1-1-2007

Mechanizing Metatheory in a Logical Framework

Robert Harper
Carnegie Mellon University, rwh@cs.cmu.edu

Daniel R. Licata
Carnegie Mellon University

Mechanizing Metatheory in a Logical Framework

Robert Harper and Daniel R. Licata
Carnegie Mellon University
(*e-mail:* {rwh,drl}@cs.cmu.edu)

Abstract

The LF logical framework codifies a methodology for representing deductive systems, such as programming languages and logics, within a dependently typed λ-calculus. In this methodology, the syntactic and deductive apparatus of a system is encoded as the canonical forms of associated LF types; an encoding is correct (*adequate*) if and only if it defines a *compositional bijection* between the apparatus of the deductive system and the associated canonical forms. Given an adequate encoding, one may establish metatheoretic properties of a deductive system by reasoning about the associated LF representation. The Twelf implementation of the LF logical framework is a convenient and powerful tool for putting this methodology into practice. Twelf supports both the representation of a deductive system and the mechanical verification of proofs of metatheorems about it.

The purpose of this paper is to provide an up-to-date overview of the LF λ-calculus, the LF methodology for adequate representation, and the Twelf methodology for mechanizing metatheory. We begin by defining a variant of the original LF language, called *Canonical LF*, in which only canonical forms (long $\beta\eta$-normal forms) are permitted. This variant is parameterized by a *subordination relation*, which enables modular reasoning about LF representations. We then give an adequate representation of a simply typed λ-calculus in Canonical LF, both to illustrate adequacy and to serve as an object of analysis. Using this representation, we formalize and verify the proofs of some metatheoretic results, including preservation, determinacy, and strengthening. Each example illustrates a significant aspect of using LF and Twelf for formalized metatheory.

1 Introduction

A formal definition of a programming language provides a precise specification for programmers, ensures compatibility among compilers, and enables rigorous analysis of its properties. However, a language definition is an intricate artifact, and the proofs of its properties are often complex and subtle. Consequently, it can be difficult to define a language correctly and prove the appropriate theorems about it, let alone to maintain the definition and associated proofs as a language evolves. Fortunately, some of this burden can be alleviated by mechanizing the definition and metatheory of a language.

In this article, we present a technical introduction to mechanizing languages using LF (Harper *et al.*, 1993) and Twelf (Pfenning & Schürmann, 1999). The literature on formalizing and verifying languages in LF and Twelf is dispersed among numerous research papers, dissertations, course notes, and on-line repositories. The

goal of this paper is to consolidate these ideas and make them more accessible to a broader audience. We present an up-to-date overview of the LF λ-calculus, the LF methodology for representing languages, and the Twelf methodology for mechanically verifying the metatheory of a language. We hope to suggest that this methodology for mechanizing languages is effective and feasible; however, the successful Twelf mechanization efforts, some of which are listed in Section 5, support this claim better than this article can.

Mechanizing Definitions in LF. Language formalization efforts frequently start from an informal, on-paper definition of the language to be mechanized—in, for example, a textbook or a research paper. What does it mean to correctly represent such a language definition in a formal framework? In the methodology of the the LF logical framework (Harper *et al.*, 1993), an LF representation of a language is *adequate* iff it is isomorphic to the informal definition of the language. Adequacy is a useful correctness criterion because any reasoning in or about an adequate LF representation applies equally well to the informal definition, and vice versa. For example, metatheoretic proofs about an adequate LF representation establish properties of the original definition as well.

LF is a minimal dependent type theory. The syntax and judgements of an *object language* (the object of study) are represented in LF as as the canonical forms (essentially long $\beta\eta$-normal forms) of associated LF types. These canonical forms are specified by an LF *signature*, which declares type and term constants, and by a *world*, which specifies the LF contexts under consideration. A representation is adequate iff there is an isomorphism between the informal presentation of the object language and the associated canonical forms. The LF methodology has been successfully used to derive representations of a wide variety of logical systems, often leading to new insights about their structure.

One can often use variable binding in LF to represent object-language binding forms and hypothetical judgements. With such higher-order representations of syntax and judgements, LF binding provides object-language operations such as α-conversion and capture-avoiding substitution and object-language judgement properties such as weakening and substitution. Because binding and substitution are present in both the informal presentation of a language and its LF representation, an isomorphism between the two must respect this structure. Thus, we say that a higher-order LF representation is adequate iff there is a *compositional bijection* between the informal presentation of the language and the associated canonical forms, where a bijection is compositional iff it commutes with substitution. This property ensures that the object language's variables and hypothetical judgements are correctly modeled.

Establishing the adequacy of a representation requires two main technical tools. First, adequacy proofs proceed by *induction on the canonical forms of LF*. The representation of an object language is not an inductive definition inside the LF type theory; indeed, higher-order encodings rely on negative occurrences of types. However, externally, the representation of a language as the canonical forms of par-

ticular types in a particular world *is* an inductive definition because the canonical forms of LF are inductively defined.

Second, modular adequacy proofs require *subordination-based transport of adequacy*. To modularly encode languages in LF, it is necessary to consider each fragment of the object language in only the LF signature and contexts relevant to its encoding. Combining these fragments requires understanding whether an encoding that is adequate in one signature and world remains adequate in another. To understand when adequacy may be transported, it suffices to consider conditions under which the canonical forms of a type remain invariant when transported from one signature and world to another. The judgements of LF are parametrized by a *subordination relation* (Virga, 1999), which determines whether canonical forms of one type can appear in canonical forms of another. Using subordination, it is possible to establish general conditions under which the canonical forms of a type, and therefore the adequacy theorems proved about them, remain invariant when transported from one signature and world to another.

Though adequacy underlies the LF methodology, it is not an obstacle to mechanization in practice. Experienced users of LF often define a language solely by its LF representation, never even stating an informal description. However, even such a definition will rely on an understanding of how to represent a language as the canonical forms of particular LF types in contexts of particular forms, which is exactly what an adequacy proof clarifies. Thus, we claim that all users of LF should understand adequacy, even if they do not carry out a proof for every object language.

Mechanizing Metatheory in Twelf. One application of adequate LF encodings is the formalization and verification of a language's metatheory. Since the entire deductive apparatus of a language is represented by certain canonical forms in LF, it is possible to reason about its properties by reasoning about the associated canonical forms. Specifically, an informal proof by induction on the structure of derivations can be re-cast, via an adequate encoding, as a structural induction on the canonical forms of the associated types.

This capability is exploited by the Twelf implementation of LF (Pfenning & Schürmann, 1999), which supports the mechanized proofs of $\forall\exists$-propositions over the canonical forms of specified types. This is sufficient to capture many useful properties of programming languages and logics. The Twelf methodology has proved to be very robust: it has been used to develop the metatheory of a wide variety of systems, including Mini-ML (Michaylov & Pfenning, 1991), the typed assembly language TALT (Crary, 2003), the POPLmark Challenge problem (Aydemir *et al.*, 2005), and the Harper-Stone internal language for Standard ML (Lee *et al.*, 2007). The same technical tools required for adequacy proofs are necessary for Twelf metatheory: Twelf proofs proceed by induction on the canonical forms of LF, and modular Twelf proofs require subordination-based transport of canonical forms.

Outline. In Section 2, we give a presentation of the LF type theory. Our presentation, called *Canonical LF*, follows Watkins *et al.* (2002, 2004a) in defining the

type theory so that only canonical forms are permitted, giving a direct inductive definition of the canonical forms. Additionally, we follow Virga (1999) in supporting subordination in the type theory. Next, we use a simply typed λ-calculus as a running example of mechanizing a language and its metatheory in LF and Twelf. In Section 3, we give an adequate LF encoding of the syntax and semantics of the simply typed λ-calculus. In Section 4, we show how to mechanize some of the metatheory of the simply typed λ-calculus in Twelf. We give mechanized proofs of several of its properties, including type preservation, determinacy of the operational semantics, and strengthening.

An electronic copy of the Twelf code presented in this article is available as supplementary material from the Journal of Functional Programming Web site (`http://www.cambridge.org/jfp/`).

2 Canonical LF

As we stated above, the LF methodology consists of representing object languages as canonical forms in a dependent type theory. In a dependent type theory, terms of the type theory are allowed to appear in types. In LF, dependency arises from considering families of types indexed by LF terms; these type families are used to represent object-language judgements, as we discuss in Section 3. In the presence of dependency, it is necessary to generalize the simple function type $\mathtt{A_1} \to \mathtt{A_2}$ to a dependent function type $\Pi\,\mathtt{x{:}A_1}.\,\mathtt{A_2}$. In such a type, the variable $\mathtt{x}$, which stands for the argument to the function, may appear in the result type $\mathtt{A_2}$; this permits the result type of the function to vary with the argument provided to it.

The canonical forms of LF are the long $\beta\eta$-normal forms: a term is a canonical form iff it is not β-reducible and it is η-expanded as much as possible without introducing β-redices. Traditional presentations of LF (Harper *et al.*, 1993; Salvesen, 1990; Geuvers, 1992; Harper & Pfenning, 2005) first define the type theory so that both canonical and non-canonical forms are well-typed and then give an inductive characterization of which terms are canonical forms. However, because the LF representation methodology requires only canonical forms, the non-canonical terms can be seen as a technical device, rather than as an essential part of the type theory. Technically, the reason to admit non-canonical forms is that they arise from the substitution of one canonical form into another—i.e., canonical forms are not closed under substitution. Such a substitution occurs in the typing rule for the application of a dependent function to an argument:

$$\frac{\mathtt{M_1} : \Pi\,\mathtt{x{:}A_2}.\,\mathtt{A} \quad \mathtt{M_2} : \mathtt{A_2}}{\mathtt{M_1}\ \mathtt{M_2} : [\mathtt{M_2}/\mathtt{x}]\mathtt{A}}$$

Even if $\mathtt{M_2}$ and all the terms embedded in the type $\Pi\,\mathtt{x{:}A_2}.\,\mathtt{A}$ are in canonical form, the substitution $[\mathtt{M_2}/\mathtt{x}]\mathtt{A}$ may introduce non-canonical forms if $\mathtt{A_2}$ is itself a function type. To equate non-canonical terms with an associated canonical form, traditional presentations of LF consider terms modulo β- or $\beta\eta$-equivalence.

Canonical LF, which is a fragment of Concurrent LF (Watkins *et al.*, 2002, 2004a), is a presentation of LF in which *only* the canonical forms are well-typed. The key

Kinds	$\mathtt{K}$	::=	$\mathtt{type} \mid \Pi\,\mathtt{x{:}A}.\,\mathtt{K}$
Canonical Type Families	$\mathtt{A}$	::=	$\mathtt{P} \mid \Pi\,\mathtt{x{:}A_2}.\,\mathtt{A}$
Atomic Type Families	$\mathtt{P}$	::=	$\mathtt{a} \mid \mathtt{P\,M}$
Canonical Terms	$\mathtt{M}$	::=	$\mathtt{R} \mid \lambda\,\mathtt{x}.\,\mathtt{M}$
Atomic Terms	$\mathtt{R}$	::=	$\mathtt{x} \mid \mathtt{c} \mid \mathtt{R\,M}$
Signatures	Σ	::=	$\cdot \mid \Sigma, \mathtt{c:A} \mid \Sigma, \mathtt{a:K}$
Contexts	Γ	::=	$\cdot \mid \Gamma, \mathtt{x:A}$

Fig. 1. LF Syntax

technical contribution of the canonical-forms approach is a notion of *hereditary substitution*, which directly computes the canonical form of the result of an ordinary substitution. In Canonical LF's application typing rule, hereditary substitution is used to directly compute the canonical form resulting from an ordinary substitution $[\mathtt{M_2}/\mathtt{x}]\mathtt{A}$; the intermediate non-canonical form is never considered. When a traditional substitution would result in a redex $(\lambda\,\mathtt{x}.\,\mathtt{M_1})\,\mathtt{M_2}$, the corresponding hereditary substitution continues by hereditarily substituting $\mathtt{M_2}$ for $\mathtt{x}$ in $\mathtt{M_1}$. The key insight made in Watkins *et al.* (2004a), which enables this approach, is that hereditary substitution can be defined in a structurally recursive manner, using the same algorithm as structural cut elimination (Pfenning, 1994). The corresponding structural induction principle is used to establish the metatheoretic properties of Canonical LF. Consequently, the metatheory of Canonical LF is simpler than presentations that admit non-canonical forms—there is no need to consider $\beta\eta$-equality.

Following Virga (1999), the judgements of Canonical LF are parametrized by a subordination relation, which determines when canonical forms of one type are relevant to canonical forms of another. Under appropriate subordination conditions, the canonical forms of a type remain invariant when considered in a different signature and world. Specifically, the addition or removal of canonical forms of types that are *not* subordinate to a given type does not change the canonical forms of that type.

In Sections 2.1, 2.2, and 2.3, we overview the syntax, judgements, and properties of Canonical LF. In Section 2.4, we give a formal definition of subordination and prove the transport of canonical forms theorem.

2.1 Syntax

In Figure 1, we present the syntax of LF. The term level includes functions, application, variables $\mathtt{x}$ (which are bound in contexts Γ), and constants $\mathtt{c}$ (which are declared in signatures Σ). Functions are given dependent function types $\Pi\,\mathtt{x{:}A_2}.\,\mathtt{A}$; we write $\mathtt{A_2} \to \mathtt{A}$ as an abbreviation when $\mathtt{x}$ is not free in $\mathtt{A}$. The base types are family-level constants $\mathtt{a}$ and their applications to terms. The original presentation of LF (Harper *et al.*, 1993) included family-level λ-abstractions; because these cannot

$\boxed{\vdash_{\preceq} \Sigma\ \mathtt{sig}}$

$$\frac{}{\vdash_{\preceq} \cdot\ \mathtt{sig}}\ \text{SIG\_EMPTY} \qquad \frac{\vdash_{\preceq} \Sigma\ \mathtt{sig} \quad \cdot \vdash_{\Sigma,\preceq} \mathtt{A\ type} \quad \mathtt{c}\#\Sigma}{\vdash_{\preceq} \Sigma, \mathtt{c}:\mathtt{A}\ \mathtt{sig}}\ \text{SIG\_TERM}$$

$$\frac{\vdash_{\preceq} \Sigma\ \mathtt{sig} \quad \cdot \vdash_{\Sigma,\preceq} \mathtt{K\ kind} \quad \mathtt{a}\#\Sigma}{\vdash_{\preceq} \Sigma, \mathtt{a}:\mathtt{K}\ \mathtt{sig}}\ \text{SIG\_FAM}$$

$\boxed{\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}}$

$$\frac{}{\vdash_{\Sigma,\preceq} \cdot\ \mathtt{ctx}}\ \text{CTX\_EMPTY} \qquad \frac{\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx} \quad \Gamma \vdash_{\Sigma,\preceq} \mathtt{A\ type} \quad \mathtt{x}\#\Gamma}{\vdash_{\Sigma,\preceq} \Gamma, \mathtt{x}:\mathtt{A}\ \mathtt{ctx}}\ \text{CTX\_TERM}$$

Fig. 2. LF Signature and Context Formation

appear in canonical types, we omit them from the current presentation. The kind `type` classifies types; the inhabitants of dependent function kinds are family-level constants and their applications to terms.

To admit only canonical forms, the syntax of terms is stratified into two categories, the canonical terms `M` and the atomic terms `R`; this syntactically precludes β-redices. For example, the term $(\lambda\,\mathtt{x}.\,\mathtt{x})\ \mathtt{c}$ is not syntactically correct, let alone well-typed, because the λ-abstraction is not an atomic term `R`. The type family level is similarly stratified into `A` and `P` to differentiate between Π-types and base types; this stratification will be used in the formation rules to ensure that well-typed terms are η-long.

We implicitly consider all expressions `M`, `R`, `A`, `P`, and `K` up to α-equivalence. By convention, when we write a binding form ($\lambda\,\mathtt{x}.\,\mathtt{M}$, $\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}$, or $\Pi\,\mathtt{x}{:}\mathtt{A}.\,\mathtt{K}$), the bound variable is chosen to be fresh.

2.2 Judgements

The judgements defining LF are presented in Figures 2, 3, 4, and 5. For each judgement, we first identify the judgement form in a box, (e.g., $\boxed{\Sigma\ \mathtt{sig}}$), and then we give the inference rules defining that judgement.

Parameters to the Formation Judgements. The signature formation judgement is parametrized by a subordination relation $\preceq$, which is a binary relation between type families. The context, kind, type family, and term formation judgements are parametrized by a signature Σ and a subordination relation $\preceq$. Subordination is discussed in detail in Section 2.4. Taking $\preceq$ to be the complete relation imposes no restrictions on which judgements are derivable, recovering a presentation of Canonical LF without subordination as a special case.

Signature and Context Formation. The judgements in Figure 2 define signature and context formation. These judgements ensure that all variables or constants

$\boxed{\Gamma \vdash_{\Sigma,\preceq} \mathtt{K}\ \mathtt{kind}}$

$$\frac{}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{type}\ \mathtt{kind}}\ \text{CANON\_KIND\_TYPE}$$

$$\frac{\Gamma \vdash_{\Sigma,\preceq} \mathtt{A}\ \mathtt{type} \quad \Gamma, \mathtt{x}:\mathtt{A} \vdash_{\Sigma,\preceq} \mathtt{K}\ \mathtt{kind}}{\Gamma \vdash_{\Sigma,\preceq} \Pi\,\mathtt{x}{:}\mathtt{A}.\,\mathtt{K}\ \mathtt{kind}}\ \text{CANON\_KIND\_PI}$$

$\boxed{\Gamma \vdash_{\Sigma,\preceq} \mathtt{A}\ \mathtt{type}}$

$$\frac{\Gamma \vdash_{\Sigma,\preceq} \mathtt{P} \Rightarrow \mathtt{type}}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{P}\ \mathtt{type}}\ \text{CANON\_FAM\_ATOM}$$

$$\frac{\Gamma \vdash_{\Sigma,\preceq} \mathtt{A}_2\ \mathtt{type} \quad \Gamma, \mathtt{x}:\mathtt{A}_2 \vdash_{\Sigma,\preceq} \mathtt{A}\ \mathtt{type} \quad \mathtt{A}_2 \preceq \mathtt{A}}{\Gamma \vdash_{\Sigma,\preceq} \Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}\ \mathtt{type}}\ \text{CANON\_FAM\_PI}$$

$\boxed{\Gamma \vdash_{\Sigma,\preceq} \mathtt{P} \Rightarrow \mathtt{K}}$

$$\frac{\mathtt{a}:\mathtt{K}\ \text{in}\ \Sigma}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{a} \Rightarrow \mathtt{K}}\ \text{ATOM\_FAM\_CONST}$$

$$\frac{\Gamma \vdash_{\Sigma,\preceq} \mathtt{P}_1 \Rightarrow \Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{K}_1 \qquad \Gamma \vdash_{\Sigma,\preceq} \mathtt{M}_2 \Leftarrow \mathtt{A}_2 \quad [\mathtt{M}_2/\mathtt{x}]^k_{\mathtt{A}_2}\mathtt{K}_1 = \mathtt{K}}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{P}_1\,\mathtt{M}_2 \Rightarrow \mathtt{K}}\ \text{ATOM\_FAM\_APP}$$

$\boxed{\Gamma \vdash_{\Sigma,\preceq} \mathtt{M} \Leftarrow \mathtt{A}}$

$$\frac{\Gamma \vdash_{\Sigma,\preceq} \mathtt{R} \Rightarrow \mathtt{P}}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{R} \Leftarrow \mathtt{P}}\ \text{CANON\_TERM\_ATOM}$$

$$\frac{\Gamma, \mathtt{x}:\mathtt{A}_2 \vdash_{\Sigma,\preceq} \mathtt{M} \Leftarrow \mathtt{A}}{\Gamma \vdash_{\Sigma,\preceq} \lambda\,\mathtt{x}.\,\mathtt{M} \Leftarrow \Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}}\ \text{CANON\_TERM\_LAM}$$

$\boxed{\Gamma \vdash_{\Sigma,\preceq} \mathtt{R} \Rightarrow \mathtt{A}}$

$$\frac{\mathtt{x}:\mathtt{A}\ \text{in}\ \Gamma}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{x} \Rightarrow \mathtt{A}}\ \text{ATOM\_TERM\_VAR} \qquad \frac{\mathtt{c}:\mathtt{A}\ \text{in}\ \Sigma}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{c} \Rightarrow \mathtt{A}}\ \text{ATOM\_TERM\_CONST}$$

$$\frac{\Gamma \vdash_{\Sigma,\preceq} \mathtt{R}_1 \Rightarrow \Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}_1 \qquad \Gamma \vdash_{\Sigma,\preceq} \mathtt{M}_2 \Leftarrow \mathtt{A}_2 \quad [\mathtt{M}_2/\mathtt{x}]^a_{\mathtt{A}_2}\mathtt{A}_1 = \mathtt{A}}{\Gamma \vdash_{\Sigma,\preceq} \mathtt{R}_1\,\mathtt{M}_2 \Rightarrow \mathtt{A}}\ \text{ATOM\_TERM\_APP}$$

Fig. 3. LF Formation Judgements

Simple Types $\alpha \quad ::= \quad \mathtt{a} \mid \alpha_1 \rightarrow \alpha_2$

$\boxed{(\mathtt{A})^- = \alpha}$

$$\frac{}{(\mathtt{a})^- = \mathtt{a}} \qquad \frac{(\mathtt{P})^- = \alpha}{(\mathtt{P}\,\mathtt{M})^- = \alpha} \qquad \frac{(\mathtt{A}_2)^- = \alpha_2 \quad (\mathtt{A})^- = \alpha}{(\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A})^- = \alpha_2 \rightarrow \alpha}$$

Fig. 4. Erasure to Simple Types

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^k_{\alpha_0}\mathtt{K} = \mathtt{K}'}$

$$\frac{}{[\mathtt{M}_0/\mathtt{x}_0]^k_{\alpha_0}\mathtt{type} = \mathtt{type}}\ \text{SUBST\_K\_TYPE} \qquad \frac{[\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\mathtt{A} = \mathtt{A}' \quad [\mathtt{M}_0/\mathtt{x}_0]^k_{\alpha_0}\mathtt{K} = \mathtt{K}'}{[\mathtt{M}_0/\mathtt{x}_0]^k_{\alpha_0}\Pi\,\mathtt{x}{:}\mathtt{A}.\,\mathtt{K} = \Pi\,\mathtt{x}{:}\mathtt{A}'.\,\mathtt{K}'}\ \text{SUBST\_K\_PI}$$

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\mathtt{A} = \mathtt{A}'}$

$$\frac{[\mathtt{M}_0/\mathtt{x}_0]^p_{\alpha_0}\mathtt{P} = \mathtt{P}'}{[\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\mathtt{P} = \mathtt{P}'}\ \text{SUBST\_A\_P} \qquad \frac{[\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\mathtt{A}_2 = \mathtt{A}_2' \quad [\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\mathtt{A} = \mathtt{A}'}{[\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A} = \Pi\,\mathtt{x}{:}\mathtt{A}_2'.\,\mathtt{A}'}\ \text{SUBST\_A\_PI}$$

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^p_{\alpha_0}\mathtt{P} = \mathtt{P}'}$

$$\frac{}{[\mathtt{M}_0/\mathtt{x}_0]^p_{\alpha_0}\mathtt{a} = \mathtt{a}}\ \text{SUBST\_P\_CONST} \qquad \frac{[\mathtt{M}_0/\mathtt{x}_0]^p_{\alpha_0}\mathtt{P} = \mathtt{P}' \quad [\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M} = \mathtt{M}'}{[\mathtt{M}_0/\mathtt{x}_0]^p_{\alpha_0}\mathtt{P}\,\mathtt{M} = \mathtt{P}'\,\mathtt{M}'}\ \text{SUBST\_P\_APP}$$

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M} = \mathtt{M}'}$

$$\frac{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R} = \mathtt{M}' : \alpha'}{[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{R} = \mathtt{M}'}\ \text{SUBST\_M\_RH} \qquad \frac{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R} = \mathtt{R}'}{[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{R} = \mathtt{R}'}\ \text{SUBST\_M\_R}$$

$$\frac{[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M} = \mathtt{M}'}{[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\lambda\,\mathtt{x}.\,\mathtt{M} = \lambda\,\mathtt{x}.\,\mathtt{M}'}\ \text{SUBST\_M\_LAM}$$

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R} = \mathtt{M}' : \alpha}$

$$\frac{}{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{x}_0 = \mathtt{M}_0 : \alpha_0}\ \text{SUBST\_RH\_VAR}$$

$$\frac{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R}_1 = \lambda\,\mathtt{x}.\,\mathtt{M}_1' : \alpha_2 \to \alpha \quad [\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M}_2 = \mathtt{M}_2' \quad [\mathtt{M}_2'/\mathtt{x}]^m_{\alpha_2}\mathtt{M}_1' = \mathtt{M}'}{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R}_1\,\mathtt{M}_2 = \mathtt{M}' : \alpha}\ \text{SUBST\_RH\_APP}$$

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R} = \mathtt{R}'}$

$$\frac{\mathtt{x}\#\mathtt{x}_0}{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{x} = \mathtt{x}}\ \text{SUBST\_R\_VAR} \qquad \frac{}{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{c} = \mathtt{c}}\ \text{SUBST\_R\_CONST}$$

$$\frac{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R}_1 = \mathtt{R}_1' \quad [\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M}_2 = \mathtt{M}_2'}{[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R}_1\,\mathtt{M}_2 = \mathtt{R}_1'\,\mathtt{M}_2'}\ \text{SUBST\_R\_APP}$$

$\boxed{[\mathtt{M}_0/\mathtt{x}_0]^c_{\alpha_0}\Gamma = \Gamma'}$

$$\frac{}{[\mathtt{M}_0/\mathtt{x}_0]^c_{\alpha_0}\cdot = \cdot}\ \text{SUBST\_C\_EMPTY}$$

$$\frac{\mathtt{x}_0\#\mathtt{x} \quad \mathtt{x}\#\mathtt{M}_0 \quad [\mathtt{M}_0/\mathtt{x}_0]^c_{\alpha_0}\Gamma = \Gamma' \quad [\mathtt{M}_0/\mathtt{x}_0]^a_{\alpha_0}\mathtt{A} = \mathtt{A}'}{[\mathtt{M}_0/\mathtt{x}_0]^c_{\alpha_0}\Gamma, \mathtt{x} : \mathtt{A} = \Gamma', \mathtt{x} : \mathtt{A}'}\ \text{SUBST\_C\_TERM}$$

Fig. 5. LF Hereditary Substitution

declared in a context or signature are distinct and that all classifiers are well-formed in the preceding declarations. The judgements $\mathtt{x}\#\Gamma$, $\mathtt{c}\#\Sigma$, and $\mathtt{a}\#\Sigma$ assert that the variable or constant is not declared in the context or signature; we omit their inductive definitions. Note that the context formation judgement presupposes that the signature is well-formed, rather than checking signature formation at the leaves.

We use the term *world* to refer to a set of well-formed contexts:

Definition 2.1 (*World*)
Given Σ and $\preceq$ such that $\vdash_{\preceq} \Sigma\ \mathtt{sig}$, a world $\mathcal{W}$ is a set containing contexts Γ for which the judgement $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$ is derivable.

Kind, Family, and Term Formation. The judgements in Figure 3 define the formation of kinds, type families, and terms. We now call attention to several aspects of these rules:

- All of the formation judgements presuppose that $\vdash_{\preceq} \Sigma\ \mathtt{sig}$ and $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$. In the rules for binding forms, the presuppositions of the judgements and the premises of the rules always entail that the extended context in the premise of the rule is well-formed.
- The judgement $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{A}$ presupposes that the type $\mathtt{A}$ is well-formed in Γ. In contrast, the judgement $\Gamma \vdash \mathtt{R} \Rightarrow \mathtt{A}$ ensures that the type $\mathtt{A}$ is well-formed. These modes correspond to a bidirectional operational interpretation: a canonical term is checked against a type, whereas a type is synthesized from an atomic term. The atomic type family judgement also synthesizes its kind, while the remaining judgements check that a type or kind is well-formed. The direction of the arrow in the judgement form serves as a mnemonic for the flow of information: in $\mathtt{M} \Leftarrow \mathtt{A}$, information in the type is used to check the term; in $\mathtt{R} \Rightarrow \mathtt{A}$, information in the term is used to synthesize the type. Because λ-abstractions are always checked against a known type, they do not require a type annotation for the bound variable.
- Variables and constants must be declared in the context or signature to be well-formed. As the variable rule ATOM_TERM_VAR explicates, the notation $\mathtt{x} : \mathtt{A}$ in contexts Γ stands for hypothetical assumptions of $\vdash_{\Sigma,\preceq} \mathtt{x} \Rightarrow \mathtt{A}$. Consequently, the usual substitution principle for the hypothetical judgement permits the substitution of a derivation of $\mathtt{R} \Rightarrow \mathtt{A}$ for an assumption $\mathtt{x} \Rightarrow \mathtt{A}$. In contrast, it requires further justification to substitute a derivation of $\mathtt{M} \Leftarrow \mathtt{A}$ for an assumption $\mathtt{x} \Rightarrow \mathtt{A}$; this justification is provided by hereditary substitution. Note that we do not consider substitutions for constants.
- The subordination relation parameter to the judgements is used only in the premise of the rule CANON_FAM_PI; this premise ensures that for any well-formed type $\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}$, the relation $\mathtt{A}_2 \preceq \mathtt{A}$ holds. When $\preceq$ is taken to be the complete relation, this premise is always satisfiable. The rationale for this premise is described in Section 2.4.
- The rules ATOM_TERM_APP and ATOM_FAM_APP have hereditary substitution premises that compute the result type and kind of an application.

- In the rule CANON_TERM_ATOM, the syntactic restriction of the classifier to a $\mathtt{P}$, rather than any $\mathtt{A}$, ensures that canonical forms are η-long. For example, the judgement $\mathtt{f} : (\mathtt{a} \to \mathtt{a}) \vdash \mathtt{f} \Leftarrow (\mathtt{a} \to \mathtt{a})$ is not derivable: $(\mathtt{a} \to \mathtt{a})$ is a Π-type, so CANON_TERM_ATOM cannot be applied, and the variable $\mathtt{f}$ is not a λ-abstraction, so neither can CANON_TERM_LAM. However, in a signature containing the declaration $\mathtt{a} : \mathtt{type}$, the canonicity of the η-expansion of $\mathtt{f}$ can be derived as follows (let Γ abbreviate the context $\mathtt{f} : (\mathtt{a} \to \mathtt{a})$, $\mathtt{x} : \mathtt{a}$):

$$\dfrac{\dfrac{}{\Gamma \vdash \mathtt{f} \Rightarrow (\mathtt{a} \to \mathtt{a})} \quad \dfrac{\dfrac{}{\Gamma \vdash \mathtt{x} \Rightarrow \mathtt{a}}}{\Gamma \vdash \mathtt{x} \Leftarrow \mathtt{a}}\ \text{CTA} \quad \dfrac{\vdots}{[\mathtt{x}/\_]^{a}_{\mathtt{a}}\mathtt{a} = \mathtt{a}}}{\dfrac{\dfrac{\Gamma \vdash \mathtt{f}\,\mathtt{x} \Rightarrow \mathtt{a}}{\Gamma \vdash \mathtt{f}\,\mathtt{x} \Leftarrow \mathtt{a}}\ \text{CTA}}{\mathtt{f} : (\mathtt{a} \to \mathtt{a}) \vdash \lambda\mathtt{x}.\,\mathtt{f}\,\mathtt{x} \Leftarrow (\mathtt{a} \to \mathtt{a})}} \quad .$$

  Both inferences labeled CTA for CANON_TERM_ATOM occur at base type, as required.

These formation judgements give a direct inductive definition of the canonical forms of LF. Consequently, rule induction for these judgements may be used to reason about canonical forms.

Hereditary Substitution. Next, we define hereditary substitution, which computes the canonical result of substituting one canonical form into another. The hereditary substitution judgements are defined in Figure 5. Hereditary substitution is defined on α-equivalence classes of expressions; by our notational convention, a bound variable $\mathtt{x}$ is always distinct from both the variable being substituted for and the term being substituted for it ($\mathtt{x}_0$ and $\mathtt{M}_0$ in the rules). Written in this relational style, a standard substitution judgement $[\mathtt{M}_0/\mathtt{x}]\mathtt{M} = \mathtt{M}'$ would have four parameters: the term being substituted ($\mathtt{M}_0$), the variable being substituted for ($\mathtt{x}$), the term being substituted into ($\mathtt{M}$), and the result of the substitution ($\mathtt{M}'$). The hereditary substitution judgement $[\mathtt{M}_0/\mathtt{x}]^{m}_{\alpha_0}\mathtt{M} = \mathtt{M}'$ adds an extra parameter α_0, which is the simple type of the substituted term $\mathtt{M}_0$ (the superscript m is simply notation for differentiating the judgements for the various syntactic classes from one another). The syntax of simple types α is defined in Figure 4. The simple type guides the process of hereditary substitution: because of the simple type parameter to the judgement, it is decidable whether or not a hereditary substitution exists even when the terms involved are ill-formed (see *Theorem 2.4* below).

The key hereditary substitution judgement is $[\mathtt{M}_0/\mathtt{x}]^{r}_{\alpha_0}\mathtt{R} = \mathtt{M}' : \alpha'$. This judgement computes the canonical result of substituting a canonical form $\mathtt{M}_0$ into an atomic term $\mathtt{R}$ whose head is the variable $\mathtt{x}$; it computes a new canonical term $\mathtt{M}'$ and its simple type α'. The key rule is SUBST_RH_APP: when the substitution into the function position of an application yields a λ-abstraction, this rule continues by hereditarily substituting the argument $\mathtt{M}'_2$ into the body of the function. This is the rule where the simple type is used to guide the process of hereditary substitution: the simple type resulting from the first premise must have the form $\alpha_2 \to \alpha$, and the simple-type input to the third premise is α_2 (which, on well-typed terms, is

the simple type of $\mathtt{M}_2'$) rather than α_0. We prove below that the simple type α_2 in this final premise is always smaller than the input simple type α_0; this fact is used to establish decidability of hereditary substitution and to prove that hereditary substitutions exist under the appropriate typing premises.

The other hereditary substitution judgement on atomic terms, $[\mathtt{M}_0/\mathtt{x}]^r_{\alpha_0}\mathtt{R} = \mathtt{R}'$, is derivable only when the variable $\mathtt{x}$ is not the head variable of $\mathtt{R}$; it computes another atomic term compositionally. The remaining hereditary substitution judgements are defined compositionally as well. Because there is no single scope in which a variable declared in Γ can be renamed, the premises of the rule SUBST_C_TERM, which defines hereditary substitution into a context, insist that the variable in the context does not interfere with the substitution. The auxiliary judgement $\mathtt{x}\#\mathtt{M}$ holds when $\mathtt{x}$ is not free in the term $\mathtt{M}$; we elide its standard inductive definition. Intuitively, the premises of this rule ensure that the hereditary substitution into a context Γ is not defined when either the variable $\mathtt{x}_0$ is declared in Γ or when a free variable in $\mathtt{M}_0$ would be captured by a declaration in Γ.

The judgement $(\mathtt{A})^- = \alpha$ in Figure 4 defines an erasure relation that computes a simple type α from a dependent type $\mathtt{A}$. Observe (by induction on the structure of type families) that for all $\mathtt{A}$, there exists a unique α such that $(\mathtt{A})^- = \alpha$; this justifies using function notation for $(\mathtt{A})^-$. As a convenience, we write $[\mathtt{M}_0/\mathtt{x}]^m_{\mathtt{A}_0}\mathtt{M} = \mathtt{M}'$ to mean $[\mathtt{M}_0/\mathtt{x}]^m_{\alpha_0}\mathtt{M} = \mathtt{M}'$ where $(\mathtt{A}_0)^- = \alpha_0$; we adopt the analogous convention for the other syntactic categories. This notational convention is used by the hereditary substitution premises of the formation rules.

Notation. In the remainder of this article, we adopt several additional notational conveniences. We elide the signature Σ and the subordination relation $\preceq$ parameters to the formation judgements when they are clear from context, as they are invariant throughout a derivation. To make the default instantiation of the parameters clear in a particular context, we will say that we work in LF$[\Sigma, \preceq]$. Also, we abbreviate a hypothetical judgement in the empty context by eliding the turnstile, writing, for example, $\mathtt{R} \Rightarrow \mathtt{A}$ for $\cdot \vdash \mathtt{R} \Rightarrow \mathtt{A}$.

2.3 Metatheory of Canonical LF

In this section, we prove two major theorems about Canonical LF. First, we prove decidability results for hereditary substitution and type checking. Next, we prove that hereditary substitutions exist and preserve types when the terms involved are well-formed and the types align in the appropriate way. Watkins *et al.* (2002) discuss these theorems and their proofs in more detail.

The remainder of this article does not require understanding the proofs of these two theorems about Canonical LF, so we provide proof sketches only as a reference for interested readers. However, the adequacy proofs below do make use of these results. A reader who is not interested in the metatheoretic development of LF may skip this section on first reading and refer back to the theorem statements as necessary to understand the subsequent adequacy proofs.

As a notational convenience in the following theorem statements, we use the word

"expression" to refer generically to any of the syntactic classes of Canonical LF. We also write $\mathtt{E}$ as a general metavariable for any syntactic category $\mathtt{K}, \mathtt{A}, \mathtt{P}, \mathtt{M}, \mathtt{R}, \Gamma$, and we use e in $\{k, a, p, m, r, c\}$ for the corresponding tag on the hereditary substitution judgement. We write $\mathtt{x}\#\mathtt{E}$ to mean that the variable $\mathtt{x}$ does not occur free in the expression $\mathtt{E}$.

The staging of lemmas leading up to the main results is somewhat intricate, and in some cases it is desirable to strengthen a theorem statement so that it holds even when certain subjects are not well-formed. In particular, we will state some theorems about formation judgements whose contexts are not known to be well-formed. However, even in these cases, we tacitly assume that all expressions have correct variable scoping. Specifically, when we write a context $\mathtt{x_1} : \mathtt{A_1}, \ldots, \mathtt{x_n} : \mathtt{A_n}$, we assume that all variables $\mathtt{x_i}$ are distinct and that the free variables of $\mathtt{A_i}$ are a subset of $\{\mathtt{x_1}, \ldots, \mathtt{x_{i-1}}\}$. Additionally, we only write a judgement form such as $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{A}$ when all free variables of $\mathtt{M}$ and $\mathtt{A}$ are declared in Γ. Observe that, given a hereditary substitution $[\mathtt{M_0}/\mathtt{x_0}]^e_{\mathtt{A_0}}\mathtt{E} = \mathtt{E'}$, the free variables of $\mathtt{E'}$, written $FV(\mathtt{E'})$, are $(FV(\mathtt{E}) - \{\mathtt{x_0}\}) \cup FV(\mathtt{M_0})$.

In the following theorem statements, when we leave the signature and subordination relation parameters to a theorem statement implicit, we tacitly assume a subordination relation $\preceq$ and signature Σ such that $\vdash_{\preceq} \Sigma\ \mathtt{sig}$.

2.3.1 Decidability

Lemma 2.2 (*Head Substitution Size*)
If $[\mathtt{M_0}/\mathtt{x_0}]^r_{\alpha_0}\mathtt{R} = \mathtt{M'} : \alpha$ then α is a subexpression of α_0.

Proof
By induction on the derivation of $[\mathtt{M_0}/\mathtt{x_0}]^r_{\alpha_0}\mathtt{R} = \mathtt{M'} : \alpha$. In the case for SUBST_RH_VAR, α_0 and α are identical. In the case for SUBST_RH_APP, the inductive hypothesis gives that $\alpha_2 \to \alpha$ is a subexpression of α_0, so α is as well. □

Lemma 2.3 (*Uniqueness of Substitution and Synthesis*)

1. If $[\mathtt{M_0}/\mathtt{x_0}]^r_{\alpha_0}\mathtt{R} = \mathtt{R'}$ and $[\mathtt{M_0}/\mathtt{x_0}]^r_{\alpha_0}\mathtt{R} = \mathtt{M'} : \alpha'$ then false.
2. For any $\mathtt{E}$ in $\{\mathtt{K}, \mathtt{A}, \mathtt{P}, \mathtt{M}, \mathtt{R}\}$, if $[\mathtt{M_0}/\mathtt{x_0}]^e_{\alpha_0}\mathtt{E} = \mathtt{E'}$ and $[\mathtt{M_0}/\mathtt{x_0}]^e_{\alpha_0}\mathtt{E} = \mathtt{E''}$ then $\mathtt{E'} = \mathtt{E''}$.
3. If $\Gamma \vdash \mathtt{R} \Rightarrow \mathtt{A}$ and $\Gamma \vdash \mathtt{R} \Rightarrow \mathtt{A'}$ then $\mathtt{A} = \mathtt{A'}$.
4. If $\Gamma \vdash \mathtt{P} \Rightarrow \mathtt{K}$ and $\Gamma \vdash \mathtt{P} \Rightarrow \mathtt{K'}$ then $\mathtt{K} = \mathtt{K'}$.

Because we develop the metatheory of Canonical LF in an informal constructive logic, the following statements of decidability are sensible.

Theorem 2.4 (*Decidability of Substitution*)

1. For any $\mathtt{E}$ in $\{\mathtt{K}, \mathtt{A}, \mathtt{P}, \mathtt{M}, \Gamma\}$, given $\mathtt{M_0}$, α_0, $\mathtt{x_0}$, either there exists an $\mathtt{E'}$ such that $[\mathtt{M_0}/\mathtt{x_0}]^e_{\alpha_0}\mathtt{E} = \mathtt{E'}$ or there is no such $\mathtt{E'}$.
2. Given $\mathtt{M_0}$, α_0, $\mathtt{x_0}$, and $\mathtt{R}$, either there exists an $\mathtt{R'}$ such that $[\mathtt{M_0}/\mathtt{x_0}]^r_{\alpha_0}\mathtt{R} = \mathtt{R'}$ or there exist $\mathtt{M'}$ and α' such that $[\mathtt{M_0}/\mathtt{x_0}]^r_{\alpha_0}\mathtt{R} = \mathtt{M'} : \alpha'$ or there are no such $\mathtt{R'}$ and $\mathtt{M'}$.

Proof

First, we prove part 2 and the clause of part 1 for canonical terms $\mathtt{M}$ by mutual lexicographic induction on $(\mathtt{A}_0)^-$, the terms $\mathtt{M}$ and $\mathtt{R}$ being substituted into, and an order permitting inductive calls to the clause for atomic terms from the clause for canonical terms on the same simple type and term (to account for the silent injection from $\mathtt{R}$ to $\mathtt{M}$). Then, we prove the clauses of part 1 for $\mathtt{P}$, $\mathtt{A}$, $\mathtt{K}$, and Γ in that order, each by induction on the expression being substituted into, and each using the previous parts. □

Theorem 2.5 (*Decidability of Formation*)
Assume that the subordination relation $\mathtt{A} \preceq \mathtt{B}$ is decidable.

1. For all Σ and $\preceq$, either $\vdash_{\preceq} \Sigma\ \mathtt{sig}$ or not.

Assume $\preceq$ and Σ such that that $\vdash_{\preceq} \Sigma\ \mathtt{sig}$.

2. For all Γ and $\mathtt{K}$, either $\Gamma \vdash \mathtt{K}\,\mathtt{kind}$ or not.
3. For all Γ and $\mathtt{A}$, either $\Gamma \vdash \mathtt{A}\,\mathtt{type}$ or not.
4. For all Γ and $\mathtt{P}$, either there exists a $\mathtt{K}$ such that $\Gamma \vdash \mathtt{P} \Rightarrow \mathtt{K}$ or there is no such $\mathtt{K}$.
5. For all Γ, $\mathtt{M}$, and $\mathtt{A}$, either $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{A}$ or not.
6. For all Γ and $\mathtt{R}$, either there exists an $\mathtt{A}$ such that $\Gamma \vdash \mathtt{R} \Rightarrow \mathtt{A}$ or there is no such $\mathtt{A}$.
7. For all Γ, either $\Gamma\ \mathtt{ctx}$ or not.

Proof
First, we prove parts 5 and 6 by mutual lexicographic induction on first the term and second an order permitting inductive calls to the clause for atomic terms from the clause for canonical terms on the same term. Then, we prove parts 4, 3, 2, 7, and 1, each by induction on the expression being judged well-formed. □

2.3.2 Substitution Theorem

We now prove that under the appropriate typing conditions hereditary substitutions exist and preserve types. We write $\Gamma \subseteq \Gamma'$ iff Γ can be obtained from Γ' by removing zero or more declarations; we use the notation $\Sigma \subseteq \Sigma'$ analogously. Viewing subordination relations as sets of pairs of types, we write $\preceq \subseteq \preceq'$ for the usual subset order.

Lemma 2.6 (*Weakening of Signature and Context*)
Assume $\Gamma \subseteq \Gamma'$, $\Sigma \subseteq \Sigma'$ and $\preceq \subseteq \preceq'$.

1. For all five formation judgements $\mathcal{J}$, if $\Gamma \vdash_{\Sigma,\preceq} \mathcal{J}$ then $\Gamma' \vdash_{\Sigma',\preceq'} \mathcal{J}$.
2. If $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$ then $\vdash_{\Sigma',\preceq'} \Gamma\ \mathtt{ctx}$.
3. If $\vdash_{\preceq} \Sigma\ \mathtt{sig}$ then $\vdash_{\preceq'} \Sigma\ \mathtt{sig}$.

Lemma 2.7 (*Exchange*)
For all five formation judgements, if $\Gamma, \mathtt{x}:\mathtt{A}, \mathtt{y}:\mathtt{B}, \Gamma' \vdash \mathcal{J}$ then $\Gamma, \mathtt{y}:\mathtt{B}, \mathtt{x}:\mathtt{A}, \Gamma' \vdash \mathcal{J}$.

Exchange for the signature Σ also holds, but we do not require it in this article.

Lemma 2.8 (*Vacuous Substitutions*)
For all $\mathtt{M}_0$, $\mathtt{x}_0$, $\mathtt{A}_0$, and $\mathtt{E}$ among $\{\mathtt{K}, \mathtt{A}, \mathtt{P}, \mathtt{M}, \mathtt{R}\}$, if $\mathtt{x}_0 \# \mathtt{E}$ then $[\mathtt{M}_0/\mathtt{x}_0]^e_{\mathtt{A}_0}\mathtt{E} = \mathtt{E}$.

Lemma 2.9 (*Erasure is Invariant Under Substitution*)
For all $\mathtt{E}$ in $\{\mathtt{A}, \mathtt{P}\}$, if $[\mathtt{M}_0/\mathtt{x}_0]^e_{\alpha_0}\mathtt{E} = \mathtt{E}'$ then $(\mathtt{E})^- = (\mathtt{E}')^-$.

Ordinary substitutions enjoy the following composition property:

$$[\mathtt{e}_0/\mathtt{x}_0][\mathtt{e}_2/\mathtt{x}]\mathtt{e}_1 = [[\mathtt{e}_0/\mathtt{x}_0]\mathtt{e}_2/\mathtt{x}][\mathtt{e}_0/\mathtt{x}_0]\mathtt{e}_1$$

assuming that $\mathtt{x} \# \mathtt{x}_0$ and $\mathtt{x} \# \mathtt{e}_0$. The following lemma establishes an analogous property for hereditary substitutions. In addition to showing that the results of the two substitutions are equal, it shows that the outer two substitutions are defined if the inner three are.

Lemma 2.10 (*Composition of Substitution*)
Assume $\mathtt{x} \# \mathtt{x}_0$ and $\mathtt{x} \# \mathtt{M}_0$.

1. For all $\mathtt{E}$ in $\{\mathtt{K}, \mathtt{A}, \mathtt{P}, \mathtt{M}, \mathtt{R}\}$, if $[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M}_2 = \mathtt{M}_2'$ and $[\mathtt{M}_2/\mathtt{x}]^e_{\alpha_2}\mathtt{E}_1 = \mathtt{E}$ and $[\mathtt{M}_0/\mathtt{x}_0]^e_{\alpha_0}\mathtt{E}_1 = \mathtt{E}_1'$ then there exists an $\mathtt{E}'$ such that $[\mathtt{M}_0/\mathtt{x}_0]^e_{\alpha_0}\mathtt{E} = \mathtt{E}'$ and $[\mathtt{M}_2'/\mathtt{x}]^e_{\alpha_2}\mathtt{E}_1' = \mathtt{E}'$.
2. If $[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M}_2 = \mathtt{M}_2'$ and $[\mathtt{M}_2/\mathtt{x}]^r_{\alpha_2}\mathtt{R}_1 = \mathtt{M} : \alpha$ and $[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R}_1 = \mathtt{R}_1'$ then there exists an $\mathtt{M}'$ such that $[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M} = \mathtt{M}'$ and $[\mathtt{M}_2'/\mathtt{x}]^r_{\alpha_2}\mathtt{R}_1' = \mathtt{M}' : \alpha$.
3. If $[\mathtt{M}_0/\mathtt{x}_0]^m_{\alpha_0}\mathtt{M}_2 = \mathtt{M}_2'$ and $[\mathtt{M}_2/\mathtt{x}]^r_{\alpha_2}\mathtt{R}_1 = \mathtt{R}$ and $[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R}_1 = \mathtt{M}_1' : \alpha$ then there exists an $\mathtt{M}'$ such that $[\mathtt{M}_0/\mathtt{x}_0]^r_{\alpha_0}\mathtt{R} = \mathtt{M}' : \alpha$ and $[\mathtt{M}_2'/\mathtt{x}]^m_{\alpha_2}\mathtt{M}_1' = \mathtt{M}'$.

The additional two composition properties for atomic terms $\mathtt{R}$ apply when the head variable of $\mathtt{R}$ is $\mathtt{x}$ or $\mathtt{x}_0$.

Proof
Define $size(\mathtt{a}) = 1$ and $size(\alpha_1 \to \alpha_2) = 1 + size(\alpha_1) + size(\alpha_2)$. The proof is by mutual lexicographic induction on first $size(\alpha_0) + size(\alpha_2)$ and then the derivation of the substitution of $\mathtt{M}_2$. The size metric is necessary to justify an inductive call in which α_0 and α_2 are swapped. □

Theorem 2.11 (*Substitution*)
Assume Σ and $\preceq$ such that $\vdash_\preceq \Sigma\,\mathtt{sig}$. Assume $\Gamma_L, \mathtt{x}_0 : \mathtt{A}_0, \Gamma_R\,\mathtt{ctx}$ and $\Gamma_L \vdash \mathtt{M}_0 \Leftarrow \mathtt{A}_0$. Further, assume that subordination relationships $\mathtt{A} \preceq \mathtt{B}$ are preserved by hereditary substitution into $\mathtt{A}$ and $\mathtt{B}$. Then:

1. There exists a Γ_R' such that $[\mathtt{M}_0/\mathtt{x}_0]^c_{\mathtt{A}_0}\Gamma_R = \Gamma_R'$ and $\Gamma_L, \Gamma_R'\ \mathtt{ctx}$.
2. If $\Gamma_L, \mathtt{x}_0 : \mathtt{A}_0, \Gamma_R \vdash \mathtt{K}\,\mathtt{kind}$ then there exists a $\mathtt{K}'$ such that $[\mathtt{M}_0/\mathtt{x}_0]^k_{\mathtt{A}_0}\mathtt{K} = \mathtt{K}'$ and $\Gamma_L, \Gamma_R' \vdash \mathtt{K}'\,\mathtt{kind}$.
3. If $\Gamma_L, \mathtt{x}_0 : \mathtt{A}_0, \Gamma_R \vdash \mathtt{A}\,\mathtt{type}$ then there exists an $\mathtt{A}'$ such that $[\mathtt{M}_0/\mathtt{x}_0]^a_{\mathtt{A}_0}\mathtt{A} = \mathtt{A}'$ and $\Gamma_L, \Gamma_R' \vdash \mathtt{A}'\,\mathtt{type}$.
4. If $\Gamma_L, \mathtt{x}_0 : \mathtt{A}_0, \Gamma_R \vdash \mathtt{A}\,\mathtt{type}$ and $\Gamma_L, \mathtt{x}_0 : \mathtt{A}_0, \Gamma_R \vdash \mathtt{M} \Leftarrow \mathtt{A}$ then there exist $\mathtt{A}'$ and $\mathtt{M}'$ such that $[\mathtt{M}_0/\mathtt{x}_0]^a_{\mathtt{A}_0}\mathtt{A} = \mathtt{A}'$ and $[\mathtt{M}_0/\mathtt{x}_0]^m_{\mathtt{A}_0}\mathtt{M} = \mathtt{M}'$ and $\Gamma_L, \Gamma_R' \vdash \mathtt{M}' \Leftarrow \mathtt{A}'$.

Proof

To prove this substitution theorem, we strengthen the theorem statements so that they work over contexts and types that are not necessarily well-formed. By strengthening the theorem in this way, we may prove the clause for terms without yet knowing that substitutions into type families preserve well-formedness. For conciseness, in the following theorem statements we leave implicit the existential quantification of the results of hereditary substitution. We first prove the following two clauses:

- If $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{M} \Leftarrow \mathtt{A}$ and $\Gamma_L \vdash \mathtt{M_0} \Leftarrow \mathtt{A_0}$ and $[\mathtt{M_0}/\mathtt{x_0}]^c_{\mathtt{A_0}}\Gamma_R = \Gamma'_R$ and $[\mathtt{M_0}/\mathtt{x_0}]^a_{\mathtt{A_0}}\mathtt{A} = \mathtt{A'}$ then $[\mathtt{M_0}/\mathtt{x_0}]^m_{\mathtt{A_0}}\mathtt{M} = \mathtt{M'}$ and $\Gamma_L, \Gamma'_R \vdash \mathtt{M'} \Leftarrow \mathtt{A'}$.
- If $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{R} \Rightarrow \mathtt{A}$ and $\Gamma_L \vdash \mathtt{M_0} \Leftarrow \mathtt{A_0}$ and $[\mathtt{M_0}/\mathtt{x_0}]^c_{\mathtt{A_0}}\Gamma_R = \Gamma'_R$ then $[\mathtt{M_0}/\mathtt{x_0}]^a_{\mathtt{A_0}}\mathtt{A} = \mathtt{A'}$ and either $[\mathtt{M_0}/\mathtt{x_0}]^r_{\mathtt{A_0}}\mathtt{R} = \mathtt{R'}$ and $\Gamma_L, \Gamma'_R \vdash \mathtt{R'} \Rightarrow \mathtt{A'}$ or $[\mathtt{M_0}/\mathtt{x_0}]^r_{\mathtt{A_0}}\mathtt{R} = \mathtt{M'} : (\mathtt{A'})^-$ and $\Gamma_L, \Gamma'_R \vdash \mathtt{M'} \Leftarrow \mathtt{A'}$.

The proof is by mutual lexicographic induction on first the simple type $(\mathtt{A_0})^-$ and then the derivations of $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{M} \Leftarrow \mathtt{A}$ and $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{R} \Rightarrow \mathtt{A}$.

Next, we prove the analogous statements for the remaining syntactic categories:

- If $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{P} \Rightarrow \mathtt{K}$ and $\Gamma_L \vdash \mathtt{M_0} \Leftarrow \mathtt{A_0}$ and $[\mathtt{M_0}/\mathtt{x_0}]^c_{\mathtt{A_0}}\Gamma_R = \Gamma'_R$ then $[\mathtt{M_0}/\mathtt{x_0}]^k_{\mathtt{A_0}}\mathtt{K} = \mathtt{K'}$ and $[\mathtt{M_0}/\mathtt{x_0}]^p_{\mathtt{A_0}}\mathtt{P} = \mathtt{P'}$ and $\Gamma_L, \Gamma'_R \vdash \mathtt{P'} \Rightarrow \mathtt{K'}$.
- If $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{A}\ \mathtt{type}$ and $\Gamma_L \vdash \mathtt{M_0} \Leftarrow \mathtt{A_0}$ and $[\mathtt{M_0}/\mathtt{x_0}]^c_{\mathtt{A_0}}\Gamma_R = \Gamma'_R$ then $[\mathtt{M_0}/\mathtt{x_0}]^a_{\mathtt{A_0}}\mathtt{A} = \mathtt{A'}$ and $\Gamma_L, \Gamma'_R \vdash \mathtt{A'}\ \mathtt{type}$.
- If $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R \vdash \mathtt{K}\ \mathtt{kind}$ and $\Gamma_L \vdash \mathtt{M_0} \Leftarrow \mathtt{A_0}$ and $[\mathtt{M_0}/\mathtt{x_0}]^c_{\mathtt{A_0}}\Gamma_R = \Gamma'_R$ then $[\mathtt{M_0}/\mathtt{x_0}]^k_{\mathtt{A_0}}\mathtt{K} = \mathtt{K'}$ and $\Gamma_L, \Gamma'_R \vdash \mathtt{K'}\ \mathtt{kind}$.
- If $\Gamma_L, \mathtt{x_0} : \mathtt{A_0}, \Gamma_R\, \mathtt{ctx}$ and $\Gamma_L \vdash \mathtt{M_0} \Leftarrow \mathtt{A_0}$ then $[\mathtt{M_0}/\mathtt{x_0}]^c_{\mathtt{A_0}}\Gamma_R = \Gamma'_R$ and $\Gamma_L, \Gamma'_R\, \mathtt{ctx}$.

Each part is proved in sequence by induction on the derivation of the expression being substituted into, using the previous parts. Then the clauses of the main theorem may be obtained as simple corollaries. □

Lemma 2.12 (*Regularity*)

1. If $\Gamma\ \mathtt{ctx}$ and $\Gamma \vdash \mathtt{P} \Rightarrow \mathtt{K}$ then $\Gamma \vdash \mathtt{K}\ \mathtt{kind}$.
2. If $\Gamma\ \mathtt{ctx}$ and $\Gamma \vdash \mathtt{R} \Rightarrow \mathtt{A}$ then $\Gamma \vdash \mathtt{A}\ \mathtt{type}$.

This final lemma will be convenient in the adequacy proofs below; it asserts an n-ary hereditary substitution inversion principle for iterated Π-types and applications.

Lemma 2.13 (*Iterated Hereditary Substitution Inversion*)

1. If $[\mathtt{M}/\mathtt{x}]^a_{\mathtt{A}}\mathtt{B} = \Pi\, \mathtt{x_1}{:}\mathtt{A_1}. \ldots \Pi\, \mathtt{x_n}{:}\mathtt{A_n}.\, \mathtt{A_{n+1}}$ then there exist $\mathtt{A'_1}, \ldots, \mathtt{A'_{n+1}}$ such that $\mathtt{B} = \Pi\, \mathtt{x_1}{:}\mathtt{A'_1}. \ldots \Pi\, \mathtt{x_n}{:}\mathtt{A'_n}.\, \mathtt{A'_{n+1}}$ and $[\mathtt{M}/\mathtt{x}]^a_{\mathtt{A}}\mathtt{A'_i} = \mathtt{A_i}$ for $1 \leq \mathtt{i} \leq \mathtt{n}+1$.
2. If $[\mathtt{M}/\mathtt{x}]^a_{\mathtt{A}}\Pi\, \mathtt{x_1}{:}\mathtt{A_1}. \ldots \Pi\, \mathtt{x_n}{:}\mathtt{A_n}.\, \mathtt{A_{n+1}} = \mathtt{B}$ then there exist $\mathtt{A'_1}, \ldots, \mathtt{A'_{n+1}}$ such that $\mathtt{B} = \Pi\, \mathtt{x_1}{:}\mathtt{A'_1}. \ldots \Pi\, \mathtt{x_n}{:}\mathtt{A'_n}.\, \mathtt{A'_{n+1}}$ where $[\mathtt{M}/\mathtt{x}]^a_{\mathtt{A}}\mathtt{A_i} = \mathtt{A'_i}$ for $1 \leq \mathtt{i} \leq \mathtt{n}+1$.
3. If $[\mathtt{M}/\mathtt{x}]^a_{\mathtt{A}}\mathtt{B} = (\mathtt{a}\, \mathtt{M_1}\ \ldots\ \mathtt{M_n})$ then there exist $\mathtt{M'_1}, \ldots, \mathtt{M'_n}$ such that $\mathtt{B} = \mathtt{a}\, \mathtt{M'_1}\ \ldots\ \mathtt{M'_n}$ where $[\mathtt{M}/\mathtt{x}]^m_{\mathtt{A}}\mathtt{M'_i} = \mathtt{M_i}$ for $1 \leq \mathtt{i} \leq \mathtt{n}$.
4. If $[\mathtt{M}/\mathtt{x}]^a_{\mathtt{A}}(\mathtt{a}\, \mathtt{M_1}\ \ldots\ \mathtt{M_n}) = \mathtt{B}$ then there exist $\mathtt{M'_1}, \ldots, \mathtt{M'_n}$ such that $\mathtt{B} = \mathtt{a}\, \mathtt{M'_1}\ \ldots\ \mathtt{M'_n}$ where $[\mathtt{M}/\mathtt{x}]^m_{\mathtt{A}}\mathtt{M_i} = \mathtt{M'_i}$ for $1 \leq \mathtt{i} \leq \mathtt{n}$.

The complete metatheory of Canonical LF includes an additional theorem witnessing that a variable $\mathtt{x} : \mathtt{A}$ can be η-expanded into a canonical form of type $\mathtt{A}$.

Whereas hereditary substitution corresponds to cut admissibility for a sequent calculus, this theorem corresponds to an identity principle (i.e., that $\mathtt{A} \vdash \mathtt{A}$ for any $\mathtt{A}$). Because we do not require the identity theorem in this article, we refer the interested reader to Watkins *et al.* (2002) for details.

2.4 Subordination

Intuitively, a type family $\mathtt{a}$ is subordinate to a type family $\mathtt{b}$ if terms of type $\mathtt{a}$ can appear in either terms of type $\mathtt{b}$ or indices of the type family $\mathtt{b}$ (Virga, 1999). The definition of subordination requires an auxiliary judgement identifying the *head* constant of a type family $\mathtt{A}$.

$\boxed{|\mathtt{A}| = \mathtt{a}}$

$$\frac{}{|\mathtt{a}| = \mathtt{a}} \qquad \frac{|\mathtt{P}| = \mathtt{a}}{|\mathtt{P}\ \mathtt{M}| = \mathtt{a}} \qquad \frac{|\mathtt{A}| = \mathtt{a}}{|\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}| = \mathtt{a}}$$

This judgement identifies the family-level constant at the head of the base type to which terms of type $\mathtt{A}$ contribute, where a term of base type contributes to that base type and a term of function type contributes to the base type that results from fully applying it to arguments. Observe (by induction over the structure of type families) that for all $\mathtt{A}$ there exists a unique family-level constant $\mathtt{a}$ such that $|\mathtt{A}| = \mathtt{a}$; this justifies using $|\mathtt{A}|$ in a functional notation.

We now define the conditions under which a subordination relation is well-formed.

Definition 2.14 (*Subordination Relation*)
A *subordination relation* for a signature Σ is a binary relation $\preceq$ between family-level constants declared in Σ, presented as a list of tuples, that satisfies the following properties:

1. Well-formedness: The judgement $\vdash_{\preceq}\ \Sigma\ \mathtt{sig}$ is derivable.
2. Index subordination: For all declarations $\mathtt{a} : \Pi\,\mathtt{x}_1{:}\mathtt{A}_1 \ldots \Pi\,\mathtt{x}_\mathtt{n}{:}\mathtt{A}_\mathtt{n}.\,\mathtt{type}$ in Σ, $|\mathtt{A}_\mathtt{i}| \preceq \mathtt{a}$ for all $1 \leq \mathtt{i} \leq \mathtt{n}$.
3. Reflexivity: For all $\mathtt{a}$ declared in Σ, $\mathtt{a} \preceq \mathtt{a}$.
4. Transitivity: if $\mathtt{a}_1 \preceq \mathtt{a}_2$ and $\mathtt{a}_2 \preceq \mathtt{a}_3$ then $\mathtt{a}_1 \preceq \mathtt{a}_3$.

The first condition, $\vdash_{\preceq}\ \Sigma\ \mathtt{sig}$, implies that the subordination relation is permissive enough that the signature itself is well-formed. The second condition ensures that the types of the indices to a type family are subordinate to that family. The third and fourth ensure that subordination is a pre-order.

We tacitly extend the notation for subordination from constants to arbitrary type families by writing $\mathtt{A}_1 \preceq \mathtt{A}_2$ to mean $|\mathtt{A}_1| \preceq |\mathtt{A}_2|$. We write $\mathtt{a}_1 \not\preceq \mathtt{a}_2$ to mean that $\mathtt{a}_1$ is not subordinate to $\mathtt{a}_2$; we extend this notation to $\mathtt{A}_1 \not\preceq \mathtt{A}_2$ analogously to subordination. We also write $\mathtt{K} \preceq \mathtt{A}$, where $\mathtt{K} = \Pi\,\mathtt{x}_1{:}\mathtt{A}_1 \ldots \Pi\,\mathtt{x}_\mathtt{n}{:}\mathtt{A}_\mathtt{n}.\,\mathtt{type}$, to mean $\mathtt{A}_\mathtt{i} \preceq \mathtt{A}$ for all $1 \leq \mathtt{i} \leq \mathtt{n}$.

Theorem 2.5 assumes that the subordination relation $\mathtt{A} \preceq \mathtt{B}$ is decidable. This assumption is true of any subordination relation: $|\mathtt{A}|$ maps every type family $\mathtt{A}$ to a unique constant $\mathtt{a}$; and subordination on constants $\mathtt{a} \preceq \mathtt{b}$ is decidable because

we require a subordination relation to be presented as a list of pairs. *Theorem 2.11* assumes that $\mathtt{A} \preceq \mathtt{B}$ is preserved by substitution into $\mathtt{A}$ and $\mathtt{B}$. This assumption is true of any subordination relation by the following lemma:

Lemma 2.15 (*Head is Invariant Under Substitution*)
For all $\mathtt{M}_0$, $\mathtt{x}_0$, α_0, and $\mathtt{E}$ in $\{\mathtt{A}, \mathtt{P}\}$, if $[\mathtt{M}_0/\mathtt{x}_0]^e_{\alpha_0}\mathtt{E} = \mathtt{E}'$ then $|\mathtt{E}| = |\mathtt{E}'|$.

2.4.1 Transport of Canonical Forms

In this section, we show that the canonical forms of a type are unchanged by adding or removing canonical forms of other types that are not subordinate to it. We begin by defining the restriction of a context, signature, or subordination relation to a type, which removes all declarations or relationships that are not subordinate to that type.

Definition 2.16 (*Context, Signature, and Subordination Relation Restriction*)

$\boxed{\Gamma|^{\preceq}_{\mathtt{a}} = \Gamma'}$

$$\frac{}{\cdot|^{\preceq}_{\mathtt{a}} = \cdot} \qquad \frac{\Gamma|^{\preceq}_{\mathtt{a}} = \Gamma' \quad \mathtt{A}_2 \preceq \mathtt{a}}{(\Gamma, \mathtt{x}:\mathtt{A}_2)|^{\preceq}_{\mathtt{a}} = \Gamma', \mathtt{x}:\mathtt{A}_2} \qquad \frac{\Gamma|^{\preceq}_{\mathtt{a}} = \Gamma' \quad \mathtt{A}_2 \not\preceq \mathtt{a}}{(\Gamma, \mathtt{x}:\mathtt{A}_2)|^{\preceq}_{\mathtt{a}} = \Gamma'}$$

$\boxed{\Sigma|^{\preceq}_{\mathtt{a}} = \Sigma'}$

$$\frac{}{\cdot|^{\preceq}_{\mathtt{a}} = \cdot} \qquad \frac{\Sigma|^{\preceq}_{\mathtt{a}} = \Sigma' \quad \mathtt{A}_2 \preceq \mathtt{a}}{(\Sigma, \mathtt{c}:\mathtt{A}_2)|^{\preceq}_{\mathtt{a}} = \Sigma', \mathtt{c}:\mathtt{A}_2} \qquad \frac{\Sigma|^{\preceq}_{\mathtt{a}} = \Sigma' \quad \mathtt{A}_2 \not\preceq \mathtt{a}}{(\Sigma, \mathtt{c}:\mathtt{A}_2)|^{\preceq}_{\mathtt{a}} = \Sigma'}$$

$$\frac{\Sigma|^{\preceq}_{\mathtt{a}} = \Sigma' \quad \mathtt{a}' \preceq \mathtt{a}}{(\Sigma, \mathtt{a}':\mathtt{K})|^{\preceq}_{\mathtt{a}} = \Sigma', \mathtt{a}':\mathtt{K}} \qquad \frac{\Sigma|^{\preceq}_{\mathtt{a}} = \Sigma' \quad \mathtt{a}' \not\preceq \mathtt{a}}{(\Sigma, \mathtt{a}':\mathtt{K})|^{\preceq}_{\mathtt{a}} = \Sigma'}$$

Given a subordination relation $\preceq$, we define its restriction to a constant $\mathtt{c}$, written $\preceq|_{\mathtt{c}}$, by

$$\mathtt{a} \preceq|_{\mathtt{c}} \mathtt{b} \quad \text{iff} \quad \mathtt{a} \preceq \mathtt{b} \text{ and } \mathtt{b} \preceq \mathtt{c}.$$

Observe by induction on the structure of Γ that for all Γ, $\preceq$, $\mathtt{a}$, there exists a unique Γ' such that $\Gamma|^{\preceq}_{\mathtt{a}} = \Gamma'$; this justifies using $\Gamma|^{\preceq}_{\mathtt{a}}$ in function notation. A similar result holds for signature restriction. As a convenience, we write $\Gamma|^{\preceq}_{\mathtt{A}}$ for $\Gamma|^{\preceq}_{\mathtt{a}}$ where $|\mathtt{A}| = \mathtt{a}$; we adopt the analogous convention for signature restriction $\Sigma|^{\preceq}_{\mathtt{A}}$ and subordination restriction $\preceq|_{\mathtt{A}}$.

We now prove that canonical forms are invariant under subordination-based context, signature, and subordination relation restriction and extension:

Theorem 2.17 (*Transport of Canonical Forms*)
Assume that $\preceq$ is a subordination relation for Σ.

1. For all $\mathtt{A}$, if $\preceq|_{\mathtt{A}} = \preceq'$ and $\Sigma|^{\preceq}_{\mathtt{A}} = \Sigma'$ then $\vdash_{\preceq'} \Sigma'$ $\mathtt{sig}$.

Assume a type $\mathtt{B}$ and let $\Sigma|^{\preceq}_{\mathtt{B}} = \Sigma'$ and $\preceq|_{\mathtt{B}} = \preceq'$.

2. If $\mathtt{A} \preceq \mathtt{B}$ and $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$ and $\Gamma \vdash_{\Sigma,\preceq} \mathtt{A}\ \mathtt{type}$ and $\Gamma|_{\mathtt{B}}^{\preceq} = \Gamma'$ then $\Gamma \vdash_{\Sigma,\preceq} \mathtt{M} \Leftarrow \mathtt{A}$ iff $\Gamma' \vdash_{\Sigma',\preceq'} \mathtt{M} \Leftarrow \mathtt{A}$. The analogous statement for $\Gamma \vdash_{\Sigma,\preceq} \mathtt{R} \Rightarrow \mathtt{A}$ also holds.
3. If $\mathtt{A} \preceq \mathtt{B}$ and $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$ and $\Gamma|_{\mathtt{B}}^{\preceq} = \Gamma'$ then $\Gamma \vdash_{\Sigma,\preceq} \mathtt{A}\mathtt{type}$ iff $\Gamma' \vdash_{\Sigma',\preceq'} \mathtt{A}\mathtt{type}$. The analogous statement for $\Gamma \vdash_{\Sigma,\preceq} \mathtt{P} \Rightarrow \mathtt{K}$ also holds.
4. If $\mathtt{K} \preceq \mathtt{B}$ and $\vdash_{\Sigma,\preceq} \Gamma\mathtt{ctx}$ and $\Gamma|_{\mathtt{B}}^{\preceq} = \Gamma'$ then $\Gamma \vdash_{\Sigma,\preceq} \mathtt{K}\mathtt{kind}$ iff $\Gamma' \vdash_{\Sigma',\preceq'} \mathtt{K}\mathtt{kind}$.
5. If $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$ and $\Gamma|_{\mathtt{B}}^{\preceq} = \Gamma'$ then $\vdash_{\Sigma',\preceq'} \Gamma'\ \mathtt{ctx}$.

Proof
The "if" directions of parts 2, 3, and 4 are consequences of weakening (*Lemma 2.6*). This leaves the "only if" directions for each of these three parts. The two clauses in part 2 can be proved by mutual induction on the derivations of $\Gamma \vdash_{\Sigma,\preceq} \mathtt{M} \Leftarrow \mathtt{A}$ and $\Gamma \vdash_{\Sigma,\preceq} \mathtt{R} \Rightarrow \mathtt{A}$. Then, to prove part 3, we prove the clause for atomic families by induction on the derivation of $\Gamma \vdash_{\Sigma,\preceq} \mathtt{P} \Rightarrow \mathtt{K}$, and then we prove the clause for types by induction on the derivation of $\Gamma \vdash_{\Sigma,\preceq} \mathtt{A}\ \mathtt{type}$. Next, we prove part 4 by induction on the derivation of $\Gamma \vdash_{\Sigma,\preceq} \mathtt{K}\ \mathtt{kind}$. The proofs of these parts use *Lemma 2.12* and *Lemma 2.15*. These parts also require the following lemma: if $\preceq$ is a subordination relation for Σ and $\Gamma \vdash_{\Sigma,\preceq} \mathtt{P} \Rightarrow \mathtt{K}$ then $\mathtt{K} \preceq \mathtt{P}$; this lemma can be proved by induction on the formation derivation for $\mathtt{P}$.

Next, we prove part 5 by induction on the derivation of $\vdash_{\Sigma,\preceq} \Gamma\ \mathtt{ctx}$. For part 1, we prove that if $\preceq$ is a subordination relation for Σ and $\Sigma|_{\mathtt{A}}^{\preceq} = \Sigma'$ and $\preceq|_{\mathtt{A}} = \preceq'$ then $\vdash_{\preceq'} \Sigma'\ \mathtt{sig}$. The proof is by induction on the restriction derivation; it requires a lemma stating that a subordination relation for a signature $(\Sigma, \mathtt{c}:\mathtt{A})$ is also a subordination relation for Σ (and similarly for $\Sigma, \mathtt{a}:\mathtt{K}$). □

The proof of this theorem uses the fact that the relationship $\mathtt{A}_2 \preceq \mathtt{A}$ holds whenever a type $\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}$ is well-formed. Moreover, it is possible to construct a counterexample to this transport theorem using a type that does not satisfy this condition.

2.4.2 Strongest Subordination Relation

We define the *strongest subordination relation* for a signature Σ, written $\preceq_\Sigma$, to be the intersection of all subordination relations for Σ. If a signature Σ is well-formed without regard to subordination (i.e., it is well-formed in the complete relation), then there exists a unique strongest subordination relation for Σ. Intuitively, the strongest subordination relation establishes as few subordination relationships as possible, subject to the constraint that the signature itself be well-formed. Working with any subordination relation other than the strongest one causes *Theorem 2.17* to produce overapproximate results: extra subordination relationships prevent context and signature restriction from dropping assumptions that are, in fact, irrelevant. For example, if this theorem is applied with the complete subordination relation, the restrictions are the identity and the theorem yields no information.

In the remainder of this paper, we consider only the strongest subordination relation for each signature. By convention, when we instantiate a judgement with an LF signature Σ, we also implicitly take the subordination relation parameter to be $\preceq_\Sigma$. For example, we will write LF[Σ] to mean LF[$\Sigma, \preceq_\Sigma$]. Additionally, we write $\Sigma|_{\mathtt{A}}$ to mean $\Sigma|_{\mathtt{A}}^{\preceq_\Sigma}$ for the strongest subordination relation $\preceq_\Sigma$. We will also

$$\begin{array}{lcl} \tau & ::= & \mathsf{unit} \mid \tau_1 \to \tau_2 \\ \mathsf{e} & ::= & \mathsf{x} \mid \langle\rangle \mid \lambda\,\mathsf{x}{:}\tau.\,\mathsf{e} \mid \mathsf{e}_1\,\mathsf{e}_2 \end{array}$$

$\boxed{\mathcal{X} \vdash \mathsf{e}\ \mathsf{term}}$

$$\frac{}{\mathcal{X}, \mathsf{x}, \mathcal{X}' \vdash \mathsf{x}\ \mathsf{term}}\ \textsc{term\_var} \qquad \frac{}{\mathcal{X} \vdash \langle\rangle\ \mathsf{term}}\ \textsc{term\_empty}$$

$$\frac{\mathcal{X}, \mathsf{x} \vdash \mathsf{e}\ \mathsf{term}}{\mathcal{X} \vdash \lambda\,\mathsf{x}{:}\tau.\,\mathsf{e}\ \mathsf{term}}\ \textsc{term\_lam} \qquad \frac{\mathcal{X} \vdash \mathsf{e}_1\ \mathsf{term} \quad \mathcal{X} \vdash \mathsf{e}_2\ \mathsf{term}}{\mathcal{X} \vdash \mathsf{e}_1\,\mathsf{e}_2\ \mathsf{term}}\ \textsc{term\_app}$$

$\boxed{[\mathsf{e}_0/\mathsf{x}]\mathsf{e} = \mathsf{e}'}$

$$\frac{}{[\mathsf{e}_0/\mathsf{x}]\mathsf{x} = \mathsf{e}_0}\ \textsc{subst\_var\_match} \qquad \frac{\mathsf{y} \# \mathsf{x}}{[\mathsf{e}_0/\mathsf{x}]\mathsf{y} = \mathsf{y}}\ \textsc{subst\_var\_mismatch}$$

$$\frac{}{[\mathsf{e}_0/\mathsf{x}]\langle\rangle = \langle\rangle}\ \textsc{subst\_empty} \qquad \frac{[\mathsf{e}_0/\mathsf{x}]\mathsf{e}_1 = \mathsf{e}_1' \quad [\mathsf{e}_0/\mathsf{x}]\mathsf{e}_2 = \mathsf{e}_2'}{[\mathsf{e}_0/\mathsf{x}]\mathsf{e}_1\,\mathsf{e}_2 = \mathsf{e}_1'\,\mathsf{e}_2'}\ \textsc{subst\_app}$$

$$\frac{\mathsf{x} \# \mathsf{y} \quad \mathsf{y} \# \mathsf{e}_0 \quad [\mathsf{e}_0/\mathsf{x}]\mathsf{e} = \mathsf{e}'}{[\mathsf{e}_0/\mathsf{x}]\lambda\,\mathsf{y}{:}\tau.\,\mathsf{e} = \lambda\,\mathsf{y}{:}\tau.\,\mathsf{e}'}\ \textsc{subst\_lam}$$

Fig. 6. Syntax of the STLC

write $\Gamma|_{\mathtt{A}}$ when the signature and its strongest subordination relation are clear from context.

All of the signatures Σ we consider in this article have the property that

$$(\preceq_\Sigma)|_{\mathsf{a}} = \preceq_{\Sigma'} \text{ where } \Sigma|_{\mathsf{a}}^{\preceq_\Sigma} = \Sigma'.$$

for any family a declared in the signatures. That is, the restriction to a type of the strongest subordination relation for the signature is the strongest subordination relation for the restriction of the signature. We sometimes tacitly pass between these two subordination relations.

3 Mechanizing the Definition of the STLC in LF

3.1 Encoding of Syntax

We now begin mechanizing the simply typed λ-calculus (STLC) in LF. In this section, we encode the language's syntax. For reference, in Figure 6, we present the syntax of the STLC in informal mathematical notation. The metavariable $\mathcal{X}$ ranges over comma-separated lists of distinct variables, which stand for assumptions of $\mathsf{x}\,\mathsf{term}$; the hypothetical judgement $\mathcal{X} \vdash \mathsf{e}\ \mathsf{term}$ is derivable when the free variables of the term e are contained in $\mathcal{X}$. This judgement will be used to state the adequacy theorems below. The judgement $[\mathsf{e}_0/\mathsf{x}]\mathsf{e} = \mathsf{e}'$ defines the standard notion of capture-avoiding substitution; we omit the standard definition of $\mathsf{x} \# \mathsf{e}$.

The LF signature defined in Figure 7 represents the syntax of the STLC. It

$$
\begin{array}{rcl}
\mathtt{tp} & : & \mathtt{type} \\
\mathtt{arrow} & : & \mathtt{tp} \to \mathtt{tp} \to \mathtt{tp} \\
\mathtt{unit} & : & \mathtt{tp} \\
\\
\mathtt{tm} & : & \mathtt{type} \\
\mathtt{empty} & : & \mathtt{tm} \\
\mathtt{app} & : & \mathtt{tm} \to \mathtt{tm} \to \mathtt{tm} \\
\mathtt{lam} & : & \mathtt{tp} \to (\mathtt{tm} \to \mathtt{tm}) \to \mathtt{tm}
\end{array}
$$

Fig. 7. LF Signature for the STLC Syntax

$\boxed{\tau\ \mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}}$

$$\frac{}{\mathsf{unit}\ \mathsf{type} \gg \mathtt{unit} \Leftarrow \mathtt{tp}}\ \text{ENC\_TP\_UNIT}$$

$$\frac{\tau_1\ \mathsf{type} \gg \mathtt{M}_1 \Leftarrow \mathtt{tp} \quad \tau_2\ \mathsf{type} \gg \mathtt{M}_2 \Leftarrow \mathtt{tp}}{\tau_1 \to \tau_2\ \mathsf{type} \gg \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2 \Leftarrow \mathtt{tp}}\ \text{ENC\_TP\_ARROW}$$

$\boxed{\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}}$

$$\frac{}{\cdot\ \mathsf{terms} \gg \cdot\ \mathtt{ctx}}\ \text{ENC\_TERMS\_NIL} \qquad \frac{\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}}{\mathcal{X}, \mathsf{x}\ \mathsf{terms} \gg \Gamma, \mathtt{x} : \mathtt{tm}\ \mathtt{ctx}}\ \text{ENC\_TERMS\_TERM}$$

$\boxed{\mathcal{X} \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}}$

$$\frac{}{\mathcal{X}, \mathsf{x}, \mathcal{X}' \vdash \mathsf{x}\ \mathsf{term} \gg \Gamma, \mathtt{x} : \mathtt{tm}, \Gamma' \vdash \mathtt{x} \Leftarrow \mathtt{tm}}\ \text{ENC\_TM\_VAR}$$

$$\frac{}{\mathcal{X} \vdash \langle\rangle\ \mathsf{term} \gg \Gamma \vdash \mathtt{empty} \Leftarrow \mathtt{tm}}\ \text{ENC\_TM\_EMPTY}$$

$$\frac{\tau\ \mathsf{type} \gg \mathtt{M}_\mathsf{t} \Leftarrow \mathtt{tp} \quad \mathcal{X}, \mathsf{x} \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma, \mathtt{x} : \mathtt{tm} \vdash \mathtt{M}_\mathsf{e} \Leftarrow \mathtt{tm}}{\mathcal{X} \vdash (\lambda\,\mathsf{x}{:}\tau.\,\mathsf{e})\ \mathsf{term} \gg \Gamma \vdash \mathtt{lam}\,\mathtt{M}_\mathsf{t}\,(\lambda\,\mathtt{x}.\,\mathtt{M}_\mathsf{e}) \Leftarrow \mathtt{tm}}\ \text{ENC\_TM\_LAM}$$

$$\frac{\mathcal{X} \vdash \mathsf{e}_1\ \mathsf{term} \gg \Gamma \vdash \mathtt{M}_1 \Leftarrow \mathtt{tm} \quad \mathcal{X} \vdash \mathsf{e}_2\ \mathsf{term} \gg \Gamma \vdash \mathtt{M}_2 \Leftarrow \mathtt{tm}}{\mathcal{X} \vdash \mathsf{e}_1\,\mathsf{e}_2\ \mathsf{term} \gg \Gamma \vdash \mathtt{app}\,\mathtt{M}_1\,\mathtt{M}_2 \Leftarrow \mathtt{tm}}\ \text{ENC\_TM\_APP}$$

Fig. 8. Encoding of STLC Syntax

declares an LF type for each syntactic category (`tp` for τ and `tm` for e) along with constants inhabiting those types with the representations of the language's types and terms. The informal syntax unit has no subexpressions in the informal grammar, so it is represented by an LF constant `unit` of type `tp`; the informal syntax $\tau_1 \to \tau_2$ has two type subexpressions, so it is represented by an LF constant `arrow` of type $\mathtt{tp} \to \mathtt{tp} \to \mathtt{tp}$. The representation of terms uses a technique called *higher-order abstract syntax*: object-language variables are represented by LF variables; with this representation, the framework provides α-conversion and substitution for the object language. To build intuition, consider the following expressions and their intended representations:

Expression		LF Representation
$\mathsf{unit} \to \mathsf{unit}$	$\gg$	$(\mathtt{arrow}\ \mathtt{unit}\ \mathtt{unit})$
$\mathsf{x}\,\mathsf{y}$	$\gg$	$(\mathtt{app}\ \mathsf{x}\ \mathsf{y})$
$\lambda\,\mathsf{x}{:}\mathsf{unit}.\,\mathsf{x}$	$\gg$	$(\mathtt{lam}\ \mathtt{unit}\ (\lambda\,\mathsf{x}.\,\mathsf{x}))$

The second and third examples illustrate higher-order abstract syntax. In the second example, object-language variables are translated to LF variables. The third example illustrates the representation of binding forms: in the informal syntax $\lambda\,\mathsf{x}{:}\tau.\,\mathsf{e}$, the variable x is bound in the body e; in the LF representation, the body is represented by an LF function of type $(\mathtt{tm} \to \mathtt{tm})$ that binds the equivalent variable. This representation strategy is the reason for the higher-order type of the constant `lam`.

The judgements in Figure 8 formally define the encoding of the STLC into LF. We consider both object-language terms and LF terms up to α-equivalence, and by convention all bound variables are chosen fresh. The intended reading of these judgements is as follows:

- $\tau\ \mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$ Encode an object-language type τ to an LF term `M` of LF type `tp` in the empty context.
- $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ Encode an object-language list of variables $\mathcal{X}$ to an LF context declaring each of those variables to have type `tm`.
- $\mathcal{X} \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ Assuming $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$, encode an object-language term `e` with free variables in $\mathcal{X}$ to an LF term `M` of type `tm` in LF context Γ. This judgement has four parameters (`e`, $\mathcal{X}$, `M`, and Γ). The notation is meant to suggest the intended invariant that whenever the judgement $\mathcal{X} \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ is derivable, so are $\mathcal{X} \vdash \mathsf{e}\ \mathsf{term}$ and $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$. This invariant is verified in the adequacy proof below.

The term encoding rule ENC_TM_LAM illustrates higher-order abstract syntax: the encoding of the object language term `e` with free variables $(\mathcal{X}, \mathsf{x})$ is an LF term $\mathtt{M}_{\mathsf{e}}$ with free variables in the context $(\Gamma, \mathsf{x} : \mathtt{tm})$; the variable `x` is then bound in $\lambda\,\mathsf{x}.\,\mathtt{M}_{\mathsf{e}}$, which creates an LF term of type $\mathtt{tm} \to \mathtt{tm}$.

These encoding judgements clarify the fact that the LF representation of the object-language syntax is specified not just by the LF signature, but also by the LF contexts in which that signature is considered. For example, to know that `tm` adequately represents object-language terms with free variables, it is necessary to know not just that the constants inhabiting `tm` are `empty`, `app`, and `lam`, but also that contexts containing LF variables of type `tm` are considered.

The strongest subordination relation for the signature Σ in Figure 7, written $\preceq_\Sigma$, is $\{\mathtt{tp} \preceq \mathtt{tp}, \mathtt{tp} \preceq \mathtt{tm}, \mathtt{tm} \preceq \mathtt{tm}\}$. Intuitively, STLC types appear in the syntax of STLC terms, but STLC terms do not appear in the syntax of STLC types.

3.2 Adequacy of Syntax Encodings

Adequacy is the correctness criterion for an encoding judgement; it establishes that the encoding is an isomorphism between the informal object-language entities and

their LF representation. We break the statement of adequacy into four parts: First, we show that the subjects of the encoding judgement are well-formed, which establishes that the encoding relates the stated object-language entities to LF terms of the appropriate type. Second, we show that the encoding judgement relates every informal entity to a unique LF term; third, we show that the encoding judgement relates every canonical form of the appropriate LF type to a unique informal entity. Because the single encoding judgement is functional in both directions, these functions are mutually inverse, yielding a bijection. Finally, we show that the encoding commutes with substitution, establishing compositionality.

For the remainder of this section, we work in LF[Σ], where Σ stands for the LF signature defined in Figure 7.

3.2.1 Types

For parallelism with later adequacy statements, we use the notation $\tau\,\mathsf{type}$ to mean that τ is a syntactically well-formed type. The adequacy theorem for types is degenerate: because types do not involve binding, no compositionality condition is necessary.

Theorem 3.1 (*Adequacy for Types*)
The encoding relation $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$ defines a bijection between the types τ and the LF terms $\mathtt{M}$ such that $\mathtt{M} \Leftarrow \mathtt{tp}$:

1. If $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$ then $\tau\,\mathsf{type}$ and $\mathtt{M} \Leftarrow \mathtt{tp}$.
2. If $\tau\,\mathsf{type}$, there exists a unique LF term $\mathtt{M}$ such that $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$.
3. If $\mathtt{M} \Leftarrow \mathtt{tp}$ then there exists a unique τ such that $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$.

The third part of this theorem is proved by induction on canonical forms, using a specialized inversion principle for the canonical forms of type $\mathtt{tp}$ in the empty LF context:

Lemma 3.2 (*Inversion of Canonical Forms of Type* $\mathtt{tp}$)
If $\mathcal{D}$ derives $\mathtt{M} \Leftarrow \mathtt{tp}$ then either

- $\mathtt{M} = \mathtt{unit}$, or
- $\mathtt{M} = \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2$, and $\mathtt{M}_1 \Leftarrow \mathtt{tp}$ and $\mathtt{M}_2 \Leftarrow \mathtt{tp}$ were derived as strict subderivations of $\mathcal{D}$.

This lemma permits a proof by induction on the derivation of $\mathtt{M} \Leftarrow \mathtt{tp}$ to consider only these two cases and to appeal to induction on the strict subderivations—it is equivalent to a specialized induction principle for the canonical forms of type $\mathtt{tp}$ in the empty context.

Proof
Characterizing the canonical forms of type $\mathtt{tp}$ requires first characterizing the canonical forms of certain higher types:

1. There is no $\mathtt{R}$ such that $\mathtt{R} \Rightarrow \Pi\,\mathtt{x}_1{:}\mathtt{A}_1 \ldots \Pi\,\mathtt{x}_\mathtt{n}{:}\mathtt{A}_\mathtt{n}.\,\mathtt{tp}$ for $\mathtt{n} \geq 3$.
2. If $\mathtt{R} \Rightarrow \Pi\,\mathtt{x}_1{:}\mathtt{A}_1.\,\Pi\,\mathtt{x}_2{:}\mathtt{A}_2.\,\mathtt{tp}$ then $\mathtt{A}_1 = \mathtt{A}_2 = \mathtt{tp}$ and $\mathtt{R} = \mathtt{arrow}$.

3. If $\mathtt{R} \Rightarrow \Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{tp}$ then $\mathtt{A}_2 = \mathtt{tp}$, $\mathtt{R} = \mathtt{arrow}\,\mathtt{M}_1$, and $\mathtt{M}_1 \Leftarrow \mathtt{tp}$ was derived as a strict subderivation.

To prove the first part, assume there is such a term and then obtain a contradiction by induction on the given derivation. The proofs of the next two parts proceed by case analysis on the derivation, each using the previous part. Each proof uses *Lemma 2.13*. Then the overall lemma is proved by inverting the derivation of $\mathtt{M} \Leftarrow \mathtt{tp}$ and showing, in each case, that $\mathtt{M}$ has the required form. The proof uses *Lemma 2.13* and part 3 above. □

Using this lemma, we prove adequacy:

Proof of Theorem 3.1

1. To show: If $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$ then $\tau\,\mathsf{type}$ and $\mathtt{M} \Leftarrow \mathtt{tp}$. We show one case.
   In the case for ENC_TP_ARROW, assume as the inductive hypothesis that $\tau_1\,\mathsf{type}$, $\tau_2\,\mathsf{type}$, $\mathtt{M}_1 \Leftarrow \mathtt{tp}$, and $\mathtt{M}_2 \Leftarrow \mathtt{tp}$. Then $\tau_1 \to \tau_2\,\mathsf{type}$ and the judgement $\mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2 \Leftarrow \mathtt{tp}$ can be derived using the inductive hypotheses and the following rules: ATOM_TERM_APP, ATOM_TERM_CONST, SUBST_A_PI, SUBST_A_P, SUBST_P_CONST, and CANON_TERM_ATOM.
2. To show: For all τ, there exists a unique term $\mathtt{M}$ such that $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$. The proof is by structural induction on τ. We show one case.
   In the case for $\tau_1 \to \tau_2$, we assume that there exist unique $\mathtt{M}_1$ and $\mathtt{M}_2$ such that $\tau_1\,\mathsf{type} \gg \mathtt{M}_1 \Leftarrow \mathtt{tp}$ and $\tau_2\,\mathsf{type} \gg \mathtt{M}_2 \Leftarrow \mathtt{tp}$. We must show that there exists a unique $\mathtt{M}$ such that $\tau_1 \to \tau_2\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$. To establish existence, take $\mathtt{M}$ to be $\mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2$. Then using ENC_TP_ARROW on the inductive hypotheses proves that $\tau_1 \to \tau_2\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$. To show uniqueness, assume some other $\mathtt{M}'$ such that $\tau_1 \to \tau_2\mathsf{type} \gg \mathtt{M}' \Leftarrow \mathtt{tp}$. By inversion, the only rule that can have applied to $\tau_1 \to \tau_2$ is ENC_TP_ARROW, so $\mathtt{M}' = \mathtt{arrow}\,\mathtt{M}'_1\,\mathtt{M}'_2$ where $\tau_1\,\mathsf{type} \gg \mathtt{M}'_1 \Leftarrow \mathtt{tp}$ and $\tau_2\,\mathsf{type} \gg \mathtt{M}'_2 \Leftarrow \mathtt{tp}$. But then the inductive hypotheses that $\mathtt{M}_1$ and $\mathtt{M}_2$ are unique show that $\mathtt{M}_1 = \mathtt{M}'_1$ and $\mathtt{M}_2 = \mathtt{M}'_2$. Therefore $\mathtt{arrow}\,\mathtt{M}'_1\,\mathtt{M}'_2 = \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2$, establishing uniqueness.
3. To show: For all $\mathtt{M}$ such that $\mathtt{M} \Leftarrow \mathtt{tp}$, there exists a unique τ such that $\tau\,\mathsf{type} \gg \mathtt{M} \Leftarrow \mathtt{tp}$. The proof is by induction on the derivation of $\mathtt{M} \Leftarrow \mathtt{tp}$.
   Lemma 3.2 gives two cases to consider; we show the case for $\mathtt{M} = \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2$, where $\mathtt{M}_1 \Leftarrow \mathtt{tp}$ and $\mathtt{M}_2 \Leftarrow \mathtt{tp}$ were derived as strict subderivations. To show: there exists a unique τ such that $\tau\mathsf{type} \gg \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2 \Leftarrow \mathtt{tp}$. By the inductive hypothesis applied to the subderivations, there exist unique τ_1 and τ_2 such that $\tau_1\,\mathsf{type} \gg \mathtt{M}_1 \Leftarrow \mathtt{tp}$ and $\tau_2\,\mathsf{type} \gg \mathtt{M}_2 \Leftarrow \mathtt{tp}$. To establish existence, take $\tau = \tau_1 \to \tau_2$; then ENC_TP_ARROW applied to the derivations from the inductive hypothesis shows that $\tau\,\mathsf{type} \gg \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2 \Leftarrow \mathtt{tp}$. To establish uniqueness, assume some other τ' such that $\tau'\,\mathsf{type} \gg \mathtt{arrow}\,\mathtt{M}_1\,\mathtt{M}_2 \Leftarrow \mathtt{tp}$. By inversion, only the rule ENC_TP_ARROW could have applied, so $\tau' = \tau'_1 \to \tau'_2$ where $\tau'_1\,\mathsf{type} \gg \mathtt{M}_1 \Leftarrow \mathtt{tp}$ and $\tau'_2\,\mathsf{type} \gg \mathtt{M}_2 \Leftarrow \mathtt{tp}$. The inductive hypothesis that τ_1 and τ_2 are unique shows that $\tau_1 = \tau'_1$ and $\tau_2 = \tau'_2$, so $\tau_1 \to \tau_2 = \tau'_1 \to \tau'_2$, as we needed to show.

□

3.2.2 Terms

The following lemma, analogous to the one for tp above, gives an inversion principle for the canonical forms of type tm:

Lemma 3.3 (*Inversion of Canonical Forms of Type* tm)
If $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ and $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ then one of the following holds:

- $\mathtt{M} = \mathtt{x}$ and $\Gamma = \Gamma_1, \mathtt{x} : \mathtt{tm}, \Gamma_2$.
- $\mathtt{M} = \mathtt{empty}$.
- $\mathtt{M} = \mathtt{lam}\,\mathtt{M_t}\,\lambda\mathtt{x}.\mathtt{M_e}$, where $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$ and $\Gamma, \mathtt{x} : \mathtt{tm} \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$ were derived as strict subderivations.
- $\mathtt{M} = \mathtt{app}\,\mathtt{M_1}\,\mathtt{M_2}$, where and $\Gamma \vdash \mathtt{M_1} \Leftarrow \mathtt{tm}$ and $\Gamma \vdash \mathtt{M_2} \Leftarrow \mathtt{tm}$ were derived as strict subderivations.

This lemma states that an LF term of type tm in a context containing variables of type tm is either a variable or a constant applied to arguments which may contain those variables—i.e., it states that the LF term is parametric in the variables in the context. If, hypothetically, LF had an unrestricted case-analysis construct for analyzing terms of type tm, this lemma would not be true: a case-analysis of a variable would be an "exotic" canonical form of type tm, violating adequacy. Extensions of LF with other types must take care to preserve this property; in Concurrent LF, certain connectives are confined to a monad so that they do not interfere with this style of higher-order representation (Watkins *et al.*, 2002).

Because types appear in the syntax of terms, adequacy for terms requires adequacy for types. However, types are adequately represented in the empty LF context, whereas the judgements in Figure 8 encode terms in LF contexts of the form $\mathtt{x_1} : \mathtt{tm}, \ldots, \mathtt{x_n} : \mathtt{tm}$. Thus, *Theorem 3.1* as stated in the previous section is not strong enough, as adequacy for terms requires that types remain adequate in these extended contexts. This requirement could fail—for example, if term contexts were of the form $\mathtt{x} : \mathtt{tm}, \ldots, \mathtt{u} : \mathtt{tp}, \ldots$, these contexts would induce additional canonical forms of type tp that have no informal counterpart. Consequently, it is necessary to prove a lemma showing that types remain adequate in the contexts Γ such that $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$. This lemma is our first application of the transport of canonical forms theorem, *Theorem 2.17*.

Lemma 3.4 (*Transport of Adequacy for Terms*)
1. If $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then $\Gamma\ \mathtt{ctx}$.
2. If $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then $\cdot \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$ iff $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$.

Proof
The proof of the first part is a straightforward induction on the premise. The second part uses transport of canonical forms (*Theorem 2.17*). First, one application of *Theorem 2.17* shows that $\cdot \vdash_\Sigma \mathtt{M} \Leftarrow \mathtt{tp}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{tp}}} \mathtt{M} \Leftarrow \mathtt{tp}$. Next, because $\mathtt{tm} \not\preceq \mathtt{tp}$, a simple induction shows that if $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then $\Gamma|^{\prec\Sigma}_{\mathtt{tp}} = \cdot$. Then the proof is direct using *Theorem 2.17* and the first part. □

We now prove the main adequacy result. The judgement $\mathcal{X} \vdash \mathsf{e}\ \mathsf{term}$ is used here to state the invariant on the encoding. When we claim uniqueness for an object-language or LF entity that involves binding, we mean uniqueness up to α-conversion.

Theorem 3.5 (*Adequacy for Terms*)

1. If $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ and $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ then $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term}$ and $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$.
2. If $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term}$ and $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then there exists a unique LF term $\mathtt{M}$ such that $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$.
3. If $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ and $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then there exists a unique $\mathtt{e}$ such that $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$.

Proof

1. By rule induction on the second premise. The case for ENC_TM_LAM is the only one in which the conclusion does not follow immediately from the inductive hypotheses. In this case, by the inductive hypothesis, $\mathcal{X}, \mathtt{x} \vdash \mathtt{e}\ \mathsf{term}$ and $\Gamma, \mathtt{x} : \mathtt{tm} \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$. Then the first conclusion is immediate by TERM_LAM. To derive the second conclusion, we also need to know that $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$. By *Theorem 3.1* applied to the premise of the rule, $\cdot \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$. Then by *Lemma 3.4*, $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$.
2. By rule induction on $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term}$, we show that if $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then there exists a unique $\mathtt{M}$ such that $\mathcal{X} \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$. We give the cases that involve binding:
   - Case for TERM_VAR. To show: if $\mathcal{X}_1, \mathtt{x}, \mathcal{X}_2\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ then there exists a unique LF term $\mathtt{M}$ such that $\mathcal{X} \vdash \mathtt{x}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$. By inversion, Γ must have the form $\Gamma_1, \mathtt{x} : \mathtt{tm}, \Gamma_2$. To establish existence, use ENC_TM_VAR to derive $\mathcal{X}_1, \mathtt{x}, \mathcal{X}_2 \vdash \mathtt{x}\ \mathsf{term} \gg \Gamma \vdash \mathtt{x} \Leftarrow \mathtt{tm}$. To establish uniqueness, assume some other $\mathtt{M'}$ such that $\mathcal{X}_1, \mathtt{x}, \mathcal{X}_2 \vdash \mathtt{x}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M'} \Leftarrow \mathtt{tm}$; inversion on this derivation gives the result because only ENC_TM_VAR can have applied.
   - Case for TERM_LAM. Assume a derivation of $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$. Using the rule ENC_TERMS_TERM, we can derive $(\mathcal{X}, \mathtt{x})\ \mathsf{terms} \gg (\Gamma, \mathtt{x} : \mathtt{tm})\ \mathtt{ctx}$. Then we can appeal to the inductive hypothesis to conclude that there exists a unique canonical form $\mathtt{M_e}$ such that $\mathcal{X}, \mathtt{x} \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma, \mathtt{x} : \mathtt{tm} \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$. By *Theorem 3.1*, there exists a unique $\mathtt{M_t}$ such that $\tau\ \mathsf{type} \gg \mathtt{M_t} \Leftarrow \mathtt{tp}$. To show existence, take $\mathtt{M}$ to be $\mathtt{lam}\ \mathtt{M_t}\ \lambda \mathtt{x}.\mathtt{M_e}$; then we can derive the desired property by applying ENC_TM_LAM to the above encoding derivations. Uniqueness is immediate by inversion and the results above.
3. By induction on the derivation of $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{tm}$. *Lemma 3.3* gives four cases; we show those involving binding:
   - $\mathtt{M} = \mathtt{x}$ and $\Gamma = (\Gamma_1, \mathtt{x} : \mathtt{tm}, \Gamma_2)$.
     By inversion on the derivation of $\mathcal{X}\ \mathsf{terms} \gg (\Gamma_1, \mathtt{x} : \mathtt{tm}, \Gamma_2)\ \mathtt{ctx}$, $\mathcal{X}$ is $\mathcal{X}_1, \mathtt{x}, \mathcal{X}_2$. Thus, we can take $\mathtt{e}$ to be $\mathtt{x}$, which has the desired property by ENC_TM_VAR. Uniqueness is immediate by inversion.
   - $\mathtt{M} = \mathtt{lam}\ \mathtt{M_t}\ \lambda \mathtt{x}.\mathtt{M_e}$, where $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$ and $\Gamma, \mathtt{x} : \mathtt{tm} \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$ were derived as subderivations.
     We would like to appeal to *Theorem 3.1* on the derivation of $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$ to conclude that there exists a unique τ such that $\tau\ \mathsf{type} \gg \mathtt{M_t} \Leftarrow \mathtt{tp}$. Unfortunately, *Theorem 3.1* is only stated for typing derivations in the

empty context. Thus, we first apply *Lemma 3.4* to conclude $\cdot \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$; then we can appeal to *Theorem 3.1*.
Next, using the assumed derivation of $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$, we can create a derivation of $(\mathcal{X}, \mathtt{x})\ \mathsf{terms} \gg (\Gamma, \mathtt{x}:\mathtt{tm})\ \mathtt{ctx}$ by ENC_TERMS_TERM. Then, by the inductive hypothesis on the second subderivation, there exists a unique $\mathtt{e}'$ such that $\mathcal{X}, \mathtt{x} \vdash \mathtt{e}'\ \mathsf{term} \gg \Gamma, \mathtt{x}:\mathtt{tm} \vdash \mathtt{M}'_\mathtt{e} \Leftarrow \mathtt{tm}$. To show existence, we take $\mathtt{e}$ to be $\lambda\, \mathtt{x}{:}\tau.\, \mathtt{e}'$; ENC_TM_LAM applied to the derivations produced above shows that $\mathcal{X} \vdash \lambda\, \mathtt{x}{:}\tau.\, \mathtt{e}'\ \mathsf{term} \gg \Gamma \vdash \mathtt{lam}\, \mathtt{M_t}\, \lambda\, \mathtt{x}.\, \mathtt{M}'_\mathtt{e} \Leftarrow \mathtt{tm}$. Uniqueness is immediate by inversion and the uniqueness of $\mathtt{e}'$ and τ.

□

Next, we prove compositionality. The proof requires a lemma stating that a property similar to weakening holds for the encoding:

Lemma 3.6 (*Weakening of the Encoding*)
Assume $\mathcal{X}, \mathcal{X}'\ \mathsf{terms} \gg \Gamma, \Gamma'\ \mathtt{ctx}$ and $\mathcal{X}, \mathtt{x}, \mathcal{X}'\ \mathsf{terms} \gg \Gamma, \mathtt{x}:\mathtt{tm}, \Gamma'\ \mathtt{ctx}$. Then if $\mathcal{X}, \mathcal{X}' \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma, \Gamma' \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ then $\mathcal{X}, \mathtt{x}, \mathcal{X}' \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma, \mathtt{x}:\mathtt{tm}, \Gamma' \vdash \mathtt{M} \Leftarrow \mathtt{tm}$.

Exchange and contraction also hold, but we do not require them in this development. Compositionality is essentially a substitution principle:

Theorem 3.7 (*Compositionality for Terms*)
Assume $\mathcal{X}\ \mathsf{terms} \gg \Gamma\ \mathtt{ctx}$ and $\mathcal{X} \vdash \mathtt{e_2}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M}_2 \Leftarrow \mathtt{tm}$.
If $\mathcal{X}, \mathtt{x}, \mathcal{X}'\ \mathsf{terms} \gg \Gamma, \mathtt{x}:\mathtt{tm}, \Gamma'\ \mathtt{ctx}$ and $\mathcal{X}, \mathtt{x}, \mathcal{X}' \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma, \mathtt{x}:\mathtt{tm}, \Gamma' \vdash \mathtt{M} \Leftarrow \mathtt{tm}$ and $[\mathtt{e_2}/\mathtt{x}]\mathtt{e} = \mathtt{e}'$ and $[\mathtt{M}_2/\mathtt{x}]^m_\mathtt{tm}\mathtt{M} = \mathtt{M}'$ then $\mathcal{X}, \mathcal{X}' \vdash \mathtt{e}'\ \mathsf{term} \gg \Gamma, \Gamma' \vdash \mathtt{M}' \Leftarrow \mathtt{tm}$.

Proof
Using *Lemma 3.6*, *Theorem 3.1*, and *Lemma 2.8*, we prove this theorem by induction on the derivation of $\mathcal{X}, \mathtt{x}, \mathcal{X}' \vdash \mathtt{e}\ \mathsf{term} \gg \Gamma, \mathtt{x}:\mathtt{tm}, \Gamma' \vdash \mathtt{M} \Leftarrow \mathtt{tm}$. This theorem assumes the hereditary substitution $[\mathtt{M}_2/\mathtt{x}]^m_\mathtt{tm}\mathtt{M} = \mathtt{M}'$ only to make the inductive argument more convenient; applying *Lemma 3.4*, *Theorem 3.5*, and *Theorem 2.11* to the other assumptions shows that the substitution must exist. □

3.2.3 Discussion

Transport of adequacy lemmas like *Lemma 3.4* demonstrate the convenience of the general subordination-based transport of canonical forms theorem (*Theorem 2.17*). Absent this general theorem, we could prove each individual transport of adequacy lemma inductively. Alternatively, we could state and prove every adequacy theorem for the largest context necessary for the encoding of the entire language. Fortunately, subordination saves us from the tedium of the first alternative and the inherent non-modularity of the second, which requires knowing, in advance, all future uses of any piece of the object language.

3.3 Encoding of Judgements

The judgements in Figure 9 define the static and dynamic semantics of the STLC in informal notation. In the type system, we use γ to notate object-language contexts

$\boxed{\gamma \vdash \mathsf{e} : \tau}$

$$\frac{}{\gamma \vdash \langle\rangle : \mathsf{unit}}\ \text{OF\_EMPTY} \qquad \frac{}{\gamma, \mathsf{x}:\tau, \gamma' \vdash \mathsf{x} : \tau}\ \text{OF\_VAR}$$

$$\frac{\gamma, \mathsf{x}:\tau_2 \vdash \mathsf{e} : \tau}{\gamma \vdash \lambda\, \mathsf{x}{:}\tau_2.\, \mathsf{e} : \tau_2 \to \tau}\ \text{OF\_LAM} \qquad \frac{\gamma \vdash \mathsf{e}_1 : \tau_2 \to \tau \quad \gamma \vdash \mathsf{e}_2 : \tau_2}{\gamma \vdash \mathsf{e}_1\ \mathsf{e}_2 : \tau}\ \text{OF\_APP}$$

$\boxed{\mathsf{e}\ \mathsf{value}}$

$$\frac{}{\langle\rangle\ \mathsf{value}}\ \text{VALUE\_EMPTY} \qquad \frac{}{\lambda\, \mathsf{x}{:}\tau.\, \mathsf{e}\ \mathsf{value}}\ \text{VALUE\_LAM}$$

$\boxed{\mathsf{e} \mapsto \mathsf{e}'}$

$$\frac{\mathsf{e}_1 \mapsto \mathsf{e}_1'}{\mathsf{e}_1\ \mathsf{e}_2 \mapsto \mathsf{e}_1'\ \mathsf{e}_2}\ \text{STEP\_APP\_1} \qquad \frac{\mathsf{e}_1\ \mathsf{value} \quad \mathsf{e}_2 \mapsto \mathsf{e}_2'}{\mathsf{e}_1\ \mathsf{e}_2 \mapsto \mathsf{e}_1\ \mathsf{e}_2'}\ \text{STEP\_APP\_2}$$

$$\frac{\mathsf{e}_2\ \mathsf{value} \quad [\mathsf{e}_2/\mathsf{x}]\mathsf{e} = \mathsf{e}'}{(\lambda\, \mathsf{x}{:}\tau_2.\, \mathsf{e})\ \mathsf{e}_2 \mapsto \mathsf{e}'}\ \text{STEP\_APP\_BETA}$$

Fig. 9. Semantics of the STLC

containing assumptions of the form $\mathsf{x}_i : \tau_i$; such a context is well-formed when all variables in it are distinct. Whenever we write γ, we tacitly presuppose that this context is well-formed. The dynamic semantics are a standard call-by-value structural operational semantics on closed terms. The following property of the STLC is necessary for the adequacy proofs below:

Lemma 3.8 (*Uniqueness of Substitution Derivations*)
If $\mathcal{D}$ and $\mathcal{D}'$ both derive $[\mathsf{e}_2/\mathsf{x}]\mathsf{e} = \mathsf{e}'$ then $\mathcal{D} = \mathcal{D}'$.

The LF signature in Figure 10 extends the signature from Figure 7 to represent these judgements. The representation of the static and dynamic semantics is guided by the *judgements-as-types* principle: an object-language judgement is represented by a family of LF types indexed by the subjects of the judgement; derivations of a judgement are represented as LF terms inhabiting this type. Such a representation is adequate iff there is an isomorphism between the informal derivations of the judgement and the canonical forms of the associated type family.

For instance, the operational semantics judgement $\mathsf{e} \mapsto \mathsf{e}'$, whose subject is two terms, is represented by the LF type family $\mathtt{step} : \mathtt{tm} \to \mathtt{tm} \to \mathtt{type}$. Each constant with head $\mathtt{step}$ corresponds to the object-language inference rule with the same name. A derivation of the judgement $\mathsf{e} \mapsto \mathsf{e}'$ is represented by an LF term of type $\mathtt{step}\ \mathtt{M}_\mathsf{e}\ \mathtt{M}'_\mathsf{e}$, where $\mathtt{M}_\mathsf{e}$ and $\mathtt{M}'_\mathsf{e}$ are the encodings of e and e'. For example, assuming terms $\mathtt{M}_1$, $\mathtt{M}_2$, and $\mathtt{M}'_1$ of LF type $\mathtt{tm}$ and an LF term $\mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M}_1\ \mathtt{M}'_1$, the LF term $(\mathtt{step\_app\_1}\ \mathtt{M}_1\ \mathtt{M}_2\ \mathtt{M}'_1\ \mathtt{M})$ represents a derivation of $\mathtt{step}\ (\mathtt{app}\ \mathtt{M}_1\ \mathtt{M}_2)\ (\mathtt{app}\ \mathtt{M}'_1\ \mathtt{M}_2)$.

The object-language hypothetical judgement $\gamma \vdash \mathsf{e} : \tau$ is represented by the LF type family $\mathtt{of}$, which is indexed by the LF types $\mathtt{tm}$ and $\mathtt{tp}$. One might expect the family to also be indexed by the encoding of a context γ. However, rather

$$
\begin{array}{rcl}
\texttt{of} & : & \texttt{tm} \to \texttt{tp} \to \texttt{type} \\
\texttt{of\_empty} & : & \texttt{of}\ \texttt{empty}\ \texttt{unit} \\
\texttt{of\_app} & : & \Pi\, \texttt{E}_1, \texttt{E}_2{:}\texttt{tm}.\, \Pi\, \texttt{T}_2, \texttt{T}{:}\texttt{tp}. \\
 & & (\texttt{of}\ \texttt{E}_1\ (\texttt{arrow}\ \texttt{T}_2\ \texttt{T})) \to (\texttt{of}\ \texttt{E}_2\ \texttt{T}_2) \to \texttt{of}\ (\texttt{app}\ \texttt{E}_1\ \texttt{E}_2)\ \texttt{T} \\
\texttt{of\_lam} & : & \Pi\, \texttt{T}_2, \texttt{T}{:}\texttt{tp}.\, \Pi\, \texttt{E}{:}\texttt{tm} \to \texttt{tm}. \\
 & & (\Pi\, \texttt{x}{:}\texttt{tm}.\, (\texttt{of}\ \texttt{x}\ \texttt{T}_2) \to (\texttt{of}\ (\texttt{E}\ \texttt{x})\ \texttt{T})) \to \texttt{of}\ (\texttt{lam}\ \texttt{T}_2\ \lambda\, \texttt{x}.\, \texttt{E}\ \texttt{x})\ (\texttt{arrow}\ \texttt{T}_2\ \texttt{T}) \\
 & & \\
\texttt{value} & : & \texttt{tm} \to \texttt{type} \\
\texttt{value\_empty} & : & \texttt{value}\ \texttt{empty} \\
\texttt{value\_lam} & : & \Pi\, \texttt{T}{:}\texttt{tp}.\, \Pi\, \texttt{E}{:}\texttt{tm} \to \texttt{tm}.\, \texttt{value}\ (\texttt{lam}\ \texttt{T}\ \lambda\, \texttt{x}.\, \texttt{E}\ \texttt{x}) \\
 & & \\
\texttt{step} & : & \texttt{tm} \to \texttt{tm} \to \texttt{type} \\
\texttt{step\_app\_1} & : & \Pi\, \texttt{E}_1, \texttt{E}_2, \texttt{E}_1'{:}\texttt{tm}.\, (\texttt{step}\ \texttt{E}_1\ \texttt{E}_1') \to \texttt{step}\ (\texttt{app}\ \texttt{E}_1\ \texttt{E}_2)\ (\texttt{app}\ \texttt{E}_1'\ \texttt{E}_2) \\
\texttt{step\_app\_2} & : & \Pi\, \texttt{E}_1, \texttt{E}_2, \texttt{E}_2'{:}\texttt{tm}. \\
 & & (\texttt{value}\ \texttt{E}_1) \to (\texttt{step}\ \texttt{E}_2\ \texttt{E}_2') \to \texttt{step}\ (\texttt{app}\ \texttt{E}_1\ \texttt{E}_2)\ (\texttt{app}\ \texttt{E}_1\ \texttt{E}_2') \\
\texttt{step\_app\_beta} & : & \Pi\, \texttt{E}_2{:}\texttt{tm}.\, \Pi\, \texttt{E}{:}\texttt{tm} \to \texttt{tm}.\, \Pi\, \texttt{T}_2{:}\texttt{tp}. \\
 & & (\texttt{value}\ \texttt{E}_2) \to \texttt{step}\ (\texttt{app}\ (\texttt{lam}\ \texttt{T}_2\ \lambda\, \texttt{x}.\, \texttt{E}\ \texttt{x})\ \texttt{E}_2)\ (\texttt{E}\ \texttt{E}_2)
\end{array}
$$

Fig. 10. LF Signature for the STLC Judgements

than representing the context as an explicit argument, we represent the object-language hypotheses as LF hypotheses, identifying the object-language hypothetical judgement with a hypothetical judgement in LF. Specifically, an object-language context

$$\mathtt{x}_1 : \tau_1, \ldots, \mathtt{x}_n : \tau_n$$

is represented by an LF context

$$\mathtt{x}_1 : \mathtt{tm}, \mathtt{dx}_1 : \mathtt{of}\ \mathtt{x}_1\ \mathtt{T}_1, \ldots, \mathtt{x}_n : \mathtt{tm}, \mathtt{dx}_n : \mathtt{of}\ \mathtt{x}_n\ \mathtt{T}_n$$

where each $\mathtt{dx}_i$ is a fresh variable that is distinct from all $\mathtt{x}_i$ and from the other $\mathtt{dx}_j$. Each object-language hypothesis $\mathtt{x}_i : \tau_i$ is represented by an LF variable $\mathtt{dx}_i$. Correspondingly, context extension is represented by LF terms of higher type. Consider the object-language rule OF_LAM:

$$\frac{\gamma, \mathtt{x} : \tau_2 \vdash \mathtt{e} : \tau}{\gamma \vdash \lambda\, \mathtt{x}{:}\tau_2.\, \mathtt{e} : \tau_2 \to \tau}\ \text{OF\_LAM}$$

In the premise of the rule, the context is extended with the assumption $\mathtt{x} : \tau_2$ for a fresh variable $\mathtt{x}$. This rule is represented by the LF constant `of_lam`. The premise of `of_lam` is represented by an LF term of type of the following dependent function type:

$$\Pi\, \mathtt{x}{:}\mathtt{tm}.\, (\mathtt{of}\ \mathtt{x}\ \mathtt{T}_2) \to (\mathtt{of}\ (\mathtt{E}\ \mathtt{x})\ \mathtt{T}).$$

$\boxed{\mathsf{vars}\,\gamma\ \mathcal{X}}$

$$\frac{}{\mathsf{vars}\ \cdot\ \cdot} \qquad \frac{\mathsf{vars}\,\gamma\ \mathcal{X}}{\mathsf{vars}\,(\gamma, \mathsf{x}:\tau)\ (\mathcal{X}, \mathsf{x})}$$

$\boxed{\gamma\,\mathtt{ctx} \gg \Gamma\,\mathtt{ctx}}$

$$\frac{}{\cdot\,\mathtt{ctx} \gg \cdot\,\mathtt{ctx}}\ \text{ENC\_CTX\_EMPTY} \qquad \frac{\gamma\,\mathtt{ctx} \gg \Gamma\,\mathtt{ctx} \quad \tau\,\mathsf{type} \gg \mathtt{M_t} \Leftarrow \mathtt{tp}}{\gamma, \mathsf{x}:\tau\,\mathtt{ctx} \gg \Gamma, \mathtt{x}:\mathtt{tm}, \mathtt{dx}:\mathtt{of}\,\mathtt{x}\,\mathtt{M_t}\,\mathtt{ctx}}\ \text{ENC\_CTX\_CONS}$$

$\boxed{\mathcal{D} :: \gamma \vdash \mathsf{e}:\tau \ \gg\ \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\,\mathtt{M_e}\,\mathtt{M_t}}$

$$\frac{}{\gamma, \mathsf{x}:\tau, \gamma' \vdash \mathsf{x}:\tau \gg \Gamma, \mathtt{x}:\mathtt{tm}, \mathtt{dx}:\mathtt{of}\,\mathtt{x}\,\mathtt{M_t}, \Gamma' \vdash \mathtt{dx} \Leftarrow \mathtt{of}\,\mathtt{x}\,\mathtt{M_t}}\ \text{ENC\_OF\_VAR}$$

$$\frac{}{\gamma \vdash \mathsf{empty}:\mathsf{unit} \gg \Gamma \vdash \mathtt{of\_empty} \Leftarrow \mathtt{of}\,\mathsf{empty}\,\mathsf{unit}}\ \text{ENC\_OF\_EMPTY}$$

$$\frac{\mathcal{D}_1 :: \gamma \vdash \mathsf{e}_1:\tau_2 \to \tau \ \gg\ \Gamma \vdash \mathtt{M}_1 \Leftarrow \mathtt{of}\,\mathtt{M_{e1}}\,(\mathtt{arrow}\,\mathtt{M_{t2}}\,\mathtt{M_t}) \quad \mathcal{D}_2 :: \gamma \vdash \mathsf{e}_2:\tau_2 \ \gg\ \Gamma \vdash \mathtt{M}_2 \Leftarrow \mathtt{of}\,\mathtt{M_{e2}}\,\mathtt{M_{t2}}}{\dfrac{\mathcal{D}_1 :: \gamma \vdash \mathsf{e}_1:\tau_2 \to \tau \quad \mathcal{D}_2 :: \gamma \vdash \mathsf{e}_2:\tau_2}{\gamma \vdash \mathsf{e}_1\,\mathsf{e}_2:\tau} \gg \Gamma \vdash \mathtt{of\_app}\,\mathtt{M_{e1}}\,\mathtt{M_{e2}}\,\mathtt{M_{t2}}\,\mathtt{M_t}\,\mathtt{M_1}\,\mathtt{M_2} \Leftarrow \mathtt{of}\,(\mathtt{app}\,\mathtt{M_{e1}}\,\mathtt{M_{e2}})\,\mathtt{M_t}}\ \text{ENC\_OF\_APP}$$

$$\frac{\tau_2\,\mathsf{type} \gg \mathtt{M_{t2}} \Leftarrow \mathtt{tp} \quad \mathcal{D}_1 :: \gamma, \mathsf{x}:\tau_2 \vdash \mathsf{e}:\tau \ \gg\ \Gamma, \mathtt{x}:\mathtt{tm}, \mathtt{dx}:\mathtt{of}\,\mathtt{x}\,\mathtt{M_{t2}} \vdash \mathtt{M}_2 \Leftarrow \mathtt{of}\,\mathtt{M_e}\,\mathtt{M_t}}{\dfrac{\mathcal{D}_1 :: \gamma, \mathsf{x}:\tau_2 \vdash \mathsf{e}:\tau}{\gamma \vdash \lambda\,\mathsf{x}{:}\tau_2.\,\mathsf{e}:\tau_2 \to \tau} \gg \Gamma \vdash \mathtt{of\_lam}\,\mathtt{M_{t2}}\,\mathtt{M_t}\,(\lambda\,\mathtt{x}.\,\mathtt{M_e})\,(\lambda\,\mathtt{x}.\,\lambda\,\mathtt{dx}.\,\mathtt{M}_2) \Leftarrow \mathtt{of}\,(\mathtt{lam}\,\mathtt{M_t}\,(\lambda\,\mathtt{x}.\,\mathtt{M_e}))\,(\mathtt{arrow}\,\mathtt{M_{t2}}\,\mathtt{M_t})}\ \text{ENC\_OF\_LAM}$$

Fig. 11. Encoding of STLC Static Semantics

$\boxed{\mathcal{D} :: \mathsf{e}\ \mathsf{value}\ \gg\ \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}}$

$$\frac{}{\langle\rangle\ \mathsf{value} \gg\ \mathtt{value\_empty} \Leftarrow \mathtt{value}\ \mathtt{empty}}\ \text{ENC\_VALUE\_EMPTY}$$

$$\frac{\cdot \vdash (\lambda\, \mathsf{x}{:}\tau.\, \mathsf{e})\ \mathsf{term} \gg\ \cdot \vdash \mathtt{lam}\ \mathtt{M_t}\ (\lambda\, \mathtt{x}.\, \mathtt{M_e}) \Leftarrow \mathtt{tm}}{\lambda\, \mathsf{x}{:}\tau.\, \mathsf{e}\ \mathsf{value} \gg\ \mathtt{value\_lam}\ \mathtt{M_t}\ (\lambda\, \mathtt{x}.\, \mathtt{M_e}) \Leftarrow \mathtt{value}\ (\mathtt{lam}\ \mathtt{M_t}\ (\lambda\, \mathtt{x}.\, \mathtt{M_e}))}\ \text{ENC\_VALUE\_LAM}$$

$\boxed{\mathcal{D} :: \mathsf{e} \mapsto \mathsf{e}'\ \gg\ \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}}$

$$\frac{\cdot \vdash \mathsf{e_2}\ \mathsf{term} \gg\ \cdot \vdash \mathtt{M_{e2}} \Leftarrow \mathtt{tm} \quad \mathcal{D}_1 :: \mathsf{e}_1 \mapsto \mathsf{e}'_1 \ \gg\ \mathtt{M_1} \Leftarrow \mathtt{step}\ \mathtt{M_{e1}}\ \mathtt{M'_{e1}}}{\dfrac{\mathcal{D}_1 :: \mathsf{e}_1 \mapsto \mathsf{e}'_1}{\mathsf{e}_1\ \mathsf{e}_2 \mapsto \mathsf{e}'_1\ \mathsf{e}_2 \ \gg\ \mathtt{step\_app\_1}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}}\ \mathtt{M'_{e1}}\ \mathtt{M_1} \Leftarrow \mathtt{step}\ (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}})\ (\mathtt{app}\ \mathtt{M'_{e1}}\ \mathtt{M_{e2}})}}\ \text{ENC\_STEP\_APP\_1}$$

$$\frac{\mathcal{D}_1 :: \mathsf{e}_1\ \mathsf{value} \ \gg\ \mathtt{M_1} \Leftarrow \mathtt{value}\ \mathtt{M_{e1}} \quad \mathcal{D}_2 :: \mathsf{e}_2 \mapsto \mathsf{e}'_2 \ \gg\ \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_{e2}}\ \mathtt{M'_{e2}}}{\dfrac{\mathcal{D}_1 :: \mathsf{e}_1\ \mathsf{value} \quad \mathcal{D}_2 :: \mathsf{e}_2 \mapsto \mathsf{e}'_2}{\mathsf{e}_1\ \mathsf{e}_2 \mapsto \mathsf{e}_1\ \mathsf{e}'_2 \qquad \gg\ \mathtt{step\_app\_2}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}}\ \mathtt{M'_{e2}}\ \mathtt{M_1}\ \mathtt{M_2} \Leftarrow \mathtt{step}\ (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}})\ (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M'_{e2}})}}\ \text{ENC\_STEP\_APP\_2}$$

$$\frac{\cdot \vdash (\lambda\, \mathsf{x}{:}\tau_2.\, \mathsf{e})\ \mathsf{term} \gg\ \cdot \vdash \mathtt{lam}\ \mathtt{M_{t2}}\ (\lambda\, \mathtt{x}.\, \mathtt{M_e}) \Leftarrow \mathtt{tm} \quad \mathcal{D}_2 :: \mathsf{e}_2\ \mathsf{value} \ \gg\ \mathtt{M_2} \Leftarrow \mathtt{value}\ \mathtt{M_{e2}} \quad [\mathtt{M_{e2}}/\mathtt{x}]^m_{\mathtt{tm}}\mathtt{M_e} = \mathtt{M'_e}}{\dfrac{\mathcal{D}_2 :: \mathsf{e}_2\ \mathsf{value} \quad [\mathsf{e}_2/\mathsf{x}]\mathsf{e} = \mathsf{e}'}{(\lambda\, \mathsf{x}{:}\tau_2.\, \mathsf{e})\ \mathsf{e}_2 \mapsto \mathsf{e}' \qquad \gg\ \mathtt{step\_app\_beta}\ \mathtt{M_{e2}}\ (\lambda\, \mathtt{x}.\, \mathtt{M_e})\ \mathtt{M_{t2}}\ \mathtt{M_2} \Leftarrow \mathtt{step}\ (\mathtt{app}\ (\mathtt{lam}\ \mathtt{M_{t2}}\ (\lambda\, \mathtt{x}.\, \mathtt{M_e}))\ \mathtt{M_{e2}})\ \mathtt{M'_e}}}\ \text{ENC\_STEP\_APP\_BETA}$$

Fig. 12. Encoding of STLC Dynamic Semantics

The canonical forms of this type are terms $\lambda\,\mathtt{x}.\,\lambda\,\mathtt{dx}.\,\mathtt{M}$, where $\mathtt{M}$ has LF type $\mathtt{of}\ (\mathtt{E}\ \mathtt{x})\ \mathtt{T}$ in an LF context including $\mathtt{x}:\mathtt{tm}, \mathtt{dx}:\mathtt{of}\ \mathtt{x}\ \mathtt{T}_2$. These LF assumptions correspond exactly to a derivation under the hypothesis $\mathtt{x}:\tau_2$ for a fresh variable $\mathtt{x}$. This higher-order representation of hypotheses is natural and advantageous: there is no need to explicitly represent contexts and operations on them, and, moreover, LF provides structural properties such as weakening and substitution when hypothetical judgements are represented in such a fashion.

The correspondence between object-language derivations and their LF representations is made precise by the judgements in Figures 11 and 12. The intended reading of these judgements is as follows:

- $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ Encode an object-language context γ to an LF context Γ.
- $\mathcal{D} :: \gamma \vdash \mathsf{e}:\tau \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M}_\mathsf{e}\ \mathtt{M}_\mathsf{t}$ Assuming $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$, encode an object-language derivation $\mathcal{D}$ of $\gamma \vdash \mathsf{e}:\tau$ to an LF term $\mathtt{M}$ of type $\mathtt{of}\ \mathtt{M}_\mathsf{e}\ \mathtt{M}_\mathsf{t}$ in Γ. The notation $\mathcal{D} :: \gamma \vdash \mathsf{e}:\tau$ is simply linear notation for
$$\begin{array}{c}\mathcal{D}\\ \gamma \vdash \mathsf{e}:\tau.\end{array}$$
- $\mathcal{D} :: \mathsf{e}\ \mathsf{value} \gg \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M}_\mathsf{e}$ Encode an object-language derivation $\mathcal{D}$ of $\mathsf{e}\ \mathsf{value}$ for a closed term e to an LF term $\mathtt{M}$ of type $\mathtt{value}\ \mathtt{M}_\mathsf{e}$ in the empty LF context.
- $\mathcal{D} :: \mathsf{e} \mapsto \mathsf{e}' \gg \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M}_\mathsf{e}\ \mathtt{M}'_\mathsf{e}$ Encode an object-language derivation $\mathcal{D}$ of $\mathsf{e} \mapsto \mathsf{e}'$ for closed terms e and e' to an LF term $\mathtt{M}$ of type $\mathtt{step}\ \mathtt{M}_\mathsf{e}\ \mathtt{M}'_\mathsf{e}$ in the empty LF context.

The rule ENC_OF_LAM expresses the higher-order representation strategy described above. The rule ENC_STEP_APP_BETA includes a hereditary substitution premise because the right-hand term of the informal rule STEP_APP_BETA is $[\mathsf{e}_2/\mathsf{x}]\mathsf{e}$; compositionality of the encoding (*Theorem 3.7*) shows that this substitution can be represented by writing $(\mathtt{E}\ \mathtt{E}_2)$ as the right-hand term in `step_app_beta`.

These encoding judgements clarify the fact that the LF representation of the object language judgements is specified not just by the LF signature, but also by the LF contexts in which that signature is considered. For example, the object language operational semantics judgement $\mathsf{e} \mapsto \mathsf{e}'$ presupposes that the terms e and e' are closed. In the LF representation, this presupposition is reflected in the fact that we consider only the inhabitants of `step` in the empty LF context. Note that the same LF signature for `step`, considered in other LF contexts, could adequately represent an object-language transition system that does not presuppose closed terms.

3.4 Adequacy of the Judgement Encodings

Let Σ stand for the signature in Figure 10. When proving adequacy, we consider each judgement of the object language in the fragment of Σ relevant to it. (This is in contrast to Section 3.2, where for simplicity we considered adequacy of both types and terms in the full signature.) Observe by the definition of signature restriction that

- $\Sigma|_{\mathtt{tp}}$ contains the family declaration `tp` and the constants `unit` and `arrow`.
- $\Sigma|_{\mathtt{tm}}$ contains the family declarations `tp`, `tm` and their associated constants.
- $\Sigma|_{\mathtt{of}}$ contains the family declarations `tp`, `tm`, `of` and their associated constants.
- $\Sigma|_{\mathtt{value}}$ contains the family declarations `tp`, `tm`, `value` and their associated constants.
- $\Sigma|_{\mathtt{step}}$ contains the family declarations `tp`, `tm`, `value`, `step` and their associated constants.

The strongest subordination relation $\preceq_\Sigma$ is the reflexive-transitive closure of the following relation: $\mathtt{tp} \preceq \mathtt{tm}$, $\mathtt{tm} \preceq \mathtt{of}$, $\mathtt{tp} \preceq \mathtt{of}$, $\mathtt{tm} \preceq \mathtt{value}$, $\mathtt{tp} \preceq \mathtt{value}$, $\mathtt{tm} \preceq \mathtt{step}$, $\mathtt{tp} \preceq \mathtt{step}$, and $\mathtt{value} \preceq \mathtt{step}$. Additionally, for each $\mathtt{a}$ of these type families, $\preceq_\Sigma |_{\mathtt{a}} = \preceq_{\Sigma|_{\mathtt{a}}}$. Thus, *Theorem 2.17* shows that each of these signature restrictions is well-formed in the strongest subordination relation for it. Our previous adequacy theorems show that the encodings of terms and types are adequate in $\Sigma|_{\mathtt{tm}}$. An easy application of *Theorem 2.17* shows that adequacy for STLC types, *Theorem 3.1*, holds in $\Sigma|_{\mathtt{tp}}$ as well.

3.4.1 Static Semantics

In this section, we work in $\mathrm{LF}[\Sigma|_{\mathtt{of}}]$. Just as it was necessary to ensure that the encoding of types remained adequate in the signature and contexts for terms, it is now necessary to show that the encodings of both types and terms remain adequate in the signature and contexts we consider for typing derivations:

Lemma 3.9 (*Transport of Adequacy for Typing*)

1. $(\Sigma|_{\mathtt{of}})|_{\mathtt{tp}} = \Sigma|_{\mathtt{tp}}$ and $(\Sigma|_{\mathtt{of}})|_{\mathtt{tm}} = \Sigma|_{\mathtt{tm}}$.
2. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ then $\Gamma\ \mathtt{ctx}$.
3. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ then $\Gamma|_{\mathtt{tp}} = \cdot$.
4. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$, $\mathsf{vars}\,\gamma\ \mathcal{X}$, and $\Gamma|_{\mathtt{tm}} = \Gamma'$ then $\mathcal{X}\ \mathsf{terms} \gg \Gamma'\ \mathtt{ctx}$
5. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ then $\Gamma \vdash_{\Sigma|_{\mathtt{of}}} \mathtt{M_t} \Leftarrow \mathtt{tp}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{tp}}} \mathtt{M_t} \Leftarrow \mathtt{tp}$.
6. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ then $\Gamma \vdash_{\Sigma|_{\mathtt{of}}} \mathtt{M_e} \Leftarrow \mathtt{tm}$ iff $\Gamma|_{\mathtt{tm}} \vdash_{\Sigma|_{\mathtt{tm}}} \mathtt{M_e} \Leftarrow \mathtt{tm}$.

Next, we state the inversion lemma:

Lemma 3.10 (*Inversion of Canonical Forms of Type* `of`)
If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ and $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$ then one of the following hold:

- $\mathtt{M} = \mathtt{dx}$, $\mathtt{M_e} = \mathtt{x}$, and $\Gamma = \Gamma_1, \mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{M_t}, \Gamma_2$ for some variables $\mathtt{dx}$ and $\mathtt{x}$.
- $\mathtt{M} = \mathtt{of\_empty}$, $\mathtt{M_e} = \mathtt{empty}$, and $\mathtt{M_t} = \mathtt{unit}$.
- $\mathtt{M} = \mathtt{of\_app}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}}\ \mathtt{M_{t2}}\ \mathtt{M_t}\ \mathtt{M_1}\ \mathtt{M_2}$, $\mathtt{M_e} = (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}})$, and the following were derived as strict subderivations: $\Gamma \vdash \mathtt{M_{e1}} \Leftarrow \mathtt{tm}$ and $\Gamma \vdash \mathtt{M_{e2}} \Leftarrow \mathtt{tm}$ and $\Gamma \vdash \mathtt{M_{t2}} \Leftarrow \mathtt{tp}$ and $\Gamma \vdash \mathtt{M_t} \Leftarrow \mathtt{tp}$ and $\Gamma \vdash \mathtt{M_1} \Leftarrow \mathtt{of}\ \mathtt{M_{e1}}\ (\mathtt{arrow}\ \mathtt{M_{t2}}\ \mathtt{M_t})$ and $\Gamma \vdash \mathtt{M_2} \Leftarrow \mathtt{of}\ \mathtt{M_{e2}}\ \mathtt{M_{t2}}$.
- $\mathtt{M} = \mathtt{of\_lam}\ \mathtt{M_{t2}}\ \mathtt{M'_t}\ (\lambda\,\mathtt{x}.\,\mathtt{M'_e})\ (\lambda\,\mathtt{x}.\,\lambda\,\mathtt{dx}.\,\mathtt{M_2})$, $\mathtt{M_e} = (\mathtt{lam}\ \mathtt{M_t}\ (\lambda\,\mathtt{x}.\,\mathtt{M'_e}))$, $\mathtt{M_t} = (\mathtt{arrow}\ \mathtt{M_{t2}}\ \mathtt{M'_t})$, and the following were derived as strict subderivations: $\Gamma \vdash \mathtt{M_{t2}} \Leftarrow \mathtt{tp}$ and $\Gamma \vdash \mathtt{M'_t} \Leftarrow \mathtt{tp}$ and $\Gamma, \mathtt{x} : \mathtt{tm} \vdash \mathtt{M'_e} \Leftarrow \mathtt{tm}$ and $\Gamma, \mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{M_{t2}} \vdash \mathtt{M_2} \Leftarrow \mathtt{of}\ \mathtt{M'_e}\ \mathtt{M'_t}$.

In the statement and proof of adequacy, it will be convenient to abuse notation by writing $\gamma \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$ to mean the following: $\mathsf{vars}\gamma\,\mathcal{X}$ and $\Gamma|_{\mathtt{tm}} = \Gamma'$ and $\mathcal{X} \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma' \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$. In *Definition 2.16*, we observed that given Γ and $\mathtt{A}$, there exists a unique Γ' such that $\Gamma|_{\mathtt{A}} = \Gamma'$. Observe (by induction on γ) that given a well-formed context γ, there exists a unique $\mathcal{X}$ such that $\mathsf{vars}\ \gamma\ \mathcal{X}$. Thus, both $\mathcal{X}$ and Γ' are uniquely determined when we write $\gamma \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$.

We can now prove adequacy. The first condition states not only that the subjects are well-formed, as above, but also that the indices to the judgement are encoded in the appropriate manner.

Theorem 3.11 (*Adequacy for Typing Derivations*)

1. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ and $\mathcal{D} :: \gamma \vdash \mathsf{e} : \tau \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$ then
   - $\tau\ \mathsf{type} \gg \mathtt{M_t} \Leftarrow \mathtt{tp}$ and $\gamma \vdash \mathsf{e}\ \mathsf{term} \gg \Gamma \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$,
   - $\mathcal{D}$ derives $\gamma \vdash \mathsf{e} : \tau$, and
   - $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$.
2. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ and $\mathcal{D}$ derives $\gamma \vdash \mathsf{e} : \tau$ then there exist unique $\mathtt{M_e}$, $\mathtt{M_t}$ and $\mathtt{M}$ such that $\mathcal{D} :: \gamma \vdash \mathsf{e} : \tau \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$.
3. If $\gamma\ \mathsf{ctx} \gg \Gamma\ \mathtt{ctx}$ and $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$ then there exist unique τ, e, and $\mathcal{D}$ such that $\mathcal{D} :: \gamma \vdash \mathsf{e} : \tau \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$.

Proof
The proof follows the same general pattern as adequacy for syntax. It uses *Lemma 3.9*, *Theorem 3.5*, *Theorem 3.1*, *Lemma 2.12*, and *Lemma 3.10*. □

In informal descriptions of programming languages, we do not usually consider the operation of substituting one typing derivation into another (i.e., the computational content of the proof of the substitution theorem). However, it is possible to define such an operation and then prove a compositionality theorem, analogous to *Theorem 3.7*, for the judgement $\mathcal{D} :: \gamma \vdash \mathsf{e} : \tau \gg \Gamma \vdash \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$. However, because this compositionality result is not necessary for the remainder of this article, we elide the details.

3.4.2 Dynamic Semantics

The adequacy proof for the value judgement is simple; we elide the transport of adequacy lemma and the canonical forms inversion lemma, which are analogous to those given above.

Theorem 3.12 (*Adequacy for Values*)

1. If $\mathcal{D} :: \mathsf{e}\ \mathsf{value} \gg \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$ then
   - $\cdot \vdash \mathsf{e}\ \mathsf{term} \gg \cdot \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$,
   - $\mathcal{D}$ derives $\mathsf{e}\ \mathsf{value}$, and
   - $\cdot \vdash_{\Sigma|_{\mathtt{value}}} \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$.
2. If $\cdot \vdash \mathsf{e}\ \mathsf{term}$ and $\mathcal{D} :: \mathsf{e}\ \mathsf{value}$ then there exist unique $\mathtt{M_e}$ and $\mathtt{M}$ such that $\mathcal{D} :: \mathsf{e}\ \mathsf{value} \gg \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$.

3. If $\cdot \vdash_{\Sigma|_{\mathtt{value}}} \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$ then there exist unique e and $\mathcal{D}$ such that $\mathcal{D} :: \mathsf{e}\ \mathsf{value} \gg \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$.

In the remainder of this section, we work in $\mathrm{LF}[\Sigma|_{\mathtt{step}}]$.

Lemma 3.13 (*Transport of Adequacy for Operational Semantics*)

1. For $\mathtt{a} \in \{\mathtt{tm}, \mathtt{tp}, \mathtt{value}\}$, $(\Sigma|_{\mathtt{step}})|_{\mathtt{a}} = \Sigma|_{\mathtt{a}}$.
2. $\cdot \vdash_{\Sigma|_{\mathtt{step}}} \mathtt{M_e} \Leftarrow \mathtt{tm}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{tm}}} \mathtt{M_e} \Leftarrow \mathtt{tm}$.
3. $\cdot \vdash_{\Sigma|_{\mathtt{step}}} \mathtt{M_t} \Leftarrow \mathtt{tp}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{tp}}} \mathtt{M_t} \Leftarrow \mathtt{tp}$.
4. $\cdot \vdash_{\Sigma|_{\mathtt{step}}} \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{value}}} \mathtt{M} \Leftarrow \mathtt{value}\ \mathtt{M_e}$.

Next, we state the inversion principle:

Lemma 3.14 (*Inversion of Canonical Forms of Type* `step`)
If $\mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$ then one of the following hold:

- $\mathtt{M} = \mathtt{step\_app\_1}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}}\ \mathtt{M'_{e1}}\ \mathtt{M_1}$, $\mathtt{M_e} = (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}})$, $\mathtt{M'_e} = (\mathtt{app}\ \mathtt{M'_{e1}}\ \mathtt{M_{e2}})$, and the following were derived as strict subderivations: $\mathtt{M_{e1}} \Leftarrow \mathtt{tm}$ and $\mathtt{M_{e2}} \Leftarrow \mathtt{tm}$ and $\mathtt{M'_{e1}} \Leftarrow \mathtt{tm}$ and $\mathtt{M_1} \Leftarrow \mathtt{step}\ \mathtt{M_{e1}}\ \mathtt{M'_{e1}}$.
- $\mathtt{M} = \mathtt{step\_app\_2}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}}\ \mathtt{M'_{e2}}\ \mathtt{M_1}\ \mathtt{M_2}$, $\mathtt{M_e} = (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M_{e2}})$, $\mathtt{M'_e} = (\mathtt{app}\ \mathtt{M_{e1}}\ \mathtt{M'_{e2}})$, and the following were derived as strict subderivations: $\mathtt{M_{e1}} \Leftarrow \mathtt{tm}$ and $\mathtt{M_{e2}} \Leftarrow \mathtt{tm}$ and $\mathtt{M'_{e2}} \Leftarrow \mathtt{tm}$ and $\mathtt{M_1} \Leftarrow \mathtt{value}\ \mathtt{M_{e1}}$ and $\mathtt{M_2} \Leftarrow \mathtt{step}\ \mathtt{M_{e2}}\ \mathtt{M'_{e2}}$.
- $\mathtt{M} = \mathtt{step\_app\_beta}\ \mathtt{M_{e2}}\ (\lambda\, \mathtt{x}.\, \mathtt{M_b})\ \mathtt{M_{t2}}\ \mathtt{M_2}$, $\mathtt{M_e} = (\mathtt{app}\ (\mathtt{lam}\ \mathtt{M_{t2}}\ (\lambda\, \mathtt{x}.\, \mathtt{M_b}))\ \mathtt{M_{e2}})$, and the following were derived as strict subderivations: $\mathtt{M_{e2}} \Leftarrow \mathtt{tm}$ and $\mathtt{x} : \mathtt{tm} \vdash \mathtt{M_b} \Leftarrow \mathtt{tm}$ and $\mathtt{M_{t2}} \Leftarrow \mathtt{tp}$ and $\mathtt{M_2} \Leftarrow \mathtt{value}\ \mathtt{M_{e2}}$ and $[\mathtt{M_{e2}}/\mathtt{x}]^m_{\mathtt{tm}}\mathtt{M_b} = \mathtt{M'_e}$.

Finally, we state adequacy.

Theorem 3.15 (*Adequacy for Operational Semantics*)

1. If $\mathcal{D} :: \mathsf{e} \mapsto \mathsf{e}' \gg \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$ then
   - $\cdot \vdash \mathsf{e}\ \mathsf{term} \gg \cdot \vdash \mathtt{M_e} \Leftarrow \mathtt{tm}$ and $\cdot \vdash \mathsf{e}'\ \mathsf{term} \gg \cdot \vdash \mathtt{M'_e} \Leftarrow \mathtt{tm}$,
   - $\mathcal{D}$ derives $\mathsf{e} \mapsto \mathsf{e}'$, and
   - $\mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$.
2. If $\cdot \vdash \mathsf{e}\ \mathsf{term}$ and $\cdot \vdash \mathsf{e}'\ \mathsf{term}$ and $\mathcal{D}$ derives $\mathsf{e} \mapsto \mathsf{e}'$ then there exist unique $\mathtt{M_e}$, $\mathtt{M'_e}$ and $\mathtt{M}$ such that $\mathcal{D} :: \mathsf{e} \mapsto \mathsf{e}' \gg \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$.
3. If $\mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$ then there exist unique e, e', and $\mathcal{D}$ such that $\mathcal{D} :: \mathsf{e} \mapsto \mathsf{e}' \gg \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$.

Proof
An object-language derivation using the rule STEP_APP_BETA contains a subderivation deriving $[\mathsf{e}_2/\mathsf{x}]\mathsf{e}$. However, because substitution in the object-language is represented as substitution in LF, this substitution derivation is not encoded as an explicit LF object. Consequently, it is necessary to show that there is at most one object-language substitution derivation; *Lemma 3.8* establishes this fact. Other than this complication, the proof is similar to adequacy of typing. The proof uses *Theorem 2.11*, *Theorem 3.5*, *Theorem 3.7*, *Theorem 3.12*, *Lemma 3.13*, and *Lemma 3.14*. □

4 Mechanizing the Metatheory of the STLC

A *metatheorem* is a statement about an object language. The Twelf implementation of LF (Pfenning & Schürmann, 1999) permits the mechanization of metatheorems about languages encoded in LF.

Type Families as Relations. In describing the metatheoretic capabilities of Twelf, it is useful to regard an LF type family as defining a relation on the family's indices, where indices are related iff their instance of the type family is inhabited. In the simplest case, when a type family's kind is not dependent and when only closed LF terms are considered, a type family defines a single relation on the family's indices, where indices are related iff the corresponding type is inhabited. For example, the type family $\mathtt{step} : \mathtt{tm} \to \mathtt{tm} \to \mathtt{type}$ defines a relation between two LF terms of type $\mathtt{tm}$, where $\mathtt{E}$ and $\mathtt{E}'$ are related iff there exists an LF term $\mathtt{D}$ of type $\mathtt{step}\,\mathtt{E}_1\,\mathtt{E}_2$. More formally, we say that a type family $\mathtt{a} : \mathtt{A}_1 \to \ldots \to \mathtt{A}_n \to \mathtt{type}$ defines a relation R on terms $\cdot \vdash \mathtt{M}_1 \Leftarrow \mathtt{A}_1, \ldots, \cdot \vdash \mathtt{M}_n \Leftarrow \mathtt{A}_n$, where $R(\mathtt{M}_1, \ldots, \mathtt{M}_n)$ iff there exists an LF term $\mathtt{D}$ such that $\mathtt{D} \Leftarrow \mathtt{a}\,\mathtt{M}_1\ \ldots\ \mathtt{M}_n$.

This simple case generalizes in two ways. First, when LF terms in non-empty contexts are considered, a type family constitutes a simultaneous definition of a context-indexed family of relations on the type family's indices, where indices are related by the relation for a particular context iff their instance of the family is inhabited in that context. For example, the type family $\mathtt{of} : \mathtt{tm} \to \mathtt{tp} \to \mathtt{type}$ defines a context-indexed family of relations R, where each relation R_Γ relates terms $\mathtt{E}$ and $\mathtt{T}$ such that $\Gamma \vdash \mathtt{E} \Leftarrow \mathtt{tm}$ and $\Gamma \vdash \mathtt{T} \Leftarrow \mathtt{tp}$, and $R_\Gamma(\mathtt{E}, \mathtt{T})$ holds iff there exists an LF term $\mathtt{D}$ such that $\Gamma \vdash \mathtt{D} \Leftarrow \mathtt{of}\,\mathtt{E}\,\mathtt{T}$. Note that the type family constitutes a simultaneous inductive definition of all the relations R_Γ, as the definition of R_Γ refers to other relations $R_{\Gamma'}$ (e.g., when the premise of $\mathtt{of\_lam}$ extends the LF context). More formally, we say that a type family $\mathtt{a} : \mathtt{A}_1 \to \ldots \to \mathtt{A}_n \to \mathtt{type}$ defines a context-indexed family of relations R, where each R_Γ is a relation on terms $\Gamma \vdash \mathtt{M}_1 \Leftarrow \mathtt{A}_1, \ldots, \Gamma \vdash \mathtt{M}_n \Leftarrow \mathtt{A}_n$ and $R_\Gamma(\mathtt{M}_1, \ldots, \mathtt{M}_n)$ iff there exists a canonical form $\mathtt{D}$ such that $\Gamma \vdash \mathtt{D} \Leftarrow \mathtt{a}\,\mathtt{M}_1\ \ldots\ \mathtt{M}_n$.

Second, the kind of a type family may in general be a dependent kind of the form $\Pi\,\mathtt{x}_1{:}\mathtt{A}_1 \ldots \Pi\,\mathtt{x}_n{:}\mathtt{A}_n.\,\mathtt{type}$, in which case the type family defines a dependent relation—the types of the related terms may vary with the other indices to the relation. In full generality, a type family $\mathtt{a} : \Pi\,\mathtt{x}_1{:}\mathtt{A}_1 \ldots \Pi\,\mathtt{x}_n{:}\mathtt{A}_n.\,\mathtt{type}$ defines a context-indexed family of relations R, where R_Γ relates terms $\mathtt{M}_1$ such that $\Gamma \vdash \mathtt{M}_1 \Leftarrow \mathtt{A}_1$, $\mathtt{M}_2$ such that $[\mathtt{M}_1/\mathtt{x}_1]^a_{\mathtt{A}_1}\mathtt{A}_2 = \mathtt{A}'_2$ and $\Gamma \vdash \mathtt{M}_2 \Leftarrow \mathtt{A}'_2$, $\mathtt{M}_3$ such that $[\mathtt{M}_2/\mathtt{x}_2]^a_{\mathtt{A}_2}[\mathtt{M}_1/\mathtt{x}_1]^a_{\mathtt{A}_1}\mathtt{A}_3 = \mathtt{A}'_3$, and so on, and $R_\Gamma(\mathtt{M}_1, \ldots, \mathtt{M}_n)$ iff there exists a canonical form $\mathtt{D}$ such that $\Gamma \vdash \mathtt{D} \Leftarrow \mathtt{a}\,\mathtt{M}_1\ \ldots\ \mathtt{M}_n$. Though we have not used any dependent kinds in the previous sections of this article, we discuss many examples of them below.

Totality Assertions. Twelf has the ability to mechanically verify *totality assertions* about LF type families. To a first approximation, a totality assertion for a type family corresponds with the standard notion of totality for the relation defined by the type family: a totality assertion is specified by designating some indices as

inputs and the remaining indices as outputs; then the totality assertion asserts that for all inputs, there exist outputs that stand in the relation. However, because a type family defines a family of relations R_Γ, the specification of a totality assertion must also clarify for which contexts Γ the relations R_Γ are asserted to be total. In general, we may wish to state a totality assertion only for contexts of a particular form—because, for example, the totality assertion may be true for contexts of one form but false for contexts of another. To allow this, we parametrize the specification of a totality assertion by a world (set of contexts) $\mathcal{W}$, and the totality assertion asserts that for all contexts $\Gamma \in \mathcal{W}$, the relation R_Γ is total.

Thus, the specification of a totality assertion for a given type family consists of two declarations: a *mode* declaration, which identifies some of the family's indices as inputs (we notate these $\mathtt{A_i}$) and the others as outputs (we notate these $\mathtt{B_j}$), and a *world* declaration, which restricts the totality assertion to LF contexts in a particular world $\mathcal{W}$. Given these declarations, the totality assertion for a type family is defined as follows.

Definition 4.1 (*Totality Assertions*)
The totality assertion for a type family $\mathtt{a} : \Pi\, \mathtt{x_1}{:}\mathtt{A_1} \ldots \Pi\, \mathtt{y_1}{:}\mathtt{B_1} \ldots \mathtt{type}$ with inputs $\mathtt{A_1}, \ldots, \mathtt{A_m}$ and outputs $\mathtt{B_1}, \ldots, \mathtt{B_n}$ in world $\mathcal{W}$ is the proposition

$$\begin{array}{c}\text{For all } \Gamma \in \mathcal{W}, \text{ for all } \mathtt{M_1} \text{ such that } \Gamma \vdash \mathtt{M_1} \Leftarrow \mathtt{A'_1}, \ldots, \text{ and } \mathtt{M_m} \text{ such that} \\ \Gamma \vdash \mathtt{M_m} \Leftarrow \mathtt{A'_m}, \text{ there exist } \mathtt{N_1} \text{ such that } \Gamma \vdash \mathtt{N_1} \Leftarrow \mathtt{B'_1}, \ldots, \text{ and } \mathtt{N_n} \text{ such that} \\ \Gamma \vdash \mathtt{N_n} \Leftarrow \mathtt{B'_n}, \text{ and } \mathtt{D} \text{ such that } \Gamma \vdash \mathtt{D} \Leftarrow \mathtt{a}\, \mathtt{M_1} \ldots \mathtt{M_m}\, \mathtt{N_1} \ldots \mathtt{N_n}.\end{array}$$

where we write $\mathtt{A'_i}$ and $\mathtt{B'_j}$ for the appropriate substitutions into $\mathtt{A_i}$ and $\mathtt{B_j}$ to account for the dependent nature of the relation:

$$\begin{array}{c}[\mathtt{M_{i-1}}/\mathtt{x_{i-1}}]_{\mathtt{A_{i-1}}} \ldots [\mathtt{M_1}/\mathtt{x_1}]_{\mathtt{A_1}} \mathtt{A_i} = \mathtt{A'_i} \\ [\mathtt{N_{j-1}}/\mathtt{y_{j-1}}]_{\mathtt{B_{j-1}}} \ldots [\mathtt{N_1}/\mathtt{y_1}]_{\mathtt{B_1}} [\mathtt{M_m}/\mathtt{x_m}]_{\mathtt{A_m}} \ldots [\mathtt{M_1}/\mathtt{x_1}]_{\mathtt{A_1}} \mathtt{B_j} = \mathtt{B'_j}.\end{array}$$

For example, we may specify a totality assertion for the type family $\mathtt{step}\, \mathtt{E}\, \mathtt{E'}$ by declaring $\mathtt{E}$ an input and $\mathtt{E'}$ an output and by restricting consideration to the empty LF context. These declarations specify the following totality assertion: for all $\mathtt{E}$ such that $\mathtt{E} \Leftarrow \mathtt{tm}$, there exist $\mathtt{E'}$ and $\mathtt{D}$ such that $\mathtt{E'} \Leftarrow \mathtt{tm}$ and $\mathtt{D} \Leftarrow \mathtt{step}\, \mathtt{E}\, \mathtt{E'}$. Of course, this totality assertion for the STLC is false, as there are terms that cannot take a step. Other judgements of an object language, such as a compiler transformation, might in fact define total relations.

The Twelf implementation permits a totality assertion to be specified by a mode declaration and a *regular worlds* declaration, which defines a world $\mathcal{W}$ by a regular expression over contexts. Moreover, Twelf provides a totality declaration which instructs Twelf to (attempt to) prove a totality assertion using a specified induction metric. Twelf proves a totality assertion for a type family by interpreting the type family as a higher-order logic program and proving that the logic program is total; see Schürmann & Pfenning (2003) for details. As with any theorem prover for sufficiently rich statements, there are many true totality assertions that Twelf is unable to prove. In the remainder of this section, we present many examples of type

families that Twelf can prove total, and we discuss how totality assertions are used in mechanizing metatheory.

Mechanizing Metatheory. The machinery of totality assertions can be deployed in a particular way to permit the mechanical verification of a much wider class of metatheorems than totality assertions themselves. General $\forall\exists$*-statements* of the form

$$\text{For all } \Gamma \in \mathcal{W}, \text{ for all } \mathtt{M_1} \text{ such that } \Gamma \vdash \mathtt{M_1} \Leftarrow \mathtt{A_1}, \ldots, \text{ and } \mathtt{M_m} \text{ such that } \Gamma \vdash \mathtt{M_m} \Leftarrow \mathtt{A_m}, \text{ there exist } \mathtt{N_1} \text{ such that } \Gamma \vdash \mathtt{N_1} \Leftarrow \mathtt{B_1}, \ldots, \text{ and } \mathtt{N_n} \text{ such that } \Gamma \vdash \mathtt{N_n} \Leftarrow \mathtt{B_n}.$$

can be mechanized in Twelf. A proof of a $\forall\exists$-statement consists of a transformation from the universally quantified terms (inputs) into the existentially quantified terms (outputs), typically defined by induction on the inputs. One way of presenting such a transformation is as a total relation. Consequently, a $\forall\exists$-statement and its inductive proof can be represented as an LF type family and verified by Twelf's totality checker. Specifically, a statement of this form can be translated into the type family, mode, and worlds declarations specifying the totality of a type family $\mathtt{a} : \Pi\,\mathtt{x_1}{:}\mathtt{A_1}. \ldots \Pi\,\mathtt{y_n}{:}\mathtt{B_n}.\,\mathtt{type}$. An inductive proof of such a statement can be translated into LF constants that inductively define the type family `a` in such a way that Twelf can verify the its totality. The original $\forall\exists$-statement is a corollary of the totality assertion for this particular relation. That is, we deploy Twelf's ability to *prove* totality assertions in order to *check* proofs of general $\forall\exists$-statements by representing these proofs explicitly as LF terms. A successful proof constitutes a sufficiently detailed definition of a relation that Twelf can verify its totality.

This methodology is useful because a wide class of metatheorems can expressed as $\forall\exists$-statements about the canonical forms of LF. Because of the judgements-as-types representation strategy, the quantifiers of $\forall\exists$-statements range over not just the representations of object-language individuals such as terms and types but also the representations of derivations of object-language judgements. For example, the standard informal statement of preservation is the following:

Theorem 4.2 (*Informal Statement of Preservation*)
For all types τ and all closed terms $\mathtt{e}$ and $\mathtt{e}'$, if $\mathtt{e} \mapsto \mathtt{e}'$ and $\cdot \vdash \mathtt{e} : \tau$ then $\cdot \vdash \mathtt{e}' : \tau$.

By adequacy (*Theorem 3.1*, *Theorem 3.5*, *Theorem 3.11*, and *Theorem 3.15*), this statement can be recast as the following $\forall\exists$-statement:

Theorem 4.3 (*Statement of Preservation via the Encoding*)
For all $\mathtt{E}$ such that $\mathtt{E} \Leftarrow \mathtt{tm}$, $\mathtt{E}'$ such that $\mathtt{E}' \Leftarrow \mathtt{tm}$, $\mathtt{T}$ such that $\mathtt{T} \Leftarrow \mathtt{tp}$, $\mathtt{D_{step}}$ such that $\mathtt{D_{step}} \Leftarrow \mathtt{step}\,\mathtt{E}\,\mathtt{E}'$, and $\mathtt{D_{of}}$ such that $\mathtt{D_{of}} \Leftarrow \mathtt{of}\,\mathtt{E}\,\mathtt{T}$, there exists a $\mathtt{D'_{of}}$ such that $\mathtt{D'_{of}} \Leftarrow \mathtt{of}\,\mathtt{E}'\,\mathtt{T}$.

To prove this statement in Twelf, we recast the informal proof of preservation as constants inhabiting an LF type family

$$\mathtt{preserv} \;:\; \Pi\,\mathtt{E}{:}\mathtt{tm}.\,\Pi\,\mathtt{E}'{:}\mathtt{tm}.\,\Pi\,\mathtt{T}{:}\mathtt{tp}.\,\mathtt{step}\,\mathtt{E}\,\mathtt{E}' \rightarrow \mathtt{of}\,\mathtt{E}\,\mathtt{T} \rightarrow \mathtt{of}\,\mathtt{E}'\,\mathtt{T} \rightarrow \mathtt{type}.$$

The type family `preserv` has a dependent kind, accounting for the occurrences of the metavariables in the theorem statement. Then, we ask Twelf to verify the appropriate totality assertion:

Theorem 4.4 (*Totality of Preservation*)
For all $\mathtt{E}$ such that $\mathtt{E} \Leftarrow \mathtt{tm}$, $\mathtt{E}'$ such that $\mathtt{E}' \Leftarrow \mathtt{tm}$, $\mathtt{T}$ such that $\mathtt{T} \Leftarrow \mathtt{tp}$, $\mathtt{D}_{\mathtt{step}}$ such that $\mathtt{D}_{\mathtt{step}} \Leftarrow \mathtt{step}\,\mathtt{E}\,\mathtt{E}'$, and $\mathtt{D}_{\mathtt{of}}$ such that $\mathtt{D}_{\mathtt{of}} \Leftarrow \mathtt{of}\,\mathtt{E}\,\mathtt{T}$, there exist $\mathtt{D}'_{\mathtt{of}}$ such that $\mathtt{D}'_{\mathtt{of}} \Leftarrow \mathtt{of}\,\mathtt{E}'\,\mathtt{T}$ and $\mathtt{D}$ such that $\mathtt{D} \Leftarrow \mathtt{preserv}\,\mathtt{E}\,\mathtt{E}'\,\mathtt{T}\,\mathtt{D}_{\mathtt{of}}\,\mathtt{D}_{\mathtt{step}}\,\mathtt{D}'_{\mathtt{of}}$.

To state and prove preservation, it suffices to consider a $\forall\exists$-statement for the world containing only the empty LF context. In general, $\forall\exists$-statements about non-empty LF contexts are useful for stating properties of object languages encoded using higher-order representations. For example, we may recast a statement about STLC typing derivations in non-empty contexts γ as a statement about LF terms of type `of` in contexts Γ such that $\gamma\ \mathtt{ctx} \gg \Gamma\ \mathtt{ctx}$. We prove such a statement in Twelf by verifying the totality assertion for a corresponding type family in an appropriate world.

In summary, proving an informal $\forall\exists$-statement about an object language in Twelf requires four steps. First, via adequacy, recast the metatheorem as a $\forall\exists$-statement about canonical forms of particular types in a particular world. Second, define a corresponding LF type family `r` and annotate it with mode and worlds declarations specifying the corresponding totality assertion. Third, extend the LF signature with constants inhabiting `r` for all canonical inputs. Fourth, ask Twelf to verify that `r` satisfies the totality assertion by executing a totality declaration. In the remainder of this section, we show several examples of this methodology for mechanizing metatheorems in Twelf.

Twelf Concrete Syntax. In this section, we will use Twelf's concrete syntax for LF. The type $\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{A}$ is written as `{x:A2} A`, the kind $\Pi\,\mathtt{x}{:}\mathtt{A}_2.\,\mathtt{K}$ is written as `{x:A2} K`, and the term $\lambda\,\mathtt{x}.\,\mathtt{M}$ is written as `[x] M`. Parentheses are used for grouping, and the usual λ-calculus associativities and precedences apply: application associates to the left; $\rightarrow$ associates to the right; the scope of a binder extends as far to the right as possible. Twelf's concrete syntax also includes several conveniences that we will exploit:

- When declaring a constant in a signature, if an identifier beginning with a lower-case letter is not bound, Twelf reports an error. If an identifier beginning with an upper-case letter is not bound, Twelf implicitly binds it in a Π at the front of the constant's type or kind. The application of a constant to these implicit arguments is then inferred.
- Twelf allows a programmer to write an arrow type `A -> B` with a backward arrow—`B <- A`. This makes it easier to see the head family of a constant.
- Twelf allows arbitrary type annotations by writing `M : A` in place of any `M`.
- Twelf permits non-canonical forms, which are treated as syntactic sugar for the corresponding canonical form.

In Figure 13, we present an example of the first two conveniences: we show Twelf concrete syntax for the signature from Figure 10 defining the static and dynamic

```
of       : tm -> tp -> type.
of_empty : of empty unit.
of_lam   : of (lam T2 ([x] E x)) (arrow T2 T)
           <- ({x:tm} (of x T2) -> (of (E x) T)).
of_app   : of (app E1 E2) T
           <- of E2 T2
           <- of E1 (arrow T2 T).

value       : tm -> type.
value_empty : value empty.
value_lam   : value (lam T2 ([x] E x)).

step          : tm -> tm -> type.
step_app_1    : step (app E1 E2) (app E1' E2)
                 <- step E1 E1'.
step_app_2    : step (app E1 E2) (app E1 E2')
                 <- step E2 E2'
                 <- value E1.
step_app_beta : step (app (lam T2 ([x] E x)) E2) (E E2)
                 <- value E2.
```

Fig. 13. Static and Dynamic Semantics for the STLC in Twelf Concrete Syntax

semantics of the STLC. In idiomatic Twelf, it is common to reverse the order of the premises so that they read, from top to bottom, in the same order as they would read from left to right in an inference rule; however, to keep the order of arguments consistent with the signature in Figure 10, we have not done this here.

4.1 Type Preservation

Type preservation is our first example of a Twelf metatheorem.

4.1.1 Twelf Proof

Figure 14 contains a complete Twelf proof of preservation for the STLC. The type family `preserv` defines a relation among the appropriate types. The `%mode` and `%worlds` declarations state the appropriate totality assertion: the mode declaration declares that the first two indices of `preserv` are universally quantified (`+`), whereas the last one is existentially quantified (`-`). Implicitly quantified variables are universally quantified if they appear in any inputs and existentially quantified if they appear only in outputs, so in this case `E`, `E'`, and `T` are all inputs. The `%worlds` declaration declares that these quantifiers range over canonical forms in the world containing only the empty LF context.

The term-level constants `preserv_app_1`, `preserv_app_2`, and `preserv_app_beta` constitute an inductive definition of the `preserv` type family, codifying the cases of the proof of the $\forall\exists$-theorem. We prove preservation by induction on the operational semantics, writing one LF constant (i.e., case of the proof) for each operational semantics rule. For example, consider `preserv_app_1`. The intuitive reading of this

```
preserv : step E E' -> of E T -> of E' T -> type.
%mode preserv +Dstep +Dof -Dof'.

preserv_app_1    : preserv
                     (step_app_1 (DstepE1 : step E1 E1'))
                     (of_app (DofE1 : of E1 (arrow T2 T))
                             (DofE2 : of E2 T2))
                     (of_app DofE1' DofE2)
                     <- preserv DstepE1 DofE1 (DofE1' : of E1' (arrow T2 T)).

preserv_app_2    : preserv
                     (step_app_2 (DvalE1 : value E1) (DstepE2 : step E2 E2'))
                     (of_app (DofE1 : of E1 (arrow T2 T))
                             (DofE2 : of E2 T2))
                     (of_app DofE1 DofE2')
                     <- preserv DstepE2 DofE2 (DofE2' : of E2' T2).

preserv_app_beta : preserv
                     (step_app_beta (Dval : value E2))
                     (of_app (of_lam (([x] [dx] DofE x dx)
                                      : {x : tm} (of x T2) -> (of (E x) T)))
                             (DofE2 : of E2 T2))
                     (DofE E2 DofE2).

%worlds () (preserv _ _ _).
%total D (preserv D _ _).
```

Fig. 14. Twelf Proof of Preservation for the STLC

case is as follows: in the case for `step_app_1`, we invert the typing derivation because `of_app` is the only rule that could have applied, yielding typing derivations for `E1` and `E2`; we then appeal inductively to `preserv` on the smaller step derivation `DstepE1` and the typing derivation `DofE1` for `E1`, which yields a typing derivation $\texttt{DofE1}'$ for $\texttt{E1}'$; then we apply the rule `of_app` to this derivation and the derivation for `E2` to obtain the result. More formally, `preserv_app_1` is an LF constant inhabiting the family

$$\texttt{preserv}\ (\texttt{step\_app\_1}\ \texttt{Dstep1})\ (\texttt{of\_app}\ \texttt{DofE1}\ \texttt{DofE2})\ (\texttt{of\_app}\ \texttt{DofE1}'\ \texttt{DofE2})$$

as long as the family `preserv Dstep1 DofE1 DofE2` is inhabited. This extends the canonical forms that are related by `preserv`. The fact that the premise of the constant is on a smaller step derivation `Dstep1` is used in showing the totality of `preserv`. The constant `preserv_app_2` is similar. In `preserv_app_beta`, the intuitive reading is that we invert twice based on the syntactic form of the term on the left-hand side of the step derivation; then we apply the hypothetical typing derivation for the body of the function `DofE` to the argument `E2` and its typing derivation `DofE2`—because of the higher-order encoding, there is no need for a substitution lemma. Of course, `preserv_app_beta` is in fact just an LF constant inhabiting the type `preserv` for particular indices. The type annotations on each constant are

included only for readability; if we omitted their types, Twelf would infer them. The `%total` declaration asks Twelf to verify, by induction on the first argument `D`, that `preserv` satisfies the totality assertion in *Theorem 4.4*.

4.1.2 Correctness of the Statement of Preservation

In the introduction to this section, we stated the equivalence of the informal statement of preservation (*Theorem 4.2*) and the statement via the encoding (*Theorem 4.3*). We discuss the proof of this equivalence in more detail here. The quantifiers and their types clearly correspond, so adequacy should imply the equivalence of these two statements. However, there are two ways in which the previous adequacy statements could fail to give the necessary result:

1. Though we did not make this explicit in the introduction to this section, the Twelf proof proves *Theorem 4.3* in a different signature than the one in which the relevant types are adequate: the type family `preserv` and the terms inhabiting it extend the LF signature. It is necessary to check that the previous adequacy results can be transfered to this extended signature.
2. The `%worlds` declaration specifies the class of LF contexts for the metatheorem. It is necessary to check that the previous adequacy results give the desired correspondence in these contexts.

To address the first concern, let Σ be the original signature in Figure 10, and call the extension of this signature with `preserv` and its constants Σ'. The strongest subordination relation $\preceq_{\Sigma'}$ extends $\preceq_{\Sigma}$ with the conditions that $\mathtt{a} \preceq_{\Sigma'} \mathtt{preserv}$ for all `a` declared in Σ, but in $\preceq_{\Sigma'}$ no additional families are subordinate to any family declared in Σ. With this subordination relation, we can calculate that $\Sigma'|_{\mathtt{of}} = \Sigma|_{\mathtt{of}}$ and $\Sigma'|_{\mathtt{step}} = \Sigma|_{\mathtt{step}}$. Therefore, *Theorem 2.17* immediately implies the following lemma:

Lemma 4.5 (*Transport of Adequacy for Preservation*)

1. $\cdot \vdash_{\Sigma'} \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{of}}} \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$.
2. $\cdot \vdash_{\Sigma'} \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$ iff $\cdot \vdash_{\Sigma|_{\mathtt{step}}} \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$.

To address the second concern, observe that the `%worlds` declaration asserts the theorem only for canonical forms in the empty LF context. The previous adequacy results (*Theorem 3.1*, *Theorem 3.5*, *Theorem 3.11*, and *Theorem 3.15*) show that the empty LF context adequately represents types, closed terms, typing derivations in the empty object-language context, and operational semantics derivations.

4.1.3 Correctness of the Proof of Preservation

In processing the `%total` declaration, Twelf automatically verifies that `preserv` satisfies the totality specification in *Theorem 4.4*. However, we can also prove the totality of this relation by hand.

Proof of Theorem 4.4

```
%% identity (alpha-equivalence) of terms
id   : tm -> tm -> type.
refl : id E E.

%% application congruence lemma for identity
id_app_cong : id E1 E1'
               -> id E2 E2'
               -> id (app E1 E2) (app E1' E2')
               -> type.
%mode id_app_cong +X1 +X2 -X3.

- : id_app_cong refl refl refl.

%worlds () (id_app_cong _ _ _).
%total {} (id_app_cong _ _ _).

%% determinacy of evaluation
det : step E E' -> step E E'' -> id E' E'' -> type.
%mode det +X1 +X2 -X3.

det-1 : det (step_app_1 DstepE1') (step_app_1 DstepE1'') DidApp
         <- det DstepE1' DstepE1'' DidE1
         <- id_app_cong DidE1 refl DidApp.

det-2 : det (step_app_2 _ DstepE2') (step_app_2 _ DstepE2'') DidApp
         <- det DstepE2' DstepE2'' DidE2
         <- id_app_cong refl DidE2 DidApp.

det-b : det (step_app_beta _) (step_app_beta _) refl.

%worlds () (det _ _ _).
%total D (det D _ _).
```

Fig. 15. Twelf Proof of Determinacy of the STLC Operational Semantics

In what follows, we work in LF[Σ']. Take Γ to be $\cdot$. *Lemma 4.5* above justifies appealing to *Lemma 3.10* and *Lemma 3.14* on derivations of $\cdot \vdash_{\Sigma'} \mathtt{M} \Leftarrow \mathtt{of}\ \mathtt{M_e}\ \mathtt{M_t}$ and $\cdot \vdash_{\Sigma'} \mathtt{M} \Leftarrow \mathtt{step}\ \mathtt{M_e}\ \mathtt{M'_e}$. Because preservation is proved by induction on the dynamic semantics derivation, we can use *Lemma 3.14* on $\mathtt{D_{step}}$. This gives three cases to consider. In each case, we use *Lemma 3.10* to invert $\mathtt{D_{of}}$. In the case for `step_app_1`, the inductive hypothesis gives an inhabitant of the premise of `preserv_app_1`. The case for `step_app_2` is similar. In the case for `step_app_beta`, we use *Theorem 2.11* to instantiate the hypothetical typing derivation for the body of the function—because of the higher-order encoding, substitution for LF is used where one would expect to use substitution for the object language. □

4.2 Determinacy

Next, we show that the STLC's operational semantics are deterministic. This example illustrates several additional aspects of Twelf metatheory: we show how to give a simple representation of object-language α-equivalence as an LF type family; we show how to use a lemma in a Twelf proof; and we illustrate a circumstance in which Twelf's metatheorem checker allows vacuously-true proof cases to be elided. Determinacy is stated as follows:

Theorem 4.6 (*Informal Statement of Determinacy*)
For closed terms $\mathtt{e}$, $\mathtt{e}'$, and $\mathtt{e}''$, if $\mathtt{e} \mapsto \mathtt{e}'$ and $\mathtt{e} \mapsto \mathtt{e}''$ then $\mathtt{e}' =_\alpha \mathtt{e}''$.

Translating this informal statement into Twelf requires representing α-equivalence of STLC terms as an LF type family. Because of our higher-order representation strategy, α-equivalence is an intrinsic property of the representation: two object-language terms are α-equivalent iff their representations are α-equivalent canonical forms in LF (by *Theorem 3.5*). Consequently, we can represent object-language α-equivalence by internalizing α-equivalence of LF terms as a type family. The type family `id` at the top of Figure 15 does just this. For each `E` of type `tm`, there is one canonical form inhabiting `id`, `refl E`, expressing reflexivity (recall Twelf's convention that the variable `E` in the type of `refl` is implicitly quantified). With this definition, `id E E'` is inhabited exactly when the canonical forms `E` and `E'` are α-equivalent in LF. There is no need to give an inductive definition of α-equivalence, as one would give for raw abstract syntax trees.

Using `id`, the informal statement of determinacy can be rephrased as follows:

Theorem 4.7 (*Statement of Determinacy via the Encoding*)
For all `E`, `E'`, `E''` such that $\mathtt{E}, \mathtt{E}', \mathtt{E}'' \Leftarrow \mathtt{tm}$, `D'` such that $\mathtt{D}' \Leftarrow \mathtt{step\ E\ E}'$, and `D''` such that $\mathtt{D}'' \Leftarrow \mathtt{step\ E\ E}''$, there exists a `D` such that $\mathtt{D} \Leftarrow \mathtt{id\ E}'\ \mathtt{E}''$.

Adequacy (*Theorem 3.5*, *Theorem 3.15*, and the fact that `id` corresponds to object-language α-equivalence) implies that this theorem statement is equivalent to the informal statement in *Theorem 4.6*.

Figure 15 contains a complete Twelf proof of determinacy. The theorem statement in *Theorem 4.7* is represented by the type family `det` and its mode and world declarations in Figure 15, which state the corresponding totality assertion for the relation defined by the constants in the figure.

This proof uses a lemma, represented by the type family `id_app_cong`, its mode and world declarations, and its inhabitants. This lemma states a congruence property of α-equivalence: two applications are identical if their subterms are identical. The proof of this lemma has exactly one case, in which the two subterm equalities were both derived using reflexivity; in this case, the applications are identical, so reflexivity gives the result. (In the Twelf concrete syntax, the application of `refl` to its implicit term argument is suppressed, so all three derivations are written simply as `refl`.) The `%total` declaration for `id_app_cong` uses the empty lexicographic termination metric `{}` because the proof is not inductive. The totality of this lemma justifies using it as a premise in any future proof to produce a derivation

of `id (app E1 E2) (app E1' E2')` from two input derivations of the appropriate type.

The three constants with head `det`, named `det-1`, `det-2`, and `det-b`, define the relation that proves determinacy. In each of these three cases, the two `step` derivations conclude with the same final rule. In the case labeled `det-1` for `step_app_1`, we are given a derivation of the judgement $\texttt{step}\,(\texttt{app}\,\texttt{E}_1\,\texttt{E}_2)\,(\texttt{app}\,\texttt{E}_1'\,\texttt{E}_2)$ and a derivation of $\texttt{step}\,(\texttt{app}\,\texttt{E}_1\,\texttt{E}_2)\,(\texttt{app}\,\texttt{E}_1''\,\texttt{E}_2)$ with subderivations of $\texttt{step}\,\texttt{E}_1\,\texttt{E}_1'$ and $\texttt{step}\,\texttt{E}_1\,\texttt{E}_1''$. The `det` premise represents an appeal to the inductive hypothesis on the two subderivations, concluding that $\texttt{id}\,\texttt{E}_1'\,\texttt{E}_1''$. The `id_app_cong` premise appeals to the lemma on this identity derivation to conclude that the applications are equal. The case `det-2` is similar. When a `step` derivation ends with `step_app_beta`, the left-hand term completely determines the right-hand term, so in the case `det-b` the result is immediate by reflexivity. The `%total` declaration instructs Twelf to check that this type family represents a total relation by induction on the first `step` derivation—though, because of the symmetry, using the second derivation would also suffice. The totality of `id_app_cong` is used as a lemma in showing the totality of `det`—for example, it is used to show the existence of an inhabitant of `id_app_cong DidE1 refl DidApp` for some `DidApp` given `DidE1` and `refl`.

Twelf successfully proves the totality of `det`, even though the relation contains only cases where the final rule used in both `step` derivations is the same. This is because the off-diagonal cases are all vacuously true—and moreover, Twelf's metatheorem checker rules out these contradictory cases automatically. For example, if one derivation concluded with `step_app_1` and the other with `step_app_2`, then there would be subderivations concluding both $\texttt{value}\,\texttt{E}_1$ and $\texttt{step}\,\texttt{E}_1\,\texttt{E}_1'$. These two types can never be simultaneously inhabited: `step` is only inhabited when $\texttt{E}_1$ is an application, and there is no rule inhabiting `value` for an application. Similarly, if one derivation concluded with `step_app_1` and the other with `step_app_beta`, subderivations would give a `step` derivation whose left-hand side is a `lam`, which cannot exist. The other off-diagonal cases can be contradicted in a similar manner. Twelf's coverage checker rules out cases like these where the inputs to a metatheorem result in an uninhabited instance of some type family (Schürmann & Pfenning, 2003). This feature both eases the development of Twelf proofs and keeps these details from cluttering the final product. As another example, in the companion Twelf code, we give a proof of progress for the STLC that exploits this feature to avoid an explicit canonical forms lemma.

4.3 Strengthening

The strengthening property of the STLC is stated as follows:

Theorem 4.8 (*Informal Statement of Strengthening*)
If $\gamma, \mathtt{y} : \tau_0 \vdash \mathtt{e} : \tau$ and $\mathtt{y} \# \mathtt{e}$ then $\gamma \vdash \mathtt{e} : \tau$.

Strengthening is easy to prove if all types are inhabited (simply substitute any term of the appropriate type for `y`), but the proof we give in this section handles uninhabited types as well. This theorem might, naïvely, seem difficult to prove for

our higher-order representation of the object language, as it requires a detailed statement about variables that would seem to require their representation as data. In this section, we give a simple Twelf proof of this property without sacrificing the higher-order representation. Strengthening illustrates three additional features of Twelf metatheory: it is a theorem stated for a non-trivial world; its proof appeals to induction in an extended LF context; and its proof illustrates a common Twelf device for handling the variable case of a theorem.

To formalize the statement of strengthening in Twelf, we will need to capture the condition $\mathtt{y} \# \mathtt{e}$, that the variable $\mathtt{y}$ is not free in the term $\mathtt{e}$. However, under the representation strategy defined in Figure 8, the variable $\mathtt{y}$ is free in the term $\mathtt{e}$ exactly when the LF variable $\mathtt{y}$ is free in the term $\mathtt{M_e}$ representing $\mathtt{e}$. Therefore, adequacy (*Theorem 3.1*, *Theorem 3.5*, *Theorem 3.11*) implies that this informal statement of strengthening is equivalent to the following, where we let $\mathcal{W}_{\mathtt{of}}$ consist of all contexts Γ such that $\gamma\ \mathtt{ctx} \gg \Gamma\ \mathtt{ctx}$ and we work in LF[Σ], where Σ is the signature for the STLC in Section 3.

Theorem 4.9 (*Statement of Strengthening via the Encoding*)
For all $\Gamma \in \mathcal{W}_{\mathtt{of}}$, for all $\mathtt{E}$ such that $\Gamma \vdash \mathtt{E} \Leftarrow \mathtt{tm}$, $\mathtt{T}$ and $\mathtt{T_0}$ such that $\Gamma \vdash \mathtt{T}, \mathtt{T_0} \Leftarrow \mathtt{tp}$, and $\mathtt{D}$ such that $\Gamma, \mathtt{y} : \mathtt{tm}, \mathtt{dy} : \mathtt{of}\ \mathtt{y}\ \mathtt{T_0} \vdash \mathtt{D} \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$, there exists a $\mathtt{D'}$ such that $\Gamma \vdash \mathtt{D'} \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$.

Unlike preservation, which was stated for the world $\{\cdot\}$, this theorem is stated for a world $\mathcal{W}_{\mathtt{of}}$ containing the representations of object-language typing hypotheses. The world declaration is critical for equivalence with the informal statement in *Theorem 4.8*—for example, if this theorem were stated for the world $\{\cdot\}$ like preservation, the LF statement would only adequately correspond to a weaker theorem about object-language typing derivations in the empty context. Moreover, the proof we give below requires the more general inductive hypothesis, so it would not suffice to prove the weaker theorem, even if that were the informal theorem of interest.

The statement of *Theorem 4.9* is almost in the form supported by Twelf, but the condition $\Gamma, \mathtt{y} : \mathtt{tm}, \mathtt{dy} : \mathtt{of}\ \mathtt{y}\ \mathtt{T_0} \vdash \mathtt{D} \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$ is in a context other than Γ, which is not permitted. However, it is simple to put it in the correct form by abstracting over the extra variables $\mathtt{y}$ and $\mathtt{dy}$, premising the theorem on a term of higher LF type. The revised theorem is as follows:

Theorem 4.10 (*Revised Statement of Strengthening*)
For all $\Gamma \in \mathcal{W}_{\mathtt{of}}$, for all $\mathtt{E}$ such that $\Gamma \vdash \mathtt{E} \Leftarrow \mathtt{tm}$, $\mathtt{T}$ and $\mathtt{T_0}$ such that $\Gamma \vdash \mathtt{T}, \mathtt{T_0} \Leftarrow \mathtt{tp}$, and $\mathtt{D}$ such that $\boxed{\Gamma \vdash \mathtt{D} \Leftarrow \Pi\, \mathtt{y}{:}\mathtt{tm}.\, \mathtt{of}\ \mathtt{y}\ \mathtt{T_0} \rightarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}}$, there exists a $\mathtt{D'}$ such that $\Gamma \vdash \mathtt{D'} \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$.

4.3.1 Twelf Proof

This $\forall\exists$-statement is translated into the Twelf type family, mode, and worlds declaration in Figure 16. The only subtlety in the type family and mode declarations is that $\mathtt{T_0}$, $\mathtt{E}$, and $\mathtt{T}$ are tacitly universally quantified at the outside by Twelf's implicit

```
strengthen : ({y : tm} of y T0 -> of E T)
              -> of E T
              -> type.
%mode strengthen +X1 -X2.

str-e : strengthen
         ([y] [dy : of y T0] of_empty)
         of_empty.

str-a : strengthen
         ([y] [dy : of y T0]
            (of_app
               ((Dof1 : {y} of y T0 -> of E1 (arrow T2 T)) y dy)
               ((Dof2 : {y} of y T0 -> of E2 T2) y dy)))
         (of_app Dof1' Dof2')
         <- strengthen Dof1 (Dof1' : of E1 (arrow T2 T))
         <- strengthen Dof2 (Dof2' : of E2 T2).

str-l : strengthen
         ([y] [dy : of y T0]
            (of_lam ([x] [dx : of x T2]
                        (Dof : {y} of y T0 -> {x} of x T2 -> of (E x) T)
                        y dy x dx)))
         (of_lam Dof')
         <- ({x} {dx : of x T2}
               strengthen ([y] [dy : of y T0] Dof y dy x dx)
                          ((Dof' : {x} of x T2 -> of (E x) T) x dx)).

%block ofblock : some {T : tp} block {x : tm} {dx : of x T}.
%worlds (ofblock) (strengthen _ _).
%total D (strengthen D _).
```

Fig. 16. Incomplete First Attempt to Prove Strengthening for the STLC

arguments convention. The `%block` and `%worlds` declaration are Twelf's concrete syntax for defining the world $\mathcal{W}_{\mathtt{of}}$. The block declaration defines a context block called `ofblock` containing the declarations $\mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{T}$ for some term $\mathtt{T} \Leftarrow \mathtt{tp}$. The `%worlds` declaration states `strengthen` for the world of LF contexts containing any number of `ofblock`s. This set of contexts corresponds exactly to those for which the judgement $\gamma\ \mathtt{ctx} \gg \Gamma\ \mathtt{ctx}$ is derivable.

Figure 16 contains an incomplete first proof attempt; it is instructive to see where this proof fails, and the modification necessary to correct the proof is small. The constants `str-e`, `str-a`, and `str-l` give an inductive definition of the relation `strengthen`. The case `str-e` is simple: when the open typing derivation was derived by the rule `of_empty`, the conclusion can be derived by `of_empty` because this constant does not mention the variables (this corresponds to the fact that the on-paper rule OF_EMPTY is stated for an arbitrary context γ). The case `str-a` applies when the open derivation was constructed with the rule `of_app`, in which case the subderivations `Dof1` and `Dof2` can potentially mention `y` and `dy`. However, because

`y` is not free in `(app E1 E2)`, it is not free in the subterms, so by induction on each subderivation (two `strengthen` premises to the constant), we can create derivations of the same facts that do not mention `y` and `dy`; then reapplying the constant `of_app` gives the result.

The case `str-l` is slightly more involved because it passes under a binder. It is helpful to first consider how this case would proceed on paper. The input is a derivation of

$$\frac{\gamma, \mathtt{y} : \tau_0, \mathtt{x} : \tau_2 \vdash \mathtt{e} : \tau}{\gamma, \mathtt{y} : \tau_0 \vdash \lambda\, \mathtt{x}{:}\tau_2.\, \mathtt{e} : \tau_2 \to \tau}\ \textsc{of\_lam}$$

where $\mathtt{y} \# \lambda\, \mathtt{x}{:}\tau_2.\, \mathtt{e}$. Under this condition, $\mathtt{y} \# \mathtt{e}$ as well, so we can use exchange for the object language typing judgement to permute the assumptions $\mathtt{y} : \tau_0$ and $\mathtt{x} : \tau_2$ and then appeal to the inductive hypothesis on a derivation in the extended context $\Gamma, \mathtt{x} : \tau_2, \mathtt{y} : \tau_0$ to give a derivation of $\gamma, \mathtt{x} : \tau_2 \vdash \mathtt{e} : \tau$. Then reapplying OF_LAM gives the result.

The Twelf case is precisely analogous. When the last rule applied was OF_LAM, the subderivation `Dof` in an extended context is represented by an LF term of function type `{y} of y T0 -> {x} of x T2 -> of (E x) T`. Observe that the arguments to the function are the encoding of the assumptions (other than γ) in the premise of OF_LAM. Because the term `lam ([x] E x)` is well-formed without `y` in the context, `E` can only mention the free variable `x`. The inductive call to strengthen takes place in a context extended with $\mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{T}_2$; this is represented by giving the constant `str-l` a higher-order premise that abstracts over these two variables. Observe that this context extension stays in $\mathcal{W}_{\mathtt{of}}$; if this were not the case, Twelf would report an error, signaling that the inductive hypothesis of the totality assertion for this relation, which is stated only for contexts in $\mathcal{W}_{\mathtt{of}}$, would not apply. The use of exchange is implicit in the fact that the strengthening variables for the inductive call are bound inside this dependent function type. The inductive call outputs a derivation `Dof'` that can mention only the bound variables `x` and `dx`, and then applying the constant `of_lam` gives the result.

Though each of these three cases is correct, they do not satisfy the totality assertion defined by the mode and worlds declarations. Specifically, this relation does not cover the typing derivation given by a variable `dx` in the context—there is no inhabitant of the type `strengthen ([y] [dy] dx) D` for variables `dx`. In an informal proof, this is the case for OF_VAR: assume $\mathtt{y} \# \mathtt{x}$ and $\gamma, \mathtt{y} : \tau_0 \vdash \mathtt{x} : \tau$, then the variable `x` must be bound in γ, so OF_VAR derives the conclusion.

It may not be immediately obvious how to formalize a case for a variable—where can one even mention an arbitrary LF variable `dx` from the context? In Twelf, such cases can be covered by putting the case for theorem *in the context itself*. Proofs of metatheorems are simply constants of particular types, so the standard apparatus of hypotheses can be deployed to cover cases for variables. For this theorem, whenever we assume a typing derivation `dx`, we also assume a case of `strengthen` for it. Instead of proving the theorem for contexts in $\mathcal{W}_{\mathtt{of}}$, we prove it for a world $\mathcal{W}_{\mathtt{str}}$, which contains contexts of the form

$$\mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{T}_2, \mathtt{dstr} : \Pi\, \mathtt{T}_0{:}\mathtt{tp}.\ \mathtt{strengthen}\ (\lambda\, \mathtt{y}.\, \lambda\, \mathtt{dy}.\, \mathtt{dx})\ \mathtt{dx}, \ldots$$

```
strengthen : ({y : tm} of y T0 -> of E T)
             -> of E T
             -> type.
%mode strengthen +X1 -X2.

%% str-e and str-a are unchanged

str-l : strengthen
         ([y] [dy] (of_lam ([x] [dx] Dof x dx y dy)))
         (of_lam Dof')
         <- ({x} {dx : of x T2}
               {_ : {T0:tp} strengthen ([y] [dy : of y T0] dx) dx}
               strengthen ([y] [dy] Dof x dx y dy) (Dof' x dx)).

%block strblock : some {T : tp}
                  block {x : tm} {dx : of x T}
                  {_ : {T0:tp} strengthen ([y] [dy : of y T0] dx) dx}.

%worlds (strblock) (strengthen _ _).
%total D (strengthen D _).
```

Fig. 17. Completed Proof of Strengthening for the STLC

As in the informal proof, the variable typing derivation `dx` is the necessary output in this case. Another way to understand these contexts is to recall that an LF variable stands for all possible substitution instances—by insisting that contexts have this particular form, we are restricting the substitution instances for `x` and `dx` to those for which strengthening holds.

Figure 17 contains the completed proof. The theorem statement has changed according to the above discussion: the block `strblock` declares blocks of the above form, and the worlds declaration states the theorem for the world generated by this block. The theorem case in the block defines the relation `strengthen` for the canonical forms that were not covered in the previous attempt. Only one other change to the previous proof is necessary: the inductive call in `str-l` adds the `strengthen` case to the context to stay in the world for which the theorem is stated.

4.3.2 Correctness of the Statement of Strengthening

Let Σ' stand for the extension of the signature Σ from Section 3 with the constants for `strengthen`. The above Twelf proof implies the following statement of strengthening, which is stated in LF[Σ'].

Theorem 4.11 (*Statement of Strengthening in $\mathcal{W}_{\mathtt{str}}$*)
For all $\Gamma \in \mathcal{W}_{\mathtt{str}}$, for all $\mathtt{E}$ such that $\Gamma \vdash \mathtt{E} \Leftarrow \mathtt{tm}$, $\mathtt{T}$ and $\mathtt{T}_0$ such that $\Gamma \vdash \mathtt{T}, \mathtt{T}_0 \Leftarrow \mathtt{tp}$, and $\mathtt{D}$ such that $\Gamma \vdash \mathtt{D} \Leftarrow \Pi\, \mathtt{y}{:}\mathtt{tm}.\, \mathtt{of}\ \mathtt{y}\ \mathtt{T}_0 \to \mathtt{of}\ \mathtt{E}\ \mathtt{T}$, there exists a $\mathtt{D}'$ such that $\Gamma \vdash \mathtt{D}' \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$.

```
weaken : {T0} of E T -> ({x} (of x T0) -> of E T) -> type.
%mode weaken +X1 +X2 -X3.

- : weaken T0 D ([x] [dx] D).

%worlds (ofblock) (weaken _ _ _).
%total {} (weaken _ _ _).
```

Fig. 18. Direct Twelf Proof of Weakening for the STLC

This theorem differs from our desired theorem statement (*Theorem 4.10*) in two ways: it is stated for an extended signature Σ' and for a different world $\mathcal{W}_{\mathtt{str}}$. Thus, it is possible that we successfully proved the wrong theorem. For example, *Theorem 4.11* might not cover all object-language typing derivations, or it might assume some additional typing derivations that have no informal counterpart. To assuage these concerns, we should check that *Theorem 4.11* implies *Theorem 4.10*. Because the only differences between these two theorems are the LF signature and world, it should not be surprising that the content of this theorem is a transport of adequacy result:

Lemma 4.12 (*Transport of Adequacy for* `strengthen`)

1. $\Sigma'|_{\mathtt{of}} = \Sigma|_{\mathtt{of}}$.
2. For all $\Gamma' \in \mathcal{W}_{\mathtt{of}}$, there exists a $\Gamma \in \mathcal{W}_{\mathtt{str}}$ such that $\Gamma|_{\mathtt{of}}^{\preceq_{\Sigma'}} = \Gamma'$.

Proof
Observe that $\preceq_{\Sigma'}$ adds the edges $\mathtt{tm} \preceq_{\Sigma'} \mathtt{strengthen}$, $\mathtt{tp} \preceq_{\Sigma'} \mathtt{strengthen}$, and $\mathtt{of} \preceq_{\Sigma'} \mathtt{strengthen}$ to $\preceq_{\Sigma}$. The first part is true by calculation. The second part can be proved by a simple induction on the number of blocks in Γ. For any `ofblock` of the form $\mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{T}$ there is a `strblock`

$$\mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{T}, \mathtt{strx} : \Pi\, \mathtt{T}_0{:}\mathtt{tp}.\, \mathtt{strengthen}\ (\lambda\, \mathtt{y}.\, \lambda\, \mathtt{dy}.\, \mathtt{dx})\ \mathtt{dx}$$

because the indices to the theorem case for `strengthen` mention only terms bound in the `ofblock`; the restriction of this block to `of` is the original block. □

Using this lemma and *Theorem 2.17*, the proof that *Theorem 4.11* implies *Theorem 4.10* is direct. Thus, the Twelf proof does indeed prove the informal statement that we wanted to show.

4.3.3 Contrasting Strengthening with Weakening

Whereas strengthening requires an inductive proof, weakening can be proved directly, as we show in Figure 18. The type argument `T0` is made explicit so that it can be universally quantified; otherwise, it would be existentially quantified because it would appear only in an output. The proof of `weaken` includes only one case, in which, given a derivation of $\mathtt{of}\ \mathtt{E}\ \mathtt{T}$, we introduce functions to create a derivation that is abstracted over `x` and `dx`. The fact that this LF constant is well-typed

```
%% previously proved congruence lemma for identity
id_app_cong : id E1 E1'
               -> id E2 E2'
               -> id (app E1 E2) (app E1' E2')
               -> type.
%mode id_app_cong +X1 +X2 -X3.
%worlds () (id_app_cong _ _ _).

%% all free variables of a well-typed term must have a typing assumption
%% in the context
all_declared : ({y : tm} of (E y) T)
                -> ({y : tm} id (E y) E')
                -> type.
%mode all_declared +X1 -X2.

- : all_declared ([y] of_app (D1 y) (D2 y)) DidApp
     <- all_declared D1 (Did1 : {y:tm} id (E1 y) E1')
     <- all_declared D2 (Did2 : {y:tm} id (E2 y) E2')
     <- ({y : tm} id_app_cong (Did1 y) (Did2 y) (DidApp y)).

...

%worlds (ofblock) (all_declared _ _).
%total D (all_declared D _).
```

Fig. 19. Excerpt of Twelf Proof of Motivating Example for World Subsumption

corresponds with weakening being admissible in LF: the use of the variable D is insensitive to which other variables are in the LF context, so we may derive

$$\ldots, \mathtt{D} : \mathtt{of}\ \mathtt{E}\ \mathtt{T}, \mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of}\ \mathtt{x}\ \mathtt{T}_0 \vdash \mathtt{D} \Rightarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$$

by ATOM_TERM_VAR, and then use CANON_TERM_ATOM and CANON_TERM_LAM to introduce the functions.

It may at first seem curious that there is such a direct proof of weakening but not of strengthening—why does strengthening for LF not give the result? In this instance, strengthening for LF would prove the following:
If $\Gamma, \mathtt{y} : \mathtt{tm}, \mathtt{dy} : \mathtt{of}\ \mathtt{y}\ \mathtt{T}_0 \vdash \mathtt{D} \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$ and neither y nor dy are free in E or D, then $\Gamma \vdash \mathtt{D} \Leftarrow \mathtt{of}\ \mathtt{E}\ \mathtt{T}$.
That is, strengthening in LF only allows the variables to be dropped when they are known not to be free *in the derivation* D. In the statement of strengthening in *Theorem 4.9*, the variable y is not free in the term E, but both variables may appear in the derivation D. This stronger statement of strengthening requires an inductive proof even when the weaker version does not; indeed, one can imagine object languages where the weaker statement is true but the stronger one is not.

4.4 World Subsumption

4.4.1 Motivating Example

When proving theorems that are stated for a non-trivial LF world, it is common to want to use a lemma that is stated for one world in the proof of a theorem stated for another. We illustrate this with the following (somewhat contrived) theorem:

Theorem 4.13
If $\gamma \vdash \mathtt{e} : \tau$, where $\mathcal{X}, \mathtt{y}, \mathcal{X}' \vdash \mathtt{e}\ \mathsf{term}$ but $\mathtt{y}$ is not in γ, then $\mathtt{y} \# \mathtt{e}$.

That is, all free variables in a well-typed term must be declared in the context.

Figure 19 contains an excerpt of a Twelf proof of this theorem (see the companion Twelf code for the remaining cases, which are simple but not necessary for the discussion in this section). The premise that $\gamma \vdash \mathtt{e} : \tau$, where `y` is potentially free in `e` but is not declared in γ, is represented by the input LF term of type `({y : tm} of (E y) T)`—the term `E` may potentially mention the variable `y`, but there is no typing derivation for `y` given as a hypothesis. The conclusion, that $\mathtt{y} \# \mathtt{e}$ is represented by the output LF term of type `({y : tm} id (E y) E')`—the term `E`, which may mention the variable `y`, is α-equivalent to a term $\mathtt{E}'$ in which `y` does not occur (recall the definition of `id` in Section 4.2). To adequately represent the informal statement, which is stated for arbitrary object-language contexts γ, the Twelf statement is stated for the world consisting of `ofblock`s.

The excerpt of the proof shows the case when the typing derivation was derived using `of_app`. Appealing to the inductive hypothesis on the two subderivations shows that `y` is not free in either `E1` (with evidence `Did1`) or `E2` (with evidence `Did2`), and we must use these facts to show that it is not free in `app E1 E2`. To do so, the case appeals to the congruence lemma `id_app_cong` proved in Figure 15 (for reference, the statement of this lemma is reiterated at the top of Figure 19). This call is in an extended context binding the variable `y`.

Should this call be allowed? The current theorem, `all_declared`, is stated for the world $\mathcal{W}_{\mathtt{of}}$, whereas the lemma, `id_app_cong`, was proved in the world $\{\cdot\}$. Unfortunately, a lemma proved for one world may not be true in another, and, in this case, there is indeed a problem. The world $\mathcal{W}_{\mathtt{of}}$ contains additional canonical forms of type `tm` that are not present in $\{\cdot\}$, so it is possible that there are input derivations of $\mathtt{id}\,\mathtt{E}\,\mathtt{E}'$ in $\mathcal{W}_{\mathtt{of}}$ that `id_app_cong` is not prepared to handle. More formally, the totality assertion for `id_app_cong` is the following:

For all $\Gamma \in \{\cdot\}$, if $\Gamma \vdash \mathtt{D1} \Leftarrow \mathtt{id}\,\mathtt{E1}\,\mathtt{E1}'$ and $\Gamma \vdash \mathtt{D2} \Leftarrow \mathtt{id}\,\mathtt{E2}\,\mathtt{E2}'$ then there exist `D3` and `D` such that $\Gamma \vdash \mathtt{D3} \Leftarrow \mathtt{id}\,(\mathtt{app}\,\mathtt{E1}\,\mathtt{E2})\,(\mathtt{app}\,\mathtt{E1}'\,\mathtt{E2}')$ and
$\Gamma \vdash \mathtt{D} \Leftarrow \mathtt{id\_app\_cong}\,\mathtt{D1}\,\mathtt{D2}\,\mathtt{D3}$.

Appealing to this theorem from $\mathcal{W}_{\mathtt{of}}$ instantiates Γ with a context other than $\cdot$. This is impermissible, absent some additional reasoning justifying that the theorem still holds in the extended world.

To solve this problem, we can prove `id_app_cong` in the larger world. Intuitively, we need to show that the application congruence rule for `id` is admissible not just on closed terms, but also on open terms. Fortunately, our original proof does

```
id_app_cong : id E1 E1'
              -> id E2 E2'
              -> id (app E1 E2) (app E1' E2')
              -> type.
%mode id_app_cong +X1 +X2 -X3.

- : id_app_cong refl refl refl.

%worlds (ofblock) (id_app_cong _ _ _).
%total {} (id_app_cong _ _ _).
```

Fig. 20. Twelf Proof of Lemma `id_app_cong` in $\mathcal{W}_{\mathtt{of}}$

indeed prove the stronger theorem—in Section 4.2 we were concerned only with closed terms, so we did not consider the extended theorem statement. This can be expressed with the Twelf code in Figure 20. The only difference from the proof in Figure 15 is the worlds declaration, which states the theorem for $\mathcal{W}_{\mathtt{of}}$. The proof goes through as before.

Unfortunately, this revised theorem is still insufficient for two reasons. The first is that the call to it from the proof of `all_declared` is not in $\mathcal{W}_{\mathtt{of}}$, as the extended context contains a variable `y` but no typing derivation. This problem can be solved by revising the case of the theorem to add an unused typing hypothesis for `y` to the context. The more serious issue is one of modularity: the statement and proof of `id_app_cong` have nothing to do with typing derivations, but yet the lemma needed to be stated for a world containing typing derivations to support its later use. This issue is analogous to modularity issues we encountered with adequacy proofs, and we can deploy the same machinery to solve it.

4.4.2 Definition of World Subsumption

Calling a lemma stated for a world $\mathcal{W}$ from a theorem stated for a world $\mathcal{W}'$ is not always permissible: it is possible that $\mathcal{W}'$ contains additional canonical forms that the lemma is not prepared to handle (as in the above example), or that the lemma outputs canonical forms in $\mathcal{W}$ that do not exist in $\mathcal{W}'$. In either case, the lemma proved in $\mathcal{W}$ may no longer be true in $\mathcal{W}'$. Fortunately, the same techniques that we used to transfer adequacy from one world to another can be used to transfer proofs of metatheorems from one world to another—the key issue in both cases is the characterization of the relevant canonical forms. In particular, we define an order on worlds as follows, where the relation $\Gamma \sim \Gamma'$ is identity modulo permutation of independent hypotheses:

Definition 4.14 (*World Subsumption*)
$\mathcal{W} \lesssim_{\mathtt{a}} \mathcal{W}'$ iff for all $\Gamma' \in \mathcal{W}'$, there exists a $\Gamma \in \mathcal{W}$ such that $\Gamma|_{\mathtt{a}} \sim \Gamma'|_{\mathtt{a}}$.

Because the proof that one world subsumes another may be non-trivial, Twelf implements a conservative but useful approximation of the relation $\mathcal{W} \lesssim_{\mathtt{a}} \mathcal{W}'$.

The intuitive justification for choosing this orientation of the relation $\lesssim_{\mathtt{a}}$ is that world subsumption is a sufficient condition for weakening the world of a totality assertion (i.e., for transporting a theorem proved in a smaller world to a larger one):

Theorem 4.15 (*Totality Assertions Remain True in Larger Worlds*)
Assume an LF constant $\mathtt{a} \Rightarrow \Pi\,\mathtt{x}_1{:}\mathtt{A}_1 \ldots \Pi\,\mathtt{y}_1{:}\mathtt{B}_1 \ldots .\,\mathtt{type}$ that satisfies the following totality assertion:

$$\begin{gathered}\text{For all } \Gamma \in \mathcal{W}, \text{ for all } \mathtt{M}_1 \text{ such that } \Gamma \vdash \mathtt{M}_1 \Leftarrow \mathtt{A}'_1, \ldots, \text{ and } \mathtt{M}_\mathtt{m} \text{ such that} \\ \Gamma \vdash \mathtt{M}_\mathtt{m} \Leftarrow \mathtt{A}'_\mathtt{m}, \text{ there exist } \mathtt{N}_1 \text{ such that } \Gamma \vdash \mathtt{N}_1 \Leftarrow \mathtt{B}'_1, \ldots, \text{ and } \mathtt{N}_\mathtt{n} \text{ such that} \\ \Gamma \vdash \mathtt{N}_\mathtt{n} \Leftarrow \mathtt{B}'_\mathtt{n}, \text{ and } \mathtt{D} \text{ such that } \Gamma \vdash \mathtt{D} \Leftarrow \mathtt{a}\,\mathtt{M}_1 \ldots \mathtt{M}_\mathtt{m}\,\mathtt{N}_1 \ldots \mathtt{N}_\mathtt{n}.\end{gathered}$$

where

$$\begin{gathered}[\mathtt{M}_{\mathtt{i}-1}/\mathtt{x}_{\mathtt{i}-1}]_{\mathtt{A}_{\mathtt{i}-1}} \ldots [\mathtt{M}_1/\mathtt{x}_1]_{\mathtt{A}_1}\mathtt{A}_\mathtt{i} = \mathtt{A}'_\mathtt{i} \\ [\mathtt{N}_{\mathtt{j}-1}/\mathtt{y}_{\mathtt{j}-1}]_{\mathtt{B}_{\mathtt{j}-1}} \ldots [\mathtt{N}_1/\mathtt{y}_1]_{\mathtt{B}_1}[\mathtt{M}_\mathtt{m}/\mathtt{x}_\mathtt{m}]_{\mathtt{A}_\mathtt{m}} \ldots [\mathtt{M}_1/\mathtt{x}_1]_{\mathtt{A}_1}\mathtt{B}_\mathtt{j} = \mathtt{B}'_\mathtt{j}.\end{gathered}$$

Then for all $\mathcal{W}'$ such that $\mathcal{W} \lesssim_{\mathtt{thm}} \mathcal{W}'$, the totality assertion for $\mathcal{W}'$ is also true.

Proof
The proof is direct using *Theorem 2.17*, *Theorem 2.11*, and two additional lemmas:

1. If $\Gamma \sim \Gamma'$ then $\Gamma|_{\mathtt{a}} \sim \Gamma'|_{\mathtt{a}}$.
2. If $\Gamma \vdash \mathtt{M} \Leftarrow \mathtt{A}$ and $\Gamma \sim \Gamma'$ then $\Gamma' \vdash \mathtt{M} \Leftarrow \mathtt{A}$.

The proof of the second part uses exchange (*Lemma 2.7*). □

Intuitively, the condition $\mathcal{W} \lesssim_{\mathtt{a}} \mathcal{W}'$ ensures that all canonical forms of type $\mathtt{a}$ in $\mathcal{W}'$ also exist in $\mathcal{W}$. This condition is necessary to use a lemma proved for $\mathcal{W}$ from a theorem stated for $\mathcal{W}'$, as it ensures that $\mathcal{W}'$ does not add additional inputs that the lemma is not prepared to handle. However, the condition $\mathcal{W} \lesssim_{\mathtt{a}} \mathcal{W}'$ does not imply that all of the canonical forms of type $\mathtt{a}$ in $\mathcal{W}$ exist in $\mathcal{W}'$. This may intuitively seem problematic for the above theorem—what if the lemma returns a canonical form in $\mathcal{W}$ that does not exist in $\mathcal{W}'$? The lemma cannot return such a derivation because totality assertions quantify over the context Γ at the outside: when the lemma is given canonical inputs in a particular Γ, it returns canonical outputs in the same Γ.

To understand *Theorem 4.15*, it may be helpful to consider an example in which it does not apply. Recall the transport of adequacy theorem for strengthening, *Lemma 4.12* above. The second part of this lemma is equivalent to stating that $\mathcal{W}_{\mathtt{str}} \lesssim_{\mathtt{of}} \mathcal{W}_{\mathtt{of}}$. Moreover, the proof that the statement of strengthening in $\mathcal{W}_{\mathtt{str}}$ (*Theorem 4.11*) implies the statement of strengthening in $\mathcal{W}_{\mathtt{of}}$ (*Theorem 4.10*) is similar to the proof of *Theorem 4.15*. However, we cannot use *Theorem 4.15* to give an alternate proof of this theorem. *Lemma 4.12* proves $\mathcal{W}_{\mathtt{str}} \lesssim_{\mathtt{of}} \mathcal{W}_{\mathtt{of}}$, but it does not prove $\mathcal{W}_{\mathtt{str}} \lesssim_{\mathtt{strengthen}} \mathcal{W}_{\mathtt{of}}$, which would be necessary to satisfy the premise of *Theorem 4.15* (in fact, it is not true that $\mathcal{W}_{\mathtt{str}} \lesssim_{\mathtt{strengthen}} \mathcal{W}_{\mathtt{of}}$). In this example, the $\forall\exists$-statement of strengthening transfers from $\mathcal{W}_{\mathtt{str}}$ to $\mathcal{W}_{\mathtt{of}}$, as we proved above, but the totality assertion for `strengthen` does not—precisely because the contexts in $\mathcal{W}_{\mathtt{str}}$ contribute proof cases to `strengthen`.

```
id_app_cong : id E1 E1'
              -> id E2 E2'
              -> id (app E1 E2) (app E1' E2')
              -> type.
%mode id_app_cong +X1 +X2 -X3.

- : id_app_cong refl refl refl.

%block tmblock : block {x : tm}.
%worlds (tmblock) (id_app_cong _ _ _).
%total {} (id_app_cong _ _ _).
```

Fig. 21. Twelf Proof of Lemma `id_app_cong` in $\mathcal{W}_{\mathtt{tm}}$

4.4.3 Using World Subsumption

The proof of `all_declared` at the beginning of this section required a lemma asserting that identity is a congruence on open terms. The correct statement and proof of this lemma is presented in Figure 21. The theorem must be stated for the world containing term variables (call this world $\mathcal{W}_{\mathtt{tm}}$) to account for open terms. However, unlike the previous statement, this theorem does not mention typing derivations, which are irrelevant to this particular lemma and its proof. It is easy to show that $\mathcal{W}_{\mathtt{tm}} \lesssim_{\mathtt{id\_app\_cong}} \mathcal{W}_{\mathtt{of}}$: `of` $\not\preceq$ `id_app_cong`, so for any context Γ of the form $\mathtt{x} : \mathtt{tm}, \mathtt{dx} : \mathtt{of\ x\ T}, \ldots$, it is true that $\Gamma|_{\mathtt{id\_app\_cong}} \in \mathcal{W}_{\mathtt{tm}}$. Thus, *Theorem 4.15* implies that `id_app_cong` holds in $\mathcal{W}_{\mathtt{of}}$ as well. This authorizes the call from the proof of `all_declared` in Figure 19.

Observe that $\{\cdot\} \not\lesssim_{\mathtt{id\_app\_cong}} \mathcal{W}_{\mathtt{of}}$ because `tm` $\preceq$ `id_app_cong`, so *Theorem 4.15* does not permit the theorem to call the weaker version of the lemma, which was stated only for the empty context. Of course, had we anticipated needing the open-term congruence lemma, we could have proved that lemma in the first place, as it includes closed terms as a special case. It is worth noting that, though these examples have to do with worlds, the general principle has nothing to do with Twelf: in a paper proof, one might choose to prove a weaker version of a lemma for use at one point in a proof, and then, later in the proof, discover that a stronger version is necessary.

The world subordination criterion emphasizes that transport of canonical forms is important not just for modular adequacy proofs, but also for modular Twelf proofs.

4.5 Discussion

Though Twelf *proofs* are mechanically checked, a programmer still must verify that a Twelf theorem *statement* corresponds to the informal statement of the theorem he wants to prove. As we have seen in this section, the adequacy methodology permits a precise account of the correspondence between informal and mechanized theorem statements. That said, a Twelf programmer does not need to make this correspondence for every theorem he proves. In a large Twelf proof, there are usually

a few top-level theorems stating the major results about a system (e.g., progress and preservation for a language) that are proved using many lemmas. In this case, it is necessary only to verify that the top-level theorem statements correspond to the desired informal theorems; there is no need to consider the informal analogues of lemmas that are not of independent interest.

5 Related Work

A reader who is interested in learning more about Twelf should visit the Twelf Wiki (`http://twelf.plparty.org/`), which contains numerous tutorials and case studies.

LF and Twelf. Harper *et al.* (1993) introduce the LF type theory and representation methodology. The canonical-forms presentation of LF in this article is due to the treatment of Concurrent LF by Watkins *et al.* (2002, 2004a); it has since been applied to several other type theories (for example, see Nanevski *et al.* (2007); Nanevski & Morrisett (2006); Lee *et al.* (2007)). The general method of defining a type theory algorithmically goes back to the AUTOMATH languages (de Bruijn, 1993; van Daalen, 1980). Previous treatments of the metatheory of LF include the following: Harper *et al.* (1993) discuss the metatheory of LF with β-conversion but not η-conversion. Salvesen (1990), Geuvers (1992), and Goguen (1999) discuss the extension to $\beta\eta$-conversion. Coquand (1991) gives a shape-directed equality algorithm for LF. Harper & Pfenning (2005) present a type-directed equality algorithm. Felty (1991) presents LF with formation judgements that admit only canonical forms, but defines equality using β-reduction on non-canonical forms.

Several logical frameworks extend LF with additional features. Linear LF (LLF) extends LF with linear connectives (Cervesato & Pfenning, 2002). Concurrent LF (CLF) extends LLF with more linear connectives and with an intrinsic notion of concurrency (Watkins *et al.*, 2002, 2004b; Cervesato *et al.*, 2002). These extensions ease the representation of language features for which LF provides no native support—e.g., a language with state can conveniently be represented using the linear connectives in LLF.

The Twelf User's Guide (Pfenning & Schürmann, 2002) describes the features of Twelf. The metatheory of Twelf is discussed in several sources. Pfenning (1991) introduces the logic-programming operational semantics for LF. Pfenning & Rohwedder (1992) give an early overview of the Twelf methodology. Rohwedder & Pfenning (1996) discuss mode and termination checking. Pientka & Pfenning (2000) discuss an extended termination checker. Schürmann (2000) and Schürmann & Pfenning (2003) discuss coverage and world checking.

Several extensions of Twelf have been studied but not yet fully implemented. Schürmann & Pfenning (1998) discuss a metatheorem prover. Anderson & Pfenning (2004) discuss uniqueness checking. Schürmann *et al.* (2005) describe an extended metatheorem language that would permit more general theorems than $\forall\exists$-statements. Reed (2006) describes an approach to extending the metatheoretic capabilities of Twelf to Linear LF.

Applications of LF and Twelf. Several papers discuss applications of LF and Twelf. Lee *et al.* (2007) discuss a Twelf proof of type safety for the Harper-Stone internal language for Standard ML. Avron *et al.* (1989) discuss LF representation and present several examples. Michaylov & Pfenning (1991) present some of the metatheory of Mini-ML. Pfenning (1992) gives a proof of the Church-Rosser theorem for the simply typed λ-calculus. Pfenning (1994) gives a proof of cut elimination for intuitionistic logic. Schürmann *et al.* (2001) present some of the metatheory of F^{ω}. Schürmann & Stehr (2005) present a formalization of Howe's embedding of HOL into NuPRL. Appel (2001) presents a foundational proof-carrying code system. Appel & Felty (2002) use Twelf to implement tactical theorem provers. Crary & Sarkar (2003) present a typed assembly language and applications to proof-carrying code. Simmons (2005) presents the metatheory of a functional language with references. Pfenning (1999) surveys various methods of representing deductive systems in logical frameworks. The `examples` directory of the Twelf distribution contains additional examples of deductive systems and their metatheory.

Though they do not focus on mechanization, several additional papers are accompanied by Twelf appendices mechanizing the metatheory of the presented languages. Crary (2003) formalizes a foundational typed assembly language. Murphy VII *et al.* (2004) and Murphy VII *et al.* (2005) formalize modal type systems for distributed computing. Licata & Harper (2005) formalize a language extending ML with a form of dependent types. Garg & Pfenning (2006) formalize an authorization logic. Acar *et al.* (2007) prove consistency and correctness of a semantics for self-adjusting computation. Fluet *et al.* (2006) formalize a substructural type system for region-based memory management.

Other Mechanization Tools. The LF logical framework is similar in spirit to the AUTOMATH languages (Nederpelt *et al.*, 1994)—both provide frameworks for representing machine-checkable mathematical arguments, while neither make any foundational commitment about what logics can be represented.

Other proof assistants that have been used to formalize mathematics and computer science include Coq (Coq Development Team, 2007; Bertot & Castéran, 2004), Isabelle/HOL (Nipkow *et al.*, 2002), NuPRL (Constable *et al.*, 1986), HOL Light (Harrison, 1996), and ACL2 (Kaufmann *et al.*, 2000). Many of these proof assistants have been applied to the domain of programming languages and logics. For example, Miculan (1997) shows how to encode several logical systems in Coq; Leroy (2006) reports on implementing a certified compiler in Coq. Klein & Nipkow (2006) formalize a semantics for a Java-like language in Isabelle/HOL.

Twelf, Coq, Isabelle/HOL, and other proof assistants differ significantly in the representation techniques they support, the classes of metatheorems that can be proved, the style in which proofs are written, and the degree of automation they provide. A detailed analysis of the trade-offs between these tools is beyond the scope of this article. In the literature, several efforts have been made to compare and contrast proofs of the same theorem formalized in different proof assistants. For example, the POPLmark Challenge (Aydemir *et al.*, 2005) is one benchmark on which proof assistants can be compared; solutions have been presented in Twelf,

Coq, Isabelle/HOL, and other systems. Appel & Leroy (2006) compare Coq and Twelf proofs of a first-order list-machine benchmark.

6 Conclusion

In this article, we have surveyed the LF λ-calculus, the LF methodology for representing languages, and the Twelf methodology for mechanizing metatheory. Following Watkins *et al.* (2002), we have shown how LF, a minimal dependent type theory, can be defined so that only canonical forms are well-typed, giving a direct inductive definition of the canonical forms. We have shown how the minimality of LF enables higher-order representations of syntax and judgements, whereby an object language inherits α-equivalence, capture-avoiding substitution, and properties of hypothetical judgements from the logical framework. We have shown how these representations are proved adequate by simple structural inductions on the informal object-language entities and on the canonical forms of LF; and we have shown how subordination-based transport of adequacy facilitates modular adequacy proofs. Finally, we have shown how the proofs of $\forall\exists$-statements over LF types can be represented relationally within LF and mechanically verified by Twelf, again using subordination and induction on canonical forms.

There are several important and useful features of Twelf that we have not discussed in this article. Chiefly, Twelf implements a logic-programming operational semantics for LF (Pfenning, 1991). Using this operational semantics, one can run the LF specification of a programming language directly. For example, the type family `step` defined in Figure 13 can be run as an interpreter for the STLC, and the type family `of` can be run as a type checker. The interested reader should consult the Twelf User's Guide (Pfenning & Schürmann, 2002) or the Twelf Wiki to learn about the additional features of Twelf.

In future work, we expect that the methodologies that we have surveyed in this paper will be scaled to richer logical frameworks, to tools that provide more automated theorem proving, and to more-general metatheorem languages. Research on Linear LF and Concurrent LF has shown how how these frameworks permit facile representations of systems that are cumbersome to represent in LF (Cervesato & Pfenning, 2002; Cervesato *et al.*, 2002). The LF methodology for adequate representations scales to these richer frameworks; the programmer simply has a richer collection of types available for generating canonical forms. The extension of Twelf's logic-programming operational semantics and totality checker to these logical frameworks is current and future work (Reed, 2006).

At present, Twelf is most often used as a metatheorem checker rather than as a metatheorem prover, in the sense that most often a proof of a $\forall\exists$-statement is represented explicitly as an LF type family and then verified by a totality assertion. A metatheorem prover (i.e., an extension of Twelf with the ability to automatically generate an LF type family satisfying a given totality assertion) has been studied and implemented in a previous release of Twelf (Schürmann & Pfenning, 1998). However, the current Twelf implementation of this metatheorem prover does not output the LF type family that it discovers, limiting the feature's utility. This

metatheorem prover's implementation may eventually be completed, and the development of more-sophisticated metatheorem provers is a subject of active research.

Because Twelf metatheorems are proved using totality assertions about LF type families, the class of metatheorems that can be mechanized is restricted to $\forall\exists$-statements over LF types. On the one hand, as the successful Twelf formalizations cited in Section 5 demonstrate, these $\forall\exists$-statements have proved to be sufficient for formalizing a wide variety of metatheorems about programming languages and logics. On the other hand, we have no way to quantify when metatheorems of this form will be sufficient, and there are some well-known examples of proofs that cannot be formalized directly using Twelf's metatheorem language. For example, proofs by logical relations often require more quantifier complexity than $\forall\exists$-statements afford. In future work, we expect that this restriction may be lifted by moving to a different metatheorem language that permits more general statements (see Schürmann *et al.* (2005) for some preliminary work). Though such a metatheorem language is a departure from the specific methodology outlined in this paper, it is in essence just a different way of expressing proofs by induction on the canonical forms of LF.

In summary, we believe that the LF and Twelf methodology for mechanizing languages and their metatheory is both useful in practice today and the foundation of interesting future research. We hope that the interested reader will use LF and Twelf and assess their utility himself.

Acknoweldgements. We thank Todd Wilson, Rob Simmons, Steve Brookes, William Lovas, Tom Murphy, Jason Reed, Ruy Ley-Wild, Daniel Lee, Daniel Spoonhower, Akiva Leffert, Neel Krishnaswami, Jake Donham, Sean McLaughlin, Karl Crary, and Frank Pfenning for discussions about, and feedback on, previous drafts of this article. We thank the anonymous referees for their helpful feedback on previous drafts.

References

Acar, Umut A., Blume, Matthias, & Donham, Jacob. (2007). A consistent semantics of self-adjusting computation. *European Symposium on Programming.* Springer-Verlag.

Anderson, Penny, & Pfenning, Frank. (2004). Verifying uniqueness in a logical framework. *Pages 18–33 of:* Slind, K., Bunker, A., & Gopalakrishnan, G. (eds), *International Conference on Theorem Proving in Higher-Order Logics.* Lecture Notes in Computer Science, vol. 3223. Springer.

Appel, Andrew, & Leroy, Xavier. (2006). A list-machine benchmark for mechanized metatheory. *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice.* Electronic Notes in Theoretical Computer Science.

Appel, Andrew W. (2001). Foundational proof-carrying code. *Page 247 of: IEEE Symposium on Logic in Computer Science.* IEEE Computer Society.

Appel, Andrew W., & Felty, Amy P. (2002). Dependent types ensure partial correctness of theorem provers. *Journal of Functional Programming*, **14**(1), 3–19.

Avron, Arnon, Honsell, Furio, & Mason, Ian A. (1989). An overview of the Edinburgh Logical Framework. Birtwistle, Graham, & Subrahmanyam, P.A. (eds), *Current trends in hardware verification*. Springer Verlag.

Aydemir, Brian E., Bohannon, Aaron, Fairbairn, Matthew, Foster, J. Nathan, Pierce, Benjamin C., Sewell, Peter, Vytiniotis, Dimitrios, Washburn, Geoffrey, Weirich, Stephanie, & Zdancewic, Steve. (2005). Mechanized metatheory for the masses: The POPLmark challenge. *International Conference on Theorem Proving in Higher-Order Logics*. Springer-Verlag.

Bertot, Yves, & Castéran, Pierre. (2004). *Interactive theorem proving and program development: Coq'art: The calculus of inductive constructions*. Texts in Theoretical Computer Science. Springer.

Cervesato, Iliano, & Pfenning, Frank. (2002). A linear logical framework. *Information and Computation*, **179**(1), 19–75.

Cervesato, Iliano, Pfenning, Frank, Walker, David, & Watkins, Kevin. (2002). *A concurrent logical framework II: Examples and applications*. Tech. rept. CMU-CS-02-102. Department of Computer Science, Carnegie Mellon University. Revised May 2003.

Constable, Robert L., Allen, Stuart F., Bromley, H. M., Cleaveland, W. R., Cremer, J. F., Harper, R. W., Howe, Douglas J., Knoblock, T. B., Mendler, N. P., Panangaden, P., Sasaki, James T., & Smith, Scott F. (1986). *Implementing mathematics with the NuPRL proof development system*. Prentice Hall.

Coq Development Team. (2007). *The Coq proof assistant reference manual*. INRIA. Available from `http://coq.inria.fr/`.

Coquand, Thierry. (1991). An algorithm for testing conversion in type theory. *Pages 255–279 of:* Huet, Gérard, & Plotkin, Gordon D. (eds), *Logical frameworks*. New York, NY, USA: Cambridge University Press.

Crary, Karl. (2003). Toward a foundational typed assembly language. *Pages 198–212 of: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press.

Crary, Karl, & Sarkar, Susmit. (2003). Foundational certified code in a metalogical framework. *Pages 106–120 of: International Conference on Automated Deduction*. Springer-Verlag.

de Bruijn, Nicolaas Govert. (1993). Algorithmic definition of lambda-typed lambda calculus. *Pages 131–145 of:* Huet, Gérard, & Plotkin, Gordon D. (eds), *Logical environment*. Cambridge University Press.

Felty, Amy. (1991). Encoding dependent types in an intuitionistic logic. *Pages 214–251 of:* Huet, Gérard, & Plotkin, Gordon D. (eds), *Logical frameworks*. Cambridge University Press.

Fluet, Matthew, Morrisett, Greg, & Ahmed, Amal. (2006). Linear regions are all you need. *Pages 7–21 of: European Symposium on Programming*. Springer-Verlag.

Garg, Deepak, & Pfenning, Frank. (2006). Non-interference in constructive authorization logic. *Pages 183–293 of: Computer security foundations workshop*.

Geuvers, Herman. (1992). The Church-Rosser property for $\beta\eta$-reduction in typed λ-calculi. *Pages 453–460 of:* Scedrov, A. (ed), *IEEE Symposium on Logic in Computer Science*. IEEE Press.

Goguen, Healfdene. (1999). Soundness of the logical framework for its typed operational semantics. *International Conference on Typed Lambda Calculi and Applications*. Lecture Notes in Computer Science, vol. 1581. Springer-Verlag.

Harper, Robert, & Pfenning, Frank. (2005). On equivalence and canonical forms in the LF type theory. *ACM Transactions on Computational Logic*, **6**, 61–101.

Harper, Robert, Honsell, Furio, & Plotkin, Gordon. (1993). A framework for defining logics. *Journal of the Association for Computing Machinery*, **40**(1).

Harrison, John. (1996). HOL Light: A tutorial introduction. *Pages 265–269 of: Formal Methods in Computer-Aided Design*. Lecture Notes in Computer Science, vol. 1166. Springer-Verlag.

Kaufmann, Matt, Manolios, Panagiotis, & Moore, J Strother. (2000). *Computer-aided reasoning: An approach*. Kluwer Academic Publishers.

Klein, Gerwin, & Nipkow, Tobias. (2006). A machine-checked model for a Java-like language, virtual machine and compiler. *ACM Transactions on Programming Languages and Systems*, **28**(4), 619–695.

Lee, Daniel K., Crary, Karl, & Harper, Robert. (2007). Towards a mechanized metatheory of Standard ML. *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press.

Leroy, Xavier. (2006). Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. *Pages 42–54 of: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press.

Licata, Daniel R., & Harper, Robert. (2005). *A formulation of Dependent ML with explicit equality proofs*. Tech. rept. CMU-CS-05-178. Department of Computer Science, Carnegie Mellon University.

Michaylov, Spiro, & Pfenning, Frank. (1991). Natural semantics and some of its meta-theory in Elf. *Pages 299–344 of:* Eriksson, L.H., Hallnäs, L., & Schroeder-Heister, P. (eds), *International Workshop on Extensions of Logic Programming*. Lecture Notes in Artificial Intelligence, vol. 596. Springer-Verlag.

Miculan, Marino. (1997). *Encoding logical theories of programs*. Ph.D. thesis, Dipartimento di Informatica, Universita di Pisa.

Murphy VII, Tom, Crary, Karl, Harper, Robert, & Pfenning, Frank. (2004). A symmetric modal lambda calculus for distributed computing. *Pages 286–295 of:* Ganzinger, H. (ed), *IEEE Symposium on Logic in Computer Science*. IEEE Press.

Murphy VII, Tom, Crary, Karl, & Harper, Robert. (2005). Distributed control flow with classical modal logic. *Pages 51–69 of: Computer Science Logic*. Lecture Notes in Computer Science, vol. 3634. Springer-Verlag.

Nanevski, Aleksandar, & Morrisett, Greg. (2006). *Dependent type theory of stateful higher-order functions*. Tech. rept. TR-24-05. Harvard Computer Science.

Nanevski, Aleksandar, Pfenning, Frank, & Pientka, Brigitte. (2007). Contextual modal type theory. *ACM Transactions on Computational Logic*. To appear.

Nederpelt, R.P., Geuvers, J.H., & de Vrijer, R.C. (eds). (1994). *Selected papers on AUTOMATH*. Studies in Logic and the Foundations of Mathematics, vol. 133. North-Holland.

Nipkow, Tobias, Paulson, Lawrence C., & Wenzel, Markus. (2002). *Isabelle/HOL*

— *a proof assistant for higher-order logic*. Lecture Notes in Computer Science, vol. 2283. Springer-Verlag.

Pfenning, Frank. (1991). Logic programming in the LF logical framework. *Pages 149–181 of:* Huet, Gérard, & Plotkin, Gordon D. (eds), *Logical frameworks*. Cambridge University Press.

Pfenning, Frank. (1992). *A proof of the Church-Rosser theorem and its representation in a logical framework*. Tech. rept. CMU-CS-92-186. Department of Computer Science, Carnegie Mellon University.

Pfenning, Frank. (1994). *A structural proof of cut elimination and its representation in a logical framework*. Tech. rept. CMU-CS-94-218. Department of Computer Science, Carnegie Mellon University.

Pfenning, Frank. (1999). Logical Frameworks. Robinson, Alan, & Voronkov, Andrei (eds), *Handbook of Automated Reasoning*. Elsevier Science and MIT Press.

Pfenning, Frank, & Rohwedder, Ekkehard. (1992). Implementing the meta-theory of deductive systems. *Pages 537–551 of:* Kapur, D. (ed), *International Conference on Automated Deduction*. Lecture Notes in Artificial Intelligence, vol. 607. Springer-Verlag.

Pfenning, Frank, & Schürmann, Carsten. (1999). System description: Twelf - a meta-logical framework for deductive systems. *Pages 202–206 of:* Ganzinger, Harald (ed), *International Conference on Automated Deduction*.

Pfenning, Frank, & Schürmann, Carsten. (2002). *Twelf user's guide, version 1.4*.

Pientka, Brigitte, & Pfenning, Frank. (2000). Termination and reduction checking in the logical framework. Schürmann, Carsten (ed), *Workshop on Automation of Proofs by Mathematical Induction*.

Reed, Jason. (2006). Hybridizing a logical framework. *International workshop on hybrid logic*. Electronic Notes in Theoeretical Computer Science. Elsevier.

Rohwedder, Ekkehard, & Pfenning, Frank. (1996). Mode and termination checking for higher-order logic programs. *Pages 296–310 of:* Nielson, Hanne Riis (ed), *European Symposium on Programming*. Lecture Notes in Computer Science, vol. 1058. Springer-Verlag.

Salvesen, Anne. (1990). The Church-Rosser theorem for LF with $\beta\eta$-reduction. *Unpublished notes to a talk given at the First Workshop on Logical Frameworks*.

Schürmann, Carsten. (2000). *Automating the meta-theory of deductive systems*. Ph.D. thesis, Carnegie Mellon University.

Schürmann, Carsten, & Pfenning, Frank. (1998). Automated theorem proving in a simple meta-logic for LF. *Pages 286–300 of:* Kirchner, Claude, & Kirchner, Hélène (eds), *International Conference on Automated Deduction*. Lecture Notes in Computer Science, vol. 1421. Springer-Verlag.

Schürmann, Carsten, & Pfenning, Frank. (2003). A coverage checking algorithm for LF. *International Conference on Theorem Proving in Higher-Order Logics*. Springer-Verlag.

Schürmann, Carsten, & Stehr, Mark-Oliver. (2005). An executable formalization of the HOL/NuPRL connection in Twelf. *International conference on logic for programming artificial intelligence and reasoning*. Springer-Verlag.

Schürmann, Carsten, Yu, Dachuan, & Ni, Zhaozhong. (2001). A representation of F_ω in LF. *Electronic notes in Theoretical Computer Science*, **58**(1).

Schürmann, Carsten, Poswolsky, Adam, & Sarnat, Jeffrey. (2005). The ∇-calculus: Functional programming with higher-order encodings. *International Conference on Typed Lambda Calculi and Applications*. Springer-Verlag.

Simmons, Robert. (2005). *Twelf as a unified framework for language formalization and implementation*. Tech. rept. Princeton University. Undergraduate Senior Thesis 18679.

van Daalen, D. T. (1980). *The language theory of AUTOMATH*. Ph.D. thesis, Technical University of Eindhoven, Eindhoven, Netherlands.

Virga, Roberto. (1999). *Higher-order rewriting with dependent types*. Ph.D. thesis, Carnegie Mellon University.

Watkins, Kevin, Cervesato, Iliano, Pfenning, Frank, & Walker, David. (2002). *A concurrent logical framework I: Judgments and properties*. Tech. rept. CMU-CS-02-101. Department of Computer Science, Carnegie Mellon University. Revised May 2003.

Watkins, Kevin, Cervesato, Iliano, Pfenning, Frank, , & Walker, David. (2004a). A concurrent logical framework: the propositional fragment. *Pages 355–377 of:* Berardi, S., Coppo, M., & Damiani, F. (eds), *Types for proofs and programs*. Lecture Notes in Computer Science, vol. 3085. Springer-Verlag.

Watkins, Kevin, Cervesato, Iliano, Pfenning, Frank, & Walker, David. (2004b). Specifying properties of concurrent computations in CLF. Schürmann, Carsten (ed), *International workshop on logical frameworks and meta-languages*.