Towards Normalization by Evaluation for the $\beta\eta$-Calculus of Constructions

Andreas Abel

Project PI.R2, INRIA Rocquencourt and PPS, Paris
andreas.abel@ifi.lmu.de

Abstract. We consider the Calculus of Constructions with typed beta-eta equality and an algorithm which computes long normal forms. The normalization algorithm evaluates terms into a semantic domain, and reifies the values back to terms in normal form. To show termination, we interpret types as partial equivalence relations between values and type constructors as operators on PERs. This models also yields consistency of the beta-eta-Calculus of Constructions. The model construction can be carried out directly in impredicative type theory, enabling a formalization in Coq.

1 Introduction

The proof assistant Coq [INR08] based on intensional type theory is used for large verification projects in mathematics [Gon04] and computer science [Ler06]. However, to this day no complete meta theory of its logical core, the Calculus of Inductive Constructions (CIC) exists. The CIC is a dependent type theory with at least one impredicative base universe (`Set` or `Prop` or both) and an infinite cumulative hierarchy of predicative universes ($\texttt{Type}_i$) above this base. Inductive types with large (aka strong) eliminations exist at every level. The CIC is formulated with untyped equality, leading to complications in model constructions [MW03] and in the treatment of η-equality. As η-reduction, the subject reduction property requires contravariant subtyping, which is especially hard to model (I am only aware of Miquel's coherence space model [Miq00]). And it cannot be formulated as η-expansion in an untyped setting. The lack of η-equality in Coq is an annoyance both for its implementers and its users.

Recently, formulations of CIC with typed equality, aka judgemental equality, have been considered since they admit simple set-theoretical models [Bar09]. Judgemental equality also integrates η-equality nicely. On the downside, injectivity of the function space constructor Π, crucial for the implementation of type checking, is notoriously difficult to establish. Goguen [Gog94] has obtained injectivity of Π in the Extended Calculus of Constructions via his Typed Operational Semantics, a typed Kripke term model with standardizing reduction. In predicative Martin-Löf Type Theory, it is the byproduct of a PER model construction which also yields Normalization by Evaluation (NbE) [ACD07].

In this article, we investigate NbE for the Calculus of Constructions (CoC), a fragment of the CIC with just one impredicative and one predicative universe, with typed $\beta\eta$-equality. By constructing a PER model, we obtain termination and completeness for

NbE, the latter meaning that all judgmentally equal terms normalize to the same expression. As a consequence of the model, we obtain logical consistency of the $\beta\eta$-CoC.

The missing property of soundness of NbE, meaning that each term is judgmentally equal to its computed normal form, is implied by injectivity of Π and vice versa. Decidability of typing also hinges on injectivity. This leaves two options to complete this work and obtain a sound and complete type checker for the CoC with η: Prove soundness of NbE by Kripke logical relations between syntax and semantics as in [ACP09], or obtain injectivity by syntactical means. Adams [Ada06] obtained injectivity for functional pure type systems with judgemental β-equality; his proof might extend to $\beta\eta$.

Overview. This article is organized as follows: In Section 2 we introduce CoC with typed equality and explicit substitutions. In Section 3 we define normalization by evaluation for CoC using partial applicative structures, and we specify a type inference algorithm. In Section 4 we recapitulate a simple method how to classify CoC expressions into terms, types, and kinds, a device which helps us to bootstrap the PER model construction in Section 5. Section 6 proves the rules of CoC sound wrt. our model, and as a corollary we obtain termination and completeness of NbE and consistency of CoC. Loose ends are listed in the conclusions (Section 7).

2 Syntax

We present the Calculus of Constructions (CoC) as a pure type system (PTS) with annotated λ-abstraction, typed equality and explicit substitutions.

1. *Annotated λ-abstraction* (as in $\lambda\, M\, N$) enables us to compute the type of a term from the type of its free variables and its semantics from the semantics of its free variables (see Section 6). Thus, a term makes already sense in a context alone, it does not need an ascribed type.
2. *Typed equality* is the natural choice in the presence of η since untyped η-reduction is badly behaved in set-theoretical models and type-theoretical models without subtyping —this includes the semantics we are constructing. (Untyped η-expansion cannot be defined.)
3. Lambda calculi with *explicit substitutions* have more models than lambda calculi with substitution implemented as an operation. In particular, the model of closures in weak head normal form we will use in Section 3. Recent meta theoretic studies involving explicit substitutions include [Dan07,Cha09,Gra09,ACP09]. In the presence of explicit substitutions, variables are most naturally represented as de Bruijn indices [ACCL91].

2.1 Expressions and Typing

The CoC is a dependently typed lambda calculus with expressions on three levels: *terms* t, u, the data structures and programs of the language; the *types* T, U of terms, generalized to a lambda-calculus of type constructors[1]; and the kinds κ, ι, the types of types.

[1] E.g., List is a type constructor which produces a type of homogeneous lists $\mathsf{List}\, T$ for each element type T.

In the PTS-style presentation, there is just one language of *expressions* M, N for all three levels, and the classification of expressions into terms, type constructors, and kinds is a byproduct of typing, using the two *sorts* $s ::= *, \Box$. The inhabitants of sort $\Box$ are kinds, one of which is $*$, and the inhabitants of sort $*$ are types, whose inhabitants in turn are terms.

From our perspective, the CoC is a dependent version of System F^ω. In terms of the Calculus of Inductive Constructions, the core language of Coq [INR08], the sort $*$ is the impredicative `Set`, and the sort $\Box$ the predicative $\texttt{Type}_0$. Alternatively [Wer92], one could identify $*$ with the impredicative `Prop` and refer to the levels as *proof terms, predicates, and kinds* instead.

Syntax of expressions, substitutions, and contexts. We represent bound variables by de Bruijn indices; the 0th variable is represented by the expression v_0, the ith variable by the i-fold application of the lifting substitution $\uparrow$ to the expression v_0. Consequently, we use the expression v_i as a shorthand for the expression $\mathsf{v}_0 \uparrow^i$. Also, we abbreviate (id, N) by $[N]$.

$$
\begin{array}{lll}
\mathsf{Sort} & \ni s & ::= * \mid \Box \\
\mathsf{Exp} & \ni M, N, t, u, T, U, \kappa, \iota & ::= s \mid \mathsf{v}_0 \mid \lambda\, M\, N \mid M\, N \mid \Pi\, M\, N \mid M\, \sigma \\
\mathsf{Subst} & \ni \sigma, \tau & ::= \uparrow \mid \mathsf{id} \mid \sigma\, \tau \mid (\sigma, M) \\
\mathsf{Cxt} & \ni \Gamma, \Delta & ::= () \mid \Gamma, M
\end{array}
$$

We denote the length of context Γ by $\|\Gamma\|$. We use $\equiv$ for literal identity of expressions, and the dot notations $\Pi\, U.\, T$ and $\lambda\, U.\, M$ to save parentheses.

Normal forms are those expressions that do not contain substitutions (except lifting of an index) or β-redexes. Normal forms starting with a variable are called *neutral*.

$$
\begin{array}{lll}
\mathsf{Norm} \ni v, w, V, W & ::= s \mid \lambda V w \mid \Pi\, V\, W \mid n & \beta\text{-normal form} \\
\mathsf{Neut} \ni n, N & ::= \mathsf{v}_i \mid n\, v & \text{neutral normal form}
\end{array}
$$

Typing. The judgements $\Gamma \vdash$ "Γ is a well-formed context" and $\Gamma \vdash M : N$ "M has type N in context Γ" are given inductively by the following rules.

$$
\frac{}{() \vdash} \qquad \frac{\Gamma \vdash \qquad \Gamma \vdash T : s}{\Gamma, T \vdash}
$$

$$
\frac{\Gamma \vdash}{\Gamma \vdash * : \Box} \qquad \frac{\Gamma \vdash U : s \qquad \Gamma, U \vdash T : s'}{\Gamma \vdash \Pi\, U\, T : s'}
$$

$$
\frac{\Gamma \vdash T : s}{\Gamma, T \vdash \mathsf{v}_0 : T \uparrow} \qquad \frac{\Gamma \vdash U : s \qquad \Gamma, U \vdash T : s' \qquad \Gamma, U \vdash M : T}{\Gamma \vdash \lambda\, U\, M : \Pi\, U\, T}
$$

$$
\frac{\Gamma \vdash M : \Pi\, U\, T \qquad \Gamma \vdash N : U}{\Gamma \vdash M\, N : T[N]} \qquad \frac{\Gamma \vdash M : T \qquad \Gamma \vdash T = T' : s}{\Gamma \vdash M : T'}
$$

We come to the judgement for type equality, $\Gamma \vdash T = T' : s$, later. The typing rules for substitutions are:

$$\frac{\Gamma \vdash \sigma : \Delta \qquad \Delta \vdash M : T}{\Gamma \vdash M\,\sigma : T\,\sigma} \qquad \frac{\Gamma \vdash \sigma : \Delta \qquad \Delta \vdash T : s \qquad \Gamma \vdash M : T\,\sigma}{\Gamma \vdash (\sigma, M) : \Delta, T}$$

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{id} : \Gamma} \qquad \frac{\Gamma_1 \vdash \tau : \Gamma_2 \qquad \Gamma_2 \vdash \sigma : \Gamma_3}{\Gamma_1 \vdash \sigma\,\tau : \Gamma_3} \qquad \frac{\Gamma \vdash T : s}{\Gamma, T \vdash \uparrow : \Gamma}$$

For $i < \|\Gamma\|$, we define context look-up $\Gamma(i)$ by $(\Gamma, T)(0) = T\uparrow$ and $(\Gamma, T)(i+1) = \Gamma(i)\uparrow$. It is easy to see that the general variable rule is derivable by induction on i:

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{v}_i : \Gamma(i)}$$

Using this rule, we can understand typing of expressions from the first set of rules alone, under an abstract view on substitution and equality.

2.2 Typed Equality

We formalize $\beta\eta\sigma$-equality by the judgements $\Gamma \vdash M = M' : T$ and $\Gamma \vdash \sigma = \sigma' : \Delta$. Equality holds only between well-formed objects of syntax, thus the rules have to be formulated such that they entail $\Gamma \vdash M : T$ (and likewise for M', σ, and σ'). For instance, the η-rule reads:

$$\frac{\Gamma \vdash M : \Pi\,U\,T}{\Gamma \vdash M = \lambda\,U.\,(M\uparrow)\,\mathsf{v}_0 : \Pi\,U\,T}$$

We will not spell out the rules will types and typing assumptions. Instead, we will just write down axioms in the form $M = M'$ and $\sigma = \sigma'$, the typing can be reconstructed. Also we will skip all congruence rules expressing that equality is an equivalence relation and that it is closed under all syntactic constructions. We have taken a similar approach before [ACP09] which is justified by Cartmell's work on generalized algebraic theories [Car86].

Computation: β, resolution of substitutions, pushing substitution under constructions.

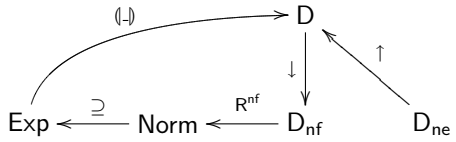
$$\begin{array}{ll} (\lambda U M)\,N = M[N] & (M\,N)\,\sigma = (M\,\sigma)\,(N\,\sigma) \\ \mathsf{v}_0\,(\sigma, M) = M & (\lambda\,U\,M)\,\sigma = \lambda\,U\,\sigma.\,M\,(\sigma\uparrow, \mathsf{v}_0) \\ \uparrow(\sigma, M) = \sigma & (\Pi\,U\,T)\,\sigma = \Pi\,U\,\sigma.\,T\,(\sigma\uparrow, \mathsf{v}_0) \\ M\,\mathsf{id} = M & (M\,\sigma)\,\tau = M\,(\sigma\,\tau) \\ s\,\sigma = s & (\sigma, M)\,\tau = (\sigma\,\tau, M\,\tau) \end{array}$$

Non-computational rules: Extensionality and rules of the category of substitutions.

$$\begin{array}{ll} M = \lambda\,U.\,(M\uparrow)\,\mathsf{v}_0 & \mathsf{id}\,\sigma = \sigma \\ \mathsf{id} = (\uparrow, \mathsf{v}_0) & \sigma\,\mathsf{id} = \sigma \\ & (\sigma_1\,\sigma_2)\,\sigma_3 = \sigma_1\,(\sigma_2\,\sigma_3) \end{array}$$

3 Normalization by Evaluation

We conceive *normalization by evaluation* as the composition of a standard interpreter $(\!|\_|\!) : \mathsf{Exp} \to \mathsf{D}$ mapping expressions into a semantics D and a *reifier* which computes a long normal form from a value in D. Coquand [ACP09] observed that the η-expansion part of reification can be carried out entirely within the semantics which splits reification into an *η-expansion* phase $\downarrow : \mathsf{D} \to \mathsf{D}_{\mathsf{nf}}$ and a *read-back* phase $\mathsf{R}^{\mathsf{nf}} : \mathsf{D}_{\mathsf{nf}} \to \mathsf{Norm} \subseteq \mathsf{Exp}$.



η-expansion $\downarrow$ is mutually defined with reflection $\uparrow : \mathsf{D}_{\mathsf{ne}} \to \mathsf{D}$ which injects variables (de Bruijn levels), and more generally neutral values $e \in \mathsf{D}_{\mathsf{ne}}$, into the semantics in η-expanded form.

3.1 Weak head evaluation

While having used a Scott domain to represent values in previous work [ACP09], we now change to *partial applicative structures* which subsume Scott domains (and, indeed, all λ-models and -algebras). The following representation of values by closures is such a structure and it can be directly formalized in type theory which is not the case for any effective total applicative structure.

Values are defined in terms of closures $(\underline{\lambda} t)\eta$ and de Bruijn levels. Level x_j represents the jth free variable. In effect, we are using a *locally nameless* presentation of values [Pol94] where bound variables are represented as relative references (de Bruijn indices v_i) and free variables as absolute references, i. e., names (de Bruijn levels x_j). The delayed η-expansions $\uparrow^{\underline{\Pi}\, A\, F} e$ and $\downarrow^{\underline{\Pi}\, A\, F} f$ are the defunctionalization of reflection and reification in the Scott domain (as closures are the defunctionalization of evaluation).

D	$\ni a, b, f, A, B, F, G, K, L$	$::= s \mid \underline{\Pi}\, A\, F \mid (\underline{\lambda} t)\eta \mid \uparrow^{\underline{\Pi}\, A\, F} e \mid e$	value
DNe	$\ni e, E$	$::= \mathsf{x}_j\ (j \in \mathbb{N}) \mid e\, d$	neutral value
DNf	$\ni d, D$	$::= \downarrow^{\underline{\Pi}\, A\, F} f \mid a$	normal value
Env	$\ni \eta$	$::= \mathsf{id} \mid (\eta, a)$	environment

We let $\uparrow^B e = e$ and $\downarrow^B a = a$ if B is not a Π-type.

Evaluation and application. We introduce the judgements $(\!|M|\!)_\eta \searrow a$ "in environment η, expression M evaluates to a", $(\!|\sigma|\!)_\eta \searrow \eta'$ "in environment η, substitution σ evaluates to environment η'", and $f \cdot a \searrow b$ "value f applied to value a evaluates to b" inductively

by the following rules.

$$\frac{}{(\!|s|\!)_\eta \searrow s} \qquad \frac{}{(\!|\mathsf{v}_0|\!)_{(\eta,a)} \searrow a} \qquad \frac{}{(\!|\lambda\, U\, t|\!)_\eta \searrow (\underline{\lambda} t)\eta} \qquad \frac{(\!|U|\!)_\eta \searrow A}{(\!|\Pi\, U\, T|\!)_\eta \searrow \underline{\Pi}\, A\, (\underline{\lambda} T)\eta}$$

$$\frac{(\!|t|\!)_\eta \searrow f \qquad (\!|u|\!)_\eta \searrow a \qquad f \cdot a \searrow b}{(\!|t\, u|\!)_\eta \searrow b} \qquad \frac{(\!|\sigma|\!)_\eta \searrow \eta' \qquad (\!|t|\!)_{\eta'} \searrow a}{(\!|t\, \sigma|\!)_\eta \searrow a}$$

$$\frac{}{(\!|\uparrow|\!)_{(\eta,a)} \searrow \eta} \qquad \frac{}{(\!|\mathsf{id}|\!)_\eta \searrow \eta} \qquad \frac{(\!|\sigma|\!)_\eta \searrow \eta' \qquad (\!|t|\!)_\eta \searrow a}{(\!|(\sigma, t)|\!)_\eta \searrow (\eta', a)} \qquad \frac{(\!|\tau|\!)_\eta \searrow \eta' \qquad (\!|\sigma|\!)_{\eta'} \searrow \eta''}{(\!|\sigma\, \tau|\!)_\eta \searrow \eta''}$$

$$\frac{(\!|t|\!)_{(\eta,a)} \searrow b}{(\underline{\lambda} t)\eta \cdot a \searrow b} \qquad \frac{F \cdot a \searrow B}{(\uparrow^{\underline{\Pi}\, A\, F} e) \cdot a \searrow \uparrow^B (e \downarrow^A a)}$$

These three relations are deterministic, thus, they can be turned into partial functions $(\!|\_|\!)_\_ : \mathsf{Exp} \times \mathsf{Env} \rightharpoonup \mathsf{D}$, $(\!|\_|\!)_\_ : \mathsf{Subst} \times \mathsf{Env} \rightharpoonup \mathsf{Env}$, and $\_ \cdot \_ : \mathsf{D} \times \mathsf{D} \rightharpoonup \mathsf{D}$.

3.2 Read-back (aka Reification)

We introduce two judgements $m \vdash d \searrow v$ "at level m, normal value d reifies to normal form v", and $m \vdash^{\mathsf{ne}} e \searrow n$ "at level m, neutral value reifies to neutral normal form n" inductively by the following rules. The natural number m corresponds to the de Bruijn level of the next fresh variable.

$$\frac{}{m \vdash * \searrow *} \qquad \frac{m \vdash A \searrow V \qquad F \cdot \uparrow^A \mathsf{x}_m \searrow B \qquad m+1 \vdash B \searrow W}{m \vdash \underline{\Pi}\, A\, F \searrow \Pi\, V\, W}$$

$$\frac{m \vdash A \searrow V \qquad F \cdot \uparrow^A \mathsf{x}_m \searrow B \qquad f \cdot \uparrow^A \mathsf{x}_m \searrow b \qquad m+1 \vdash \downarrow^B b \searrow w}{m \vdash \downarrow^{\underline{\Pi}\, A\, F} f \searrow \lambda\, V\, w}$$

$$\frac{m \vdash^{\mathsf{ne}} e \searrow n}{m \vdash e \searrow n} \qquad \frac{}{m \vdash^{\mathsf{ne}} \mathsf{x}_j \searrow \mathsf{v}_{m-(j+1)}} \qquad \frac{m \vdash^{\mathsf{ne}} e \searrow n \qquad m \vdash d \searrow v}{m \vdash^{\mathsf{ne}} e\, d \searrow n\, v}$$

In the but last rule, we use the "monus" function on $\mathbb{N}$ where $m - m' = 0$ if $m' \geq m$.

Read-back is deterministic, so we introduce two partial functions by $\mathsf{R}^{\mathsf{nf}}_m\, d = v$ iff $m \vdash d \searrow v$ and $\mathsf{R}^{\mathsf{ne}}_m\, e = n$ iff $m \vdash^{\mathsf{ne}} e \searrow n$. These correspond to the read-back function by Gregoire and Leroy [GL02] and own previous work [ACP09].

3.3 Type Inference

In the following we adopt bidirectional value-based type checking [Coq96,ACD08] to de Bruijn style. Since we have typed abstraction, the type of every well-typed term is inferable. We specify the type inference algorithm by a deterministic inductive judgement

$$\Delta \vdash t \rightrightarrows A$$

meaning that in context Δ, the principal type of t is A. We keep context Δ and type A in evaluated form. The algorithm is very similar to Huet's *constructive engine* [Hue89] as refined by Pollack [Pol06].

By induction on a context of values Δ, we define the η-expanded environment id_Δ which maps de Bruijn indices to their corresponding de Bruijn levels. We write x_Δ for $\mathsf{x}_{\|\Delta\|}$.

$$\mathsf{id}_{()} = \mathsf{id} \qquad\qquad \mathsf{id}_{\Delta,A} = (\mathsf{id}_\Delta, \uparrow^A \mathsf{x}_\Delta)$$

Type inference is given inductively by the following rules. Note that only type checked terms are evaluated, and only values enter the contexts or are returned.

$$\frac{}{\Delta \vdash * \rightrightarrows \Box} \qquad \frac{\Delta \vdash U \rightrightarrows s \qquad \Delta, (\!|U|\!)_{\mathsf{id}_\Delta} \vdash T \rightrightarrows s'}{\Delta \vdash \Pi\, U\, T \rightrightarrows s'} \qquad \frac{}{\Delta \vdash \mathsf{v}_i \rightrightarrows \Delta(i)}$$

$$\frac{\Delta \vdash U \rightrightarrows s \qquad (\!|U|\!)_{\mathsf{id}_\Delta} \searrow A \qquad \Delta, A \vdash t \rightrightarrows B \qquad \Delta, A \vdash B \xrightarrow{\lambda} F}{\Delta \vdash \lambda\, U\, t \rightrightarrows \underline{\Pi}\, A\, F}$$

$$\frac{\Delta \vdash t \rightrightarrows \underline{\Pi}\, A\, F \qquad \Delta \vdash u \rightrightarrows B \qquad \Delta \vdash A \cong B}{\Delta \vdash t\, u \rightrightarrows F \cdot (\!|u|\!)_{\mathsf{id}_\Delta}}$$

In the last two rules we have used two auxiliary judgements. In the application rule, we check the ascribed type A and the inferred type B for $\beta\eta$-equality using $\Delta \vdash A \cong B$. In the application rule, we need to turn the type value B of the function body t into a function over the last variable x_Δ, which is of type A. We write $\Delta, A \vdash B \xrightarrow{\lambda} F$ for this abstraction operation.

$$\begin{aligned} \Delta \vdash A \cong B \qquad &\iff \|\Delta\| \vdash A \searrow V \text{ and } \|\Delta\| \vdash B \searrow V \\ \Delta, A \vdash B \xrightarrow{\lambda} F \quad &\iff \|\Delta, A\| \vdash B \searrow V \text{ and } F \equiv (\underline{\lambda} V)\mathsf{id} \end{aligned}$$

Type values A and B are equal if they reify to the same normal form V. Due to our locally nameless style values, abstraction $\Delta, A \vdash B \xrightarrow{\lambda} F$ is a bit cumbersome. We implement it by first reifying value B in context Δ, A to term V and then building the closure $F \equiv (\underline{\lambda} V)\mathsf{id}$.

3.4 Normalization

During type inference, values are reified to normal forms to test equality. We can also compose evaluation and read-back to obtain a normalization function for terms. Let $(\!|\Gamma|\!)$ be evaluation of contexts partially defined by

$$(\!|()|\!) = () \qquad\qquad (\!|\Gamma, U|\!) = (\!|\Gamma|\!), (\!|U|\!)_{\mathsf{id}_{(\!|\Gamma|\!)}}$$

Using the partially defined identity environment $\eta_\Gamma := \mathsf{id}_{(\!|\Gamma|\!)}$, the partial normalization function is now obtained as

$$\begin{aligned} \mathsf{nbe}^T_\Gamma(t) \quad &= \mathsf{R}^{\mathsf{nf}}_{\|\Gamma\|} \left(\downarrow^{(\!|T|\!)_{\eta_\Gamma}} (\!|t|\!)_{\eta_\Gamma}\right) \\ \mathsf{Nbe}_\Gamma(T) &= \mathsf{nbe}^{\Box}_\Gamma(T). \end{aligned}$$

The goal of this work is to show its correctness on well-formed expressions, i. e.:

1. Soundness: if $\Gamma \vdash t : T$ then $\Gamma \vdash t = \mathsf{nbe}^T_\Gamma(t) : T$.
2. Completeness: if $\Gamma \vdash t = t' : T$ then $\mathsf{nbe}^T_\Gamma(t) \equiv \mathsf{nbe}^T_\Gamma(t')$.
3. Termination: if $\Gamma \vdash t : T$ then $\mathsf{nbe}^T_\Gamma(t)$ is defined.

The termination property is a consequence of soundness and also of completeness, since judgemental equality and expression equality presuppose definedness. Remarks on soundness can be found in the long version of this paper [Abe10]. In the remainder of the paper, we will focus on completeness, which will be established by a PER model construction.

4 Classification of Expressions

If $\Gamma \vdash \kappa : \Box$, then κ is a called a *kind*. If $\Gamma \vdash T : \kappa$ for a kind κ, then T is called a *type constructor*. In particular, if $\Gamma \vdash T : *$, then T is called a *type*. If $\Gamma \vdash t : T$ for a type T, then t is called a *term*. We obtain three syntactic subclasses Kind, Ty, Tm of Exp. There are no kind variables, only term variables and type (constructor) variables.

$$\begin{array}{lll} \mathsf{Kind} \ni \kappa, \iota & ::= * \mid \Pi\, \kappa\, \kappa' \mid \Pi\, U\, \kappa \\ \mathsf{Ty} \quad \ni T, U & ::= \Pi\, U\, T \mid \Pi\, \kappa\, T \\ & \quad \mid X \mid \lambda\, \kappa\, T \mid T\, U \mid \lambda\, U\, T \mid T\, u \\ \mathsf{Tm} \quad \ni t, u & ::= x \mid \lambda\, U\, t \mid t\, u \mid \lambda\, \kappa\, t \mid t\, U \end{array}$$

It is well-known that all dependencies in pure CoC can be *erased* such that one ends up with the terms, type constructors, and kinds of System F^ω. This way, one can inherit normalization of CoC from F^ω [GN91]. However, since we want to add inductive types in $*$ with large eliminations into $*$ (just as Werner [Wer92]), we cannot pursue this path; CoC with natural numbers has types defined by recursion on a number, so it can express types of functions with varying arity, like

$$(X : *) \to (n : \mathsf{Nat}) \to \underbrace{X \to \cdots \to X}_{n \text{ times}} \to X$$

which have no counterpart in F^ω. However, without large eliminations into $\Box$, so no kinds defined by recursion, the structure of kinds is still simple and dependencies can be erased on the kind level. This observation by Coquand and Gallier [CG90] and Werner [Wer92] has been exploited by Barras and Werner [BW97] to completely formalize strong normalization of pure CoC in Coq. Following their lead, we will use erased kinds to bootstrap our model construction in Section 5.

Simple kinds. We enrich the kinds of F^ω by a base kind $\diamond$ of terms.

$$\begin{array}{lll} \mathsf{SKi} \ni k & ::= \diamond \mid l & \text{simple kind} \\ \mathsf{SKiP} \ni l & ::= * \mid k \to l & \text{proper simple kind} \\ \mathsf{SCxt} \ni \gamma, \delta & ::= () \mid \gamma, k & \text{simple kinding context} \end{array}$$

The simple kind $\diamond \to k$ is the erasure of the indexed kind $\Pi\, T\, \kappa$. The grammar forbids $k \to \diamond$, the kind of functions from constructors of simple kind k to terms, which is a subset of the terms, so we set $k \to \diamond := \diamond$.

We define the judgements $\gamma \vdash M \div k$ "in context γ, expression M has simple kind k", $\gamma \vdash \sigma \div \delta$ "in context γ, substitution σ has simple kinding δ, and $\gamma \vdash M \doteq k$ "in context γ, expression M has skeleton k" inductively by the rules to follow. These rules are basically an erasure of the typing rules. Kinds κ are related to their skeleton k by $\gamma \vdash \kappa \doteq k$, where Types $T : *$ are assigned skeleton $\diamond$. Type constructors $T : \kappa$ are be related to the skeleton k of κ by judgement $\gamma \vdash T \div k$, and terms to skeleton $\diamond$.

$$\frac{}{\gamma \vdash * \doteq *} \qquad \frac{\gamma \vdash U \doteq k \qquad \gamma, k \vdash T \doteq k'}{\gamma \vdash \Pi\, U\, T \doteq k \to k'} \qquad \frac{\gamma \vdash T \div *}{\gamma \vdash T \doteq \diamond}$$

$$\frac{}{\gamma, k \vdash \mathsf{v}_0 \div k} \qquad \frac{\gamma \vdash U \doteq k \qquad \gamma, k \vdash M \div k'}{\gamma \vdash \lambda\, U\, M \div k \to k'}$$

$$\frac{\gamma \vdash M \div k \to k' \qquad \gamma \vdash N \div k}{\gamma \vdash M\, N \div k'} \qquad \frac{\gamma \vdash \sigma \div \delta \qquad \delta \vdash M \star k}{\gamma \vdash M\, \sigma \star k}\ \star \in \{\div, \doteq\}$$

$$\frac{}{\gamma, k \vdash \uparrow \div \gamma} \qquad \frac{}{\gamma \vdash \mathsf{id} \div \gamma} \qquad \frac{\gamma \vdash \sigma \div \delta \qquad \gamma \vdash M \div k}{\gamma \vdash (\sigma, M) \div \delta, k}$$

$$\frac{\gamma_1 \vdash \tau \div \gamma_2 \qquad \gamma_2 \vdash \sigma \div \gamma_3}{\gamma_1 \vdash \sigma\, \tau \div \gamma_3}$$

Shape computation. We now define two (total) functions, $|M|^{\div}_{\gamma}$ "the kind of M in simple context γ", and $|M|^{\doteq}_{\gamma}$ "the skeleton of M in simple context γ", by means of a general *shape* function $|M|_{\gamma}$ which returns a pair $(\star, k)$ with $\star \in \{\doteq, \div\}$:

$$|M|^{\doteq}_{\gamma} = \begin{cases} k & \text{if } |M|_{\gamma} = (\doteq, k) \\ \diamond & \text{otherwise} \end{cases} \qquad |M|^{\div}_{\gamma} = \begin{cases} k & \text{if } |M|_{\gamma} = (\div, k) \\ * & \text{otherwise} \end{cases}$$

The shape function is defined mutually with the function $|\sigma|^{\div}_{\gamma}$, written $|\sigma|_{\gamma}$, which computes the kinds of the expressions in σ. We also define the skeleton $|\Gamma|$ of a context.

$$\begin{aligned}
|*|_{\gamma} &= (\doteq, *) \\
|\Pi\, U\, T|_{\gamma} &= (\doteq, |U|^{\doteq}_{\gamma} \to |T|^{\doteq}_{\gamma, |U|^{\doteq}_{\gamma}}) \\
|\mathsf{v}_0|_{\gamma, k} &= (\div, k) \\
|\mathsf{v}_0|_{()} &= (\div, \diamond) \\
|\lambda\, U\, M|_{\gamma} &= (\div, |U|^{\doteq}_{\gamma} \to |M|^{\div}_{\gamma, |U|^{\doteq}_{\gamma}}) \\
|M\, N|_{\gamma} &= \begin{cases} (\div, k') & \text{if } |M|^{\div}_{\gamma} = k \to k' \\ (\div, \diamond) & \text{otherwise} \end{cases} \\
|M\, \sigma|_{\gamma} &= |M|_{|\sigma|_{\gamma}}
\end{aligned}$$

$$\begin{aligned}
|\uparrow|_{\gamma, k} &= \gamma \\
|\uparrow|_{()} &= () \\
|\mathsf{id}|_{\gamma} &= \gamma \\
|(\sigma, M)|_{\gamma} &= |\sigma|_{\gamma}, |M|^{\div}_{\gamma} \\
|\sigma\, \tau|_{\gamma} &= |\sigma|_{|\tau|_{\gamma}} \\
\\
|()| &= () \\
|\Gamma, T| &= |\Gamma|, |T|^{\doteq}_{|\Gamma|}
\end{aligned}$$

Lemma 1 (Soundness of shape computation). *For $L ::= M \mid \sigma$ and $R ::= k \mid \gamma$ and $\gamma \vdash L \star R$ we have $R = |L|^{\star}_{\gamma}$.*

Lemma 2 (Kind skeleton independence). $|M|^{\doteq}_{\gamma} = |M|^{\doteq}_{\gamma'}$ *for all* γ, γ'.

Therefore, we may suppress γ and just write $|M|^{\doteq}$.

Theorem 1 (Shapes of wellformed expressions). *Let* $\gamma = |\Gamma|$.

1. *If* $\Gamma \vdash \kappa : \Box$ *then* $\gamma \vdash \kappa \doteq |\kappa|^{\doteq}_{\gamma}$.
2. *If* $\Gamma \vdash \kappa = \kappa' : \Box$ *then* $|\kappa|^{\doteq}_{\gamma} = |\kappa'|^{\doteq}_{\gamma}$.
3. *If* $\Gamma \vdash T : *$ *then* $\gamma \vdash T \doteq |T|^{\doteq}_{\gamma} = \diamond$.
4. *If* $\Gamma \vdash M : T \not\equiv \Box$ *then* $\gamma \vdash M \div |M|^{\div}_{\gamma} = |T|^{\doteq}_{\gamma}$.
5. *If* $\Gamma \vdash \sigma : \Delta$ *then* $\gamma \vdash \sigma \div |\sigma|_{\gamma} = |\Delta|$.

Proof. Simultaneously by induction on the typing/equality derivation. Note that $\Gamma \vdash M = M' : T \not\equiv \Box$ implies $|M|^{\div}_{\gamma} = |M'|^{\div}_{\gamma}$ since the kinds of M and M' equal the skeleton of T.

5 A Model for the $\beta\eta$-CoC with Large Eliminations

In this section, we present a PER model of CoC. Each expression M is modeled by a pair $(F, \mathcal{F})$ where $F : \mathsf{D}$ is simply the value of M and $\mathcal{F}$ is the semantic role of M. *Terms* $t \div \diamond$ have no semantic role, they are modeled by a pair $(a, ())$. *Types* $T \div *$ are modeled by a pair $(A, \mathcal{A})$ where $\mathcal{A}$ is a partial equivalence relation (PER) between semantic terms. The objects $(a, ())$ and $(a', ())$ are related by $\mathcal{A}$ iff, intuitively, a and a' are $\beta\eta$-equal values of type A. Formally that means that $\downarrow^A a$ and $\downarrow^A a'$ must read back as the same expression; this connection between A and $\mathcal{A}$ is written $A \Vdash \mathcal{A}$ and pronounced "A realizes $\mathcal{A}$". *Type constructors* $T \div k \to k'$ are modeled as $(F, \mathcal{F})$ where $\mathcal{F}$ is a higher-order operator on PERs, it maps constructors $(G, \mathcal{G})$ of kind k to constructors $\mathcal{F}(G, \mathcal{G})$ of kind k'. Note that unlike in System F^ω or erased versions of the CoC [BW97,Geu94], $\mathcal{F}$ also depends on a value G (see [Wer92,SG96]). Finally kinds $\kappa \doteq k$ are modeled as $(K, \mathcal{K})$ where the PER $\mathcal{K}$ relates constructors $(F, \mathcal{F})$ and $(F', \mathcal{F}')$ of kind k if $\mathcal{F}$ and $\mathcal{F}'$ are extensionally equal operators and $\downarrow^K F$ and $\downarrow^K F'$ have the same normal form (thus, K realizes $\mathcal{K}$). To avoid duplication we have modeled types and kinds uniformly in the formal presentation of the semantics, probably at the cost of readability; may this informal exposition serve as an Ariadne thread in the maze to follow.

Meta language. We use an impredicative type-theoretic meta language, i. e., we will not speak in terms of sets, but in terms of types and predicates. However, we will use some set-theoretic notation with care. For a type α, the type $\mathcal{P}(\alpha)$ contains the predicates over α, and for $P : \mathcal{P}(\alpha)$ and $a : \alpha$ we write $a \in P$ if $P(a)$ holds. The subset type $\{a : \alpha \mid P(a)\}$ is the type of pairs (a, p) such that p is a proof of $P(a)$. Usually, we suppress the proof and write just $a \in \{a : \alpha \mid P(a)\}$. The value $f(a, p)$ of a function $f : \{a : \alpha \mid P(a)\} \to \beta$ may not depend on the form of the proof p.

A setoid is a pair of a type α and an equivalence relation $=_\alpha : \mathcal{P}(\alpha \times \alpha)$. We write α for the setoid. A function $f : \alpha \to \beta$ is a (setoid) morphism, $f \in \alpha \to \beta$, if it respects setoid equality, i. e., $a =_\alpha a'$ implies $f(a) =_\beta f(a')$. Two morphisms f, f' are equal, $f =_{\alpha\to\beta} f'$ iff $a =_\alpha a'$ implies $f(a) =_\beta f'(a)$. This makes $(\alpha \to \beta, =_{\alpha\to\beta})$ a setoid in turn.

A partial equivalence relation $\mathcal{A} : \mathsf{Per}(\alpha)$ is a binary relation over type α which is symmetric and transitive. We write $a = a' \in \mathcal{A}$ for $a, a' : \alpha$ with $(a, a') \in \mathcal{A}$, and $a \in \mathcal{A}$ for $a = a \in \mathcal{A}$. Equality $\mathcal{A} = \mathcal{A}'$ of PERs holds extensionally if for all $a, a' : \alpha$, $a = a' \in \mathcal{A}$ iff $a = a' \in \mathcal{A}'$. Each PER $\mathcal{A}$ can be coerced into an associated setoid $\{a : \alpha \mid a \in \mathcal{A}\}$ with setoid equality $\mathcal{A}$. This defines the notion of morphism $\mathcal{F} \in \mathcal{A} \to \beta$ from PER $\mathcal{A}$ to setoid β. We define $\mathsf{Ne} : \mathsf{Per}(\mathsf{DNe})$ and $\mathsf{Nf} : \mathsf{Per}(\mathsf{DNf})$ by

$$\begin{aligned} e = e' \in \mathsf{Ne} &\iff \forall m : \mathbb{N}.\, \exists n : \mathsf{Neut}.\, m \vdash^{\mathsf{ne}} e \searrow n \text{ and } m \vdash^{\mathsf{ne}} e' \searrow n \\ d = d' \in \mathsf{Nf} &\iff \forall m : \mathbb{N}.\, \exists v : \mathsf{Norm}.\, m \vdash d \searrow v \text{ and } m \vdash d' \searrow v. \end{aligned}$$

Note that transitivity follows from determinism of read-back.

A partial function $\mathcal{H} : \alpha \rightharpoonup \beta$ is a pair $(\mathsf{dom}(\mathcal{H}) : \mathcal{P}(\alpha), \mathsf{apply}(\mathcal{H}) : \{a : \alpha \mid a \in \mathsf{dom}\,\mathcal{H}\} \to \beta)$ where $\mathsf{dom}(\mathcal{H})$ and $\mathsf{apply}(\mathcal{H})$ respect the setoid equalities associated to α and β. We will write $\mathcal{H}(a) \downarrow$ "$\mathcal{H}(a)$ is defined" if there is a proof p that $a \in \mathsf{dom}(\mathcal{H})$, and then $\mathcal{H}(a)$ stands for $\mathsf{apply}(\mathcal{H})(a, p)$. Two partial functions $\mathcal{H}, \mathcal{H}'$ are equal $\mathcal{H} =_{\alpha\rightharpoonup\beta} \mathcal{H}'$ if they have equal domains and coincide pointwise (wrt. to setoid equalities); this makes $\alpha \rightharpoonup \beta$ a setoid.

Raw interpretation of kinds. Let $()$ denote the unit type with single inhabitant $()$. We define the candidate space $\langle k \rangle$ for simple kind k and an inhabitant $\bot^k : \langle k \rangle$ by recursion on k:

$$\begin{array}{lll} \langle \diamond \rangle & = () & \bot^\diamond & = () \\ \langle * \rangle & = \mathsf{Per}(\mathsf{D} \times ()) & \bot^* & = \mathsf{Ne}^{()} \\ \langle k \to k' \rangle & = (\mathsf{D} \times \langle k \rangle) \rightharpoonup \langle k' \rangle & \bot^{k\to k'}(G : \mathsf{D}, \mathcal{G} : \langle k \rangle) & = \bot^{k'} \end{array}$$

where $\mathsf{Ne}^{()} = \mathsf{Ne} \times () = \{((e, ()), (e', ())) \mid e = e' \in \mathsf{Ne}\}$. Extensional setoid equality $\mathcal{F} =_{\langle k \rangle} \mathcal{F}'$ is defined along the way. We set $\langle\!\langle k \rangle\!\rangle = \mathsf{Per}(\mathsf{D} \times \langle k \rangle)$. Note that $\langle\!\langle \diamond \rangle\!\rangle = \langle * \rangle$.

Sort interpretation. For $K : \mathsf{D}, \mathcal{K} : \langle\!\langle k \rangle\!\rangle$ we define $\underline{K}, \overline{K} : \langle\!\langle k \rangle\!\rangle$ and $K \Vdash \mathcal{K}$, "K realizes $\mathcal{K}$", by

$$\begin{aligned} (F, \mathcal{F}) = (F', \mathcal{F}') \in \overline{K} &\iff \downarrow^K F = \downarrow^K F' \in \mathsf{Nf} \\ (\uparrow^K E, \bot^k) = (\uparrow^K E', \bot^k) \in \underline{K} &\iff E = E' \in \mathsf{Ne} \\ K \Vdash \mathcal{K} &\iff \underline{K} \subseteq \mathcal{K} \subseteq \overline{K} \end{aligned}$$

In words, code K realizes PER $\mathcal{K}$ iff equal inhabitants F of $\mathcal{K}$ reify to the same normal form, where K directs the amount of η-expansion during reification; and neutrals E can be reflected at code K into $\mathcal{K}$. Equivalently, we could say $K \Vdash \mathcal{K}$ iff $\downarrow^K \circ \pi_1 \in \mathcal{K} \to \mathsf{Nf}$, "$\downarrow^K$ after the first projection is a PER morphism from $\mathcal{K}$ to Nf", and $(E : \mathsf{DNe} \mapsto (\uparrow^K E, \bot^k)) \in \mathsf{Ne} \to \mathcal{K}$, "$\uparrow^K$ paired with $\bot^k$ is a PER morphism from Ne to $\mathcal{K}$".

Lemma 3 (Least PER is a candidate). *For all $E \in \mathsf{Ne}$, $E \Vdash \bot^* : \langle\!\langle \diamond \rangle\!\rangle$.*

Equality of candidates $(K, \mathcal{K}) = (K', \mathcal{K}') \in \overline{\overline{k}}$ shall hold iff

1. $K \Vdash \mathcal{K}$ and $K' \Vdash \mathcal{K}'$.
2. $\mathcal{K} =_{\langle\langle k \rangle\rangle} \mathcal{K}'$ and $\downarrow K = \downarrow K' \in \mathsf{Nf}$.
3. $\uparrow^K = \uparrow^{K'} \in \mathsf{Ne} \times \{\bot^k\} \to \mathcal{K}$ and $\downarrow^K = \downarrow^{K'} \in \pi_1(\mathcal{K}) \to \mathsf{Nf}$.

This states that $\mathcal{K}$ and $\mathcal{K}'$ are extensionally equal PERs, plus the associated codes K and K' are reifiable, plus they are indistinguishable with respect to their own normal form and their directive behaviour during reification of inhabitants of $\mathcal{K}$ and reflection of neutrals into $\mathcal{K}$. Now sort $*$ is interpreted by $\overline{\overline{\diamond}}$ and $\Box$ (informally) by $\bigcup_{k \neq \diamond} \overline{\overline{k}}$.

Lemma 4. *For all k, $\overline{\overline{k}}$ is a PER over $\mathsf{D} \times \langle k \rangle$. If $K \Vdash \mathcal{K}$ then $(K, \mathcal{K}) \in \overline{\overline{k}}$.*

Lemma 5 (Interpretation of $*$). *We have $* \Vdash \overline{\overline{\diamond}} : \langle\langle * \rangle\rangle$ and $(*, \overline{\overline{\diamond}}) = (*, \overline{\overline{\diamond}}) \in \overline{\overline{*}}$.*

Function space construction. For simple kinds $k, l : \mathsf{SKi}$ we define

$$\begin{aligned}
&\textstyle\prod^{k,l} \quad : \ (\mathcal{K} \in \langle\langle k \rangle\rangle) \to (\mathcal{K} \to \langle\langle l \rangle\rangle) \to \langle\langle k \to l \rangle\rangle \\
&\textstyle\prod^{k,l} \mathcal{K}\, \mathcal{L} = \{((F, \mathcal{F}), (F', \mathcal{F})) : (\mathsf{D} \times \langle k \to l \rangle)^2 \mid \text{ for all } (G, \mathcal{G}) = (G', \mathcal{G}') \in \mathcal{K}, \\
&\qquad\qquad F \cdot G \downarrow, F' \cdot G' \downarrow, \mathcal{F}(G, \mathcal{G}) \downarrow, \mathcal{F}'(G', \mathcal{G}') \downarrow, \text{ and} \\
&\qquad\qquad (F \cdot G, \mathcal{F}(G, \mathcal{G})) = (F' \cdot G', \mathcal{F}'(G', \mathcal{G}')) \in \mathcal{L}(G, \mathcal{G})\}.
\end{aligned}$$

In case $l = \diamond$ both $\mathcal{F}, \mathcal{F}' : \langle k \to l \rangle = ()$ are trivial and we let the application $\mathcal{F}(G, \mathcal{G})$ be defined as $()$. Otherwise, $\mathcal{F}, \mathcal{F}' : \mathsf{D} \times \langle k \rangle \rightharpoonup \langle l \rangle$.

$\prod^{k,l} \mathcal{K}\, \mathcal{L}$ is indeed a PER, the proof is standard. The definition above is a bit abstract, the following table conveys some intuition about $\prod^{k,l}$.

k	l	description	PTS rule
$\diamond$	$\diamond$	dependent function space	$(*, *, *)$
not $\diamond$	$\diamond$	universal quantification	$(\Box, *, *)$
$\diamond$	not $\diamond$	indexed kind formation	$(*, \Box, \Box)$
not $\diamond$	not $\diamond$	function kind formation	$(\Box, \Box, \Box)$

Where is the impredicativity? For $k = *$ and $l = \diamond$ we should recover the impredicative quantification of System F. Let us look at the definition of this instance $\prod^{*,\diamond} : (\mathcal{K} \in \mathsf{Per}(\mathsf{D} \times \langle * \rangle)) \to (\mathcal{K} \to \langle * \rangle) \to \langle * \rangle$. Remember that $\langle * \rangle = \mathsf{Per}(\mathsf{D} \times ())$, thus, modulo the isomorphism $\mathsf{D} \times () \cong \mathsf{D}$ we get

$$\begin{aligned}
\textstyle\prod^{*,\diamond} \mathcal{K}\, \mathcal{L} = \{(F, F') : \mathsf{D}^2 \mid \forall G, G' : \mathsf{D}, \mathcal{G}, \mathcal{G}' : \langle * \rangle. \\
(G, \mathcal{G}) = (G', \mathcal{G}') \in \mathcal{K} \implies F \cdot G = F' \cdot G' \in \mathcal{L}(G, \mathcal{G})\}.
\end{aligned}$$

Hence, to obtain a new element $\prod \mathcal{K}\, \mathcal{L} : \langle * \rangle$ we quantify over all $\mathcal{G}, \mathcal{G}' : \langle * \rangle$ — there is System F impredicativity.

Lemma 6 (Function type formation, introduction, and elimination). *Let $\mathcal{K}, \mathcal{K}' : \langle\langle k \rangle\rangle$, $\mathcal{L}, \mathcal{L}' \in \mathcal{K} \to \langle\langle l \rangle\rangle$, and $\mathcal{F}, \mathcal{F}' \in \mathcal{K} \to \langle l \rangle$. The following inferences are valid in*

the model.

$$\frac{(K,\mathcal{K}) = (K',\mathcal{K}') \in \overline{\overline{k}} \qquad (L \cdot G, \mathcal{L}(G,\mathcal{G})) = (L' \cdot G', \mathcal{L}'(G',\mathcal{G}')) \in \bar{\bar{l}} \text{ for all } (G,\mathcal{G}) = (G',\mathcal{G}') \in \mathcal{K}}{(\underline{\Pi}\, K\, L, \prod \mathcal{K}\, \mathcal{L}) = (\underline{\Pi}\, K'\, L', \prod \mathcal{K}'\, \mathcal{L}') \in \overline{\overline{k \to l}}}$$

$$\frac{(F \cdot G, \mathcal{F}(G,\mathcal{G})) = (F' \cdot G', \mathcal{F}'(G',\mathcal{G}')) \in \mathcal{L}(G,\mathcal{G}) \text{ for all } (G,\mathcal{G}) = (G',\mathcal{G}') \in \mathcal{K}}{(F,\mathcal{F}) = (F',\mathcal{F}') \in \prod \mathcal{K}\, \mathcal{L}}$$

$$\frac{(F,\mathcal{F}) = (F',\mathcal{F}') \in \prod \mathcal{K}\, \mathcal{L} \qquad (G,\mathcal{G}) = (G',\mathcal{G}') \in \mathcal{K}}{(F \cdot G, \mathcal{F}(G,\mathcal{G})) = (F' \cdot G', \mathcal{F}'(G',\mathcal{G}')) \in \mathcal{L}(G,\mathcal{G})}$$

Proof. Introduction and elimination follow directly from the definition of $\prod \mathcal{K}\, \mathcal{L}$. The formation rule follows from properties of reflection and reification at function types [ACD07].

6 Soundness of the Model

We extend the raw semantic interpretation to shapes by setting $\langle \div, k \rangle = \langle k \rangle$ and $\langle \doteq, k \rangle = \langle\!\langle k \rangle\!\rangle$, and to simple kinding contexts via $\langle () \rangle = ()$ and $\langle \gamma, k \rangle = \langle \gamma \rangle \times \langle k \rangle$.

Interpretation. By induction on M/σ we simultaneously define the partial functions $[\![M]\!]_{\eta;\gamma;\_} : (\rho : \langle \gamma \rangle) \rightharpoonup \langle |M|_\gamma \rangle$ and $[\![\sigma]\!]_{\eta;\gamma;\_} : (\rho : \langle \gamma \rangle) \rightharpoonup \langle |\sigma|_\gamma \rangle$.

$$\begin{aligned}
[\![*]\!]_{\eta;\gamma;\rho} &= \overline{\overline{\diamond}} : \langle\!\langle * \rangle\!\rangle \\
[\![\Pi\, U\, T]\!]_{\eta;\gamma;\rho} &= \textstyle\prod^{|U|^{\doteq}, |T|^{\doteq}} [\![U]\!]_{\eta;\gamma;\rho}\, ((G,\mathcal{G}) \in [\![U]\!]_{\eta;\gamma;\rho} \mapsto [\![T]\!]_{\eta,G;\gamma,|U|^{\doteq};\rho,\mathcal{G}}) \\
[\![\mathsf{v}_0]\!]_{\eta;\gamma,k;\rho,\mathcal{G}} &= \mathcal{G} : \langle k \rangle \\
[\![\lambda\, U\, M]\!]_{\eta;\gamma;\rho} &= ((G,\mathcal{G}) \in [\![U]\!]_{\eta;\gamma;\rho} \mapsto [\![M]\!]_{\eta,G;\gamma,|U|^{\doteq};\rho,\mathcal{G}}) \\
[\![M\, N]\!]_{\eta;\gamma;\rho} &= [\![M]\!]_{\eta;\gamma;\rho}(\!(N)\!)_\eta, [\![N]\!]_{\eta;\gamma;\rho}) \\
[\![M\, \sigma]\!]_{\eta;\gamma;\rho} &= [\![M]\!]_{(\!(\sigma)\!)_\eta;|\sigma|_\gamma;[\![\sigma]\!]_{\eta;\gamma;\rho}}
\end{aligned}$$

$$\begin{aligned}
[\![\uparrow]\!]_{\eta;\gamma,k;\rho,o} &= \rho : \langle \gamma \rangle \\
[\![\mathsf{id}]\!]_{\eta;\gamma;\rho} &= \rho : \langle \gamma \rangle \\
[\![(\sigma, M)]\!]_{\eta;\gamma;\rho} &= [\![\sigma]\!]_{\eta;\gamma;\rho}, [\![M]\!]_{\eta;\gamma;\rho} : \langle |\sigma|_\gamma, |M|^{\div}_{\gamma} \rangle \\
[\![\sigma\, \tau]\!]_{\eta;\gamma;\rho} &= [\![\sigma]\!]_{(\!(\tau)\!)_\eta;|\tau|_\gamma;[\![\tau]\!]_{\eta;\gamma;\rho}} : \langle |\sigma|_{|\tau|_\gamma} \rangle
\end{aligned}$$

The prime source of partiality is the potential undefinedness of $(\!(N)\!)_\eta$ in the interpretation of the application $M\, N$. Contrast this to Barras and Werner [BW97] where $(\!(N)\!)_\eta$ is gone and with it the large eliminations.

The precise conditions for definedness can be extracted mechanically from this definition, for instance, $[\![\Pi\, U\, T]\!]_{\eta;\gamma;\rho}$ is defined iff $[\![U]\!]_{\eta;\gamma;\rho}$ is defined and for all $(G,\mathcal{G}) \in [\![U]\!]_{\eta;\gamma;\rho}$, $[\![T]\!]_{\eta,G;\gamma,|U|^{\doteq};\eta;\mathcal{G}}$ is defined.

Validity. We define the relation $(\eta;\rho) = (\eta';\rho') \in [\![\Gamma]\!]$ for $\eta, \eta' : \mathsf{Env}$ and $\rho, \rho' : \langle|\Gamma|\rangle$ inductively by the rules

$$\frac{}{(();()) = (();()) \in [\![()]\!]} \qquad \frac{(\eta;\rho) = (\eta';\rho') \in [\![\Gamma]\!] \quad (G,\mathcal{G}) = (G',\mathcal{G}') \in [\![T]\!]_{\eta;|\Gamma|;\rho}}{(\eta,G;\rho,\mathcal{G}) = (\eta,G';\rho,\mathcal{G}') \in [\![\Gamma,T]\!]}$$

There is an implicit premise $[\![T]\!]_{\eta;|\Gamma|;\rho} \downarrow$ in the second rule. By induction on Γ we simultaneously define the following propositions:

$$\begin{aligned}
() \models &\quad :\Longleftrightarrow \text{ true}\\
\Gamma, T \models &\quad :\Longleftrightarrow \Gamma \models T : \Box\\
\Gamma \models \kappa = \kappa' : \Box &\quad :\Longleftrightarrow \Gamma \models \text{ and for all } (\eta,\rho) = (\eta',\rho') \in [\![\Gamma]\!],\\
&\qquad\qquad (\!|\kappa|\!)_\eta, [\![\kappa]\!]_{\eta;|\Gamma|;\rho}) = ((\!|\kappa'|\!)_{\eta'}, [\![\kappa']\!]_{\eta';|\Gamma|;\rho'}) \in \overline{\overline{|\kappa|^{\doteq}}}\\
\Gamma \models M = M' : T \not\equiv \Box &\quad :\Longleftrightarrow \Gamma \models T : \Box \text{ and for all } (\eta,\rho) = (\eta',\rho') \in [\![\Gamma]\!],\\
&\qquad (\!|M|\!)_\eta, [\![M]\!]_{\eta;|\Gamma|;\rho}) = ((\!|M'|\!)_{\eta'}, [\![M']\!]_{\eta';|\Gamma|;\rho'}) \in [\![T]\!]_{\eta;|\Gamma|;\rho}\\
\Gamma \models M : T &\quad :\Longleftrightarrow \Gamma \models M = M : T\\
\Gamma \models \sigma = \sigma' : \Delta &\quad :\Longleftrightarrow \Gamma \models \text{ and } \Delta \models \text{ and for all } (\eta,\rho) = (\eta',\rho') \in [\![\Gamma]\!],\\
&\qquad\qquad (\!|\sigma|\!)_\eta, [\![\sigma]\!]_{\eta;|\Gamma|;\rho}) = ((\!|\sigma'|\!)_{\eta'}, [\![\sigma']\!]_{\eta';|\Gamma|;\rho'}) \in [\![\Delta]\!]\\
\Gamma \models \sigma : \Delta &\quad :\Longleftrightarrow \Gamma \models \sigma = \sigma : \Delta
\end{aligned}$$

Lemma 7. *If $\Gamma \models$ then $\_ = \_ \in [\![\Gamma]\!]$ is a PER.*

Theorem 2 (Fundamental theorem). *If $\Gamma \vdash J$ then $\Gamma \models J$.*

Proof. Simultaneously by for all judgements J by induction on $\Gamma \vdash J$.

Theorem 3 (Completeness of NbE). *If $\Gamma \vdash M = M' : T$ then $\mathsf{nbe}^T_\Gamma M \equiv \mathsf{nbe}^T_\Gamma M'$.*

Proof. Define ρ_Γ by the clauses $\rho_{()} = ()$ and $\rho_{\Gamma,U} = \rho_\Gamma, \bot^{|U|^{\doteq}}$ and prove that $(\eta_\Gamma, \rho_\Gamma) \in [\![\Gamma]\!]$ by induction on Γ. Let $(F,\mathcal{F}) = ((\!|M|\!)_{\eta_\Gamma}, [\![M]\!]_{\eta_\Gamma;|\Gamma|;\rho_\Gamma})$ and $(F',\mathcal{F}')$ analogously. If $T \not\equiv \Box$ then with $(K,\mathcal{K}) = ((\!|T|\!)_{\eta_\Gamma}, [\![T]\!]_{\eta_\Gamma;|\Gamma|;\rho_\Gamma})$ we obtain $(F,\mathcal{F}) = (F',\mathcal{F}') \in \mathcal{K}$ and $K \Vdash \mathcal{K}$ by the fundamental theorem. Hence, $\downarrow^K F = \downarrow^K F' \in \mathsf{Nf}$, so in particular $\mathsf{R}^{\mathsf{nf}}_{\|\Gamma\|}\,(\downarrow^K F) \equiv \mathsf{R}^{\mathsf{nf}}_{\|\Gamma\|}\,(\downarrow^K F')$. If $T \equiv \Box$ then the fundamental theorem yields $(F,\mathcal{F}) = (F',\mathcal{F}') \in \overline{\overline{k}}$ for $k = |M|^{\doteq}$ which also implies $\downarrow F = \downarrow F' \in \mathsf{Nf}$.

6.1 On Consistency

From our PER model we can prove the consistency of CoC as follows. Add a new type constant $\emptyset$, the empty type, with rules $\Gamma \vdash \emptyset : *$ and $\Gamma \vdash \emptyset = \emptyset : *$ and semantics $[\![\emptyset]\!]_{\eta;\gamma;\rho} = \mathsf{Ne}^{()}$.

Theorem 4 (Consistency). $\not\vdash t : \emptyset$.

Proof. Let $a = (\!|t|\!)_{\mathsf{id}}$ and observe that a cannot mention a de Bruijn level x_j. Since $a \in \mathsf{Ne}$ by the fundamental theorem (Thm .2), we have $0 \vdash^{\mathsf{ne}} a \searrow n$ for some n. But this means that a is of the shape $\mathsf{x}_j\, \boldsymbol{d}$ for some j, contradiction!

7 Conclusion

We have built a model for the Calculus of Constructions with typed $\beta\eta$-equality which proves termination and completeness of normalization by evaluation. The model supports extensions of the CoC by small inductive types ("small" meaning "in $*$") and large (aka strong) eliminations into $*$, i. e., types defined by recursion [Wer92]. The model is formalizable directly in impredicative dependent type theories with inductive definitions, e. g., in the Calculus of Inductive Constructions (CIC) [INR08].

This work is a first step towards a metatheory of CIC with typed $\beta\eta$-equality using normalization by evaluation. The long term goal is a correctness proof for the Coq type checker in the presence of η. However, a number of things need to be done:

- Construct a Kripke logical relation between expressions and values that proves the soundness of NbE without external reference to injectivity. As a consequence, we obtain soundness and completeness of the type checker of Section 3.
- Extend the calculus to more universes and to large inductive types. This will break simple kinding, requiring a new way to bootstrap the model. The usual technique are inaccessible cardinals [Luo89,Gog94,Miq00], however, we seek a more direct representation in type theory.

Acknowledgments. The author acknowledges the support of INRIA through a guest researcher fellowship. He would like to thank Hugo Herbelin and Pierre-Louis Curien for the invitation, and Bruno Barras and Benjamin Werner for discussions on the topic of this work. Their [Wer92,BW97] and Stefanova and Geuver's work [SG96] were a major help in understanding models for the CoC. Thanks also to the anonymous referees for their comments which helped to improve the quality of this paper.

References

[Abe10] Andreas Abel. Towards Normalization by Evaluation for the Calculus of Constructions (Extended Version). Available on the author's homepage, 2010.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *JFP*, 1(4):375–416, 1991.

[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.

[ACP09] Andreas Abel, Thierry Coquand, and Miguel Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. In *TLCA'09*, volume 5608 of *LNCS*, pages 5–19. Springer, 2009.

[Ada06] Robin Adams. Pure type systems with judgemental equality. *JFP*, 16(2):219–246, 2006.

[Bar09] Bruno Barras. Sets in Coq, Coq in sets. In *The 1st Coq Workshop, Proceedings*. Technische Universität München, 2009.

[BW97] Bruno Barras and Benjamin Werner. Coq in Coq. Available on the WWW, 1997.

[Car86] John Cartmell. Generalised algebraic theories and contextual categories. *APAL*, pages 32–209, 1986.

[CG90] Thierry Coquand and Jean Gallier. A proof of strong normalization for the theory of constructions using a kripke-like interpretation. In *Proceedings of the First Workshop on Logical Frameworks*, 1990.

[Cha09] James Chapman. *Type Checking and Normalization*. PhD thesis, School of Computer Science, University of Nottingham, 2009.

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

[Dan07] Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. In *TYPES'06*, volume 4502 of *LNCS*, pages 93–109. Springer, 2007.

[Geu94] Herman Geuvers. A short and flexible proof of strong normalization for the Calculus of Constructions. In *Types for Proofs and Programs, Int. Workshop TYPES '94*, volume 996 of *LNCS*, pages 14–38, Båstad, Sweden, 1994. Springer.

[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.

[GN91] Herman Geuvers and Mark-Jan Nederhof. Modular proof of strong normalization for the calculus of constructions. *JFP*, 1(2):155–189, 1991.

[Gog94] Healfdene Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, 1994. Available as LFCS Report ECS-LFCS-94-304.

[Gon04] Georges Gonthier. A computer-checked proof of the four colour theorem. Technical report, Microsoft Research, 2004. Available at http://research.microsoft.com/˜gonthier/.

[Gra09] Johan Granström. *Reference and Computation in Intuitionistic Type Theory*. PhD thesis, Mathematical Logic, Uppsala University, 2009.

[Hue89] Gérard Huet. The constructive engine. In *2nd European Symposium on Programming, Nancy, March 1988*, 1989. Final version in anniversary volume Theoretical Computer Science in memory of Gift Siromoney, Ed. R. Narasimhan, World Scientific Publishing, 1989.

[INR08] INRIA. *The Coq Proof Assistant Reference Manual*. INRIA, version 8.2 edition, 2008. http://coq.inria.fr/.

[Ler06] Xavier Leroy. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *POPL'06*, pages 42–54. ACM, 2006.

[Luo89] Zhaohui Luo. ECC, an Extended Calculus of Constructions. In *LICS'89*, pages 386–395. IEEE CS Press, 1989.

[Miq00] Alexandre Miquel. A model for impredicative type systems, universes, intersection types and subtyping. In *LICS'00*, pages 18–29, 2000.

[MW03] Alexandre Miquel and Benjamin Werner. The not so simple proof-irrelevant model of CC. In *TYPES'02*, volume 2646 of *LNCS*, pages 240–258. Springer, 2003.

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

[Pol06] Randy Pollack. The constructive engine. Talk presented at the TYPES Workshop Curry-Howard Implementation Techniques - Connecting Humans And Theorem provers, CHIT-CHAT 2006, Radboud University, Nijmegen, The Netherlands, 2006.

[SG96] Milena Stefanova and Herman Geuvers. A simple model construction for the calculus of constructions. In *TYPES'95*, volume 1158 of *LNCS*, pages 249–264. Springer, 1996.

[Wer92] Benjamin Werner. A normalization proof for an impredicative type system with large eliminations over integers. In *TYPES'92*, pages 341–357, 1992.