Normalization by Evaluation for Martin-Löf Type Theory with One Universe

Andreas Abel[1,2]

Institut für Informatik, Ludwig-Maximilians-Universität
Oettingenstr. 67, D-80538 München

Klaus Aehlig[3,4]

Department of Computer Science, University of Wales Swansea
Singleton Park, Swansea SA2 8PP

Peter Dybjer[5,6]

Department of Computer Science, Chalmers University of Technology
Rännvägen 6, S-41296 Göteborg

Abstract

We present an algorithm for computing normal terms and types in Martin-Löf type theory with one universe and eta-conversion. We prove that two terms or types are equal in the theory iff the normal forms are identical (as de Bruijn terms). It thus follows that our algorithm can be used for deciding equality in Martin-Löf type theory. The algorithm uses the technique of normalization by evaluation; normal forms are computed by first evaluating terms and types in a suitable model. The normal forms are then extracted from the semantic elements. We prove its completeness by a PER model and its soundness by a Kripke logical relation.

Keywords: Dependent Types, Domain Semantics, Normalization by Evaluation, Type Theory, Universe

[1] Partially supported by the EU coordination action *TYPES* (510996).

[2] Email: `abel@tcs.ifi.lmu.de`

[3] Partially supported by grant EP/D03809X/1 of the British Engineering and Physical Sciences Research Council (EPSRC). Currently visiting University of Toronto, supported by grant Ae 102/1-1 of the Deutsche Forschungsgemeinschaft (DFG).

[4] Email: `k.t.aehlig@swan.ac.uk`

[5] Partially supported by project TLCA, Vetenskapsrådet.

[6] Email: `peterd@cs.chalmers.se`

1 Introduction

Normalization by Evaluation (NbE) is a method for computing normal forms of λ-terms by first interpreting them in some semantic realm and then *reifying them*, i. e., bringing them "down" to the syntactic level, arriving at a normal form. We exploit this method for Martin-Löf type theory with one universe [19], a theory where types can depend on values. Such dependent types are not only restrictions of larger non-dependent types (as the types of the logical framework or refinement types). In Martin-Löf type theory with a universe a type can be defined by *recursion* on a value of some other type. (This is sometimes called definition by *large* elimination.) Such dependencies cannot be erased, so that values with such dependent types cannot always be assigned simple types.

Large eliminations have repercussions on the design of a NbE algorithm. Firstly, types need to be normalized as well as terms. Furthermore, in Martin-Löf type theory well-typed terms denote "total" elements, such as total natural numbers, total functions between natural numbers, etc. It is thus tempting to consider a semantics of total elements for the interpretation of terms in NbE, as in the simply-typed case. This, however, leads to great complications in the case of dependent types, where typing derivations depend on proofs of equality, a scenario one could call the "dependent types nightmare". Instead we were able to prove the correctness of our NbE algorithm by choosing a different approach: inspired by untyped NbE [4,13] we evaluate terms in a reflexive domain and ignore their types. We then define a PER model, to pick out (equal) total elements of this domain and show that well-typed term denote such total elements and $\beta\eta$-convertible terms denote equal total elements. The model construction is surprisingly painless, since it is sufficient to recover raw terms from the domain and not typing derivations (Church terms, resp.), i. e., we are not interested in whether the reified term is well-typed.

Our algorithm returns η-long forms, which can only be correctly produced when the type of a term is normalized before the term. We use normal types to reflect variables into the semantics and to reify semantic objects to η-long forms. During reflection we η-expand variables semantically, and we maintain the invariant that variables are always fully applied in the semantics.

Contributions.

We present a normalization algorithm for Martin-Löf type theory with one universe [19] and η-conversion. We prove that this algorithm returns unique representatives from each convertibility class and hence can be used for deciding convertibility of terms and of types. This is a new result; the decidability property for the theory with β-conversion but *without* η follows from a standard reduction-based normalization proof by Martin-Löf [19] (see also C. Coquand [8]).

A side-effect of our paper is that we provide the first account of NbE for the typed lambda calculus with $\beta\eta$-conversion and inductive datatypes such as natural numbers, where correctness is proved directly without relying on normalizability of the reduction relation.

As pointed out to us by Thierry Coquand, our result is important for proof assistants based on Martin-Löf type theory such as the Agda system [7]. The reason is that the core of Agda is a version of Martin-Löf's *logical framework* which has η-conversion both on the level of *types* and on the level of the universe of *sets*. Our result provides the key step in the proof of decidability of type-checking of this theory including a set of natural numbers. It appears unproblematic to extend our proof to other sets (*data types*) given by (generalized) inductive definitions such as Brouwer ordinals and well-orderings used in applications of Martin-Löf type theory.

Related work.

Normalization by evaluation for Martin-Löf type theory has to our knowledge only been considered once before in the literature, namely in the first published paper on Martin-Löf type theory [15]. However, this version of the theory had a weak notion of conversion (no conversion under λ) and was later abandoned.

Recently, Martin-Löf has in unpublished work adapted this normalization algorithm to the present version of type theory [20]. The theory considered by Martin-Löf [20] differs from ours since it has only η-conversion on the level of types and not on the level of sets. [18,21]. (Note that the type of "sets" in this theory plays the rule of the universe in [19]. The reason for this change in terminology is that conceptually, the logical framework in this theory is intended as a metalogic for formalizing the rules of Martin-Löf type theory.) Another important difference is that we use an approach to NbE which relies on evaluation of untyped terms in a semantic domain with partial values. Martin-Löf's instead uses an informal typed intuitionistic metalanguage (see for example the discussion in [14]) which is presumably some strong version of Martin-Löf type theory with inductive-recursive definitions. The third author has in vain tried to formalize a strongly typed version of NbE for Martin-Löf type theory in a proof assistant for Martin-Löf type theory. However, Danielsson [9] has recently made progress in this direction.

Berger, Eberl, and Schwichtenberg [6] describe how to construct an NbE algorithm from a set of computation rules formulated as a term rewriting system (TRS), thus, also covering primitive recursion for natural numbers. However, they assume the TRS to be terminating and consider only simple types.

Filinski [12] has earlier used domain-theoretic models for NbE. In particular Filinski and Rohde [13] gave a domain-theoretic treatment of untyped NbE.

2 Syntax and Inference Rules

We first present the syntax and inference rules of Martin-Löf type theory with one universe. As already mentioned, an essential point is that we extend the theory as presented by Martin-Löf [19] by the rule of η-conversion. In this paper we show only the rules for dependent function types ("cartesian product of a family of types"), the type of natural numbers, and the type of small types, but we believe that our method can be extended to all type formers considered by Martin-Löf [19], that is, also for dependent product types ("disjoint union of a family of types"), binary disjoint unions, and finite types. It also appears unproblematic to include generalized inductive definitions such as the type of Brouwer ordinals [17] and the well-orderings [16].

As in Martin-Löf [19], we consider a formulation where conversion is a relation between raw terms; we do not have *equality judgements* as in the later versions of Martin-Löf type theory. Also like in Martin-Löf [19], our universe is formulated a la Russell, where small types are types. (When universes are formulated a la Tarski as in Aczel [3] and Martin-Löf [17], elements of universes are *codes* for small types and each such code a denotes a small type $T\ a$. We have also written the algorithm for the system a la Tarski, but the a la Russell version is shorter.)

We use de Bruijn's nameless representation of lambda terms, whereas Martin-Löf used ordinary named variables. Small types are called "sets" to conform with the current usage in Agda. So the first universe which was called U by Martin-Löf is now called Set.

Raw terms.

We begin by defining the set Tm of raw de Bruijn terms of the theory, see Fig. 1. Since universes are formulated a la Russell, type expressions are just special kinds of raw terms. We use the letters r, s, t, z, A, B, C as metavariables for raw terms, where A, B, C are used when we expect that the raw terms are type expressions. Syntactic equality of terms is denoted by $t \equiv t'$. Binders are λ and Π; λt binds index 0 in t and $\Pi\, A\, B$ binds index 0 in B. Based on this binding convention we define the set of free de Bruijn indices $\mathsf{FV}(t)$ for a term t in the obvious way; in particular we have the following clauses.

$$\begin{array}{lcl}
i \in \mathsf{FV}(v_j) & \iff & i = j \\
i \in \mathsf{FV}(\lambda t) & \iff & i+1 \in \mathsf{FV}(t) \\
i \in \mathsf{FV}(\Pi\, A\, B) & \iff & i \in \mathsf{FV}(A) \text{ or } i+1 \in \mathsf{FV}(B) \\
i \in \mathsf{FV}(\mathit{Rec}\, A\, z\, s\, t) & \iff & i \in \mathsf{FV}(A) \cup \mathsf{FV}(z) \cup \mathsf{FV}(s) \cup \mathsf{FV}(t)
\end{array}$$

We denote the operation of *lifting* the free de Bruijn indices in a term t by $k \in \mathbb{N}$ steps by $\Uparrow^k$, where $\Uparrow^k = \Uparrow^k_0$, for an auxiliary operation $\Uparrow^k_n t$ with $n \in \mathbb{N}$ that is defined by induction on t and lifts the free variables from index n onwards by k; in particular

$$\begin{array}{lcl}
\Uparrow^k_n v_i & = & \begin{cases} v_i & \text{if } i < n \\ v_{i+k} & \text{otherwise} \end{cases} \\
\Uparrow^k_n \lambda t & = & \lambda(\Uparrow^k_{n+1} t) \\
\Uparrow^k_n (\Pi\, A\, B) & = & \Pi (\Uparrow^k_n A)\, (\Uparrow^k_{n+1} B)
\end{array}$$

We define the non-dependent function space $A \Rightarrow B$ as an abbreviation for $\Pi\, A\, (\Uparrow^1 B)$.

Let $t[s/i]$ denote the collapsing substitution of s for index i in t, that is

$$v_j[s/i] = \begin{cases} v_j & j < i \\ s & j = i \\ v_{j-1} & j > i \end{cases}$$

and $(\lambda t)[s/i] = \lambda(t[\Uparrow^1 s/i+1])$, as usual. We write $t[s]$ as a shorthand for $t[s/0]$. One-step $\beta\eta$-reduction $t \longrightarrow t'$ is given as the congruence-closure of the following

Raw terms with de Bruijn indices.

$$\begin{array}{rcll} Tm \ni r,s,t,z,A,B,C & ::= & v_i & \text{de Bruijn index} \\ & \mid & \lambda t & \text{abstracting 0th variable} \\ & \mid & r\,s & \text{application} \\ & \mid & \mathit{Zero} & \text{natural number ``0''} \\ & \mid & \mathit{Succ}\,t & \text{successor} \\ & \mid & \mathit{Rec}\,A\,z\,s\,t & \text{primitive recursion} \\ & \mid & \Pi\,A\,B & \text{dependent function type} \\ & \mid & \mathit{Nat} & \text{natural number type} \\ & \mid & \mathit{Set} & \text{universe} \end{array}$$

Well-formed contexts $\Gamma \vdash$.

$$\frac{}{\diamond \vdash} \qquad \frac{\Gamma \vdash A}{\Gamma, A \vdash}$$

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

$$\frac{\Gamma \vdash A : \mathit{Set}}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathit{Set}} \qquad \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash \Pi\,A\,B}$$

Typing $\Gamma \vdash t : A$.

$$\frac{\Gamma \vdash}{\Gamma \vdash v_i : \Gamma(i)}\ 0 \leq i < |\Gamma|$$

$$\frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda t : \Pi\,A\,B} \qquad \frac{\Gamma \vdash r : \Pi\,A\,B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\,s : B[s]} \qquad \frac{\Gamma \vdash A : \mathit{Set} \qquad \Gamma, A \vdash B : \mathit{Set}}{\Gamma \vdash \Pi\,A\,B : \mathit{Set}}$$

$$\frac{}{\Gamma \vdash \mathit{Nat} : \mathit{Set}} \qquad \frac{}{\Gamma \vdash \mathit{Zero} : \mathit{Nat}} \qquad \frac{\Gamma \vdash t : \mathit{Nat}}{\Gamma \vdash \mathit{Succ}\,t : \mathit{Nat}}$$

$$\frac{\begin{array}{c}\Gamma, \mathit{Nat} \vdash C \\ \Gamma \vdash z : C[\mathit{Zero}] \qquad \Gamma \vdash s : \Pi\,\mathit{Nat}\,(C \Rightarrow C[\mathit{Succ}\,v_0]) \qquad \Gamma \vdash t : \mathit{Nat}\end{array}}{\Gamma \vdash \mathit{Rec}\,(\lambda C)\,z\,s\,t : C[t]}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A'}{\Gamma \vdash t : A'}\ A =_{\beta\eta} A'$$

Fig. 1. Terms and inference rules.

contractions.

$$\begin{array}{lll} (\lambda t)\,s & \longrightarrow t[s] & (\beta\text{-}\lambda) \\ \lambda.(\Uparrow^1 t)\,v_0 & \longrightarrow t & (\eta) \\ \mathit{Rec}\,A\,z\,s\,\mathit{Zero} & \longrightarrow z & (\beta\text{-}\mathit{Rec}\text{-}\mathit{Zero}) \\ \mathit{Rec}\,A\,z\,s\,(\mathit{Succ}\,r) & \longrightarrow s\,r\,(\mathit{Rec}\,A\,z\,s\,r) & (\beta\text{-}\mathit{Rec}\text{-}\mathit{Succ}) \end{array}$$

Its reflexive-transitive closure $\longrightarrow^*$ is confluent, so we can define $t =_{\beta\eta} t'$ as $\exists s.t \longrightarrow^* s \;{}^*\!\!\longleftarrow t'$.

Typing contexts are lists of types, inductively defined by $\Gamma ::= \diamond \mid \Gamma, A$. Context lookup $\Gamma(n)$ performs the necessary liftings:

$$\begin{array}{ll} (\Gamma, A)(0) & = \Uparrow^1 A \\ (\Gamma, A)(n+1) & = \Uparrow^1 \Gamma(n) \end{array}$$

Inference rules.

We define the inference rules for the following three forms of judgements.

$$\begin{array}{ll} \Gamma \vdash & \Gamma \text{ is a well-formed context} \\ \Gamma \vdash A & A \text{ is a well-formed type in context } \Gamma \\ \Gamma \vdash t : A & t \text{ has type } A \text{ in context } \Gamma \end{array}$$

The rules are listed in Fig. 1. The judgements enjoy standard properties like weakening, strengthening and substitution, however, we require *no* syntactical properties of these judgements in this work.

3 Domain Model

In this section, we present the NbE algorithm by defining a suitable semantic domain D into which terms are evaluated, before they are brought back *down* onto the syntactical level.

Let D be a set, and let environments ρ range over $\mathsf{Env} := \mathbb{N} \to \mathsf{D}$. We define *environment update* ρ, d as the environment ρ' such that $\rho'(0) = d$ and $\rho'(i+1) = \rho(i)$. Furthermore, let $[\![\_]\!]_\_ \in Tm \to \mathsf{Env} \to \mathsf{D}$ an evaluation function, $\_ \cdot \_ \in \mathsf{D} \to \mathsf{D} \to \mathsf{D}$ an application function, and $\mathsf{rec} \in \mathsf{D} \to \mathsf{D} \to \mathsf{D} \to \mathsf{D} \to \mathsf{D}$ a primitive recursion operator. Adopting Barendregt's notion [5] to our setting, we say that $(\mathsf{D}, [\![\_]\!]_\_, \_ \cdot \_, \mathsf{rec})$ is a *weakly extensional λ-model*, if the following holds.

$$\begin{array}{lll} [\![v_i]\!]_\rho & = \rho(i) & \\ [\![r\, s]\!]_\rho & = [\![r]\!]_\rho \cdot [\![s]\!]_\rho & \\ [\![Rec\, C\, z\, s\, t]\!]_\rho & = \mathsf{rec}\ [\![C]\!]_\rho\, [\![z]\!]_\rho\, [\![s]\!]_\rho\, [\![t]\!]_\rho & \\[1ex] [\![\lambda t]\!]_\rho \cdot d & = [\![t]\!]_{\rho,d} & \\ \mathsf{rec}\ a\, d_z\, d_s\, [\![Zero]\!]_\rho & = d_z & \\ \mathsf{rec}\ a\, d_z\, d_s\, [\![Succ\, t]\!]_\rho & = d_s \cdot [\![t]\!]_\rho \cdot (\mathsf{rec}\ a\, d_z\, d_s\, [\![t]\!]_\rho) & \\[1ex] [\![c]\!]_\rho & = [\![c]\!]_{\rho'} & \text{for } c \in \{Zero, Nat, Set\} \\ [\![\lambda t]\!]_\rho & = [\![\lambda t']\!]_{\rho'} & \text{if } [\![t]\!]_{\rho,d} = [\![t']\!]_{\rho',d} \text{ for all } d \in \mathsf{D} \\ [\![\Pi A\, B]\!]_\rho & = [\![\Pi A'\, B']\!]_{\rho'} & \text{if } [\![A]\!]_\rho = [\![A']\!]_{\rho'} \\ & & \text{and } [\![B]\!]_{\rho,d} = [\![B']\!]_{\rho',d} \text{ for all } d \in \mathsf{D} \\ [\![Succ\, t]\!]_\rho & = [\![Succ\, t']\!]_{\rho'} & \text{if } [\![t]\!]_\rho = [\![t']\!]_{\rho'} \end{array}$$

Lemma 3.1 (Properties of weakly extensional λ-models [5]) *A weakly extensional λ-model* $(\mathsf{D}, [\![\_]\!]_\_, \_ \cdot \_, \mathsf{rec})$ *has the following properties.*

(i) *(Irrelevance) If* $\rho(i) = \rho'(i)$ *for all* $i \in \mathsf{FV}(t)$, *then* $[\![t]\!]_\rho = [\![t]\!]_{\rho'}$.

(ii) *(Substitution)* $[\![t[s]]\!]_\rho = [\![t]\!]_{\rho,[\![s]\!]_\rho}$.

(iii) *(β-Invariance) If* $t =_\beta t'$ *then* $[\![t]\!]_\rho = [\![t']\!]_\rho$.

Liftable terms.

Following Aehlig and Joachimski [4] we delay the lifting operation, to avoid the need of liftings in semantical objects. Morally, a liftable term is nothing but a function that maps k to the way this term would look like under k binders; usually this is just the term lifted by k. However, to allow for bound variables to occur we have to accept partiality; a term containing a variable bound by the ℓ'th binder can only present itself under at least ℓ binders — a bound variable can never occur outside the scope of its binder.

Formally, we define $Tm_{\mathbb{Z}}$ to be as our raw terms, but allowing also negative de Bruijn indices and we define the set of *liftable terms* $TM = \mathbb{N} \to Tm_{\mathbb{Z}}$ as the total functions from the naturals to $Tm_{\mathbb{Z}}$. For $\hat{t}, \hat{t}' \in TM$ we overload application by setting $(\hat{t}\,\hat{t}')(k) = \hat{t}(k)\,\hat{t}'(k)$; similarly for Rec. Equality $\hat{t} \equiv \hat{t}'$ is point-wise. We denote the liftable term $k \mapsto \Uparrow^k t$ simply as $\Uparrow t$. The special liftable term

$$\hat{v}_{-(k+1)}(l) = v_{l-(k+1)}$$

where $k \in \mathbb{N}$, is sometimes denoted by $\Uparrow v_{-(k+1)}$.

We define the semantic domain D with information order $\sqsubseteq$ as the least solution of the domain equation

$$\mathsf{D} = [\mathsf{D} \to \mathsf{D}] \oplus \mathbf{O} \oplus \mathsf{D} \oplus (\mathsf{D} \times [\mathsf{D} \to \mathsf{D}]) \oplus \mathbf{O} \oplus \mathbf{O} \oplus TM_\perp$$

We work in a suitable category of domains such as the category of Scott domains [22] (consistently complete pointed cpos and continuous functions), where $\mathbf{O}$ means the two point domain $\{\perp, \top\}$ with $\perp \sqsubseteq \top$ (the Sierpinski space), $\oplus$ means coalesced sum, $\times$ means cartesian product, $[\_ \to \_]$ means continuous function space, and $TM_\perp$ is the flat domain obtained by adjoining a least element $\perp$ to the set TM of liftable terms. We also extend application on TM to $TM_\perp$ so that $\hat{t}\,\perp = \perp\,\hat{t}' = \perp\,\perp = \perp$. If we write $\hat{t} \in TM_\perp$ then $\hat{t} \neq \perp$ always denotes a proper liftable term.

The role of the seven components of the RHS of the domain equation will be clear by introducing the following names of the strict injections (constructors):

$$\begin{array}{llll}
\mathsf{Lam} : [\mathsf{D} \to \mathsf{D}] \to \mathsf{D} & & \mathsf{Pi} : \mathsf{D} \times [\mathsf{D} \to \mathsf{D}] \to \mathsf{D} \\
\mathsf{Zero} : \mathbf{O} \to \mathsf{D} & & \mathsf{Nat} : \mathbf{O} \to \mathsf{D} \\
\mathsf{Succ} : \mathsf{D} \to \mathsf{D} & & \mathsf{Set} : \mathbf{O} \to \mathsf{D} \\
 & & \mathsf{Ne} : TM_\perp \to \mathsf{D}
\end{array}$$

Although formally, Pi has type $\mathsf{D} \times [\mathsf{D} \to \mathsf{D}] \to \mathsf{D}$, we write $\mathsf{Pi}\, a\, g$ instead of $\mathsf{Pi}\,(a, g)$. Moreover, although Zero has type $\mathbf{O} \to \mathsf{D}$ we write Zero instead of $\mathsf{Zero}\,\top$, and similarly for Nat and Set.

Elements of D are denoted by a, b, c (for types) and d, e for objects. Functions in $[\mathsf{D} \to \mathsf{D}]$ are denoted by f (object valued) and g (type valued). We overload the notation $a \Rightarrow b$ on D to mean $\mathsf{Pi}\, a\, (\_ \mapsto b)$.

The reason for having coalesced sums in the domain equation is that our proof of semantical η-conversion (Lemma 2 below) relies on the strictness of Lam, i. e., $\mathsf{Lam}\,\bot = \bot$.

If we replace the coalesced sums in the definition of D by separated sums, and replace $TM_\bot$ by the non-flat domain of lazy term families, we will get a domain equation which gives the intended domain semantics of the type D in our Haskell program (see the Appendix). However, there is an obvious embedding of the "strictified" domain used in the proof into the lazy domain used in the program. Using this embedding it follows that the normalization function in the proof must return an answer which is less defined than or equal to the normalization function computed by the Haskell program. Since we prove that the normalization function on the strictified domain returns correct total elements, it follows that the Haskell program also returns correct total elements.

We define application on D as the function

$$\begin{array}{lll} \mathsf{app} : [\mathsf{D} \to [\mathsf{D} \to \mathsf{D}]] & & \\ \mathsf{app}\,(\mathsf{Lam}\,f) & = f & \\ \mathsf{app}\,e & = \bot & \text{if } e \text{ is not a } \mathsf{Lam} \end{array}$$

where in the following "default $\bot$ clauses" like the last one are always tacitly assumed. Observe that $\mathsf{app}\,(\mathsf{Lam}\,f)\,d = f(d)$. We write $e \cdot d$ for $\mathsf{app}\,e\,d$.

Although we are working with a weakly extensional model D which only identifies β-equal terms, we can show that η-reduction is mapped to the order $\sqsubseteq$ on D.

Lemma 3.2 (Semantical η-contraction) $\mathsf{Lam}\,(\mathsf{app}\,e) \sqsubseteq e$.

Proof. By cases on e. If $e = \mathsf{Lam}\,f$, then $\mathsf{Lam}\,(\mathsf{app}\,e) = \mathsf{Lam}\,f = e$. Otherwise $\mathsf{Lam}\,(\mathsf{app}\,e) = \mathsf{Lam}\,\bot = \bot \sqsubseteq e$. $\square$

By mutual recursion, we define the reflection function $\uparrow^a$ from neutral term families into D and the reification function $\downarrow^a$ from D into normal term families as follows. Herein, we write $\downarrow^a_k d$ for $(\downarrow^a d)(k)$, and similarly $\Downarrow_k a$ for $(\Downarrow a)(k)$. The defining clauses are listed in Fig. 2, together with the semantical version of *Rec* and the evaluation function $[\![t]\!]_\rho$.

Note that to justify the types of the reification functions $\Downarrow$ and $\downarrow$ we must show that they return either $\bot$ or a totally defined function in TM. This is so because the argument $k : \mathbb{N}$ is only used in a parametric way, so definedness cannot depend on it.

Lemma 3.3 $(\mathsf{D}, [\![\_]\!]_\_, \_ \cdot \_, \mathsf{rec})$ *is a weakly extensional λ-model.*

Also, note that $[\![\Uparrow^1 t]\!]_{\rho,d} = [\![t]\!]_\rho$.

Lemma 3.4 *If $t \longrightarrow t'$ then $[\![t]\!]_\rho \sqsubseteq [\![t']\!]_\rho$ for all $\rho \in \mathsf{Env}$.*

Proof. By induction on $t \longrightarrow t'$. For the three rules $(\beta\text{-}\lambda)$, $(\beta\text{-}\mathit{Rec\text{-}Zero})$, and $(\beta\text{-}\mathit{Rec\text{-}Succ})$ it even follows from Lemma 3.3 that $t \longrightarrow t'$ implies $[\![t]\!]_\rho = [\![t']\!]_\rho$. Hence, it remains to check η-reduction and the congruence rules.

- *Case*

$$\lambda.\,(\Uparrow^1 t)\,v_0 \longrightarrow t.$$

Reflection and reification.

$$
\begin{array}{lll}
\uparrow \,:\, [\mathsf{D} \to [\mathit{TM}_\bot \to \mathsf{D}]] & & \\
\uparrow^{\mathsf{Pi}\, a\, g} \hat{t} & = \mathsf{Lam}\, f & \text{where } f(d) = \uparrow^{g(d)}(\hat{t} \downarrow^{a} d) \\
\uparrow^{c} \hat{t} & = \mathsf{Ne}\, \hat{t} & \text{if } c \neq \bot, c \neq \mathsf{Pi} \ldots \\
\Downarrow \,:\, [\mathsf{D} \to \mathit{TM}_\bot] & & \\
\Downarrow_k (\mathsf{Pi}\, a\, g) & = \Pi(\Downarrow_k a)\,(\Downarrow_{k+1} g(d)) & \text{where } d = \uparrow^{a} \hat{v}_{-(k+1)} \\
\Downarrow_k \mathsf{Nat} & = \mathit{Nat} & \\
\Downarrow_k \mathsf{Set} & = \mathit{Set} & \\
\Downarrow_k (\mathsf{Ne}\, \hat{t}) & = \hat{t}(k) & \\
\downarrow \,:\, [\mathsf{D} \to [\mathsf{D} \to \mathit{TM}_\bot]] & & \\
\downarrow_k^{\mathsf{Set}} a & = \Downarrow_k a & \\
\downarrow_k^{\mathsf{Pi}\, a\, g} e & = \lambda \downarrow_{k+1}^{g(d)}(e \cdot d) & \text{where } d = \uparrow^{a} \hat{v}_{-(k+1)} \\
\downarrow_k^{\mathsf{Nat}} \mathsf{Zero} & = \mathit{Zero} & \\
\downarrow_k^{\mathsf{Nat}} (\mathsf{Succ}\, d) & = \mathit{Succ}\,(\downarrow_k^{\mathsf{Nat}} d) & \\
\downarrow_k^{c} (\mathsf{Ne}\, \hat{t}) & = \hat{t}(k) & \text{if } c \neq \bot, c \neq \mathsf{Pi} \ldots
\end{array}
$$

Primitive recursion in D.

$$
\begin{array}{ll}
\mathsf{rec} : [\mathsf{D} \to [\mathsf{D} \to [\mathsf{D} \to [\mathsf{D} \to \mathsf{D}]]]] & \\
\mathsf{rec}\, a\, d_z\, d_s\, \mathsf{Zero} & = d_z \\
\mathsf{rec}\, a\, d_z\, d_s\, (\mathsf{Succ}\, e) & = d_s \cdot e \cdot (\mathsf{rec}\, a\, d_z\, d_s\, e) \\
\mathsf{rec}\, a\, d_z\, d_s\, (\mathsf{Ne}\, \hat{t}) & = \uparrow^{a \cdot (\mathsf{Ne}\, \hat{t})}(k \mapsto \mathit{Rec}\,(\downarrow_k^{\mathsf{Nat} \Rightarrow \mathsf{Set}} a) \\
& \qquad (\downarrow_k^{a \cdot \mathsf{Zero}} d_z) \\
& \qquad (\downarrow_k^{\Pi\, \mathsf{Nat}\, (d \mapsto a \cdot d \Rightarrow a \cdot (\mathsf{Succ}\, d))} d_s)\ \hat{t}(k))
\end{array}
$$

Denotation (evaluation) function $[\![\_]\!]_{\_} : \mathit{Tm} \to [\mathsf{Env} \to \mathsf{D}]$.

$$
\begin{array}{lll}
[\![v_i]\!]_\rho & = \rho(i) & \\
[\![\lambda t]\!]_\rho & = \mathsf{Lam}\, f & \text{where } f(d) = [\![t]\!]_{\rho,d} \\
[\![r\, s]\!]_\rho & = [\![r]\!]_\rho \cdot [\![s]\!]_\rho & \\
[\![\mathit{Zero}]\!]_\rho & = \mathsf{Zero} & \\
[\![\mathit{Succ}\, t]\!]_\rho & = \mathsf{Succ}\, [\![t]\!]_\rho & \\
[\![\mathit{Rec}\, A\, z\, s\, t]\!]_\rho & = \mathsf{rec}\, [\![A]\!]_\rho\, [\![z]\!]_\rho\, [\![s]\!]_\rho\, [\![t]\!]_\rho & \\
[\![\Pi A\, B]\!]_\rho & = \mathsf{Pi}\, [\![A]\!]_\rho\, g & \text{where } g(d) = [\![B]\!]_{\rho,d} \\
[\![\mathit{Nat}]\!]_\rho & = \mathsf{Nat} & \\
[\![\mathit{Set}]\!]_\rho & = \mathsf{Set} &
\end{array}
$$

Fig. 2. Key ingredients of NbE.

We have $[\![\lambda. (\Uparrow^1 t)\, v_0]\!]_\rho = \mathsf{Lam}\,(d \mapsto [\![\Uparrow^1 t]\!]_{\rho,d} \cdot [\![v_0]\!]_{\rho,d}) = \mathsf{Lam}\,(d \mapsto [\![t]\!]_\rho \cdot d) = \mathsf{Lam}\,(\mathsf{app}\,[\![t]\!]_\rho) \sqsubseteq [\![t]\!]_\rho$ by Lemma 3.2.

- *Case*
$$\frac{t \longrightarrow t'}{\lambda t \longrightarrow \lambda t'}$$
By induction hypothesis $[\![t]\!]_{\rho,d} \sqsubseteq [\![t']\!]_{\rho,d}$ for all ρ, d. Hence, $[\![\lambda t]\!]_\rho = \mathsf{Lam}\,(d \mapsto [\![t]\!]_{\rho,d}) \sqsubseteq \mathsf{Lam}\,(d \mapsto [\![t']\!]_{\rho,d}) = [\![\lambda t']\!]_\rho$.
- *Case*
$$\frac{A \longrightarrow A'}{\mathit{Rec}\,A\,z\,s\,t \longrightarrow \mathit{Rec}\,A'\,z\,s\,t}$$
By induction hypothesis $[\![A]\!]_\rho \sqsubseteq [\![A']\!]_\rho$. Since rec is continuous, $[\![\mathit{Rec}\,A\,z\,s\,t]\!]_\rho = \mathsf{rec}\,[\![A]\!]_\rho\,[\![z]\!]_\rho\,[\![s]\!]_\rho\,[\![t]\!]_\rho \sqsubseteq \mathsf{rec}\,[\![A']\!]_\rho\,[\![z]\!]_\rho\,[\![s]\!]_\rho\,[\![t]\!]_\rho = [\![\mathit{Rec}\,A\,z\,s\,t]\!]_\rho$.

The other cases are analogous. □

Identity valuation.

We define a special valuation $\rho_\Gamma \in \mathsf{Env}$ by induction on Γ:

$$\begin{aligned} \rho_\diamond(i) &= \bot \\ \rho_{\Gamma,A} &= \rho_\Gamma, (\uparrow^{[\![A]\!]_{\rho_\Gamma}} \hat{v}_{-|\Gamma,A|}) \end{aligned}$$

The valuation ρ_Γ is the semantic equivalent of the syntactical identity substitution, σ_0, defined by $\sigma_0(i) = v_i$.

Lemma 3.5 $\rho_\Gamma(i) = \uparrow^{[\![\Gamma(i)]\!]_{\rho_\Gamma}} \hat{v}_{i-|\Gamma|}$ *for* $0 \le i < |\Gamma|$.

Proof. By induction on Γ. In case $\Gamma = \diamond$, there is nothing to show.

$$\begin{aligned} \rho_{\Gamma,A}(0) &= \uparrow^{[\![A]\!]_{\rho_\Gamma}} \hat{v}_{-|\Gamma,A|} \\ &= \uparrow^{[\![(\Gamma,A)(0)]\!]_{\rho_{\Gamma,A}}} \hat{v}_{0-|\Gamma,A|} && \text{since } (\Gamma,A)(0) = \Uparrow^1 A \\ \rho_{\Gamma,A}(i+1) &= \rho_\Gamma(i) \\ &= \uparrow^{[\![\Gamma(i)]\!]_{\rho_\Gamma}} \hat{v}_{i-|\Gamma|} && \text{by ind.hyp.} \\ &= \uparrow^{[\![(\Gamma,A)(i+1)]\!]_{\rho_{\Gamma,A}}} \hat{v}_{i+1-|\Gamma,A|} && \text{since } (\Gamma,A)(i+1) = \Uparrow^1 \Gamma(i). \end{aligned}$$

□

Normalization by evaluation for terms and types is now implemented by these two functions:

$$\begin{aligned} \mathsf{nbe}^A_\Gamma t &:= \downarrow^{[\![A]\!]_{\rho_\Gamma}}_{|\Gamma|} [\![t]\!]_{\rho_\Gamma} \\ \mathsf{Nbe}_\Gamma A &:= \Downarrow_{|\Gamma|} [\![A]\!]_{\rho_\Gamma} \end{aligned}$$

4 Completeness of NbE

In this section we establish the fact that well-typed $\beta\eta$-equal terms evaluate to the same long normal form.

$$\Gamma \vdash t, t' : A \;\&\; t =_{\beta\eta} t' \Longrightarrow \mathsf{nbe}^A_\Gamma t \equiv \mathsf{nbe}^A_\Gamma t' \in \mathit{Tm}$$

We also establish the analogous fact for types:

$$\Gamma \vdash A, A' \ \&\ A =_{\beta\eta} A' \Longrightarrow \mathsf{Nbe}_\Gamma A \equiv \mathsf{Nbe}_\Gamma A' \in Tm$$

We proceed by constructing an extensional PER model of total elements in D (similar to the one in [1]) and show that the denotation of $\beta\eta$-equal terms are related in the PER assigned to their type. Finally we prove that such related objects are reified ("brought down") to syntactically identical terms.

PER model.

Let Rel denote the set of relations on D and $\mathsf{Per} \subseteq \mathsf{Rel}$ the set of partial equivalence relations on D. If $\mathcal{A} \in \mathsf{Rel}$, we write $d = d' \in \mathcal{A}$ for $(d, d') \in \mathcal{A}$ and $d \in \mathcal{A}$ for $d = d \in \mathcal{A}$. If $\mathcal{A} \in \mathsf{Rel}$ and $\mathcal{G}(d) \in \mathsf{Rel}$ for each $d \in \mathcal{A}$ we let

$$\Pi \mathcal{A}\, \mathcal{G} = \{(e, e') \mid (e \cdot d, e' \cdot d') \in \mathcal{G}(d) \text{ for all } (d, d') \in \mathcal{A}\}.$$

If $\mathcal{A} \in \mathsf{Per}$ and $\mathcal{G}(d) \in \mathsf{Per}$ for all $d \in \mathcal{A}$, then $\Pi \mathcal{A}\, \mathcal{G} \in \mathsf{Per}$. Let $\mathcal{N}e \in \mathsf{Per}$ be $\{(\mathsf{Ne}\, \hat{s}, \mathsf{Ne}\, \hat{s}) \mid \hat{s} \in TM\}$.

Semantical natural numbers.

We inductively define $\mathcal{N}at \in \mathsf{Per}$ by the following rules.

$$\frac{}{\mathsf{Zero} = \mathsf{Zero} \in \mathcal{N}at} \qquad \frac{d = d' \in \mathcal{N}at}{\mathsf{Succ}\, d = \mathsf{Succ}\, d' \in \mathcal{N}at} \qquad \frac{}{\mathsf{Ne}\, \hat{t} = \mathsf{Ne}\, \hat{t} \in \mathcal{N}at}$$

Semantical "sets".

We shall now define a partial equivalence relation $\mathcal{S}et$ for "equal sets" together with partial equivalence relations $[c]$ for "equal elements" of a set $c \in \mathcal{S}et$. One naturally tries to generate $\mathcal{S}et$ inductively and $[c]$ by structural recursion on $\mathcal{S}et$. However, the introduction rule for Π for $\mathcal{S}et$ will then refer negatively to $[-]$, and we therefore do not have a positive inductive definition in the usual sense. Instead this is an example of a simultaneous *inductive-recursive* definition. Such definitions are both constructively and classically meaningful [10,11] and are often needed in the metatheory of dependent type theory. We shall now show how the inductive-recursive definition of $\mathcal{S}et$ and $[c]$ can be understood classically by first giving a monotone inductive definition of $[c]$ which is then used in the definition of $\mathcal{S}et$.

Lemma 4.1 (Interpretation function) *There is a partial function* $[\_] \in \mathsf{D} \rightharpoonup \mathsf{Per}$ *satisfying the equations:*

$$\begin{array}{ll} [\mathsf{Pi}\, a\, g] & = \Pi\, [a]\, (d \mapsto [g(d)]) \\ [\mathsf{Nat}] & = \mathcal{N}at \\ [\mathsf{Ne}\, \hat{t}] & = \mathcal{N}e. \end{array}$$

Proof. We define the graph $\mathsf{T} \subseteq \mathcal{P}(\mathsf{D} \times \mathsf{Per})$ of $[\_]$ inductively by the following rules.

$$\frac{(a, \mathcal{A}) \in \mathsf{T} \quad (g(d), \mathcal{G}(d)) \in \mathsf{T} \text{ for all } d \in \mathcal{A}}{(\mathsf{Pi}\, a\, g, \Pi \mathcal{A}\, \mathcal{G}) \in \mathsf{T}} \qquad \frac{}{(\mathsf{Nat}, \mathcal{N}at) \in \mathsf{T}} \qquad \frac{}{(\mathsf{Ne}\, \hat{t}, \mathcal{N}e) \in \mathsf{T}}$$

(This is a monotone inductive definition on sets, see for example Aczel [2].) By an easy induction on the membership in T we prove that $(a, \mathcal{A}) \in \mathsf{T}$ and $(a, \mathcal{A}') \in \mathsf{T})$ imply $\mathcal{A} = \mathcal{A}'$, hence, $[a] = \mathcal{A} \iff (a, \mathcal{A}) \in \mathsf{T}$ defines a partial function. □

We inductively define $\mathcal{S}et \in \mathsf{Rel}$ by the following rules.

$$\frac{a = a' \in \mathcal{S}et \qquad g(d) = g'(d') \in \mathcal{S}et \text{ for all } d = d' \in [a]}{\mathsf{Pi}\, a\, g = \mathsf{Pi}\, a'\, g' \in \mathcal{S}et}$$

$$\frac{}{\mathsf{Nat} = \mathsf{Nat} \in \mathcal{S}et} \qquad \frac{}{\mathsf{Ne}\, \hat{t} = \mathsf{Ne}\, \hat{t} \in \mathcal{S}et}$$

The following lemma shows that $\mathcal{S}et \in \mathsf{Per}$ and $[\_] \in \mathcal{S}et \to \mathsf{Per}$, thus, $[\_]$ is a total interpretation function for semantical sets.

Lemma 4.2 (Well-definedness of $\mathcal{S}et$ and interpretation)

(i) *If $c = c' \in \mathcal{S}et$ then $[c], [c']$ are defined and $[c] = [c']$.*

(ii) *If $a = b \in \mathcal{S}et$ and $b = c \in \mathcal{S}et$ then $a = c \in \mathcal{S}et$.*

(iii) *If $a = b \in \mathcal{S}et$ then $b = a \in \mathcal{S}et$.*

Proof. Each by induction on the (first) derivation of $\_ = \_ \in \mathcal{S}et$. Analogous proofs can be found in [1, Appendix B]. □

Note that $\perp \notin \mathcal{S}et$ and that $\perp \notin [c]$ for all $c \in \mathcal{S}et$.

Lemma 4.3 (Semantical sets are upward-closed)

(i) *If $c \in \mathcal{S}et$ and $c \sqsubseteq c'$ then $c = c' \in \mathcal{S}et$.*

(ii) *If $c \in \mathcal{S}et$, $e \in [c]$, and $e \sqsubseteq e'$, then $e = e' \in [c]$.*

Proof. Each induction on $c \in \mathcal{S}et$. For the first proposition, consider the case $c = \mathsf{Ne}\, \hat{t} \sqsubseteq c'$. This implies $c' = \mathsf{Ne}\, \hat{t}$. Next, consider the case $c = \mathsf{Pi}\, a\, g \in \mathcal{S}et$ and assume $\mathsf{Pi}\, a\, g \sqsubseteq c'$. Then c' must have the shape $\mathsf{Pi}\, a'\, g'$ with $a \sqsubseteq a'$ and $g(d') \sqsubseteq g'(d')$ for all $d' \in \mathsf{D}$. By induction hypothesis $a = a' \in \mathcal{S}et$ and $g(d) = g(d') = g'(d') \in \mathcal{S}et$ for all $d = d' \in [a]$, so $\mathsf{Pi}\, a\, g = \mathsf{Pi}\, a'\, g' \in \mathcal{S}et$.

For the second proposition, the only interesting case is $e \in [\mathsf{Pi}\, a\, g]$ and $e \sqsubseteq e'$. By monotonicity of app, $e \cdot d \sqsubseteq e' \cdot d$ holds for all $d \in \mathsf{D}$. Hence, by induction hypothesis, $e \cdot d' = e' \cdot d' \in [g(d')]$ for all $d' \in [a]$, For arbitrary $d = d' \in [a]$ we have by assumption $e \cdot d = e \cdot d' \in [g(d)]$; together, since $[g(d')] = [g(d)]$, $e \cdot d = e' \cdot d' \in [g(d)]$. Thus, $e \sqsubseteq e' \in [\mathsf{Pi}\, a\, g]$. □

Semantical types.

We extend the interpretation function by the clause $[\mathsf{Set}] = \mathcal{S}et$ and define an inductive judgement $\_ = \_ \in \mathcal{T}ype$ as follows.

$$\frac{c = c' \in \mathcal{S}et}{c = c' \in \mathcal{T}ype} \qquad \frac{}{\mathsf{Set} = \mathsf{Set} \in \mathcal{T}ype}$$

$$\frac{a = a' \in \mathcal{T}ype \qquad g(d) = g'(d') \in \mathcal{T}ype \text{ for all } d = d' \in [a]}{\mathsf{Pi}\, a\, g = \mathsf{Pi}\, a'\, g' \in \mathcal{T}ype}$$

As for semantical sets, $c = c' \in \mathcal{T}ype$ implies $[c] = [c'] \in \mathsf{Per}$, and $\mathcal{T}ype \in \mathsf{Per}$. Also, $\bot \notin \mathcal{T}ype$.

Lemma 4.4 (Semantical types are upward-closed)

(i) *If $c \in \mathcal{T}ype$ and $c \sqsubseteq c'$ then $c = c' \in \mathcal{T}ype$.*

(ii) *If $c \in \mathcal{T}ype$, $e \in [c]$, and $e \sqsubseteq e'$, then $e = e' \in [c]$.*

Proof. Analogously to Lemma 4.3. □

Lemma 4.5 (Up and down)

(i) *If $c = c' \in \mathcal{T}ype$ then $\uparrow^c \hat{t} = \uparrow^{c'} \hat{t} \in [c]$.*

(ii) *If $c = c' \in \mathcal{S}et$ then $\Downarrow c \equiv \Downarrow c' \in TM$.*

(iii) *If $c = c' \in \mathcal{T}ype$ then $\Downarrow c \equiv \Downarrow c' \in TM$.*

(iv) *If $c = c' \in \mathcal{T}ype$ and $e = e' \in [c]$ then $\downarrow^c e \equiv \downarrow^{c'} e' \in TM$.*

Proof. Simultaneously by induction on $c = c' \in \mathcal{T}ype$ or $\mathcal{S}et$, respectively. We show the proof of the first proposition.

If $c = \mathsf{Pi}\, a\, g$ and $c' = \mathsf{Pi}\, a'\, g'$ then $\uparrow^c \hat{t} = \mathsf{Lam}\,(d \mapsto \uparrow^{g(d)}(\hat{t} \downarrow^a d))$ and $\uparrow^{c'} \hat{t} = \mathsf{Lam}\,(d \mapsto \uparrow^{g'(d)}(\hat{t} \downarrow^{a'} d))$. By induction hypothesis, $\downarrow^a d$ and $\downarrow^{a'} d$ are well-defined and identical term families for $d \in [a] = [a']$, and hence, $\uparrow^{g(d)}(\hat{t} \downarrow^a d) = \uparrow^{g'(d)}(\hat{t} \downarrow^{a'} d) \in [g(d)]$, again by induction hypothesis.

If c, c' are not Pis, then $\uparrow^c \hat{t} = \uparrow^{c'} \hat{t} = \mathsf{Ne}\, \hat{t} \in [c]$.

For the fourth proposition, consider the case $c = \mathsf{Pi}\, a\, g$ and $c' = \mathsf{Pi}\, a'\, g'$ and $e = e' \in [\mathsf{Pi}\, a\, g]$. We show $\downarrow_k^{\mathsf{Pi}\, a\, g} e \equiv \downarrow_k^{\mathsf{Pi}\, a'\, g'} e'$ for arbitrary $k \in \mathbb{N}$. Let $d := \uparrow^a \hat{v}_{-(k+1)}$ and $d' := \uparrow^{a'} \hat{v}_{-(k+1)}$. Since $d = d' \in [a]$ by induction hypothesis 1, we have $e \cdot d = e' \cdot d' \in [g(d)]$, and by induction hypothesis 4, $\downarrow_{k+1}^{g(d)}(e \cdot d) \equiv \downarrow_{k+1}^{g'(d')}(e' \cdot d')$. Hence, $\lambda \downarrow_{k+1}^{g(d)}(e \cdot d) \equiv \lambda \downarrow_{k+1}^{g'(d')}(e' \cdot d')$, which was to be shown. □

Lemma 4.6 (Soundness of recursion) *Assume*

(i) *$a \cdot d = a' \cdot d' \in \mathcal{T}ype$ for all $d = d' \in \mathcal{N}at$,*

(ii) *$d_z = d'_z \in [a \cdot \mathsf{Zero}]$,*

(iii) *$d_s = d'_s \in [\mathsf{Pi}\,\mathsf{Nat}\,(d_m \mapsto a \cdot d_m \Rightarrow a \cdot (\mathsf{Succ}\, d_m))]$,*

(iv) *and $e = e' \in \mathcal{N}at$.*

Then $\mathsf{rec}\ a\ d_z\ d_s\ e = \mathsf{rec}\ a'\ d'_z\ d'_s\ e' \in [a \cdot e]$.

Proof. By induction on $e = e' \in \mathcal{N}at$. The interesting case is when $e = e' = \mathsf{Ne}\, \hat{t}$. By Lemma 4.5, the following are well-defined liftable terms:

$$
\begin{array}{lll}
\hat{A}(k) & := \downarrow_k^{\mathsf{Nat} \Rightarrow \mathsf{Set}} a & \\
& = \lambda. \Downarrow_{k+1}(a \cdot (\mathsf{Ne}\, \hat{v}_{-(k+1)})) & \equiv \lambda. \Downarrow_{k+1}(a' \cdot (\mathsf{Ne}\, \hat{v}_{-(k+1)})) \\
\hat{z} & := \downarrow^{a \cdot \mathsf{Zero}} d_z & \equiv \downarrow^{a' \cdot \mathsf{Zero}} d'_z \\
\hat{s} & := \downarrow^{\Pi\, \mathsf{Nat}\, (d \mapsto a \cdot d \Rightarrow a \cdot (\mathsf{Succ}\, d))} d_s & \equiv \downarrow^{\Pi\, \mathsf{Nat}\, (d \mapsto a' \cdot d \Rightarrow a' \cdot (\mathsf{Succ}\, d))} d'_s
\end{array}
$$

Hence, $\hat{r} := Rec\, \hat{A}\, \hat{z}\, \hat{s}\, \hat{t} \in TM$. Again by Lemma 4.5, we have $\uparrow^{a \cdot (\mathsf{Ne}\, \hat{t})} \hat{r} = \mathsf{rec}\ a\ d_z\ d_s\ (\mathsf{Ne}\, \hat{t}) = \mathsf{rec}\ a'\ d'_z\ d'_s\ (\mathsf{Ne}\, \hat{t}') = \uparrow^{a' \cdot (\mathsf{Ne}\, \hat{t})} \hat{r} \in [a \cdot (\mathsf{Ne}\, \hat{t})]$. □

Semantical contexts.

Let

$$\rho = \rho' \in [\Gamma] \;:\Longleftrightarrow\; \rho(i) = \rho'(i) \in [\llbracket \Gamma(i) \rrbracket_\rho] \text{ for } 0 \le i < |\Gamma|$$

Lemma 4.7 (Context extension) *$(\rho, d) = (\rho', d') \in [\Gamma, A]$ iff $\rho = \rho' \in [\Gamma]$ and $d = d' \in [\llbracket A \rrbracket_\rho]$.*

Proof. Let $0 \le i < |\Gamma, A|$. We consider the proposition $(\rho, d)(i) = (\rho', d')(i) \in [\llbracket (\Gamma, A)(i) \rrbracket_{\rho,d}]$ for the principal values of i. If $i = 0$, this proposition reduces to $d = d' \in [\llbracket \Uparrow^1 A \rrbracket_{\rho,d}] = [\llbracket A \rrbracket_\rho]$. Otherwise, it reduces to $\rho(i-1) = \rho'(i-1) \in [\llbracket \Uparrow^1 (\Gamma(i)) \rrbracket_{\rho,d}] = [\llbracket \Gamma(i) \rrbracket_\rho]$ where $0 \le i - 1 < |\Gamma|$. $\square$

We define valid contexts $\Gamma \models$ inductively by the following rules:

$$\frac{}{\diamond \models} \qquad \frac{\Gamma \models \qquad \llbracket A \rrbracket_\rho = \llbracket A \rrbracket_{\rho'} \in \mathcal{T}ype \text{ for all } \rho = \rho' \in [\Gamma]}{\Gamma, A \models}$$

Validity.

We let

$$\begin{array}{lcl}
\Gamma \models A & :\Longleftrightarrow & \Gamma \models \text{ and } \forall \rho = \rho' \in [\Gamma].\ \llbracket A \rrbracket_\rho = \llbracket A \rrbracket_{\rho'} \in \mathcal{T}ype \\
\Gamma \models A = A' & :\Longleftrightarrow & \Gamma \models \text{ and } \forall \rho = \rho' \in [\Gamma].\ \llbracket A \rrbracket_\rho = \llbracket A' \rrbracket_{\rho'} \in \mathcal{T}ype \\
\Gamma \models t : A & :\Longleftrightarrow & \Gamma \models A \text{ and } \forall \rho = \rho' \in [\Gamma].\ \llbracket t \rrbracket_\rho = \llbracket t \rrbracket_{\rho'} \in [\llbracket A \rrbracket_\rho] \\
\Gamma \models t = t' : A & :\Longleftrightarrow & \Gamma \models A \text{ and } \forall \rho = \rho' \in [\Gamma].\ \llbracket t \rrbracket_\rho = \llbracket t' \rrbracket_{\rho'} \in [\llbracket A \rrbracket_\rho]
\end{array}$$

Lemma 4.8 (Convertible terms are semantically related)

(i) *If $\Gamma \models A, A'$ and $A =_{\beta\eta} A'$ then $\Gamma \models A = A'$.*

(ii) *If $\Gamma \models t, t' : A$ and $t =_{\beta\eta} t'$ then $\Gamma \models t = t' : A$.*

Proof. Fix some $\rho = \rho' \in [\Gamma]$. By assumption, $a := \llbracket A \rrbracket_\rho = \llbracket A \rrbracket_{\rho'} \in \mathcal{T}ype$ and $a' = \llbracket A' \rrbracket_{\rho'} \in \mathcal{T}ype$. Further, $A \longrightarrow^* B \,{}^*\!\!\longleftarrow A'$, which implies $\llbracket A \rrbracket_{\rho'} \sqsubseteq \llbracket B \rrbracket_{\rho'} =: b$ and $a' \sqsubseteq b$. Since $\mathcal{T}ype$ is upward-closed, $a = \llbracket A \rrbracket_{\rho'} = b = a' \in \mathcal{T}ype$. $\square$

The next theorem establishes the soundness of the inference rules w. r. t. our PER model. A simple consequence is that NbE is complete, i. e., will answer "yes" on $\beta\eta$-equal terms if used as an equality test.

Theorem 4.9 (Validity)

(i) *If $\Gamma \vdash$ then $\Gamma \models$.*

(ii) *If $\Gamma \vdash A$ then $\Gamma \models A$.*

(iii) *If $\Gamma \vdash t : A$ then $\Gamma \models t : A$.*

Proof. Simultaneously by induction on the derivation.

- *Case*

$$\frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda t : \Pi\, A\, B}$$

Assume $\rho = \rho' \in [\Gamma]$. Let $a = [\![A]\!]_\rho$, $a' = [\![A]\!]_{\rho'}$, $g(d) = [\![B]\!]_{\rho,d}$, $g'(d) = [\![B]\!]_{\rho',d}$, $f(d) = [\![t]\!]_{\rho,d}$, and $f'(d) = [\![t]\!]_{\rho',d}$. We have $[\![\Pi\, A\, B]\!]_\rho = \mathsf{Pi}\, a\, g = \mathsf{Pi}\, a'\, g' = [\![\Pi A\, B]\!]_{\rho'} \in \mathcal{T}\!ype$, since by induction hypothesis, $a = a' \in \mathcal{T}\!ype$ and $g(d) = g'(d') \in \mathcal{T}\!ype$ for all $d = d' \in [a]$. Furthermore, by induction hypothesis, $f(d) = f'(d') \in [g(d)]$ for all $d = d' \in [a]$, so we have $[\![\lambda t]\!]_\rho = \mathsf{Lam}\, f = \mathsf{Lam}\, f' = [\![\lambda t]\!]_{\rho'} \in [\mathsf{Pi}\, a\, g]$.

- *Case*

$$\frac{\Gamma \vdash r : \Pi\, A\, B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B[s]}$$

  By induction hypothesis and definition of $[\mathsf{Pi}\, [\![A]\!]_\rho\, (d \mapsto [\![B]\!]_{\rho,d})]$, using the identity $[\![B[s]]\!]_\rho = [\![B]\!]_{\rho,[\![s]\!]\rho}$.

- *Case*

$$\frac{\Gamma \vdash z : C[\mathit{Zero}] \qquad \begin{array}{c}\Gamma, \mathit{Nat} \vdash C \\ \Gamma \vdash s : \Pi\, \mathit{Nat}\, (C \Rightarrow C[\mathit{Succ}\, v_0])\end{array} \qquad \Gamma \vdash t : \mathit{Nat}}{\Gamma \vdash \mathit{Rec}\, (\lambda C)\, z\, s\, t : C[t]}$$

  By Lemma 4.6.

- *Case*

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A'}{\Gamma \vdash t : A'}\ A =_{\beta\eta} A'$$

  By induction hypothesis, $\Gamma \models A$ and $\Gamma \models A'$. By Lemma 4.8, $\Gamma \models A = A'$, meaning that for any $\rho \in [\Gamma]$, $[\![A]\!]_\rho = [\![A']\!]_\rho \in \mathcal{T}\!ype$. Hence $[[\![A]\!]_\rho] = [[\![A']\!]_\rho]$, which entails the goal.

□

Corollary 4.10 (Completeness of NbE)

(i) *If* $\Gamma \vdash t, t' : A$ *and* $t =_{\beta\eta} t'$ *then* $\mathsf{nbe}^A_\Gamma t \equiv \mathsf{nbe}^A_\Gamma t' \in \mathit{Tm}$.

(ii) *If* $\Gamma \vdash A, A'$ *and* $A =_{\beta\eta} A'$ *then* $\mathsf{Nbe}_\Gamma A \equiv \mathsf{Nbe}_\Gamma A' \in \mathit{Tm}$.

As a consequence of the corollary, NbE is terminating on well-typed terms.

5 Term Model and Soundness of NbE

Soundness of NbE means that the algorithm returns a term which is $\beta\eta$-equal to the input. To prove this property we use a term model where the denotation function is just parallel substitution. Fortunately, we do not have to go all the way and give an interpretation of syntactical types. For our purposes, it is sufficient to interpret each semantical type by a Kripke logical relation between terms t and domain elements d which expresses that the reification of the domain element d is $\beta\eta$-equal to the term t it is related to. By then showing that for each well-typed term, its denotation in the term model is logically related to its denotation in the domain model, we establish soundness of NbE.

Substitutions.

For $\sigma \in \mathbb{N} \to \mathit{Tm}$ we define an update operation σ, s as follows:

$$\begin{aligned}(\sigma, s)(0) &= s \\ (\sigma, s)(i+1) &= \sigma(i)\end{aligned}$$

Let $\Uparrow^1\sigma$ be a shorthand for $\Uparrow^1 \circ\, \sigma$. Lifting shall bind stronger than update, thus, $\Uparrow^k\sigma, s$ is to be read as $(\Uparrow^k\sigma), s$.

We inductively define parallel substitution $(\!|\_|\!)_\_ \in \mathit{Tm} \to (\mathbb{N} \to \mathit{Tm}) \to \mathit{Tm}$ by the following clauses:

$$\begin{aligned}(\!|v_i|\!)_\sigma &= \sigma(i) \\ (\!|\lambda t|\!)_\sigma &= \lambda.\, (\!|t|\!)_{\Uparrow^1\sigma, v_0} \\ (\!|r\, s|\!)_\sigma &= (\!|r|\!)_\sigma\, (\!|s|\!)_\sigma \\ (\!|\mathit{Zero}|\!)_\sigma &= \mathit{Zero} \\ (\!|\mathit{Succ}\, t|\!)_\sigma &= \mathit{Succ}\, (\!|t|\!)_\sigma\end{aligned}
\qquad
\begin{aligned}(\!|\mathit{Rec}\, A\, z\, s\, t|\!)_\sigma &= \mathit{Rec}\, (\!|A|\!)_\sigma\, (\!|z|\!)_\sigma\, (\!|s|\!)_\sigma\, (\!|t|\!)_\sigma \\ (\!|\Pi\, A\, B|\!)_\sigma &= \Pi\, (\!|A|\!)_\sigma\, (\!|B|\!)_{\Uparrow^1\sigma, v_0} \\ (\!|\mathit{Nat}|\!)_\sigma &= \mathit{Nat} \\ (\!|\mathit{Set}|\!)_\sigma &= \mathit{Set}\end{aligned}$$

Lemma 5.1 *Let $\boldsymbol{v} = v_{k-1}, \dots, v_0$ with $|\boldsymbol{v}| = k$. Then $(\!|t|\!)_{\Uparrow^{k+1}\sigma, v_k, \boldsymbol{v}}[s/k] = (\!|t|\!)_{\Uparrow^k\sigma, s, \boldsymbol{v}}$.*

Proof. By induction on t. We spell out the proof for variables and for a binder.

- *Case v_ℓ.*
  If $\ell \geq k+1$, then $(\!|v_\ell|\!)_{\Uparrow^{k+1}\sigma, v_k, \boldsymbol{v}}[s/k] = (\Uparrow^{k+1}\sigma(\ell - (k+1)))[s/k] = \Uparrow^k\sigma(\ell - (k+1)) = (\!|v_\ell|\!)_{\Uparrow^k\sigma, s, \boldsymbol{v}}$.
  If $\ell = k$ then $(\!|v_k|\!)_{\Uparrow^{k+1}\sigma, v_k, \boldsymbol{v}}[s/k] = v_k[s/k] = s = (\!|v_k|\!)_{\Uparrow^k\sigma, s, \boldsymbol{v}}$.
  If $\ell < k$ then $(\!|v_k|\!)_{\Uparrow^{k+1}\sigma, v_k, \boldsymbol{v}}[s/k] = v_\ell[s/k] = v_\ell = (\!|v_\ell|\!)_{\Uparrow^k\sigma, s, \boldsymbol{v}}$.
- *Case λ.*

$$\begin{aligned}(\!|\lambda t|\!)_{\Uparrow^{k+1}\sigma, v_k, \boldsymbol{v}}[s/k] &= (\lambda (\!|t|\!)_{\Uparrow^{k+2}\sigma, v_{k+1}, \Uparrow^1\boldsymbol{v}, v_0})[s/k] \\ &= \lambda((\!|t|\!)_{\Uparrow^{k+2}\sigma, v_{k+1}, \Uparrow^1\boldsymbol{v}, v_0}[\Uparrow^1 s/(k+1)]) \\ &= \lambda(\!|t|\!)_{\Uparrow^{k+1}\sigma, \Uparrow^1 s, \Uparrow^1\boldsymbol{v}, v_0} \\ &= (\!|\lambda t|\!)_{\Uparrow^k\sigma, s, \boldsymbol{v}}.\end{aligned}$$

□

Corollary 5.2 $(\!|t|\!)_{\Uparrow^1\sigma, v_0}[s] = (\!|t|\!)_{\sigma, s}$

Now we can construct the term model $\mathsf{T} := \mathit{Tm}/{=_{\beta\eta}}$ by identifying $\beta\eta$-equal terms. Let the notation $t = t' \in \mathsf{T}$ mean that t, t' are well-defined terms in Tm and $t =_{\beta\eta} t'$. In particular, if t or t' is an instance $\hat{t}(k)$ of a liftable term $\hat{t}$, *well-defined* means that there are no negative indices in $\hat{t}(k)$.

Lemma 5.3 (Term model) $(\mathsf{T}, (\!|\_|\!)_\_, \_\,\_, \mathit{Rec})$ *is a weakly extensional λ-model.*

Proof. Most conditions are trivially satisfied, we show $(\!|\lambda t|\!)_\sigma s = (\!|t|\!)_{\sigma, s}$ in T:

$$(\!|\lambda t|\!)_\sigma s = (\lambda (\!|t|\!)_{\Uparrow^1\sigma, v_0})\, s = (\!|t|\!)_{\Uparrow^1\sigma, v_0}[s] = (\!|t|\!)_{\sigma, s}.$$

□

Kripke logical relations.

By induction on $a \in \mathcal{T}ype$ we define the relation $\mathsf{R}^a_k \subseteq \mathsf{T} \times [a]$ for $k \in \mathbb{N}$.

$$r \ \mathsf{R}^{\mathsf{Pi}\,a\,g}_k \ e \iff (\Uparrow^\ell r)\, s \ \mathsf{R}^{g(d)}_{k+\ell} \ e \cdot d \quad \text{for all } \ell \in \mathbb{N} \text{ and } s \ \mathsf{R}^a_{k+\ell} \ d$$

$$s \quad \mathsf{R}^c_k \quad d \iff \Uparrow^\ell s = \downarrow^c_{k+\ell} d \in \mathsf{T} \quad \text{for all } \ell \in \mathbb{N} \text{ where } c \neq \mathsf{Pi} \ldots$$

Lemma 5.4 (Equality) *If $c = c' \in \mathcal{T}ype$ then $\mathsf{R}^c_k = \mathsf{R}^{c'}_k$.*

Proof. By induction on $c = c' \in \mathcal{T}ype$. If c, c' are not function types, then we have already $c = c'$, so the claim follows trivially. Otherwise, $c = \mathsf{Pi}\,a\,g$ and $c' = \mathsf{Pi}\,a'\,g'$ with $a = a' \in \mathcal{T}ype$ and $g(d) = g'(d') \in \mathcal{T}ype$ for all $d = d' \in [a]$. Assume $r \ \mathsf{R}^c_k \ e$ and $s \ \mathsf{R}^{a'}_{k+\ell} \ d$. By induction hypothesis, $s \ \mathsf{R}^a_{k+\ell} \ d$, hence, $(\Uparrow^\ell r)\, s \ \mathsf{R}^{g(d)}_{k+\ell} \ e \cdot d$. Again, by induction hypothesis, $(\Uparrow^\ell r)\, s \ \mathsf{R}^{g'(d)}_{k+\ell} \ e \cdot d$, thus, $r \ \mathsf{R}^{c'}_k \ e$. □

Lemma 5.5 (Monotonicity) *If $c \in \mathcal{T}ype$ and $r \ \mathsf{R}^c_k \ e$ then $\Uparrow^\ell r \ \mathsf{R}^c_{k+\ell} \ e$ for all $\ell \in \mathbb{N}$.*

Proof. By induction on $c \in \mathcal{T}ype$. □

Lemma 5.6 (Up and down for R^c_k) *Let $c \in \mathcal{T}ype$.*

(i) *If $\Uparrow^\ell r = \hat{r}(k + \ell) \in \mathsf{T}$ for all $\ell \in \mathbb{N}$, then $r \ \mathsf{R}^c_k \uparrow^c \hat{r}$.*

(ii) *If $r \ \mathsf{R}^c_k \ e$ then $\Uparrow^\ell r = \downarrow^c_{k+\ell} e \in \mathsf{T}$ for all $\ell \in \mathbb{N}$.*

Proof. Simultaneously by induction on $c \in \mathcal{T}ype$. First proposition:

- *Case $c \neq \mathsf{Pi} \ldots$.* Let $\ell \in \mathbb{N}$. Then $\Uparrow^\ell r = \hat{r}(k + \ell) = \downarrow^c_{k+\ell} \mathsf{Ne}\,\hat{r} = \downarrow^c_{k+\ell} \uparrow^c \hat{r} \in \mathsf{T}$, hence, $r \ \mathsf{R}^c_k \uparrow^c \hat{r}$ by definition.
- *Case $c = \mathsf{Pi}\,a\,g$.* To show $r \ \mathsf{R}^{\mathsf{Pi}\,a\,g}_k \ \hat{r}$, we assume $\ell \in \mathbb{N}$ and $s \ \mathsf{R}^a_{k+\ell} \ d$ and prove $(\Uparrow^\ell r)\, s \ \mathsf{R}^{g(d)}_{k+\ell} \ (\uparrow^{\mathsf{Pi}\,a\,g} \hat{r}) \cdot d$, where the r.h.s. simplifies to $\uparrow^{g(d)}(\hat{r} \downarrow^a d)$. Applying the induction hypothesis, it remains to show for arbitrary $\ell' \in \mathbb{N}$ that $\Uparrow^{\ell'}((\Uparrow^\ell r)\, s) = (\hat{r} \downarrow^a d)(k+\ell+\ell') \in \mathsf{T}$, or equivalently, $(\Uparrow^{\ell+\ell'} r)\,(\Uparrow^{\ell'} s) = \hat{r}(k+\ell+\ell') \downarrow^a_{k+\ell+\ell'} d \in \mathsf{T}$. But this equation holds, since by induction hypothesis 2, $\Uparrow^{\ell'} s = \downarrow^a_{k+\ell+\ell'} d \in \mathsf{T}$.

Second proposition:

- *Case $c \neq \mathsf{Pi} \ldots$.* By definition.
- *Case $c = \mathsf{Pi}\,a\,g$.* Fix some $\ell \in \mathbb{N}$ and let $d = \uparrow^a \hat{v}_{-(k+\ell+1)}$. Given $r \ \mathsf{R}^{\mathsf{Pi}\,a\,g}_k \ e$, we have to show $\Uparrow^\ell r = \downarrow^{\mathsf{Pi}\,a\,g}_{k+\ell} e \in \mathsf{T}$. By induction hypothesis 1, $v_0 \ \mathsf{R}^a_{k+\ell+1} \ d$, hence from the assumption, $(\Uparrow^{\ell+1} r)\, v_0 \ \mathsf{R}^{g(d)}_{k+\ell+1} \ e \cdot d$. By induction hypothesis 2, $(\Uparrow^{\ell+1} r)\, v_0 = \downarrow^{g(d)}_{k+\ell+1}(e \cdot d) \in \mathsf{T}$, so we have $\Uparrow^\ell r =_{\beta\eta} \lambda.(\Uparrow^{\ell+1} r)\, v_0 =_{\beta\eta} \lambda. \downarrow^{g(d)}_{k+\ell+1}(e \cdot d) = \downarrow^{\mathsf{Pi}\,a\,g}_{k+\ell} e \in \mathsf{T}$.

□

Logical Relations for Contexts.

We define

$$\sigma \ \mathsf{R}^\Gamma_k \ \rho \ :\iff \ \forall i.\, \Gamma(i) = A \implies \sigma(i) \ \mathsf{R}^{[\![A]\!]_\rho}_k \ \rho(i)$$

Theorem 5.7 *Let* $\sigma \ \mathsf{R}^{\Gamma}_{k} \ \rho$.

(i) *If* $\Gamma \vdash t : A$ *then* $(\!|t|\!)_\sigma \ \mathsf{R}_k^{[\![A]\!]_\rho} \ [\![t]\!]_\rho$.

(ii) *If* $\Gamma \vdash A$ *then* $(\!|A|\!)_\sigma = \Downarrow_k [\![A]\!]_\rho \in \mathsf{T}$.

Proof. Each by induction on the typing derivation.

- *Case*
$$\frac{\Gamma \vdash}{\Gamma \vdash v_i : \Gamma(i)} \ 0 \le i < |\Gamma|$$
Let $a := [\![\Gamma(i)]\!]_\rho$. By assumption, $(\!|v_i|\!)_\sigma = \sigma(i) \ \mathsf{R}^a_k \ \rho(i) = [\![v_i]\!]_\rho$.
- *Case*
$$\frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda t : \Pi \, A \, B}$$
Let $a := [\![A]\!]_\rho \in \mathcal{T}ype$ and $g(d) := [\![B]\!]_{\rho,d} \in [a] \to \mathcal{T}ype$. We have to show $(\!|\lambda t|\!)_\sigma \ \mathsf{R}_k^{\mathsf{Pi}\, a\, g} \ [\![\lambda t]\!]_\rho = \mathsf{Lam}\,(d \mapsto [\![t]\!]_{\rho,d})$ which amounts to showing $(\Uparrow^\ell (\!|\lambda t|\!)_\sigma)\, s = (\!|\lambda t|\!)_{\Uparrow^\ell \sigma}\, s =_{\beta\eta} (\!|t|\!)_{\Uparrow^\ell \sigma, s} \ \mathsf{R}^{g(d)}_{k+\ell} \ [\![t]\!]_{\rho,d}$ for arbitrary $\ell \in \mathbb{N}$ and $s \ \mathsf{R}^a_{k+\ell} \ d$. Since $(\Uparrow^\ell \sigma, s) \ \mathsf{R}^{\Gamma,A}_{k+\ell} \ (\rho, d)$ by monotonicity of R (Lemma 5.5), this is just an instance of the induction hypothesis.
- *Case*
$$\frac{\Gamma \vdash r : \Pi \, A \, B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B[s]}$$
Let $a := [\![A]\!]_\rho \in \mathcal{T}ype$ and $g(d) := [\![B]\!]_{\rho,d} \in [a] \to \mathcal{T}ype$. Further, set $d := [\![s]\!]_\rho \in [a]$. Observe that $[\![B[s]]\!]_\rho = [\![B]\!]_{\rho,[\![s]\!]_\rho} = g(d)$. We have to show that $(\!|r\, s|\!)_\sigma \ \mathsf{R}^{g(d)}_k \ [\![r\, s]\!]_\rho$ which follows by the induction hypotheses $(\!|r|\!)_\sigma \ \mathsf{R}^{\mathsf{Pi}\, a\, g}_k \ [\![r]\!]_\rho$ and $(\!|s|\!)_\sigma \ \mathsf{R}^a_k \ [\![s]\!]_\rho$.
- *Case*
$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A'}{\Gamma \vdash t : A'} \ A =_{\beta\eta} A'$$
Since $[\![A]\!]_\rho = [\![A']\!]_\rho \in \mathcal{T}ype$ by Theorem 4.9, we have $\mathsf{R}_k^{[\![A]\!]_\rho} = \mathsf{R}_k^{[\![A']\!]_\rho}$ by Lemma 5.4.
- *Case*
$$\frac{\Gamma \vdash A : \mathit{Set} \qquad \Gamma, A \vdash B : \mathit{Set}}{\Gamma \vdash \Pi \, A \, B : \mathit{Set}}$$
Recall that $\downarrow^{\mathsf{Set}} = \Downarrow$. Let $a := [\![A]\!]_\rho \in \mathcal{T}ype$ and $d := \uparrow^a \hat{v}_{-(k+1)}$. By monotonicity of the logical relation $\Uparrow^1 \sigma \ \mathsf{R}^\Gamma_{k+1} \ \rho$ and since $v_0 \ \mathsf{R}^a_{k+1} \ d$ by Lemma 5.6, we have $(\Uparrow^1 \sigma, v_0) \ \mathsf{R}^{\Gamma,A}_{k+1} \ (\rho, d)$. Hence, by induction hypothesis, $(\!|B|\!)_{\Uparrow^1 \sigma, v_0} = \Downarrow_{k+1} [\![B]\!]_{\rho,d} \in \mathsf{T}$. Also, by induction hypothesis, $(\!|A|\!)_\sigma = \Downarrow_k [\![A]\!]_\rho \in \mathsf{T}$. Together,
$$\begin{aligned} (\!|\Pi \, A \, B|\!)_\sigma &= \Pi (\!|A|\!)_\sigma \, (\!|B|\!)_{\Uparrow^1 \sigma, v_0} \\ &=_{\beta\eta} \Pi (\Downarrow_k [\![A]\!]_\rho)(\Downarrow_{k+1} [\![B]\!]_{\rho,d}) \\ &= \Downarrow_k (\mathsf{Pi}\, [\![A]\!]_\rho \, (d' \mapsto [\![B]\!]_{\rho,d'})) \\ &= \Downarrow_k [\![\Pi \, A \, B]\!]_\rho. \end{aligned}$$
The same proof works for $\Gamma \vdash \Pi \, A \, B$.

□

Recall that $\sigma_0(i) = v_i$ and that ρ_Γ is the semantical counterpart of σ_0.

Lemma 5.8 (Context satisfiable)

(i) *If* $\Gamma \models$ *then* $\rho_\Gamma \in [\Gamma]$.

(ii) *If* $\Gamma \models$ *then* $\sigma_0 \ \mathsf{R}^{\Gamma}_{|\Gamma|} \ \rho_\Gamma$.

Proof. Let $0 \leq i < |\Gamma|$ and $a := [\![\Gamma(i)]\!]_{\rho_\Gamma} \in \mathcal{T}ype$. First, $(\rho_\Gamma)(i) = \uparrow^a \hat{v}_{i-|\Gamma|} \in [a]$ by Lemma 4.5. Secondly, $\sigma_0(i) = v_i \ \mathsf{R}^a_{|\Gamma|} \uparrow^a \hat{v}_{i-|\Gamma|} = (\rho_\Gamma)(i)$ by Lemma 5.6. □

Corollary 5.9 (Soundness of NbE)

(i) *If* $\Gamma \vdash t : A$ *then* $t =_{\beta\eta} \mathsf{nbe}^A_\Gamma t$.

(ii) *If* $\Gamma \vdash A$ *then* $A =_{\beta\eta} \mathsf{Nbe}_\Gamma A$.

Proof. Let $a := [\![A]\!]_{\rho_\Gamma}$ which is in $\mathcal{T}ype$ by validity. By the logical relations theorem, $t = (\!|t|\!)_{\sigma_0} \ \mathsf{R}^a_{|\Gamma|} \ [\![t]\!]_{\rho_\Gamma}$. Hence, by Lemma 5.6, $t =_{\beta\eta} \downarrow^a_{|\Gamma|} [\![t]\!]_{\rho_\Gamma}$. Similarly, $A = (\!|A|\!)_{\sigma_0} =_{\beta\eta} \Downarrow_{|\Gamma|} [\![A]\!]_{\rho_\Gamma}$. □

6 Conclusion

In this article, we have provided a normalization-by-evaluation algorithm for lambda-terms and primitive recursion. By constructing a PER model and a Kripke logical relation we have proven that it decides $\beta\eta$-equality of Martin-Löf Type Theory with an Universe a la Russell. With NbE sound and complete, we can replace the side condition $A =_{\beta\eta} A'$ in the conversion rule by the test $\mathsf{Nbe}_\Gamma A \equiv \mathsf{Nbe}_\Gamma A'$. This is the crucial step towards a (bidirectional) type checking algorithm for the system presented.

Acknowledgment.

We are grateful to Thierry Coquand for many discussions and much insight into normalization for type theory in general and feedback on normalization by evaluation in particular. For example, he pointed out to us that it is more elegant to write the algorithm for lambda terms a la Curry than for the lambda terms a la Church that we used in a preliminary version. We also thank the anonymous referees for detailed comments on draft versions of this article.

References

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

[2] Aczel, P., *An introduction to inductive definitions*, in: J. Barwise, editor, *Handbook of Mathematical Logic*, North-Holland, 1977 pp. 739–782.

[3] Aczel, P., *The strength of Martin-Löf's type theory with one universe*, in: S. Miettinen and J. Väänanen, editors, *Proceedings of the Symposium on Mathematical Logic (Oulu 1974)*, 1977, pp. 1–32, report No 2 of Dept. Philosophy, University of Helsinki.

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

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

[6] Berger, U., M. Eberl and H. Schwichtenberg, *Term rewriting for normalization by evaluation*, Information and Computation **183** (2003), pp. 19–42.

[7] Coquand, C., *Agda: An interactive proof editor*, http://agda.sourceforge.net/.

[8] Coquand, C., *A realizability interpretation of Martin-Löf's type theory*, in: G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory* (1998).

[9] Danielsson, N. A., *A partial formalisation of a dependently typed language as an inductive-recursive family* (2006), to appear in TYPES'06 proceedings.

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

[11] Dybjer, P. and A. Setzer, *Indexed induction-recursion*, Journal of Logic and Algebraic Programming (2006).

[12] Filinski, A., *A semantic account of type-directed partial evaluation*, in: G. Nadathur, editor, *International Conference on Principles and Practice of Declarative Programming*, number 1702 in LNCS, 1999, pp. 378–395.

[13] Filinski, A. and H. K. Rohde, *A denotational account of untyped normalization by evaluation.*, in: I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, Lecture Notes in Computer Science **2987** (2004), pp. 167–181.

[14] Martin-Löf, P., *About models for intuitionistic type theories and the notion of definitional equality*, in: S. Kanger, editor, *Proceedings of the 3rd Scandinavian Logic Symposium*, 1975, pp. 81–109.

[15] Martin-Löf, P., *An intuitionistic theory of types: Predicative part*, in: H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73* (1975), pp. 73–118.

[16] Martin-Löf, P., *Constructive mathematics and computer programming*, in: *Logic, Methodology and Philosophy of Science, VI, 1979* (1982), pp. 153–175.

[17] Martin-Löf, P., "Intuitionistic Type Theory," Bibliopolis, 1984.

[18] Martin-Löf, P., *Amendment to intuitionistic type theory* (1986), notes from a lecture given in Göteborg.

[19] Martin-Löf, P., *An intuitionistic theory of types*, in: G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory* (1998), reprinted version of an unpublished report from 1972.

[20] Martin-Löf, P., *Normalization by evaluation and by the method of computability* (2004), talk at JAIST, Japan Advanced Institute of Science and Technology, Kanazawa.

[21] Nordström, B., K. Petersson and J. Smith, "Programming in Martin-Löf's Type Theory: an Introduction," Oxford University Press, 1990.

[22] Scott, D. S., *Domains for denotational semantics*, in: *Automata, Languages and Programming, Proceedings of the 9th International Colloquium* (1982), pp. 577–613.

A Haskell program

This appendix contains a `Haskell` program implementing normalization by evaluation as described in this article. The *only* difference between `Haskell`'s semantics and the semantics developed in Section 3 is that the Haskell program may return a partially defined value at certain places, whereas our argument assumes the totally undefined value $\perp$. This is due to the non-strictness of the constructors in Haskell.

However, being *more defined than we need to* does not do any harm. Since we show that for $\Gamma \vdash t : A$ the function $\mathtt{nbe}^A_\Gamma t$ yields a totally defined value, namely, a member of Tm, the program $\mathtt{nbe}\ \Gamma\ A\ t$ has to produce the same result, as there are no values that are more defined than a total one—in the usual domain theoretic order.

The discrepancy between the mathematical treatment and the Haskell program could be overcome by adding strictness annotations in the datatypes `Tm` and `D`. In this way we could model the construction of D exactly. However, as we have argued, the Haskell program works correctly without these annotations. Alternatively, one could switch to a strict language like ML to get a one-to-one correspondence with the mathematical development.

Terms (including types).

```
data Tm = Var Int | App Tm Tm | Lam Tm
          | Zero | Succ Tm | Rec Tm Tm Tm Tm
          | Nat | Pi Tm Tm | Set
            deriving (Show,Eq)
type TM = Int -> Tm
```

Domain Semantics.

```
data D = PiD D (D -> D)  -- pi-type
       | NatD            -- code for Nat
       | SetD            -- universe
       | LamD (D -> D)   -- function
       | ZeroD           -- natural numbers
       | SuccD D
       | NeD TM          -- neutral terms

arrD :: D -> D -> D
arrD a b = PiD a (\ _ -> b)

appD :: D -> D -> D
appD (LamD f) d = f d

varD :: D -> Int -> D
varD a k = up a (\ l -> Var (l+k))
```

Reflection (up) and reification (down).

```
up :: D -> TM -> D
up (PiD a g) t = LamD (\ d -> up (g d) (\ k -> App (t k) (down a d k)))
up _         t = NeD t

downT :: D -> TM
downT (PiD a g) k = Pi (downT a k) $ downT (g $ varD a $ (-(k+1))) (k+1)
downT  NatD     k = Nat
downT  SetD     k = Set
downT (NeD t)   k = t k

down :: D -> D -> TM
down (PiD a g) e k = Lam $ down (g d) (appD e d) (k+1)
    where d = varD a (-(k+1))
down SetD a k = downT a k
down NatD ZeroD     k = Zero
down NatD (SuccD d) k = Succ (down NatD d k)
down _ (NeD t)      k = t k
```

Primitive Recursion.

```
recD :: D -> D -> D -> D -> D
recD a z s ZeroD     = z
recD a z s (SuccD d) = s `appD` d `appD` (recD a z s d)
recD a z s d         = up (a `appD` d) (\ k ->
  Rec (down (NatD `arrD` SetD) a k)
      (down (a `appD` ZeroD) z k)
      (down (PiD NatD (\ n -> (a `appD` n) `arrD` (a `appD` (SuccD n)))) s k)
      (down NatD d k))
```

Environments.

```
type Env = Int -> D

emptyEnv k = error $ "unbound index " ++ show k

ext :: Env -> D -> Env
ext rho a k = if k==0 then a else rho (k-1)
```

Evaluation.

```
eval :: Tm -> Env -> D

eval (Var k)       rho = rho k
eval (App r s)     rho = appD (eval r rho) (eval s rho)
eval (Lam r)       rho = LamD f
```

```
    where f d = eval r (ext rho d)

eval (Zero)        rho = ZeroD
eval (Succ r)      rho = SuccD (eval r rho)
eval (Rec a z s n) rho = recD (eval a rho)
                              (eval z rho) (eval s rho) (eval n rho)

eval (Nat)         rho = NatD
eval (Pi r s)      rho = PiD (eval r rho) g
    where g d = eval s (ext rho d)
eval (Set)         rho = SetD
```

Identity valuation.

```
type Cxt = [Tm]

upG' :: Int -> Cxt -> Env
upG' n [] = emptyEnv
upG' n (a:gamma) = ext rho (varD (eval a rho) (n - length (a:gamma)))
    where rho = (upG' n gamma)

upG :: Cxt -> Env
upG gamma = upG' (length gamma) gamma
```

Normalization by evaluation.

```
nbe  gamma c r = down (eval c (upG gamma)) (eval r (upG gamma)) 0
nbeT gamma a   = downT (eval a (upG gamma)) 0
```