A Framework for Defining Logics

Robert Harper* Furio Honsell† Gordon Plotkin‡

Abstract

The Edinburgh Logical Framework (LF) provides a means to define (or present) logics. It is based on a general treatment of syntax, rules, and proofs by means of a typed λ-calculus with dependent types. Syntax is treated in a style similar to, but more general than, Martin-Löf's system of arities. The treatment of rules and proofs focuses on his notion of a *judgement*. Logics are represented in LF via a new principle, the *judgements as types* principle, whereby each judgement is identified with the type of its proofs. This allows for a smooth treatment of discharge and variable occurrence conditions and leads to a uniform treatment of rules and proofs whereby rules are viewed as proofs of higher-order judgements and proof checking is reduced to type checking. The practical benefit of our treatment of formal systems is that logic-independent tools such as proof editors and proof checkers can be constructed.

Categories and subject descriptors: F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic.

General terms: algorithms, theory, verification.

Additional key words and phrases: typed lambda calculus, formal systems, proof checking, interactive theorem proving.

1 Introduction

Much work has been devoted to building systems for checking and building formal proofs in various logical systems. Research in this area was initiated by de Bruijn in the AUTOMATH project whose purpose was to formalize mathematical arguments in a language suitable for machine checking [15]. Interactive proof construction was first considered by Milner, *et. al.* in the LCF system [19]. The fundamental idea was to exploit the abstract type mechanism of ML to provide a safe means of interactively building proofs in PPλ. These ideas were subsequently taken up by Paulson [37] (for LCF) and Petersson [40] (for Martin-Löf's type theory). Coquand and Huet, inspired by the work of Girard, extended the language of AUTOMATH with impredicative features, and developed an interactive proof checker for it [10, 12, 13]. Building on the experience of AUTOMATH and LCF, the NuPRL system [9] is a full-scale interactive proof development environment for type theory that provides support not only for interactive proof construction, but also notational extension, abbreviations, library management, and automated proof search.

There are a great many logics of interest for computer science (for example, equational, first-order, higher-order, modal, temporal, relevant, and linear logics, type-theories and set theories, type assignment systems and operational semantics for programming languages). Implementing an interactive proof development environment for any style of presentation of any of these logics is a daunting task. At the level of abstract syntax, support must be provided for the management of binding operators, substitution, and formula-, term-, and rule schemes. At the level of formal proofs, a representation of formal proofs must be defined, and the mechanisms associated with proof checking and proof construction must be provided. This includes the means of instantiating rule schemes, and constructing proofs from rules, checking the associated context-sensitive applicability conditions. Further effort is needed to support automated search: tactics and tacticals as in LCF [19], unification and matching [47, 25], and so forth. It is therefore highly desirable to develop a general theory of logical systems that isolates the uniformities of a wide class of logics so that much of this

*School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
†Dipartimento di Matematica e Informatica, Università di Udine, Via Zanon, 6, Udine, Italy
‡Laboratory for Foundations of Computer Science, Edinburgh University, Edinburgh EH9-3JZ, United Kingdom

effort can be expended once and for all. In particular, it is important to define a presentation language for defining logical systems that is a suitable basis for a logic-independent proof development environment.

The Edinburgh Logical Framework (LF) is intended to provide such a means of presentation. It comprises a *formal system* yielding a formal means of presentation of logical systems, and an *informal method* of finding such presentations. An important part in presenting logics is played by a *judgements as types* principle, which can be regarded as the metatheoretic analogue of the well-known propositions-as-types principles [14, 15, 23].

The formal system of LF is based on a a typed λ-calculus with first-order dependent types that is closely related to several of the AUTOMATH languages. Its proof-theoretic strength is quite low (equivalent to that of the simply typed λ-calculus). In fact, the LF type system is chosen to be as weak as possible so as to provide scope for the development of practical unification and matching algorithms for use in applications [44, 16]. The type system has three levels of terms: objects, types (which classify objects), and kinds (which classify families of types). There is also a formal notion of *definitional equality*, which we take to be β-conversion; all matters relating to encodings of logics are treated up to this equality. A logical system is presented by a *signature* which assigns kinds and types to a finite set of constants that represent its syntax, its judgements (or assertions), and its rule schemes. The type system is sufficiently expressive to represent the conventions associated with binding operators, with schematic abstraction and instantiation, and with the variable-occurrence and discharge conditions associated with rules in systems of natural deduction. All of the formal structures of the logical system (expressions, assertions, and proofs) are encoded as LF terms; the type checking rules enforce well-formedness conditions. In particular, proof checking is reduced to type checking.

The treatment of syntax in LF is inspired by Church [6] and by Martin-Löf's system of arities [36]: binding operators are represented using the λ-abstractions of LF. However, the presence of dependent types allows for a smoother treatment of syntax in many commonly-occurring examples. Our treatment of rules and proofs focuses on the notion of *judgement* (or assertion) stressed by Martin-Löf [30]: logical systems are viewed as calculi for constructing proofs of some collection of basic judgement forms. To present logics in LF we introduce the judgements-as-types principle mentioned above: judgements are represented as types, and proofs are represented as terms whose type is the representation of the judgement that they prove. The structure of the LF type system provides for the uniform extension of the basic judgement forms to two higher-order forms introduced by Martin-Löf, the *hypothetical*, representing consequence, and the *schematic*, representing generality. By exploiting these forms, it is possible to view inference rules as primitive proofs of higher-order judgements. This allows us to collapse the notions of rule and proof into one, and eliminates the distinction between primitive and derived rules of inference. The use of higher-order judgements is essential to achieve a smooth representation of systems of natural deduction.

Having a means of presentation of a logic, it is natural to enquire as to the correctness of a presentation. A signature is said to be an *adequate* presentation of a logical system iff there an *encoding* which is a *compositional bijection* between the syntactic entities (terms, formulas, proofs) of the logical system and certain valid LF terms (the so-called "canonical forms") in that signature. By "compositional" we mean that substitution commutes with encoding; in particular substitution in the logical system is encoded as substitution in LF (which relies on the identification of object-logic variables with the variables of LF). By "adequate" we mean "full" (does not introduce any additional entities) and "faithful" (encodes all entities uniquely). There is some flexibility in the statement of adequacy for specific logical systems. For example, in the examples that we consider here we achieve an adequate presentation of proofs of consequence, not just of pure theorems. This may not, in general, be achievable, or even of interest. On the other hand, LF automatically provides a much richer structure than mere proofs of consequence. For example, it is possible to express the derivability of inference rules and to consider proofs under the assumption of these new rules of inference. The adequacy theorem ensures, however, that this additional structure is a conservative extension of the underlying logic.

This methodology for representing formal systems differs substantially from that used in NuPRL and the Calculus of Constructions. The latter are not concerned with the problem of representation of arbitrary formal systems, but rather with developing the "internal" mathematics of a specific constructive set theory. For detailed explanations and illustrations, see the NuPRL book [9] and the papers of Coquand and Huet [10, 12, 13]. There is a much closer relationship between the present work and that of the AUTOMATH project [15]. On the one hand the AUTOMATH project was concerned with formalizing traditional mathematical reasoning without foundational prejudice. In this regard the overall aims of LF are similar, and our work may be seen as carrying forward the aims of the AUTOMATH project. On the other hand, our approach differs from that of AUTOMATH in that we seek to develop a general theory of representation of formal systems on a computer. In particular, we stress the notion of adequate encoding for a number of formal systems, as will become apparent in the sequel.

This paper is organized as follows. In Section 2 we present the LF λ-calculus, and state its basic metatheoretic properties (the proofs are given in the appendix). The main result of this section is the decidability of the type system, which is essential for the reduction of proof checking to type checking. (We take it for granted that proof checking in logical systems of interest is decidable.) In Section 3 we present the LF theory of syntax. We consider two example logical systems, first-order and higher-order logic. In Section 4 we present the LF theory of rules and proofs. Once again, we take as examples first-order and higher-order logic. In Section 5 we discuss related work. The appendix is devoted to the proofs of the metatheoretic results stated in Section 2, and the consideration of a stronger notion of definitional equality.

This work was initiated in the summer of 1986, and was first reported in July, 1987 at the Symposium on Logic in Computer Science in Ithaca, New York. We gratefully acknowledge the influence of Per Martin-Löf, particularly the lectures delivered in Edinburgh in the spring of 1986, which inspired the present work. We are especially indebted to the other members of the LF project, Arnon Avron and Ian Mason, for their many valuable contributions to this work. We also thank David Pym, Don Sannella and Andrzej Tarlecki for their comments on earlier drafts of this paper. Support for this research was provided by the Science and Engineering Research Council of the United Kingdom under grant number GR/D 64612 (Computer Assisted Formal Reasoning: Logics and Modularity), by the ESPRIT Basic Research Action grant 3245 (Logical Frameworks: Design, Implementation and Experiment), by Italian MURST grants, and by the Defense Advanced Research Projects Agency under ARPA Order No. 5404.

2 The Type Theory of LF

The LF type theory is a predicative, dependently-typed λ-calculus, closely related to the Π-fragment of AUT-PI [55], a language belonging to the AUTOMATH family. LF can also be fruitfully compared to several other systems such as AUT-QE [55], Martin-Löf's early type theories [28], Huet and Coquand's Calculus of Constructions [10], and Meyer and Reinhold's λ^{π} [33]. Some of these comparisons are only superficial; others are more substantial if carried out under an appropriate notational transliteration. The variant of the LF type theory that we shall discuss in greatest detail in this paper is essentially a subsystem of the Calculus of Constructions. The study of the LF type theory greatly benefited from previous work on these related systems (in particular, [55] and [10]).

The purpose of this section is to define the LF type system and to state its key metatheoretic properties. As in all systems with dependent types, the notion of *definitional equality* — the fundamental equivalence relation imposed on expressions of the LF type theory — plays a central role [27]. Definitional equality is used to handle definitions and instantiation of schemes, and is essential for establishing the adequacy theorems. It is not to be confused with any equality that may be present in a represented logic. We consider here a version of definitional equality in which only β-like axioms are considered; in the appendix we briefly consider strengthening the definitional equality relation to admit η-like axioms. The principal result of the section is the decidability of the system. However, we also state some properties that have some bearing on the nature of the representation of logical systems in LF. We also introduce the notion of canonical form, which is needed for the proofs of adequacy given in Sections 3 and 4 below. Most of the proofs of the results stated in this section are deferred to the appendix.

2.1 Definition of the Type Theory

The LF type theory is a calculus for deriving typing and equivalence (*i.e.*definitional equality) assertions. The system is structured into three levels of terms: the level of *objects*, the level of *types* and *families of types*, or simply *families*, and the level of *kinds*. Objects (denoted by M, N, and P) are used to represent syntactic entities or proofs or inference rules in a formal system. Types and families of types (denoted by A, B, and C) are used to represent syntactic classes and judgement or assertion forms. Types classify objects; families of types may be thought of as n-ary functions mapping objects to types. Kinds (denoted by K and L) are introduced purely for technical reasons in order to classify families; in particular there is a distinguished kind Type that classifies the types. General terms — designating kinds, types, or objects — are denoted by U and V.

We assume given a countably infinite set of variables, and two countably infinite sets of constants, disjoint from each other and from the variables, one for object-level constants, the other for family-level constants. The metavariables x, y, and z range over the variables, c and d range over the object-level constants, and a and b over the family-level constants. The abstract syntax of the entities of LF is given by the following grammar:

$$\begin{array}{lll} \textit{Kinds} & K & ::= \ \mathsf{Type} \mid \Pi x{:}A.K \\ \textit{Families} & A & ::= \ a \mid \Pi x{:}A.B \mid \lambda x{:}A.B \mid AM \\ \textit{Objects} & M & ::= \ c \mid x \mid \lambda x{:}A.M \mid MN \end{array}$$

Both Π and λ are binding operators, binding the variable x in the second argument position. As usual, we identify terms that differ only in the choice of bound variable names. The notions of free and bound variables, the binding occurrence of a given variable occurrence, and capture-avoiding substitution may be defined accordingly. We write $[M_1, \ldots, M_k/x_1, \ldots, x_k]U$ for the result of simultaneously substituting M_1, $\ldots$, M_k for free occurrences of x_1, $\ldots$, x_k in U, renaming bound variables to avoid capture. We write $A \to B$ for $\Pi x{:}A.B$ when x does not occur free in B; to avoid excessive parentheses, we regard successive occurrences of "$\to$" as associating to the right (and will adopt this convention for other infix operators).

In the LF type theory *signatures* are used to keep track of the types or kinds assigned to constants, and *contexts* are used similarly to keep track of the types assigned to variables. The distinction between signature and context is introduced for pragmatic reasons (as is the distinction between constant and variable). The main differences are that in a well-formed signature variables may not occur free in the types or kinds assigned to constants, and in a well-formed context only types, and not kinds, may be assigned to variables. The abstract syntax for signatures and contexts is given by the following grammar:

$$\begin{array}{lll} \textit{Signatures} & \Sigma & ::= \ \langle\rangle \mid \Sigma, a{:}K \mid \Sigma, c{:}A \\ \textit{Contexts} & \Gamma & ::= \ \langle\rangle \mid \Gamma, x{:}A \end{array}$$

Thus both signatures and contexts consist of finite sequences of declarations. We write Γ, Γ' to indicate concatentation of the contexts Γ and Γ'.

Various versions of the LF type system can be given according to the strength of definitional equality one is willing to consider. Each can be presented using various styles and notational expedients. The system that we consider here will be presented in a style that trades off conciseness against readability. The LF type theory is a formal system for deriving assertions of one of the following forms (the intended meaning is in brackets):

$$\begin{array}{ll} \Sigma \ \mathrm{sig} & (\Sigma \text{ is a valid signature}) \\ \vdash_\Sigma \Gamma & (\Gamma \text{ is a valid context in } \Sigma) \\ \Gamma \vdash_\Sigma K & (K \text{ is a kind in } \Gamma \text{ and } \Sigma) \\ \Gamma \vdash_\Sigma A : K & (A \text{ has kind } K \text{ in } \Gamma \text{ and } \Sigma) \\ \Gamma \vdash_\Sigma M : A & (M \text{ has type } A \text{ in } \Gamma \text{ and } \Sigma) \end{array}$$

We write $\Gamma \vdash_\Sigma \alpha$ for an arbitrary assertion of one of the forms $\Gamma \vdash K$, $\Gamma \vdash A : K$, or $\Gamma \vdash M : A$. The rules for deriving the formation assertions of the LF type theory are given in Tables 1 and 2.

The inference rules of the LF type theory make use of an unspecified notion of definitional equality, consisting of the following three forms of assertion:

$$\begin{array}{ll} \Gamma \vdash_\Sigma K \equiv K' & (K \text{ and } K' \text{ are definitionally equal kinds in } \Gamma \text{ and } \Sigma) \\ \Gamma \vdash_\Sigma A \equiv A' & (A \text{ and } A' \text{ are definitionally equal families in } \Gamma \text{ and } \Sigma) \\ \Gamma \vdash_\Sigma M \equiv M' & (M \text{ and } M' \text{ are definitionally equal objects in } \Gamma \text{ and } \Sigma) \end{array}$$

The first two of these relations are used directly (rules B-CONV-FAM and B-CONV-OBJ); the third is used to define the others. Definitional equality plays an important role in our usage of the LF type theory for encoding proofs; an example of an essential use will be presented in Section 4. Choices for the definitional equality relations will be discussed shortly.

Valid Signatures

$$\frac{}{\langle\rangle \text{ sig}} \qquad \text{(B-EMPTY-SIG)}$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \text{dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}} \qquad \text{(B-KIND-SIG)}$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma A : \mathsf{Type} \quad c \notin \text{dom}(\Sigma)}{\Sigma, c{:}A \text{ sig}} \qquad \text{(B-TYPE-SIG)}$$

Valid Contexts

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \langle\rangle} \qquad \text{(B-EMPTY-CTX)}$$

$$\frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma A : \mathsf{Type} \quad x \notin \text{dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}A} \qquad \text{(B-TYPE-CTX)}$$

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathsf{Type}} \qquad \text{(B-TYPE-KIND)}$$

$$\frac{\Gamma, x{:}A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}A.K} \qquad \text{(B-PI-KIND)}$$

Table 1: The LF Type System (Part I)

We often simply write $\Gamma \vdash_\Sigma \alpha$ to mean that the indicated assertion is derivable in the system. A term is said to be *well-typed* or *valid* in a signature and context if it can be shown to either be a kind, have a kind, or have a type in that signature and context. We similarly speak of terms as being valid kinds and valid types or families in a signature and context; we also speak of valid contexts relative to a signature and of valid signatures.

2.2 Definitional Equality

The definitional equality relation that we shall consider here is extremely simple, being β-conversion of the entities of all three levels. (Stronger notions of definitional equality are discussed briefly in the appendix.) In this case the definition can be given without reference to the signature or context, and hence will usually be dropped. Thus we define the definitional equality relation, $\equiv$, between entities of all three levels to be the symmetric and transitive closure of the *parallel nested reduction* relation, $\rightarrow$, defined by the rules of Table 3. The transitive closure of parallel reduction is denoted by $\rightarrow^*$.

An immediate benefit of our choice of definitional equality is the diamond property for parallel reduction:

Proposition 2.1 (Diamond Property) *If $U \rightarrow U'$ and $U \rightarrow U''$, then there exists V such that $U' \rightarrow V$ and $U'' \rightarrow V$.* □

This result can be readily established by adapting the method of Tait and Martin-Löf [28, 42, 53] to our system. It follows that $\rightarrow^*$ satisfies the Church-Rosser property:

Corollary 2.2 (Church-Rosser Property) *If $U \rightarrow^* U'$ and $U \rightarrow^* U''$, then there exists V such that $U' \rightarrow^* V$ and $U'' \rightarrow^* V$.* □

Valid Families

$$\frac{\vdash_\Sigma \Gamma \quad c{:}K \in \Sigma}{\Gamma \vdash_\Sigma c : K} \qquad \text{(B-CONST-FAM)}$$

$$\frac{\Gamma, x{:}A \vdash_\Sigma B : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi x{:}A.B : \mathsf{Type}} \qquad \text{(B-PI-FAM)}$$

$$\frac{\Gamma, x{:}A \vdash_\Sigma B : K}{\Gamma \vdash_\Sigma \lambda x{:}A.B : \Pi x{:}A.K} \qquad \text{(B-ABS-FAM)}$$

$$\frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.K \quad \Gamma \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma AM : [M/x]K} \qquad \text{(B-APP-FAM)}$$

$$\frac{\Gamma \vdash_\Sigma A : K \quad \Gamma \vdash_\Sigma K' \quad \Gamma \vdash_\Sigma K \equiv K'}{\Gamma \vdash_\Sigma A : K'} \qquad \text{(B-CONV-FAM)}$$

Valid Objects

$$\frac{\vdash_\Sigma \Gamma \quad c{:}A \in \Sigma}{\Gamma \vdash_\Sigma c : A} \qquad \text{(B-CONST-OBJ)}$$

$$\frac{\vdash_\Sigma \Gamma \quad x{:}A \in \Gamma}{\Gamma \vdash_\Sigma x : A} \qquad \text{(B-VAR-OBJ)}$$

$$\frac{\Gamma, x{:}A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x{:}A.M : \Pi x{:}A.B} \qquad \text{(B-ABS-OBJ)}$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}A.B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma MN : [N/x]B} \qquad \text{(B-APP-OBJ)}$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma A' : \mathsf{Type} \quad \Gamma \vdash_\Sigma A \equiv A'}{\Gamma \vdash_\Sigma M : A'} \qquad \text{(B-CONV-OBJ)}$$

Table 2: The LF Type System (Part II)

$$\frac{}{M \to M} \quad \text{(R-REFL)}$$

$$\frac{M \to M' \quad N \to N'}{(\lambda x{:}A.M)N \to [N'/x]M'} \quad \text{(R-BETA-OBJ)}$$

$$\frac{B \to B' \quad N \to N'}{(\lambda x{:}A.B)N \to [N'/x]B'} \quad \text{(R-BETA-FAM)}$$

$$\frac{M \to M' \quad N \to N'}{MN \to M'N'} \quad \text{(R-APP-OBJ)}$$

$$\frac{A \to A' \quad M \to M'}{AM \to A'M'} \quad \text{(R-APP-FAM)}$$

$$\frac{A \to A' \quad M \to M'}{\lambda x{:}A.M \to \lambda x{:}A'.M'} \quad \text{(R-ABS-OBJ)}$$

$$\frac{A \to A' \quad B \to B'}{\lambda x{:}A.B \to \lambda x{:}A'.B'} \quad \text{(R-ABS-FAM)}$$

$$\frac{A \to A' \quad B \to B'}{\Pi x{:}A.B \to \Pi x{:}A'.B'} \quad \text{(R-PI-FAM)}$$

$$\frac{A \to A' \quad K \to K'}{\Pi x{:}A.K \to \Pi x{:}A'.K'} \quad \text{(R-PI-KIND)}$$

Table 3: Parallel Reduction

It is noteworthy that the Church-Rosser property holds for our notion of reduction irrespective of whether the terms are well-typed. This property is lost if η is added: the term $\lambda x{:}A.(\lambda y{:}B.M)x$ reduces via η to $\lambda y{:}B.M$ and by β to $\lambda x{:}A.[x/y]M$, which is α-convertible to $\lambda y{:}A.M$. The diamond cannot be completed unless A and B have a common reduct, which is not the case for certain ill-typed terms. This is the reason for introducing the context argument in the definitional equality relation: the Church-Rosser property can, in general, only be established for well-typed terms.

2.3 Fundamental Properties of the Type System

The turnstile symbol used in the LF type system is reminiscent of a consequence relation. The following theorem bears on the naturality with which formal systems may be encoded in LF, and is of some use in the proofs of adequacy given below.

Theorem 2.3 *The following are derived rules:*

1. *Weakening: if* $\Gamma \vdash_\Sigma \alpha$ *and* $\vdash_\Sigma \Gamma, \Gamma'$*, then* $\Gamma, \Gamma' \vdash_\Sigma \alpha$.
2. *Strengthening: if* $\Gamma, x{:}U, \Gamma' \vdash_\Sigma \alpha$*, then* $\Gamma, \Gamma' \vdash_\Sigma \alpha$ *provided that* $x \notin \mathrm{FV}(\Gamma') \cup \mathrm{FV}(\alpha)$.
3. *Transitivity: if* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma, x{:}A, \Gamma' \vdash_\Sigma \alpha$*, then* $\Gamma, [M/x]\Gamma' \vdash_\Sigma [M/x]\alpha$.
4. *Permutation: if*
$$\Gamma, x{:}U, \Gamma', y{:}V, \Gamma'' \vdash_\Sigma \alpha,$$
then
$$\Gamma, y{:}V, \Gamma', x{:}U, \Gamma'' \vdash_\Sigma \alpha,$$
provided that x *does not occur free in* Γ' *or* V*, and that* V *is a valid type or kind in* Γ. □

Rule (3) may also be viewed as a substitution principle; we prefer the name "transitivity" to stress the intended application of LF. The derivability of strengthening may be viewed as evidence for the fact that the LF assertions are, in Martin-Löf's terminology, "analytic judgements" since the derivability of an assertion $\Gamma \vdash_\Sigma \alpha$ depends only on the variables that actually occur in α [31, 30]. In contrast, such a property does not hold for the extensional type theories of Martin-Löf [29].

A natural algorithm for type checking proceeds by computing a type or kind for a term, then testing for definitional equality with the given type or kind. This approach relies on the following property of the type system:

Theorem 2.4 (Unicity of Types and Kinds)

1. *If* $\Gamma \vdash_\Sigma M : A$ *and* $\Gamma \vdash_\Sigma M : A'$*, then* $\Gamma \vdash_\Sigma A \equiv A'$.
2. *If* $\Gamma \vdash_\Sigma A : K$ *and* $\Gamma \vdash_\Sigma A : K'$*, then* $\Gamma \vdash_\Sigma K \equiv K'$. □

Parallel reduction enjoys the strong normalization property (*i.e.*, all reduction sequences arrives at a normal form):

Theorem 2.5 (Strong Normalization)

1. *If* $\Gamma \vdash_\Sigma K$*, then* K *is strongly normalizing.*
2. *If* $\Gamma \vdash_\Sigma A : K$*, then* A *is strongly normalizing.*
3. *If* $\Gamma \vdash_\Sigma M : A$*, then* M *is strongly normalizing.* □

A principal goal of LF is the uniform reduction of proof checking for an object logic to type checking in the LF type theory. This use of LF makes the decidability of the LF typing assertions of paramount importance. Martin-Löf's account of Kreisel's dictum, although in a slightly different context, seems particularly appropriate here: "...that it should be recursively decidable whether or not a closed term formally proves a given closed formula ... is the formal counterpart of the experience that we can decide whether or not a purported proof actually is a proof of a given proposition (in Kreisel's words: we recognize a proof when we see one)" [28])).

Together with the Church-Rosser property, the strong normalization theorem entails the decidability of definitional equality for well-typed expressions: to test $U \equiv V$, reduce both to their (unique) normal forms and check that they are identical up to the names of bound variables.

Theorem 2.6 (Decidability) *All assertions of the LF type system are recursively decidable.*

2.4 Canonical Forms

In the proofs of adequacy to be given below, we shall make use of a stronger notion than that of normal form, called a *canonical form.* The intention is that, the canonical forms of a given type are, so to speak, the long $\beta\eta$-normal forms of that type. In order to give the definition, we need the following:

Lemma 2.7 (Characterization of Normal Forms)

1. *A normal form kind has shape* $\Pi x_1{:}A_1.\ldots.\Pi x_n{:}A_n.\mathsf{Type}$ *for some* $n \geq 0$*, where the* A_i*'s are normal form types.*

2. *A normal form family has shape*

$$\lambda x_1{:}A_1.\ldots.\lambda x_n{:}A_n.\Pi y_1{:}B_1.\ldots.\Pi y_m{:}B_m.\xi M_1 \ldots M_k$$

for some $n, m, k \geq 0$*, where the* A_i*'s and* B_i*'s are normal form types, and the* M_i*'s are normal form objects, and* ξ *is a variable or a constant.*

3. *A normal form object has shape* $\lambda x_1{:}A_1.\ldots.\lambda x_n{:}A_n.\xi M_1 \ldots M_k$ *for some* $n, k \geq 0$*, where the* A_i*'s are normal form types, the* M_i*'s are normal form objects, and* ξ *is a variable or a constant.*

Proof By induction on the structure of terms, bearing in mind that a normal form is a term with no subterms of the form $(\lambda x{:}A.U)M$. □

The *arity* of a valid type or kind is the number of Π's in the prefix of its normal form. The arity of a constant with respect to a valid signature is the arity of its type or kind in that signature. Similarly, the arity of a variable with respect to a valid context is the arity of its type in that context. The arity of a bound variable occurrence in a valid term is the arity of the type label attached to its binding occurrence.

Definition 2.8 *An occurrence of a constant or variable* ξ *in a valid term* U *is* fully applied *with respect to* Σ *and* Γ *iff that occurrence is in a context of the form* $\xi M_1 \ldots M_n$*, where* n *is the arity of* ξ*.* □

Definition 2.9 *A valid term* U *is* canonical *with respect to the valid signature* Σ *and valid context* Γ *iff* U *is in normal form and every constant and variable occurrence in* U *is fully applied with respect to* Σ *and* Γ*. A valid signature is* canonical *iff the type or kind assigned to each constant is canonical with respect to the declarations preceding it; similarly, a valid context is* canonical *iff the type assigned to every variable is canonical with respect to the preceding declarations. A valid term* U has a canonical form *iff its normal form is canonical.* □

It is decidable whether a valid term has a canonical form. Not all terms are convertible to canonical form — consider, for example, a variable of functional type. One reason to consider stronger notions of definitional equality is to achieve the property that every well-formed term is convertible to a unique canonical form, but such strengthenings are not strictly necessary for the LF encoding methodology.

The following lemma characterizes the canonical forms:

Lemma 2.10 (Characterization of Canonical Forms)

1. *A kind K is canonical with respect to Σ and Γ iff it is of the form*

$$\Pi x_1.{:}A_1.\ldots.\Pi x_n{:}A_n.\mathsf{Type}$$

with each A_i $(1 \le i \le n)$ canonical with respect to Σ and $\Gamma, x_1{:}A_1, \ldots, x_{i-1}{:}A_{i-1}$.

2. *A family A is canonical with canonical kind*

$$\Pi x_1{:}A_1.\ldots.\Pi x_n{:}A_n.\mathsf{Type}$$

with respect to Σ and Γ iff A is of the form

$$\lambda x_1{:}A_1.\ldots.\lambda x_n{:}A_n.\Pi y_1{:}B_1.\ldots.\Pi y_m{:}B_m.\xi M_1 \ldots M_k$$

where k is the arity of the variable or constant ξ, each B_i $(1 \le i \le m)$ is canonical with respect to Σ and

$$\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_{i-1}{:}B_{i-1},$$

and each M_i $(1 \le i \le k)$ is canonical with respect to Σ and

$$\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n, y_1{:}B_1, \ldots, y_m{:}B_m.$$

3. *An object M is canonical with canonical type*

$$\Pi x_1{:}A_1.\ldots.\Pi x_n{:}A_n.\xi M_1 \ldots M_k$$

with respect to Σ and Γ iff M is of the form

$$\lambda x_1{:}A_1.\ldots.\lambda x_n{:}A_n.\xi' N_1 \ldots N_l$$

where l is the arity of ξ' and each N_i $(1 \le i \le l)$ is canonical with respect to Σ and $\Gamma, x_1{:}A_1, \ldots, x_n{:}A_n$.

Proof By the characterization of normal forms and the definition of canonical forms. Note that since normal-form applications must begin with a constant or variable, an application cannot be a canonical form of product type or kind, for then the head constant or variable would not be fully applied. □

It can be shown that if there exists an M such that we can establish $\Gamma \vdash_\Sigma M : A$, where Σ, Γ, and A are all in canonical form, then there exists a term M' in canonical form such that $\Gamma \vdash_\Sigma M' : A$ can be established. This ensures that the restriction to canonical forms does not affect the inhabitability (*i.e.*, the existence of a term) of a canonical type in a given canonical signature and context.

3 Theory of Expressions

The method for representing the syntax of a language is inspired by Church [6] and Martin-Löf [36]. The general approach is to associate an LF type to each syntactic category, and to declare a constant corresponding to each expression-forming construct of the object language, in such a way that a bijective correspondence between expressions of the object language and canonical forms of a suitable type is established. Variable-binding operators are typically represented using constants whose domain is of functional type, in contrast to the usual representations of abstract syntax in programming languages such as ML [34] and Prolog [7]. The principal advantage of this approach is that it enables the machinery associated with handling binding operators (such as α-conversion and capture-avoiding substitution) to be shifted to the metatheory, rather than be repeated for each presentation. Of course, only binding operators that behave similarly to the binding operators of LF can be represented in this way; for object logics with non-standard variable binding other means are necessary. In particular, one can use the notion of judgement (explained in the next section) to enforce context-dependent conditions that are not directly expressible using the LF type system. (See [4, 3] for examples.) It should be noted that since the LF type system is considerably richer than the system of arities, it is correspondingly better able to provide a natural representation of syntax (see, for example, the encoding of higher-order logic given below).

In this section we consider the representation of the abstract syntax of first-order logic [49] and higher-order logic [6]. For the sake of specificity, we assume in each case that the language of individuals is that of arithmetic. It will be clear that the method applies to any signature of first-order or higher-order logic. We shall treat the first-order logic example in some detail in order to illustrate the issues involved. The presentation of the abstract syntax of first-order logic will form a part of the signature Σ_{FOL} (in the next section we will discuss the extension of Σ_{FOL} to represent rules and proofs). Similarly, the presentation of the abstract syntax of higher-order logic will form a part of the signature Σ_{HOL}.

3.1 First-Order Logic

In a first-order language there are two syntactic categories: the *terms*, which stand for individuals (objects in the domain of quantification), and the *formulas*, which stand for propositions. In the case of first-order arithmetic, the language of terms t, u is given by the following abstract syntax:

$$t \quad ::= \quad x \mid 0 \mid succ(t) \mid t + u \mid t \times u$$

where x ranges over the set of variables of the first-order language. We write $t[x]$ to indicate that the variable x may occur in t, and use $t[t'/x]$, or simply $t[t']$, for the result of substituting t' for x in t. The language of formulas is given by the following abstract syntax:

$$\varphi \quad ::= \quad t{=}u \mid t{<}u \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \supset \psi \mid \forall x.\varphi \mid \exists x.\varphi$$

The metavariables φ and ψ range over the formulas of first-order arithmetic. We write $\varphi[x]$ to indicate that the variable x may occur free in φ, and $\varphi[t/x]$, or simply $\varphi[t]$, for the result of substituting t for free occurrences of x in φ.

The two syntactic categories of first-order logic are represented in LF by the types ι of individuals, and o of propositions. Thus Σ_{FOL} begins with the declarations:

$$\begin{array}{lcl} \iota & : & \mathsf{Type} \\ o & : & \mathsf{Type} \end{array}$$

In order to represent the terms, the signature Σ_{FOL} includes a constant for each term constructor in the language, as follows:

$$\begin{array}{lcl} 0 & : & \iota \\ succ & : & \iota \to \iota \\ + & : & \iota \to \iota \to \iota \\ \times & : & \iota \to \iota \to \iota \end{array}$$

Terms are encoded in LF by a function ε_X (where X is a finite set of variables) mapping terms of first-order arithmetic with free variables in X to terms of type ι in Σ_{FOL} and Γ_X. Here Γ_X is the context $x_1{:}\iota, \ldots, x_n{:}\iota$, where $x_1, \ldots, x_n$ is a standard enumeration of X. This encoding is defined by induction on the structure of the terms as follows:

$$\begin{array}{rcl} \varepsilon_X(x) & = & x \\ \varepsilon_X(0) & = & 0 \\ \varepsilon_X(succ(t)) & = & succ\ \varepsilon_X(t) \\ \varepsilon_X(t+u) & = & +\,\varepsilon_X(t)\,\varepsilon_X(u) \\ \varepsilon_X(t\times u) & = & \times\,\varepsilon_X(t)\,\varepsilon_X(u) \end{array}$$

Note that in Σ_{FOL} there is no declaration for a type of variables; the variables of the object logic are identified with the variables of LF, as can be seen from the definition of ε_X. Thus, for example, the term $+(succ\ x)0$ in a context declaring $x{:}\iota$ represents the open term $succ(x) + 0$. This approach to syntax is fundamental to the treatment of quantification described below.

The relationship between the terms of first-order arithmetic and LF terms is made precise by the following theorem:

Theorem 3.1 (Adequacy for Syntax, I) *The encoding ε_X is a bijection between the terms of first-order arithmetic with free variables in X and the canonical forms of type ι in Σ_{FOL} and Γ_X. Moreover, the encoding is compositional in the sense that for $t[x_1, \ldots, x_n]$ a term with free variables in $X = \{ x_1, \ldots, x_n \}$ and $t_1, \ldots, t_n$ terms with free variables in Y,*

$$\varepsilon_Y(t[t_1, \ldots, t_n]) = [\varepsilon_Y(t_1), \ldots, \varepsilon_Y(t_n)/x_1, \ldots, x_n]\varepsilon_X(t).$$

Proof The encoding function ε_X is evidently injective and maps every term to a canonical form of type ι in Σ_{FOL} and Γ_X. Surjectivity is proved by defining a function δ_X that is left-inverse to ε_X. The function δ_X is defined by induction on the structure of the canonical forms as follows.

$$\begin{array}{rcl} \delta_X(x) & = & x \\ \delta_X(0) & = & 0 \\ \delta_X(\mathit{succ}\ M) & = & \mathit{succ}(\delta_X(M)) \\ \delta_X(+\, M_1\, M_2) & = & \delta_X(M_1) + \delta_X(M_2) \\ \delta_X(\times\, M_1\, M_2) & = & \delta_X(M_1) \times \delta_X(M_2) \end{array}$$

By Lemma 2.10 a canonical form M of type ι in Γ_X must have the form $\xi M_1 \ldots M_k$ for some constant or variable ξ, and some canonical $M_1, \ldots, M_k$, with k being the arity of ξ. By inspection of Σ_{FOL} and Γ_X, we see that the only choices for ξ are x in X, 0, succ, $+$, and $\times$. It follows from the types of these constants that δ is total and well-defined. It is easy to show by induction on the structure of t that $\delta_X(\varepsilon_X(t)) = t$. The compositionality property is shown by a straightforward structural induction on first-order terms. □

The formulas of first-order arithmetic are represented by introducing constants for each of the formula constructors as follows:

$$\begin{array}{rcl@{\qquad}rcl} = & : & \iota \to \iota \to o & < & : & \iota \to \iota \to o \\ \neg & : & o \to o & \wedge & : & o \to o \to o \\ \vee & : & o \to o \to o & \supset & : & o \to o \to o \\ \forall & : & (\iota \to o) \to o & \exists & : & (\iota \to o) \to o \end{array}$$

Formulas are encoded in LF by the function ε_X mapping formulas whose free variables are among those in X to LF terms of type o in $\Sigma_{\mathrm{FOL}}, \Gamma_X$, where Γ_X is as above. This encoding is defined by induction on the structure of formulas as follows:

$$\begin{array}{rcl} \varepsilon_X(t = u) & = & = \varepsilon_X(t)\, \varepsilon_X(u) \\ \varepsilon_X(t < u) & = & < \varepsilon_X(t)\, \varepsilon_X(u) \\ \varepsilon_X(\neg\varphi) & = & \neg\, \varepsilon_X(\varphi) \\ \varepsilon_X(\varphi \wedge \psi) & = & \wedge\, \varepsilon_X(\varphi)\, \varepsilon_X(\psi) \\ \varepsilon_X(\varphi \vee \psi) & = & \vee\, \varepsilon_X(\varphi)\, \varepsilon_X(\psi) \\ \varepsilon_X(\varphi \supset \psi) & = & \supset \varepsilon_X(\varphi)\, \varepsilon_X(\psi) \\ \varepsilon_X(\forall x.\varphi) & = & \forall\, (\lambda x{:}\iota.\varepsilon_{X,x}(\varphi)) \\ \varepsilon_X(\exists x.\varphi) & = & \exists\, (\lambda x{:}\iota.\varepsilon_{X,x}(\psi)) \end{array}$$

In the clauses for $\forall$ and $\exists$ we assume that x is chosen so that $x \notin X$, and we write X, x for $X \cup \{ x \}$.

Note how the encoding treats the quantifiers: the fact that $\forall$ and $\exists$ are binding operators is handled by their representation as functions of type $\iota \to o$. For example, the formula $\forall x.x = x$ is encoded as the term $\forall(\lambda x{:}\iota.{=}\, x\, x)$. This approach relies on the identification of the object logic variables with the variables of LF. As we remarked above, this allows us to avoid explicitly formalizing the machinery associated with binding operators.

The relationship between the formulas of first-order arithmetic and LF terms is made precise by the following theorem:

Theorem 3.2 (Adequacy for Syntax, II) *The encoding ε_X is a bijection between the formulas of first-order arithmetic with free variables among those in X and the canonical forms of type o in Σ_{FOL} and Γ_X. Moreover, the encoding is compositional in the sense that for $\varphi[x_1, \ldots, x_n]$ with free variables in $X = \{ x_1, \ldots, x_n \}$ and $t_1, \ldots, t_n$ with free variables in Y,*

$$\varepsilon_Y(\varphi[t_1, \ldots, t_n]) = [\varepsilon_Y(t_1), \ldots, \varepsilon_Y(t_n)/x_1, \ldots, x_n]\varepsilon_X(\varphi).$$

Proof The proof is similar to that for terms. It is easy to show by induction on the structure of formulas that ε_X yields a canonical form of the appropriate type. Consider, for example, the case of $\forall x.\varphi$. We have by induction that

$$\Gamma_{X,x} \vdash_{\Sigma_{\mathrm{FOL}}} \varepsilon_{X,x}(\varphi) : o.$$

Although $\Gamma_{X,x}$ is not necessarily of the form $\Gamma_X, x{:}\iota$, it follows from Theorem 2.3 that

$$\Gamma_X, x{:}\iota \vdash_{\Sigma_{\mathrm{FOL}}} \varepsilon_{X,x}(\varphi) : o$$

and therefore that

$$\Gamma \vdash_{\Sigma_{\mathrm{FOL}}} \forall\,(\lambda x{:}\iota.\varepsilon_{X,x}(\varphi)) : o.$$

The encoding ε_X is clearly injective. Surjectivity is established by defining a decoding map δ_X that is left-inverse to ε_X. The decoding δ_X is defined by induction on the structure of the canonical forms as follows:

$$\begin{aligned}
\delta_X(=M_1\,M_2) &= \delta_X(M_1) = \delta_X(M_2)\\
\delta_X(<M_1\,M_2) &= \delta_X(M_1) < \delta_X(M_2)\\
\delta_X(\neg\,M) &= \neg\delta_X(M)\\
\delta_X(\wedge\,M_1\,M_2) &= \delta_X(M_1) \wedge \delta_X(M_2)\\
\delta_X(\vee\,M_1\,M_2) &= \delta_X(M_1) \vee \delta_X(M_2)\\
\delta_X(\supset M_1\,M_2) &= \delta_X(M_1) \supset \delta_X(M_2)\\
\delta_X(\forall\,(\lambda x{:}\iota.M)) &= \forall x.\delta_{X,x}(M)\\
\delta_X(\exists\,(\lambda x{:}\iota.M)) &= \exists x.\delta_{X,x}(M)
\end{aligned}$$

As before, we tacitly assume that in the cases for the quantifiers that x has been chosen so that $x \notin X$. That δ_X is total follows from Lemma 2.10 and inspection of Σ_{FOL} and Γ_X. In particular, the fact that the domain consists only of canonical forms entails in case of the constants $\forall$ and $\exists$ that the argument must be a λ-abstraction of the indicated form. It is easy to show by induction on the structure of φ that $\delta_X(\varepsilon_X(\varphi)) = \varphi$ (for all X); for example,

$$\begin{aligned}
\delta_X(\varepsilon_X(\forall x.\varphi)) &= \delta_X(\forall(\lambda x{:}\iota.\varepsilon_{X,x}(\varphi)))\\
&= \forall x.\delta_{X,x}(\varepsilon_{X,x}(\varphi))\\
&= \forall x.\varphi
\end{aligned}$$

The compositionality property is established by a straightforward induction on the structure of first-order formulas. □

In addition to the correspondence between open formulas and open canonical terms, the treatment of syntax also provides a formal representation of the notions of schematic abstraction and instantiation. Specifically, we may regard a canonical term M of type o with a free variable of type o as an *incomplete* formula. By abstracting with respect to the free variable we obtain a *formula scheme* that may be *instantiated* by application. The rules of β-reduction correctly formalize a capture-avoiding form of schematic instantiation. In order to achieve the effect of capture-incurring schematic instantiation, we simply abstract with respect to variables of higher type. For example, the term

$$M = \lambda F{:}\iota{\to}o.{\supset}\,(\forall(\lambda x : \iota.F x))\,(\exists(\lambda x : \iota.F x))$$

may be viewed as a representation of the formula scheme $(\forall x.\varphi) \supset (\exists x.\varphi)$, where it is understood that φ may be instantiated to a formula with free occurrences of x. To instantiate this scheme, we apply it to a term of type $\iota \to o$; for example, $M(\lambda x{:}\iota.{=}\,x\,x)$ is definitionally equal to

$$\supset(\forall\,(\lambda x{:}\iota.{=}\,x\,x))\,(\exists\,(\lambda x{:}\iota.{=}\,x\,x))$$

which is the encoding of $(\forall x.x = x) \supset (\exists x.x = x)$.

3.2 Higher-Order Logic

There are many ways to present higher-order logic (see, for example, [6, 1, 51, 54]). For our version we follow Church in using the simply-typed λ-calculus [35] to form expressions of the logic. The language of simple functional types σ, τ is given by the following abstract syntax:

$$\sigma \quad ::= \quad \iota \mid o \mid \sigma \to \tau$$

where ι and o are *basic* types (of individuals and propositions). The language of expressions e, e' is given by the following abstract syntax:

$$e \quad ::= \quad x \mid c \mid \lambda_\sigma x.e \mid e\,e'$$

where x ranges over a countably infinite set of *variables*, and c over the set of *constants* which we take to be $0, succ, +, \times, <, \neg, \wedge, \vee, \supset, =_\sigma, \forall_\sigma, \exists_\sigma$ (one for each σ). Notions of α-equivalence, substitution, and $\beta\eta$-equivalence can be defined; α-equivalent expressions are treated as identical.

The *well-formed* expressions of type σ are defined relative to *assignments* $\mathcal{A}$ of the form $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ (with the x_i's all different); when we write $\mathcal{A}, x{:}\sigma$ we assume that x does not occur in $\mathcal{A}$. To do so one needs to have types for the constants, which are given as follows:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
0 & : & \iota & \wedge & : & o \to o \to o \\
succ & : & \iota \to \iota & \vee & : & o \to o \to o \\
+ & : & \iota \to \iota \to \iota & \supset & : & o \to o \to o \\
\times & : & \iota \to \iota \to \iota & =_\sigma & : & \sigma \to \sigma \to o \\
< & : & \iota \to \iota \to o & \forall_\sigma & : & (\sigma \to o) \to o \\
\neg & : & o \to o & \exists_\sigma & : & (\sigma \to o) \to o
\end{array}$$

We write $\mathcal{A} \vdash e : \sigma$ to mean that e has type σ relative to $\mathcal{A}$. Expressions of type o are the *formulas* denoted by φ and ψ, those of type ι are the *terms* , denoted by s and t), and those of type $\sigma \to \tau$ are the *functional expressions*. Note that in contrast to first-order logic, the quantifiers are constants of functional type in the object language, and hence must be applied using the application constructor of the language of higher-order logic.

The simple functional types are represented in LF using the following declarations of the signature Σ_{HOL}:

$$\begin{array}{lcl}
holtype & : & \mathsf{Type} \\
\iota & : & holtype \\
o & : & holtype \\
\Rightarrow & : & holtype \to holtype \to holtype
\end{array}$$

Note the change in notation to avoid confusion with that of the LF type theory. The type *holtype* is the type of higher-order logic types. This type contains the base types ι and o, and is closed under $\Rightarrow$, the function space constructor. There is an obvious bijective encoding ε of types as canonical forms of type *holtype* in Σ_{HOL}.

To represent expressions, we associate to each higher-order logic type an LF type of objects of that type, encoded by the constant *obj* of kind *holtype* $\to$ Type. The expressions of each type are defined by a set of constants corresponding to the expression constructors of higher-order logic. The representation of the quantifiers in LF makes use of dependent types in an essential way: the dependency of the type of the matrix on the domain of quantification cannot be directly expressed using only simple types in the sense of adequacy given above. The constant representing equality is similarly indexed by a type, and is therefore encoded using dependent types as well. The abstraction operator of higher-order logic is written "Λ" to avoid confusion with the abstraction operator of LF, and the application operator is made explicit. Thus we have the following declarations in Σ_{HOL}, using $\Rightarrow$ as an infix operator:

$$
\begin{array}{lll}
\mathit{obj} & : & \mathit{holtype} \rightarrow \mathsf{Type} \\
0 & : & \mathit{obj}(\iota) \\
\mathit{succ} & : & \mathit{obj}(\iota \Rightarrow \iota) \\
+ & : & \mathit{obj}(\iota \Rightarrow \iota \Rightarrow \iota) \\
\times & : & \mathit{obj}(\iota \Rightarrow \iota \Rightarrow \iota) \\
< & : & \mathit{obj}(\iota \Rightarrow \iota \Rightarrow o) \\
\neg & : & \mathit{obj}(o \Rightarrow o) \\
\wedge & : & \mathit{obj}(o \Rightarrow o \Rightarrow o) \\
\vee & : & \mathit{obj}(o \Rightarrow o \Rightarrow o) \\
\supset & : & \mathit{obj}(o \Rightarrow o \Rightarrow o) \\
= & : & \Pi s{:}\mathit{holtype}.\ \mathit{obj}(s \Rightarrow s \Rightarrow o) \\
\forall & : & \Pi s{:}\mathit{holtype}.\ \mathit{obj}((s \Rightarrow o) \Rightarrow o) \\
\exists & : & \Pi s{:}\mathit{holtype}.\ \mathit{obj}((s \Rightarrow o) \Rightarrow o) \\
\Lambda & : & \Pi s{:}\mathit{holtype}.\Pi t{:}\mathit{holtype}.(\mathit{obj}(s) \rightarrow \mathit{obj}(t)) \rightarrow \mathit{obj}(s \Rightarrow t) \\
\mathit{ap} & : & \Pi s{:}\mathit{holtype}.\Pi t{:}\mathit{holtype}.\ \mathit{obj}(s \Rightarrow t) \rightarrow \mathit{obj}(s) \rightarrow \mathit{obj}(t)
\end{array}
$$

In contrast to Church's formulation, both the domain and range type of an abstraction are explicitly attached to the representation of λ-abstractions and applications. This does not, however, introduce any complications in the proof of adequacy since the domain and range types are uniquely determined for each well-formed expression of functional type.

For every assignment $\mathcal{A} = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$, define the context $\Gamma_\mathcal{A}$ to be

$$x_1{:}\ \mathit{obj}(\varepsilon(\sigma_1)), \ldots, x_n{:}\ \mathit{obj}(\varepsilon(\sigma_n)).$$

For each assignment $\mathcal{A}$ and type σ, there is an encoding function $\varepsilon_{\mathcal{A},\sigma}$ mapping expressions of type σ relative to $\mathcal{A}$ to LF terms of type $\mathit{obj}(\varepsilon(\sigma))$ in Σ_{HOL} and $\Gamma_\mathcal{A}$. This encoding is defined as follows:

$$
\begin{array}{rcl}
\varepsilon_{\mathcal{A},\sigma}(x) & = & x \\
\varepsilon_{\mathcal{A},\iota\rightarrow\iota\rightarrow\iota}(+) & = & + \\
\varepsilon_{\mathcal{A},(\sigma\rightarrow o)\rightarrow o}(\forall_\sigma) & = & \forall\varepsilon(\sigma)
\end{array}
$$

and similarly for other constants,

$$
\begin{array}{rcl}
\varepsilon_{\mathcal{A},\sigma\rightarrow\tau}(\lambda_\sigma x.e) & = & \Lambda\,\varepsilon(\sigma)\,\varepsilon(\tau)\,(\lambda x{:}\ \mathit{obj}(\varepsilon(\sigma)).\varepsilon_{(\mathcal{A},x:\sigma),\tau}(e)) \\
\varepsilon_{\mathcal{A},\tau}(ee') & = & \mathit{ap}\ \varepsilon(\sigma)\varepsilon(\tau)\varepsilon(e)\varepsilon(e')
\end{array}
$$

where in the last clause $\mathcal{A} \vdash e : \sigma \rightarrow \tau$.

Theorem 3.3

1. *The encoding ε is a bijection between the simple functional types and the canonical forms of type holtype in Σ_{HOL} and the empty context. Moreover, the encoding is compositional in that $\varepsilon(\sigma\rightarrow\tau) = \varepsilon(\sigma) \Rightarrow \varepsilon(\tau)$.*

2. *For each assignment $\mathcal{A}$ and each type σ, the encoding $\varepsilon_{\mathcal{A},\sigma}$ is a bijection between the expressions of type σ relative to $\mathcal{A}$, and the canonical forms of type $\mathit{obj}(\varepsilon(\sigma))$ in Σ_{HOL} and $\Gamma_\mathcal{A}$. Moreover, the encoding is compositional in the sense that for every expression $e[x_1, \ldots, x_n]$ of type σ relative to the assignment $\mathcal{A} = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ and expressions $e_1, \ldots, e_n$ of types $\sigma_1, \ldots, \sigma_n$ relative to the assignment $\mathcal{B}$,*

$$\varepsilon_{\mathcal{B},\sigma}(e[e_1, \ldots, e_n]) = [\varepsilon_{\mathcal{B},\sigma_1}(e_1), \ldots, \varepsilon_{\mathcal{B},\sigma_n}(e_n)/x_1, \ldots, x_n]\varepsilon_{\mathcal{A},\sigma}(e).$$

Proof Similar to that for first-order logic. Note that the second case covers terms, formulas, and functional expressions. $\square$

4 Theory of Rules and Proofs

The treatment of rules and proofs lies at the heart of LF. The approach is organized around the notion of a *judgement* (or *assertion*) [30]. Logics are viewed as systems for generating proofs of judgements. Each logic has a characteristic set of *basic* (or *atomic*) judgements. For example, in first-order logic there is one form of basic judgement, the assertion that a formula φ is true, written φ *true*. The basic judgements are uniformly extended with two *higher-order* judgements, the *hypothetical* and the *schematic* (or *general*). If J_1 and J_2 are judgements, then the hypothetical judgement $J_1 \vdash J_2$ expresses a form of consequence: J_2 is provable under the assumption of J_1. If C is a category of the language and $J(x)$ is a judgement involving a variable x ranging over C, then the schematic judgement $\bigwedge_{x \in C} J(x)$ expresses a form of generality: $J(x)$ is provable uniformly in x. The set of proofs of the logic is generated by a collection of *rule schemes*, which are viewed as functions mapping syntactic entities and proofs to proofs. Higher-order judgements are central to our treatment of systems of natural deduction. Whereas rules of pure Hilbert-type systems take as arguments proofs of basic judgements, we shall see below that rules in systems of natural deduction may be understood as taking proofs of higher-order judgements as arguments.

It is worthwhile to compare hypothetical judgements with *consequence relations* [2]. Much of traditional logic (Martin-Löf's work being a notable, and inspirational, exception) is based on using a single form of basic judgement (for which no syntactic indication is needed), and attention focuses on notions of consequence between sets or multisets of sentences (basic judgements). Consequence relations arise in a variety of ways, and there are often several different consequence relations of interest for a logical system. For example, in first-order logic there are the *truth* and *validity* consequence relations which differ in the scoping of free variables (the consequence $\varphi(x) \vdash \forall x.\varphi$ is correct under the validity interpretation, but it is incorrect under the truth interpretation). To take another example, propositional modal systems such as S4 also have two consequence relations of interest also called (somewhat confusingly) *truth* and *validity*, the former expressing consequence in each world, the latter expressing consequence for all worlds (see [2] and [4] for further details).

Hypothetical judgements are in one sense more general and in another less general than consequence relations. The additional generality stems from the ability to consider "hybrid" notions of consequence stemming from the consideration of multiple basic judgements. For example, in S4 modal logic one may consider two forms of basic judgement, φ *true* and φ *valid*, which allow the expression of a consequence such as φ *valid* $\vdash \Box\varphi$ *true* which is neither an instance of the truth nor of the validity consequence relation. On the other hand, hypothetical judgements are less general in that they are given a *fixed* interpretation in every logical system as the "external" consequence relation associated with provability from assumptions [2]. (The exact interpretation will be made precise shortly.) Although the hypothetical judgement form is always available, it is by no means immediate that it corresponds to a given consequence relation for that logical system: this is established by the adequacy theorem for the representation, at least as far as proofs of basic judgements and proofs of consequence are concerned.

The choice of type system for LF is well motivated by considering the representation of rules and proofs. The LF methodology for representing rules and proofs is based on a new principle, the *judgements-as-types* principle, under which judgements are represented as the type of their proofs. To each basic judgement form is associated a family of types (indexed by the type corresponding to the syntactic category of the subject of the judgement). For example, in first-order logic the basic judgement form φ *true* is represented by the type

$$\overline{\varphi\ \mathit{true}} = \mathit{true}(\varepsilon(\varphi)),$$

where $\mathit{true} : o \to \mathsf{Type}$ is a constant of the signature Σ_{FOL}[1]. The higher-order forms are available under the correspondences

$$\overline{J_1 \vdash J_2} = \overline{J_1} \to \overline{J_2}$$

and

$$\overline{\bigwedge_{x \in C} J(x)} = \Pi x{:}\overline{C}.\overline{J(x)}$$

where $\overline{C}$ is to be a type representing the category C.

[1]Closely related ideas appear in the AUTOMATH literature. For example in [15] section 23 a type of proofs of boolean expressions is considered.

The type system is further exploited in the encoding of inference rules. To each primitive rule is associated a constant of higher type, with arguments the values of the parameters and the proofs of the premises. As we shall see below, the context-sensitive discharge and variable-occurrence conditions associated with systems of natural deduction are directly representable using the LF type system, in contrast to the explicit side conditions and discharge conventions of the usual "first-order" presentations. In this way proofs are represented by terms of judgement type, and proof checking is reduced to type checking.

Under the judgements-as-types principle, proofs of higher-order judgements are LF terms of functional type. This has two significant consequences. First, rules are simply proofs of higher-order judgement type. Although primitive rules are represented by constants, any term of higher-order judgement type may be viewed as a rule of inference. In fact, derived rules may well be representable as terms of higher type, as we shall see below. In LF there is no distinction between rules and proofs. Second, the structural properties of the LF type theory stated in Section 2 (and proved in the appendix) provide important information about the potential of the above encoding of consequence. For if we expect to encode (as in the case of first-order logic) $\varphi_1, \ldots, \varphi_n \vdash \varphi$ as the type

$$\mathit{true}(\varepsilon(\varphi_1)) \to \cdots \to \mathit{true}(\varepsilon(\varphi_n)) \to \mathit{true}(\varepsilon(\varphi)),$$

then the consequence relation must satisfy weakening and contraction due to the properties of the LF function type. Specifically, if we have a term of the above type, then there is also a term of type

$$\mathit{true}(\varepsilon(\varphi_1)) \to \cdots \to \mathit{true}(\varepsilon(\varphi_n)) \to \mathit{true}(\varepsilon(\varphi_{n+1}) \to \mathit{true}(\varepsilon(\varphi)).$$

Similarly, if we have a term of type

$$\mathit{true}(\varepsilon(\varphi_1)) \to \cdots \to \mathit{true}(\varepsilon(\varphi_n)) \to \mathit{true}(\varepsilon(\varphi_n) \to \mathit{true}(\varepsilon(\varphi))$$

then we also have a term of type

$$\mathit{true}(\varepsilon(\varphi_1)) \to \cdots \to \mathit{true}(\varepsilon(\varphi_n)) \to \mathit{true}(\varepsilon(\varphi)).$$

Weakening and contraction are not satisfied by relevance and linear logics; in these cases either the encoding, the treatment of consequence, or, perhaps, LF itself must be changed. However, the consequence relation induced by a pure Hilbert system or a system of natural deduction is correctly captured by the above correspondences. It is a thesis of the LF approach that pure Hilbert systems or systems of natural deduction can be adequately represented in LF. By "pure" we mean that there are no non-local applicability conditions on rules such as are associated with rules of proof which may only be applied to premises that do not depend on assumptions. (See Avron [2] for much further discussion of this point.)

We stress that the judgements-as-types principle is more general than the propositions-as-types principle of Curry, Howard, and deBruijn: the judgements-as-types principle is merely the formal expression of the fact that the assertion of a judgement in a formal system is precisely the claim that it has a formal proof in that system. This principle is correct for any formal system, not just those associated with intuitionistic logic. It might be argued, however, that we are employing the propositions-as-types principle at the level of LF, rather than at the level of the object logic, since we are, in effect, treating the higher-order judgement forms as the intuitionistic connectives associated with the (simple and dependent variants of the) function space constructor of LF.

To illustrate these ideas, we consider the representation of first- and higher-order logic in LF. In each case we present a natural deduction formulation of the system. For the sake of brevity, we consider only a representative selection of the rules; the remaining cases do not present any additional difficulties.

4.1 First-Order Logic

We first give a rigorous account of proofs in natural deduction [43]; this will enable us later to give a precise statement of adequacy. Roughly speaking, a proof in natural deduction is a tree with nodes labeled by inference rules and the values of their parameters, together with a discharge function assigning rule occurrences to hypothesis occurrences specifying which hypothesis occurrences are discharged by which rule

$$\text{(RAA)} \quad \frac{\neg\neg\varphi}{\varphi} \qquad\qquad \text{(IMP-I)} \quad \frac{\begin{array}{c}(\varphi)\\ \psi\end{array}}{\varphi \supset \psi}$$

$$\text{(ALL-I}^*\text{)} \quad \frac{\varphi[x]}{\forall x.\varphi[x]} \qquad\qquad \text{(ALL-E)} \quad \frac{\forall x.\varphi[x]}{\varphi[t]}$$

$$\text{(SOME-E}^\dagger\text{)} \quad \frac{\exists x.\varphi[x] \quad \begin{array}{c}(\varphi)\\ \psi\end{array}}{\psi}$$

*Provided that x is not free in any assumption on which φ depends.
†Provided that x is not free in ψ or any assumptions, other than φ, on which ψ depends.

Table 4: Some Rules of First-Order Logic

occurrence. A proof is valid iff each rule occurrence is a correct instance of the rule scheme that labels it (in particular, if the side conditions on the applicability of the rule are satisfied.) An illustrative selection of rule schemes from the natural deduction formulation of first-order logic (taken from [43]) is given in Table 4. This presentation employs the "parenthesis convention" on the rules IMP-I and SOME-E to indicate that zero or more occurrences of the indicated hypothesis are discharged by an application of the rule. Note, however, that what, if any, hypotheses are discharged by the application are not made explicit in the rule, but instead are indicated using a discharge function. It is worth remarking that the device of discharge functions is necessitated by the fact that it is hypothesis *occurrences*, and not *hypotheses*, that are discharged: an inference that discharges φ may leave certain occurrences of φ undischarged. For the sake of precision in the proof of adequacy below, we now proceed to give a formal definition of proofs in natural deduction.

To begin with, we introduce a language of *proof expressions* Π defined by the following grammar:[2]

$$\begin{array}{rcl}\Pi & ::= & \text{HYP}_\varphi(\xi) \mid \text{RAA}_\varphi(\Pi) \mid \text{IMP-I}_{\varphi,\psi}(\xi{:}\Pi) \mid \text{ALL-I}_{x,\varphi}(\Pi) \mid \\ & & \text{ALL-E}_{x,\varphi,t}(\Pi) \mid \text{SOME-E}_{x,\varphi,\psi}(\Pi', \xi{:}\Pi)\end{array}$$

Here ξ ranges over a countably infinite set of *occurrence markers* (disjoint from the set of first-order variables). The idea is that when a hypothesis is discharged, a certain set of its occurrences is discharged and all these are marked by a particular ξ. The markers play a variable-like role. In the above grammar, the explicitly written x and ξ are binding occurrences: in cases ALL-I, ALL-E, and SOME-E, occurrences of x in φ are bound, in cases ALL-I and SOME-E, occurrences of x in Π are bound, and in cases IMP-I and SOME-E, occurrences of ξ in Π are bound; all other occurrences are free. We do not distinguish proof expressions that differ only in the choice of bound variables and bound occurrence markers. One can substitute terms $t_1, \ldots, t_m$ and proof expressions $\Pi_1, \ldots, \Pi_n$ for all free occurrences of $x_1, \ldots, x_m$ and $\xi_1, \ldots, \xi_n$ in a proof expression $\Pi[x_1, \ldots, x_m, \xi_1, \ldots, \xi_n]$ obtaining $\Pi[t_1, \ldots, t_m, \Pi_1, \ldots, \Pi_n]$; we omit the evident formal definition. The *end formula* of a proof expression is defined as follows: φ in cases HYP and RAA, $\varphi \supset \psi$ in case IMP-I, $\forall x.\varphi$ in case ALL-I, $\varphi[t]$ in case ALL-E, and ψ in case SOME-E. Note that in case ALL-E it is necessary that t appear as a parameter of the rule since the possibility of vacuous substitution precludes recovering it from the other parameters.

Not all proof expressions are valid (just as not all LF terms were valid). A *proof context* is a pair (X, Δ) with X a finite set of variables of first-order logic and Δ a sequence $\xi_1{:}\varphi_1, \ldots, \xi_n{:}\varphi_n$ with all ξ_i's different and every free variable occurring in a φ_i in X. We write $\mathrm{FV}(\Delta)$ for the set of free variables of the formulas φ_i in Δ, and $\mathrm{dom}(\Delta)$ for the set of ξ_i's in Δ. In Table 5 we give rules for proving assertions of the form $X, \Delta \vdash \Pi : \varphi$, which is to be read as "Π is a valid proof of φ with respect to the proof context (X, Δ)." The derivability of such an assertion means that a number of general rules are obeyed (*e.g.*no two occurrences of a ξ mark different formulas) and that the restrictions in the rules of Table 4 are also obeyed. We say that Π is valid with respect to the proof context (X, Δ) iff $X, \Delta \vdash \Pi : \varphi$ holds for some φ (necessarily the

[2]We consider only the selection of rules given in Table 4.

$$\frac{\xi{:}\varphi \in \Delta}{X, \Delta \vdash \textsc{hyp}_{\varphi}(\xi) : \varphi} \qquad (\textsc{v-hyp})$$

$$\frac{X, \Delta \vdash \Pi : \neg\neg\varphi}{X, \Gamma \vdash \textsc{raa}_{\varphi}(\Pi) : \varphi} \qquad (\textsc{v-not-e})$$

$$\frac{X, (\Delta, \xi{:}\varphi) \vdash \Pi : \psi \quad \xi \notin \mathrm{dom}(\Delta)}{X, \Delta \vdash \textsc{imp-i}_{\varphi,\psi}(\Pi) : \varphi \supset \psi} \qquad (\textsc{v-imp-i})$$

$$\frac{(X, x), \Delta \vdash \Pi : \varphi \quad x \notin X}{X, \Delta \vdash \textsc{all-i}_{x,\varphi}(\Pi) : \forall x.\varphi} \qquad (\textsc{v-all-i})$$

$$\frac{X, \Delta \vdash \Pi : \forall x.\varphi[x]}{X, \Delta \vdash \textsc{all-e}_{x,\varphi,t}(\Pi) : \varphi[t]} \qquad (\textsc{v-all-e})$$

$$\frac{X, \Delta \vdash \Pi' : \exists x.\varphi \quad (X, x), (\Delta, \xi{:}\varphi) \vdash \Pi : \psi \quad x \notin X, \xi \notin \mathrm{dom}(\Delta)}{X, \Delta \vdash \textsc{some-e}_{x,\varphi,\psi}(\Pi', \xi{:}\Pi) : \psi} \qquad (\textsc{v-some-e})$$

Table 5: Valid Proof Expressions

end formula of Π). There is a minimal such proof context, if any exists at all: take X to be the set of all free variables in Π, take Δ to be a list of the set of all the $\xi{:}\varphi$ such that $\textsc{hyp}_{\varphi}(\xi)$ is a sub-expression of Π (if two different φ's appear with the same ξ, then no such context exists anyway). Validity is preserved under substitution in the sense that if $\Pi[x_1, \ldots, x_m, \xi_1, \ldots, \xi_n]$ is a valid proof of $\varphi[x_1, \ldots, x_m]$ with respect to $(\{x_1, \ldots, x_m\}, (\xi_1{:}\varphi_1, \ldots, \xi_n{:}\varphi_n))$ and if $t_1, \ldots, t_m$ are terms whose variables are all in X' and if $\Pi_1, \ldots, \Pi_n$ are valid proofs of $\varphi_1, \ldots, \varphi_n$ with respect to (X', Δ'), then $\Pi[t_1, \ldots, t_m, \Pi_1, \ldots, \Pi_n]$ is a valid proof of $\varphi[t_1, \ldots, t_m]$ with respect to (X', Δ').

We turn now to the presentation of first-order logic in LF. As remarked above, there is one basic judgement form, the assertion that a formula φ is a logical truth; it is represented by including in Σ_{FOL} the declaration

$$\mathit{true} : o \rightarrow \mathsf{Type}$$

If φ is a formula, then a proof of φ in first-order logic is represented by a term of type $\mathit{true}(\varepsilon(\varphi))$ in a context providing declarations for the free variables and undischarged assumptions of the proof.

Each of the rules in Table 4 is represented by a constant of the signature Σ_{FOL} whose type encodes the variable-occurrence and discharge conditions associated with the rule. In effect, we shift the burden of implementing the machinery of natural deduction from the object- to the meta-level, just as we shifted the burden of implementing the machinery of binding operators from the object- to the meta-level in Section 3. In what follows we give the representation of each rule in LF, together with a brief justification for the choice of representation. The precise correspondence between natural deduction and its representation in LF is the content of Theorem 4.1 below.

The *reductio ad absurdum* rule is represented by the declaration:

$$\textsc{raa} : \Pi p{:}o.\mathit{true}(\neg\neg p) \rightarrow \mathit{true}(p)$$

This rule is schematic in the proposition p, and has as premise a term of type $\mathit{true}(\neg\neg p)$. Hence if φ is a proposition and M is a term of type $\mathit{true}(\neg\neg\varepsilon(\varphi))$, then $\textsc{raa}\ \varepsilon(\varphi)\ M$ is a term of type of $\mathit{true}(\varepsilon(\varphi))$.

The implication introduction rule uses non-dependent function types to model discharge. It is represented by the declaration:

$$\textsc{imp-i} : \Pi p{:}o.\Pi q{:}o.(\mathit{true}(p) \rightarrow \mathit{true}(q)) \rightarrow \mathit{true}(\supset p\, q)$$

One might say that the formulation of IMP-I in Table 4 takes as premise a *hypothetical proof* of ψ and discharges some occurences of the hypothesis φ, whereas the formulation of IMP-I in LF takes as premise a *proof of a hypothetical judgement*, thereby shifting the handling of discharge to LF. For example,

$$\textsc{imp-i}\ \varepsilon(\varphi)\,\varepsilon(\varphi)\,(\lambda x{:}\mathit{true}(\varepsilon(\varphi)).x)$$

is the LF encoding of a proof of $\varphi \supset \varphi$ in first-order logic.

The elimination rule for the universal quantifier is represented by the declaration:

$$\textsc{all-e} : \Pi F{:}\iota{\to}o.\Pi x{:}\iota.\mathit{true}(\forall(\lambda x{:}\iota.Fx)) \to \mathit{true}(Fx)$$

This rule has two parameters, one for the matrix of the universally quantified proposition, represented by the variable F, and one for the term to instantiate it , represented by the variable x. The rule also takes as argument a term which represents a proof of the universal proposition, and the result is a term which represents a proof of the instance. For example, if F is $\lambda x{:}\iota.\varepsilon(\varphi[x])$, and if M is a term of type $\mathit{true}(\forall\,(\lambda x{:}\iota.Fx))$, then ALL-E $F\,0\,M$ is a term of type $\mathit{true}(F\,0)$, which is definitionally equal to $\mathit{true}([0/x]\varepsilon(\varphi))$, the encoding of $\varphi[0]$. Thus we see that substitution is modeled by β-conversion, and that the rule B-CONV-OBJ is needed to show that we have a term of type $\mathit{true}(\varepsilon(\varphi[0]))$.

The variable-occurrence condition associated with the rule ALL-I is represented using the dependent function space type of LF:

$$\textsc{all-i} : \Pi F{:}\iota{\to}o.(\Pi x{:}\iota.\mathit{true}(Fx)) \to \mathit{true}(\forall(\lambda x{:}\iota.Fx))$$

One may say that in natural deduction ALL-I takes as premise a schematic (in x) proof of a judgement, whereas in LF it takes as premise a proof of a schematic judgement, shifting the enforcement of the variable-occurrence condition from the object logic to LF. Note the similarity between the encoding of ALL-I and IMP-I stemming from the fact that the non-dependent function space is a special case of the dependent function: variable-occurrence and discharge conditions are closely related.

The existential elimination rule has both discharge and variable-occurrence conditions:

$$\textsc{some-e} : \Pi F{:}\iota{\to}o.\Pi p{:}o.\mathit{true}(\exists(\lambda x{:}\iota.Fx)) \to (\Pi x{:}\iota.\mathit{true}(Fx) \to \mathit{true}(p)) \to \mathit{true}(p)$$

Note that the variable-occurrence condition on the conclusion of the SOME-E rule is a matter of scoping: since p is bound outside of the scope of x, no instance of p can have x free, as required by the existential elimination rule.

To encode natural deduction proofs as LF terms we assume that the LF variables include the first-order variables and the occurrence markers. For each proof context (X, Δ) we inductively define a function $\varepsilon_{X,\Delta}$ from valid proofs with respect to (X, Δ) (defined in Table 5) to LF terms as follows:

$$\begin{aligned}
\varepsilon_{X,\Delta}(\textsc{hyp}_\varphi(\xi)) &= \xi \\
\varepsilon_{X,\Delta}(\textsc{raa}_\varphi(\Pi)) &= \textsc{raa}\ \varepsilon_X(\varphi)\,\varepsilon_{X,\Delta}(\Pi) \\
\varepsilon_{X,\Delta}(\textsc{imp-i}_{\varphi,\psi}(\xi{:}\Pi)) &= \textsc{imp-i}\ \varepsilon_X(\varphi)\,\varepsilon_X(\psi)\,\lambda\xi{:}\mathit{true}(\varepsilon_X(\varphi)).\varepsilon_{X,(\Delta,\xi:\varphi)}(\Pi) \\
\varepsilon_{X,\Delta}(\textsc{all-i}_{x,\varphi}(\Pi)) &= \textsc{all-i}\,(\lambda x{:}\iota.\varepsilon_{X,x}(\varphi))\,(\lambda x{:}\iota.\varepsilon_{(X,x),\Delta}(\Pi)) \\
\varepsilon_{X,\Delta}(\textsc{all-e}_{x,\varphi,t}(\Pi)) &= \textsc{all-e}\,(\lambda x{:}\iota.\varepsilon_{X,x}(\varphi))\,\varepsilon_X(t)\,\varepsilon_{X,\Delta}(\Pi) \\
\varepsilon_{X,\Delta}(\textsc{some-e}_{x,\varphi,\psi}(\Pi', \xi{:}\Pi)) &= \textsc{some-e}\,(\lambda x{:}\iota.\varepsilon_{X,x}(\varphi))\,\varepsilon_X(\psi)\,\varepsilon_{X,\Delta}(\Pi') \\
&\quad\ (\lambda x{:}\iota.\lambda\xi{:}\mathit{true}(\varepsilon_{X,x}(\varphi)).\varepsilon_{(X,x),(\Delta,\xi:\varphi)}(\Pi))
\end{aligned}$$

In the cases involving individual variables x and occurrence markers ξ, it is assumed that x and ξ are chosen to be the first such variable or occurrence marker (in some standard enumeration) not occurring in X or Δ, respectively.

To illustrate the encoding, consider the proof

$$\Pi = \textsc{imp-i}_{\varphi,\psi\supset\varphi}(\xi{:}\, \textsc{imp-i}_{\psi,\varphi}(\xi'{:}\, \textsc{hyp}_{\varphi}(\xi)))$$

of $\varphi \supset (\psi \supset \varphi)$ in the empty proof context. (The discharge at the second occurence of IMP-I is vacuous). The encoding of Π with respect to the empty proof context is the canonical LF term

$$\textsc{imp-i}\ \varepsilon(\varphi)\, \varepsilon(\psi \supset \varphi)\, (\lambda\xi{:}\mathit{true}(\varepsilon(\varphi)).\, \textsc{imp-i}\ \varepsilon(\psi)\, \varepsilon(\varphi)\, (\lambda\xi'{:}\mathit{true}(\varepsilon(\psi)).\xi)).$$

It is easy to verify that this term is of type $\mathit{true}(\varepsilon(\varphi{\supset}(\psi{\supset}\varphi)))$ in Σ_{FOL} and the empty context. Notice that the vacuous discharge in the natural deduction proof corresponds to the non-occurrence of ξ' in the body of the innermost λ-abstraction in the corresponding LF term.

We now give the relationship between the valid proofs in first-order logic and the terms of LF. First some notation: if (X, Δ) is a proof context with $X = \{ x_1, \ldots, x_m \}$ and $\Delta = (\xi_1{:}\varphi_1, \ldots, \xi_n{:}\varphi_n)$, then $\Gamma_{X,\Delta}$ is the LF context

$$x_1{:}\iota, \ldots, x_m{:}\iota, \xi_1{:}\mathit{true}(\varepsilon_X(\varphi_1)), \ldots, \xi_n{:}\mathit{true}(\varepsilon_X(\varphi_n)).$$

Theorem 4.1 (Adequacy for Proofs, I) *For every first-order formula φ, the encoding $\varepsilon_{X,\Delta}$ is a bijection between valid proofs of φ with respect to (X, Δ) to canonical terms of type $\mathit{true}(\varepsilon_X(\varphi))$ in Σ_{FOL} and $\Gamma_{X,\Delta}$. Furthermore, $\varepsilon_{X,\Delta}$ is compositional in the sense that for any proof contexts (X, Δ) and (X', Δ') with $X = \{ x_1, \ldots, x_m \}$ and $\Delta = (\xi_1{:}\varphi_1, \ldots, \xi_n{:}\varphi_n)$, if $t_1, \ldots, t_m$ are first-order terms whose variables are all in X' and if $\Pi_1, \ldots, \Pi_n$ are valid proofs of $\varphi_1, \ldots, \varphi_n$ with respect to (X', Δ'), then for any proof expression $\Pi[x_1, \ldots, x_m, \xi_1, \ldots, \xi_n]$ valid with respect to (X, Δ),*

$$\begin{gathered}\varepsilon_{X',\Delta'}(\Pi[t_1, \ldots, t_m, \Pi_1, \ldots, \Pi_n]) \\ = \\ [\varepsilon_{X'}(t_1), \ldots, \varepsilon_{X'}(t_m), \varepsilon_{X',\Delta}(\Pi_1), \ldots, \varepsilon_{X',\Delta'}(\Pi_n)/x_1, \ldots, x_m, \xi_1, \ldots, \xi_n]\varepsilon_{X,\Delta}(\Pi).\end{gathered}$$

Proof It is straightforward to verify by induction on the structure of proof expressions that, given the hypothesis of the theorem, $\varepsilon_{X,\Delta}(\Pi)$ is a canonical term of type $\mathit{true}(\varepsilon_X(\varphi))$ in Σ_{FOL} and $\Gamma_{X,\Delta}$. For example, consider the case $\varepsilon_{X,\Delta}(\textsc{imp-i}_{\varphi,\psi}(\xi{:}\Pi))$. Then Π is a valid proof of ψ wrt $(X, (\Delta, \xi{:}\varphi))$. So, by induction hypothesis, $\varepsilon_{X,(\Delta,\xi:\varphi)}(\Pi)$ is a canonical term of type $\mathit{true}(\varepsilon_X(\psi))$ in Σ_{FOL} and $\Gamma_{X,(\Delta,\xi:\varphi)}$, and the result follows. Again, consider the case $\varepsilon_{X,\Delta}(\textsc{all-i}_{x,\varphi}(\Pi))$. Then Π is a valid proof of φ wrt $((X, x), \Delta)$ and $x \notin X$ (hence $x \notin \mathrm{FV}(\Delta)$ because $\mathrm{FV}(\Delta) \subseteq X$). By induction hypothesis $\varepsilon_{(X,x),\Delta}(\Pi)$ is a canonical term of type $\mathit{true}(\varepsilon_X(\varphi))$ in Σ_{FOL} and $\Gamma_{(X,x),\Delta}$, and so also in Σ_{FOL} and $\Gamma_{X,\Delta}, x{:}\iota$ (by permutation, since $x \notin \mathrm{FV}(\Delta)$); the result then follows.

It is a routine matter to show by induction on proof expressions that $\varepsilon_{X,\Delta}$ is injective. To establish surjectivity we exhibit a left-inverse $\delta_{X,\Delta}$ defined by induction on the structure of the canonical forms as follows:

$$\begin{array}{rcl}
\delta_{X,\Delta}(\xi) & = & \textsc{hyp}_{\delta_X(M)}(\xi) \text{ where } \xi{:}\mathit{true}(M) \in \Gamma_{X,\Delta} \\[2ex]
\delta_{X,\Delta}(\textsc{raa}\ M\, P) & = & \textsc{raa}_{\delta_X(M)}(\delta_{X,\Gamma}(P)) \\[2ex]
\delta_{X,\Delta}(\textsc{imp-i}\ M\, N\, (\lambda\xi{:}\mathit{true}(M).P)) & = & \textsc{imp-i}_{\delta_X(M),\delta_X(N)}(\xi{:}\delta_{X,(\Delta,\xi:\delta_X(N))}(\Pi)) \\[2ex]
\delta_{X,\Delta}(\textsc{all-i}\, (\lambda x{:}\iota.M)\, (\lambda x{:}\iota.P)) & = & \textsc{all-i}_{x,\delta_{X,x}(M)}(\delta_{(X,x),\Delta}(P)) \\[2ex]
\delta_{X,\Delta}(\textsc{all-e}\, (\lambda x{:}\iota.M)\, N\, P) & = & \textsc{all-e}_{x,\delta_{X,x}(M),\delta_X(N)}(\delta_{X,\Delta}(P)) \\[2ex]
\delta_{X,\Delta}\left(\begin{array}{c}\textsc{some-e}\, (\lambda x{:}\iota.M)\, N\, P' \\ (\lambda x{:}\iota.\lambda\xi{:}\mathit{true}(M).P)\end{array}\right) & = & \textsc{some-e}_{x,\delta_{X,x}(M),\delta_X(N)}(\Pi', \xi{:}\Pi) \\
& & \text{where } \Pi' = \delta_{X,\Delta}(P'), \\
& & \text{and } \Pi = \delta_{(X,x),(\Delta,\xi:\delta_{X,x}(M))}(\Pi)
\end{array}$$

Here x and ξ are chosen as before to be the first (in some standard enumeration) variable or occurrence marker not already in the proof context.

That $\delta_{X,\Delta}$ is total and well-defined follows from the definition of canonical forms and inspection of the signature Σ_{FOL}, together with the definition of validity of proof expressions. It remains to show that $\delta_{X,\Delta}(\varepsilon_{X,\Delta}(\Pi)) = \Pi$. This is proved by induction on the structure of Π. We just illustrate the proof with the case of implication introduction:

$$\begin{array}{rcl} \delta_{X,\Delta}(\varepsilon_{X,\Delta}(\text{IMP-I}_{\varphi,\psi}(\Pi))) & = & \delta_{X,\Delta}(\text{IMP-I}\ \varepsilon_X(\varphi)\ \varepsilon_X(\psi)\ (\lambda\xi{:}true(\varepsilon_X(\varphi)).\varepsilon_{X,(\Delta,\xi:\varphi)}(\Pi))) \\ & = & \text{IMP-I}_{\delta_X(\varepsilon_X(\varphi)),\delta_X(\varepsilon_X(\psi))}(\xi{:}\delta_{X,(\Delta,\xi:\delta_X(\varepsilon_X(\varphi))}(\varepsilon_{X,(\Delta,\xi:\varphi)}(\Pi)) \\ & = & \text{IMP-I}_{\varphi,\psi}(\xi{:}\Pi) \end{array}$$

Note that by the conventions on bound variables and bound occurrence markers, we may assume that ξ is chosen to be the same in both the LF term and in the proof expression.

The compositionality of the encoding is established by another straightforward induction on the relevant proof expression, Π. □

It is important to stress that the way in which we have defined the set of free variables in a proof is crucial to the correctness of the adequacy theorem. For example,

$$\text{IMP-I}_{\forall x.\varphi,\exists x.\varphi}(\xi{:}\ \text{SOME-I}_{x,\varphi,y}(\text{ALL-E}_{x,\varphi,y}(\text{HYP}_{\forall x.\varphi}(\xi)))$$

is a valid proof of $(\forall x.\varphi) \supset (\exists x.\varphi)$.[3] The variable y occurs free in the proof (as the argument to ALL-E), but not in its end formula. Nevertheless, it must be accounted for in the LF encoding, otherwise the resulting term is not well-typed. By glossing over such details the usual presentations of systems of natural deduction appear to "build-in" the assumption that the domain of quantification is non-empty, but we see from the LF representation that this is not the case.

The adequacy theorem deals with pure first-order logic over the language of Peano arithmetic, but does not deal with the logic of equality or arithmetic. One way to do this is to add suitable axiom and rule schemes to the natural deduction system sketched above. For example, we could add the substitution rule

$$(\text{SUB}) \quad \frac{t = u \quad \varphi[t]}{\varphi[u]}$$

and the induction rule

$$(\text{IND}) \quad \frac{\varphi[0] \quad \begin{array}{c}(\varphi[x]) \\ \varphi[succ(x)]\end{array}}{\forall x.\varphi[x]}$$

(with the side condition that x not occur free in any assumption other than those discharged by the application of the rule). These rules are presented in LF by the declarations

$$\text{SUB} : \Pi t{:}\iota.\Pi u{:}\iota.\Pi\varphi{:}\iota{\to}o.true(t{=}v) \to true(\varphi\, t) \to true(\varphi\, u)$$

and

$$\text{IND} : \Pi\varphi{:}\iota{\to}o.true(\varphi\, 0) \to (\Pi x{:}\iota.true(\varphi\, x) \to true(\varphi\,(succ\ x))) \to true(\forall\,(\lambda x{:}\iota.\varphi\, x))$$

It is straightforward to extend the above formal treatment of natural deduction to include equality and arithmetic. The encoding of proofs can then be suitably extended , and an adequacy theorem for the representation in LF can be obtained. It would be interesting to etablish an adequacy theorem for a general notion of extension of first-order logic by additional axiom schemes and rules of inference.

The adequacy theorem is a minimal correctness criterion, and does not delineate the extent to which the type structure of LF may be exploited in representing forms of inference that are not characteristic of the logical system being represented. For example, it is possible to express in LF the derivability of rules of inference by constructing a term of higher judgement type. We illustrate this potential by showing that the

[3] SOME-I is the rule of existential introduction, $\frac{\varphi[t]}{\exists x.\varphi}$.

$$(\text{ALL-I}^*) \quad \frac{ex\ (\mathcal{A}, x{:}\sigma)}{\forall_\sigma e\ (\mathcal{A})} \qquad\qquad (\text{ALL-E}^\dagger) \quad \frac{\forall_\sigma e\ (\mathcal{A})}{ee'\ (\mathcal{A})}$$

$$(\text{LAM}) \quad \frac{e =_\tau e'\ (\mathcal{A}, x{:}\sigma)}{\lambda_\sigma x.e =_{\sigma\to\tau} \lambda_\sigma x.e'\ (\mathcal{A})} \qquad\qquad (\text{EQ}) \quad \frac{\varphi\ (\mathcal{A}) \qquad \varphi =_o \psi\ (\mathcal{A})}{\psi\ (\mathcal{A})}$$

$$(\beta^\ddagger) \quad (\lambda_\sigma x.e[x])\, e' =_\tau e[e']\ (\mathcal{A}) \qquad\qquad (\eta^{**}) \quad \lambda_\sigma x.(e\, x) =_{\sigma\to\tau} e\ (\mathcal{A})$$

**Provided that* $\mathcal{A} \vdash e : \sigma{\to}o$ *and* x *does not occur free in any assumption on which* ex *depends.*
†*Provided that* $\mathcal{A} \vdash e : \sigma{\to}o$ *and* $\mathcal{A} \vdash e' : \sigma$.
‡*Provided that* $\mathcal{A}, x{:}\sigma \vdash e : \tau$ *and* $\mathcal{A} \vdash e' : \sigma$.
***Provided that* $\mathcal{A} \vdash e : \sigma \to \tau$ *and* x *does not occur in* $\mathcal{A}$.

Table 6: Some Axioms and Rules of Higher-Order Logic

elimination rule for the universal quantifier given by Schröder-Heister in [50] has a formal counterpart as a term of LF in the signature Σ_{FOL}. The Schröder-Heister elimination rule is specified as follows:

$$\text{ALL-E}_{SH} : \Pi F{:}\iota{\to}o.\Pi a{:}o.\mathit{true}(\forall(\lambda x{:}\iota.Fx)) \to ((\Pi x{:}\iota.\mathit{true}(Fx)) \to \mathit{true}(a)) \to \mathit{true}(a)$$

It can be easily (even mechanically) verified that the term

$$\lambda F{:}\iota{\to}o.\lambda a{:}o.\lambda p{:}\mathit{true}(\forall(\lambda x{:}\iota.Fx)).\lambda q{:}(\Pi x{:}\iota.\mathit{true}(Fx)){\to}\mathit{true}(a).q(\lambda x{:}\iota.\ \text{ALL-E}(\lambda x{:}\iota.Fx)(x)(p))$$

has the above type. This term may be viewed as a witness to the derivability of Schröder-Heister's rule in first-order logic.

With regard to derived rules, it is important to stress that in view of the fact that weakening is an admissible rule of the LF type theory, judgements are "open" concepts. This precludes the encoding of a proof of admissibility of an inference rule that makes use of a principle of induction over a type of proofs. For example, the proof of the deduction theorem for a Hilbert-style formalization of first-order logic cannot be encoded as an LF term in the usual encoding of Hilbert systems in LF. A closely-related point is the representation of rules of proof in a Hilbert system, which are rules that may be applied only if the premises are pure theorems (the rule of necessitation in the Hilbert-style formulation of S4 modal logic is a typical example). In many cases it is possible to exploit multiple judgements to achieve a faithful representation of such a system. (See [4] for further details.)

4.2 Higher-Order Logic

A natural deduction-style presentation of higher-order logic can be given along much the same lines as in the case of first-order logic. A difference is that one proves a formula φ relative to an assignment $\mathcal{A}$ governing the free variables of the proof; the assignments are made explicit by writing them in parentheses after the formula φ. The rules for equality include those for $\beta\eta$ conversion; there is also a rule EQ governing the interaction between truth and equality. As in first-order logic, only an illustrative selection of rules is presented here; see Table 6. The remaining rules — including those for arithmetic and choice — present no additional difficulties.

We turn now to the representation of higher-order logic in LF. We depart from Church [6] in that we use a natural deduction presentation, rather than a Hilbert-type system. We declare a constant in Σ_{HOL} representing the basic judgement form asserting that a formula is a logical truth. Since the formulas of higher-order logic are just the terms of o, we have the following declaration:

$$\mathit{true} : \mathit{obj}(o) \to \mathsf{Type}$$

The adequacy theorem for the syntax of higher-order logic ensures that the type $\mathit{obj}(o)$ faithfully represents the formulas, and so we are justified in introducing such a constant.

The inference rules are presented in LF using techniques similar to those in the case of first-order logic. We present declarations corresponding to the selection of rules given in Table 6. The declarations corresponding to the rules governing the universal quantifier are as follows, (we write arguments to applications as subscripts to enhance readability):

$$\text{ALL-I} : \Pi s{:}\mathit{holtype}.\Pi F{:}\, \mathit{obj}(s{\Rightarrow}o).(\Pi x{:}\, \mathit{obj}(s).\mathit{true}(\mathit{ap}_{s,o}\ F\, x)) \to \mathit{true}(\mathit{ap}_{s\Rightarrow o,o}\ \forall_s F)$$

$$\text{ALL-E} : \Pi s{:}\mathit{holtype}.\Pi F{:}\, \mathit{obj}(s{\Rightarrow}o).\Pi x{:}\, \mathit{obj}(s).\mathit{true}(\mathit{ap}_{(s\Rightarrow o),o}\ \forall_s F) \to \mathit{true}(\mathit{ap}_{s,o}\ F\, x)$$

The remaining rules involve the equality relation of higher-order logic. As a notational expedient, we make use of the following "externalization" of the equality constant:

$$\approx\ \equiv \lambda s{:}\mathit{holtype}.\lambda x{:}\, \mathit{obj}(s), \lambda y{:}\, \mathit{obj}(s).\ \mathit{ap}_{s,o}(\mathit{ap}_{s,s\Rightarrow o} =_s x)\, y$$

which has type

$$\Pi s{:}\mathit{holtype}.\ \mathit{obj}(s) \to \mathit{obj}(s) \to \mathit{obj}(o)$$

and which we write in infix form. The rule EQ is presented as follows:

$$\text{EQ} \quad : \quad \Pi\varphi{:}\, \mathit{obj}(o).\Pi\psi{:}\, \mathit{obj}(o).\mathit{true}(\varphi) \to \mathit{true}(\varphi \approx_o \psi) \to \mathit{true}(\psi)$$

The rule LAM of equality for λ-abstractions is presented by the declaration:

$$\begin{aligned}\text{LAM} \quad : \quad & \Pi s, t{:}\mathit{holtype}.\Pi f, g{:}\, \mathit{obj}(s) \to \mathit{obj}(t).(\Pi x{:}\, \mathit{obj}(s).\mathit{true}(f\, x \approx_t g\, x)) \to \\ & \mathit{true}(\Lambda_{s,t}\, \lambda x{:}\, \mathit{obj}(s).fx \approx_{s\Rightarrow t} \Lambda_{s,t}\, \lambda x{:}\, \mathit{obj}(s).gx)\end{aligned}$$

(using an obvious notational convention whereby several variables of the same type are introduced with a single Π). The β and η axioms for equality are presented as follows:

$$\beta \quad : \quad \Pi s, t{:}\mathit{holtype}.\Pi f{:}\, \mathit{obj}(s) \to \mathit{obj}(t).\Pi x{:}\, \mathit{obj}(s).\mathit{true}(\mathit{ap}_{s,t}\, (\Lambda_{s,t}\, (\lambda x{:}\, \mathit{obj}(s).fx))\, x \approx_t f\, x)$$

$$\eta \quad : \quad \Pi s, t{:}\mathit{holtype}.\Pi f{:}\, \mathit{obj}(s{\Rightarrow}t).\mathit{true}(\Lambda_{s,t}\, (\lambda x{:}\, \mathit{obj}(s).\ \mathit{ap}_{s,t}\ f\, x) \approx_{(s\Rightarrow t)} f)$$

By proceeding analogously to the first-order case, one can give a language of proof expressions for higher-order logic, and a formal system for deriving valid proofs with respect to a proof context $(\mathcal{A}, \Delta)$ consisting of an assignment $\mathcal{A}$ and a labeled hypothesis list Δ. The LF context $\Gamma_{\mathcal{A},\Delta}$ is defined in the obvious way, following the pattern of first-order logic. The encoding of valid proofs as canonical LF terms is defined in a manner similar to that of first-order logic. The adequacy of this encoding is expressed by the following theorem:

Theorem 4.2 (Adequacy for Higher-Order Logic) *Let $\mathcal{A}$ be an assignment and let Δ be a labeled set of hypotheses with free variables declared in $\mathcal{A}$. There is a compositional bijection $\varepsilon_{\mathcal{A},\Delta}$ mapping valid proofs of a formula φ with respect to $(\mathcal{A}, \Delta)$ to canonical LF terms of type $\varepsilon_{\mathcal{A},o}(\varphi)$ in Σ_{HOL} and $\Gamma_{\mathcal{A},\Delta}$.* □

5 Related Work

The design of LF has been strongly influenced by AUTOMATH [15] and by Martin-Löf's work on the foundations of intuitionistic logic [30]. The seminal work on machine-assisted proof was initiated by de Bruijn in the late 1960's, and was subsequently developed by him and his co-workers. The overall goal was to develop a framework for expressing arbitrary mathematical arguments in a notation suitable for checking by a machine. Their approach was based on representing mathematical texts as terms in a typed λ-calculus, reducing proof checking to type checking. A variety of mathematical theories have been developed and checked, most notably the formalization of Landau's textbook on Mathematical Analysis [26]. Their work has been of considerable importance in the development of machine-assisted proof, especially the NuPRL system of Constable and his co-workers [9] and the Calculus of Constructions of Coquand and Huet [10, 12, 13]. However, this subsequent work differs in spirit from AUTOMATH in that the latter two are concerned only with the formalization of constructive mathematics, whereas AUTOMATH sought to encompass classical mathematics as well. Our work can be viewed as a development of the AUTOMATH ideas that seeks to keep a clear distinction between the object- and meta-level, and seeks to handle proof checking for a wider class of logical systems.

As remarked earlier, the work of Martin-Löf has been of considerable importance in the design of LF. We were particularly influenced by his emphasis on the notion of judgement and on its uniform extension to higher-order forms. To a large extent our work has proceeded in parallel. Since the LF work began, he has developed his system of "logical types" (as yet unpublished, but see [36]) as the basis for his intuitionistic set theory. Formally, the system of logical types is quite similar to the LF type theory, but the applications are substantially different. In particular, we are concerned with encoding formal proofs in arbitrary logical systems, and are not concerned with specifically intuitionistic problems such as proof normalization. In contrast, Martin-Löf uses the system of logical types as the foundation for his set theory, and does not consider its application to general formal systems.

Since this research was initiated, there has been further work conducted both by members of the LF project and elsewhere. We begin by surveying the work of the LF project. A number of example logical systems have been encoded in LF. These include two different variations on Hoare logic [4], [3], modal logics from K to S4 [4], various λ-calculi, including λ-I, λ_v, and linear λ-calculus [4], [3], various type theories, including the LF type system itself, Martin-Löf's type theory, and the Damas-Milner type assignment system [22]. A natural deduction approach to operational semantics based on the ideas of LF has also been developed [5]. A general theory of search, including a unification algorithm for the LF type theory, has been developed [44, 45]. Elliott has also developed a unification algorithm for LF [16] which has been used by Pfenning as the basis for a logic programming language based on the LF type theory [41]. An equational variant of LF is introduced, along with the basic model-theoretic results, in [21]. Three different implementations have been undertaken, one by Griffin [20] (based on the Cornell Program Synthesizer [46]), one by Hagino (in ML), and one by Pollack (also in ML) [4].

Concurrently with the later development of LF, Paulson extended his Isabelle system using ideas very similar to those of LF in the context of higher-order logic [38, 39]. Constable and Howe [8] demonstrated the use of NuPRL as a logical framework, emphasizing the use of the richer type structure of the NuPRL type theory in an encoding. More recently, Felty has studied the representation of logics in λ-Prolog, in particular, the LF type theory itself [18]. Mendler and Aczel [32] are developing the theory of MaThImP, as system for doing interactive mathematics on a machine that is similarly based on a general theory of logical systems, albeit of a rather different flavor than that considered here. Feferman has proposed a theory of formal systems based on a general system of finitary inductive definitions [17].

References

[1] Andrews, P. B. Resolution in type theory. *Journal of Symbolic Logic 36* (1971), 414–432.

[2] Avron, A. Simple consequence relations. *Information and Computation 91*, 1 (1991), 105–139.

[3] Avron, A., Honsell, F., and Mason, I. A. An overview of the Edinburgh logical framework. In *Current Trends in Hardware Verification and Automated Theorem Proving*, G. Birtwistle and P. A. Subramanyam, Eds. Springer-Verlag, 1989, pp. 323–240.

[4] Avron, A., Honsell, F., Mason, I. A., and Pollack, R. Using typed lambda calculus to implement formal systems on a machine. Tech. Rep. ECS–LFCS–87–31, Laboratory for the Foundations of Computer Science, Edinburgh University, June 1987. To appear, *Journal of Automated Reasoning*.

[5] Burstall, R., and Honsell, F. Operational semantics in a natural deduction setting. In Huet and Plotkin [24].

[6] Church, A. A formulation of the simple theory of types. *Journal of Symbolic Logic 5* (1940), 56–68.

[7] Clocksin, W., and Mellish, C. S. *Programming in PROLOG*. Springer-Verlag, 1981.

[8] Constable, R., and Howe, D. NuPRL as a general logic. In *Logic and Computation*, P. Odifreddi, Ed. Academic Press, 1990.

[9] Constable, *et. al.*, R. L. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice-Hall, 1986.

[10] Coquand, T. *Une Théorie des Constructions*. PhD thesis, Université Paris VII, Jan. 1985.

[11] Coquand, T. An algorithm for testing conversion in type theory. In Huet and Plotkin [24].

[12] Coquand, T., and Huet, G. Constructions: A higher-order proof system for mechanizing mathematics. In *EUROCAL '85: European Conference on Computer Algebra* (1985), B. Buchberger, Ed., vol. 203 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 151–184.

[13] Coquand, T., and Huet, G. The Calculus of Constructions. *Information and Computation 76*, 2/3 (February/March 1988), 95–120.

[14] Curry, H. B., and Feys, R. *Combinatory Logic*. North-Holland, 1958.

[15] de Bruijn, N. G. A survey of the project AUTOMATH. In Seldin and Hindley [52], pp. 589–606.

[16] Elliott, C. *Extensions and Applications of Higher-Order Unification*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 1990.

[17] Feferman, S. Finitary inductively presented logics. In *Logic Colloquium, '88* (Amsterdam, 1989), North Holland, pp. 191–220.

[18] Felty, A. *Specifying and Implementing Theorem Provers in a Higher-Order Logic Programming Language*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, July 1989.

[19] Gordon, M., Milner, R., and Wadsworth, C. *Edinburgh LCF: A Mechanized Logic of Computation*, vol. 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[20] Griffin, T. An environment for formal systems. Tech. Rep. 87–846, Cornell University, Ithaca, New York, June 1987.

[21] Harper, R. An equational formulation of LF. Tech. Rep. ECS–LFCS–88–67, Laboratory for the Foundations of Computer Science, Edinburgh University, Oct. 1988.

[22] Harper, R. Systems of polymorphic type assignment in LF. Tech. Rep. CMU–CS–90–144, School of Computer Science, Carnegie Mellon University, June 1990.

[23] Howard, W. A. The formulas-as-types notion of construction. In Seldin and Hindley [52], pp. 479–490.

[24] Huet, G., and Plotkin, G., Eds. *Logical Frameworks*. Cambridge University Press, 1991.

[25] Huet, G. P. Unification for typed lambda calculus. *Theoretical Computer Science 1*, 1 (June 1975), 27–58.

[26] Jutting, L. S. *Checking Landau's Grundlagen in the AUTOMATH System*. PhD thesis, Eindhoven University, Netherlands, 1977.

[27] Martin-Löf, P. About models for intuitionistic type theories and the notion of definitional equality. In *Proceedings of the Third Scandinavian Logic Symposium* (1975), S. Kanger, Ed., Studies in Logic and the Foundations of Mathematics, North-Holland, pp. 81–109.

[28] Martin-Löf, P. An intuitionistic theory of types: Predicative part. In *Logic Colloquium '73* (1975), H. E. Rose and J. C. Shepherdson, Eds., vol. 80 of *Studies in Logic and the Foundations of Mathematics*, North-Holland, pp. 73–118.

[29] Martin-Löf, P. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science* (1982), North-Holland, pp. 153–175.

[30] Martin-Löf, P. On the meanings of the logical constants and the justifications of the logical laws. Tech. Rep. 2, Scuola di Specializzazione in Logica Matematica, Dipartimento di Matematica, Università di Siena, 1985.

[31] Martin-Löf, P. Truth of a proposition, evidence of a judgement, validity of a proof. Lecture given at the workshop "Theories of Meaning", Florence, Italy., June 1985.

[32] Mendler, P. F., and Aczel, P. The notion of a framework and a framework for LTC. In *Third Symposium on Logic in Computer Science* (Edinburgh, July 1988), pp. 392–401.

[33] Meyer, A., and Reinhold, M. 'Type' is not a type: Preliminary report. In *Thirteenth ACM Symposium on Principles of Programming Languages* (1986).

[34] Milner, R., Tofte, M., and Harper, R. *The Definition of Standard ML.* MIT Press, 1990.

[35] Mitchell, J. C. Type systems for programming languages. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. B: Formal Models and Semantics. Elsevier, Amsterdam, 1991, ch. 8, pp. 365–458.

[36] Norström, B., Petersson, K., and Smith, J. M. *Programming in Martin-Löf's Type Theory: An Introduction*, vol. 7 of *International Series of Monographs on Computer Science.* Oxford University Press, 1990.

[37] Paulson, L. Interactive theorem proving with Cambridge LCF. Tech. Rep. 80, Computer Laboratory, University of Cambridge, Nov. 1985.

[38] Paulson, L. Natural deduction proof as higher-order resolution. *Journal of Logic Programming 3* (1986), 237–258.

[39] Paulson, L. The foundations of a generic theorem prover. Tech. Rep. 130, Computer Laboratory, University of Cambridge, 1987.

[40] Petersson, K. A programming system for type theory. Tech. Rep. 21, Programming Methodology Group, University of Göteborg/Chalmers Institute of Technology, Mar. 1982.

[41] Pfenning, F. Logic programming in the LF logical framework. In Huet and Plotkin [24].

[42] Plotkin, G. Call-by-name, call-by-value, and the lambda calculus. *Theoretical Computer Science 1* (1975), 125–159.

[43] Prawitz, D. *Natural Deduction: A Proof-Theoretical Study.* Almquist & Wiksell, 1965.

[44] Pym, D. *Proofs, Search and Computation in General Logic.* PhD thesis, Edinburgh University, 1990. Available as Edinburgh University Computer Science Department Technical Report ECS–LFCS–90–125.

[45] Pym, D., and Wallen, L. Proof search in the $\lambda\Pi$-calculus. In Huet and Plotkin [24].

[46] Reps, T. W., and Teitelbaum, T. The synthesizer generator reference manual. Tech. rep., Cornell University, Ithaca, New York, 1987.

[47] Robinson, J. A. A machine-oriented logic based on the resolution principle. *Journal of the ACM 12* (1965), 23–41.

[48] Salvesen, A. A proof of the Church-Rosser property for the Edinburgh LF with η-conversion. Lecture given at the First Workshop on Logical Frameworks, Sophia-Antipolis, France., May 1990.

[49] Schoenfield, J. R. *Mathematical Logic.* Addison-Wesley, 1967.

[50] Schröder–Heister, P. A natural extension of natural deduction. *Journal of Symbolic Logic 49*, 4 (Dec. 1984).

[51] Schütte, K. *Proof Theory.* Springer-Verlag, 1977.

[52] Seldin, J. P., and Hindley, J. R., Eds. *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism* (1980), Academic Press.

[53] Stenlund, S. *Combinators, λ-terms and Proof Theory.* D. Reidel, 1972.

[54] Takeuti, G. *Proof Theory.* North-Holland, 1975.

[55] van Daalen, D. T. *The Language Theory of AUTOMATH.* PhD thesis, Technical University of Eindhoven, Eindhoven, Netherlands, 1980.

A Metatheoretic Properties of the LF Type System

In this appendix we outline the proofs of some of the results given in Section 2 . Our ultimate goal is to give a proof of the decidability of the LF type system. This is a lengthy and difficult task (which can be surprisingly complicated for stronger notions of definitional equality). In order to simplify the proofs, we work with a variant of the type theory that lacks signatures and the context validity judgement. This system is related to the LF type system in a very simple way, and hence we may transfer the metatheoretic results directly to the latter system. The proof of decidability requires a number of properties to be established for the system. Some are interesting in themselves, others are merely technical lemmas to simplify the proofs. Then, as a tool for establishing decidability, we introduce an algorthmic presentation of the system, closer in spirit to implementations of LF. The algorithmic version is also used in the proof of the derivability of strengthening. Finally, we outline a possible decidability proof for the extension of LF which admits η as a rule of definitional equality.

A.1 A Simplified Type Theory

To simplify our work we introduce a variant of the LF type theory that is somewhat easier to handle. We eliminate constants and signatures in favor of variables and a more general form of context admitting kind, as well as type, declarations. Further, we eliminate the context validity assertion in favor of the derivability of $\Gamma \vdash \mathsf{Type}$, as in AUTOMATH. In compensation for the generalization of contexts, we must introduce premises on the λ and Π rules to ensure that the domain is a type and not a kind. The definitional equality relation is the same as that given in Section 2. The rules for this system appear in Tables 7 and 8.

By regarding the constants of the LF type system defined in Section 2 as variables of the simplified system, we may obtain the following equivalence theorem:

Theorem A.1 *Let Γ be a context containing only declarations of the form $x{:}A$, and let Σ be a signature.*

1. Σ sig *iff* $\Sigma \vdash \mathsf{Type}$.
2. $\vdash_\Sigma \Gamma$ *iff* $\Sigma, \Gamma \vdash \mathsf{Type}$.
3. $\Gamma \vdash_\Sigma \alpha$ *iff* $\Sigma, \Gamma \vdash \alpha$.

Proof By induction on the definitions of the two type systems. □

The theorems stated in Section 2 for the LF type theory are obtained as immediate corollaries of the corresponding results for the simplified type system and the above theorem relating the two.

Valid Kinds

$$\frac{}{\vdash \mathsf{Type}} \quad \text{(S-TYPE-KIND)}$$

$$\frac{\Gamma \vdash K \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x{:}K \vdash \mathsf{Type}} \quad \text{(S-K-VAR)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathsf{Type}} \quad \text{(S-T-VAR)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x{:}A \vdash K}{\Gamma \vdash \Pi x{:}A.K} \quad \text{(S-PI-KIND)}$$

Valid Families

$$\frac{\Gamma \vdash \mathsf{Type} \quad x{:}K \in \Gamma}{\Gamma \vdash x : K} \quad \text{(S-VAR-FAM)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x{:}A \vdash B : \mathsf{Type}}{\Gamma \vdash \Pi x{:}A.B : \mathsf{Type}} \quad \text{(S-PI-FAM)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x{:}A \vdash B : K}{\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K} \quad \text{(S-ABS-FAM)}$$

$$\frac{\Gamma \vdash A : \Pi x{:}B.K \quad \Gamma \vdash M : B}{\Gamma \vdash AM : [M/x]K} \quad \text{(S-APP-FAM)}$$

$$\frac{\Gamma \vdash A : K \quad \Gamma \vdash K' \quad \Gamma \vdash K \equiv K'}{\Gamma \vdash A : K'} \quad \text{(S-CONV-FAM)}$$

Table 7: Simplified Variant of LF

Valid Objects

$$\frac{\Gamma \vdash \mathsf{Type} \quad x{:}A \in \Gamma}{\Gamma \vdash x : A} \quad \text{(S-VAR-OBJ)}$$

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B} \quad \text{(S-ABS-OBJ)}$$

$$\frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : [N/x]B} \quad \text{(S-APP-OBJ)}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A' : \mathsf{Type} \quad \Gamma \vdash A \equiv A'}{\Gamma \vdash M : A'} \quad \text{(S-CONV-OBJ)}$$

Table 8: Simplified Variant of LF (continued)

A.2 Fundamental Properties of the Simplified System

Theorem A.2

1. *Transitivity is a derived rule: if* $\Gamma \vdash M : A$ *and* $\Gamma, x{:}A, \Gamma' \vdash \alpha$*, then* $\Gamma, [M/x]\Gamma' \vdash [M/x]\alpha$*.*

2. *Weakening and permutation are derived rules: if* Γ *and* Γ' *are valid contexts, and every declaration occurring in* Γ *also occurs in* Γ'*, then* $\Gamma \vdash \alpha$ *implies* $\Gamma' \vdash \alpha$*.*

Proof By induction on the structure of proofs of the hypotheses. The proof of the strengthening property is given below (Theorem A.18). □

The following lemma simplifies many of the proofs to come:

Lemma A.3 (Subderivation Property)

1. *Every proof of* $\Gamma \vdash \alpha$ *has a proof of* $\Gamma \vdash \mathsf{Type}$ *as a sub-proof.*

2. *Every proof of* $\Gamma, x{:}A \vdash \mathsf{Type}$ *has a proof of* $\Gamma \vdash A : \mathsf{Type}$ *as a sub-proof.*

3. *Every proof of* $\Gamma, x{:}K \vdash \mathsf{Type}$ *has a proof of* $\Gamma \vdash K$ *as a sub-proof.*

4. *If* $\Gamma \vdash A : K$*, then* $\Gamma \vdash K$*.*

5. *If* $\Gamma \vdash M : A$*, then* $\Gamma \vdash A : \mathsf{Type}$*.*

Proof By induction on the structure of the proof of the premise. □

Again by induction on the structure of proofs we can show:

Theorem A.4 (Unicity of Types and Kinds)

1. *If* $\Gamma \vdash M : A$ *and* $\Gamma \vdash M : A'$*, then* $\Gamma \vdash A \equiv A'$*.*

2. *If* $\Gamma \vdash A : K$ *and* $\Gamma \vdash A : K'$*, then* $\Gamma \vdash K \equiv K'$*.* □

This is particularly easy to establish because of the simple nature of our definitional equality relation.

Unicity of types and kinds, together with the Church-Rosser property, allow a straightforward derivation of the following lemma, which is needed for the proof of the subject reduction theorem.

Lemma A.5 (Abstraction Typing)

1. *If* $\Gamma \vdash \lambda x{:}A.B : \Pi x{:}C.K$*, then* $\Gamma \vdash A \equiv C$*.*

2. *if* $\Gamma \vdash \lambda x{:}A.M : \Pi x{:}B.C$*, then* $\Gamma \vdash A \equiv B$*.*

3. *If* $\Gamma \vdash \lambda x{:}A.B : \Pi x{:}A.K$*, then* $\Gamma, x{:}A \vdash B : K$*.*

4. *If* $\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B$*, then* $\Gamma, x{:}A \vdash M : B$*.* □

The proof of this lemma for stronger systems of definitional equality is extremely delicate. It is the first non-trivial property one has to prove. And when Church-Rosser is not available, the proof of this lemma has to include part of the complexity of the full Church-Rosser proof.

The subject reduction theorem is stated for the relation $\rightarrow_1$ of *one-step reduction*. This relation is defined to be the restriction of the parallel reduction relation defined in Table 3 (which we now read for the simplified system) obtained by dropping the rule of reflexivity, eliminating the premises on the β rules, and duplicating the remaining rules so that reduction is performed in only one of the two possible subterms. It is easy to see that $\rightarrow_1^*$ and $\rightarrow^*$ coincide.

Theorem A.6 (Subject Reduction)

1. *If $\Gamma \vdash K$ and $K \rightarrow_1 K'$, then $\Gamma \vdash K'$.*
2. *If $\Gamma \vdash A : K$ and $A \rightarrow_1 A'$, then $\Gamma \vdash A' : K$.*
3. *If $\Gamma \vdash M : A$ and $M \rightarrow_1 M'$, then $\Gamma \vdash M' : A$.*

Proof By simultaneous induction on the structure of the derivation of the premises. We illustrate the case in which the last step of the typing derivation is rule S-APP-OBJ and the last reduction step is rule R-BETA-OBJ, *i.e.*,

$$\Gamma \vdash (\lambda x{:}A.M)N : B$$

and

$$(\lambda x{:}A.M)N \rightarrow_1 [N/x]M,$$

with

$$\Gamma \vdash \lambda x{:}A.M : \Pi x{:}C.D,$$

and

$$\Gamma \vdash N : C,$$

and $[N/x]D = B$. By the abstraction typing lemma and rule S-CONV-OBJ, we have $\Gamma \vdash N : A$ and $\Gamma, x{:}A \vdash M : D$, from which the result follows by an application of transitivity. □

Another essential ingredient, needed in order to achieve the decidability property is strong normalization for $\rightarrow_1$. The proof we will give here is interesting for two reasons. The first is that it does not depend on the Church-Rosser property and is therefore applicable to LF with stronger notions of definitional equality. The second is that it yields an interesting corollary as a by-product, the predicativity theorem.

Theorem A.7 (Strong Normalization)

1. *If $\Gamma \vdash K$, then K is strongly normalizing.*
2. *If $\Gamma \vdash A : K$, then A is strongly normalizing.*
3. *If $\Gamma \vdash M : A$, then M is strongly normalizing.*

Theorem A.8 (Predicativity) *If $\Gamma \vdash M : A$, then* Erase(M) *can be typed in Curry's type assignment system, where* Erase(M) *denotes the untyped λ-term obtained from M by removing type labels from λ-abstractions.*

The proof of strong normalization proceeds as follows. We start by defining "dependency-less" translations τ of kinds and type families to S, the set of simple types over a given base type ω, and $|\cdot|$, of type families and objects to $\Lambda(K)$, the set of untyped λ-terms over a set of constants $K = \{\, \pi_\sigma \mid \sigma \in S \,\}$. The τ translation is extended to contexts in the natural way.

Definition A.9

$$\begin{array}{lcl}
\tau(\mathsf{Type}) & = & \omega \\
\tau(\Pi x{:}A.K) & = & \tau(A) \rightarrow \tau(K) \\
\tau(x) & = & \omega \\
\tau(\lambda x{:}A.B) & = & \tau(B) \\
\tau(AM) & = & \tau(A) \\
\tau(\Pi x{:}A.B) & = & \tau(A) \rightarrow \tau(B) \\
|x| & = & x \\
|AM| & = & |A||M| \\
|MN| & = & |M||N| \\
|\Pi x{:}A.B| & = & \pi_{\tau(A)}|A|(\lambda x.|B|) \\
|\lambda x{:}A.M| & = & (\lambda y.\lambda x.|M|)|A| \quad (y \notin \mathrm{FV}(M)) \\
|\lambda x{:}A.B| & = & (\lambda y.\lambda x.|B|)|A| \quad (y \notin \mathrm{FV}(B))
\end{array}$$

Note that $\tau(A) = \tau([N/x]A)$ — the dependency of A on ordinary variables is eliminated by the τ translation.

The idea of the proof is to embed LF into Curry-typeable terms of the untyped λ-calculus in a structure-preserving way (*cf.* the last three clauses of the definition of $|\cdot|$). The translation is sufficiently faithful as to preserve the number of β-reductions, and so strong normalization for LF follows from strong normalization for simply-typed λ-calculus.

We need two lemmas.

Lemma A.10

1. *If* $\Gamma \vdash A \equiv A'$, *then* $\tau(A) = \tau(A')$.
2. *If* $\Gamma \vdash K \equiv K'$, *then* $\tau(K) = \tau(K')$.

Proof It is easy to show by induction on the structure of proofs that if $A \rightarrow A'$, then $\tau(A) = \tau(A')$, and similarly that if $K \rightarrow K'$, then $\tau(K) = \tau(K')$. The result then follows from the definition of definitional equality. □

Lemma A.11

1. $|[N/x]M| = [|N|/x]|M|$.
2. $|[N/x]B| = [|N|/x]|B|$.

Proof By induction on the structure of M and B. □

The following lemma shows that the translation was consistently given:

Lemma A.12

1. *If* $\Gamma \vdash A : K$, *then* $\tau(\Gamma) \vdash_\Sigma |A| : \tau(K)$;
2. *If* $\Gamma \vdash M : A$, *then* $\tau(\Gamma) \vdash_\Sigma |M| : \tau(A)$.

where $\vdash_\Sigma$ *denotes the type assignment system of Curry augmented by the infinite set of rules for K*

$$\vdash_\Sigma \pi_\sigma : \omega \rightarrow (\sigma \rightarrow \omega) \rightarrow \omega$$

for each $\sigma \in S$.

Proof By induction on the structure of the proof of $\Gamma \vdash A : K$ and $\Gamma \vdash M : A$. Since $\tau(A)$ and $\tau(K)$ are always well-formed simple types, the results hold trivially if the last rules applied in the LF derivation are rules S-VAR-FAM and S-VAR-OBJ. Now we deal with some other cases:

(S-PI-FAM) By induction hypothesis we have both

$$\tau(\Gamma), x{:}\tau(A) \vdash_\Sigma |B| : \omega$$

and

$$\tau(\Gamma) \vdash_\Sigma |A| : \omega.$$

Therefore

$$\tau(\Gamma) \vdash_\Sigma \lambda x.|B| : \tau(A) \rightarrow \omega$$

and

$$\tau(\Gamma) \vdash_\Sigma \pi_{\tau(A)}|A|(\lambda x.|B|) : \omega.$$

(S-ABS-FAM) By induction hypothesis we have both

$$\tau(\Gamma), x{:}\tau(A) \vdash_\Sigma |B| : \tau(K)$$

and

$$\tau(\Gamma) \vdash_\Sigma |A| : \omega$$

and therefore

$$\tau(\Gamma) \vdash_\Sigma (\lambda y.\lambda x.|B|)|A| : \tau(A) \to \tau(K).$$

(S-APP-FAM) By induction hypothesis we have

$$\tau(\Gamma) \vdash_\Sigma |A| : \tau(B) \to \tau(K)$$

and

$$\tau(\Gamma) \vdash_\Sigma |M| : \tau(B)$$

and since $\tau([M/x]K) = \tau(K)$, the result is achieved.

(S-CONV-FAM) By induction hypothesis we have

$$\tau(\Gamma) \vdash_\Sigma |A| : \tau(K)$$

and thus by Lemma A.10 we have $\tau(\Gamma) \vdash_\Sigma |A| : \tau(K')$.

The remaining cases are handled similarly. □

The extra combinatorial complexity of LF terms due to the possibility of reductions within type labels is not lost by the translation, as can be seen by the following lemma:

Lemma A.13

1. *If $A \to_1 A'$, then $|A| \to_1^+ |A'|$;*
2. *If $M \to_1 M'$, then $|M| \to_1^+ |M'|$.*

where $\to_1^+$ is the transitive closure of $\to_1$ for the untyped λ-calculus.

Proof By induction on the proof of $A \to_1 A'$ and $M \to_1 M'$. The only non-trivial cases arise when the last rule applied is one of the β-rules, or one of the Π-rules. In the first case we have, for example,

$$|(\lambda x{:}A.M)N| \to_1^+ (\lambda x.|M|)|N| \to_1^+ [|N|/x]|M|$$

which by Lemma A.11 is $|[N/x]M|$. In the second case Lemma A.10 suffices for the result. □

We can now prove strong normalization. Lemma A.13 implies that the translation of a reduct can be reached from the translation of the redex with at least one reduction. Now, since translations of well-typed LF terms are Curry-typable λ-terms, and since the Curry-typable terms are strongly normalizing, we can conclude that no infinite reduction sequence can start from a well-typed LF term. It follows that the relation of definitional equality is decidable for well-typed terms, since by Church-Rosser and strong normalization, $\Gamma \vdash U \equiv V$ iff U and V have identical normal forms.

The proof of the predicativity corollary is by induction on the structure of M: it can be easily seen that $|M| \to_1^+ \text{Erase}(M)$, and Lemma A.12 and subject reduction for Curry type assignment yield the result.

We are now ready to prove the major result of this section, the decidability of the relations $\Gamma \vdash \alpha$. We achieve this by introducing an implementation-oriented variant of LF that is equivalent to the original system and for which we may establish decidability by a simple induction on the complexity of assertions. The algorithmic system defines three forms of assertion

$$\Gamma \vdash K \Rightarrow \text{kind} \qquad \Gamma \vdash A \Rightarrow K \qquad \Gamma \vdash M \Rightarrow A$$

$$\frac{}{\vdash \mathsf{Type} \Rightarrow \mathrm{kind}} \qquad \text{(A-TYPE-KIND)}$$

$$\frac{\Gamma \vdash K \Rightarrow \mathrm{kind} \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x{:}K \vdash \mathsf{Type} \Rightarrow \mathrm{kind}} \qquad \text{(A-K-VAR)}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{Type} \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x{:}A \vdash \mathsf{Type} \Rightarrow \mathrm{kind}} \qquad \text{(A-T-VAR)}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{Type} \quad \Gamma, x{:}A \vdash K \Rightarrow \mathrm{kind}}{\Gamma \vdash \Pi x{:}A.K \Rightarrow \mathrm{kind}} \qquad \text{(A-PI-KIND)}$$

$$\frac{\Gamma \vdash \mathsf{Type} \Rightarrow \mathrm{kind} \quad x : K \in \Gamma}{\Gamma \vdash x \Rightarrow \mathrm{NF}(K)} \qquad \text{(A-VAR-FAM)}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{Type} \quad \Gamma, x{:}A \vdash B \Rightarrow \mathsf{Type}}{\Gamma \vdash \Pi x{:}A.B \Rightarrow \mathsf{Type}} \qquad \text{(A-PI-FAM)}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{Type} \quad \Gamma, x{:}A \vdash B \Rightarrow K}{\Gamma \vdash \lambda x{:}A.B \Rightarrow \Pi x{:}\mathrm{NF}(A).K} \qquad \text{(A-ABS-FAM)}$$

$$\frac{\Gamma \vdash A \Rightarrow \Pi x{:}B.K \quad \Gamma \vdash M \Rightarrow B}{\Gamma \vdash AM \Rightarrow \mathrm{NF}([M/x]K)} \qquad \text{(A-APP-FAM)}$$

$$\frac{\Gamma \vdash \mathsf{Type} \Rightarrow \mathrm{kind} \quad x{:}A \in \Gamma}{\Gamma \vdash x \Rightarrow \mathrm{NF}(A)} \qquad \text{(A-VAR-OBJ)}$$

$$\frac{\Gamma \vdash A \Rightarrow \mathsf{Type} \quad \Gamma, x{:}A \vdash M \Rightarrow B}{\Gamma \vdash \lambda x{:}A.M \Rightarrow \Pi x{:}\mathrm{NF}(A).B} \qquad \text{(A-ABS-OBJ)}$$

$$\frac{\Gamma \vdash M \Rightarrow \Pi x{:}A.B \quad \Gamma \vdash N \Rightarrow A}{\Gamma \vdash MN \Rightarrow \mathrm{NF}([N/x]B)} \qquad \text{(A-APP-OBJ)}$$

Table 9: Algorithmic Version of LF

with intended meaning that K is a kind, A has normal form kind K, and M has normal form type A in context Γ, respectively. The rules of derivation for these assertions appear in Table 9. These rules make use of a function $\mathrm{NF}(U)$ which yields the normal form of an expression U with respect to the leftmost-outermost reduction strategy. Several of the rules given in Table 9 make use of NF in the conclusion of the rule. We temporarily adopt the convention that such a rule does not apply unless the required normal form exists, for it will be a direct consequence of the soundness theorem given below that the normal forms in question will always exist. By an easy induction on the structure of terms U, we can easily see that there is at most one V such that $\Gamma \vdash U \Rightarrow V$, and V is in normal form.

The relationship between the algorithmic system and the system of Tables 7 and 8 is made precise by the following two theorems.

Theorem A.14 (Soundness)

1. *If* $\Gamma \vdash K \Rightarrow \mathrm{kind}$*, then* $\Gamma \vdash K$*.*
2. *If* $\Gamma \vdash A \Rightarrow K$*, then* $\Gamma \vdash A : K$*.*
3. *If* $\Gamma \vdash M \Rightarrow A$*, then* $\Gamma \vdash M : A$*.*

Proof By induction on the structure of the proof of the premises. We consider here the case when the last rule application is A-APP-OBJ; the other cases are dealt with similarly. We have by induction $\Gamma \vdash M : \Pi x{:}A.B$ and $\Gamma \vdash N : A$, and therefore by rule S-APP-OBJ, $\Gamma \vdash MN : [N/x]B$. By assumption the normal form $\mathrm{NF}([N/x]B)$ exists, and by Theorem A.6, $\Gamma \vdash \mathrm{NF}([N/x]B) : \mathsf{Type}$. Therefore, by an application of rule S-CONV-OBJ, $\Gamma \vdash MN : \mathrm{NF}([N/x]B)$, as required. □

Theorem A.15 (Completeness)

1. *If* $\Gamma \vdash K$*, then* $\Gamma \vdash K \Rightarrow \mathrm{kind}$*.*
2. *If* $\Gamma \vdash A : K$*, then* $\Gamma \vdash A \Rightarrow \mathrm{NF}(K)$*.*
3. *If* $\Gamma \vdash M : A$*, then* $\Gamma \vdash M \Rightarrow \mathrm{NF}(A)$*.*

Proof By induction on the structure of the proofs of the premises. We consider two cases; the others are handled similarly.

(S-CONV-OBJ) By induction hypothesis we have $\Gamma \vdash M \Rightarrow \mathrm{NF}(A)$. Since $\Gamma \vdash A' : \mathsf{Type}$, A' is strongly normalizing and so as $\Gamma \vdash A \equiv A'$, $\mathrm{NF}(A) = \mathrm{NF}(A')$ by the Church-Rosser theorem, which establishes the result.

(S-APP-OBJ) By induction hypothesis we have

$$\Gamma \vdash M \Rightarrow \mathrm{NF}(\Pi x{:}A.B)$$

and

$$\Gamma \vdash N \Rightarrow \mathrm{NF}(A).$$

Noting that $\mathrm{NF}(\Pi x{:}A.B)$ is $\Pi x{:}\mathrm{NF}(A).\,\mathrm{NF}(B)$ and that

$$\mathrm{NF}([N/x]\,\mathrm{NF}(B)) = \mathrm{NF}([N/x]B)$$

(by the Church-Rosser theorem), the result follows by an application of rule A-APP-OBJ. □

We are now ready to establish the crucial lemma:

Lemma A.16

1. *It can be recursively decided whether or not* $\Gamma \vdash K \Rightarrow \mathrm{kind}$*.*

2. *It can be recursively decided whether or not there exists K such that $\Gamma \vdash A \Rightarrow K$.*

3. *It can be recursively decided whether or not there exists A such that $\Gamma \vdash M \Rightarrow A$.*

Proof The result follows from two observations. First, the algorithmic system is deterministic in the sense that a proof of an assertion is completely determined by the form of the assertion itself. Second, if we measure the complexity of an assertion $\Gamma \vdash K \Rightarrow \text{kind}$ (respectively, $\Gamma \vdash A \Rightarrow K$ and $\Gamma \vdash M \Rightarrow A$) by a suitable measure of the size of Γ and K (respectively, Γ and A, and Γ and M), then the proof of an assertion is determined by proofs of strictly smaller measure. Note that it follows from the soundness theorem that all required normal forms exist. □

The decidability of the simplified system follows immediately from the soundness, completeness, and decidability of the algorithmic system:

Theorem A.17 (Decidability) *The assertions $\Gamma \vdash \alpha$ are recursively decidable.*

The algorithmic system introduced to establish the decidability theorem is also extremely valuable for establishing the derivability of the structural rule of strengthening.

Theorem A.18 (Strengthening) *Strengthening is a derived rule: if $\Gamma, x{:}U, \Gamma' \vdash \alpha$, then $\Gamma, \Gamma' \vdash \alpha$ provided that $x \notin \mathrm{FV}(\Gamma') \cup \mathrm{FV}(\alpha)$.*

Proof It is relatively straightforward to prove that the analogue of the strengthening property holds of the algorithmic system. To do so we prove by induction on derivations that if $\Gamma, x{:}U, \Delta \vdash P \Rightarrow R$ holds and x does not occur free in Δ or P, then x is not free in R. The strengthening property of the LF type theory now follows easily from the strengthening property of the algorithmic system, keeping in mind the subderivation property, and the soundness and completeness of the algorithmic system. For example, suppose that $\Gamma, x{:}U, \Gamma' \vdash M : A$ with $x \notin \mathrm{FV}(\Gamma') \cup \mathrm{FV}(M) \cup \mathrm{FV}(A)$. By completeness, $\Gamma, x{:}U, \Gamma' \vdash M \Rightarrow \mathrm{NF}(A)$ and hence $\Gamma, \Gamma' \vdash M \Rightarrow \mathrm{NF}(A)$, and so by soundness $\Gamma, \Gamma' \vdash M : \mathrm{NF}(A)$. On the other hand, by the subderivation lemma, $\Gamma, x{:}U, \Gamma' \vdash A : \mathsf{Type}$, and by reasoning as above, $\Gamma, \Gamma' \vdash A : \mathsf{Type}$. Then, by an application of rule S-CONV-OBJ, $\Gamma, \Gamma' \vdash M : A$, as required. □

A.3 Systems with Stronger Definitional Equality

The relation of definitional equality considered so far in the Appendix is the weakest form that is adequate for our purposes. It is possible to consider strengthening this relation to include rules for δ-, R-, or η-reduction. It is for these systems that the environments Γ have to be explicitly mentioned in the definitional equality assertions in order for the resulting systems to be well-behaved. See [21] for an analysis of LF systems of this kind.

In this subsection we will outline the theory of the LF system which arises from the extension of definitional equality considered above obtained by including η-reduction. To this end the rules for definitional equality must be stated for well-typed terms, and typing premises have to be introduced in some of the rules. Here is a sample list of rules:

$$\frac{\Gamma, x{:}A \vdash M \equiv M'}{\Gamma \vdash \lambda x{:}A.M \equiv \lambda x{:}A.M'}$$

$$\frac{\Gamma \vdash (\lambda x{:}A.M)N : B \quad \Gamma \vdash [N/x]M : C}{\Gamma \vdash (\lambda x{:}A.M)N \equiv [N/x]M}$$

$$\frac{\Gamma \vdash \lambda x{:}A.Mx : B \quad \Gamma \vdash M : C \quad x \notin \mathrm{FV}(M)}{\Gamma \vdash \lambda x{:}A.Mx \equiv M}$$

A number of variations are possible; the version we consider here is inspired by the work of van Daalen [55].

Such a system (henceforth called LFη) clearly has the advantage of allowing a smoother treatment of adequacy since every well-typed term will be convertible to a unique canonical form. Note that in LFη no term of type o in Σ_{FOL} is ruled out as a correct encoding of a first-order formula: *e.g.*, $\forall(\lambda x{:}\iota.\forall(=x))$ can be safely treated as a representation of $\forall x.\forall y.x = y$. Another advantage of considering η is that it simplifies considerably the unification problem for LF [44]. We preferred, however, to study in detail a more elementary system since LFη has an extremely complicated and delicate metatheory. Achieving decidability of LFη is a surprisingly difficult task which, however, does not provide any deeper insight into the uses or virtues of LF, which is the principal concern of this paper.

Some of the difficulties arising in establishing the decidability of LFη were pointed out in Section 2. We will now sketch what we conjecture would be a proof of decidability for LFη by adapting some of the techniques van Daalen has introduced in the study of AUT-QE. This account will be lacking in two respects: no details are given and no specification for the shape of the equality rules will be given. Clearly various systems can be chosen according to which notion of $\beta\eta$-reduction (*e.g.*, parallel or one-step) is modelled or which typing constraints are enforced.

The major difficulty in handling LFη arises from the fact that the Church-Rosser property cannot be established for arbitrary terms, but can only be proved for well-typed terms at a stage where a great number of syntactic properties of the system are already available. It is for this reason that the order in which the results are proved is essential. Fortunately, the proof of strong normalization we have for the weaker system readily adapts to LFη. Thus strong normalization for LFη can be established very early, as soon as the first structural properties are proved.

Since the Church-Rosser property is not available at the outset, the proofs of "unicity of domains" (Theorem A.5) and the "subject reduction" theorem cannot be established immediately. It should be possible, nonetheless, to establish the first property with a more elaborate argument than the one used in the previous case. On the other hand the subject reduction theorem cannot be shown at this stage for this system. The only way to prove subject reduction, then, seems to be to assume strengthening as a rule of the system from the very beginning and to prove for this new system all the properties we have so far established. At this stage Church-Rosser could be proved for well-typed terms in LFη with strengthening, using a modification of van Daalen's technique of label conversion.

Now the decidability of LFη with strengthening should be provable in a very similar way to the one we have given for the weaker system. In particular, an appropriate equivalent algorithmic system should be introduced. Using this system, strengthening should be proved redundant in a manner similar to that used to prove the admissibility of strengthening for the weaker system. Many of the variants of LFη obtained by altering the choice of conversion rules could now be shown to be equivalent.

Note added in proof: Since this paper was written, Salvesen, inspired by the above outline, has carried out a complete proof of the Church-Rosser property for LFη [48]. Taking a different approach, Coquand has also given a proof of the Church-Rosser property for a very similar type system [11].