A Modular Type-Checking Algorithm for Type Theory with Singleton Types and Proof Irrelevance

Andreas Abel[1], Thierry Coquand[2], and Miguel Pagano[3]

[1] Ludwig-Maximilians-Universität München, abel@informatik.uni-muenchen.de
[2] Göteborg University, coquand@cs.chalmers.se
[3] Universidad Nacional de Córdoba, miguel.pagano@gmail.com

Abstract. We define a logical framework with singleton types and one universe of small types. We give the semantics using a PER model; it is used for constructing a normalisation-by-evaluation algorithm. We prove completeness and soundness of the algorithm; and get as a corollary the injectivity of type constructors. Then we give the definition of a correct and complete type-checking algorithm for terms in normal form. We extend the results to proof-irrelevant propositions.

1 Introduction and Related Work

One of the raisons d'être of proof-checkers like Agda [26], Coq [18], and Epigram [23] is to decide if a given term has some type; i.e., if a term corresponds to a proof of a proposition [17]. Hence, the convenience of such a system is, in part, determined by the types for which the system can check membership. We extend the decidability of type-checking done in previous works [1, 2] for Martin-Löf type-theories [21, 25] by considering singleton types and proof-irrelevant propositions.

Singleton types were introduced by Aspinall [8] in the context of specification languages. An important use of singletons is as definitions by abbreviations (see [8, 14]); they were also used to model translucent sums in the formalisation of SML [19]. It is interesting to consider singleton types because beta-eta phase separation fails: one cannot do eta-expansion before beta-normalisation because the shape of the types at which to eta-expand is still unknown at this point; and one cannot postpone eta-expansion after beta-normalisation, because eta-expansion can trigger new beta-reductions. Stone and Harper [29] decide type checking in a LF with singleton types and subtyping. Yet it is not clear whether their method extends to computation on the type level. As far as we know, our work is the first where singleton types are considered together with a universe.

De Bruijn proposed the concept of irrelevance of proofs [11], for reducing the burden in the formalisation of mathematics. As shown by Werner [30], the use of proof-irrelevance types together with sigma types is one way to get subset types à la PVS [27] in type-theories having the eta rule—this direction was explored by Sozeau [28, Sec. 3.3].

Checking dependent types relies on checking types for equality. To this end, we compute η-long normal forms using *normalisation by evaluation* (NbE) [22]. Syntactic expressions are evaluated into a semantic domain and then *reified* back to expressions in normal form. To handle functional and open expressions, the semantic domain has to be equipped with variables; a major challenge in rigorous treatments of NbE has been the problem to generate fresh identifiers. Solutions include term families [10], liftable de Bruijn terms [7], or Kripke semantics [4]. In this work we present a novel formulation of NbE which avoids the problem completely: reification is split into an η-expansion phase ($\downarrow$) in the semantics, followed by a read back function (R) into the syntax which is indexed by the number of already used variables. This way, a standard PER model is sufficient, and technical difficulties are avoided.

Outline. The definitions of two calculi are presented in section 2. In section 3 we define the semantics of this LF in a PER model, and we show soundness of the model wrt. the derived rules of the calculus. We use this model to introduce a NbE algorithm, for which we prove completeness (if $t = s$ is derivable, then $\mathbf{nbe}(t)$ and $\mathbf{nbe}(s)$ are identical). In section 4 we prove, using logical relations, the soundness of the algorithm (i.e., $t = \mathbf{nbe}(t)$ is derivable). In section 5 we define a bi-directional algorithm for checking the type of normal forms and inferring the type of neutral terms.

2 The calculus as a Generalised Algebraic Theory

In the section, we introduce the calculus. For ease of reading, and for showing the modularity of our approach, we present it as two calculi: the first one has dependent function spaces, singleton types, and a universe closed under function spaces and singletons. In the second calculus we leave out singleton types and we add proof-irrelevant types.

We present the calculi using the formalism proposed by Cartmell for generalised algebraic theories (GAT) [12]; however, our calculi are not proper GATs (the rules are written in the so-called "informal syntax" and the rule for application is ambiguous). We give only the introductory rules and the axioms; the rules stating that equality is a congruence relation, called derived rules, are omitted. An example of a derived rule is

$$\frac{A = B \in \mathsf{Type}(\Gamma) \qquad \gamma = \delta \in \Delta \to \Gamma}{A\,\gamma = B\,\delta \in \mathsf{Type}(\Delta)}\ .$$

Calculus with singleton types.
Sorts. The set of sort symbols is $\{\mathsf{Ctx}, \to, \mathsf{Type}, \mathsf{Term}\}$.

$$\frac{}{\mathsf{Ctx}\text{ is a type}}\ (\textsc{ctx-sort}) \qquad \frac{\Gamma, \Delta \in \mathsf{Ctx}}{\Gamma \to \Delta\text{ is a type}}\ (\textsc{subs-sort})$$

$$\frac{\Gamma \in \mathsf{Ctx}}{\mathsf{Type}(\Gamma)\text{ is a type}}\ (\textsc{type-sort}) \qquad \frac{\Gamma \in \mathsf{Ctx} \qquad A \in \mathsf{Type}(\Gamma)}{\mathsf{Term}(\Gamma, A)\text{ is a type}}\ (\textsc{term-sort})$$

In the following, whenever a rule has a hypothesis $A \in \mathsf{Type}(\Gamma)$, then $\Gamma \in \mathsf{Ctx}$ shall be a further, implicit hypothesis. Similarly, $\sigma \in \Gamma \to \Delta$ presupposes $\Gamma \in \mathsf{Ctx}$ and $\Delta \in \mathsf{Ctx}$, and $t \in \mathsf{Term}(\Gamma, A)$ presupposes $A \in \mathsf{Type}(\Gamma)$, which in turn presupposes $\Gamma \in \mathsf{Ctx}$. Note that judgements of the form $\Gamma \in \mathsf{Ctx}$, $A \in \mathsf{Type}(\Gamma)$, $t \in \mathsf{Term}(\Gamma, A)$, and $\sigma \in \Gamma \to \Delta$ correspond to the more conventional forms $\Gamma \vdash$, $\Gamma \vdash A$, $\Gamma \vdash t : A$, and $\Gamma \vdash \sigma : \Delta$, resp. In the rest of the paper we use the latter.

Operators. The set of operators is quite large and instead of giving it at once, we define it as the union of the disjoint sets of operators for contexts, substitutions, types, and terms.

Contexts. There are two operators for contexts: $S_C = \{\diamond, \_.\_\}$.

$$\frac{}{\diamond \in \mathsf{Ctx}}\ (\textsc{empty-ctx}) \qquad \frac{\Gamma \in \mathsf{Ctx} \qquad A \in \mathsf{Type}(\Gamma)}{\Gamma.A \in \mathsf{Ctx}}\ (\textsc{ext-ctx})$$

Substitutions. For substitutions we have five operators: $S_S = \{\mathsf{id}_\_, \langle\rangle, (\_,\_), \_\,\_, \mathsf{p}\}$.

$$\frac{\Gamma \in \mathsf{Ctx}}{\mathsf{id}_\Gamma \in \Gamma \to \Gamma}\ (\textsc{id-subs}) \qquad \frac{\Gamma \in \mathsf{Ctx}}{\langle\rangle \in \Gamma \to \diamond}\ (\textsc{empty-subs})$$

$$\frac{\delta \in \Gamma \to \Theta \qquad \sigma \in \Theta \to \Delta}{\sigma\,\delta \in \Gamma \to \Delta}\ (\textsc{comp-subs})$$

$$\frac{\sigma \in \Gamma \to \Delta \qquad t \in \mathsf{Term}(\Gamma, A\,\sigma)}{(\sigma, t) \in \Gamma \to \Delta.A}\ (\textsc{ext-subs}) \qquad \frac{A \in \mathsf{Type}(\Gamma)}{\mathsf{p} \in \Gamma.A \to \Gamma}\ (\textsc{fst-subs})$$

Types. The set of operators for types is $S_T = \{\mathsf{U}, \mathsf{Fun}\,\_\,\_, \_\,\_, \{\_\}_\_\}$.

$$\frac{\Gamma \in \mathsf{Ctx}}{\mathsf{U} \in \mathsf{Type}(\Gamma)}\ (\textsc{u-f}) \quad \frac{A \in \mathsf{Term}(\Gamma, \mathsf{U})}{A \in \mathsf{Type}(\Gamma)}\ (\textsc{u-el}) \quad \frac{A \in \mathsf{Type}(\Gamma) \qquad B \in \mathsf{Type}(\Gamma.A)}{\mathsf{Fun}\,A\,B \in \mathsf{Type}(\Gamma)}\ (\textsc{fun-f})$$

$$\frac{A \in \mathsf{Type}(\Gamma) \qquad t \in \mathsf{Term}(\Gamma, A)}{\{t\}_A \in \mathsf{Type}(\Gamma)}\ (\textsc{sing-f}) \quad \frac{A \in \mathsf{Type}(\Delta) \qquad \sigma \in \Gamma \to \Delta}{A\,\sigma \in \mathsf{Type}(\Gamma)}\ (\textsc{subs-type})$$

Terms. The set of operators for terms is $S_E = \{\mathsf{Fun}\,\_\,\_, \{\_\}_\_, \_\,\_, \mathsf{q}, \lambda\_, \mathsf{App}\ \_\ \_\}$.

$$\frac{A \in \mathsf{Term}(\Gamma, \mathsf{U}) \qquad B \in \mathsf{Term}(\Gamma.A, \mathsf{U})}{\mathsf{Fun}\,A\,B \in \mathsf{Term}(\Gamma, \mathsf{U})}\ (\textsc{fun-u-i}) \quad \frac{t \in \mathsf{Term}(\Gamma.A, B)}{\lambda t \in \mathsf{Term}(\Gamma, \mathsf{Fun}\,A\,B)}\ (\textsc{fun-i})$$

$$\frac{B \in \mathsf{Type}(\Gamma.A) \qquad t \in \mathsf{Term}(\Gamma, \mathsf{Fun}\,A\,B) \qquad u \in \mathsf{Term}(\Gamma, A)}{\mathsf{App}\ t\ u \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}_\Gamma, u))}\ (\textsc{fun-el})$$

$$\frac{\sigma \in \Gamma \to \Delta \qquad t \in \mathsf{Term}(\Delta, A)}{t\,\sigma \in \mathsf{Term}(\Gamma, A\,\sigma)}\ (\textsc{subs-term}) \quad \frac{A \in \mathsf{Type}(\Gamma)}{\mathsf{q} \in \mathsf{Term}(\Gamma.A, A\,\mathsf{p})}\ (\textsc{hyp})$$

$$\frac{A \in \mathsf{Term}(\Gamma, \mathsf{U}) \qquad t \in \mathsf{Term}(\Gamma, A)}{\{t\}_A \in \mathsf{Term}(\Gamma, \mathsf{U})}\ (\textsc{sing-u-i}) \quad \frac{t \in \mathsf{Term}(\Gamma, A)}{t \in \mathsf{Term}(\Gamma, \{t\}_A)}\ (\textsc{sing-i})$$

$$\frac{a \in \mathsf{Term}(\Gamma, A) \qquad t \in \mathsf{Term}(\Gamma, \{a\}_A)}{t \in \mathsf{Term}(\Gamma, A)}\ (\textsc{sing-el})$$

Axioms. We give the axioms without the premises, except in the cases where they can not be inferred.

Substitutions.

$$\begin{aligned} (\sigma\,\delta)\,\gamma &= \sigma\,(\delta\,\gamma) & \langle\rangle\,\sigma &= \langle\rangle \\ \mathsf{id}_\Gamma\,\sigma &= \sigma & \sigma\,\mathsf{id}_\Gamma &= \sigma \\ \mathsf{id}_\diamond &= \langle\rangle & \mathsf{id}_{\Gamma.A} &= (\mathsf{p},\mathsf{q}) \\ \mathsf{p}\,(\sigma,t) &= \sigma & (\sigma,t)\,\delta &= (\sigma\,\delta, t\,\delta) \end{aligned}$$

Substitutions on types, and terms; η and β-axioms.

$$\begin{aligned} \mathsf{U}\,\gamma &= \mathsf{U} & \{t\}_A\,\sigma &= \{t\,\sigma\}_{A\,\sigma} \\ (\mathsf{Fun}\,A\,B)\,\sigma &= \mathsf{Fun}\,(A\,\sigma)\,(B\,(\sigma\,\mathsf{p},\mathsf{q})) & \mathsf{q}\,(\sigma,t) &= t \\ t\,(\sigma\,\delta) &= (t\,\sigma)\,\delta & t\,\mathsf{id}_\Gamma &= t \\ (\lambda t)\,\sigma &= \lambda(t\,(\sigma\,\mathsf{p},\mathsf{q})) & (\mathsf{App}\ r\ s)\,\sigma &= \mathsf{App}\ (r\,\sigma)\ (s\,\sigma) \\ \mathsf{App}\ (\lambda t)\ r &= t\,(\mathsf{id}_\Gamma, r) & \lambda(\mathsf{App}\ (t\,\mathsf{p})\ \mathsf{q}) &= t \end{aligned}$$

$$\frac{t, t' \in \mathsf{Term}(\Gamma, \{a\}_A)}{t = t' \in \mathsf{Term}(\Gamma, \{a\}_A)}\ (\textsc{sing-eq-i}) \qquad \frac{t = t' \in \mathsf{Term}(\Gamma, \{a\}_A)}{t = t' \in \mathsf{Term}(\Gamma, A)}\ (\textsc{sing-eq-el})$$

Notation. We denote with $|\Gamma|$ the length of the context Γ; and $\Gamma!i$ is the projection of the i-th component of Γ, for $0 \leqslant i < |\Gamma|$. We say $\Delta \leqslant^i \Gamma$ if $\Delta \vdash \mathsf{p}^i : \Gamma$; where p^i is the i-fold composition of p with itself. We denote with *Terms* the set of words freely generated using symbols in $S_S \cup S_T \cup S_E$. We write $t \equiv_T t'$ for denoting syntactically equality of t and t' in $T \subseteq \mathit{Terms}$. We call A the *tag* of $\{a\}_A$.

Definition 1 (Neutral terms, and normal forms).

$$\begin{aligned} \mathit{Ne} \ni k &::= \mathsf{q} \mid \mathsf{q}\mathsf{p}^{i+1} \mid \mathsf{App}\ k\ v \\ \mathit{Nf} \ni v, V, W &::= \mathsf{U} \mid \mathsf{Fun}\,V\,W \mid \{v\}_V \mid \lambda v \mid k \end{aligned}$$

Remark 1 (Weakening of judgements). Let $\Delta \leqslant^i \Gamma$, $\Gamma \vdash A = A'$, and $\Gamma \vdash t = t' : A$; then $\Delta \vdash A\,\mathsf{p}^i = A'\,\mathsf{p}^i$, and $\Delta \vdash t\,\mathsf{p}^i = t'\,\mathsf{p}^i : A\,\mathsf{p}^i$.

Remark 2 (Syntactic validity).

1. If $\Gamma \vdash t : A$, then $\Gamma \vdash A$.
2. If $\Gamma \vdash t = t' : A$, then both $\Gamma \vdash t : A$, and $\Gamma \vdash t' : A$.
3. If $\Gamma \vdash A = A'$, then both $\Gamma \vdash A$, and $\Gamma \vdash A'$.

Lemma 1 (Inversion of types).

1. *If $\Gamma \vdash \mathsf{Fun}\,A\,B$, then $\Gamma \vdash A$, and $\Gamma.A \vdash B$.*
2. *If $\Gamma \vdash \{a\}_A$, then $\Gamma \vdash A$, and $\Gamma \vdash a : A$.*
3. *If $\Gamma \vdash k$, then $\Gamma \vdash k : \mathsf{U}$.*

Lemma 2 (Inversion of typing).

1. *If* $\Gamma \vdash \mathsf{Fun}\, A'\, B' : A$*, then* $\Gamma \vdash A' : \mathsf{U}$*, and also* $\Gamma.A' \vdash B' : \mathsf{U}$*;*
2. *If* $\Gamma \vdash \{b\}_B : A$*, then* $\Gamma \vdash B : \mathsf{U}$*, and also* $\Gamma \vdash b : B$*;*
3. *If* $\Gamma \vdash \lambda t : A$*, then* $\Gamma.A' \vdash t : B'$*.*
4. *If* $\Gamma \vdash t : \{a\}_A$*, then* $\Gamma \vdash t : A$*, and* $\Gamma \vdash t = a : A$*.*
5. *If* $\Gamma \vdash \mathsf{q}\,\mathsf{p}^i : A$*, then either* $\Gamma \vdash A = (\Gamma!i)\,\mathsf{p}^{i+1}$*; or* $\Gamma \vdash A = \{a\}_{A'}$*, and* $\Gamma \vdash a = \mathsf{q}\,\mathsf{p}^i : A'$*.*

Calculus with Proof-Irrelevance. Our treatment of proof-irrelevance is based on [9, 20]. The motivation for a canonical element witnessing the existence of a proof is to keep the modularity of the algorithm for deciding equality; but since its introduction breaks completeness of type-checking, we consider two calculi: the proof (programming) developments are done in a calculus without PRF-TM, and the type-checking is performed in a calculus with it. We show then that this is a conservative extension.

Introductory rules.

$$\frac{A \in \mathsf{Type}(\Gamma)}{\mathsf{Prf}\, A \in \mathsf{Type}(\Gamma)}\ (\textsc{prf-f}) \qquad \frac{a \in \mathsf{Term}(\Gamma, A)}{[a] \in \mathsf{Term}(\Gamma, \mathsf{Prf}\, A)}\ (\textsc{prf-i}) \qquad \frac{t \in \mathsf{Term}(\Gamma, A)}{\mathsf{O} \in \mathsf{Term}(\Gamma, \mathsf{Prf}\, A)}\ (\textsc{prf-tm})$$

$$\frac{A \in \mathsf{Type}(\Gamma) \qquad t, t' \in \mathsf{Term}(\Gamma, \mathsf{Prf}\, A)}{t = t' \in \mathsf{Term}(\Gamma, \mathsf{Prf}\, A)}\ (\textsc{prf-eq})$$

$$\frac{B \in \mathsf{Type}(\Gamma) \qquad b \in \mathsf{Term}(\Gamma.A, B\,\mathsf{p}) \qquad t \in \mathsf{Term}(\Gamma, \mathsf{Prf}\, A)}{b\,\mathsf{where}^B\, t \in \mathsf{Term}(\Gamma, \mathsf{Prf}\, B)}\ (\textsc{prf-el})$$

$$(\mathsf{Prf}\, A)\,\delta = \mathsf{Prf}\,(A\,\delta) \qquad [t]\,\delta = [t\,\delta] \qquad \mathsf{O}\,\delta = \mathsf{O}$$

$$(b\,\mathsf{where}^B\, t)\,\delta = b\,(\delta\,\mathsf{p}, \mathsf{q})\,\mathsf{where}^{B\,\delta}\,(t\,\delta) \qquad b\,\mathsf{where}^B\,[t] = [b\,(\mathsf{id}, t)]$$

Lemma 3 (Inversion).

1. *If* $\Gamma \vdash [t] : A$*, then* $\Gamma \vdash A = \mathsf{Prf}\, A'$ *and* $\Gamma \vdash t' : A'$*.*
2. *If* $\Gamma \vdash b\,\mathsf{where}^B\, t : A$*, then* $\Gamma \vdash A = \mathsf{Prf}\, B$*, and* $\Gamma \vdash t : \mathsf{Prf}\, A'$*, and* $\Gamma.A' \vdash b : B\,\mathsf{p}$*.*

As is expected we have now more normal forms, and more neutral terms:

$$\begin{aligned} Ne \ni k &::= \ldots \mid v\,\mathsf{where}^V\, k \\ Nf \ni v, V &::= \ldots \mid \mathsf{Prf}\, V \mid [v] \mid \mathsf{O} \end{aligned}$$

Now we prove that the calculus with PRF-TM is a conservative extension of the one without it. We decorate the turnstile, and the equality symbol with * for referring to judgements in the extended calculus.

Definition 2. *A term t' is called a* lifting *of a term t, if all the occurrences of* O *in t have been replaced by terms $s_0, \ldots, s_{n-1}$, and* O *does not occur in any s_i. We extend this definition to substitutions, contexts, and equality judgements.*

If Γ' is a lifting of Γ, and $\Gamma =^* \Gamma'$, and also $\Gamma' \vdash$ then we say that Γ' is a good-lifting *of Γ. We extend the definition of good-lifting to the others kinds of judgement.*

Lemma 4. *Let $\Gamma \vdash^* J$, then there exists a good-lifting $\Gamma' \vdash J'$; moreover for any other good-lifting $\Gamma'' \vdash J''$ of $\Gamma \vdash^* J$, we have $\Gamma' = \Gamma''$, and $\Gamma' \vdash J' = J''$.*

Corollary 1. *The calculus $\vdash^*$ is a conservative extension of $\vdash$.*

3 Semantics

In this section we define a PER model of the calculus presented in the previous section. The model is used to define a normalisation function later.

3.1 PER semantics

Definition 3. *We define a domain $D = \mathbb{O} \oplus Var_\bot \oplus [D \to D] \oplus D \times D \oplus D \times D \oplus \mathbb{O} \oplus D \times [D \to D] \oplus D \times D$, where Var is a denumerable set of variables (as usual we write x_i and assume $x_i \neq x_j$ if $i \neq j$, for $i, j \in \mathbb{N}$), $E_\bot = E \cup \{\bot\}$ is lifting, $\mathbb{O} = \{\top\}_\bot$ is the Sierpinski space, $[D \to D]$ is the set of continuous functions from D to D, $\oplus$ is the coalesced sum, and $D \times D$ is the Cartesian product of D [6].*

An element of D which is not $\bot$ can be of one of the forms:

$\top$	(d, d')	for $d, d' \in D$
$\mathsf{Var}\, x_i$	U	for $x_i \in Var$
$\mathsf{Lam}\, f$	$\mathsf{Fun}\, d\, f$	for $d \in D$, and $f \in [D \to D]$
$\mathsf{App}\, d\, d'$	$\mathsf{Sing}\, d\, d'$	for $d, d' \in D$.

We define application $\_ \cdot \_ : [D \times D \to D]$ and the projections $\mathsf{p}, \mathsf{q} : [D \to D]$ by

$$\begin{aligned} f \cdot d &= \text{if } f = \mathsf{Lam}\, f' \text{ then } f'\, d \text{ else } \bot, \\ \mathsf{p}\, d &= \text{if } d = (d_1, d_2) \text{ then } d_1 \text{ else } \bot, \\ \mathsf{q}\, d &= \text{if } d = (d_1, d_2) \text{ then } d_2 \text{ else } \bot. \end{aligned}$$

We define a partial function $\mathsf{R}_{\_}\, \_ : \mathbb{N} \to D \to Terms$ which reifies elements from the model into terms; this function is similar to the read-back function of Gregoire and Leroy's [16].

Definition 4 (Read-back function).

$$\begin{aligned} \mathsf{R}_j\, \mathsf{U} &= \mathsf{U} & \mathsf{R}_j\, (\mathsf{App}\, d\, d') &= \mathsf{App}\ (\mathsf{R}_j\, d)\ (\mathsf{R}_j\, d') \\ \mathsf{R}_j\, (\mathsf{Fun}\, X\, F) &= \mathsf{Fun}\, (\mathsf{R}_j\, X) & \mathsf{R}_j\, (\mathsf{Lam}\, f) &= \lambda(\mathsf{R}_{j+1}\, (f(\mathsf{Var}\, x_j))) \\ &\qquad (\mathsf{R}_{j+1}\, (F(\mathsf{Var}\, x_j))) & \mathsf{R}_j\, (\mathsf{Var}\, x_i) &= \begin{cases} \mathsf{q} & \textit{if } j \leqslant i \\ \mathsf{q}\, \mathsf{p}^{j-i-1} & \textit{if } j > i \end{cases} \\ \mathsf{R}_j\, (\mathsf{Sing}\, d\, X) &= \{\mathsf{R}_j\, d\}_{\mathsf{R}_j\, X} \end{aligned}$$

Partial Equivalence Relations. A partial equivalence relation (PER) over a set D is a binary relation over D which is symmetric and transitive.

If $\mathcal{R}$ is a PER over D, and $(d, d') \in \mathcal{R}$ then it is clear that $(d, d) \in \mathcal{R}$. We define $dom(R) = \{d \in D \mid (d, d) \in \mathcal{R}\}$. If $(d, d') \in \mathcal{R}$, sometimes we will write $d = d' \in \mathcal{R}$, and $d \in \mathcal{R}$ if $d \in dom(\mathcal{R})$. We denote with PER(D) the set of all PERs over D.

If $\mathcal{R} \in \text{PER}(D)$ and $\mathcal{F} : dom(\mathcal{R}) \to \text{PER}(D)$, we say that $\mathcal{F}$ is *a family of PERs indexed by* $\mathcal{R}$ iff for all $d = d' \in \mathcal{R}$, $\mathcal{F}\ d = \mathcal{F}\ d'$. If $\mathcal{F}$ is a family indexed by $\mathcal{R}$, we write $\mathcal{F} : \mathcal{R} \to \text{PER}(D)$.

We define two binary relations over D: one for neutral terms and the other for normal forms.

$$\begin{aligned} d = d' \in \mathcal{N}e &:\iff \forall i \in \mathbb{N}.\, \mathsf{R}_i\, d \text{ and } \mathsf{R}_i\, d' \text{ are defined and } \mathsf{R}_i\, d \equiv_{Ne} \mathsf{R}_i\, d' \\ d = d' \in \mathcal{N}f &:\iff \forall i \in \mathbb{N}.\, \mathsf{R}_i\, d \text{ and } \mathsf{R}_i\, d' \text{ are defined and } \mathsf{R}_i\, d \equiv_{Nf} \mathsf{R}_i\, d' \end{aligned}$$

The following definitions are standard [8, 14] (except for **1**); they will be used in the definition of the model.

Definition 5. *Let $\mathcal{X} \in PER(D)$ and $\mathcal{F} \in \mathcal{X} \to PER(D)$.*

- $\mathbf{1} = \{(\top, \top)\}$;
- $\coprod \mathcal{X}\, \mathcal{F} = \{(d, d') \mid \mathsf{p}\ d = \mathsf{p}\ d' \in \mathcal{X} \textit{ and } \mathsf{q}\ d = \mathsf{q}\ d' \in \mathcal{F}\ (\mathsf{p}\ d)\}$;
- $\prod \mathcal{X}\, \mathcal{F} = \{(f, f') \mid f \cdot d = f' \cdot d' \in \mathcal{F}\ d, \textit{ for all } d = d' \in \mathcal{X}\}$;
- $\{\!\{d\}\!\}_{\mathcal{X}} = \{(e, e') \mid d = e \in \mathcal{X} \textit{ and } d = e' \in \mathcal{X}\}$.

We define $\mathcal{U}, \mathcal{T} \in \text{PER}(D)$ and $[\_] : dom(\mathcal{T}) \to \text{PER}(D)$ using Dybjer's schema of inductive-recursive definition [15]. We show then that $[\_]$ is a family of PERs over D.

Definition 6 (PER model).

- *Inductive definition of* $\mathcal{U} \in PER(D)$.
  - $\mathcal{N}e \subseteq \mathcal{U}$,
  - *if* $X = X' \in \mathcal{U}$ *and* $d = d' \in [X]$, *then* $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X' \in \mathcal{U}$,
  - *if* $X = X' \in \mathcal{U}$ *and for all* $d = d' \in [X]$, $F\ d = F'\ d' \in \mathcal{U}$ *then* $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{U}$.
- *Inductive definition of* $\mathcal{T} \in PER(D)$.
  - $\mathcal{U} \subset \mathcal{T}$,
  - $\mathsf{U} = \mathsf{U} \in \mathcal{T}$,
  - *if* $X = X' \in \mathcal{T}$, *and* $d = d' \in [X]$ *then* $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X' \in \mathcal{T}$,
  - *if* $X = X' \in \mathcal{T}$, *and for all* $d = d' \in [X]$, $F\ d = F'\ d' \in \mathcal{T}$, *then* $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{T}$.
- *Recursive definition of* $[\_] \in dom(\mathcal{T}) \to PER(D)$.
  - $[\mathsf{U}] = \mathcal{U}$,
  - $[\mathsf{Sing}\, d\, X] = \{\!\{d\}\!\}_{[X]}$,
  - $[\mathsf{Fun}\, X\, F] = \prod [X]\, (d \mapsto [F\ d])$,
  - $[d] = \mathcal{N}e$, *in all other cases.*

Lemma 5. *The function* $[\_]$ *is a family of* $PER(D)$ *over* $\mathcal{T}$.

3.2 Normalisation and η-Expansion in the Model

The usual way to define NbE [7] is to introduce a reification function which maps elements from the model into normal forms; and a function mapping neutral terms to elements of the model (the former function is called the inverse of the evaluation function, and the later "make self evaluating" in [10]). A tricky point of the algorithm is to find a new variable when reifying functions as abstractions.

In this work we do not need to worry about variable capturing when reifying, because we can define functions corresponding to reification, and lifting of neutrals *in the model* avoiding completely the need to deal with fresh variables.

Definition 7. *The partial functions* $\uparrow_{\_}\,\_, \downarrow_{\_}\,\_ : D \to D \to D$ *and* $\Downarrow : D \to D$ *are given as follows:*

$$\begin{aligned}
\uparrow_{\mathsf{Fun}\,X\,F} d &= \mathsf{Lam}\,(e \mapsto \uparrow_{F\,e} \mathsf{App}\,d \downarrow_X e) & \downarrow_{\mathsf{Fun}\,X\,F} d &= \mathsf{Lam}\,(e \mapsto \downarrow_{F\,\uparrow_X e} (d \cdot \uparrow_X e))\\
\uparrow_{\mathsf{Sing}\,d\,X} e &= d & \downarrow_{\mathsf{Sing}\,d\,X} e &= \downarrow_X d\\
\uparrow_{\mathsf{U}} d &= d & \downarrow_{\mathsf{U}} d &= \Downarrow d\\
\uparrow_d e &= e & \downarrow_d e &= e, \textit{ in all other cases.}
\end{aligned}$$

$$\begin{aligned}
\Downarrow(\mathsf{Fun}\,X\,F) &= \mathsf{Fun}\,(\Downarrow X)\,(d \mapsto \Downarrow(F \uparrow_X d)) & \Downarrow \mathsf{U} &= \mathsf{U}\\
\Downarrow(\mathsf{Sing}\,d\,X) &= \mathsf{Sing}\,(\downarrow_X d)\,(\Downarrow X) & \Downarrow d &= d, \textit{ in all other cases.}
\end{aligned}$$

Lemma 6 (Characterisation of $\uparrow$, $\downarrow$, and $\Downarrow$). *Let $X = X' \in \mathcal{T}$, then*

1. *if $k = k' \in \mathcal{N}e$ then $\uparrow_X k = \uparrow_{X'} k' \in [X]$;*
2. *if $d = d' \in [X]$, then $\downarrow_X d = \downarrow_{X'} d' \in \mathcal{N}f$;*
3. *and also $\Downarrow X = \Downarrow X' \in \mathcal{N}f$.*

Definition 8 (Semantics).

Contexts.

$$[\![\diamond]\!] = \mathbf{1} \qquad [\![\Gamma.A]\!] = \coprod [\![\Gamma]\!]\,(d \mapsto [[\![A]\!]d])$$

Substitutions.

$$\begin{aligned}
[\![\diamond]\!]d &= \top & [\![\mathsf{id}]\!]d &= d\\
[\![(\gamma, t)]\!]d &= ([\![\gamma]\!]d, [\![t]\!]d) & [\![\mathsf{p}]\!]d &= \mathsf{p}\ d\\
[\![\gamma\,\delta]\!]d &= [\![\gamma]\!]([\![\delta]\!]d)
\end{aligned}$$

Terms (and types).

$$\begin{aligned}
[\![\mathsf{U}]\!]d &= \mathsf{U} & [\![\mathsf{Fun}\,A\,B]\!]d &= \mathsf{Fun}\,([\![A]\!]d)\,(e \mapsto [\![B]\!](d,e))\\
[\![\{a\}_A]\!]d &= \mathsf{Sing}\,([\![a]\!]d)\,([\![A]\!]d) & [\![\mathsf{App}\ t\ u]\!]d &= [\![t]\!]d \cdot [\![u]\!]d\\
[\![\lambda t]\!]d &= \mathsf{Lam}\,(d' \mapsto [\![t]\!](d, d')) & [\![t\,\gamma]\!]d &= [\![t]\!]([\![\gamma]\!]d)\\
[\![\mathsf{q}]\!]d &= \mathsf{q}\ d
\end{aligned}$$

Definition 9 (Validity).

1. $\diamond \vDash$ *iff true*
2. $\Gamma.A \vDash$ *iff* $\Gamma \vDash A$
3. $\Gamma \vDash A$ *iff* $\Gamma \vDash A = A$
4. $\Gamma \vDash A = A'$ *iff* $\Gamma \vDash$ *and for all* $d = d' \in [\![\Gamma]\!]$, $[\![A]\!]d = [\![A']\!]d' \in \mathcal{T}$
5. $\Gamma \vDash t : A$ *iff* $\Gamma \vDash t = t : A$
6. $\Gamma \vDash t = t' : A$ *iff* $\Gamma \vDash A$ *and for all* $d = d' \in [\![\Gamma]\!]$, $[\![t]\!]d = [\![t']\!]d' \in [[\![A]\!]d]$
7. $\Gamma \vDash \sigma : \Delta$ *iff* $\Gamma \vDash \sigma = \sigma : \Delta$
8. $\Gamma \vDash \sigma = \sigma' : \Delta$ *iff* $\Gamma \vDash$, $\Delta \vDash$, *and for all* $d = d' \in [\![\Gamma]\!]$, $[\![\sigma]\!]d = [\![\sigma']\!]d' \in [\![\Delta]\!]$.

Theorem 1 (Soundness of the Judgements). *if* $\Gamma \vdash J$, *then* $\Gamma \vDash J$.

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

Theorem 2 (Completeness of NbE). *If* $\vdash t = t' : A$, *then* $\downarrow_{[\![A]\!]} [\![t]\!] = \downarrow_{[\![A]\!]} [\![t']\!] \in \mathcal{N}f$.

Proof. By Thm. 1 we have $[\![t]\!] = [\![t']\!] \in [[\![A]\!]]$ and we conclude by Lem. 6.

Calculus with Proof-Irrelevance. We extend all the definition concerning the construction of the model;

$$D = \ldots \oplus D \oplus \mathbb{O} \;;$$

the new inhabitants will be written as $\mathsf{Prf}\, d$, and $\star$, respectively. The read-back function is extended by the equations $\mathsf{R}_j\,(\mathsf{Prf}\, d) = \mathsf{Prf}\,(\mathsf{R}_j\, d)$ and $\mathsf{R}_j \star = \mathsf{O}$. We add a new clause in the definition of $\mathcal{T}$,

$$\text{if } X = X' \in \mathcal{T}\text{, then } \mathsf{Prf}\, X = \mathsf{Prf}\, X' \in \mathcal{T}\text{, and } [\mathsf{Prf}\, X] = \{(\star, \star)\} \;.$$

The definitions of normalisation and expansion are extended for $\mathsf{Prf}\, X$,

$$\uparrow_{\mathsf{Prf}\, X} d = \star \qquad \downarrow_{\mathsf{Prf}\, X} d = \star \qquad \Downarrow \mathsf{Prf}\, X = \mathsf{Prf} \Downarrow X \;.$$

The semantic equations for the new constructions are

$$\begin{array}{rr} [\![\mathsf{Prf}\, A]\!]d = \mathsf{Prf}\, [\![A]\!]d & [\![[a]]\!]d = \star \\ [\![b\, \mathsf{where}^B\, t]\!]d = \star & [\![\mathsf{O}]\!]d = \star \;. \end{array}$$

Remark 3. All of lemmata 5, 6, and theorems 1, and 2 are valid for the calculus with proof-irrelevance.

4 Logical relations

In order to prove soundness of our normalisation algorithm we define logical relations [24] between types and elements in the domain of $\mathcal{T}$, and between terms and elements in the domain of the PER corresponding to elements of $\mathcal{T}$.

Definition 10 (Logical relations). *The relations $\Gamma \vdash A \sim X \in \mathcal{T}$ (ternary) and $\Gamma \vdash t : A \sim d \in [X]$ are defined simultaneously by induction on $X \in \mathcal{T}$.*

- *Neutral types: $X \in \mathcal{N}e$.*
  - $\Gamma \vdash A \sim X \in \mathcal{T}$ *iff for all* $\Delta \leqslant^i \Gamma$, $\Delta \vdash A\,\mathsf{p}^i = \mathsf{R}_{|\Delta|} \Downarrow X$.
  - $\Gamma \vdash t : A \sim d \in [X]$ *iff* $\Gamma \vdash A \sim X \in \mathcal{T}$, *and for all* $\Delta \leqslant^i \Gamma$, $\Delta \vdash t\,\mathsf{p}^i = \mathsf{R}_{|\Delta|} \downarrow_X d : A\,\mathsf{p}^i$.
- *Universe $X = \mathsf{U}$.*
  - $\Gamma \vdash A \sim \mathsf{U} \in \mathcal{T}$ *iff* $\Gamma \vdash A = \mathsf{U}$.
  - $\Gamma \vdash t : A \sim X \in [\mathsf{U}]$ *iff* $\Gamma \vdash A = \mathsf{U}$, *and* $\Gamma \vdash t \sim X \in \mathcal{T}$.
- *Singletons.*
  - $\Gamma \vdash A \sim \mathsf{Sing}\,d\,X \in \mathcal{T}$ *iff* $\Gamma \vdash A = \{a\}_{A'}$ *for some* A', a, *and* $\Gamma \vdash a : A' \sim d \in [X]$.
  - $\Gamma \vdash t : A \sim d' \in [\mathsf{Sing}\,d\,X]$ *iff* $\Gamma \vdash A = \{a\}_{A'}$ *for some* A', a, *such that* $\Gamma \vdash t : A' \sim d \in [X]$, *and* $\Gamma \vdash A' \sim X \in \mathcal{T}$.
- *Function spaces.*
  - $\Gamma \vdash A \sim \mathsf{Fun}\,X\,F \in \mathcal{T}$ *iff* $\Gamma \vdash A = \mathsf{Fun}\,A'\,B$, *and* $\Gamma \vdash A' \sim X \in \mathcal{T}$, *and* $\Delta \vdash B\,(\mathsf{p}^i, s) \sim F\,d \in \mathcal{T}$ *for all* $\Delta \leqslant^i \Gamma$ *and* $\Delta \vdash s : A'\,\mathsf{p}^i \sim d \in [X]$.
  - $\Gamma \vdash t : A \sim f \in [\mathsf{Fun}\,X\,F]$ *iff* $\Gamma \vdash A = \mathsf{Fun}\,A'\,B$, $\Gamma \vdash A' \sim X$, *and* $\Delta \vdash \mathsf{App}\ (t\,\mathsf{p}^i)\ s : B\,(\mathsf{p}^i, s) \sim f \cdot d \in [F\,d]$ *for all* $\Delta \leqslant^i \Gamma$ *and* $\Delta \vdash s : A'\,\mathsf{p}^i \sim d \in [X]$.

The following lemmata show that the logical relations are preserved by judgemental equality, weakening of the judgement, and the equalities on the corresponding PERs.

Lemma 7. *Let* $\Gamma \vdash A = A'$, $\Gamma \vdash t = t' : A$, $\Gamma \vdash A \sim X \in \mathcal{T}$, *and* $\Gamma \vdash t : A \sim d \in [X]$; *then* $\Gamma \vdash A' \sim X \in \mathcal{T}$, *and* $\Gamma \vdash t' : A' \sim d \in [X]$.

Lemma 8 (Monotonicity). *Let* $\Delta \leqslant^i \Gamma$, *then*

1. *if* $\Gamma \vdash A \sim X \in \mathcal{T}$, *then* $\Delta \vdash A\,\mathsf{p}^i \sim X \in \mathcal{T}$; *and*
2. *if* $\Gamma \vdash t : A \sim d \in [X]$, *then* $\Delta \vdash t\,\mathsf{p}^i : A\,\mathsf{p}^i \sim d \in [X]$.

Lemma 9. *Let* $\Gamma \vdash A \sim X \in \mathcal{T}$ *and* $\Gamma \vdash t : A \sim d \in [X]$, *then*

1. *if* $X = X' \in \mathcal{T}$, *then* $\Gamma \vdash A \sim X' \in \mathcal{T}$; *and*
2. *if* $d = d' \in [X]$, *then* $\Gamma \vdash t : A \sim d' \in [X]$.

The following lemma plays a key role in the proof of soundness. It proves that if a term is related to some element in (some PER), then it is convertible to the reification of the corresponding element in the PER of normal forms.

Lemma 10. *Let* $\Gamma \vdash A \sim X \in \mathcal{T}$, $\Gamma \vdash t : A \sim d \in [X]$, *and* $k \in \mathcal{N}e$, *then*

1. $\Gamma \vdash A = \mathsf{R}_{|\Gamma|} \Downarrow X$,
2. $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_X d : A$; *and*
3. *if for all* $\Delta \leqslant^i \Gamma$, $\Delta \vdash t\,\mathsf{p}^i = \mathsf{R}_{|\Delta|}\,k : A\,\mathsf{p}^i$, *then* $\Gamma \vdash t : A \sim \uparrow_X k \in [X]$.

In order to finish the proof of soundness we have to prove that each well-typed term (and each well-formed type) is logically related to its denotation; with that aim we extend the definition of logical relations to substitutions and prove the fundamental theorem of logical relations.

Definition 11 (Logical relation for substitutions).

– $\Gamma \vdash \sigma : \diamond \sim d \in \mathbf{1}$.
– $\Gamma \vdash (\sigma, t) : \Delta.A \sim (d, d') \in \coprod \mathcal{X}\,(d \mapsto [F\ d])$ *iff* $\Gamma \vdash \sigma : \Delta \sim d \in \mathcal{X}$, $\Gamma \vdash A\,\sigma \sim F\ d \in \mathcal{T}$, *and* $\Gamma \vdash t : A\,\sigma \sim d' \in [F\ d]$.

After proving the counterparts of 7, 8 and 9 for substitutions, we can proceed with the proof of the main theorem of logical relations.

Theorem 3 (Fundamental theorem of logical relations). *Let* $\Delta \vdash \delta : \Gamma \sim d \in [\![\Gamma]\!]$.

1. *If* $\Gamma \vdash A$, *then* $\Delta \vdash A\,\delta \sim [\![A]\!]d \in \mathcal{T}$;
2. *if* $\Gamma \vdash t : A$, *then* $\Delta \vdash t\,\delta : A\,\delta \sim [\![t]\!]d \in [[\![A]\!]d]$; *and*
3. *if* $\Gamma \vdash \gamma : \Theta$ *then* $\Delta \vdash \gamma\,\delta : \Theta \sim [\![\gamma]\!]d \in [\![\Theta]\!]$.

We define for each context Γ an element ρ_Γ of D, that is, by construction, logically related to id_Γ. This environment will be used to define the normalisation function; also notice that if we instantiate Thm. 3 with ρ_Γ, then a well-typed term under Γ will be logically related to its denotation.

Definition 12. *Let* $\rho_\Gamma = P_\Gamma\ \top$, *where* $P_\diamond\ d = d$ *and* $P_{\Gamma.A}\ d = (d', \uparrow_{[\![A]\!]d'} \mathsf{Var}\, x_{|\Gamma|})$ *with* $d' = P_\Gamma\ d$. *Then* $\Gamma \vdash \mathsf{id}_\Gamma : \Gamma \sim \rho_\Gamma \in [\![\Gamma]\!]$ *for* $\Gamma \in \mathsf{Ctx}$.

Definition 13 (Normalisation algorithm). *Let* $\Gamma \vdash A$, *and* $\Gamma \vdash t : A$.

$$\mathbf{nbe}_\Gamma(A) = \mathsf{R}_{|\Gamma|} \Downarrow [\![A]\!]\rho_\Gamma$$
$$\mathbf{nbe}^A_\Gamma(t) = \mathsf{R}_{|\Gamma|} \downarrow_{[\![A]\!]\rho_\Gamma} [\![t]\!]\rho_\Gamma$$

The first point of soundness is a direct consequence of Thm. 3 and Lem. 7; and the second point is obtained using Lem. 10.

Corollary 2 (Soundness of NbE). *Let* $\Gamma \vdash A$, *and* $\Gamma \vdash t : A$, *then*

1. $\Gamma \vdash A \sim [\![A]\!]\rho_\Gamma \in \mathcal{T}$, *and* $\Gamma \vdash t : A \sim [\![t]\!]\rho_\Gamma \in [[\![A]\!]\rho_\Gamma]$; *and*
2. $\Gamma \vdash A = \mathbf{nbe}(A)$, *and* $\Gamma \vdash t = \mathbf{nbe}(t) : A$.

Remark 4. By expanding the definitions, we easily check

1. $\mathbf{nbe}_\Gamma(\mathsf{Fun}\, A\, B) = \mathsf{Fun}\,(\mathbf{nbe}_\Gamma(A))\,(\mathbf{nbe}_{\Gamma.A}(B))$, and
2. $\mathbf{nbe}_\Gamma(\{a\}_A) = \{\mathbf{nbe}^A_\Gamma(a)\}_{\mathbf{nbe}_\Gamma(A)}$.

Corollary 3. *If* $\Gamma \vdash A$, *and* $\Gamma \vdash A'$, *then we can decide* $\Gamma \vdash A = A'$. *Also if* $\Gamma \vdash t : A$, *and* $\Gamma \vdash t' : A$, *we can decide* $\Gamma \vdash t = t' : A$.

Corollary 4 (Injectivity of $\mathsf{Fun}\,\_\,\_$ and of $\{\_\}_\_$). *If* $\Gamma \vdash \mathsf{Fun}\, A\, B = \mathsf{Fun}\, A'\, B'$, *then* $\Gamma \vdash A = A'$, *and* $\Gamma.A \vdash B = B'$. *Also* $\Gamma \vdash \{t\}_A = \{t'\}_{A'}$, *then* $\Gamma \vdash A = A'$, *and* $\Gamma \vdash t = t' : A$.

Calculus with Proof-Irrelevance. We add the corresponding cases in the definition of logical relations,

$$\Gamma \vdash A \sim \mathsf{Prf}\, X \in \mathcal{T}, \text{ iff } \Gamma \vdash A = \mathsf{Prf}\, A', \text{ and } \Gamma \vdash A' \sim X \in \mathcal{T}; \text{ and}$$
$$\Gamma \vdash t : A \sim d \in [\mathsf{Prf}\, X], \text{ iff } \Gamma \vdash A \sim \mathsf{Prf}\, X \in \mathcal{T}.$$

Remark 5. All the lemmata 7, 8, 9, 10, theorem 3, and remarks 2, 4 are still valid. Moreover we also have $\mathbf{nbe}(\mathsf{Prf}\, A) = \mathsf{Prf}\,(\mathbf{nbe}(A))$.

5 Type-checking algorithm

In this section we define a bi-directional type-checking algorithm for terms in normal form, and a type-inference algorithm for neutral terms. We prove its correctness and completeness.

The algorithm is similar to previous ones [13, 3]. The only difference is due to the presence of singleton types. We deal with this by η-normalising the type, and considering first if the normalised type is a singleton (side-condition in type-checking of neutrals); in that case we check that the term is typeable with the tag of the singleton type, and that it is equal to the term of the singleton.

We stress the importance of having a normalisation function with the property stated in Rem. 4, and also to have decidability of equality. In fact, it is enough to have a function $\mathbf{nbe}(\_)$ such that:

1. $\mathbf{nbe}(\{a\}_A) = \{\mathbf{nbe}(a)\}_{\mathbf{nbe}(A)}$, and $\mathbf{nbe}(\mathsf{Fun}\, A\, B) = \mathsf{Fun}\, \mathbf{nbe}(A) \mathbf{nbe}(B)$;
2. $\mathbf{nbe}_\Gamma(A) = \mathbf{nbe}_\Gamma(B)$ if and only if $\Gamma \vdash A = B$, and $\mathbf{nbe}^A_\Gamma(t) = \mathbf{nbe}^A_\Gamma(t')$, if and only if $\Gamma \vdash t = t' : A$.

In this section, let $V, V', W, v, v', w \in Nf$, and $k \in Ne$. We define a function to get the deepest tag of a singleton, that is essentially the same as in [8],

$$\overline{V} = \begin{cases} \overline{W} & \text{if } V \equiv \{w\}_W \\ V & \text{otherwise.} \end{cases}$$

The predicates for type-checking are defined mutually inductively, together with the function for inferring types.

Definition 14 (Type-checking and type-inference).

Types $\Gamma \Leftarrow V$. We presuppose $\Gamma \vdash$.

$$\frac{}{\Gamma \Leftarrow \mathsf{U}} \quad \frac{\Gamma \Leftarrow V \qquad \Gamma.V \Leftarrow W}{\Gamma \Leftarrow \mathsf{Fun}\, V\, W} \quad \frac{\Gamma \Leftarrow V \qquad \Gamma \vdash v \Leftarrow \mathbf{nbe}(V)}{\Gamma \Leftarrow \{v\}_V} \quad \frac{\Gamma \vdash k \Leftarrow \mathsf{U}}{\Gamma \Leftarrow k}$$

Terms $\Gamma \vdash v \Leftarrow V$. We presuppose $\Gamma \vdash V$, and V in η-long normal form with respect to Γ.

$$\frac{\Gamma \vdash V \Leftarrow \mathsf{U} \qquad \Gamma.V \vdash W \Leftarrow \mathsf{U}}{\Gamma \vdash \mathsf{Fun}\, V\, W \Leftarrow \mathsf{U}} \qquad \frac{\Gamma.V \vdash v \Leftarrow W}{\Gamma \vdash \lambda v \Leftarrow \mathsf{Fun}\, V\, W}$$

$$\frac{\Gamma \vdash V \Leftarrow \mathsf{U} \qquad \Gamma \vdash v \Leftarrow \mathbf{nbe}(V)}{\Gamma \vdash \{v\}_V \Leftarrow \mathsf{U}} \qquad \frac{\Gamma \vdash v \Leftarrow V' \qquad \Gamma \vdash v' = v : V'}{\Gamma \vdash v \Leftarrow \{v'\}_{V'}}$$

$$\frac{\Gamma \vdash k \Rightarrow V' \qquad \Gamma \vdash \overline{V'} = V}{\Gamma \vdash k \Leftarrow V}\ V \not\equiv \{w\}_W$$

Type inference $\Gamma \vdash k \Rightarrow V$. We presuppose $\Gamma \vdash$.

$$\frac{}{\Gamma.A_i.\dots A_0 \vdash \mathsf{q}\, \mathsf{p}^i \Rightarrow \mathbf{nbe}(A_i\, \mathsf{p}^{i+1})} \qquad \frac{\Gamma \vdash k \Rightarrow V \qquad \Gamma \vdash \overline{V} = \mathsf{Fun}\, V'\, W \qquad \Gamma \vdash v \Leftarrow V'}{\Gamma \vdash \mathsf{App}\ k\ v \Rightarrow \mathbf{nbe}(W\, (\mathsf{id}, v))}$$

Theorem 4 (Correctness of type-checking).

1. *If $\Gamma \Leftarrow V$, then $\Gamma \vdash V$.*
2. *If $\Gamma \vdash v \Leftarrow V$, then $\Gamma \vdash v : V$.*
3. *If $\Gamma \vdash k \Rightarrow V$, then $\Gamma \vdash k : V$.*

Proof. By simultaneous induction on the type-checking judgement.

In order to prove completeness we define a lexicographic order on pairs of terms and types, in this way we can make induction over the term, and the type.

Definition 15. *Let $v, v' \in Nf$, and $A, A' \in \mathsf{Type}(\Gamma)$, then $(v, A) \prec (v', A')$ is the lexicographic order on $Nf \times \mathsf{Type}(\Gamma)$. The corresponding orders are $v \prec v'$ iff v is an immediate sub-term of v'; and $A \prec^\Gamma A'$, iff $\mathbf{nbe}(A') \equiv \{w\}_{\mathbf{nbe}(A)}$.*

Theorem 5 (Completeness of type-checking).

1. *If $\Gamma \vdash V$, then $\Gamma \Leftarrow V$.*
2. *If $\Gamma \vdash v : A$, then $\Gamma \vdash v \Leftarrow \mathbf{nbe}(A)$.*
3. *If $\Gamma \vdash k : A$, and $\Gamma \vdash k \Rightarrow V'$, then $\Gamma \vdash \overline{\mathbf{nbe}(A)} = \overline{V'}$.*

Proof. By simultaneous induction on V, and well-founded induction on (v, A).

Calculus with Proof-Irrelevance.

Definition 16 (Type-checking and type-inference).

$$\frac{\Gamma \Leftarrow V}{\Gamma \Leftarrow \mathsf{Prf}\, V} \quad \frac{\Gamma \vdash v \Leftarrow V}{\Gamma \vdash [v] \Leftarrow \mathsf{Prf}\, V} \quad \frac{\Gamma \vdash k \Rightarrow \mathsf{Prf}\, V' \qquad \Gamma.V' \vdash v \Leftarrow \mathbf{nbe}(V\, \mathsf{p})}{\Gamma \vdash v\, \mathsf{where}^V\ k \Rightarrow \mathsf{Prf}\, V}$$

Remark 6. Thm. 4 is still valid for the calculus with PRF-TM. Moreover, Thm. 5 is valid if we add the axiom $\Gamma \vdash \mathsf{O} \Leftarrow \mathsf{Prf}\, V$.

Remark 7. Type checking happens always *before* normalisation. If the term to type-check does not contain O, the case $\Gamma \vdash \mathsf{O} \Leftarrow \mathsf{Prf}\, V$ will never be reached—although occurrences of O may be created by normalisation.

Corollary 5. *The type-checking algorithm is correct (by Cor. 1) and complete (by last remark) with respect to the calculus without* PRF-TM*.*

6 Conclusion

The main contributions of the paper are the definition of a correct and complete type-checking algorithm, and the simplification of the NbE algorithm for a calculus with singletons, one universe, and proof-irrelevant types. The type-checker is based on the NbE algorithm which is used to decide equality and to prove the injectivity of the type constructors. We emphasise that the type-checking algorithm is modular with respect to the normalisation algorithm. All the results can be extended to a calculus with annotated lambda abstractions, yielding a type-checking algorithm for terms not necessarily in normal forms.

The full version [5] extends this work by sigma-types and data types and an implementation of the type checker in Haskell.

References

1. Abel, A., Aehlig, K., Dybjer, P.: Normalization by evaluation for Martin-Löf type theory with one universe. In: Fiore, M., ed., Proc. of the 23rd Conf. on the Mathematical Foundations of Programming Semantics (MFPS XXIII), volume 173 of Electr. Notes in Theor. Comp. Sci. Elsevier (2007), 17–39
2. Abel, A., Coquand, T., Dybjer, P.: Normalization by evaluation for Martin-Löf Type Theory with typed equality judgements. In: Proc. of the 22nd IEEE Symp. on Logic in Computer Science (LICS 2007). IEEE Computer Soc. Press (2007), 3–12
3. Abel, A., Coquand, T., Dybjer, P.: On the algebraic foundation of proof assistants for intuitionistic type theory. In: Garrigue, J., Hermenegildo, M. V., eds., Proc. of the 9th Int. Symp. on Functional and Logic Programming, FLOPS 2008, volume 4989 of Lect. Notes in Comput. Sci. Springer-Verlag (2008), 3–13
4. Abel, A., Coquand, T., Dybjer, P.: Verifying a semantic $\beta\eta$-conversion test for Martin-Löf type theory. volume 5133 of Lect. Notes in Comput. Sci. Springer-Verlag (2008), 29–56
5. Abel, A., Coquand, T., Pagano, M.: A modular type-checking algorithm for type theory with singleton types and proof irrelevance (full version) (2009). Available on http://www.tcs.ifi.lmu.de/˜abel/singleton.pdf
6. Abramsky, S., Jung, A.: Handbook of Logic in Computer Science, chapter Domain Theory. Oxford University Press (1994), 1–168
7. Aehlig, K., Joachimski, F.: Operational aspects of untyped normalization by evaluation. Math. Struct. in Comput. Sci. **14** (2004) 587–611
8. Aspinall, D.: Subtyping with singleton types. In: Pacholski, L., Tiuryn, J., eds., Computer Science Logic, 8th Int. Wksh., CSL '94, volume 933 of Lect. Notes in Comput. Sci. Springer-Verlag (1995), 1–15

9. Awodey, S., Bauer, A.: Propositions as [Types]. J. Log. Comput. **14** (2004) 447–471
10. Berger, U., Schwichtenberg, H.: An inverse to the evaluation functional for typed λ-calculus. In: Proc. of the 6th IEEE Symp. on Logic in Computer Science (LICS'91). IEEE Computer Soc. Press (1991), 203–211
11. Bruijn, N. G. d.: Some extensions of Automath : the AUT-4 family (1994)
12. Cartmell, J.: Generalised algebraic theories and contextual categories. Annals of Pure and Applied Logic (1986) 32–209
13. Coquand, T.: An algorithm for type-checking dependent types. Science of Computer Programming **26** (1996) 167–177
14. Coquand, T., Pollack, R., Takeyama, M.: A logical framework with dependently typed records. Fundam. Inform. **65** (2005) 113–134
15. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. The Journal of Symbolic Logic **65** (2000) 525–549
16. Grégoire, B., Leroy, X.: A compiled implementation of strong reduction. In: Proc. of the 7th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP '02), volume 37 of SIGPLAN Notices. ACM Press (2002), 235–246
17. Harper, R., Honsell, F., Plotkin, G.: A Framework for Defining Logics. Journal of the Association of Computing Machinery **40** (1993) 143–184
18. INRIA: The Coq Proof Assistant, Version 8.1. INRIA (2007). http://coq.inria.fr/
19. Lee, D. K., Crary, K., Harper, R.: Towards a mechanized metatheory of Standard ML. In: Hofmann, M., Felleisen, M., eds., Proc. of the 34th ACM Symp. on Principles of Programming Languages, POPL 2007. ACM Press (2007), 173–184
20. Maillard, O.-A.: Proof-irrelevance, strong-normalisation in Type-Theory and PER. Technical report, Chalmers Institute of Technology (2006)
21. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
22. Martin-Löf, P.: Normalization by evaluation and by the method of computability (2004). Talk at JAIST, Japan Advanced Institute of Science and Technology, Kanazawa
23. McBride, C.: Epigram: Practical programming with dependent types. In: Vene, V., Uustalu, T., eds., 5th Int. School on Advanced Functional Programming, AFP 2004, Revised Lectures, volume 3622 of Lect. Notes in Comput. Sci. Springer-Verlag (2005), 130–170
24. Mitchell, J. C., Moggi, E.: Kripke-Style models for typed lambda calculus. In: LICS (1987), 303–314
25. Nordström, B., Petersson, K., Smith, J. M.: Programming in Martin Löf's Type Theory: An Introduction. Clarendon Press, Oxford (1990)
26. Norell, U.: Towards a practical programming language based on dependent type theory. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden (2007)
27. Shankar, N., Owre, S.: Principles and Pragmatics of Subtyping in PVS. In: WADT '99: Selected papers from the 14th International Workshop on Recent Trends in Algebraic Development Techniques. Springer-Verlag, London, UK (2000), 37–52
28. Sozeau, M.: Subset coercions in Coq. In: Altenkirch, T., McBride, C., eds., Types for Proofs and Programs, Int. Wksh., TYPES 2006, volume 4502 of Lect. Notes in Comput. Sci. Springer-Verlag (2007), 237–252
29. Stone, C. A., Harper, R.: Extensional equivalence and singleton types. ACM Trans. Comput. Logic **7** (2006) 676–722
30. Werner, B.: On the strength of proof-irrelevant type theories. Logical Meth. in Comput. Sci. **4** (2008)