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 [28], Coq [20], and Epigram [25] is to decide if a given term has some type; i.e., if a term corresponds to a proof of a proposition [19]. 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 [23, 27] by considering singleton types and proof-irrelevant propositions.

The universe of small types was introduced by Martin-Löf [22] for formalising category theory (in mathematics, universes were introduced by Grothendieck [18] for giving a set-theoretical foundation of category theory[11]); in [23] two different styles of universes are introduced which are called *à la Russell* (the one considered here), and *à la Tarski*.

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, 15]); they were also used to model translucent sums in the formalisation of SML [21]. 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 [33] 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 [12], for reducing the burden in the formalisation of mathematics. As shown by Werner [34], the use of proof-irrelevance types together with sigma types is one way to get subset types à la PVS [30] in type-theories having the eta rule—this direction was explored by Sozeau [31, Sec. 3.3] (for other presentation of subset types in Martin-Löf type-theory see [29]). Berardi conjectured in an unpublished work that (impredicative) type-theory with proof-irrelevance is equivalent to constructive mathematics [9].

Checking dependent types relies on checking types for equality. To this end, we compute η-long normal forms using *normalisation by evaluation* (NbE) [24]. 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. The Haskell programs corresponding to the NbE, and type-checking algorithms are shown in the appendices A, and B, respectively.

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) [13]; 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

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

2.1 Calculus with singleton types

Notation. We use capital Greek letters for variables ranging over Ctx; capital letters from the beginning of the Latin alphabet for variables ranging on $\mathsf{Type}(\Gamma)$; small Greek letters are used for variables ranging on $\Gamma \to \Delta$; and minuscule Latin characters, for variables on $\mathsf{Term}(\Gamma, A)$. Words in sans face denote constants.

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 $S_T = \{\mathsf{U}, \mathsf{Fun}\ \_\ \_, \{\_\}\_\}$.

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

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

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

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

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

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

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

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

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.

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

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

In the last two rules we have a choice on how to express them, we could replace them with the rules

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

The rule SING-EQ-EL is essential; in fact, since we have eta-expansion for singletons, we would like to derive

$$\Gamma.\{\lambda t\}_{\mathsf{Fun}\, A\, B} \vdash \mathsf{App}\ \mathsf{q}\ a = t\, (\mathsf{id}, a) : A$$

from $\Gamma.\{\lambda t\}_{\mathsf{Fun}\, A\, B} \vdash \mathsf{q} = \lambda t : \{\lambda t\}_{\mathsf{Fun}\, A\, B}$, and $\Gamma \vdash a : A$. Which would be impossible if SING-EQ-EL were not a rule.

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 write $\mathcal{D} :: \mathcal{J}$, to denote that $\mathcal{D}$ is a derivation with conclusion $\mathcal{J}$. 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 $\{a\}_A$.

Remark 1. Note that if $\Delta \vdash \mathsf{p}^i : \Gamma$, and $\Gamma \vdash \mathsf{p}^j : \Theta$, then $\Delta \vdash \mathsf{p}^{i+j} : \Theta$.

Definition 1 (Neutral terms, and normal forms).

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

An advantage of introducing the calculus as a GAT is that we can derive several syntactical results from the meta-theory of GATs.

Remark 2 (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 3 (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 $\Gamma \vdash k : \mathsf{U}$.*

Because of the presence of singletons the proof of inversion of the typing rule is more involved than usual. We first show a weaker version of the inversion lemma; and then we prove the usual one.

Let $\mathcal{D}$ be a derivation, then $P(\mathcal{D})$ denotes the predicate "if $\mathcal{D} :: \Gamma \vdash t : A$, then there exists a derivation $\mathcal{D}' :: \Gamma \vdash A = \{a\}_{A'}$, and three sub-derivations of $\mathcal{D}$, such that $\mathcal{D}^1 :: \Gamma \vdash a : A'$, $\mathcal{D}^2 :: \Gamma \vdash t : A'$, and $\mathcal{D}^3 :: \Gamma \vdash a = t : A'$, respectively".

Lemma 2 (Inversion of terms).

1. *If $\mathcal{D} :: \Gamma \vdash \mathsf{Fun}\, A'\, B' : A$, then*
   (a) *either there exist a derivation $\mathcal{D}' :: \Gamma \vdash A = \mathsf{U}$, and two sub-derivations of $\mathcal{D}$, such that $\mathcal{D}^1 :: \Gamma \vdash A' : \mathsf{U}$, and also $\mathcal{D}^2 :: \Gamma.A' \vdash B' : \mathsf{U}$;*
   (b) *or $P(\mathcal{D})$.*
2. *If $\mathcal{D} :: \Gamma \vdash \{b\}_B : A$, then*
   (a) *either there exist a derivation $\mathcal{D}' :: \Gamma \vdash A = \mathsf{U}$, and two sub-derivations of $\mathcal{D}$, such that $\mathcal{D}^1 :: \Gamma \vdash B : \mathsf{U}$, and also $\mathcal{D}^2 :: \Gamma \vdash b : B$;*

   (b) *or* $P(\mathcal{D})$.
3. *If* $\mathcal{D} :: \Gamma \vdash \lambda t : A$, *then*
   (a) *either there exist a derivation* $\mathcal{D}' :: \Gamma \vdash A = \mathsf{Fun}\, A'\, B'$, *and a sub-derivations of* $\mathcal{D}$, *such that* $\mathcal{D}^1 :: \Gamma.A' \vdash t : B'$;
   (b) *or* $P(\mathcal{D})$.
4. *If* $\mathcal{D} :: \Gamma \vdash \mathsf{App}\ t\ r : A$, *then*
   (a) *either there exist a derivation* $\mathcal{D}' :: \Gamma \vdash A = B'\,(\mathsf{id}, r)$, *and sub-derivations of* $\mathcal{D}$, *such that* $\mathcal{D}^1 :: \Gamma \vdash t : \mathsf{Fun}\, A'\, B'$, *and* $\mathcal{D}^2 :: \Gamma \vdash r : A'$;
   (b) *or* $P(\mathcal{D})$.

Proof. (By induction on $\mathcal{D}$.) We show the proof only for $\mathcal{D} :: \Gamma \vdash \mathsf{Fun}\, A\, B : C$, for the other cases are very similar.

We should consider only three cases FUN-U-I, SING-I, and CONV.

1. It is immediate for FUN-U-I, we prove the first disjunct; the only missing derivation is obtained by reflexivity.
2. It is also trivial for SING-I.
3. The most tedious case is CONV. We have a derivation $\mathcal{D}$

$$\frac{\mathcal{D}_1 :: \Gamma \vdash \mathsf{Fun}\, A\, B : C \qquad \mathcal{D}_2 :: \Gamma \vdash C = C'}{\Gamma \vdash \mathsf{Fun}\, A\, B : C'}$$

   We can apply the inductive hypothesis on $\mathcal{D}_1$; from that we know
   (a) either there are sub-derivations of $\mathcal{D}_1$, such that $\mathcal{D}_1^1 :: \Gamma \vdash A : \mathsf{U}$, and $\mathcal{D}_1^2 :: \Gamma.A \vdash B : \mathsf{U}$, and a derivation $\mathcal{D}* :: \Gamma \vdash C = \mathsf{U}$;
   (b) or there are a derivation $\mathcal{D}* :: \Gamma \vdash C = \{a\}_{A'}$, and three sub-derivations of $\mathcal{D}_1$, such that $\mathcal{D}_1^1 :: \Gamma \vdash a : A'$, and $\mathcal{D}_1^2 :: \Gamma \vdash \mathsf{Fun}\, A\, B : A'$, and $\mathcal{D}_1^3 :: \Gamma \vdash a = \mathsf{Fun}\, A\, B : A'$.
   In either case $\mathcal{D}_1^i$ is a sub-derivation of $\mathcal{D}$. And from $\mathcal{D}*$ and $\mathcal{D}_2$, we can conclude by symmetry and transitivity $\Gamma \vdash C' = \mathsf{U}$, or $\Gamma \vdash C' = \{a\}_{A'}$, respectively.

From the last lemma we can conclude, using the fact that derivations are well-founded, the stronger inversion lemma.

Corollary 1 (Inversion of typing).

1. *If* $\mathcal{D} :: \Gamma \vdash \mathsf{Fun}\, A'\, B' : A$, *then there exist two sub-derivations of* $\mathcal{D}$, *such that* $\mathcal{D}^1 :: \Gamma \vdash A' : \mathsf{U}$, *and also* $\mathcal{D}^2 :: \Gamma.A' \vdash B' : \mathsf{U}$;
2. *If* $\mathcal{D} :: \Gamma \vdash \{b\}_B : A$, *then there exist two sub-derivations of* $\mathcal{D}$, *such that* $\mathcal{D}^1 :: \Gamma \vdash B : \mathsf{U}$, *and also* $\mathcal{D}^2 :: \Gamma \vdash b : B$;
3. *If* $\mathcal{D} :: \Gamma \vdash \lambda t : A$, *then there exists a sub-derivation of* $\mathcal{D}$, *such that* $\mathcal{D}^1 :: \Gamma.A' \vdash t : B'$.

Proof. First we note that, albeit not formally done in this work, derivations are defined inductively. Hence the induced order on derivations ($\mathcal{D}' < \mathcal{D}$ if and only if $\mathcal{D}'$ is a sub-derivation of $\mathcal{D}$) is well-founded.

We make the reasoning for the first clause; all the rest are similar. Let $\mathcal{D} :: \Gamma \vdash \mathsf{Fun}\, A\, B : C$

Suppose 1 is false, hence by lemma 2, there exists a sub-derivation $\mathcal{D}' :: \Gamma \vdash \mathsf{Fun}\, A\, B : C'$. And we can continue ad-infinitum, hence the order on derivations is not well-founded.

Remark 4 (Inversion of substitution). It is clear that any substitution $\Delta \vdash \sigma : \Gamma.A$ is equal to some substitution $\Delta \vdash (\sigma', t) : \Gamma.A$

Calculus with Proof-Irrelevance.

In this section we keep the basic rules, and introduce types for natural numbers, enumeration sets, and proof-irrelevant types. The main difference with other presentations [27, 23], in the syntactic level, is that the eliminator operator (for each type) has as an argument the type of the result. The presence of the resulting type in the eliminator is needed in order to define the normalisation function; it is also necessary for the type-inference algorithm.

The lost of decidability of type-checking is because we should decide the inhabitation of a type in order to check that O has some type $\mathsf{Prf}\ A$[1].

Introductory rules.

$$\frac{\Gamma \in \mathsf{Ctx} \qquad A \in \mathsf{Type}(\Gamma) \qquad B \in \mathsf{Type}(\Gamma.A)}{\Sigma\, A\, B \in \mathsf{Type}(\Gamma)}\ (\textsc{sum-i})$$

$$\frac{t \in \mathsf{Term}(\Gamma, A) \qquad b \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, t))}{(t, b) \in \mathsf{Term}(\Gamma, \Sigma\, A\, B)}\ (\textsc{sum-in})$$

$$\frac{t \in \mathsf{Term}(\Gamma, \Sigma\, A\, B)}{\mathsf{fst}\ t \in \mathsf{Term}(\Gamma, A)}\ (\textsc{sum-el}1) \qquad \frac{t \in \mathsf{Term}(\Gamma, \Sigma\, A\, B)}{\mathsf{snd}\ t \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, \mathsf{fst}\ t))}\ (\textsc{sum-el}2)$$

$$\frac{B \in \mathsf{Term}(\Gamma, \mathsf{U}) \qquad B \in \mathsf{Term}(\Gamma.A, \mathsf{U})}{\Sigma\, A\, B \in \mathsf{Term}(\Gamma, \mathsf{U})}\ (\textsc{sum-u-i})$$

$$\frac{\Gamma \in \mathsf{Ctx}}{\mathsf{Nat} \in \mathsf{Type}(\Gamma)}\ (\textsc{nat-i}) \qquad \frac{\Gamma \in \mathsf{Ctx}}{\mathsf{Nat} \in \mathsf{Term}(\Gamma, \mathsf{U})}\ (\textsc{nat-u-i})$$

$$\frac{\Gamma \in \mathsf{Ctx}}{\mathsf{zero} \in \mathsf{Term}(\Gamma, \mathsf{Nat})}\ (\textsc{nat-z-i}) \qquad \frac{\Gamma \in \mathsf{Ctx} \qquad t \in \mathsf{Term}(\Gamma, \mathsf{Nat})}{\mathsf{suc}(t) \in \mathsf{Term}(\Gamma, \mathsf{Nat})}\ (\textsc{nat-s-i})$$

$$\frac{\begin{array}{c} B \in \mathsf{Type}(\Gamma.\mathsf{Nat}) \qquad t \in \mathsf{Term}(\Gamma, \mathsf{Nat}) \\ z \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, \mathsf{zero})) \qquad s \in \mathsf{Term}(\Gamma, \mathsf{Rec}(B)) \end{array}}{\mathsf{natrec}(B, z, s, t) \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, t))}\ (\textsc{nat-el})$$

$$\frac{\Gamma \in \mathsf{Ctx}}{\mathsf{N}_n \in \mathsf{Type}(\Gamma)}\ (\textsc{N}_n\textsc{-f}) \qquad \frac{\Gamma \in \mathsf{Ctx}}{\mathsf{N}_n \in \mathsf{Term}(\Gamma, \mathsf{U})}\ (\textsc{N}_n\textsc{-u-i})$$

$$\frac{\Gamma \in \mathsf{Ctx} \qquad i < n}{\mathsf{c}_i^n \in \mathsf{Term}(\Gamma, \mathsf{N}_n)}\ (\textsc{N}_n\textsc{-i})$$

$$\frac{\begin{array}{c} B \in \mathsf{Type}(\Gamma.\mathsf{N}_n) \qquad t_0 \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, \mathsf{c}_0^n)) \cdots \\ t_{n-1} \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, \mathsf{c}_{n-1}^n)) \qquad t \in \mathsf{Term}(\Gamma, \mathsf{N}_n) \end{array}}{\mathsf{elim}^n(B, t_0, \ldots, t_{n-1}, t) \in \mathsf{Term}(\Gamma, B\,(\mathsf{id}, t))}\ (\textsc{N}_n\textsc{-e})$$

[1] Another possibility is to consider type-checking as an interactive process. For example, we could use the same approach as in [32], and introduce proof-obligations when type-checking O.

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

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

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

Axioms.

$$\begin{aligned}
\Sigma\, A\, B\, \sigma &= \Sigma\, (A\,\sigma)\, (B\, (\sigma\, \mathsf{p}, \mathsf{q})) \\
\mathsf{Nat}\, \sigma &= \mathsf{Nat} \\
\mathsf{N}_i\, \sigma &= \mathsf{N}_i \\
\mathsf{fst}\ t\, \sigma &= \mathsf{fst}\ t\, \sigma \\
\mathsf{snd}\ t\, \sigma &= \mathsf{snd}\ t\, \sigma \\
(t, b)\, \sigma &= (t\, \sigma, b\, \sigma) \\
\mathsf{zero}\, \delta &= \mathsf{zero} \\
\mathsf{suc}(t)\, \delta &= \mathsf{suc}(t\, \delta) \\
\mathsf{natrec}(B, z, s, t)\, \delta &= \mathsf{natrec}(B\, (\delta\, \mathsf{p}, \mathsf{q}), z\, \delta, s\, \delta, t\, \delta) \\
(\mathsf{Prf}\ A)\, \delta &= \mathsf{Prf}\ (A\, \delta) \qquad\qquad [t]\, \delta = [t\, \delta] \qquad\qquad \mathsf{O}\, \delta = \mathsf{O} \\
(b\ \mathsf{where}^{B}\ t)\, \delta &= b\, (\delta\, \mathsf{p}, \mathsf{q})\ \mathsf{where}^{B\, \delta}\ (t\, \delta) \qquad\qquad \mathsf{c}_i^n\, \delta = \mathsf{c}_i^n \\
\mathsf{elim}^n(B, t_0, \ldots, t_{n-1}, t)\, \delta &= \mathsf{elim}^n(B\, \delta, t_0\, \delta, \ldots, t_{n-1}\, \delta, t\, \delta) \\
\mathsf{fst}\ (t, b) &= t \\
\mathsf{snd}\ (t, b) &= b \\
(\mathsf{fst}\ t, \mathsf{snd}\ t) &= t \\
\mathsf{natrec}(B, z, s, \mathsf{zero}) &= z \\
\mathsf{natrec}(B, z, s, \mathsf{suc}(t)) &= \mathsf{App}\ (\mathsf{App}\ s\ t)\ \mathsf{natrec}(B, z, s, t) \\
\mathsf{elim}^n(B, t_0, \ldots, t_{n-1}, \mathsf{c}_i^n) &= t_i \\
b\ \mathsf{where}^{B}\ [t] &= [b\, (\mathsf{id}, t)]
\end{aligned}$$

Notation In the sequel we will be (more) flexible with the notation, and we will omit the indices in c_i, and in $\mathsf{elim}(B, \boldsymbol{t}, r)$.

Lemma 3 (Inversion).

1. *If $\Gamma \vdash \Sigma\, A\, B$, then $\Gamma \vdash A$, and $\Gamma.A \vdash B$.*
2. *If $\Gamma \vdash \Sigma\, A'\, B : A$, then $\Gamma \vdash A = \mathsf{U}$, and $\Gamma \vdash A : \mathsf{U}$, and $\Gamma.A \vdash B : \mathsf{U}$;*
3. *If $\Gamma \vdash (t, b) : A$, then $\Gamma \vdash A = \Sigma\, A'\, B$, and $\Gamma \vdash t : A'$, and $\Gamma \vdash b : B\, (\mathsf{id}, t)$;*
4. *If $\Gamma \vdash \mathsf{fst}\ t : A$, then $\Gamma \vdash A = A'$, and $\Gamma \vdash t : \Sigma\, A'\, B$, for some A', and B;*
5. *If $\Gamma \vdash \mathsf{snd}\ t : B$, then $\Gamma \vdash B = B'\, (\mathsf{id}, \mathsf{fst}\ t)$, and $\Gamma \vdash t : \Sigma\, A\, B'$, for some A, and B';*

6. *If* $\Gamma \vdash \mathsf{Nat} : A$, *then* $\Gamma \vdash A = \mathsf{U}$.
7. *If* $\Gamma \vdash \mathsf{zero} : A$, *then* $\Gamma \vdash A = \mathsf{Nat}$;
8. *If* $\Gamma \vdash \mathsf{suc}(t) : A$, *then* $\Gamma \vdash t : \mathsf{Nat}$, *and* $\Gamma \vdash A = \mathsf{Nat}$;
9. *If* $\Gamma \vdash \mathsf{natrec}(B, z, s, t) : A$, *then* $\Gamma.\mathsf{Nat} \vdash B$, $\Gamma \vdash z : B\,(\mathsf{id}, \mathsf{zero})$, $\Gamma \vdash s : \mathsf{Rec}(B)$, *and* $\Gamma \vdash t : \mathsf{Nat}$, *and* $\Gamma \vdash A = B\,(\mathsf{id}, t)$;
10. *if* $\Gamma \vdash \mathsf{c}_i^n : A$, *then* $\Gamma \vdash A = \mathsf{N}_n$;
11. *If* $\Gamma \vdash \mathsf{elim}(B, \boldsymbol{t}, t') : A$, *then* $\Gamma.\mathsf{N}_n \vdash B$, $\Gamma \vdash t_i : B\,(\mathsf{id}, \mathsf{c}_i)$, *and* $\Gamma \vdash t' : \mathsf{N}_n$, *and* $\Gamma \vdash A = B\,(\mathsf{id}, t)$.
12. *If* $\Gamma \vdash [t] : A$, *then* $\Gamma \vdash A = \mathsf{Prf}\ A'$ *and* $\Gamma \vdash t' : A'$.
13. *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:

$$\begin{aligned} Ne \ni k ::= \ldots \mid \mathsf{fst}\ k \mid \mathsf{snd}\ k \mid \mathsf{natrec}(V, v, v, k) \mid \mathsf{O} \\ \mid v\ \mathsf{where}^V\ k \mid \mathsf{elim}^n(V, v_0, \ldots, v_{n-1}, k) \\ Nf \ni v, V ::= \ldots \mid \Sigma\, V\, W \mid \mathsf{Nat} \mid \mathsf{N}_n \mid \mathsf{Prf}\ V \mid (v, v') \mid \\ \mathsf{zero} \mid \mathsf{suc}(v) \mid [v] \mid \mathsf{O} \mid \mathsf{c}_i^n \end{aligned}$$

The problem of losing decidability of type-checking for the calculus with the rule PRF-TM does not affect the usability of a system based on a calculus without it. In fact we can consider two calculi, one which is used internally by the system, and one another on which the user writes proofs.

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.*

For example, $\Gamma' \vdash t' : A'$ is a good-lifting of $\Gamma \vdash t : A$, if Γ' is a good-lifting of Γ, and A' is a lifting of A, such that $\Gamma \vdash A = A'$, and t' is a lifting of t, such that $\Gamma \vdash t = t' : A$.

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* $\Gamma' \vdash J' = J''$.

Proof. By induction on $\Gamma \vdash J$, in each rule we use i.h., and build up the corresponding entity to the good-lifting for each part of the judgement; then, given any other good-lifting of the whole judgement, we do inversion on the definition of good-lifting, and get the equalities for each part; we finish using congruence for showing that both good-lifting are judgemental equal.

We show the case for PRF-TM. Let

$$\Gamma \vdash \mathsf{O} : \mathsf{Prf}\ A, \tag{1}$$

then we know $\Gamma \vdash t : A$; hence, by i.h. there is a good-lifting $\Gamma' \vdash t' : A'$, and we can conclude $\Gamma' \vdash [t'] : \mathsf{Prf}\ A'$, which is a good-lifting of 1. Now suppose $\Gamma'' \vdash s : B$ is another good-lifting of 1, then, by inversion, we know Γ'' is a good-lifting of Γ, B is a good-lifting of $\mathsf{Prf}\ A$, hence $B \equiv \mathsf{Prf}\ A''$, and A'' is a good-lifting of A; moreover we know s is a good-lifting of O, hence $s \equiv [t'']$; therefore we can conclude $\Gamma' \vdash [t'] = [t''] : \mathsf{Prf}\ A'$, by PRF-EQ, after using conversion.

Corollary 2. *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 $D = \mathbb{O} \oplus X_\perp \oplus [D \to D] \oplus D \times D \oplus D \times D \oplus \mathbb{O} \oplus D \times [D \to D] \oplus D \times D$, where X is a denumerable set of variables (as usual we write x_i and assume $x_i \neq x_j$ if $i \neq j$, for $i, j \in \mathbb{N}$), $E_\perp = E \cup \{\perp\}$ is lifting, $\mathbb{O} = \{\top\}_\perp$ is the Sierpinski space, $[D \to D]$ is the set of continuous functions from D to D, $\oplus$ is the coalesced sum (this is the disjoint union where all the bottoms elements are identified), and $D \times D$ is the Cartesian product of D [6]. The injections are:*

$$\begin{aligned}
\iota_{\mathsf{O}} &: \mathbb{O} \to D & \iota_{\mathsf{Pair}} &: D \times D \to D \\
\iota_{\mathsf{Var}} &: X \to D & \iota_{\mathsf{U}} &: \mathbb{O} \to D \\
\iota_{\mathsf{Lam}} &: [D \to D] \to D & \iota_{\mathsf{Fun}} &: D \times [D \to D] \to D \\
\iota_{\mathsf{App}} &: D \times D \to D & \iota_{\mathsf{Sing}} &: D \times D \to D
\end{aligned}$$

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

$\top$	(d, d')	for $d, d' \in D$
$\mathsf{Var}\, x_i$	U	for $x_i \in X$
$\mathsf{Lam}\, f$	$\mathsf{Fun}\, d\, f$	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

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

We define a partial function $\mathsf{R}\_\_ : \mathbb{N} \to D \to \mathit{Terms}$ which reifies elements from the model into terms; this function is similar to the read-back function of Gregoire and Leroy's [17].

Definition 4 (Read-back function).

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

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 $(d, d) \in R$. We define $dom(R) = \{d \in D \mid (d, d) \in \mathcal{R}\}$; clearly, R is an equivalence relation over $dom(\mathcal{R})$. If $(d, d') \in R$, sometimes we will write $d = d' \in \mathcal{R}$, and $d \in \mathcal{R}$ if $d \in dom(\mathcal{R})$. We denote with $\mathrm{PER}(D)$ the set of all PERs over D.

If $\mathcal{R} \in \mathrm{PER}(D)$ and $\mathcal{F} : dom(\mathcal{R}) \to \mathrm{PER}(D)$, 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 $\mathcal{F} : \mathcal{R} \to \mathrm{PER}(D)$.

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

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

Remark 5. These are clearly PERs over D: symmetry is trivial and transitivity follows from transitivity of the syntactical equality.

The following definitions are standard [8, 15] (except for **1**); they will be used in the definition of the model.

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

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

We define $\mathcal{U}, \mathcal{T} \in \mathrm{PER}(D)$ and $[\_] : dom(\mathcal{T}) \to \mathrm{PER}(D)$ using Dybjer's schema of inductive-recursive definition [16]. We show then that $[\_]$ is a family of PERs over D.

Definition 6 (PER model).

- *Inductive definition of $\mathcal{U} \in PER(D)$.*
  - $\mathsf{Ne} \subseteq \mathcal{U}$,
  - *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* $[\_] \in dom(\mathcal{T}) \to PER(D)$.
- $[\mathsf{U}] = \mathcal{U}$,
- $[\mathsf{Sing}\, d\, X] = \{\!\{d\}\!\}_{[X]}$,
- $[\mathsf{Fun}\, X\, F] = \prod [X]\, (d \mapsto [F\ d])$,
- $[d] = \mathsf{Ne}$, *in all other cases.*

Remark 6. The generation order $\sqsubset$ on $\mathcal{T}$ is well-founded. The minimal elements are U, and elements in Ne; $X \sqsubset \mathsf{Fun}\, X\, F$, and for all $d \in [X]$, $F\ d \sqsubset \mathsf{Fun}\, X\, F$; and, finally, $X \sqsubset \mathsf{Sing}\, d\, X$.

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

Proof. We prove, by induction on $X = X' \in \mathcal{T}$, that $[X] = [X']$.

– It is trivial for $d = d' \in \mathsf{Ne}$.
– It is also trivial for $\mathsf{U} = \mathsf{U}$.
– Let $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X' \in \mathcal{U}$, show $\{\!\{d\}\!\}_X = \{\!\{d'\}\!\}_{X'}$. By i.h. we have $[X] = [X']$, hence $d = d' \in [X]$. Now let $e = d \in [X]$ and $e' = d \in [X]$, by transitivity we have $e = d' \in [X]$ and $e' = d' \in [X]$; therefore $\{\!\{d\}\!\}_X = \{\!\{d'\}\!\}_{X'}$. Note that this arguments is the same for $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X' \in \mathcal{T}$.
– Let $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{U}$. By i.h. we know $[X] = [X']$, and by definition we have $F\ d = F'\ d$, for all $d \in dom([X])$; hence if $f \cdot d = f' \cdot d' \in F\ d$, for all $d = d' \in [X]$, then also $f \cdot d = f' \cdot d' \in F'\ d$. As in the last case this reasoning applies for $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F' \in \mathcal{T}$.

Remark 7. If $d = d' \in \mathsf{Nf}$ and $X = X' \in \mathsf{Nf}$, then $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X' \in \mathsf{Nf}$. Also, if $d = d' \in \mathsf{Ne}$ and $e = e' \in \mathsf{Nf}$, then $\mathsf{App}\, d\, e = \mathsf{App}\, d'\, e' \in \mathsf{Ne}$.

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:*

$$\begin{aligned}
\uparrow_{\mathsf{Fun}\ X\ F} d &= \mathsf{Lam}\,(e \mapsto \uparrow_{F\ e} \mathsf{App}\, d \downarrow_X e)\\
\uparrow_{\mathsf{Sing}\ d\ X} e &= d\\
\uparrow_{\mathsf{U}} d &= d\\
\uparrow_{d} e &= e, \textit{ for all } d \in \mathsf{Ne}
\end{aligned}$$

$$\begin{aligned}
\downarrow_{\mathsf{Fun}\ X\ F} d &= \mathsf{Lam}\,(e \mapsto \downarrow_{F\ \uparrow_X e} d \cdot \uparrow_X e)\\
\downarrow_{\mathsf{Sing}\ d\ X} e &= \downarrow_X d\\
\downarrow_{\mathsf{U}} d &= \Downarrow d\\
\downarrow_{d} e &= e, \textit{ for all } d \in \mathsf{Ne}
\end{aligned}$$

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

We will show that $\uparrow_X : \mathsf{Ne} \to [X]$ and $\downarrow_X : [X] \to \mathsf{Nf}$. As usual, we need to make it stronger and get that as a corollary of the following lemma.

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

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

Proof. (By induction on $X = X' \in \mathcal{T}$)

- $\mathsf{U} = \mathsf{U}$: the first and third parts are clearly seen to hold. The second part is shown by induction on $d = d' \in \mathcal{U}$, and is proved by the third point in the following three cases.
- $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X'$: by the rule we have $d = d' \in [X]$, and $X = X' \in \mathcal{T}$; hence $[X] = [X']$. The first part follows directly from those facts. The second part is by i.h. on $d = d' \in [X]$. The third part follows from the i.h. and from Rem. 7.
- $d = d' \in \mathsf{Ne}$: trivial.
- $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F'$:
  - let $k = k' \in \mathsf{Ne}$, for showing the first part we need to take $d = d' \in [X]$ and show $\uparrow_{\mathsf{Fun}\ X\ F} \mathsf{App}\, k \downarrow_X d = \uparrow_{\mathsf{Fun}\ X'\ F'} \mathsf{App}\, k' \downarrow_{X'} d' \in [F\ d]$. By i.h. on X we have $\downarrow_X d = \downarrow_{X'} d' \in \mathsf{Nf}$, hence by Rem. 7 and i.h. on $F\ d$, we have $\uparrow_{F\ d} \mathsf{App}\, k \downarrow_X d = \uparrow_{F'\ d'} \mathsf{App}\, k' \downarrow_{X'} d' \in [F\ d]$.

- Let $f = f' \in [\mathsf{Fun}\, X\, F]$, for showing $\downarrow_{\mathsf{Fun}\, X\, F} f = \downarrow_{\mathsf{Fun}\, X'\, F'} f' \in \mathsf{Nf}$ we should prove that $\mathsf{R}_i\,(\downarrow_{\mathsf{Fun}\, X\, F} f)$ and $\mathsf{R}_i\,(\downarrow_{\mathsf{Fun}\, X\, F'} f')$ are both defined and syntactically equal normal forms. From Rem. 7 and by i.h. on X we have $\uparrow_X \mathsf{Var}\, x_j = \uparrow_X \mathsf{Var}\, x_j \in [X]$ and then $f \cdot \mathsf{Var}\, x_j = f' \cdot \mathsf{Var}\, x_j \in [F\ \uparrow_X \mathsf{Var}\, x_j]$, therefore we conclude (by i.h. on the second part) $\downarrow_{F\ \uparrow_X \mathsf{Var}\, x_j} f \cdot \mathsf{Var}\, x_j = \downarrow_{F'\ \uparrow_X \mathsf{Var}\, x_j} f' \cdot \mathsf{Var}\, x_j \in \mathsf{Nf}$. So, by definition of Nf they are syntactically equal once read back, finally we end using congruence of equality under λ.
- let's show $\Downarrow \mathsf{Fun}\, X\, F = \Downarrow \mathsf{Fun}\, X'\, F' \in \mathsf{Nf}$. By definition we have $\Downarrow \mathsf{Fun}\, X\, F = \mathsf{Fun} \Downarrow X\, e \mapsto \Downarrow(F\ \uparrow_X e)$. By i.h. on X, we have $\Downarrow X = \Downarrow X' \in \mathsf{Nf}$. Moreover if $k = k' \in \mathsf{Ne}$, $\Downarrow F\ \uparrow_X k = \Downarrow F'\ \uparrow_{X'} k' \in \mathsf{Nf}$ (this is again by i.h. and by definition of $\mathcal{T}$). Now we need to show that $\mathsf{R}_i\,(\mathsf{Fun}\, X\, F)$ and $\mathsf{R}_i\,(\mathsf{Fun}\, X'\, F')$ are both defined, and they are syntactically equal in $\mathcal{T}$. We can conclude that by the fact that the syntactically equality is a congruence for $\mathsf{Fun}\ \_\ \_$; the only thing that remains to be proved is that $\mathsf{R}_{i+1}\, F(\uparrow_X \mathsf{Var}\, x_i)$ and $\mathsf{R}_{i+1}\, F'(\uparrow_{X'} \mathsf{Var}\, x_i)$ are both syntactically equal, which follows from $\Downarrow F\ \uparrow_X k = \Downarrow F'\ \uparrow_{X'} k' \in \mathsf{Nf}$ instantiating k and k' with $\mathsf{Var}\, x_i$.

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). *We define inductively the notion of validity of judgements in a model.*

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* $\downarrow_{[\![A]\!]} [\![t]\!] = \downarrow_{[\![A]\!]} [\![t']\!] \in \mathsf{Nf}$.

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.

$$\begin{aligned} D = \ldots &\oplus D \times [D \to D] \oplus 2 \times D \\ &\oplus \mathbb{O} \oplus \mathbb{O} \oplus D \oplus [D \to D] \times D \times [D \to [D \to D]] \times D \\ &\oplus D \oplus \mathbb{O} \oplus \mathbb{N} \oplus \mathbb{N} \times \mathbb{N} \oplus \mathbb{N} \times [D \to D] \times D^* \times D \ ; \end{aligned}$$

the corresponding injections are

$$\begin{aligned} \iota_{Sum} &: D \times [D \to D] \to D & \iota_{Proj} &: 2 \times D \to D \\ \iota_{Nat} &: \mathbb{O} \to D & \iota_{Zero} &: \mathbb{O} \to D \\ \iota_{Suc} &: D \to D & \iota_{Rec} &: [D \to D] \times D \times [D \to [D \to D]] \times D \to D \\ \iota_{\mathsf{Prf}} &: D \to D & \iota_{\star} &: \mathbb{O} \to D \\ \iota_{\mathsf{N}} &: \mathbb{N} \to D & \iota_{\mathsf{c}} &: \mathbb{N} \times \mathbb{N} \to D \\ \iota_{\mathsf{elim}} &\mathbb{N} \times [D \to D] \times D^* \times D \to D \end{aligned}$$

As before we use a more convenient notation for the injections, let $F, f \in [D \to D]$, $d, d' \in D$, $g \in [D \to D] \times D \times [D \to [D \to D]] \times D$, $n, i \in \mathbb{N}$, and $\boldsymbol{d} \in D^*$.

$$\begin{aligned} &\mathsf{Sum}(d, f) & &\mathsf{Proj}_{\{1,2\}} d \\ &\mathsf{zero} & &\mathsf{Nat} \\ &\mathsf{suc}(d) & &\mathsf{nrec}(F, d, g, d') \\ &\star & &\mathsf{Prf}(d) \\ &\mathsf{N}_n & &\mathsf{c}_i^n \\ &\mathsf{elim}^n(F, \boldsymbol{d}, d') \end{aligned}$$

In preparation for giving the semantics of the new constructs, we define a new extension for functions defined only on pairs to any element of D. If $g : D \times D \to D$, then $g^- : 2 \to D \to D$, as follows:

$$g^i d = \begin{cases} g\ (e, e') & \text{if } d = (e, e') \\ \mathsf{Proj}_i(d) & \text{if } d \in \mathsf{Ne} \\ \bot & \text{otherwise} \end{cases}$$

We will use this extension for defining the semantics of fst and snd :

$$\begin{aligned} \mathsf{p}^* &= \pi_1^1 \\ \mathsf{q}^* &= \pi_2^2 \end{aligned}$$

Definition 10.

$$\begin{aligned} \mathsf{R}_j\, \mathsf{Sum}(X, F) &= \Sigma\, (\mathsf{R}_j\, X)\, (\mathsf{R}_{j+1}\, F\ \mathsf{Var}\, x_j) \\ \mathsf{R}_j\, (d, d') &= (\mathsf{R}_j\, d, \mathsf{R}_j\, d') \\ \mathsf{R}_j\, \mathsf{Proj}_1(d) &= \mathsf{fst}\ \mathsf{R}_j\, d \\ \mathsf{R}_j\, \mathsf{Proj}_2(d) &= \mathsf{snd}\ \mathsf{R}_j\, d \\ \mathsf{R}_j\, \mathsf{Nat} &= \mathsf{Nat} \\ \mathsf{R}_j\, \mathsf{zero} &= \mathsf{zero} \\ \mathsf{R}_j\, \mathsf{suc}(d) &= \mathsf{suc}(\mathsf{R}_j\, d) \\ \mathsf{R}_j\, \mathsf{nrec}(F, d, f, e) &= \mathsf{natrec}(\mathsf{R}_{j+1}\, F\ \mathsf{Var}\, x_j, \mathsf{R}_j\, d, \mathsf{R}_j\, f, \mathsf{R}_j\, e) \\ \mathsf{R}_j\, \mathsf{Prf}(d) &= \mathsf{Prf}\ \mathsf{R}_j\, d \\ \mathsf{R}_j \star &= \mathsf{O} \\ \mathsf{R}_j\, \mathsf{N}_n &= \mathsf{N}_n \\ \mathsf{R}_j\, \mathsf{c}_i^n &= \mathsf{c}_i^n \qquad \textit{if } i < n \\ \mathsf{R}_j\, \mathsf{elim}^n(F, \langle d_0, \ldots, d_{n-1} \rangle, e) &= \mathsf{elim}^n(\mathsf{R}_{j+1}\, F\ \mathsf{Var}\, x_j, \mathsf{R}_j\, d_0, \ldots, \mathsf{R}_j\, d_{n-1}, \mathsf{R}_j\, e) \end{aligned}$$

Note that if $d = d' \in \mathsf{Ne}$, $e = e' \in \mathsf{Nf}$, $f = f' \in \mathsf{Nf}$, and $F = F' \in \mathsf{Nf}$, then $\mathsf{nrec}(F, d, f, e) = \mathsf{nrec}(F', d', f', e') \in \mathsf{Ne}$.

We define inductively a new PER for interpreting natural numbers:

Definition 11 (PER for natural numbers).

- $\mathsf{Ne} \subset \mathcal{N}$
- $\mathsf{zero} = \mathsf{zero} \in \mathcal{N}$
- $\mathsf{suc}(d) = \mathsf{suc}(d') \in \mathcal{N}$, *if* $d = d' \in \mathcal{N}$

Definition 12. *Let* $n \in \mathbb{N}$, *then* $\mathsf{Const}_n = \{(\mathsf{c}_i^n, \mathsf{c}_i^n) \mid i < n\} \cup \mathsf{Ne}$.

Remark 8. The operator $\overline{X}$ is a closure operator; i.e.,

1. it is extensive, $X \subseteq \overline{X}$;
2. it is monotonic, if $X \subseteq X'$, then $\overline{X} \subseteq \overline{X'}$; and
3. it is idempotent, $\overline{\overline{X}} = \overline{X}$.

Definition 13.

- *If* $X = X' \in \mathcal{U}$, *and for all* $d = d' \in [X]$, $F\ d = F'\ d' \in \mathcal{U}$, *then* $\mathsf{Sum}(X, F) = \mathsf{Sum}(X', F') \in \mathcal{U}$.
- *If* $X = X' \in \mathcal{T}$, *and for all* $d = d' \in [X]$, $F\ d = F'\ d' \in \mathcal{T}$, *then* $\mathsf{Sum}(X, F) = \mathsf{Sum}(X', F') \in \mathcal{T}$.
- *finally we add the corresponding PER,* $[\mathsf{Sum}(X, F)] = \coprod\ X\ F$
- $\mathsf{Nat} = \mathsf{Nat} \in \mathcal{U}$,
- $[\mathsf{Nat}] = \mathcal{N}$
- $\mathsf{N}_n = \mathsf{N}_n \in \mathcal{U}$,
- $[\mathsf{N}_n] = \mathsf{Const}_n$
- *if* $X = X' \in \mathcal{T}$, *then* $\mathsf{Prf}(X) = \mathsf{Prf}(X') \in \mathcal{T}$.
- *if* $X \in dom(\mathcal{T})$, *then* $[\mathsf{Prf}(X)] = \{(\star, \star)\}$

Not that in the PER model, all propositions $\mathsf{Prf}(X)$ are inhabited. In fact, all types are inhabited, since there is a reflection from variables into any type, be it empty or not. So, the PER model is unsuited for refuting propositions. However, the logical relation we define in the next section will only be inhabited for non-empty types.

Definition 14.

$$\uparrow_{\mathsf{Sum}(X,F)} d = (\uparrow_X \mathsf{p}^*\ d, \uparrow_{F\ \ \uparrow_X (\mathsf{p}^*\ d)} \mathsf{q}^*\ d)$$
$$\downarrow_{\mathsf{Sum}(X,F)} d = (\downarrow_X \mathsf{p}^*\ d, \downarrow_{F\ \ (\mathsf{p}^*\ d)} \mathsf{q}^*\ d)$$

$$\uparrow_{\mathsf{Nat}} k = k \qquad \downarrow_{\mathsf{Nat}} \mathsf{suc}(d) = \mathsf{suc}(\downarrow_{\mathsf{Nat}} d)$$
$$\downarrow_{\mathsf{Nat}} d = d$$
$$\uparrow_{\mathsf{Prf}(X)} d = \star \qquad \downarrow_{\mathsf{Prf}(X)} d = \star$$
$$\uparrow_{\mathsf{N}_n} k = k \qquad \downarrow_{\mathsf{N}_n} d = d$$
$$\Downarrow \mathsf{Sum}(X, F) = \mathsf{Sum}(\Downarrow X, d \mapsto \Downarrow F\ \uparrow_X d) \qquad \Downarrow \mathsf{Nat} = \mathsf{Nat}$$
$$\Downarrow \mathsf{Prf}(X) = \mathsf{Prf}(\Downarrow X) \qquad \Downarrow \mathsf{N}_n = \mathsf{N}_n$$

$$\uparrow_{\mathsf{Prf}(X)} d = \star \qquad \downarrow_{\mathsf{Prf}(X)} d = \star \qquad \Downarrow \mathsf{Prf}(X) = \mathsf{Prf}(\Downarrow X)$$

Proof (6 on page 13).

1. let $k = k' \in \mathsf{Ne}$, then $\uparrow_{\mathsf{Nat}} k = \uparrow_{\mathsf{Nat}} k' \in [\mathsf{Nat}]$. Trivial, since $\mathsf{Ne} \subset \mathcal{N}$.
2. Let $d = d' \in [\mathsf{Nat}]$, and show $\downarrow_{\mathsf{Nat}} d = \downarrow_{\mathsf{Nat}} d' \in \mathsf{Nf}$. By induction on $d = d' \in \mathcal{N}$.
3. We have to show $\Downarrow \mathsf{Nat} = \Downarrow \mathsf{Nat} \in \mathsf{Nf}$. Trivial.
4. It is immediate that if $k = k' \in \mathsf{Ne}$, then $\uparrow_{\mathsf{Prf}(X)} k = \uparrow_{\mathsf{Prf}(X')} k' \in [\mathsf{Prf}(X)]$, because both sides are by definition $\star$, and $\star = \star \in [\mathsf{Prf}(X)]$.
5. Since $\mathsf{O} \in Nf$, and $\mathsf{R}_i \star = \mathsf{O}$, then $\star = \star \in \mathsf{Nf}$; hence if $d = d' \in [\mathsf{Prf}(X)]$, $\downarrow_{\mathsf{Prf}(X)} d = \downarrow_{\mathsf{Prf}(X')} d$.
6. by i.h. we have $\Downarrow X = \Downarrow X' \in \mathsf{Nf}$, and clearly $\mathsf{Prf}(\Downarrow X) = \mathsf{Prf}(\Downarrow X') \in \mathsf{Nf}$.

We define a partial function $\mathsf{rec} : [D \to D] \times D \times D \times D \to D$ as:

Definition 15.

$$\begin{aligned}
\mathsf{rec}(F, d, f, \mathsf{zero}) &= d \\
\mathsf{rec}(F, d, f, \mathsf{suc}(e)) &= (f \cdot e) \cdot \mathsf{rec}(F, d, f, e) \\
\mathsf{rec}(F, d, f, k) &= \uparrow_{F\ k} (\mathsf{nrec}(d' \mapsto \Downarrow F\ d', \\
&\qquad \downarrow_{F\ \mathsf{zero}} d, \\
&\qquad \mathsf{Lam}\, d' \mapsto (\mathsf{Lam}\, e' \mapsto \downarrow_{F\ \mathsf{suc}(d')} f \cdot d' \cdot e'), \\
&\qquad k))
\end{aligned}$$

Definition 16.

$$\begin{aligned}
&\mathsf{case}^n(F, \langle d_0, \ldots, d_{n-1}\rangle, \mathsf{c}^n_i) = d_i \\
&\mathsf{case}^n(F, \langle d_0, \ldots, d_{n-1}\rangle, k) = \uparrow_{F\ k} \mathsf{elim}^n(e \mapsto \Downarrow F\ e, \langle \downarrow_{F\ \mathsf{c}^n_0} d_0, \ldots, \downarrow_{F\ \mathsf{c}^n_{n-1}} d_{n-1}\rangle, k)
\end{aligned}$$

Remark 9. If for all $d = d' \in \mathcal{N}$, $F\ d = F'd' \in \mathcal{T}$, and $z = z' \in [F\ \mathsf{zero}]$, and for all $d = d' \in \mathcal{N}$, and $e = e' \in [F\ d]$, $s \cdot d \cdot e = s' \cdot d' \cdot e' \in [F\mathsf{suc}(d)]$, and $d = d' \in \mathcal{N}$ then $\mathsf{rec}(F, z, s, d) = \mathsf{rec}(F, z, s, d') \in [F\ d]$.

With these new definitions we can now give the semantic equations for the new constructs.

Definition 17.

$$\begin{aligned}
[\![\Sigma\, A\, B]\!]d &= \mathsf{Sum}([\![A]\!]d, d' \mapsto [\![B]\!]((d, d'))) \\
[\![\mathsf{fst}\ t]\!]d &= \mathsf{p}^*\ [\![t]\!]d \\
[\![\mathsf{snd}\ t]\!]d &= \mathsf{q}^*\ [\![t]\!]d \\
[\![(t, t')]\!]d &= ([\![t]\!]d, [\![t']\!]d) \\
[\![\mathsf{Nat}]\!]d &= \mathsf{Nat} \\
[\![\mathsf{zero}]\!]d &= \mathsf{zero} \\
[\![\mathsf{suc}(t)]\!]d &= \mathsf{suc}([\![t]\!]d) \\
[\![\mathsf{natrec}(B, z, s, t)]\!]d &= \mathsf{rec}(e \mapsto [\![B]\!](d, e), [\![z]\!]d, [\![s]\!]d, [\![t]\!]d) \\
[\![\mathsf{Prf}\ A]\!]d &= \mathsf{Prf}([\![A]\!]d) \\
[\![[a]]\!]d &= \star \\
[\![\mathsf{O}]\!]d &= \star \\
[\![b\ \mathsf{where}^B\ t]\!]d &= \star \\
[\![\mathsf{N}_n]\!]d &= \mathsf{N}_n \\
[\![\mathsf{c}^n_i]\!]d &= \mathsf{c}^n_i \\
[\![\mathsf{elim}^n(B, t_0, \ldots, t_{n-1}, t)]\!]d &= \mathsf{case}^n(e \mapsto [\![B]\!](d, e), \langle [\![t_0]\!]d, \ldots, [\![t_{n-1}]\!]d\rangle, [\![t]\!]d)
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{Prf}\ A]\!]d &= \mathsf{Prf}([\![A]\!]d) & [\![[a]]\!]d &= \star \\
[\![b\ \mathsf{where}^B\ t]\!]d &= \star & [\![\mathsf{O}]\!]d &= \star
\end{aligned}$$

Remark 10. 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 [**?**] 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 $\mathcal{T}$.

These relations are defined inductively on the structure of the elements in the domain of $\mathcal{T}$.

Definition 18 (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: $X \in \mathsf{Ne}$.*
  - $\Gamma \vdash A \sim X \in \mathcal{T}$ *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 $\Gamma \vdash t' : A' \sim d \in [X]$.*

Proof. By induction on $X \in \mathcal{T}$.

- Types.
  - $X = \mathsf{U}$: by hypothesis $\Gamma \vdash A = U$, hence by symmetry and transitivity $\Gamma \vdash A' = U$.
  - $X \in \mathsf{Ne}$: by hypothesis $\Gamma \vdash A = \mathsf{R}_{|\Gamma|}\,X$, again by symmetry and transitivity $\Gamma \vdash A' = \mathsf{R}_{|\Gamma|}\,X$.
  - $X = \mathsf{Sing}\,d\,X$: by hypothesis $\Gamma \vdash A = \{a\}_B$, and $\Gamma \vdash a : B \sim d \in [X]$. As before we use symmetry and transitivity, and conclude $\Gamma \vdash A' = \{a\}_B$.
  - $X = \mathsf{Fun}\,X'\,F$: by hypothesis $\Gamma \vdash A = \mathsf{Fun}\,B\,C$, $\Gamma \vdash B \sim X'$, and $\Delta \vdash C\,(\mathsf{p}^i, s) \sim F\ d \in \mathcal{T}$ for all $\Delta \leqslant^i \Gamma$ and $\Delta \vdash s : B\,\mathsf{p}^i \sim d \in [X']$. Using symmetry and transitivity, we have $\Gamma \vdash A' = \mathsf{Fun}\,B\,C$.

– Terms.
  - $X = \mathsf{U}$: proved by the three last cases in the previous part.
  - $X \in \mathsf{Ne}$: by hypothesis $\Gamma \vdash t = \mathsf{R}_{|\Gamma|}\, d : A$; and by conversion $\Gamma \vdash t' = t' : A'$, and $\Gamma \vdash t = \mathsf{R}_{|\Gamma|}\, d : A'$ from which we easily conclude $\Gamma \vdash t' = \mathsf{R}_{|\Gamma|}\, d : A'$.
  - $X = \mathsf{Sing}\, d\, X$: we have to show $\Gamma \vdash t' : A' \sim d' \in \{\!\{d\}\!\}_X$. From the hypothesis we know $\Gamma \vdash t : A \sim d' \in \{\!\{d\}\!\}_X$, hence there exist B and a, such that $\Gamma \vdash t : \{a\}_B$, and $\Gamma \vdash t : B \sim d \in [X]$. As $\Gamma \vdash t = t' : B$, we can use the i.h. and conclude $\Gamma \vdash t' : B \sim d \in [X]$.
  - $X = \mathsf{Fun}\, X'\, F$: we need to show $\Gamma \vdash t' : A' \sim f \in [\mathsf{Fun}\, X\, F]$. From the hypothesis we have $\Gamma \vdash A = \mathsf{Fun}\, B\, C$, and $\Gamma \vdash B \sim X \in \mathcal{T}$; by symmetry and transitivity we conclude $\Gamma \vdash A' = \mathsf{Fun}\, B\, C$.
    Now we have to show $\Delta \vdash \mathsf{App}\ (t'\, \mathsf{p}^i)\ s : C\,(\mathsf{p}^i, s) \sim f \cdot d \in [F\ d]$. Which can be derived using the i.h. on $\Delta \vdash C\,(\mathsf{p}^i, s) = C\,(\mathsf{p}^i, s)$, $\Delta \vdash \mathsf{App}\ (t\, \mathsf{p}^i)\ s = \mathsf{App}\ (t'\, \mathsf{p}^i)\ s : C\,(\mathsf{p}^i, s)$, and $\Delta \vdash \mathsf{App}\ (t\, \mathsf{p}^i)\ s : C\,(\mathsf{p}^i, s) \sim f \cdot \in [F\ d]$.

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 $\Delta \vdash t\, \mathsf{p}^i : A\, \mathsf{p}^i \sim d \in [X]$.*

Proof. By induction on $X \in \mathcal{T}$. We will show first the proof of the first point, and then the proof of the second point.

– Types:
  - U: the first part is trivial because $\mathsf{U}\, \mathsf{p}^i = \mathsf{U}$ for any i.
  - $\mathsf{Var}\, x_j$ and $\mathsf{App}\, k\, v$: by definition, and remark 1.
  - $\mathsf{Sing}\, d\, X$: by 2 we have $\Delta \vdash A\, \mathsf{p}^i = \{a\}_{A'\, \mathsf{p}^i}$, hence $\Delta \vdash A\, \mathsf{p}^i = \{a\, \mathsf{p}^i\}_{A'\, \mathsf{p}^i}$. And by i.h. we have $\Delta \vdash a\, \mathsf{p}^i : A'\, \mathsf{p}^i \sim d \in [X]$.
  - $\mathsf{Fun}\, X\, F$: by the congruence of equality with respect to the application of substitutions, we know $\Delta \vdash A\, \mathsf{p}^i = \mathsf{Fun}\,(A'\, \mathsf{p}^i)\,(B\,(\mathsf{p}^i\, \mathsf{p}, \mathsf{q}))$. By i.h. we have $\Delta \vdash A'\, \mathsf{p}^i \sim X$.
    For the second part, let $\Theta \leqslant^j \Delta$ and $\Theta \vdash s : (A'\, \mathsf{p}^i)\, \mathsf{p}^j \sim d \in [X]$; then we have to prove $\Theta \vdash B\,(\mathsf{p}^i\, \mathsf{p}, \mathsf{q})\,(\mathsf{p}^j, s) \sim F\ d$; and that follows from the definition, because $B\,(\mathsf{p}^i\, \mathsf{p}, \mathsf{q})\,(\mathsf{p}^j, s) = B\,(\mathsf{p}^{i+j}, s)$.
– Terms:
  - U: By the last three cases for the first part.
  - $\mathsf{Var}\, x_j$ and $\mathsf{App}\, k\, v$: The first part to show, namely $\Delta \vdash A \sim \mathsf{Var}\, x_i$, was shown in the previous point; for showing $\Delta \vdash t\, \mathsf{p}^i = \mathsf{R}_{|\Delta|}\, d : A$, we use the same reasoning used in the previous paragraph.
  - $\mathsf{Sing}\, d\, X$: We have $\Delta \vdash A\, \mathsf{p}^i = \{a\, \mathsf{p}^i\}_{A'\, \mathsf{p}^i}$ and by i.h. we have $\Delta \vdash t\, \mathsf{p}^i : A'\, \mathsf{p}^i \sim d \in [X]$.
  - $\mathsf{Fun}\, X\, F$: For proving $\Delta \vdash \lambda t\, \mathsf{p}^i : A\, \mathsf{p}^i \sim f \in [\mathsf{Fun}\, X\, F]$, first we note that $\Delta \vdash A\, \mathsf{p}^i = \mathsf{Fun}\,(A'\, \mathsf{p}^i)\,(B\,(\mathsf{p}^{i+1}, \mathsf{q}))$. The second thing to show is that $\Theta \vdash \mathsf{App}\ ((\lambda t\, \mathsf{p}^i)\, \mathsf{p}^j)\ s : B\,(\mathsf{p}^{i+1}, \mathsf{q})\,(\mathsf{p}^j, s) \sim f \cdot d \in [F\ d]$, for any $\Theta \leqslant^j \Delta$, and $\Delta \vdash s : A'\, \mathsf{p}^j \sim d \in [X]$, which follows from the definition of $\Gamma \vdash \lambda t : \mathsf{Fun}\, A'\, B \sim f \in [\mathsf{Fun}\, X\, F]$.

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* $\Gamma \vdash t : A \sim d' \in [X]$*.*

Proof. By induction on $X = X' \in \mathcal{T}$.

- $\mathsf{U} = \mathsf{U}$: trivial for the first part. The second part is proved in the first part of the following cases.
- $X = X' \in \mathsf{Ne}$: by definition of Ne, we know that $\mathsf{R}_{|\Gamma|}\, X = \mathsf{R}_{|\Gamma|}\, X'$ (this is syntactically equal), from that is trivial to deduce $\Gamma \vdash A = \mathsf{R}_{|\Gamma|}\, X'$.
  By definition $[X] = \mathsf{Ne}$, then we can use the same reasoning as before: $d = d' \in \mathsf{Ne}$, hence $\Gamma \vdash a = \mathsf{R}_{|\Gamma|}\, d : A$ imply $\Gamma \vdash \mathsf{R}_{|\Gamma|}\, d = \mathsf{R}_{|\Gamma|}\, d' : A$; by transitivity we have: $\Gamma \vdash a = \mathsf{R}_{|\Gamma|}\, d' : A$.
- $\mathsf{Sing}\, d\, X = \mathsf{Sing}\, d'\, X'$. By hypothesis, $\Gamma \vdash A = \{a\}_{A'}$, $\Gamma \vdash A' \sim X$, and also $\Gamma \vdash a : A' \sim d \in [X]$. By i.h. we have $\Gamma \vdash A' \sim X$, and $\Gamma \vdash a : A' \sim d' \in [X]$; by $[X] = [X']$ we conclude $\Gamma \vdash a : A' \sim d' \in [X']$.
  Let $e = e' \in \{\!\{d\}\!\}_X$ and $\Gamma \vdash a : A \sim e \in \{\!\{d\}\!\}_X$, show $\Gamma \vdash a : A \sim e' \in \{\!\{d'\}\!\}_{X'}$. By hypothesis we know $\Gamma \vdash A = \{a'\}_{A'}$, and $\Gamma \vdash a' : A' \sim d \in [X]$ and by i.h. we conclude $\Gamma \vdash a' : A' \sim d' \in [X']$, because we have $d = d' \in [X]$ and $X = X' \in \mathcal{T}$.
- $\mathsf{Fun}\, X\, F = \mathsf{Fun}\, X'\, F'$. By hypothesis, $\Gamma \vdash A = \mathsf{Fun}\, A'\, B$, $\Gamma \vdash A' \sim X \in \mathcal{T}$, and hence by i.h. $\Gamma \vdash A' \sim X' \in \mathcal{T}$. We need to prove $\Delta \vdash B\,(\mathsf{p}^i, s) \sim F'\, d \in \mathcal{T}$, where $\Delta \leqslant^i \Gamma$, and $\Delta \vdash s : A\,\mathsf{p}^i \sim d \in [X]$; that is proved by mean of the i.h. with $\Delta \vdash B\,(\mathsf{p}^i, s) \sim F\ d \in \mathcal{T}$ and $\Delta \vdash s : A\,\mathsf{p}^i \sim d \in [X']$.
  Let $\Gamma \vdash t : A \sim f \in \prod\, X\, F$, and $f = f' \in \prod\, X\, F$, show $\Gamma \vdash t : A \sim f' \in \prod\, X'\, F'$. By hypothesis we know $\Gamma \vdash A = \mathsf{Fun}\, A'\, B$, and

$$\Delta \vdash \mathsf{App}\ (t\,\mathsf{p}^i)\ s : B\,(\mathsf{p}^i, s) \sim f \cdot d \in [F\ d] \tag{*}$$

  for any $\Delta \leqslant^i \Gamma$ and

$$\Delta \vdash s : A'\,\mathsf{p}^i \sim d \in [X] \tag{**}$$

  By i.h. on (**), we have $\Delta \vdash s : A'\,\mathsf{p}^i \sim d' \in [X']$. And we also know that $f \cdot d = f' \cdot d \in [F\ d]$; hence by i.h. on (*), we conclude $\Delta \vdash \mathsf{App}\ (t\,\mathsf{p}^i)\ s : B\,(\mathsf{p}^i, s) \sim f' \cdot d \in [F'\ d]$.

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* $k \in \mathsf{Ne}$*, 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* $\Gamma \vdash t : A \sim \uparrow_X k \in [X]$*.*

Proof. By induction on $X \in \mathcal{T}$. For organising better the proof we show the proofs for each point separately.

– $\Gamma \vdash A = \mathsf{R}_{|\Gamma|} \Downarrow X$
  - $X = \mathsf{U}$: trivial for the first part.
  - $X \in \mathsf{Ne}$: also trivial.
  - $X = \mathsf{Sing}\, d\, X$: by induction hypothesis we have $\Gamma \vdash A' = \mathsf{R}_{|\Gamma|} \Downarrow X$, and $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_X d : A'$. By congruence we conclude $\Gamma \vdash \{a\}_{A'} = \{\mathsf{R}_{|\Gamma|} \downarrow_X d\}_{\mathsf{R}_{|\Gamma|} \Downarrow X}$.
  - $X = \mathsf{Fun}\, X\, F$: by i.h. we have $\Gamma \vdash A' = \mathsf{R}_{|\Gamma|} \Downarrow X$, and $\Delta \vdash B\,(\mathsf{p}^i, s) = \mathsf{R}_{|\Delta|} \Downarrow F\ d$, for any $\Delta \leqslant^i \Gamma$ and $\Delta \vdash s : A'\,\mathsf{p}^i \sim d \in [X]$. By **??** and i.h. on the third part we have $\Gamma.A' \vdash \mathsf{q} : A'\,\mathsf{p} \sim \uparrow_X \mathsf{Var}\, x_{|\Gamma|}$. Hence we conclude $\Gamma.A' \vdash B\,(\mathsf{p}, \mathsf{q}) = \mathsf{R}_{|\Gamma.A'|} \Downarrow F\ \uparrow_X \mathsf{Var}\, x_{|\Gamma|}$, hence $\Gamma.A' \vdash B = \mathsf{R}_{|\Gamma.A'|} \Downarrow F\ \uparrow_X \mathsf{Var}\, x_{|\Gamma|}$, because $(\mathsf{p}, \mathsf{q}) = \mathsf{id}_{\Gamma.A'}$. From this we conclude $\Gamma \vdash \mathsf{Fun}\, A'\, B = \mathsf{R}_{|\Gamma|}\, \mathsf{Fun}\, X\, F$.

– $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_X d : A$
  - $d \in \mathcal{U}$: this is shown by the previous part.
  - $d \in [X]$, and $X \in \mathsf{Ne}$: also trivial.
  - $d' \in [\mathsf{Sing}\, d\, X]$: let $\Gamma \vdash A \sim \mathsf{Sing}\, d\, X$, and $(*)$ $\Gamma \vdash t : A \sim d' \in \mathsf{Sing}\, d\, X$; show $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_{\mathsf{Sing}\, d\, X} d' : A$.
    From $(*)$, we get $\Gamma \vdash A = \{a\}_{A'}$, hence $\Gamma \vdash t = A : \{a\}_{A'}$, and $\Gamma \vdash A' : t \sim d \in [X]$; by i.h. in the last fact, we know $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_X d : A'$, hence $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_X d : \{t\}_{A'}$, and using conversion twice we conclude $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_X d : A$.
  - $f \in [\mathsf{Fun}\, X\, F]$: let $\Gamma \vdash t : A \sim f \in [\mathsf{Fun}\, X\, F]$, show $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_{\mathsf{Fun}\, X\, F} f : A$. First note that $\Gamma \vdash \mathsf{q} : A' \sim \uparrow_X \mathsf{Var}\, x_{|\Gamma|}$, by i.h. on the third part. Hence by definition of the logical relation: $\Gamma.A' \vdash \mathsf{App}\,(t\,\mathsf{p})\,\mathsf{q} : B\,(\mathsf{p}, \mathsf{q}) \sim f \cdot \uparrow_X \mathsf{Var}\, x_{|\Gamma|} \in [F\ \uparrow_X \mathsf{Var}\, x_{|\Gamma|}]$. Therefore by i.h. we conclude: $\Gamma.A' \vdash \mathsf{App}\,(t\,\mathsf{p})\,\mathsf{q} = \mathsf{R}_{|\Gamma.A|} \downarrow_{F\ \uparrow_X \mathsf{Var}\, x_{|\Gamma|}} f \cdot \uparrow_X \mathsf{Var}\, x_{|\Gamma|} : B\,(\mathsf{p}, \mathsf{q})$. By conversion we have $\Gamma.A' \vdash \mathsf{App}\,(t\,\mathsf{p})\,\mathsf{q} = \mathsf{R}_{|\Gamma.A|} \downarrow_{F\ \uparrow_X \mathsf{Var}\, x_{|\Gamma|}} f \cdot \uparrow_X \mathsf{Var}\, x_{|\Gamma|} : B$, and by congruence we have $\Gamma \vdash \lambda(\mathsf{App}\,(t\,\mathsf{p})\,\mathsf{q}) = \lambda(\mathsf{R}_{|\Gamma.A|} \downarrow_{F\ \uparrow_X \mathsf{Var}\, x_{|\Gamma|}} f \cdot \uparrow_X \mathsf{Var}\, x_{|\Gamma|}) : \mathsf{Fun}\, A'\, B$; and by transitivity we conclude $\Gamma \vdash t = \mathsf{R}_{|\Gamma|} \downarrow_{\mathsf{Fun}\, X\, F} f : \mathsf{Fun}\, A'\, B$.

–
  - $X = \mathsf{U}$: we have to show $\Gamma \vdash t : A \sim \uparrow_{\mathsf{U}} k$. the first part $\Gamma \vdash A = \mathsf{U}$, is obtained from the hypothesis. The second part $\Gamma \vdash t \sim d \in \mathcal{T}$, is trivial from the hypothesis.
  - $X \in \mathsf{Ne}$: by hypothesis.
  - $X = \mathsf{Sing}\, d\, X$: we need to show $\Gamma \vdash t : A' \sim d \in [X]$. From the hypothesis we know: $\Gamma \vdash t : \{a\}_{A'}$ and $\Gamma \vdash a : A' \sim d \in [X]$; hence we have $\Gamma \vdash a = t : A'$. By lemma 7, we conclude $\Gamma \vdash t : A' \sim d \in [X]$.
  - $X = \mathsf{Fun}\, X\, F$: let $\Delta \leqslant^i \Gamma$, and $(*)$ $\Delta \vdash s : A'\,\mathsf{p}^i \sim d' \in [X]$. We have to show $\Delta \vdash \mathsf{App}\ t\,\mathsf{p}^i\ s : B\,(\mathsf{p}^i, s) \sim \uparrow_{F\ d'} \mathsf{App}\, d \downarrow_X d'$. From $(*)$, using the i.h., we have $\Delta \vdash s = \mathsf{R}_{|\Delta|} \downarrow_X d' : A'\,\mathsf{p}^i$. Hence we can use that and the hypothesis (after lifting) to deduce $\Delta \vdash \mathsf{App}\,(t\,\mathsf{p}^i)\ s = \mathsf{App}\ ((\mathsf{R}_{|\Gamma|}\, d)\ \mathsf{p}^i)\ (\mathsf{R}_{|\Delta|}\,(\downarrow_X d')) : B\,(\mathsf{p}^i, s)$. The r.h.s. of the equality is equal by definition to $\mathsf{R}_{|\Delta|}\, \mathsf{App}\, d \downarrow_X d'$; hence using the i.h. we conclude $\Delta \vdash \mathsf{App}\,(t\,\mathsf{p}^i)\ s : B\,(\mathsf{p}^i, s) \sim \uparrow_{F\ d'} \mathsf{App}\, d \downarrow_X d' \in [F\ d']$.

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 19 (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* $\Gamma \vdash \sigma : \Delta \sim d \in \mathcal{X}$ *and* $\Gamma \vdash t : A\,\sigma \sim d' \in [F\ d]$.

By the way this relation is defined, the counterparts of 7, 8 and 9 are easily proved by induction on the co-domain of the substitutions.

Remark 11. If $\gamma = \delta : \Gamma \to \Delta$, and $\Gamma \vdash \gamma : \Delta \sim d \in X$, then $\Gamma \vdash \delta : \Delta \sim d \in X$.

Remark 12. If $\Delta \vdash \delta : \Gamma \sim d \in [\![\Gamma]\!]$, then for any $\Theta \leqslant^i \Delta$, $\Theta \vdash \delta\,\mathsf{p}^i : \Gamma \sim d \in [\![\Gamma]\!]$.

Remark 13. If $\Gamma \vdash \gamma : \Delta \sim d \in X$, and $d = d' \in X$, then $\Gamma \vdash \gamma : \Delta \sim d' \in X$.

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* $\Delta \vdash \gamma\,\delta : \Theta \sim [\![\gamma]\!]d \in [\![\Theta]\!]$.

Proof. By mutual induction on the derivations. We note that for terms we show only the cases when the last rule used was the introductory rule, or the rule for introducing elements in singletons; the case when the last rule used was the conversion rule, we can conclude by i.h., and lemma 7.

- Types:
  - $\diamond \vdash \mathsf{U}$: trivial using 7 and 9.
  - $\Gamma \vdash \mathsf{Fun}\,A\,B$: we will show $\Theta \vdash B\,(\delta\,p^i, s) \sim [\![B]\!](d, e) \in \mathcal{T}$, for any $\Theta \leqslant^i \Delta$, and $(*)$ $\Theta \vdash s : (A\,\delta)\,\mathsf{p}^i \sim e \in [[\![A]\!]d]$. By 12 we have $\Theta \vdash \delta\,\mathsf{p}^i : \Gamma \sim d \in [\![\Gamma]\!]$, and from $(*)$ we conclude $\Theta \vdash (\delta\,\mathsf{p}^i, s) : \Gamma.A \sim (d, e)$, hence we can use the i.h. on $\Gamma.A \vdash B$ and the rest is by 7 and 9.
  - $\Gamma \vdash A$ (by $\Gamma \vdash A : \mathsf{U}$): by i.h. $\Delta \vdash A\,\delta : \mathsf{U} \sim [\![A]\!]d \in \mathcal{U}$.
  - $\Gamma \vdash \{t\}_A$: by i.h. $\Delta \vdash t\,\delta : A\,\delta \sim [\![t]\!]d \in [\![A]\!]d$.
  - $\Gamma \vdash A\,\sigma$: by i.h. $\Delta \vdash \sigma\,\delta : \Theta \sim [\![\sigma]\!]d \in [\![\Theta]\!]$; and, again by i.h., $\Delta \vdash A\,\sigma\,\delta \sim [\![A]\!]([\![\sigma]\!]d) \in \mathcal{T}$.
- Terms:
  - $\Gamma \vdash \mathsf{Fun}\,A\,B : \mathsf{U}$ and $\Gamma \vdash \{t\}_A : \mathsf{U}$ are the same as before.
  - $\Gamma \vdash \lambda t : \mathsf{Fun}\,A\,B$: we will show $\Theta \vdash t\,(\delta\,p^i, s) : B\,(\delta\,p^i, s) \sim [\![\lambda t]\!]d \cdot e \in [[\![B]\!](d, e)]$, for any $\Theta \leqslant^i \Delta$, and $(*)$ $\Theta \vdash s : (A\,\delta)\,\mathsf{p}^i \sim e \in [[\![A]\!]d]$. By 12 we have $\Theta \vdash \delta\,\mathsf{p}^i : \Gamma \sim d \in [\![\Gamma]\!]$, and from $(*)$ we conclude $\Theta \vdash (\delta\,\mathsf{p}^i, s) : \Gamma.A \sim (d, e)$, hence we can use the i.h. on $\Gamma.A \vdash t : B$ and the rest is by 7 and 9.

- $\Gamma \vdash \mathsf{App}\ t\ r : B\,(\mathsf{id}_\Gamma, r)$: by i.h. we have $\Delta \vdash r\,\delta : A\,\delta \sim [\![r]\!]d \in [[\![A]\!]d]$ and, $\Delta \vdash t\,\delta : \mathsf{Fun}\,A\,B\,\delta \sim [\![t]\!]d \in [[\![\mathsf{Fun}\,A\,B]\!]d]$; hence $\Delta \vdash \mathsf{App}\ t\,\delta\ r\,\delta : B\,(\mathsf{id}_\Delta, r\,\delta) \sim [\![t]\!]d \cdot [\![r]\!]d \in [[\![B]\!](d, [\![r]\!]d)]$. The rest is again using 7 and 9.
- $\Gamma.A \vdash \mathsf{q} : A\,\mathsf{p}$: by remark 4 we know $\delta = (\gamma, t)$, for some γ and t; hence $d = (e, e')$, $\Delta \vdash \gamma : \Gamma \sim e \in [\![\Gamma]\!]$, and $\Delta \vdash t : A\,\delta \sim e' \in [[\![A]\!]e]$. Now we know that $\mathsf{q}\,(\gamma, t) = t$, and $[\![A\,\mathsf{p}]\!](e, e') = [\![A]\!]e$. Hence we are done by 7 and 9.
- $\Gamma \vdash t\,\sigma : A\,\sigma$: by i.h. $\Delta \vdash \sigma\,\delta : \Theta \sim [\![\sigma]\!]d \in [\![\Theta]\!]$; and, again by i.h., $\Delta \vdash t\,\sigma\,\delta : A\,\sigma\,\delta \sim [\![t]\!]([\![\sigma]\!]d) \in [[\![A]\!]([\![\sigma]\!]d)]$.
- $\Gamma \vdash t : \{a\}_A$: by i.h. we have $\Delta \vdash t\,\delta : A\,\delta \sim [\![t]\!]d \in [[\![A]\!]d]$, and by soundness $[\![t]\!]d = [\![a]\!] \in [[\![A]\!]d]$, hence $\Delta \vdash t\,\delta : A\,\delta \sim [\![a]\!]d \in [[\![A]\!]d]$.

– Substitutions:
- $\Gamma \vdash t\,\sigma : A\,\sigma$: by i.h. $\Delta \vdash \sigma\,\delta : \Theta \sim [\![\sigma]\!]d \in [\![\Theta]\!]$; and, again by i.h., $\Delta \vdash t\,\sigma\,\delta : A\,\sigma\,\delta \sim [\![t]\!]([\![\sigma]\!]d) \in [[\![A]\!]([\![\sigma]\!]d)]$.
- $\Gamma \vdash \mathsf{id}_\Gamma : \Gamma$: show that $\Delta \vdash \mathsf{id}_\Gamma\,\delta : \Gamma \sim [\![\mathsf{id}_\Gamma]\!]d \in [\![\Gamma]\!]$. First note that $\mathsf{id}_\Gamma\,\delta = \delta$ and $[\![\mathsf{id}_\Gamma]\!]d = d$, hence using 11 and 13 we conclude what was needed.
- $\Gamma \vdash \langle\rangle : \diamond$: Again we use 11 and 13 to conclude .
- $\Gamma \vdash (\gamma, t) : \Theta.A$: we use the i.h. on $\Gamma \vdash \gamma : \Theta$ and $\Gamma \vdash t : A\,\gamma$; so we have $\Delta \vdash \gamma\,\delta : \Theta \sim [\![\gamma]\!]d \in [\![\Theta]\!]$, and $\Delta \vdash t\,\delta : (A\,\gamma)\,\delta \sim [\![t]\!]d \in [[\![A\,\gamma]\!]d]$. From that we get that $([\![\gamma]\!]d, [\![t]\!]d) \in \coprod [\![\Theta]\!]\,(e \mapsto [[\![A\,\gamma]\!]e])$. Finally $(\gamma, t)\,\delta = (\gamma\,\delta, t\,\delta)$ and $[\![(\gamma, t)]\!]d = ([\![\gamma]\!]d, [\![t]\!]d)$, hence we use 11 and 13 and we are done.
- $\Gamma \vdash \gamma'\,\gamma : \Theta'$: by i.h. $\Delta \vdash \gamma\,\delta : \Gamma \sim [\![\gamma]\!]d \in [\![\Gamma]\!]$, and using this we conclude by i.h. $\Delta \vdash \gamma'\,(\gamma\,\delta) : \Theta' \sim [\![\gamma']\!]([\![\gamma]\!]d) \in [\![\Theta']\!]$. The rest is by 11 and 13.
- $\Gamma.A \vdash \mathsf{p} : \Gamma$: by remark 4 we know $\delta = (\gamma, t)$, for some γ and t; hence $d = (e, e')$, and $\Delta \vdash \gamma : \Gamma \sim e \in [\![\Gamma]\!]$. Finally, $\mathsf{p}\,(\gamma, t) = \gamma$, and $[\![\mathsf{p}]\!](e, e') = e$.

At this point we have all the ingredients to define a normalisation algorithm for our calculus. First we define recursively on the structure of contexts an element of D, which is related with the identity substitution of that context; then we use that value for calculating the semantics of well-typed term $t : A$ (resp. well-formed type A); by the last theorem the value of t is in the PER corresponding to the value of A (resp. the value of A is in $\mathcal{T}$), and is related to t (resp. to A). Then we can use the key lemma 10 and conclude that t (resp. A) is provable equal to the reification of the normalised value of t (resp. A).

Definition 20. *We define $\rho_\Gamma = P_\Gamma\ \top$, where*

$$P_\diamond\ d = d$$
$$P_{\Gamma.A}\ d = (d', \uparrow_{[\![A]\!]d'} \mathsf{Var}\,x_n) \qquad \text{where } n = |\Gamma|,\ \text{and } d' = P_\Gamma\ d.$$

Then $\Gamma \vdash \mathsf{id}_\Gamma : \Gamma \sim \rho_\Gamma \in [\![\Gamma]\!]$ for $\Gamma \in \mathsf{Ctx}$.

Definition 21 (Normalisation algorithm). *Let $\Gamma \vdash A$, and $\Gamma \vdash t : A$.*

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

If Γ, and A are clear from the context, then we will omit them and write $\mathbf{nbe}(A)$, and $\mathbf{nbe}(t)$, respectively.

Remark 14. Let $\Gamma \vdash A$, and $\Gamma \vdash t : A$, then

1. by fundamental theorem of logical relations (and lemma 7),
   - $\Gamma \vdash A \sim [\![A]\!]\rho_\Gamma \in \mathcal{T}$; and
   - $\Gamma \vdash t : A \sim [\![t]\!]\rho_\Gamma \in [[\![A]\!]\rho_\Gamma]$,
2. then it is immediate using lemma 10, that
   - $\Gamma \vdash A = \mathbf{nbe}(A)$, and
   - $\Gamma \vdash t = \mathbf{nbe}(t) : A$.

Remark 15. 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.

Logical relations

Definition 22.

- $\Gamma \vdash A \sim \mathsf{Sum}(X, F)$, *iff* $\Gamma \vdash A = \Sigma\, A'\, B'$, *and* $\Gamma \vdash A' \sim X$, *and for all* $\Delta \leqslant^i \Gamma$, *and* $\Delta \vdash s : A'\, \mathsf{p}^i \sim d \in [X]$, $\Delta \vdash B'\, (\mathsf{p}^i, s) \sim F d$.
- $\Gamma \vdash t : A \sim d \in [\mathsf{Sum}(X, F)]$, *iff* $\Gamma \vdash A = \Sigma\, A'\, B'$, *and* $\Gamma \vdash \mathsf{fst}\ t : A' \sim \mathsf{p}^*\, d \in [X]$, *and* $\Gamma \vdash \mathsf{snd}\ t : B'\, (\mathsf{id}_\Gamma, \mathsf{fst}\ t) \sim \mathsf{q}^*\, d \in [F \mathsf{p}^*\, d]$.
- $\Gamma \vdash A \sim \mathsf{Nat}$, *iff* $\Gamma \vdash A = \mathsf{Nat}$.
- $\Gamma \vdash t : A \sim d \in [\mathsf{Nat}]$, *iff* $\Gamma \vdash A \sim \mathsf{Nat}$, *and for all* $\Delta \leqslant^i \Gamma$, $\Delta \vdash t\, \mathsf{p}^i = \mathsf{R}_{|\Delta|}\, d : A\, \mathsf{p}^i$.
- $\Gamma \vdash A \sim \mathsf{N}_n$, *iff* $\Gamma \vdash A = \mathsf{N}_n$.
- $\Gamma \vdash t : A \sim d \in [\mathsf{N}_n]$, *iff* $\Gamma \vdash A \sim \mathsf{N}_n$, *and for all* $\Delta \leqslant^i \Gamma$, $\Delta \vdash t\, \mathsf{p}^i = \mathsf{R}_{|\Delta|}\, d : A\, \mathsf{p}^i$.
- $\Gamma \vdash A \sim \mathsf{Prf}(X) \in \mathcal{T}$, *iff* $\Gamma \vdash A = \mathsf{Prf}\ A'$, *and* $\Gamma \vdash A' \sim X \in \mathcal{T}$.
- $\Gamma \vdash t : A \sim d \in [\mathsf{Prf}(X)]$, *iff* $\Gamma \vdash A \sim \mathsf{Prf}(X)$.

Lemma 11. *1.* $\mathbf{nbe}_\Gamma(\Sigma\, A\, B) = \Sigma\, \mathbf{nbe}_\Gamma(A)\, \mathbf{nbe}_{\Gamma.A}(B)$;
2. $\mathbf{nbe}^{\Sigma\, A\, B}_\Gamma((t, b)) = (\mathbf{nbe}^A_\Gamma(t), \mathbf{nbe}^{B\, (\mathsf{id}, t)}_\Gamma(b))$;
3. $\mathbf{nbe}(\mathsf{suc}(t)) = \mathsf{suc}(\mathbf{nbe}(t))$.
4. $\mathbf{nbe}(\mathsf{Prf}\ A) = \mathsf{Prf}\ \mathbf{nbe}(A)$.

Corollary 5 (). *If $\Gamma \vdash \Sigma\, A\, B = \Sigma\, A'\, B'$, then $\Gamma \vdash A = A'$, and $\Gamma.A \vdash B = B'$.*

5 Type-checking algorithm

In this section we define three predicates that represent algorithms for checking the well formation of contexts in normal form, the derivability of a type under a certain context, the typing of a normal form, and a partial function from contexts and neutral terms to types, that represents the algorithm for inferring types of neutral terms. We implemented them in Haskell (see the appendix B).

The algorithm is similar to previous ones [14, 3], in that it proceeds by analysing the possible types for each normal form, and succeeds only if the type corresponds to the type corresponding to the type of the introductory rule of the term. The only difference is introduced by the presence of singleton types; now we should take into account that a normal form can also have a singleton as its type.

This situation can be dealt in two possible ways; either one check that the deepest tag of the normalised type (see 24) has the form of the type of the introductory rule; or one add a rule for checking any term against singleton types. The first approach requires to have more rules (this is due to the combination of singletons and a universe). We take the second approach, which requires to compute the eta-long normal form of the type before type-checking. We also note that the proof of completeness is more involved, because now the algorithm is not only driven by the term being checked, but also by the type.

In both cases it is crucial to have a normalisation function with the following properties.

Definition 23 (Properties for normalisation).

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* $\Gamma \vdash t = t' : A$.

Notation. In this section, we fix C for a context where all the types are in normal form; $V, V', W, v, v', w \in Nf$, and $k \in Ne$. For obtaining the deepest tag, we define an operation on types, which is essentially the same as the one defined in [8]

Definition 24 (Singleton's tag).

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

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

Definition 25 (Type-checking and type-inference).
Contexts $\Gamma \Leftarrow$.

$$\frac{}{\diamond \Leftarrow} \qquad \frac{\Gamma \Leftarrow \qquad \Gamma \Leftarrow V}{\Gamma.V \Leftarrow}$$

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* $\Gamma \vdash$.

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

Theorem 4 (Correctness of type-checking).

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

Proof. By simultaneous induction on $\Gamma \Leftarrow$, $\Gamma \Leftarrow V$, and $\Gamma \vdash v \Leftarrow A$.

- Contexts:
  - For the empty context the proof is trivial: the derivation is $\diamond \vdash$.
  - For the case of an extension, we know $\Gamma \Leftarrow$, and also $\Gamma \Leftarrow V$; hence by i.h. we have derivations with conclusions $\Gamma \vdash$, and $\Gamma \vdash V$, so we can derive $\Gamma.V \vdash$.
- Types:
  - the case for U is trivial.
  - the case for $\mathsf{Fun}\, V\, W$ is also obtained directly from the derivations we get using the i.h. on $\Gamma \Leftarrow V$, and $\Gamma.V \Leftarrow W$; and use them for deriving $\Gamma \vdash \mathsf{Fun}\, V\, W$
  - for $\{v\}_V$, we can apply the same reasoning as before: by i.h. on $\Gamma \Leftarrow V$, and $\Gamma \vdash v \Leftarrow \mathbf{nbe}(V)$ we know that there are, respectively, derivations with conclusions $\Gamma \vdash V$, and $\Gamma \vdash v : V$; from which we can conclude $\Gamma \vdash \{v\}_V$
  - here we'll consider the three cases when V is a neutral term, because the reasoning is the same. By i.h. on $\Gamma \vdash V \Leftarrow \mathsf{U}$, we have a derivation with conclusion $\Gamma \vdash V : \mathsf{U}$; hence we use U-EL.

– Terms:
  - let $V = \mathsf{U}$, and $v = \mathsf{Fun}\, V'\, W$. By i.h. $\Gamma \vdash V' : \mathsf{U}$, and $\Gamma.V' \vdash W : \mathsf{U}$, and using both derivations we can derive $\Gamma \vdash \mathsf{Fun}\, V\, W : \mathsf{U}$.
  - consider $V = \mathsf{U}$, and $v = \{v'\}_{V'}$. by i.h. on $\Gamma \vdash V' \Leftarrow \mathsf{U}$, and $\Gamma \vdash v' \Leftarrow \mathbf{nbe}(V)$, we have $\Gamma \vdash V : \mathsf{U}$, and $\Gamma \vdash v' : \mathbf{nbe}(V)$, and using conversion we derive $\Gamma \vdash v' : V$; and these are the premises we need to show $\Gamma \vdash \{v'\}_V : \mathsf{U}$.
  - $V = \mathsf{Fun}\, V'\, W$, and $v = \lambda v'$: we have $\Gamma.V' \vdash v' \Leftarrow W$. From this we can conclude by i.h. $\Gamma.V' \vdash v' : W$; and this is the key premise for concluding $\Gamma \vdash \lambda v' : \mathsf{Fun}\, V'\, W$.
  - $V = \{w\}_W$: by hypothesis we know $\Gamma \vdash w : W$, and $\Gamma \vdash v \Leftarrow W$, and $\Gamma \vdash w = v : W$; by the i.h. on the second one we get $\Gamma \vdash v : W$; then we can conclude using SING-I.
  - $v = k \in Ne$, and $V \not\equiv \{w\}_W$: let $\Gamma \vdash k \Rightarrow V'$, then we distinguish the cases when V' is a singleton, and when V' is not a singleton. In the later case, the derivation is obtained directly from the correctness of type-inference. In the first case we use the rule SING-EL, with the derivation obtained by i.h. and then we conclude with conversion.

– Inference:
  - for $\mathsf{q}\,\mathsf{p}^i$, if $i = 0$, then we use HYP, and conversion; if $i > 0$, then we have a derivation with conclusion $\Gamma \vdash \mathsf{q} : A_i\, \mathsf{p}$, and clearly $\Gamma \vdash \mathsf{p}^i : \Gamma.A_i \ldots.A_0$, hence by SUBS-TERM, we have $\Gamma.A_i \ldots.A_0 \vdash \mathsf{q}\,\mathsf{p}^i : A_i\, \mathsf{p}^{i+1}$, we conclude by the fact $\Gamma \vdash \mathbf{nbe}(A) = A$, and conversion.
  - by i.h. we have derivations with conclusions $\Gamma \vdash k : V'$, with $\overline{V'} = \mathsf{Fun}\, V\, W$, hence we have a derivation $\Gamma \vdash k : \mathsf{Fun}\, V\, W$ (using SING-EL if necessary) and $\Gamma \vdash v : V$, hence by the rule FUN-EL, we have $\Gamma \vdash \mathsf{App}\ k\ v : W\,(\mathsf{id}, v)$. We conclude with conversion and the fact that $\Gamma \vdash A = \mathbf{nbe}(A)$.

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

Proof. We prove simultaneously all the points. The first two points are by induction on the structure of the context, and the type, respectively. In the last two points we use well-founded induction on the order $\prec$.

– Contexts:

  - the case for $\diamond$ is trivial;
  - for $C = C'.V$, we have by i.h. $C' \Leftarrow$, and $C' \Leftarrow V$; hence $C'.V \Leftarrow$.
- Types:
  - U, trivial.
  - $\Gamma \vdash \mathsf{Fun}\, V'\, W$; by inversion we know $\Gamma \vdash V'$, and $\Gamma.V' \vdash W$; hence by i.h. we have respectively $\Gamma \Leftarrow V'$, and $\Gamma.V' \Leftarrow W$.
  - $V = \{v\}_{V'}$: by inversion we have $\Gamma \vdash V'$, and $\Gamma \vdash v : \mathbf{nbe}(V')$, hence by i.h. we have both $\Gamma \Leftarrow V'$, and $\Gamma \vdash v \Leftarrow V'$.
  - $\Gamma \vdash k$, we have to show $\Gamma \Leftarrow k$. By lemma **??**, we know $\Gamma \vdash k : \mathsf{U}$; hence by i.h. we have $\Gamma \vdash k \Rightarrow A$, and $\Gamma \vdash A = \mathsf{U}$, hence $\Gamma \vdash k \Leftarrow \mathsf{U}$.
- Terms: We omit the trivial cases, e.g. (U, A); we have re-arranged the order of the cases for the sake of clarity.
  - $v = \mathsf{Fun}\, V'\, W$:
    1. either $\Gamma \vdash A = \mathsf{U}$, $\Gamma \vdash V' : \mathsf{U}$, and $\Gamma.V' \vdash W : \mathsf{U}$; hence, by i.h. we know both $\Gamma \vdash V' \Leftarrow \mathsf{U}$, and $\Gamma.V' \vdash W \Leftarrow \mathsf{U}$; hence we can conclude $\Gamma \vdash \mathsf{Fun}\, V\, W \Leftarrow \mathsf{U}$.
    2. Or $\Gamma \vdash A = \{a\}_{A'}$, $\Gamma \vdash v : A'$, and $\Gamma \vdash v = a : A'$, hence by i.h. we know $\Gamma \vdash v \Leftarrow \mathbf{nbe}(B)$, by conversion we also have and transitivity of the equality $\Gamma \vdash \mathbf{nbe}(a) = v : \mathbf{nbe}(B)$, hence $\Gamma \vdash v \Leftarrow \{\mathbf{nbe}(a)\}_{\mathbf{nbe}(B)}$.
  - $v = \{v'\}_V$:
    1. $\Gamma \vdash V : \mathsf{U}$, and $\Gamma \vdash v' : V$. From those derivations we have by i.h. $\Gamma \vdash V \Leftarrow \mathsf{U}$, and $\Gamma \vdash v' \Leftarrow \mathbf{nbe}(V)$, respectively; from which we conclude $\Gamma \vdash \{v'\}_V \Leftarrow \mathsf{U}$
    2. $\Gamma \vdash A = \{a\}_{A'}$, with $\Gamma \vdash v : A'$, and $\Gamma \vdash v = a : A'$, hence by i.h. we know $\Gamma \vdash v \Leftarrow \mathbf{nbe}(B)$. We can also derive $\Gamma \vdash \mathbf{nbe}(a) = v : \mathbf{nbe}(B)$, hence $\Gamma \vdash v \Leftarrow \{\mathbf{nbe}(a)\}_{\mathbf{nbe}(B)}$.
  - $v = \lambda v'$
    1. $\Gamma \vdash V = \mathsf{Fun}\, A'\, B$, and $\Gamma.A' \vdash v' : B$; from this we can conclude $\Gamma.\mathbf{nbe}(A') \vdash v' : B$ by i.h. we get $\Gamma.\mathbf{nbe}(A') \vdash v' \Leftarrow \mathbf{nbe}(B)$; hence $\Gamma \vdash \lambda v' \Leftarrow \mathsf{Fun}\, \mathbf{nbe}(A')\, \mathbf{nbe}(B')$.
    2. Or $\Gamma \vdash A = \{a\}_{A'}$, $\Gamma \vdash v : A'$, and $\Gamma \vdash v = a : A'$, hence by i.h. we know $\Gamma \vdash v \Leftarrow \mathbf{nbe}(B)$, by conversion we also have and transitivity of the equality $\Gamma \vdash \mathbf{nbe}(a) = v : \mathbf{nbe}(B)$, hence $\Gamma \vdash v \Leftarrow \{\mathbf{nbe}(a)\}_{\mathbf{nbe}(B)}$.
  - $v \in \mathit{Ne}$: then we do case analysis on $\mathbf{nbe}(A)$.
    1. If $\mathbf{nbe}(A) = \{w\}_W$, then by soundness of $\mathbf{nbe}(\_)$, and conversion we have $\Gamma \vdash k : \{w\}_W$; and by inversion of singletons we have $\Gamma \vdash k : W$, and also $\Gamma \vdash k = w : W(*)$. Clearly $(k, W) \prec (k, A)$, hence we can apply the inductive hypothesis and conclude $\Gamma \vdash k \Leftarrow W$; from that and $(*)$, we conclude $\Gamma \vdash k \Leftarrow \{w\}_W$, i.e., $\Gamma \vdash k \Leftarrow \mathbf{nbe}(A)$.
    2. If $V \not\equiv \{w\}_W$, then $\overline{V} \equiv V$. We use the last clause for concluding $\Gamma \vdash k \Leftarrow \mathbf{nbe}(A)$; but we need to show that if $\Gamma \vdash k \Rightarrow V'$, then $\Gamma \vdash \overline{V} = \overline{V'}$; we show this in the next point.
- Inference: let $\Gamma \vdash k : A$, $\Gamma \vdash k \Rightarrow V'$, and $V = \mathbf{nbe}(A)$. Show $\Gamma \vdash \overline{V} = \overline{V'}$.

- let us consider first the case when $V = \{w\}_W$; by inversion we have derivations $\Gamma \vdash k : W$, and $\Gamma \vdash k = w : W$. Hence by i.h. we know that $\Gamma \vdash \overline{V'} = \overline{W}$, and $\overline{W} = \overline{\{w\}_W}$.
- Now we consider the case when V is not a singleton, and $k = \mathsf{q}\,\mathsf{p}^i$; this case is trivial because by inversion we know that $\Gamma \vdash V = \mathbf{nbe}((\Gamma!i)\,\mathsf{p}^{i+1})$.
- the last case to consider is $k = \mathsf{App}\ k'\ v$ and V not a singleton. By inversion we know $\Gamma \vdash \mathsf{App}\ k\ v : B\,(\mathsf{id}, v)$, and $\Gamma \vdash k : \mathsf{Fun}\,A\,B$, hence $\Gamma \vdash k : \mathsf{Fun}\,\mathbf{nbe}(A)\,\mathbf{nbe}(B)$, and $\Gamma \vdash v : A$, hence $\Gamma \vdash v : \mathbf{nbe}(A)$. By i.h. we know that if $\Gamma \vdash k \Rightarrow V'$, then $\overline{V'} = \mathsf{Fun}\,\mathbf{nbe}(A)\,\mathbf{nbe}(B)$, and also $\Gamma \vdash v \Leftarrow \mathbf{nbe}(A)$. Hence we can conclude $\Gamma \vdash \mathsf{App}\ k\ v \Rightarrow \mathbf{nbe}(\mathbf{nbe}(B)\,(\mathsf{id}, v))$. And $\Gamma \vdash \mathbf{nbe}(\mathbf{nbe}(B)\,(\mathsf{id}, v)) = \mathbf{nbe}(B\,(\mathsf{id}, v))$ (using the derived rule shown in example **??** together with the correctness of the $\mathbf{nbe}(\_)$ algorithm).

Calculus with Proof-Irrelevance.

We give the additional rules for type-checking, and type-inference algorithms for the new constructs of the calculus. We show that the type-checking algorithm is correct and complete for the calculus without PRF-TM.

Definition 27 (Type-checking and type-inference).

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

$$\frac{\Gamma \Leftarrow V \qquad \Gamma.V \Leftarrow W}{\Gamma \Leftarrow \Sigma\,V\,W} \qquad \frac{}{\Gamma \Leftarrow \mathsf{Nat}}$$

$$\frac{}{\Gamma \Leftarrow \mathsf{N}_n} \qquad \frac{\Gamma \Leftarrow V}{\Gamma \Leftarrow \mathsf{Prf}\ V}$$

Terms $\Gamma \vdash v \Leftarrow V$.

$$\frac{\Gamma \vdash V \Leftarrow \mathsf{U} \qquad \Gamma.V \vdash W \Leftarrow \mathsf{U}}{\Gamma \vdash \Sigma\,V\,W \Leftarrow \mathsf{U}}$$

$$\frac{\Gamma \vdash v \Leftarrow V \qquad \Gamma \vdash v' \Leftarrow \mathbf{nbe}(W\,(\mathsf{id}_\Gamma, v))}{\Gamma \vdash (v, v') \Leftarrow \Sigma\,V\,W}$$

$$\frac{}{\Gamma \vdash \mathsf{Nat} \Leftarrow \mathsf{U}}$$

$$\frac{}{\Gamma \vdash \mathsf{zero} \Leftarrow \mathsf{Nat}} \qquad \frac{\Gamma \vdash v \Leftarrow \mathsf{Nat}}{\Gamma \vdash \mathsf{suc}(v) \Leftarrow \mathsf{Nat}}$$

$$\frac{}{\Gamma \vdash \mathsf{N}_n \Leftarrow \mathsf{U}}$$

$$\frac{i < n}{\Gamma \vdash \mathsf{c}_i^n \Leftarrow \mathsf{N}_n} \qquad \frac{\Gamma \vdash v \Leftarrow V}{\Gamma \vdash [v] \Leftarrow \mathsf{Prf}\ V}$$

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

$$\frac{\Gamma \vdash k \Rightarrow \Sigma\, V\, W}{\Gamma \vdash \mathsf{fst}\ k \Rightarrow V} \qquad \frac{\Gamma \vdash k \Rightarrow \Sigma\, V\, W}{\Gamma \vdash \mathsf{snd}\ k \Rightarrow \mathbf{nbe}(W\,(\mathsf{id}_\Gamma, \mathsf{fst}\ k))}$$

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

$$\frac{\Gamma.\mathsf{Nat} \Leftarrow V \qquad \Gamma \vdash k \Rightarrow \mathsf{Nat} \qquad \Gamma \vdash v \Leftarrow \mathbf{nbe}(V\,(\mathsf{id}_\Gamma, \mathsf{zero})) \qquad \Gamma.\mathsf{Nat} \vdash v' \Leftarrow \mathbf{nbe}(\mathsf{Fun}\,V\,(\mathsf{id},\mathsf{q})\,V\,(\mathsf{id}, \mathsf{suc}(\mathsf{q}\,\mathsf{p})))}{\Gamma \vdash \mathsf{natrec}(V, v, v', k) \Rightarrow \mathbf{nbe}(V\,(\mathsf{id}, k))}$$

$$\frac{\Gamma.\mathsf{N}_n \Leftarrow V \qquad \Gamma \vdash k \Rightarrow \mathsf{N}_n \qquad \Gamma \vdash v_i \Leftarrow \mathbf{nbe}(V\,(\mathsf{id}_\Gamma, \mathsf{c}_i^n))}{\Gamma \vdash \mathsf{elim}^n(V, v_0, \ldots, v_{n-1}, k) \Rightarrow \mathbf{nbe}(V\,(\mathsf{id}, k))}$$

Theorem 6. *The type-checking algorithm is correct with respect to the calculus with* PRF-TM.

Proof. By simultaneous induction on the derivability of the type-checking judgements.

Corollary 6. *The type-checking algorithm is sound with respect to the calculus without* PRF-TM.

Lemma 12. *The type-checking algorithm is complete with respect to the calculus with* PRF-TM *if we add the axiom* $\Gamma \vdash \mathsf{O} \Leftarrow \mathsf{Prf}\ V$.

Proof. We need to consider the typing rules introduced in this section.

- let $\Gamma \vdash \mathsf{Prf}\ V$, then by inversion we know $\Gamma \vdash V$, hence by i.h. $\Gamma \Leftarrow V$, and we are done.
- if $\Gamma \vdash A : [v]$, we know by inversion $\Gamma \vdash A' : v$, and $\Gamma \vdash A = \mathsf{Prf}\ A'$, and by i.h. we have $\Gamma \vdash v \Leftarrow \mathbf{nbe}(A')$, and by 15 we know $\mathbf{nbe}(A) \equiv \mathsf{Prf}\ \mathbf{nbe}(A')$, hence $\Gamma \vdash [v] \Leftarrow \mathbf{nbe}(A)$ (we don't consider the case when by inversion we have that $\Gamma \vdash A = \{a\}_{A'}$, for it is not different to other similar cases).
- if $\Gamma \vdash \mathsf{O} : A$, then we use the inversion lemma and we conclude by using the axiom considered in this lemma, and 15.

Remark 16. If we consider context, types, and terms where O does not occur, then we can drop the axiom for PRF-AX. Indeed, we prove by inspecting the type-checking rules that we never normalise a term in the type-checking algorithms (except for deciding equalities); hence we do not need to type-check O if it does not occur in the term being checked.

Corollary 7. *The type-checking algorithm is correct (by Cor. 2) 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.

6.1 Future work

Mention Sozeau's work, use $\mathsf{Prf}\ A$ to model uniformity (as in [**?**,**?**], this was suggested by Baas), using the same technique for adding more equalities (this is Miguel thesis, hopefully), relation with program extraction facilities (adding Prop?), how to handle functions defined by pattern-matching (cite the calculus of definitions by Thierry?). Prove that the model is a CwF(?).

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. Berardi, S.: About the sets-as-propositions embedding of HOL in CC (2004)
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. Blass, A.: The interaction between category theory and set theory. Contemporary Mathematics (1984)
12. Bruijn, N. G. d.: Some extensions of Automath : the AUT-4 family (1994)
13. Cartmell, J.: Generalised algebraic theories and contextual categories. Annals of Pure and Applied Logic (1986) 32–209
14. Coquand, T.: An algorithm for type-checking dependent types. Science of Computer Programming **26** (1996) 167–177
15. Coquand, T., Pollack, R., Takeyama, M.: A logical framework with dependently typed records. Fundam. Inform. **65** (2005) 113–134
16. Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. The Journal of Symbolic Logic **65** (2000) 525–549
17. 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
18. Grothendieck, A.: Séminaire de Géométrie Algébrique du Bois Marie - 1963-64 - Théorie des topos et cohomologie étale des schémas, volume 269 of Lecture notes in mathematics. Springer-Verlag, Berlin; New York (1972)
19. Harper, R., Honsell, F., Plotkin, G.: A Framework for Defining Logics. Journal of the Association of Computing Machinery **40** (1993) 143–184
20. INRIA: The Coq Proof Assistant, Version 8.1. INRIA (2007). http://coq.inria.fr/
21. 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
22. Martin-Löf, P.: An Intuitionistic Theory of Types. Technical report (1972)
23. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
24. Martin-Löf, P.: Normalization by evaluation and by the method of computability (2004). Talk at JAIST, Japan Advanced Institute of Science and Technology, Kanazawa
25. 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
26. Mitchell, J. C., Moggi, E.: Kripke-Style models for typed lambda calculus. In: LICS (1987), 303–314
27. Nordström, B., Petersson, K., Smith, J. M.: Programming in Martin Löf's Type Theory: An Introduction. Clarendon Press, Oxford (1990)
28. 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)
29. Sambin, G., Valentini, S.: Building up a toolbox for Martin-Löf's type theory: subset theory. In: Twenty-five years of constructive type theory (Venice, 1995), volume 36, chapter Oxford Logic Guides. Oxford University Press, New York (1998), 221–244
30. 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

31. 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
32. Sozeau, M.: Un environnement pour la programmation avec types dépendants. Ph.D. thesis, Université Paris 11, Orsay, France (2008)
33. Stone, C. A., Harper, R.: Extensional equivalence and singleton types. ACM Trans. Comput. Logic **7** (2006) 676–722
34. Werner, B.: On the strength of proof-irrelevant type theories. Logical Meth. in Comput. Sci. **4** (2008)

A Normalisation by evaluation

```
data Term = U                -- Universe
          | Fun Term Term    -- dependent function space
          | Singl Term Term  -- Singleton type ({a}_A)
          | Term :@ Term     -- application
          | Lam Term         -- abstraction
          | Q                -- variable
          | Sub Term Subst   -- substitution
          | Sigma Term Term  -- dependent pair type
          | Fst Term         -- first projection
          | Snd Term         -- second projection
          | Pair Term Term   -- dependent pair
          | Nat              -- Naturals
          | Zero             --
          | Suc Term         --
          | Rec Term Term Term Term -- Elimination for Nat
          | Prf Term         -- Proof (with Proof-Irrelevance)
          | Box Term         -- A term in Prf
          | Oh               -- Canonical element of Prf t
          | Where Term Term Term
          | Enum Int         -- Enum n has n elements
          | Const Int Int    --
          | Elim Int Term [Term] Term -- Elimination for (Enum n)
            deriving (Eq,Show)

data Subst = E                -- empty substitution
           | Is               -- identity substitution
           | Ext Subst Term   -- extension
           | P                -- weakening
           | Comp Subst Subst  -- composition
             deriving (Eq,Show)

data D = T                    -- terminal object (empty context)
       | Ld (D -> D)          -- curryfication
       | FunD D (D -> D)       -- dependent products
```

```
        | UD                    -- Universe
        | SingD D D             -- Singleton
        | PairD D D              -- context comprehension
        | Vd Int                -- variables
        | AppD D D              -- neutrals
        | SumD D (D -> D)       -- dependent products
        | Proj Bool D           -- projections of neutrals
        | NatD
        | ZeroD
        | SucD D
        | RecD (D -> D) D D D
        | PrfD D
        | StarD
        | EnumD Int
        | ConstD Int Int
        | ElimD Int (D -> D) [D] D


type Ctx = [Term]

pi1,pi2 :: D -> D
pi1 (PairD d d') = d
pi2 (PairD d d') = d'

app :: D -> D
app (PairD (Ld f) d) = f d

ap :: D -> D -> D
ap f d = app (PairD f d)

neutralD :: D -> Bool
neutralD (Vd _) = True
neutralD (AppD _ _) = True
neutralD (Proj _ _) = True
neutralD (RecD _ _ _ _) = True
neutralD (ElimD _ _ _ _) = True
neutralD _ = False

dec :: (D -> D) -> Bool -> D -> D
dec f b (PairD d d')  = f (PairD d d')
dec f b d | neutralD d = Proj b d

p = dec pi1 False
q = dec pi2 True
```

```
rec :: (D -> D) -> D -> D -> D -> D
rec b z s ZeroD    = z
rec b z s (SucD e) = (s `ap` e) `ap` (rec b z s e)
rec b z s d | neutralD d = up (b d)
                           (RecD (\e -> (downT (b e)))
                              (down (b ZeroD) z)
                              (down
                               (FunD NatD (\n -> FunD (b n) (\e -> b (SucD n)))) s)
                              d)

downs :: Int -> (D -> D) -> [D] -> Int -> [D]
downs _ _ []     _ = []
downs n f (d:ds) i = (down (f (ConstD n i)) d):(downs n f ds (i+1))

elim :: Int -> (D -> D) -> [D] -> D -> D
elim n b ds (ConstD m i) | n == m && i < n = ds!!i
elim n b ds d | neutralD d = up (b d)
                                (ElimD n (\e -> (downT (b e)))
                                       (downs n b ds 0)
                                       d
                                )
up :: D -> D -> D
up (SingD a x) k = a
up (FunD a f)  k = Ld (\d -> up (f d) (AppD k (down a d)))
up (SumD a f)  k = PairD (up a (p k)) (up (f (up a (p k))) (q k))
up NatD        k = k
up (PrfD a)    k = StarD
up d           k = k

down :: D -> D -> D
down UD d          = downT d
down (SingD a x) d = down x a
down (FunD a f)  d = Ld (\e -> down (f (up a e)) (d `ap` (up a e)))
down (SumD a b)  d = PairD (down a (p a)) (down (b (p d)) (q d))
down NatD        (SucD e) = SucD (down NatD e)
down (PrfD a)    d = StarD
down d           e = e

downT (SingD a x) = SingD (down x a) (downT x)
downT (FunD a f)  = FunD (downT a) (\d -> downT (f (up a d)))
downT (SumD a b)  = SumD (downT a) (\d -> downT (b (up a d)))
downT (PrfD a)    = PrfD (downT a)
downT d           = d

readback :: Int -> D -> Term
```

```
readback i UD             = U
readback i (FunD a f)     = Fun (readback i a) $ readback (i+1) (f (Vd i))
readback i (SingD a x)    = Singl (readback i a) (readback i x)
readback i (Ld f)         = Lam $ readback (i+1) (f (Vd i))
readback i (Vd n)         = mkvar (i-n-1)
readback i (AppD k d)     = (readback i k) :@ (readback i d)
readback i (Proj False d) = Fst (readback i d)
readback i (Proj True d)  = Snd (readback i d)
readback i (PairD d e)    = Pair (readback i d) (readback i e)
readback i (SumD a b)     = Sigma (readback i a) $ readback (i+1) (b (Vd i))
readback i NatD           = Nat
readback i ZeroD          = Zero
readback i (SucD e)       = Suc (readback i e)
readback i (RecD b z s e) = Rec (Fun Nat (readback (i+1) (b (Vd i))))
                            (readback i z)
                            (readback i s)
                            (readback i e)

readback i (PrfD d)        = Prf (readback i d)
readback i StarD           = Oh
readback i (EnumD n)       = Enum n
readback i (ConstD n j)    = Const n j
readback i (ElimD n b ds d) = Elim n (readback (i+1) (b (Vd i)))
                                (map (readback i) ds) (readback i d)

-- Evaluation

eval :: Term -> D -> D
eval U           d = UD
eval (Fun t f)   d = FunD (eval t d) (\d' -> eval f (PairD d d'))
eval (Singl t a) d = SingD (eval t d) (eval a d)
eval (Lam t)     d = Ld (\d' -> eval t (PairD d d'))
eval (t :@ r)    d = (eval t d) `ap` (eval r d)
eval Q           d = pi2 d
eval (Sub t s)   d = eval t (evalS s d)

eval (Sigma t r) d = SumD (eval t d) (\e -> eval r (PairD d e))
eval (Fst t)     d = p (eval t d)
eval (Snd t)     d = q (eval t d)
eval (Pair t r)  d = PairD (eval t d) (eval r d)

eval Nat         d = NatD
eval Zero        d = ZeroD
eval (Suc t)     d = SucD (eval t d)
eval (Rec b z s t) d = rec (\e -> eval b (PairD d e))
```

```
                          (eval z d)
                          (eval s d)
                          (eval t d)


eval (Prf t)     d = PrfD (eval t d)
eval (Box t)     d = eval t d
eval Oh          d = StarD
eval (Where t b p) d   = eval b (PairD d (eval t d))
eval (Enum n)        d = EnumD n
eval (Const n i)     d = ConstD n i
eval (Elim n b ts t) d = elim n (\e -> eval b (PairD d e))
                             (map ((flip eval) d) ts) (eval t d)

evalS :: Subst -> D -> D
evalS E           d = T
evalS Is          d = d
evalS (Ext s t)   d = PairD (evalS s d) (eval t d)
evalS P           d = pi1 d
evalS (Comp s s') d = (evalS s . evalS s') d

nbe :: Term -> Term -> Term
nbe ty t = readback 0 . down (eval ty T) $ eval t T

nbeTy :: Term -> Term
nbeTy ty = readback 0 $ downT (eval ty T)

nbeOpen :: Ctx -> Term -> Term -> Term
nbeOpen ctx ty t = readback n . down (eval ty env) $ eval t env
    where n   = length ctx
          env = mkenv n ctx T



nbeOpenTy :: Ctx -> Term -> Term
nbeOpenTy ctx ty = readback n $ downT (eval ty env)
    where n   = length ctx
          env = mkenv n ctx T


mkenv :: Int -> Ctx -> D -> D
mkenv 0 []     d = d
mkenv n (t:ts) d = PairD d' (up td (Vd (n-1)))
    where d' = mkenv (n-1) ts d
          td = eval t d'
```

```
mkvar :: Int -> Term
mkvar n | n == 0    = Q
        | otherwise = Sub Q $ subs (n-1)

subs n | n == 0    = P
subs n | otherwise = Comp P $ subs (n-1)
```

B Type-checking algorithm

Type checking algorithm for normal forms, and type inference algorithm for neutral terms.

Checking contexts

```
chkCtx :: Ctx -> Bool
chkCtx [] = True
chkCtx (t:ts) = chkCtx ts && chkType ts t
```

Checking types

```
chkType :: Ctx -> Term -> Bool
chkType ts U                = True
chkType ts (Fun t r)        = chkType ts t && chkType (t:ts) r
chkType ts (Singl a t)      = chkType ts t && chkTerm ts t a
chkType ts (Sigma t r)      = chkType ts t && chkType (t:ts) r
chkType ts Nat              = True
chkType ts (Prf t)          = chkType ts t
chkType ts (Enum n)         = True
chkType ts Q                = chkNeTerm ts U Q
chkType ts w@(Sub Q s)      = chkNeTerm ts U w
chkType ts w@(k :@ v)       = chkNeTerm ts U w
chkType ts w@(Fst k)        = chkNeTerm ts U w
chkType ts w@(Snd k)        = chkNeTerm ts U w
chkType ts w@(Rec t' v v' k) = chkNeTerm ts U w
chkType _ _ = False
```

Checking the types of terms

```
idsub :: Term -> Term -> Term
idsub t t' = Sub t (Ext Is t')
```

```
chkTerm :: Ctx -> Term -> Term -> Bool
chkTerm ts U           (Fun t t')   = chkTerm ts U t &&
                                      chkTerm (t:ts) U t'
chkTerm ts U           (Singl e t)  = chkTerm ts U t &&
                                      chkTerm ts t e
chkTerm ts U           (Sigma t t') = chkTerm ts U t &&
                                      chkTerm (t:ts) U t'
chkTerm ts U            Nat         = True
chkTerm ts (Fun t t')  (Lam e)      = chkTerm (t:ts) t' e
chkTerm ts (Singl e t) e'           = chkTerm ts (nbeOpenTy ts t) e' &&
                                      (nbeOpen ts e t) == (nbeOpen ts e' t)
chkTerm ts (Sigma t r) (Pair e e')  = chkTerm ts t e &&
                                      chkTerm ts (nbeOpenTy ts (idsub r e)) e'
chkTerm ts Nat         Zero         = True
chkTerm ts Nat         (Suc t)      = chkTerm ts Nat t
chkTerm ts (Prf t)     (Box e)      = chkTerm ts t e
chkTerm ts (Enum n)    (Const m i)  = m == n && i < n
chkTerm ts t e | neutral e          = chkNeTerm ts t e
chkTerm _ _ _ = False

neutral :: Term -> Bool
neutral Q               = True
neutral (Sub Q s)       = True
neutral (k :@ v)        = True
neutral (Fst k)         = True
neutral (Snd k)         = True
neutral (Rec t' v v' k) = True
neutral (Elim n b ts t) = True
neutral (Where b t p)   = True
neutral _ = False

maybeEr :: Maybe Term -> Maybe Term
maybeEr = maybe Nothing (Just . erase)

chkNeTerm :: Ctx -> Term -> Term -> Bool
chkNeTerm ts t e = case maybeEr (infType ts e) of
                     Just t' -> t == t'
                     Nothing -> False
```

Inferring the types of neutral terms

```
nbeType :: Ctx -> Term -> Maybe Term
nbeType ctx t = Just (nbeOpenTy ctx t)

infType :: Ctx -> Term -> Maybe Term
```

```
infType (t:ts) Q         = nbeType (t:ts) (Sub t P)
infType ts     (Sub Q s) = case infType (infCtx ts s) Q of
                               Just t -> nbeType ts (Sub t s)
                               _ -> Nothing

infType ts (e :@ e') = case maybeEr (infType ts e) of
                           Just (Fun t t') ->
                               if chkTerm ts t e'
                               then nbeType ts (idsub t' e')
                               else Nothing
                           _ -> Nothing

infType ts (Fst e)   = case maybeEr(infType ts e) of
                           Just (Sigma t t') ->  Just t
                           _ -> Nothing
infType ts (Snd e)   = case maybeEr (infType ts e) of
                           Just (Sigma t t') -> nbeType ts (idsub t' (Fst e))
                           _ -> Nothing
infType ts (Rec t v w k) = case maybeEr (infType ts k) of
                               Just Nat -> if
                                   chkType (Nat:ts) t &&
                                   chkTerm ts (nbeOpenTy ts (idsub t Zero)) v &&
                                   chkTerm (Nat:ts)
                                             (Fun (idsub t Q)
                                                  (idsub t (Suc (Sub Q P)))) w
                                   then nbeType ts (idsub t k)
                                   else Nothing
                               _ -> Nothing
infType ts (Where t b k) = case maybeEr (infType ts k) of
                               Just (Prf t') -> if chkType ts t &&
                                                   chkTerm (t':ts) t b
                                                then Just (Prf t)
                                                else Nothing
                               _ -> Nothing
infType ts (Elim n b cs k) = case maybeEr (infType ts k) of
                                 Just (Enum m) -> if m == n &&
                                                     chkType (Enum n:ts) b &&
                                                     chkList ts n b 0 cs
                                                  then nbeType ts (idsub b k)
                                                  else Nothing
                                 _ -> Nothing

infType _ _ = Nothing

chkList :: Ctx -> Int -> Term -> Int -> [Term] -> Bool
```

```
chkList ts _ _ _ []     = True
chkList ts n b i (e:es) = chkTerm ts (nbeOpenTy ts (idsub b (Const n i))) e &&
                          chkList ts n b (i+1) es

infCtx :: Ctx -> Subst -> Ctx
infCtx  (t:ts) P          = ts
infCtx  (t:ts) (Comp P s) = infCtx ts s

erase :: Term -> Term
erase (Singl e t) = erase t
erase t           = t
```