

NORMALISATION BY EVALUATION FOR TYPE THEORY, IN TYPE THEORY

THORSTEN ALTENKIRCH AND AMBRUS KAPOSI

School for Computer Science, University of Nottingham, Nottingham, United Kingdom
e-mail address: txa@cs.nott.ac.uk

Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest,
Hungary
e-mail address: akaposi@inf.elte.hu

ABSTRACT. We develop normalisation by evaluation (NBE) for dependent types based on presheaf categories. Our construction is formulated in the metalanguage of type theory using quotient inductive types. We use a typed presentation hence there are no preterms or realizers in our construction, and every construction respects the conversion relation. NBE for simple types uses a logical relation between the syntax and the presheaf interpretation. In our construction, we merge the presheaf interpretation and the logical relation into a proof-relevant logical predicate. We prove normalisation, completeness, stability and decidability of definitional equality. Most of the constructions were formalized in Agda.

1. INTRODUCTION

Normalisation by evaluation (NBE) is a technique to compute normal forms of typed λ -terms by evaluating them in an appropriate semantics. The idea was pioneered by Schwichtenberg and Berger [14], subsequently a categorical account using presheaf categories was given [8] and this approach was extended to System F [9, 10] and coproducts [7].

In the present paper we extend NBE to a basic type theory with dependent types which has Π -types and an uninterpreted family using a presheaf interpretation. We take advantage of our recent work on an intrinsic representation of type theory in type theory [13] which only defines typed objects avoiding any reference to untyped preterms or typing relations and which forms the basis of our formal development in Agda.

The present paper is an expanded version of our conference paper [12]. In particular we show here for the first time that our normalisation construction implies decidability of equality. This isn't entirely obvious because our normal forms are indexed by contexts and types of which it is a priori not known whether equality is decidable. However, we observe that mimicking the bidirectional approach to type checking [17] we can actually decide equality

Key words and phrases: normalisation by evaluation, dependent types, internal type theory, logical relations, Agda.

This research was supported by EPSRC grant EP/M016951/1, USAF grant FA9550-16-1-0029 and COST Action EUTypes CA15123.

of normal forms and hence, after combining it with normalisation, we obtain decidability for conversion.

1.1. Specifying normalisation. Normalisation can be given the following specification.

We denote the type of well typed terms of type A in context Γ by $\mathsf{Tm}\ \Gamma\ A$. We are not interested in preterms, all of our constructions will be well-typed. In addition, this type is defined as a quotient inductive type (QIT, see [13]) which means that terms are quotiented with the conversion relation. It follows that on one hand if two terms $t, t' : \mathsf{Tm}\ \Gamma\ A$ are convertible then they are equal: $t \equiv_{\mathsf{Tm}\ \Gamma\ A} t'$. On the other hand, the eliminator of $\mathsf{Tm}\ \Gamma\ A$ ensures that every function defined from this type respects the conversion relation. This enforces a high level of abstraction when reasoning about the syntax: all of our constructions need to respect convertibility as well.

The type of normal forms is denoted $\mathsf{Nf}\ \Gamma\ A$ and there is an embedding from it to terms $\ulcorner - \urcorner : \mathsf{Nf}\ \Gamma\ A \rightarrow \mathsf{Tm}\ \Gamma\ A$. Normal forms are defined as a usual inductive type (as opposed to quotient inductive types).

Normalisation is given by a function `norm` which takes a term to a normal form. It needs to be an isomorphism:

$$\text{completeness } \smile \quad \text{norm } \downarrow \frac{\mathsf{Tm}\ \Gamma\ A}{\mathsf{Nf}\ \Gamma\ A} \quad \uparrow \ulcorner - \urcorner \quad \smile \text{ stability}$$

If we normalise a term, we obtain a term which is convertible to it: $t \equiv \ulcorner \text{norm } t \urcorner$. This is called completeness. The other direction is called stability: $n \equiv \text{norm } \ulcorner n \urcorner$. It expresses that there is no redundancy in the type of normal forms. Soundness, that is, if $t \equiv t'$ then $\text{norm } t \equiv \text{norm } t'$ is given by congruence of equality.

1.2. NBE for simple type theory. Normalisation by evaluation (NBE) is one way to implement this specification. It works by a complete model construction (figure 1). We define a model of the syntax and hence the eliminator gives us a function from the syntax to the model. Then we define a quote function which is a map from the model back to the syntax, but it targets normal forms (a subset of the syntax via the operator $\ulcorner - \urcorner$).

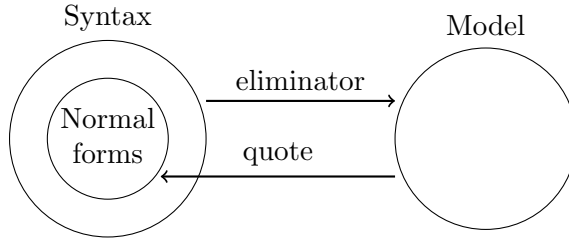


Figure 1: Normalisation by evaluation.

In this subsection, we summarize the approach of [8] for NBE for simple types. Here the model we choose is a presheaf model. Presheaf models are proof-relevant versions of Kripke models (possible world semantics) for intuitionistic logic: they are parameterised over a category instead of a poset. The category that we choose here is the category of renamings REN . The objects in REN are contexts and morphisms are variable renamings. The presheaf model interprets contexts and types as presheaves, e.g. the interpretation of A

denoted $\llbracket A \rrbracket$ is a $\text{REN}^{\text{op}} \rightarrow \text{Set}$ functor. Terms and substitutions are natural transformations between the corresponding presheaves, e.g. for $t : \text{Tm } \Gamma A$ we have a natural transformation $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$. A function type is interpreted as the presheaf exponential (a function for all future worlds), the base type is interpreted as normal forms of the base type.

Because REN has contexts as objects, we can embed types into presheaves (Yoneda embedding): a type A is embedded into the presheaf TM_A by setting $\text{TM}_A \Psi = \text{Tm } \Psi A$ i.e. a type at a given context is interpreted as the set of terms of that type in that context. Analogously, we can embed a type A into the presheaves of normal forms NF_A and neutral terms NE_A . Normal forms are terms with no redexes (they include neutral terms) while neutral terms are either variables or an eliminator applied to a neutral term.

The quote function is defined by induction on types as a natural transformation $q_A : \llbracket A \rrbracket \rightarrow \text{NF}_A$. Quote is defined mutually with unquote which maps neutral terms into semantic elements: $u_A : \text{NE}_A \rightarrow \llbracket A \rrbracket$.

To normalise a term, we also need to define unquote for neutral substitutions (lists of neutral terms). Then we get normalisation by calling unquote on the identity neutral substitution, then interpreting the term at this semantic element and finally quoting.

$$\text{norm}_A (t : \text{Tm } \Gamma A) : \text{Nf } \Gamma A := q_A (\llbracket t \rrbracket (u_\Gamma \text{id}))$$

We can prove completeness using a logical relation R between TM and the presheaf model. The logical relation is equality at the base type. We extend quote and unquote to produce witnesses and require a witness of this logical relation, respectively. This is depicted in figure 2. The commutativity of the right hand triangle gives completeness: starting with a term, a semantic value and a witness that these are related, we get a normal form, and then if we embed it back into terms, we get a term equal to the one we started with.

Stability can be proven by mutual induction on terms and normal forms.

$$\begin{array}{ccccc}
 \text{NE}_A & \xrightarrow{u'_A} & \Sigma (\text{TM}_A \times \llbracket A \rrbracket) R_A & \xrightarrow{q'_A} & \text{NF}_A \\
 & \searrow \ulcorner _ \urcorner & \downarrow \text{proj} & \swarrow \ulcorner _ \urcorner & \\
 & & \text{TM}_A & &
 \end{array}$$

Figure 2: The type of quote and unquote for a type A in NBE for simple types. We use primed notations for the unquote and quote functions to denote that they include the completeness proof. This is a diagram in the category of presheaves.

A nice property of this normalisation proof is that the part of unquote (and quote) which gives (and uses) $\llbracket A \rrbracket$ can be defined separately from the part which gives relatedness. This means that the normalisation function can be defined independently from the proof that it is complete.

1.3. NBE for type theory. In this subsection, we explain why the naive generalisation of the proof for the simply typed case does not work in the presence of dependent types and how we solve this problem.

In the case of simple type theory, types are closed, so they are interpreted as presheaves just as contexts. When we have dependent types, types depend on contexts, hence they are interpreted as families of presheaves in the presheaf model (we omit functoriality).

$$\begin{aligned} \llbracket \Gamma \rrbracket & : |\mathbf{REN}| \rightarrow \mathbf{Set} \\ \llbracket \Gamma \vdash A \rrbracket : (\Psi : |\mathbf{REN}|) & \rightarrow \llbracket \Gamma \rrbracket \Psi \rightarrow \mathbf{Set} \end{aligned}$$

We can declare quote for contexts the same way as for simple types, but quote for types has to be more subtle. Our first candidate is the following where it depends on quote for contexts (we omit the naturality properties).

$$\begin{aligned} \mathbf{q}_\Gamma & : (\Psi : |\mathbf{REN}|) \rightarrow \llbracket \Gamma \rrbracket \Psi \rightarrow \mathbf{Tms} \Psi \Gamma \\ \mathbf{q}_{\Gamma \vdash A} & : (\Psi : |\mathbf{REN}|)(\alpha : \llbracket \Gamma \rrbracket \Psi) \rightarrow \llbracket A \rrbracket_\Psi \alpha \rightarrow \mathbf{Nf} \Psi (A[\mathbf{q}_{\Gamma, \Psi} \alpha]) \end{aligned}$$

The type of unquote also depends on quote for contexts.

$$\mathbf{u}_{\Gamma \vdash A} : (\Psi : |\mathbf{REN}|)(\alpha : \llbracket \Gamma \rrbracket \Psi) \rightarrow \mathbf{Ne} \Psi (A[\mathbf{q}_{\Gamma, \Psi} \alpha]) \rightarrow \llbracket A \rrbracket \Psi \alpha$$

When we try to define quote and unquote following this specification, we observe that we need some new equations to typecheck our definition. E.g. quote for function types needs that quote after unquote is the identity up to embedding: $\ulcorner - \urcorner \circ \mathbf{q}_A \circ \mathbf{u}_A \equiv \ulcorner - \urcorner$. This is however the consequence of the logical relation between the syntax and the presheaf model: we can read it off figure 2 by the commutativity of the diagram: if we embed a neutral term into terms, it is the same as unquoting, then quoting, then embedding.

Hence, our second attempt is defining quote and unquote mutually with their correctness proofs. It is not very surprising that when moving to dependent types the well-typedness of normalisation depends on completeness. The types of quote and unquote become the following.

$$\begin{aligned} \mathbf{q}_{\Gamma \vdash A} & : (\Psi : |\mathbf{REN}|)(\rho : \mathbf{Tms} \Psi \Gamma)(\alpha : \llbracket \Gamma \rrbracket \Psi)(p : \mathbf{R}_\Gamma \Psi \rho \alpha) \\ & (t : \mathbf{Tm} \Psi A[\rho])(v : \llbracket A \rrbracket_\Psi \alpha) \rightarrow \mathbf{R}_{A \Psi p} t v \rightarrow \Sigma(n : \mathbf{NF}_A \rho). t \equiv \ulcorner n \urcorner \\ \mathbf{u}_{\Gamma \vdash A} \Psi \rho \alpha p & : (n : \mathbf{Ne} \Psi A[\rho]) \rightarrow \Sigma(v : \llbracket A \rrbracket_\Psi \alpha). \mathbf{R}_{A \Psi p} \ulcorner n \urcorner v \end{aligned}$$

However there seems to be no way to define quote and unquote this way because quote does not preserve the logical relation. The problem is that when defining unquote at Π we need to define a semantic function which works for arbitrary inputs, not only those which are related to a term. The first component of unquote at Π has the following type.

$$\begin{aligned} & \mathbf{proj}_1 \left(\mathbf{u}_{\Gamma \vdash \Pi A B} \Psi \rho \alpha p (n : \mathbf{Ne} \Psi (\Pi A B)[\rho]) \right) \\ & : \forall \Omega. (\beta : \mathbf{REN}(\Omega, \Psi))(x : \llbracket A \rrbracket_\Omega (\llbracket \Gamma \rrbracket \beta \alpha)) \rightarrow \llbracket B \rrbracket_\Omega (\llbracket \Gamma \rrbracket \beta \alpha, x) \end{aligned}$$

We should define this as unquoting the application of the neutral function n and quoting the input x . However we can't quote an arbitrary semantic x , we also need a witness that it is related to a term. It seems that we have to restrict the presheaf model to only contain semantic elements which are related to some term.

Indeed, this is our solution: we merge the presheaf model and the logical relation into a single proof-relevant logical predicate. We denote the logical predicate at a context Γ by \mathbf{P}_Γ . We define normalisation following the diagram in figure 3.

In the presheaf model, the interpretation of the base type was normal forms at the base type and the logical relation at the base type was equality of the term and the normal form. In our case, the logical predicate at the base type will say that there exists a normal

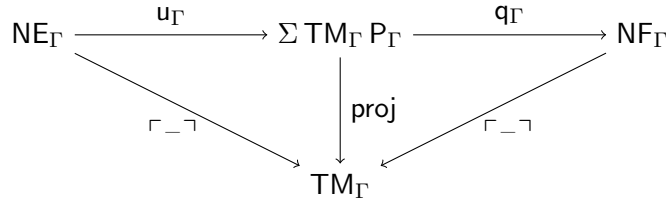


Figure 3: The types of quote and unquote for a context Γ in our proof.

form which is equal to the term (this is why it needs to be proof-relevant). This solves the problem mentioned before: now the semantics of a term will be the same term together with a witness of the predicate for that term.

1.4. Structure of the proof and the paper. In this subsection, we give a high level sketch of the proof. Sections 3, 4, 5, 7 are fully formalised in Agda, the computational parts of sections 6, 8 and 9 are formalised, but some of the naturality and functoriality properties are left as holes. The formalisation is available online [11]. The proofs are available in full detail on paper (including everything that we omitted in this paper and which is not finished in the formalisation) in the second author’s thesis [25].

In section 2 we briefly summarize the metatheory we are working in.

In section 3 we define the syntax for type theory as a quotient inductive inductive type (QIIT) [13]. The arguments of the eliminator for the QIIT form a model of type theory.

In section 4 we prove injectivity of context extension and the type formers EI and Π . We will need these for proving decidability of equality for normal forms.

In section 5 we define the category of renamings REN : objects are contexts and morphisms are renamings.

In section 6 we define the proof-relevant presheaf logical predicate interpretation of the syntax. The interpretation has REN as the base category and two parameters for the interpretations of U and EI . This interpretation can be seen as a dependent version of the presheaf model of type theory. E.g. a context in the presheaf model is interpreted as a presheaf. Now it is a family of presheaves dependent on a substitution into that context. The interpretations of base types can depend on the actual elements of the base types. The interpretation of substitutions and terms are what are usually called fundamental theorems.

In section 7 we define neutral terms and normal forms together with their renamings and embeddings into the syntax ($\ulcorner _ \urcorner$). With the help of these, we define the interpretations of U and EI . The interpretation of U at a term of type U will be a neutral term of type U which is equal to the term. We also prove decidability of equality for normal forms.

In section 8 we mutually define the natural transformations quote and unquote. We define them by induction on contexts and types as shown in figure 3. Quote takes a term and a semantic value at that term into a normal term and a proof that the normal term is equal to it. Unquote takes a neutral term into a semantic value at the neutral term.

Finally, in section 9, we put together the pieces by defining the normalisation function and showing that it is complete and stable. In addition, we show decidability of equality and consistency.

1.5. Related work. Normalisation by evaluation was first formulated by Schwichtenberg and Berger [14], subsequently a categorical account using presheaf categories was given [8] and this approach was extended to System F [9,10] and coproducts [7]. The present work can be seen as a continuation of this line of research. A fully detailed description of our proof can be found in the PhD thesis of the second author [25].

The term normalisation by evaluation is also more generally used to describe semantic based normalisation functions. E.g. Danvy is using semantic normalisation for partial evaluation [20]. Normalisation by evaluation using untyped realizers has been applied to dependent types by Abel et al. [2–4]. Danielsson [19] has formalized NBE for dependent types but he doesn’t prove soundness of normalisation.

Our proof of injectivity of type formers is reminiscent in [22] and the proof of decidability of normal forms is similar to that of [5].

2. METATHEORY AND NOTATION

We are working in intensional Martin-Löf Type Theory with postulated extensionality principles using Agda as a vehicle [1,27]. We make use of quotient inductive inductive types (QIITs, see section 6 of [13]). QIITs are a combination of inductive inductive types [26] and higher inductive types [29]. The metatheory of QIITs is not developed yet, however we hope that they can be justified by a setoid model [6]. We only use one instance of a QIIT, the definition of the syntax. We extend Agda with this QIIT using axioms and rewrite rules [15]. The usage of rewrite rules guarantees that injectivity and disjointness of constructors of the QIIT are not available to the unification mechanisms of Agda. Also, pattern matching on constructors of the QIIT is not available, the only way to define a function from the QIIT is to use the eliminator.

When defining an inductive type A , we first declare the type by `data A : S` where S is the sort, then we list the constructors. For inductive inductive types we first declare all the types, then following a second `data` keyword we list the constructors. We also postulate functional extensionality which is a consequence of having an interval QIIT anyway. We assume \mathbf{K} , that is, we work in a strict type theory.

We follow Agda’s convention of denoting the universe of types by `Set`, we write function types as $(x : A) \rightarrow B$ or $\forall x.B$, implicit arguments are written in curly braces $\{x : A\} \rightarrow B$ and can be omitted or given in curly braces or lower index. If some arguments are omitted, we assume universal quantification, e.g. $(y : B x) \rightarrow C$ means $\forall x.(y : B x) \rightarrow C$ if x is not given in the context. We write $\Sigma(x : A).B$ for Σ types. We overload names e.g. the action on objects and morphisms of a functor is denoted by the same symbol.

The identity type (propositional equality) is denoted $- \equiv -$ and its constructor is `refl`. Transport of a term $u : P a$ along an equality $p : a \equiv a'$ is denoted $_{p*}u : P a'$. We denote $(_{p*}u) \equiv u'$ by $u \equiv^p u'$. We write `ap` for congruence, that is `ap f p : f a \equiv f a'` if $p : a \equiv a'$. We write $- \cdot -$ for transitivity and $-^{-1}$ for symmetry of equality. For readability, we will omit writing transports in the informal presentation most of the time, that is, our informal notation is that of extensional type theory. This choice is justified by the conservativity of extensional type theory over intensional type theory with \mathbf{K} and functional extensionality [23,28]. This allows writing e.g. $f a$ where $f : A \rightarrow B$ and $a : A'$ in case there is an equality in scope which justifies $A \equiv A'$.

Sometimes we use Coq-style definitions: we write `d (x : A) : B := t` for defining `d` of type $(x : A) \rightarrow B$ by $\lambda x.t$. We also use Agda-style pattern matching definitions. We use the

underscore $_$ to denote arguments that we don't need e.g. the constant function is written $\text{const } x _ := x$.

3. OBJECT THEORY

The object theory is a basic type theory with dependent function space, an uninterpreted base type \mathbf{U} and an uninterpreted family over this base type \mathbf{El} . We use intrinsic typing (that is, we only define well typed terms, term formers and derivation rules are indented), and we present the theory as a QIIT, that is, we add conversion rules as equality constructors. We define an explicit substitution calculus, hence substitutions are part of the syntax and the syntax is purely inductive (as opposed to inductive recursive). For a more detailed presentation, see [25].

The syntax constitutes of contexts, types, substitutions and terms. We declare the QIIT of the syntax as follows.

```

data Con : Set
data Ty   : Con → Set
data Tms : Con → Con → Set
data Tm   : (Γ : Con) → Ty Γ → Set
    
```

We use the convention of naming contexts Γ, Δ, Θ , types A, B , terms t, u , substitutions σ, ν, δ .

The point constructors are listed in the left column and the equality constructors in the right.

data	data
\cdot : Con	$[\text{id}]$: $A[\text{id}] \equiv A$
$-, -$: $(\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Con}$	$[\]$: $A[\sigma][\nu] \equiv A[\sigma \circ \nu]$
$-[-]$: $\text{Ty } \Delta \rightarrow \text{Tms } \Gamma \Delta \rightarrow \text{Ty } \Gamma$	$\mathbf{U}[\]$: $\mathbf{U}[\sigma] \equiv \mathbf{U}$
\mathbf{U} : Ty Γ	$\mathbf{El}[\]$: $(\mathbf{El } \hat{A})[\sigma] \equiv \mathbf{El} (\mathbf{U}[\]_* \hat{A}[\sigma])$
\mathbf{El} : Tm $\Gamma \mathbf{U} \rightarrow \text{Ty } \Gamma$	$\mathbf{\Pi}[\]$: $(\mathbf{\Pi } A B)[\sigma] \equiv \mathbf{\Pi} (A[\sigma]) (B[\sigma \uparrow A])$
$\mathbf{\Pi}$: $(A : \text{Ty } \Gamma) \rightarrow \text{Ty } (\Gamma, A) \rightarrow \text{Ty } \Gamma$	$\text{id} \circ$: $\text{id} \circ \sigma \equiv \sigma$
id : Tms $\Gamma \Gamma$	$\circ \text{id}$: $\sigma \circ \text{id} \equiv \sigma$
$- \circ -$: Tms $\Theta \Delta \rightarrow \text{Tms } \Gamma \Theta \rightarrow \text{Tms } \Gamma \Delta$	$\circ \circ$: $(\sigma \circ \nu) \circ \delta \equiv \sigma \circ (\nu \circ \delta)$
ϵ : Tms $\Gamma \cdot$	$\epsilon \eta$: $\{\sigma : \text{Tms } \Gamma \cdot\} \rightarrow \sigma \equiv \epsilon$
$-, -$: $(\sigma : \text{Tms } \Gamma \Delta) \rightarrow \text{Tm } \Gamma A[\sigma] \rightarrow \text{Tms } \Gamma (\Delta, A)$	$\pi_1 \beta$: $\pi_1 (\sigma, t) \equiv \sigma$
π_1 : Tms $\Gamma (\Delta, A) \rightarrow \text{Tms } \Gamma \Delta$	$\pi \eta$: $(\pi_1 \sigma, \pi_2 \sigma) \equiv \sigma$
$-[-]$: Tm $\Delta A \rightarrow (\sigma : \text{Tms } \Gamma \Delta) \rightarrow \text{Tm } \Gamma A[\sigma]$	$, \circ$: $(\sigma, t) \circ \nu \equiv (\sigma \circ \nu), ([\]_* t[\nu])$
π_2 : $(\sigma : \text{Tms } \Gamma (\Delta, A)) \rightarrow \text{Tm } \Gamma A[\pi_1 \sigma]$	$\pi_2 \beta$: $\pi_2 (\sigma, t) \equiv^{\pi_1 \beta} t$
lam : Tm $(\Gamma, A) B \rightarrow \text{Tm } \Gamma (\mathbf{\Pi } A B)$	$\mathbf{\Pi} \beta$: $\text{app } (\text{lam } t) \equiv t$
app : Tm $\Gamma (\mathbf{\Pi } A B) \rightarrow \text{Tm } (\Gamma, A) B$	$\mathbf{\Pi} \eta$: $\text{lam } (\text{app } t) \equiv t$
	$\text{lam}[\]$: $(\text{lam } t)[\sigma] \equiv^{\mathbf{\Pi}[\]} \text{lam } (t[\sigma \uparrow A])$

The constructors can be summarized as follows.

- Substitutions form a category with a terminal object. This includes the categorical substitution laws for types $[\text{id}]$ and $[\]$.
- Substitution laws for types $\text{U}[\]$, $\text{El}[\]$, $\text{II}[\]$.
- The laws of comprehension which state that we have the natural isomorphism

$$\pi_1\beta, \pi_2\beta \cup \quad -, - \downarrow \frac{\sigma : \text{Tms } \Gamma \Delta \quad \text{Tm } \Gamma A[\sigma]}{\text{Tms } \Gamma (\Delta, A)} \uparrow \pi_1, \pi_2 \quad \curvearrowright \pi\eta$$

where naturality¹ is given by $, \circ$.

- The laws for function space which are given by the natural isomorphism

$$\text{II}\beta \cup \quad \text{lam} \downarrow \frac{\text{Tm } (\Gamma, A) B}{\text{Tm } \Gamma (\text{II } A B)} \uparrow \text{app} \quad \curvearrowright \text{II}\eta$$

where naturality is given by $\text{lam}[\]$.

Note that the equality $\pi_2\beta$ lives over $\pi_1\beta$. Also, we had to use transport to typecheck $\text{El}[\]$ and $, \circ$. We used lifting of a substitution in the types of $\text{II}[\]$ and $\text{lam}[\]$. It is defined as follows.

$$\begin{aligned} - \uparrow - &: (\sigma : \text{Tms } \Gamma \Delta) \rightarrow \text{Ty } \Delta \rightarrow \text{Tms } (\Gamma, A[\sigma]) (\Delta, A) \\ \sigma \uparrow A &:= (\sigma \circ \pi_1 \text{id}), (\text{II}[\] * \pi_2 \text{id}) \end{aligned}$$

We use the categorical app operator but the usual one $(-\$-)$ can also be derived.

$$\begin{aligned} \langle (u : \text{Tm } \Gamma A) \rangle & \quad \quad \quad : \text{Tms } \Gamma (\Gamma, A) := \text{id}, [\text{id}]^{-1} * u \\ (t : \text{Tm } \Gamma (\text{II } A B)) \$ (u : \text{Tm } \Gamma A) : B[\langle u \rangle] & \quad \quad \quad := (\text{app } t)[\langle u \rangle] \end{aligned}$$

When we define a function from the above syntax, we need to use the eliminator. The eliminator has four motives corresponding to what Con , Ty , Tms and Tm get mapped to and one method for each constructor including the equality constructors. The methods for point constructors are the elements of the motives to which the constructor is mapped. The methods for the equality constructors demonstrate soundness, that is, the semantic constructions respect the syntactic equalities. The eliminator comes in two different flavours: the non-dependent and dependent version. In our constructions we use the dependent version. The motives and methods for the non-dependent eliminator (recursor) collected together form a model of type theory, they are equivalent to Dybjer's Categories with Families [21].

To give an idea of what the eliminator looks like we list its motives and some of its methods. For a complete presentation and an algorithm for deriving these from the constructors, see [25]. As names we use the names of the constructors followed by an upper

¹If one direction of an isomorphism is natural, so is the other. This is why it is enough to state naturality for $-, -$ and not for π_1, π_2 .

index M .

$$\begin{aligned}
 \text{Con}^M & : \text{Con} \rightarrow \text{Set} \\
 \text{Ty}^M & : (\text{Con}^M \Gamma) \rightarrow \text{Ty } \Gamma \rightarrow \text{Set} \\
 \text{Tms}^M & : (\text{Con}^M \Gamma) \rightarrow (\text{Con}^M \Delta) \rightarrow \text{Tms } \Gamma \Delta \rightarrow \text{Set} \\
 \text{Tm}^M & : (\Gamma^M : \text{Con}^M \Gamma) \rightarrow \text{Ty}^M \Gamma^M A \rightarrow \text{Tm } \Gamma A \rightarrow \text{Set} \\
 \cdot^M & : \text{Con}^M . \\
 -,^M - & : (\Gamma^M : \text{Con}^M \Gamma) \rightarrow \text{Ty}^M \Gamma^M A \rightarrow \text{Con}^M (\Gamma, A) \\
 \text{id}^M & : \text{Tms}^M \Gamma^M \Gamma^M \text{id} \\
 - \circ^M - & : \text{Tms}^M \Theta^M \Delta^M \sigma \rightarrow \text{Tms}^M \Gamma^M \Theta^M \nu \rightarrow \text{Tms}^M \Gamma^M \Delta^M (\sigma \circ \nu) \\
 \text{oid}^M & : \sigma^M \circ^M \text{id}^M \equiv^{\text{oid}} \sigma^M \\
 \pi_2 \beta^M & : \pi_2^M (\rho^M, {}^M t^M) \equiv^{\pi_1 \beta^M, \pi_2 \beta} t^M
 \end{aligned}$$

Note that the method equality oid^M lives over the constructor oid while the method equality $\pi_2 \beta^M$ lives both over the method equality $\pi_1 \beta^M$ and the equality constructor $\pi_2 \beta$.

There are four eliminators for the four constituent types. These are understood in the presence of all the motives and methods given above.

$$\begin{aligned}
 \text{ElimCon} & : (\Gamma : \text{Con}) \quad \rightarrow \text{Con}^M \Gamma \\
 \text{ElimTy} & : (A : \text{Ty } \Gamma) \quad \rightarrow \text{Ty}^M (\text{ElimCon } \Gamma) A \\
 \text{ElimTms} & : (\sigma : \text{Tms } \Gamma \Delta) \rightarrow \text{Tms}^M (\text{ElimCon } \Gamma) (\text{ElimCon } \Delta) \sigma \\
 \text{ElimTm} & : (t : \text{Tm } \Gamma A) \quad \rightarrow \text{Tm}^M (\text{ElimCon } \Gamma) (\text{ElimTy } A) t
 \end{aligned}$$

We have the usual β computation rules such as the following.

$$\begin{aligned}
 \text{ElimCon } (\Gamma, A) & = \text{ElimCon } \Gamma, {}^M \text{ElimTy } A \\
 \text{ElimTms } (\sigma \circ \nu) & = \text{ElimTms } \sigma \circ^M \text{ElimTms } \nu
 \end{aligned}$$

There are no β rules for the equality constructors (such rules would be only interesting in a setting without \mathbf{K}).

4. INJECTIVITY OF CONTEXT AND TYPE FORMERS

As examples of using the eliminator we prove injectivity of context and type constructors. We will need these results when proving decidability of equality for normal forms in section 7.

We start by injectivity of context extension. As there are no equality constructors for contexts, the proof follows the usual argument for injectivity of constructors for inductive types.

First, given $\Gamma_0 : \mathbf{Con}$ and $A_0 : \mathbf{Ty} \Gamma_0$ we define a family over contexts $P : \mathbf{Con} \rightarrow \mathbf{Set}$ using the eliminator. We specify the motives and methods as follows.

$$\begin{aligned}
\mathbf{Con}^M _ &:= \mathbf{Set} \\
\mathbf{Ty}^M _ _ &:= \top \\
\mathbf{Tms}^M _ _ _ &:= \top \\
\mathbf{Tm}^M _ _ _ &:= \top \\
\cdot^M &:= \perp \\
_,^M _ \{ \Gamma : \mathbf{Con} \} _ \{ A : \mathbf{Ty} \Gamma \} _ &:= \Sigma(q : \Gamma_0 \equiv \Gamma). A_0 \equiv^q A \\
\mathbf{id}^M, \dots &:= \mathbf{tt} \\
\mathbf{oid}^M, \dots &:= \mathbf{refl}
\end{aligned}$$

A context is interpreted as a type. Types, substitutions and terms are interpreted as elements of the unit type, hence the interpretations of all the type formers, substitution and term constructors are trivially \mathbf{tt} and all the equalities hold by reflexivity. The empty context is interpreted as the empty type (we will never need this later) and an extended context (Γ, A) is interpreted as a pair of equalities between Γ_0 and Γ and A_0 and A (the latter depends on the former equality). When defining $_,^M _$ we wrote underscores for the interpretations of Γ and A (having types \mathbf{Set} and \top , respectively) thus ignoring these arguments, we only used Γ and A themselves which are implicit arguments of the eliminator. Using the above motives and methods, we define $P := \mathbf{ElimCon} : \mathbf{Con} \rightarrow \mathbf{Set}$ and the β rule tells us that

$$P(\Gamma_0, A_0) = \Sigma(q : \Gamma_0 \equiv \Gamma_0). A_0 \equiv^q A_0$$

and

$$P(\Gamma_1, A_1) = \Sigma(q : \Gamma_0 \equiv \Gamma_1). A_0 \equiv^q A_1.$$

We can prove the first one by $(\mathbf{refl}, \mathbf{refl})$ and given an equality w between the indices (Γ_0, A_0) and (Γ_1, A_1) , we can transport it to the second one. This proves injectivity:

$$\mathbf{inj}, : (w : (\Gamma_0, A_0) \equiv (\Gamma_1, A_1)) := w_*(\mathbf{refl}, \mathbf{refl}) : \Sigma(q : \Gamma_0 \equiv \Gamma_1). A_0 \equiv^q A_1$$

To show injectivity of type formers, we start by the definition of normal types. These are either \mathbf{U} , \mathbf{El} or $\mathbf{\Pi}$, but not substituted types. Then we show normalisation of types using the eliminator (this just means pushing down the substitutions until we reach a \mathbf{U} or \mathbf{El}). Finally we prove the first injectivity lemma for $\mathbf{\Pi}$ using normalisation.

Normal types are given by the following indexed inductive type which is defined mutually with the embedding back into types. In spite of their name, these are not fully normal types: they can include arbitrary non-normal terms through \mathbf{El} . Note that we use overloaded

constructor names.

```

data NTy : (Γ : Con) → Set
  ⌈_⌋⌋      : NTy Γ → Ty Γ
data NTy
  U        : NTy Γ
  El       : Tm Γ U → NTy Γ
  Π       : (A : NTy Γ) → NTy (Γ, ⌈A⌋⌋) → NTy Γ
  ⌈Π A B⌋⌋ := Π ⌈A⌋⌋ ⌈B⌋⌋
  ⌈U⌋⌋      := U
  ⌈El Â⌋⌋   := El Â
    
```

Substitution of normal types can be defined by ignoring the substitution for U , applying it to the term for El and substituting recursively for Π . We need to mutually prove a lemma saying that the embedding is compatible with substitution. As NTy is a simple inductive type (no equality constructors) we use pattern matching notation when defining these functions.

```

  -[-]      : NTy Δ → Tms Γ Δ → NTy Γ
  ⌈⌋⌋      : (A : NTy Δ)(σ : Tms Γ Δ) → ⌈A⌋⌋[σ] ≡ ⌈A[σ]⌋⌋
  (Π A B)[σ] := Π (A[σ]) (B[⌈⌋⌋Aσ]*σ ↑ A])
  U[σ]       := U
  (El Â)[σ]  := El (Â[σ])
  ⌈⌋⌋(Π A B)σ := Π ⌋ • apΠ (⌈⌋⌋Aσ) (⌈⌋⌋B (σ ↑ A))
  ⌈⌋⌋Uσ       := U⌋
  ⌈⌋⌋(El Â)σ  := El⌋
    
```

When defining substitution of Π , we need to use $\lceil \lceil \rceil$ to transport the lifted substitution $\sigma \uparrow A$ to the expected type. The lemma $\lceil \lceil \rceil$ is proved using the substitution laws of the syntax and the induction hypothesis in the case of Π . $\text{ap}\Pi$ denotes the congruence rule for Π , its type is $(p_A : A \equiv A') \rightarrow B \equiv^{p_A} B' \rightarrow \Pi A B \equiv \Pi A' B'$.

By induction on normal types, we prove the following two lemmas as well.

```

[id] : (A : NTy Γ) → A[id] ≡ A
⌋⌋ : (A : NTy Γ).∀σ ν.A[σ][ν] ≡ A[σ ∘ ν]
    
```

Now we can define the model of normal types using the following motives for the eliminator.

```

ConM _      := ⊤
TyM {Γ} _ A := Σ(A' : NTy Γ).A ≡ ⌈A'⌋⌋
TmsM _ _ _ := ⊤
TmM _ _ _ := ⊤
    
```

That is, the eliminator will map a type to a normal type and a proof that the embedding of the normal type is equal to the original type. Contexts, substitutions and terms are mapped

to the trivial type. Hence, the methods for contexts, substitutions and terms will be all trivial and the equality methods for them can be proven by `refl`.

The methods for types are given as follows.

$$\begin{aligned} -[-]^M \{A\} (A', p_A) \{\sigma\} - & := (A'[\sigma], \mathbf{ap}(-[\sigma]) p_A \cdot \ulcorner \urcorner A' \sigma) \\ \mathbf{U}^M & := (\mathbf{U}, \mathbf{refl}) \\ \mathbf{El}^M \{\hat{A}\} - & := (\mathbf{El} \hat{A}, \mathbf{refl}) \\ \mathbf{\Pi}^M \{A\} (A', p_A) \{B\} (B', p_B) & := (\mathbf{\Pi} A' (p_{A*} B'), \mathbf{ap} \mathbf{\Pi} p_A p_B) \end{aligned}$$

$-[-]^M$ receives a type A as an implicit argument, a normal type A' and a proof p_A that they are equal, a substitution σ as an implicit argument and the semantic version of the substitution which does not carry information. We use the above defined $-[-]$ for substituting A' and we need the concatenation of the equalities p_A and $\ulcorner \urcorner$ to provide the equality $\ulcorner A'[\sigma] \urcorner \equiv \ulcorner A[\sigma] \urcorner$. Mapping \mathbf{U} and $\mathbf{El} \hat{A}$ to normal types is trivial, while in the case of $\mathbf{\Pi} A B$ we use the inductive hypotheses A' and B' to construct $\mathbf{\Pi} A' B'$, and in a similar way we use p_A and p_B to construct the equality.

When proving the equality methods $[\mathbf{id}]^M$, $[\ulcorner \urcorner]^M$, $[\mathbf{U}]^M$, $[\mathbf{El}]^M$ and $[\mathbf{\Pi}]^M$, it is enough to show that the first components of the pairs (the normal types) are equal, the p_A proofs will be equal by `K`. The equality methods $[\mathbf{id}]^M$ and $[\ulcorner \urcorner]^M$ are given by the above lemmas `[id]` and `[ulcorner]`. The semantic counterparts of the substitution laws $[\mathbf{U}]$ and $[\mathbf{El}]$ are trivial, while $[\mathbf{\Pi}]^M$ is given by a straightforward induction.

Using the eliminator, we define normalisation of types as follows.

$$\mathbf{norm} (A : \mathbf{Ty} \Gamma) : \mathbf{NTy} \Gamma := \mathbf{proj}_1 (\mathbf{Elim} \mathbf{Ty} A)$$

We can also show completeness and stability of normalisation (see section 1.1 for this nomenclature).

$$\begin{aligned} \mathbf{compl} (A : \mathbf{Ty} \Gamma) : A & \equiv \ulcorner \mathbf{norm} A \urcorner := \mathbf{proj}_2 (\mathbf{Elim} \mathbf{Ty} A) \\ \mathbf{stab} (A' : \mathbf{NTy} \Gamma) : A' & \equiv \mathbf{norm} \ulcorner A' \urcorner \end{aligned}$$

Stability is proven by a straightforward induction on normal types.

Injectivity of $\mathbf{\Pi}^{\mathbf{NTy}}$ (the $\mathbf{\Pi}$ constructor for normal types) is proven the same way as we did for context extension: the family `P` can be simply given by pattern matching as `NTy` doesn't have equality constructors. Given a type A_0 , we define `P` as follows.

$$\begin{aligned} \mathbf{P} : \mathbf{NTy} \Gamma & \rightarrow \mathbf{Set} \\ \mathbf{P} (\mathbf{\Pi}^{\mathbf{NTy}} A B) & := A_0 \equiv A \\ \mathbf{P} X & := \perp \end{aligned}$$

Note that we can't define the same family over `Ty` (using the eliminator of the syntax) because it does not respect the equality $\mathbf{\Pi} \ulcorner \urcorner$. With the help of this `P`, we can prove injectivity by transporting the reflexivity proof of $\mathbf{P} (\mathbf{\Pi}^{\mathbf{NTy}} A_0 B_0) = (A_0 \equiv A_0)$ to that of $\mathbf{P} (\mathbf{\Pi}^{\mathbf{NTy}} A_1 B_1) = (A_0 \equiv A_1)$.

$$\mathbf{inj} \mathbf{\Pi}^{\mathbf{NTy}} (w : \mathbf{\Pi}^{\mathbf{NTy}} A_0 B_0 \equiv \mathbf{\Pi}^{\mathbf{NTy}} A_1 B_1) : A_0 \equiv A_1 := w_* \mathbf{refl}$$

We put together these pieces to prove injectivity of $\mathbf{\Pi}^{\mathbf{Ty}}$ with the diagram in figure 4. We start with a proof $p : \mathbf{\Pi}^{\mathbf{Ty}} A_0 B_0 \equiv \mathbf{\Pi}^{\mathbf{Ty}} A_1 B_1$, then use completeness to get a proof $q : \ulcorner \mathbf{norm} (\mathbf{\Pi}^{\mathbf{Ty}} A_0 B_0) \urcorner \equiv \ulcorner \mathbf{norm} (\mathbf{\Pi}^{\mathbf{Ty}} A_1 B_1) \urcorner$. Applying `norm` to both sides and

using stability we get $r : \text{norm}(\Pi^{\text{Ty}} A_0 B_0) \equiv \text{norm}(\Pi^{\text{Ty}} A_1 B_1)$. The type of r reduces to $\Pi^{\text{NTy}}(\text{norm} A_0)(\text{norm} B_0) \equiv \Pi^{\text{NTy}}(\text{norm} A_1)(\text{norm} B_1)$ and now we can apply the injectivity of normal Π to get that $\text{norm} A_0 \equiv \text{norm} A_1$. As a last step we apply $\ulcorner - \urcorner$ to both sides of this equality and use completeness on A_0 and A_1 to obtain $A_0 \equiv A_1$.

$$\begin{array}{ccc}
 \Pi^{\text{Ty}} A_0 B_0 & \xrightarrow{p} & \Pi^{\text{Ty}} A_1 B_1 \\
 \text{compl}(\Pi A_0 B_0) \Big| & & \Big| \text{compl}(\Pi A_0 B_0) \\
 \ulcorner \text{norm}(\Pi^{\text{Ty}} A_0 B_0) \urcorner & \xrightarrow{q} & \ulcorner \text{norm}(\Pi^{\text{Ty}} A_1 B_1) \urcorner \\
 \\
 \text{norm} \ulcorner \text{norm}(\Pi^{\text{Ty}} A_0 B_0) \urcorner & \xrightarrow{\text{ap norm } q} & \text{norm} \ulcorner \text{norm}(\Pi^{\text{Ty}} A_1 B_1) \urcorner \\
 \text{stab}(\text{norm}(\Pi^{\text{Ty}} A_0 B_0)) \Big| & & \Big| \text{stab}(\text{norm}(\Pi^{\text{Ty}} A_1 B_1)) \\
 \text{norm}(\Pi^{\text{Ty}} A_0 B_0) & & \text{norm}(\Pi^{\text{Ty}} A_1 B_1) \\
 \parallel & & \parallel \\
 \Pi^{\text{NTy}}(\text{norm} A_0)(\text{norm} B_0) & \xrightarrow{r} & \Pi^{\text{NTy}}(\text{norm} A_1)(\text{norm} B_1) \\
 \\
 \text{norm} A_0 & \xrightarrow{\text{inj} \Pi^{\text{NTy}} r} & \text{norm} A_1 \\
 \\
 \ulcorner \text{norm} A_0 \urcorner & \xrightarrow{\text{ap} \ulcorner - \urcorner (\text{inj} \Pi^{\text{NTy}} r)} & \ulcorner \text{norm} A_1 \urcorner \\
 \text{compl} A_0 \Big| & & \Big| \text{compl} A_1 \\
 A_0 & \xrightarrow{\text{inj} \Pi^{\text{Ty}} p} & A_1
 \end{array}$$

Figure 4: Proof of injectivity of Π^{Ty} in the domain. The dashed lines are given by the fillers of the squares. The double lines are definitional equalities.

We only state the other two injectivity lemmas, they can be proved analogously to $\text{inj} \Pi^{\text{Ty}}$.

$$\begin{aligned}
 \text{inj} \Pi^{\text{Ty}'} & : (w : \Pi^{\text{Ty}} A_0 B_0 \equiv \Pi^{\text{Ty}} A_1 B_1) \rightarrow B_0 \equiv \text{inj} \Pi^{\text{Ty}} w B_1 \\
 \text{inj} \text{El} & : \text{El}^{\text{Ty}} \hat{A}_0 \equiv \text{El}^{\text{Ty}} \hat{A}_1 \rightarrow \hat{A}_0 \equiv \hat{A}_1
 \end{aligned}$$

The above proof works for our small type theory but it is not obvious how it would scale to a type theory with large elimination. In that case the injectivity proof would depend on normalisation of terms.

5. THE CATEGORY OF RENAMINGS

In this section we define the category of renamings \mathbf{REN} . Objects in this category are contexts, morphisms are renamings of variables (\mathbf{Vars}).

We define typed De Bruijn variables \mathbf{Var} and renamings \mathbf{Vars} together with their embeddings into substitutions.

$$\begin{aligned}
\mathbf{data\ Var} & : (\Psi : \mathbf{Con}) \rightarrow \mathbf{Ty\ \Psi} \rightarrow \mathbf{Set} \\
\mathbf{vze} & : \mathbf{Var\ (\Psi, A)} (A[\pi_1 \mathbf{id}]) \\
\mathbf{vsu} & : \mathbf{Var\ \Psi\ A} \rightarrow \mathbf{Var\ (\Psi, B)} (A[\pi_1 \mathbf{id}]) \\
\lceil - \rceil & : \mathbf{Vars\ \Omega\ \Psi} \rightarrow \mathbf{Tms\ \Omega\ \Psi} \\
\mathbf{data\ Vars} & : \mathbf{Con} \rightarrow \mathbf{Con} \rightarrow \mathbf{Set} \\
\epsilon & : \mathbf{Vars\ \Psi} \cdot \\
-, - & : (\beta : \mathbf{Vars\ \Omega\ \Psi}) \rightarrow \mathbf{Var\ \Omega\ A}[\lceil \beta \rceil] \rightarrow \mathbf{Vars\ \Omega\ (\Psi, A)} \\
\lceil - \rceil & : \mathbf{Var\ \Psi\ A} \rightarrow \mathbf{Tm\ \Psi\ A} \\
\lceil \mathbf{vze} \rceil & := \pi_2 \mathbf{id} \\
\lceil \mathbf{vsu}\ x \rceil & := \lceil x \rceil[\pi_1 \mathbf{id}] \\
\lceil \epsilon \rceil & := \epsilon \\
\lceil \beta, x \rceil & := \lceil \beta \rceil, \lceil x \rceil
\end{aligned}$$

Variables are typed de Bruijn indices. \mathbf{vze} projects out the last element of the context, \mathbf{vsu} extends the context, and the type $A : \mathbf{Ty\ \Psi}$ needs to be weakened in both cases because we need to interpret it in Ψ extended by another type. Renamings are lists of variables with the appropriate types. Embedding of variables into terms uses the projections and the identity substitution, and embedding renamings is pointwise.

We use the names Ψ, Ω, Ξ for objects of \mathbf{REN} , x, y for variables, β, γ for renamings.

We need identity and composition of renamings for the categorical structure. To define them, we also need weakening and renaming of variables together with laws relating their embeddings to terms. We only list the types as the definitions are straightforward inductions.

$$\begin{aligned}
\mathbf{wkV} & : \mathbf{Vars\ \Omega\ \Psi} \rightarrow \mathbf{Vars\ (\Omega, A)\ \Psi} & \lceil \mathbf{wkV} \rceil & : \lceil \beta \rceil \circ \pi_1 \mathbf{id} \equiv \lceil \mathbf{wkV}\ \beta \rceil \\
\mathbf{id} & : \mathbf{Vars\ \Psi\ \Psi} & \lceil \mathbf{id} \rceil & : \lceil \mathbf{id} \rceil \equiv \mathbf{id} \\
- \circ - & : \mathbf{Vars\ \Xi\ \Psi} \rightarrow \mathbf{Vars\ \Omega\ \Xi} \rightarrow \mathbf{Vars\ \Omega\ \Psi} & \lceil - \circ - \rceil & : \lceil \beta \rceil \circ \lceil \gamma \rceil \equiv \lceil \beta \circ \gamma \rceil \\
-[-] & : \mathbf{Var\ \Psi\ A} \rightarrow (\beta : \mathbf{Vars\ \Omega\ \Psi}) \rightarrow \mathbf{Var\ \Omega\ A}[\lceil \beta \rceil] & \lceil [-] \rceil & : \lceil x \rceil[\lceil \beta \rceil] \equiv \lceil x[\beta] \rceil
\end{aligned}$$

Renamings form a category, we omit the statement and proofs of the categorical laws.

6. THE LOGICAL PREDICATE INTERPRETATION

In this section, we define the proof-relevant presheaf logical predicate interpretation of the type theory given in section 3. It can be seen as a dependent version of the presheaf model of type theory [24]: for example, in the presheaf model contexts are interpreted as presheaves (a contravariant functor into \mathbf{Set} , that is, a set for each object with some more structure). Now these sets depend not only on the object, but also on a substitution into the corresponding context. The categorically inclined reader can view this construction as the categorical glueing [18] over the Yoneda embedding from the term model to the presheaf model.

We start by defining the Yoneda embedding \mathbf{TM} which embeds a context into the sets of substitutions into that context, a type into the sets of terms into that context, and substitutions and terms into maps between the corresponding sets. We denote contravariant presheaves over \mathcal{C} by $\mathbf{PSh}\mathcal{C}$, families of presheaves over a presheaf by \mathbf{FamPSh} and natural transformations and sections by $\dot{\rightarrow}$ and $\dot{\xrightarrow{S}}$, respectively. See appendix A for the definitions of these categorical notions.

$$\begin{array}{llll}
 \Delta : \mathbf{Con} & \mathbf{TM}_\Delta : \mathbf{PSh}\mathbf{REN} & \mathbf{TM}_\Delta \Psi := \mathbf{Tms}\Psi \Delta & \mathbf{TM}_\Delta \beta \rho := \rho \circ \ulcorner \beta \urcorner \\
 A : \mathbf{Ty}\Gamma & \mathbf{TM}_A : \mathbf{FamPSh}\mathbf{TM}_\Gamma & \mathbf{TM}_A \Psi \rho := \mathbf{Tm}\Psi A[\rho] & \mathbf{TM}_A \beta t := t[\ulcorner \beta \urcorner] \\
 \sigma : \mathbf{Tms}\Gamma \Delta & \mathbf{TM}_\sigma : \mathbf{TM}_\Gamma \dot{\rightarrow} \mathbf{TM}_\Delta & \mathbf{TM}_{\sigma\Psi} \rho := \sigma \circ \rho & \\
 t : \mathbf{Tm}\Gamma A & \mathbf{TM}_t : \mathbf{TM}_\Gamma \dot{\xrightarrow{S}} \mathbf{TM}_A & \mathbf{TM}_{t\Psi} \rho := t[\rho] &
 \end{array}$$

\mathbf{TM}_Δ is a presheaf over \mathbf{REN} . The functor laws hold as $\rho \circ \ulcorner \mathbf{id} \urcorner \equiv \rho$ and $(\rho \circ \ulcorner \beta \urcorner) \circ \ulcorner \gamma \urcorner \equiv \rho \circ \ulcorner \beta \circ \gamma \urcorner$. \mathbf{TM}_A is a family of presheaves over \mathbf{TM}_Γ , and similarly we have $t[\ulcorner \mathbf{id} \urcorner] \equiv t$ and $t[\ulcorner \beta \urcorner][\ulcorner \gamma \urcorner] \equiv t[\ulcorner \beta \circ \gamma \urcorner]$. \mathbf{TM}_σ is a natural transformation, naturality is given by associativity: $(\sigma \circ \rho) \circ \ulcorner \beta \urcorner \equiv \sigma \circ (\rho \circ \ulcorner \beta \urcorner)$. \mathbf{TM}_t is a section, its naturality law can be verified as $t[\rho][\ulcorner \beta \urcorner] \equiv t[\rho \circ \ulcorner \beta \urcorner]$. \mathbf{TM} can be seen as a (weak) morphism in the category of models of type theory from the syntax into the presheaf model.

The motives for the presheaf logical predicate interpretation are given as families over the Yoneda embedding \mathbf{TM} . In contrast with section 4, here we use a recursive notation for defining the motives and methods.²

$$\begin{array}{ll}
 \Delta : \mathbf{Con} & \mathbf{P}_\Delta : \mathbf{FamPSh}\mathbf{TM}_\Delta \\
 A : \mathbf{Ty}\Gamma & \mathbf{P}_A : \mathbf{FamPSh}(\Sigma(\Sigma\mathbf{TM}_\Gamma\mathbf{TM}_A)\mathbf{P}_\Gamma[\mathbf{wk}]) \\
 \sigma : \mathbf{Tms}\Gamma \Delta & \mathbf{P}_\sigma : \Sigma\mathbf{TM}_\Gamma\mathbf{P}_\Gamma \dot{\xrightarrow{S}} \mathbf{P}_\Delta[\mathbf{TM}_\sigma][\mathbf{wk}] \\
 t : \mathbf{Tm}\Gamma A & \mathbf{P}_t : \Sigma\mathbf{TM}_\Gamma\mathbf{P}_\Gamma \dot{\xrightarrow{S}} \mathbf{P}_A[\mathbf{TM}_t \uparrow \mathbf{P}_\Gamma]
 \end{array}$$

Note that $-[-]$ above is the composition of a family of presheaves with a natural transformation, \mathbf{wk} is the weakening natural transformation and $-\uparrow-$ is lifting of a section (see appendix A for details).

We unfold these definitions below following appendix A.

A context Δ is mapped to a family of presheaves over \mathbf{TM}_Δ . That is, for every substitution $\rho : \mathbf{TM}_\Delta \Psi$ we have a type $\mathbf{P}_{\Delta\Psi} \rho$ expressing that the logical predicate holds for ρ . Moreover, we have the renaming $\mathbf{P}_\Delta \beta : \mathbf{P}_{\Delta\Psi} \rho \rightarrow \mathbf{P}_{\Delta\Omega}(\mathbf{TM}_\Delta \beta \rho)$ for a $\beta : \mathbf{REN}(\Omega, \Psi)$. Sometimes we omit the parameter Ψ , also for the \mathbf{P} operations on types, substitutions and terms.

\mathbf{P}_A is the logical predicate at a type A . It depends on a substitution (for which the predicate needs to hold as well) and a term. $\mathbf{P}_{A\Psi}(\rho, s, \alpha)$ expresses that the logical predicate

²For reference, the ‘‘arguments for the eliminator’’ notation for the motives looks as follows.

$$\begin{array}{ll}
 \mathbf{Con}^M \Delta & := \mathbf{FamPSh}\mathbf{TM}_\Delta \\
 \mathbf{Ty}^M \Gamma^M A & := \mathbf{FamPSh}(\Sigma(\Sigma\mathbf{TM}_\Gamma\mathbf{TM}_A)\Gamma^M[\mathbf{wk}]) \\
 \mathbf{Tms}^M \Gamma^M \Delta^M \sigma & := \Sigma\mathbf{TM}_\Gamma\Gamma^M \dot{\xrightarrow{S}} \Delta^M[\mathbf{TM}_\sigma][\mathbf{wk}] \\
 \mathbf{Tm}^M \Gamma^M A^M t & := \Sigma\mathbf{TM}_\Gamma\Gamma^M \dot{\xrightarrow{S}} A^M[\mathbf{TM}_t \uparrow \Gamma^M]
 \end{array}$$

holds for term $s : \text{Tm } \Psi A[\rho]$.

$$\frac{A : \text{Ty } \Gamma \quad \Psi : |\text{REN}| \quad \rho : \text{TM}_\Gamma \Psi \quad s : \text{TM}_A \rho \quad \alpha : \text{P}_{\Gamma \Psi} \rho}{\text{P}_{A\Psi}(\rho, s, \alpha) : \text{Set}}$$

It is also stable under renamings. That is, for a $\beta : \text{REN}(\Omega, \Psi)$ we have

$$\text{P}_A \beta : \text{P}_{A\Psi}(\rho, s, \alpha) \rightarrow \text{P}_{A\Omega}(\text{TM}_\Gamma \beta \rho, \text{TM}_A \beta s, \text{P}_\Gamma \beta \alpha).$$

A substitution σ is mapped to P_σ which expresses the fundamental theorem of the logical predicate at σ : for any other substitution ρ for which the predicate holds, we can compose it with σ and the predicate will hold for the composition.

$$\frac{\sigma : \text{Tms } \Gamma \Delta \quad \Psi : |\text{REN}| \quad \rho : \text{TM}_\Gamma \Psi \quad \alpha : \text{P}_{\Gamma \Psi} \rho}{\text{P}_{\sigma\Psi}(\rho, \alpha) : \text{P}_{\Delta\Psi}(\sigma \circ \rho)}$$

The fundamental theorem is also natural.

$$\text{P}_\Delta \beta (\text{P}_{\sigma\Psi}(\rho, \alpha)) \equiv \text{P}_{\sigma\Omega}(\text{TM}_\Gamma \beta \rho, \text{P}_\Gamma \beta \alpha)$$

A term t is mapped to the fundamental theorem at the term: given a substitution ρ for which the predicate holds, it also holds for $t[\rho]$ in a natural way.

$$\frac{t : \text{Tm } \Gamma A \quad \Psi : |\text{REN}^{\text{op}}| \quad \rho : \text{TM}_\Gamma \Psi \quad \alpha : \text{P}_{\Gamma \Psi} \rho}{\text{P}_{t\Psi}(\rho, \alpha) : \text{P}_{A\Psi}(\rho, t[\rho], \alpha)}$$

$$\text{P}_A \beta (\text{P}_{t\Psi}(\rho, \alpha)) \equiv \text{P}_{t\Omega}(\text{TM}_\Gamma \beta \rho, \text{P}_\Gamma \beta \alpha)$$

We define the presheaf $\text{TM}^{\text{U}} : \text{PSh } \text{REN}$ and a family over it $\text{TM}^{\text{El}} : \text{FamPSh } \text{TM}^{\text{U}}$. The actions on objects are $\text{TM}^{\text{U}} \Psi := \text{Tm } \Psi \text{U}$ and $\text{TM}^{\text{El}} \Psi \hat{A} := \text{Tm } \Psi (\text{El } \hat{A})$. The action on a morphism β is just substitution $-[\ulcorner \beta \urcorner]$ for both.

Note that the base category of the logical predicate interpretation is fixed to REN . However we parameterise the interpretation by the predicate at the base type U and base family El . These are denoted by $\bar{\text{U}}$ and $\bar{\text{El}}$ and have the following types.

$$\begin{aligned} \bar{\text{U}} &: \text{FamPSh } \text{TM}^{\text{U}} \\ \bar{\text{El}} &: \text{FamPSh } (\Sigma (\Sigma \text{TM}^{\text{U}} \text{TM}^{\text{El}}) \bar{\text{U}}[\text{wk}]) \end{aligned}$$

Now we list the methods for each constructor in the same order as we have given them in section 3. We omit the proofs of functoriality/naturality only for reasons of space (for these details see [25]).

The logical predicate trivially holds at the empty context, and it holds at an extended context for ρ if it holds at the smaller context at $\pi_1 \rho$ and if it holds at the type which extends the context for $\pi_2 \rho$. The second part obviously depends on the first. The action on morphisms for context extension is pointwise. Here we omitted some usages of $-_*-$ e.g. $\text{P}_\Delta \beta \alpha$ is only well-typed in that position when we transport along the equality $\pi_1 \rho \circ \ulcorner \beta \urcorner \equiv \pi_1 (\rho \circ \ulcorner \beta \urcorner)$. From now on we will omit transports and the usages of $\ulcorner - \urcorner$ in most cases for readability.

$$\begin{aligned} \text{P} . (\rho : \text{TM} . \Psi) &:= \top \\ \text{P} . \beta_- &:= \text{tt} \\ \text{P}_{\Delta, A} (\rho : \text{TM}_{\Delta, A} \Psi) &:= \Sigma(\alpha : \text{P}_\Delta(\pi_1 \rho)) . \text{P}_A(\pi_1 \rho, \pi_2 \rho, \alpha) \\ \text{P}_{\Delta, A} (\beta : \text{REN}(\Omega, \Psi)) (\alpha, a) &:= (\text{P}_\Delta \beta \alpha, \text{P}_A \beta a) \end{aligned}$$

The logical predicate at a substituted type is the logical predicate at the type and we need to use the fundamental theorem at the substitution to lift the witness of the predicate for

the substitution. Renaming a substituted type is the same as renaming in the original type (hence the functor laws hold immediately by the inductive hypothesis). This is well-typed because of naturality of TM_σ and P_σ as shown below.

$$\begin{aligned} \text{P}_{A[\sigma]}(\rho, s, \alpha) &:= \text{P}_A(\text{TM}_\sigma \rho, s, \text{P}_\sigma(\rho, \alpha)) \\ \text{P}_{A[\sigma]} \beta a &:= \text{P}_A \beta a : \underbrace{\text{P}_A(\text{TM}_\Delta \beta(\text{TM}_\sigma \rho), \text{TM}_A \beta s, \text{P}_\Delta \beta(\text{P}_\sigma(\rho, \alpha)))}_{\equiv \text{P}_A(\text{TM}_\sigma(\text{TM}_\Gamma \beta \rho), \text{TM}_A \beta s, \text{P}_\sigma(\text{TM}_\Gamma \beta \rho, \text{P}_\Gamma \beta \alpha))} \end{aligned}$$

The logical predicate at the base type and family says what we have given as parameters. Renaming also comes from these parameters.

$$\begin{aligned} \text{P}_U(\rho, s, \alpha) &:= \bar{U}(\text{U}[\ast]s) & \text{P}_U \beta a &:= \bar{U} \beta a \\ \text{P}_{\text{El}\hat{A}}(\rho, s, \alpha) &:= \bar{\text{El}}(\text{El}[\ast]\text{TM}_{\hat{A}}\rho, s, \text{P}_{\hat{A}}(\rho, \alpha)) & \text{P}_{\text{El}\hat{A}} \beta a &:= \bar{\text{El}} \beta a \end{aligned}$$

The logical predicate holds for a function s when we have that if the predicate holds for an argument u (at A , witnessed by v), so it holds for $s\$u$ at B . In addition, we have a Kripke style generalisation: this should be true for $\text{TM}_{\Pi AB} \beta s$ for any morphism β in a natural way. Renaming a witness of the logical predicate at the function type is postcomposing the Kripke morphism by it.

$$\begin{aligned} &\text{P}_{\Pi AB\Psi}((\rho : \text{TM}_\Gamma \Psi), (s : \text{TM}_{\Pi AB} \rho), (\alpha : \text{P}_\Gamma \rho)) \\ &:= \Sigma(\text{map} : (\beta : \text{REN}(\Omega, \Psi))(u : \text{TM}_A(\text{TM}_\Gamma \beta \rho))(v : \text{P}_{A\Omega}(\text{TM}_\Gamma \beta \rho, u, \text{P}_\Gamma \beta \alpha)) \\ &\quad \rightarrow \text{P}_{B\Omega}((\text{TM}_\Gamma \beta \rho, u), (\text{TM}_{\Pi AB} \beta s)\$u, (\text{P}_\Gamma \beta \alpha, v))) \\ &\quad \forall \beta, u, v, \gamma. \text{P}_B \gamma(\text{map} \beta u v) \equiv \text{map}(\beta \circ \gamma)(\text{TM}_A \gamma u)(\text{P}_A \gamma v) \\ &\text{P}_{\Pi AB} \beta'(\text{map}, \text{nat}) := \lambda \beta. \text{map}(\beta' \circ \beta), \lambda \beta. \text{nat}(\beta' \circ \beta) \end{aligned}$$

Now we list the methods for the substitution constructors, that is, we prove the fundamental theorem for substitutions. We omit the naturality proofs. The object theoretic constructs `map` to their metatheoretic counterparts: identity becomes identity, composition becomes composition, the empty substitution becomes the element of the unit type, substitution extension becomes pairing, first projection becomes first projection.

$$\begin{aligned} \text{P}_{\text{id}}(\rho, \alpha) &:= \alpha \\ \text{P}_{\sigma \circ \nu}(\rho, \alpha) &:= \text{P}_\sigma(\text{TM}_\nu \rho, \text{P}_\nu(\rho, \alpha)) \\ \text{P}_\epsilon(\rho, \alpha) &:= \text{tt} \\ \text{P}_{\sigma, t}(\rho, \alpha) &:= \text{P}_\sigma(\rho, \alpha), \text{P}_t(\rho, \alpha) \\ \text{P}_{\pi_1 \sigma}(\rho, \alpha) &:= \text{proj}_1(\text{P}_\sigma(\rho, \alpha)) \end{aligned}$$

The fundamental theorem for substituted terms and the second projection are again just composition and second projection.

$$\begin{aligned} \text{P}_{t[\sigma]}(\rho, \alpha) &:= \text{P}_t(\text{TM}_\sigma \rho, \text{P}_\sigma(\rho, \alpha)) \\ \text{P}_{\pi_2 \sigma}(\rho, \alpha) &:= \text{proj}_2(\text{P}_\sigma(\rho, \alpha)) \end{aligned}$$

Now we prove the fundamental theorem for `lam` and `app` (i.e. provide the corresponding methods). For `lam`, the `map` function is using the fundamental theorem for t which is in the context extended by the domain type $A : \text{Ty } \Gamma$, so we need to supply an extended substitution and a witness of the predicate. Moreover, we need to rename the substitution ρ and the

witness of the predicate α to account for the Kripke property. The naturality is given by the naturality of the term itself.

$$\begin{aligned} P_{\text{lam } t}(\rho, \alpha) &:= \left(\lambda \beta u v. P_t((\text{TM}_\Gamma \beta \rho, u), (P_\Gamma \beta \alpha, v)) \right. \\ &\quad \left. , \lambda \beta u v \gamma. \text{natS } P_t((\text{TM}_\Gamma \beta \rho, u), (P_\Gamma \beta \alpha, v)) \gamma \right) \end{aligned}$$

Application uses the `map` part of the logical predicate and the identity renaming.

$$P_{\text{app } t}(\rho, \alpha) := \text{map } (P_t(\pi_1 \rho, \text{proj}_1 \alpha)) \text{ id } (\pi_2 \rho) (\text{proj}_2 \alpha)$$

Lastly, we need to provide methods for the equality constructors. We won't list all of these proofs as they are quite straightforward, but as examples we show the semantic versions of the laws $\square\square$ and $\pi_2\beta$. For $\square\square$, we have to show that the two families of presheaves $P_{A[\sigma][\nu]}$ and $P_{A[\sigma\nu]}$ are equal. It is enough to show that their action on objects and morphisms coincides as the equalities will be equal by K . Note that we use function extensionality to show the equality of the presheaves from the pointwise equality of actions. When we unfold the definitions for the actions on objects we see that the results are equal by associativity.

$$\begin{aligned} &P_{A[\sigma][\nu]}(\rho, s, \alpha) \\ &= P_{A[\sigma]}(\text{TM}_\nu \rho, s, P_\nu(\rho, \alpha)) \\ &= P_A(\text{TM}_\sigma(\text{TM}_\nu \rho), s, P_\sigma(\text{TM}_\nu \rho, P_\nu(\rho, \alpha))) \\ &\equiv P_A(\text{TM}_{\sigma\nu} \rho, s, P_\sigma(\text{TM}_\nu \rho, P_\nu(\rho, \alpha))) \\ &= P_A(\text{TM}_{\sigma\nu} \rho, s, P_{\sigma\nu}(\rho, \alpha)) \\ &= P_{A[\sigma\nu]}(\rho, s, \alpha) \end{aligned}$$

The actions on morphisms are equal by unfolding the definitions.

$$P_{A[\sigma][\nu]} \beta a = P_A \beta a = P_{A[\sigma\nu]} \beta a$$

For $\pi_2\beta$ we need to show that two sections $P_{\pi_2(\sigma, t)}$ and P_t are equal, and again, the law parts of the sections will be equal by K .

$$\pi_2\beta^M : P_{\pi_2(\sigma, t)}(\rho, \alpha) = \text{proj}_2(P_{\sigma, t}(\rho, \alpha)) = \text{proj}_2(P_\sigma(\rho, \alpha), P_t(\rho, \alpha)) = P_t(\rho, \alpha)$$

7. NORMAL FORMS

We define η -long β -normal forms mutually with neutral terms. Neutral terms are terms where a variable is in a key position which precludes the application of the rule $\Pi\beta$. Embeddings back into the syntax are defined mutually in the obvious way. Note that neutral terms and normal forms are indexed by types, not normal types.

<pre> data Ne : (Γ : Con) → Ty Γ → Set data Nf : (Γ : Con) → Ty Γ → Set ⌈ − ⌋ : Nf Γ A → Tm Γ A data Ne var : Var Γ A → Ne Γ A app : Ne Γ (Π A B) → (v : Nf Γ A) → Ne Γ (B[⌈ v ⌋]) </pre>	<pre> data Nf neuU : Ne Γ U → Nf Γ U neuEl : Ne Γ (El Â) → Nf Γ (El Â) lam : Nf (Γ, A) B → Nf Γ (Π A B) ⌈ − ⌋ : Ne Γ A → Tm Γ A </pre>
---	---

We define lists of neutral terms and normal forms. X is a parameter of the list, it can stand for both **Ne** and **Nf**.

$$\begin{aligned} \text{data } \text{-s} (X : (\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Set}) : \text{Con} \rightarrow \text{Con} \rightarrow \text{Set} \\ \lceil _ \rceil : X \text{s } \Gamma \Delta \rightarrow \text{Tms } \Gamma \Delta \\ \text{data } X \text{s} \\ \epsilon \quad : X \text{s } \Gamma \cdot \\ \text{-, -} : (\tau : X \text{s } \Gamma \Delta) \rightarrow X \Gamma A [\lceil \tau \rceil] \rightarrow X \text{s } \Gamma (\Delta, A) \end{aligned}$$

We also need renamings of (lists of) normal forms and neutral terms together with lemmas relating their embeddings to terms. Again, X can stand for both **Ne** and **Nf**.

$$\begin{aligned} \text{-}[_] : X \Gamma A \rightarrow (\beta : \text{Vars } \Psi \Gamma) \rightarrow X \Psi A [\lceil \beta \rceil] \quad \lceil [_] \rceil : \lceil n \rceil [\lceil \beta \rceil] \equiv \lceil n[\beta] \rceil \\ \text{-} \circ \text{-} : X \text{s } \Gamma \Delta \rightarrow \text{Vars } \Psi \Gamma \rightarrow X \text{s } \Psi \Delta \quad \lceil \tau \rceil \circ \lceil \beta \rceil \equiv \lceil \tau \circ \beta \rceil \end{aligned}$$

Now we can define the presheaf X_Δ and families of presheaves X_A where X is either **NE** or **NF** (we use uppercase for the families of presheaves and lowercase for the inductive types, just as in the case of **Tm** and **TM**). The definitions follow that of **TM**.

$$\begin{aligned} \Delta : \text{Con} \quad X_\Delta : \text{PSh } \text{REN} \quad X_\Delta \Psi \quad &:= X \text{s } \Psi \Delta \quad X_\Delta \beta \tau := \tau \circ \beta \\ A : \text{Ty } \Gamma \quad X_A : \text{FamPSh } \text{TM}_\Gamma \quad X_A (\rho : \text{TM}_\Gamma \Psi) &:= X \Psi A [\rho] \quad X_A \beta n := n[\beta] \end{aligned}$$

We set the parameters of the logical predicate at the base type and family by defining \bar{U} and $\bar{E}l$. The predicate holds for a term if there is a neutral term of the corresponding type which is equal to the term. The action on morphisms is just renaming.

$$\begin{aligned} \bar{U} : \text{FamPSh } \text{TM}^U \\ \bar{U}_\Psi (\hat{A} : \text{Tm } \Psi U) := \Sigma (n : \text{Ne } \Psi U). \hat{A} \equiv \lceil n \rceil \\ \bar{E}l : \text{FamPSh } (\Sigma (\Sigma \text{TM}^U \text{TM}^{El}) \bar{U}[\text{wk}]) \\ \bar{E}l_\Psi (\hat{A}, t : \text{Tm } \Psi (El \hat{A}), p) := \Sigma (n : \text{Ne } \Psi (El \hat{A})). t \equiv \lceil n \rceil \end{aligned}$$

Now we can interpret any term in the logical predicate interpretation over **REN** with base type interpretations \bar{U} and $\bar{E}l$. We denote the interpretation of t by P_t .

Now we turn to the proof of decidability of equality for normal forms. Decidability of a type X is defined as the sum type $\text{Dec } X := X + (X \rightarrow \perp)$. We start by describing a failed attempt of the proof.

Our first try is to prove decidability directly by mutual induction on variables, neutral terms and normal forms as given below where X can be either **Var**, **Ne** or **Nf**.

$$\text{dec}_X : (n_0 n_1 : X \Gamma A) \rightarrow \text{Dec } (n_0 \equiv n_1)$$

However there is a problem with the constructor **app** for neutral terms. Given $\text{app } n_0 v_0 : \text{Ne } \Gamma (B_0[\lceil v_0 \rceil])$ and $\text{app } n_1 v_1 : \text{Ne } \Gamma (B_1[\lceil v_1 \rceil])$, we would like to first decide the equality for n_0 and n_1 . If they are not equal, we use injectivity of the neutral term constructor **app** to show $\text{app } n_0 v_0 \equiv \text{app } n_1 v_1 \rightarrow \perp$. If they are equal, then we use the other induction hypothesis, decidability of v_0 and v_1 . However, we can't even apply the induction hypothesis for n_0 and n_1 : we would need $\Pi A_0 B_0 \equiv \Pi A_1 B_1$ but we only have $B_0[\lceil v_0 \rceil] \equiv B_1[\lceil v_1 \rceil]$ and this doesn't give us the former equality. It seems that we need to decide whether the types are equal and only compare the terms afterwards (note that A_0 and B_0 are implicit parameters of the constructor **app** before the n_0 and v_0 parameters).

Hence, our next try is to generalise our induction hypothesis and decide equality of types, not only that of terms:

$$\text{dec}_X : (n_0 : X \Gamma A_0)(n_1 : X \Gamma A_1) \rightarrow \text{Dec} (\Sigma(q : A_0 \equiv A_1).n_0 \equiv^q n_1)$$

However this wouldn't work for `lam` because when we decide the equality of `lam v_0 : Nf \Gamma (\Pi A_0 B_0)` and `lam v_1 : Nf \Gamma (\Pi A_1 B_1)`, we would need to decide whether `v_0 : Nf (\Gamma, A_0) B_0` is equal to `v_1 : Nf (\Gamma, A_1) B_1` where the contexts are different. It seems that we need to prove decidability of contexts, types, variables, neutral terms and normal forms at the same time:

$$\text{dec}_X : (n_0 : X \Gamma_0 A_0)(n_1 : X \Gamma_1 A_1) \rightarrow \text{Dec} (\Sigma(p : \Gamma_0 \equiv \Gamma_1, q : A_0 \equiv^p A_1).n_0 \equiv^{p,q} n_1)$$

However types can include non-normal terms (using `El`), so our current definition of normal forms seems to be not suitable for deciding equality: we would need normal contexts and normal types defined mutually with normal forms and neutral terms.

Before abandoning our definition of normal forms for a more complicated one, we look at bidirectional type checking [17]. This teaches us that for neutral terms, the context determines the type (type inference), while for normal forms we need a type as an input (type checking). We observe that we can organise our induction this way: given two variables or neutral terms of an arbitrary type in the same context, we will be able to decide whether they are equal (including their types, which are determined by the context). Given two normal forms of the same type (this is the input type), we can decide whether they are equal. The following mutual induction actually works:

$$\text{dec}_{\text{Var}} : (x_0 : \text{Var } \Gamma A_0)(x_1 : \text{Var } \Gamma A_1) \rightarrow \text{Dec} (\Sigma(q : A_0 \equiv A_1).x_0 \equiv^q x_1)$$

$$\text{dec}_{\text{Ne}} : (n_0 : \text{Ne } \Gamma A_0)(n_1 : \text{Ne } \Gamma A_1) \rightarrow \text{Dec} (\Sigma(q : A_0 \equiv A_1).n_0 \equiv^q n_1)$$

$$\text{dec}_{\text{Nf}} : (v_0 v_1 : \text{Nf } \Gamma A) \rightarrow \text{Dec} (v_0 \equiv v_1)$$

If the variables are both `vze`, they need to have the same type (the last element of the context). If they are both constructed by `vsu`, we can use the induction hypothesis. We check whether the induction hypothesis gives us a positive or negative result. In the positive case, we just return the positive answer again, and in the negative case we need to construct `x_0 \equiv x_1` from `vsu x_0 \equiv vsu x_1` to prove \perp . This comes from injectivity of `vsu`.

If the two neutral terms are variables, we use the induction hypothesis for variables. If they are both applications, we first decide equality of the neutral functions which will also give us an equality of the function types. By injectivity of `\Pi` (section 4) we get that the domains are equal, hence we can compare the normal arguments at this type.

If the two normal forms are neutral terms, we use the induction hypothesis for neutral terms. If they are both λ -abstractions, we can use the induction hypothesis thanks to injectivity of `\Pi` again.

If two variables, neutral terms or normal forms are constructed by different constructors, they are non equal by disjointness of constructors.

In the above proof, we (sometimes implicitly) used injectivity of context extension, `El` and `\Pi` (section 4) and injectivity and disjointness of the constructors for `Var`, `Ne` and `Nf`. For all the technical details, see the formalisation [11].

8. QUOTE AND UNQUOTE

By the logical predicate interpretation using \bar{U} and $\bar{E}l$ we have the following two things:

- terms at the base type and base family are equal to a normal form,

- this property is preserved by the other type formers — this is what the logical predicate says at function types and substituted types.

We make use of this fact to lift the first property to any type. We do this by defining a quote function by induction on the type. Quote takes a term which preserves the predicate and maps it to a normal form which is equal to it. Because of function spaces, we need a function in the other direction as well, mapping neutral terms to the witness of the predicate.

More precisely, we define the quote function \mathfrak{q} and unquote \mathfrak{u} by induction on the structure of contexts and types. For this, we need to define a model of type theory in which only the motives for contexts and types are interesting.

First we define families of presheaves for contexts and types which express that there is an equal normal form. The actions on objects are given as follows.

$$\begin{aligned} \text{NF}^\equiv_\Delta : \text{FamPSh } \text{TM}_\Delta & & \text{NF}^\equiv_A : \text{FamPSh } (\Sigma \text{TM}_\Gamma \text{TM}_A) \\ \text{NF}^\equiv_\Delta (\rho : \text{TM}_\Delta \Psi) := \Sigma(\rho' : \text{NF}_\Delta \Psi). \rho \equiv \ulcorner \rho' \urcorner & & \text{NF}^\equiv_A (\rho, s) := \Sigma(s' : \text{NF}_A \rho). s \equiv \ulcorner s' \urcorner \end{aligned}$$

We use these to write down the motives for contexts and types. We use sections to express the commutativity of the diagram in figure 3. We only write Σ once for iterated usage.

$$\begin{aligned} \mathfrak{u}_\Delta : \text{NE}_\Delta & \xrightarrow{\mathfrak{s}} \text{P}_\Delta[\ulcorner - \urcorner] & \mathfrak{u}_A : \Sigma \text{TM}_\Gamma \text{NE}_A (\text{P}_\Gamma[\text{wk}]) & \xrightarrow{\mathfrak{s}} \text{P}_A[\text{id}, \ulcorner - \urcorner, \text{id}] \\ \mathfrak{q}_\Delta : \Sigma \text{TM}_\Delta \text{P}_\Delta & \xrightarrow{\mathfrak{s}} \text{NF}^\equiv_\Delta[\text{wk}] & \mathfrak{q}_A : \Sigma \text{TM}_\Gamma \text{TM}_A (\text{P}_\Gamma[\text{wk}]) \text{P}_A & \xrightarrow{\mathfrak{s}} \text{NF}^\equiv_A[\text{wk}][\text{wk}] \end{aligned}$$

Unquote for a context takes a neutral substitution and returns a proof that the logical predicate holds for it. Quote takes a substitution for which the predicate holds and returns a normal substitution together with a proof that the original substitution is equal (convertible) to the normal one (embedded into substitutions by $\ulcorner - \urcorner$). The types of unquote and quote for types are more involved as they depend on a substitution for which the predicate needs to hold. Unquote for a type takes such a substitution and a neutral term at the type substituted by this substitution and returns a proof that the predicate holds for this neutral term. The natural transformation $\text{id}, \ulcorner - \urcorner, \text{id}$ is defined in the obvious way, it just embeds the second component (the neutral term) into terms. Quote for a type takes a term of this type for which the predicate holds and returns a normal form at this type together with a proof that it is equal to the term. Here again, another substitution is involved.

The motives for substitutions and terms are the constant unit families.

We will list the methods for contexts and types omitting the naturality proofs for brevity.

Unquote and quote for the empty context are trivial, for extended contexts they are pointwise. ap , is the congruence law of substitution extension $- , -$.

$$\begin{aligned} \mathfrak{u}. (\tau : \text{NE}. \Psi) : \top & := \text{tt} \\ \mathfrak{q}. ((\sigma : \text{TM}. \Psi), (\alpha : \top)) : \Sigma(\rho' : \text{NF}. \Psi). \rho \equiv \ulcorner \rho' \urcorner & := (\epsilon, \epsilon\eta) \\ \mathfrak{u}_{\Delta, A} ((\tau, n) : \text{NE}_{\Delta, A} \Psi) : \Sigma(\alpha : \text{P}_\Delta (\pi_1 \ulcorner \tau, n \urcorner)). \text{P}_A (\pi_1 \ulcorner \tau, n \urcorner, \pi_2 \ulcorner \tau, n \urcorner, \alpha) & \\ & := \mathfrak{u}_\Delta \tau, \mathfrak{u}_A (\ulcorner \tau \urcorner, n, \mathfrak{u}_\Delta \tau) \\ \mathfrak{q}_{\Delta, A} ((\rho : \text{TM}_{\Delta, A} \Psi), (\alpha, a) : \text{P}_{\Delta, A} \rho) : \Sigma(\rho' : \text{NF}_{\Delta, A} \Psi). \rho \equiv \ulcorner \rho' \urcorner & \\ & := \text{let } (\tau, p) := \mathfrak{q}_\Delta (\pi_1 \rho, \alpha); (n, q) := \mathfrak{q}_A (\pi_1 \rho, \pi_2 \rho, \alpha, a) \text{ in } ((\tau, n), (\text{ap}, p q)) \end{aligned}$$

(Un)quoting a substituted type is the same as (un)quoting at the type and using the fundamental theorem at the substitution to lift the witness of the predicate α . As expected, unquoting at the base type simply returns the neutral term itself and the witness of the

predicate will be reflexivity, while quote just returns the witness of the predicate.

$$\begin{aligned}
\mathbf{u}_{A[\sigma]}(\rho, n, \alpha) & : \mathbf{P}_A(\sigma \circ \rho, \ulcorner n \urcorner, \mathbf{P}_\sigma(\rho, \alpha)) := \mathbf{u}_A(\sigma \circ \rho, n, \mathbf{P}_\sigma(\rho, \alpha)) \\
\mathbf{q}_{A[\sigma]}(\rho, s, \alpha, a) & : \Sigma(s' : \mathbf{NF}_{A[\sigma]} \rho). s \equiv \ulcorner s' \urcorner := \mathbf{q}_A(\sigma \circ \rho, s, \mathbf{P}_\sigma(\rho, \alpha), a) \\
\mathbf{u}_U((\rho : \mathbf{TM}_\Gamma \Psi), (n : \mathbf{Ne} \Psi \mathbf{U}[\rho]), \alpha) & : \Sigma(n' : \mathbf{NF}_U \text{id}). \ulcorner n \urcorner \equiv \ulcorner n' \urcorner := \mathbf{neuU}(\mathbf{U}[\ulcorner * \urcorner] n), \text{refl} \\
\mathbf{q}_U(\rho, t, \alpha, (a : \mathbf{NF}^{\equiv}_U(\text{id}, t))) & : \mathbf{NF}^{\equiv}_U(\rho, t) := \mathbf{U}[\ulcorner * \urcorner] a \\
\mathbf{u}_{\text{El}\hat{A}}((\rho : \mathbf{TM}_\Gamma \Psi), (n : \mathbf{Ne} \Psi (\text{El}\hat{A}[\rho])), \alpha) & : \Sigma(n' : \mathbf{NF}_{\text{El}\hat{A}} \text{id}). \ulcorner n \urcorner \equiv \ulcorner n' \urcorner := \mathbf{neuEl}(\mathbf{El}[\ulcorner * \urcorner] n), \text{refl} \\
\mathbf{q}_{\text{El}\hat{A}}(\rho, t, \alpha, (a : \mathbf{NF}^{\equiv}_{\text{El}\hat{A}}(\text{id}, t))) & : \mathbf{NF}^{\equiv}_{\text{El}\hat{A}}(\rho, t) := \mathbf{El}[\ulcorner * \urcorner] a
\end{aligned}$$

We only show the mapping part of unquoting a function. To show that n preserves the predicate, we show that it preserves the predicate for every argument u for which the predicate holds (by v). We quote the argument, thereby getting it in normal form (m), and now we can unquote the neutral term ($\mathbf{app} \ n[\beta] \ m$) to get the result. We also need to transport the result along the proof p that $u \equiv \ulcorner m \urcorner$.

$$\begin{aligned}
& \mathbf{map}\left(\mathbf{u}_{\Pi AB}((\rho : \mathbf{TM}_\Gamma \Psi), (n : \mathbf{NE}_{\Pi AB} \rho), \alpha)\right) (\beta : \mathbf{Vars} \ \Omega \ \Psi) (u : \mathbf{TM}_A(\rho \circ \ulcorner \beta \urcorner)) \\
& \quad (v : \mathbf{P}_{A\Omega}(\rho \circ \ulcorner \beta \urcorner, u, \mathbf{P}_\Gamma \beta \alpha)) : \mathbf{P}_{B\Omega}((\rho \circ \ulcorner \beta \urcorner, u), (\ulcorner n \urcorner[\ulcorner \beta \urcorner]) \$ u, (\mathbf{P}_\Gamma \beta \alpha, v)) \\
& := \mathbf{let} \ (m, p) := \mathbf{q}_A(\rho \circ \ulcorner \beta \urcorner, u, \mathbf{P}_\Gamma \beta \alpha, v) \ \mathbf{in} \ \mathbf{u}_B((\rho \circ \ulcorner \beta \urcorner, u), (p * \mathbf{app} \ n[\beta] \ m), (\mathbf{P}_\Gamma \beta \alpha, v))
\end{aligned}$$

The normal form of a function t is $\mathbf{lam} \ n$ for some normal form n which is in the extended context. We get this n by quoting $\mathbf{app} \ t$ in the extended context. f is the witness that t preserves the relation for any renaming, and we use the renaming $\mathbf{wkV} \ \text{id}$ to use f in the extended context. The argument of f in this case will be the zero de Bruijn index \mathbf{vze} and we need to unquote it to get the witness that it preserves the logical predicate. This is the place where the Kripke property of the logical relation is needed: the base category of the Kripke logical relation needs to minimally include the morphism $\mathbf{wkV} \ \text{id}$ (in our case it has type $\mathbf{Vars}(\Gamma, A) \ \Gamma$).

$$\begin{aligned}
\mathbf{q}_{\Pi AB}(\rho, t, \alpha, f) & : \Sigma(t' : \mathbf{NF}_{\Pi AB} \rho). t \equiv \ulcorner t' \urcorner \\
& := \mathbf{let} \ a := \mathbf{u}_A(\rho \circ \ulcorner \mathbf{wkV} \ \text{id} \urcorner, \mathbf{var} \ \mathbf{vze}, \mathbf{P}_\Gamma(\mathbf{wkV} \ \text{id}) \ \alpha) \\
& \quad (n, p) := \mathbf{q}_B(\rho^A, \mathbf{app} \ t, (\mathbf{P}_\Gamma(\mathbf{wkV} \ \text{id}) \ \alpha, a), \mathbf{map} \ f(\mathbf{wkV} \ \text{id}) \ulcorner \mathbf{vze} \urcorner a) \\
& \quad \mathbf{in} \ (\mathbf{lam} \ n, \Pi \eta^{-1} \cdot \mathbf{ap} \ \mathbf{lam} \ p)
\end{aligned}$$

We have to verify the equality laws for types. Note that we use function extensionality to show that the corresponding quote and unquote functions are equal. The naturality proofs will be equal by \mathbf{K} .

(Un)quote preserves [id] by the left identity law.

$$\begin{aligned}
\mathbf{u}_{A[\text{id}]}(\rho, n, \alpha) & = \mathbf{u}_A(\text{id} \circ \rho, n, \alpha) \equiv \mathbf{u}_A(\rho, n, \alpha) \\
\mathbf{q}_{A[\text{id}]}(\rho, s, \alpha, a) & = \mathbf{q}_A(\text{id} \circ \rho, s, \alpha, a) \equiv \mathbf{q}_A(\rho, s, \alpha, a)
\end{aligned}$$

(Un)quote preserves $\llbracket \cdot \rrbracket$ by associativity for substitutions.

$$\begin{aligned}
 & \mathbf{u}_{A[\sigma][\nu]}(\rho, n, \alpha) \\
 &= \mathbf{u}_A(\sigma \circ (\nu \circ \rho), n, \mathbf{P}_\sigma(\nu \circ \rho, \mathbf{P}_\nu(\rho, \alpha))) \\
 &\equiv \mathbf{u}_A((\sigma \circ \nu) \circ \rho, n, \mathbf{P}_\sigma(\nu \circ \rho, \mathbf{P}_\nu(\rho, \alpha))) \\
 &= \mathbf{u}_{A[\sigma \circ \nu]}(\rho, n, \alpha) \\
 & \mathbf{q}_{A[\sigma][\nu]}(\rho, s, \alpha, a) \\
 &= \mathbf{q}_A(\sigma \circ (\nu \circ \rho), s, \mathbf{P}_\sigma(\nu \circ \rho, \mathbf{P}_\nu(\rho, \alpha)), a) \\
 &\equiv \mathbf{q}_A((\sigma \circ \nu) \circ \rho, s, \mathbf{P}_\sigma(\nu \circ \rho, \mathbf{P}_\nu(\rho, \alpha)), a) \\
 &= \mathbf{q}_{A[\sigma \circ \nu]}(\rho, s, \alpha, a)
 \end{aligned}$$

The semantic counterparts of $\mathbf{U}[\cdot]$ and $\mathbf{EI}[\cdot]$ are verified as follows.

$$\begin{aligned}
 \mathbf{u}_{\mathbf{U}[\sigma]}(\rho, n, \alpha) &= \mathbf{u}_{\mathbf{U}}(\sigma \circ \rho, n, \mathbf{P}_\sigma(\rho, \alpha)) = (n, \text{refl}) = \mathbf{u}_{\mathbf{U}}(\rho, n, \alpha) \\
 \mathbf{q}_{\mathbf{U}[\sigma]}(\rho, t, \alpha, a) &= \mathbf{q}_{\mathbf{U}}(\sigma \circ \rho, t, \mathbf{P}_\sigma(\rho, \alpha), a) = a = \mathbf{q}_{\mathbf{U}}(\rho, t, \alpha, a) \\
 \mathbf{u}_{\mathbf{EI}[\hat{A}][\sigma]}(\rho, n, \alpha) &= \mathbf{u}_{\mathbf{EI}[\hat{A}]}\left(\sigma \circ \rho, n, \mathbf{P}_\sigma(\rho, \alpha)\right) = (n, \text{refl}) = \mathbf{u}_{\mathbf{EI}[\hat{A}][\sigma]}(\rho, n, \alpha) \\
 \mathbf{q}_{\mathbf{EI}[\hat{A}][\sigma]}(\rho, t, \alpha, a) &= \mathbf{q}_{\mathbf{EI}[\hat{A}]}\left(\sigma \circ \rho, t, \mathbf{P}_\sigma(\rho, \alpha), a\right) = a = \mathbf{q}_{\mathbf{EI}[\hat{A}][\sigma]}(\rho, t, \alpha, a)
 \end{aligned}$$

For reasons of space, we only state what we need to verify for $\mathbf{\Pi}[\cdot]$. It is enough to show that the mapping parts of the unquoted functions are equal and that the first components of the results of quote are equal because the other parts are equalities.

$$\begin{aligned}
 \text{map}(\mathbf{u}_{(\mathbf{\Pi} A B)[\sigma]}(\rho, n, \alpha)) &\equiv \text{map}(\mathbf{u}_{\mathbf{\Pi} A[\sigma] B[\sigma \uparrow A]}(\rho, n, \alpha)) \\
 \text{proj}_1(\mathbf{q}_{(\mathbf{\Pi} A B)[\sigma]}(\rho, t, \alpha, f)) &\equiv \text{proj}_1(\mathbf{q}_{\mathbf{\Pi} A[\sigma] B[\sigma \uparrow A]}(\rho, t, \alpha, f))
 \end{aligned}$$

The methods for substitutions and terms (including the equality methods) are all trivial.

9. REAPING THE FRUITS

Now we can define the normalisation function and show that it is complete as follows. We quote at the type of the input term and as parameters we provide the identity substitution, the term itself, a witness that the logical predicate holds for the identity (neutral) substitution (this is given by unquote) and a witness that the predicate holds for the input term. This is given by the fundamental theorem of the logical relation, which needs identity again.

$$\begin{aligned}
 \text{norm}_A(t : \text{Tm } \Gamma A) : \text{Nf } \Gamma A &:= \text{proj}_1\left(\mathbf{q}_A(\text{id}, t, \mathbf{u}_\Gamma \text{id}, \mathbf{P}_t(\text{id}, \mathbf{u}_\Gamma \text{id}))\right) \\
 \text{compl}_A(t : \text{Tm } \Gamma A) : t \equiv \ulcorner \text{norm}_A t \urcorner &:= \text{proj}_2\left(\mathbf{q}_A(\text{id}, t, \mathbf{u}_\Gamma \text{id}, \mathbf{P}_t(\text{id}, \mathbf{u}_\Gamma \text{id}))\right)
 \end{aligned}$$

Note that we implicitly used the equality $A[\text{id}] \equiv A$ in the above definitions.

We prove stability by mutual induction on neutral terms and normal forms.

$$\begin{aligned}
 \text{stab}_{\text{Ne}} : (n : \text{Ne } \Gamma A) \rightarrow \mathbf{P}_{\ulcorner n \urcorner}(\text{id}, \mathbf{u}_\Gamma \text{id}) &\equiv \mathbf{u}_A(\text{id}, n, \mathbf{u}_\Gamma \text{id}) \\
 \text{stab}_{\text{Nf}} : (n : \text{Nf } \Gamma A) \rightarrow \text{norm}_A \ulcorner n \urcorner &\equiv n
 \end{aligned}$$

Decidability of equality comes from that for normal forms.

$$\text{dec}(t_0 t_1 : \text{Tm } \Gamma A) : \text{Dec}(t_0 \equiv t_1) := \text{compl}_A t_1 * (\text{compl}_A t_0 * \text{dec}_{\text{Nf}}(\text{norm}_A t_0)(\text{norm}_A t_1))$$

Similarly, consistency of normal forms can be proven by the following mutual induction.

$$\text{cons}_{\text{Var}} : \text{Var} \cdot A \rightarrow \perp \qquad \text{cons}_{\text{Ne}} : \text{Ne} \cdot A \rightarrow \perp \qquad \text{cons}_{\text{Nf}} : \text{Nf} \cdot U \rightarrow \perp$$

It follows that our theory is consistent.

$$\text{cons}(t : \text{Tm} \cdot U) : \perp := \text{compl}_U t * \text{cons}_{\text{Nf}}(\text{norm}_A t)$$

10. CONCLUSIONS AND FURTHER WORK

We proved normalisation for a basic type theory with dependent types by the technique of NBE. We evaluate terms into a proof relevant logical predicate model. The model is depending on the syntax, we need to use the dependent eliminator of the syntax. Our approach can be seen as merging the presheaf model and the logical relation used in NBE for simple types [8] into a single logical predicate. This seems to be necessary because of the combination of type indexing and dependent types: the well-typedness of normalisation depends on completeness. Another property to note is that we don't normalise types, we just index normal terms by not necessarily normal types.

QIITs make it possible to define the syntax of type theory in a very concise way, however because of missing computation rules, reasoning with them involves lots of boilerplate. We expect that a cubical metatheory [16] with its systematic way of expressing equalities depending on equalities and its additional computation rules would significantly reduce the amount of boilerplate. The metatheoretic status of QIITs is not yet clear, however we hope that we can justify them using a setoid model [6]. In addition, work on the more general higher inductive types would also validate these constructions.

Another challenge is to extend our basic type theory with inductive types, universes and large elimination. Also, it would be interesting to see how the work fits into the setting of homotopy type theory (without assuming \mathbf{K}). We would also like to investigate whether the logical predicate interpretation can be generalised to work over arbitrary presheaf models and study its relation to categorical glueing.

ACKNOWLEDGEMENTS

We would like to thank Bernhard Reus and the anonymous reviewers for their helpful comments and suggestions.

REFERENCES

- [1] The Agda development team. The Agda Wiki, 2017. Available online.
- [2] Andreas Abel. Towards normalization by evaluation for the $\beta\eta$ -calculus of constructions. In *Functional and Logic Programming*, pages 224–239. Springer, 2010.
- [3] Andreas Abel. *Normalization by Evaluation: Dependent Types and Impredicativity*. PhD thesis, Habilitation, Ludwig-Maximilians-Universität München, 2013.
- [4] Andreas Abel, Thierry Coquand, and Peter Dybjer. Normalization by evaluation for Martin-Löf type theory with typed equality judgements. In *Logic in Computer Science, 2007. LICS 2007. 22nd Annual IEEE Symposium on*, pages 3–12. IEEE, 2007.
- [5] Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science*, 8(1), 2012.
- [6] Thorsten Altenkirch. Extensional equality in intensional type theory. In *14th Symposium on Logic in Computer Science*, pages 412 – 420, 1999.

- [7] Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Phil Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 303–310, 2001.
- [8] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, LNCS 953, pages 182–199, 1995.
- [9] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for a polymorphic system. In *11th Annual IEEE Symposium on Logic in Computer Science*, pages 98–106, 1996.
- [10] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for system *F*. 1997.
- [11] Thorsten Altenkirch and Ambrus Kaposi. Agda formalisation for the paper Normalisation by Evaluation for Type Theory, in *Type Theory*, 2016. Available online at the second author’s website.
- [12] Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for dependent types. In *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, pages 6:1–6:16, 2016.
- [13] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL 2016, pages 18–29, New York, NY, USA, 2016. ACM.
- [14] Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ -calculus. In *Logic in Computer Science, 1991. LICS’91., Proceedings of Sixth Annual IEEE Symposium on*, pages 203–211. IEEE, 1991.
- [15] Jesper Cockx and Andreas Abel. Sprinkles of extensionality for your vanilla type theory. In Silvia Ghilezan and Iveti Jelena, editors, *22nd International Conference on Types for Proofs and Programs, TYPES 2016*. University of Novi Sad, 2016.
- [16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. December 2015.
- [17] Thierry Coquand. An algorithm for type-checking dependent types. *Science of Computer Programming*, 26:167–177, 1996.
- [18] Roy L. Crole. *Categories for types*. Cambridge mathematical textbooks. Cambridge University Press, Cambridge, New York, 1993.
- [19] Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. In *Types for Proofs and Programs*, pages 93–109. Springer, 2006.
- [20] Olivier Danvy. *Type-directed partial evaluation*. Springer, 1999.
- [21] Peter Dybjer. Internal type theory. In *Types for Proofs and Programs*, pages 120–134. Springer, 1996.
- [22] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the λ type theory. *ACM Trans. Comput. Logic*, 6(1):61–101, January 2005.
- [23] Martin Hofmann. Conservativity of equality reflection over intensional type theory. In *TYPES 95*, pages 153–164, 1995.
- [24] Martin Hofmann. Syntax and semantics of dependent types. In *Extensional Constructs in Intensional Type Theory*, pages 13–54. Springer, 1997.
- [25] Ambrus Kaposi. *Type theory in a type theory with quotient inductive types*. PhD thesis, University of Nottingham, 2016.
- [26] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.
- [27] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.
- [28] Nicolas Oury. *Extensionality in the calculus of constructions*, pages 278–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
- [29] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.

APPENDIX A. CATEGORICAL DEFINITIONS

We use the following categorical definitions in sections 6, 7 and 8. Note that we work in a setting with \mathbf{K} (uniqueness of identity proofs).

A category \mathcal{C} is given by a type of objects $|\mathcal{C}|$ and given $I, J : |\mathcal{C}|$, a type $\mathcal{C}(I, J)$ which we call the type of morphisms between I and J . A category is equipped with an operation for composing morphisms $- \circ - : \mathcal{C}(J, K) \rightarrow \mathcal{C}(I, J) \rightarrow \mathcal{C}(I, K)$ and an identity morphism at each object $\text{id}_I : \mathcal{C}(I, I)$. In addition we have the associativity law $(f \circ g) \circ h \equiv f \circ (g \circ h)$ and the identity laws $\text{id} \circ f \equiv f$ and $f \circ \text{id} \equiv f$.

A contravariant presheaf over a category \mathcal{C} is denoted $\Gamma : \text{PSh } \mathcal{C}$. It is given by the following data: given $I : |\mathcal{C}|$, a set ΓI , and given $f : \mathcal{C}(J, I)$ a function $\Gamma f : \Gamma I \rightarrow \Gamma J$. Moreover, we have $\text{idP } \Gamma : \Gamma \text{id} \alpha \equiv \alpha$ and $\text{compP } \Gamma : \Gamma(f \circ g) \alpha \equiv \Gamma g(\Gamma f \alpha)$ for $\alpha : \Gamma I$, $f : \mathcal{C}(J, I)$, $g : \mathcal{C}(K, J)$.

Given $\Gamma : \text{PSh } \mathcal{C}$, a family of presheaves over Γ is denoted $A : \text{FamPSh } \Gamma$. It is given by the following data (indeed, this is equivalent to a presheaf over the category of elements $\int \Gamma$): given $\alpha : \Gamma I$, a set $A_I \alpha$ and given $f : \mathcal{C}(J, I)$, a function $A f : A_I \alpha \rightarrow A_J(\Gamma f \alpha)$. In addition, we have the functor laws $\text{idF } A : A \text{id} v \equiv^{\text{idP}} v$ and $\text{compF } A : A(f \circ g) v \equiv^{\text{compP}} A g(A f v)$ for $\alpha : \Gamma I$, $v : A_I \alpha$, $f : \mathcal{C}(J, I)$, $g : \mathcal{C}(K, J)$.

A natural transformation between presheaves Γ and Δ is denoted $\sigma : \Gamma \rightarrow \Delta$. It is given by a function $\sigma : \{I : |\mathcal{C}|\} \rightarrow \Gamma I \rightarrow \Delta I$ together with the condition $\text{natn } \sigma : \Delta f(\sigma_I \alpha) \equiv \sigma_J(\Gamma f \alpha)$ for $\alpha : \Gamma I$, $f : \mathcal{C}(J, I)$.

A section from a presheaf Γ to a family of presheaves A over Γ is denoted $t : \Gamma \xrightarrow{\text{S}} A$. It is given by a function $t : \{I : |\mathcal{C}|\} \rightarrow (\alpha : \Gamma I) \rightarrow A_I \alpha$ together with the naturality condition $\text{natS } t \alpha f : A f(t \alpha) \equiv t(\Gamma f \alpha)$ for $f : \mathcal{C}(J, I)$. We call this a section as it can be viewed as a section of the first projection from $\Sigma \Gamma A$ to Γ but we define it without using the projection.

Given $\Gamma : \text{PSh } \mathcal{C}$ and $A : \text{FamPSh } \Gamma$ we can define $\Sigma \Gamma A : \text{PSh } \mathcal{C}$ by $(\Sigma \Gamma A) I := \Sigma(\alpha : \Gamma I). A_I \alpha$ and $(\Sigma \Gamma A) f(\alpha, a) := (\Gamma f \alpha, A f a)$.

Given $\sigma : \Gamma \rightarrow \Delta$ and $A : \text{FamPSh } \Delta$, we define $A[\sigma] : \text{FamPSh } \Gamma$ by $A[\sigma]_I \alpha := A_I(\sigma_I \alpha)$ and $A[\sigma] f a := \text{natn } \sigma_*(A f a)$ for $\alpha : \Gamma I$, $a : A[\sigma] \alpha$ and $f : \mathcal{C}(J, I)$.

The weakening natural transformation $\text{wk} : \Sigma \Gamma A \rightarrow \Gamma$ is defined by $\text{wk}_I(\alpha, a) := \alpha$.

Lifting of a section $t : \Gamma \xrightarrow{\text{S}} A$ by a family of presheaves $B : \text{FamPSh } \Gamma$ is a natural transformation $t \uparrow B : \Sigma \Gamma B \rightarrow \Sigma(\Sigma(\Gamma A)) B[\text{wk}]$. It is defined as $(t \uparrow B)_I(\alpha, b) := (\alpha, t_I \alpha, b)$.