Typed Applicative Structures and Normalization by Evaluation for System F^ω

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich

Abstract. We present a normalization-by-evaluation (NbE) algorithm for System F^ω with $\beta\eta$-equality, the simplest impredicative type theory with computation on the type level. Values are kept abstract and requirements on values are kept to a minimum, allowing many different implementations of the algorithm. The algorithm is verified through a general model construction using typed applicative structures, called type and object structures. Both soundness and completeness of NbE are conceived as an instance of a single fundamental theorem.

1 Introduction and Related Work

In theorem provers, such as Coq [INR07], Agda [Nor07], Epigram [CAM07], which are based on intensional type theory, checking the validity of proofs or typings relies on deciding equality of types. Types are recognized as equal if they have the same normal form. This is why normalization plays a key role in type theories, such as the Calculus of Constructions (CC) which underlies Coq, and Martin-Löf Type Theory which is the basis of Agda and Epigram. The hardwired type equality of Coq is restricted to computational equality (β), as opposed to Agda and Epigram which have $\beta\eta$-equality. Our goal is to integrate η-laws into Coq's equality. As a prerequisite, we have to show normalization for the $\beta\eta$-CC.

Normalization by evaluation (NbE) [BS91,Dan99] is a systematic method to perform $\beta\eta$-normalization. In a first step, the object t of type T is evaluated. The resulting value d is then *reified* to an η-long β-normal form v. The reification process is directed by the shape of type T. NbE has proven a valid tool to structure extensional normalization, especially in the notoriously difficult case of sum types [ADHS01,BCF04,Bar08]. In previous work [ACD07], we have adapted NbE to a dependent type theory with one predicative universe and judgmental $\beta\eta$-equality. What is the challenge when stepping up to impredicativity? Predicative type theories allow to define the semantics of types from below via induction-recursion [Dyb00], and the reification function can be defined by induction on types. This fails in the presence of impredicativity, where one first has to lay out a lattice of semantic type candidates and then define impredicative quantification using an intersection over all candidates [GLT89]. Hence, the semantic type structure is not inductive, and reification cannot be defined by induction on types. There are at least two ways out of this dilemma: Altenkirch, Hofmann, and Streicher [AHS96] construct a total normalization function type-wise while building a model for System F. In previous work [Abe08], I have conceived reification as a deterministic

relation between value d and normal form v and their type T, and showed through a model construction that it corresponds to a total function.

In this work, we are moving one step closer to NbE for the CC: we are considering the simplest type system which features impredicativity and computation on the type level: the *higher-order* polymorphic lambda-calculus F^ω. It adds to the problem of impredicativity the difficulty that types are no longer fixed syntactic expressions as in System F, but they need to be normalized as well. Of course, due to the simply-kinded structure of types they could be kept in long normal form using simple structural normalization. This does not scale to the CC, so we resist this temptation.

In our solution, reification of objects is directed by *type values* A. Syntactic types T are interpreted by a pair $(A, \mathcal{A})$ of a type value A and a semantic type $\mathcal{A}$ which is a set of objects that are reifiable at type A. Furthermore, type value A reifies to a normal form V which is $\beta\eta$-equal to T. These considerations lead us to the concept of a *type structure* which captures the similarities between syntactic types, type values, and semantic types. Consequently, syntactic *objects* and their values both form an *object structure* over a type structure, the syntactical type structure in case of syntactic objects and the structure of type values in case of (object) values.

Or notions of type and object structures are very general, in essence typed versions of Barendregt's syntactical applicative structures [Bar84, Def. 5.3.1]. The fundamental theorems we prove are also very general since we do not fix an interpretation of types; we only require that semantic types inhabit a *candidate space*. By choosing different candidate spaces we can harvest different results from the same fundamental theorem, e. g., soundness of NbE, completeness of NbE, or weak normalization of β- or $\beta\eta$-reduction [Abe08].

In the following developments we omit most proofs due to lack of space. They can be found in the full version of this article on the author's homepage [Abe09].

Preliminaries. Contexts $\Xi, \Theta, \Gamma, \Delta, \Phi, \Psi$ are functions from variables to some codomain. We write $\diamond$ for the totally undefined function and $\Phi, x : a$ for the function Φ' with domain $\mathsf{dom}(\Phi) \uplus \{x\}$ such that $\Phi'(x) = a$ and $\Phi'(y) = \Phi(y)$ for $y \neq x$. We say Ψ' *extends* Ψ, written $\Psi' \leq \Psi$, if $\Psi'(x) = \Psi(x)$ for all $x \in \mathsf{dom}(\Psi)$.

Families $\mathcal{T}_\Xi$ indexed by a context Ξ are always understood to be *Kripke*, i. e., $\Xi' \leq \Xi$ implies $\mathcal{T}_\Xi \subseteq \mathcal{T}_{\Xi'}$. The notion *Kripke family* is also used for maps M_Ξ. There it implies that M does not depend on the context parameter, i. e., $M_\Xi(a) = M_{\Xi'}(a)$ for $a \in \mathsf{dom}(M_\Xi)$ and $\Xi' \leq \Xi$. (Note that $\mathsf{dom}(M_\Xi) \subseteq \mathsf{dom}(M_{\Xi'})$ since M is Kripke.)

We write $(a \in \mathcal{A}) \to \mathcal{B}(a)$ for the *dependent function space* $\{f \in \mathcal{A} \to \bigcup_{a \in \mathcal{A}} \mathcal{B}(a) \mid f(a) \in \mathcal{B}(a) \text{ for all } a \in \mathcal{A}\}$.

2 Syntax

In this section, we present the syntax and inference rules for System F^ω. The system consists of three levels: On the lowest level there live the *objects*, meaning polymorphic, purely functional programs. On the middle level live the *types* of objects, and the *type constructors*, which are classified by *kinds* that themselves inhabit the highest level.

Kinding $\Xi \vdash T : \kappa$. "In context Ξ, type T has kind κ."

$$\frac{}{\Xi \vdash C : \Sigma(C)} \qquad \frac{}{\Xi \vdash X : \Xi(X)}$$

$$\frac{\Xi, X{:}\kappa \vdash T : \kappa'}{\Xi \vdash \lambda X{:}\kappa.\, T : \kappa \to \kappa'} \qquad \frac{\Xi \vdash T : \kappa \to \kappa' \qquad \Xi \vdash U : \kappa}{\Xi \vdash T\,U : \kappa'}$$

Type equality $\Xi \vdash T = T' : \kappa$. "In context Ξ, types T and T' are $\beta\eta$-equal of kind κ." Congruence closure of the following β- and η-axioms.

$$\frac{\Xi, X{:}\kappa \vdash T : \kappa' \qquad \Xi \vdash U : \kappa}{\Xi \vdash (\lambda X{:}\kappa.\, T)\,U = T[U/X] : \kappa'} \qquad \frac{\Xi \vdash T : \kappa \to \kappa'}{\Xi \vdash \lambda X{:}\kappa.\, T\,X = T : \kappa \to \kappa'}\; X \notin \mathsf{dom}(\Xi)$$

Typing $\Xi; \Gamma \vdash t : T$. "In contexts Ξ, Γ, object t has type T."

$$\frac{\Xi \vdash \Gamma}{\Xi; \Gamma \vdash x : \Gamma(x)} \qquad \frac{\Xi; \Gamma, x{:}U \vdash t : T}{\Xi; \Gamma \vdash \lambda x{:}U.\, t : U \to T} \qquad \frac{\Xi; \Gamma \vdash t : U \to T \qquad \Xi; \Gamma \vdash u : U}{\Xi; \Gamma \vdash t\,u : T}$$

$$\frac{\Xi \vdash T : \kappa \to \star \qquad \Xi, X{:}\kappa; \Gamma \vdash t : T\,X}{\Xi; \Gamma \vdash \Lambda X{:}\kappa.\, t : \forall^\kappa T}\; X \notin \mathsf{dom}(\Xi) \qquad \frac{\Xi; \Gamma \vdash t : \forall^\kappa T \qquad \Xi \vdash U : \kappa}{\Xi; \Gamma \vdash t\,U : T\,U}$$

$$\frac{\Xi; \Gamma \vdash t : T \qquad \Xi \vdash T = T' : \star}{\Xi; \Gamma \vdash t : T'}$$

Object equality $\Xi; \Gamma \vdash t = t' : T$. "In contexts Ξ, Γ, objects t and t' are $\beta\eta$-equal of type T." Congruence closure of the following β- and η-axioms.

$$\frac{\Xi; \Gamma, x{:}U \vdash t : T \qquad \Xi; \Gamma \vdash u : U}{\Xi; \Gamma \vdash (\lambda x{:}U.\, t)\,u = t[u/x] : T} \qquad \frac{\Xi; \Gamma \vdash t : U \to T}{\Xi; \Gamma \vdash \lambda x{:}U.\, t\,x = t : U \to T}\; x \notin \mathsf{dom}(\Gamma)$$

$$\frac{\Xi, X{:}\kappa; \Gamma \vdash t : T \qquad \Xi \vdash U : \kappa}{\Xi; \Gamma \vdash (\Lambda X{:}\kappa.\, t)\,U = t[U/X] : T[U/X]} \qquad \frac{\Xi; \Gamma \vdash t : \forall^\kappa T}{\Xi; \Gamma \vdash \Lambda X{:}\kappa.\, t\,X = t : \forall^\kappa T}\; X \notin \mathsf{dom}(\Xi)$$

Fig. 1. F^ω: kinding, type equality, typing, object equality.

Kinds $\kappa \in \mathsf{Ki}$ are given by the grammar $\kappa ::= \star \mid \kappa \to \kappa'$. Kind $\star$ classifies type constructors which are actually types, and kind $\kappa \to \kappa'$ classifies the type constructors which map type constructors of kind κ to type constructors of kind κ'. In the following, we will refer to all type constructors as *types*.

Assume a countably infinite set of *type variables* TyVar whose members are denoted by X, Y, Z. *Kinding contexts* $\Xi, \Theta \in \mathsf{KiCxt}$ are partial maps from the type variables into Ki. The set $\mathsf{TyCst} = \{\to, \forall^\kappa \mid \kappa \in \mathsf{Ki}\}$ contains the *type constants* C. Their kinds are given by the signature $\Sigma \in \mathsf{TyCst} \to \mathsf{Ki}$ defined by $\Sigma(\to) = \star \to \star \to \star$ and $\Sigma(\forall^\kappa) = (\kappa \to \star) \to \star$ for all $\kappa \in \mathsf{Ki}$.

Types are given by the grammar $T, U, V ::= C \mid X \mid \lambda X{:}\kappa.\, T \mid T\,U$, where $X \in \mathsf{TyVar}$, and form a "simply-kinded" lambda calculus. As usual, we write $T \to U$ for $\to T\,U$. *Objects* are given by the grammar $t, u, v ::= x \mid \lambda x{:}T.\, t \mid t\,u \mid \Lambda X{:}\kappa.\, t \mid t\,U$ and form a polymorphic lambda-calculus with type abstraction and type application. Herein, object variables x are drawn from a countably infinite set ObjVar which is

disjoint from TyVar. We write $b[a/x]$ for capture-avoiding substitution of a for variable x in syntactic expression b, and FV for the function returning the set of all free type and object variables of a syntactic expression.

The judgements and inference rules of F^ω are displayed in Figure 1. Herein, the auxiliary judgement $\Xi \vdash \Gamma$, read "*Γ is a well-formed typing context in Ξ*", is defined as $\Xi \vdash \Gamma(x) : \star$ for all $x \in \mathsf{dom}(\Gamma)$.

3 Abstract Normalization by Evaluation

In the following, we present normalization by evaluation (NbE) for System F^ω for an abstract domain D of *values* and type values. This leaves the freedom to implement values in different ways, e. g., β-normal forms, weak head normal forms (as in Pollack's constructive engine [Pol94]), closures (as in Coquand's type checker [Coq96]), tagged functions (Epigram 2 [CAM07]) or virtual machine instructions (compiled reduction in Coq [GL02]). All implementations of values that satisfy the interface given in the following can be used with our NbE algorithm, and in this article we provide a framework to prove all these implementations correct.

In this section, we will understand functions in terms of a programming language, i. e., partial and possibly non-terminating. We unify the syntax of kinds, types, and objects into a grammar of expressions Exp. Let $\mathsf{Var} = \mathsf{TyVar} \cup \mathsf{ObjVar}$.

Expressions	$\mathsf{Exp} \ni M, N ::= \star \mid C \mid X \mid x \mid \lambda x{:}M.\,N \mid \Lambda X{:}M.\,N \mid M\,N$	
Values	$\mathsf{D} \ni d, e, f, A, B, F, G$	(abstract)

Environments Env are finite maps from variables to values. Look-up of variable x in environment ρ is written $\rho(x)$, update of environment ρ with new value v for variable x is written $\rho[x \mapsto v]$, and the empty environment is written $\diamond$. The call $\mathsf{fresh}(\rho)$ returns a variable x which is not in $\mathsf{dom}(\rho)$.

Application and evaluation (see Fig. 2) make values into a *syntactical applicative structure* [Bar84, 5.3.1], provided the equations below are satisfied. Such structures will appear later, in a sorted setting, as type and object structures (defs. 1 and 13). Note that establishing the laws of evaluation can be arbitrarily hard, e. g., if $(\!|_|\!)_$ involves an optimizing compiler.

Values are converted back to expressions through reification. However, this process can only be implemented for *term-like* value domains, in particular, we require an embedding of variables into D, and an analysis neView of values that arise as iterated application of a variable (a so-called *neutral* value) or as iterated application of a constant (a *constructed* value). Some constructed values are types or kinds, they are analyzed by tyView, which can actually be defined from neView.

Values d of type V in context Δ, which assigns type values to variables, are reified by a call to $\searrow^{\Uparrow}(\Delta, d, V)$. It is mutually defined with $\searrow^{\Uparrow}(\Delta, n)$ which returns the normal form M and type V of neutral value n. Later in this article, reification will be presented as two relations $\Delta \vdash d \searrow M \Downarrow\Uparrow V$ such that $\Delta \vdash d \searrow M \Uparrow V$ iff $\searrow^{\Uparrow}(\Delta, d, V) = M$ and $\Delta \vdash d \searrow M \Downarrow V$ iff $\searrow^{\Downarrow}(\Delta, d) = (M, V)$.

NbE is now obtained as reification after evaluation. For closed expressions M of type or kind N we define $\mathsf{nbe}(M, N) = \searrow^{\Uparrow}(\diamond, (\!|M|\!)_\diamond, \mathsf{tyView}(\!|N|\!)_\diamond)$.

Applicative structure D of values.

$$
\begin{array}{lll}
\text{Application} & _ \cdot _ & : \ \mathsf{D} \to \mathsf{D} \to \mathsf{D} \\
\text{Evaluation} & (\!|_|\!)_ & : \ \mathsf{Exp} \to \mathsf{Env} \to \mathsf{D} \\
 & (\!|x|\!)_\rho & = \rho(x) \\
 & (\!|\lambda x\!:\!M.\,N|\!)_\rho \cdot d & = (\!|N|\!)_{\rho[x \mapsto d]} \\
 & (\!|X|\!)_\rho & = \rho(X) \\
 & (\!|\Lambda X\!:\!M.\,N|\!)_\rho \cdot G & = (\!|N|\!)_{\rho[X \mapsto G]} \\
 & (\!|M\ N|\!)_\rho & = (\!|M|\!)_\rho \cdot (\!|N|\!)_\rho
\end{array}
$$

D is term-like.

$$
\begin{array}{llll}
\text{Embedding} & \mathsf{var} & : & \mathsf{Var} \to \mathsf{D} \\
\text{View as neutral} & \mathsf{NeView} \ni n & ::= & C \mid X \mid x \mid e\,d \\
 & \mathsf{neView} & : & \mathsf{D} \to \mathsf{NeView} \\
 & \mathsf{neView}(\!|C|\!)_\rho & = & C \\
 & \mathsf{neView}(\mathsf{var}\,X) & = & X \\
 & \mathsf{neView}(\mathsf{var}\,x) & = & x \\
 & \mathsf{neView}(e \cdot d) & = & e\,d \quad \text{if } \mathsf{neView}\ e \text{ is defined} \\
\text{View as type} & \mathsf{TyView} \ni V & ::= & \star \mid A \to B \mid \forall^\kappa F \\
 & \mathsf{tyView} & : & \mathsf{D} \to \mathsf{TyView} \\
 & \mathsf{tyView}\ (\!|\star|\!)_\rho & = & \star \\
 & \mathsf{tyView}\ (\!|M \to N|\!)_\rho & = & \mathsf{tyView}\ (\!|M|\!)_\rho \to \mathsf{tyView}\ (\!|N|\!)_\rho \\
 & \mathsf{tyView}\ (\!|\forall^\kappa M|\!)_\rho & = & \forall^\kappa\ \mathsf{tyView}\ (\!|M|\!)_\rho
\end{array}
$$

Reification.

$$
\begin{array}{ll}
\searrow^{\Uparrow} & : \ \mathsf{Env} \to \mathsf{D} \to \mathsf{TyView} \to \mathsf{Exp} \\
\searrow^{\Uparrow}(\Delta, f, A \to B) & = \mathsf{let}\ \ x \quad\ = \mathsf{fresh}(\Delta) \\
 & \qquad (U, _) = \searrow^{\Downarrow}(\Delta, \mathsf{neView}\ A) \\
 & \quad \mathsf{in}\ \lambda x\!:\!U.\ \searrow^{\Uparrow}(\Delta[x \mapsto A], f \cdot \mathsf{var}\ x, \mathsf{tyView}\ B) \\
\searrow^{\Uparrow}(\Delta, d, \forall^\kappa F) & = \mathsf{let}\ X = \mathsf{fresh}(\Delta)\ \mathsf{in}\ \Lambda X\!:\!\kappa.\ \searrow^{\Uparrow}(\Delta[X \mapsto \kappa], d \cdot \mathsf{var}\ X, \mathsf{tyView}(F \cdot \mathsf{var}\ X)) \\
\searrow^{\Uparrow}(\Delta, e, \star) & = \mathsf{let}\ (M, _) = \searrow^{\Downarrow}(\Delta, \mathsf{neView}\ e)\ \mathsf{in}\ M
\end{array}
$$

$$
\begin{array}{ll}
\searrow^{\Downarrow} & : \ \mathsf{Env} \to \mathsf{NeView} \to \mathsf{Exp} \times \mathsf{TyView} \\
\searrow^{\Downarrow}(\Delta, C) & = (C, \Sigma(C)) \\
\searrow^{\Downarrow}(\Delta, X) & = (X, \mathsf{tyView}(\Delta(X))) \\
\searrow^{\Downarrow}(\Delta, x) & = (x, \mathsf{tyView}(\Delta(x))) \\
\searrow^{\Downarrow}(\Delta, e\,d) & = \mathsf{let}\ (M, V) = \searrow^{\Downarrow}(\Delta, e)\ \mathsf{in\ case}\ V\ \mathsf{of} \\
 & \qquad A \to B \mapsto (M\ (\searrow^{\Uparrow}(\Delta, d, \mathsf{tyView}\ A)), \mathsf{tyView}\ B) \\
 & \qquad \forall^\kappa F \quad\ \mapsto (M\ (\searrow^{\Uparrow}(\Delta, d, \kappa)), \mathsf{tyView}(F \cdot d))
\end{array}
$$

Normalization by evaluation.

$$
\mathsf{nbe}(M, N) = \searrow^{\Uparrow}(\diamond, (\!|M|\!)_\diamond, \mathsf{tyView}(\!|N|\!)_\diamond)
$$

Fig. 2. Specification of an NbE algorithm.

A concrete instance of NbE is obtained by defining a recursive data type D with the constructors:

$$\begin{array}{ll} \mathsf{Constr} & : \mathsf{TyCst} \to \mathsf{D}^* \to \mathsf{D} \\ \mathsf{Ne} & : \mathsf{Var} \to \mathsf{D}^* \to \mathsf{D} \\ \mathsf{Abs} & : (\mathsf{D} \to \mathsf{D}) \to \mathsf{D} \end{array}$$

Application, evaluation, and variable embedding are given by the following equations.

$$\begin{array}{lll} (\mathsf{Constr}\, C\; Gs) \cdot G & = \mathsf{Constr}\, C\, (Gs, G) & \\ (\mathsf{Ne}\, x\; ds) \quad \cdot d & = \mathsf{Ne}\, x\, (ds, d) & \\ (\mathsf{Abs}\, f) \quad \cdot d & = f\, d & \\ (\!|\lambda x : M.\, N|\!)_\rho & = \mathsf{Abs}\, f & \text{where } f\, d = (\!|N|\!)_{\rho[x \mapsto d]} \\ (\!|\Lambda X : M.\, N|\!)_\rho & = \mathsf{Abs}\, f & \text{where } f\, G = (\!|N|\!)_{\rho[X \mapsto G]} \\ (\!|C|\!)_\rho & = \mathsf{Constr}\, C\, () & \end{array}$$

Variables are embedded via $\mathsf{var}\, x = \mathsf{Ne}\, x\, ()$. This instance of NbE is now easily completed using the equations of the specification, and can be implemented directly in Haskell.

In this article we show that *any instance* of the NbE-specification terminates with the correct result for well-formed expressions of F^ω, i. e., we show the following two properties:

1. Soundness: if $\vdash M : N$, then $\vdash \mathsf{nbe}(M, N) = M : N$.
2. Completeness: if $\vdash M : N$ and $\vdash M' : N$, then $\mathsf{nbe}(M, N) = \mathsf{nbe}(M', N)$ (same expression up to α).

In contrast to the untyped presentation in this section, which saves us from some repetition, we will distinguish the three levels of F^ω consequently in the remainder of the article.

4 Type Structures

In this section, we define type structures as an abstraction over syntactic types, type values, and semantic types. Type structures form a category which has finite products. Let $\mathsf{Ty}^\kappa_\Xi = \{T \mid \Xi \vdash T : \kappa\}$.

Definition 1 (Type structure). *An* (F^ω) type structure *is a tuple* $(\mathcal{T}, \mathsf{Cst}, \mathsf{App}, [\![_]\!]__)$ *where* $\mathcal{T}$ *is a Kripke family* $\mathcal{T}^\kappa_\Xi$ *of sets with the following Kripke families of maps:*

$$\begin{array}{ll} \mathsf{Cst}_\Xi & \in (C \in \mathsf{TyCst}) \to \mathcal{T}^{\Sigma(C)}_\Xi \\ \mathsf{App}^{\kappa \to \kappa'}_\Xi & \in \mathcal{T}^{\kappa \to \kappa'}_\Xi \to \mathcal{T}^\kappa_\Xi \to \mathcal{T}^{\kappa'}_\Xi \end{array}$$

Usually, we will just write $F \cdot G$ *for* $\mathsf{App}^{\kappa \to \kappa'}_\Xi(F, G)$. *Let* $\rho \in \mathcal{T}^\Xi_\Theta$ *iff* $\rho(X) \in \mathcal{T}^{\Xi(X)}_\Theta$ *for all* $X \in \mathsf{dom}(\Xi)$. *The interpretation function has the following properties:*

$$\begin{array}{lll} [\![_]\!]__ & \in \mathsf{Ty}^\kappa_\Xi \to \mathcal{T}^\Xi_\Theta \to \mathcal{T}^\kappa_\Theta & \\ [\![C]\!]_\rho & = \mathsf{Cst}_\Theta(C) & \\ [\![X]\!]_\rho & = \rho(X) & \end{array} \qquad \begin{array}{lll} [\![\lambda X : \kappa.\, T]\!]_\rho \cdot G & = [\![T]\!]_{\rho[X \mapsto G]} & \\ [\![T\, U]\!]_\rho & = [\![T]\!]_\rho \cdot [\![U]\!]_\rho & \\ [\![T[U/X]]\!]_\rho & = [\![T]\!]_{\rho[X \mapsto [\![U]\!]_\rho]} & (*) \end{array}$$

If the condition $(*)$ is fulfilled, we speak of a *combinatory* type structure, since $(*)$ is a characterizing property of combinatory algebras. The condition $(*)$ is only necessary since we chose to use eager substitution in the inference rules of F^ω, it can be dropped when switching to explicit substitutions [ACD08].

We use "interpretation" and "evaluation" synonymously. Note that while the equations determine the interpretation of constants, variables, and application, there is some freedom in the interpretation of functions $[\![\lambda X : \kappa.\, T]\!]_\rho$. It could be lambda-terms (taking $\mathcal{T} = \mathsf{Ty}$), set-theoretical functions (see Def. 20), functional values in an interpreter, machine code etc.

Since Cst_Ξ is independent of Ξ, we have $\mathsf{Cst}_\Xi = \mathsf{Cxt}_\diamond$, we usually suppress the index Ξ in Cst_Ξ. We may even drop Cst altogether, i. e., we just write $\to \in \mathcal{T}_\Xi^{\star\to\star\to\star}$ instead of $\mathsf{Cst}(\to) \in \mathcal{T}_\Xi^{\star\to\star\to\star}$.

To avoid ambiguities when different type structures are in scope, we may write $\to_{\mathcal{T}}$, $\forall_{\mathcal{T}}^\kappa$, $_ \cdot_{\mathcal{T}} _$ and $\mathcal{T}[\![_]\!]_$ to emphasize that we mean the type structure operations of $\mathcal{T}$.

Simple examples of type structures are Ty and Ty modulo β, $\beta\eta$, or judgmental equality. In these instances, the interpretation function is parallel substitution.

Definition 2 (Type structure morphism). *Given two type structures $\mathcal{S}$ and $\mathcal{T}$, a* type structure morphism *$M : \mathcal{S} \to \mathcal{T}$ is a Kripke family of maps $M_\Xi^\kappa \in \mathcal{S}_\Xi^\kappa \to \mathcal{T}_\Xi^\kappa$ that commute with the operations of $\mathcal{S}$, i. e.,*

$$\begin{array}{lcl} M_\Xi^\kappa(C_{\mathcal{S}}) & = & C_{\mathcal{T}} \\ M_\Xi^{\kappa'}(F \cdot_{\mathcal{S}} G) & = & M_\Xi^{\kappa\to\kappa'}(F) \cdot_{\mathcal{T}} M_\Xi^\kappa(G) \\ M_\Theta^\kappa(\mathcal{S}[\![T]\!]_\rho) & = & \mathcal{T}[\![T]\!]_{M_\Theta^\Xi \circ \rho} \qquad \textit{where } (M_\Theta^\Xi \circ \rho)(X) := M_\Theta^{\Xi(X)}(\rho(X)). \end{array}$$

The Cartesian product $\mathcal{S} \times \mathcal{T}$ of two type structures forms a type structure with pointwise application and tupled interpretation. The two projections $\pi_1 : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ and $\pi_2 : \mathcal{S} \times \mathcal{T} \to \mathcal{T}$ are trivially type structure morphisms, and $\times$ is a product in the category of type structures and their morphisms.

4.1 Type Substructures and the Fundamental Theorem for Kinding

Definition 3 (Type substructure). *The Kripke family $\mathcal{S}_\Xi^\kappa \subseteq \mathcal{T}_\Xi^\kappa$ is a* type substructure of $\mathcal{T}$ *if all of $\mathcal{T}$'s operations are well-defined on $\mathcal{S}$, i. e., $C_{\mathcal{T}} \in \mathcal{S}_\Xi^\kappa$, $_ \cdot_{\mathcal{T}} _ \in \mathcal{S}_\Xi^{\kappa\to\kappa'} \to \mathcal{S}_\Xi^\kappa \to \mathcal{S}_\Xi^{\kappa'}$, and $\mathcal{T}[\![_]\!]_ \in \mathsf{Ty}_\Xi^\kappa \to \mathcal{S}_\Theta^\Xi \to \mathcal{S}_\Theta^\kappa$.*

In the following we simply write $\mathcal{S} \subseteq \mathcal{T}$ to mean $\mathcal{S}_\Xi^\kappa \subseteq \mathcal{T}_\Xi^\kappa$ for all κ, Ξ.

Lemma 1 (Projection type substructure). *If $\mathcal{S} \subseteq \mathcal{T}_1 \times \mathcal{T}_2$ is a type substructure, so are $\pi_1(\mathcal{S}) \subseteq \mathcal{T}_1$ and $\pi_2(\mathcal{S}) \subseteq \mathcal{T}_2$.*

Definition 4 (Function space). *We write $K \in \widehat{\mathcal{T}}^\kappa$ if K is a Kripke family of subsets $K_\Xi \subseteq \mathcal{T}_\Xi^\kappa$. Given $K \in \widehat{\mathcal{T}}^\kappa$ and $K' \in \widehat{\mathcal{T}}^{\kappa'}$ we define the* Kripke function space

$$(K \to_{\widehat{\mathcal{T}}} K')_\Xi = \{F \in \mathcal{T}_\Xi^{\kappa\to\kappa'} \mid F \cdot G \in K'_{\Xi'} \textit{ for all } \Xi' \leq \Xi \textit{ and } G \in K_{\Xi'}\}$$

If no ambiguities arise, we write $\to$ for $\to_{\widehat{\mathcal{T}}}$.

Definition 5 (Induced type structure). *Let $\mathcal{T}$ be a type structure and $\mathcal{S} \subseteq \mathcal{T}$ be Kripke. If $C_{\mathcal{T}} \in \mathcal{S}_{\Xi}^{\Sigma(C)}$ for all constants C and $\mathcal{S}_{\Xi}^{\kappa \to \kappa'} = (\mathcal{S}^{\kappa} \to_{\widehat{\mathcal{T}}} \mathcal{S}^{\kappa'})_{\Xi}$ then $\mathcal{S}$ is called* induced *or an* induced type substructure *of $\mathcal{T}$ (see Thm. 1).*

Such an $\mathcal{S}$ is called induced since it is already determined by the choice of the denotation of the base kind $\mathcal{S}^{\star}$.

Theorem 1 (Fundamental theorem of kinding). *Let $\mathcal{T}$ be a type structure. If $\mathcal{S} \subseteq \mathcal{T}$ is induced, then $\mathcal{S}$ is a type substructure of $\mathcal{T}$.*

Proof. We mainly need to show that evaluation is well-defined. This is shown by induction on the kinding derivation, as usual. □

4.2 NbE for Types and Its Soundness

We are ready to define the reification relation for type values and show that NbE, i. e., the composition of evaluation of a syntactic type T and reification to a normal form V, is sound, i. e., T and V are judgmentally equal. As a byproduct, we show totality of NbE on well-kinded types. The structure $\mathcal{T}$ of type values is left abstract. However, not every $\mathcal{T}$ permits reification of its inhabitants. It needs to include the variables which need to be distinguishable from each other and other type values. *Neutral* types, i. e., of the shape $X \cdot \boldsymbol{G}$, need to be analyzable into head X and spine $\boldsymbol{G}$. We call a suitable $\mathcal{T}$ *term-like*; on such a $\mathcal{T}$ we can define contextual reification [ACD08,Abe08].

Definition 6 (Term-like type structure). *A type structure $\mathcal{T}$ is* term-like *if there exists exists a Kripke family of maps $\mathsf{Var}_{\Xi} \in (X \in \mathsf{dom}(\Xi)) \to \mathcal{T}_{\Xi}^{\Xi(X)}$ and a Kripke family of partial maps*

$$\begin{aligned}\mathsf{View}_{\Xi}^{\kappa} \in \mathcal{T}_{\Xi}^{\kappa} \rightharpoonup \{(C, \boldsymbol{G}) \in \mathsf{TyCst} \times \mathcal{T}_{\Xi}^{\boldsymbol{\kappa}} \mid \Sigma(C) = \boldsymbol{\kappa} \to \kappa\} \\ + \ \{(X, \boldsymbol{G}) \in \mathsf{TyVar} \times \mathcal{T}_{\Xi}^{\boldsymbol{\kappa}} \mid \Xi(X) = \boldsymbol{\kappa} \to \kappa\}\end{aligned}$$

such that $\mathsf{View}(F) = (C, \boldsymbol{G})$ iff $F = \mathsf{Cst}(C) \cdot \boldsymbol{G}$ ("F is constructed") and $\mathsf{View}(F) = (X, \boldsymbol{G})$ iff $F = \mathsf{Var}(X) \cdot \boldsymbol{G}$ ("F is neutral").

Usually, we drop the index Ξ to Var. We may write $\mathsf{Var}_{\mathcal{T}}$ to refer to the variable embedding of type structure $\mathcal{T}$.

Definition 7 (Type reification). *On a term-like type structure $\mathcal{T}$ we define* reification relations

$$\begin{array}{ll} \Xi \vdash F \searrow V \Uparrow \kappa & \text{in } \Xi,\ F \text{ reifies to } V \text{ at kind } \kappa, \\ \Xi \vdash H \searrow U \Downarrow \kappa & \text{in } \Xi,\ H \text{ reifies to } U, \text{ inferring kind } \kappa, \end{array}$$

(where $F, H \in \mathcal{T}_{\Xi}^{\kappa}$ with H neutral or constructed, and $V, U \in \mathsf{Ty}_{\Xi}^{\kappa}$) inductively by the following rules:

$$\frac{}{\Xi \vdash X \searrow X \Downarrow \Xi(X)} \qquad \frac{\Xi \vdash H \searrow U \Downarrow \kappa \to \kappa' \qquad \Xi \vdash G \searrow V \Uparrow \kappa}{\Xi \vdash H \cdot G \searrow U\,V \Downarrow \kappa'}$$

$$\frac{}{\Xi \vdash C \searrow C \Downarrow \Sigma(C)} \qquad \frac{\Xi \vdash H \searrow U \Downarrow \star}{\Xi \vdash H \searrow U \Uparrow \star} \qquad \frac{\Xi, X{:}\kappa \vdash F \cdot X \searrow V \Uparrow \kappa'}{\Xi \vdash F \searrow \lambda X{:}\kappa.\, V \Uparrow \kappa \to \kappa'}$$

Reification is *deterministic* in the following sense: For all Ξ, κ, F (inputs) and neutral or constructed H (input) there is at most one V (output) such that $\Xi \vdash F \searrow V \Uparrow \kappa$ and at most one U and κ' (outputs) such that $\Xi \vdash H \searrow U \Downarrow \kappa'$.

Seen as logic programs with inputs and outputs as indicated above, these relations denote partial functions, where $\searrow^{\Uparrow}$ is defined by cases on the kind κ and and $\searrow^{\Downarrow}$ by cases on the neutral value H.

We continue by constructing a model of the kinding rules which proves soundness of NbE for types. Kinds κ are interpreted as sets $\mathsf{G}^{\kappa}_{\Xi}$ of pairs (F, T) *glued together* [CD97] by reification, i. e., the type value F reifies to syntactic type T up to $\beta\eta$-equality. Function kinds are interpreted via Tait's function space (see Def. 4), thus, the fundamental theorem of kinding yields that G is indeed a type structure.

Definition 8 (Glueing candidate). *Fix a term-like type structure $\mathcal{T}$. We define the families $\underline{\mathsf{Gl}}, \overline{\mathsf{Gl}} \subseteq \mathcal{T} \times \mathsf{Ty}$ by*

$$\overline{\mathsf{Gl}}^{\kappa}_{\Xi} = \{(F, T) \in \mathcal{T}^{\kappa}_{\Xi} \times \mathsf{Ty}^{\kappa}_{\Xi} \mid \Xi \vdash F \searrow V \Uparrow \kappa \text{ and } \Xi \vdash T = V : \kappa\},$$
$$\underline{\mathsf{Gl}}^{\kappa}_{\Xi} = \{(H, T) \in \mathcal{T}^{\kappa}_{\Xi} \times \mathsf{Ty}^{\kappa}_{\Xi} \mid \Xi \vdash H \searrow U \Downarrow \kappa \text{ and } \Xi \vdash T = U : \kappa\}.$$

A family $\mathcal{S}$ with $\underline{\mathsf{Gl}}^{\kappa} \subseteq \mathcal{S}^{\kappa} \subseteq \overline{\mathsf{Gl}}^{\kappa}$ is called a glueing candidate.

Def. and Lem. 2 (Kind candidate space) *$\underline{\mathsf{Gl}}^{\kappa}$, $\overline{\mathsf{Gl}}^{\kappa}$ form a* kind candidate space*, i. e., satisfy the following laws, where we write $\underline{\kappa}$ for $\underline{\mathsf{Gl}}^{\kappa}$ and $\overline{\kappa}$ for $\overline{\mathsf{Gl}}^{\kappa}$.*

$$\underline{\star} \sqsubseteq \overline{\star}, \qquad \underline{\kappa} \to \overline{\kappa'} \sqsubseteq \overline{\kappa \to \kappa'}, \qquad \underline{\kappa \to \kappa'} \sqsubseteq \overline{\kappa} \to \underline{\kappa'}.$$

Def. and Lem. 3 (Glueing type structure) *Given a type structure $\mathcal{T}$, we define $\mathsf{G} \subseteq \mathcal{T} \times \mathsf{Ty}$ by $\mathsf{G}^{\star}_{\cdot} = \underline{\star}$ and $\mathsf{G}^{\kappa \to \kappa'}_{\cdot} = \mathsf{G}^{\kappa} \to_{\widehat{\mathcal{T} \times \mathsf{Ty}}} \mathsf{G}^{\kappa'}$. G is a glueing candidate, i. e., $\underline{\mathsf{Gl}}^{\kappa} \subseteq \mathsf{G}^{\kappa} \subseteq \overline{\mathsf{Gl}}^{\kappa}$ for all κ.*

Since G is induced, by the fundamental theorem of kinding it is a type substructure of $\mathcal{T} \times \mathsf{Ty}$. Thus, for all $T \in \mathsf{Ty}^{\kappa}_{\Xi}$, $\mathsf{G}[\![T]\!]_{\mathsf{Var}_{\mathsf{G}}} = (\mathcal{T}[\![T]\!]_{\mathsf{Var}_{\mathcal{T}}}, T) \in \mathsf{G}^{\kappa}_{\Xi} \subseteq \overline{\mathsf{Gl}}^{\kappa}_{\Xi}$, entailing soundness.

Theorem 4 (Soundness of NbE for types). *Let $\mathcal{T}$ be a term-like type structure. If $\Xi \vdash T : \kappa$ then there is a $V \in \mathsf{Ty}^{\kappa}_{\Xi}$ such that $\Xi \vdash \mathcal{T}[\![T]\!]_{\mathsf{Var}_{\mathcal{T}}} \searrow V \Uparrow \kappa$ and $\Xi \vdash T = V : \kappa$.*

5 Type Groupoids

Completeness of NbE means that it models judgmental type equality, i. e., if $\Xi \vdash T = T' : \kappa$ then $\Xi \vdash [\![T]\!] \searrow V \Uparrow \kappa$ and $\Xi \vdash [\![T']\!] \searrow V \Uparrow \kappa$. Completeness will be shown by a fundamental theorem of type equality. Judgmental equality is usually modelled by partial equivalence relations (PERs), which can be seen as groupoids. Hence, we introduce the notion of a *groupoidal type structure*, or *type groupoid*. The advantage over PERs is that we can directly reuse the fundamental theorem of kinding, instantiated to a groupoidal type structure ${}^{2}\mathcal{T}$ of pairs of types, instead of having to prove this theorem again for kinds modelled as PERs.

A *groupoid* is a set $\mathcal{G}$ with inversion $_^{-1} : \mathcal{G} \to \mathcal{G}$ and partial but associative composition $_*_ : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ such that $a^{-1} * a$ and $a * a^{-1}$ are always defined, and if $a * b$ is defined, then $a * b * b^{-1} = a$, and $a^{-1} * a * b = b$. One easily shows that $(a^{-1})^{-1} = a$ and if $a * b$ is defined then $(a * b)^{-1} = b^{-1} * a^{-1}$. Examples of groupoids include partial equivalence relations R, where $(s, t)^{-1} = (t, s)$ and $(r, s) * (s, t) = (r, t)$, and any set S with the trivial groupoidal structure: $s^{-1} = s$ and $r * s$ is defined iff $r = s$, and then $s * s = s$.

A *subgroupoid* is a subset $\mathcal{H} \subseteq \mathcal{G}$ that is closed under inversion and composition.

5.1 Type Groupoids and the Fundamental Theorem of Type Equality

Definition 9 (Type groupoid). *A type structure is* groupoidal *if each $\mathcal{T}^{\kappa}_{\Xi}$ is a groupoid, constants are preserved under inversion, and inversion and composition distribute over application, i. e., $C^{-1} = C$, $(F \cdot G)^{-1} = F^{-1} \cdot G^{-1}$, and $(F \cdot G) * (F' \cdot G') = (F * F') \cdot (G * G')$.*

Given a type structure $\mathcal{T}$ we define the square type groupoid ${}^2\mathcal{T}$ as the product type structure $\mathcal{T} \times \mathcal{T}$ equipped with $(F, G)^{-1} = (G, F)$ and $(F, G) * (G, H) = (F, H)$. If $K \in \widehat{\mathcal{T}}^{\kappa}$ and $K' \in \widehat{\mathcal{T}}^{\kappa'}$ are groupoids, so is $K \to_{\widehat{\mathcal{T}}} K' \in \widehat{\mathcal{T}}^{\kappa \to \kappa'}$

Definition 10 (Induced type groupoid). *Let $\mathcal{T}$ be a type structure and $\mathcal{E} \subseteq {}^2\mathcal{T}$. We say $\mathcal{E}$ is* induced *if $\mathcal{E}$ is an induced type structure and $\mathcal{E}^{\star}_{\Xi}$ is groupoidal for all Ξ.*

Since type equality refers to kinding, we will have to refer to the fundamental theorem of kinding in the proof of the fundamental theorem of type equality.

Lemma 2 (Fundamental theorem of kinding for type groupoids). *Let $\mathcal{T}$ be a type structure and $\mathcal{E} \subseteq {}^2\mathcal{T}$ be induced. Then,*

1. *$\mathcal{E}$ is a type subgroupoid of ${}^2\mathcal{T}$, and*
2. *if $\Xi \vdash T : \kappa$ and $(\rho, \rho') \in \mathcal{E}^{\Xi}_{\Theta}$ then $(\mathcal{T}[\![T]\!]_{\rho}, \mathcal{T}[\![T]\!]_{\rho'}) \in \mathcal{E}^{\kappa}_{\Theta}$.*

Definition 11 (Model/respect type equality). $\mathcal{E} \subseteq {}^2\mathcal{T}$ models type equality *if $\Xi \vdash T = T' : \kappa$ and $(\rho, \rho') \in \mathcal{E}^{\Xi}_{\Theta}$ imply $(\mathcal{T}[\![T]\!]_{\rho}, \mathcal{T}[\![T']\!]_{\rho'}) \in \mathcal{E}^{\kappa}_{\Theta}$. A type structure $\mathcal{T}'$* respects type equality *if $\Xi \vdash T = T' : \kappa$ implies $\mathcal{T}'[\![T]\!]_{\rho} = \mathcal{T}'[\![T']\!]_{\rho}$ for all $\rho \in \mathcal{T}'^{\Xi}_{\Theta}$.*

Theorem 5 (Fundamental theorem of type equality). *Let $\mathcal{T}$ be a combinatory type structure and $\mathcal{E} \subseteq {}^2\mathcal{T}$ an induced type structure. Then $\mathcal{E}$ models type equality.*

5.2 Completeness of NbE for Types

In the following we show that the relation "reify to the same η-long form" gives rise to an equivalence relation on types which models type equality. This implies that NbE is complete.

Def. and Lem. 6 (Kind candidate space for completeness) *Let $\mathcal{T}$ be term-like.*

$$\overline{\mathsf{Per}}^{\kappa}_{\Xi} = \{(F, F') \in {}^2\mathcal{T}^{\kappa}_{\Xi} \mid \Xi \vdash F \searrow V \Uparrow \kappa \text{ and } \Xi \vdash F' \searrow V \Uparrow \kappa \text{ for some } V \in \mathsf{Ty}^{\kappa}_{\Xi}\}$$
$$\underline{\mathsf{Per}}^{\kappa}_{\Xi} = \{(F, F') \in {}^2\mathcal{T}^{\kappa}_{\Xi} \mid \Xi \vdash F \searrow V \Downarrow \kappa \text{ and } \Xi \vdash F' \searrow V \Downarrow \kappa \text{ for some } V \in \mathsf{Ty}^{\kappa}_{\Xi}\}$$

$\underline{\mathsf{Per}}^{\kappa}$ and $\overline{\mathsf{Per}}^{\kappa}$ are Kripke families of subgroupoids, and form a kind candidate space.

Def. and Lem. 7 (Type groupoid for completeness) *Let $\mathcal{T}$ be a type structure. We define $\mathsf{P}^\kappa \subseteq {}^2\mathcal{T}^\kappa$ by recursion on κ: $\mathsf{P}^\star := \underline{\mathsf{Per}}^\star$ and $\mathsf{P}^{\kappa\to\kappa'} = \mathsf{P}^\kappa \to_{{}^2\widehat{\mathcal{T}}} \mathsf{P}^{\kappa'}$. P is an induced type groupoid.*

Theorem 8 (Completeness of NbE for types). *Let $\mathcal{T}$ be a term-like type structure. If $\Xi \vdash T = T' : \kappa$ then $\Xi \vdash [\![T]\!]_{\mathsf{Var}} \searrow V \Uparrow \kappa$ and $\Xi \vdash [\![T']\!]_{\mathsf{Var}} \searrow V \Uparrow \kappa$ for some V.*

Proof. Since $(\mathsf{Var}, \mathsf{Var}) \in \mathsf{P}^\Xi_\Xi$, by the fundamental theorem of type equality we have $([\![T]\!]_{\mathsf{Var}}, [\![T']\!]_{\mathsf{Var}}) \in \mathsf{P}^\kappa_\Xi \subseteq \overline{\mathsf{Per}}^\kappa_\Xi$ which entails the goal. □

6 Object Structures

In this section, we introduce object structures which model both the syntactic object structure Obj indexed by syntactic types in Ty and structures of values D indexed by type values from a structure $\mathcal{T}$. The following development, leading up the fundamental theorem of typing and the soundness of NbE for objects, parallels the preceding one on the type level. However, while we could define the glueing type structure G^κ by induction on kind κ, we cannot define a similar glueing objects structure gl by induction on types, due to impredicativity. Hence, we will define gl as a structure of *candidates* for semantic types.

Definition 12 (Typing context). *Given a type structure $\mathcal{T}$, a $\mathcal{T}_\Xi$-context $\Delta \in \mathcal{T}^{\mathsf{cxt}}_\Xi$ is a partial map from the term variables into $\mathcal{T}^\star_\Xi$. If $\Gamma \in \mathsf{Ty}^{\mathsf{cxt}}_\Xi$ and $\rho \in \mathcal{T}^\Xi_\Theta$ then $[\![\Gamma]\!]_\rho \in \mathcal{T}^{\mathsf{cxt}}_\Theta$ is defined by $[\![\Gamma]\!]_\rho(x) = [\![\Gamma(x)]\!]_\rho$.*

Let $\mathsf{Obj}^{\Xi\vdash T}_\Gamma = \{t \mid \Xi; \Gamma \vdash t : T\}$.

Definition 13 (Object structure). *Let $\mathcal{T}$ be a type structure. An* object structure over $\mathcal{T}$ *is a family $D^{\Xi\vdash A}$ ($A \in \mathcal{T}^\star_\Xi$) of Kripke sets indexed by $\mathcal{T}_\Xi$-contexts Δ such that $\Xi' \leq \Xi$ implies $D^{\Xi\vdash A}_\Delta = D^{\Xi'\vdash A}_\Delta$. It respects type equality, i. e., $\Xi \vdash T = T' : \star$ implies $D^{\Theta\vdash[\![T]\!]_\rho} = D^{\Theta\vdash[\![T']\!]_\rho}$ for any $\rho \in \mathcal{T}^\Xi_\Theta$, and there are operations:*

$$\begin{aligned}
\mathsf{app}^{\Xi\vdash A\to B}_\Delta &\in D^{\Xi\vdash A\to B}_\Delta \to D^{\Xi\vdash A}_\Delta \to D^{\Xi\vdash B}_\Delta,\\
\mathsf{TyApp}^{\Xi\vdash\forall^\kappa F}_\Delta &\in D^{\Xi\vdash\forall^\kappa F}_\Delta \to (G \in \mathcal{T}^\kappa_\Xi) \to D^{\Xi\vdash F\cdot G}_\Delta.
\end{aligned}$$

We write $_\cdot_$ for both of these operations. For $\Delta, \Psi \in \mathcal{T}^{\mathsf{cxt}}_\Theta$, let $\eta \in D^{\Theta\vdash\Psi}_\Delta$ iff $\eta(x) \in D^{\Theta\vdash\Psi(x)}_\Delta$ for all $x \in \mathsf{dom}(\Psi)$. We stipulate a family of evaluation functions

$$(\!|_|\!)^\rho_{_} \in \mathsf{Obj}^{\Xi\vdash T}_\Gamma \to D^{\Theta\vdash[\![\Gamma]\!]_\rho}_\Delta \to D^{\Theta\vdash[\![T]\!]_\rho}_\Delta$$

indexed by $\rho \in \mathcal{T}^\Xi_\Theta$ which satisfy the following equations:

$$\begin{aligned}
(\!|x|\!)^\rho_\eta &= \eta(x)\\
(\!|r\ s|\!)^\rho_\eta &= (\!|r|\!)^\rho_\eta \cdot (\!|s|\!)^\rho_\eta\\
(\!|t\ U|\!)^\rho_\eta &= (\!|t|\!)^\rho_\eta \cdot [\![U]\!]_\rho\\
(\!|t[u/x]|\!)^\sigma_\eta &= (\!|t|\!)^\sigma_{\eta[x\mapsto(\!|u|\!)^\sigma_\eta]}\ (*)
\end{aligned}
\qquad
\begin{aligned}
(\!|\lambda x\!:\!U.\,t|\!)^\rho_\eta \cdot d &= (\!|t|\!)^\rho_{\eta[x\mapsto d]} && \text{if } d \in D^{\Theta\vdash[\![U]\!]_\rho}_\Delta\\
(\!|\Lambda X\!:\!\kappa.\,t|\!)^\rho_\eta \cdot G &= (\!|t|\!)^{\rho[X\mapsto G]}_\eta && \text{if } G \in \mathcal{T}^\kappa_\Theta\\
(\!|t[U/X]|\!)^\sigma_\eta &= (\!|t|\!)^{\sigma[X\mapsto[\![U]\!]_\sigma]}_\eta\ (*)
\end{aligned}$$

Again, $(*)$ have to hold only in *combinatory* object structures.

With parallel substitution, Obj (modulo β, $\beta\eta$, or judgmental equality) forms an object structure over Ty (modulo the same equality).

Definition 14 (Object substructure). *Let $\mathcal{S}, \mathcal{T}$ be type structures with $\mathcal{S} \subseteq \mathcal{T}$ and let D be an object structure over $\mathcal{T}$. Let $E^{\Xi \vdash A} \subseteq D^{\Xi \vdash A}$ be a Kripke family of subsets indexed by $\Delta \in \mathcal{S}^{\mathsf{cxt}}_{\Xi}$ for all $A \in \mathcal{S}^{\star}_{\Xi}$. Then E is an* object substructure of D over $\mathcal{S}$ *if application and evaluation are well defined on E.*

Definition 15 (Reindexed object structure). *Let $M : \mathcal{S} \to \mathcal{T}$ be a type structure morphism and D an object structure over $\mathcal{T}$. The type structure $E^{\Xi \vdash A} := D^{\Xi \vdash M(A)}$ over $\mathcal{S}$ with $f \cdot_E d := f \cdot_D d$, $d \cdot_E G := d \cdot_D (M(G))$, and $E(\!|_|\!)^{\rho}_{_} := D(\!|_|\!)^{M \circ \rho}_{_}$ is called D* reindexed by M.

Given object structures D_1 over $\mathcal{T}_1$ and D_2 over T_1 we define the *product object structure* $D_1 \times D_2$ over $\mathcal{T}_1 \times \mathcal{T}_2$ in the obvious way.

6.1 Realizability Type Structure and the Fundamental Theorem of Typing

Fix some term-like type structure $\mathcal{T}$ and an object structure D over $\mathcal{T}$. Let $\mathcal{A} \in \widehat{D}^{A}_{\Xi}$ if $\mathcal{A}_{\Delta} \subseteq D^{\Xi \vdash A}_{\Delta}$ and $\mathcal{A}$ is Kripke. $\widehat{D}^{A}_{\Xi}$ forms a complete lattice for all Ξ, A.

Definition 16 (Function space and type abstraction on $\widehat{D}$).

$$\begin{array}{ll} _ \to_{\widehat{D}} _ & \in \widehat{D}^{A}_{\Xi} \to \widehat{D}^{B}_{\Xi} \to \widehat{D}^{A \to B}_{\Xi} \\ (\mathcal{A} \to \mathcal{B})_{\Delta} & := \{ f \in D^{\Xi \vdash A \to B}_{\Delta} \mid \textit{for all } d, \Delta' \leq \Delta, d \in \mathcal{A}_{\Delta'} \textit{ implies } f \cdot d \in \mathcal{B}_{\Delta'} \} \end{array}$$

$$\begin{array}{ll} (_._)^{\forall^{\kappa} F} & \in (G \in \mathcal{T}^{\kappa}_{\Xi}) \to \widehat{D}^{F \cdot G}_{\Xi} \to \widehat{D}^{\forall^{\kappa} F}_{\Xi} \\ (G.\mathcal{A})^{\forall^{\kappa} F}_{\Delta} & := \{ d \in D^{\Xi \vdash \forall^{\kappa} F}_{\Delta} \mid d \cdot G \in \mathcal{A}_{\Delta} \} \end{array}$$

Constructors of higher kind are interpreted as operators on Kripke sets.

Definition 17 (Kripke operators of higher kind). *We define $\widehat{D}^{F:\kappa}_{\Xi}$ by $\widehat{D}^{A:\star}_{\Xi} := \widehat{D}^{A}_{\Xi}$ and $\widehat{D}^{F:\kappa \to \kappa'}_{\Xi} := (G \in \mathcal{T}^{\kappa}_{\Xi}) \to \widehat{D}^{G:\kappa}_{\Xi} \to \widehat{D}^{F \cdot G:\kappa'}_{\Xi}$.*

Definition 18 (Type candidate space). *A* type candidate space $\mathcal{C}$ for D over $\mathcal{T}$ *consists of two Kripke sets $\underline{\mathcal{C}}^{\Xi \vdash A}, \overline{\mathcal{C}}^{\Xi \vdash A} \in \widehat{D}^{A}_{\Xi}$, (written $\underline{A}, \overline{A}$ if no ambiguities arise) for each type $A \in \mathcal{T}^{\star}_{\Xi}$ such that the following conditions hold.*

$$\begin{array}{llll} \underline{H} & \subseteq \overline{H} & \in \widehat{D}^{H}_{\Xi} & (H \textit{ neutral}) \\ \underline{\forall^{\kappa} F} & \subseteq G.\underline{F \cdot G} & \in \widehat{D}^{\forall^{\kappa} F}_{\Xi} & (G \in \mathcal{T}^{\kappa}_{\Xi}) \\ X.\overline{F \cdot X} & \subseteq \overline{\forall^{\kappa} F} & \in \widehat{D}^{\forall^{\kappa} F}_{\Xi} & (X \notin \mathsf{dom}(\Xi)) \end{array} \qquad \begin{array}{lll} \underline{A \to B} & \subseteq \overline{A} \to_{\widehat{D}} \underline{B} & \in \widehat{D}^{A \to B}_{\Xi} \\ \underline{A} \to_{\widehat{D}} \overline{B} & \subseteq \overline{A \to B} & \in \widehat{D}^{A \to B}_{\Xi} \end{array}$$

Definition 19 (Realizable semantic types). *If $F \in \mathcal{T}^{\kappa}_{\Xi}$ and $\mathcal{F} \in \widehat{D}^{F:\kappa}_{\Xi}$ then $F \Vdash^{\kappa}_{\mathcal{C}} \mathcal{F}$ (pronounced F* realizes *$\mathcal{F}$) is defined by induction on κ as follows:*

$$\begin{array}{ll} A \Vdash^{\star}_{\mathcal{C}} \mathcal{A} & :\Longleftrightarrow \underline{A} \subseteq \mathcal{A} \subseteq \overline{A} \\ F \Vdash^{\kappa \to \kappa'}_{\mathcal{C}} \mathcal{F} & :\Longleftrightarrow F \cdot G \Vdash^{\kappa'}_{\mathcal{C}} \mathcal{F}(G, \mathcal{G}) \textit{ for all } G \Vdash^{\kappa}_{\mathcal{C}} \mathcal{G} \end{array}$$

We define the Kripke families $\mathcal{T}\widehat{D}^{\kappa}_{\Xi} = \{(F, \mathcal{F}) \in \mathcal{T}^{\kappa}_{\Xi} \times \widehat{D}^{F:\kappa}_{\Xi}\}$ and $\mathcal{C}^{\kappa}_{\Xi} = \{(F, \mathcal{F}) \in \mathcal{T}^{\kappa}_{\Xi} \times \widehat{D}^{F:\kappa}_{\Xi} \mid F \Vdash^{\kappa}_{\mathcal{C}} \mathcal{F}\}$. For the remainder of this section, we fix a type candidate space $\mathcal{C}$.

Definition 20 (Interpretation into $\widehat{D}$). *For $T \in \mathsf{Ty}^{\kappa}_{\Xi}$ and $(\sigma, \rho) \in \mathcal{T}\widehat{D}^{\Xi}_{\Theta}$ we define $\widehat{D}[\![T]\!]_{\sigma,\rho} \in \widehat{D}^{\mathcal{T}[\![T]\!]_{\sigma}:\kappa}_{\Theta}$ as follows:*

$$\begin{array}{lll}
 & \widehat{D}[\![X]\!]_{\sigma,\rho} & := \rho(X) \\
 & \widehat{D}[\![\lambda X\!:\!\kappa.\, T]\!]_{\sigma,\rho} & := ((G, \mathcal{G}) \in \mathcal{T}\widehat{D}^{\kappa}_{\Theta}) \mapsto \widehat{D}[\![T]\!]_{(\sigma,\rho)[X \mapsto (G,\mathcal{G})]} \\
 & \widehat{D}[\![T\,U]\!]_{\sigma,\rho} & := \widehat{D}[\![T]\!]_{\sigma,\rho}(\mathcal{T}[\![U]\!]_{\sigma}, \widehat{D}[\![U]\!]_{\sigma,\rho}) \\
 & \widehat{D}[\![C]\!]_{\sigma,\rho} & := C_{\widehat{D}} \\
\textit{where} & \forall^{\kappa}_{\widehat{D}} & \in \widehat{D}^{\forall^{\kappa}:(\kappa \to \star) \to \star}_{\Xi} \\
 & \forall^{\kappa}_{\widehat{D}}(F, \mathcal{F}) & := \bigcap_{G \Vdash^{\kappa} \mathcal{G}} G.\mathcal{F}(G, \mathcal{G})
\end{array}$$

Since the kind function space is the full set-theoretic one, $\widehat{D}$ is combinatory and respects type equality.

Theorem 9 (Realizability). *$\mathcal{T}\widehat{D}$ is a type structure with application $(F, \mathcal{F}) \cdot (G, \mathcal{G}) = (F \cdot G, \mathcal{F}(G, \mathcal{G}))$ and evaluation $\mathcal{T}\widehat{D}[\![T]\!]_{\sigma,\rho} = (\mathcal{T}[\![T]\!]_{\sigma}, \widehat{D}[\![T]\!]_{\sigma,\rho})$. $\mathcal{C}$ is a type substructure of $\mathcal{T}\widehat{D}$.*

Theorem 10 (Fundamental theorem of typing). *Let D be an object structure over $\mathcal{T}$ and $\underline{\mathcal{C}}, \overline{\mathcal{C}} \in \widehat{D}$ a type candidate space. Let $\mathcal{S} \subseteq \mathcal{C}$ be a type substructure of the associated realizability type structure $\mathcal{C}$. Then the family $E^{\Xi \vdash (A,\mathcal{A})}_{(\Delta,\Phi)} := \mathcal{A}_{\Delta}$ is an object substructure of D reindexed by $\pi_1 : \mathcal{S} \to \mathcal{T}$.*

6.2 Soundness of NbE for Objects

Term-like object structures and neutral objects are now defined analogously to term-like type structures.

Definition 21 (Object reification). *Given a term-like type structure $\mathcal{T}$ and a term-like object structure D over $\mathcal{T}$, we define the relations*

$$\begin{array}{ll}
\Xi; \Delta \vdash d \searrow v \Uparrow A & \textit{d reifies to } v \textit{ at type } A, \\
\Xi; \Delta \vdash e \searrow u \Downarrow A & \textit{e reifies to } u\textit{, inferring type } A,
\end{array}$$

(where $d, e \in D^{\Xi \vdash A}_{\Delta}$ and v, u are syntactical objects) inductively by the following rules:

$$\frac{}{\Xi; \Delta \vdash x \searrow x \Downarrow \Delta(x)} \qquad \frac{\Xi; \Delta \vdash e \searrow u \Downarrow A \to B \qquad \Xi; \Delta \vdash d \searrow v \Uparrow A}{\Xi; \Delta \vdash e\,d \searrow u\,v \Downarrow B}$$

$$\frac{\Xi; \Delta \vdash e \searrow u \Downarrow \forall^{\kappa} F \qquad \Xi \vdash G \searrow V \Uparrow \kappa}{\Xi; \Delta \vdash e\,G \searrow u\,V \Downarrow F \cdot G} \qquad \frac{\Xi, X\!:\!\kappa; \Delta \vdash f \cdot X \searrow v \Uparrow F \cdot X}{\Xi; \Delta \vdash f \searrow \Lambda X\!:\!\kappa.\, v \Uparrow \forall^{\kappa} F}$$

$$\frac{\Xi; \Delta, x\!:\!A \vdash f \cdot x \searrow v \Uparrow B \qquad \Xi \vdash A \searrow U \Uparrow \star}{\Xi; \Delta \vdash f \searrow \lambda x\!:\!U.\, v \Uparrow A \to B} \qquad \frac{\Xi; \Delta \vdash e \searrow u \Downarrow H}{\Xi; \Delta \vdash e \searrow u \Uparrow H}\ H \textit{ neutral}$$

As for types, object reification is deterministic.

Note that we cannot say now in which $\mathsf{Obj}_{\Gamma}^{\Xi \vdash T}$ the objects u and v live. The conjecture is those Γ, T with $\Xi \vdash \Delta \searrow \Gamma$ and $\Xi \vdash A \searrow T \Uparrow \star$. However, this does not follow directly from the definition, it is a consequence of Thm. 12.

Def. and Lem. 11 (Glueing type candidate space) *Let $\mathcal{S} \subseteq \mathcal{T} \times \mathsf{Ty}$ a glueing candidate, $\Vdash_{\mathsf{Gl}} \mathcal{S}$. For $(A, T) \in \mathcal{S}_{\Xi}^{\star}$ we define the Kripke families $\underline{\mathsf{gl}}^{\Xi \vdash (A,T)}, \overline{\mathsf{gl}}^{\Xi \vdash (A,T)} \in \widehat{D \times \mathsf{Obj}_{\Xi}}^{(A,T)}$ by*

$$\begin{aligned}
\overline{\mathsf{gl}}_{(\Delta,\Gamma)}^{\Xi \vdash (A,T)} &:= \{(d,t) \mid \Xi; \Delta \vdash d \searrow v \Uparrow A \textit{ and } \Xi; \Gamma \vdash t = v : T \textit{ for some } v\}, \\
\underline{\mathsf{gl}}_{(\Delta,\Gamma)}^{\Xi \vdash (A,T)} &:= \{(e,t) \mid \Xi; \Delta \vdash e \searrow u \Downarrow A \textit{ and } \Xi; \Gamma \vdash t = u : T \textit{ for some } u\}.
\end{aligned}$$

gl *is a type candidate space.*

Theorem 12 (Soundness of NbE for objects). *Let D be a term-like object structure over a term-like type structure $\mathcal{T}$. If $\Xi; \Gamma \vdash t : T$ then there is a long normal form v such that $\Xi; \mathcal{T}[\![\Gamma]\!]_{\mathsf{Var}} \vdash D(\!|t|\!)_{\mathsf{var}}^{\mathsf{Var}} \searrow v \Uparrow \mathcal{T}[\![T]\!]_{\mathsf{Var}}$ and $\Xi; \Gamma \vdash t = v : T$.*

6.3 Completeness of NbE for objects

Completeness on the object level is shown analogously to completeness on the type level. Define *object groupoids* as groupoidal object structures and show that Kripke function space and impredicative quantification on $\widehat{D}$ preserve the groupoid structure. Then prove a fundamental theorem of object equality and instantiate it to the type candidate space defined analogously to Per. Due to lack of space, we cannot spell out the details and refer to the full version of this article instead [Abe09].

7 Conclusion

We have developed type and object structures, which are sorted applicative structures on type and object level, in order to facilitate generic model constructions for System F^{ω}—which are an alternative to categorical semantics [See87] and Bruce-Meyer-Mitchell models [BM84]. Using special instances of kind candidate spaces we have shown soundness and completeness of an abstract normalization by evaluation algorithm for types. We have gone on to show soundness and completeness of NbE for objects.

We seek to extend NbE to the Calculus of Constructions, using ideas from this work. Due to dependency, type and object levels cannot be defined in sequence, but must be defined simultaneously; this seems to be the main remaining technical difficulty.

References

[Abe08] Andreas Abel. Weak $\beta\eta$-normalization and normalization by evaluation for System F. In *LPAR'08*, volume 5330 of *LNAI*, pages 497–511. Springer, 2008.

[Abe09] Andreas Abel. Typed applicative structures and normalization by evaluation for System F^{ω} (full version). `http://www.tcs.ifi.lmu.de/~abel/fomegaNbe.pdf`, 2009.

[ACD07] Andreas Abel, Thierry Coquand, and Peter Dybjer. Normalization by evaluation for Martin-Löf Type Theory with typed equality judgements. In *LICS'07*, pages 3–12. IEEE CS Press, 2007.

[ACD08] Andreas Abel, Thierry Coquand, and Peter Dybjer. Verifying a semantic $\beta\eta$-conversion test for Martin-Löf type theory. In *MPC'08*, volume 5133 of *LNCS*, pages 29–56. Springer, 2008.

[ADHS01] Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Philip J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *LICS'01*, pages 303–310. IEEE CS Press, 2001.

[AHS96] Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for a polymorphic system. In *LICS'96*, pages 98–106. IEEE CS Press, 1996.

[Bar84] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, Amsterdam, 1984.

[Bar08] Freiric Barral. *Decidability for non-standard conversions in lambda-calculus*. PhD thesis, Ludwig-Maximilians-University Munich, 2008.

[BCF04] Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *POPL'04*, pages 64–76. ACM, 2004.

[BM84] Kim B. Bruce and Albert R. Meyer. The semantics of second order polymorphic lambda calculus. In *Semantics of Data Types*, volume 173 of *LNCS*, pages 131–144. Springer, 1984.

[BS91] Ulrich Berger and Helmut Schwichtenberg. An inverse to the evaluation functional for typed λ-calculus. In *LICS'91*, pages 203–211. IEEE CS Press, 1991.

[CAM07] James Chapman, Thorsten Altenkirch, and Conor McBride. Epigram reloaded: a standalone typechecker for ETT. In *TFP'05*, volume 6 of *Trends in Functional Programming*, pages 79–94. Intellect, 2007.

[CD97] Thierry Coquand and Peter Dybjer. Intuitionistic model constructions and normalization proofs. *MSCS*, 7(1):75–94, 1997.

[Coq96] Thierry Coquand. An algorithm for type-checking dependent types. In *MPC'95*, volume 26 of *SCP*, pages 167–177. Elsevier, 1996.

[Dan99] Olivier Danvy. Type-directed partial evaluation. In *Partial Evaluation Summer School '98*, volume 1706 of *LNCS*, pages 367–411. Springer, 1999.

[Dyb00] Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *JSL*, 65(2):525–549, 2000.

[GL02] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *ICFP'02*, volume 37 of *SIGPLAN Notices*, pages 235–246. ACM, 2002.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in TCS*. CUP, 1989.

[INR07] INRIA. *The Coq Proof Assistant, Version 8.1*. INRIA, 2007. http://coq.inria.fr/.

[Nor07] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers, Göteborg, Sweden, 2007.

[Pol94] Robert Pollack. Closure under alpha-conversion. In *TYPES'93*, volume 806 of *LNCS*, pages 313–332. Springer, 1994.

[See87] R. A. G. Seely. Categorical semantics for higher order polymorphic lambda calculus. *JSL*, 52(4):969–989, 1987.