Fundamenta Informaticae ISSUE 1001–1047
IOS Press

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

(Draft of December 30, 2005)

Andreas Abel* [C]

Institut für Informatik, Ludwigs-Maximilians-Universität München

abel@informatik.uni-muenchen.de

Thierry Coquand*

Department of Computer Science, Chalmers University of Technology

coquand@cs.chalmers.se

Abstract. Martin-Löf's Logical Framework is extended by strong Σ-types and presented via judgmental equality with rules for extensionality and surjective pairing. Soundness of the framework rules is proven via a generic PER model on untyped terms. An algorithmic version of the framework is given through an untyped $\beta\eta$-equality test and a bidirectional type checking algorithm. Completeness is proven by instantiating the PER model with η-equality on β-normal forms, which is shown equivalent to the algorithmic equality.

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 [15]. Furthermore, the surjective-pairing reduction does not preserve types.

*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).
[C]Corresponding author

We therefore advocate the incremental $\beta\eta$-convertibility test which has been given by the second author for dependently typed λ-terms [6], 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 [13]. The advantage of this approach is that it can handle cases where the type provides extra information which is not present already 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 erase the dependencies and obtain simple types to direct the equality algorithm. In the theory they consider, the Edinburgh Logical Framework [12], 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 type-checking algorithm of the second author [6] 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 [3] 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, plus a bidirectional type checking algorithm for normal terms (Section 3). The goal of this article is to show that the algorithmic presentation of MLF_Σ is equivalent to the declarative one. Soundness is proven rather directly in Section 4, requiring inversion for

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

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 (partial equivalence relation) 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 and show that η-equality of β-normal forms is a partial equivalence, hence, gives rise to a PER model. In Section 7 we give a proof that η-equivalence is decided by the algorithmic equality which implies that the algorithmic equality serves as basis for a PER model as well. This entails completeness of the algorithm. We could have done a more direct proof, without the intermediate model involving η-equality, and this (rather technical) path is taken in Section 8. Decidability of judgmental equality on well-typed terms in MLF_Σ ensues, which entails that type checking of normal forms is decidable as well (Section 9).

2. Declarative Presentation of MLF_Σ

This section presents the typing and equality rules for an extension of Martin-Löf's logical framework [16] 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.

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	$::=$	$c \mid x \mid \lambda x t \mid r\, s \mid (t, t') \mid r\, p$	expressions
Ty	$\ni$	A, B, C	$::=$	$\mathsf{Set} \mid \mathsf{El}\, t \mid \mathsf{Fun}\, A\, (\lambda x B) \mid \mathsf{Pair}\, A\, (\lambda x B)$	types
Cxt	$\ni$	Γ	$::=$	$\diamond \mid \Gamma, x : A$	typing contexts

Types $\mathsf{Ty} \subseteq \mathsf{Exp}$ are distinguished expressions. 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 $\lambda a \lambda x x$.

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

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

Typing $\Gamma \vdash t : A$.

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

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

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

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

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

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

Figure 1. MLF_Σ rules for contexts and typing.

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.

Remark 2.1. (Subject reduction fails)
In the context $z : \mathsf{Pair}\, A\, (\lambda x B)$, the η-redex $(z\, \mathsf{L},\ z\, \mathsf{R})$ can be given the non-dependent type $\mathsf{Pair}\, A\, (\lambda\_.\, B[z\, \mathsf{L}/x])$, but the reduct z not. A closer analysis of this problem leads us to rule PAIR-I: the types of s and t do not determine the type of (s, t). If the term s appears in $B[s/x]$, then there are at least two different expressions B_1 and B_2 such that $B_1[s/x] \equiv B_2[s/x] \equiv B[s/x]$, which lead to different types of (s, t).

For the remainder of this section we present properties of MLF_Σ which have easy syntactical proofs. In this, we follow roughly the path outlined by Harper and Pfenning [13]. However, there is a methodological difference: In all judgements $\Gamma \vdash J$, we presuppose $\Gamma \vdash \mathsf{ok}$, which is not true for Harper and Pfenning's presentation of the logical framework.

Lemma 2.1. (Admissible rules)

1. Reflexivity: If $\mathcal{D} :: \Gamma \vdash t : A$ then $\Gamma \vdash t = t : A$.
2. Weakening: If $\mathcal{D} :: \Gamma, \Gamma' \vdash J$ and both $\Gamma \vdash A : \mathsf{Type}$ and $(x : B) \notin (\Gamma, \Gamma')$ for any B, then $\Gamma, x\!:\!A, \Gamma' \vdash J$.
3. Syntactic validity of hypotheses: If $\mathcal{D} :: \Gamma \vdash J$ and $(x\!:\!A) \in \Gamma$ then $\mathcal{D}' :: \Gamma \vdash A : \mathsf{Type}$ and the derivation $\mathcal{D}'$ is shorter than $\mathcal{D}$.
4. Context conversion: If $\mathcal{D} :: \Gamma, x\!:\!A, \Gamma' \vdash J$ and $\Gamma \vdash A = B : \mathsf{Type}$ then $\Gamma, x\!:\!B, \Gamma' \vdash J$.
5. Substitution: If $\mathcal{D} :: \Gamma, x\!:\!A, \Gamma' \vdash J$ and $\Gamma \vdash s : A$ then $\Gamma, \Gamma'[s/x] \vdash J[s/x]$.

Proof:
Each by induction on $\mathcal{D}$. Syntactic validity of hypotheses requires weakening in case CXT-EXT. Substitution requires weakening in case EQ-HYP. The only interesting case for context conversion is EQ-HYP, which needs an application of EQ-CONV. □

Lemma 2.2. (Inversion for types)

1. If $\mathcal{D} :: \Gamma \vdash \mathsf{El}\ t : \mathsf{Type}$ then $\mathcal{D}' :: \Gamma \vdash t : \mathsf{Set}$.

Equivalence, hypotheses, conversion.

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

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

Sets.

$$\textsc{eq-set-f}\ \frac{\Gamma \vdash \mathsf{ok}}{\Gamma \vdash \mathsf{Set} = \mathsf{Set} : \mathsf{Type}} \qquad \textsc{eq-set-e}\ \frac{\Gamma \vdash t = t' : \mathsf{Set}}{\Gamma \vdash \mathsf{El}\, t = \mathsf{El}\, t' : \mathsf{Type}}$$

Dependent functions.

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

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

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

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

$$\textsc{eq-fun-}\eta\ \frac{\Gamma \vdash t : \mathsf{Fun}\, A\, (\lambda x B)}{\Gamma \vdash (\lambda x.\, t\, x) = t : \mathsf{Fun}\, A\, (\lambda x B)}\ x \not\in \mathsf{FV}(t)$$

Dependent pairs.

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

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

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

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

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

Figure 2. MLF_Σ equality rules.

2. Let $c \in \{\mathsf{Fun}, \mathsf{Pair}\}$. If $\mathcal{D} :: \Gamma \vdash c\, A\, (\lambda x B) : \mathsf{Type}$ then $\mathcal{D}_1 :: \Gamma \vdash A : \mathsf{Type}$ and $\mathcal{D}_2 :: \Gamma, x\!:\!A \vdash B : \mathsf{Type}$.

In all cases, the derivations $\mathcal{D}'$, $\mathcal{D}_1$, and $\mathcal{D}_2$ are shorter than $\mathcal{D}$.

Proof:
By cases on $\mathcal{D}$, using syntactic validity of hypotheses (2.1.3) for part 2. □

Lemma 2.3. (Functionality for typing)
Let $\Gamma \vdash s = s' : A$ and $\Gamma \vdash s : A$. If $\mathcal{D} :: \Gamma, x : A, \Gamma' \vdash t : C$ then $\Gamma, \Gamma'[s/x] \vdash t[s/x] = t[s'/x] : C[s/x]$.

Proof:
By induction on $\mathcal{D}$. We spell out some cases:

- In the case of an hypothesis rule, we have $\Gamma, x : A, \Gamma' \vdash \mathsf{ok}$, hence, by the substitution lemma, $\Gamma, \Gamma'[s/x] \vdash \mathsf{ok}$. We consider the following subcases:
  - The used hypothesis is $x : A$. Since all types in $\Gamma'[s/x]$ are wellformed, we can iteratively weaken the assumption of this lemma to obtain the desired $\Gamma, \Gamma'[s/x] \vdash s = s' : A$. Note that $A \equiv A[s/x]$ since x cannot be free in A.
  - The used hypothesis is $(y\!:\!B) \in \Gamma$. Then x cannot be free in B and $\Gamma, \Gamma'[s/x] \vdash y = y : B$ is an instance of rule EQ-HYP.
  - The used hypothesis is $(y : B) \in \Gamma'$. Then $(y : B[s/x]) \in \Gamma'[s/x]$ and we can again use EQ-HYP.

- Case:

$$\textsc{conv}\ \frac{\Gamma, x\!:\!A, \Gamma' \vdash t : B \qquad \Gamma, x\!:\!A, \Gamma' \vdash B = C : \mathsf{Type}}{\Gamma, x\!:\!A, \Gamma' \vdash t : C}$$

$\Gamma, \Gamma'[s/x] \vdash t[s/x] = t[s'/x] : B[s/x]$	induction hypothesis
$\Gamma \vdash s : A$	assumption
$\Gamma, \Gamma'[s/x] \vdash B[s/x] = C[s/x] : \mathsf{Type}$	substitution lemma
$\Gamma, \Gamma'[s/x] \vdash t[s/x] = t[s'/x] : C[s/x]$	rule EQ-CONV

- Case:

$$\mathcal{D} :: \Gamma, x\!:\!A, \Gamma' \vdash \mathsf{Fun}\, B\, \lambda y C : \mathsf{Type}$$

$\mathcal{D}_1 :: \Gamma, x\!:\!A, \Gamma' \vdash B : \mathsf{Type}$	inversion for types
$\quad \Gamma, \Gamma'[s/x] \vdash B[s/x] = B[s'/x] : \mathsf{Type}$	ind. hyp. ($\mathcal{D}_1 < \mathcal{D}$)
$\mathcal{D}_2 :: \Gamma, x\!:\!A, \Gamma', y\!:\!B \vdash C : \mathsf{Type}$	inversion for types
$\quad \Gamma, \Gamma'[s/x], y\!:\!B[s/x] \vdash C[s/x] = C[s'/x] : \mathsf{Type}$	ind. hyp. ($\mathcal{D}_2 < \mathcal{D}$)
$\quad \Gamma, \Gamma'[s/x] \vdash \mathsf{Fun}\ (B[s/x])\ \lambda y.\, C[s/x]$	
$\quad\quad = \mathsf{Fun}\ (B[s'/x])\ \lambda y.\, C[s'/x] : \mathsf{Type}$	rule EQ-FUN-F
$\quad \Gamma, \Gamma'[s/x] \vdash (\mathsf{Fun}\, B\, \lambda y C)[s/x] = (\mathsf{Fun}\, B\, \lambda y C)[s'/x] : \mathsf{Type}$	properties of substitution

- Case:

$$\text{FUN-I}\ \frac{\Gamma, x\!:\!A, \Gamma', y\!:\!B \vdash t : C}{\Gamma, x\!:\!A, \Gamma' \vdash \lambda y t : \mathsf{Fun}\, B\, \lambda y C}$$

$\Gamma, \Gamma'[s/x], y\!:\!B[s/x] \vdash t[s/x] = t[s'/x] : C[s/x]$ — induction hypothesis
$\Gamma, \Gamma'[s/x] \vdash \lambda y.\, t[s/x] = \lambda y.\, t[s'/x] : \mathsf{Fun}\ (B[s/x])\ \lambda y.\, C[s/x]$ — rule EQ-FUN-I
$\Gamma, \Gamma'[s/x] \vdash (\lambda y t)[s/x] = (\lambda y t)[s'/x] : (\mathsf{Fun}\, B\, \lambda y C)[s/x]$ — properties of substitution

- Case:

$$\text{PAIR-E-R}\ \frac{\Gamma, x\!:\!A, \Gamma' \vdash r : \mathsf{Pair}\, B\, \lambda y C}{\Gamma, x\!:\!A, \Gamma' \vdash r\, \mathsf{R} : C[r\, \mathsf{L}/y]}$$

$\Gamma, \Gamma'[s/x] \vdash r[s/x] = r[s'/x] : \mathsf{Pair}\ (B[s/x])\ \lambda y.\, C[s/x]$ — induction hypothesis
$\Gamma, \Gamma'[s/x] \vdash r\, \mathsf{R}[s/x] = r\, \mathsf{R}[s'/x] : (C[s/x])[(r[s/x]\, \mathsf{L})/y]$ — rule EQ-PAIR-E-R
$\Gamma, \Gamma'[s/x] \vdash r\, \mathsf{R}[s/x] = r\, \mathsf{R}[s'/x] : (C[r\, \mathsf{L}/y])[s/x]$ — properties of substitution

□

Lemma 2.4. (Injectivity)

1. If $\mathcal{D} :: \Gamma \vdash \mathsf{Set} = C : \mathsf{Type}$ or $\mathcal{D} :: \Gamma \vdash C = \mathsf{Set} : \mathsf{Type}$ then $C \equiv \mathsf{Set}$.
2. If $\mathcal{D} :: \Gamma \vdash \mathsf{El}\ t = C : \mathsf{Type}$ or $\mathcal{D} :: \Gamma \vdash C = \mathsf{El}\ t : \mathsf{Type}$ then $C \equiv \mathsf{El}\ t'$ and $\Gamma \vdash t = t' : \mathsf{Set}$.
3. Let $c \in \{\mathsf{Fun}, \mathsf{Pair}\}$. If $\mathcal{D} :: \Gamma \vdash c\, A\, (\lambda x B) = C : \mathsf{Type}$ or $\mathcal{D} :: \Gamma \vdash C = c\, A\, (\lambda x B) : \mathsf{Type}$ then $C \equiv c\, A'\, (\lambda x B')$ with $\Gamma \vdash A = A' : \mathsf{Type}$ and $\Gamma, x\!:\!A \vdash B = B' : \mathsf{Type}$.

Proof:
By induction on $\mathcal{D}$. 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 [13]. In the Edinburgh LF with type-level λ it involves a normalization argument and is proven using logical relations [20]. □

Lemma 2.5. (Syntactic validity)

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

Proof:
Simultaneously by induction on $\mathcal{D}$. A few interesting cases are:

- Case:

$$\text{CONV}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A = B : \mathsf{Type}}{\Gamma \vdash t : B}$$

By induction hypothesis (2.), $\Gamma \vdash B : \mathsf{Type}$.

- Case:

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

By inversion (Lemma 2.2) on the induction hypothesis, $\Gamma, x : A \vdash B : \mathsf{Type}$. Also, by rule PAIR-E-L, $\Gamma \vdash r\, \mathsf{L} : A$. Hence, $\Gamma \vdash B[r\, \mathsf{L}/x] : \mathsf{Type}$ by substitution.

- Case:

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

By induction hypothesis, $\Gamma \vdash A, A' : \mathsf{Type}$ and $\Gamma, x : A \vdash B, B' : \mathsf{Type}$. We infer $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}$ directly, by FUN-F, whereas $\Gamma \vdash \mathsf{Fun}\, A'\, (\lambda x B') : \mathsf{Type}$ follows only after we converted the type of x in the context to A'.

- Case:

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

$\Gamma \vdash s, s' : A$	induction hypothesis
$\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}$	induction hypothesis
$\Gamma, x\!:\!A \vdash B : \mathsf{Type}$	inversion for types
$\Gamma \vdash B[s/x] : \mathsf{Type}$	substitution lemma
$\Gamma \vdash B[s/x] = B[s'/x] : \mathsf{Type}$	functionality for typing
$\Gamma \vdash r, r' : \mathsf{Fun}\, A\, (\lambda x B)$	induction hypothesis
$\Gamma \vdash r\, s : B[s/x]$	rule FUN-E
$\Gamma \vdash r'\, s' : B[s'/x]$	rule FUN-E
$\Gamma \vdash r'\, s' : B[s/x]$	rules EQ-SYM, CONV

- Case:

$$\textsc{eq-fun-}\beta\ \frac{\Gamma, x\!:\!A \vdash t = t' : B \qquad \Gamma \vdash s = s' : A}{\Gamma \vdash (\lambda x t)\, s = t'[s'/x] : B[s/x]}$$

By induction hypothesis, $\Gamma \vdash s : A$ and $\Gamma, x : A \vdash B : \mathsf{Type}$, hence we get the first goal $\Gamma \vdash B[s/x] : \mathsf{Type}$ by the substitution lemma. By functionality for typing we also have $\Gamma \vdash B[s/x] = B[s'/x] : \mathsf{Type}$. Another induction hypothesis is $\Gamma, x\!:\!A \vdash t : B$ from which we obtain the second goal $\Gamma \vdash t[s/x] : B[s/x]$ again by substitution. Using substitution on the induction hypotheses $\Gamma, x : A \vdash t' : B$ and $\Gamma \vdash s' : A$ entails $\Gamma \vdash t'[s'/x] : B[s'/x]$ and we can use our derived type equality with EQ-SYM and CONV to finally arrive at $\Gamma \vdash t'[s'/x] : B[s/x]$.

- Case:

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

W. l. o. g., x is not bound by context Γ. By induction hypothesis, $\Gamma \vdash t, t' : \mathsf{Fun}\, A\, (\lambda x B)$ and $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}$. By inversion for types, $\Gamma \vdash A : \mathsf{Type}$, hence we can apply weakening to obtain $\Gamma, x\!:\!A \vdash t : \mathsf{Fun}\, A\, (\lambda x B)$. This entails $\Gamma \vdash \lambda x.\, t\, x : \mathsf{Fun}\, A\, (\lambda x B)$.

□

Using syntactic validity, the functionality lemma (2.3) needs fewer hypotheses:

Corollary 2.1. (Functionality for typing)
If $\Gamma \vdash s = s' : A$ and $\Gamma, x\!:\!A, \Gamma' \vdash t : C$ then $\Gamma, \Gamma'[s/x] \vdash t[s/x] = t[s'/x] : C[s/x]$.

Lemma 2.6. (Functionality for equality)
If $\Gamma, x\!:\!A, \Gamma' \vdash t = t' : C$ and $\Gamma \vdash s = s' : A$ then $\Gamma, \Gamma'[s/x] \vdash t[s/x] = t'[s'/x] : C[s/x]$.

Proof:
Direct (cf. Harper and Pfenning [13]).

$\Gamma \vdash s : A$	syntactic validity
$\Gamma, \Gamma[s/x] \vdash t[s/x] = t'[s/x] : C[s/x]$	substitution lemma
$\Gamma, x\!:\!A, \Gamma' \vdash t' : C$	syntactic validity
$\Gamma, \Gamma[s/x] \vdash t'[s/x] = t'[s'/x] : C[s/x]$	functionality for typing
$\Gamma, \Gamma[s/x] \vdash t[s/x] = t'[s'/x] : C[s/x]$	rule EQ-TRANS

□

Lemma 2.7. (Inversion of Typing)
Let $C \not\equiv \mathsf{Type}$.

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

Proof:
By induction on $\mathcal{D}$. For each shape of term t in $\Gamma \vdash t : C$, there are two matching rules. One is the introduction resp. elimination rule fitting t, which entails the inversion property trivially. The other one is rule CONV:

- Case:
$$\textsc{conv}\ \frac{\Gamma \vdash \lambda xt : C \qquad \Gamma \vdash C = C' : \mathsf{Type}}{\Gamma \vdash \lambda xt : C'}$$
By induction hypothesis $C \equiv \mathsf{Fun}\, A\, (\lambda xB)$ and $\Gamma, x : A \vdash t : B$. By injectivity, $C' \equiv \mathsf{Fun}\, A'\, (\lambda xB')$ with $\Gamma \vdash A = A' : \mathsf{Type}$ and $\Gamma, x : A \vdash B = B' : \mathsf{Type}$. By conversion and context conversion we conclude $\Gamma, x\!:\!A' \vdash t : B'$.

- Case:

$$\textsc{conv}\ \frac{\Gamma \vdash r\,s : C \qquad \Gamma \vdash C = C' : \mathsf{Type}}{\Gamma \vdash r\,s : C'}$$

  By induction hypothesis $\Gamma \vdash r : \mathsf{Fun}\,A\,(\lambda x B)$ for some A, B with $\Gamma \vdash s : A$ and $\Gamma \vdash B[s/x] = C : \mathsf{Type}$. We infer $\Gamma \vdash B[s/x] = C' : \mathsf{Type}$ by transitivity.

- Case:

$$\textsc{conv}\ \frac{\Gamma \vdash r\,\mathsf{L} : A \qquad \Gamma \vdash A = A' : \mathsf{Type}}{\Gamma \vdash r\,\mathsf{L} : A'}$$

  By induction hypothesis, $\Gamma \vdash r : \mathsf{Pair}\,A\,(\lambda x B)$. Syntactic validity (Lemma 2.5), inversion for types (Lemma 2.2), and reflexivity entail $\Gamma, x : A \vdash B = B : \mathsf{Type}$, hence, $\Gamma \vdash \mathsf{Pair}\,A\,(\lambda x B) = \mathsf{Pair}\,A'\,(\lambda x B) : \mathsf{Type}$ by rule EQ-PAIR-F. The desired $\Gamma \vdash r : \mathsf{Pair}\,A'\,(\lambda x B)$ follows by CONV.

□

Remark 2.2. (Weaker inversion property for left projection)
The statement “if $\Gamma \vdash r\mathsf{L} : C$ then $\Gamma \vdash r : \mathsf{Pair}\,A\,(\lambda x B)$ and $\Gamma \vdash A = C : \mathsf{Type}$” can be proven without reference to syntactic validity.

3. Algorithmic Presentation

In this section, we present algorithms for deciding equality and for type-checking. The goal of this article is to show these algorithms sound and complete.

Syntactic classes. The algorithms work 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.

WElim	$\ni$	e	$::=$	$s \mid p$	eliminations
WNe	$\ni$	n	$::=$	$c \mid x \mid n\,e$	neutral expressions
WFun	$\ni$	w_f	$::=$	$n \mid \lambda x t$	weak head function values
WPair	$\ni$	w_p	$::=$	$n \mid (t, t')$	weak head pair values
WVal	$\ni$	w	$::=$	$w_f \mid w_p$	weak head values

Note that types $A \in \mathsf{Ty} \subseteq \mathsf{WNe}$ are always neutral weak head values.

Weak head evaluation. We define simultaneously two judgements:

$$\_ \searrow \_ \quad \subseteq \quad \mathsf{Exp} \times \mathsf{WVal}$$
$$\_@\_ \searrow \_ \quad \subseteq \quad \mathsf{WVal} \times \mathsf{WElim} \times \mathsf{WVal}$$

Weak head evaluation $t \searrow w$.

$$\text{EVAL-C}\ \frac{}{c \searrow c} \qquad \text{EVAL-VAR}\ \frac{}{x \searrow x}$$

$$\text{EVAL-FUN-I}\ \frac{}{\lambda xt \searrow \lambda xt} \qquad \text{EVAL-FUN-E}\ \frac{r \searrow w_f \qquad w_f@s \searrow w}{r\, s \searrow w}$$

$$\text{EVAL-PAIR-I}\ \frac{}{(t, t') \searrow (t, t')} \qquad \text{EVAL-PAIR-E}\ \frac{r \searrow w_p \qquad w_p@p \searrow w}{r\, p \searrow w}$$

Active elimination $w@e \searrow w'$.

$$\text{ELIM-NE}\ \frac{}{n@e \searrow n\, e} \qquad \text{ELIM-FUN}\ \frac{t[s/x] \searrow w}{(\lambda xt)@s \searrow w}$$

$$\text{ELIM-PAIR-L}\ \frac{t \searrow w}{(t, t')@\mathsf{L} \searrow w} \qquad \text{ELIM-PAIR-R}\ \frac{t' \searrow w}{(t, t')@\mathsf{R} \searrow w}$$

Weak head evaluation $t \searrow w$ is equivalent to multi-step weak head reduction to normal form.

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

$$\text{AQ-C}\ \frac{}{c \sim c} \qquad \text{AQ-VAR}\ \frac{}{x \sim x}$$

$$\text{AQ-NE-FUN}\ \frac{n \sim n' \qquad s{\downarrow} \sim s'{\downarrow}}{n\, s \sim n'\, s'} \qquad \text{AQ-NE-PAIR}\ \frac{n \sim n'}{n\, p \sim n'\, p}$$

$$\text{AQ-EXT-FUN}\ \frac{w_f@x \sim w'_f@x}{w_f \sim w'_f}\ x \notin \mathsf{FV}(w_f, w'_f)$$

$$\text{AQ-EXT-PAIR}\ \frac{w_p@\mathsf{L} \sim w'_p@\mathsf{L} \qquad w_p@\mathsf{R} \sim w'_p@\mathsf{R}}{w_p \sim w'_p}$$

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 [13] and Sarnat [19], 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 3.1. In contrast to the corresponding equality for λ-terms without pairs [6] (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 $\lambda x.\, n\, x \sim (n\mathsf{L}, n\mathsf{R})$.

Type checking. In the following, we give a bidirectional type checking algorithm [7, 17, 13] for normal terms. We define simultaneously two judgements:

$$\begin{array}{rcl} \_ \vdash \_ \Downarrow \_ & \subseteq & \mathsf{Cxt} \times \mathsf{Exp} \times (\mathsf{Ty} \cup \{\mathsf{Type}\}) \\ \_ \vdash \_ \Uparrow \_ & \subseteq & \mathsf{Cxt} \times \mathsf{Exp} \times \mathsf{Ty} \end{array}$$

The judgement $\Gamma \vdash t \Downarrow A$ infers type A from neutral terms t, $\Gamma \vdash t \Uparrow C$ checks whether the β-normal term t has type C, and $\Gamma \vdash A \Downarrow \mathsf{Type}$ identifies wellformed types $A \in \mathsf{Ty}$.

Type inference $\Gamma \vdash t \Downarrow A$.

$$\text{INF-VAR}\ \frac{}{\Gamma \vdash x \Downarrow \Gamma(x)} \qquad \text{INF-FUN-E}\ \frac{\Gamma \vdash r \Downarrow \mathsf{Fun}\, A\, (\lambda x B) \qquad \Gamma \vdash s \Uparrow A}{\Gamma \vdash r\, s \Downarrow B[s/x]}$$

$$\text{INF-PAIR-E-L}\ \frac{\Gamma \vdash r \Downarrow \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash r\mathsf{L} \Downarrow A} \qquad \text{INF-PAIR-E-R}\ \frac{\Gamma \vdash r \Downarrow \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash r\mathsf{R} \Downarrow B[r\mathsf{L}/x]}$$

Type checking $\Gamma \vdash t \Uparrow A$.

$$\text{CHK-INF}\ \frac{\Gamma \vdash r \Downarrow A \qquad A \sim B}{\Gamma \vdash r \Uparrow B} \qquad \text{CHK-FUN-I}\ \frac{\Gamma, x\!:\!A \vdash t \Uparrow B}{\Gamma \vdash \lambda x t \Uparrow \mathsf{Fun}\, A\, (\lambda x B)}$$

$$\text{CHK-PAIR-I}\ \frac{\Gamma \vdash t \Uparrow A \qquad \Gamma \vdash t' \Uparrow B[t/x]}{\Gamma \vdash (t, t') \Uparrow \mathsf{Pair}\, A\, (\lambda x B)}$$

Type well-formedness $\Gamma \vdash A \Downarrow \mathsf{Type}$.

$$\text{CHK-SET-F}\ \frac{}{\Gamma \vdash \mathsf{Set} \Downarrow \mathsf{Type}} \qquad \text{CHK-SET-E}\ \frac{\Gamma \vdash t \Uparrow \mathsf{Set}}{\Gamma \vdash \mathsf{El}\, t \Downarrow \mathsf{Type}}$$

$$\text{CHK-DEP-F}\ \frac{\Gamma \vdash A \Downarrow \mathsf{Type} \qquad \Gamma,\ x\!:\!A \vdash B \Downarrow \mathsf{Type}}{\Gamma \vdash c\, A\, (\lambda x B) \Downarrow \mathsf{Type}}\ c \in \{\mathsf{Fun}, \mathsf{Pair}\}$$

Besides the fact that in both judgements and in the context, types are always in weak head normal form, the algorithm has the invariant that every expression which is evaluated has been checked before. This principle ensures termination, a byproduct of soundness which we show in the next section.

The algorithms in this section have been prototypically implemented in Haskell using explicit substitutions [1].

4. Soundness

The soundness proofs for conversion and type-checking in this section are entirely syntactical and rely crucially on injectivity of El, Fun and Pair (Lemma 2.4) and inversion of typing (Lemma 2.7). First, we show soundness of weak head evaluation, which subsumes subject reduction.

Lemma 4.1. (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.

- Case:

$$\textsc{eval-fun-e}\ \frac{r \searrow w_f \qquad w_f@s \searrow w}{r\, s \searrow w}$$

$\Gamma \vdash r\, s : C$	hypothesis
$\Gamma \vdash r : \mathsf{Fun}\, A\, (\lambda x B)$	&
$\Gamma \vdash s : A$	&
$\Gamma \vdash B[s/x] = C : \mathsf{Type}$	inversion
$\Gamma \vdash r = w_f : \mathsf{Fun}\, A\, (\lambda x B)$	first ind. hyp.
$\Gamma \vdash r\ s = w_f\ s : B[s/x]$	EQ-FUN-E
$\Gamma \vdash w_f\, s : C$	syntactic validity, CONV
$\Gamma \vdash w_f\, s = w : C$	second ind. hyp.
$\Gamma \vdash r\, s = w : C$	EQ-TRANS

- Case:

$$\textsc{elim-fun}\ \frac{t[s/x] \searrow w}{(\lambda x t)@s \searrow w}$$

$\Gamma \vdash (\lambda x t)\ s : C$	hypothesis
$\Gamma \vdash \lambda x t : \mathsf{Fun}\, A\, (\lambda x B)$	&
$\Gamma \vdash s : A$	&
$\Gamma \vdash B[s/x] = C : \mathsf{Type}$	inversion
$\Gamma, x\!:\!A \vdash t : B$	inversion
$\Gamma \vdash (\lambda x t)\, s = t[s/x] : B[s/x]$	EQ-FUN-β
$\Gamma \vdash (\lambda x t)\ s = t[s/x] : C$	EQ-CONV
$\Gamma \vdash t[s/x] : C$	syntactic validity
$\Gamma \vdash t[s/x] = w : C$	ind. hyp.
$\Gamma \vdash (\lambda x t)\ s = w : C$	EQ-TRANS

□

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 4.2. (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 4.1) and transitivity. We prove the first two propositions simultaneously by induction on $\mathcal{D}$.

- Case:

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

$\Gamma \vdash n\,s : C$	hypothesis
$\Gamma \vdash n : \mathsf{Fun}\,A\,(\lambda xB)$	&
$\Gamma \vdash s : A$	&
$\Gamma \vdash B[s/x] = C : \mathsf{Type}$	IM-NE-FUN, inversion
$\Gamma \vdash n'\,s' : C'$	hypothesis
$\Gamma \vdash n' : \mathsf{Fun}\,A'\,(\lambda xB')$	&
$\Gamma \vdash s' : A'$	&
$\Gamma \vdash B'[s'/x] = C' : \mathsf{Type}$	IM-NE-FUN, inversion
$\Gamma \vdash n = n' : \mathsf{Fun}\,A\,(\lambda xB)$	&
$\Gamma \vdash \mathsf{Fun}\,A\,(\lambda xB) = \mathsf{Fun}\,A'\,(\lambda xB') : \mathsf{Type}$	first ind. hyp.
$\Gamma \vdash A = A' : \mathsf{Type}$	injectivity
$\Gamma \vdash s' : A$	rule CONV
$\Gamma \vdash s = s' : A$	second ind. hyp. (3.)
$\Gamma, x\!:\!A \vdash B = B' : \mathsf{Type}$	injectivity
$\Gamma \vdash B[s/x] = B'[s'/x] : \mathsf{Type}$	functionality
$\Gamma \vdash C = C' : \mathsf{Type}$	transitivity, symmetry
$\Gamma \vdash n\,s = n'\,s' : C$	EQ-FUN-E

- Case (instance of AQ-EXT-FUN with $v_f \equiv \lambda xt$ and $v'_f = n$):

$$\textsc{aq-ext-fun}\ \frac{(\lambda xt)@x \searrow w \qquad w \sim n\,x \qquad n@x \searrow n\,x}{\lambda xt \sim n}\ x \notin \mathsf{FV}(n)$$

$\Gamma \vdash \lambda x t : C$	hypothesis
$C \equiv \mathsf{Fun}\, A\, (\lambda x B)$	&
$\Gamma, x{:}A \vdash t : B$	inversion
$t \searrow w$	assumption
$\Gamma, x{:}A \vdash t = w : B$	eval. sound (Lemma 4.1)
$\Gamma \vdash n : \mathsf{Fun}\, A\, (\lambda x B)$	hypothesis, def. C
$\Gamma \vdash \lambda x.\, n\, x = n : \mathsf{Fun}\, A\, (\lambda x B)$	EQ-FUN-η, $x \notin \mathsf{FV}(n)$
$\Gamma, x{:}A \vdash n : \mathsf{Fun}\, A\, (\lambda x B)$	weakening
$\Gamma, x{:}A \vdash n\, x : B$	FUN-E, HYP
$\Gamma, x{:}A \vdash w = n\, x : B$	ind. hyp.
$\Gamma, x{:}A \vdash t = n\, x : B$	transitivity (EQ-TRANS)
$\Gamma \vdash \lambda x t = \lambda x.\, n\, x : \mathsf{Fun}\, A\, (\lambda x B)$	EQ-FUN-I
$\Gamma \vdash \lambda x t = n : C$	EQ-TRANS

□

It follows that also type checking is correct, if started in a correct context and with a well-formed type.

Theorem 4.1. (Soundness of bidirectional type checking)

1. If $\mathcal{D} :: \Gamma \vdash t \Downarrow A$ and $\Gamma \vdash \mathsf{ok}$ then $\Gamma \vdash t : A$ and $A \in \mathsf{Ty} \cup \{\mathsf{Type}\}$.

2. If $\mathcal{D} :: \Gamma \vdash t \Uparrow C$ and $\Gamma \vdash C : \mathsf{Type}$, then $\Gamma \vdash t : C$.

Proof:
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 along the lines of Goguen [9, 10], 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

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

Herein, we use the following entities:

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

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:

$$
\begin{array}{lrcll}
\textsc{den-const} & c\rho & = & c & \text{if } c \in \mathsf{Const} \\
\textsc{den-var} & x\rho & = & \rho(x) & \\
\textsc{den-fun-e} & (r\,s)\rho & = & r\rho\,(s\rho) & \\
\textsc{den-pair-e} & (r\,p)\rho & = & r\rho\,p & \\
 & & & & \\
\textsc{den-}\beta & t\rho & = & t'\rho & \text{if } t =_\beta t' \\
\textsc{den-irr} & t\rho & = & t\rho' & \text{if } \rho(x) = \rho'(x) \text{ for all } x \in \mathsf{FV}(t)
\end{array}
$$

This notion of model, which does not admit weak (ξ) and strong extensionality rules, but still has the substitution property (see Lemma 5.1), is an invention of Benzmüller, Brown, and Kohlhase [5, Def. 3.18]. They consider it in the context of typed λ-calculus as a basis for a model of higher-order logics. We have adapted it to the untyped setting, extended it by projections and added injectivity for the type constructors.

The following laws for β are admissible:

$$
\begin{array}{lrcl}
\textsc{den-fun-}\beta & (\lambda x t)\rho\ v & = & t(\rho, x = v) \\
 & & & \\
\textsc{den-pair-}\beta\textsc{-l} & (r, s)\rho\,\mathsf{L} & = & r\rho \\
\textsc{den-pair-}\beta\textsc{-r} & (r, s)\rho\,\mathsf{R} & = & s\rho
\end{array}
$$

Proof:
We show soundness of DEN-FUN-β.

$$
\begin{array}{cll}
 & (\lambda x t)\rho\ v & \\
= & (\lambda x t)\rho\ x(\rho, x = v) & \textsc{den-var} \\
= & (\lambda x t)(\rho, x = v)\ x(\rho, x = v) & \textsc{den-irr} \\
= & ((\lambda x t)\ x)(\rho, x = v) & \textsc{den-fun-e} \\
= & t(\rho, x = v) & \textsc{den-}\beta.
\end{array}
$$

□

The subsitution property is a consequence of β-equality:

Lemma 5.1. (Soundness of substitution)
$(t[s/x])\rho = t(\rho, x\!=\!s\rho)$.

Proof:
$(t[s/x])\rho = ((\lambda x t)\, s)\rho = (\lambda x t)\rho\ s\rho = t(\rho, x\!=\!s\rho)$. □

Remark 5.1. (Comparison to standard λ-model)
Barendregt et. al. [4] axiomatize a λ-model by DEN-VAR, DEN-FUN-E, DEN-FUN-β, and weak extensionality:

$$\text{DEN-FUN-}\xi \quad (\lambda x t)\rho = (\lambda x' t')\rho' \quad \text{ if } t(\rho, x\!=\!v) = t'(\rho', x'\!=\!v) \text{ for all } v \in \mathsf{D}.$$

Then irrelevance (DEN-IRR) and substitution (Lemma 5.1) are provable by induction on t, and DEN-β follows. However, in Benzmüller, Brown, and Kohlhase's notion of λ-model, weak extensionality is not admissible: Consider D to be closed λ-terms over the empty set of constants modulo $\beta\eta$-equality, where denotation $t\rho$ is interpreted as parallel substitution. This clearly models DEN-VAR, DEN-FUN-E, DEN-β, and DEN-IRR, hence, is a model in the sense of Benzmüller, Brown, and Kohlhase. But Plotkin [18] showed that the ω-rule does not hold in $\lambda\beta\eta$-calculus, i.e., there are (closed) terms r, s such that for all closed terms t it holds that $r\,t =_{\beta\eta} s\,t$, but not $r =_{\beta\eta} s$. It follows for a fresh variable x that $(r\,x)(\rho, x = t) = (s\,x)(\rho, x = t)$ for all $t \in \mathsf{D}$, but not $(\lambda x.\, r\,x)\rho = (\lambda x.\, s\,x)\rho$, hence ξ fails. Thus, Benzmüller, Brown, and Kohlhase's notion of a model is strictly weaker than the standard one, even in the untyped setting. For typed models, this has already been demonstrated [5, Example 5.8], but the counterexample provided does not carry over to untyped models.

Injectivity laws. We require the type constructors in the model to be injective. This is necessary since we want to interpret distinguished elements of $\mathcal{D}$, the *types*, as semantical types later. In the following, let $c, c' \in \{\mathsf{Fun}, \mathsf{Pair}\}$.

DEN-SET-NOT-EL	Set	$\neq$	$\mathsf{El}\ v$	
DEN-SET-NOT-DEP	Set	$\neq$	$c\ V\ F$	
DEN-EL-NOT-DEP	$\mathsf{El}\ v$	$\neq$	$c\ V\ F$	
DEN-EL-INJ	$\mathsf{El}\ v$	$=$	$\mathsf{El}\ v'$	implies $v = v'$
DEN-DEP-INJ	$c\ V\ F$	$=$	$c'\ V'\ F'$	implies $c = c'$ and $V = V'$ and $F = F'$

5.2. PER Models

In the definition of PER models, we follow a paper of the second author with Pollack and Takeyama [8] and Vaux [21]. 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 $t\rho$.

Relations on D. Let Rel denote the set of relations over D. If $\mathcal{A} \in \mathsf{Rel}$, we say $v \in \mathcal{A}$ if v is in the carrier of $\mathcal{A}$, i. e., $(v, w) \in \mathcal{A}$ or $(w, v) \in \mathcal{A}$ for some $w \in \mathsf{D}$.

Partial equivalence relation (PER). A PER is a symmetric and transitive relation. Let $\mathsf{Per} \subseteq \mathsf{Rel}$ denote the set of PERs over D. If $\mathcal{A} \in \mathsf{Per}$, we write $v = v' \in \mathcal{A}$ if $(v, v') \in \mathcal{A}$. For $\mathcal{A}$ a PER, $v \in \mathcal{A}$ means $v = v \in \mathcal{A}$. 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. 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 $\mathcal{A} \in \mathsf{Rel}$ and $\mathcal{F} \in \mathcal{A} \to \mathsf{Rel}$. We define $\mathcal{F}un(\mathcal{A}, \mathcal{F}), \mathcal{P}air(\mathcal{A}, \mathcal{F}) \in \mathsf{Rel}$:

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

Lemma 5.2. ($\mathcal{F}un$ and $\mathcal{P}air$ operate on PERs)
If $\mathcal{A} \in \mathsf{Per}$ and $\mathcal{F} \in \mathsf{Fam}(\mathcal{A})$ then $\mathcal{F}un(\mathcal{A}, \mathcal{F}), \mathcal{P}air(\mathcal{A}, \mathcal{F}) \in \mathsf{Per}$.

In the following, assume some $\mathcal{S}et \in \mathsf{Per}$ and some $\mathcal{E}\ell \in \mathsf{Fam}(\mathcal{S}et)$.

Semantical types. 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 $\mathcal{F}un([V], v \mapsto [F\ v])$.

$\mathsf{Pair}\ V\ F = \mathsf{Pair}\ V'\ F' \in \mathcal{T}ype$ 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 $\mathcal{P}air([V], v \mapsto [F\ v])$.

This definition is possible by the injectivity laws. Notice that in the last two clauses, we have

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

Remark 5.2. $\mathcal{T}ype$ and $[\_]$ are an instance of an inductive-recursive definition. A formulation alternative via a relation which is not a priori a PER, and a partial function, is given in Appendix C.

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:

$$\text{SEM-CXT-EMPTY}\ \frac{}{\diamond \in \mathcal{C}xt}$$

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

Theorem 5.1. (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:
Simultaneously by induction on $\mathcal{D}$, using lemma 5.1.

- Case:

$$\text{FUN-I}\ \frac{\Gamma, x\!:\!A \vdash t : B}{\Gamma \vdash \lambda xt : \mathsf{Fun}\, A\, (\lambda xB)}$$

$(\Gamma, x\!:\!A) \in \mathcal{C}xt$	ind. hyp. (*)
$\Gamma \in \mathcal{C}xt$	inversion
$\rho = \rho' \in [\Gamma]$	assumption
$A\rho = A\rho' \in \mathcal{T}ype$	from (*)
$v = v' \in [A\rho]$	assumption (v, v' arbitrary)
$(\rho, x\!=\!v) = (\rho', x\!=\!v') \in [\Gamma, x\!:\!A]$	def. $[\Gamma, x\!:\!A]$
$B(\rho, x\!=\!v) = B(\rho', x\!=\!v') \in \mathcal{T}ype$	ind. hyp.
$(\lambda xB)\rho\, v = (\lambda xB)\rho'\, v' \in \mathcal{T}ype$	DEN-FUN-β
$(\mathsf{Fun}\, A\, \lambda xB)\rho = (\mathsf{Fun}\, A\, \lambda xB)\rho' \in \mathcal{T}ype$	def. $\mathcal{T}ype$, DEN-FUN-E, DEN-CONST
$t(\rho, x\!=\!v) = t(\rho', x\!=\!v') \in [B(\rho, x\!=\!v)]$	ind. hyp.
$(\lambda xt)\rho\, v = (\lambda xt)\rho'\, v' \in [(\lambda xB)\rho\, v]$	DEN-FUN-β
$(\lambda xt)\rho = (\lambda xt)\rho' \in [(\mathsf{Fun}\, A\, \lambda xB)\rho]$	def. $\mathcal{F}un$, DEN-FUN-E, DEN-CONST

- Case:

$$\text{FUN-E}\ \frac{\Gamma \vdash r : \mathsf{Fun}\, A\, (\lambda xB) \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B[s/x]}$$

$\Gamma \in \mathcal{C}xt$ ind. hyp.

$\mathsf{Fun}\ (A\rho)\ ((\lambda x.B)\rho) = \mathsf{Fun}\ (A\rho')\ ((\lambda x.B)\rho') \in \mathcal{T}ype$ ind. hyp.
$s\rho = s\rho' \in [A\rho]$ ind. hyp.
$B(\rho, x\!=\!s\rho) = B(\rho', x\!=\!s\rho') \in \mathcal{T}ype$ def. $\mathcal{T}ype$
$(B[s/x])\rho = (B[s/x])\rho' \in \mathcal{T}ype$ subst. (Lemma 5.1)

$r\rho = r\rho' \in \mathcal{F}un([A\rho], v \mapsto [B(\rho, x\!=\!v)])$ ind. hyp.
$r\rho\,(s\rho) = r\rho'\,(s\rho') \in [B(\rho, x\!=\!s\rho)]$ def. $\mathcal{F}un$
$(r\,s)\rho = (r\,s)\rho' \in [(B[s/x])\rho]$ DEN-FUN-E, Lemma 5.1

- Case:

$$\text{EQ-FUN-}\beta\ \frac{\Gamma, x\!:\!A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda xt)\, s = t[s/x] : B[s/x]}$$

$\Gamma \in \mathcal{C}xt$ ind. hyp.

$\rho = \rho' \in [\Gamma]$ assumption
$A\rho = A\rho' \in \mathcal{T}ype$ ind. hyp.
$s\rho = s\rho' \in [A\rho]$ ind. hyp.
$(\rho, x\!=\!s\rho) = (\rho', x\!=\!s\rho') \in [\Gamma, x\!:\!A]$ def. $[\Gamma, x\!:\!A]$
$t(\rho, x\!=\!s\rho) = t(\rho', x\!=\!s\rho') \in [B(\rho, x\!=\!s\rho)]$ ind. hyp.
$(\lambda xt)\rho\ (s\rho) = (t[s/x])\rho' \in [(B[s/x])\rho]$ DEN-FUN-β, subst.

$B(\rho, x\!=\!s\rho) = B(\rho', x\!=\!s\rho') \in \mathcal{T}ype$ ind. hyp.
$(B[s/x])\rho = (B[s/x])\rho' \in \mathcal{T}ype$ subst. (Lemma 5.1)

- Case:

$$\text{EQ-FUN-}\eta\ \frac{\Gamma \vdash t : \mathsf{Fun}\, A\, (\lambda xB)}{\Gamma \vdash (\lambda x.\, t\, x) = t : \mathsf{Fun}\, A\, (\lambda xB)}\ x \notin \mathsf{FV}(t)$$

$\Gamma \in \mathcal{C}xt$	ind. hyp.
$\rho = \rho' \in [\Gamma]$	assumption
$(\mathsf{Fun}\, A\, \lambda x B)\rho = (\mathsf{Fun}\, A\, \lambda x B)\rho' \in \mathcal{T}ype$	ind. hyp.
$A\rho = A\rho' \in \mathcal{T}ype$	inversion on $\mathcal{T}ype$
$v = v' \in [A\rho]$	assumption (v, v' arbitrary)
$t\rho = t\rho' \in [(\mathsf{Fun}\, A\, \lambda x B)\rho]$	ind. hyp.
$t\rho\, v = t\rho'\, v' \in [(\lambda x B)\rho\, v]$	def. $\mathcal{F}un$
$t(\rho, x{=}v)\, v = t\rho'\, v' \in [(\lambda x B)\rho\, v]$	irrelevance DEN-IRR
$(t\, x)(\rho, x{=}v) = t\rho'\, v' \in [(\lambda x B)\rho\, v]$	DEN-FUN-E, DEN-VAR
$(\lambda x.\, t\, x)\rho\, v = t\rho'\, v' \in [(\lambda x B)\rho\, v]$	DEN-FUN-β
$(\lambda x.\, t\, x)\rho = t\rho' \in [(\mathsf{Fun}\, A\, \lambda x B)\rho]$	since v, v' arb.

- Case:

$$\text{EQ-PAIR-}\eta\ \frac{\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)}{\Gamma \vdash (r\, \mathsf{L},\ r\, \mathsf{R}) = r : \mathsf{Pair}\, A\, (\lambda x B)}$$

$\Gamma \in \mathcal{C}xt$	ind. hyp.
$\rho = \rho' \in [\Gamma]$	assumption
$(\mathsf{Pair}\, A\, \lambda x B)\rho = (\mathsf{Pair}\, A\, \lambda x B)\rho' \in \mathcal{T}ype$	ind. hyp.
$r\rho = r\rho' \in [(\mathsf{Pair}\, A\, \lambda x B)\rho]$	ind. hyp.
$(r\, \mathsf{L})\rho = r\rho'\, \mathsf{L} \in [A\rho]$	def. $\mathcal{P}air$, DEN-PAIR-E
$(r\, \mathsf{L},\ r\, \mathsf{R})\rho\, \mathsf{L} = r\rho'\, \mathsf{L} \in [(\mathsf{Pair}\, A\, \lambda x B)\rho]$	DEN-PAIR-β-L
$(r\, \mathsf{R})\rho = r\rho'\, \mathsf{R} \in [(\lambda x B)\rho\, (r\, \mathsf{L})\rho]$	def. $\mathcal{P}air$, DEN-PAIR-E
$(r\, \mathsf{L},\ r\, \mathsf{R})\rho\, \mathsf{R} = r\rho'\, \mathsf{R} \in [(\lambda x B)\rho\, ((r\, \mathsf{L},\ r\, \mathsf{R})\rho\, \mathsf{L})]$	DEN-PAIR-β-R
$(r\, \mathsf{L},\ r\, \mathsf{R})\rho = r\rho' \in [(\mathsf{Pair}\, A\, \lambda x B)\rho]$	def. $\mathcal{P}air$

□

5.4. Safe Types

We define an abstract notion of *safety*, similar to what Vaux calls "saturation" [21]. A PER is safe if it lies between a PER $\mathcal{N}$ on *neutral* expressions and a PER $\mathcal{S}$ on *safe* expressions [22]. 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	$\mathcal{N} \subseteq \mathcal{S} = \mathcal{S}_{fun} \cup \mathcal{S}_{pair}$	
SAFE-NE-FUN	$u\,v = u'\,v' \in \mathcal{N}$	if $u = u' \in \mathcal{N}$ and $v = v' \in \mathcal{S}$
SAFE-NE-PAIR	$u\,p = u'\,p \in \mathcal{N}$	if $u = u' \in \mathcal{N}$
SAFE-EXT-FUN	$v = v' \in \mathcal{S}_{fun}$	if $v\,u = v'\,u' \in \mathcal{S}$ for all $u = u' \in \mathcal{N}$
SAFE-EXT-PAIR	$v = v' \in \mathcal{S}_{pair}$	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 5.3. (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:

SAFE-NE-FUN	$\mathcal{N} \subseteq \mathcal{F}un(\mathcal{S}, \_ \mapsto \mathcal{N})$
SAFE-NE-PAIR	$\mathcal{N} \subseteq \mathcal{P}air(\mathcal{N}, \_ \mapsto \mathcal{N})$
SAFE-EXT-FUN	$\mathcal{F}un(\mathcal{N}, \_ \mapsto \mathcal{S}) \subseteq \mathcal{S}_{fun}$
SAFE-EXT-PAIR	$\mathcal{P}air(\mathcal{S}, \_ \mapsto \mathcal{S}) \subseteq \mathcal{S}_{pair}$

□

Lemma 5.4. (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 5.3. □

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 $=_\beta$.

$$
\begin{array}{lcl}
\mathsf{D} & := & \mathsf{Exp}/\!=_\beta \\
\overline{r} \cdot \overline{s} & := & \overline{r\,s} \\
\overline{r}\,\mathsf{L} & := & \overline{r\,\mathsf{L}} \\
\overline{r}\,\mathsf{R} & := & \overline{r\,\mathsf{R}} \\
t\rho & := & \overline{t[\rho]}
\end{array}
$$

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.

Lemma 6.1. $\mathsf{Exp}/=_\beta$ is a λ model in the sense of the last section.

Proof:
We have to show that all operations are well-defined. For application, consider pairs of equivalent members $r =_\beta r'$ and $s =_\beta s'$. Since $r\,s =_\beta r'\,s'$, application is well-defined. The projections are similarly easy. For the denotation operation, let t a term with $FV(t) = \vec{x}$. We assume two equivalent valuations ρ and ρ', meaning that $\rho(x) =_\beta \rho'(x)$ for all variables x. Now [2]

$$\begin{array}{lllllll} t[\rho] & =_\beta & ((\lambda\vec{x}t)\,\vec{x})[\rho] & =_\beta & (\lambda\vec{x}t)[\rho]\;\vec{x}[\rho] & =_\beta & (\lambda\vec{x}t)\;\vec{x}[\rho] \\ & =_\beta & (\lambda\vec{x}t)[\rho']\;\vec{x}[\rho] & =_\beta & (\lambda\vec{x}t)[\rho']\;\vec{x}[\rho'] & =_\beta & ((\lambda\vec{x}t)\,\vec{x})[\rho'] \quad =_\beta \quad t[\rho']. \end{array}$$

If we weaken the assumption such that ρ and ρ' have to equivalent only on the free variables of t, the calculation is still sound and validates DEN-IRR. The laws DEN-CONST, DEN-VAR, DEN-FUN-E and DEN-PAIR-E follow directly by the definition of parallel substitution, with a little work also DEN-β. The injectivity requirements hold since El, Fun, and Pair are unanimated constants.

□

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

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

η-reduction on β-normal forms. In order to obtain an η-equality on values, we define one-step η-reduction $v \longrightarrow_\eta v'$ for $v, v' \in \mathsf{Val}$ inductively by the following rules.

$$\text{ETA-FUN-RED}\ \frac{}{\lambda x.\,u\,x \longrightarrow_\eta u} \qquad \text{ETA-PAIR-RED}\ \frac{}{(u\,\mathsf{L},\ u\,\mathsf{R}) \longrightarrow_\eta u}$$

$$\text{ETA-FUN-I}\ \frac{v \longrightarrow_\eta v'}{\lambda x v \longrightarrow_\eta \lambda x v'}$$

$$\text{ETA-FUN-E-L}\ \frac{u \longrightarrow_\eta u'}{u\,v \longrightarrow_\eta u'\,v} \qquad \text{ETA-FUN-E-R}\ \frac{v \longrightarrow_\eta v'}{u\,v \longrightarrow_\eta u\,v'}$$

$$\text{ETA-PAIR-I-L}\ \frac{v_1 \longrightarrow_\eta v_1'}{(v_1,\ v_2) \longrightarrow_\eta (v_1',\ v_2)} \qquad \text{ETA-PAIR-E}\ \frac{u \longrightarrow_\eta u'}{u\,p \longrightarrow_\eta u'\,p}$$

Note that η-reduction on β-normal forms does not create β-redexes, hence it is well-defined. Neutral values reduce to neutral values, so it is even well-defined on VNe. It does *not* preserve typing, e. g.,

[2] Benzmüller, Brown, and Kohlhase [5] prove a similar result by converting t into an SK-combinatorical term. Our argument seems simpler.

$z : \mathsf{Pair}\,A\,B \ \vdash\ (z\,\mathsf{L},\ z\,\mathsf{R})\ :\ \mathsf{Pair}\,A\,(\lambda\_.\,B\,(z\,\mathsf{L}))$, but not $z : \mathsf{Pair}\,A\,B \ \vdash\ z\ :\ \mathsf{Pair}\,A\,(\lambda\_.\,B\,(z\,\mathsf{L}))$. In contrast to η-reduction on arbitrary terms, it is locally confluent. Let $\longrightarrow_\eta^*$ denote the reflexive-transitive closure of $\longrightarrow_\eta$. As usual, η-equality $v =_\eta v'$ holds iff $v \longrightarrow_\eta^* v_0 \ {}_\eta^*\!\!\longleftarrow v'$ for some v_0. Note that all ETA-rules above are admissible for both $\longrightarrow_\eta^*$ and $=_\eta$.

Lemma 6.2. (Local confluence)
If $\mathcal{D}_1 :: v_0 \longrightarrow_\eta v_1$ and $\mathcal{D}_2 :: v_0 \longrightarrow_\eta v_2$ then $v_1 \longrightarrow_\eta^* v_3$ and $v_2 \longrightarrow_\eta^* v_3$ for some v_3.

Proof:
By simultaneous induction on $\mathcal{D}_1$ and $\mathcal{D}_2$. Some cases:

- Case $v_1 = v_2$. Then $v_3 = v_1 \longrightarrow_\eta^* v_3$.
- Case $\mathcal{D}_1 :: \lambda x.\,u\,x \longrightarrow_\eta u$ and $\mathcal{D}_2 :: \lambda x.\,u\,x \longrightarrow_\eta \lambda x.\,u'\,x$ where $u \longrightarrow_\eta u'$. Then $\lambda x.\,u'\,x \longrightarrow_\eta u'$.
- Case $\mathcal{D}_1 :: (u\,\mathsf{L},\ u\,\mathsf{R}) \longrightarrow_\eta u$ and $\mathcal{D}_2 :: (u\,\mathsf{L},\ u\,\mathsf{R}) \longrightarrow_\eta (u'\,\mathsf{L},\ u\,\mathsf{R})$ where $u \longrightarrow_\eta u'$. Then $u \longrightarrow_\eta^* u'$ and $(u'\,\mathsf{L},\ u\,\mathsf{R}) \longrightarrow_\eta (u'\,\mathsf{L},\ u'\,\mathsf{R}) \longrightarrow_\eta u'$.
- Case $\mathcal{D}_1 :: u\,v \longrightarrow_\eta u'\,v$ with $u \longrightarrow_\eta u'$ and $\mathcal{D}_2 :: u\,v \longrightarrow_\eta u\,v'$ with $v \longrightarrow_\eta v'$. Then $u'\,v \longrightarrow_\eta u'\,v'$ and $u\,v' \longrightarrow_\eta u'\,v'$.
- Case $\mathcal{D}_1 :: \lambda x v_0 \longrightarrow_\eta \lambda x v_1$ with $v_0 \longrightarrow_\eta v_1$ and $\mathcal{D}_2 :: \lambda x v_0 \longrightarrow_\eta \lambda x v_2$ with $v_0 \longrightarrow_\eta v_2$. By induction hypothesis $v_1 \longrightarrow_\eta v_3$ and $v_2 \longrightarrow_\eta v_3$, hence, $\lambda x v_1 \longrightarrow_\eta \lambda x v_3$ and $\lambda x v_2 \longrightarrow_\eta \lambda x v_3$.

□

Corollary 6.1. (Confluence)
If $v_0 \longrightarrow_\eta^* v_1$ and $v_0 \longrightarrow_\eta^* v_2$ then $v_1 \longrightarrow_\eta^* v_3$ and $v_2 \longrightarrow_\eta^* v_3$ for some v_3.

Proof:
By Newman's lemma, it is sufficient to show that $\longrightarrow_\eta^*$ is strongly normalizing. This is easy to see: Each reduction step decreases the number of introductions (λs and pairs), and no step creates an introduction.

□

Lemma 6.3. (Inversion properties of $\longrightarrow_\eta^*$)

1. If $\mathcal{D} :: x \longrightarrow_\eta^* v_0$ then $v_0 = x$. If $\mathcal{D} :: c \longrightarrow_\eta^* v_0$ then $v_0 = c$.
2. If $\mathcal{D} :: u\,v \longrightarrow_\eta^* v_0$ then $v_0 = u'\,v'$ with $u \longrightarrow_\eta^* u'$ and $v \longrightarrow_\eta^* v'$.
3. If $\mathcal{D} :: u\,p \longrightarrow_\eta^* v_0$ then $v_0 = u'\,p$ with $u \longrightarrow_\eta^* u'$.
4. If $\mathcal{D} :: \lambda x v \longrightarrow_\eta^* v_0$ then either
   - $v_0 = u$ neutral and $v \longrightarrow_\eta^* u\,x$, or
   - $v_0 = \lambda x v'$ and $v \longrightarrow_\eta^* v'$.
5. If $\mathcal{D} :: (v_1, v_2) \longrightarrow_\eta^* v_0$ then either
   - $v_0 = u$ neutral and both $v_1 \longrightarrow_\eta^* u\,\mathsf{L}$ and $v_2 \longrightarrow_\eta^* u\,\mathsf{R}$, or

- $v_0 = (v_1', v_2')$ and both $v_1 \longrightarrow_\eta^* v_1'$ and $v_2 \longrightarrow_\eta^* v_2'$.

Proof:
Each by induction on $\mathcal{D}$. □

Corollary 6.2. (Inversion on $=_\eta$)

1. If $x =_\eta u_0$ then $u_0 = x$. If $c =_\eta u_0$ then $u_0 = c$.
2. If $u\,v =_\eta u_0$ then $u_0 = u'\,v'$ with $u =_\eta u'$ and $v =_\eta v'$.
3. If $u\,p =_\eta u_0$ then $u_0 = u'\,p$ with $u =_\eta u'$.
4. If $\lambda xv =_\eta u$ then $v =_\eta u\,x$.
5. If $\lambda xv =_\eta \lambda xv'$ then $v =_\eta v'$.
6. If $(v_1, v_2) =_\eta u$ then $v_1 =_\eta u\,\mathsf{L}$ and $v_2 =_\eta u\,\mathsf{R}$.
7. If $(v_1, v_2) =_\eta (v_1', v_2')$ then $v_1 =_\eta v_1'$ and $v_2 =_\eta v_2'$.
8. If $(v_1, v_2) =_\eta \lambda xv$ then $v_1 \longrightarrow_\eta^* u\,\mathsf{L}$, $v_2 \longrightarrow_\eta^* u\,\mathsf{R}$, and $u\,x \;{}_\eta^*\!\!\longleftarrow v$ for some u.

An η-equality on β-equivalence classes. Since $\longrightarrow_\eta^*$ is confluent, η-equality $v_1 =_\eta v_2$, which holds iff $v_1 \longrightarrow_\eta^* v \;{}_\eta^*\!\!\longleftarrow v_2$ for some v, is transitive and, hence, an equivalence relation on Val. Thus, the relation

$$t \simeq t' \;:\iff\; t =_\beta v \text{ and } t' =_\beta v' \text{ for some } v, v' \text{ with } v =_\eta v'$$

is a partial equivalence on Exp. Note that if $t \simeq t'$, then t and t' are β-normalizable. If t, t' are β-normal forms, then $t \simeq t'$ if $t =_\eta t'$. We lift $\simeq$ to β-equivalence classes: $\overline{t} \simeq \overline{t'}$ iff $t \simeq t'$. Two classes are only related if both contain a β-normal form. Choosing these normal forms as representatives, we have

$$\overline{v} \simeq \overline{v'} \iff v =_\eta v'.$$

Safety range. We define the following sub-relations $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair} \subseteq \mathcal{S} := \simeq$.

$$\begin{array}{lll} (\overline{u},\, \overline{u'}) \in \mathcal{N} & :\iff & u =_\eta u' \\ (\overline{v_f},\, \overline{v_f'}) \in \mathcal{S}_{fun} & :\iff & v_f =_\eta v_f' \\ (\overline{v_p},\, \overline{v_p'}) \in \mathcal{S}_{pair} & :\iff & v_p =_\eta v_p' \end{array}$$

Lemma 6.4. $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair} \in \mathsf{Per}$.

Lemma 6.5. (Extensionality for functions)
If $v\,x \simeq v'\,x$ with $x \notin \mathsf{FV}(v, v')$, then $v, v' \in \mathsf{VFun}$ and $v =_\eta v'$.

Proof:
Consider the cases:

- Case v, v' neutral. Then $v\,x =_\eta v'\,x$, and $v =_\eta v'$ follows by Cor. 6.2, item 2.

- Case $v = \lambda x v_0$ and $v' = u$ neutral. W. l. o. g., $x \notin \mathsf{FV}(u)$. By assumption $v\,x \simeq u\,x$, and since $(\lambda x v_0)\,x \longrightarrow_\beta v_0$, we have $v_0 =_\eta u\,x$. Hence, $\lambda x v_0 =_\eta \lambda x.\, u\,x =_\eta u$.
- Case $v = \lambda x v_0$ and $v' = \lambda x v'_0$. From the assumption we get $v_0 =_\eta v'_0$ by β-reduction. Hence, $\lambda x v_0 =_\eta \lambda x v'_0$.
- Case $v = (v_1,\, v_2)$. Then $(v_1,\, v_2)\,x$ does not reduce to β-normal form, which is a contradiction to the assumption.

□

Corollary 6.3. (SAFE-EXT-FUN)
If $\overline{v\,u} = \overline{v'\,u'} \in \mathcal{S}$ for all $\overline{u} = \overline{u'} \in \mathcal{N}$, then $\overline{v} = \overline{v'} \in \mathcal{S}_{fun}$.

Proof:
By the previous lemma with $u = u' = x \notin \mathsf{FV}(v, v')$. □

Lemma 6.6. (SAFE-EXT-PAIR)
If $v\,\mathsf{L} \simeq v'\,\mathsf{L}$ and $v\,\mathsf{R} \simeq v'\,\mathsf{R}$ then $v, v' \in \mathsf{VPair}$ and $v =_\eta v'$.

Proof:
By cases, similar to last lemma. □

Corollary 6.4. (Safety range)
$\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair}$ form a safety range.

Proof:
SAFE-INT holds by definition of $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair}$. Requirements SAFE-NE-FUN and SAFE-NE-PAIR are simple closure properties of η-equality. SAFE-EXT-FUN is satisfied by Cor. 6.3 and SAFE-EXT-PAIR by Lemma 6.6. □

Now we can instantiate our generic PER model of MLF_Σ. We let $\mathcal{S}et := \mathcal{S}$ and $\mathcal{E}\ell(\overline{t}) := \mathcal{S}$. From this we get a decision procedure for judgemental equality.

Lemma 6.7. (Context satisfiable)
Let $\rho_0(x) := \overline{x}$ for all $x \in \mathsf{Var}$. If $\Gamma \vdash \mathsf{ok}$, then $\rho_0 \in [\Gamma]$.

Corollary 6.5. (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. 5.1), $t\rho_0 = t'\rho_0 \in [C\rho_0]$. The claim follows since $[C\rho_0] \subseteq \mathcal{S}$ by Lemma 5.4. □

We have shown that each well-typed term is β-normalizable and two judgementally equal terms $\beta\eta$-reduce to the same normal form. This gives us a decision procedure for equality of well-typed terms.

It remains to show that our algorithmic equality is also a decision procedure. In the next section, we demonstrate 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. Then we have proven completeness of the algorithmic equality.

7. Completeness

In this section, we show completeness of the algorithmic presentation of MLF_Σ by relating it to the term model of the last section.

7.1. A Transitive Extension of Algorithmic Equality

To relate the η-equality on β-normalforms $\simeq$ to the algorithmic equality $\sim$, we first present a transitive extension $\stackrel{+}{\sim}$ of the algorithmic equality which is conservative for terms of the same type. We then show that this extension $\stackrel{+}{\sim}$ is equivalent to $\simeq$. Since $\simeq$ has been shown complete through the PER model, the algorithmic equality is also complete for terms of the same type.

Algorithmic equality, restated. We recapitulate the rules of algorithmic equality, this time without use of active elimination @.

Rules for neutral terms:

$$\text{AQ-C}\ \frac{}{c \sim c} \qquad \text{AQ-VAR}\ \frac{}{x \sim x}$$

$$\text{AQ-NE-FUN}\ \frac{n \sim n' \qquad s\!\downarrow\, \sim s'\!\downarrow}{n\, s \sim n'\, s'} \qquad \text{AQ-NE-PAIR}\ \frac{n \sim n'}{n\, p \sim n'\, p}$$

The following three rules are a synonym for AQ-EXT-FUN.

$$\text{AQ-EXT-FUN-FUN}\ \frac{t\!\downarrow\, \sim t'\!\downarrow}{\lambda x t \sim \lambda x t'}$$

$$\text{AQ-EXT-FUN-NE}\ \frac{t\!\downarrow\, \sim n\, x}{\lambda x t \sim n}\ x \notin \mathsf{FV}(n) \qquad \text{AQ-EXT-NE-FUN}\ \frac{n\, x \sim t\!\downarrow}{n \sim \lambda x t}\ x \notin \mathsf{FV}(n)$$

And these three rules are a synonym for AQ-EXT-PAIR.

$$\text{AQ-EXT-PAIR-PAIR}\ \frac{r\!\downarrow\, \sim r'\!\downarrow \qquad s\!\downarrow\, \sim s'\!\downarrow}{(r, s) \sim (r', s')}$$

$$\text{AQ-EXT-PAIR-NE}\ \frac{r\!\downarrow\, \sim n\, \mathsf{L} \qquad s\!\downarrow\, \sim n\, \mathsf{R}}{(r, s) \sim n} \qquad \text{AQ-EXT-NE-PAIR}\ \frac{n\, \mathsf{L} \sim r\!\downarrow \qquad n\, \mathsf{R} \sim s\!\downarrow}{n \sim (r, s)}$$

A transitive extension. Let $w \stackrel{+}{\sim} w'$ be given by the rules for algorithmic equality plus the following two:

$$\text{AQ}^{+}\text{-FUN-PAIR}\ \frac{t\!\downarrow\, \stackrel{+}{\sim} n\, x \qquad n\, \mathsf{L} \stackrel{+}{\sim} r\!\downarrow \qquad n\, \mathsf{R} \stackrel{+}{\sim} s\!\downarrow}{\lambda x t \stackrel{+}{\sim} (r, s)}\ x \notin \mathsf{FV}(n)$$

$$\text{AQ}^{+}\text{-PAIR-FUN}\ \frac{r\!\downarrow\, \stackrel{+}{\sim} n\, \mathsf{L} \qquad s\!\downarrow\, \stackrel{+}{\sim} n\, \mathsf{R} \qquad n\, x \stackrel{+}{\sim} t\!\downarrow}{(r, s) \stackrel{+}{\sim} \lambda x t}\ x \notin \mathsf{FV}(n)$$

These rules destroy the algorithmic character, since the neutral term n has to be guessed if one reads the rules from bottom to top as in logic programming.

Lemma 7.1. (The extension $\stackrel{+}{\sim}$ is conservative for same-typed terms)

1. If $\mathcal{D} :: n \stackrel{+}{\sim} n'$ and $\Gamma \vdash n : C$ and $\Gamma \vdash n' : C'$ then $n \sim n'$.

2. If $\mathcal{D} :: t{\downarrow} \stackrel{+}{\sim} t'{\downarrow}$ and $\Gamma \vdash t, t' : C$ then $t{\downarrow} \sim t'{\downarrow}$.

Proof:
Simultaneously by induction on $\mathcal{D}$ using subject reduction for weak head evaluation which is implied by its soundness (Lemma 4.1). The requirement of being of the same type in (2.) prevents $\mathcal{D}$ from applying rules AQ$^+$-FUN-PAIR and AQ$^+$-PAIR-FUN. Hence $\mathcal{D}$ contains only the counterparts of the rules for the algorithmic equality. □

As a consequence, the algorithmic equality is transitive for terms of the same type, provided $\stackrel{+}{\sim}$ is indeed transitive. This claim will be validated through equivalence with the transitive $\simeq$.

7.2. Soundness of the Extended Algorithmic Equality

In this section, we show that the extended algorithmic equality $\stackrel{+}{\sim}$ is sound w. r. t. the model equality $\simeq$. Together with the dual result of the next section we establish equivalence of these two notions of equality. As a byproduct, we obtain transitivity of $\stackrel{+}{\sim}$, which we will later also obtain directly (see Section 8). However, for the completeness of the algorithmic equality, which is the main theme of this article, the soundness result of this section is not relevant.

Lemma 7.2. (Standardization)

1. If $t =_\beta x$ then $t \searrow x$. If $t =_\beta c$ then $t \searrow c$.

2. If $t =_\beta n\, s$ then $t \searrow n'\, s'$ with $n =_\beta n'$ and $s =_\beta s'$.

3. If $t =_\beta n\, p$ then $t \searrow n'\, p$ with $n =_\beta n'$.

4. If $t =_\beta \lambda x r$ then $t \searrow \lambda x r'$ with $r =_\beta r'$.

5. If $t =_\beta (r, s)$ then $t \searrow (r', s')$ with $r =_\beta r'$ and $s =_\beta s'$.

Proof:
Fact about the λ-calculus [3]. □

Lemma 7.3. (Soundness of $\stackrel{+}{\sim}$ w. r. t. $\simeq$)
If $\mathcal{D} :: t{\downarrow} \stackrel{+}{\sim} t'{\downarrow}$ then $t \simeq t'$.

Proof:
By induction on $\mathcal{D}$, using standardization. All cases are easy, for example:

- Case

$$\text{AQ}^{+}\text{-NE-FUN}\ \frac{n \overset{+}{\sim} n' \qquad s{\downarrow} \overset{+}{\sim} s'{\downarrow}}{n\,s \overset{+}{\sim} n'\,s'}$$

By induction hypothesis and standardization, $n =_\beta u =_\eta u' =_\beta n'$ and $s =_\beta v =_\eta v' =_\beta s'$. Thus, $n\,s =_\beta u\,v =_\eta u'\,v' =_\beta n'\,s'$.

- Case

$$\text{AQ}^{+}\text{-EXT-FUN-NE}\ \frac{t{\downarrow} \overset{+}{\sim} n\,x}{\lambda xt \overset{+}{\sim} n}\ x \notin \mathsf{FV}(n)$$

By induction hypothesis and standardization, $t =_\beta v =_\eta u\,x =_\beta n\,x$, hence, $\lambda xt =_\beta \lambda xv =_\eta \lambda x.\, u\,x =_\eta u =_\beta n$.

- Case

$$\text{AQ}^{+}\text{-FUN-PAIR}\ \frac{t{\downarrow} \overset{+}{\sim} n\,x \qquad n\,\mathsf{L} \overset{+}{\sim} r{\downarrow} \qquad n\,\mathsf{R} \overset{+}{\sim} s{\downarrow}}{\lambda xt \overset{+}{\sim} (r, s)}\ x \notin \mathsf{FV}(n)$$

By induction hypothesis and standardization, $t =_\beta v =_\eta u\,x =_\beta n\,x$, hence, $\lambda xv =_\eta \lambda x.\, u\,x =_\eta u$. Further, $n\,\mathsf{L} =_\beta u\,\mathsf{L} =_\eta v_1 =_\beta r$ and $n\,\mathsf{R} =_\beta u\,\mathsf{R} =_\eta v_2 =_\eta s$, thus, $u =_\eta (u\,\mathsf{L},\ u\,\mathsf{R}) =_\eta (v_1, v_2)$. Together, $\lambda xt =_\beta \lambda xv =_\eta (v_1, v_2) =_\beta (r, s)$.

□

Corollary 7.1. If $t{\downarrow} \overset{+}{\sim} t{\downarrow}$ then t is β-normalizable.

Remark 7.1. A consequence of the lemma is that $v \overset{+}{\sim} v'$ implies $v =_\eta v'$. This can also be proven directly without the use of standardization.

7.3. Completeness of the Extended Algorithmic Equality

Lemma 7.4. (Completeness of $\overset{+}{\sim}$ on β-normal forms)
If $v =_\eta v'$ then $v \overset{+}{\sim} v'$.

For the proof we need an induction measure $|\cdot|$ on terms which is compatible with the subterm ordering and gives extra weight to introductions, such that $|\lambda xr| + |t| > |r| + |t\,x|$ and $|(r, s)| + |t| > |r| + |t\,\mathsf{L}|$. These conditions are also met by Goguen's [10] measure for proving termination of Coquand's [6] algorithmic equality restricted to pure λ-terms. But we need the extra conditions $|\lambda xt| > 2|t|$ and both $|(r, s)| > 2|r|$ and $|(r, s)| > 2|s|$.

$$\begin{array}{rcl} |x| & := & |c| := 1 \\ |r\,s| & := & |r| + |s| \\ |r\,p| & := & |r| + 1 \\ |\lambda xt| & := & 2|t| + 1 \\ |(r, s)| & := & 2|r| + 2|s| \end{array}$$

Observe that the conditions are met since $|t| \geq 1$ for all terms t. This measure is compatible with η-reduction, i. e., if $v \longrightarrow_\eta v'$ then $|v| > |v'|$.

Proof:
[of Lemma 7.4] By induction on $|v| + |v'|$. We first treat the cases for neutral terms $u =_\eta u'$.

- Case $u = c$. Then $u' = c$ by Cor. 6.2 and $c \stackrel{+}{\sim} c$.
- Case $u = x$. Similar.
- Case $u = u_1\, v_1$. Then by Cor. 6.2 $u' = u_2\, v_2$ with $u_1 =_\eta u_2$ and $v_1 =_\eta v_2$. By induction hypothesis $u_1 \stackrel{+}{\sim} u_2$ and $v_1 \stackrel{+}{\sim} v_2$, hence $u \stackrel{+}{\sim} u'$ by AQ$^+$-NE-FUN.
- Case $u = u_1\, p$. Similar.

Now we look at the general form $v =_\eta v'$, where we omit symmetrical cases.

- Case $\lambda x v =_\eta u$. By Cor. 6.2, $v =_\eta u\, x$. Since $|v| + |u\, x| = |v| + |u| + 1 < (2|v| + 1) + |u| = |\lambda x v| + |u|$, we can apply the induction hypothesis and obtain $v \stackrel{+}{\sim} u\, x$. Thus $\lambda x v \stackrel{+}{\sim} u$ by AQ$^+$-EXT-FUN-NE.
- Case $\lambda x v =_\eta \lambda x v'$. By Cor. 6.2, $v =_\eta v'$, on which we apply the induction hypothesis and AQ$^+$-EXT-FUN-FUN.
- Case $\lambda x v =_\eta (v_1, v_2)$. By Cor. 6.2 there exists a neutral u such that $v \longrightarrow_\eta^* u\, x$ and both $u\, \mathsf{L} \;{}_\eta^*\!\!\longleftarrow v_1$ and $u\, \mathsf{R} \;{}_\eta^*\!\!\longleftarrow v_2$. Since reduction is compatible with the measure, we have $|v| + |u\, x| \leq 2|v| < 2|v| + 1 = |\lambda x v|$ and can apply the induction hypothesis to obtain $v \stackrel{+}{\sim} u\, x$. Further, we have $|u\, \mathsf{L}| + |v_1| \leq 2|v_1| < 2|v_1 + v_2| = |(v_1, v_2)|$, thus, by induction hypothesis, $u\, \mathsf{L} \stackrel{+}{\sim} v_1$, and similarly, $u\, \mathsf{R} \stackrel{+}{\sim} v_2$. By AQ$^+$-FUN-PAIR we get $\lambda x v \stackrel{+}{\sim} (v_1, v_2)$.
- Case $(v_1, v_2) =_\eta u$. By Cor. 6.2, $v_1 =_\eta u\, \mathsf{L}$ and $v_2 =_\eta u\, \mathsf{R}$. Since $|v_1| + |u\, \mathsf{L}| = |v_1| + |u| + 1 < 2(|v_1| + |v_2|) + |u| = |(v_1, v_2)| + |u|$, by induction hypothesis, $v_1 \stackrel{+}{\sim} u\, \mathsf{L}$, and with a similar calculation, $v_2 \stackrel{+}{\sim} u\, \mathsf{R}$. Thus, $(v_1, v_2) \stackrel{+}{\sim} u$ by AQ$^+$-NE-PAIR.
- Case $(v_1, v_2) =_\eta (v_1', v_2')$. By inversion, induction hypothesis, and rule AQ$^+$-EXT-PAIR-PAIR.

□

Remark 7.2. (Alternative proof)
First, show reflexivity $v \stackrel{+}{\sim} v$ for all β-normal forms v by induction on v. Then prove that $\stackrel{+}{\sim}$ is closed under η-expansion. More precisely, show that

1. $u \longrightarrow_\eta u'$ and $\mathcal{D} :: u'\, \vec{e} \stackrel{+}{\sim} v$ imply $u\, \vec{e} \stackrel{+}{\sim} v$ for a vector of eliminations $\vec{e}$, and
2. $v_1 \longrightarrow_\eta v_2$ and $\mathcal{D} :: v_2 \stackrel{+}{\sim} v_3$ imply $v_1 \stackrel{+}{\sim} v_3$

simultaneously by induction on $\mathcal{D}$. For reasons of symmetry, $\stackrel{+}{\sim}$ is also closed by η-expansion on the right hand side. Finally, assuming $v_1 \longrightarrow_\eta^* v_2 \;{}_\eta^*\!\!\longleftarrow v_3$ we can show $v_1 \stackrel{+}{\sim} v_3$ from $v_2 \stackrel{+}{\sim} v_2$ by induction on the number of reduction steps.

Lemma 7.5. (From normal to normalizing terms)
1. If $n =_\beta u$ and $n' =_\beta u'$ and $\mathcal{D} :: u \stackrel{+}{\sim} u'$, then $n \stackrel{+}{\sim} n'$.

2. If $t =_\beta v$ and $t' =_\beta v'$ and $\mathcal{D} :: v \overset{+}{\sim} v'$, then $t{\downarrow} \overset{+}{\sim} t'{\downarrow}$.

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

- Case $n =_\beta u\,v$ and $n' =_\beta u'\,v'$ and

$$\text{AQ}^+\text{-NE-FUN}\ \frac{u \overset{+}{\sim} u' \qquad v \overset{+}{\sim} v'}{u\,v \overset{+}{\sim} u'\,v'}$$

$n \searrow n_0\,s$ with $n_0 =_\beta u$ and $s =_\beta v$	standardization
$n' \searrow n'_0\,s'$ with $n'_0 =_\beta u'$ and $s' =_\beta v'$	standardization
$n_0 \overset{+}{\sim} n'_0$	first ind. hyp.
$s{\downarrow} \overset{+}{\sim} s'{\downarrow}$	second ind. hyp.
$n_0\,s \overset{+}{\sim} n'_0\,s'$	AQ$^+$-NE-FUN
$n \equiv n_0\,s$ and $n \equiv n'_0\,s'$	$n \searrow n$ for $n \in \mathsf{WNe}$
$n \overset{+}{\sim} n'$	

- Case $t =_\beta \lambda xv$ and $t' =_\beta u$ and

$$\text{AQ}^+\text{-EXT-FUN-NE}\ \frac{v \overset{+}{\sim} u\,x}{\lambda xv \overset{+}{\sim} u}\ x \notin \mathsf{FV}(u)$$

$t \searrow \lambda xr$ with $r =_\beta v$	standardization
$t' \searrow n$ with $n =_\beta u$	standardization
$x \notin \mathsf{FV}(n)$	renaming
$n\,x =_\beta u\,x$	$=_\beta$ is a congruence
$r \overset{+}{\sim} n\,x$	induction hypothesis
$\lambda xr \overset{+}{\sim} n$	AQ$^+$-EXT-FUN-NE

- Case $t =_\beta \lambda xv$ and $t' =_\beta (v_1, v_2)$ and

$$\text{AQ}^+\text{-FUN-PAIR}\ \frac{v \overset{+}{\sim} u\,x \qquad u\,\mathsf{L} \overset{+}{\sim} v_1 \qquad u\,\mathsf{R} \overset{+}{\sim} v_2}{\lambda xv \overset{+}{\sim} (v_1, v_2)}\ x \notin \mathsf{FV}(u)$$

$t \searrow \lambda xr$ with $r =_\beta v$	standardization
$t' \searrow (r_1, r_2)$ with $r_1 =_\beta v_1$ and $r_2 =_\beta v_2$	standardization
$r \overset{+}{\sim} u\,x$ and $u\,\mathsf{L} \overset{+}{\sim} r_1$ and $u\,\mathsf{R} \overset{+}{\sim} r_2$	induction hypotheses
$\lambda xr \overset{+}{\sim} (r_1, r_2)$	AQ$^+$-FUN-PAIR

□

Corollary 7.2. [Completeness of $\stackrel{+}{\sim}$] If $t \simeq t'$ then $t\!\downarrow \stackrel{+}{\sim} t'\!\downarrow$.

Proof:
By assumption $t =_\beta v =_\eta v' =_\beta t'$. First $v \stackrel{+}{\sim} v'$ by Lemma 7.4, then also $t\!\downarrow \stackrel{+}{\sim} t'\!\downarrow$ by Lemma 7.5. □

Corollary 7.3. If t is β-normalizable, then $t\!\downarrow \stackrel{+}{\sim} t\!\downarrow$.

Together with Cor. 7.1 we see that the diagonal of extended algorithmic equalty—which coincides with the diagonal of pure algorithmic equality—characterizes the weakly normalizing terms t. Therefore, we can define $w \in \mathsf{WN} \;:\Longleftrightarrow\; w \stackrel{+}{\sim} w$ and $t \in \mathsf{WN} \;:\Longleftrightarrow\; t \searrow w \in \mathsf{WN}$. Let us specialize the rules of algorithmic equality to WN:

$$\frac{}{c \in \mathsf{WN}} \qquad \frac{}{x \in \mathsf{WN}} \qquad \frac{n \in \mathsf{WN} \qquad s \in \mathsf{WN}}{n\,s \in \mathsf{WN}} \qquad \frac{n \in \mathsf{WN}}{n\,p \in \mathsf{WN}}$$

$$\frac{r \in \mathsf{WN}}{\lambda x r \in \mathsf{WN}} \qquad \frac{r \in \mathsf{WN} \qquad s \in \mathsf{WN}}{(r, s) \in \mathsf{WN}} \qquad \frac{t \searrow w \qquad w \in \mathsf{WN}}{t \in \mathsf{WN}}$$

This predicate corresponds Joachimski and Matthes' [14] inductive characterization of weakly normalizing λ-terms. (Only that they use weak head reduction instead of weak head evaluation.)

7.4. Completeness of Algorithmic Equality

Now we can assemble the pieces of the jigsaw puzzle.

Theorem 7.1. (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 $A \sim A'$.

Proof:
Completeness for terms (1): By Cor. 6.5 we have $\overline{t} \simeq \overline{t}'$, which entails $t\!\downarrow \stackrel{+}{\sim} t'\!\downarrow$ by Cor. 7.2. Since $\Gamma \vdash t, t' : C$, we infer $t\!\downarrow \sim t'\!\downarrow$ by Lemma 7.1. The completeness for types (2) is then shown by induction on $\mathcal{D}$, using completeness for terms in case EQ-SET-E. □

8. A Shortcut: Disposing of η-Reduction

In sections 7.2 and 7.3 we have shown that the extended algorithmic equality $\stackrel{+}{\sim}$ is equivalent to η-equality on β-normal forms. Hence, we could define more directly $\overline{v} \simeq \overline{v'}$ iff $v \stackrel{+}{\sim} v'$. The requirement SAFE-EXT-FUN is simply fulfilled by rule AQ$^+$-EXT-FUN, and SAFE-EXT-PAIR by AQ$^+$-EXT-PAIR. It remains to show—without reference to $=_\eta$—that $\stackrel{+}{\sim}$ is transitive. We dedicate the remainder of this section to that task.

Let $\#\mathcal{D} \geq 1$ denote the following measure on derivations $\mathcal{D} :: w \stackrel{+}{\sim} w'$:

$$\begin{array}{lcl} \#\text{AQ}^{+}\text{-FUN-PAIR}(\mathcal{D}_1, \mathcal{D}_{21}, \mathcal{D}_{22}) & = & 1 + \#\mathcal{D}_1 + \max(\#\mathcal{D}_{21}, \#\mathcal{D}_{22}) \\ \#\text{AQ}^{+}\text{-PAIR-FUN}(\mathcal{D}_{11}, \mathcal{D}_{12}, \mathcal{D}_2) & = & 1 + \max(\#\mathcal{D}_{11}, \#\mathcal{D}_{12}) + \#\mathcal{D}_2 \\ \#r(\mathcal{D}_1, \dots, \mathcal{D}_n) & = & 1 + \max\{\#\mathcal{D}_i \mid 1 \leq i \leq n\} \end{array}$$

Here, r stands for any other rule application, or more precisely, a rule which has a counterpart in the original algorithmic equality judgement $w \sim w'$. Hence, $\#\mathcal{D}$ is just the height of derivation $\mathcal{D}$ if $\mathcal{D}$ corresponds to a derivation of $w \sim w'$. Since rule AQ$^+$-FUN-PAIR stands for a pair of derivations $\mathcal{D}_1 :: \lambda xt \sim n$ and $\mathcal{D}_2 :: n \sim (r, s)$, its weight is derived for the sum of the weight of these derivations; and similarly for AQ$^+$-PAIR-FUN.

Lemma 8.1. ($\stackrel{+}{\sim}$ is transitive)
Let $\vec{e}$ be a possibly empty list of eliminations.

1. If $\mathcal{D}_1 :: n \stackrel{+}{\sim} w$ and $\mathcal{D}_2 :: w \stackrel{+}{\sim} n'$ then $\mathcal{E} :: n \stackrel{+}{\sim} n'$.
2. If $\mathcal{D}_1 :: w \stackrel{+}{\sim} n\,\vec{e}$ and $\mathcal{D}_2 :: n \stackrel{+}{\sim} n'$ then $\mathcal{E} :: w \stackrel{+}{\sim} n'\,\vec{e}$.
3. If $\mathcal{D}_1 :: n \stackrel{+}{\sim} n'$ and $\mathcal{D}_2 :: n'\,\vec{e} \stackrel{+}{\sim} w$ then $\mathcal{E} :: n \stackrel{+}{\sim} w$.
4. If $\mathcal{D}_1 :: w_1 \stackrel{+}{\sim} w_2$ and $\mathcal{D}_2 :: w_2 \stackrel{+}{\sim} w_3$ then $\mathcal{E} :: w_1 \stackrel{+}{\sim} w_3$.

In all cases, $\#\mathcal{E} < \#\mathcal{D}_1 + \#\mathcal{D}_2$.

Proof:
Simultaneously by induction on $\#\mathcal{D}_1 + \#\mathcal{D}_2$. In the remainder of this proof, leave $\#$ implicit. First, we prove (1):

- Case $\mathcal{D}_1, \mathcal{D}_2 :: x \stackrel{+}{\sim} x$. Then $\mathcal{E} :: x \stackrel{+}{\sim} x$ with $1 = \mathcal{E} < \mathcal{D}_1 + \mathcal{D}_2 = 2$.
- Case

$$\mathcal{D}_1 = \frac{\begin{array}{cc}\mathcal{D}_{11} & \mathcal{D}_{12} \\ n_1 \stackrel{+}{\sim} n_2 & s_1{\downarrow} \stackrel{+}{\sim} s_2{\downarrow}\end{array}}{n_1\, s_1 \stackrel{+}{\sim} n_2\, s_2} \qquad \mathcal{D}_2 = \frac{\begin{array}{cc}\mathcal{D}_{21} & \mathcal{D}_{22} \\ n_2 \stackrel{+}{\sim} n_3 & s_2{\downarrow} \stackrel{+}{\sim} s_3{\downarrow}\end{array}}{n_2\, s_2 \stackrel{+}{\sim} n_3\, s_3}$$

$\mathcal{E}_1 :: n_1 \stackrel{+}{\sim} n_3$	$\mathcal{E}_1 < \mathcal{D}_{11} + \mathcal{D}_{21}$	first ind. hyp.
$\mathcal{E}_2 :: s_1{\downarrow} \stackrel{+}{\sim} s_3{\downarrow}$	$\mathcal{E}_2 < \mathcal{D}_{12} + \mathcal{D}_{22}$	second ind. hyp.
$\mathcal{E} :: n_1\, s_1{\downarrow} \stackrel{+}{\sim} n_3\, s_3{\downarrow}$	$\mathcal{E} = \mathcal{E}_1 + \mathcal{E}_2 + 1 < \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-NE-FUN

- Case $n_1\, p \stackrel{+}{\sim} n_2\, p$ and $n_2\, p \stackrel{+}{\sim} n_3\, p$: Similarly.

- Case

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}'_1\\ n\,x \overset{+}{\sim} t{\downarrow}\end{array}}{n \overset{+}{\sim} \lambda x t}\; x \notin \mathsf{FV}(n) \qquad \mathcal{D}_2 = \dfrac{\begin{array}{c}\mathcal{D}'_2\\ t{\downarrow} \overset{+}{\sim} n'\,x\end{array}}{\lambda x t \overset{+}{\sim} n'}\; x \notin \mathsf{FV}(n')$$

$\mathcal{E}' :: n\,x \overset{+}{\sim} n'\,x$	$\mathcal{E}' < \mathcal{D}'_1 + \mathcal{D}'_2$	ind. hyp.
$\mathcal{E} :: n \overset{+}{\sim} n'$	$\mathcal{E} < \mathcal{E}' < \mathcal{D}_1 + \mathcal{D}_2$	inversion

- Case

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}_{11}\\ n\,\mathsf{L} \overset{+}{\sim} r{\downarrow}\end{array} \qquad \begin{array}{c}\mathcal{D}_{12}\\ n\,\mathsf{R} \overset{+}{\sim} s{\downarrow}\end{array}}{n \overset{+}{\sim} (r, s)} \qquad \mathcal{D}_2 = \dfrac{\begin{array}{c}\mathcal{D}_{21}\\ r{\downarrow} \overset{+}{\sim} n'\,\mathsf{L}\end{array} \qquad \begin{array}{c}\mathcal{D}_{22}\\ s{\downarrow} \overset{+}{\sim} n'\,\mathsf{R}\end{array}}{(r, s) \overset{+}{\sim} n'}$$

$\mathcal{E}_1 :: n\,\mathsf{L} \overset{+}{\sim} n'\,\mathsf{L}$	$\mathcal{E}_1 < \mathcal{D}_{11} + \mathcal{D}_{21}$	ind. hyp.
$\mathcal{E} :: n \overset{+}{\sim} n'$	$\mathcal{E} < \mathcal{E}_1 < \mathcal{D}_1 + \mathcal{D}_2$	inversion

For (2), consider the cases:

- Case w is neutral and $\vec{e}$ is empty: By (1).
- Case $w = n_0\,s_0$ and

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}_{11}\\ n_0 \overset{+}{\sim} n\,\vec{e}\end{array} \qquad \begin{array}{c}\mathcal{D}_{12}\\ s_0{\downarrow} \overset{+}{\sim} s{\downarrow}\end{array}}{n_0\,s_0 \overset{+}{\sim} n\,\vec{e}\,s} \qquad \begin{array}{c}\mathcal{D}_2\\ n \overset{+}{\sim} n'\end{array}$$

$\mathcal{E}' :: n_0 \overset{+}{\sim} n'\,\vec{e}$	$\mathcal{E}' < \mathcal{D}_{11} + \mathcal{D}_2$	ind. hyp. $(\mathcal{D}_{11} + \mathcal{D}_2 < \mathcal{D}_1 + \mathcal{D}_2)$
$\mathcal{E} :: n_0\,s_0 \overset{+}{\sim} n'\,\vec{e}\,s$	$\mathcal{E} = 1 + \max(\mathcal{E}', \mathcal{D}_{12}) < \mathcal{D}_1 + \mathcal{D}_2$	AQ^{+}-NE-FUN

- Case $w = n_0\,p$ similar.
- Case $w = \lambda x t$ and

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}'_1\\ t{\downarrow} \overset{+}{\sim} n\,\vec{e}\,x\end{array}}{\lambda x t \overset{+}{\sim} n\,\vec{e}} \qquad \begin{array}{c}\mathcal{D}_2\\ n \overset{+}{\sim} n'\end{array}$$

$\mathcal{E}' :: t{\downarrow} \overset{+}{\sim} n'\,\vec{e}\,x$	$\mathcal{E}' < \mathcal{D}_1' + \mathcal{D}_2$	ind. hyp.
$\mathcal{E} :: \lambda xt \overset{+}{\sim} n'\,\vec{e}$	$\mathcal{E} < \mathcal{D}_1' + \mathcal{D}_2 + 1 = \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-EXT-FUN-NE

- Case

$$\mathcal{D}_1 = \dfrac{\overset{\mathcal{D}_{11}}{r{\downarrow} \overset{+}{\sim} n\,\vec{e}\,\mathsf{L}} \qquad \overset{\mathcal{D}_{12}}{s{\downarrow} \overset{+}{\sim} n\,\vec{e}\,\mathsf{R}}}{(r,s) \overset{+}{\sim} n\,\vec{e}} \qquad \overset{D_2}{n \overset{+}{\sim} n'}$$

$\mathcal{E}_1 :: r{\downarrow} \overset{+}{\sim} n'\,\vec{e}\,\mathsf{L}$	$\mathcal{E}_1 < \mathcal{D}_{11} + \mathcal{D}_2$	ind. hyp.
$\mathcal{E}_2 :: s{\downarrow} \overset{+}{\sim} n'\,\vec{e}\,\mathsf{R}$	$\mathcal{E}_2 < \mathcal{D}_{12} + \mathcal{D}_2$	ind. hyp.
$\mathcal{E} :: (r,s) \overset{+}{\sim} n'\,\vec{e}$	$\mathcal{E} = 1 + \max(\mathcal{E}_1, \mathcal{E}_2) < \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-EXT-PAIR-NE

Statement (3) is symmetrical to (2) and can be proven analogously. For (4), all of the following cases are easy:

- Case $\lambda xt \overset{+}{\sim} n$ and $n \overset{+}{\sim} \lambda xt'$.
- Case $\lambda xt \overset{+}{\sim} \lambda xt'$ and $\lambda xt' \overset{+}{\sim} n$ (plus symmtrical case).
- Case $\lambda xt_1 \overset{+}{\sim} \lambda xt_2$ and $\lambda xt_2 \overset{+}{\sim} \lambda xt_3$.
- Case $(r,s) \overset{+}{\sim} n$ and $n \overset{+}{\sim} (r',s')$.
- Case $(r,s) \overset{+}{\sim} (r',s')$ and $(r',s') \overset{+}{\sim} n$ (plus symmtrical case).
- Case $(r_1,s_1) \overset{+}{\sim} (r_2,s_2)$ and $(r_2,s_2) \overset{+}{\sim} (r_3,s_3)$.

The following cases introduce a relation between a function and a pair.

- Case

$$\mathcal{D}_1 = \dfrac{\overset{\mathcal{D}_1'}{t{\downarrow} \overset{+}{\sim} n\,x}}{\lambda xt \overset{+}{\sim} n}\; x \notin \mathsf{FV}(n) \qquad \mathcal{D}_2 = \dfrac{\overset{\mathcal{D}_{21}}{n\,\mathsf{L} \overset{+}{\sim} r{\downarrow}} \qquad \overset{\mathcal{D}_{22}}{n\,\mathsf{R} \overset{+}{\sim} s{\downarrow}}}{n \overset{+}{\sim} (r,s)}$$

  $\mathcal{E} :: \lambda xt \overset{+}{\sim} (r,s)$ by AQ$^+$-FUN-PAIR. $\mathcal{E} = 1 + \mathcal{D}_1' + \max(\mathcal{D}_{21}, \mathcal{D}_{22}) < \mathcal{D}_1 + \mathcal{D}_2$.

- Case $(r,s) \overset{+}{\sim} n$ and $n \overset{+}{\sim} \lambda xt$. Symmetrical.

The remaining cases eliminate a relation between a function or a pair. We only spell out these cases where the second relation is of this kind, the other cases are analogously.

- Case ($x \notin \mathsf{FV}(n, n')$)

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}'_1 \\ n\,x \overset{+}{\sim} t{\downarrow}\end{array}}{n \overset{+}{\sim} \lambda x t} \qquad \mathcal{D}_2 = \dfrac{\begin{array}{c}\mathcal{D}'_2 \\ t{\downarrow} \overset{+}{\sim} n'\,x\end{array} \qquad \begin{array}{c}\mathcal{D}'_3 \\ n'\,\mathsf{L} \overset{+}{\sim} r{\downarrow}\end{array} \qquad \begin{array}{c}\mathcal{D}'_4 \\ n'\,\mathsf{R} \overset{+}{\sim} s{\downarrow}\end{array}}{\lambda x t \overset{+}{\sim} (r, s)}$$

$\mathcal{E}_1 :: n\,x \overset{+}{\sim} n'\,x$	$\mathcal{E}_1 < \mathcal{D}'_1 + \mathcal{D}'_2$	ind.hyp. on $\mathcal{D}'_1, \mathcal{D}'_2$
$\mathcal{E}_2 :: n \overset{+}{\sim} n'$	$1 + \mathcal{E}_2 < \mathcal{D}'_1 + \mathcal{D}'_2$	inversion on $\mathcal{E}_1$
$\mathcal{E}_3 :: n\,\mathsf{L} \overset{+}{\sim} n'\,\mathsf{L}$	$\mathcal{E}_3 < \mathcal{D}'_1 + \mathcal{D}'_2$	AQ$^+$-NE-PAIR
$\mathcal{E}_4 :: n\,\mathsf{R} \overset{+}{\sim} n'\,\mathsf{R}$	$\mathcal{E}_4 < \mathcal{D}'_1 + \mathcal{D}'_2$	AQ$^+$-NE-PAIR
$\mathcal{E}_5 :: n\,\mathsf{L} \overset{+}{\sim} r{\downarrow}$	$\mathcal{E}_5 < \mathcal{E}_3 + \mathcal{D}'_3 < \mathcal{D}_1 + \mathcal{D}_2$	ind.hyp. on $\mathcal{E}_3, \mathcal{D}'_3$
$\mathcal{E}_6 :: n\,\mathsf{R} \overset{+}{\sim} s{\downarrow}$	$\mathcal{E}_6 < \mathcal{E}_4 + \mathcal{D}'_4 < \mathcal{D}_1 + \mathcal{D}_2$	ind.hyp. on $\mathcal{E}_4, \mathcal{D}'_4$
$\mathcal{E} :: n \overset{+}{\sim} (r, s)$	$\mathcal{E} = 1 + \max(\mathcal{E}_5, \mathcal{E}_6) < \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-EXT-NE-PAIR

- Case ($x \notin \mathsf{FV}(n, n')$)

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}'_1 \\ t{\downarrow} \overset{+}{\sim} t'{\downarrow}\end{array}}{\lambda x t \overset{+}{\sim} \lambda x t'} \qquad \mathcal{D}_2 = \dfrac{\begin{array}{c}\mathcal{D}_{21} \\ t'{\downarrow} \overset{+}{\sim} n'\,x\end{array} \qquad \begin{array}{c}\mathcal{D}_{22} \\ n'\,\mathsf{L} \overset{+}{\sim} r{\downarrow}\end{array} \qquad \begin{array}{c}\mathcal{D}_{23} \\ n'\,\mathsf{R} \overset{+}{\sim} s{\downarrow}\end{array}}{\lambda x t' \overset{+}{\sim} (r, s)}$$

$\mathcal{E}' :: t{\downarrow} \overset{+}{\sim} n'\,x$	$\mathcal{E}' < \mathcal{D}'_1 + \mathcal{D}_{21}$	ind.hyp. on $\mathcal{D}'_1, \mathcal{D}_{21}$
$\mathcal{E} :: \lambda x t \overset{+}{\sim} (r, s)$	$\mathcal{E} = 1 + \mathcal{E}' + \max(\mathcal{D}_{22}, \mathcal{D}_{23}) < \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-FUN-PAIR

- Case

$$\mathcal{D}_1 = \dfrac{\begin{array}{c}\mathcal{D}_{11} \\ r{\downarrow} \overset{+}{\sim} n\,\mathsf{L}\end{array} \qquad \begin{array}{c}\mathcal{D}_{12} \\ s{\downarrow} \overset{+}{\sim} n\,\mathsf{R}\end{array} \qquad \begin{array}{c}\mathcal{D}_{13} \\ n\,x \overset{+}{\sim} t'{\downarrow}\end{array}}{(r, s) \overset{+}{\sim} \lambda x t'}\; x \notin \mathsf{FV}(n)$$

$$\mathcal{D}_2 = \dfrac{\begin{array}{c}\mathcal{D}_{21} \\ t{\downarrow} \overset{+}{\sim} n'\,x\end{array} \qquad \begin{array}{c}\mathcal{D}_{22} \\ n'\,\mathsf{L} \overset{+}{\sim} r{\downarrow}\end{array} \qquad \begin{array}{c}\mathcal{D}_{23} \\ n'\,\mathsf{R} \overset{+}{\sim} s{\downarrow}\end{array}}{\lambda x t \overset{+}{\sim} (r, s)}\; x \notin \mathsf{FV}(n')$$

$\mathcal{E}_1 :: n\,x \overset{+}{\sim} n'\,x$	$\mathcal{E}_1 < \mathcal{D}_{13} + \mathcal{D}_{21}$	ind. hyp.
$\mathcal{E}_2 :: n\,\mathsf{L} \overset{+}{\sim} n'\,\mathsf{L}$	$\mathcal{E}_2 < \mathcal{D}_{13} + \mathcal{D}_{21}$	inversion
$\mathcal{E}_3 :: n\,\mathsf{R} \overset{+}{\sim} n'\,\mathsf{R}$	$\mathcal{E}_3 < \mathcal{D}_{13} + \mathcal{D}_{21}$	inversion
$\mathcal{E}_4 :: r\!\downarrow \overset{+}{\sim} n'\,\mathsf{L}$	$\mathcal{E}_4 < \mathcal{D}_{11} + \mathcal{E}_2 < \mathcal{D}_{11} + \mathcal{D}_{13} + \mathcal{D}_{21}$	ind. hyp.
$\mathcal{E}_5 :: s\!\downarrow \overset{+}{\sim} n'\,\mathsf{R}$	$\mathcal{E}_5 < \mathcal{D}_{12} + \mathcal{E}_3 < \mathcal{D}_{12} + \mathcal{D}_{13} + \mathcal{D}_{21}$	ind. hyp.
$\mathcal{E}_6 :: r\!\downarrow \overset{+}{\sim} r'\!\downarrow$	$\mathcal{E}_6 < \mathcal{E}_4 + \mathcal{D}_{22} < \mathcal{D}_{11} + \mathcal{D}_{13} + \mathcal{D}_{21} + \mathcal{D}_{22}$	ind. hyp.
$\mathcal{E}_7 :: s\!\downarrow \overset{+}{\sim} s'\!\downarrow$	$\mathcal{E}_7 < \mathcal{E}_5 + \mathcal{D}_{22} < \mathcal{D}_{12} + \mathcal{D}_{13} + \mathcal{D}_{21} + \mathcal{D}_{22}$	ind. hyp.
$\mathcal{E} :: (r,s) \overset{+}{\sim} (r',s')$	$\mathcal{E} = 1 + \max(\mathcal{E}_6, \mathcal{E}_7) < \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-EXT-PAIR-PAIR

We have three cases left, which can be proven similarly to the previous ones.

- Case $n \overset{+}{\sim} (r,s)$ and $(r,s) \overset{+}{\sim} \lambda xt$.
- Case $(r,s) \overset{+}{\sim} (r',s')$ and $(r',s') \overset{+}{\sim} \lambda xt$.
- Case $\lambda xt \overset{+}{\sim} (r,s)$ and $(r,s) \overset{+}{\sim} \lambda xt'$.

□

9. Decidability

By completeness of algorithmic equality, every welltyped term is weakly normalizing (Cor 7.1). On weakly normalizing terms, the equality algorithm terminates, as we will see in this section.

9.1. Decidability of Equality

We have shown that two judgmentally equal terms t, t' weak-head normalize to w, w' and there exists a derivation of $w \sim w'$, 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 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*.[3]

Lemma 9.1. (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 $|\mathcal{D}_1| + |\mathcal{D}_2|$. There are many cases to consider. First we consider neutral w_1, w_2, for instance:

[3]A similar measure is used by Goguen [10] to prove termination of algorithmic equality restricted to pure λ-terms [6].

- Case: $w_1 \equiv x$ and $w_2 \equiv n_2\, s_2$. Since there is no rule with a conclusion of the shape $x \sim n_2\, s_2$, the query fails.

- Case: $w_1 \equiv n_1\, s_1$ and $w_2 \equiv n_2\, s_2$. Rule AQ-NE-FUN matches. By the first induction hypothesis, $n_1 \sim n_1$ and $n_2 \sim n_2$, hence, the subquery $n_1 \sim n_2$ terminates. Since by the second induction hypothesis, $s_1 \searrow w_1'$, $s_2 \searrow w_2'$, $w_1' \sim w_1'$, and $w_2' \sim w_2'$, the subquery $w_1' \sim w_2'$ terminates as well. Hence, the whole query terminates.

The other neutral cases work similarly. Let us consider some cases where at least one of the weak head normal forms is not neutral.

- Case $w_1 \equiv \lambda x r$ and $w_2 \equiv (t, t')$. There is no matching rule, the query fails.

- Case $w_1 \equiv n$ and $w_2 \equiv (t, t')$. Rule AQ-EXT-PAIR matches. We apply the induction hypothesis to the derivations $\hat{\mathcal{D}}_1 :: n\,\mathsf{L} \sim n\,\mathsf{L}$ and $\mathcal{D}_2' :: t{\downarrow} \sim t{\downarrow}$, which is legal since $|\mathcal{D}_1| + |\mathcal{D}_2| = |\mathcal{D}_1| + |\mathcal{D}_2'| + 2 > (|\mathcal{D}_1| + 1) + |\mathcal{D}_2'| = |\hat{\mathcal{D}}_1| + |\mathcal{D}_2'|$. Hence, the first subquery $n\,\mathsf{L} \sim t{\downarrow}$ terminates, and, by a similar argument, also the second subquery $n\,\mathsf{R} \sim t'{\downarrow}$.

- Case $w_1 \equiv n$ and $w_2 \equiv \lambda x r$. Rule AQ-EXT-FUN matches. Since $x \sim x$ is a derivation of height one, we can apply the induction hypothesis, with justification similar to the last case, on the only subquery $n\,x \sim r{\downarrow}$.

□

Theorem 9.1. (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 7.1, $t \searrow w$, $t' \searrow w'$, $w \sim w$, 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. □

9.2. Termination of Type Checking

The termination of the type checker is a consequence of termination of equality for welltyped objects.

Lemma 9.2. (Termination of type checking)
Let $\Gamma \vdash \mathsf{ok}$.

1. The query $\Gamma \vdash t \Downarrow ?\not\equiv \mathsf{Type}$ terminates.

2. If $\Gamma \vdash C : \mathsf{Type}$ then the query $\Gamma \vdash t \Uparrow C$ terminates.

Proof:
Simultaneously by induction on t. The inference succeeds directly in case $t \equiv x$ with rule INF-VAR, and fails immediately in case $t \equiv c$, $t \equiv \lambda x r$, or $t \equiv (t_1, t_2)$. We consider $t \equiv r\, s$. Then rule INF-FUN-E matches.

$$\text{INF-FUN-E}\ \frac{\Gamma \vdash r \Downarrow \mathsf{Fun}\, A\, (\lambda x B) \qquad \Gamma \vdash s \Uparrow A}{\Gamma \vdash r\, s \Downarrow B[s/x]}$$

query $\Gamma \vdash r \Downarrow ?$ terminates	induction hypothesis
$\Gamma \vdash r \Downarrow C$	&
$C \equiv \mathsf{Fun}\, A\,(\lambda x B)$	otherwise fail
$\Gamma \vdash r : \mathsf{Fun}\, A\,(\lambda x B)$	inference sound (Thm. 4.1)
$\Gamma \vdash \mathsf{Fun}\, A\,(\lambda x B) : \mathsf{Type}$	syntactic validity
$\Gamma \vdash A : \mathsf{Type}$	inversion
query $\Gamma \vdash s \Uparrow A$ terminates	induction hypothesis
$\Gamma \vdash s \Uparrow A$	otherwise fail
$\Gamma \vdash s : A$	checking sound (Thm. 4.1)
$\Gamma, x{:}A \vdash B : \mathsf{Type}$	inversion
$\Gamma \vdash B[s/x] : \mathsf{Type}$	substitution (Lemma 2.1)
$\Gamma \vdash r\, s \Downarrow B,$ query successful	

The remaining case $t \equiv r\,p$ is treated analogously. For the termination of checking, let us start with case $t \equiv (t_1, t_2)$, where rule CHK-PAIR-I matches.

$$\text{CHK-PAIR-I}\ \frac{\Gamma \vdash t_1 \Uparrow A \qquad \Gamma \vdash t_2 \Uparrow B[t_1/x]}{\Gamma \vdash (t_1, t_2) \Uparrow \mathsf{Pair}\, A\,(\lambda x B)}$$

Using the induction hypotheses, we basically need to show that $\Gamma \vdash B[t_1/x] : \mathsf{Type}$ if $\Gamma \vdash t_1 \Uparrow A$ succeeds. The case $t \equiv \lambda x r$ matches rule CHK-FUN-I and is treated similarly. In the remaining cases, rule CHK-INF fires.

$$\text{CHK-INF}\ \frac{\Gamma \vdash r \Downarrow A \qquad A \sim C}{\Gamma \vdash r \Uparrow C}$$

By induction hypothesis, the inference algorithm terminates. If $\Gamma \vdash r \Downarrow A$ then $\Gamma \vdash A : \mathsf{Type}$, hence the equality check terminates by Lemma 9.1, which implies termination of the type checker. □

Lemma 9.3. (Termination of type well-formedness)
If $\Gamma \vdash \mathsf{ok}$ then the query $\Gamma \vdash A \Downarrow \mathsf{Type}$ terminates.

Proof:
By induction on A, using the previous lemma in case $A \equiv \mathsf{El}\, t$. □

9.3. Completeness of Type Checking

Once we have solved the hard problem of deciding equality, the decidability of typing is easy, provided we restrict to *normal* terms.

Normal and neutral terms. We introduce two predicates $t \Uparrow$ (t is normal) and $t \Downarrow$ (t is neutral).

$$\frac{}{c \Downarrow} \qquad \frac{}{x \Downarrow} \qquad \frac{r \Downarrow \qquad s \Uparrow}{r\, s \Downarrow} \qquad \frac{r \Downarrow}{r\, p \Downarrow} \qquad \frac{r \Downarrow}{r \Uparrow} \qquad \frac{t \Uparrow}{\lambda x t \Uparrow} \qquad \frac{r \Uparrow \qquad s \Uparrow}{(r, s) \Uparrow}$$

Theorem 9.2. (Completeness of type checking)

1. If $\mathcal{D} :: t \Downarrow$ and $\Gamma \vdash t : C \not\equiv \mathsf{Type}$ then $\Gamma \vdash t \Downarrow A$ and $A \sim C$.

2. If $\mathcal{D} :: t \Uparrow$ and $\Gamma \vdash t : C \not\equiv \mathsf{Type}$ then $\Gamma \vdash t \Uparrow C$.

Proof:
Simultaneously by induction on $\mathcal{D}$. □

Corollary 9.1. (Completeness of type well-formedness)
If $\mathcal{D} :: \Gamma \vdash A : \mathsf{Type}$ and $A \Downarrow$ then $\Gamma \vdash A \Downarrow \mathsf{Type}$.

Proof:
By induction on $\mathcal{D}$. In case $A \equiv \mathsf{El}\, t$, the premise $A \Downarrow$ forces $t \Uparrow$, hence we can apply the previous theorem. □

10. 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 [13]—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 [8].

A more general proof of completeness? Our proof uses a λ-model with full β-equality thanks to the rule DEN-β. We had also considered a weaker model (without DEN-β and DEN-IRR, but with DEN-FUN-β and DEN-PAIR-β) 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 [8] 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 [19] have been working on an extension of the Edinburgh Logical Framework (ELF) by Σ-types (LF$_\Sigma$), following Harper and Pfenning [13]. In comparison to MLF_Σ, syntactic validity (Lemma 2.5) and injectivity are non-trivial in their formulation of ELF. Robin Adams [2] 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 [9] 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 [10] has proven termination and completeness for both the type-directed [13] and the shape-directed equality [6] from the standard meta-theoretical properties (strong normalization, confluence, subject reduction, etc.) of the logical framework. He also proposes a method to check $\beta\eta$-equality for Σ- and singleton types by a sequence of full η-expansion followed by β-reduction [11].

Acknowledgments. We are grateful to Lionel Vaux whose clear presentation of models for this implicit calculus [21] provided a guideline for our model construction. Thanks to Ulf Norell for proof-reading an earlier version of this article. The first author is indebted to Frank Pfenning who taught him type-directed equality and bidirectional type-checking at Carnegie Mellon University in 2000, and to Carsten Schürmann for communication on LF_Σ.

APPENDIX.

A. Surjective Pairing Destroys Confluence

Klop [15, pp. 195–208] shows that the untyped λ-calculus with the surjective pairing reduction $(r\,\mathsf{L},\ r\,\mathsf{R}) \longrightarrow r$ is not confluent (Church-Rosser). It is, however, locally confluent (weakly Church-Rosser), hence, because of Newman's Lemma, only a term with an infinite reduction sequence can fail to be confluent. Klop provides the following example.

$$\begin{array}{lcll} \mathsf{Y} & := & (\lambda x \lambda y.\, y\,(x\,x\,y)) & \text{Turing's fixed-point combinator} \\ \mathsf{e} & := & z & \text{free variable (or the term } \Omega) \\ \mathsf{c} & := & \mathsf{Y}\ (\lambda c \lambda a.\, \mathsf{e}\,(a\,\mathsf{L},\ (c\,a)\,\mathsf{R})) & \\ \mathsf{a} & := & \mathsf{Y}\,\mathsf{c} & \end{array}$$

Since $\mathsf{c}\,t \longrightarrow^+ \mathsf{e}\,(t\,\mathsf{L},\ (\mathsf{c}\,t)\,\mathsf{R})$ and $\mathsf{a} \longrightarrow^+ \mathsf{c}\,\mathsf{a}$, we can construct the following reduction sequences:

$$\begin{array}{l} \mathsf{c\,a} \longrightarrow^+ \mathsf{e}\,(\mathsf{a}\,\mathsf{L},\ (\mathsf{c\,a})\,\mathsf{R}) \longrightarrow^+ \mathsf{e}\,((\mathsf{c\,a})\,\mathsf{L},\ (\mathsf{c\,a})\,\mathsf{R}) \longrightarrow^+ \mathsf{e}\,(\mathsf{c\,a}) \\ \mathsf{c\,a} \longrightarrow^+ \mathsf{c}\,(\mathsf{c\,a}) \longrightarrow^+ \mathsf{c}\,(\mathsf{e}\,(\mathsf{c\,a})) \end{array}$$

The end reducts of both sequences cannot be joined again.

B. On Transitivity of Algorithmic Equality

While transitivity does not hold for the pure algorithmic equality (see Remark 3.1), 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*.[4] We will use this measure for the proof of transitivity and termination of algorithmic equality.

[4] A similar measure is used by Goguen [10] to prove termination of algorithmic equality restricted to pure λ-terms [6].

Lemma B.1. (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.

- Case:

$$\textsc{aq-ne-fun}\ \frac{n_1 \sim n_2 \qquad s_1{\downarrow} \sim s_2{\downarrow}}{n_1\, s_1 \sim n_2\, s_2} \qquad \textsc{aq-ne-fun}\ \frac{n_2 \sim n_3 \qquad s_2{\downarrow} \sim s_3{\downarrow}}{n_2\, s_2 \sim n_3\, s_3}$$

$\Gamma \vdash n_i : \mathsf{Fun}\, A_i\, (\lambda x B_i)$	&
$\Gamma \vdash s_i : A_i$	inversion for $i = 1, 2, 3$
$n_1 \sim n_3$	first ind. hyp.
$\Gamma \vdash n_1 = n_2 = n_3 : \mathsf{Fun}\, A_1\, (\lambda x B_1)$	&
$\Gamma \vdash \mathsf{Fun}\, A_1\, (\lambda x B_1) = \mathsf{Fun}\, A_2\, (\lambda x B_2) : \mathsf{Type}$	&
$\Gamma \vdash \mathsf{Fun}\, A_2\, (\lambda x B_2) = \mathsf{Fun}\, A_3\, (\lambda x B_3) : \mathsf{Type}$	soundness of $\sim$
$\Gamma \vdash A_1 = A_2 = A_3 : \mathsf{Type}$	injectivity
$\Gamma \vdash s_i : A_1$	$i = 1, 2, 3$, EQ-CONV
$s_1{\downarrow} \sim s_3{\downarrow}$	second ind. hyp.
$n_1\, s_1{\downarrow} \sim n_3\, s_3{\downarrow}$	AQ-NE-FUN

- In the following case, x is chosen such that $x \notin \mathsf{FV}(n)$.

$$\textsc{aq-ext-fun}\ \frac{(\lambda x t_1)@x \sim (\lambda x t_2)@x}{\lambda x t_1 \sim \lambda x t_2} \qquad \textsc{aq-ext-fun}\ \frac{(\lambda x t_2)@x \sim n@x}{\lambda x t_2 \sim n}$$

$C \equiv \mathsf{Fun}\, A\, (\lambda x B)$	&
$\Gamma, x{:}A \vdash t_1, t_2 : B$	inversion
$(\lambda x t_i)@x \searrow w_i$	for $i = 1, 2$, assumption
$t_i \searrow w_i$	for $i = 1, 2$, def. of $@$
$\Gamma \vdash t_i = w_i : B$	soundness of evaluation
$\Gamma, x{:}A \vdash n\, x : B$	weakening, FUN-E
$w_1 \sim n\, x$	ind. hyp.
$(\lambda x t_1)@x \sim n@x$	since $n@x \searrow n\, x$
$\lambda x t_1 \sim n$	AQ-EXT-FUN

- Case:

$$\text{AQ-EXT-FUN}\ \frac{(\lambda x t_1)@x \sim n_2@x}{\lambda x t_1 \sim n_2} \qquad n_2 \sim n_3$$

$C \equiv \mathsf{Fun}\, A\, (\lambda x B)$	&
$\Gamma, x\!:\!A \vdash t_1 : B$	inversion
$\Gamma, x\!:\!A \vdash n_i@x : B$	for $i = 2, 3$, weakening, FUN-E
$n_2@x \sim n_3@x$	AQ-NE-FUN
$(\lambda x t_1)@x \sim n_3@x$	ind. hyp.
$\lambda x t_1 \sim n_3$	AQ-EXT-FUN

□

C. Alternative to Inductive-Recursive Definition

In section 5.2 we have defined intensional type equality $V = V' \in \mathcal{T}\!ype$ and type interpretation $[V]$ simultaneously by induction-recursion. In the following, we give conventional definitions of the two concepts.

Type interpretation. Type interpretation $[\_] \in \mathsf{D} \rightharpoonup \mathsf{Rel}$ is a partial function specified by the following equations.

$$\begin{array}{lrcl}
\text{INT-SET-F} & [\mathsf{Set}] & = & \mathcal{S}et \\
\text{INT-SET-E} & [\mathsf{El}\ v] & = & \mathcal{E}\ell(v) \\
\text{INT-FUN-F} & [\mathsf{Fun}\, V\ F] & = & \mathcal{F}un([V], v \mapsto [F\ v]) \\
\text{INT-PAIR-F} & [\mathsf{Pair}\, V\ F] & = & \mathcal{P}air([V], v \mapsto [F\ v])
\end{array}$$

Lemma C.1. Type interpretation $[\_] \in \mathsf{D} \rightharpoonup \mathsf{Rel}$ is a well-defined partial function.

Proof:
Well-definedness, i. e., that $V = V'$ implies $[V] = [V']$, follows by injectivity and pairwise distinctness of type constructors. The latter guarantees that we can define the type interpretation by pattern matching although D is not necessarily a free structure. For instance, in the absence of the inequality $\mathsf{Set} \neq \mathsf{Fun}\ V\ F$ (DEN-SET-NOT-DEP), the defining equations of type interpretation could imply the inconsistency $\mathcal{S}et = \mathcal{F}un([V], v \mapsto [F\ v])$. Injectivity proves that, e. g., $[\mathsf{Fun}\ V\ F] = [\mathsf{Fun}\ V'\ F']$ if $\mathsf{Fun}\ V\ F = \mathsf{Fun}\ V'\ F'$, since then $V = V'$ and $F = F'$ by law DEN-DEP-INJ. □

Intensional type equality $\mathcal{T}ype \in$ Rel is is given inductively by the following rules. Note that rule TYEQ-DEP has an infinitary premise.

$$\text{TYEQ-SET-F}\ \frac{}{\mathsf{Set} = \mathsf{Set} \in \mathcal{T}ype} \qquad \text{TYEQ-SET-E}\ \frac{v = v' \in \mathcal{S}et}{\mathsf{El}\ v = \mathsf{El}\ v' \in \mathcal{T}ype}$$

$$\text{TYEQ-DEP}\ \frac{V = V' \in \mathcal{T}ype \qquad F\ v = F'\ v' \in \mathcal{T}ype \text{ for all } (v, v') \in [V]}{c\ V F = c\ V'\ F' \in \mathcal{T}ype}\ c \in \{\mathsf{Fun}, \mathsf{Pair}\}$$

In the last rule, if $[V]$ is not defined, the quantification is to be read as empty.

The next lemma proves the following: For all semantical types $V \in \mathcal{T}ype$, the interpretation $[V]$ is a well-defined PER, and intensionally equal types have the same interpretation. Together, $[\_] \in \mathsf{Fam}(\mathcal{T}ype)$.

Lemma C.2. (Soundness of intensional type equality)
If $\mathcal{D} :: V = V' \in \mathcal{T}ype$ then $[V], [V'] \in$ Per and $[V] = [V']$.

Proof:
By induction on the ordinal height of $\mathcal{D}$. We consider the following case:

$$\frac{V = V' \in \mathcal{T}ype \qquad F\ v = F'\ v' \in \mathcal{T}ype \text{ for all } v = v' \in [V]}{\mathsf{Fun}\ V\ F = \mathsf{Fun}\ V'\ F' \in \mathcal{T}ype}$$

We have to show that $\mathcal{F}un([V], v \mapsto [F\ v])$ and $\mathcal{F}un([V'], v \mapsto [F'\ v])$ are PERs and equal. By induction hypothesis, $[V]$ and $[V']$ are PERs and equal. Assume $v = v' \in [V]$ arbitrary. We may use the induction hypothesis on the assumptions $F\ v = F'\ v, F\ v' = F'\ v' \in \mathcal{T}ype$ to deduce $[F\ v] = [F\ v'] \in$ Per, hence, the family $\mathcal{F}$, defined by $\mathcal{F}(v) := [F\ v]$, is in $\mathsf{Fam}([V])$, since v and v' were arbitrary. Analogously, the second family $\mathcal{F}'$, where $\mathcal{F}'(v) := [F'\ v]$, it holds that $\mathcal{F}' \in \mathsf{Fam}([V'])$. By Lemma 5.2, $\mathcal{F}un([V], \mathcal{F})$ and $\mathcal{F}un([V'], \mathcal{F}')$ are PERs. Also by induction hypothesis, we obtain $[F\ v] = [F'\ v]$ for arbitrary v, so the two families $\mathcal{F}$ and $\mathcal{F}'$ are equal. This entails our goal. □

Finally, we can prove that $\mathcal{T}ype$ is a itself a PER.

Lemma C.3. (Soundness of intensional type equality)

1. If $\mathcal{D} :: V_1 = V_2 \in \mathcal{T}ype$ and $V_2 = V_3 \in \mathcal{T}ype$ then $V_1 = V_3 \in \mathcal{T}ype$.
2. If $\mathcal{D} :: V = V' \in \mathcal{T}ype$ then $V' = V \in \mathcal{T}ype$.

Proof:
Each by induction on the ordinal height of $\mathcal{D}$. For transitivity (1.), we consider the case:

$$\frac{V_1 = V_2 \in \mathcal{T}ype \qquad F_1\ v_1 = F_2\ v_2 \in \mathcal{T}ype \text{ for all } v_1 = v_2 \in [V]}{\mathsf{Fun}\ V_1\ F_1 = \mathsf{Fun}\ V_2\ F_2 \in \mathcal{T}ype}$$

$$\frac{V_2 = V_3 \in \mathcal{T}ype \qquad F_2\ v_2 = F_3\ v_3 \in \mathcal{T}ype \text{ for all } v_2 = v_3 \in [V]}{\mathsf{Fun}\ V_2\ F_2 = \mathsf{Fun}\ V_3\ F_3 \in \mathcal{T}ype}$$

By soundness of intensional type equality (Lemma C.2), we have $[V_1] = [V_2] \in \mathsf{Per}$, and by the first induction hypothesis, $V_1 = V_3 \in \mathcal{T}ype$. Assume arbitrary $v = v' \in [V_1]$. Since $[V_1]$ is a PER, $v' = v' \in [V_1]$, hence, also $v' = v' \in [V_2]$. By assumption $F_1\ v = F_2\ v' \in \mathcal{T}ype$ and $F_2\ v' = F_3\ v' \in \mathcal{T}ype$, hence, we can apply the induction hypothesis to obtain $F_1\ v = F_3\ v' \in \mathcal{T}ype$. Since v and v' were arbitrary $\mathsf{Fun}\ V_1\ F_1 = \mathsf{Fun}\ V_3\ F_3 \in \mathcal{T}ype$ by rule TYEQ-DEP. □

References

[1] Abel, A.: An Implementation of the Logical Framework with Σ-Types, Haskell code, available at http://www.tcs.ifi.lmu.de/~abel/MLFSigma.lhs, November 2004.

[2] Adams, R.: Decidable Equality in a Logical Framework with Sigma Kinds, 2001, Unpublished technical report, available on http://www.cs.man.ac.uk/~radams/.

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

[4] Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A Filter Lambda Model and the Completeness of Type Assignment, *The Journal of Symbolic Logic*, **48**(4), 1983, 931–940.

[5] Benzmüller, C., Brown, C. E., Kohlhase, M.: Higher-Order Semantics and Extensionality, *The Journal of Symbolic Logic*, **69**(4), December 2004, 1027–1088.

[6] Coquand, T.: An Algorithm for Testing Conversion in Type Theory, in: *Logical Frameworks* (G. Huet, G. Plotkin, Eds.), Cambridge University Press, 1991, 255–279.

[7] Coquand, T.: An Algorithm for Type-Checking Dependent Types, *Mathematics of Program Construction. Selected Papers from the Third International Conference on the Mathematics of Program Construction (July 17–21, 1995, Kloster Irsee, Germany)*, 26, Elsevier Science, May 1996.

[8] Coquand, T., Pollack, R., Takeyama, M.: A Logical Framework with Dependently Typed Records, *Typed Lambda Calculus and Applications, TLCA'03*, 2701, Springer, 2003.

[9] Goguen, H.: Soundness of the Logical Framework for Its Typed Operational Semantics, *Typed Lambda Calculi and Applications, TLCA 1999* (J.-Y. Girard, Ed.), 1581, Springer, L'Aquila, Italy, 1999.

[10] Goguen, H.: Justifying Algorithms for $\beta\eta$ Conversion, *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings* (V. Sassone, Ed.), 3441, Springer, 2005, ISBN 3-540-25388-2.

[11] Goguen, H.: A Syntactic Approach to Eta Equality in Type Theory, *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005* (J. Palsberg, M. Abadi, Eds.), ACM, January 2005, ISBN 1-58113-830-X.

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

[13] Harper, R., Pfenning, F.: On Equivalence and Canonical Forms in the LF Type Theory, *ACM Transactions on Computational Logic*, **6**(1), 2005, 61–101, ISSN 1529-3785.

[14] Joachimski, F., Matthes, R.: Short Proofs of Normalization, *Archive of Mathematical Logic*, **42**(1), 2003, 59–87.

[15] Klop, J. W.: Combinatory Reducion Systems, *Mathematical Center Tracts*, **27**, 1980.

[16] Nordström, B., Petersson, K., Smith, J.: Martin-Löf's Type Theory, *Handbook of Logic in Computer Science*, 5, Oxford University Press, October 2000.

[17] Pierce, B. C., Turner, D. N.: Local Type Inference, *POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Diego, California, 1998.

[18] Plotkin, G. D.: The lambda-Calculus is omega-Incomplete, *The Journal of Symbolic Logic*, **39**(2), 1974, 313–317.

[19] Sarnat, J.: LF_{Σ}: The Metatheory of LF with Σ types, 2004, Unpublished technical report, kindly provided by Carsten Schürmann.

[20] Vanderwaart, J. C., Crary, K.: *A Simplified Account of the Metatheory of Linear LF*, Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2002.

[21] Vaux, L.: A type system with implicit types, June 2004, English version of his mémoire de maîtrise.

[22] Vouillon, J.: Subtyping Union Types, *Computer Science Logic, CSL'04* (J. Marcinkowski, A. Tarlecki, Eds.), 3210, Springer, 2004.