):

LEMMA 15 (ALGORITHMIC 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 16 (DETERMINACY). *Suppose that* $\vdash \Sigma$ *ssig and* $\vdash \Delta$ *sctx.*
(1) *If* $M \stackrel{\text{whr}}{\longrightarrow} M'$ *and* $M \stackrel{\text{whr}}{\longrightarrow} M''$ *then* $M' = M''$.
(2) *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\nexists M'.\ M \stackrel{\text{whr}}{\longrightarrow} M'$.
(3) *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\nexists N'.\ N \stackrel{\text{whr}}{\longrightarrow} N'$.
(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 parts 4 and 5 in the proof of transitivity (Thm. 4). These properties are also later used in Thm. 13 in proving decidability of the algorithmic rules.

LEMMA 17 (GENERALIZED 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* $\vdash \Sigma$ *ssig 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

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

Fig. 5. Logical relation definition

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. 17) 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 article, but we found it helpful in the proof of soundness. We first need an (easily established) freshness-preservation property of weak head reduction.

Lemma 18 (Weak head reduction preserves freshness).
If $M \xrightarrow{\text{whr}} N$ *and* $x \# M$ *then* $x \# N$.

With this property in hand, strengthening for algorithmic and structural equivalence can be established by induction on the structure of judgments, making use of basic properties of freshness, valid contexts, and the previous lemma as necessary.

Lemma 19 (Strengthening of algorithmic equivalence). *Suppose that* $x \# (\Delta',\, M,\, N)$. *Then:*
(*1*) *If* $\Delta' @ [(x,\, \tau')] @ \Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *then* $\Delta' @ \Delta \vdash_\Sigma M \Leftrightarrow N : \tau$.
(*2*) *If* $\Delta' @ [(x,\, \tau')] @ \Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\Delta' @ \Delta \vdash_\Sigma M \leftrightarrow N : \tau$.

Proof. Straightforward induction on derivations, using properties of freshness. Lem. 18 is needed in the cases involving weak head reduction to maintain the freshness constraints needed for the induction hypothesis. □

3.3 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. 5. This kind of logical relation is called Kripke-style because it is indexed by a variable context Δ and in the case for function types and kinds, we quantify over all valid extensions to Δ when considering the argument terms M', N'.

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 20 (Logical relation 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 (Logically related terms are algorithmically equivalent). *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 21 (Closure under head expansion).
(1) *If* $M \xrightarrow{\text{whr}} M'$ *and* $\Delta \vdash_\Sigma M' = N \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$.
(2) *If* $N \xrightarrow{\text{whr}} N'$ *and* $\Delta \vdash_\Sigma M = N' \in [\![\tau]\!]$ *then* $\Delta \vdash_\Sigma M = N \in [\![\tau]\!]$.

Lemma 22 (Logical relation 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 23 (Logical relation 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. Establishing this (via the following lemma) required identifying and proving a number of standard properties of simultaneous substitutions. In contrast, reasoning about single substitutions sufficed almost everywhere else in the formalization.

Lemma 24. *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 25. *If* $\vdash_\Sigma \Gamma$ *ctx then* $\Gamma^- \vdash_{\Sigma^-} id_\Gamma = id_\Gamma \in [\![\Gamma^-]\!]$.

Theorem 6 (Definitionally equal terms are logically related).

(*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* $\Gamma \vdash_\Sigma K = L : \mathit{kind}$ *then* $\Gamma^- \vdash_{\Sigma^-} K \Leftrightarrow L : \mathit{kind}^-$.

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.

3.4 Soundness

Soundness of algorithmic equivalence 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 26 (Subject reduction). *Suppose* $M \xrightarrow{\text{whr}} M'$ *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:

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

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 27 (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

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

$$\frac{(a,\,\kappa) \in \Sigma \quad \vdash \Sigma\ \mathit{ssig} \quad \vdash \Delta\ \mathit{sctx}}{\Delta \vdash_\Sigma a \rightleftharpoons a : \kappa} \qquad \frac{\Delta \vdash_\Sigma A \rightleftharpoons B : \tau \to \kappa \quad \Delta \vdash_\Sigma M \Leftrightarrow N : \tau}{\Delta \vdash_\Sigma A\ M \rightleftharpoons B\ N : \kappa}$$

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

Fig. 6. Weak algorithmic type equivalence judgment

$A\ M$, 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. 27, we can complete the proof of the first part of Thm. 2 as described in HP05:

Lemma 28 (Soundness of algorithmic object equivalence). *Suppose $\Gamma \vdash_\Sigma M : A$ and $\Gamma \vdash_\Sigma N : A$. Then:*
(1) If $\Gamma^- \vdash_{\Sigma^-} M \Leftrightarrow N : A^-$ then $\Gamma \vdash_\Sigma M = N : A$.
(2) If $\Gamma^- \vdash_{\Sigma^-} M \leftrightarrow N : \tau$ then $\Gamma \vdash_\Sigma M = N : A$ and $\Gamma \vdash_\Sigma A = B : \mathit{type}$ and $A^- = \tau$ and $B^- = \tau$.

Second problem. The second problem is more serious. It 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:

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

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

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma A : \Pi x{:}C.\ K \\ \Gamma \vdash_\Sigma B : \Pi x{:}C.\ K \quad \Gamma \vdash_\Sigma C : \mathit{type} \quad (x,\,C)::\Gamma \vdash_\Sigma A\ x = B\ x : K \quad x \# \Gamma\end{array}}{\Gamma \vdash_\Sigma A = B : \Pi x{:}C.\ K}$$

that would permit 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 above 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[1], 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. 6 suffice. We can easily prove that these rules are sound with respect to definitional type equivalence:

[1]personal communication

LEMMA 29 (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* $\Gamma \vdash_\Sigma A = B : K$, $\Gamma \vdash_\Sigma K = L : \mathit{kind}$, $K^- = \kappa$ *and* $L^- = \kappa$.

PROOF. Similar to the proof of soundness of algorithmic and structural type equivalence from HP05. Requires soundness of object equivalence (Lem. 28). □

Moreover, we can prove completeness using a slightly modified logical relation: the type-level logical relation needs to be redefined as

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

The first two solutions however establish soundness only for variants of the definitions in HP05. In particular, the first shows that the original algorithmic rules are sound with respect to a stronger notion of definitional equality, while the second gives a correct modified algorithm for the original definitional rules. But neither solution tells us whether the *original* equivalence algorithm is sound with respect to the *original* definitional system in HP05; that is, whether the results in HP05 hold as stated. We resolved this question in the affirmative by finding a third solution that establishes soundness for the original definitions using the weak type equivalence algorithm introduced above.

Since we already established that weak type equivalence implies definitional equivalence (for well-formed terms), it suffices to show that the original algorithmic type equivalence judgments imply weak type equivalence. To do so, we need to show that weak type equivalence admits extensionality (Lem. 34 below). This is nontrivial: 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 30. *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 31. *If* $(y, \tau')::\Delta \vdash_0 M_0 : \tau$ *and* $y \# M_0$ *then* $\Delta \vdash_0 M_0 : \tau$.

Furthermore, if an applied variable is algorithmically or structurally equivalent to itself, then it is weakly well-formed:

LEMMA 32. *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. 30 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

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

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. 31 to show that $\Delta \vdash_0 M_0 : \tau_1 \to \tau_2$, which follows since $x \,\#\, M_0$. $\square$

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 33 (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. 19 here in the case for structural equivalence of type applications. $\square$

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

Lemma 34 (Extensionality of weak type equivalence). *If* $(x,\,\tau)::\Delta \vdash_\Sigma A\ x \rightleftharpoons B\ x : \kappa$ *and* $x \,\#\, (\Delta,\, A,\, B)$ *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. 33 we conclude $\Delta \vdash_\Sigma A \rightleftharpoons B : \tau \to \kappa$. $\square$

Lemma 35. *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. 34. $\square$

The proof of Thm. 2 is completed as follows.

Lemma 36 (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. 35 and 29. $\square$

Lemma 37 (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. 36 as necessary. $\square$

Thm. 2 follows immediately from Lem. 28, 36 and 37.

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

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

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

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

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

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

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad (x,\, A) \in \Gamma}{\Gamma \vdash_\Sigma x \Rightarrow A}$$

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad (c,\, A) \in \Sigma}{\Gamma \vdash_\Sigma c \Rightarrow A}$$

$$\frac{\Gamma \vdash_\Sigma M_1 \Rightarrow \Pi x{:}A_2{}'.\ A_1 \quad \Gamma \vdash_\Sigma M_2 \Rightarrow A_2 \quad \Gamma^- \vdash_{\Sigma^-} A_2 \Leftrightarrow A_2{}' : type^- \quad x \# \Gamma}{\Gamma \vdash_\Sigma M_1\ M_2 \Rightarrow A_1[x{:=}M_2]}$$

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

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

$$\frac{\vdash_\Sigma \Gamma \Rightarrow ctx \quad (a,\, K) \in \Sigma}{\Gamma \vdash_\Sigma a \Rightarrow K}$$

$$\frac{\Gamma \vdash_\Sigma A \Rightarrow \Pi x{:}A_2{}'.\ K_1 \quad \Gamma \vdash_\Sigma M \Rightarrow A_2 \quad \Gamma^- \vdash_{\Sigma^-} A_2 \Leftrightarrow A_2{}' : type^- \quad x \# \Gamma}{\Gamma \vdash_\Sigma A\ M \Rightarrow K_1[x{:=}M]}$$

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

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

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

Fig. 7. Algorithmic typechecking rules

3.5 Algorithmic typechecking

After the soundness and completeness proof, HP05 introduces an algorithmic version of the typechecking judgment, proves additional syntactic properties of definitional equivalence, sketches proofs of decidability, and discusses quasicanonical forms and adequacy of LF encodings of object languages. We will treat them in turn.

Algorithmic typechecking. The typechecking algorithm in HP05 traverses terms, types and kinds in a syntax-directed manner, using the algorithmic equivalence judgment in certain places. The definition of algorithmic typechecking in HP05 omitted explicit definitions of algorithmic signature and context validity. In our formalization, we added these (obvious) rules, as shown in Fig. 7. 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 type-checking is a (mostly) straightforward exercise using soundness and completeness of algorithmic equivalence and various syntactic properties:

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* $\Gamma \vdash_\Sigma A : K$.
(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* $\exists A'.\ \Gamma \vdash_\Sigma M \Rightarrow A'$ *and* $\Gamma \vdash_\Sigma A = A' : type$.
(4) If $\Gamma \vdash_\Sigma A : K$ *then* $\exists K'.\ \Gamma \vdash_\Sigma A \Rightarrow K'$ *and* $\Gamma \vdash_\Sigma K = K' : kind$.
(5) If $\Gamma \vdash_\Sigma K : kind$ *then* $\Gamma \vdash_\Sigma K \Rightarrow kind$.

3.6 Strengthening and strong extensionality

The strengthening property states that all of the definitional judgments are preserved by removing an unused variable from the context. We already established strengthening for the algorithmic equivalence judgments (Lem. 19). In order to establish strengthening for the algorithmic typechecking judgments, we need a stronger freshness lemma for algorithmic typechecking, which was not discussed in HP05:

LEMMA 38 (STRONG ALGORITHMIC FRESHNESS). *Let* $\Gamma = \Gamma_1 @ [(x, B)] @ \Gamma_2$.
(1) If $\Gamma \vdash_\Sigma M \Rightarrow A$ *and* $x \# (\Gamma_1, M)$ *then* $x \# A$.
(2) If $\Gamma \vdash_\Sigma A \Rightarrow K$ *and* $x \# (\Gamma_1, A)$ *then* $x \# K$.

We can now prove strengthening for algorithmic typechecking by induction on derivations:

THEOREM 9 (STRENGTHENING OF ALGORITHMIC TYPECHECKING).
Let $\Gamma = \Gamma_1 @ [(x, B)] @ \Gamma_2$.
(1) If $\vdash_\Sigma \Gamma \Rightarrow ctx$ *and* $x \# \Gamma_1$ *then* $\vdash_\Sigma \Gamma_1 @ \Gamma_2 \Rightarrow ctx$.
(2) If $\Gamma \vdash_\Sigma K \Rightarrow kind$ *and* $x \# (\Gamma_1, K)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma K \Rightarrow kind$.
(3) If $\Gamma \vdash_\Sigma A \Rightarrow K$ *and* $x \# (\Gamma_1, A)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma A \Rightarrow K$.
(4) If $\Gamma \vdash_\Sigma M \Rightarrow A$ *and* $x \# (\Gamma_1, M)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma M \Rightarrow A$.

PROOF. The proof is straightforward, using strengthening for algorithmic equivalence; parts (1–4) need to be proved in the order stated above since we need strengthening for contexts everywhere, we need strengthening for kinds to prove strengthening for types, and so on. Lem. 38 is needed in the cases for object and type application. □

Finally, we can prove strengthening for the definitional system.

THEOREM 10 (STRENGTHENING). *Let* $\Gamma = \Gamma_1 @ [(x, B)] @ \Gamma_2$.
(1) If $\vdash_\Sigma \Gamma\ ctx$ *and* $x \# \Gamma_1$ *then* $\vdash_\Sigma \Gamma_1 @ \Gamma_2\ ctx$.

(*2*) *If* $\Gamma \vdash_\Sigma K : kind$ *and* $x \# (\Gamma_1,\, K)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma K : kind$.
(*3*) *If* $\Gamma \vdash_\Sigma K = L : kind$ *and* $x \# (\Gamma_1,\, K,\, L)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma K = L : kind$.
(*4*) *If* $\Gamma \vdash_\Sigma A : K$ *and* $x \# (\Gamma_1,\, A)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma A : K$.
(*5*) *If* $\Gamma \vdash_\Sigma A = B : K$ *and* $x \# (\Gamma_1,\, A,\, B)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma A = B : K$.
(*6*) *If* $\Gamma \vdash_\Sigma M : A$ *and* $x \# (\Gamma_1,\, M)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma M : A$.
(*7*) *If* $\Gamma \vdash_\Sigma M = N : A$ *and* $x \# (\Gamma_1,\, M,\, N)$ *then* $\Gamma_1 @ \Gamma_2 \vdash_\Sigma M = N : A$.

PROOF. The proof follows the sketch in the article, using algorithmic strengthening and soundness and completeness of the algorithmic judgments, but some care is needed. Part 1 is straightforward, but we must prove the remaining cases in the specific order listed: first kind validity, then kind equivalence, then type validity, etc. The reason is that to prove strengthening for the equivalence judgments, we need strengthening for the corresponding validity judgments because of the validity side-conditions on Thm. 2. In turn, to prove strengthening for the object and type validity judgments, we need strengthening for type and kind equivalence respectively, because of the respective type and kind equivalence judgments in the conclusions of Thm. 8. Lem. 38 is needed in parts (4) and (6). □

HP05 also sketched a proof of admissibility of a stronger version of the extensionality rule which omits the well-formedness checks:

$$\frac{(x,\, A_1) {::} \Gamma \vdash_\Sigma M\, x = N\, x : A_2 \quad x \# (M,\, N)}{\Gamma \vdash_\Sigma M = N : \Pi x{:}A_1.\; A_2}$$

We were also able to verify this property. However, the short proof sketched in the article actually requires a substantial amount of work to formalize. The first two steps of their informal proof were as follows:
(1) By validity, we have $(x,\, A_1){::}\Gamma \vdash_\Sigma M\, x : A_2$.
(2) By inversion, we have $(x,\, A_1){::}\Gamma \vdash_\Sigma M : \Pi x{:}B_1.\; B_2$ and $(x,\, A_1){::}\Gamma \vdash_\Sigma x : B_1$
However, step (2) above does not follow immediately from the inversion lemmas proved earlier. In particular, we only know that M will have a type of the form $\Pi y{:}B_1.\; B_2$ for some y, B_1 and B_2 such that $(x,\, A_1){::}\Gamma \vdash_\Sigma M : \Pi y{:}B_1.\; B_2$ and $(x,\, A_1){::}\Gamma \vdash_\Sigma y : B_1$ and $(x,\, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2[y{:=}x] : type$. Moreover, in this case we cannot use the strong version of the inversion lemma to avoid this problem, because x is already in use in the context.

Although their proof looks rigorous and detailed, here Harper and Pfenning appear to be employ implicit "without loss of generality" reasoning about inversion and renaming that is not easy to formalize directly. Instead we needed to show carefully that:

LEMMA 39 (STRONG EXTENSIONAL VALIDITY).
If $(x,\, A_1){::}\Gamma \vdash_\Sigma M\, x : A_2$ *and* $x \# M$ *then* $\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\; A_2$.

PROOF. The proof proceeds by applying validity and inversion principles, as discussed above. One subtle freshness side-condition is the fact that x is fresh for $\Pi y{:}B_1.\; B_2$, and this is proved by translating to the algorithmic typechecking system and using Lem. 38. □

Strong extensionality then follows essentially as in HP05, using strong extensional validity to fill the gap identified above:

THEOREM 11 (STRONG EXTENSIONALITY).
If $(x,\, A_1)::\Gamma \vdash_\Sigma M\,x = N\,x : A_2$ *and* $x \,\#\, (M,\, N)$ *then* $\Gamma \vdash_\Sigma M = N : \Pi x{:}A_1.\ A_2$.

3.7 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 infer decidability of P simply by proving $P \vee \neg P$. Furthermore, given a relation R definable in Isabelle/HOL, it is non-obvious how to formalize the informal statement "R is decidable".

As a sanity check, we have shown that weak head reduction is strongly normalizing for well-formed terms. We write $M\Downarrow$ to indicate that M is strongly normalizing under weak head reduction. This proof uses techniques and definitions from the example formalization of strong normalization for the simply-typed lambda calculus in the Nominal Datatype Package.

THEOREM 12. *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. □

Turning now to the issue of formalizing decidability properties in Isabelle/HOL, we considered the following options:

Formalizing computability theory. It should be possible to define Turing machines (or some other universal model of computation) within Isabelle/HOL and derive enough of the theory of computation to be able to prove that the algorithmic equivalence and typechecking relations are decidable. It appears to be an open question how to formalize proofs of decidability in Isabelle/HOL, especially for algorithms over complex data structures such as nominal datatypes. Although this would probably be the most satisfying solution, it would also require a major additional formalization effort, including a great deal of work that is orthogonal to the issues addressed here. We therefore view fully formalizing decidability in this way as beyond the scope of this article. Instead, we consider other techniques that fall short of full formalization while providing some convincing evidence for decidability.

Bounded-height derivations. We could define height-bounded versions of the algorithmic typechecking relations and prove that there is a computable bound on the height needed to derive any derivable judgment in the system. That is, there exists a computable h such that for any inputs $x_1, \ldots, x_n$, there is a derivation of $J(x_1, \ldots, x_n)$ if and only if there is a derivation of height at most $h(x_1, \ldots, x_n)$.

This seems reasonable intuitively, but there are several problems. First, it is not obvious how to obtain a closed-form, recursively defined height bound for the number of steps needed for algorithmic equivalence for the same reason it is difficult to give an explicit termination measure for weak head normalization. Second, even if we could find such an h, this approach begs the question of how to prove that h is computable. It is clearly not enough to simply require that some h exists, because the Axiom of Choice can be used to define h nonconstructively. Finally, inductively defined judgments in Isabelle/HOL may themselves involve nonconstructive features, including equality at or quantification over infinite types, negation of undecidable properties, and choice operators. Although the definitions we have in mind do not use these facilities, there is no easy way to certify this within Isabelle/HOL.

Inductive definability. We have formalized what we believe is the essence of the decidability proof using the following methodology. For each inductively defined relation R we wish to prove decidable, possibly under some constraints P:

(1) Inductively define a complement relation R'.
(2) (Exclusion) Prove that $\neg$ (R *and* R').
(3) (Exhaustion) Prove that P implies $R \vee R'$.
(4) Observe (informally) that R and R' are recursively enumerable since they are defined inductively by rules without recourse to nonconstructive features such as negation or universal quantification in the hypotheses. Conclude (informally) that P implies R is both r.e. and co-r.e., hence decidable.

This approach exploits an intuitive connection between inductively definable predicates and recursively enumerable sets in step (4). It is important to note that this intuition is not rigorously formalized. We argue that this approach does force us to perform all of the case analysis that would be necessary in a proper decidability proof, but the only way to be certain of this is to fully formalize a substantial amount of computability theory in Isabelle/HOL, which as we have discussed above would be a major research contribution in its own right. Although we believe that this approach provides greater confidence in the decidability results compared to the proof sketches in HP05, there is still room for improvement. However, reasoning about decidability in Isabelle/HOL seems to be an open problem, involving several orthogonal issues. We leave the question of fully formalizing decidability to future work.

In the rest of this section, we describe the inductive definability argument for decidability in detail. We have introduced inductively defined complements for the algorithmic equivalence and typechecking judgments and verified exhaustivity and exclisivity for each of them. We have not verified step (4).

We call a formula R *quasidecidable* if both R and its negation are equivalent to inductively defined relations, as described above. This is an informal (and intensional) property; we have *not* defined quasidecidability explicitly in Isabelle/HOL. We were able to prove the following lemma, analogous to HP05's Lemma 6.1:

THEOREM 13 (QUASIDECIDABILITY OF ALGORITHMIC EQUIVALENCE).

(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.*

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

$$\frac{M \xrightarrow{\text{whr}} M' \quad \Delta \vdash_\Sigma M' \Leftrightarrow N : a^- \Uparrow \bar{\bar{O}}}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^- \Uparrow \bar{\bar{O}}} \qquad \frac{N \xrightarrow{\text{whr}} N' \quad \Delta \vdash_\Sigma M \Leftrightarrow N' : a^- \Uparrow \bar{\bar{O}}}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^- \Uparrow \bar{\bar{O}}}$$

$$\frac{\Delta \vdash_\Sigma M \leftrightarrow N : a^- \downarrow \bar{O}}{\Delta \vdash_\Sigma M \Leftrightarrow N : a^- \Uparrow \bar{O}} \qquad \frac{(x, \tau)::\Delta \vdash_\Sigma M\,x \Leftrightarrow N\,x : \tau' \Uparrow \bar{\bar{O}} \quad x \;\#\; (\Delta, M, N)}{\Delta \vdash_\Sigma M \Leftrightarrow N : \tau \to \tau' \Uparrow \lambda x.\bar{\bar{O}}}$$

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

$$\frac{(x, \tau) \in \Delta \quad \vdash \Sigma\ ssig \quad \vdash \Delta\ sctx}{\Delta \vdash_\Sigma x \leftrightarrow x : \tau \downarrow x} \qquad \frac{(c, \kappa) \in \Sigma \quad \vdash \Sigma\ ssig \quad \vdash \Delta\ sctx}{\Delta \vdash_\Sigma c \leftrightarrow c : \kappa \downarrow c}$$

$$\frac{\Delta \vdash_\Sigma M_1 \leftrightarrow N_1 : \tau_2 \to \tau_1 \downarrow \bar{O}_1 \quad \Delta \vdash_\Sigma M_2 \Leftrightarrow N_2 : \tau_2 \Uparrow \bar{\bar{O}}_2}{\Delta \vdash_\Sigma M_1\ M_2 \leftrightarrow N_1\ N_2 : \tau_1 \downarrow \bar{O}_1\ \bar{\bar{O}}_2}$$

Fig. 8. Algorithmic equivalence rules instrumented to produce quasicanonical forms.

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

LEMMA 40 (UNIQUENESS OF ALGORITHMIC TYPES).
(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* $K = K'$.

Equipped with Thm. 13 and the uniqueness lemma above, we can show a form of HP05's Theorem 6.2. Note that uses of Thm. 13 are safe because we always call the algorithmic equivalence judgments on terms that are well-formed, and hence (by Thm. 2) algorithmically equivalent to themselves.

THEOREM 14 (QUASIDECIDABILITY OF ALGORITHMIC TYPECHECKING).
(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,K, *if* $\vdash_\Sigma \Gamma \Rightarrow ctx$ *holds then* $\Gamma \vdash_\Sigma K \Rightarrow kind$ *is quasidecidable.*

3.8 Quasicanonical forms

Section 7 of HP05 discusses *quasicanonical forms* which can be used to study the *adequacy*, or correctness, of LF encodings. Quasicanonical forms are untyped λ-terms that correspond to the β-normal, η-long forms of well-typed LF terms. Quasicanonical forms $\bar{\bar{O}}$ and quasiatomic forms $\bar{O}$ are given by the grammar rules:

$$\bar{\bar{O}} ::= \bar{O} \mid \lambda x.\bar{\bar{O}} \qquad \bar{O} ::= x \mid c \mid \bar{O}\ \bar{\bar{O}}$$

HP05 introduces instrumented algorithmic equivalence judgments that construct quasicanonical forms for algorithmically and structurally equivalent terms, respectively. The rules are shown in Fig. 8.

It is straightforward to show that quasi-canonical and quasi-atomic forms exist and are unique (provided that Σ and Δ are valid).

Lemma 41 Properties of quasicanonical forms.
(1) *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$ *then* $\exists\, QC.\ \Delta \vdash_\Sigma M \Leftrightarrow N : \tau \Uparrow QC$.
(2) *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$ *then* $\exists\, QA.\ \Delta \vdash_\Sigma M \leftrightarrow N : \tau \downarrow QA$.
(3) *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau \Uparrow \bar{\bar{O}}$ *then* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau$.
(4) *If* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau \downarrow \bar{O}$ *then* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau$.
(5) *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau \Uparrow \bar{\bar{O}}$ *and* $M \xrightarrow{\text{whr}} M'$ *then* $\Delta \vdash_\Sigma M' \Leftrightarrow N : \tau \Uparrow \bar{\bar{O}}$.
(6) *If* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau \Uparrow \bar{\bar{O}}$ *and* $N \xrightarrow{\text{whr}} N'$ *then* $\Delta \vdash_\Sigma M \Leftrightarrow N' : \tau \Uparrow \bar{\bar{O}}$.

Theorem 15 Uniqueness of quasicanonical forms.
(1) *If* $\vdash \Delta$ *sctx and* $\vdash \Sigma$ *ssig and* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau \Uparrow \bar{\bar{O}}_1$ *and* $\Delta \vdash_\Sigma M \Leftrightarrow N : \tau \Uparrow \bar{\bar{O}}_2$ *then* $\bar{\bar{O}}_1 = \bar{\bar{O}}_2$.
(2) *If* $\vdash \Delta$ *sctx and* $\vdash \Sigma$ *ssig and* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau \downarrow \bar{O}_1$ *and* $\Delta \vdash_\Sigma M \leftrightarrow N : \tau' \downarrow \bar{O}_2$ *then* $\tau = \tau'$ *and* $\bar{O}_1 = \bar{O}_2$.

Proof. By induction on derivations, using Lem. 41(5,6) in the cases involving weak head reduction. □

The main result about these forms in HP05 is that well-formed LF terms can be recovered from quasicanonical forms and type information. To show this, we write $N \Uparrow \bar{\bar{O}}$ or $N \downarrow \bar{O}$ for the relations that relate objects N with their quasicanonical forms $\bar{\bar{O}}$ or quasiatomic forms $\bar{O}$, respectively, where the type-labels have been erased. (HP05 defined this notion as a partial function, which would be difficult to define with the Nominal Datatype Package at the time of writing.) These relations are defined as follows:

$$\frac{}{x \downarrow x} \qquad \frac{}{c \downarrow c} \qquad \frac{M \downarrow \bar{O} \quad N \Uparrow \bar{\bar{O}}}{(M\ N) \downarrow \bar{O}\ \bar{\bar{O}}} \qquad \frac{M \Uparrow \bar{\bar{O}}}{(\lambda x{:}A.\ M) \Uparrow \lambda x.\bar{\bar{O}}} \qquad \frac{M \downarrow \bar{O}}{M \Uparrow \bar{O}}$$

In the proof of the Quasicanonical Forms theorem (Theorem 7.1 of HP05) we found it necessary to prove several nontrivial auxiliary lemmas such as the admissibility of η-equivalence (which was not discussed in HP05):

Lemma 42 (Eta-equivalence). *If* $x \# \Gamma$ *and* $\Gamma \vdash_\Sigma M : \Pi x{:}A_1.\ A_2$ *then* $\Gamma \vdash_\Sigma M = \lambda x{:}A_1.\ M\ x : \Pi x{:}A_1.\ A_2$.

The following theorem is stated slightly differently than the corresponding theorem in HP05 (Theorem 7.1), but their version follows immediately from this version.

Theorem 16 (Quasicanonical forms).
(1) *If* $\Gamma^- \vdash_{\Sigma^-} M_1 \Leftrightarrow M_2 : A^- \Uparrow \bar{\bar{O}}$ *and* $\Gamma \vdash_\Sigma M_1 : A$ *and* $\Gamma \vdash_\Sigma M_2 : A$ *then* $\exists N.\ N \Uparrow \bar{\bar{O}}$ *and* $\Gamma \vdash_\Sigma N : A$ *and* $\Gamma \vdash_\Sigma M_1 = N : A$ *and* $\Gamma \vdash_\Sigma M_2 = N : A$.
(2) *If* $\Gamma^- \vdash_{\Sigma^-} M_1 \leftrightarrow M_2 : \tau \downarrow \bar{O}$ *and* $\Gamma \vdash_\Sigma M_1 : A_1$ *and* $\Gamma \vdash_\Sigma M_2 : A_2$ *then* $\Gamma \vdash_\Sigma A_1 = A_2 : type$ *and* $A_1{}^- = \tau$ *and* $A_2{}^- = \tau$ *and* $(\exists N.\ N \downarrow \bar{O}$ *and* $\Gamma \vdash_\Sigma N : A_1$ *and* $\Gamma \vdash_\Sigma M_1 = N : A_1$ *and* $\Gamma \vdash_\Sigma M_2 = N : A_2)$.

3.9 Adequacy

Conventionally, adequacy is the property that the terms of the object language are in a bijective correspondence with the well-formed LF terms of a given type,

$\boxed{\Gamma \vdash t \leftrightsquigarrow \bar{\bar{M}} : \iota}$

$$\frac{(x,\,\iota) \in \Gamma}{\Gamma \vdash x \leftrightsquigarrow x : \iota} \qquad \frac{\Gamma \vdash t_1 \leftrightsquigarrow \bar{\bar{M}}_1 : \iota \quad \Gamma \vdash t_2 \leftrightsquigarrow \bar{\bar{M}}_2 : \iota}{\Gamma \vdash f(t_1{,}t_2) \leftrightsquigarrow c_f\ \bar{\bar{M}}_1\ \bar{\bar{M}}_2 : \iota}$$

$\boxed{\Gamma \vdash \varphi \leftrightsquigarrow \bar{\bar{M}} : o}$

$$\frac{\Gamma \vdash t_1 \leftrightsquigarrow \bar{\bar{M}}_1 : \iota \quad \Gamma \vdash t_2 \leftrightsquigarrow \bar{\bar{M}}_2 : \iota}{\Gamma \vdash t_1 = t_2 \leftrightsquigarrow c_=\ \bar{\bar{M}}_1\ \bar{\bar{M}}_2 : o} \qquad \frac{\Gamma \vdash \varphi_1 \leftrightsquigarrow \bar{\bar{M}}_1 : o \quad \Gamma \vdash \varphi_2 \leftrightsquigarrow \bar{\bar{M}}_2 : o}{\Gamma \vdash \varphi_1 \wedge \varphi_2 \leftrightsquigarrow c_\wedge\ \bar{\bar{M}}_1\ \bar{\bar{M}}_2 : o}$$

$$\frac{(x,\,\iota){::}\Gamma \vdash \varphi \leftrightsquigarrow \bar{\bar{M}} : o \quad x \,\#\, \Gamma}{\Gamma \vdash \forall x.\varphi \leftrightsquigarrow c_\forall\ \lambda x.\bar{\bar{M}} : o}$$

Fig. 9. Adequacy translation

modulo LF equality. Moreover, the bijection should be *compositional*[2] in the sense that substitution for the object language is preserved and reflected by substitution in LF. The exact statement of the adequacy theorem for a given language depends on the language and its definition of substitution. To illustrate how quasicanonical forms could be used for reasoning about adequacy, HP05 introduces a small example language of first-order terms t and formulas φ, similar to the following:

$$t{,}u ::= x \mid f(t{,}u) \qquad \varphi{,}\psi ::= t = u \mid \varphi \wedge \psi \mid \forall x.\varphi$$

along with an appropriate LF signature Σ_{FO} with types ι for first-order terms, o for first-order formulas, and constants

$$\begin{array}{lcl@{\qquad}lcl} c_f & : & \iota \to \iota \to \iota & c_= & : & \iota \to \iota \to o \\ c_\wedge & : & o \to o \to o & c_\forall & : & (\iota \to o) \to o\ . \end{array}$$

HP05 then defines translation judgments $\Gamma \vdash t \leftrightsquigarrow M : \iota$ and $\Gamma \vdash \varphi \leftrightsquigarrow M : o$ relating LF terms M with first-order terms and formulas $t : \iota$ and $\varphi : o$. Note that unlike most other judgments in this article, the translations are *not* implicitly parametrized by a signature Σ since they only refer to constants from the fixed signature Σ_{FO}. The rules for the translation are shown in Fig. 9.

Harper and Pfenning then formulate the adequacy property for this language in their Theorem 7.2 as follows:

THEOREM 17 (ADEQUACY FOR SYNTAX OF FIRST-ORDER LOGIC). *Let Γ be a context of the form $x_1 : \iota, \ldots, x_n : \iota$ for some $n \geq 0$.*

(*1*) *The relation $\Gamma \vdash t \leftrightsquigarrow \bar{\bar{M}} : \iota$ is a compositional bijection between terms t of first-order logic over variables $x_1, \ldots, x_n$ and quasi-canonical forms $\bar{\bar{M}}$ of type ι relative to Γ.*

(*2*) *The relation $\Gamma \vdash \varphi \leftrightsquigarrow \bar{\bar{M}} : o$ is a compositional bijection between formulas t of first-order logic over variables $x_1, \ldots, x_n$ and quasi-canonical forms $\bar{\bar{M}}$ of type o relative to Γ.*

Their proof sketch involves first showing that (for all appropriate Γ) the translations are bijections, and then proving compositionality by induction over the

[2]This term is used in HP05 without being defined, but this is the definition used in other papers which discuss adequacy [Harper et al. 1993; Pfenning 2001].

structure of terms and formulas.

Unfortunately, the statement of this theorem is ambiguous or at least incomplete. The reason is that Harper and Pfenning do not explicitly define what it means for a bijection to be compositional. Even assuming the standard definition of compositionality as substitution preservation, HP05 did not provide a definition of substitution for quasicanonical forms.

If we wish to substitute a quasicanonical form for a variable y in another quasicanonical form, the result is not always quasicanonical. For example, if we substitute $\lambda x.M$ for y in $y\ N$, we get $(\lambda x.M)\ N$, which is not quasicanonical. This illustrates that quasicanonical forms are not closed under substitution of quasicanonical forms for variables, because variables are quasiatomic forms and substituting a λ-expression for a variable may introduce β-redexes.

It has been observed elsewhere (apparently first by Watkins et al. [2003]) that substitution can be defined for well-formed quasicanonical expressions in a *hereditary* way that recursively renormalizes any β-redexes introduced by substitution. Harper and Licata [2007] have shown how this idea can be used as the basis for a variant of LF called *Canonical LF* in which all expressions are maintained in canonical form.

In our initial formalization (reported in [Urban et al. 2008]) we misinterpreted the definition of the translation slightly by defining the adequacy translations to relate first-order terms and formulas to *quasiatomic* forms. It is easy to define substitution of quasiatomic forms for variables since no reduction can be introduced in doing so. Consequently, we were able to prove Theorem 7.2 with the word "quasicanonical" replaced by "quasiatomic". However, even with this modification, the formal proof is not as easy as the sketch in HP05 suggests; for example, we needed to prove weakening, exchange, and substitution lemmas for the translation judgment in order to establish compositionality.

After we discovered and corrected the mismatch between our definition and the original translation, we were still able to prove that the translations are bijections. To establish compositionality, we also formalized hereditary substitution (using a simple form of Harper and Licata's definition) and showed that the translation maps object-language substitution to hereditary substitution.

Formalizing HP05's Theorem 7.2 thus appears to require either changing their translation or introducing hereditary substitution, a nontrivial concept that was not mentioned in HP05. The Canonical LF approach now appears to be the preferred starting point for research on extensions to LF. Developing a full and satisfying formalization of hereditary substitutions and adequacy properties (and relating HP05's version of LF to Harper and Licata's development of Canonical LF [2007]) would be a significant independent undertaking. Therefore, we prefer to leave further study of adequacy based on hereditary substitution for future work.

4. CODE GENERATION

Since type checking in LF can be part of the trusted code base of proof-carrying code, Appel et al. [2003] were very careful to implement it as cleanly as possible and in as few lines of code as possible. Their motivation was that a small and clean implementation can be manually inspected and hence can be made robust against, for example, Thompson-style attacks [Thompson 1984]. For this they explicitly set out to minimize the number of library functions they have to trust in order for their implementation to be correct. However, Appel mentioned[3] also that he had to trust the correctness of the type-checking algorithm itself, since he was unable to ascertain whether the algorithm is sound and complete w.r.t. its specification. In this paper we have formally proved that both the equivalence checking and type-checking algorithms from HP05 are sound and complete. Consequently, we can remove this aspect from our "trusted code base". In this section we like to show that our formalisation is also helpful for producing an executable ML-implementation of the type-checking algorithm.

Isabelle/HOL contains a code generator implemented by Berghofer and Nipkow [2002] which can translate automatically inductive definitions into executable pure ML-code. To be able to use this code generator, however, we need to invest some further work. The present version of this code generator can only deal with rules involving datatypes, not *nominal* datatypes. To surmount this problem we translate our nominal representation of kinds, types and terms into a locally nameless representation [McKinna and Pollack 1999; Aydemir et al. 2008], which can be implemented in Isabelle/HOL as datatype. For the LF-syntax this gives rise to the definition:

$$\begin{array}{lcl} \textit{Kinds} & ::= & \textit{type} \mid \Pi A.\ K \\ \textit{Types} & ::= & a \mid \Pi A_1.\ A_2 \mid A\ M \\ \textit{Objects} & ::= & c \mid x \mid n \mid \lambda A.\ M \mid M_1\ M_2 \end{array}$$

where terms contain de Bruijn indices n for bound variables [de Bruijn 1972]. In comparison with "pure" de Bruijn representations, in the locally nameless representation free variables still have names. This means we can continue using our implementation of signatures and contexts in judgments. With a "pure" de Bruijn representation, contexts would need to be referenced by numbers and positions.

While the locally nameless representation is straightforward to implement in Isabelle/HOL, the translations between the nominal and locally nameless representation involve quite a lot of formalisation work. First we have to define a well-formedness predicate that ensures that there are no loose de Bruijn indices. We also need three substitution operations, namely substituting (well-formed) terms for free variables, written $(-)[x := M]$, substituting terms for de Bruijn indices, written $(-)[n := M]$, and substituting de Bruijn indices for variables, written $(-)[x := n]$. In the latter we have to increase the de Bruijn index whenever the substitution moves under a binder. Also the translation functions between the nominal and locally nameless representations turned out to be non-trivial to work with. In one direction the translation is a partial function and only total over well-formed locally nameless terms. In the other direction we use a translation depending on an explicit list of variables. The idea is to push a variable onto the list whenever the translation goes under a λ- or a Π-abstraction. Now the de Bruijn index for a variable occurrence is the position of the variable in this list. The translation, written $|-|_{xs}$, can be formally defined as

[3] personal communication

$$
\begin{array}{rcll}
|type|_{xs} & = & type & \\
|\Pi x{:}A.\ K|_{xs} & = & \Pi |A|_{xs}.\ |K|_{(x::xs)} & \text{provided } x \# xs \\
|a|_{xs} & = & a & \\
|A\ M|_{xs} & = & |A|_{xs}\ |M|_{xs} & \\
|\Pi x{:}A_1.\ A_2|_{xs} & = & \Pi |A_1|_{xs}.\ |A_2|_{(x::xs)} & \text{provided } x \# xs \\
|c|_{xs} & = & c & \\
|x|_{xs} & = & \mathit{index\ x\ xs\ 0} & \\
|M\ N|_{xs} & = & |M|_{xs}\ |N|_{xs} & \\
|\lambda x{:}A.\ M|_{xs} & = & \lambda |A|_{xs}.\ |M|_{(x::xs)} & \text{provided } x \# xs
\end{array}
$$

where the variable case is defined in terms of the auxiliary function *index x xs n*:

$$
\begin{array}{l}
\mathit{index}\ x\ []\ n = x \\
\mathit{index}\ x\ (y{::}ys)\ n = (\mathit{if}\ x = y\ \mathit{then}\ n\ \mathit{else}\ \mathit{index}\ x\ ys\ (\mathit{Suc}\ n))
\end{array}
$$

The problem with this definition arises from the fact that inductions need to be appropriately generalised in order to take the potentially growing list of variables into account. This is sometimes easy to do, but sometimes we needed a lot of ingenuity to find the right lemmas to get inductions through.

Having translated all our terms in to the locally nameless representation, we solved th technical problem with the code generator in Isabelle/HOL. However, there is a further problem that needs to solved: the algorithms specified so far are not yet concrete enough to be translated directly into runnable ML-code. For this consider again the algorithmic equivalence rule

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

from Fig. 4. This rule decides the equivalence between the terms M and N having function type. When read bottom-up, it states that we need to introduce a variable x (any will do) that is fresh for Δ, M and N. ML does not have any built-in facilities for choosing such a fresh name (unlike, for example, FreshML by Shinwell et al. [2003]). This means for an ML-implementation of type and equivalence checking that we need to make explicit which fresh name should be chosen. An obvious choice is to inspect all free variables occurring in Δ, M and N, and produce a variable with a higher index. In our case, it suffices to compute the maximum index of all variables in scope and increase by one to obtain a fresh variable index. We are able to compute this index because variables in the Nominal Datatype Package have a natural number as index and thus can be ordered. This allows us to formulate algorithmic equivalence rules as follows

$$
\frac{(x,\ \tau_1){::}\Delta\ {}_{ln}{\vdash_{\Sigma}} M\ x \Leftrightarrow N\ x : \tau_2 \quad x = \mathit{maxi}\ (\mathit{fv}\ \Delta\ @\mathit{fv}\ M\ @\mathit{fv}\ N)}{\Delta\ {}_{ln}{\vdash_{\Sigma}} M \Leftrightarrow N : \tau_1 \to \tau_2}
$$

where *fv* is a polymorphic function producing a list of free variables of a term or context, and the function *maxi* scans through a list of variables and returns the highest variable increased by one.

In Fig. 10 we show the rules for type checking in the locally nameless representation and with the explicit choice of fresh variables. The locally nameless variants of these judgments are marked by the subscript ln. We omit the locally nameless versions of the algorithmic equivalence rules but they are similar. The functions $fi\ (-)$ and $fv\ (-)$ calculate the free identifiers and free variables of their arguments, respectively.

It is important to note that it would be extremely inconvenient to build the concrete choice for a fresh variable into the rules that are used in the soundness and completeness proofs described in the earlier sections. The reason is that several of the proofs would *not* go through as stated in HP05 since the choice is not fresh enough for all entities considered in some lemmas (an example is the weakening property, where the variable x is assumed to be not just fresh for Δ, M and N, but also for a larger context Δ'). It is however relatively straightforward to show the equivalence (i.e., they derive the same judgments, modulo translation) between the original rules and the rules with the concrete choice for fresh variables. We can show:

LEMMA 43 EQUIVALENCE.
(1) $\vdash \Sigma \Rightarrow sig$ *if and only if* ${}_{ln}\vdash |\Sigma|_{[]} \Rightarrow sig$.
(2) $\vdash_{\Sigma} \Gamma \Rightarrow ctx$ *if and only if* ${}_{ln}\vdash_{|\Sigma|_{[]}} |\Gamma|_{[]} \Rightarrow ctx$.
(3) $\Gamma \vdash_{\Sigma} M \Rightarrow A$ *if and only if* $|\Gamma|_{[]}\ {}_{ln}\vdash_{|\Sigma|_{[]}} |M|_{[]} \Rightarrow |A|_{[]}$.
(4) $\Gamma \vdash_{\Sigma} A \Rightarrow K$ *if and only if* $|\Gamma|_{[]}\ {}_{ln}\vdash_{|\Sigma|_{[]}} |A|_{[]} \Rightarrow |K|_{[]}$.
(5) $\Gamma \vdash_{\Sigma} K \Rightarrow kind$ *if and only if* $|\Gamma|_{[]}\ {}_{ln}\vdash_{|\Sigma|_{[]}} |K|_{[]} \Rightarrow kind$.

From the rules in Fig. 10 the code generator of Isabelle/HOL can generate ML-code. Of course the correctness of this code depends on the correctness of the generator. However it is relatively easy to inspect the generated ML-code and we are confident that it implements correctly the inductive definitions that have been proved to be sound and complete w.r.t. specification. We have used the extracted ML-code to type-check several LF example signatures.

5. 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 roughly three months' experience with these tools prior to starting the formalization. We estimate that the total effort involved in conducting the formalizations in Sec. 3 was at most three person-months. We worked on the code generation part intermittently therefore de not have any information about timing. Although there is still 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 experienced users.

It took approximately six person-weeks to formalize everything up to the soundness proof (including pondering why the omitted case for type extensionality 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. Rechecking the proof on paper would have meant reviewing approximately 31 pages of proofs. Subsequently we checked the validity of a solution suggested by Harper and found another solution for the problem. As a practical matter, the ability to

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

$$\frac{}{{}_{ln}\vdash\ [] \Rightarrow sig} \qquad \frac{{}_{ln}\vdash\ \Sigma \Rightarrow sig \quad []\ {}_{ln}\vdash_{\Sigma}\ A \Rightarrow type \quad c \notin \mathit{fi}\ \Sigma}{{}_{ln}\vdash\ (c,\ A)::\Sigma \Rightarrow sig}$$

$$\frac{{}_{ln}\vdash\ \Sigma \Rightarrow sig \quad []\ {}_{ln}\vdash_{\Sigma}\ K \Rightarrow kind \quad a \notin \mathit{fi}\ \Sigma}{{}_{ln}\vdash\ (a,\ K)::\Sigma \Rightarrow sig}$$

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

$$\frac{{}_{ln}\vdash\ \Sigma \Rightarrow sig}{{}_{ln}\vdash_{\Sigma}\ [] \Rightarrow ctx} \qquad \frac{{}_{ln}\vdash_{\Sigma}\ \Gamma \Rightarrow ctx \quad \Gamma\ {}_{ln}\vdash_{\Sigma}\ A \Rightarrow type \quad x \notin \mathit{fv}\ \Gamma}{{}_{ln}\vdash_{\Sigma}\ (x,\ A)::\Gamma \Rightarrow ctx}$$

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

$$\frac{{}_{ln}\vdash_{\Sigma}\ \Gamma \Rightarrow ctx \quad (x,\ A) \in \Gamma}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ x \Rightarrow A} \qquad \frac{{}_{ln}\vdash_{\Sigma}\ \Gamma \Rightarrow ctx \quad (c,\ A) \in \Sigma}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ c \Rightarrow A}$$

$$\frac{\Gamma\ {}_{ln}\vdash_{\Sigma}\ M_1 \Rightarrow \Pi A_2{}'.\ A_1 \quad \Gamma\ {}_{ln}\vdash_{\Sigma}\ M_2 \Rightarrow A_2 \quad \Gamma^-\ {}_{ln}\vdash_{\Sigma^-}\ A_2 \Leftrightarrow A_2{}' : type^-}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ M_1\ M_2 \Rightarrow A_1[0 := M_2]}$$

$$\frac{\begin{array}{c}\Gamma\ {}_{ln}\vdash_{\Sigma}\ A_1 \Rightarrow type\\ (x,\ A_1)::\Gamma\ {}_{ln}\vdash_{\Sigma}\ M_2[0 := x] \Rightarrow A_2 \quad x = maxi\ (\mathit{fv}\ \Gamma\ @\mathit{fv}\ M_2\ @\mathit{fv}\ A_1) \quad A_2{}' = A_2[x := 0]\end{array}}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ \lambda A_1.\ M_2 \Rightarrow \Pi A_1.\ A_2{}'}$$

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

$$\frac{{}_{ln}\vdash_{\Sigma}\ \Gamma \Rightarrow ctx \quad (a,\ K) \in \Sigma}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ a \Rightarrow K}$$

$$\frac{\Gamma\ {}_{ln}\vdash_{\Sigma}\ A \Rightarrow \Pi A_2{}'.\ K_1 \quad \Gamma\ {}_{ln}\vdash_{\Sigma}\ M \Rightarrow A_2 \quad \Gamma^-\ {}_{ln}\vdash_{\Sigma^-}\ A_2 \Leftrightarrow A_2{}' : type^-}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ A\ M \Rightarrow K_1[0 := M]}$$

$$\frac{\Gamma\ {}_{ln}\vdash_{\Sigma}\ A_1 \Rightarrow type \quad (x,\ A_1)::\Gamma\ {}_{ln}\vdash_{\Sigma}\ A_2[0 := x] \Rightarrow type \quad x = maxi\ (\mathit{fv}\ \Gamma\ @\mathit{fv}\ A_1\ @\mathit{fv}\ A_2)}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ \Pi A_1.\ A_2 \Rightarrow type}$$

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

$$\frac{{}_{ln}\vdash_{\Sigma}\ \Gamma \Rightarrow ctx}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ type \Rightarrow kind}$$

$$\frac{\Gamma\ {}_{ln}\vdash_{\Sigma}\ A \Rightarrow type \quad (x,\ A)::\Gamma\ {}_{ln}\vdash_{\Sigma}\ K[0 := x] \Rightarrow kind \quad x = maxi\ (\mathit{fv}\ \Gamma\ @\mathit{fv}\ A\ @\mathit{fv}\ K)}{\Gamma\ {}_{ln}\vdash_{\Sigma}\ \Pi A.\ K \Rightarrow kind}$$

Fig. 10. Algorithmic typechecking rules used for generating executable code.

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 [Urban and Zhu 2008].

Our formalization using nominal datatypes follows that given in HP05 very closely. Our experience suggests that nominal techniques can be used to both state and prove results almost exactly as they are presented on paper—no other currently available technique appears able to do this. To illustrate this point, we have prepared this paper using Isabelle's documentation facilities [Nipkow et al. 2002]. Most

Table I. Summary of the formalization

Theory	Description	Size (bytes)	Lines	Lemmas
`LF`	Syntax and definitional judgments of LF	125,975	2,631	103
`Erasure`	Simple types and kinds, erasure	14,860	463	35
`PairOrdering`	Pair ordering used for transitivity	962	29	3
`EquivAlg`	Algorithmic equivalence judgments and properties	47,480	1,015	46
`Completeness`	Logical relation, completeness proof	54,575	778	22
`WeakEquivAlg`	Weak algorithmic typechecking	9,373	219	7
`Soundness`	Subject reduction, soundness proofs	31,235	562	8
`TypeAlg`	Algorithmic typechecking	13,139	244	5
`Decidability`	Quasidecidability	104,939	2,087	50
`Strengthening`	Strengthening and strong extensionality	28,940	591	15
`Canonical`	Quasicanonical forms	27,702	556	13
`Adequacy`	Adequacy example	29,777	736	45
`LocallyN`	Translation to locally nameless syntax	179,148	4,674	223
Total		668,105	14585	575

lemmas, theorems, and definitions have been generated directly from the formalization (the main exceptions are the quasidecidability and adequacy properties, which are paraphrased).

In Table I, we report some simple metrics about our formalization such as the sizes, number of lines of text, and number of lemmas in each theory in the main formalization. As Table I shows, the core `LF` theory accounts for about 20% 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 each be handled individually. The `Decidability` theory accounts for another 15%; the quasidecidability proofs are verbose but largely straightforward. The `LocallyN` theory proves that the nominal datatypes version of LF is equivalent to a locally nameless formulation; this accounts for about 25% of the development. The efort involved in this part was therefore quite substantial: it can be explained by the lack of automatic infrastructure for the locally nameless representation of binders in Isabelle/HOL. But also by the inherent subtleties when working with this represenation. A number of lemmas need to be carfully stated, and in a few cases in rather non-intuitive ways. The remaining theories account for at most 5–10% of the formalization each; the `WeakAlgorithm` theory defines the weak algorithmic equivalence judgment and proves the additional 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 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 their use may not be persuasive to readers unfamiliar with nominal logic and abstract syntax. Thus, a skeptical reader might ask whether these representations, definitions and reasoning principles are really *correct*; that is, whether they are equivalent to the definitions in HP05, as formalized using some more 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 [Harper et al. 1993], due to the potential problems involved in reasoning about higher-order terms modulo alpha, beta and eta-equivalence.

Adequacy is also important for nominal techniques and deserves further study. We believe that the techniques explored in existing work on the semantics of nominal abstract syntax and its implementation in the Nominal Datatype Package [Gabbay and Pitts 2002; Pitts 2003; Cheney 2006; Pitts 2006; Urban 2008] suffices for informally judging the correctness of our formalization. There has also been some prior work on formalizing adequacy results for nominal datatypes via isomorphisms. Urban [2008] proves a bijective correspondence between nominal datatypes and a conventional named implementation of the λ-calculus modulo α-equivalence. Norrish and Vestergaard [2007] have formalized isomorphisms between nominal and de Bruijn representations, and they provide further citations to several other isomorphism results. Our proof of equivalence to a locally nameless representation described in Sec. 4 also gives evidence for the correctness of the nominal datatype representation.

In any case, whether or not nominal datatypes in Isabelle/HOL really capture our informal intuitions about abstract syntax with binding, our formalization has exposed some subtle issues which make sense in the context of LF.

6. RELATED AND FUTURE WORK

McKinna and Pollack [1999]'s LEGO formalization of Pure Type Systems is probably the most extensive formalization of a dependent type theory in a theorem prover. Their formalization introduced the locally nameless variant of de Bruijn's name-free approach [de Bruijn 1972] and considered primarily syntactic properties of pure type systems with β-equivalence, including a proof of strengthening. Pollack [1995] subsequently verified the partial correctness of typechecking algorithms for certain classes of Pure Type Systems including LF.

Aydemir et al. [2008] have developed a methodology for formalizing metatheory in Coq using the locally nameless representation to manage binding, and using cofinite quantification to handle fresh names. Chlipala's *parametric higher-order abstract syntax* is another recently developed technique for reasoning about abstract syntax in Coq, and has been applied to good effect in reasoning about compiler transformations [Chlipala 2008]. Westbrook et al. [2009] are developing CINIC, a variant of Coq that provides built-in support for nominal abstract syntax (generalizing a simple nominal type theory developed by Cheney [2009]). Using these methodologies to formalize the results in this article would provide a useful comparison of these approaches, particularly concerning decidability proofs, which ought to be easier in constructive logics.

Algorithms for equivalence and canonicalization for dependent type theories have been studied by several authors. Prior work on equivalence checking for LF has focused on first checking well-formedness with respect to simple types, then β- or $\beta\eta$-normalizing; these approaches are discussed in detail by Harper and Pfenning [2005]. Coquand's algorithm [1991] is similar to Harper and Pfenning's but operates on untyped terms. Goguen's approach [2005b] involves first type-directed η-expansion and then β-normalization, and relies on standard properties such as the Church-Rosser theorem, strong normalization of β-reduction and strengthening. Goguen [2005a] extends this proof technique to show termination of Coquand's and Harper and Pfenning's algorithms, and gives a terminating type-directed algorithm for checking $\beta\eta$-equivalence in System F. It may be interesting to verify these algorithms and proofs and compare with Harper and Pfenning's proof.

We chose to formalize the approach taken by Harper and Pfenning [2005] because it is the most recent and most detailed among the above developments. Another reason is that the quality standards in the LF-community are very high, and peer-reviewed work is generally trusted. Appel, for example, told us[4] that he trusts the implementation of a type-checker for LF presented by Appel et al. [2003], 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 meta-reasoning about LF specifications, following Harper and Licata's detailed informal development of Canonical LF [2007]. Doing so could make it possible to extract Isabelle/HOL theorems from Twelf proofs, but as discussed earlier, formalizing Canonical LF, hereditary substitutions, and the rest of Harper and Licata's work appears to be a substantial challenge.

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 by Harper and Pfenning [2005, 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.

7. CONCLUSIONS

LF is an extremely convenient tool for defining logics and other calculi involving binding syntax. 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.

[4] personal communication

For our formalization we had the advantage of working from Harper and Pfenning's carefully-written informal proof, which withstood rigorous mechanical formalization rather well. Still we found in this informal proof one gap and numerous 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 rapidly. By replaying a modified formalization in a theorem prover one can immediately focus on places where the proof fails and attempt to repair them rather than re-checking the many cases that are unchanged. This capability was essential in fixing the soundness proof, and it illustrates one of the distinctive advantages of performing such a formalization. Had we attempted to repair the gap using only the paper proof, experimenting with different solutions would have required manually re-checking the roughly 31 pages of paper proofs for each change.

Our formalization is not an end in itself but also provides a foundation for further study in several directions. Researchers developing extensions to LF may find our formalization useful as a starting point for verifying the metatheory of such extensions. We plan to further investigate hereditary substitutions and adequacy proofs in LF and Canonical LF. More ambitiously, we contemplate formalizing the meaning and correctness of metatheoretic reasoning about LF specifications (as provided by the Twelf system) inside Isabelle/HOL, and extracting Isabelle/HOL theorems from Twelf proofs.

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

APPENDIX

2. FULL STATEMENTS OF SYNTACTIC RESULTS

LEMMA 44 (FRESHNESS).

(*1*) *If* $\vdash \Sigma$ *sig then* $x \# \Sigma$.
(*2*) *If* $\vdash_\Sigma \Gamma$ *ctx then* $x \# \Sigma$.
(*3*) *If* $\Gamma \vdash_\Sigma M : A$ *and* $x \# \Gamma$ *then* $x \# M$ *and* $x \# A$.
(*4*) *If* $\Gamma \vdash_\Sigma A : K$ *and* $x \# \Gamma$ *then* $x \# A$ *and* $x \# K$.
(*5*) *If* $\Gamma \vdash_\Sigma K : kind$ *and* $x \# \Gamma$ *then* $x \# K$.
(*6*) *If* $\Gamma \vdash_\Sigma M = N : A$ *and* $x \# \Gamma$ *then* $x \# M$ *and* $x \# N$ *and* $x \# A$.
(*7*) *If* $\Gamma \vdash_\Sigma A = B : K$ *and* $x \# \Gamma$ *then* $x \# A$ *and* $x \# B$ *and* $x \# K$.
(*8*) *If* $\Gamma \vdash_\Sigma K = L : kind$ *and* $x \# \Gamma$ *then* $x \# K$ *and* $x \# L$.

LEMMA 45 (IMPLICIT VALIDITY).

(*1*) *If* $\vdash_\Sigma \Gamma$ *ctx then* $\vdash \Sigma$ *sig*.
(*2*) *If* $\Gamma \vdash_\Sigma M : A$ *then* $\vdash_\Sigma \Gamma$ *ctx and* $\vdash \Sigma$ *sig*.
(*3*) *If* $\Gamma \vdash_\Sigma A : K$ *then* $\vdash_\Sigma \Gamma$ *ctx and* $\vdash \Sigma$ *sig*.

(4) *If* $\Gamma \vdash_\Sigma K : kind$ *then* $\vdash_\Sigma \Gamma$ *ctx and* $\vdash \Sigma$ *sig.*
(5) *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\vdash_\Sigma \Gamma$ *ctx and* $\vdash \Sigma$ *sig.*
(6) *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\vdash_\Sigma \Gamma$ *ctx and* $\vdash \Sigma$ *sig.*
(7) *If* $\Gamma \vdash_\Sigma K = L : kind$ *then* $\vdash_\Sigma \Gamma$ *ctx and* $\vdash \Sigma$ *sig.*

LEMMA 46 (IMPLICIT VALIDITY). *If* $\Gamma \vdash_\Sigma M : A$ *then* $\vdash \Sigma$ *sig and* $\vdash_\Sigma \Gamma$ *ctx.*

LEMMA 47 (WEAKENING). *Suppose* $\vdash_\Sigma \Gamma_2$ *ctx and* $\Gamma_1 \subseteq \Gamma_2$.
(1) *If* $\Gamma_1 \vdash_\Sigma M : A$ *then* $\Gamma_2 \vdash_\Sigma M : A$.
(2) *If* $\Gamma_1 \vdash_\Sigma A : K$ *then* $\Gamma_2 \vdash_\Sigma A : K$.
(3) *If* $\Gamma_1 \vdash_\Sigma K : kind$ *then* $\Gamma_2 \vdash_\Sigma K : kind$.
(4) *If* $\Gamma_1 \vdash_\Sigma M = N : A$ *then* $\Gamma_2 \vdash_\Sigma M = N : A$.
(5) *If* $\Gamma_1 \vdash_\Sigma A = B : K$ *then* $\Gamma_2 \vdash_\Sigma A = B : K$.
(6) *If* $\Gamma_1 \vdash_\Sigma K = L : kind$ *then* $\Gamma_2 \vdash_\Sigma K = L : kind$.

LEMMA 48 (SUBSTITUTION). *Suppose* $\Gamma_2 \vdash_\Sigma P : C$ *and let* $\Gamma = \Gamma_1 \mathbin{@} [(y,\ C)] \mathbin{@} \Gamma_2$.
(1) *If* $\vdash_\Sigma \Gamma$ *ctx then* $\vdash_\Sigma \Gamma_1[y{:=}P] \mathbin{@} \Gamma_2$ *ctx.*
(2) *If* $\Gamma \vdash_\Sigma M : B$ *then* $\Gamma_1[y{:=}P] \mathbin{@} \Gamma_2 \vdash_\Sigma M[y{:=}P] : B[y{:=}P]$.
(3) *If* $\Gamma \vdash_\Sigma B : K$ *then* $\Gamma_1[y{:=}P] \mathbin{@} \Gamma_2 \vdash_\Sigma B[y{:=}P] : K[y{:=}P]$.
(4) *If* $\Gamma \vdash_\Sigma K : kind$ *then* $\Gamma_1[y{:=}P] \mathbin{@} \Gamma_2 \vdash_\Sigma K[y{:=}P] : kind$.
(5) *If* $\Gamma \vdash_\Sigma M = N : A$ *then* $\Gamma_1[y{:=}P] \mathbin{@} \Gamma_2 \vdash_\Sigma M[y{:=}P] = N[y{:=}P] : A[y{:=}P]$.
(6) *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\Gamma_1[y{:=}P] \mathbin{@} \Gamma_2 \vdash_\Sigma A[y{:=}P] = B[y{:=}P] : K[y{:=}P]$.
(7) *If* $\Gamma \vdash_\Sigma K = L : kind$ *then* $\Gamma_1[y{:=}P] \mathbin{@} \Gamma_2 \vdash_\Sigma K[y{:=}P] = L[y{:=}P] : kind$.

LEMMA 49 (CONTEXT CONVERSION). *Assume that* $\Gamma \vdash_\Sigma B : type$ *and* $\Gamma \vdash_\Sigma A = B : type$. *Then:*
(1) *If* $(x,\ A){::}\Gamma \vdash_\Sigma M : C$ *then* $(x,\ B){::}\Gamma \vdash_\Sigma M : C$
(2) *If* $(x,\ A){::}\Gamma \vdash_\Sigma C : K$ *then* $(x,\ B){::}\Gamma \vdash_\Sigma C : K$
(3) *If* $(x,\ A){::}\Gamma \vdash_\Sigma K : kind$ *then* $(x,\ B){::}\Gamma \vdash_\Sigma K : kind$
(4) *If* $(x,\ A){::}\Gamma \vdash_\Sigma C = D : K$ *then* $(x,\ B){::}\Gamma \vdash_\Sigma C = D : K$
(5) *If* $(x,\ A){::}\Gamma \vdash_\Sigma K = L : kind$ *then* $(x,\ B){::}\Gamma \vdash_\Sigma K = L : kind$

LEMMA 50 (FUNCTIONALITY FOR TYPING). *Assume that* $\Gamma \vdash_\Sigma M : C$ *and* $\Gamma \vdash_\Sigma N : C$ *and* $\Gamma \vdash_\Sigma M = N : C$. *Then:*
(1) *If* $\Gamma' \mathbin{@} [(y,\ C)] \mathbin{@} \Gamma \vdash_\Sigma P : B$ *then* $\Gamma'[y{:=}M] \mathbin{@} \Gamma \vdash_\Sigma P[y{:=}M] = P[y{:=}N] : B[y{:=}M]$
(2) *If* $\Gamma' \mathbin{@} [(y,\ C)] \mathbin{@} \Gamma \vdash_\Sigma B : K$ *then* $\Gamma'[y{:=}M] \mathbin{@} \Gamma \vdash_\Sigma B[y{:=}M] = B[y{:=}N] : K[y{:=}M]$
(3) *If* $\Gamma' \mathbin{@} [(y,\ C)] \mathbin{@} \Gamma \vdash_\Sigma K : kind$ *then* $\Gamma'[y{:=}M] \mathbin{@} \Gamma \vdash_\Sigma K[y{:=}M] = K[y{:=}N] : kind$

LEMMA 51 (VALIDITY). *Objects, types and kinds appearing in derivable judgments are valid, that is*
(1) *If* $\Gamma \vdash_\Sigma M : A$ *then* $\Gamma \vdash_\Sigma A : type$.
(2) *If* $\Gamma \vdash_\Sigma A : K$ *then* $\Gamma \vdash_\Sigma K : kind$.
(3) *If* $\Gamma \vdash_\Sigma M = N : B$ *then* $\Gamma \vdash_\Sigma M : B$ *and* $\Gamma \vdash_\Sigma N : B$ *and* $\Gamma \vdash_\Sigma B : type$.
(4) *If* $\Gamma \vdash_\Sigma A = B : K$ *then* $\Gamma \vdash_\Sigma A : K$ *and* $\Gamma \vdash_\Sigma B : K$ *and* $\Gamma \vdash_\Sigma K : kind$.
(5) *If* $\Gamma \vdash_\Sigma K = L : kind$ *then* $\Gamma \vdash_\Sigma K : kind$ *and* $\Gamma \vdash_\Sigma L : kind$.

LEMMA 52 (TYPING INVERSION). *The validity rules are invertible, up to conversion of types and kinds.*

(*1*) *If* $\Gamma \vdash_\Sigma x : A$ *then* $\exists B.\ (x, B) \in \Gamma$ *and* $\Gamma \vdash_\Sigma A = B : type$.
(*2*) *If* $\Gamma \vdash_\Sigma c : A$ *then* $\exists B.\ (c, B) \in \Sigma$ *and* $\Gamma \vdash_\Sigma A = B : type$.
(*3*) *If* $\Gamma \vdash_\Sigma M_1\ M_2 : A$ *then* $\exists x\ A_1\ A_2.\ \Gamma \vdash_\Sigma M_1 : \Pi x{:}A_2.\ A_1$ *and* $\Gamma \vdash_\Sigma M_2 : A_2$ *and* $\Gamma \vdash_\Sigma A = A_1[x{:=}M_2] : type$.
(*4*) *If* $\Gamma \vdash_\Sigma \lambda x{:}A.\ M : B$ *and* $x \# \Gamma$ *then* $\exists A'.\ \Gamma \vdash_\Sigma B = \Pi x{:}A.\ A' : type$ *and* $\Gamma \vdash_\Sigma A : type$ *and* $(x, A){::}\Gamma \vdash_\Sigma M : A'$.
(*5*) *If* $\Gamma \vdash_\Sigma \Pi x{:}A_1.\ A_2 : K$ *and* $x \# \Gamma$ *then* $\Gamma \vdash_\Sigma K = type : kind$ *and* $\Gamma \vdash_\Sigma A_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma A_2 : type$.
(*6*) *If* $\Gamma \vdash_\Sigma c : K$ *then* $\exists L.\ (c, L) \in \Sigma$ *and* $\Gamma \vdash_\Sigma K = L : kind$.
(*7*) *If* $\Gamma \vdash_\Sigma A\ M : K$ *then* $\exists x\ A1\ K2.\ \Gamma \vdash_\Sigma A : \Pi x{:}A1.\ K2$ *and* $\Gamma \vdash_\Sigma M : A1$ *and* $\Gamma \vdash_\Sigma K = K2[x{:=}M] : kind$.
(*8*) *If* $\Gamma \vdash_\Sigma \Pi x{:}A_1.\ K_2 : kind$ *and* $x \# \Gamma$ *then* $\Gamma \vdash_\Sigma A_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma K_2 : kind$.

LEMMA 53 (EQUALITY INVERSION).
(*1*) *If* $\Gamma \vdash_\Sigma type = L : kind$ *then* $L = type$.
(*2*) *If* $\Gamma \vdash_\Sigma L = type : kind$ *then* $L = type$.
(*3*) *If* $\Gamma \vdash_\Sigma A = \Pi x{:}B_1.\ B_2 : type$ *and* $x \# \Gamma$ *then* $\exists A_1\ A_2.\ A = \Pi x{:}A_1.\ A_2$ *and* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2 : type$.
(*4*) *If* $\Gamma \vdash_\Sigma \Pi x{:}B_1.\ B_2 = B : type$ *and* $x \# \Gamma$ *then* $\exists A_1\ A_2.\ B = \Pi x{:}A_1.\ A_2$ *and* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2 : type$.
(*5*) *If* $\Gamma \vdash_\Sigma K = \Pi x{:}B_1.\ L_2 : kind$ *and* $x \# \Gamma$ *then* $\exists A_1\ K_2.\ K = \Pi x{:}A_1.\ K_2$ *and* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma K_2 = L_2 : kind$.
(*6*) *If* $\Gamma \vdash_\Sigma \Pi x{:}B_1.\ L_2 = L : kind$ *and* $x \# \Gamma$ *then* $\exists A_1\ K_2.\ L = \Pi x{:}A_1.\ K_2$ *and* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma K_2 = L_2 : kind$.

LEMMA 54 (PRODUCT INJECTIVITY). *Suppose* $x \# \Gamma$.
(*1*) *If* $\Gamma \vdash_\Sigma \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : type$ *then* $\Gamma \vdash_\Sigma A_1 = B_1 : type$ *and* $(x, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2 : type$.
(*2*) *If* $\Gamma \vdash_\Sigma \Pi x{:}A.\ K = \Pi x{:}B.\ L : kind$ *then* $\Gamma \vdash_\Sigma A = B : type$ *and* $(x, A){::}\Gamma \vdash_\Sigma K = L : kind$.

LEMMA 55 STRONG VERSIONS OF RULES. *The following rules are admissible:*

(*1*) $$\frac{\Gamma \vdash_\Sigma M_1 : \Pi x{:}A_2.\ A_1 \quad \Gamma \vdash_\Sigma M_2 : A_2}{\Gamma \vdash_\Sigma M_1\ M_2 : A_1[x{:=}M_2]}$$

(*2*) $$\frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.\ K \quad \Gamma \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma A\ M : K[x{:=}M]}$$

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

(*4*) $$\frac{\Gamma \vdash_\Sigma A_1 = B_1 : type \quad (x, A_1){::}\Gamma \vdash_\Sigma A_2 = B_2 : type \quad x \# \Gamma}{\Gamma \vdash_\Sigma \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : type}$$

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

REFERENCES

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

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

Berghofer, S. and Nipkow, T. 2002. Executing higher order logic. In *Proc. of the International Workshop on Types for Proofs and Programs*. Number 2277 in LNCS. 24–40.

Cheney, J. 2006. Completeness and Herbrand theorems for nominal logic. *Journal of Symbolic Logic 71,* 1, 299–320.

Cheney, J. 2009. A simple nominal type theory. *Electr. Notes Theor. Comput. Sci. 228*, 37–52. LFMTP '08: Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages.

Chlipala, A. J. 2008. Parametric higher-order abstract syntax for mechanized semantics. In *ICFP*, J. Hook and P. Thiemann, Eds. ACM, 143–156.

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

de Bruijn, N. G. 1972. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indagationes Mathematicae 34,* 5, 381–392.

Gabbay, M. J. and Pitts, A. M. 2002. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing 13*, 341–363.

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

Goguen, H. 2005a. Justifying algorithms for β-η-conversion. In *FoSSaCS*, V. Sassone, Ed. LNCS, vol. 3441. Springer, 410–424.

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

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

Harper, R. and Licata, D. 2007. Mechanizing metatheory in a logical framework. *J. Funct. Program. 17,* 4-5, 613–673.

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

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

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

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

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

Norrish, M. and Vestergaard, R. 2007. Proof pearl: de Bruijn terms really do work. In *TPHOLs*. LNCS, vol. 4732. Springer, 207–222.

Pfenning, F. 2001. Logical frameworks. In *Handbook of Automated Reasoning*, J. A. Robinson and A. Voronkov, Eds. Elsevier and MIT Press, 1063–1147.

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

Pitts, A. M. 2003. Nominal logic, a first order theory of names and binding. *Information and Computation 183*, 165–193.

Pitts, A. M. 2006. Alpha-structural recursion and induction. *Journal of the ACM 53,* 3 (May), 459–506.

Pollack, R. 1995. A verified typechecker. In *TLCA*, M. Dezani-Ciancaglini and G. D. Plotkin, Eds. LNCS, vol. 902. Springer, 365–380.

Schürmann, C. and Sarnat, J. 2008. Structural logical relations. In *LICS*. IEEE Computer Society, 69–80.

SHINWELL, M. R., PITTS, A. M., AND GABBAY, M. J. 2003. FreshML: Programming with binders made simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*. ACM Press, 263–274.

THOMPSON, K. 1984. Reflections on trusting trust. *Communications of the ACM 27,* 8, 761–763.

URBAN, C. 2008. Nominal techniques in Isabelle/HOL. *Journal of Automatic Reasoning 40,* 4, 327–356.

URBAN, C., BERGHOFER, S., AND NORRISH, M. 2007. Barendregt's variable convention in rule inductions. In *CADE*. LNAI, vol. 4603. 35–50.

URBAN, C., CHENEY, J., AND BERGHOFER, S. 2008. Mechanizing the metatheory of LF. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*. 45–56.

URBAN, C. AND TASSON, C. 2005. Nominal techniques in Isabelle/HOL. In *CADE*. LNCS, vol. 3632. 38–53.

URBAN, C. AND ZHU, B. 2008. Revisiting cut-elimination: One difficult proof is really a proof. In *RTA*, A. Voronkov, Ed. Lecture Notes in Computer Science, vol. 5117. Springer, 409–424.

WATKINS, K., CERVESATO, I., PFENNING, F., AND WALKER, D. 2003. A concurrent logical framework I: Judgments and properties. Tech. Rep. CMU-CS-02-101, Carnegie Mellon University. May.

WESTBROOK, E., STUMP, A., AND AUSTIN, E. 2009. The calculus of nominal inductive constructions: an intensional approach to encoding name-bindings. In *LFMTP '09: Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages*. ACM, New York, NY, USA, 74–83.

Received Month Year; revised Month Year; accepted Month Year