PROOF-IRRELEVANT MODEL OF CC WITH PREDICATIVE INDUCTION AND JUDGMENTAL EQUALITY

GYESIK LEE\textsuperscript{a} AND BENJAMIN WERNER\textsuperscript{b}

\textsuperscript{a} Hankyong National University, Anseong-si, Kyonggi-do, Korea
e-mail address: gslee@hknu.ac.kr
\textsuperscript{b} INRIA Saclay and LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, France
e-mail address: Benjamin.Werner@inria.fr

Abstract. We present a set-theoretic, proof-irrelevant model for Calculus of Constructions (CC) with predicative induction and judgmental equality in Zermelo-Fraenkel set theory with an axiom for countably many inaccessible cardinals. We use Aczel's trace encoding which is universally defined for any function type, regardless of being impredicative. Direct and concrete interpretations of simultaneous induction and mutually recursive functions are also provided by extending Dybjer's interpretations on the basis of Aczel's rule sets. Our model can be regarded as a higher-order generalization of the truth-table methods. We provide a relatively simple consistency proof of type theory, which can be used as the basis for a theorem prover.

1. Introduction

Informal motivation. The \textit{types-as-sets} interpretation of type theory in a sufficiently strong classical axiomatic set theory, such as the Zermelo-Fraenkel (ZF) set theory, has been regarded as the most straightforward approach to demonstrating the consistency of type theory (cf. \cite{Aczel1998} and \cite{Coquand1990}). It can be construed as a higher-order generalization of the truth-table methods. Such a model captures the intuitive meaning of the constructs: the product, $\lambda$-abstraction, and application correspond to the ordinary set-theoretic product, function, and application, respectively.

A straightforward model of type theory is very useful for establishing the consistency of type theory, and it can be used to determine the proof-theoretic strength of type theory (cf. \cite{Aczel1998,Dybjer1991,Dybjer2000, Werner1997}). However, a higher-order generalization of the trivial Boolean model is \textit{not so simple} (cf. \cite{Miquel2003}). The main cause of this problem, as identified by \cite{Reynolds1984}, is the fact that type systems...
containing Girard-Reynolds’ second-order calculus cannot have the usual set-theoretic interpretation of types. The only way to provide a set-theoretic meaning for an impredicative proposition type is to identify all the proof terms of that proposition type: Proposition types are interpreted either by the empty set or a singleton with a canonical element. Thus, proof-irrelevant models are necessary for interpreting reasonable higher-order type systems.

Set-theoretic models of type theory can be understood in a straightforward manner. Werner(2008) showed that they can be used as the basis of proof assistants in programming with dependent types. This is because they provide a mechanism to distinguish between computational and logical parts. Werner’s system is a proof-irrelevant version of Luo’s Extended Calculus of Constructions (ECC; Luo(1989)), and the set-theoretic model is an extension of that of Calculus of Constructions (CC) defined by Miquel and Werner(2003).

Luo’s ECC is a Martin-Löf-style extension of CC, with strong sum types and a fully cumulative type hierarchy. At the lowest level, there is an impredicative type Prop of propositions. This is followed by a hierarchy of predicative type universes Typeᵢ, i = 0, 1, 2, . . . :

- Prop is of type Type₀;
- Typeᵢ is of type Typeᵢ₊₁;
- Prop ⊊ Type₀ ⊊ Type₁ ⊊ . . . .

Werner’s system, however, does not include the subtyping rule Prop ⊊ Type₀, which could complicate the model construction, as identified by Miquel and Werner(2003). Their model constructions cannot be extended to ECC. We will explain this in detail in Remark 3.1.

In this paper, we investigate the inclusion of Prop ⊊ Type₀, and we show that type theory with judgmental equality, à la Martin-Löf(1984), can have a simple proof-irrelevant model. We expect our results to play a key role in the theoretical justification of proof systems based on Martin-Löf-style type theory.

Overview of the work. Martin-Löf type theory and Logical Framework include typing rules for the equality of objects and types:

\[
\Gamma \vdash M = N : A \quad \text{and} \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B}{\Gamma \vdash M : B} \quad (\text{conv})
\]

In particular, Barendregt’s PTS-style β-conversion side condition turns into an explicit judgment. Two objects are not just equal; they are equal with respect to a type (cf. Nordström et al.(1990) Nordström, Petersson, and Smith [Goguen(1994), Aczel(1998)]).

The type system considered in our study is CC with predicative induction and judgmental equality. It is a type system with the following features: dependent types, impredicative type (Prop) of propositions, a cumulative hierarchy of predicative universes (Typeᵢ), predicative inductions, and judgmental equality.

The main difficulty in the construction of a set-theoretic model of our system stems from the impredicativity of Prop and the subtyping property Prop ⊊ Type₀. Without subtyping, one could use the solution provided by Miquel and Werner(2003) and Werner(2008), whereby proof-terms are syntactically distinguished from other function terms. Thus, the problem lies in the case distinction between the impredicative type Prop and the predicative types Typeᵢ, whereas the subsumption eliminates the difference. An interpretation function \( f : \{0, 1\} \to V \) is required, where \( V \) is a set universe, that is different from the identity function. See Section 3 for further details.
For a set-theoretic interpretation of the cumulative type universes and predicative inductions, it is sufficient to assume countably many (strongly) inaccessible cardinals. [Werner(1997)] showed that ZF with an axiom guaranteeing the existence of infinitely many inaccessible cardinals is a good candidate. However, it is not clear whether the inaccessible cardinal axiom is necessary for our construction. The required feature of an inaccessible cardinal $\kappa$ is the closure property of the universe $\mathcal{V}_\kappa$ under the powerset operation. This is a necessary condition for the interpretation of inductive types. Following [Dybjer(1991)], we use Aczel’s rule sets to obtain a direct and concrete interpretation of induction and recursion rules.

The remainder of this paper is organized as follows. In Section 2 we provide a formal presentation of CC with predicative induction and judgmental equality. Examples are presented to enable the reader to understand the syntax and typing rules. This section can be regarded as an introduction to the base theory of the proof assistant Coq. Indeed, the syntax we have provided is as close to Coq syntax as that used in practice, except for the judgmental equality and the restriction on predicative inductions.

The difficulties in providing set-theoretic interpretations of impredicative or polymorphic types, subtypes, etc., are discussed in Section 3. We use the computational information about the domains saved in the interpretation of $a : A$ to avoid these difficulties. This means that for the construction of set-theoretic models, type systems with judgmental equality are more explicit than systems without it. Using some typical examples, we explain the construction of a set-theoretic interpretation of inductive types and recursive functions.

Finally, in Section 4 we prove the soundness of our interpretation. The proof itself is relatively simple, and it can also be used to verify the consistency of our system. This is because some types such as $\Pi(\alpha : \text{Prop}).\alpha$ will be interpreted as the empty set; hence, they cannot be inhabited in the type system.

In Section 5 we summarize the main results, and we discuss related work for future investigation.

2. Formal presentation of CC with judgmental equality

First, we provide the full presentation of the system, i.e., Coquand’s CC with judgmental equality and predicative induction over infinitely many cumulative universes.

2.1. Syntax. We assume an infinite set of countably many variables, and we let $x, x_i, X, X_i, \ldots$ vary over the variables. We also use special constants $\text{Prop}$ and $\text{Type}_i$, $i \in \mathbb{N}$. They are called sorts. Sorts are usually denoted by $s, s_i, \text{etc.}$

1We remark that many impredicative inductive types can be coded by impredicative definitions (cf. Girard et al.(1989) Girard, Taylor, and Lafont, Coquand(1990) Werner(1997)).

2In this paper, we do not consider the sort $\text{Set}$. Indeed, when (the impredicative or predicative sort) $\text{Set}$ is placed at the lowest level in the hierarchy of sorts, as in the case of the current development of Coq, there is no way to provide a universal set-theoretic interpretation of both $\text{Set}$ and $\text{Prop}$, as identified by Reynolds(1984). Note, however, that $\text{Type}_0$ in our system plays the role of the predicative $\text{Set}$. 
Definition 2.1 (Terms and contexts). The syntax of the objects is given as follows.
\[
\begin{align*}
\text{terms} &::= \text{var} \mid \text{seq} \mid \lambda \text{var} : \text{term} | \text{let} \text{var} := \text{term} | \text{fix} \text{var} : \text{term} \\
\text{declarations} &::= \emptyset | (\text{var} := \text{term}) \\
\text{contexts} &::= \emptyset | (\text{var} : \text{term}) | (\text{var} := \text{term}) | (\text{var} : \text{term})
\end{align*}
\]

(terms)
\[
\begin{align*}
\text{var} &::= \text{var} \mid \text{seq} \\
\text{seq} &::= \text{seq} \mid \lambda \text{var} : \text{term} | \text{let} \text{var} := \text{term} | \text{fix} \text{var} : \text{term}
\end{align*}
\]

(declarations)
\[
\begin{align*}
\text{var} &::= \text{var} \mid \text{seq} \\
\text{seq} &::= \text{seq} \mid \lambda \text{var} : \text{term} | \text{let} \text{var} := \text{term}
\end{align*}
\]

(contexts)

Remark 2.4.

(1) Vector notations are used instead of some sequences of expressions:
\[
\begin{align*}
\vec{t}, \vec{t}' &::= t, t' \\
\vec{a}, \vec{a}' &::= a, a'
\end{align*}
\]

(2) Note that we use two subscript styles. One is of the form \( \text{var} : \text{term} \) and the other is of the form \( \text{var} \), where \( \text{var} \) is a natural number. The latter style will be used only in the definition of mutually recursive functions, i.e., in combination with \( \text{fix} \).

(3) Given a sequence \( \vec{t} \), let \( \text{lh}(\vec{t}) \) denote its length.

(4) In the examples presented below, character strings are used instead of single character variables in order to emphasize the correspondence with real Coq-expressions.

(5) Given a declaration \( \Delta \) and a variable \( \text{var} \), let \( \Delta(\text{var}) = \text{term} \) when \( \text{var} \) is the only term such that \( \text{var} : \text{term} \) occurs in \( \Delta \).

(6) There are standard definitions of the sets of free variables in a context or a term, and of the substitution \( t[\text{var} \setminus \text{term}] \), where \( t \), \( \text{var} \), and \( \text{term} \) are terms and \( \text{var} \) a variable. Formal definitions are given in Appendix A.

(7) Given a sequence \( \delta \), let \( \text{fix}(\delta) := (x_1 \setminus t_1) \cdots (x_n \setminus t_n) \)

denote consecutive substitution. On the other hand, the simultaneous substitution of terms \( t_1, \ldots, t_n \) for \( x_1, \ldots, x_n \), respectively, in \( t \) is denoted by \( t[\delta] := t[x_1 \setminus t_1 \ldots x_n \setminus t_n] \).

To enable the reader to understand the intended meaning of terms and contexts, we explain some notations with examples. The examples will also be used in Section 3 to explain our model.

Remark 2.5. The expression \( \text{Ind}_n(\Delta_I := \Delta_C) \) denotes a (mutually) inductive type, and the subscript \( n \) denotes the number of parameters. \( \Delta_I \) and \( \Delta_C \) are two declarations containing inductive types and their constructors, respectively. The Parameters are binders shared by all the constructors of the definition, and they are used to construct polymorphic types. The
parameters differ from other non-parametric binders in that the conclusion of each type of constructor invokes the inductive type with the same parameter values as its specification. We refer to Lemma 3.8 and Lemma 3.9 which show the difference between parameters and non-parametric binders.

The mutual definition of trees and forests can be represented, for instance, by

$$D_{TF} = \text{Ind}_1 \{ \Delta_I := \Delta_C \}$$

where

$$\Delta_I = \text{tree} : \text{Type}_0 \rightarrow \text{Type}_0, \quad \text{forest} : \text{Type}_0 \rightarrow \text{Type}_0,$$

$$\Delta_C = \text{node} : \Pi(A : \text{Type}_0). A \rightarrow \text{forest} A \rightarrow \text{tree} A,$$

$$\text{emptyf} : \Pi(A : \text{Type}_0). \text{forest} A,$$

$$\text{consf} : \Pi(A : \text{Type}_0). \text{tree} A \rightarrow \text{forest} A \rightarrow \text{forest} A.$$ 

The subscript 1 implies that \((A : \text{Type}_0)\) is a parameter.

**Remark 2.6.** If \(D = \text{Ind}_n \{ \Delta_I := \Delta_C \}\) and \(x \in \text{dom}(\Delta_I, \Delta_C)\), then \(D \cdot x\) corresponds to the names of defined inductive types or their constructors.

The type for natural numbers and its two constructors can be represented by \(D_N \cdot \text{nat}, D_N \cdot 0,\) and \(D_N \cdot \text{S},\) respectively, where \(D_N = \text{Ind}_0 \{ \Delta_I := \Delta_C \}\), \(\Delta_I = \text{nat} : \text{Type}_0,\) and \(\Delta_C = 0 : \text{nat}, \text{S} : \text{nat} \rightarrow \text{nat}.$$ 

In the examples presented below, however, we use character strings for better readability. Thus, for example, \(\text{nat}, 0,\) and \(\text{S}\) are used instead of \(D_N \cdot \text{nat}, D_N \cdot 0,\) and \(D_N \cdot \text{S},\) respectively.

**Remark 2.7 (case and fix).** The term \(\text{case}(e, Q, \vec{h})\) corresponds to the following Coq-expression

\[
\text{match } e \text{ as } y \text{ in } I \vec{p} \vec{u} \text{ return } Q \vec{u} y \text{ with } \cdots | C_i \vec{p} \vec{v} \Rightarrow h_i | \cdots \text{ end}
\]

where

- the term \(e\) is of an inductive type \(I \vec{p} \vec{u}\) for some terms \(\vec{p}, \vec{u},\)
- \(lh(\vec{p}) = lh(\vec{u}),\)
- \(Q = \lambda \vec{u}: \vec{U}. \lambda y: I \vec{p} \vec{u}. Q',\) for some terms \(\vec{U}, Q',\)
- the term \(y\) is a fresh variable bound in \(Q \vec{u} y,\)
- each \(C_i\) is a constructor of type \(\Pi \vec{p}: \vec{P}, \Pi \vec{u}: \vec{V}, I \vec{p} \vec{u} \vec{v} \vec{i}\) for some terms \(\vec{P}, \vec{V}, \vec{w}, \vec{i},\) and
- each \(h_i = \lambda \vec{v}: \vec{V}. h'_i\) for some term \(h'_i.\)

The term \(\text{fix } f_i \{ f/k : A := t \}\) denotes the \((i + 1)\)th function defined by a mutual recursion. The number \(k_i\) denotes the position of the inductive binder on which recursion is performed for \(f_i.\) It corresponds to Coq’s \textit{struct} annotation used for the “guarded” condition in the termination check (cf. [Giménez(1995)]).

(1) The addition function \(\text{plus}\) can be defined as follows:

\[
\text{plus} = \text{fix } f \{ f/1 : \Pi(m, n : \text{nat}). \text{nat} := \lambda(m, n : \text{nat}). \text{case}(n, Q, h_0, h_1)\},
\]

where \(Q = \lambda(\ell : \text{nat}). \text{nat}, h_0 = m,\) and \(h_1 = \lambda(p : \text{nat}). \text{S} (f m p).\)
(2) The functions for measuring the size of trees and forests can be represented by \( \text{Ts} = \text{fix} \ g_0 \{ R \} \) and \( \text{F} = \text{fix} \ g_1 \{ R \} \), where \( R = g/k : B := t \), \( k_0 = k_1 = 1 \), and

\[
\begin{align*}
B_0 & = \Pi(A : \text{Type}_0). \Pi(t : \text{tree} A). \text{nat}, \\
B_1 & = \Pi(A : \text{Type}_0). \Pi(f : \text{forest} A). \text{nat}, \\
t_0 & = \lambda(A : \text{Type}_0). \lambda(t : \text{tree} A). \text{case}(t, Q_0, h_0), \\
t_1 & = \lambda(A : \text{Type}_0). \lambda(f : \text{forest} A). \text{case}(f, Q_1, h_1, h_2), \\
Q_0 & = \lambda(t : \text{tree} A). \text{nat}, \\
Q_1 & = \lambda(f : \text{forest} A). \text{nat}, \\
h_0 & = \lambda(a : A). \lambda(f : \text{forest} A). \text{S}(g_1 A f), \\
h_1 & = 0, \\
h_2 & = \lambda(t : \text{tree} A). \lambda(f : \text{forest} A). \text{plus}(g_0 A t)(g_1 A f).
\end{align*}
\]

2.2. Typing rules. The typing judgment \( \Gamma \vdash M : A \) or \( \Gamma \vdash M = N : A \) is defined simultaneously with the property \( W_F(\Gamma) \) of a well-formed valid context and the property \( \Gamma \vdash M \prec N \) of cumulativity of types in Figures 1–4. We provide short explanations of some rules. For a more detailed explanation, refer to [Bertot and Castérán(2004)], [Letouzey(2004)], or [Paulin-Mohring(1996)].

Typing rules for basic terms and valid contexts (Figure 1). Typing rules for standard constructions of \( \lambda \)- and \( \Pi \)-terms are given.

\( \text{(wf)} \): Well-formed contexts contain well-typed terms, and they can be extended by well-typed inductive types, as in rule \( \text{(ind-wf)} \) of Figure 2

(\( \Pi \)) and \( \text{(II-eq)} \): \( P(s_1, s_2, s_3) \) implies that

- \( s_2 = s_3 = \text{Prop} \), or
- \( s_1 \in \text{Type}_i \), \( s_2 = \text{Type}_j \) and \( s_3 = \text{Type}_k \) where \( k \geq \max\{i, j\} \).

Typing rules for inductive types and recursive functions (Figure 2). Typing rules for (mutually) inductive types, case distinctions, and (mutually) recursive functions are given.

\( \text{(ind-wf)} \): The positivity condition is crucial for defining an inductive type. A term \( A \) is an arity ending in sort \( s \), \( \text{Arity}(A, s) \), if it is convertible to \( s \) or a product \( \Pi x : A.B \), where \( B \) is an arity ending in sort \( s \). \( A \) is called an arity, \( \text{Arity}(A) \), if \( A \) is an arity ending in sort \( s \) for some sort \( s \).

A term \( M \) satisfies the positivity condition for a variable \( x \) when \( M = \Pi \bar{y} : \bar{A}. x \bar{u} \) for some terms \( \bar{A}, \bar{u} \) and the variable \( x \) occurs strictly positively in \( \bar{A} \). A variable \( x \) occurs strictly positively in \( M \) when

- \( x \) does not occur in \( M \), or
- \( M \equiv \Pi \bar{y} : \bar{A}. (x \bar{B}) \) and \( x \) does not occur in \( \bar{A}, \bar{B} \).

Now, \( I_n(\Delta_I, \Delta_C) \) represents the following conditions:

- All the names contained in the domains of \( \Delta_I \) and \( \Delta_C \) must be mutually distinct and new.
For all $T$, we use $\Gamma \vdash \text{Prop} : \text{Type}_i$ for further details. 

\[
\begin{array}{c}
\text{WF}(\emptyset) \\
\frac{\Gamma \vdash A : s \quad x \notin \text{dom}(\Gamma)}{\text{WF}(\Gamma, (x : A))} \quad \frac{\Gamma \vdash t : A \quad x \notin \text{dom}(\Gamma)}{\text{WF}(\Gamma, (x := t : A))} \\
\end{array}
\]

\[
\begin{array}{c}
\text{WF}(\Gamma) \\
\frac{\Delta \vdash \text{Prop} : \text{Type}_i}{\Gamma \vdash \text{Prop} : \text{Type}_i} \quad \frac{\Gamma \vdash \Delta : \text{Type}_i}{\Gamma \vdash \Delta : \text{Type}_j} \\
\quad \frac{\Gamma \vdash (x : A) \in \Gamma \quad \text{or} \quad (x := t : A) \in \Gamma}{\Gamma \vdash x : A} \\
\quad \frac{\Gamma \vdash t = t' : A \quad \Gamma, (x := t : A) \vdash u = u' : U}{\Gamma \vdash (\text{let } x := t \text{ in } u) = (\text{let } x := t' \text{ in } u') : U[x \backslash t]} \\
\quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \mathcal{P}(s_1, s_2, s_3)}{\Gamma \vdash \Pi x : A.B : s_3} \\
\quad \frac{\Gamma \vdash A = A' : s_1 \quad \Gamma, x : A \vdash B = B' : s_2 \quad \mathcal{P}(s_1, s_2, s_3)}{\Gamma \vdash \Pi x : A.B : s_3} \\
\quad \frac{\Gamma \vdash \lambda x : A.M : \Pi x : A.B}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B} \\
\quad \frac{\Gamma \vdash A = A' : s \quad \Gamma, x : A \vdash M = M' : B \quad \Gamma \vdash \Pi x : A.B : s'}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B} \\
\end{array}
\]

Figure 1: Basic terms and valid contexts

- All the types of $\Delta_I$ and $\Delta_C$ start with the same $n$ products, say, $\overline{p} : \overline{P}$.
- Any occurrence of some $d \in \text{dom}(\Delta_I)$ in $\Delta_C$ is of the form $(\overline{d} \overline{p} \overline{u})$, which is not applicable any more.
- For all $d : A \in \Delta_I$, $A$ is an arity ending in sort $s_d$ such that $s_d \neq \text{Prop}$. Thus, we do not use inductive definitions of type $\text{Prop}$. Some propositions defined inductively can be constructed using an impredicative coding. See [Girard et al. (1989) Girard, Taylor, and Lafont, Coquand(1990)], and [Werner(1997)] for further details.
- For all $c : T \in \Delta_C$, $T$ is the type of a constructor for an inductive type $d \in \text{dom}(\Delta_I)$, i.e., $T$ is of the form $\Pi \overline{p} : \overline{P}, \Pi \overline{x} : \overline{Z}. (d \overline{p} \overline{u})$. In this case, the sort $s_c$ in the third premise of the rule must be $s_d$.
- $T$ satisfies the positivity condition for all $x \in \text{dom}(\Delta_I)$.

**Notation.** We use $\Gamma \vdash \text{Ind}_d(\Delta_I := \Delta_C)$ when all the premises of (ind-wf) are satisfied.
\[
\begin{align*}
I_\nu(\Delta_I, \Delta_C) \\
\Gamma \vdash A : s_d \quad \text{for all} \quad (d : A) \in \Delta_I \\
\Gamma, \Delta_I \vdash T : s_c \quad \text{for all} \quad (c : T) \in \Delta_C \\
\mathcal{W} \mathcal{F}(\Gamma, \text{ind}_{\nu}(\Delta_I := \Delta_C))
\end{align*}
\]

\[
\mathcal{W} \mathcal{F}(\Gamma) \quad \mathcal{D} = \text{ind}_{\nu}(\Delta_I := \Delta_C) \in \Gamma \quad d \in \text{dom}(\Delta_I)
\]

\[
\mathcal{W} \mathcal{F}(\Gamma) \quad \mathcal{D} = \text{ind}_{\nu}(\Delta_I := \Delta_C) \in \Gamma \quad c \in \text{dom}(\Delta_C)
\]

\[
\text{Ind}_{\nu}(\Delta_I := \Delta_C) \in \Gamma \quad (d_i : \Pi \tilde{p} : \tilde{P}. A) \in \Delta_I \quad \text{lh}(\tilde{p}) = n \\
\Gamma \vdash Q : B \quad C(d_i \tilde{p} : A; B) \quad \Gamma \vdash e : d_i \tilde{p} \tilde{u} \\
\Gamma \vdash h_k : \Pi \tilde{v} : \tilde{V}_k. Q \tilde{w}_k(c_k \tilde{p} \tilde{v}) \quad \text{for all} \quad (c_k : \Pi \tilde{p} : \tilde{P}. \Pi \tilde{v} : \tilde{V}_k. d_i \tilde{p} \tilde{w}_k) \in \Delta_C
\]

\[
\Gamma \vdash \text{case}(e, Q, (h_k)_k) : Q \tilde{u} \tilde{e}
\]

\[
\text{Ind}_{\nu}(\Delta_I := \Delta_C) \in \Gamma \quad (d_i : \Pi \tilde{p} : \tilde{P}. A) \in \Delta_I \quad \text{lh}(\tilde{p}) = n \\
\Gamma \vdash Q = Q' : B \quad C(d_i \tilde{p} : A; B) \quad \Gamma \vdash e = e' : d_i \tilde{p} \tilde{u} \\
\Gamma \vdash h_k = h'_k : \Pi \tilde{v} : \tilde{V}_k. Q \tilde{w}_k(c_k \tilde{p} \tilde{v}) \quad \text{for all} \quad (c_k : \Pi \tilde{p} : \tilde{P}. \Pi \tilde{v} : \tilde{V}_k. d_i \tilde{p} \tilde{w}_k) \in \Delta_C
\]

\[
\Gamma \vdash \text{case}(e, Q, (h_k)_k) = \text{case}(e', Q', (h'_k)_k) : Q \tilde{u} \tilde{e}
\]

\[
\mathcal{F}(\tilde{f}, \tilde{A}, \tilde{k}, \tilde{l}) \quad n = \text{lh}(\tilde{k}) \quad (\Gamma \vdash A_i : s_i)_{\forall i \leq n} \\
(\Gamma, \tilde{f} : \tilde{A} \vdash t_i : A_i)_{\forall i \leq n} \quad j \leq n
\]

\[
\Gamma \vdash \text{fix} \; f_j \{f/k : A := t_i\} : A_j
\]

\[
\mathcal{F}(\tilde{f}, \tilde{A}, \tilde{k}, \tilde{l}) \quad \mathcal{F}(\tilde{f}, \tilde{A}', \tilde{k}, \tilde{l'}) \quad n = \text{lh}(\tilde{k}) \\
(\Gamma \vdash A_i = A'_i : s_i)_{\forall i \leq n} \\
(\Gamma, \tilde{f} : \tilde{A} \vdash t_i = t'_i : A_i)_{\forall i \leq n} \quad j \leq n
\]

\[
\Gamma \vdash \text{fix} \; f_j \{f/k : A := t\} = \text{fix} \; f_j \{f/k : A' := t'\} : A_j
\]

Figure 2: Inductive types and recursive functions

\((ind\text{-}type)\) and \((ind\text{-}const)\): Given \( \mathcal{D} = \text{ind}_{\nu}(\Delta_I := \Delta_C) \) and a term \( A, A[\\setminus \mathcal{D}] \) implies that every occurrence of \( z \in \text{dom}(\Delta_I, \Delta_C) \) in \( A \) is replaced with \( \mathcal{D} \cdot z \).

\((case)\) and \((case\text{-}eq)\): \( d_i \) and \( c_k \) denote \( \text{ind}_{\nu}(\Delta_I := \Delta_C) \cdot d_i \) and \( \text{ind}_{\nu}(\Delta_I := \Delta_C) \cdot c_k \), respectively. Furthermore, \( \text{lh}(\tilde{u}) = \text{lh}(\tilde{w}_k) \).

For an inductive type \( d \) and an arity \( B \), the relation \( C(d \tilde{q} : A; B) \) is defined as follows:

- \( C(d \tilde{q} : \text{Prop}; d \tilde{q} \rightarrow \text{Prop}) \);
- \( C(d \tilde{q} : \text{Prop}; d \tilde{q} \rightarrow \text{Type}_j) \) if \( d \) is an inductive type that is empty or has only one constructor\(^3\) such that all the non-parametric arguments are of sort \( \text{Prop} \);
- \( C(d \tilde{q} : \text{Type}_j; d \tilde{q} \rightarrow s) \) for any sort \( s \);
- \( C(d \tilde{q} : (\Pi u : U. A); (\Pi u : U. B)) \) if \( C(d \tilde{q} u : A; B) \).

\(^3\)This reflects the fact that no pattern matching is allowed on proof-terms, which would otherwise result in a paradox, as shown by [Coquand(1990)].
\[
\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A} \quad \frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A} \quad \text{(ref)(sym)}
\]

\[
\frac{\Gamma \vdash M = N : A \quad \Gamma \vdash N = P : A}{\Gamma \vdash M = P : A} \quad \text{(trans)}
\]

\[
\frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B : s}{\Gamma \vdash M : B} \quad \frac{\Gamma \vdash M = N : A \quad \Gamma \vdash A = B : s}{\Gamma \vdash M = N : B} \quad \text{(conv)(conv-eq)}
\]

\[
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.M)N = M[x\backslash N] : B[x\backslash N]} \quad \text{(\(\beta\))}\n\]

\[
\frac{\forall \mathcal{F}(\Gamma) \quad (x := t : A) \in \Gamma}{\Gamma \vdash x = t : A} \quad \text{(\(\delta\))}
\]

\[
\frac{\Gamma \vdash \text{let } x := t \text{ in } u : U}{\Gamma \vdash (\text{let } x := t \text{ in } u) = u[x\backslash t] : U[x\backslash t]} \quad \text{(\(\zeta\))}
\]

\[
\text{Ind}_{\Delta_f := \Delta_C}(\{\Delta_i : \Pi \bar{p} : \bar{P}, \bar{A}) \in \Delta_f \quad \text{lh}(\bar{p}) = n \quad \Gamma \vdash Q : B \quad \mathcal{C}(d_i, \bar{p} : A; B) \quad \Gamma \vdash c_j \bar{p} \bar{a} : d_i \bar{p} \bar{u}} \quad \Gamma \vdash h_k : \Pi \bar{v} : \bar{V}_k. Q \bar{w}_k (c_k \bar{p} \bar{v}) \text{ for all } (c_k : \Pi \bar{p} : \bar{P}, \Pi \bar{v} : \bar{V}_k, d_i \bar{p} \bar{v}_k) \in \Delta_C \quad \Gamma \vdash \text{case}(c_j \bar{p} \bar{a}, Q,(h_k)_k) = h_j \bar{a} : Q \bar{u} (c_j \bar{p} \bar{a})}
\]

\[
\frac{\Gamma \vdash A_i : s_i \forall i \leq n \quad (\Gamma, \bar{f} : \bar{A} \vdash t_i : A_i) \forall i \leq n \quad \mathcal{F}(\bar{f}, \bar{A}, \bar{k}, \bar{r}) \quad j \leq n \quad R \equiv \bar{f}/k : A := \bar{t} \quad A_j \equiv \Pi \bar{x}_j : \bar{B}_j, A_j' \quad \Gamma \vdash \bar{a} : \bar{B}_j \quad \text{lh}(\bar{B}_j) = k_j + 1}{\Gamma \vdash (\text{fix } f_i \{R\}) \bar{a} = (t_j[\bar{f}_i(\text{fix } f_i \{R\})]] \bar{a} : A_j'\{\bar{x} : \bar{a}\}}
\]

\[
\vdash \text{Prop} \prec \text{Type}_0 \quad \vdash \text{Type}_j \prec \text{Type}_{j+1} \quad \text{(inc)}
\]

\[
\frac{\Gamma \vdash M \prec N \quad \Gamma \vdash N < P}{\Gamma \vdash M < P} \quad \text{(trans-inc)}
\]

\[
\frac{\Gamma \vdash A \prec B \quad \Gamma \vdash C : s \quad x \notin \text{dom}(\Gamma)}{\Gamma, x : C \vdash A < B} \quad \text{(weak-inc)}
\]

\[
\frac{\Gamma \vdash A_1 = B_1 : s \quad \Gamma, x : A_1 \vdash A_2 < B_2}{\Gamma \vdash \Pi x : A_1.A_2 \prec \Pi x : B_1.B_2} \quad \text{(\(\Pi\)-inc)}
\]

\[
\frac{\Gamma \vdash M = N : s \quad \Gamma \vdash M < P \quad \Gamma \vdash M = N : s \quad \Gamma \vdash P < M \quad \Gamma \vdash P < N}{\Gamma \vdash N < P \quad \Gamma \vdash M = N : s \quad \Gamma \vdash P < M} \quad \text{(eq-inc)}
\]

\[
\frac{\Gamma \vdash M : A \quad \Gamma \vdash A < B \quad \Gamma \vdash M : N : A \quad \Gamma \vdash A < B}{\Gamma \vdash M = N : B} \quad \text{(cum)(cum-eq)}
\]

Figure 3: Judgmental equality and cumulative type universes.
This means that an object of the inductive type $d$ can be eliminated for proving a property $P$ of type $B$. Let $C(d \bar{q}; B)$ denote $C(d \bar{q}: A; B)$, where $A$ is the type of $d \bar{q}$.

\[ (\text{fix}) \quad \text{and} \quad (\text{fix-eq}): F(\bar{f}, A, \bar{k}, \bar{t}) \]

represents the following conditions:

- $lh(\bar{f}) = lh(A) = lh(\bar{k}) = lh(\bar{t})$,
- for each $t_i \in \bar{t}$, there is an inductive type $D \cdot \bar{d}$, where $D = \text{Ind}_n\{\Delta_I := \Delta_C\}$, and a term $T_i$ such that
  - $t_i = \lambda \bar{y}: \bar{A}. \lambda z: (D \cdot \bar{d}) \bar{u}. t'_i$, where $lh(\bar{Y}) = k_i$, and
  - there is a constrained derivation with respect to $z$ and $\Delta_I, \Delta_C$ such that
    \[ \Gamma^e, (\bar{g}: \bar{Y})^e, z : \epsilon (D \cdot \bar{d}) \bar{u}, (\bar{f}: A)^e \bar{A}^e_k \vdash t'_i : \epsilon T_i^e \]

where $(\bar{f}: A)^e_k$ is the context composed of

\[ f_j : \epsilon \Pi(\bar{a}: \bar{B}_j)^e. \Pi v : \epsilon X_j, P_j^e \]

if $f_j : \Pi(\bar{a}: \bar{B}_j). \Pi v : X_j, P_j$ is from $\bar{f}: A$ and $lh(B_j) = k_j$.

The condition for constrained derivation ensures that the constructed terms are normalizing terms. A formal definition is given in Appendix [B]. Informally, it means that $t_i$ can only contain decreasing recursive calls: if $f_j$ appears in $t_i$, then it must have at least $k_j + 1$ arguments, and its $(k_j + 1)$th argument must be structurally smaller than the initial inductive argument $z$ (Thus, any subterm of an inductive term obtained by going through at least one constructor is structurally smaller than the initial term.).

### Judgmental equality and type universes (Figure 3)

The rules in Figure 3 stipulate that the judgmental equality based on reductions is an equivalence relation. De Bruijn’s telescope notation is very useful: $\Gamma \vdash \bar{t}: \bar{A}$ with $lh(\bar{t}) = n$ implies that

- $\Gamma, x_1 : A_1, ..., x_{j-1} : A_{j-1} \vdash A_j : s_j$ for all $j \in \{1, ..., n\}$, and
- $\Gamma \vdash t_j : A_j[x_1 \setminus t_1] \cdot \cdot \cdot [x_{j-1} \setminus t_{j-1}]$ for all $j \in \{1, ..., n\}$.

### 3. Set-theoretic model construction

#### 3.1. Background

We must resolve a dilemma related to the construction of a set-theoretic model of CC and its extensions. In a proof-irrelevant model, each type expression should have an obvious set-theoretic interpretation; however, it is well known that impredicative or polymorphic types, such as $\text{Prop}$, can only have a trivial set-theoretic interpretation, as shown by [Reynolds(1984)].

Hence, it is necessary to assign a singleton or the empty set to each term of type $\text{Prop}$.

In constructing a set-theoretic model of Coquand’s CC, [Miquel and Werner(2003)] provided the following solution. Under the assumption of the existence of a urelement $\bullet$ that does not belong to the standard universe of set theory, the sort $\text{Prop}$ is associated with
Lemma 3.3. Furthermore, the application and $\lambda$-abstraction terms are interpreted by $\text{app}$ and $\text{lam}$, respectively, which are defined as follows:

\[
\begin{align*}
\text{app}(u, x) & := \begin{cases} 
\bullet & \text{if } u = \bullet, \\
 u(x) & \text{otherwise.}
\end{cases} \\
\text{lam}(f) & := \begin{cases} 
\bullet & \text{if } f(x) = \bullet \text{ for all } x \in \text{dom}(f), \\
 f & \text{otherwise.}
\end{cases}
\end{align*}
\]

Remark 3.1. This construction does not correctly model the cumulative relation between $\text{Prop}$ and $\text{Type}_i$, as demonstrated in the following example; [Werner(2008)] showed that it can be easily extended to the cumulative type universes when the subtyping relation between $\text{Prop}$ and $\text{Type}_i$ does not exist.

Consider $I = \lambda(A : \text{Type}_0). A \rightarrow A$. Then, its type of set-theoretic interpretation is not deterministic. Suppose that $P$ is a true proposition. Then, $[I P]$ depends on the type we have assigned to $P$, that is, $\text{Prop}$ or $\text{Type}_0$. In the former case, $[I P] = \{\bullet\}$ since $P \rightarrow P$ is a tautology, whereas in the latter case, $[I P] = [I ([P])] = \{f \mid f : \bullet \rightarrow \bullet\} \neq \{\bullet\}$ since $[P] = \{\bullet\}$.

Another solution was provided by [Aczel(1998)]. He used the trace encoding of functions in order to provide an adequate interpretation of the impredicative type $\text{Prop}$ of propositions and its relationship with $\text{Type}_i$. For this reason, we adopt Aczel’s solution.

Definition 3.2 (Trace encoding of set-theoretic functions). Let $u, x, f$ denote sets. Then,

\[
\begin{align*}
\text{app}(u, x) & := \{z \mid (x, z) \in u\}, \\
\text{lam}(f) & := \mathop{\bigcup}_{(x, y) \in f} \{(x) \times y\} = \mathop{\bigcup}_{(x, y) \in f} \{(x, z) \mid z \in y\}.
\end{align*}
\]

Note that for any function $f$ and any $x \in \text{dom}(f)$, we have

\[
\text{app}(\text{lam}(f), x) = \{y \mid (x, y) \in \text{lam}(f)\} = \{y \mid y \in f(x)\} = f(x).
\]

Notations.

(1) In the remainder of this paper, $\downarrow$ is used if something is well defined, and $\uparrow$ is used otherwise.

(2) Given sets $A, B(x), x \in A$, let $\prod_{x \in A} B(x)$ denote the set of all functions $f$ such that $\text{dom}(f) = A$ and $f(x) \in B(x)$ for all $x \in A$.

(3) Given a function $f \in \prod_{x_1 \in A_1} \ldots \prod_{x_n \in A_n} B(x_1, \ldots, x_n)$, we use the notation $\text{lam}_n(f)$ (resp. and $\text{app}(f, \vec{x})$) for the $n$-times application of $\text{lam}$ (resp. $\text{app}$):

\[
\begin{align*}
\text{lam}_n(f) & := \{(x_1, \ldots, x_n, y) \mid x_1 \in A_1, \ldots, x_n \in A_n, y \in f(x_1, \ldots, x_n)\} \\
\text{app}_n(f, \vec{x}) & := \text{app}(\ldots(\text{app}(f(x_1), x_2), \ldots), x_n)
\end{align*}
\]

We suppress the subscript $n$ when the number of times we want to apply $\text{lam}$ or $\text{app}$ is obvious from the context. Note that $\text{lam}_0(f) = f$ and $\text{app}_0(f, \text{nil}) = f$.

Lemma 3.3 ([Aczel(1998)]). Given a set $A$, assume $B(x) \subseteq 1$ for all $x \in A$.

(1) $\{\text{lam}(f) \mid f \in \prod_{x \in A} B(x)\} \subseteq 1$.

(2) $\{\text{lam}(f) \mid f \in \prod_{x \in A} B(x)\} = 1$ iff $\forall x \in A (B(x) = 1)$. 

Proof. Let \( f \in \prod_{x \in A} B(x) \), i.e.,
\[
    f = \begin{cases} 
        \uparrow & \text{if } \exists x \in A \, (B(x) = \emptyset), \\
        \{ (x, \emptyset) \mid x \in A \} & \text{otherwise.}
    \end{cases}
\]
Then, we have
\[
    \lambda f = \begin{cases} 
        \uparrow & \text{if } \exists x \in A \, (B(x) = \emptyset), \\
        \emptyset & \text{otherwise.}
    \end{cases}
\]
This also implies that \( \{ \lambda f \mid f \in \prod_{x \in A} B(x) \} = 1 \) iff \( \forall x \in A \, (B(x) = 1) \).

**Remark 3.4.** A useful feature of trace encoding is that \( \text{app}(u, a) \) and \( \lambda f \) are always defined for any sets \( u, a, f \), including the empty set. This, however, implies that we sometimes lose the information of the domain of a given function \( f \), i.e., we cannot trace back to \( \text{dom}(f) \) starting from \( \lambda f \). We will see that the use of judgmental equality enables us to avoid such a loss when only well-typed terms are involved.

### 3.2. Inductive types and rule sets

Here, we follow the approaches of [Aczel(1998)] and [Dybjer(1991)] for the construction of a set-theoretic interpretation of inductive types. We are particularly interested in rule sets.

We are going to work on the basis of ZF set theory with an axiom guaranteeing the existence of countably many (strongly) inaccessible cardinals. Note that such an axiom is independent of ZFC. [Werner(1997)] showed that such an axiom is sufficient for a set-theoretic interpretation of the cumulative type universes and predicatively inductive types. However, it is not clear whether this axiom is necessary for our construction. Indeed, the required feature of an inaccessible cardinal \( \kappa \) is the closure property of the universe \( V_\kappa \) under the powerset operation. This is a necessary condition for the interpretation of inductive types.

Henceforth, assume that there are countably many (strongly) inaccessible cardinals. Let \( \kappa_0 = \omega \) and \( \kappa_1, \kappa_2, \ldots \) enumerate these inaccessible cardinals. We associate each sort \( \text{Type}_i \) with its rank \( \text{rank}(\text{Type}_i) := \kappa_i \). If \( (V_\alpha)_{\alpha \in \text{Ord}} \) denotes the (standard) universe of sets defined as follows, then \( V_\kappa \) is a model of ZF:
\[
    V_0 := \emptyset \quad \text{and} \quad V_\alpha := \bigcup_{\beta < \alpha} \mathcal{P}(V_\beta) \quad \text{if } \alpha > 0
\]
\( \text{Ord} \) denotes the class of all ordinals, \( \lambda \) denotes a limit ordinal, and \( \mathcal{P} \) denotes the power set operator. In particular, if \( \kappa \) is an inaccessible cardinal, \( A \in V_\kappa \), and for every \( a \in A \), \( B_a \in V_\kappa \), then, \( \prod_{a \in A} B_a \in V_\kappa \). Let \( \text{rank}(\text{Prop}) := -1 \) and \( V_{-1} = \{0, 1\} \) for convenience. Refer to [Drake(1974)] for further details about inaccessible cardinals.

A rule on a base set \( U \) is a pair of sets \( \langle u, v \rangle \), often written as \( \frac{u}{v} \), such that \( u \subseteq U \) and \( v \in U \). A set of rules on \( U \) is called a rule set on \( U \). Given a rule set \( \Phi \) on \( U \), a set \( w \subseteq U \) is \( \Phi \)-closed if for any \( \frac{u}{v} \in \Phi \), \( v \in w \) whenever \( u \subseteq w \). Note that there is the least \( \Phi \)-closed set
\[
    I(\Phi) := \bigcap \{ w \subseteq U \mid w \, \Phi \text{-closed} \}.
\]
In fact, it is well known that each rule set \( \Phi \) on \( U \) generates a monotone operator on \( \mathcal{P}(U) \)
\[
    \Gamma_\Phi(X) := \{ v \in U \mid \text{there exists some } u \subseteq X \text{ such that } \frac{u}{v} \in \Phi \}.
\]
Then, there is some rule set $\Phi$ mentioned before, we suppress ($\Phi$ satisfies the soundness of the rules\footnote{\emph{Notations.}}\footnote{\emph{Interpretation of inductive types.}} of inductive types that satisfy the soundness of the rules \footnote{\emph{Examples.}}\footnote{\emph{Given a well-defined (mutually) inductive type $\text{Ind}_n(\Delta_f := \Delta_C)$, where}}\footnote{\emph{Here, we claim the existence of the interpretations of inductive and recursive types with some examples. Given a well-defined (mutually) inductive type $\text{Ind}_n(\Delta_f := \Delta_C)$, where}}\footnote{\emph{Let $\Delta_f := d_0 : A_0, ..., d_\ell : A_\ell, \Delta_C := c_1 : T_1, ..., c_m : T_m$,}}\footnote{\emph{Further suppose that}}\footnote{\emph{Then, there is some rule set $\Phi$ such that the following interpretation of $D \cdot d_i$ and $D \cdot c_k$ satisfies the soundness of the rules (ind-type) and (ind-const):}}\footnote{\emph{Lemma 3.5. Suppose $\Gamma \vdash D$, where $D = \text{Ind}_n(\Delta_f := \Delta_C)$. Let $\gamma \in [\Gamma]$ be given. As mentioned before, we suppress $\Gamma$ and $\gamma$ for better readability. Further suppose that}}\footnote{\emph{PROOF-IRRELEVANT MODEL OF CC WITH PRED. INDUCTION AND JUDG. EQUALITY 13}}

A rule set is \emph{deterministic} provided that it contains at most one rule with a given conclusion. The rule sets defined below by an inductive definition are deterministic. This makes it possible to interpret functions defined by structural recursion on a certain inductive type as set-theoretic functions. The interpretations are defined on the corresponding set-theoretically defined set, which is the fixpoint of a monotone operator. Refer to \footnote{\emph{Aczel(1977)} and \emph{Moschovakis(1974)}, \emph{Moschovakis(1980)}} for further details about rule sets, monotone operators, and fixpoints.

Below, we describe the interpretations of inductive and recursive types with some examples. Given a well-defined (mutually) inductive type $\text{Ind}_n(\Delta_f := \Delta_C)$, where $\Delta_f = x_0 : A_1, ..., x_\ell : A_\ell$ and $A_i = \Pi \overrightarrow{b}_i : \overrightarrow{B}_i, s_i$, let $\text{rank}(x_i) := \text{rank}(s_i)$.

\section*{Notations.}

1. With each context $\Gamma$, we associate a set $[\Gamma]$ of $\Gamma$-valuations of the form $\langle \alpha_1, \ldots, \alpha_n \rangle$, where $n$ is the length of $\Gamma$ and $\langle , \ldots \rangle$ denotes a sequence of a finite length. Given a sequence $L = \langle \alpha_1, \ldots, \alpha_n \rangle$ and a natural number $i < n$, we set $(L)_i = \alpha_{i+1}$. If $\alpha_i$ itself is a sequence of length $m$, then we write $(L)_{i,j}$ for $(\alpha_{i+1})_j$ if $j < m$, etc.

2. $\alpha, \beta, \alpha_i, \beta_i$ vary over single values while $\gamma, \delta, \gamma_i, \delta_i$ vary over valuations. nil denotes the empty sequence. Given two valuations $\gamma$ and $\delta$, the notation $\gamma, \delta$ denotes their concatenation. If $\delta = \langle \alpha \rangle$, then we write $\gamma, \alpha$ instead of $\gamma, \langle \alpha \rangle$.

3. With each pair $(\Gamma, t)$ formed by a context $\Gamma$ and a term $t$, we associate a function $[\Gamma \vdash t]$ that is partially defined on $\Gamma$-valuations: $[\Gamma \vdash t]_\gamma$ denotes $[\Gamma \vdash t]_\gamma(\gamma)$ when $\gamma \in [\Gamma]$.

4. In the following, we write $[t]$ for $[\Gamma \vdash t]_\gamma$ if $\Gamma$ and $\gamma \in [\Gamma]$ are fixed in the context.

Similarly, we use the notation $\overrightarrow{u} \in [A]$ for $u_1 \in [\Gamma \vdash A_1]_\gamma, ..., u_n \in [\Gamma, x_1 : A_1, ..., x_n : A_n]_\gamma$ for some context $\Gamma$ and $\Gamma$-valuation $\gamma \in [\Gamma]$.

\section*{3.3. Interpretation of inductive types.}

Here, we claim the existence of the interpretations of inductive types that satisfy the soundness of the rules (ind-wf), (ind-type), and (ind-const) when the conditions in the typing rules are fullfilled. The formal definition is given in Appendix \footnote{\emph{Refer to Dybjer(1991)}, whose idea is generalized in this paper.}

\begin{lemma}
Suppose $\Gamma \vdash D$, where $D = \text{Ind}_n(\Delta_f := \Delta_C)$. Let $\gamma \in [\Gamma]$ be given. As mentioned before, we suppress $\Gamma$ and $\gamma$ for better readability. Further suppose that

$$
\Delta_f := d_0 : A_0, ..., d_\ell : A_\ell, \Delta_C := c_1 : T_1, ..., c_m : T_m,
$$

$$
A_i := \Pi \overrightarrow{\overrightarrow{b}}_i : \overrightarrow{B}_i, s_i, T_k := \Pi \overrightarrow{\overrightarrow{b}}_k : \overrightarrow{B}_k, \overrightarrow{t}_k.
$$

Then, there is some rule set $\Phi$ such that the following interpretation of $D \cdot d_i$ and $D \cdot c_k$ satisfies the soundness of the rules (ind-type) and (ind-const):

- $[D \cdot d_i] := D\overrightarrow{m}(f_i)$ where $f_i(\overrightarrow{\overrightarrow{b}}_i) := [\mathcal{I}\mathcal{F}(\Phi)(i, \overrightarrow{\overrightarrow{b}}_i) : \overrightarrow{\overrightarrow{P}}, \overrightarrow{\overrightarrow{B}}]$, $\overrightarrow{m}(f_i(\overrightarrow{\overrightarrow{b}}_i)) := \overrightarrow{\overrightarrow{f}}(\overrightarrow{\overrightarrow{b}}_i)$ for $\overrightarrow{\overrightarrow{b}}_i \in [\overrightarrow{\overrightarrow{P}}, \overrightarrow{\overrightarrow{B}}]$,

- $[D \cdot c_k] := D\overrightarrow{m}(g_k)$ where $g_k(\overrightarrow{\overrightarrow{b}}_k) := \langle k, \overrightarrow{\overrightarrow{z}}_k \rangle$ for $\overrightarrow{\overrightarrow{b}}_k \in [\overrightarrow{\overrightarrow{P}}, \overrightarrow{\overrightarrow{Z}}_k]$.
\end{lemma}
Remark 3.6. The positivity condition is crucial for showing that the construction of the rule set $\Phi$ in Appendix C is well defined.

The elements of our rule sets $\Phi$ are of the form $\frac{\mathcal{I}F(\Phi_{\text{nat}})(0), [0] = \langle 1 \rangle, \text{and } \text{app}([S], n) = \langle 2, n \rangle \text{ for any } n \in [\text{nat}]}$, where $\mathcal{I}F(\Phi_{\text{nat}})(0), [0] = \langle 1 \rangle, \text{and } \text{app}([S], n) = \langle 2, n \rangle \text{ for any } n \in [\text{nat}]

Example 3.7 (Natural numbers). Let $\mathcal{D}_N$ be the inductive type for natural numbers, as in Remark 2.6 Then,

$\Phi_{\text{nat}} = \left\{ \begin{array}{c}
\mathcal{N} \\
(0, \langle 1 \rangle)
\end{array} \right\} \cup \left\{ \begin{array}{c}
\mathcal{N} \\
(0, \langle 2, v_1, v_2 \rangle)
\end{array} \right\} \mid v \in \mathcal{V}_{\kappa_0}, \ x, v_1, v_2 \in \mathcal{V}_{\kappa_1},

\text{app}([\text{nat}], x) = \mathcal{I}F(\Phi_{\text{nat}})(0, x), \text{app}([Y_1], x) = \langle 1, x \rangle, \text{and } \text{app}([Y_2], x, a, b) = \langle 2, x, a, b \rangle,

\text{where } x \in \mathcal{V}_{\kappa_1}, a \in \text{app}([\text{titi}], [\text{nat}]), \text{and } b \in \text{app}([\text{titi}], x).

Example 3.8 (Inductive families). The following Coq-expression shows a typical use of inductive families.

Inductive toto : Type -> Type :=
| Y1 : forall x : Type, toto x
| Y2 : forall x : Type, toto nat -> toto x -> toto x.

The inductive type toto can be represented by $\mathcal{D}_{\text{toto}} = \text{Ind}_0(\Delta_1 := \Delta_C)$, where

$\Delta_1 := \text{toto : Type}_{1} \rightarrow \text{Type}_{1},$

$\Delta_C := \Pi x : \text{Type}_{1}. \text{toto } x, \quad \Pi x : \text{Type}_{1}. \text{toto nat } \rightarrow \text{toto } x \rightarrow \text{toto } x.$

Then,

$\Phi_{\text{toto}} = \left\{ \begin{array}{c}
\mathcal{N} \\
(0, x, \langle 1, x \rangle)
\end{array} \right\} \cup \left\{ \begin{array}{c}
\mathcal{N} \\
(0, x, \langle 2, x, v_1, v_2 \rangle)
\end{array} \right\} \mid x, v_1, v_2 \in \mathcal{V}_{\kappa_1},

\text{app}([\text{titi}], x) = \mathcal{I}F(\Phi_{\text{titi}})(0, x), \text{app}([Z_1], x) = \langle 1, x \rangle, \text{and } \text{app}([Z_2], x, a, b) = \langle 2, x, a, b \rangle,

\text{where } x \in \mathcal{V}_{\kappa_1}, a \in \text{app}([\text{titi}], [\text{nat}]), \text{and } b \in \text{app}([\text{titi}], x).

Example 3.9 (Inductive types with parameters). The following Coq-expression shows a typical use of parametric inductive types.

Inductive titi (x : Type) : Type :=
| Z1 : titi x
| Z2 : titi nat -> titi x -> titi x.

The inductive type titi can be represented by $\mathcal{D}_{\text{titi}} = \text{Ind}_1(\Delta_1 := \Delta_C)$, where

$\Delta_1 := \text{titi : Type}_{0} \rightarrow \text{Type}_{0},$

$\Delta_C := \Pi x : \text{Type}_{1}. \text{titi } x, \quad \Pi x : \text{Type}_{1}. \text{titi nat } \rightarrow \text{titi } x \rightarrow \text{titi } x.$

Then,

$\Phi_{\text{titi}} = \left\{ \begin{array}{c}
\mathcal{N} \\
(0, x, \langle 1 \rangle)
\end{array} \right\} \cup \left\{ \begin{array}{c}
\mathcal{N} \\
(0, x, \langle 2, v_1, v_2 \rangle)
\end{array} \right\} \mid x, v_1, v_2 \in \mathcal{V}_{\kappa_0},

\text{app}([\text{titi}], x) = \mathcal{I}F(\Phi_{\text{titi}})(0, x), \text{app}([\text{Z1}], x) = \langle 1, x \rangle, \text{and } \text{app}([\text{Z2}], x, a, b) = \langle 2, x, a, b \rangle,

\text{where } x \in \mathcal{V}_{\kappa_1}, a \in \text{app}([\text{titi}], [\text{nat}]), \text{and } b \in \text{app}([\text{titi}], x).
Remark 3.10. Note that in Coq, toto cannot have Type -> Set as its type, unlike titi. This difference is also reflected in their interpretations.

Example 3.11 (Mutually inductive types with parameters). Two inductive types tree and forest defined by \(D_{TF}\) in Remark 2.5 can be interpreted by means of the following rule set:
\[
\Phi := \{ \{1, A, v_1\} \mid A, v_1 \in \mathcal{V}_{\kappa_0}, a \in A \} \\
\cup \{ \emptyset \mid A \in \mathcal{V}_{\kappa_0} \} \cup \{ \{0, A, v_1\}, \langle 1, A, \{v_2\}\rangle \mid A, v_1, v_2 \in \mathcal{V}_{\kappa_0} \}.
\]

3.4. Interpretation of well-founded structured recursion. A set defined by a (mutual) induction generates a canonical well-founded relation on the set, i.e., the relation defined according to the inductive construction of the elements, the so-called structurally-smaller-than-relation. This is the basis for the discipline of structural recursion, which stipulates that recursive calls consume structurally smaller data.

Here, we claim the existence of the interpretations of recursive types that satisfy the soundness of the rules (fix), (fix-eq), and (\(\iota\)). A formal definition is given in Appendix D. Refer to [Dybjer(1991)], whose study provides the basic idea.

**Lemma 3.12.** Suppose \(\Gamma \vdash \text{fix } f : \{R\} : A_j\), where
\[
R = \frac{f/k : A := t}{f/k : A := t}, \quad A_i \equiv \Pi x_i : B_i, A_i', \quad lh(B_i) = k_i + 1, \quad \ell \leq n,
\]
\[
(\Gamma \vdash A_i : s_i)_{i \leq n}, \quad (\Gamma, f : A \vdash t_i : A_i)_{i \leq n}, \quad F(f, A, k, i).
\]
Let \(\gamma \in \llbracket \Gamma \rrbracket\) be given. We suppress \(\Gamma\) and \(\gamma\) for better readability. Then, there is a rule set \(\Psi\) such that the following interpretation of \(\text{fix } f : \{R\}\) satisfies the soundness of the rules (fix), (fix-eq), and (\(\iota\)):

- \(\llbracket \text{fix } f : \{R\} \rrbracket = \text{lam}(h)\), where \(h(a_1, \ldots, a_{k_\ell}, \langle k, z_k\rangle) = I F(\Psi)(a_1, \ldots, a_{k_\ell}, \langle k, Z_k\rangle)\) for \(a, \langle k, z_k\rangle \in \llbracket B_i \rrbracket\).

**Remark 3.13.** The condition for constrained derivation is essential. Indeed, constrained derivation corresponds to guarded recursion defined by [Giménez(1995)]; hence, it guarantees that the construction of the rule set \(\Psi\) in Appendix D is well defined.

The elements of our rule sets \(\Psi\) are of the form \(\alpha(x, k, y, b)\), where \(k\) denotes the \(k\)th constructor of the inductive type \(d\) on which the recursion is performed, \(\bar{y}\) denote the non-parametric arguments of \(d\), \(\bar{x}\) denotes the list of rest arguments of the constructor, and \(b\) is the result of the function. Note that \(\bar{x}, \bar{y}\) could be empty.

**Example 3.14** (Primitive recursion). The Coq-expression stated below is a general form of primitive recursion.

```plaintext
match n with
| 0 => g
| S p => h p (PRec A g h p)
end.
```
The corresponding term is \( \text{PRec} := \text{fix } f_0 \{ f_0/3 : B := t \} \), where

\[
B = \Pi(A : \text{Type}). \Pi(g : A). \Pi(h : \text{nat} \to A \to A). \Pi(n : \text{nat}). n, \\
t = \lambda(A : \text{Type}). \lambda(g : A). \lambda(h : \text{nat} \to A \to A). \lambda(n : \text{nat}). \text{case}(n, P, h_1, h_2), \\
P = \lambda(\ell : \text{nat}). A, h_1 = g, \text{ and } h_2 = \lambda(p : \text{nat}). hp(f_0Agp).
\]

\( \text{PRec} \) is characterized by the following rule set \( \Psi \_	ext{PRec} := \)

\[
\begin{align*}
& \emptyset \\
& \cup \left\{ \langle A, g, h, \langle 1, g \rangle \rangle \mid A \in \mathcal{V}_{\kappa_1}, g \in A, h \in [\text{nat} \to A \to A] \right\} \\
& \cup \left\{ \langle A, g, h, \langle 2, p \rangle, \text{app}(h, p, v) \rangle \mid A \in \mathcal{V}_{\kappa_1}, g \in A, h \in [\text{nat} \to A \to A], p \in [\text{nat}], v \in A \right\}.
\end{align*}
\]

Given a type \( A \), \( [\text{PRec}A] \) denotes the primitive recursor with values from \( A \). For instance,

\[ \text{app}([\text{plus}], m, n) = I.F([\text{PRec}])([\text{nat}], m, [h], n) \]

where \( h = \lambda(p : \text{nat}). \lambda(\ell : \text{nat}). S\ell \) and \( m, n \in [\text{nat}] \).

**Example 3.15** (Mutually recursive functions). The interpretations of \( T\text{size} \) and \( F\text{size} \) from Remark 2.7 are characterized by the following rule set \( \Psi_{\text{Size}} := \)

\[
\begin{align*}
& \emptyset \\
& \cup \left\{ \langle A, f, v' \rangle \mid A \in \mathcal{V}_{\kappa_0}, f \in \text{app}([\text{forest}], A), v' \in [\text{nat}], a \in A \right\} \\
& \cup \left\{ \langle A, \langle 1, a, f \rangle, [S v'] \rangle \mid A \in \mathcal{V}_{\kappa_0} \right\} \\
& \cup \left\{ \langle A, \langle 2 \rangle, [0] \rangle \mid A \in \mathcal{V}_{\kappa_0} \right\} \\
& \cup \left\{ \langle A, t, v' \rangle, \langle A, f', v'_2 \rangle \rangle \mid A \in \mathcal{V}_{\kappa_0}, t \in \text{app}([\text{tree}], A), \\
& f' \in \text{app}([\text{forest}], A), v_1, v'_2 \in [\text{nat}] \right\}.
\end{align*}
\]

4. **Set-theoretic model and soundness**

Since the denotations \( [\Gamma] \) and \( [\Gamma \vdash t] \) will be defined by mutual induction on the size of their arguments, we need a size function \( | \cdot | \) that guarantees the termination. In particular, the following properties should be satisfied:

- \( |\Gamma| < |\Gamma | A | < |\Gamma, x : A| \),
- \( |\Gamma \vdash t|, |\Gamma \vdash A| < |\Gamma, t := t : A| \),
- \( |\Delta_I(d)|, |\Delta_C(c)| < |\text{ind}_0\Delta_I := \Delta_C| \cdot x| \) for all \( x \in \text{dom}(\Delta_I, \Delta_C) \), \( d \in \text{dom}(\Delta_I) \), and \( c \in \text{dom}(\Delta_C) \),
- \( |A_i|, |t_j| < |f/k : A := t| \) for all \( A_i \in \mathcal{A} \) and \( t_j \in \mathcal{T} \).

An adequate size function can be defined by a simple extension of the one defined by Miquel and Werner (2003):

- The size \( |t| \) of a term \( t \) is (recursively) defined as the sum of the sizes of its immediate subterms plus 1.
– The immediate subterms of $\text{Ind}_n\{\Delta_I := \Delta_C\} \cdot x$ are $\Delta_I(d)$ and $\Delta_C(c)$, where $d \in \text{dom}(\Delta_I)$ and $c \in \text{dom}(\Delta_C)$.
– The immediate subterms of $\text{fix} \ f_j \{f/k : A \rightarrow\} \}$ are $\bar{A}, \bar{t}$.

• The size of a context $\Gamma$ is defined as follows:
  1. $|[]| = 1/2$,
  2. $|\Gamma, (x : t)| = |\Gamma| + |t|$,
  3. $|\Gamma, (x := t : A)| = |\Gamma| + |t| + |A|$,
  4. $|\Gamma, \text{Ind}_n\{\Delta_I := \Delta_C\}| = |\Gamma| + 1$.

• $|\Gamma \vdash t| = |\Gamma| + |t| - \frac{1}{2}$.

**Remark 4.1.** As mentioned in Remark 3.6 and Remark 3.13, the positivity condition and the condition for constrained derivation play a crucial role for establishing the soundness proof of Theorem 4.3.

**Definition 4.2.** The set-theoretic interpretations of $[\Gamma]$ and $[\gamma \vdash t]$ are defined by a mutual induction on the size of their arguments.

1. For each context $\Gamma$, the set $[\Gamma]$ is defined as follows:
   
   \[
   [\Gamma] := \{\text{nil}\},
   \]
   
   \[
   G, x : A := \{\gamma, \alpha \mid \gamma \in G, [\Gamma \vdash A]_\gamma \downarrow \text{ and } \alpha \in [\Gamma \vdash A]_\gamma\},
   \]
   
   \[
   G, x := t : A := \{\gamma, \alpha \mid \gamma \in G, [\Gamma \vdash A]_\gamma \downarrow, [\Gamma \vdash t]_\gamma \downarrow \text{ and } \alpha = [\Gamma \vdash t]_\gamma \in [\Gamma \vdash A]_\gamma\},
   \]
   
   \[
   G, \text{Ind}_n\{\Delta_I := \Delta_C\} := \{\gamma \mid \gamma \in G\}.
   \]

2. The interpretation $[\Gamma \vdash t]$ of a term $t$ in a context $\Gamma$ is a partial function defined on $[\Gamma]$:
   Given $\gamma \in [\Gamma]$, $[\Gamma \vdash \text{Prop}]_\gamma := \{0, 1\}$,
   $[\Gamma \vdash \text{Type}]_\gamma := \forall \eta_i$,
   $[\Gamma \vdash x]_{(a_0, ..., a_n)} := \alpha_i$ if $x$ is the $i$th declared variable in $\Gamma$, \quad (*)
   $[\Gamma \vdash \Pi x : A.B]_\gamma := \{\lambda \text{m}(f) : f \in \Pi \alpha \in [\Gamma \vdash A], [\Gamma, x : A \vdash B]_{\gamma(\alpha)}\}$,
   $[\Gamma \vdash \lambda x : A.t]_\gamma := \lambda \text{m}(\alpha \in [\Gamma \vdash A]_\gamma \mapsto [\Gamma, x : A \vdash t]_{\gamma(\alpha)})$,
   $[\Gamma \vdash t u]_\gamma := \text{app}(([\Gamma \vdash t]_\gamma, [\Gamma \vdash u]_\gamma)$,
   $[\Gamma \vdash \text{let} \ x := t \ in \ u]_\gamma := [\Gamma, (x := t : A) \vdash u]_{\gamma[\gamma-t]}$, \quad (†)
   where $A$ is such that $[\Gamma \vdash t] \in [\Gamma \vdash A]$,
   $[\Gamma \vdash \text{Ind}_n\{\Delta_I := \Delta_C\} \cdot z]_\gamma := \text{as explained above if defined},$
   $[\Gamma \vdash \text{case}\(e, P, M_1, \ldots, M_n\)]_\gamma := \text{app}(([\Gamma \vdash e], ([\Gamma \vdash e]_0), \ldots, ([\Gamma \vdash e]_q)$
   \quad if $([\Gamma \vdash e]_0 = j$ where $lh(e) = q + 1$, $\text{if} \ \text{defined}$.

\quad (†) $A$ could be any term with the given property since the interpretation, when defined, is independent of it.

The following lemma is crucial for the soundness proof.
Lemma 4.3 (Substitutivity). Let \( \Gamma \) be a context and let \( u, A \) be terms such that \([\Gamma \vdash u]_\gamma \in [\Gamma \vdash A]_\gamma\) for some \( \gamma \in [\Gamma] \) (assuming that both of them are defined), and write \( \alpha = [\Gamma \vdash u]_\gamma \).

1. Suppose \((\gamma, \alpha), \delta \in [\Gamma, x : A, \Delta]\). Then, \(\gamma, \delta \in [\Gamma, \Delta[x/u]]\).
2. Suppose \((\gamma, \alpha), \delta \in [\Gamma, x : A, \Delta]\) and \([\Gamma, x : A, \Delta \vdash t]_{(\gamma, \alpha), \delta} \downarrow\). Then,
   - \([\Gamma, \Delta[x/u] \vdash t[x/u]]_{\gamma, \delta} \downarrow\).
   - \([\Gamma, \Delta[x/u] \vdash t[x/u]]_{\gamma, \delta} = [\Gamma, x : A, \Delta \vdash t]_{(\gamma, \alpha), \delta} = [\Gamma, x := u : A, \Delta \vdash t]_{(\gamma, \alpha), \delta}\).

Proof. The assertions are proved for each \( \Delta \) and \( t \) by a mutual induction on the size of their arguments. In particular, given \( \Delta \), the first assertion is proved before the second one for all \( t \). In the case of \( \Delta = [] \), the claims are obvious. Assume that \( \Delta = \Delta_0, y : B \) and \( \delta = \delta_0, \beta \). The other cases can be considered similarly.

1. \((\gamma, \alpha), \delta_0, \beta \in [\Gamma, x : A, \Delta_0, y : B]\). Then, using the I.H. of the second claim, we have
   \[\beta \in [\Gamma, x : A, \Delta_0 \vdash B]_{\gamma, \alpha, \delta_0} = [\Gamma, \Delta_0[x/u] \vdash B[x/u]]_{\gamma, \delta_0} = [\Gamma, u : B[x/u]]\].
   This is because the interpretation of \( u \) does not depend on \( \text{dom}(\Delta) \).

2. We proceed by induction on \( t \). If \( t = x \), the claim follows because \([\Gamma \vdash u]_{\gamma} \downarrow\) implies that \([\Gamma, \Delta[x/u] \vdash u]_{\gamma, \delta} \downarrow\) and \([\Gamma, \Delta[x/u] \vdash u]_{\gamma, \delta} = [\Gamma, u : B[x/u]]\). This is because the interpretation of \( u \) does not depend on \( \text{dom}(\Delta) \).

Theorem 4.4 (Soundness). Our type system is sound with respect to the set-theoretic interpretation defined in Definition 4.3 in the following sense:

1. If \( \text{WF}(\Gamma) \), then \([\Gamma]\) is defined.
2. If \( \Gamma \vdash M : A \), then \([\Gamma]\) is defined, and for any \( \gamma \in [\Gamma] \), it holds that \([\Gamma \vdash M]_\gamma\) and \([\Gamma \vdash A]_\gamma\) are defined, and that \([\Gamma \vdash M]_\gamma \in [\Gamma \vdash A]_\gamma\).
3. If \( \Gamma \vdash M = N : A \), then \([\Gamma]\) is defined, and for any \( \gamma \in [\Gamma] \), it holds that \([\Gamma \vdash M]_\gamma\), \([\Gamma \vdash N]_\gamma\), and \([\Gamma \vdash A]_\gamma\) are defined, and that \([\Gamma \vdash M]_\gamma = [\Gamma \vdash N]_\gamma \in [\Gamma \vdash A]_\gamma\).
4. If \( \Gamma \vdash M < N \), then \([\Gamma]\) is defined, and for any \( \gamma \in [\Gamma] \), it holds that \([\Gamma \vdash M]_\gamma\) and \([\Gamma \vdash N]_\gamma\) are defined, and that \([\Gamma \vdash M]_\gamma \subseteq [\Gamma \vdash N]_\gamma\).

Proof. We proceed by a simultaneous induction over the typing derivation. The cases \((wf), (ax), (var), (weak), \) and \((weak-eq)\) are obvious.

(P) Suppose

\[
\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \\
\Gamma \vdash \Pi x : A.B : s_3
\]

By I.H., it holds that \([\Gamma \vdash A]_\gamma \in [\Gamma \vdash s_1]_\gamma\) and \([\Gamma, x : A \vdash B]_{\gamma, \alpha} \in [\Gamma, x : A \vdash s_2]_{\gamma, \alpha}\) for all \( \alpha \in [\Gamma \vdash A]_\gamma \). Now, we need to show that
\[\Gamma \vdash \Pi x : A.B \in [\Gamma \vdash s_3]_\gamma\].
If \( s_2 = s_3 = \text{Prop} \), then Lemma 3.3 implies the claim. Assume \( s_1 = \text{Type}_i, s_2 = \text{Type}_j, s_3 = \text{Type}_k \), and \( i, j \leq k \). Then, \( [\Gamma \vdash s_3] = V_{\kappa_k} \), where \( \kappa_k \) is the \( k \)th inaccessible cardinal; hence, \( V_{\kappa_k} \) is closed under the power set operation.

The cases \((\Pi\text{-eq}), (\lambda)\), and \((\lambda\text{-eq})\) are obvious.

\((\text{app})\) Suppose
\[
\Gamma \vdash M : \Pi x : A.B \quad \Gamma \vdash N : A
\]

By induction hypothesis, it holds that \( [\Gamma \vdash N]_\gamma \in [\Gamma \vdash A]_\gamma \) and \( [\Gamma \vdash M]_\gamma = \text{lam}(f) \) for some function \( f \) with \( \text{dom}(f) = [\Gamma \vdash A] \) and \( f(\alpha) \in [\Gamma, x : A \vdash B]_\gamma,\alpha \) for any \( \alpha \in [\Gamma \vdash A]_\gamma \). Thus, we have
\[
[\Gamma \vdash MN]_\gamma = \text{app}([\Gamma \vdash M]_\gamma, [\Gamma \vdash N]_\gamma) = f([\Gamma \vdash N]_\gamma)
\]
\[
\in [\Gamma, x : A \vdash B]_\gamma,\gamma = [\Gamma \vdash B[x \backslash N]]_\gamma.
\]

The cases \((\text{app-eq}), (\text{let}), \) and \((\text{let-eq})\) are similar.

The soundness of \((\text{ind-wf}), (\text{ind-type})\), and \((\text{ind-cons})\) are obvious from the interpretation constructions. The interpretations of inductive types and constructors are possible because of the induction hypotheses. This is the same for \((\text{fix}), (\text{fix-eq}), (\text{case}), \) and \((\text{case-eq})\).

The cases \((\text{ref}), (\text{sym}), (\text{trans}), (\text{conv})\), and \((\text{conv-eq})\) are obvious.

\((\beta)\) Suppose
\[
\Gamma, x : A \vdash M : B \quad \Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \Gamma \vdash N : A
\]

It remains to show that \([\Gamma \vdash (\lambda x : A.M)N]_\gamma = [M[x \backslash N]]_\gamma\):
\[
[\Gamma \vdash (\lambda x : A.M)N]_\gamma = \text{app}([\Gamma \vdash \lambda x : A.M]_\gamma, [\Gamma \vdash N]_\gamma)
\]
\[
= \text{app}(\text{lam}(\alpha \in [\Gamma \vdash A]_\gamma \mapsto [\Gamma, x : A \vdash M]_\gamma,\alpha), [\Gamma \vdash N]_\gamma)
\]
\[
= [\Gamma, x : A \vdash M]_\gamma,\gamma = [M[x \backslash N]]_\gamma
\]

by Lemma 4.3 because we know that \([\Gamma \vdash N]_\gamma \in [\Gamma \vdash A]_\gamma\) by induction hypothesis. The judgmental equality plays a crucial role in this case.

The case \((\delta)\) is obvious, and the case \((\zeta)\) follows from Lemma 4.3 and induction hypothesis. The case \((\iota)\) is obvious by definition. The cumulativity rules are obviously sound. Finally, the soundness of the the constrained typing rules in Figure 4 follows directly from the arguments stated above.

\[\square\]

**Theorem 4.5** (Consistency). There is no term \( t \) such that \( \vdash t : \Pi x : \ast. x \).

**Proof.** Note that \([\vdash \Pi x : \ast. x]_{\text{nll}} = \varnothing\).

\[\square\]
5. Conclusion

We identified some critical issues in constructing a set-theoretic, proof-irrelevant model of CC with cumulative type universes. Our construction reconfirmed that proof-irrelevance is a subtle and difficult subject to tackle when it is combined with the subtyping of the universes, in particular, Prop \(\prec\) Type. We showed that the set-theoretic interpretation can be relatively easy when we work with judgmental equality. We believe that our study provides a (relatively) easy way for justifying the correctness of type theory in Martin-Löf-styled, i.e., with simple model and, in particular, no proof of the strong normalization, which is usually very difficult to establish.

Besides the historical importance of Martin-Löf-style type theory and the technical difficulties with external \(\beta\)-reduction, there is another theoretical and practical reason for studying type systems with judgmental equality. In general, the equivalence of two systems with or without judgmental equality remains an open problem. Proving the equivalence of two systems with or without judgmental equality is not a simple task, even though some positive results have been achieved by [Coquand(1991), Goguen(1994), Goguen(1999), Adams(2006)], and [Siles and Herbelin(2010)]. However, they are not sufficiently general to cover the case with cumulative type universes. Although Adams(2006) mentioned that it might be possible to extend his proof to more general systems with unique principal types instead of type uniqueness as in the case of Luo’s ECC [Luo(1990), Luo(1994)] and Coquand’s CIC, it still remains an open question.

A positive consequence of the work of [Adams(2006)] and [Siles and Herbelin(2010)] is that the failed attempt of [Miquel and Werner(2003)], i.e., without using sorted variables, would work if one first considers the system CC with judgmental equality and uses its equivalence to the usual CC. This is indeed the case for the model construction described in this paper, where we restrict the model construction to CC.

Acknowledgment

We would like to express our sincere gratitude to Hugo Herbelin and Bruno Barras for their insightful discussions and advice. We would also like to thank the anonymous referees whose comments enabled us to improve this paper significantly.

References


APPENDIX A. Definition of free variables and substitution

The definitions of the sets of free variables in a context or term are standard.

\[ y[x\setminus u] := \begin{cases} u & \text{if } x = y, \\ y & \text{otherwise}, \end{cases} \]

\[ s[x\setminus u] := s, \]

\[ (\Pi y : A.B)[x\setminus u] := \begin{cases} \Pi y : A[x\setminus u].B & \text{if } x = y, \\ \Pi y : A[x\setminus u].(B[x\setminus u]) & \text{otherwise}, \end{cases} \quad (\star) \]

\[ (\lambda y : A.B)[x\setminus u] := \begin{cases} \lambda y : A[x\setminus u].B & \text{if } x = y, \\ \lambda y : A[x\setminus u].(B[x\setminus u]) & \text{otherwise}, \end{cases} \quad (\star) \]

\[ (\text{let } y := t_1 \text{ in } t_2)[x\setminus u] := \begin{cases} (\text{let } y := t_1[x\setminus u] \text{ in } t_2) & \text{if } x = y, \\ (\text{let } y := t_1[x\setminus u] \text{ in } t_2[x\setminus u]) & \text{otherwise}, \end{cases} \quad (\star) \]

\[ (t_1 t_2)[x\setminus u] := (t_1[x\setminus u])(t_2[x\setminus u]), \]

\[ \text{case}(e, P, f)[x\setminus u] := \text{case}(e[x\setminus u], P[x\setminus u], f[x\setminus u]), \]

\[ (\text{Ind}_n\{\Delta_I := \Delta_C\} \cdot y)[x\setminus u] := \begin{cases} (\text{Ind}_n\{\Delta_I := \Delta_C\} \cdot y) & \text{if } x \in \text{dom}(\Delta_I, \Delta_C) \text{ or } FV(u) \cap \text{dom}(\Delta_I, \Delta_C) \neq \emptyset, \\ (\text{Ind}_n\{\Delta_I[x\setminus u] := \Delta_C[x\setminus u]\} \cdot y) & \text{otherwise}, \end{cases} \quad (\dagger) \]

\[ (\text{fix } y_i \{y/k : A := t\})[x\setminus u] := \begin{cases} \text{fix } y_i \{y/k : A := t\} & \text{if } x \in \{\bar{y}\} \text{ or } FV(u) \cap \{\bar{y}\} \neq \emptyset, \\ \text{fix } y_i \{y/k : (A[x\setminus u]) := t[x\setminus u]\} & \text{otherwise}. \end{cases} \quad (\dagger) \]

(\star) By using \(\alpha\)-conversion, if needed, \(y\) is assumed to be not free in \(u\) such that the variable condition is satisfied.

(\dagger) The variable condition here implies that the names of inductive types, constructors, and recursive functions are uniquely determined, and that they will never be changed once they are defined. Thus, these names are bound variables that differ from variables bound by \(\Pi\) and \(\lambda\).

APPENDIX B. Constrained typing

Note that not all fix-point definitions can be accepted because of the possibility of non-normalizing terms. If one of the arguments belongs to an inductive type, then the function starts with a case analysis, and recursive calls are performed on variables coming from patterns and representing subterms. This is the usual restriction implemented in Coq when a case distinction with respect to a distinguished inductive type in a definition of a (mutual) recursive function occurs. These restrictions are imposed by the so-called guarded-by-destructors condition defined by [Giménez(1995)]. Here, we follow the simplified version given by [Paulin-Mohring(1996)] by using constrained typing.

The constraints will be imposed with respect to a variable \(z\) and an inductive specification \(\Delta_I, \Delta_C\), and they have three forms: the empty constraint \(\epsilon\), the constraint \(<_z\), which describes the structural smallness with respect to \(z\), and the constraint \(\equiv_z\), which describes the equivalence to \(z\). The constraints will be added to any occurrence of a variable in a term. Let \(c, d, \ldots\) vary over constraints. The judgments of constrained typing have the form
\[
\text{Ind}_\epsilon(\Delta_I := \Delta_C) \in \Gamma \quad \text{(} d_i : \Pi \bar{\nu} : \bar{P} \cdot A) \in \Delta_I \quad \text{lh}(\bar{p}) = n
\]
\[
\Gamma \vdash Q : B \quad \text{C}(d_i \bar{p} : A; B) \quad \Gamma \vdash \epsilon \vdash d_i \bar{p} \bar{\nu}
\]
\[
\Gamma \vdash h_k : \epsilon \cdot \Pi(\bar{\nu} : \bar{V}_k)^\epsilon \cdot Q \bar{w}_k (c_k \bar{p} \bar{v}) \text{ for all } (c_k : \Pi \bar{p} : \bar{P} \cdot \Pi \bar{v} : \bar{V}_k, d_i \bar{p} \bar{w}_k) \in \Delta_C
\]
\[
\begin{align*}
\Gamma \vdash M : s & \quad \mathcal{WF}(\Gamma) \quad x : \epsilon \ A \in \Gamma \quad \Gamma \vdash t : \epsilon \ A \\
\mathcal{WF}(\Gamma, x : \epsilon \ M) & \quad \Gamma \vdash x : \epsilon \ A \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash A : \epsilon \ s_1 & \quad \Gamma, x : \epsilon \ A \vdash B : \epsilon \ s_2 \quad \mathcal{P}(s_1, s_2, s_3) \\
\Gamma \vdash \Pi x : \epsilon \ A.B : \epsilon \ s_3 & \quad \Gamma \vdash A : \epsilon \ s_1, x : \epsilon \ A \vdash B : \epsilon \ s_2 \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash A = A' : \epsilon \ s_1 & \quad \Gamma, x : \epsilon \ A \vdash B = B' : \epsilon \ s_2 \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash A = A' & \quad \Gamma, x : \epsilon \ A \vdash B : \epsilon \ s_2 \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash A = A' & \quad s_1 \quad \Gamma, x : \epsilon \ A \vdash B : \epsilon \ s_2 \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash \lambda x : \epsilon \ A.M : \epsilon \ s_3 & \quad \Pi x : \epsilon \ A.B
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash \lambda x : \epsilon \ A.M = \lambda x : \epsilon \ A'.M' & : \epsilon \ s_3 \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash M : \epsilon \ s_1 & \quad \Pi x : \epsilon \ A.B
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash M = M' & \quad \Pi x : \epsilon \ A.B \\
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash N = N & \quad \text{if } FV(A) \cap \text{dom}(\Delta_I, \Delta_C) = \emptyset,
\end{align*}
\]
\[
\begin{align*}
\Gamma \vdash M.N = M'.N' & \quad \text{if } FV(A) \cap \text{dom}(\Delta_I, \Delta_C) \neq \emptyset.
\end{align*}
\]
\[
\text{Γ} \vdash \text{M} : \epsilon \text{N}, \text{ where the constraints are added to all the variables from dom(Γ). M} \epsilon \text{ and } \Gamma \epsilon \text{ denote the term } M \text{ and the term sequence } \Gamma, \text{ respectively, where only the constraint } \epsilon \text{ is added.}
\]
Given a constraint c, the constraint < c is defined as follows:
\[
< \epsilon := \epsilon, \quad <\epsilon := <\epsilon, \quad <\epsilon := <\epsilon.
\]
The following defines the restriction of a recursive call of inductive type when defining a mutual recursion. Given a declaration Δ, Δ<Δ is defined as follows:
\[
[[[]]<^\epsilon := [[],\quad
(\Delta, x : A)<^\epsilon := \Delta<^\epsilon, x : \epsilon A \quad \text{if } FV(A) \cap \text{dom}(\Delta_I, \Delta_C) = \emptyset,
(\Delta, x : A)<^\epsilon := \Delta<^\epsilon, x :<^\epsilon A \quad \text{if } FV(A) \cap \text{dom}(\Delta_I, \Delta_C) \neq \emptyset.
\]
\]
Γ<Δ is defined similarly for a term sequence Γ. In Figure 4, we list the rules for constrained typing. The omitted rules contain only the empty constraint ε.
Suppose $Γ ⊢ D$, where $D = \text{Ind}_d(Δ_l := Δ_C)$, and $γ ∈ [Γ]$. As mentioned before, we suppress $Γ$ and $γ$ for better readability. Suppose that

$$Δ_l := d_0 : A_0, ..., d_ℓ : A_ℓ, Δ_C := c_1 : T_1, ..., c_m : T_m,$$

$$A_i := Π\vec{p} : _BACKGROUND_{B_i}.s_i, T_k := Π\vec{p} : _BACKGROUND_{Z_k}.d_k, Z_k,j := Π\vec{u}_{k,j} : _BACKGROUND{\vec{H}_{k,j}}, l\vec{t}_{k,j} := Π\vec{v}_{k,j} : _BACKGROUND{\vec{H}_{k,j}},$$

where $lh(\vec{p}) = n, lh(\vec{B}_i) = ℓ_i, lh(T_k) = ℓ_k, j ∈ ν(k) := \{j | FV(Z_{k,j}) ∩ dom(Δ_l) ≠ \emptyset\}$, and $i_k, i_{k,j} ≤ ℓ$. Furthermore, $Z_{k,j}’$ is defined as

$$Z_{k,j}’ := \begin{cases} Z_{k,j} & \text{if } j \notin ν_k, \\ Π\vec{u}_{k,j} : _BACKGROUND{\vec{H}_{k,j}}, l\vec{t}_{ik,j} & \text{if } j \in ν_k, \end{cases}$$

where $d_{ik,j}’$ are fresh variables. Further, we suppose that $[\vec{P}], [\vec{Z}_k]_ρ_k, ...$ are already well defined below in the definition of $Φ$ (This will be the case by induction hypothesis.).

Then, we set $[D] := I(Φ)$, where $Φ :=$

$$\bigcup_{i ≤ ℓ} \bigcup_{k ∈ μ_i} \left\{ \langle i, k, BACKGROUND{\vec{p}, \vec{w}_{ik,j}}, \vec{u} \rangle | \vec{u} ∈ [BACKGROUND{\vec{H}_{k,j}}], \vec{z}_k \in [\vec{P}], [\vec{Z}_k]_ρ_k \rangle \right\}.$$

Here, $μ_i := \{k | d_{ik} = d_i\}$, and $ρ_k$ associates $ν_{r(k,j)}$ with $d_{ik,j}’$, where $r(k, j) := rank(d_{ik,j})$. We also set $[D · d_k] := \text{lam}(f_k)$, where $f_k(\vec{p}, \vec{b}_i) = I(Φ)(i, \vec{p}, \vec{b}_i)$ for $\vec{p}, \vec{b}_i : [\vec{P}, \vec{B}_i]$, and $[D · c_k] := \text{lam}(g_k)$, where $g_k(\vec{p}, \vec{z}_k) = (k, \vec{z}_k)$ for $\vec{p}, \vec{z}_k : [\vec{P}, \vec{Z}_k]$. 

**APPENDIX D. INTERPRETATION OF WELL-FOUNDED STRUCTURED RECURSION**

Below, we use the same notation as that used in Appendix C for the inductive types on which the recursive call is running.

Suppose $Γ ⊢ \text{fix}\ f_ℓ \{R\} : A_j$, where

$$R := f/k : A := t, \quad A_i := Π\vec{x}_i : _BACKGROUND{\vec{B}_i}, A_i’, \quad lh(\vec{B}_i) = k + 1, \quad ℓ ≤ n,$$

$$(Γ ⊢ A_i : s_i)_{i ≤ n}, \quad (Γ, f : _BACKGROUND{\vec{A} ≡ t_i : A_i})_{i ≤ n}, \quad F(\vec{f}, \vec{A}, \vec{f}, \vec{t}).$$

Let $γ ∈ [Γ]$ be given. We suppress $Γ$ and $γ$ for better readability. Then, $[\text{fix}\ f_ℓ \{R\}]$ will depend on the $\iota$-reduction.

Suppose $Γ ⊢ \vec{a}, a_{k+1} : \vec{B}_ℓ$, where $a_{k+1} = T_k \vec{p}\vec{u}_k$, and $B_{ℓ,k+1} = x_i \vec{p}\vec{t}_k$, i.e., $k ∈ μ_i$, and that $\vec{a}, \vec{p}$ are all fresh variables, while $\vec{u}_k$ represents a branch in the tree-like structure. All the free variables occurring in $\vec{u}_k$ should be fresh. Then, $(t_ℓ[f_ℓ(\text{fix}\ f_ℓ \{R\})])\vec{a}(T_k \vec{p}\vec{u}_k)$ $β$-, $\iota$-reduces to the term $M_{ℓ,k}$ which is obtained from the node term of the branch which $T_k \vec{p}\vec{u}_k$ represents.

Suppose that for some $g_ℓ,k, h_ℓ,k ∈ \mathbb{N}$, $u_1', ..., u_{g_ℓ,k}', u_{g_ℓ,k+1}', ..., u_{h_ℓ,k}'$ list all the subterms of $M_{ℓ,k}$ that are structurally smaller than $\vec{u}_k$. Each $u_q'$ with $q ≤ g_ℓ,k$ occurs as the $(k_n + 1)$th argument of $(\text{fix}\ f_{n_q} \{R\})$. Thus,

$$((\text{fix}\ f_{n_q} \{R\}) b_{n_q} u_q')$$
is a subterm of $M_{j,k}$ for some $\bar{b}_{n_q} : B_{n_q,1}, ..., B_{n_q,k_{n_q}}$. Note that each $u'_q$ is an argument of some constructor $T_{m_q} : \bar{P}. \Pi \bar{Z}_{m_q} : x_{i_{m_q}} \bar{P} \bar{r}_{m_q}$ such that the head of $u'_q$ is of some type $Z_{m_q,j_q} = \Pi \bar{u}_{m_q,j_q} \bar{m}_{m_q,j_q} x_{i_{m_q}} \bar{P} \bar{w}_{m_q,j_q}$ and that $x_{i_{m_q,j_q}} = x_{i_{n_q}}$ if $q \leq g_{k,k}$. Furthermore, we suppose that $u'_{q_1}, ..., u'_{q_h}$ are all terms among $u'_1, ..., u'_{h_{l,k}}$, which are headed by some variables.

Thus, $u'_{q_r} = a_{q_r} \bar{u}_{m_q,j_{q_r}}$ for a variable $a_{q_r}$ of type

$$Z_{m_q,j_{q_r}} = \Pi \bar{u}_{m_q,j_{q_r}} \bar{m}_{m_q,j_{q_r}} \bar{P} \bar{w}_{m_q,j_{q_r}}$$

and for some terms $\bar{u}_{m_q,j_{q_r}}$. Note that $a_{q_1}, ..., a_{q_h}$ are exactly the free variables occurring in $\bar{u}_k$. Suppose that $u'_{q_1}, ..., u'_{q_h}$ are all such terms structurally smaller than $u'_q$. Then,

$$u'_q = \lambda \bar{u}_{m_q,j_{q_1}} : \bar{H}_{m_q,j_{q_1}} ... \lambda \bar{u}_{m_q,j_{q_h}} : \bar{H}_{m_q,j_{q_h}} (\forall \bar{u}_{m_q,j_{q_1}} ... \bar{u}_{m_q,j_{q_h}}) \bar{u}_m \bar{u}_{m_q,j_{q_1}} ... \bar{u}_{m_q,j_{q_h}}$$

for some $\bar{u}_m$. Similarly, $M_{l,k}$ can be written as follows:

$$(\lambda \bar{u}_{m_q,j_{q_1}} : \bar{H}_{m_q,j_{q_1}} ... \lambda \bar{u}_{m_q,j_{q_h}} : \bar{H}_{m_q,j_{q_h}} (\forall \bar{u}_{m_q,j_{q_1}} ... \bar{u}_{m_q,j_{q_h}}) \bar{b}_{n_q} \bar{u}'_q) \bar{u}_{m_q,j_{q_1}} ... \bar{u}_{m_q,j_{q_h}}$$

where

$$\bar{u}'_q = \lambda \bar{u}_{m_q,j_{q_1}} : \bar{H}_{m_q,j_{q_1}} ... \lambda \bar{u}_{m_q,j_{q_h}} : \bar{H}_{m_q,j_{q_h}} \bar{u}'_{q_1} ... \bar{u}'_{q_h}$$

Further, set

$$M'_{l,k} = (\lambda \bar{u}_{m_q,j_{q_1}} : \bar{H}_{m_q,j_{q_1}} ... \lambda \bar{u}_{m_q,j_{q_h}} : \bar{H}_{m_q,j_{q_h}} (\forall \bar{u}_{m_q,j_{q_1}} ... \bar{u}_{m_q,j_{q_h}}) \bar{b}_{n_q} \bar{u}'_q)$$

of type $(\lambda \bar{m}_{m_q,j_{q_1}} : \bar{H}_{m_q,j_{q_1}} ... \lambda \bar{m}_{m_q,j_{q_h}} : \bar{H}_{m_q,j_{q_h}} (\forall \bar{m}_{m_q,j_{q_1}} ... \bar{m}_{m_q,j_{q_h}}) \bar{c}_{n_q} \bar{c}'_q)$, where $c$ ranges over $1, ..., q$. Lastly, let $M'_{l,k}$ be obtained from $M_{l,k}$ by replacing $M'_{l,k}$ with a fresh variable $X_{n_q}$.

Then, $\bigcup_{\ell} \llbracket f_{\ell} \{ R \} \rrbracket$ will correspond to the fixpoint of the following rule set:

$$\Psi := \bigcup_{\ell \leq n} \bigcup_{k \in k_{l,k}} \left\{ \begin{array}{l}
\{ \langle a_{\ell_1}, ..., a_{k_1}, \langle k, \bar{u}_k \rangle, \langle \bar{u}_{m_q,j_{q_1}}, ..., \bar{u}_{m_q,j_{q_h}} \rangle \rangle \} \quad | \\
\langle \alpha_1, ..., \alpha_{k_1}, \langle k, \bar{u}_k \rangle, \langle \bar{M} \rangle \rangle \}
\end{array} \right\}
$$

$$\begin{array}{l}
\{ C \} \equiv \{ \bar{u}_{m_q,j_{q_1}} \in \bar{H}_{m_q,j_{q_1}} ... \bar{u}_{m_q,j_{q_h}} \in \bar{H}_{m_q,j_{q_h}} \}, \\
\bar{a} \in \{ B_{l,1}, ..., B_{l,k_1} \}, \\
v_{q_r} \in \{ \Pi \bar{u}_{m_q,j_{q_r}} : \bar{H}_{m_q,j_{q_r}} \bar{P} \bar{w}_{m_q,j_{q_r}} \}, \\
r \in \{ 1, ..., h \}, \\
v'_{q_r} \in \{ \Pi \bar{u}_{m_q,j_{q_r}} : \bar{H}_{m_q,j_{q_r}} \bar{c}_{n_q} \bar{c}'_{q_r} \}, \\
\eta \in \{ \bar{a}, \bar{v}_{q_r} \}, \\
r \in \{ 1, ..., h \}, \end{array}
$$

We set $\llbracket f_{\ell} \{ R \} \rrbracket := \bar{m}(h)$, where $h$ is a function such that

$$h(a_1, ..., a_{k_1}, \langle k, \bar{z}_k \rangle) = \mathcal{T} \{ \Psi \}(a_1, ..., a_{k_1}, \langle k, \bar{z}_k \rangle)$$

where $\bar{a}, \langle k, \bar{z}_k \rangle \in \bar{B}_l$. 

---

This work is licensed under the Creative Commons Attribution-NoDerivs License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/2.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany.