Normalization by Evaluation for Martin-Löf Type Theory with Typed Equality Judgements

Andreas Abel *
Institut für Informatik
Ludwig-Maximilians-Universität
Oettingenstr. 67, D-80538 München
abel@tcs.ifi.lmu.de

Thierry Coquand
Department of Computer Science and Engineering
Chalmers University of Technology
Rännvägen 6, S-41296 Göteborg
coquand@cs.chalmers.se

Peter Dybjer †
Department of Computer Science and Engineering
Chalmers University of Technology
Rännvägen 6, S-41296 Göteborg
peterd@cs.chalmers.se

Abstract

The decidability of equality is proved for Martin-Löf type theory with a universe á la Russell and typed beta-eta-equality judgements. A corollary of this result is that the constructor for dependent function types is injective, a property which is crucial for establishing the correctness of the type-checking algorithm. The decision procedure uses normalization by evaluation, an algorithm which first interprets terms in a domain with untyped semantic elements and then extracts normal forms. The correctness of this algorithm is established using a PER-model and a logical relation between syntax and semantics.

1. Introduction

The most important metatheorems of Martin-Löf's intensional intuitionistic type theory [21, 26] (and related intensional type theories such as the Calculus of Constructions) state that its judgements are decidable. These theorems are fundamental both philosophically and practically. They are the main reasons for preferring intensional to extensional type theory [22, 11]. It is a tenet of the philosophy of constructivism that it should be mechanically decidable whether a certain construction a is a witness to the truth of a given proposition A. Moreover, proof checkers for intensional type theories such as Coq, Agda, and Epigram, all rely on the decision algorithm for $a : A$ using normalization: the user attempts to prove the theorem A by building a construction a, and the system checks that a is indeed a proof of A.

Typed equality judgements. In spite of many years of research into the metatheory of type theory a completely satisfactory story has yet to be told. Systems with an untyped notion of conversion, such as Barendregt's pure type systems, are well-understood, but there are reasons why theories with a typed notion of conversion (*equality judgements*) are more fundamental. Firstly, the constructive meaning of equality in the sense of Martin-Löf [22, 23] is relative to a type; two objects are not just equal, they are equal with respect to a type. For this reason all systems of Martin-Löf type theory from 1973 and onwards [21, 22, 26] use equality judgements rather than untyped conversion. Secondly, systems with a typed notion of conversion have clearer mathematical semantics: we can compute the semantics of an element $a : A$ in an arbitrary model of type theory, for example, defined as a category with families [15].

Although some authors have studied the decidability problem for various systems with typed conversion (Martin-Löf [21], Coquand [12], Altenkirch [7], Adams [4]), there is still a need for a clear treatment of the main system of Martin-Löf type theory. Goguen [17] uses typed operational semantics to develop the metatheory of Luo's system UTT including normalization and Church-Rosser. UTT is an impredicative system having a predicative subsystem which is a version of Martin-Löf type theory. It differs from our sys-

*Research partially supported by the EU coordination action *TYPES* (510996).

†Research partially supported by VR-project *Typed Lambda Calculus and Applications*.

tem since UTT's rules for the universe are formulated à la Tarski rather than à la Russell. Moreover, unlike our system, UTT does not have η-conversion on the universe level. More recently, Harper and Pfenning [18] have addressed the problem for the logical framework (LF), with the aim of finding a simpler treatment of its metatheory. However, their method requires injectivity of the dependent function type constructor to be proved early in the development of LF's metatheory. It is not clear how to extend this method to a system of dependent type theory with a universe of small types. The reader is referred to Harper and Pfenning [18] for discussion and references to related work.

Normalization by evaluation (nbe). We shall employ this method for proving the main decidability theorems for the theory $\lambda^{\Pi\mathsf{U}\mathsf{N}}$ of Martin-Löf type theory with one universe and natural numbers and typed equality judgements. An important advantage of nbe is that it bypasses the need for a separate proof of the Church-Rosser property, a discovery due to Peter Hancock [21]. Instead it follows directly from the model construction that convertible terms have equal normal forms.

The nbe-method has previously been used for the combinatory version of Martin-Löf type theory [21] with a weak notion of conversion. This theory was however abandoned, and the most recent version of Martin-Löf type theory is based on a logical framework with a type of "sets" (a universe of small types) and dependent function types with β and η-conversion. In recent unpublished work Martin-Löf [24] has shown how to construct an nbe-algorithm for this system, incorporating the technique developed by Berger and Schwichtenberg [10] for the simply-typed λ-calculus. Martin-Löf did however not consider a system with a universe of small types closed under dependent function sets which is the main difficulty here. Such a system was instead considered by Abel, Aehlig, and Dybjer [1] who used ideas from Aehlig and Joachimski's work on untyped nbe [5]. However, the system in [1] has an untyped notion of conversion and thus differs from the present one; it was not known whether the two systems are equivalent.

The use of nbe has made it possible both to strengthen and to simplify previous treatments of the metatheory of Martin-Löf type theory with a universe and typed equality judgements. Furthermore, our method generalizes to dependent product types (Σ-types) and also to singleton types, and thus generalizes results by Stone and Harper [30].

We expect that our results will play a key role in the theoretical justification of proof systems for Martin-Löf type theory, including the new version of the Agda system currently being developed in Göteborg.

Plan of the paper. In Section 2 we introduce our version of Martin-Löf type theory $\lambda^{\Pi\mathsf{U}\mathsf{N}}$ with typed equality judgements. The main point here is to give an nbe-algorithm for this theory and to prove it correct. This algorithm consists of an evaluation function given in 3.1 and reification and reflection functions given in 3.2, and studied further as a system of rewrite rules in 3.3. The central property of the nbe-algorithm is that it returns unique representatives (normal forms) for equivalence class of convertible terms and convertible types. We prove this property in two steps: *completeness* in Section 3 and *soundness* in Section 4.

By completeness we mean that terms or types which are convertible (we can prove the respective equality judgements) map to identical normal forms. We establish this in Section 3 by defining a family of PERs expressing a notion of typed equality in the model. We then show that the interpretation function maps convertible terms and types to equivalent elements in the model and that reification maps equivalent elements to equal normal forms.

By soundness we mean that a term or type is convertible to its normal form as returned by the nbe-algorithm. To this end we define a logical relation between syntactic expressions and elements in the model in Section 4. Having proved both soundness and completeness the two main results of the paper (decidability of equality and injectivity of Π) follow as corollaries.

In Section 5 we explain how our method extends if we add unit types and dependent product types (Σ-types), and also can be adapted to some other variations of the theory.

2. Syntax and inference rules

In this section we introduce a version of Martin-Löf type theory $\lambda^{\Pi\mathsf{U}\mathsf{N}}$ with a type of natural numbers N and a universe U of small types. In this system we can build families of small types (elements of U) by primitive recursion. As a consequence, we cannot erase type dependencies, a technique which can otherwise be used in the metatheoretic analysis of the system [18].

The inference rules of the theory are quite standard: we have the usual rules for the dependently typed lambda calculus formulated with equality judgements, the usual rules for N, and the usual rules for a universe U à la Russell. We remark that we have both β and η-conversion for Π-types. Moreover, the substitution rule is primitive rather than admissible, and thus standard properties of the system are proved easily. We also remark that the nbe-method is quite robust and not so dependent on the precise formulation of the inference rules. It is only when proving decidability of type-checking that we need to know that typing rules are invertible (see Lemma 2), something we can show for our system, even though we have a primitive substitution rule.

Expressions (terms and types). Since we work with Russell-style universes it is natural to have a common syn-

tactic category Λ for terms and types:

$$\begin{array}{llll} \mathsf{Var} & \ni & x, y, z & \\ \mathsf{Const} & \ni & c & ::= \mathsf{Fun} \mid \mathsf{U} \mid \mathsf{N} \mid \mathsf{s} \mid \mathsf{z} \mid \mathsf{rec} \\ \Lambda & \ni & r, s, t & ::= c \mid x \mid \lambda x.t \mid r\, s \end{array}$$

The contexts are generated by

$$\mathsf{Cxt} \quad \ni \quad \Gamma, \Delta \quad ::= \quad \diamond \mid \Gamma, x\!:\!A$$

To aid the reader, we use the letters A, B, C for expressions which are to be understood as types and r, s, t for terms. Dependent function types, usually written $\Pi x : A.\, B$, are written $\mathsf{Fun}\, A\, (\lambda x.B)$. When B does not depend on x we have a non-dependent function type and write $A \to B$.

We identify expressions up to α-conversion and adopt the convention that in contexts Γ all variables must be distinct. Hence, we can view Γ as a map from variables to types with finite domain $\mathsf{dom}(\Gamma)$ and let $\Gamma(x) = A$ iff $(x\!:\!A) \in \Gamma$. In context extensions $\Gamma, x\!:\!A$ we presuppose $x \notin \mathsf{dom}(\Gamma)$. As usual $\mathsf{FV}(t)$ is the set of free variables of t. We let $\mathsf{FV}(t_1, \ldots, t_n) = \mathsf{FV}(t_1) \cup \cdots \cup \mathsf{FV}(t_n)$ and $\mathsf{FV}(\Gamma) = \bigcup_{x \in \mathsf{dom}(\Gamma)} \mathsf{FV}(\Gamma(x))$.

Simultaneous substitutions $\sigma, \tau \in \mathsf{Var} \to \Lambda$ are mappings from variables to expressions where the set $\{x \mid \sigma(x) \neq x\}$, called $\mathsf{dom}(\sigma)$, is finite. We set $\mathsf{FV}(\sigma) = \bigcup_{x \in \mathsf{dom}(\sigma)} \mathsf{FV}(\sigma(x))$. The identity substitution is denoted by σ_{id}. When $f \in \mathsf{Var} \to S$ for some set S, $x \in \mathsf{Var}$ and $a \in S$, let $f[x \mapsto a]$ be the function f' defined by $f'(x) = a$ and $f'(y) = f(y)$ for $y \neq x$. This update operation will be used for substitutions and later also for environments mapping variables to elements of a semantic domain.

The term $t[\sigma]$ is the result of substituting $\sigma(x)$ for x in t for all $x \in \mathsf{FV}(t)$ in parallel. Parallel substitution $\Delta[\sigma]$ into contexts is defined by $(\Delta[\sigma])(x) = (\Delta(x))[\sigma]$ and composition of substitutions $\sigma[\sigma']$ by $(\sigma[\sigma'])(x) = (\sigma(x))[\sigma']$.

Judgements. The type theory $\lambda^{\Pi\mathsf{UN}}$ has the following forms of judgement:

$\Gamma \vdash$	Γ is a well-formed context
$\Gamma \vdash A$	A is a well-formed type in Γ
$\Gamma \vdash t : A$	t has type A in Γ
$\Gamma \vdash \sigma : \Delta$	σ is a well-formed substitution in Γ
$\Gamma \vdash A = A'$	A and A' are equal types in Γ
$\Gamma \vdash t = t' : A$	t and t' are equal terms of type A in Γ
$\Gamma \vdash \sigma = \sigma' : \Delta$	σ and σ' are equal substitutions in Γ

Note that the judgement $\Gamma \vdash A : \mathsf{U}$ stating that A is a well-formed *small* type in Γ, is a special case of the third form.

For an arbitrary judgement, we write $\Gamma \vdash J$, where J is a collection of the syntactic entities (terms, contexts, substitutions) to the right of $\vdash$ in a judgement. The notation $J[\sigma]$ is understood as the substitution of σ into all entities in J, and $\mathsf{FV}(J)$ is the union of the free variable sets of all entities in J. Exceptions are $\mathsf{FV}(\sigma : \Delta)$, which is defined as $\bigcup_{x \in \mathsf{dom}(\Delta)} \mathsf{FV}(\Delta(x), \sigma(x))$, and $\mathsf{FV}(\sigma = \sigma' : \Delta) = \bigcup_{x \in \mathsf{dom}(\Delta)} \mathsf{FV}(\Delta(x), \sigma(x), \sigma'(x))$.

In our inference rules, we use the abbreviation $\mathsf{Rec}\,(\lambda x.C)$ for the type of the primitive recursion combinator rec with the result type C depending on $x : \mathsf{N}$.

$$\begin{array}{l} \mathsf{Rec}\,(\lambda x.C) = C[\mathsf{z}/x] \to \\ \qquad\qquad (\mathsf{Fun}\ \mathsf{N}\ (\lambda n.\, C[n/x] \to C[\mathsf{s}\, n/x])) \to \\ \qquad\qquad \mathsf{Fun}\ \mathsf{N}\ (\lambda x.C) \end{array}$$

The inference rules are given in Figure 1.

In the following, we state some simple syntactic properties of $\lambda^{\Pi\mathsf{UN}}$ which will be needed in Section 4. Since substitution is a primitive rule these properties are easier to establish than in systems without this primitive rule [18, 2]. On the other hand, if substitution is a primitive rule, the injectivity of Π does not follow easily by syntactic considerations. Instead, this will be one of our main results.

Lemma 1 (Basic properties)

1. *(Scope:) If* $\Gamma \vdash J$ *then* $\mathsf{FV}(J) \subseteq \mathsf{dom}(\Gamma)$.
2. *(Context well-formedness:) If* $\Gamma, x : A, \Gamma' \vdash J$*, then* $\Gamma \vdash A$.
3. *(Weakening:) If* $\Gamma, \Gamma' \vdash J$ *and both* $\Gamma \vdash A$ *and* $x \notin \mathsf{dom}(\Gamma, \Gamma')$*, then* $\Gamma, x\!:\!A, \Gamma' \vdash J$.
4. *(Context conversion:) Let* $\Gamma \vdash B = A$ *and* $\Gamma \vdash B$*. If* $\Gamma, x\!:\!A, \Gamma' \vdash J$ *then* $\Gamma, x\!:\!B, \Gamma' \vdash J$.
5. *(Substitution:)*
   - *(a) If* $\Gamma \vdash \sigma : \Delta$ *and* $x \in \mathsf{dom}(\Delta)$ *then* $\Gamma \vdash \sigma(x) : \Delta(x)[\sigma]$.
   - *(b)* $\Gamma \vdash _ = _ : \Delta$ *is an equivalence relation.*
   - *(c) If* $\Gamma \vdash$ *then* $\Gamma \vdash \sigma_{\mathsf{id}} : \Gamma$ *and* $\Gamma \vdash \sigma_{\mathsf{id}} = \sigma_{\mathsf{id}} : \Gamma$.

Lemma 2 (Inversion)

1. *If* $\Gamma \vdash \mathsf{z} : C$ *then* $\Gamma \vdash \mathsf{N} = C$.
2. *If* $\Gamma \vdash \mathsf{s} : C$ *then* $\Gamma \vdash \mathsf{N} \to \mathsf{N} = C$.
3. *If* $\Gamma \vdash \mathsf{rec}\,(\lambda x.A) : C$ *then* $\Gamma \vdash \mathsf{Rec}\,(\lambda x.A) = C$.
4. *If* $\Gamma \vdash x : C$ *then* $\Gamma \vdash \Gamma(x) = C$.
5. *If* $\Gamma \vdash \lambda x.t : C$ *then* $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x.B) = C$ *and* $\Gamma, x\!:\!A \vdash t : B$.
6. *If* $\Gamma \vdash r\, s : C$ *then* $\Gamma \vdash r : \mathsf{Fun}\, A\, (\lambda x.B)$ *with* $\Gamma \vdash s : A$ *and* $\Gamma \vdash B[s/x] = C$.
7. *If* $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x.B) : C$ *then* $\Gamma \vdash A : \mathsf{U}$ *and* $\Gamma, x : A \vdash B : \mathsf{U}$ *with* $\Gamma \vdash \mathsf{U} = C$.
8. *If* $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x.B)$ *then* $\Gamma \vdash A$ *and* $\Gamma, x\!:\!A \vdash B$.

Well-formed contexts $\Gamma \vdash$.

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

Well-formed types $\Gamma \vdash A$.

$$\textsc{n-f}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{N}} \qquad \textsc{univ-f}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{U}} \qquad \textsc{fun-f}\ \frac{\Gamma \vdash A \qquad \Gamma, x{:}A \vdash B}{\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x.B)} \qquad \textsc{univ-e}\ \frac{\Gamma \vdash A : \mathsf{U}}{\Gamma \vdash A}$$

Well-typed terms $\Gamma \vdash t : A$.

$$\textsc{hyp}\ \frac{\Gamma \vdash \qquad (x{:}A) \in \Gamma}{\Gamma \vdash x : A} \qquad \textsc{conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A = A'}{\Gamma \vdash t : A'}$$

$$\textsc{univ-i-n}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{N} : \mathsf{U}} \qquad \textsc{univ-i-fun}\ \frac{\Gamma \vdash A : \mathsf{U} \qquad \Gamma, x{:}A \vdash B : \mathsf{U}}{\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x.B) : \mathsf{U}}$$

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

$$\textsc{n-i-z}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{z} : \mathsf{N}} \qquad \textsc{n-i-s}\ \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{s} : \mathsf{N} \to \mathsf{N}} \qquad \textsc{n-e}\ \frac{\Gamma, x{:}\mathsf{N} \vdash C}{\Gamma \vdash \mathsf{rec}\, (\lambda x.C) : \mathsf{Rec}\, (\lambda x.C)}$$

Well-formed substitutions $\Gamma \vdash \sigma : \Delta$ and parallel substitution.

$$\textsc{subst-empty}\ \frac{\Gamma \vdash}{\Gamma \vdash \sigma : \diamond} \qquad \textsc{subst-ext}\ \frac{\Gamma \vdash \sigma : \Delta \qquad \Delta \vdash A \qquad \Gamma \vdash \sigma(x) : A[\sigma]}{\Gamma \vdash \sigma : (\Delta, x{:}A)} \qquad \textsc{subst}\ \frac{\Gamma \vdash \sigma : \Delta \qquad \Delta \vdash J}{\Gamma \vdash J[\sigma]}$$

Equal types $\Gamma \vdash A = A'$. (Congruence and equivalence rules which are omitted.)

Equal terms $\Gamma \vdash t = t' : A$. (Congruence and equivalence rules and the type conversion rule are omitted.)

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

$$\textsc{eq-n-}\iota\textsc{-z}\ \frac{\Gamma, x{:}\mathsf{N} \vdash C \qquad \Gamma \vdash z : C[\mathsf{z}/x] \qquad \Gamma \vdash s : \mathsf{Fun}\ \mathsf{N}\ (\lambda n.\, C[n/x] \to C[\mathsf{s}\, n/x])}{\Gamma \vdash \mathsf{rec}\, (\lambda x.C)\, z\, s\, \mathsf{z} = z : C[\mathsf{z}/x]}$$

$$\textsc{eq-n-}\iota\textsc{-s}\ \frac{\Gamma, x{:}\mathsf{N} \vdash C \qquad \Gamma \vdash z : C[\mathsf{z}/x] \qquad \Gamma \vdash s : \mathsf{Fun}\ \mathsf{N}\ (\lambda n.\, C[n/x] \to C[\mathsf{s}\, n/x]) \qquad \Gamma \vdash n : \mathsf{N}}{\Gamma \vdash \mathsf{rec}\, (\lambda x.C)\, z\, s\, (\mathsf{s}\, n) = s\, n\, (\mathsf{rec}\, (\lambda x.C)\, z\, s\, n) : C[\mathsf{s}\, n/x]}$$

Equal substitutions $\Gamma \vdash \sigma = \sigma' : \Delta$ and functionality.

$$\textsc{eq-subst-empty}\ \frac{\Gamma \vdash}{\Gamma \vdash \sigma = \sigma' : \diamond} \qquad \textsc{eq-subst-ext}\ \frac{\Gamma \vdash \sigma = \sigma' : \Delta \qquad \Delta \vdash A \qquad \Gamma \vdash \sigma(x) = \sigma'(x) : A[\sigma]}{\Gamma \vdash \sigma = \sigma' : (\Delta, x{:}A)}$$

$$\textsc{eq-func-ty}\ \frac{\Gamma \vdash \sigma = \sigma' : \Delta \qquad \Delta \vdash A}{\Gamma \vdash A[\sigma] = A[\sigma']} \qquad \textsc{eq-func}\ \frac{\Gamma \vdash \sigma = \sigma' : \Delta \qquad \Delta \vdash t : A}{\Gamma \vdash t[\sigma] = t[\sigma'] : A[\sigma]}$$

Figure 1. Inference rules for $\lambda^{\Pi\mathsf{UN}}$.

Lemma 3 (Syntactic validity)

1. *Typing: If* $\Gamma \vdash t : A$ *then* $\Gamma \vdash A$.
2. *Equality: If* $\Gamma \vdash t = t' : A$ *then* $\Gamma \vdash A$ *and both* $\Gamma \vdash t : A$ *and* $\Gamma \vdash t' : A$.
3. *Type equality: If* $\Gamma \vdash A = A'$ *then* $\Gamma \vdash A$ *and* $\Gamma \vdash A'$.

3. Models and Completeness of NbE

We now show that if we can prove an equality judgement expressing that two types or terms are equal, then they have the same normal form, as computed by our algorithm. To this end, we build a model of $\lambda^{\Pi\mathsf{UN}}$ over a model of the untyped λ-calculus.

In 3.1 we consider the class of all PER models of $\lambda^{\Pi\mathsf{UN}}$ and show soundness of the inference rules. In 3.2 we add a few new operations and equations to our notion of syntactic combinatory algebra. One of these operations is *reification*. An abstract nbe-algorithm can be obtained by composing evaluation with the reification function of that algebra. To establish the completeness of this nbe-algorithm we define a *residualizing PER-model*. Completeness of nbe follows from the facts that (i) provably equal syntactic terms and types are evaluated to PER-equivalent elements in the model, and (ii) PER-equivalent elements in the model are reified to identical normal forms.

In 3.3 we show a new way to instantiate the underlying syntactic combinatorial algebra of the residualizing PER-model by an extended lambda calculus.

3.1. PER models

A syntactical combinatory algebra consists of a set D with an application operation (juxtaposition) $_\,_ \in \mathsf{D} \times \mathsf{D} \to \mathsf{D}$, constructor constants $\mathsf{Fun}, \mathsf{U}, \mathsf{N}, \mathsf{z}, \mathsf{s}$ and a constant for primitive recursion rec. Each constructor c is injective: $c\,\vec{d} = c\,\vec{e}$ implies $\vec{d} = \vec{e}$. Primitive recursion satisfies the usual equations:

$$\begin{array}{lrcl}
\textsc{den-}\iota\textsc{-z} & \mathsf{rec}\ F\ d_z\ d_s\ \mathsf{z} & = & d_z \\
\textsc{den-}\iota\textsc{-s} & \mathsf{rec}\ F\ d_z\ d_s\ (\mathsf{s}\ e) & = & d_s\ e\ (\mathsf{rec}\ F\ d_z\ d_s\ e)
\end{array}$$

Note that the first argument of rec is a dependent type, the type of the result of the function defined by rec. Furthermore, there is an evaluation function $[\![_]\!]_ \in \Lambda \times \mathsf{Env} \to \mathsf{D}$ satisfying

$$\begin{array}{lrcll}
\textsc{den-const} & [\![c]\!]_\rho & = & c & (c \text{ constant}) \\
\textsc{den-var} & [\![x]\!]_\rho & = & \rho(x) & \\
\textsc{den-fun-e} & [\![r\ s]\!]_\rho & = & [\![r]\!]_\rho\ [\![s]\!]_\rho & \\
\textsc{den-}\beta & [\![\lambda x.t]\!]_\rho\ d & = & [\![t]\!]_{\rho[x \mapsto d]} & \\
\textsc{den-subst} & [\![t[\sigma]]\!]_\rho & = & [\![t]\!]_{[\![\sigma]\!]_\rho} &
\end{array}$$

In the following, we let ρ range over environments in $\mathsf{Env} := \mathsf{Var} \to \mathsf{D}$. We use X, Y, F to range over elements of D which denote types. The denotation $[\![\sigma]\!]_\rho$ of a substitution σ is the environment $[\![\sigma]\!]_\rho(x) = [\![\sigma(x)]\!]_\rho$.

PER models. Let $\mathsf{Per}(\mathsf{D})$ denote the set of all partial equivalences over D. If $\mathcal{A} \in \mathsf{Per}(\mathsf{D})$ we write $d = d' \in \mathcal{A}$ for $(d, d') \in \mathcal{A}$ and $|\mathcal{A}| = \{d \mid d = d \in \mathcal{A}\}$ for the domain of $\mathcal{A}$. We often simply write $d \in \mathcal{A}$ for $d \in |\mathcal{A}|$. If $\mathcal{A} \in \mathsf{Per}(\mathsf{D})$, let $\mathsf{Fam}(\mathcal{A})$ be the set of functions $\mathcal{F} \in |\mathcal{A}| \to \mathsf{Per}(\mathsf{D})$ such that $\mathcal{F}(d) = \mathcal{F}'(d')$ for all $d = d' \in \mathcal{A}$.

A *PER model* of $\lambda^{\Pi\mathsf{UN}}$ consists of a syntactic combinatory algebra and two PERs $\mathcal{U} \subset \mathcal{T}ype \in \mathsf{Per}(\mathsf{D})$ and a family of PERs $[_] \in \mathsf{Fam}(\mathcal{T}ype)$ with the following closure conditions. The first two conditions express that the PER $\mathcal{U}$ of small types contains N and is closed under Π:

PER-1 $\mathsf{N} = \mathsf{N} \in \mathcal{U}$; then $\mathsf{z} = \mathsf{z} \in [\mathsf{N}]$, and $\mathsf{s}\ d = \mathsf{s}\ d' \in [\mathsf{N}]$ if $d = d' \in [\mathsf{N}]$.

PER-2 $\mathsf{Fun}\ X\ F = \mathsf{Fun}\ X'\ F' \in \mathcal{U}$ if $X = X' \in \mathcal{U}$ and $F\,d = F'\,d' \in \mathcal{U}$ for all $d = d' \in [X]$; then $f = f' \in [\mathsf{Fun}\,X\,F]$ if $f\,d = f'\,d' \in [F\,d]$ for all $d = d' \in [X]$.

Moreover, the PER $\mathcal{T}ype$ of all types contains all small types and U, and is closed under Π:

PER-3 $\mathcal{T}ype \supset \mathcal{U}$.

PER-4 $\mathsf{U} = \mathsf{U} \in \mathcal{T}ype$; then $[\mathsf{U}] = \mathcal{U}$.

PER-5 $\mathsf{Fun}\ X\ F = \mathsf{Fun}\ X'\ F' \in \mathcal{T}ype$ if $X = X' \in \mathcal{T}ype$ and $F\,d = F'\,d' \in \mathcal{T}ype$ for all $d = d' \in [X]$; then $f = f' \in [\mathsf{Fun}\,X\,F]$ if $f\,d = f'\,d' \in [F\,d]$ for all $d = d' \in [X]$.

Finally, since the first argument of the recursion-constant rec is a family of types it is not covered by the rules for Π above. Hence to validate the rule EQ-N-E it is necessary to stipulate separately that it preserves PER-equalities:

PER-6 If $F\,d = F'\,d' \in \mathcal{T}ype$ for all $d = d' \in [\mathsf{N}]$, $d_z = d'_z \in [F\,\mathsf{z}]$, $d_s\,d_n\,d_r = d'_s\,d'_n\,d'_r \in [F\,(\mathsf{s}\,d_n)]$ for all $d_n = d'_n \in [\mathsf{N}]$ and $d_r = d'_r \in [F\,d_n]$, and $e = e' \in [\mathsf{N}]$ then $\mathsf{rec}\,F\,d_z\,d_s\,e = \mathsf{rec}\,F'\,d'_z\,d'_s\,e' \in [F\,e]$.

From now on we often write $d = d' \in X$ for $d = d' \in [X]$.

Validity of the judgements in $\lambda^{\Pi\mathsf{UN}}$. Let $\rho = \rho' \in \Gamma$ hold if $\rho(x) = \rho'(x) \in [\![A]\!]_\rho$ for all $(x : A) \in \Gamma$. Define $\Gamma \models J$, meaning that $\Gamma \vdash J$ is valid in a given PER-model,

as follows:

$$
\begin{array}{lcl}
\models & :\Longleftrightarrow & \text{true} \\
\Gamma, x\!:\!A \models & :\Longleftrightarrow & \Gamma \models A \\
\Gamma \models A & :\Longleftrightarrow & \Gamma \models A = A \\
\Gamma \models A = A' & :\Longleftrightarrow & \Gamma \models \text{ and} \\
\quad \forall \rho = \rho' \in \Gamma.\ [\![A]\!]_\rho = [\![A']\!]_{\rho'} \in \mathcal{T}ype & & \\
\Gamma \models t : A & :\Longleftrightarrow & \Gamma \models t = t : A \\
\Gamma \models t = t' : A & :\Longleftrightarrow & \Gamma \models A \text{ and} \\
\quad \forall \rho = \rho' \in \Gamma.\ [\![t]\!]_\rho = [\![t']\!]_{\rho'} \in [\![A]\!]_\rho & & \\
\Gamma \models \sigma : \Delta & :\Longleftrightarrow & \Gamma \models \sigma = \sigma : \Delta \\
\Gamma \models \sigma = \sigma' : \Delta & :\Longleftrightarrow & \Gamma \models \text{ and } \Delta \models \text{ and} \\
\quad \forall \rho = \rho' \in \Gamma.\ [\![\sigma]\!]_\rho = [\![\sigma']\!]_{\rho'} \in \Delta & &
\end{array}
$$

Lemma 4 (Soundness of EQ-FUN-η**)** *If* $\Gamma \models t : \mathsf{Fun}\, A\, (\lambda x.B)$ *and* $x \notin \mathsf{dom}(\Gamma)$ *then* $\Gamma \models \lambda x.\, t\, x = t : \mathsf{Fun}\, A\, (\lambda x.B)$.

Proof. Assume $\rho = \rho' \in \Gamma$ and $d = d' \in [\![A]\!]_\rho$. By the laws of the syntactic combinatory algebra, $[\![\lambda x.\, t\, x]\!]_\rho\, d = [\![t\, x]\!]_{\rho[x \mapsto d]} = [\![t]\!]_{\rho[x \mapsto d]}\, d$. Since $x \notin \mathsf{dom}(\Gamma)$ we have $\rho[x \mapsto d] = \rho' \in \Gamma$, hence, $[\![\lambda x.\, t\, x]\!]_\rho\, d = [\![t]\!]_{\rho[x \mapsto d]}\, d = [\![t]\!]_{\rho'}\, d' \in [\![B]\!]_{\rho[x \mapsto d]}$. □

Lemma 5 (Soundness of EQ-FUNC-TY**)** *If* $\Gamma \models \sigma = \sigma' : \Delta$ *and* $\Delta \models A$ *then* $\Gamma \models A[\sigma] = A[\sigma']$.

Proof. Assume $\rho = \rho' \in \Gamma$. Then $[\![\sigma]\!]_\rho = [\![\sigma']\!]_{\rho'} \in \Delta$ and $[\![A]\!]_{[\![\sigma]\!]_\rho} = [\![A]\!]_{[\![\sigma']\!]_{\rho'}} \in \mathcal{T}ype$ which implies $[\![A[\sigma]]\!]_\rho = [\![A[\sigma']]\!]_{\rho'} \in \mathcal{T}ype$ by DEN-SUBST. □

Theorem 6 (Soundness of the inference rules) *If* $\Gamma \vdash J$ *then* $\Gamma \models J$.

Proof. Standard, by induction on the derivation of $\Gamma \vdash J$. This proof is straightforward since the inference rules were designed with the PER model in mind. □

3.2 An abstract nbe-algorithm and its completeness

We shall now extend our syntactical combinatory algebras with a new constructor Up which maps *neutral terms* (see below) to elements in D and operations for *reification* and *reflection*. Then we can define a normalization by evaluation function by composing the evaluation function with the reification function. To prove the completeness of this algorithm we define a *residualizing* PER-model. We emphasize again that in this section we only define an "abstract" nbe-algorithm, which leaves the choice of the exact implementation of D open. In Section 3.3 we will instantiate D with an extended lambda calculus $\Lambda^{\downarrow\uparrow}$.

We define the set $\mathsf{Nf} \subseteq \Lambda$ of $\beta\iota$-normal forms simultaneously with the set $\mathsf{Ne} \subseteq \Lambda$ of neutral terms (normal forms with a head variable).

$$
\begin{array}{ll}
\mathsf{Nf} \ni v, w, V, W & ::= \lambda x.v \mid \mathsf{z} \mid \mathsf{s}\, v \mid \mathsf{U} \mid \mathsf{Fun}\, V\, \lambda x.W \mid u \\
\mathsf{Ne} \ni u & ::= x \mid u\, v \mid \mathsf{rec}\, V\, v\, w\, u
\end{array}
$$

A residualizing PER-model. Given a syntactical combinatory algebra with an additional injective constructor $\mathsf{Up} : \mathsf{Ne} \to \mathsf{D}$, we define a *residualizing* PER model as follows.

First define $\mathcal{N} \in \mathsf{Per}(\mathsf{D})$ inductively by the following rules.

$$
\frac{}{\mathsf{z} = \mathsf{z} \in \mathcal{N}} \qquad \frac{d = d' \in \mathcal{N}}{\mathsf{s}\, d = \mathsf{s}\, d' \in \mathcal{N}} \qquad \frac{}{\mathsf{Up}\, u = \mathsf{Up}\, u \in \mathcal{N}}
$$

We then give a simultaneous inductive-recursive definition [16] of the PERs $\mathcal{U}$ and $[d]$. There are several ways to understand such definitions set-theoretically [28, 3, 9]. If $\mathsf{Rel}(\mathsf{D})$ is the set of relations on D, we can e. g. follow [6, 1] and inductively define the graph of a partial function $[_] : \mathsf{D} \rightharpoonup \mathsf{Rel}(\mathsf{D})$ such that the following equations hold:

$$
\begin{array}{lcl}
[\mathsf{Up}\, u] & = & \{(\mathsf{Up}\, u',\ \mathsf{Up}\, u') \mid u' \in \mathsf{Ne}\} \\
[\mathsf{N}] & = & \mathcal{N} \\
[\mathsf{Fun}\, X\, F] & = & \{(f, f') \mid (f\, d,\ f'\, d') \in [F\, d] \\
 & & \qquad \text{for all } (d, d') \in [X]\}
\end{array}
$$

To prove univalence of this relation we use that Up, N, and Fun are constructors. Then we define the universe of small types $\mathcal{U} \in \mathsf{Rel}(\mathsf{D})$ inductively (using PER notation, although we have not yet proved that it is a PER):

$$
\frac{}{\mathsf{Up}\, u = \mathsf{Up}\, u \in \mathcal{U}} \qquad \frac{}{\mathsf{N} = \mathsf{N} \in \mathcal{U}}
$$

$$
\frac{X = X' \in \mathcal{U} \qquad F\, d = F'\, d' \in \mathcal{U} \text{ for all } d = d' \in X}{\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{U}}
$$

We then show by induction on this relation that if $X = X' \in \mathcal{U}$ then $[X]$ and $[X']$ are well-defined and equal PERs. It follows that $\mathcal{U} \in \mathsf{Per}(\mathsf{D})$. We extend the partial function by $[\mathsf{U}] = \mathcal{U}$ and define the universe of all types $\mathcal{T}ype \in \mathsf{Rel}(\mathsf{D})$ inductively.

$$
\mathcal{U} \subseteq \mathcal{T}ype \qquad \frac{}{\mathsf{U} = \mathsf{U} \in \mathcal{T}ype}
$$

$$
\frac{X = X' \in \mathcal{T}ype \quad F\, d = F'\, d' \in \mathcal{T}ype \text{ for all } d = d' \in X}{\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{T}ype}
$$

As for $\mathcal{U}$, we can now show that $[_] \in \mathsf{Fam}(\mathcal{T}ype)$ and $\mathcal{T}ype \in \mathsf{Per}(\mathsf{D})$, and it is immediate to check that $(\mathcal{U}, \mathcal{T}ype, [_])$ constitutes a PER model of $\lambda^{\Pi\mathsf{UN}}$, validating PER-1-5. The remaining requirement PER-6 will be fulfilled in the next section.

Reflection and reification. Assume furthermore, that for each $X \in \mathcal{T}ype$ there are functions

$$\begin{array}{lcll} \uparrow^X & \in & \mathsf{Ne} \to |[X]| & \text{reflection,} \\ \downarrow^X & \in & |[X]| \to \mathsf{Nf} & \text{reification,} \\ \Downarrow & \in & |\mathcal{T}ype| \to \mathsf{Nf} & \text{type reification,} \end{array}$$

such that the following equations in D hold:

$$\begin{array}{lcl} (\uparrow^{\mathsf{Fun}\,X\,F} u)\, d & = & \uparrow^{F\,d}(u \downarrow^X d) \\ \uparrow^X u & = & \mathsf{Up}\, u \end{array}$$

where X in the second equation is a term beginning with a constructor different from Fun. Moreover, we have the following syntactical identities in Λ:

$$\begin{array}{lcll} \downarrow^{\mathsf{Fun}\,X\,F} f & = & \lambda x. \downarrow^{F\,(\uparrow^X x)}(f\,(\uparrow^X x)) & (*) \\ \downarrow^{\mathsf{N}} \mathsf{z} & = & \mathsf{z} & \\ \downarrow^{\mathsf{N}} (\mathsf{s}\, d) & = & \mathsf{s}\,(\downarrow^{\mathsf{N}} d) & \\ \downarrow^{\mathsf{N}} (\mathsf{Up}\, u) & = & u & \\ \downarrow^{\mathsf{Up}\,u'} (\mathsf{Up}\, u) & = & u & \\ \downarrow^{\mathsf{U}} X & = & \Downarrow X & \\ \Downarrow(\mathsf{Fun}\, X\, F) & = & \mathsf{Fun}\,(\Downarrow X)\,(\lambda x. \Downarrow(F\,(\uparrow^X x))) & (*) \\ \Downarrow \mathsf{N} & = & \mathsf{N} & \\ \Downarrow(\mathsf{Up}\, u) & = & u & \\ \Downarrow \mathsf{U} & = & \mathsf{U} & \end{array}$$

The equations (*) shall hold for *all but finitely many* $x \in \mathsf{Var}$. Since we consider λ-terms modulo α-equivalence, this is equivalent to postulating them for some "fresh" x. One way to make this precise is to require D to be a nominal set [27] or FM-domain [29]. We let D be the λ-calculus modulo β and rewrite rules for recursion, reflection and reification. We come to that later, for now let us just assume that the reification functions always choose fresh variables x.

The following central lemma shows that the above equations define total reflection and reification functions on the PERs of our model, and their result does not depend on the choice of representative for each equivalence class.

Lemma 7 (Characterization of reflection and reification) *Let $X = X' \in \mathcal{T}ype$. Then $\uparrow^X u = \uparrow^{X'} u \in X$ and $\Downarrow X = \Downarrow X'$, and if $d = d' \in X$ then $\downarrow^X d = \downarrow^{X'} d'$.*

Proof. Simultaneously by induction on $X = X' \in \mathcal{T}ype$, showing the lemma for $X = X' \in \mathcal{U}$ first. In case of function types $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{U}$ and $f = f' \in \mathsf{Fun}\, X\, F$ we have by induction hypothesis $\uparrow^X x = \uparrow^{X'} x \in X$ for a fresh x, hence, $f \uparrow^X x = f' \uparrow^{X'} x \in F \uparrow^X x$. Since $F \uparrow^X x = F' \uparrow^{X'} x \in \mathcal{U}$, by induction hypothesis $\downarrow^{F \uparrow^X x}(f \uparrow^X x)$ and $\downarrow^{F' \uparrow^{X'} x}(f' \uparrow^{X'} x)$ are identical $\beta\iota$-normal forms. Hence, we can abstract x in both terms, which shows that $\downarrow^{\mathsf{Fun}\,X\,F} f = \downarrow^{\mathsf{Fun}\,X'\,F'} f' \in \mathsf{Nf}$. This was the most interesting case. □

Since we have a new constructor $e = \mathsf{Up}\, r$ for neutral values, we need to add a new equation for rec:

$$\begin{array}{l} \mathsf{rec}\; F\; d_z\; d_s\; (\mathsf{Up}\, r) = \\ \quad \uparrow^{F\,(\mathsf{Up}\, r)}(\mathsf{rec}\;\; (\lambda x. \Downarrow(F\;(\uparrow^N x))) \\ \qquad\qquad\qquad (\downarrow^{F\,\mathsf{z}} d_z) \\ \qquad\qquad\qquad (\lambda n \lambda y. \downarrow^{F\,(\mathsf{s}\, n)}(d_s\;(\uparrow^N n)\;(\uparrow^{F\,n} y))) \\ \qquad\qquad\qquad r) \end{array}$$

The variables x, n, y must be fresh for F, s. We can now prove PER-6, the totality of rec, by induction on $\mathcal{N}$.

Summarizing the developments in this section, we have a complete method for checking judgmental equality:

Corollary 8 (Completeness of NbE) *Let $\rho = \rho \in \Gamma$.*

1. *If $\Gamma \vdash A = A'$ then $\Downarrow [\![A]\!]_\rho = \Downarrow [\![A']\!]_\rho$.*
2. *If $\Gamma \vdash t = t' : A$ then $\downarrow^{[\![A]\!]_\rho} [\![t]\!]_\rho = \downarrow^{[\![A]\!]_\rho} [\![t']\!]_\rho$.*

Proof. We have $[\![A]\!]_\rho = [\![A']\!]_\rho \in \mathcal{T}ype$ by Thm. 6, hence $\Downarrow [\![A]\!]_\rho$ is syntactically equal to $\Downarrow [\![A']\!]_\rho$ by the lemma. Analogously for term equality. □

3.3 Extended λ-calculus

We shall now instantiate the domain D with a lambda calculus $\Lambda^{\downarrow\uparrow} \supseteq \Lambda$ extended by four new constants $\mathsf{Up}, \mathsf{up}, \mathsf{down}, \mathsf{Down}$. On $\Lambda^{\downarrow\uparrow}$ we consider $\beta\iota$-reduction $\longrightarrow$, given as the reflexive-transitive compatible closure of the β-axiom $(\lambda x.t)\, s \longrightarrow t[s/x]$ and the ι-axioms listed in Figure 2.

In the three rules which create an abstraction, $x \notin \mathsf{FV}(A, C, t)$. Under this assumption, each of the ι-reductions preserves the set of free variables. Since there are no reductions for terms headed by constants $c \in \{\mathsf{Fun}, \mathsf{N}, \mathsf{z}, \mathsf{s}, \mathsf{U}, \mathsf{Up}\}$, these c are constructors, whereas the other constants $f \in \{\mathsf{rec}, \mathsf{up}, \mathsf{down}, \mathsf{Down}\}$ satisfy pattern matching equations. The left hand sides of all ι-rules are algebraic, linear, and non-overlapping. Reduction $\longrightarrow$ is a orthogonal *constructor combinatory reduction system* [20], hence, it is confluent. Thus, $\longrightarrow \circ \longleftarrow$ is an equivalence, which we denote by $=_{\beta\iota}$. Reduction is also standardizing; one can apply the call-by-name strategy to find the normal form, if it exists.

Let $\overline{r}$ denote the equivalence class of r modulo $=_{\beta\iota}$. The set of equivalence classes $\mathsf{D} = \Lambda^{\downarrow\uparrow}/{=_{\beta\iota}}$ forms a syntactical combinatory algebra with application operation $\overline{r} \cdot \overline{s} = \overline{r\, s}$ and denotation operation $[\![r]\!]_\rho = \overline{r[\sigma]}$ where σ is arbitrary with $\overline{\sigma(x)} = \rho(x)$ for all $x \in \mathsf{Var}$. (D is even a syntactical λ-model [8].)

One easily checks that all equations assumed in the last section are satisfied by D. Since recursion is well-defined

by induction on $\mathcal{N}$ and $\uparrow^X, \downarrow^X, \Downarrow X$ are well-defined by induction on $X \in \mathcal{T}ype$, the implementation of these function by rewriting is terminating on arguments which inhabit the PERs of our model. Also, for $X \in \mathcal{T}ype$, the free variables of the normal form of $\mathsf{up}\, X\, u$ for $u \in \mathsf{Ne}$ are exactly the free variables of u, the free variables of the normal form of $\mathsf{down}\, X\, v$ for $v \in X$ normal are exactly the free variables of v, and for $X \in \mathcal{T}ype$ normal the free variables of the normal form of $\mathsf{Down}\, X$ are exactly the free variables of X. This simplifies the task of finding fresh variables during reflection and reification, since suitable candidates can be fixed in the beginning.

$$
\begin{array}{ll}
\mathsf{rec}\, C\, z\, s\, \mathsf{z} & \longrightarrow z \\
\mathsf{rec}\, C\, z\, s\, (\mathsf{s}\, n) & \longrightarrow s\, n\, (\mathsf{rec}\, C\, s\, z\, n) \\
\mathsf{rec}\, C\, z\, s\, (\mathsf{Up}\, r) & \longrightarrow \mathsf{up}\, (C\, (\mathsf{Up}\, r)) \\
\quad (\mathsf{rec}\ \ (\mathsf{down}\, (\mathsf{N} \to \mathsf{U})\, C) & \\
\qquad (\mathsf{down}\, (C\, \mathsf{z})\, z) & \\
\qquad (\mathsf{down}\, (\mathsf{Fun}\, \mathsf{N}\, (\lambda n.\, C\, n \to C\, (\mathsf{s}\, n)))\, s) & \\
\qquad r) & \\
\mathsf{up}\, (\mathsf{Fun}\, A\, C)\, t & \longrightarrow \lambda x.\, \mathsf{up}\, (C\, x)\, (t\, (\mathsf{down}\, A\, x)) \\
\mathsf{up}\, \mathsf{N}\, t & \longrightarrow \mathsf{Up}\, t \\
\mathsf{up}\, (\mathsf{Up}\, r)\, t & \longrightarrow \mathsf{Up}\, t \\
\mathsf{up}\, \mathsf{U}\, t & \longrightarrow \mathsf{Up}\, t \\
\mathsf{down}\, (\mathsf{Fun}\, A\, C)\, t & \longrightarrow \lambda x.\, \mathsf{down}\, (C\, (\mathsf{up}\, A\, x)) \\
 & \qquad\qquad (t\, (\mathsf{up}\, A\, x)) \\
\mathsf{down}\, \mathsf{N}\, \mathsf{z} & \longrightarrow \mathsf{z} \\
\mathsf{down}\, \mathsf{N}\, (\mathsf{s}\, n) & \longrightarrow \mathsf{s}\, (\mathsf{down}\, \mathsf{N}\, n) \\
\mathsf{down}\, \mathsf{N}\, (\mathsf{Up}\, t) & \longrightarrow t \\
\mathsf{down}\, (\mathsf{Up}\, r)\, (\mathsf{Up}\, t) & \longrightarrow t \\
\mathsf{down}\, \mathsf{U} & \longrightarrow \mathsf{Down} \\
\mathsf{Down}\, (\mathsf{Fun}\, A\, C) & \longrightarrow \mathsf{Fun}\, (\mathsf{Down}\, A) \\
 & \qquad\qquad (\lambda x.\, \mathsf{Down}\, (C\, (\mathsf{up}\, A\, x))) \\
\mathsf{Down}\, \mathsf{N} & \longrightarrow \mathsf{N} \\
\mathsf{Down}\, (\mathsf{Up}\, t) & \longrightarrow t \\
\mathsf{Down}\, \mathsf{U} & \longrightarrow \mathsf{U}
\end{array}
$$

Figure 2. ι-reduction in $\Lambda^{\downarrow\uparrow}$.

Our solution to freshness is similar to the work of Danvy [14] and Aehlig and Joachimski [5] who introduce a second-level lambda-calculus to model computation in D. However, our approach is simpler, we do not need two levels, so we can apply standard results of λ-calculus. Furthermore, in our case recursion, reflection, and reification are always terminating, which is not the case in untyped NbE.

4 Logical Relations and Soundness of NbE

In this section we show soundness of NbE for $\lambda^{\Pi\mathsf{UN}}$, that is, that a term (or type) is provably equal to its normal form as computed by the nbe-algorithm: $\Gamma \vdash a = \Downarrow[\![a]\!] : A$. Decidability of judgmental equality is then a direct corollary of soundness and completeness.

Let $\mathsf{Ty}(\Gamma) = \{C \mid \Gamma \vdash C\}$ be the set of well-formed types in context Γ and $\mathsf{Tm}(\Gamma, C) = \{t \mid \Gamma \vdash t : C\}$ be the set of terms of type C in Γ. We say that Δ *extends* a well-formed context Γ, written $\Delta \geq \Gamma$, if $\Delta \vdash$ and $\Gamma(x) = \Delta(x)$ for all $x \in \mathsf{dom}(\Gamma)$.

By induction on $X \in \mathcal{T}ype$, we simultaneously define relations

$$
\begin{array}{l}
_ \vdash _ \circledR X \\
_ \vdash _ : _ \circledR _ \in X
\end{array}
$$

such that $(\Gamma \vdash _ \circledR X) \subseteq \mathsf{Ty}(\Gamma)$ and $(\Gamma \vdash _ : C \circledR _ \in X) \subseteq \mathsf{Tm}(\Gamma, C) \times [X]$. We always assume that Γ is well-formed.

$$
\begin{array}{l}
\Gamma \vdash C \circledR \mathsf{Fun}\, X\, F :\Longleftrightarrow \\
\quad \Gamma \vdash C = \mathsf{Fun}\, A\, (\lambda x.B) \text{ for some } A, B \text{ and} \\
\quad \Gamma \vdash A \circledR X \text{ and} \\
\quad \Delta \vdash B[s/x] \circledR F\, d \text{ for all } \Delta \geq \Gamma \\
\qquad \text{and } \Delta \vdash s : A \circledR d \in X
\end{array}
$$

$$
\Gamma \vdash C \circledR X :\Longleftrightarrow \Gamma \vdash C = \Downarrow X \text{ for } X \neq \mathsf{Fun}\, Y\, F
$$

$$
\begin{array}{l}
\Gamma \vdash r : C \circledR f \in \mathsf{Fun} X F :\Longleftrightarrow \\
\quad \Gamma \vdash C = \mathsf{Fun}\, A\, (\lambda x.B) \text{ for some } A, B \text{ and} \\
\quad \Gamma \vdash A \circledR X \text{ and} \\
\quad \Delta \vdash r\, s : B[s/x] \circledR f\, d \in F\, d \text{ for all } \Delta \geq \Gamma \\
\qquad \text{and } \Delta \vdash s : A \circledR d \in X
\end{array}
$$

$$
\Gamma \vdash A : C \circledR X \in \mathsf{U} :\Longleftrightarrow \Gamma \vdash C = \mathsf{U} \text{ and } \Gamma \vdash A \circledR X
$$

$$
\begin{array}{l}
\Gamma \vdash r : C \circledR d \in X :\Longleftrightarrow \Gamma \vdash r = \downarrow^X d : C \\
\quad \text{for } X = \mathsf{N}, \mathsf{Up}\, u
\end{array}
$$

Note that these definitions do not depend on the choice of the representative X, the relations are invariant under replacement of X by X' if $X = X' \in \mathcal{T}ype$.

Lemma 9 *Let $X = X' \in \mathcal{T}ype$.*

1. *If $\Gamma \vdash C \circledR X$ then $\Gamma \vdash C \circledR X'$.*
2. *If $\Gamma \vdash t : C \circledR d \in X$ and $d = d' \in X$ then $\Gamma \vdash t : C \circledR d' \in X'$.*

Simultaneously with the definition one proves the following lemma.

Lemma 10 (Properties of the logical relations) *Let $X \in \mathcal{T}ype$, $\Gamma \vdash C \circledR X$, and $\Gamma \vdash t : C \circledR d \in X$.*

1. *(Monotonicity:) If $\Delta \geq \Gamma$ then $\Delta \vdash C \circledR X$ and $\Delta \vdash t : C \circledR d \in X$.*

2. *(In:) If* $\Gamma \vdash r = u : C$ *then* $\Gamma \vdash r : C \circledR \uparrow^X u \in X$.

3. *(Out:)* $\Gamma \vdash C = \Downarrow X$ *and* $\Gamma \vdash t = \downarrow^X d : C$.

A consequence of *Out* is that the choice of A, B in the defining clauses of the relations for $\mathsf{Fun}\, X\, F$ does not matter, since A is determined by X up to judgmental equality uniquely, and B by F. Hence, the following lemma is easily proved by induction on $X \in \mathcal{T}ype$:

Lemma 11 *Let* $X \in \mathcal{T}ype$ *and* $\Gamma \vdash C = C'$.

1. *If* $\Gamma \vdash C \circledR X$ *then* $\Gamma \vdash C' \circledR X$.

2. *If* $\Gamma \vdash t : C \circledR d \in X$ *and* $\Gamma \vdash t = t' : C$ *then* $\Gamma \vdash t' : C' \circledR d \in X$.

We relate substitutions $\Delta \vdash \sigma : \Gamma$ to environments $\rho \in \Gamma$ by the following definition:

$$\begin{array}{l} \Delta \vdash \sigma : \Gamma \circledR \rho :\Longleftrightarrow \text{for all } x \in \mathsf{dom}(\Gamma), \\ \quad \Delta \vdash \sigma(x) : \Gamma(x)[\sigma] \circledR \rho(x) \in [\![\Gamma(x)]\!]_\rho \end{array}$$

We define the propositions $\Gamma \Vdash J$ as follows:

$$\begin{array}{lcl} \Vdash & :\Longleftrightarrow & \text{true} \\ \Gamma, x{:}A \Vdash & :\Longleftrightarrow & \Gamma \Vdash A \\ \Gamma \Vdash A & :\Longleftrightarrow & \Gamma \Vdash A = A \\ \Gamma \Vdash A = A' & :\Longleftrightarrow & \Gamma \Vdash \text{ and} \\ & & \quad \Delta \vdash A[\sigma] \circledR [\![A']\!]_\rho \\ & & \quad \text{for all } \Delta \vdash \sigma : \Gamma \circledR \rho. \\ \Gamma \Vdash t : A & :\Longleftrightarrow & \Gamma \Vdash t = t : A \\ \Gamma \Vdash t = t' : A & :\Longleftrightarrow & \Gamma \Vdash A \text{ and} \\ & & \quad \Delta \vdash t[\sigma] : A[\sigma] \circledR [\![t']\!]_\rho \in [\![A]\!]_\rho \\ & & \quad \text{for all } \Delta \vdash \sigma : \Gamma \circledR \rho. \\ \Gamma \Vdash \tau : \Gamma' & :\Longleftrightarrow & \Gamma \Vdash \tau = \tau : \Gamma' \\ \Gamma \Vdash \tau = \tau' : \Gamma' & :\Longleftrightarrow & \Gamma \Vdash \text{ and } \Gamma' \Vdash \text{ and} \\ & & \quad \Delta \vdash \tau[\sigma] : \Gamma' \circledR [\![\tau']\!]_\rho \\ & & \quad \text{for all } \Delta \vdash \sigma : \Gamma \circledR \rho \end{array}$$

Theorem 12 (Fundamental theorem of logical relations) *If* $\Gamma \vdash J$ *then* $\Gamma \Vdash J$.

Proof. By induction on $\Gamma \vdash J$. □

We define a special valuation ρ_Γ by induction on Γ, where $\rho_\diamond(x)$ is arbitrary and $\rho_{(\Gamma, x:A)} = \rho_\Gamma[x \mapsto \uparrow^{[\![A]\!]_{\rho_\Gamma}} x]$. It follows that $\rho_\Gamma \in \Gamma$ and $\Gamma \vdash \sigma_{\mathsf{id}} : \Gamma \circledR \rho_\Gamma$.

Corollary 13 (Soundness of NbE)

1. *If* $\Gamma \vdash t : A$ *then* $\Gamma \vdash t = \downarrow^{[\![A]\!]_{\rho_\Gamma}} [\![t]\!]_{\rho_\Gamma} : A$.

2. *If* $\Gamma \vdash A$ *then* $\Gamma \vdash A = \Downarrow [\![A]\!]_{\rho_\Gamma}$.

Corollary 14 (Decidability of equality)

1. *If* $\Gamma \vdash t, t' : A$ *we can decide whether* $\Gamma \vdash t = t' : A$.

2. *If* $\Gamma \vdash A, A'$ *we can decide whether* $\Gamma \vdash A = A'$.

Proof. It follows from soundness and completeness that two terms or types are equal (in the sense of judgmental equality) iff their normal forms, as computed by the nbe-function, are equal. □

Corollary 15 (Injectivity of Π) *If* $\Gamma \vdash \mathsf{Fun}\, A\, (\lambda x.B) = \mathsf{Fun}\, A'\, (\lambda x.B')$ *then* $\Gamma \vdash A = A'$ *and* $\Gamma, x{:}A \vdash B = B'$.

Proof. Let $\rho = \rho_\Gamma$. By Thm. 6, $[\![\mathsf{Fun}\, A\, (\lambda x.B)]\!]_\rho = [\![\mathsf{Fun}\, A'\, (\lambda x.B')]\!]_\rho \in \mathcal{T}ype$. First, this implies $[\![A]\!]_\rho = [\![A']\!]_\rho \in \mathcal{T}ype$ by inversion on $\mathcal{T}ype$, hence, $\Downarrow [\![A]\!]_\rho$ and $\Downarrow [\![A']\!]_\rho$ are syntactically identical. With soundness of NbE, this implies $\Gamma \vdash A = A'$. Secondly, since $d := \uparrow^{[\![A]\!]_\rho} x \in [\![A]\!]_\rho$, we get $[\![\lambda x.B]\!]_\rho\, d = [\![\lambda x.B']\!]_\rho\, d \in \mathcal{T}ype$. With $\rho' = \rho[x \mapsto \uparrow^{[\![A]\!]_\rho} x] = \rho_{(\Gamma, x:A)}$ this implies that $[\![B]\!]_{\rho'} = [\![B']\!]_{\rho'} \in \mathcal{T}ype$, so with soundness we get $\Gamma, x : A \vdash B = B'$. □

5 Extensions

Since we work with judgmental equality it is simple to extend our method to a system with a unit type 1 with inhabitant $\Gamma \vdash () : 1$ and η-law $\Gamma \vdash t = () : 1$ for $\Gamma \vdash t : 1$. In the PER model, we simply let $d = d' \in [1]$ for all d, d'. Reification $\downarrow^1 d = ()$ and reflection $\uparrow^1 u = ()$ return both the constant $()$, so the result of nbe never depends on a particular inhabitant t of the unit type.

Similarly, we can extend our method to deal with Σ-types and singleton types, constructions which present problems in systems with untyped conversion. Such types are handled by Harper and Pfenning's [18] type-directed algorithmic equality. However, their method is based on erasure of type dependencies and fails for systems which also include universes like $\lambda^{\Pi\mathsf{UN}}$.

Finally, our approach is robust to changes of syntax. For instance, it can deal with typed abstraction $\lambda x : A.t$. In the PER model, which is based on the untyped λ-calculus, A is simply ignored. The reification function $\downarrow^{\mathsf{Fun}\, X\, F} t$ then returns an abstraction annotated with type $\Downarrow X$. We could also adapt it to the more explicit syntax for abstraction and application suggested in [31]. Yet another possibility is to use our method for proving the basic properties of the initial category with families (cwf) [15, 19], which is a more algebraic way to present type theory.

We can now prove the correctness of a standard bidirectional type checking algorithm [18] for normal terms. This proof is a corollary of decidability of judgmental

equality together with the syntactical inversion properties (Lemma 2). Such algorithms are currently used in the core languages of the dependently typed languages Agda [13] developed at Chalmers and Epigram [25] developed at Nottingham University. With typed abstraction, type checking is decidable also for non-normal terms.

References

[1] A. Abel, K. Aehlig, and P. Dybjer. Normalization by evaluation for Martin-Löf type theory with one universe. In *23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS XXIII*, Electronic Notes in Theoretical Computer Science. Elsevier, 2007. To appear.

[2] A. Abel and T. Coquand. Untyped algorithmic equality for Martin-Löf's logical framework with surjective pairs. *Fundamenta Informaticæ*, 2007. TLCA'05 special issue. To appear.

[3] P. Aczel. *Frege Structures and the Notions of Proposition, Truth, and Set*, pages 31–59. North-Holland, 1980.

[4] R. Adams. Pure type systems with judgemental equality. *Journal of Functional Programming*, 16(2):219–246, 2006.

[5] K. Aehlig and F. Joachimski. Operational aspects of untyped normalization by evaluation. *Mathematical Structures in Computer Science*, 14(4):587–611, Aug. 2004.

[6] S. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language*. PhD thesis, Department of Computer Science, Cornell University, 1987.

[7] T. Altenkirch. *Constructions, Inductive Types and Strong Normalization*. PhD thesis, University of Edinburgh, Nov. 1993.

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

[9] M. Beeson. *Foundations of constructive mathematics. Metamathematical studies*, volume 6 of *Ergebnisse der Mathematik und ihrer Grenzgebiete, 3. Folge [A Series of Modern Surveys in Mathematics]*. Springer-Verlag, 1985.

[10] U. Berger and H. Schwichtenberg. An inverse to the evaluation functional for typed λ-calculus. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science, Amsterdam*, pages 203–211, July 1991.

[11] R. Constable and team. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

[12] T. Coquand. An algorithm for testing conversion in type theory. In *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[13] T. Coquand. An algorithm for type-checking dependent types. In *Mathematics of Program Construction. Selected Papers from the Third International Conference on the Mathematics of Program Construction (July 17–21, 1995, Kloster Irsee, Germany)*, volume 26 of *Science of Computer Programming*, pages 167–177. Elsevier Science, May 1996.

[14] O. Danvy. Type-directed partial evaluation. In J. Hatcliff, T. Æ. Mogensen, and P. Thiemann, editors, *Partial Evaluation – Practice and Theory, DIKU 1998 International Summer School, Copenhagen, Denmark, June 29 - July 10, 1998*, volume 1706 of *Lecture Notes in Computer Science*, pages 367–411. Springer-Verlag, 1999.

[15] P. Dybjer. Internal type theory. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, number 1158 in Lecture Notes in Computer Science, pages 120–134. Springer-Verlag, 1996.

[16] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, (2):525–549, June 2000.

[17] H. Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, The University of Edinburgh, Department of Computer Science, 1994.

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

[19] M. Hofmann. Syntax and semantics of dependent types. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*. Cambridge University Press, 1996.

[20] J. W. Klop. Combinatory reduction systems. *Mathematical Center Tracts*, 27, 1980.

[21] P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.

[22] P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, pages 153–175. North-Holland, 1982.

[23] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[24] P. Martin-Löf. Normalization by evaluation and by the method of computability. Talk at JAIST, Japan Advanced Institute of Science and Technology, Kanazawa, June 2004.

[25] C. McBride and J. McKinna. The view from the left. *Journal of Functional Programming*, 2004.

[26] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin Löf's Type Theory: An Introduction*. Clarendon Press, Oxford, 1990.

[27] A. M. Pitts. Alpha-structural recursion and induction. In J. Hurd and T. F. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3603 of *Lecture Notes in Computer Science*, pages 17–34. Springer-Verlag, 2005.

[28] D. S. Scott. Combinators and classes. In C. Böhm, editor, *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, March 25-27, 1975*, volume 37 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag, 1975.

[29] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342(1):28–55, 2005.

[30] C. A. Stone and R. Harper. Extensional equivalence and singleton types. *ACM Transactions on Computational Logic*, 7(4):676–722, 2006.

[31] T. Streicher. *Semantics of Type Theory*. Progress in Theoretical Computer Science. Birkhaeuser Verlag, Basel, 1991.