Intensionality, Extensionality, and Proof Irrelevance in Modal Type Theory

Frank Pfenning*
Department of Computer Science
Carnegie Mellon University
fp@cs.cmu.edu

Abstract

We develop a uniform type theory that integrates intensionality, extensionality, and proof irrelevance as judgmental concepts. Any object may be treated intensionally (subject only to α-conversion), extensionally (subject also to $\beta\eta$-conversion), or as irrelevant (equal to any other object at the same type), depending on where it occurs. Modal restrictions developed in prior work for simple types are generalized and employed to guarantee consistency between these views of objects. Potential applications are in logical frameworks, functional programming, and the foundations of first-order modal logics.

Our type theory contrasts with previous approaches that a priori distinguish propositions (whose proofs are all identified—only their existence is important) from specifications (whose implementations are subject to some definitional equalities).

1 Introduction

In the development of type theory, there has been considerable debate about the degree of extensionality or intensionality that should be inherent in its formulation. In an extensional theory such as the one underlying Nuprl [4] type-checking is undecidable. In a non-extensional theory[1] such as later versions of Martin-Löf's type theory [17], we distinguish a *definitional equality* (also called *judgmental equality*) which is not extensional and decidable, from a *propositional equality* which is extensional and undecidable. There are a number of tradeoffs, both from the philosophical and pragmatic points of view. In an undecidable, extensional theory, programs are significantly more compact than in a decidable, non-extensional theory. On the other hand, we need external arguments to validate the correctness of programs, defeating at least in part the motivations underlying the separation of judgments from propositions [11, 12]. Furthermore, the development of extensional concepts in a non-extensional type theory is far from straightforward, as can be seen from Hofmann's systematic study [10].

Related is the issue of *proof irrelevance*, which plays an important role in the development of mathematical concepts in type theory via subset types or quotient types. For example, the type $\{x{:}A \mid B(x)\}$ should contain the elements M of type A that satisfy property B. If we want type-checking to be decidable, we require evidence that $B(M)$ is satisfied, but we should not distinguish between different proofs of $B(M)$—they are *irrelevant*.

In this paper we present a type theory that internalizes the concepts of intensionality, extensionality, and proof irrelevance via distinctions familiar from modal logic. We strictly follow Martin-Löf's separation of judgments from propositions and both type-checking and definitional equality are decidable.

At the heart of our modal type theory are three judgments

$M :: A$	M is an *expression* of type A,
$M : A$	M is an *term* of type A, and
$M \div A$	M is a *proof* of type A,

constructed from the same set of objects M and types A. Expressions are treated *intensionally*: they are subject only to α-conversion. Terms are treated *extensionally*: they are additionally subject to β and η-conversion. Proofs are treated as if *irrelevant*: any two proofs of the same type are identified. All these are part of the definitional equality of the type theory, which therefore combines intensionality, extensionality, and irrelevance into a single system in a coherent way.

It is a critical property of our type theory that the distinction between expressions, terms, and proofs is not made at the time the constituent constants are declared, but at the

*This work was partially supported by NSF Grant CCR-9988281.

[1]Such type theories are often called *intensional*, but this is somewhat misleading since the meaning of objects is still subject to some conversion rules.

time those constants are used. Any type A can be seen as the type of an expression, the type of a term (= a specification), or the type of a proof (= a proposition). Similarly, an object M may be seen as an expression, as a term, or as a proof, depending only on whether some conditions on its free variables are satisfied. We believe that this flexibility is an inherent advantage of our approach compared to a priori separating propositions (inhabited by proofs that are always irrelevant) from specifications (inhabited by terms that are never irrelevant). This is the approach mostly taken in the literature (see, for example, [18] or, allowing even for some classical reasoning, [2]).

Our system is also interesting in its relation to intuitionistic modal logic when we ignore the objects. Our default judgment $M : A$ can be interpreted as "*A is true*". The judgment $M :: A$ can be read as "*A is valid*". The judgment $M \div A$ can be read as "*A is provable*", hiding the proof object. These can be seen as modes of truth, and the work presented here is an extension of prior work on proof term calculi for the modal logic S4 [20] where validity corresponds to necessary truth.

In a type theory as a foundation for functional programming, irrelevant objects (that is, proofs) are erased before execution without affecting the observable outcome. From this point of view, our type system internally captures a notion of dead-code elimination (see, for example, [1] for a survey and position paper on related type-based approaches). However, we need to extend our type theory with first-class modal operators in order to use it in the context of a complete functional language. Two non-dependent theories in this style are given in [20], explaining an intuitionistic modal logic with necessity ($\Box A$) and possibility ($\Diamond A$). A proper treatment of the fully dependent version of these theories would seem to require an equational theory with commuting conversions and is therefore left to future work. Fortunately, it is possible to develop a consistent and useful type theory where these judgments are considered primarily as hypotheses. Instead of internalizing them as modal operators, we internalize the corresponding hypothetical judgment as function types. Such a restriction is not new—it goes back to similar treatments of linear logic [9] and linear type theory [3] with similar motivations.

In the remainder of the paper we present our type theory, investigate its properties, and sketch some further developments and potential applications.

2 A Modal Type Theory

Our modal type theory is a conservative extension of LF [7]. Our approach follows the outline of [8], adapted here to our more general type theory. The interested reader may find additional details in [19].

2.1 Syntax

The syntax is stratified into objects, families, and kinds as for LF.

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

Here, $M_1 \bullet M_2$ is an application whose argument (M_2) is treated as an expression (intensionally), while $M_1 \circ M_2$ is an application whose argument is treated as a proof (irrelevant for equality). We use K for kinds, A, B, C for type families, M, N, P for objects, Γ for contexts and Σ for signatures. We also use the symbol "kind" to classify the valid kinds. We consider terms that differ only in the names of their bound variables as identical. We write $[N/x]M$, $[N/x]A$ and $[N/x]K$ for capture-avoiding substitution. Signatures and contexts may declare each constant and variable at most once. For example, when we write $\Gamma, x{:}A$ we assume that x is not already declared in Γ. If necessary, we tacitly rename x before adding it to the context Γ. Since a signature is generally fixed, and constants may be used anywhere, we have permitted only two forms of constant declaration, namely $a{:}K$ and $c{:}A$. Note that this is not a restriction for our applications, since it is the *use* not the definition of a constant which determines its status with respect to definitional equality.

2.2 Judgments

The modal type theory is defined by the following principal judgments.

$\vdash \Sigma$ sig	Σ is a valid signature
$\vdash_\Sigma \Gamma$ ctx	Γ is a valid context
$\Gamma \vdash_\Sigma M : A$	M has type A
$\Gamma \vdash_\Sigma A : K$	A has type K
$\Gamma \vdash_\Sigma K$: kind	K is a valid kind
$\Gamma \vdash_\Sigma M = N : A$	M extensionally equals N
$\Gamma \vdash_\Sigma A = B : K$	A extensionally equals B
$\Gamma \vdash_\Sigma K = L$: kind	K extensionally equals L
$\Gamma \vdash_\Sigma M \equiv N : A$	M intensionally equals N

As explained later, intensional equality for types and kinds is not needed directly, and proof irrelevance is a derived concept.

For the judgment $\vdash_\Sigma \Gamma$ ctx we presuppose that Σ is a valid signature. For the remaining judgments of the form $\Gamma \vdash_\Sigma J$ we presuppose that Σ is a valid signature and that Γ is valid in Σ. For the sake of brevity we omit the signature Σ from all judgments but the first, since it does not change throughout a derivation.

If J is a typing or equality judgment, then we write $[M/x]J$ for the obvious substitution of M for x in J. For example, if J is $N : B$, then $[M/x]J$ stands for the judgment $[M/x]N : [M/x]B$.

We also have several derived judgments that are central the nature of our type theory. Each of them is defined by only a single rule. In order to explain these additional judgments we need two critical operations on contexts. The first, $\Gamma^\ominus$, hides all term variables $x{:}A$ by converting them to proof variables $x{\div}A$. The second, $\Gamma^\oplus$, resurrects all proof variables $x{\div}A$ by converting them to term variables $x{:}A$. Other declarations are unaffected in both cases.

$$\begin{array}{rlrl}
(\cdot)^\ominus &= \cdot & (\cdot)^\oplus &= \cdot \\
(\Gamma, x{:}A)^\ominus &= \Gamma^\ominus, x{\div}A & (\Gamma, x{:}A)^\oplus &= \Gamma^\oplus, x{:}A \\
(\Gamma, x{::}A)^\ominus &= \Gamma^\ominus, x{::}A & (\Gamma, x{::}A)^\oplus &= \Gamma^\oplus, x{::}A \\
(\Gamma, x{\div}A)^\ominus &= \Gamma^\ominus, x{\div}A & (\Gamma, x{\div}A)^\oplus &= \Gamma^\oplus, x{:}A
\end{array}$$

Intensional Expressions. The new judgments

$\Gamma \vdash_\Sigma M :: A$	M is an expression of type A
$\Gamma \vdash_\Sigma A :: K$	A is an expression type of kind K
$\Gamma \vdash_\Sigma M = N :: A$	M and N are equal expressions
$\Gamma \vdash_\Sigma A = B :: K$	A and B are equal expression types

are defined by the following rules

$$\frac{\Gamma^\ominus \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma M :: A} \qquad \frac{\Gamma^\ominus \vdash_\Sigma A : K}{\Gamma \vdash_\Sigma A :: K}$$

$$\frac{\Gamma^\ominus \vdash_\Sigma M \equiv N : A}{\Gamma \vdash_\Sigma M = N :: A} \qquad \frac{\Gamma^\ominus \vdash_\Sigma A = B : K}{\Gamma \vdash_\Sigma A = B :: K}$$

The idea is that an expression cannot refer to a term variable $x{:}B$, which would violate intensionality. Thus we mark these variables as irrelevant, $x{\div}B$, which is accomplished by the $(\,)^\ominus$ operation. Note, however, that intensionality and irrelevance interact: proof variables may still occur in an intensional expression, but only inside other proofs! The rules for equality indicate that only intensionally equal terms are considered as equal expressions. We do not directly refer to α-convertibility here because expressions may contain proofs that must be identified, even as subterms of expressions. Note that expression types are not intensional, but that there is a restriction regarding their validity: expression types can not depend on term variables directly.

In general, $M :: A$ is inherently stronger than $M : A$, that is, $M :: A$ implies $M : A$ but not vice versa. In particular, $x{:}A \not\vdash_\Sigma x :: A$.

Irrelevant Proofs. The new judgments

$\Gamma \vdash_\Sigma M \div A$	M is a proof of type A
$\Gamma \vdash_\Sigma A \div K$	A is a proof type of kind K
$\Gamma \vdash_\Sigma M = N \div A$	M and N are equal proofs
$\Gamma \vdash_\Sigma A = B \div K$	A and B are equal proof types

are defined by the following rules

$$\frac{\Gamma^\oplus \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma M \div A} \qquad \frac{\Gamma^\oplus \vdash_\Sigma A : K}{\Gamma \vdash_\Sigma A \div K}$$

$$\frac{\Gamma^\oplus \vdash_\Sigma M = M : A \qquad \Gamma^\oplus \vdash_\Sigma N = N : A}{\Gamma \vdash_\Sigma M = N \div A}$$

$$\frac{\Gamma^\oplus \vdash_\Sigma A = B : K}{\Gamma \vdash_\Sigma A = B \div K}$$

The idea is that a proof may depend on expression variables, term variables, and proof variables. This effect is achieved by relabelling hypotheses $x{\div}B$ to $x{:}B$ in the $(\,)^\oplus$ operation. Note that equality between proofs implements *proof irrelevance* in the classical sense. We could replace the premise $\Gamma^\oplus \vdash_\Sigma M = M : A$ with $\Gamma^\oplus \vdash_\Sigma M : A$ (and similarly for N), but for technical reasons it is simpler if the equality judgment does not refer to the typing judgment here.

It is important that $M \div A$ is inherently weaker than $M : A$. In particular, $x{\div}A \not\vdash_\Sigma x : A$. In other words, terms can not depend on proof variables, but other proofs can. Under a functional interpretation, it is this property which allows the consistent erasure of all proof objects without affecting the observable outcome (assuming proofs are not observable).

Note that, unlike the systems in [5, 20], the rules have the property of *variable monotonicity*: when viewed bottom-up, every variable is preserved—only its status might change from the conclusion to the premise of a rule. This is inspired by a similar idea in [13] and is needed for a clean interaction between expressions and proofs.

2.3 Typing Rules

Our formulation of the typing rules is similar to the second version given in [7] and directly based on [8]. In preparation for the various algorithms we presuppose and inductively preserve the validity of contexts involved in the judgments, instead of checking these properties at the leaves. This is a matter of expediency rather than necessity. Furthermore, in order to the shorten the presentation we use the following notation:

> "$\star$" stands for either ":", "::", or "$\div$" were all occurrences in a rule must be consistent.

Objects.

$$\frac{c{:}A \text{ in } \Sigma}{\Gamma \vdash c : A} \qquad \frac{}{\Gamma, x{:}A, \Gamma' \vdash x : A} \qquad \frac{}{\Gamma, x{::}A, \Gamma' \vdash x : A} \qquad \textit{no rule for } x{\div}A$$

$$\frac{\Gamma \vdash A_1 \star \text{type} \qquad \Gamma, x{\star}A_1 \vdash M_2 : A_2}{\Gamma \vdash \lambda x{\star}A_1.\ M_2 : \Pi x{\star}A_1.\ A_2} \qquad \frac{\Gamma \vdash M_1 : \Pi x{\star}A_2.\ A_1 \qquad \Gamma \vdash M_2 \star A_2}{\Gamma \vdash M_1 * M_2 : [M_2/x]A_1}$$

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

Families.

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

$$\frac{\Gamma \vdash A_1 \star \text{type} \qquad \Gamma, x{\star}A_1 \vdash A_2 : \text{type}}{\Gamma \vdash \Pi x{\star}A_1.\ A_2 : \text{type}} \qquad \frac{\Gamma \vdash A : K \qquad \Gamma \vdash K = L : \text{kind}}{\Gamma \vdash A : L}$$

Figure 1. Rules for Validity of Objects and Families

"$*$" stands for either juxtaposition (an application of a function of type $\Pi x{:}A.\ B$), "$\bullet$" (an application of a function of type $\Pi x{::}A.\ B$), or "$\circ$" (an application of a function of type $\Pi x{\div}A.\ B$). Occurrences of "$*$" must be coordinated with occurrences of "$\star$" in a rule schema in the indicated manner.

Signatures. The rules for validity of signatures are straightforward and omitted here. From now on we fix a valid signature Σ and omit it from the judgments.

Contexts. Validity of contexts must guarantee that we cannot incorrectly refer to a proof variable in a term or expression, or a term variable in an expression. This is achieved by the following rules.

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

Note that the second rule schema actually stands for three rules, depending on whether $x{:}A$, $x{::}A$, or $x{\div}A$ appear in the conclusion and premise.

Objects. Here we proceed as in LF, except that we need to make sure that arguments fit the type and disposition (intensional, extensional, or irrelevant) of the function. The rules can be found in Figure 1. The rule schema for application is the most complex and has three instances. One of them, for example, replaces $\star$ by $::$ and $*$ by $\bullet$.

Families and Kinds. The rules for application and conversion are copies of the rules from the level of objects. Valid function types restrict occurrences of the dependent variable based on whether the corresponding argument is interpreted as an expression, a term, or a proof. This is necessary to guarantee that the type of an application, which is obtained by substitution, is valid. The rules at the level of kinds mirror the ones at the level of families and are elided here.

Generally, in our theory the judgments on families only reflect the judgments on the objects embedded in them. This is typical of type theories such as the one underlying LF.

2.4 Definitional Equality

The rules for definitional are written with the presupposition that a valid signature Σ is fixed and that all contexts Γ are valid. The intent is that equality implies validity of the objects, families, or kinds involved (see Lemma 2). In contrast to the original formulation of LF in [7], equality of terms is based on a notion of parallel conversion plus extensionality, rather than $\beta\eta$-conversion, but the two definitions turn out to be equivalent. In addition we have to take care of intensionality for expressions and irrelevance of proofs. This is reflected in the rules for intensional application $M \bullet N$ and irrelevant application $M \circ N$.

Some of the typing premises in the rules are redundant, but for technical reasons we cannot prove this until validity has been established. Such premises are enclosed in $\{$braces$\}$.

Simultaneous Congruence.

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

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

$$\frac{\Gamma \vdash A_1' = A_1 \star \mathsf{type} \qquad \Gamma \vdash A_1'' = A_1 \star \mathsf{type} \qquad \Gamma, x{\star}A_1 \vdash M_2 = N_2 : A_2}{\Gamma \vdash \lambda x{\star}A_1'.\, M_2 = \lambda x{\star}A_1''.\, N_2 : \Pi x{\star}A_1.\, A_2}$$

Extensionality.

$$\frac{\Gamma \vdash A_1 \star \mathsf{type} \quad \{\Gamma \vdash M : \Pi x{\star}A_1.\, A_2\} \quad \{\Gamma \vdash N : \Pi x{\star}A_1.\, A_2\} \quad \Gamma, x{\star}A_1 \vdash M * x = N * x : A_2}{\Gamma \vdash M = N : \Pi x{\star}A_1.\, A_2}$$

Parallel Reduction.

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

Type Conversion.

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

Figure 2. Extensional Equality Between Objects

Objects. The extensional equality rules for objects are shown in Figure 2, where we have elided rules stating symmetry and transitivity. Conversion is modelled by parallel reduction, a choice motivated by technical concerns. Reflexivity is admissible, which is typical for equality based on parallel reduction.

The crux of intensionality and irrelevance is in the cases for the corresponding applications, $M \bullet N$ and $M \circ N$. We therefore explicitly consider the second premise in the rule schema for application in its three specific instances.

If we compare $M_1\, M_2 = N_1\, N_2$, then the second premise requires $M_2 = N_2 : A_2$, just as in LF.

If we compare $M_1 \bullet M_2 = N_1 \bullet N_2$ then the arguments are treated intensionally and equality will only succeed if M_2 and N_2 are well-typed and intensionally equal expressions. This is enforced with the judgment $\Gamma \vdash M_2 = N_2 :: A_2$ defined before, which holds if and only if $\Gamma^{\ominus} \vdash M_2 \equiv N_2 : A_2$.

If we compare $M_1 \circ M_2 = N_1 \circ N_2$ then the arguments are proofs and are always considered equal. We only need to check that they are well-typed, which is accomplished with the judgment $\Gamma \vdash M_2 = N_2 \div A_2$ defined before. This holds if and only if $\Gamma^{\oplus} \vdash M_2 : A_2$ and $\Gamma^{\oplus} \vdash N_2 : A_2$.

Since the main equality judgment compares terms and not expressions or proofs, the extensionality principle holds for all three kinds of functions. Modulo the construction of the right kind of context and some redundant premises required for technical reasons, these are straightforward. Similarly, the rule of parallel reduction is available for all three kinds of functions.

Families and Kinds. The rules in Figure 2 are repeated with straightforward adaptations at the levels of families and kinds and omitted here. Details can be found in the technical report [19].

Intensional Equality. The intensional equality between objects, $\Gamma \vdash M \equiv N : A$, is defined as a simultaneous congruence just as the extensional equality, but we delete the rules for extensionality and parallel conversion. In the modified rules, arguments to functions that are to be treated as proofs, however, are considered irrelevant for equality as before. Hence irrelevance takes precedence over intensionality, which seems most appropriate for the intended applications as outlined in Section 7. The reader can find the full set of rules in [19].

2.5 Elementary Properties

We establish some elementary properties of the judgments pertaining to the interpretation of contexts. All of these have standard or straightforward proofs on the structure of derivations. First we show weakening for all judgments of the type theory. Secondly, reflexivity holds for valid objects, families, and kinds.

For all lemmas and theorems from here on we tacitly assume that the contexts in the given derivations are well-formed. Furthermore, in the statement of a meta-theoretic property, several occurrences of "$\star$" must still be instantiated consistently as for inference rules.

Lemma 1 (Substitution) *If* $\Gamma, x{\star}A, \Gamma' \vdash J$ *and* $\Gamma \vdash M \star A$ *then* $\Gamma, [M/x]\Gamma' \vdash [M/x]J$.

Proof: By induction over the structure of the first given derivation. □

Note that this is shorthand for several separate substitution properties. Now there is a series of technical lemmas (which we omit), culminating in validity and functionality.

Lemma 2 (Validity)

1. *If* $\Gamma \vdash M \star A$ *then* $\Gamma \vdash A \star \mathsf{type}$.
2. *If* $\Gamma \vdash M = N \star A$, *then* $\Gamma \vdash M \star A$, $\Gamma \vdash N \star A$, *and* $\Gamma \vdash A \star \mathsf{type}$.

Analogous properties hold at the levels of families and kinds.

Lemma 3 (Functionality) *If* $\Gamma \vdash M = N \star A$ *and* $\Gamma, x{\star}A \vdash O = P : B$ *then* $\Gamma \vdash [M/x]O = [N/x]P : [M/x]B$ *and similarly at the level of types and kinds.*

Another consequence of validity is a collection of standard inversion properties. In the interest of space, we elide these properties here. We can further show, from validity, that the premises enclosed in $\{\ldots\}$ are indeed redundant, that is, follow from the other premises.

3 An Algorithm for Deciding Equality

The algorithm for deciding definitional equality can be summarized as follows:

1. When comparing objects at function type, apply extensionality.
2. When comparing objects at base type, reduce both sides to weak head-normal form and then compare heads directly. If they are equal, we compare each corresponding pair of arguments according to their status.
   (a) When the corresponding arguments are extensional (terms), recursively compare for extensional equality.
   (b) When the corresponding arguments are intensional (expressions), compare for syntactic equality modulo α-conversion, ignoring only embedded proof terms.
   (c) When the corresponding arguments are irrelevant (proofs), we always treat them as equal.

Since this algorithm is type-directed in case (1) we need to carry types. Unfortunately, this makes it difficult to prove correctness of the algorithm in the presence of dependent types, because transitivity is not an obvious property. Fortunately, we do not need to know the precise type of the objects we are comparing.

We therefore define a calculus of simple approximate types and an erasure function $()^-$ that eliminates dependencies for the purpose of this algorithm. Note that there are three forms of non-dependent function type which we write as $\tau_1 \overset{\star}{\to} \tau_2$ and similarly for kinds.

We write α to stands for simple base types and we have two special type constants, type^- and kind^-, for the equality judgments at the level of types and kinds.

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

We use τ, θ, δ for simple types and Δ for contexts declaring simple types for variables. We also use "kind^-" *in a* similar role to "kind" in the LF type theory.

We write A^- for the simple type that results from erasing dependencies in A, and similarly K^-. We translate each constant type family a to a base type a^- and extend this to all type families. We extend it further to contexts by applying it to each declaration.

$$\begin{array}{rcl}
(a)^- & = & a^- \\
(A * M)^- & = & A^- \\
(\Pi x{\star}A_1.\, A_2)^- & = & A_1^- \overset{\star}{\to} A_2^-
\end{array}$$

We now present the algorithm in the form of four judgments. These can be interpreted as an algorithm in the manner of logic programming.

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

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

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

$\Delta \vdash M \langle\equiv\rangle N$ (*M is intensionally equal to N*) Algorithmically, we assume that Δ, M, and N are given and we either succeed or fail.

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

Weak Head Reduction.

$$\frac{}{(\lambda x{\star}A_1.\ M_2) * M_1 \overset{\text{whr}}{\longrightarrow} [M_1/x]M_2}$$

$$\frac{M_1 \overset{\text{whr}}{\longrightarrow} M_1'}{M_1 * M_2 \overset{\text{whr}}{\longrightarrow} M_1' * M_2}$$

Type-Directed Object Equality.

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

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

$$\frac{\Delta \vdash M \longleftrightarrow N : \alpha}{\Delta \vdash M \Longleftrightarrow N : \alpha}$$

$$\frac{\Delta, x{\star}\tau_1 \vdash M * x \Longleftrightarrow N * x : \tau_2}{\Delta \vdash M \Longleftrightarrow N : \tau_1 \overset{\star}{\to} \tau_2}$$

Structural Object Equality.

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

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

$$\frac{\Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \overset{::}{\to} \tau_1 \qquad \Delta \vdash M_2 \langle\equiv\rangle N_2}{\Delta \vdash M_1 \bullet M_2 \longleftrightarrow N_1 \bullet N_2 : \tau_1}$$

$$\frac{\Delta \vdash M_1 \longleftrightarrow N_1 : \tau_2 \overset{\div}{\to} \tau_1}{\Delta \vdash M_1 \circ M_2 \longleftrightarrow N_1 \circ N_2 : \tau_1}$$

Structural Intensional Object Equality.

$$\frac{c{:}A \text{ in } \Sigma}{\Delta \vdash c \langle\equiv\rangle c} \qquad \frac{x{:}\tau \text{ or } x{::}\tau \text{ in } \Delta}{\Delta \vdash x \langle\equiv\rangle x}$$

$$\frac{\Delta \vdash A \Longleftrightarrow B : \text{type}^- \qquad \Delta, x{\star}A^- \vdash M \langle\equiv\rangle N}{\Delta \vdash \lambda x{\star}A.\ M \langle\equiv\rangle \lambda x{\star}B.\ N}$$

$$\frac{\Delta \vdash M_1 \langle\equiv\rangle N_1 \qquad \Delta \vdash M_2 \langle\equiv\rangle N_2}{\Delta \vdash M_1\, M_2 \langle\equiv\rangle N_1\, N_2}$$

$$\frac{\Delta \vdash M_1 \langle\equiv\rangle N_1 \qquad \Delta \vdash M_2 \langle\equiv\rangle N_2}{\Delta \vdash M_1 \bullet M_2 \langle\equiv\rangle N_1 \bullet N_2}$$

$$\frac{\Delta \vdash M_1 \langle\equiv\rangle N_1}{\Delta \vdash M_1 \circ M_2 \langle\equiv\rangle N_1 \circ N_2}$$

The crux of the definitions above are the rules for structural equality for applications. We omit the corresponding rules at the level of families. Briefly, kind-directed equality simple decomposes Π-types, while structural type equality reprises the rules for structural object equality above.

The algorithmic equality judgments satisfy some straightforward structural properties, including weakening. Furthermore, the algorithm is essentially deterministic in the sense that when comparing terms at base type we have to weakly head-normalize both sides and compare the results structurally. This is because terms that are weakly head reducible will never be considered structurally equal. This property, as well as the symmetry and transitivity of the algorithm are completely straightforward.

4 Completeness of the Equality Algorithm

In this section we summarize the completeness theorem for the type-directed equality algorithm. That is, if two terms are definitionally equal, the algorithm will succeed. The central idea is to proceed by an argument via logical relations defined inductively on the approximate type of an object, where the approximate type arises from erasing all dependencies.

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

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

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

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

can be proved.

The development can be found in [19], following [8] quite closely, so we omit it here in the interest of brevity.

Theorem 4 (Completeness of the Equality Algorithm) *If* $\Gamma \vdash M = N : A$ *then* $\Gamma^- \vdash M \Longleftrightarrow N : A^-$. *Furthermore, an analogous property holds at the level of families.*

5 Soundness of the Equality Algorithm

In general, the algorithm for type-directed equality is not sound. However, when applied to valid objects of the same type, it is sound and relates only equal terms. This direction requires a number of syntactic lemmas from Section 2.5, but is otherwise mostly straightforward.

Lemma 5 (Subject Reduction) *If* $M \xrightarrow{\text{whr}} M'$ *and* $\Gamma \vdash M : A$ *then* $\Gamma \vdash M' : A$ *and* $\Gamma \vdash M = M' : A$.

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

For the soundness of the equality algorithm we need subject reduction and validity (Lemma 2).

Theorem 6 (Soundness of the Equality Algorithm)

1. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$ *and* $\Gamma^- \vdash M \Longleftrightarrow N : A^-$, *then* $\Gamma \vdash M = N : A$.

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

3. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : B$ *and* $\Gamma^- \vdash M \;(\equiv)\; N$ *then* $\Gamma \vdash A = B : \text{type}$ *and* $\Gamma \vdash M \equiv N : A$.

Analogous properties hold for types and kinds.

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

6 Decidability

We can now show that the judgments for the equality algorithm constitute a decision procedure on valid terms of the same type. This result is then lifted to yield decidability of all judgments in the type theory. This part of the development is relatively standard. An exposition of the necessary auxiliary judgments and lemmas can be found in [19]. We only show the final result.

Theorem 7 (Decidability)

1. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash N : A$ *then it is decidable whether* $\Gamma \vdash M = N : A$.

2. *Given a valid* Γ, M, *and* A, *it is decidable whether* $\Gamma \vdash M : A$.

Corresponding properties hold at the level of families and kinds and for other equality judgments.

We also have that our type theory is conservative over LF. This is important for logical framework applications, since previously established adequacy theorems for encodings will continue to hold in the modal framework.

7 Further Developments and Potential Applications

In this section we consider various possible further developments and potential applications of our ideas.

7.1 Logical Frameworks

The addition of intensional expressions and irrelevant proofs to the logical framework may leads to more direct and more compact encodings in a number of examples.

First, the intensional nature of expressions constitutes a weak form of reflection: arbitrary LF terms are accessible in LF without regard to $\beta\eta$-conversion. At present we do not have any concrete applications for this added expressive power—the primary application of intensional expressions we have in mind is in the richer setting of functional programming explained in Section 7.2 below.

Second, the irrelevant nature of proofs can be used to encode similar situations in object theories, which is quite frequent. For example, in an encoding of linear functions in LF we often have to deal with pairs consisting of the actual function and the proof certifying its linearity. The nature of this proof is, however, irrelevant, as long as it exists. An encoding of this kind might look as shown below. Here we use $A \to B$ for $\Pi x{:}A.\ B$ where x does not occur in B.

$$
\begin{array}{rcl}
\mathit{rawterm} & : & \text{type} \\
\mathit{lam} & : & (\mathit{rawterm} \to \mathit{rawterm}) \to \mathit{rawterm} \\
\mathit{app} & : & \mathit{rawterm} \to \mathit{rawterm} \to \mathit{rawterm} \\
\mathit{linear} & : & \mathit{rawterm} \to \text{type} \\
\ldots & & \\
\mathit{linterm} & : & \Pi E{:}\mathit{rawterm}.\ \Pi L{\div}\mathit{linear}\ E.\ \text{type}
\end{array}
$$

The definitional equality at type *linterm* now ignores the proofs that the expressions E are indeed linear. A similar situation arises in the encoding of object languages with subtyping, where often all proofs of subtype relationships should be considered equal. The logic programming interpretation of such encodings can go from infeasible to practical if all choice points are discarded after the first proof has been found. Such an optimization is justified by our modal type theory without any loss of soundness or completeness.

Moreover, the Twelf system [21] can verify automatically that type families (such as *linear* or one implementing object-language subtyping) are in fact decidable using mode and termination analysis [22]. If we agree that irrelevant objects need not be shown in the user interface, then the proofs of type *linear* E that occur in linear terms actually do not need to be represented at all, leading to a potentially significant space savings that may be critical in applications such as proof-carrying code [14] and certifying decision procedures [23]. Another situation in which an implementation may mark objects as irrelevant is if they are uniquely determined, either for syntactic [15] or semantic [16] reasons. While our modal analysis does not cover all of these optimizations, it generalizes some of the core ideas from a fragment of LF to the full type theory.

7.2 Functional Programming

Our given type theory is fully adequate as a logical framework, but clearly not expressive enough to develop verified functional programs as in various implementations of type theory such as Nuprl [4] or Coq [6]. Besides standard constructs such as inductive types or Σ-types that are orthogonal to our considerations, we need to internalize expressions and proofs as modal operators, rather than just arguments to functions. The blueprint for such an integration for expressions has been given in prior work [5, 20], the correct notion of definitional equality in the presence of dependencies was the main missing ingredient. The presence of both expressions and proofs allows a new twist. We show the formation and introduction rules for the corresponding modal operators, expanding the derived judgments:

$$\frac{\Gamma^{\ominus} \vdash A : \text{type}}{\Gamma \vdash \Box A : \text{type}} \qquad \frac{\Gamma^{\ominus} \vdash M : A}{\Gamma \vdash \text{box}\, M : \Box A}$$

$$\frac{\Gamma^{\oplus} \vdash A : \text{type}}{\Gamma \vdash \triangle A : \text{type}} \qquad \frac{\Gamma^{\oplus} \vdash M : A}{\Gamma \vdash \text{tri}\, M : \triangle A}$$

The elimination rules (especially for the $\triangle$ modality) are unfortunately quite complex. To give the idea: we can now represent, for example, the subset type as a proof-irrelevant version of the the strong sum.

$$\{x{:}A \mid B\} \quad = \quad \Sigma x{:}A.\ \triangle B$$

The triangle operator appears to serve the same purpose as the squash type in [10], except here it derived directly from the judgmental level rather than from identity types.

If our operational interpretation of type theory is based on staged computation [5], then the $\triangle$ modality is necessary to reason about staged programs. Besides a natural symmetry between intensionality and irrelevance as extreme forms of decidable equality, this has been our main motivation for developing a type theory that simultaneously supports these concepts. As an example, consider the specification of a staged power function (presuming a type *nat* and a propositional equality $\doteq$):

$$\not\vdash \Pi n{:}nat.\ \Box(\Pi b{:}nat.\ \Sigma m{:}nat.\ m \doteq b^n) : \text{type}$$

This not well-formed because the term variable n is not available in the expression underneath the $\Box$ constructor. This problem is neatly solved with the $\triangle$ modality as follows:

$$\vdash \Pi n{:}nat.\ \Box(\Pi b{:}nat.\ \Sigma m{:}nat.\ \triangle(m \doteq b^n)) : \text{type}$$

This further specifies that the correctness proof for the staged power function may be erased before execution since it is computationally irrelevant.

7.3 First-Order Intuitionistic Modal Logic

If we consider the first-order fragment of our type theory, the three forms of Π-abstraction correspond to three forms of universal quantification. In terms of a Kripke semantics with varying domains, $\Pi x{:}A.\ B$ quantifies over the elements of the current domain only. This means, for example, that $\Pi x{:}A.\ \Box P(x)$ is only well-formed if P has kind $\Pi x{\div}A.$ type, because otherwise the truth of $P(x)$ may need to be investigated in worlds in which x does not exist. Yet it is still possible that x occurs, even if P can only talk about elements of the current world, as in $\Pi x{:}A.\ P(x) \rightarrow \Box\triangle P(x)$ (which is true, incidentally). The quantifier $\Pi x{::}A.\ B$ quantifies over elements existing in all domains and thus, in general, fewer than $\Pi x{:}A.\ B$. Finally, $\Pi x{\div}A.\ B$ quantifies over all elements of the current world and also elements that existed in some past world. Thus our approach has the potential to shed new light on old debates by allowing various interpretations of quantification to coexist peacefully in a single modal logic.

8 Conclusion

We have presented a dependent type theory that integrates intensionality, extensionality, and proof irrelevance as judgmental notions, based on considerations from modal logic. We proved that equality and type-checking are decidable on the fragment presented here and sketched some possible applications.

The most pressing item of future work is the inclusion of first-class modal operators important for applications in functional programming. The most difficult question here is the right notion of the "default" equality on terms. In this paper, the term equality was fully *extensional*; for functional programming applications, this will not be tenable and must be replaced by a decidable judgmental equality that is sound with respect to the operational semantics. We conjecture that this can be done without upsetting the "extreme" equalities of expressions and proofs for which there appears to be little leeway. Furthermore, some type theoretic constructs such as universes may require generalizations of our proof techniques.

Acknowledgments. We would like to thank the anonymous referees for various helpful comments and suggestions.

References

[1] S. Berardi, M. Coppo, F. Damiani, and P. Giannini. Type-based useless-code elimination for functional programs. In W. Taha, editor, *Proceedings of the International Workshop on Semantics, Applications, and Implementation of Program Generation (SAIG 2000)*, pages 172–189, Montreal, Canada, Sept. 2000. Springer-Verlag LNCS 1924.

[2] U. Berger, W. Buchholz, and H. Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 2001. To appear.

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

[4] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[5] R. Davies and F. Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 2000. To appear. Preliminary version available as Technical Report CMU-CS-99-153, August 1999.

[6] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Coq proof assistant user's guide. Rapport Techniques 154, INRIA, Rocquencourt, France, 1993. Version 5.8.

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

[8] R. Harper and F. Pfenning. On equivalence and canonical forms in the LF type theory. Technical Report CMU-CS-00-148, Department of Computer Science, Carnegie Mellon University, July 2000.

[9] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[10] M. Hofmann. *Extensional Concepts in Intensional Type Theory*. PhD thesis, Department of Computer Science, University of Edinburgh, July 1995. Available as Technical Report CST-117-95.

[11] P. Martin-Löf. Analytic and synthetic judgements in type theory. In P. Parrini, editor, *Kant and Contemporary Epistemology*, pages 87–99. Kluwer Academic Publishers, 1994.

[12] P. Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[13] A. Momigliano. *Elimination of Negation in a Logical Framework.* PhD thesis, Department of Philosophy, Carnegie Mellon University, Aug. 2000. Available as Technical Report CMU-CS-00-175.

[14] G. C. Necula. Proof-carrying code. In N. D. Jones, editor, *Conference Record of the 24th Symposium on Principles of Programming Languages (POPL'97)*, pages 106–119, Paris, France, Jan. 1997. ACM Press.

[15] G. C. Necula and P. Lee. Efficient representation and validation of logical proofs. In *Proceedings of the 13th Annual Symposium on Logic in Computer Science (LICS'98)*, pages 93–104, Indianapolis, Indiana, June 1998. IEEE Computer Society Press.

[16] G. C. Necula and S. Rahul. Oracle-based checking of untrusted software. In H. R. Nielson, editor, *Conference Record of the 28th Annual Symposium on Principles of Programming Languages (POPL'01)*, pages 142–154, London, England, Jan. 2001. ACM Press.

[17] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. Oxford University Press, 1990.

[18] C. Paulin-Mohring. *Extraction de Programmes dans le Calcul des Constructions*. PhD thesis, Université Paris VII, Jan. 1989.

[19] F. Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. Technical Report CMU-CS-01-116, Department of Computer Science, Carnegie Mellon University, Apr. 2001.

[20] F. Pfenning and R. Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11, 2001. To appear. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

[21] F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[22] E. Rohwedder and F. Pfenning. Mode and termination checking for higher-order logic programs. In H. R. Nielson, editor, *Proceedings of the European Symposium on Programming*, pages 296–310, Linköping, Sweden, Apr. 1996. Springer-Verlag LNCS 1058.

[23] A. Stump and D. L. Dill. Generating proofs from a decision procedure. In A. Pnueli and P. Traverso, editors, *Proceedings of the FLoC Workshop on Run-Time Result Verification*, Trento, Italy, July 1999.