Fundamenta Informaticae TLCA'05 1–53
IOS Press

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

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

Central to dependent type theories is the rule of *conversion*: The type of an expression can be converted to an equal type, where in intensional type theories the notion of equality between types is decidable. In the past, research has focused on β-equality, and since β-reduction is confluent, two types are equal iff they have a common β-reduct. This both suggest an implementation for equality—β-normalize and compare for syntactical identity—and provides a specification sufficient to reason about the meta-theoretic properties of the type theory: From confluence one gets injectivity of the dependent function space constructor Π, an important mile stone to prove subject reduction and, hence, soundness of the type theory. Because β is well behaved, one can consider β-reduction on untyped terms, which simplifies the reasoning significantly.

[*]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

If one adds η-laws to the notion of equality, this strategy does not work any more. Surjective pairing, the strong η-law for Σ-types, destroys confluence of reduction for untyped terms [19]. In the presence of a unit type, η-reduction is not even locally confluent on well-typed terms [20]. Furthermore, for the type theory considered in this article, MLF_Σ, an extension of Martin-Löf's logical framework with $\beta\eta$-equality by dependent surjective pairs (strong Σ types), subject reduction fails.

On the specification side, the short comings of η are salvaged by *judgmental equality*. Equality of two expressions is stated with reference to their type and a valid typing context. Soundness of type theories with judgmental equality follows directly from a PER model which interprets types as extensional partial equivalence relations. More problematic could be injectivity of Π; in our case, however, it is trivial since we follow Martin-L"of's later approach and separate terms and types conceptually: Terms can appear in types only inside the El constructor.

Another issue is decidability of judgmental equality: the rules do not suggest an algorithm immediately. We take the incremental $\beta\eta$-convertibility test which has been given by the second author for dependently typed λ-terms [8], 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 [17]. 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 [16], 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 [8] 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 [6] (see Appendix A). Therefore, one cannot *specify* this type system

with conversion defined on raw terms.[1] However, since surjective pairing does not destroy local confluence, it is confluent on strongly normalizing terms. Thus, once the type theory is proven sound which implies that all terms are normalizing, one can use reduction to check for equality. One byproduct of the completeness proof of our equality algorithm is that we can decide equality also by the following strategy: first β-normalize, then check η-equality of the β-normal forms (which in our case can be done by η-reduction).

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. Figure 1 summarizes the technical developments of this article. 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 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).

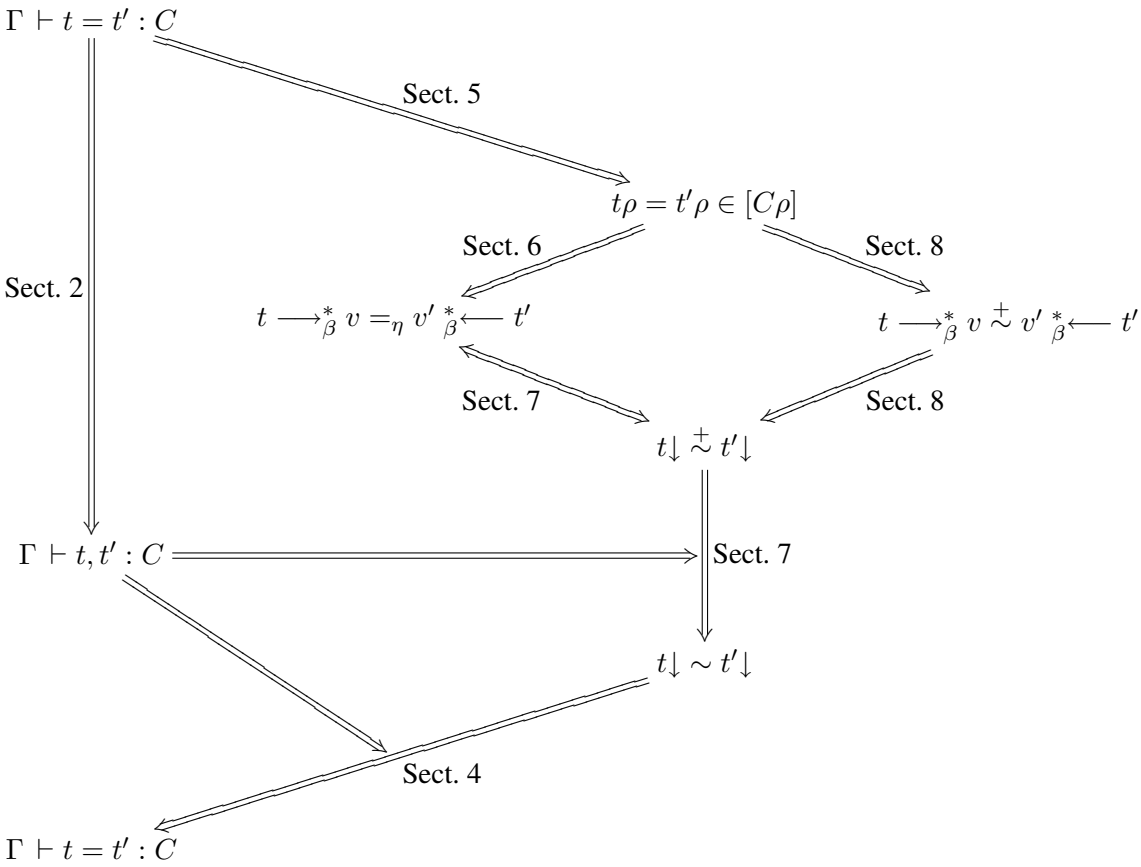
This article is an extended and revised version of a conference contribution with the same title [3].

2. Declarative Presentation of MLF_Σ

This section presents the typing and equality rules for an extension of Martin-Löf's logical framework [21] 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

[1] In the absence of confluence, one cannot show injectivity of type constructors, on which usually the proof of subject reduction rests. Adams [5] has shown the equivalence of pure type systems with judgmental and untyped equality, but only for β.



Legend:

Γ	typing context	$\_ \vdash \_ = \_ : \_$	equality judgement
C	type	$\_\downarrow$	weak head evaluation
t	term	$\_ \sim \_$	algorithmic equality
v	β-normal form	$\_ \overset{+}{\sim} \_$	extended algorithmic equality
$\_ \longrightarrow^*_\beta \_$	β-reduction	$\_\rho$	denotation in environment ρ
$\_ =_\eta \_$	η-equality	$[\_]$	type interpretation
$\_ \vdash \_ : \_$	typing judgement	$\_ = \_ \in \_$	partial equivalence relation

Figure 1. Outline of Results.

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

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

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

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

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

Typing $\Gamma \vdash t : A$.

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

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

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

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

Figure 2. MLF_Σ rules for contexts, types and typing.

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

The only binder is abstraction $\lambda x t$, which binds variable x in expression t. Let $\mathsf{FV}(t)$ denote the set of free variables of expression t. We identify terms and types up to α-conversion and write $t \equiv t'$ to express syntactic equality between t and t' (modulo α), as opposed to β-, η-, or judgemental equality which will be defined later. We adopt the convention that in contexts Γ, all variables must be distinct; hence, the context extension $\Gamma, x : A$ presupposes $x \notin \mathsf{dom}(\Gamma)$, where $\mathsf{dom}(\Gamma)$ denotes the domain of Γ viewed as a finite map from variables to types. This view also explains the notation $\Gamma(x)$. Capture-avoiding subsitution of expression s for variable x in expression t is written as $t[s/x]$.

Types $\mathsf{Ty} \subseteq \mathsf{Exp}$ are distinguished expressions; they are closed under substitution. The inhabitants of Set are type codes; El maps type codes to types. For instance, $\mathsf{Fun}\ \mathsf{Set}\ (\lambda a.\ \mathsf{Fun}\ (\mathsf{El}\, a)\ (\lambda \_.\, \mathsf{El}\, a))$ denotes the type of the polymorphic identity $\lambda a \lambda x x$.

Judgements are inductively defined relations. If $\mathcal{D}$ is a derivation of judgement J, we write $\mathcal{D} :: J$. The height of derivation $\mathcal{D}$ is denoted by $\#\mathcal{D}$. 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

A variable x is free in a judgement $\Gamma \vdash J$, written $x \in \mathsf{FV}(J)$, if it is free in one of the expressions in J. Typing rules are given in Figure 2, together with the rules for well-formed contexts and types. The rules for the equality judgements are given in Figures 3 and 4.

Remark 2.1. (Subject reduction fails)
In the typing 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 its 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 [17]. 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.

$$\text{EQ-SET-F}\ \frac{\Gamma \vdash \mathsf{ok}}{\Gamma \vdash \mathsf{Set} = \mathsf{Set} : \mathsf{Type}} \qquad \text{EQ-SET-E}\ \frac{\Gamma \vdash t = t' : \mathsf{Set}}{\Gamma \vdash \mathsf{El}\ t = \mathsf{El}\ t' : \mathsf{Type}}$$

$$\text{EQ-FUN-F}\ \frac{\Gamma \vdash A = A' : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B = B' : \mathsf{Type}}{\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) = \mathsf{Fun}\, A'\, (\lambda x B') : \mathsf{Type}}$$

$$\text{EQ-PAIR-F}\ \frac{\Gamma \vdash A = A' : \mathsf{Type} \qquad \Gamma, x{:}A \vdash B = B' : \mathsf{Type}}{\Gamma \vdash \mathsf{Pair}\, A\, (\lambda x B) = \mathsf{Pair}\, A'\, (\lambda x B') : \mathsf{Type}}$$

Figure 3. MLF_Σ type equality rules.

Type equality (see Figure 3) is reflexive, symmetric, and transitive, which can only be shown after establishing context conversion (Lemma 2.9) and syntactic validity (Theorem 2.1). But injectivity and non-confusion for type-constructors is completely trivial in our formulation of type equality, which lacks computation on the type level and rules for symmetry and transitivity.

Lemma 2.1. (Injectivity)

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

For Harper and Pfenning's version of the Edinburgh LF which lacks type-level λ-abstraction [17], injectivity is also not hard to prove. In the Edinburgh LF *with* type-level λ it involves a normalization argument and is proven using logical relations [26].

In the following, we prove a sequence of technical lemmata that bring us to the important theorem of syntactic validity which states that all syntactic entities in a derived judgement are well-formed.

Lemma 2.2. (Scoping)

1. If $\mathcal{D} :: \Gamma \vdash J$ then $\mathsf{FV}(J) \subseteq \mathsf{dom}(\Gamma)$.
2. If $\mathcal{D} :: \Gamma', x{:}A, \Gamma'' \vdash J$ then $\mathsf{FV}(A) \subseteq \mathsf{dom}(\Gamma')$.

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

Equivalence, hypotheses, conversion.

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

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

Dependent functions.

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

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

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

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

Dependent pairs.

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

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

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

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

Figure 4. MLF_Σ term equality rules.

Lemma 2.3. (Weakening)
If $\mathcal{D}_1 :: \Gamma, \Gamma' \vdash J$ and both $\mathcal{D}_2 :: \Gamma \vdash A : \mathsf{Type}$ and $x \notin \mathsf{dom}(\Gamma, \Gamma')$, then $\mathcal{E} :: \Gamma, x : A, \Gamma' \vdash J$.

Remark 2.2. The result derivation $\mathcal{E}$ is at most as long as the input derivations together: $\#\mathcal{E} \leq \#\mathcal{D}_1 + \#\mathcal{D}_2$. Note that the special case $\Gamma' = \diamond$ cannot be proven by itself; e.g., the case FUN-F fails since the order of the hypotheses matters in the context.

Proof:
By induction on $\mathcal{D}_1$. First, consider $J = \mathsf{ok}$. Γ' can be empty or not:

1. Case $\mathcal{D}_1 :: \Gamma, \Gamma' \vdash \mathsf{ok}$ and $\Gamma' = \diamond$. From $\mathcal{D}_2$ we get $\mathcal{E} :: \Gamma, x : A \vdash \mathsf{ok}$ by rule CXT-EXT.

2. Case $\mathcal{D}_1 :: \Gamma, \Gamma' \vdash \mathsf{ok}$ and $\Gamma' \neq \diamond$. The only matching rule is:

$$\text{CXT-EXT}\ \frac{\Gamma, \Gamma'' \vdash B : \mathsf{Type}}{\Gamma, \Gamma'', y : B \vdash \mathsf{ok}}$$

   By induction hypothesis, $\Gamma, x : A, \Gamma'' \vdash B : \mathsf{Type}$. We conclude by rule CXT-EXT.

Now we look at the other judgements. There are two principal cases for the last rule in $\mathcal{D}_1$: First, the rule discharges a hypothesis, and second, the context is left unchanged by the rule.

1. A hypothesis is discharged in the last rule of $\mathcal{D}_1$, for instance:

$$\text{FUN-F}\ \frac{\Gamma, \Gamma', y : B \vdash C : \mathsf{Type}}{\Gamma, \Gamma' \vdash \mathsf{Fun}\, B\, (\lambda y C) : \mathsf{Type}}$$

   Extending Γ' by the assumption $y : B$ for the induction hypothesis, we obtain $\Gamma, x : A, \Gamma', y : B \vdash C : \mathsf{Type}$. The goal follows by FUN-F.

2. The last rule application in $\mathcal{D}_1$ leaves the context unchanged. Most rules fall into this pattern, for instance:

$$\text{HYP}\ \frac{\Gamma, \Gamma' \vdash \mathsf{ok} \qquad (y : B) \in (\Gamma, \Gamma')}{\Gamma, \Gamma' \vdash y : B}$$

   By induction hypothesis, $\Gamma, x : A, \Gamma' \vdash \mathsf{ok}$, hence, $\Gamma, x : A, \Gamma' \vdash y : B$ by HYP.

Mixed forms, like EQ-FUN-F, can be treated analogously. □

The following lemma materializes the fact that in all judgements, the context is well-formed, which implies that all types in the contexts must be well-formed.

Lemma 2.4. (Context well-formedness)
1. If $\mathcal{D} :: \Gamma, \Gamma' \vdash J$ then $\mathcal{E} :: \Gamma \vdash \mathsf{ok}$ and the derivation $\mathcal{E}$ is at most as long as $\mathcal{D}$.

2. If $\mathcal{D} :: \Gamma, x : A, \Gamma' \vdash J$, then $\mathcal{E} :: \Gamma \vdash A : \mathsf{Type}$ and $\mathcal{E}$ is shorter than $\mathcal{D}$.

Proof:
The first proposition holds since in the leaves of type formation, typing, and equality derivations (rules SET-F, HYP, EQ-HYP, and EQ-SET-F) we require well-formed contexts. Formally, it is proven by induction on $\mathcal{D}$. All cases are easy, for instance:

$$\text{FUN-F}\ \frac{\Gamma, \Gamma', y : B \vdash C : \mathsf{Type}}{\Gamma, \Gamma' \vdash \mathsf{Fun}\, B\, (\lambda y C) : \mathsf{Type}}$$

By induction hypothesis, $\Gamma \vdash \mathsf{ok}$.

For the second proposition, use the first proposition to derive $\mathcal{E} :: \Gamma, x : A \vdash \mathsf{ok}$. The last rule in $\mathcal{E}$ must be CXT-EXT with premise $\mathcal{E}' :: \Gamma \vdash A : \mathsf{Type}$. We have $\#\mathcal{E}' < \#\mathcal{E} \leq \#\mathcal{D}$. □

Corollary 2.1. (Iterative weakening)
If $\Gamma \vdash J$ and $\Gamma, \Gamma' \vdash J'$ then $\Gamma, \Gamma' \vdash J$.

Proof:
By induction on the length of Γ'. If Γ' is empty, there is nothing to show, otherwise $\Gamma' = (x : A, \Gamma'')$. By the lemma, $\Gamma \vdash A : \mathsf{Type}$, hence by weakening, $\Gamma, x : A \vdash J$. By induction hypothesis, $\Gamma, x : A, \Gamma'' \vdash J$. □

Another consequence of context well-formedness is that we can decompose well-formed types:

Lemma 2.5. (Inversion for types)

1. If $\mathcal{D} :: \Gamma \vdash \mathsf{El}\ t : \mathsf{Type}$ then $\mathcal{D}' :: \Gamma \vdash t : \mathsf{Set}$.
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 Lemma 2.4 for part 2. For instance:

$$\text{FUN-F}\ \frac{\Gamma, x : A \vdash B : \mathsf{Type}}{\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}}$$

From the premise, $\mathcal{D}_2$, we get $\mathcal{D}_1 :: \Gamma \vdash A : \mathsf{Type}$ by context well-formedness. □

Similarly, context well-formedness is required to establish reflexivity.

Lemma 2.6. (Reflexivity)

1. If $\mathcal{D} :: \Gamma \vdash t : A$ then $\Gamma \vdash t = t : A$.
2. If $\mathcal{D} :: \Gamma \vdash A : \mathsf{Type}$ then $\Gamma \vdash A = A : \mathsf{Type}$.

Proof:
Each by induction on $\mathcal{D}$. Reflexivity for terms is proven mechanically, by replacing typing rules $\mathcal{R}$ in $\mathcal{D}$

with their counterpart EQ-$\mathcal{R}$ from the set of equality rules. Reflexivity for types requires some attention in case of FUN-F and PAIR-F, for instance:

$$\text{FUN-F}\ \frac{\Gamma, x{:}A \vdash B : \mathsf{Type}}{\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}}$$

First, the induction hypothesis gives us $\Gamma, x : A \vdash B = B : \mathsf{Type}$. Secondly, by the Lemma 2.4, we obtain a derivation $\Gamma \vdash A : \mathsf{Type}$ which is shorter than $\mathcal{D}$, so we can apply the induction hypothesis again to obtain $\Gamma \vdash A = A : \mathsf{Type}$. The goal follows by EQ-FUN-F. □

The substitution lemma allows us to replace a variable by a term of the correct type in a derivation. It relies on weakening and context well-formedness.

Lemma 2.7. (Substitution)
Let $\Gamma \vdash s : A$. If $\mathcal{D} :: \Gamma, x{:}A, \Gamma' \vdash J$ then $\Gamma, \Gamma'[s/x] \vdash J[s/x]$.

Proof:
By induction on $\mathcal{D}$.

- Case:
$$\text{CXT-EXT}\ \frac{\Gamma \vdash A : \mathsf{Type}}{\Gamma, x{:}A \vdash \mathsf{ok}}$$
Then $\Gamma \vdash \mathsf{ok}$ by context well-formedness.

- Case:
$$\text{CXT-EXT}\ \frac{\Gamma, x{:}A, \Gamma' \vdash B : \mathsf{Type}}{\Gamma, x{:}A, \Gamma', y{:}B \vdash \mathsf{ok}}$$
By induction hypothesis, $\Gamma, \Gamma'[s/x] \vdash B[s/x] : \mathsf{Type}$, hence $\Gamma, \Gamma'[s/x], y{:}B[s/x] \vdash \mathsf{ok}$.

- Cases HYP and EQ-HYP, e. g.:
$$\text{HYP}\ \frac{\Gamma, x{:}A, \Gamma' \vdash \mathsf{ok}}{\Gamma, x{:}A, \Gamma' \vdash x : A}$$
First observe that by scoping (Lemma 2.2) we have $\mathsf{FV}(A) \subseteq \mathsf{dom}(\Gamma)$, so $x \notin \mathsf{FV}(A)$ and $A[s/x] \equiv A$. By induction hypothesis, $\Gamma, \Gamma'[s/x] \vdash \mathsf{ok}$. We apply iterative weakening on $\Gamma \vdash s : A$ to conclude $\Gamma, \Gamma'[s/x] \vdash s : A$.

All other cases go through by induction hypothesis, using the properties of the substitution operation. □

In the following lemmata, which are all required to prove syntactic validity, there are some hypotheses (underlined) which will turn out to be redundant after we have proven syntactic validity. But until then, they are required for "boot-strapping".

Lemma 2.8. (Functionality for typing and type formation)
Let $\Gamma \vdash s = s' : A$ and $\underline{\Gamma \vdash s : A}$.

1. If $\mathcal{D} :: \Gamma, x{:}A, \Gamma' \vdash t : C$ then $\Gamma, \Gamma'[s/x] \vdash t[s/x] = t[s'/x] : C[s/x]$.
2. If $\mathcal{D} :: \Gamma, x{:}A, \Gamma' \vdash C : \mathsf{Type}$ then $\Gamma, \Gamma'[s/x] \vdash C[s/x] = C[s'/x] : \mathsf{Type}$.

Proof:
Each by induction on $\mathcal{D}$. We spell out some cases for the first proposition, for the second, there are no surprises.

- 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 (Cor. 2.1) 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 (Lemma 2.2).
  - 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:
$$\textsc{fun-i}\ \frac{\Gamma, x : A, \Gamma', y : B \vdash t : C}{\Gamma, x : A, \Gamma' \vdash \lambda y t : \mathsf{Fun}\, B\, \lambda y C}$$

$\Gamma, \Gamma'[s/x], y : B[s/x] \vdash t[s/x] = t[s'/x] : C[s/x]$	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:
$$\textsc{pair-e-r}\ \frac{\Gamma, x : A, \Gamma' \vdash r : \mathsf{Pair}\, B\, \lambda y C}{\Gamma, x : A, \Gamma' \vdash r\, \mathsf{R} : C[r\, \mathsf{L}/y]}$$

$\Gamma, \Gamma'[s/x] \vdash r[s/x] = r[s'/x] : \mathsf{Pair}\ (B[s/x])\ \lambda y.\, C[s/x]$	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 2.9. (Context conversion)
Let $\Gamma \vdash B = A : \mathsf{Type}$ and $\underline{\Gamma \vdash B : \mathsf{Type}}$. If $\mathcal{D} :: \Gamma, x:A, \Gamma' \vdash J$ then $\Gamma, x:B, \Gamma' \vdash J$.

Proof:
By induction on $\mathcal{D}$.

- Case $\Gamma' = \diamond$ and $J = \mathsf{ok}$:

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

  From the assumption $\Gamma \vdash B : \mathsf{Type}$ we get $\Gamma, x:B \vdash \mathsf{ok}$.

- Case:

$$\textsc{hyp}\ \frac{\Gamma, x:A, \Gamma' \vdash \mathsf{ok}}{\Gamma, x:A, \Gamma' \vdash x : A}$$

  By induction hypothesis $\Gamma, x:B, \Gamma' \vdash \mathsf{ok}$, hence, $\Gamma, x:B, \Gamma' \vdash x : B$. The goal follows by rule CONV.

The other cases can be handled mechanically using the induction hypothesis. □

Next, we will establish that type equality is an equivalence relation. Symmetry is needed to use a context conversion the other way round in the syntactic validity lemma.

Lemma 2.10. (Symmetry)
Let $\underline{\Gamma \vdash C' : \mathsf{Type}}$. If $\mathcal{D} :: \Gamma \vdash C = C' : \mathsf{Type}$ then $\Gamma \vdash C' = C : \mathsf{Type}$.

Proof:
By induction on $\mathcal{D}$.

- Case:

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

  By EQ-SYM, $\Gamma \vdash t' = t : \mathsf{Set}$. The goal follows by EQ-SET-E.

- Case:

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

  First, $\Gamma \vdash A : \mathsf{Type}$ by context well-formedness. By inversion on $\underline{\Gamma \vdash \mathsf{Fun}\, A'\, (\lambda x B') : \mathsf{Type}}$ we have $\Gamma \vdash A' : \mathsf{Type}$, hence by induction hypothesis, $\Gamma \vdash A' = A : \mathsf{Type}$. Again, by inversion, $\Gamma, x:A' \vdash B' : \mathsf{Type}$, and we apply context conversion to obtain $\Gamma, x:A \vdash B' : \mathsf{Type}$. We obtain the second induction hypothesis, $\Gamma, x:A \vdash B' = B : \mathsf{Type}$, from which we infer $\Gamma, x:A' \vdash B' = B : \mathsf{Type}$ by context conversion. We conclude $\Gamma \vdash \mathsf{Fun}\, A'\, (\lambda x B') = \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}$.

□

Lemma 2.11. (Transitivity)
If $\mathcal{D} :: \Gamma \vdash C_1 = C_2 : \mathsf{Type}$ and $\Gamma \vdash C_2 = C_3 : \mathsf{Type}$ then $\Gamma \vdash C_1 = C_3 : \mathsf{Type}$.

Proof:
By induction on $\mathcal{D}$. The case EQ-SET-E can be handled by EQ-TRANS, interesting is the following one:

$$\frac{\Gamma \vdash A_1 = A_2 : \mathsf{Type} \qquad \Gamma, x{:}A_1 \vdash B_1 = B_2 : \mathsf{Type}}{\Gamma \vdash \mathsf{Fun}\, A_1\, (\lambda x B_1) = \mathsf{Fun}\, A_2\, (\lambda x B_2) : \mathsf{Type}} \qquad \frac{\Gamma \vdash A_2 = A_3 : \mathsf{Type} \qquad \Gamma, x{:}A_2 \vdash B_2 = B_3 : \mathsf{Type}}{\Gamma \vdash \mathsf{Fun}\, A_2\, (\lambda x B_2) = \mathsf{Fun}\, A_3\, (\lambda x B_3) : \mathsf{Type}}$$

By context well-formedness, $\Gamma \vdash A_1 : \mathsf{Type}$, hence, we can apply context conversion to obtain $\Gamma, x : A_1 \vdash B_2 = B_3 : \mathsf{Type}$. By induction hypothesis it follows that $\Gamma, x{:}A_1 \vdash B_1 = B_3 : \mathsf{Type}$. The rest is easy. □

Theorem 2.1. (Syntactic validity)

1. Typing: If $\mathcal{D} :: \Gamma \vdash t : A$ then $\Gamma \vdash A : \mathsf{Type}$.
2. Equality: If $\mathcal{D} :: \Gamma \vdash t = t' : A$ then $\Gamma \vdash A : \mathsf{Type}$ and both $\Gamma \vdash t : A$ and $\Gamma \vdash t' : A$.
3. Type equality: If $\mathcal{D} :: \Gamma \vdash A = A' : \mathsf{Type}$ then $\Gamma \vdash A : \mathsf{Type}$ and $\Gamma \vdash A' : \mathsf{Type}$.

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

- Case:
$$\textsc{conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A = B : \mathsf{Type}}{\Gamma \vdash t : B}$$
By induction hypothesis (3.), $\Gamma \vdash B : \mathsf{Type}$.

- Case:
$$\textsc{pair-i}\ \frac{\Gamma, x{:}A \vdash B : \mathsf{Type} \qquad \Gamma \vdash s : A \qquad \Gamma \vdash t : B[s/x]}{\Gamma \vdash (s, t) : \mathsf{Pair}\, A\, (\lambda x B)}$$
By induction hypotheses, $\Gamma \vdash A : \mathsf{Type}$, and $\Gamma \vdash B[s/x] : \mathsf{Type}$. By assumption $\Gamma, x : A \vdash B : \mathsf{Type}$, from which we conclude $\Gamma \vdash \mathsf{Pair}\, A\, (\lambda x B) : \mathsf{Type}$ by rule PAIR-F.

- 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 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' (Lemma 2.9). This conversion is possible since we obtain $\Gamma \vdash A' = A : \mathsf{Type}$ by symmetry (Lemma 2.10).

- Case:

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

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

- Case:

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

By induction hypothesis, $\Gamma, x : A \vdash B : \mathsf{Type}$, and we get the first goal $\Gamma \vdash B[s/x] : \mathsf{Type}$ by the substitution lemma. Directly, we obtain $\Gamma \vdash \lambda x t : \mathsf{Fun}\, A\, (\lambda x B)$ and the second goal $\Gamma \vdash (\lambda x t)\, s : B[s/x]$. Again by substitution, the last goal $\Gamma \vdash t[s/x] : B[s/x]$ follows.

- Case:

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

W. l. o. g., x is not bound by context Γ. By induction hypothesis, $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x B) : \mathsf{Type}$. 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 $\Gamma, x{:}A \vdash t\, x : B$ by FUN-E and $\Gamma \vdash \lambda x.\, t\, x : \mathsf{Fun}\, A\, (\lambda x B)$ by FUN-I. □

After having established syntactical validity, we can drop all underlined hypotheses from the lemmata.

Figure 5 recapitulates the dependencies between the lemmata that lead up to syntactic validity.

The following lemma is more or less a consequence of substitution and functionality for typing, but it will allow for a more concise reasoning in Section 4.

Lemma 2.12. (Functionality for equality)

1. If $\Gamma, x{:}A, \Gamma' \vdash t = t' : C$ and $\Gamma \vdash s = s' : A$ then $\Gamma, \Gamma'[s/x] \vdash t[s/x] = t'[s'/x] : C[s/x]$.
2. If $\Gamma, x{:}A, \Gamma' \vdash C = C' : \mathsf{Type}$ and $\Gamma \vdash s = s' : A$ then $\Gamma, \Gamma'[s/x] \vdash C[s/x] = C'[s'/x] : \mathsf{Type}$.

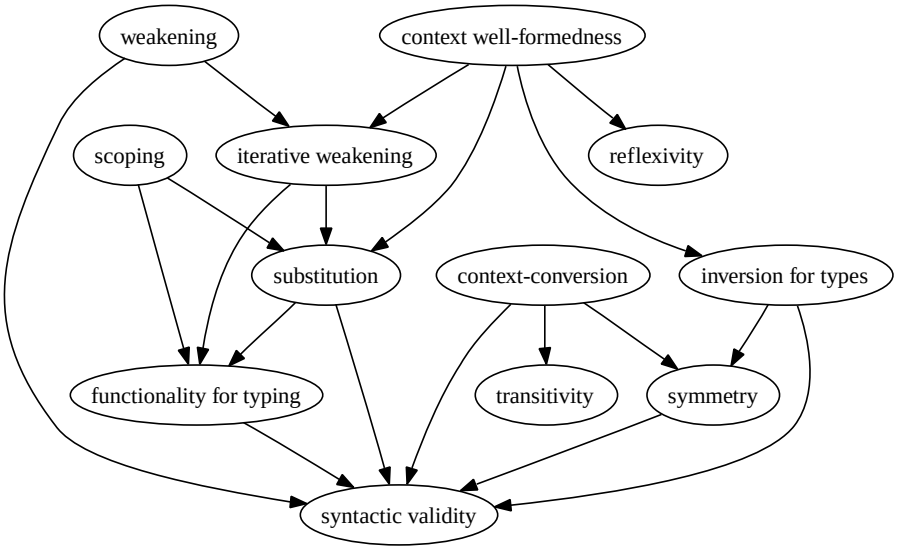


Figure 5. Lemma dependencies.

Proof:
Direct (cf. Harper and Pfenning [17]). We show the proof of the first proposition:

$\Gamma \vdash s : A$	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

□

The next lemma will be central for the soundness proof of algorithmic equality in Section 4. Other names for the theorem are *generation lemma* or *stripping lemma*, and it is central for proving subject reduction (which we will not do in this article).

Lemma 2.13. (Inversion of Typing)

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

5. If $\mathcal{D} :: \Gamma \vdash r\mathsf{L} : A$ then $\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)$.

6. If $\mathcal{D} :: \Gamma \vdash r\mathsf{R} : C$ then $\Gamma \vdash r : \mathsf{Pair}\, A\, (\lambda x B)$ and $\Gamma \vdash B[r\mathsf{L}/x] = C : \mathsf{Type}$.

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:
$$\textsc{conv}\ \frac{\Gamma \vdash \lambda x t : C \qquad \Gamma \vdash C = C' : \mathsf{Type}}{\Gamma \vdash \lambda x t : C'}$$
By induction hypothesis $C \equiv \mathsf{Fun}\, A\, (\lambda x B)$ and $\Gamma, x : A \vdash t : B$. By injectivity, $C' \equiv \mathsf{Fun}\, A'\, (\lambda x B')$ 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 (Theorem 2.1), inversion, 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 2.3. (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.

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

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

Weak head evaluation. We define simultaneously two judgements:

$$
\begin{array}{lll}
\_ \searrow \_ & \subseteq & \mathsf{Exp} \times \mathsf{WVal} \\
\_@\_ \searrow \_ & \subseteq & \mathsf{WVal} \times \mathsf{WElim} \times \mathsf{WVal}
\end{array}
$$

Weak head evaluation $t \searrow w$.

$$
\textsc{eval-c}\ \frac{}{c \searrow c} \qquad \textsc{eval-var}\ \frac{}{x \searrow x}
$$

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

$$
\textsc{eval-pair-i}\ \frac{}{(t, t') \searrow (t, t')} \qquad \textsc{eval-pair-e}\ \frac{r \searrow w_p \qquad w_p @ p \searrow w}{r\,p \searrow w}
$$

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

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

$$
\textsc{elim-pair-l}\ \frac{t \searrow w}{(t, t') @ \mathsf{L} \searrow w} \qquad \textsc{elim-pair-r}\ \frac{t' \searrow w}{(t, t') @ \mathsf{R} \searrow w}
$$

Weak head evaluation $t \searrow w$ is equivalent to multi-step weak head reduction to weak head 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 [17] and Sarnat [25], the first four rules AQ-C, AQ-VAR, AQ-NE-FUN and AQ-NE-PAIR compute *structural equality*, whereas the remaining 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 [8], which we obtain by removing 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 [9, 22, 17] for β-normal terms. The judgement $\Gamma \vdash t \Downarrow A$ infers type A from neutral terms t and the simultaneously defined judgement $\Gamma \vdash t \Uparrow C$ checks whether the β-normal term t has type C. A third judgement $\Gamma \vdash A \Downarrow \mathsf{Type}$ identifies wellformed types $A \in \mathsf{Ty}$ and depends on the type-checking judgement. Although the algorithm works only for well-formed contexts, well-formed types, and β-normal terms, we do not presuppose these properties in the *definition* of the judgements. However, these conditions will appear in the soundness (4.1) and completeness (9.2) theorems.

Type inference $\Gamma \vdash t \Downarrow A$. (Input: Γ well-formed, t neutral and β-normal. Output: A with $\Gamma \vdash t : A$.)

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

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

Type checking $\Gamma \vdash t \Uparrow A$. (Input: Γ, C with $\Gamma \vdash C : \mathsf{Type}$, t β-normal. Output: none.)

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

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

Type well-formedness $\Gamma \vdash A \Downarrow \mathsf{Type}$. (Input: Γ well-formed, A type. Output: none.)

$$\textsc{chk-set-f}\ \frac{}{\Gamma \vdash \mathsf{Set} \Downarrow \mathsf{Type}} \qquad \textsc{chk-set-e}\ \frac{\Gamma \vdash t \Uparrow \mathsf{Set}}{\Gamma \vdash \mathsf{El}\, t \Downarrow \mathsf{Type}}$$

$$\textsc{chk-dep-f}\ \frac{\Gamma \vdash A \Downarrow \mathsf{Type} \qquad \Gamma,\, x{:}A \vdash B \Downarrow \mathsf{Type}}{\Gamma \vdash c\, A\, (\lambda x B) \Downarrow \mathsf{Type}}\ c \in \{\mathsf{Fun}, \mathsf{Pair}\}$$

When starting type-checking $\Gamma \vdash t \Uparrow C$ of term t, we suppose that C has been checked for well-formedness before. Also, we suppose one starts the type-checker with an empty context in the beginning and adds only checked types to the context. Under these assumptions, one can see that all rules maintain a checked context—the only rules that extend the context by a type A are CHK-FUN-I and CHK-DEP-F, and in both cases A has been checked before. Another observation is that only checked terms are substituted into types (rules INF-FUN-E, INF-PAIR-E-R, and CHK-PAIR-I). This implies that inference $\Gamma \vdash t \Downarrow A$ always returns a well-formed type: in case INF-VAR, since it comes from the checked context, and in the other cases, since it results from a substitution of a checked term into a well-formed type. Finally, we can conclude that equality test $A \sim B$ is only invoked on well-formed types A, B. All these observations will be proven formally in the next section and in Section 9.

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 2.1) and inversion of typing (Lemma 2.13). 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 $\Gamma \vdash w\, e = w' : C$.

Note that we do not need a corresponding lemma for weak head evaluation of types, since they are neutral and therefore always weak head normal.

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
$\Gamma \vdash r\, s = w_f\, s : C$	CONV
$\Gamma \vdash w_f\, s : C$	syntactic validity
$\Gamma \vdash w_f\, s = w : C$	second ind. hyp.
$\Gamma \vdash r\, s = w : C$	EQ-TRANS

- Case:

$$\text{ELIM-FUN}\ \frac{t[s/x] \searrow w}{(\lambda x t)@s \searrow w}$$

$\Gamma \vdash (\lambda x t)\ s : C$	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 terms: If $\mathcal{D} :: n \sim n'$ and $\Gamma \vdash n : C$ and $\Gamma \vdash n' : C'$ 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 terms: 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
$\Gamma \vdash n : \mathsf{Fun}\,A\,(\lambda x B)$	&
$\Gamma \vdash s : A$	&
$\Gamma \vdash B[s/x] = C : \mathsf{Type}$	inversion
$\Gamma \vdash n'\,s' : C'$	hypothesis
$\Gamma \vdash n' : \mathsf{Fun}\,A'\,(\lambda x B')$	&
$\Gamma \vdash s' : A'$	&
$\Gamma \vdash B'[s'/x] = C' : \mathsf{Type}$	inversion
$\Gamma \vdash n = n' : \mathsf{Fun}\,A\,(\lambda x B)$	&
$\Gamma \vdash \mathsf{Fun}\,A\,(\lambda x B) = \mathsf{Fun}\,A'\,(\lambda x B') : \mathsf{Type}$	first ind. hyp.
$\Gamma \vdash A = A' : \mathsf{Type}$	injectivity
$\Gamma \vdash s' : A$	symmetry, 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
$\Gamma \vdash n\,s = n'\,s' : C$	rules EQ-FUN-E,CONV

- Case (instance of AQ-EXT-FUN with $w_f \equiv \lambda x t$ and $w'_f = n$):

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

$\Gamma \vdash \lambda x t : C$	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, CONV

□

Lemma 4.3. (Soundness of type conversion)
Let $C \sim C'$. If $\mathcal{D} :: \Gamma \vdash C : \mathsf{Type}$ and $\Gamma \vdash C' : \mathsf{Type}$, then $\Gamma \vdash C = C' : \mathsf{Type}$.

Proof:
By induction on $\mathcal{D}$. For instance, if $C \equiv \mathsf{Fun}\, A\, (\lambda x B)$, we obtain by inversion for types the two derivations $\Gamma \vdash A : \mathsf{Type}$ and $\Gamma, x : A \vdash B : \mathsf{Type}$ which are shorter than $\mathcal{D}$. By inversion on the derivation of $\mathsf{Fun}\, A\, (\lambda x B) \sim C'$ using the fact that C' is a type, we can determine $C' \equiv \mathsf{Fun}\, A'\, (\lambda x B')$ and both $A \sim A'$ and $\lambda x B \sim \lambda x B'$. The last proposition can only be derived from $B \sim B'$. Inverting the well-formedness derivation of C' and using context conversion gives us $\Gamma \vdash A' : \mathsf{Type}$ and $\Gamma, x{:}A \vdash B' : \mathsf{Type}$. Now we can assemble the induction hypotheses $\Gamma \vdash A = A' : \mathsf{Type}$ and $\Gamma, x{:}A \vdash B = B' : \mathsf{Type}$ to conclude $\Gamma \vdash C = C' : \mathsf{Type}$. □

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

Proof:
All three propositions by induction on $\mathcal{D}$, the first two simultaneously, then the third. □

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 [13, 14], 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

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

Herein, we use the following entities:

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

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

$$\begin{array}{lrcll} \text{DEN-CONST} & c\rho & = & c & \text{if } c\in\mathsf{Const}\\ \text{DEN-VAR} & x\rho & = & \rho(x) & \\ \text{DEN-FUN-E} & (r\,s)\rho & = & r\rho\,(s\rho) & \\ \text{DEN-PAIR-E} & (r\,p)\rho & = & r\rho\,p & \\ & & & & \\ \text{DEN-}\beta & t\rho & = & t'\rho & \text{if } t=_\beta t'\\ \text{DEN-IRR} & t\rho & = & t\rho' & \text{if } \rho(x)=\rho'(x) \text{ for all } x\in\mathsf{FV}(t)\end{array}$$

These laws axiomatize a *syntactical λ-algebra* [6, 5.3.2.(ii)] extended by projections. In an earlier version of this work [3] we required D to be a *syntactical λ-model*, with weak extensionality. However, this is an unnecessary requirement and excludes closed term models.

The following laws for β are admissible:

$$\begin{array}{lrcl} \text{DEN-FUN-}\beta & (\lambda x t)\rho\,v & = & t(\rho, x=v)\\ & & & \\ \text{DEN-PAIR-}\beta\text{-L} & (r,s)\rho\,\mathsf{L} & = & r\rho\\ \text{DEN-PAIR-}\beta\text{-R} & (r,s)\rho\,\mathsf{R} & = & s\rho\end{array}$$

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

$$\begin{array}{lll} & (\lambda xt)\rho\ v & \\ = & (\lambda xt)\rho\ x(\rho, x{=}v) & \text{DEN-VAR} \\ = & (\lambda xt)(\rho, x{=}v)\ x(\rho, x{=}v) & \text{DEN-IRR} \\ = & ((\lambda xt)\ x)(\rho, x{=}v) & \text{DEN-FUN-E} \\ = & t(\rho, x{=}v) & \text{DEN-}\beta. \end{array}$$

□

The substitution property is a consequence of β-equality:

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

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

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

$$\begin{array}{lrcll} \text{DEN-SET-NOT-EL} & \mathsf{Set} & \neq & \mathsf{El}\ v & \\ \text{DEN-SET-NOT-DEP} & \mathsf{Set} & \neq & c\ V\ F & \\ \text{DEN-EL-NOT-DEP} & \mathsf{El}\ v & \neq & c\ V\ F & \\ \\ \text{DEN-EL-INJ} & \mathsf{El}\ v & = & \mathsf{El}\ v' & \text{implies } v = v' \\ \text{DEN-DEP-INJ} & c\ V\ F & = & c'\ V'\ F' & \text{implies } c = c' \text{ and } V = V' \text{ and } F = F' \end{array}$$

5.2. PER Models

In the definition of PER models, we follow a paper of the second author with Pollack and Takeyama [10] and Vaux [27]. 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 first cited work defines a new syntactic class of type expressions on top of D, there being the untyped λ-calculus, and the second cited work introduces 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 relations (PERs) and families. 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 $v \in \mathcal{A}$. Let $\mathsf{Fam}(\mathcal{A})$ be the set of functions $\mathcal{F} \in \mathcal{A} \to \mathsf{Per}$ such that $\mathcal{F}(v) = \mathcal{F}(v')$ if $(v, v') \in \mathcal{A}$.

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 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 5.1. $\mathcal{T}ype$ and $[\_]$ are an instance of an inductive-recursive definition [12]. Appendix B presents an alternative formulation, via a relation which is not a priori a PER, and a partial function.

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:

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

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

- Case:

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

$\Gamma \in \mathcal{C}xt$	ind. hyp.
$\rho = \rho' \in [\Gamma]$	assumption
$\mathsf{Fun}\ (A\rho)\ ((\lambda x.B)\rho) = \mathsf{Fun}\ (A\rho')\ ((\lambda x.B)\rho') \in \mathcal{T}ype$	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:

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

$\Gamma \in \mathcal{C}xt$	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. $[\Gamma, x\!:\!A]$
$B(\rho, x{=}s\rho) = B(\rho', x{=}s\rho') \in \mathcal{T}ype$	ind. hyp.
$(B[s/x])\rho = (B[s/x])\rho' \in \mathcal{T}ype$	subst. (Lemma 5.1)
$t(\rho, x{=}s\rho) = t(\rho', x{=}s\rho') \in [B(\rho, x{=}s\rho)]$	ind. hyp.
$(\lambda x t)\rho\ (s\rho) = (t[s/x])\rho' \in [(B[s/x])\rho]$	DEN-FUN-β, subst.

- Case:

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

$\Gamma \in \mathcal{C}xt$	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
$(r\,\mathsf{L},\, r\,\mathsf{R})\rho\,\mathsf{L} = r\rho'\,\mathsf{L} \in [A\rho]$ 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" [27]. A PER is safe if it lies between a PER $\mathcal{N}$ on *neutral* expressions and a PER $\mathcal{S}$ on *safe* expressions [28]. 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 $\mathcal{N} \subseteq \mathcal{A} \subseteq \mathcal{S}$. Usually there is just one safety range in scope and then all uses of "safe" refer to this one.

Lemma 5.3. (Fun and Pair preserve safety)
Let $\mathcal{R} = (\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair})$ be a safety range. If $\mathcal{A} \in \mathsf{Per}$ is safe (w. r. t. $\mathcal{R}$) and $\mathcal{F} \in \mathsf{Fam}(\mathcal{A})$ is such that $\mathcal{F}(v)$ is safe (w. r. t. $\mathcal{R}$) for all $v \in \mathcal{A}$ then $\mathcal{F}un(\mathcal{A}, \mathcal{F})$ and $\mathcal{P}air(\mathcal{A}, \mathcal{F})$ are safe (w. r. t. $\mathcal{R}$).

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 $(\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair})$ be a safety range. 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.

We define β-reduction $\longrightarrow_\beta$ as the closure of the axioms $(\lambda x t)\, s \longrightarrow_\beta t[s/x]$, $(t, t')\,\mathsf{L} \longrightarrow_\beta t$, and $(t, t')\,\mathsf{R} \longrightarrow_\beta t'$ under all term constructors. Its reflexive-transitive closure $\longrightarrow_\beta^*$ is confluent, which enables us to define β-equality $t =_\beta t'$ as *there exists some* t_0 *such that* $t \longrightarrow_\beta^* t_0 \,{}_\beta^*\!\longleftarrow t'$.

Let $\overline{r} \in \mathsf{D}$ denote the equivalence class of $r \in \mathsf{Exp}$ with regard to $=_\beta$. Similarly, let $\overline{\rho} \in \mathsf{Var} \to \mathsf{D}$ denote the equivalence class of $\rho \in \mathsf{Var} \to \mathsf{Exp}$, meaning that $\overline{\rho}(x) = \{t \mid t =_\beta \rho(x)\}$.

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

Herein, $t[\rho]$ denotes the substitution of $\rho(x)$ for x in t, carried out in parallel for all $x \in \mathsf{FV}(t)$.

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]

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

If we weaken the assumption such that ρ and ρ' coincide 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 follow from confluence and the fact that El, Fun, and Pair are unanimated constants. □

[2] Benzmüller, Brown, and Kohlhase [7] prove a similar result by converting t into an SK-combinatorial term.

η-reduction and -equality. Let one-step η-reduction be the least congruence generated by the axioms $\lambda x.\, r\, x \longrightarrow_\eta r$ if $x \notin \mathsf{FV}(r)$ and $(r\,\mathsf{L},\ r\,\mathsf{R}) \longrightarrow_\eta r$. It is locally confluent and preserves free variables. Its reflexive-transitive closure $\longrightarrow_\eta^*$ is strongly normalizing (each step removes an abstraction or a pair) and confluent (Newman's Lemma). Because of confluence, we can define η-equality $t =_\eta t'$ as $t \longrightarrow_\eta^* t_0 \,{}_\eta^*\!\longleftarrow t'$. In this article, we are only interested in η for β-normal forms.

Value classes. The β-normal forms $v \in \mathsf{Val}$ can be described by the following grammar. As it will become clear at the end of this section, they completely represent the β-equivalence classes $\bar{t} \in \mathsf{Exp}/{=_\beta}$ of well-typed terms t.

VNe	$\ni$	u	$::=$	$c \mid x \mid u\,v \mid u\,p$	neutral values
VFun	$\ni$	v_f	$::=$	$u \mid \lambda x v$	function values
VPair	$\ni$	v_p	$::=$	$u \mid (v, v')$	pair values
Val	$\ni$	v	$::=$	$v_f \mid v_p$	values

Note that η-reduction on β-normal forms does not create β-redexes, hence it is well-defined on Val. Neutral values reduce to neutral values, so it is even well-defined on VNe. Even on values, it does not preserve typing, see Remark 2.1.

Lemma 6.2. (Inversion properties of $\longrightarrow_\eta^*$)

1. If $\mathcal{D} :: x \longrightarrow_\eta^* t$ then $t \equiv x$. If $\mathcal{D} :: c \longrightarrow_\eta^* t$ then $t \equiv c$.
2. If $\mathcal{D} :: r\,s \longrightarrow_\eta^* t$ then $t \equiv r'\,s'$ with $r \longrightarrow_\eta^* r'$ and $s \longrightarrow_\eta^* s'$.
3. If $\mathcal{D} :: r\,p \longrightarrow_\eta^* t$ then $t \equiv r'\,p$ with $r \longrightarrow_\eta^* r'$.
4. If $\mathcal{D} :: \lambda x v \longrightarrow_\eta^* t$ then either
   - $t \equiv u$ neutral value, $x \notin \mathsf{FV}(u)$, and $v \longrightarrow_\eta^* u\,x$, or
   - $t \equiv \lambda x v'$ and $v \longrightarrow_\eta^* v'$.
5. If $\mathcal{D} :: (v_1, v_2) \longrightarrow_\eta^* t$ then either
   - $t \equiv u$ neutral and both $v_1 \longrightarrow_\eta^* u\,\mathsf{L}$ and $v_2 \longrightarrow_\eta^* u\,\mathsf{R}$, or
   - $t \equiv (v_1', v_2')$ and both $v_1 \longrightarrow_\eta^* v_1'$ and $v_2 \longrightarrow_\eta^* v_2'$.

Proof:
Each by induction on $\mathcal{D}$. Proposition 4 relies on the fact that if $\lambda x.\, r\, x$ is β-normal, then r must be neutral. □

As a consequence of these inversion properties, we can decompose η-equations between values as follows. Some of these decompositions, e. g. 5., do not carry over to arbitrary terms: We have $\lambda x.\, (\lambda x z)\, x =_\eta \lambda x z$, but not $(\lambda x z)\, x =_\eta z$.

Corollary 6.1. (Inversion on $=_\eta$)

1. If $x =_\eta u_0$ then $u_0 \equiv x$. If $c =_\eta u_0$ then $u_0 \equiv c$.

2. If $u\,v =_\eta u_0$ then $u_0 \equiv u'\,v'$ with $u =_\eta u'$ and $v =_\eta v'$.
3. If $u\,p =_\eta u_0$ then $u_0 \equiv u'\,p$ with $u =_\eta u'$.
4. If $\lambda x v =_\eta u$ then $v =_\eta u\,x$ and $x \notin \mathsf{FV}(u)$.
5. If $\lambda x v =_\eta \lambda x v'$ 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 $(v_1, v_2) =_\eta \lambda x v$ 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 η-equality is an equivalence on Val, the relation

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

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 can only be 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 $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair} \subseteq \mathcal{S} := \simeq$.

$$\begin{array}{lcll}
(\overline{t},\, \overline{t'}) \in \mathcal{N} & :\Longleftrightarrow & t =_\beta u \;=_\eta u' =_\beta t' & \text{for some } u, u' \in \mathsf{VNe} \\
(\overline{t},\, \overline{t'}) \in \mathcal{S}_{fun} & :\Longleftrightarrow & t =_\beta v_f =_\eta v_f' =_\beta t' & \text{for some } v_f, v_f' \in \mathsf{VFun} \\
(\overline{t},\, \overline{t'}) \in \mathcal{S}_{pair} & :\Longleftrightarrow & t =_\beta v_p =_\eta v_p' =_\beta t' & \text{for some } v_p, v_p' \in \mathsf{VPair}
\end{array}$$

Lemma 6.3. $\mathcal{N}, \mathcal{S}_{fun}, \mathcal{S}_{pair} \in \mathsf{Per}$.

Lemma 6.4. (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. 6.1, item 2.
- Case $v \equiv \lambda x v_0$ and $v' \equiv u$ neutral. By assumption, $x \notin \mathsf{FV}(u)$ and $v\,x \simeq u\,x$, 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 $v \equiv \lambda x v_0$ and $v' \equiv \lambda x v_0'$. From the assumption we get $v_0 =_\eta v_0'$ by β-reduction. Hence, $\lambda x v_0 =_\eta \lambda x v_0'$.
- Case $v \equiv (v_1,\, v_2)$. Then $(v_1,\, v_2)\,x$ does not reduce to β-normal form, which is a contradiction to the assumption.

□

Corollary 6.2. (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 $u \equiv u' \equiv x \notin \mathsf{FV}(v, v')$. □

Lemma 6.5. (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 6.3. (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. 6.2 and SAFE-EXT-PAIR by Lemma 6.5. □

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

Lemma 6.6. (Context satisfiable)
Let $\rho_0(x) := x$ for all $x \in \mathsf{Var}$. If $\Gamma \vdash \mathsf{ok}$, then $\overline{\rho_0} \in [\Gamma]$.

Theorem 6.1. (Equal terms are related)
If $\Gamma \vdash t = t' : C$ then $\bar{t} \simeq \bar{t}'$.

Proof:
Note that $t\overline{\rho_0} = \bar{t}$. By soundness of MLF_Σ (Thm. 5.1), $t\overline{\rho_0} = t'\overline{\rho_0} \in [C\overline{\rho_0}]$. The claim follows since $[C\overline{\rho_0}] \subseteq \mathcal{S}$ by Lemma 5.4. □

Corollary 6.4. (Equal types are related)
If $\mathcal{D} :: \Gamma \vdash A = A' : \mathsf{Type}$ then $\overline{A} \simeq \overline{A}'$.

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

We have shown that each well-typed term is β-normalizable and two judgmentally equal terms $\beta\eta$-reduce to the same normal form. This gives us a decision procedure for equality of well-typed terms: By Theorem 5.1, instantiated to the term model of this section, well-typed terms $\Gamma \vdash t, t' : C$ are β-normalizable, hence, the test $t \simeq t'$ terminates. If the test succeeds, then $\Gamma \vdash t = t' : C$ holds; this is a consequence of Cor. 7.2, Lemma 7.1, and Lemma 4.2, and will be formally established in the next section. If the test fails, then by Theorem 6.1, $\Gamma \vdash t = t' : C$ cannot be true.

It remains to show that our algorithmic equality is also a decision procedure. In the next section, we demonstrate that $\bar{t} \simeq \bar{t}'$ 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-normal forms $\simeq$ to the algorithmic equality $\sim$, we first present a transitive extension $\overset{+}{\sim}$ of the algorithmic equality which is conservative for terms of the same type. We then show that this extension $\overset{+}{\sim}$ 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 $w \overset{+}{\sim} w'$ be given by the rules for algorithmic equality plus the following two:

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

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

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 $\overset{+}{\sim}$ is conservative for same-typed terms)

1. If $\mathcal{D} :: n \overset{+}{\sim} n'$ and $\Gamma \vdash n : C$ and $\Gamma \vdash n' : C'$ then $n \sim n'$ and $\Gamma \vdash C = C' : \mathsf{Type}$.
2. If $\mathcal{D} :: w \overset{+}{\sim} w'$ and $\Gamma \vdash w, w' : C$ then $w \sim w'$.
3. If $t\!\downarrow \overset{+}{\sim} t'\!\downarrow$ and $\Gamma \vdash t, t' : C$ then $t\!\downarrow \sim t'\!\downarrow$.

Proof:
The third proposition is a direct consequence of the second, using subject reduction for weak head evaluation which is implied by its soundness (Lemma 4.1) together with syntactic validity (Theorem 2.1). The first two propositions are proven simultaneously by induction on $\mathcal{D}$. 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. The most interesting case is the following one:

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

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

□

As a consequence, the algorithmic equality is transitive for terms of the same type, provided $\overset{+}{\sim}$ 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 $\overset{+}{\sim}$ 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 $\overset{+}{\sim}$, 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. (Soundness of weak head evaluation w. r. t. β-equality)
If $t \searrow w$ then $t =_\beta w$.

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

Proof:
By induction on $\mathcal{D}$, using the previous lemma. 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, $n =_\beta u =_\eta u' =_\beta n'$ and $s =_\beta v =_\eta v' =_\beta s'$. Thus, $n\,s =_\beta u\,v =_\eta u'\,v' =_\beta n'\,s'$.

- Case
$$\text{AQ}^+\text{-EXT-FUN-NE}\ \frac{t{\downarrow} \overset{+}{\sim} n\,x}{\lambda x t \overset{+}{\sim} n}\ x \notin \mathsf{FV}(n)$$
By assumption $t \searrow w$ and by induction hypothesis $w \simeq n\,x$. Together with the previous lemma, $t =_\beta w =_\beta v =_\eta u\,x =_\beta n\,x$, hence, $\lambda x t =_\beta \lambda x v =_\eta \lambda x.\, u\,x =_\eta u =_\beta n$.

- Case
$$\text{AQ}^+\text{-FUN-PAIR}\ \frac{t{\downarrow} \overset{+}{\sim} n\,x \qquad n\,\mathsf{L} \overset{+}{\sim} r{\downarrow} \qquad n\,\mathsf{R} \overset{+}{\sim} s{\downarrow}}{\lambda x t \overset{+}{\sim} (r, s)}\ x \notin \mathsf{FV}(n)$$
By the previous lemma and induction hypothesis, $t =_\beta v =_\eta u\,x =_\beta n\,x$, hence, $\lambda x v =_\eta \lambda x.\, u\,x =_\eta u$. Further, $n\,\mathsf{L} =_\beta u\,\mathsf{L} =_\eta v_1 =_\beta r$ and $n\,\mathsf{R} =_\beta u\,\mathsf{R} =_\eta v_2 =_\beta s$, thus, $u =_\eta (u\,\mathsf{L},\ u\,\mathsf{R}) =_\eta (v_1, v_2)$. Together, $\lambda x t =_\beta \lambda x v =_\eta (v_1, v_2) =_\beta (r, s)$.

□

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

7.3. Completeness of the Extended Algorithmic Equality

In the following, we demonstrate that two terms $t \simeq t'$ that are related in the term-PER model are also related by 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 $|\lambda x r| + |t| > |r| + |t\,x|$ and $|(r,s)| + |t| > |r| + |t\,\mathsf{L}|$. These conditions are also met by Goguen's [14] measure for proving termination of Coquand's [8] algorithmic equality restricted to pure λ-terms. But we need the extra conditions $|\lambda x t| > 2|t|$ and both $|(r,s)| > 2|r|$ and $|(r,s)| > 2|s|$.

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

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:
Lemma 7.4 is proved by induction on $|v| + |v'|$. We first treat the cases for neutral terms $u =_\eta u'$.

- Case $u \equiv c$. Then $u' \equiv c$ by Cor. 6.1 and $c \overset{+}{\sim} c$.
- Case $u \equiv x$. Similar.
- Case $u \equiv u_1\,v_1$. Then by Cor. 6.1 $u' \equiv u_2\,v_2$ with $u_1 =_\eta u_2$ and $v_1 =_\eta v_2$. By induction hypothesis $u_1 \overset{+}{\sim} u_2$ and $v_1 \overset{+}{\sim} v_2$, hence $u \overset{+}{\sim} u'$ by AQ$^+$-NE-FUN.
- Case $u \equiv u_1\,p$. Similar.

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

- Case $\lambda x v =_\eta u$. By Cor. 6.1, $v =_\eta u\,x$. Since $|v| + |u\,x| = |v| + |u| + 1 < (2|v| + 1) + |u| = |\lambda x v| + |u|$, we can apply the induction hypothesis and obtain $v \overset{+}{\sim} u\,x$. Thus $\lambda x v \overset{+}{\sim} u$ by AQ$^+$-EXT-FUN-NE.
- Case $\lambda x v =_\eta \lambda x v'$. By Cor. 6.1, $v =_\eta v'$, on which we apply the induction hypothesis and AQ$^+$-EXT-FUN-FUN.
- Case $\lambda x v =_\eta (v_1, v_2)$. By Cor. 6.1 there exists a neutral u such that $v \longrightarrow_\eta^* u\,x$ and both $u\,\mathsf{L} \overset{*}{\underset{\eta}{\longleftarrow}} v_1$ and $u\,\mathsf{R} \overset{*}{\underset{\eta}{\longleftarrow}} v_2$. 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 $v \overset{+}{\sim} u\,x$. Further, we have $|u\,\mathsf{L}| + |v_1| \leq 2|v_1| < 2|v_1| + 2|v_2| = |(v_1, v_2)|$, thus, by induction hypothesis, $u\,\mathsf{L} \overset{+}{\sim} v_1$, and similarly, $u\,\mathsf{R} \overset{+}{\sim} v_2$. By AQ$^+$-FUN-PAIR we get $\lambda x v \overset{+}{\sim} (v_1, v_2)$.

- Case $(v_1, v_2) =_\eta u$. By Cor. 6.1, $v_1 =_\eta u\,\mathsf{L}$ 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, $v_1 \overset{+}{\sim} u\,\mathsf{L}$, and with a similar calculation, $v_2 \overset{+}{\sim} u\,\mathsf{R}$. Thus, $(v_1, v_2) \overset{+}{\sim} u$ by AQ$^+$-EXT-PAIR-NE.
- Case $(v_1, v_2) =_\eta (v_1', v_2')$. By inversion, induction hypothesis, and rule AQ$^+$-EXT-PAIR-PAIR.

□

Remark 7.1. (Alternative proof)
First, show reflexivity $v \overset{+}{\sim} v$ for all β-normal forms v by induction on v. Then prove that $\overset{+}{\sim}$ is closed under η-expansion. More precisely, show that

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

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

Lemma 7.5. (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 [6]. □

Lemma 7.6. (From normal to normalizing terms)
1. If $n =_\beta u$ and $n' =_\beta u'$ and $\mathcal{D} :: u \overset{+}{\sim} u'$, then $n \overset{+}{\sim} n'$.
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

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

$n \equiv n_0\, s$ with $n_0 =_\beta u$ and $s =_\beta v$	neutral $n \longrightarrow^*_\beta u\, v$ application
$n' \equiv n'_0\, s'$ with $n'_0 =_\beta u'$ and $s' =_\beta v'$	neutral $n' \longrightarrow^*_\beta u'\, v'$ application
$n_0 \overset{+}{\sim} n'_0$	first ind. hyp.
$s{\downarrow} \overset{+}{\sim} s'{\downarrow}$	second ind. hyp.
$n_0\, s \overset{+}{\sim} n'_0\, s'$	AQ$^+$-NE-FUN

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

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

$t \searrow \lambda x r$ 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
$r{\downarrow} \overset{+}{\sim} n\, x$	induction hypothesis
$\lambda x r \overset{+}{\sim} n$	AQ$^+$-EXT-FUN-NE

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

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

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

□

Corollary 7.2. (Completeness of $\overset{+}{\sim}$)
If $t \simeq t'$ then $t{\downarrow} \overset{+}{\sim} t'{\downarrow}$.

Proof:
By assumption $t =_\beta v =_\eta v' =_\beta t'$. First $v \overset{+}{\sim} v'$ by Lemma 7.4, then also $t{\downarrow} \overset{+}{\sim} t'{\downarrow}$ by Lemma 7.6. □

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

Together with Cor. 7.1 we see that the diagonal of extended algorithmic equality—which coincides with the diagonal of pure algorithmic equality—characterizes the weakly normalizing terms t. Here, we consider a term weakly normalizing if is has a *junk-free* normal form v, which excludes β-normal forms like $(\lambda x x)\,\mathsf{L}$ or $(x, y)\,z$. We 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 to Joachimski and Matthes' [18] inductive characterization of weakly normalizing λ-terms. (Only that they use weak head reduction instead of weak head evaluation.)[3]

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 $\Gamma \vdash t = t' : C$ 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 Theorem 6.1 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 the admissible rule

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

and SAFE-EXT-PAIR by a similar admissible rule 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 $\mathcal{D} :: w \stackrel{+}{\sim} w'$:

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

[3] The inductive characterization originates from van Raamsdonk et. al. [24].

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 $\mathcal{D}_1 :: \lambda x t \sim n$ 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. ($\overset{+}{\sim}$ is transitive)
Let $\vec{e}$ be a possibly empty list of eliminations.

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

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

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

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

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

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

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

- Case

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

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

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

$\mathcal{E}' :: n_0 \overset{+}{\sim} n'\,\vec{e}$	$\mathcal{E}' < \mathcal{D}_{11} + \mathcal{D}_{2}$	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.
$\mathcal{E} :: \lambda x t \overset{+}{\sim} n'\,\vec{e}$	$\mathcal{E} < \mathcal{D}'_1 + \mathcal{D}_2 + 1 = \mathcal{D}_1 + \mathcal{D}_2$	AQ$^+$-EXT-FUN-NE

- Case

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

$\mathcal{E}_1 :: r\downarrow \overset{+}{\sim} n'\,\vec{e}\,\mathsf{L}$	$\mathcal{E}_1 < \mathcal{D}_{11} + \mathcal{D}_2$	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 $\lambda x t \overset{+}{\sim} n$ and $n \overset{+}{\sim} \lambda x t'$.
- Case $\lambda x t \overset{+}{\sim} \lambda x t'$ and $\lambda x t' \overset{+}{\sim} n$ (plus symmetrical case).
- Case $\lambda x t_1 \overset{+}{\sim} \lambda x t_2$ and $\lambda x t_2 \overset{+}{\sim} \lambda x t_3$.
- 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 symmetrical 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

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

  $\mathcal{E} :: \lambda x t \overset{+}{\sim} (r,s)$ 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 $n \overset{+}{\sim} \lambda x t$. Symmetrical.

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

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

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

$\mathcal{E}_1 :: n\,x \overset{+}{\sim} n'\,x$	$\mathcal{E}_1 < \mathcal{D}'_1 + \mathcal{D}'_2$	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 ($x \notin \mathsf{FV}(n, n')$)

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

$\mathcal{E}' :: t{\downarrow} \overset{+}{\sim} n'\,x$	$\mathcal{E}' < \mathcal{D}'_1 + \mathcal{D}_{21}$	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

$$\mathcal{D}_1 = \frac{\begin{array}{ccc}\mathcal{D}_{11} & \mathcal{D}_{12} & \mathcal{D}_{13} \\ r{\downarrow} \overset{+}{\sim} n\,\mathsf{L} & s{\downarrow} \overset{+}{\sim} n\,\mathsf{R} & n\,x \overset{+}{\sim} t{\downarrow}\end{array}}{(r, s) \overset{+}{\sim} \lambda x t}\; x \notin \mathsf{FV}(n)$$

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

$\mathcal{E}_1 :: n\,x \overset{+}{\sim} n'\,x$	$\mathcal{E}_1 < \mathcal{D}_{13} + \mathcal{D}_{21}$	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.
$\mathcal{E}_7 :: s{\downarrow} \overset{+}{\sim} s'{\downarrow}$	$\mathcal{E}_7 < \mathcal{E}_5 + \mathcal{D}_{23} < \mathcal{D}_{12} + \mathcal{D}_{13} + \mathcal{D}_{21} + \mathcal{D}_{23}$	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 $(r,s) \overset{+}{\sim} \lambda x t$.
- Case $(r,s) \overset{+}{\sim} (r',s')$ and $(r',s') \overset{+}{\sim} \lambda x t$.
- Case $\lambda x t \overset{+}{\sim} (r,s)$ and $(r,s) \overset{+}{\sim} \lambda x t'$.

□

9. Decidability

By completeness of algorithmic equality, every well-typed 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 well-typed 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*.[4]

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:

[4] A similar measure is used by Goguen [14] to prove termination of algorithmic equality restricted to pure λ-terms [8].

- 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 $|\mathcal{D}_1| + |\mathcal{D}_2| \geq |\mathcal{D}_1| + |\mathcal{D}_2'| + 2 > (|\mathcal{D}_1| + 1) + |\mathcal{D}_2'| = |\hat{\mathcal{D}}_1| + |\mathcal{D}_2'|$. 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. □

Corollary 9.1. (Decidability of type equality)
If $\mathcal{D} :: \Gamma \vdash A : \mathsf{Type}$ and $\Gamma \vdash A' : \mathsf{Type}$ then the query $A \sim A'$ succeeds or fails finitely and decides $\Gamma \vdash A = A' : \mathsf{Type}$.

Proof:
By induction on $\mathcal{D}$, using the theorem in case $A = \mathsf{El}\ t$. □

9.2. Termination of Type Checking

The termination of the type checker is a consequence of termination of equality for well-typed objects.

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

1. The query $\Gamma \vdash t \Downarrow ?$ terminates ($? \not\equiv \mathsf{Type}$).
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 2.7)
$\Gamma \vdash r\,s \Downarrow B[s/x]$, 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 additionally 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 $\Gamma \vdash t : C$ then $\Gamma \vdash t \Downarrow A$ and $A \sim C$.
2. If $\mathcal{D} :: t \Uparrow$ and $\Gamma \vdash t : C$ then $\Gamma \vdash t \Uparrow C$.

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

Corollary 9.2. (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. □

Our bidirectional algorithm can only check β-normal terms; however, it can be extended to arbitrary terms by requiring type annotations in redexes [11, 2], some of which can also be inferred by suitable heuristics [23].

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 [17]—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 its shape, e. g., unit types, singleton types and signatures with manifest fields [10].

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 [10] 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 [25] have been working on an extension of the Edinburgh Logical Framework (ELF) by Σ-types (LF_Σ), following Harper and Pfenning [17]. In comparison to MLF_Σ, syntactic validity (Theorem 2.1) and injectivity are non-trivial in their formulation of ELF. Robin Adams [4] 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 [13] 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 [14] has proven termination and completeness for both the type-directed [17] and the shape-directed equality [8] 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 [15].

Acknowledgments. We are grateful to Lionel Vaux whose clear presentation of models for this implicit calculus [27] 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. Thanks to the anonymous referees for their helpful comments on the draft version, especially to the one who spotted many small mistakes by rigorously checking our proofs.

APPENDIX.

A. Surjective Pairing Destroys Confluence

Klop [19, pp. 195–208] shows that the untyped λ-calculus with the reduction $(r\,\mathsf{L},\ r\,\mathsf{R}) \longrightarrow r$, called surjective pairing, 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.

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

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:

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

The end reducts of both sequences cannot be joined again.

Not that only the last step in both sequences was a surjective pairing reduction, all others were β-steps. Thus, the example refutes even confluence of the relation $\longrightarrow_\beta^* \longrightarrow_\eta^*$.

Remark A.1. (η-postponement fails)
Surjective pairing destroys also η-postponement. In the reduction sequence $((\lambda xt)\,\mathsf{L},\ (\lambda xt)\,\mathsf{R})\,s \longrightarrow_\eta (\lambda xt)\,s \longrightarrow_\beta t[s/x]$, the β-redex is created by η-reduction and cannot be reduced first.

B. 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.

$$\begin{array}{lrcl}
\textsc{int-set-f} & [\mathsf{Set}] & = & \mathcal{S}et \\
\textsc{int-set-e} & [\mathsf{El}\ v] & = & \mathcal{E}\ell(v) \\
\textsc{int-fun-f} & [\mathsf{Fun}\ V\ F] & = & \mathcal{F}un([V], v \mapsto [F\ v]) \\
\textsc{int-pair-f} & [\mathsf{Pair}\ V\ F] & = & \mathcal{P}air([V], v \mapsto [F\ v])
\end{array}$$

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

$$\textsc{tyeq-set-f}\ \frac{}{\mathsf{Set} = \mathsf{Set} \in \mathcal{T}ype} \qquad \textsc{tyeq-set-e}\ \frac{v = v' \in \mathcal{S}et}{\mathsf{El}\ v = \mathsf{El}\ v' \in \mathcal{T}ype}$$

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

In the last rule, if $[V]$ is not defined, the quantification is to be read as empty. The name *intensional type equality* [10] shall distinguish it from the more extensional type equality given by $[V] = [V']$.

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

Proof:
By structural induction on $\mathcal{D}$. 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 $[F\ v] = [F\ v'] \in \mathsf{Per}$, hence, the family $\mathcal{F}$, defined by $\mathcal{F}(v) := [F\ v]$, is in $\mathsf{Fam}([V])$, since v and v' were arbitrary. Analogously, for 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 itself a PER.

Lemma B.3. (Intensional type equality is a PER)

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 structural induction on $\mathcal{D}$. For transitivity (1.), we consider the case:

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

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

By soundness of intensional type equality (Lemma B.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] Abel, A.: Termination Checking with Types, *RAIRO – Theoretical Informatics and Applications*, **38**(4), 2004, 277–319, Special Issue: Fixed Points in Computer Science (FICS'03).

[3] Abel, A., Coquand, T.: Untyped Algorithmic Equality for Martin-Löf's Logical Framework with Surjective Pairs, *Typed Lambda Calculi and Applications (TLCA 2005), Nara, Japan* (P. Urzyczyn, Ed.), 3461, Springer-Verlag, April 2005.

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

[5] Adams, R.: Pure Type Systems with Judgemental Equality, *Journal of Functional Programming*, **16**(2), 2006, 219–246.

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

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

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

[9] 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, May 1996.

[10] Coquand, T., Pollack, R., Takeyama, M.: A Logical Framework with Dependently Typed Records, *Fundamenta Informaticae*, **65**(1-2), 2005, 113–134.

[11] Davies, R., Pfenning, F.: Intersection Types and Computational Effects, *Proceedings of the fifth ACM SIGPLAN International Conference on Functional Programming (ICFP 2000), Montreal, Canada*, September 2000.

[12] Dybjer, P.: A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory, *The Journal of Symbolic Logic*, **65**(2), 2000, 525–549.

[13] Goguen, H.: Soundness of the Logical Framework for Its Typed Operational Semantics, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings* (J.-Y. Girard, Ed.), 1581, Springer-Verlag, 1999, ISBN 3-540-65763-0.

[14] 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-Verlag, 2005, ISBN 3-540-25388-2.

[15] 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 Press, January 2005, ISBN 1-58113-830-X.

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

[17] 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.

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

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

[20] Lambek, J., Scott, P. J.: *Introduction to Higher Order Categorical Logic*, Cambridge University Press, New York, NY, USA, 1988, ISBN 0-521-35653-9.

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

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

[23] Pottier, F.: A modern eye on ML type inference: old techniques and recent developments, September 2005, Lecture notes for the APPSEM Summer School.

[24] van Raamsdonk, F., Severi, P., Sørensen, M. H., Xi, H.: Perpetual Reductions in Lambda Calculus, *Information and Computation*, **149**(2), March 1999, 173–225.

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

[26] 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.

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

[28] Vouillon, J.: Subtyping Union Types, *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings* (J. Marcinkowski, A. Tarlecki, Eds.), 3210, Springer-Verlag, 2004.