Untyped Algorithmic Equality for Martin-Löf's Logical Framework with Surjective Pairs

Andreas Abel[⋆] and Thierry Coquand

Department of Computer Science, Chalmers University of Technology
abel,coquand@cs.chalmers.se

Abstract. An untyped algorithm to test $\beta\eta$-equality for Martin-Löf's Logical Framework with strong Σ-types is presented and proven complete using a model of partial equivalence relations between untyped terms.

1 Introduction

Type checking in dependent type theories requires comparison of expressions for equality. In theories with β-equality, an apparent method is to normalize the objects and then compare their β-normal forms syntactically. In the theory we want to consider, an extension of Martin-Löf's logical framework with $\beta\eta$-equality by dependent surjective pairs (strong Σ types), which we call MLF_{Σ}, a naive *normalize and compare syntactically* approach fails since $\beta\eta$-reduction with surjective pairing is known to be non-confluent [Klo80].

We therefore advocate the incremental $\beta\eta$-convertibility test which has been given by the second author for dependently typed λ-terms [Coq91,Coq96], and extend it to pairs. The algorithm computes the weak head normal forms of the conversion candidates, and then analyzes the shape of the normal forms. In case the head symbols do not match, conversion fails early. Otherwise, the subterms are recursively weak head normalized and compared. There are two flavors of this algorithm.

Type-directed conversion. In this style, the type of the two candidates dictates the next step in the algorithm. If the candidates are of function type, both are applied to a fresh variable, if they are of pair type, their left and right projections are recursively compared, and if they are of base type, they are compared structurally, i. e., their head symbols and subterms are compared. Type-directed conversion has been investigated by Harper and Pfenning [HP05]. The advantage of this approach is that it can handle cases where the type provides extra information which is not already present in the shape of terms. An example is the unit type: any two terms of unit type, e. g., two variables, can be considered equal. Harper and Pfenning report difficulties in showing transitivity of the conversion algorithm, in case of dependent types. To circumvent this problem, they

[⋆] Research supported by the coordination action *TYPES* (510996) and thematic network *Applied Semantics II* (IST-2001-38957) of the European Union and the project *Cover* of the Swedish Foundation of Strategic Research (SSF).

erase the dependencies and obtain simple types to direct the equality algorithm. In the theory they consider, the Edinburgh Logical Framework [HHP93], erasure is sound, but in theories with types defined by cases (large eliminations), erasure is unsound and it is not clear how to make their method work. In this article, we investigate an alternative approach.

Shape-directed (untyped) conversion. As the name suggests, the shape of the candidates directs the next step. If one of the objects is a λ-abstraction, both objects are applied to a fresh variable, if one object is a pair, the algorithm continues with the left and right projections of the candidates, and otherwise, they are compared structurally. Since the algorithm does not depend on types, it is in principle applicable to many type theories with functions and pairs. In this article, we prove it complete for MLF_Σ, but since we are not using erasure, we expect the proof to extend to theories with large eliminations.

Main technical contributions of this article.

1. We extend the untyped conversion algorithm of the second author [Coq91] to a type system with Σ-types and surjective pairing. Recall that reduction in the untyped λ-calculus with surjective pairing is not Church-Rosser [Bar84] and, thus, one cannot use a presentation of this type system with conversion defined on raw terms.[1]
2. We take a modular approach for showing the completeness of the conversion algorithm. This result is obtained using a special instance of a general PER model construction. Furthermore this special instance can be described *a priori* without references to the typing rules.

Contents. We start with a syntactical description of MLF_Σ, in the style of equality-as-judgement (Section 2). Then, we give an untyped algorithm to check $\beta\eta$-equality of two expressions, which alternates weak head reduction and comparison phases (Section 3). The goal of this article is to show that the algorithmic equality of MLF_Σ is equivalent to the declarative one. Soundness is proven rather directly in Section 4, requiring inversion for the typing judgement in order to establish subject reduction for weak head evaluation. Completeness, which implies decidability of MLF_Σ, requires construction of a model. Before giving a specific model, we describe a class of PER models of MLF_Σ based on a generic model of the λ-calculus with pairs (Section 5). In Section 6 we turn to the specific model of expressions modulo β-equality, on which we define an inductive η-equality. Its transitive closure is regarded as the "universe" $\mathcal{S}$ of type interpretations, each interpretation is shown to be a subset of $\mathcal{S}$. As a consequence, two declaratively equal terms are related by $\mathcal{S}$. We complete the circle in Section 7 where we show that well-typed $\mathcal{S}$-related terms are algorithmically equal, using standardization for λ-terms. Decidability of judgmental equality on well-typed terms in MLF_Σ ensues, which entails that type checking of normal forms is decidable as well.

[1] In the absence of confluence, one cannot show injectivity of type constructors, hence subject reduction fails.

The full version of the article, which contains additionally a bidirectional type-checking algorithm for MLF_Σ and more detailed proofs, is available on the homepage of the first author [AC05].

2 Declarative Presentation of MLF_Σ

This section presents the typing and equality rules for an extension of Martin-Löf's logical framework [NPS00] by dependent pairs. We show some standard properties like weakening and substitution, as well as injectivity of function and pair types and inversion of typing, which will be crucial for the further development.

Wellformed contexts $\Gamma \vdash \mathsf{ok}$.

$$\textsc{cxt-empty}\ \frac{}{\diamond \vdash \mathsf{ok}} \qquad \textsc{cxt-ext}\ \frac{\Gamma \vdash A : \mathsf{Type}}{\Gamma, x\!:\!A \vdash \mathsf{ok}}$$

Type formation $\Gamma \vdash A : \mathsf{Type}$.

$$\textsc{set-f}\ \frac{\Gamma \vdash \mathsf{ok}}{\Gamma \vdash \mathsf{Set} : \mathsf{Type}} \qquad \textsc{set-e}\ \frac{\Gamma \vdash t : \mathsf{Set}}{\Gamma \vdash \mathsf{El}\, t : \mathsf{Type}}$$

$$\textsc{fun-f}\ \frac{\Gamma, x\!:\!A \vdash B : \mathsf{Type}}{\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}} \qquad \textsc{pair-f}\ \frac{\Gamma, x\!:\!A \vdash B : \mathsf{Type}}{\Gamma \vdash \mathsf{Pair}\, A\, (\lambda x B) : \mathsf{Type}}$$

Typing $\Gamma \vdash t : A$.

$$\textsc{hyp}\ \frac{\Gamma \vdash \mathsf{ok} \qquad (x\!:\!A) \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A = B : \mathsf{Type}}{\Gamma \vdash t : B}$$

$$\textsc{fun-i}\ \frac{\Gamma, x\!:\!A \vdash t : B}{\Gamma \vdash \lambda x t : \mathsf{Fun}\, A\, (\lambda x B)} \qquad \textsc{fun-e}\ \frac{\Gamma \vdash r : \mathsf{Fun}\, A\, (\lambda x B) \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B[s/x]}$$

$$\textsc{pair-i}\ \frac{\Gamma, x\!:\!A \vdash B : \mathsf{Type} \qquad \Gamma \vdash s : A \qquad \Gamma \vdash t : B[s/x]}{\Gamma \vdash (s, t) : \mathsf{Pair}\, A\, (\lambda x B)}$$

$$\textsc{pair-e-l}\ \frac{\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash r\, \mathsf{L} : A} \qquad \textsc{pair-e-r}\ \frac{\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash r\, \mathsf{R} : B[r\, \mathsf{L}/x]}$$

Fig. 1. MLF_Σ rules for contexts, types and typing.

Expressions (terms and types). We do not distinguish between terms and types syntactically. Dependent function types, usually written $\Pi x : A.\, B$, are written $\mathsf{Fun}\, A\, (\lambda x B)$; similarly, dependent pair types $\Sigma x : A.\, B$ are represented by

$\mathsf{Pair}\,A\,(\lambda x B)$. We write projections L and R postfix. The syntactic entities of MLF_Σ are given by the following grammar.

Var	$\ni x, y, z$		variables
Const	$\ni c$	$::= \mathsf{Fun} \mid \mathsf{Pair} \mid \mathsf{El} \mid \mathsf{Set}$	constants
Proj	$\ni p$	$::= \mathsf{L} \mid \mathsf{R}$	left and right projection
Exp	$\ni r, s, t, A, B, C$	$::= c \mid x \mid \lambda x t \mid r\,s \mid (t, t') \mid r\,p$	expressions
Cxt	$\ni \Gamma$	$::= \diamond \mid \Gamma, x\!:\!A$	typing contexts

We identify terms and types up to α-conversion and adopt the convention that in contexts Γ, all variables must be distinct; hence, the context extension $\Gamma, x\!:\!A$ presupposes $(x\!:\!B) \notin \Gamma$ for any B.

The inhabitants of Set are type codes; El maps type codes to types. E. g., $\mathsf{Fun}\ \mathsf{Set}\ (\lambda a.\ \mathsf{Fun}\ (\mathsf{El}\,a)\ (\lambda\_.\ \mathsf{El}\,a))$ is the type of the polymorphic identity $\lambda a \lambda x x$.

Judgements are inductively defined relations. If $\mathcal{D}$ is a derivation of judgement J, we write $\mathcal{D} :: J$. The type theory MLF_Σ is presented via five judgements:

$\Gamma \vdash \mathsf{ok}$	Γ is a well-formed context
$\Gamma \vdash A : \mathsf{Type}$	A is a well-formed type
$\Gamma \vdash t : A$	t has type A
$\Gamma \vdash A = A' : \mathsf{Type}$	A and A' are equal types
$\Gamma \vdash t = t' : A$	t and t' are equal terms of type A

Typing and well-formedness of types both have the form $\Gamma \vdash \_ : \_$. We will refer to them by the same judgement $\Gamma \vdash t : A$. If we mean typing only, we will require $A \not\equiv \mathsf{Type}$. The same applies to the equality judgements. Typing rules are given in Figure 1, together with the rules for well-formed contexts. The rules for the equality judgements are given in Figure 2. Observe that we have chosen a "parallel reduction" version for β- and η-rules, which has been inspired by Harper and Pfenning [HP05] and Sarnat [Sar04], in order to make the proof of functionality easier. In the following, we present properties of MLF_Σ which have easy syntactical proofs.

Admissible rules. MLF_Σ enjoys the usual properties of weakening, context conversion, substitution, functionality and inversion and injectivity for the type expressions $\mathsf{El}\ t$, $\mathsf{Fun}\,A\,(\lambda x B)$ and $\mathsf{Pair}\,A\,(\lambda x B)$. These rules can be found in the extended version of this article [AC05]. Note that in Martin-Löf's LF, injectivity is almost trivial since computation is restricted to the level of terms. This is also true for Harper and Pfenning's version of the Edinburgh LF which lacks type-level λ-abstraction [HP05]. In the Edinburgh LF with type-level λ it involves a normalization argument and is proven using logical relations [VC02].

Lemma 1 (Syntactic validity).

1. *Typing: If $\Gamma \vdash t : A$ then $\Gamma \vdash \mathsf{ok}$ and either $A \equiv \mathsf{Type}$ or $\Gamma \vdash A : \mathsf{Type}$.*
2. *Equality: If $\Gamma \vdash t = t' : A$ then $\Gamma \vdash t : A$ and $\Gamma \vdash t' : A$.*

Equivalence, hypotheses, conversion.

$$\textsc{eq-sym}\ \frac{\Gamma \vdash t = t' : A}{\Gamma \vdash t' = t : A} \qquad \textsc{eq-trans}\ \frac{\Gamma \vdash r = s : A \qquad \Gamma \vdash s = t : A}{\Gamma \vdash r = t : A}$$

$$\textsc{eq-hyp}\ \frac{\Gamma \vdash \mathsf{ok} \qquad (x{:}A) \in \Gamma}{\Gamma \vdash x = x : A} \qquad \textsc{eq-conv}\ \frac{\Gamma \vdash t = t' : A \qquad \Gamma \vdash A = B : \mathsf{Type}}{\Gamma \vdash t = t' : B}$$

Dependent functions.

$$\textsc{eq-fun-i}\ \frac{\Gamma, x{:}A \vdash t = t' : B}{\Gamma \vdash \lambda x t = \lambda x t' : \mathsf{Fun}\, A\, (\lambda x B)}$$

$$\textsc{eq-fun-e}\ \frac{\Gamma \vdash r = r' : \mathsf{Fun}\, A\, (\lambda x B) \qquad \Gamma \vdash s = s' : A}{\Gamma \vdash r\, s = r'\, s' : B[s/x]}$$

$$\textsc{eq-fun-}\beta\ \frac{\Gamma, x{:}A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x t)\, s = t[s/x] : B[s/x]}$$

$$\textsc{eq-fun-}\eta\ \frac{\Gamma \vdash t : \mathsf{Fun}\, A\, (\lambda x B)}{\Gamma \vdash (\lambda x.\, t\, x) = t : \mathsf{Fun}\, A\, (\lambda x B)}\ x \notin \mathsf{FV}(t)$$

Dependent pairs.

$$\textsc{eq-pair-i}\ \frac{\Gamma \vdash s = s' : A \qquad \Gamma \vdash t = t' : B[s/x]}{\Gamma \vdash (s, t) = (s', t') : \mathsf{Pair}\, A\, (\lambda x B)}$$

$$\textsc{eq-pair-e-l}\ \frac{\Gamma \vdash r = r' : \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash r\, \mathsf{L} = r'\, \mathsf{L} : A} \qquad \textsc{eq-pair-e-r}\ \frac{\Gamma \vdash r = r' : \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash r\, \mathsf{R} = r'\, \mathsf{R} : B[r\, \mathsf{L}/x]}$$

$$\textsc{eq-pair-}\beta\textsc{-l}\ \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash (s, t)\, \mathsf{L} = s : A} \qquad \textsc{eq-pair-}\beta\textsc{-r}\ \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash (s, t)\, \mathsf{R} = t : B}$$

$$\textsc{eq-pair-}\eta\ \frac{\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash (r\, \mathsf{L},\ r\, \mathsf{R}) = r : \mathsf{Pair}\, A\, (\lambda x B)}$$

Fig. 2. MLF_Σ term equality rules.

Lemma 2 (Inversion of Typing). *Let $C \not\equiv \mathsf{Type}$.*

1. *If $\Gamma \vdash x : C$ then $\Gamma \vdash \Gamma(x) = C : \mathsf{Type}$.*
2. *If $\Gamma \vdash \lambda x t : C$ then $C \equiv \mathsf{Fun}\, A\, (\lambda x B)$ and $\Gamma, x{:}A \vdash t : B$.*
3. *If $\Gamma \vdash r\, s : C$ then $\Gamma \vdash r : \mathsf{Fun}\, A\, (\lambda x B)$ with $\Gamma \vdash s : A$ and $\Gamma \vdash B[s/x] = C : \mathsf{Type}$.*
4. *If $\Gamma \vdash (r, s) : C$ then $C \equiv \mathsf{Pair}\, A\, (\lambda x B)$ with $\Gamma \vdash r : A$ and $\Gamma \vdash s : B[r/x]$.*
5. *If $\Gamma \vdash r\mathsf{L} : A$ then $\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)$.*
6. *If $\Gamma \vdash r\mathsf{R} : C$ then $\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)$ and $\Gamma \vdash B[r\mathsf{L}/x] = C : \mathsf{Type}$.*

3 Algorithmic Presentation

In this section, we present an algorithm for deciding equality. The goal of this article is to prove it sound and complete.

Syntactic classes. The algorithm works on weak head normal forms WVal. For convenience, we introduce separate categories for normal forms which can denote a function and for those which can denote a pair. In the intersection of these categories live the neutral expressions.

$$\begin{array}{lll}
\mathsf{WElim} \ni e & ::= s \mid p & \text{eliminations} \\
\mathsf{WNe} \ni n & ::= c \mid x \mid n\, e & \text{neutral expressions} \\
\mathsf{WFun} \ni w_f & ::= n \mid \lambda x t & \text{weak head function values} \\
\mathsf{WPair} \ni w_p & ::= n \mid (t, t') & \text{weak head pair values} \\
\mathsf{WVal} \ni w, W & ::= w_f \mid w_p & \text{weak head values}
\end{array}$$

Weak head evaluation $t \searrow w$ and active elimination $w@e \searrow w'$ are simultaneously given by the following rules:

$$\frac{r \searrow w_f \qquad w_f@s \searrow w}{r\, s \searrow w} \qquad \frac{r \searrow w_p \qquad w_p@p \searrow w}{r\, p \searrow w} \qquad \frac{}{t \searrow t}\; t \not\equiv r\, s \mid r\, p$$

$$\frac{}{n@e \searrow n\, e} \qquad \frac{t[w/x] \searrow w'}{(\lambda x t)@w \searrow w'} \qquad \frac{t \searrow w}{(t, t')@\mathsf{L} \searrow w} \qquad \frac{t' \searrow w}{(t, t')@\mathsf{R} \searrow w}$$

Weak head evaluation $t \searrow w$ is equivalent to multi-step weak head reduction to normal form. Since both judgements are deterministic, we can interpret them by two partial functions

$$\begin{array}{ll}
\downarrow\; \in \mathsf{Exp} \rightharpoonup \mathsf{WVal} & \text{weak head evaluation,} \\
@ \in \mathsf{WVal} \times \mathsf{WElim} \rightharpoonup \mathsf{WVal} & \text{active application.}
\end{array}$$

Conversion. Two terms t, t' are *algorithmically equal* if $t \searrow w$, $t' \searrow w'$, and $w \sim w'$. We combine these three propositions to $t{\downarrow} \sim t'{\downarrow}$. The algorithmic equality on weak head normal forms $w \sim w'$ is given inductively by these rules:

$$\textsc{aq-c}\ \frac{}{c \sim c} \qquad \textsc{aq-var}\ \frac{}{x \sim x}$$

$$\textsc{aq-ne-fun}\ \frac{n \sim n' \qquad s{\downarrow} \sim s'{\downarrow}}{n\,s \sim n'\,s'} \qquad \textsc{aq-ne-pair}\ \frac{n \sim n'}{n\,p \sim n'\,p}$$

$$\textsc{aq-ext-fun}\ \frac{w_f@x \sim w'_f@x}{w_f \sim w'_f}\ x \notin \mathsf{FV}(w_f, w'_f)$$

$$\textsc{aq-ext-pair}\ \frac{w_p@\mathsf{L} \sim w'_p@\mathsf{L} \qquad w_p@\mathsf{R} \sim w'_p@\mathsf{R}}{w_p \sim w'_p}$$

For two neutral values, the rules (AQ-NE-X) are preferred over AQ-EXT-FUN and AQ-EXT-PAIR. Thus, conversion is deterministic. It is easy to see that it is symmetric as well.

In our presentation, untyped conversion resembles type-directed conversion. In the terminology of Harper and Pfenning [HP05,Sar04], the first four rules AQ-C, AQ-VAR, AQ-NE-FUN and AQ-NE-PAIR compute *structural equality*, whereas the remaining two, the extensionality rules AQ-EXT-FUN and AQ-EXT-PAIR, compute type-directed equality. The difference is that in our formulation, the *shape* of a value—function or pair— triggers application of the extensionality rules.

Remark 3. In contrast to the corresponding equality for λ-terms without pairs [Coq91] (taking away AQ-NE-PAIR and AQ-EXT-PAIR), this relation is *not* transitive. For instance, $\lambda x.\, n\, x \sim n$ and $n \sim (n\mathsf{L},\ n\mathsf{R})$, but not $\lambda x.\, n\, x \sim (n\mathsf{L},\ n\mathsf{R})$.

4 Soundness

The soundness proof for conversion in this section is entirely syntactical and relies crucially on injectivity of El, Fun and Pair and inversion of typing. First, we show soundness of weak head evaluation, which subsumes subject reduction.

Lemma 4 (Soundness of weak head evaluation).

1. *If $\mathcal{D} :: t \searrow w$ and $\Gamma \vdash t : C$ then $\Gamma \vdash t = w : C$.*
2. *If $\mathcal{D} :: w@e \searrow w'$ and $\Gamma \vdash w\, e : C$ then $\Gamma \vdash w\, e = w' : C$.*

Proof. Simultaneously by induction on $\mathcal{D}$, making essential use of inversion laws.

Two algorithmically convertible well-typed expressions must also be equal in the declarative sense. In case of neutral terms, we also obtain that their types are equal. This is due to the fact that we can read off the type of the common head variable and break it down through the sequence of eliminations.

Lemma 5 (Soundness of conversion).

1. *Neutral non-types: If* $\mathcal{D} :: n \sim n'$ *and* $\Gamma \vdash n : C \not\equiv \mathsf{Type}$ *and* $\Gamma \vdash n' : C' \not\equiv \mathsf{Type}$ *then* $\Gamma \vdash n = n' : C$ *and* $\Gamma \vdash C = C' : \mathsf{Type}$*.*
2. *Weak head values: If* $\mathcal{D} :: w \sim w'$ *and* $\Gamma \vdash w, w' : C$ *then* $\Gamma \vdash w = w' : C$*.*
3. *All expressions: If* $t{\downarrow} \sim t'{\downarrow}$ *and* $\Gamma \vdash t, t' : C$ *then* $\Gamma \vdash t = t' : C$*.*

Proof. The third proposition is a consequence of the second, using soundness of evaluation (Lemma 4) and transitivity. We prove the first two propositions simultaneously by induction on $\mathcal{D}$.

5 Models

To show completeness of algorithmic equality, we leave the syntactic discipline. Although a syntactical proof should be possible following Goguen [Gog99,Gog05], we prefer a model construction since it is more apt to extensions of the type theory.

The contribution of this section is that *any* PER model over a λ-model with full β-equality is a model of MLF_Σ. Only in the next section will we decide on a particular model which enables the completeness proof.

5.1 λ Models

We assume a set D with the four operations

$$\begin{array}{lll}
\_\cdot\_ & \in \mathsf{D} \times \mathsf{D} \to \mathsf{D} & \text{application,} \\
\_\mathsf{L} & \in \mathsf{D} \to \mathsf{D} & \text{left projection,} \\
\_\mathsf{R} & \in \mathsf{D} \to \mathsf{D} & \text{right projection, and} \\
\_\,\_ & \in \mathsf{Exp} \times \mathsf{Env} \to \mathsf{D} & \text{denotation.}
\end{array}$$

Herein, we use the following entities:

$$\begin{array}{llll}
c & \in \mathsf{Const} & := \{\mathsf{Set}, \mathsf{El}, \mathsf{Fun}, \mathsf{Pair}\} & \text{constants} \\
u, v, f, V, F & \in \mathsf{D} & \supseteq \mathsf{Const} & \text{domain of the model} \\
\rho, \sigma & \in \mathsf{Env} & := \mathsf{Var} \to \mathsf{D} & \text{environments}
\end{array}$$

Let p range over the projection functions L and R. To simplify the notation, we write also $f\ v$ for $f \cdot v$. Update of environment ρ by the binding $x = v$ is written $\rho, x = v$. The operations $f \cdot v$, $v\,p$ and $t\rho$ must satisfy the following laws:

$$\begin{array}{lrcll}
\text{DEN-CONST} & c\rho & = & c & \text{if } c \in \mathsf{Const} \\
\text{DEN-VAR} & x\rho & = & \rho(x) & \\
\text{DEN-FUN-E} & (r\ s)\rho & = & r\rho\ (s\rho) & \\
\text{DEN-PAIR-E} & (r\,p)\rho & = & r\rho\,p & \\
 & & & & \\
\text{DEN-FUN-}\beta & (\lambda x t)\rho\ v & = & t(\rho, x = v) & \\
\text{DEN-PAIR-}\beta\text{-L} & (r, s)\rho\,\mathsf{L} & = & r\rho & \\
\text{DEN-PAIR-}\beta\text{-R} & (r, s)\rho\,\mathsf{R} & = & s\rho &
\end{array}$$

DEN-FUN-ξ	$(\lambda xt)\rho = (\lambda xt')\rho'$	if $t(\rho, x{=}v) = t'(\rho', x{=}v)$ for all $v \in \mathsf{D}$
DEN-PAIR-ξ	$(r, s)\rho = (r', s')\rho'$	if $r\rho = r'\rho'$ and $s\rho = s'\rho'$

DEN-SET-F-INJ	$\mathsf{El}\,v = \mathsf{El}\,v'$	implies $v = v'$
DEN-FUN-F-INJ	$\mathsf{Fun}\,V\,F = \mathsf{Fun}\,V'\,F'$	implies $V = V'$ and $F = F'$
DEN-PAIR-F-INJ	$\mathsf{Pair}\,V\,F = \mathsf{Pair}\,V'\,F'$	implies $V = V'$ and $F = F'$

Lemma 6 (Irrelevance). *If $\rho(x) = \rho'(x)$ for all $x \in \mathsf{FV}(t)$, then $t\rho = t\rho'$.*

Proof. By induction on t. Makes crucial use of the ξ rules.

Lemma 7 (Soundness of substitution). $(t[s/x])\rho = t(\rho, x{=}s\rho)$.

Proof. By induction on t, using the ξ rules and Lemma 6.

5.2 PER Models

In the definition of PER models, we follow a paper of the second author with Pollack and Takeyama [CPT03] and Vaux [Vau04]. The only difference is, since we have codes for types in D, we can define the semantical property of *being a type* directly on elements of D, whereas the cited works introduce an *intensional type equality* on closures $t\rho$.

Partial equivalence relation (PER). A PER is a symmetric and transitive relation. Let Per denote the set of PERs over D. If $\mathcal{A} \in \mathsf{Per}$, we write $v = v' \in \mathcal{A}$ if $(v, v') \in \mathcal{A}$. We say $v \in \mathcal{A}$ if v is in the carrier of $\mathcal{A}$, i. e., $v = v \in \mathcal{A}$. On the other hand, each set $\mathcal{A} \subseteq \mathsf{D}$ can be understood as the discrete PER where $v = v' \in \mathcal{A}$ holds iff $v = v'$ and $v \in \mathcal{A}$.

Equivalence classes and families. Let $\mathcal{A} \in \mathsf{Per}$. If $v \in \mathcal{A}$, then $\overline{v}_{\mathcal{A}} := \{v' \in \mathsf{D} \mid v = v' \in \mathcal{A}\}$ denotes the equivalence class of v in $\mathcal{A}$. We write $\mathsf{D}/\mathcal{A}$ for the set of all equivalence classes in $\mathcal{A}$. Let $\mathsf{Fam}(\mathcal{A}) = \mathsf{D}/\mathcal{A} \to \mathsf{Per}$. If $\mathcal{F} \in \mathsf{Fam}(\mathcal{A})$ and $v \in \mathcal{A}$, we use $\mathcal{F}(v)$ as a shorthand for $\mathcal{F}(\overline{v}_{\mathcal{A}})$.

Constructions on PERs. Let $\mathcal{A} \in \mathsf{Per}$ and $\mathcal{F} \in \mathsf{Fam}(\mathcal{A})$. We define two PERs $\mathcal{F}un(\mathcal{A}, \mathcal{F})$ and $\mathcal{P}air(\mathcal{A}, \mathcal{F})$ by

$$\begin{array}{lll} (f, f') \in \mathcal{F}un(\mathcal{A}, \mathcal{F}) & \text{iff} & f\ v = f'\ v' \in \mathcal{F}(v) \text{ for all } v = v' \in \mathcal{A}, \\ (v, v') \in \mathcal{P}air(\mathcal{A}, \mathcal{F}) & \text{iff} & v\,\mathsf{L} = v'\,\mathsf{L} \in \mathcal{A} \text{ and } v\,\mathsf{R} = v'\,\mathsf{R} \in \mathcal{F}(v\,\mathsf{L}). \end{array}$$

Semantical types. In the following, assume some $\mathcal{S}et \in \mathsf{Per}$ and some $\mathcal{E}\ell \in \mathsf{Fam}(\mathcal{S}et)$. We define inductively a new relation $\mathcal{T}ype \in \mathsf{Per}$ and a new function $[\_] \in \mathsf{Fam}(\mathcal{T}ype)$:

$\mathsf{Set} = \mathsf{Set} \in \mathcal{T}ype$ and $[\mathsf{Set}]$ is $\mathcal{S}et$.

$\mathsf{El}\ v = \mathsf{El}\ v' \in \mathcal{T}ype$ if $v = v' \in \mathcal{S}et$. Then $[\mathsf{El}\ v]$ is $\mathcal{E}\ell(v)$.

$\mathsf{Fun}\ V\ F = \mathsf{Fun}\ V'\ F' \in \mathcal{T}ype$ if $V = V' \in \mathcal{T}ype$ and $v = v' \in [V]$ implies $F\ v = F'\ v' \in \mathcal{T}ype$. We define then $[\mathsf{Fun}\ V\ F]$ to be $\mathcal{F}un([V], v \longmapsto [F\ v])$.

$\mathsf{Pair}\ V\ F = \mathsf{Pair}\ V'\ F' \in \mathcal{T}ype$ if $V = V' \in \mathcal{T}ype$ and $v = v' \in [V]$ implies $F\ v = F'\ v' \in \mathcal{T}ype$. We define then $[\mathsf{Pair}\ V\ F]$ to be $\mathcal{P}air([V], v \longmapsto [F\ v])$.

This definition is possible by the laws DEN-SET-F-INJ, DEN-FUN-F-INJ, and DEN-PAIR-F-INJ. Notice that in the last two clauses, we have

$\mathcal{F}un([V], v \longmapsto [F\ v]) = \mathcal{F}un([V'], v \longmapsto [F'\ v])$, and
$\mathcal{P}air([V], v \longmapsto [F\ v]) = \mathcal{P}air([V'], v \longmapsto [F'\ v])$.

5.3 Validity

If Γ is a context, we define a corresponding PER on Env, written $[\Gamma]$. We define $\rho = \rho' \in [\Gamma]$ to mean that, for all $x{:}A$ in Γ, we have $A\rho = A\rho' \in \mathcal{T}ype$ and $\rho(x) = \rho'(x) \in [A\rho]$. Semantical contexts $\Gamma \in \mathcal{C}xt$ are defined inductively by the following rules:

$$\frac{}{\diamond \in \mathcal{C}xt} \qquad \frac{\Gamma \in \mathcal{C}xt \qquad A\rho = A\rho' \in \mathcal{T}ype \text{ for all } \rho = \rho' \in [\Gamma]}{(\Gamma, x\,{:}\,A) \in \mathcal{C}xt}$$

Theorem 8 (Soundness of the rules of MLF_Σ).

1. *If $\mathcal{D} :: \Gamma \vdash \mathsf{ok}$ then $\Gamma \in \mathcal{C}xt$.*
2. *If $\mathcal{D} :: \Gamma \vdash A : \mathsf{Type}$ then $\Gamma \in \mathcal{C}xt$, and if $\rho = \rho' \in [\Gamma]$ then $A\rho = A\rho' \in \mathcal{T}ype$.*
3. *If $\mathcal{D} :: \Gamma \vdash t : A$ then $\Gamma \in \mathcal{C}xt$, and if $\rho = \rho' \in [\Gamma]$ then $A\rho = A\rho' \in \mathcal{T}ype$ and $t\rho = t\rho' \in [A\rho]$.*
4. *If $\mathcal{D} :: \Gamma \vdash A = A' : \mathsf{Type}$ then $\Gamma \in \mathcal{C}xt$, and if $\rho = \rho' \in [\Gamma]$ then $A\rho = A'\rho' \in \mathcal{T}ype$.*
5. *If $\mathcal{D} :: \Gamma \vdash t = t' : A$ then $\Gamma \in \mathcal{C}xt$, and if $\rho = \rho' \in [\Gamma]$ then $A\rho = A\rho' \in \mathcal{T}ype$ and $t\rho = t'\rho' \in [A\rho]$.*

Proof. Each by induction on $\mathcal{D}$, using lemmas 6 and 7.

5.4 Safe Types

We define an abstract notion of *safety*, similar to what Vaux calls "saturation" [Vau04]. A PER is safe if it lies between a PER $\mathcal{N}$ on *neutral* expressions and a PER $\mathcal{S}$ on *safe* expressions [Vou04]. In the following, we use set notation $\subseteq$ and $\cup$ also for PERs.

Safety. $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair} \in \mathsf{Per}$ form a *safety range* if the following conditions are met:

SAFE-INT $\quad \mathcal{N} \subseteq \mathcal{S} = \mathcal{S}_{fun} \cup \mathcal{S}_{pair}$
SAFE-NE-FUN $\quad u\,v = u'\,v' \in \mathcal{N}$ if $u = u' \in \mathcal{N}$ and $v = v' \in \mathcal{S}$
SAFE-NE-PAIR $\quad u\,p = u'\,p \in \mathcal{N}$ if $u = u' \in \mathcal{N}$
SAFE-EXT-FUN $\quad v = v' \in \mathcal{S}_{fun}$ if $v\,u = v'\,u' \in \mathcal{S}$ for all $u = u' \in \mathcal{N}$
SAFE-EXT-PAIR $\quad v = v' \in \mathcal{S}_{pair}$ if $v\,\mathsf{L} = v'\,\mathsf{L} \in \mathcal{S}$ and $v\,\mathsf{R} = v'\,\mathsf{R} \in \mathcal{S}$

A relation $\mathcal{A} \in \mathsf{Per}$ is called *safe* w. r. t. to a safety range $(\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair})$ if $\mathcal{N} \subseteq \mathcal{A} \subseteq \mathcal{S}$.

Lemma 9 (Fun and Pair preserve safety). *If $\mathcal{A} \in \mathsf{Per}$ is safe and $\mathcal{F} \in \mathsf{Fam}(\mathcal{A})$ is such that $\mathcal{F}(v)$ is safe for all $v \in \mathcal{A}$ then $\mathcal{F}un(\mathcal{A}, \mathcal{F})$ and $\mathcal{P}air(\mathcal{A}, \mathcal{F})$ are safe.*

Proof. By monotonicity of $\mathcal{F}un$ and $\mathcal{P}air$, if one considers the following reformulation of the conditions:

$$\begin{array}{ll}
\textsc{safe-ne-fun} & \mathcal{N} \subseteq \mathcal{F}un(\mathcal{S}, \_ \longmapsto \mathcal{N}) \\
\textsc{safe-ne-pair} & \mathcal{N} \subseteq \mathcal{P}air(\mathcal{N}, \_ \longmapsto \mathcal{N}) \\
\textsc{safe-ext-fun} & \mathcal{F}un(\mathcal{N}, \_ \longmapsto \mathcal{S}) \subseteq \mathcal{S}_{fun} \\
\textsc{safe-ext-pair} & \mathcal{P}air(\mathcal{S}, \_ \longmapsto \mathcal{S}) \subseteq \mathcal{S}_{pair}
\end{array}$$

Lemma 10 (Type interpretations are safe). *Let $\mathcal{S}et$ be safe and $\mathcal{E}\ell(v)$ be safe for all $v \in \mathcal{S}et$. If $v \in \mathcal{T}ype$ then $[v]$ is safe.*

Proof. By induction on the proof that $v \in \mathcal{T}ype$, using Lemma 9.

6 Term Model

In this section, we instantiate the model of the previous section to the set of expressions modulo β-equality. Application is interpreted as expression application and the projections of the model are mapped to projections for expressions.

Let $\overline{r} \in \mathcal{D}$ denote the equivalence class of $r \in \mathsf{Exp}$ with regard to $=_\beta$. We set $\mathsf{D} := \mathsf{Exp}/{=_\beta}$, $\overline{r} \cdot \overline{s} := \overline{r\,s}$, $\overline{r}\,\mathsf{L} := \overline{r\,\mathsf{L}}$, $\overline{r}\,\mathsf{R} := \overline{r\,\mathsf{R}}$, and $t\rho := \overline{t[\rho]}$. Herein, $t[\rho]$ denotes the substitution of $\rho(x)$ for x in t, carried out in parallel for all $x \in \mathsf{FV}(t)$. In the following, we abbreviate the equivalence class $\overline{r}$ by its representative r, if clear from the context.

Value classes. The β-normal forms $v \in \mathsf{Val}$, which can be described by the following grammar, completely represent the β-equivalence classes $\overline{t} \in \mathsf{Exp}/{=_\beta}$.

$$\begin{array}{llll}
\mathsf{VNe} \ni u & ::= c \mid x \mid u\,v \mid u\,p & & \text{neutral values} \\
\mathsf{VFun} \ni v_f & ::= u \mid \lambda x v & & \text{function values} \\
\mathsf{VPair} \ni v_p & ::= u \mid (v, v') & & \text{pair values} \\
\mathsf{Val} \ni v & ::= v_f \mid v_p & & \text{values}
\end{array}$$

An η-equality on β-equivalence classes. We define a relation $\simeq\ \subseteq \mathsf{Val} \times \mathsf{Val}$ inductively by the following rules.

$$\textsc{eta-var}\ \frac{}{x \simeq x} \qquad \textsc{eta-ne-fun}\ \frac{u \simeq u' \qquad v \simeq v'}{u\,v \simeq u'\,v'} \qquad \textsc{eta-ne-pair}\ \frac{u \simeq u'}{u\,p \simeq u'\,p}$$

$$\textsc{eta-c}\ \frac{}{c \simeq c} \qquad \textsc{eta-ext-fun}\ \frac{v_f\,x \simeq v'_f\,x}{v_f \simeq v'_f}\ x \notin \mathsf{FV}(v_f, v'_f)$$

$$\textsc{eta-ext-pair}\ \frac{v_p\,\mathsf{L} \simeq v'_p\,\mathsf{L} \qquad v_p\,\mathsf{R} \simeq v'_p\,\mathsf{R}}{v_p \simeq v'_p}$$

Note, since we are talking about equivalence classes, in the extensionality rules ETA-EXT-FUN and ETA-EXT-PAIR we actually mean the normal forms of the expressions appearing in the hypotheses. In the conclusion of an extensionality rule, we require one of the two values to be non-neutral.

As algorithmic equality, the relation $\simeq$ is symmetric, but not transitive. To turn it into a PER, we need to take the transitive closure $\simeq^+$ explicitly.

Lemma 11 (Admissible rules for $\simeq^+$). *If we replace $\simeq$ by $\simeq^+$ consistently in the rules for $\simeq$, we get admissible rules for $\simeq^+$. We denote the admissible rule by appending a $^+$ to the rule name.*

Lemma 12 (Safety range). *Let $\mathcal{S} := \simeq^+$, $\mathcal{N} := \mathcal{S} \cap (\mathsf{VNe} \times \mathsf{VNe})$, $\mathcal{S}_{fun} := \mathcal{S} \cap (\mathsf{VFun} \times \mathsf{VFun})$, and $\mathcal{S}_{pair} := \mathcal{S} \cap (\mathsf{VPair} \times \mathsf{VPair})$. Then $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair}$ are PERs and form a safety range.*

Proof. SAFE-INT is shown by definition of $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair}$. SAFE-EXT-FUN is satisfied by rule ETA-EXT-FUN$^+$ since $x = x \in \mathcal{N}$ for each variable. Each other requirement has its directly matching admissible rule.

Lemma 13 (Context satisfiable). *Let $\rho_0(x) := x$ for all $x \in \mathsf{Var}$. If $\Gamma \vdash \mathsf{ok}$, then $\rho_0 \in [\Gamma]$.*

Corollary 14 (Equal terms are related). *If $\Gamma \vdash t = t' : C \not\equiv \mathsf{Type}$ then $\overline{t} \simeq^+ \overline{t}'$.*

Proof. By soundness of MLF_Σ (Thm. 8), $t\rho_0 = t'\rho_0 \in [C\rho_0]$. The claim follows since $[C\rho_0] \subseteq \mathcal{S}$ by Lemma 10.

It remains to show that $\overline{t} \simeq^+ \overline{t}'$ implies $t{\downarrow} \sim t'{\downarrow}$, which means that both t and t' weak head normalize and these normal forms are algorithmically equal.

7 Completeness

We establish completeness of the algorithmic equality in two steps. First we prove that η-equality of β-normal forms entails equality in the algorithmic sense. Then we show that for well-typed terms, transitivity is admissible for algorithmic equality. Combining this with the result of the last section, we are done.

Lemma 15 (Standardization).

1. *If $t =_\beta u\,v$ then $t \searrow n\,s$ with $n =_\beta u$ and $s =_\beta v$.*
2. *If $t =_\beta u\,p$ then $t \searrow n\,p$ with $n =_\beta u$.*
3. *If $t =_\beta v_f$ then $t \searrow w_f$ with $w_f =_\beta v_f$.*
4. *If $t =_\beta v_p$ then $t \searrow w_p$ with $w_p =_\beta v_p$.*

Proof. Fact about the λ-calculus [Bar84].

Lemma 16 (Completeness of $\sim$ w. r. t. $\simeq$). *If $\mathcal{D} :: \overline{n} \simeq \overline{n}'$ then $n \sim n'$ and if $\mathcal{D} :: \overline{t} \simeq \overline{t}'$ then $t{\downarrow} \sim t'{\downarrow}$.*

Proof. Simultaneously by induction on $\mathcal{D}$, using standardization.

While transitivity does not hold for the pure algorithmic equality (see Remark 3), it can be established for terms of the same type. The presence of types forbids comparison of function values with pair values, the stepping stone for transitivity of the untyped equality.

For a derivation $\mathcal{D}$ of algorithmic equality, we define the measure $|\mathcal{D}|$ which denotes the number of rule applications on the longest branch of $\mathcal{D}$, counting the rules AQ-EXT-FUN and AQ-EXT-PAIR *twice.*[2] We will use this measure for the proof of transitivity and termination of algorithmic equality.

Lemma 17 (Transitivity of typed algorithmic equality).

1. *Let $\Gamma \vdash n_1 : C_1$, $\Gamma \vdash n_2 : C_2$, and $\Gamma \vdash n_3 : C_3$. If $\mathcal{D} :: n_1 \sim n_2$ and $\mathcal{D}' :: n_2 \sim n_3$ then $n_1 \sim n_3$.*
2. *Let $\Gamma \vdash w_1, w_2, w_3 : C$. If $\mathcal{D} :: w_1 \sim w_2$ and $\mathcal{D}' :: w_2 \sim w_3$ then $w_1 \sim w_3$.*
3. *Let $\Gamma \vdash t_1, t_2, t_3 : C$. If $t_1{\downarrow} \sim t_2{\downarrow}$ and $t_2{\downarrow} \sim t_3{\downarrow}$ then $t_1{\downarrow} \sim t_3{\downarrow}$.*

Proof. The third proposition is an immediate consequence of the second, using soundness of weak head evaluation. We prove 1. and 2. simultaneously by induction on $|\mathcal{D}| + |\mathcal{D}'|$, using inversion for typing and soundness of algorithmic equality.

Theorem 18 (Completeness of algorithmic equality).

1. *If $\Gamma \vdash t = t' : C \not\equiv \mathsf{Type}$ then $t{\downarrow} \sim t'{\downarrow}$.*
2. *If $\mathcal{D} :: \Gamma \vdash A = A' : \mathsf{Type}$ then $A{\downarrow} \sim A'{\downarrow}$.*

Proof. Completeness for terms (1): By Cor. 14 we have $\overline{t} \simeq^+ \overline{t}'$. Lemma 16 entails $t{\downarrow} \sim^+ t'{\downarrow}$, and since $\Gamma \vdash t, t' : C$, we infer $t{\downarrow} \sim t'{\downarrow}$ by transitivity. The completeness for types (2) is then shown by induction on $\mathcal{D}$, using completeness for terms in case EQ-SET-E.

We have shown that two judgmentally equal terms t, t' weak-head normalize to w, w' and a derivation of $w \sim w'$ exists, hence the equality algorithm, which searches deterministically for such a derivation, terminates with success. What remains to show is that the query $t{\downarrow} \sim t'{\downarrow}$ terminates for all welltyped t, t', either with success, if the derivation can be closed, or with failure, in case the search arrives at a point where there is no matching rule. For the following lemma, observe that $w \sim w$ iff w is weakly normalizing.

Lemma 19 (Termination of equality). *If $\mathcal{D}_1 :: w_1 \sim w_1$ and $\mathcal{D}_2 :: w_2 \sim w_2$ then the query $w_1 \sim w_2$ terminates.*

Proof. By induction on $|\mathcal{D}_1| + |\mathcal{D}_2|$.

[2] A similar measure is used by Goguen [Gog05] to prove termination of algorithmic equality restricted to pure λ-terms [Coq91].

Theorem 20 (Decidability of equality). *If $\Gamma \vdash t, t' : C$ then the query $t\!\downarrow\, \sim t'\!\downarrow$ succeeds or fails finitely and decides $\Gamma \vdash t = t' : C$.*

Proof. By Theorem 18, $t \searrow w$, $t' \searrow w'$, $w \sim w$, and $w' \sim w'$. By the previous lemma, the query $w \sim w'$ terminates. Since by soundness and completeness of the algorithmic equality, $w \sim w'$ if and only if $\Gamma \vdash t = t' : C$, the query decides judgmental equality.

8 Conclusion

We have presented a sound and complete conversion algorithm for MLF_Σ. The completeness proof builds on PERs over untyped expressions, hence, we need—in contrast to Harper and Pfenning's completeness proof for type-directed conversion [HP05]—no Kripke model and no notion of erasure, what we consider an arguably simpler procedure. We see in principle no obstacle to generalize our results to type theories with type definition by cases (large eliminations), whereas it is not clear how to treat them with a technique based on erasure.

The disadvantage of untyped conversion, compared to type-directed conversion, is that it cannot handle cases where the type of a term provides more information on equality than the shape of a terms, e. g., unit types, singleton types and signatures with manifest fields [CPT03].

A more general proof of completeness? Our proof uses a λ-model with full β-equality thanks to the ξ-rules. We had also considered a weaker model without ξ-rules which only equates weakly convertible objects. Combined with extensional PERs this would have been the model closest to our algorithm. But due to the use of substitution in the declarative formulation, we could not show MLF_Σ's rules to be valid in such a model. Whether it still can be done, remains an open question.

Related work. The second author, Pollack, and Takeyama [CPT03] present a model for $\beta\eta$-equality for an extension of the logical framework by singleton types and signatures with manifest fields. Equality is tested by η-expansion, followed by β-normalization and syntactic comparison. In contrast to this work, no syntactic specification of the framework and no incremental conversion algorithm are given.

Schürmann and Sarnat [Sar04] have been working on an extension of the Edinburgh Logical Framework (ELF) by Σ-types (LF$_\Sigma$), following Harper and Pfenning [HP05]. In comparison to MLF_Σ, syntactic validity (Lemma 1) and injectivity are non-trivial in their formulation of ELF. Robin Adams [Ada01] has extended Harper and Pfenning's algorithm to Luo's logical framework (i. e., MLF with typed λ-abstraction) with Σ-types and unit.

Goguen [Gog99] gives a typed operational semantics for Martin-Löf's logical framework. An extension to Σ-types has to our knowledge not yet been considered. Recently, Goguen [Gog05] has proven termination and completeness for both the type-directed [HP05] and the shape-directed equality [Coq91] from the standard meta-theoretical properties (strong normalization, confluence, subject reduction, etc.) of the logical framework.

Acknowledgments. We are grateful to Lionel Vaux whose clear presentation of models for this implicit calculus [Vau04] provided a guideline for our model construction. Thanks to Ulf Norell for proof-reading. The first author is indebted to Frank Pfenning who taught him type-directed equality at Carnegie Mellon University in 2000, and to Carsten Schürmann for communication on LF$_\Sigma$.

References

AC05. A. Abel and T. Coquand. Untyped algorithmic equality for Martin-Löf's logical framework with surjective pairs (extended version). Tech. rep., Department of Computer Science, Chalmers, Göteborg, Sweden, 2005.

Ada01. R. Adams. Decidable equality in a logical framework with sigma kinds, 2001. Unpublished note, see http://www.cs.man.ac.uk/˜radams/.

Bar84. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, Amsterdam, 1984.

Coq91. T. Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. Plotkin, eds., *Logical Frameworks*, pp. 255–279. Cambridge University Press, 1991.

Coq96. T. Coquand. An algorithm for type-checking dependent types. In *Mathematics of Program Construction (MPC 1995)*, vol. 26 of *Science of Computer Programming*, pp. 167–177. Elsevier Science, 1996.

CPT03. T. Coquand, R. Pollack, and M. Takeyama. A logical framework with dependently typed records. In *Typed Lambda Calculus and Applications, TLCA'03*, vol. 2701 of *Lecture Notes in Computer Science*. Springer, 2003.

Gog99. H. Goguen. Soundness of the logical framework for its typed operational semantics. In J.-Y. Girard, ed., *Typed Lambda Calculi and Applications, TLCA 1999*, vol. 1581 of *Lecture Notes in Computer Science*. Springer, 1999.

Gog05. H. Goguen. Justifying algorithms for $\beta\eta$ conversion. In *FoSSaCS 2005*. To appear.

HHP93. R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the Association of Computing Machinery*, 40(1):143–184, 1993.

HP05. R. Harper and F. Pfenning. On equivalence and canonical forms in the LF type theory. *ACM Transactions on Computational Logic*, 6(1):61–101, 2005.

Klo80. J. W. Klop. Combinatory reducion systems. *Mathematical Center Tracts*, 27, 1980.

NPS00. B. Nordström, K. Petersson, and J. Smith. Martin-löf's type theory. In *Handbook of Logic in Computer Science*, vol. 5. Oxford University Press, 2000.

Sar04. J. Sarnat. LF$_\Sigma$: The metatheory of LF with Σ types, 2004. Unpublished technical report, kindly provided by Carsten Schürmann.

Vau04. L. Vaux. A type system with implicit types, 2004. English version of his mémoire de maîtrise.

VC02. J. C. Vanderwaart and K. Crary. A simplified account of the metatheory of Linear LF. Tech. rep., Dept. of Comp. Sci., Carnegie Mellon, 2002.

Vou04. J. Vouillon. Subtyping union types. In J. Marcinkowski and A. Tarlecki, eds., *Computer Science Logic, CSL'04*, vol. 3210 of *Lecture Notes in Computer Science*, pp. 415–429. Springer, 2004.