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

GYESIK LEE [a] AND BENJAMIN WERNER [b]

[a] Hankyong National University, Anseong-si, Kyonggi-do, Korea
   e-mail address: gslee@hknu.ac.kr

[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 *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. [Aczel(1998)] and [Coquand(1990)]). 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, λ-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. [Aczel(1998), Dybjer(1991), Dybjer(2000), Werner(1997)]). However, a higher-order generalization of the trivial Boolean model is *not so simple* (cf. [Miquel and Werner(2003)]). The main cause of this problem, as identified by [Reynolds(1984)], 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 $\mathtt{Prop}$ of propositions. This is followed by a hierarchy of predicative type universes $\mathtt{Type}_i$, $i = 0, 1, 2, \ldots$:

- $\mathtt{Prop}$ is of type $\mathtt{Type}_0$;
- $\mathtt{Type}_i$ is of type $\mathtt{Type}_{i+1}$;
- $\mathtt{Prop} \prec \mathtt{Type}_0 \prec \mathtt{Type}_1 \prec \cdots$.

Werner's system, however, does not include the subtyping rule $\mathtt{Prop} \prec \mathtt{Type}_0$, 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 $\mathtt{Prop} \prec \mathtt{Type}_0$, 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:

$$\Gamma \vdash M = N : A \qquad \text{and} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash A = B}{\Gamma \vdash M : B} \; (conv)$$

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 ($\mathtt{Prop}$) of propositions, a cumulative hierarchy of predicative universes ($\mathtt{Type}_i$), predicative inductions, and judgmental equality.

The main difficulty in the construction of a set-theoretic model of our system stems from the impredicativity of $\mathtt{Prop}$ and the subtyping property $\mathtt{Prop} \prec \mathtt{Type}_0$. 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 $\mathtt{Prop}$ and the predicative types $\mathtt{Type}_i$, whereas the subsumption eliminates the difference. An interpretation function $f : \{0, 1\} \to \mathcal{V}$ is required, where $\mathcal{V}$ 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 κ 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.[1]

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 $\Pi(\alpha : \mathtt{Prop}).\alpha$ 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 $x, x_i, X, X_i, ...$ vary over the variables. We also use special constants $\mathtt{Prop}$ and $\mathtt{Type}_i$, $i \in \mathbb{N}$. They are called *sorts*. Sorts are usually denoted by s, s_i, etc.[2]

---

[1]We 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)]).

[2]In this paper, we do not consider the sort $\mathtt{Set}$. Indeed, when (the impredicative or predicative sort) $\mathtt{Set}$ 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 $\mathtt{Set}$ and $\mathtt{Prop}$, as identified by [Reynolds(1984)]. Note, however, that $\mathtt{Type}_0$ in our system plays the role of the predicative $\mathtt{Set}$.

Definition 2.1 (Terms and contexts)**.** The syntax of the objects is given as follows.

$$t, t', t_i, A, A_i \quad ::= \quad x \mid s \mid \Pi x : t.t' \mid \lambda x : t.t' \mid \mathsf{let}\ x := t\ \mathsf{in}\ t' \mid t\,t' \qquad (terms)$$
$$\mid \mathsf{case}(t, t', \vec{t}) \mid \mathsf{Ind}_n\{\Delta := \Delta'\} \cdot x$$
$$\mid \mathsf{fix}\ x_i\ \{x_0/k_0 : A_0 := t_0, \ldots, x_n/k_n : A_n := t_n\}$$

$$\Delta, \Delta' \quad ::= \quad [\,] \mid \Delta, (x : t) \qquad\qquad\qquad\qquad\qquad\qquad (declarations)$$

$$\Gamma, \Gamma' \quad ::= \quad [\,] \mid \Gamma, (x : t) \mid \Gamma, (x := t : t') \mid \Gamma, \mathsf{Ind}_n\{\Delta := \Delta'\} \quad (contexts)$$

Here, $[\,]$ denotes the empty sequence.

Definition 2.2 (Atomic terms)**.** *Atomic terms* are either variables, sorts, or terms of the form $\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \cdot x$.

Definition 2.3 (Domain of contexts)**.** The domain of a context is defined as follows:

$$\mathrm{dom}([\,]) := \varnothing, \qquad \mathrm{dom}(\Gamma, \mathsf{Ind}_n\{\Delta_I := \Delta_C\}) := \mathrm{dom}(\Gamma),$$
$$\mathrm{dom}(\Gamma, x = t : A) := \mathrm{dom}(\Gamma, x : A) := \mathrm{dom}(\Gamma) \cup \{x\}.$$

Remark 2.4.

(1) Vector notations are used instead of some sequences of expressions:

> - $\vec{t} := t_1, ..., t_n$ - $f\,\vec{t} := f\,t_1 \cdots t_n$
> - $\Pi\vec{x} : \vec{A}.t := \Pi x_1 : A_1. ... \Pi x_n : A_n. t$ - $\lambda\vec{x} : \vec{A}.t := \lambda x_1 : A_1. ... \lambda x_n : A_n. t$
> - $\overrightarrow{x/k : A := t} := x_0/k_0 : A_0 := t_0, ..., x_n/k_n : A_n := t_n$

(2) Note that we use two subscript styles. One is of the form $t_1, ..., t_n$, and the other is of the form $t_0, ..., t_n$, where n is a natural number. The latter style will be used only in the definition of mutually recursive functions, i.e., in combination with fix.

(3) Given a sequence $\vec{\ell}$, let $lh(\vec{\ell})$ 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 Δ and a variable x, let $\Delta(x) = A$ when A is the only term such that $x : A$ occurs in Δ.

(6) There are standard definitions of the sets of free variables in a context or a term, and of the substitution $t[x \backslash u]$, where t, u are terms and x a variable. Formal definitions are given in Appendix A.

(7) Given a sequence $\delta = x_1 : t_1, ..., x_n : t_n$ and a term t, let $t\{\delta\} := t[x_1 \backslash t_1] \cdots [x_n \backslash t_n]$ denote consecutive substitution. On the other hand, the simultaneous substitution of terms $t_1, ..., t_n$ for $x_1, ..., x_n$, respectively, in t is denoted by $t[\delta] := t[x_1 \backslash t_1, ..., x_n \backslash t_n]$.

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 $\mathsf{Ind}_n\{\Delta_I := \Delta_C\}$ denotes a *(mutually) inductive type*, and the subscript n denotes the number of *parameters*. Δ_I and Δ_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 $\mathcal{D}_{TF} = \mathsf{Ind}_1\{\Delta_I := \Delta_C\}$, where

$$\begin{aligned}
\Delta_I &= \mathtt{tree} : \mathtt{Type}_0 \to \mathtt{Type}_0, \mathtt{forest} : \mathtt{Type}_0 \to \mathtt{Type}_0\,, \\
\Delta_C &= \mathtt{node} : \Pi(A : \mathtt{Type}_0).\, A \to \mathtt{forest}\, A \to \mathtt{tree}\, A, \\
&\quad\ \mathtt{emptyf} : \Pi(A : \mathtt{Type}_0).\, \mathtt{forest}\, A, \\
&\quad\ \mathtt{consf} : \Pi(A : \mathtt{Type}_0).\, \mathtt{tree}\, A \to \mathtt{forest}\, A \to \mathtt{forest}\, A\,.
\end{aligned}$$

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

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

The type for natural numbers and its two constructors can be represented by $\mathcal{D}_N \cdot \mathtt{nat}$, $\quad \mathcal{D}_N \cdot \mathtt{0}$, and $\mathcal{D}_N \cdot \mathtt{S}$, respectively, where $\mathcal{D}_N = \mathsf{Ind}_0\{\Delta_I := \Delta_C\}, \Delta_I = \mathtt{nat} : \mathtt{Type}_0$, and $\Delta_C = \mathtt{0} : \mathtt{nat}, \mathtt{S} : \mathtt{nat} \to \mathtt{nat}$.

In the examples presented below, however, we use character strings for better readability. Thus, for example, $\mathtt{nat}$, $\mathtt{0}$, and $\mathtt{S}$ are used instead of $\mathcal{D}_N \cdot \mathtt{nat}, \mathcal{D}_N \cdot \mathtt{0}$, and $\mathcal{D}_N \cdot \mathtt{S}$, respectively.

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

$$\mathtt{match}\ e\ \mathtt{as}\ y\ \mathtt{in}\ I\ \vec{\_}\, \vec{u}\ \mathtt{return}\ Q\, \vec{u}\, y\ \mathtt{with}\ \cdots \mid C_i\, \vec{p}\, \vec{v}\ \mathtt{=>}\ h_i \mid \ldots\ \mathtt{end}$$

where
- the term e is of an inductive type $I\, \vec{p}\, \vec{u}$ for some terms $\vec{p}, \vec{u}$,
- $lh(\vec{p}) = lh(\vec{\_})$,
- $Q = \lambda \vec{u} : \vec{U}.\, \lambda y : I\, \vec{p}\, \vec{u}.\, Q'$ 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 $\Pi \vec{p} : \vec{P}.\Pi \vec{v} : \vec{V_i}.I\, \vec{p}\, \vec{w_i}$ for some terms $\vec{P}, \vec{V_i}, \vec{w_i}$, and
- each $h_i = \lambda \vec{v} : \vec{V_i}.h_i'$ for some term h_i'.

The term $\mathsf{fix}\ f_i\, \overrightarrow{\{f/k : A := t\}}$ 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 $\mathtt{struct}$ annotation used for the "*guarded*" condition in the termination check (cf. [Giménez(1995)]).

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

$$\mathtt{plus} = \mathsf{fix}\ f\, \{f/1 : \Pi(m, n : \mathtt{nat}).\, \mathtt{nat} := \lambda(m, n : \mathtt{nat}).\, \mathsf{case}(n, Q, h_0, h_1)\}\,,$$

where $Q = \lambda(\ell : \mathtt{nat}).\, \mathtt{nat}$, $h_0 = m$, and $h_1 = \lambda(p : \mathtt{nat}).\, \mathtt{S}\, (f\, m\, p)$.

(2) The functions for measuring the size of trees and forests can be represented by $\mathtt{Tsize} = \mathsf{fix}\, g_0\,\{R\}$ and $\mathtt{Fsize} = \mathsf{fix}\, g_1\,\{R\}$, where $R = \overrightarrow{g/k : B := t}$, $k_0 = k_1 = 1$, and

$$
\begin{aligned}
B_0 &= \Pi(A : \mathtt{Type}_0).\,\Pi(t : \mathtt{tree}\,A).\,\mathtt{nat}\,, \\
B_1 &= \Pi(A : \mathtt{Type}_0).\,\Pi(f : \mathtt{forest}\,A).\,\mathtt{nat}\,, \\[4pt]
t_0 &= \lambda(A : \mathtt{Type}_0).\,\lambda(t : \mathtt{tree}\,A).\,\mathsf{case}(t, Q_0, h_0)\,, \\
t_1 &= \lambda(A : \mathtt{Type}_0).\,\lambda(f : \mathtt{forest}\,A).\,\mathsf{case}(f, Q_1, h_1, h_2)\,, \\
Q_0 &= \lambda(t : \mathtt{tree}\,A).\,\mathtt{nat}\,, \\
Q_1 &= \lambda(f : \mathtt{forest}\,A).\,\mathtt{nat}\,, \\
h_0 &= \lambda(a : A).\lambda(f : \mathtt{forest}\,A).\,\mathtt{S}\,(g_1\,A\,f)\,, \\
h_1 &= \mathtt{O}\,, \\
h_2 &= \lambda(t : \mathtt{tree}\ A).\,\lambda(f : \mathtt{forest}\ A).\,\mathtt{plus}\,(g_0\,A\,t)\,(g_1\,A\,f).
\end{aligned}
$$

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

Typing rules for basic terms and valid contexts (Figure 1). Typing rules for standard constructions of λ- and Π-terms are given.

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

(Π) and $(\Pi\text{-}eq)$: $\mathcal{P}(s_1, s_2, s_3)$ implies that
- $s_2 = s_3 = \mathtt{Prop}$, or
- $s_1 \in \mathtt{Type}_i$, $s_2 = \mathtt{Type}_j$ and $s_3 = \mathtt{Type}_k$ 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.

$(ind\text{-}wf)$: The positivity condition is crucial for defining an inductive type. A term A is an *arity ending in sort s*, $\mathsf{Arity}(A, s)$, 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*, $\mathsf{Arity}(A)$, 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 $M = \Pi \vec{y} : \vec{A}.\,x\,\vec{u}$ for some terms $\vec{A}, \vec{u}$ and the variable x occurs *strictly positively* in $\vec{A}$. A variable x occurs *strictly positively* in M when
- x does not occur in M, or
- $M \equiv \Pi \vec{y} : \vec{A}.\,(x\,\vec{B})$ and x does not occur in $\vec{A}, \vec{B}$.

Now, $\mathcal{I}_n(\Delta_I, \Delta_C)$ represents the following conditions:
- All the names contained in the domains of Δ_I and Δ_C must be mutually distinct and new.

$$\mathcal{WF}(\varnothing) \qquad \frac{\Gamma \vdash A : s \quad x \notin \mathrm{dom}(\Gamma)}{\mathcal{WF}(\Gamma, (x : A))} \qquad \frac{\Gamma \vdash t : A \quad x \notin \mathrm{dom}(\Gamma)}{\mathcal{WF}(\Gamma, (x := t : A))} \qquad (wf)$$

$$\frac{\mathcal{WF}(\Gamma)}{\Gamma \vdash \mathtt{Prop} : \mathtt{Type}_i} \qquad \frac{\mathcal{WF}(\Gamma) \quad i < j}{\Gamma \vdash \mathtt{Type}_i : \mathtt{Type}_j} \qquad (ax)$$

$$\frac{\mathcal{WF}(\Gamma) \quad (x : A) \in \Gamma \quad \mathrm{or} \quad (x := t : A) \in \Gamma}{\Gamma \vdash x : A} \qquad (var)$$

$$\frac{\Gamma, (x := t : A) \vdash u : U}{\Gamma \vdash \mathsf{let}\ x := t\ \mathsf{in}\ u : U[x\backslash t]} \qquad (let)$$

$$\frac{\Gamma \vdash t = t' : A \quad \Gamma, (x := t : A) \vdash u = u' : U}{\Gamma \vdash (\mathsf{let}\ x := t\ \mathsf{in}\ u) = (\mathsf{let}\ x := t'\ \mathsf{in}\ u') : U[x\backslash t]} \qquad (let\text{-}eq)$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \mathcal{P}(s_1, s_2, s_3)}{\Gamma \vdash \Pi x : A.B : s_3} \qquad (\Pi)$$

$$\frac{\Gamma \vdash A = A' : s_1 \quad \Gamma, x : A \vdash B = B' : s_2 \quad \mathcal{P}(s_1, s_2, s_3)}{\Gamma \vdash \Pi x : A.B = \Pi x : A'.B' : s_3} \qquad (\Pi\text{-}eq)$$

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash \Pi x : A.B : s}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B} \qquad (\lambda)$$

$$\frac{\Gamma \vdash A = A' : s \quad \Gamma, x : A \vdash M = M' : B \quad \Gamma \vdash \Pi x : A.B : s'}{\Gamma \vdash \lambda x : A.M = \lambda x : A'.M' : \Pi x : A.B} \qquad (\lambda\text{-}eq)$$

$$\frac{\Gamma \vdash M : \Pi x : A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x\backslash N]} \qquad (app)$$

$$\frac{\Gamma \vdash M = M' : \Pi x : A.B \quad \Gamma \vdash N = N' : A}{\Gamma \vdash MN = M'N' : B[x\backslash N]} \qquad (app\text{-}eq)$$

Figure 1: Basic terms and valid contexts

- All the types of Δ_I and Δ_C start with the same n products, say, $\vec{p} : \vec{P}$.
- Any occurrence of some $d \in \mathrm{dom}(\Delta_I)$ in Δ_C is of the form $(d\,\vec{p}\,\vec{u})$, which is not applicable any more.
- For all $d : A \in \Delta_I$, A is an arity ending in sort s_d such that $s_d \neq \mathtt{Prop}$. Thus, we do not use inductive definitions of type $\mathtt{Prop}$. 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 $d \in \mathrm{dom}(\Delta_I)$, i.e., T is of the form $\Pi\vec{p} : \vec{P}. \Pi\vec{z} : \vec{Z}. (d\,\vec{p}\,\vec{u})$. 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 $x \in \mathrm{dom}(\Delta_I)$.

Notation. We use $\Gamma \vdash \mathsf{Ind}_n\{\Delta_I := \Delta_C\}$ when all the premises of $(ind\text{-}wf)$ are satisfied.

$$\mathcal{I}_n(\Delta_I, \Delta_C)$$
$$\frac{\Gamma \vdash A : s_d \quad \text{for all } (d : A) \in \Delta_I}{\Gamma, \Delta_I \vdash T : s_c \quad \text{for all } (c : T) \in \Delta_C}$$
$$\frac{}{\mathcal{WF}(\Gamma, \mathsf{Ind}_n\{\Delta_I := \Delta_C\})} \qquad (ind\text{-}wf)$$

$$\frac{\mathcal{WF}(\Gamma) \quad \mathcal{D} = \mathsf{Ind}_n\{\Delta_I := \Delta_C\} \in \Gamma \quad d \in \mathrm{dom}(\Delta_I)}{\Gamma \vdash \mathcal{D} \cdot d : \Delta_I(d)[.\backslash \mathcal{D}]} \qquad (ind\text{-}type)$$

$$\frac{\mathcal{WF}(\Gamma) \quad \mathcal{D} = \mathsf{Ind}_n\{\Delta_I := \Delta_C\} \in \Gamma \quad c \in \mathrm{dom}(\Delta_C)}{\Gamma \vdash \mathcal{D} \cdot c : \Delta_C(c)[.\backslash \mathcal{D}]} \qquad (ind\text{-}const)$$

$$\frac{\begin{array}{c}\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \in \Gamma \quad (d_i : \Pi\vec{p} : \vec{P}. A) \in \Delta_I \quad lh(\vec{p}) = n \\ \Gamma \vdash Q : B \quad \mathcal{C}(d_i\,\vec{p} : A; B) \quad \Gamma \vdash e : d_i\,\vec{p}\,\vec{u} \\ \Gamma \vdash h_k : \Pi\vec{v} : \vec{V}_k.\, Q\,\vec{w}_k\,(c_k\,\vec{p}\,\vec{v}) \text{ for all } (c_k : \Pi\vec{p} : \vec{P}.\,\Pi\vec{v} : \vec{V}_k.\, d_i\,\vec{p}\,\vec{w}_k) \in \Delta_C\end{array}}{\Gamma \vdash \mathsf{case}(e, Q, (h_k)_k) : Q\,\vec{u}\,e} \qquad (case)$$

$$\frac{\begin{array}{c}\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \in \Gamma \quad (d_i : \Pi\vec{p} : \vec{P}. A) \in \Delta_I \quad lh(\vec{p}) = n \\ \Gamma \vdash Q = Q' : B \quad \mathcal{C}(d_i\,\vec{p} : A; B) \quad \Gamma \vdash e = e' : d_i\,\vec{p}\,\vec{u} \\ \Gamma \vdash h_k = h'_k : \Pi\vec{v} : \vec{V}_k.\, Q\,\vec{w}_k\,(c_k\,\vec{p}\,\vec{v}) \text{ for all } (c_k : \Pi\vec{p} : \vec{P}.\,\Pi\vec{v} : \vec{V}_k.\, d_i\,\vec{p}\,\vec{w}_k) \in \Delta_C\end{array}}{\Gamma \vdash \mathsf{case}(e, Q, (h_k)_k) = \mathsf{case}(e', Q', (h'_k)_k) : Q\,\vec{u}\,e} \qquad (case\text{-}eq)$$

$$\frac{\mathcal{F}(\vec{f}, \vec{A}, \vec{k}, \vec{t}) \quad n = lh(\vec{k}) \quad (\Gamma \vdash A_i : s_i)_{\forall i \leq n} \quad (\Gamma, \vec{f} : \vec{A} \vdash t_i : A_i)_{\forall i \leq n} \quad j \leq n}{\Gamma \vdash \mathsf{fix}\ f_j\,\overrightarrow{\{f/k : A := t\}} : A_j} \qquad (fix)$$

$$\frac{\begin{array}{c}\mathcal{F}(\vec{f}, \vec{A}, \vec{k}, \vec{t}) \quad \mathcal{F}(\vec{f}, \vec{A}', \vec{k}, \vec{t}') \quad n = lh(\vec{k}) \\ (\Gamma \vdash A_i = A'_i : s_i)_{\forall i \leq n} \quad (\Gamma, \vec{f} : \vec{A} \vdash t_i = t'_i : A_i)_{\forall i \leq n} \quad j \leq n\end{array}}{\Gamma \vdash \mathsf{fix}\ f_j\,\overrightarrow{\{f/k : A := t\}} = \mathsf{fix}\ f_j\,\overrightarrow{\{f/k : A' := t'\}} : A_j} \qquad (fix\text{-}eq)$$

Figure 2: Inductive types and recursive functions

(ind-$type$) and (ind-$const$): Given $\mathcal{D} = \mathsf{Ind}_n\{\Delta_I := \Delta_C\}$ and a term A, $A[.\backslash \mathcal{D}]$ implies that every occurrence of $z \in \mathrm{dom}(\Delta_I, \Delta_C)$ in A is replaced with $\mathcal{D} \cdot z$.

($case$) and ($case$-eq): d_i and c_k denote $\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \cdot d_i$ and $\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \cdot c_k$, respectively. Furthermore, $lh(\vec{u}) = lh(\vec{w}_k)$.

For an inductive type d and an arity B, the relation $\mathcal{C}(d\,\vec{q} : A; B)$ is defined as follows:
- $\mathcal{C}(d\,\vec{q} : \mathtt{Prop}; d\,\vec{q} \to \mathtt{Prop})$;
- $\mathcal{C}(d\,\vec{q} : \mathtt{Prop}; d\,\vec{q} \to \mathtt{Type}_j)$ iff d is an inductive type that is empty or has only one constructor[3] such that all the non-parametric arguments are of sort $\mathtt{Prop}$;
- $\mathcal{C}(d\,\vec{q} : \mathtt{Type}_j; d\,\vec{q} \to s)$ for any sort s;
- $\mathcal{C}(d\,\vec{q} : (\Pi u : U . A); (\Pi u : U . B))$ iff $\mathcal{C}(d\,\vec{q}\,u : A; B)$.

---

[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)].

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M = M : A} \qquad \frac{\Gamma \vdash M = N : A}{\Gamma \vdash N = M : A} \qquad (ref)(sym)$$

$$\frac{\Gamma \vdash M = N : A \quad \Gamma \vdash N = P : A}{\Gamma \vdash M = P : A} \qquad (trans)$$

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

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash \Pi x : A.B : s \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x : A.M)N = M[x\backslash N] : B[x\backslash N]} \qquad (\beta)$$

$$\frac{\mathcal{WF}(\Gamma) \quad (x := t : A) \in \Gamma}{\Gamma \vdash x = t : A} \qquad (\delta)$$

$$\frac{\Gamma \vdash t : A \quad \Gamma, (x := t : A) \vdash u : U}{\Gamma \vdash (\mathsf{let}\ x := t\ \mathsf{in}\ u) = u[x\backslash t] : U[x\backslash t]} \qquad (\zeta)$$

$$\frac{\begin{array}{c} \mathsf{Ind}_n\{\Delta_I := \Delta_C\} \in \Gamma \quad (d_i : \Pi\vec{p} : \vec{P}.\,A) \in \Delta_I \quad lh(\vec{p}) = n \\ \Gamma \vdash Q : B \quad \mathcal{C}(d_i\,\vec{p} : A; B) \quad \Gamma \vdash c_j\,\vec{p}\vec{a} : d_i\,\vec{p}\vec{u} \\ \Gamma \vdash h_k : \Pi\vec{v} : \vec{V}_k.\,Q\,\vec{w}_k\,(c_k\,\vec{p}\vec{v})\ \text{for all}\ (c_k : \Pi\vec{p} : \vec{P}.\,\Pi\vec{v} : \vec{V}_k.\,d_i\,\vec{p}\vec{w}_k) \in \Delta_C \end{array}}{\Gamma \vdash \mathsf{case}(c_j\,\vec{p}\vec{a}, Q, (h_k)_k) = h_j\,\vec{a} : Q\,\vec{u}\,(c_j\,\vec{p}\vec{a})} \qquad (\iota)$$

$$\frac{\begin{array}{c} (\Gamma \vdash A_i : s_i)_{\forall i \leq n} \quad (\Gamma, \vec{f} : \vec{A} \vdash t_i : A_i)_{\forall i \leq n} \quad \mathcal{F}(\vec{f}, \vec{A}, \vec{k}, \vec{t}) \quad j \leq n \\ R \equiv \overrightarrow{f/k : A := t} \quad A_j \equiv \Pi\vec{x}_j : \vec{B}_j.\,A_j' \quad \Gamma \vdash \vec{a} : \vec{B}_j \quad lh(\vec{B}_j) = k_j + 1 \end{array}}{\Gamma \vdash (\mathsf{fix}\ f_j\,\{R\})\,\vec{a} = (t_j[f_i\backslash(\mathsf{fix}\ f_i\,\{R\})]\,\vec{a}) : A_j'\{\vec{x} : \vec{a}\}} \qquad (\iota)$$

$$\vdash \mathtt{Prop} \prec \mathtt{Type}_0 \qquad\qquad \vdash \mathtt{Type}_j \prec \mathtt{Type}_{j+1} \qquad (inc)$$

$$\frac{\Gamma \vdash M \prec N \quad \Gamma \vdash N \prec P}{\Gamma \vdash M \prec P} \qquad (trans\text{-}inc)$$

$$\frac{\Gamma \vdash A \prec B \quad \Gamma \vdash C : s \quad x \notin \mathrm{dom}(\Gamma)}{\Gamma, x : C \vdash A \prec B} \qquad (weak\text{-}inc)$$

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

$$\frac{\Gamma \vdash M = N : s \quad \Gamma \vdash M \prec P}{\Gamma \vdash N \prec P} \qquad \frac{\Gamma \vdash M = N : s \quad \Gamma \vdash P \prec M}{\Gamma \vdash P \prec N} \qquad (eq\text{-}inc)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A \prec B}{\Gamma \vdash M : B} \qquad \frac{\Gamma \vdash M = N : A \quad \Gamma \vdash A \prec B}{\Gamma \vdash M = N : B} \qquad (cum)(cum\text{-}eq)$$

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 $\mathcal{C}(d\,\vec{q}; B)$ denote $\mathcal{C}(d\,\vec{q} : A; B)$, where A is the type of $d\,\vec{q}$.

(fix) and $(fix\text{-}eq)$: $\mathcal{F}(\vec{f}, \vec{A}, \vec{k}, \vec{t})$ represents the following conditions:

- $lh(\vec{f}) = lh(\vec{A}) = lh(\vec{k}) = lh(\vec{t})$,
- for each $t_i \in \vec{t}$, there is an inductive type $\mathcal{D} \cdot d$, where $\mathcal{D} = \mathsf{Ind}_n\{\Delta_I := \Delta_C\}$, and a term T_i such that
  - $t_i = \lambda \vec{y} : \vec{Y}. \lambda z : (\mathcal{D} \cdot d)\,\vec{u}.\,t'_i$, where $lh(\vec{Y}) = k_i$, and
  - there is a *constrained derivation* with respect to z and Δ_I, Δ_C such that

$$\Gamma^\epsilon, (\vec{y} : \vec{Y})^\epsilon, z :^{=z} (\mathcal{D} \cdot d)\,\vec{u}, (\vec{f} : \vec{A})_{\vec{k}}^{\leq z} \vdash t'_i :^\epsilon T_i^\epsilon$$

  where $(\vec{f} : \vec{A})_{\vec{k}}^{\leq z}$ is the context composed of

$$f_j :^\epsilon \Pi(\vec{u} : \vec{B}_j)^\epsilon.\Pi v :^{\leq z} X_j\,.\,P_j^\epsilon$$

  if $f_j : \Pi(\vec{u} : \vec{B}_j).\Pi v : X_j\,.\,P_j$ is from $\vec{f} : \vec{A}$ 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: $\Gamma \vdash \vec{t} : \vec{A}$ with $lh(t) = n$ 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
- $\Gamma \vdash t_j : A_j[x_1 \backslash t_1] \cdots [x_{j-1} \backslash t_{j-1}]$ 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 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 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 Prop is associated with

$\{\varnothing, \{\bullet\}\}$. Furthermore, the application and λ-abstraction terms are interpreted by app and lam, respectively, which are defined as follows:

$$\mathsf{app}(u,x) \quad := \quad \begin{cases} \bullet & \text{if } u = \bullet, \\ u(x) & \text{otherwise.} \end{cases}$$

$$\mathsf{lam}(f) \quad := \quad \begin{cases} \bullet & \text{if } f(x) = \bullet \text{ for all } x \in \mathrm{dom}(f), \\ f & \text{otherwise.} \end{cases}$$

Remark 3.1. This construction does not correctly model the cumulative relation between $\mathtt{Prop}$ and $\mathtt{Type}_i$, 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 $\mathtt{Prop}$ and $\mathtt{Type}_i$ does not exist

Consider $I = \lambda(A : \mathtt{Type}_0). A \to A$. 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, $\mathtt{Prop}$ or $\mathtt{Type}_0$. In the former case, $[\![I\,P]\!] = \{\bullet\}$ since $P \to P$ is a tautology, whereas in the latter case, $[\![I\,P]\!] = [\![I]\!]([\![P]\!]) = \{f \,|\, f : \{\bullet\} \to \{\bullet\}\} \neq \{\bullet\}$ 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 $\mathtt{Prop}$ of propositions and its relationship with $\mathtt{Type}_i$. For this reason, we adopt Aczel's solution.

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

$$\mathsf{app}(u,x) \quad := \quad \{z \,|\, (x,z) \in u\},$$
$$\mathsf{lam}(f) \quad := \quad \bigcup_{(x,y)\in f} (\{x\} \times y) \quad = \quad \bigcup_{(x,y)\in f} \{(x,z) \,|\, z \in y\}.$$

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

$$\mathsf{app}(\mathsf{lam}(f),x) = \{y \,|\, (x,y) \in \mathsf{lam}(f)\} = \{y \,|\, y \in f(x)\} = f(x).$$

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 $\mathrm{dom}(f) = A$ and $f(x) \in B(x)$ for all $x \in A$.

(3) Given a function $f \in \prod_{x_1 \in A_1} \cdots \prod_{x_n \in A_n(x_1,...,x_{n-1})} B(x_1,...,x_n)$, we use the notation $\vec{\mathsf{lam}}_n(f)$ (resp. and $\vec{\mathsf{app}}(f, \vec{x})$) for the n-times application of lam (resp. app):

$$\vec{\mathsf{lam}}_n(f) \quad := \quad \{(x_1,...,x_n,y) \,|\, x_1 \in A_1, ..., x_n \in A_n(x_1,...,x_{n-1}), y \in f(x_1,...,x_n)\}$$
$$\vec{\mathsf{app}}_n(f, \vec{x}) \quad := \quad \mathsf{app}(...(\mathsf{app}(\mathsf{app}(f, x_1), x_2), ...), x_n)$$

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

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

(1) $\{\mathsf{lam}(f) \,|\, f \in \prod_{x \in A} B(x)\} \subseteq 1$.

(2) $\{\mathsf{lam}(f) \,|\, f \in \prod_{x \in A} B(x)\} = 1$ *iff* $\forall x \in A\,(B(x) = 1)$.

Proof. Let $f \in \prod_{x \in A} B(x)$, i.e.,

$$f = \begin{cases} \uparrow & \text{if } \exists x \in A \, (B(x) = \varnothing), \\ \{(x, \varnothing) \mid x \in A\} & \text{otherwise.} \end{cases}$$

Then, we have

$$\mathsf{lam}(f) = \begin{cases} \uparrow & \text{if } \exists x \in A \, (B(x) = \varnothing), \\ \varnothing & \text{otherwise.} \end{cases}$$

This also implies that $\{\mathsf{lam}(f) \mid f \in \prod_{x \in A} B(x)\} = 1$ iff $\forall x \in A \, (B(x) = 1)$. $\qquad\square$

Remark 3.4. A useful feature of trace encoding is that $\mathsf{app}(u, a)$ and $\mathsf{lam}(f)$ 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 $\mathrm{dom}(f)$ starting from $\mathsf{lam}(f)$. 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 κ 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.

   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 $\mathtt{Type}_i$ with its $rank(\mathtt{Type}_i) := \kappa_i$. If $(\mathcal{V}_\alpha)_{\alpha \in Ord}$ denotes the (standard) universe of sets defined as follows, then $\mathcal{V}_\kappa$ is a model of ZF:

$$\mathcal{V}_0 := \varnothing \quad \text{and} \quad \mathcal{V}_\alpha := \bigcup_{\beta \in \alpha} \mathcal{P}(\mathcal{V}_\beta) \quad \text{if } \alpha > 0$$

Ord denotes the class of all ordinals, λ denotes a limit ordinal, and $\mathcal{P}$ denotes the power set operator. In particular, if κ is an inaccessible cardinal, $A \in \mathcal{V}_\kappa$, and for every $a \in A$, $B_a \in \mathcal{V}_\kappa$, then, $\prod_{a \in A} B_a \in \mathcal{V}_\kappa$. Let $rank(\mathtt{Prop}) := -1$ and $\mathcal{V}_{-1} = \{0, 1\}$ 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 Φ on U, a set $w \subseteq U$ is Φ-closed if for any $\frac{u}{v} \in \Phi$, $v \in w$ whenever $u \subseteq w$. Note that there is the least Φ-closed set

$$\mathcal{I}(\Phi) := \bigcap \{w \subseteq U \mid w \ \Phi\text{-closed}\} .$$

In fact, it is well known that each rule set Φ on U generates a monotone operator on $\mathcal{P}(U)$

$$\Gamma_\Phi(X) := \{v \in U \mid \text{there exists some } u \subseteq X \text{ such that } \frac{u}{v} \in \Phi\}$$

such that $\mathcal{I}(\Phi)$ is the least fixed point of Γ_Φ. Assuming that Φ is a rule set on $I \times U$, Φ defines a family $\mathcal{IF}(\Phi)$ of sets in U over I as

$$\mathcal{IF}(\Phi)(i) := \{u \in U \mid \langle i, u \rangle \in \mathcal{I}(\Phi)\}$$

for each $i \in I$.

A rule set is *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-theoretic inductively defined set, which is the fixpoint of a monotone operator. Refer to [Aczel(1977)] and [Moschovakis(1974), 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 $\mathsf{Ind}_n\{\Delta_I := \Delta_C\}$, where

$$\Delta_I = x_0 : A_1, ..., x_\ell : A_\ell \quad \text{and} \quad A_i = \Pi\vec{p} : \vec{P}. \, \Pi\vec{a}_i : \vec{B}_i. \, s_i \,,$$

let $rank(x_i) := rank(s_i)$.

Notations.
(1) With each context Γ, we associate a set $[\![\Gamma]\!]$ of Γ-*valuations* of the form $\langle \alpha_1, \cdots, \alpha_n \rangle$, where n is the length of Γ and $\langle, ..., \rangle$ denotes a sequence of a finite length. Given a sequence $L = \langle \alpha_1, \cdots, \alpha_n \rangle$ and a natural number $i < n$, we set $(L)_i = \alpha_{i+1}$. If α_{i+1} 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 γ and δ, the notation γ, δ denotes their concatenation. If $\delta = \langle \alpha \rangle$, then we write γ, α instead of $\gamma, \langle \alpha \rangle$.
(3) With each pair (Γ, t) formed by a context Γ and a term t, we associate a function $[\![\Gamma \vdash t]\!]$ that is partially defined on Γ-valuations: $[\![\Gamma \vdash t]\!]_\gamma$ denotes $[\![\Gamma \vdash t]\!](\gamma)$ when $\gamma \in [\![\Gamma]\!]$.
(4) In the following, we write $[\![t]\!]$ for $[\![\Gamma \vdash t]\!]_\gamma$ if Γ and $\gamma \in [\![\Gamma]\!]$ are fixed in the context. Similarly, we use the notation $\vec{u} \in [\![\vec{A}]\!]$ for $u_1 \in [\![\Gamma \vdash A_1]\!]_\gamma$, ..., $u_n \in [\![\Gamma, x_1 : A_1, ...x_{n-1} : A_{n-1} \vdash A_n]\!]_{\gamma, u_1...u_{n-1}}$ for some context Γ and Γ-valuation $\gamma \in [\![\Gamma]\!]$.

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

Lemma 3.5. *Suppose $\Gamma \vdash \mathcal{D}$, where $\mathcal{D} = \mathsf{Ind}_n\{\Delta_I := \Delta_C\}$. Let $\gamma \in [\![\Gamma]\!]$ be given. As mentioned before, we suppress Γ and γ for better readability. Further suppose that*

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

$$A_i := \Pi\vec{p} : \vec{P}. \, \Pi\vec{b}_i : \vec{B}_i. \, s_i \,, \ T_k := \Pi\vec{p} : \vec{P}. \, \Pi\vec{z}_k : \vec{Z}_k. \, d_{i_k} \, \vec{p}\vec{t}_k \,.$$

Then, there is some rule set Φ such that the following interpretation of $\mathcal{D} \cdot d_i$ and $\mathcal{D} \cdot c_k$ satisfies the soundness of the rules $(ind\text{-}type)$ and $(ind\text{-}const)$:
- $[\![\mathcal{D} \cdot d_i]\!] := \vec{\mathsf{lam}}(f_i)$ *where* $f_i(\vec{p}, \vec{b}_i) := \mathcal{IF}(\Phi)(i, \vec{p}, \vec{b}_i)$ *for* $\vec{p}, \vec{b}_i : [\![\vec{P}, \vec{B}_i]\!]$ *,*
- $[\![\mathcal{D} \cdot c_k]\!] := \vec{\mathsf{lam}}(g_k)$ *where* $g_k(\vec{p}, \vec{z}_k) := \langle k, \vec{z}_k \rangle$ *for* $\vec{p}, \vec{z}_k : [\![\vec{P}, \vec{Z}_k]\!]$ *.*

Remark 3.6. The positivity condition is crucial for showing that the construction of the rule set Φ in Appendix C is well defined.

The elements of our rule sets Φ are of the form $\frac{u}{\langle i, \vec{p}, \vec{t}, \langle j, \vec{v} \rangle \rangle}$, where i denotes the ith inductive type d_i, $\vec{p}$ denote the parameters, $\vec{t}$ denote the non-parametric arguments of d_i, j denotes the jth constructor of d_i, and $\vec{v}$ denote the non-parametric arguments of the jth constructor. Note that $\vec{p}, \vec{t}, \vec{v}$ could be empty.

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

$$\Phi_{\mathtt{nat}} \;=\; \left\{ \frac{\varnothing}{\langle 0, \langle 1 \rangle \rangle} \right\} \cup \left\{ \frac{\{v\}}{\langle 0, \langle 2, v \rangle \rangle} \;\middle|\; v \in \mathcal{V}_{\kappa_0} \right\},$$

$[\![\mathtt{nat}]\!] = \mathcal{IF}(\Phi_{\mathtt{nat}})(0)$, $[\![\mathtt{0}]\!] = \langle 1 \rangle$, and $\mathsf{app}([\![\mathtt{S}]\!], n) = \langle 2, n \rangle$ for any $n \in [\![\mathtt{nat}]\!]$.

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 $\mathtt{toto}$ can be represented by $\mathcal{D}_{toto} = \mathsf{Ind}_0\{\Delta_I := \Delta_C\}$, where

$$\Delta_I \;:=\; \mathtt{toto} : \mathsf{Type}_1 \to \mathsf{Type}_1\,,$$
$$\Delta_C \;:=\; \mathtt{Y}_1 : \Pi x : \mathsf{Type}_1. \mathtt{toto}\, x, \quad \mathtt{Y}_2 : \Pi x : \mathsf{Type}_1. \mathtt{toto}\, \mathtt{nat} \to \mathtt{toto}\, x \to \mathtt{toto}\, x\,.$$

Then,

$$\Phi_{\mathtt{toto}} = \left\{ \frac{\varnothing}{\langle 0, x, \langle 1, x \rangle \rangle} \;\middle|\; x \in \mathcal{V}_{\kappa_1} \right\} \cup \left\{ \frac{\{\langle 0, [\![\mathtt{nat}]\!], v_1 \rangle, \langle 0, x, v_2 \rangle\}}{\langle 0, x, \langle 2, x, v_1, v_2 \rangle \rangle} \;\middle|\; x, v_1, v_2 \in \mathcal{V}_{\kappa_1} \right\},$$

$\mathsf{app}([\![\mathtt{toto}]\!], x) = \mathcal{IF}(\Phi_{\mathtt{toto}})(0, x)$, $\mathsf{app}([\![\mathtt{Y}_1]\!], x) = \langle 1, x \rangle$, and $\mathsf{app}([\![\mathtt{Y}_2]\!], x, a, b) = \langle 2, x, a, b \rangle$, where $x \in \mathcal{V}_{\kappa_1}$, $a \in \mathsf{app}([\![\mathtt{toto}]\!], [\![\mathtt{nat}]\!])$, and $b \in \mathsf{app}([\![\mathtt{toto}]\!], x)$.

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 $\mathtt{titi}$ can be represented by $\mathcal{D}_{titi} = \mathsf{Ind}_1\{\Delta_I := \Delta_C\}$, where

$$\Delta_I \;:=\; \mathtt{titi} : \mathsf{Type}_1 \to \mathsf{Type}_0\,,$$
$$\Delta_C \;:=\; \mathtt{Z}_1 : \Pi x : \mathsf{Type}_1. \mathtt{titi}\, x, \quad \mathtt{Z}_2 : \Pi x : \mathsf{Type}_1. \mathtt{titi}\, \mathtt{nat} \to \mathtt{titi}\, x \to \mathtt{titi}\, x\,.$$

Then,

$$\Phi_{\mathtt{titi}} = \left\{ \frac{\varnothing}{\langle 0, x, \langle 1 \rangle \rangle} \;\middle|\; x \in \mathcal{V}_{\kappa_1} \right\} \cup \left\{ \frac{\{\langle 0, [\![\mathtt{nat}]\!], v_1 \rangle, \langle 0, x, v_2 \rangle\}}{\langle 0, x, \langle 2, v_1, v_2 \rangle \rangle} \;\middle|\; x \in \mathcal{V}_{\kappa_1},\; v_1, v_2 \in \mathcal{V}_{\kappa_0} \right\},$$

$\mathsf{app}([\![\mathtt{titi}]\!], x) = \mathcal{IF}(\Phi_{\mathtt{titi}})(0, x)$, $\mathsf{app}([\![\mathtt{Z}_1]\!], x) := \langle 1 \rangle$, and $\mathsf{app}([\![\mathtt{Z}_2]\!], x, a, b) := \langle 2, a, b \rangle$, where $x \in \mathcal{V}_{\kappa_1}$, $a \in \mathsf{app}([\![\mathtt{titi}]\!], [\![\mathtt{nat}]\!])$, and $b \in \mathsf{app}([\![\mathtt{titi}]\!], x)$.

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 $\mathcal{D}_{TF}$ in Remark 2.5 can be interpreted by means of the following rule set:

$$\Phi \quad := \quad \left\{ \frac{\{\langle 1, A, v_1 \rangle\}}{\langle 0, A, \langle 1, a, v_1 \rangle\rangle} \;\middle|\; A, v_1 \in \mathcal{V}_{\kappa_0}, a \in A \right\}$$

$$\cup \left\{ \frac{\varnothing}{\langle 1, A, \langle 2 \rangle\rangle} \;\middle|\; A \in \mathcal{V}_{\kappa_0} \right\} \cup \left\{ \frac{\{\langle 0, A, v_1 \rangle, \langle 1, A, v_2 \rangle\}}{\langle 1, A, \langle 3, v_1, v_2 \rangle\rangle} \;\middle|\; A, v_1, v_2 \in \mathcal{V}_{\kappa_0} \right\}.$$

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), $(fix\text{-}eq)$, and (ι). A formal definition is given in Appendix D. Refer to [Dybjer(1991)], whose study provides the basic idea.

Lemma 3.12. *Suppose $\Gamma \vdash \mathsf{fix}\ f_\ell\ \{R\} : A_j$, where*

$$R = \overrightarrow{f/k : A := t}, \quad A_i \equiv \Pi \vec{x}_i : \vec{B}_i.\, A_i', \quad lh(\vec{B}_i) = k_i + 1, \quad \ell \le n,$$

$$(\Gamma \vdash A_i : s_i)_{\forall i \le n}, \quad (\Gamma, \vec{f} : \vec{A} \vdash t_i : A_i)_{\forall i \le n}, \quad \mathcal{F}(\vec{f}, \vec{A}, \vec{k}, \vec{t}).$$

Let $\gamma \in [\![\Gamma]\!]$ be given. We suppress Γ and γ for better readability. Then, there is a rule set Ψ such that the following interpretation of $\mathsf{fix}\ f_\ell\ \{R\}$ satisfies the soundness of the rules (fix), $(fix\text{-}eq)$, and (ι):

- $[\![\mathsf{fix}\ f_\ell\ \{R\}]\!] = \vec{\mathsf{lam}}(h)$*, where $h(a_1, ..., a_{k_\ell}, \langle k, \vec{z}_k \rangle) = \mathcal{IF}(\Psi)(a_1, ..., a_{k_\ell}, \langle k, \vec{z}_k \rangle)$ for $\vec{a}, \langle k, \vec{z}_k \rangle \in [\![\vec{B}_\ell]\!]$.*

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 Ψ in Appendix D is well defined.

The elements of our rule sets Ψ are of the form $\frac{u}{\langle \vec{x}, \langle k, \vec{y} \rangle, b \rangle}$, where k denotes the kth constructor of the inductive type d on which the recursion is performed, $\vec{y}$ denote the non-parametric arguments of d, $\vec{x}$ denotes the list of rest arguments of the constructor, and b is the result of the function. Note that $\vec{x}, \vec{y}$ could be empty.

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

```
Fixpoint PRec (A:Type)(g:A)(h:nat -> A -> A)(n:nat) {struct n} : A :=
  match n with
    | O => g
    | S p => h p (PRec A g h p)
  end.
```

The corresponding term is $\mathsf{PRec} := \mathsf{fix}\, f_0\, \{f_0/3 : B := t\}$, where

$$B \;=\; \Pi(A : \mathsf{Type}_i).\, \Pi(g : A).\, \Pi(h : \mathtt{nat} \to A \to A).\, \Pi(n : \mathtt{nat}).\, \mathtt{nat}\,,$$

$$t \;=\; \lambda(A : \mathsf{Type}_i).\, \lambda(g : A).\, \lambda(h : \mathtt{nat} \to A \to A).\, \lambda(n : \mathtt{nat}).\, \mathsf{case}(n, P, h_1, h_2)\,,$$

$P = \lambda(\ell : \mathtt{nat}).\, A$, $h_1 = g$, and $h_2 = \lambda(p : \mathtt{nat}).\, h\, p\, (f_0\, A\, g\, h\, p)$. $[\![\mathsf{PRec}]\!]$ is characterized by the following rule set $\Psi_{\mathsf{PRec}} :=$

$$\left\{ \frac{\varnothing}{\langle A, g, h, \langle 1 \rangle, g \rangle} \;\middle|\; A \in \mathcal{V}_{\kappa_i}, g \in A, h \in [\![\mathtt{nat} \to A \to A]\!] \right\}$$

$$\cup \left\{ \frac{\{\langle A, g, h, p, v \rangle\}}{\langle A, g, h, \langle 2, p \rangle, \vec{\mathsf{app}}(h, p, v) \rangle} \;\middle|\; A \in \mathcal{V}_{\kappa_i}, g \in A, h \in [\![\mathtt{nat} \to A \to A]\!], p \in [\![\mathtt{nat}]\!], v \in A \right\}.$$

Given a type A, $[\![\mathsf{PRec}\, A]\!]$ denotes the primitive recursor with values from A. For instance,

$$\vec{\mathsf{app}}([\![\mathtt{plus}]\!], m, n) = \mathcal{IF}(\mathsf{Prec})([\![\mathtt{nat}]\!], m, [\![h]\!], n)$$

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

Example 3.15 (Mutually recursive functions). The interpretations of $\mathtt{Tsize}$ and $\mathtt{Fsize}$ from Remark 2.7 are characterized by the following rule set $\Psi_{\mathsf{Size}} :=$

$$\left\{ \frac{\{\langle A, f, v' \rangle\}}{\langle A, \langle 1, a, f \rangle, [\![\mathsf{S}\, v']\!] \rangle} \;\middle|\; A \in \mathcal{V}_{\kappa_0}, f \in \mathsf{app}([\![\mathtt{forest}]\!], A), v' \in [\![\mathtt{nat}]\!], a \in A \right\}$$

$$\cup \left\{ \frac{\varnothing}{\langle A, \langle 2 \rangle, [\![\mathtt{0}]\!] \rangle} \;\middle|\; A \in \mathcal{V}_{\kappa_0} \right\}$$

$$\cup \left\{ \frac{\{\langle A, t, v'_1 \rangle, \langle A, f', v'_2 \rangle\}}{\langle A, \langle 3, t, f' \rangle, \mathsf{app}([\![\mathtt{plus}]\!], v'_1, v'_2) \rangle} \;\middle|\; A \in \mathcal{V}_{\kappa_0}, t \in \mathsf{app}([\![\mathtt{tree}]\!], A), \right.$$

$$\left. f' \in \mathsf{app}([\![\mathtt{forest}]\!], A), v'_1, v'_2 \in [\![\mathtt{nat}]\!] \right\}.$$

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:

- $|\Gamma| < |\Gamma \vdash A| < |\Gamma, x : A|$,
- $|\Gamma \vdash t|, |\Gamma \vdash A| < |\Gamma, x := t : A|$,
- $|\Delta_I(d)|, |\Delta_C(c)| < |\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \cdot x|$ for all $x \in \mathrm{dom}(\Delta_I, \Delta_C)$, $d \in \mathrm{dom}(\Delta_I)$, and $c \in \mathrm{dom}(\Delta_C)$,
- $|A_i|, |t_j| < |\overrightarrow{f/k : A := t}|$ for all $A_i \in \vec{A}$ and $t_j \in \vec{t}$.

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 $\mathsf{Ind}_n\{\Delta_I := \Delta_C\} \cdot x$ are $\Delta_I(d)$ and $\Delta_C(c)$, where $d \in \mathrm{dom}(\Delta_I)$ and $c \in \mathrm{dom}(\Delta_C)$.
- The immediate subterms of $\mathsf{fix}\ f_j\ \{\overrightarrow{f/k : A := t}\}$ are $\vec{A}, \vec{t}$.

- The size of a context Γ is defined as follows:
  - $\|[]\| = 1/2$,
  - $|\Gamma, (x : t)| = |\Gamma| + |t|$,
  - $|\Gamma, (x := t : A)| = |\Gamma| + |t| + |A|$,
  - $|\Gamma, \mathsf{Ind}_n\{\Delta_I := \Delta_C\}| = |\Gamma| + 1$.
- $|\Gamma \vdash t| = |\Gamma| + |t| - \frac{1}{2}$.

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 4.4.

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 Γ, the set $[\![\Gamma]\!]$ is defined as follows:

$$[\![[]\,]\!] := \{\mathtt{nil}\},$$

$$[\![\Gamma, x : A]\!] := \{\gamma, \alpha \mid \gamma \in [\![\Gamma]\!], [\![\Gamma \vdash A]\!]_\gamma \downarrow \text{ and } \alpha \in [\![\Gamma \vdash A]\!]_\gamma\},$$

$$[\![\Gamma, x := t : A]\!] := \{\gamma, \alpha \mid \gamma \in [\![\Gamma]\!], [\![\Gamma \vdash A]\!]_\gamma \downarrow, [\![\Gamma \vdash t]\!]_\gamma \downarrow$$
$$\text{and } \alpha = [\![\Gamma \vdash t]\!]_\gamma \in [\![\Gamma \vdash A]\!]_\gamma\},$$

$$[\![\Gamma, \mathsf{Ind}_n\{\Delta_I := \Delta_C\}]\!] := \{\gamma \mid \gamma \in [\![\Gamma]\!]\}.$$

(2) The interpretation $[\![\Gamma \vdash t]\!]$ of a term t in a context Γ is a partial function defined on $[\![\Gamma]\!]$: Given $\gamma \in [\![\Gamma]\!]$,

$$[\![\Gamma \vdash \mathtt{Prop}]\!]_\gamma := \{0, 1\},$$

$$[\![\Gamma \vdash \mathtt{Type}_i]\!]_\gamma := \mathcal{V}_{\kappa_i},$$

$$[\![\Gamma \vdash x]\!]_{(\alpha_1, \dots, \alpha_n)} := \alpha_i \quad \text{if } x \text{ is the } i\text{th declared variable in } \Gamma, \qquad (*)$$

$$[\![\Gamma \vdash \Pi x : A.B]\!]_\gamma := \{\mathsf{lam}(f) : f \in \Pi_{\alpha \in [\![\Gamma \vdash A]\!]_\gamma} [\![\Gamma, x : A \vdash B]\!]_{(\gamma, \alpha)}\},$$

$$[\![\Gamma \vdash \lambda x : A.t]\!]_\gamma := \mathsf{lam}(\alpha \in [\![\Gamma \vdash A]\!]_\gamma \mapsto [\![\Gamma, x : A \vdash t]\!]_{(\gamma, \alpha)}),$$

$$[\![\Gamma \vdash t\,u]\!]_\gamma := \mathsf{app}([\![\Gamma \vdash t]\!]_\gamma, [\![\Gamma \vdash u]\!]_\gamma),$$

$$[\![\Gamma \vdash \mathsf{let}\ x := t\ \mathsf{in}\ u]\!]_\gamma := [\![\Gamma, (x := t : A) \vdash u]\!]_{\gamma, [\![\Gamma \vdash t]\!]_\gamma},$$
$$\text{where } A \text{ is such that } [\![\Gamma \vdash t]\!] \in [\![\Gamma \vdash A]\!], \qquad (\dagger)$$

$$[\![\Gamma \vdash \mathsf{Ind}_n\{\Delta_I := \Delta_C\} \cdot z]\!]_\gamma := \text{as explained above if defined},$$

$$[\![\Gamma \vdash \mathsf{case}(e, P, M_1, \dots, M_\ell)]\!]_\gamma := \mathsf{a\vec{p}p}([\![\Gamma \vdash M_j]\!], ([\![\Gamma \vdash e]\!])_1, \dots, ([\![\Gamma \vdash e]\!])_q)$$
$$\text{if } ([\![\Gamma \vdash e]\!])_0 = j \text{ where } lh(e) = q + 1,$$

$$[\![\Gamma \vdash (\mathsf{fix}\ f_j\ \{\overrightarrow{f/k : A := t}\})]\!]_\gamma := \text{as explained above if defined}.$$

$(*)$ If $x \in \mathrm{dom}(\Gamma)$, then the occurrence should be unique.

$(\dagger)$ 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* Γ *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,* $\gamma, \delta \in [\![\Gamma, \Delta[x\backslash u]]\!]$.
(2) *Suppose* $(\gamma, \alpha), \delta \in [\![\Gamma, x : A, \Delta]\!]$ *and* $[\![\Gamma, x : A, \Delta \vdash t]\!]_{(\gamma,\alpha),\delta} \downarrow$. *Then,*
   - $[\![\Gamma, \Delta[x\backslash u] \vdash t[x\backslash u]]\!]_{\gamma,\delta} \downarrow$.
   - $[\![\Gamma, \Delta[x\backslash u] \vdash t[x\backslash u]]\!]_{\gamma,\delta} = [\![\Gamma, x : A, \Delta \vdash t]\!]_{(\gamma,\alpha),\delta} = [\![\Gamma, x := u : A, \Delta \vdash t]\!]_{(\gamma,\alpha),\delta}$.

Proof. The assertions are proved for each Δ and t by a mutual induction on the size of their arguments. In particular, given Δ, 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
$$\beta \in [\![\Gamma, x : A, \Delta_0 \vdash B]\!]_{\gamma,\alpha,\delta_0} = [\![\Gamma, \Delta_0[x\backslash u] \vdash B[x\backslash u]]\!]_{\gamma,\delta_0} .$$

That is, $\gamma, \delta_0, \beta \in [\![\Gamma, \Delta_0[x\backslash u], y : B[x\backslash u]]\!]$.

(2) We proceed by induction on t. If $t = x$, the claim follows because $[\![\Gamma \vdash u]\!]_\gamma \downarrow$ implies that $[\![\Gamma, \Delta[x\backslash u] \vdash u]\!]_{\gamma,\delta} \downarrow$ and $[\![\Gamma, \Delta[x\backslash u] \vdash u]\!]_{\gamma,\delta} = [\![\Gamma \vdash u]\!]_\gamma$. This is because the interpretation of u does not depend on $\mathrm{dom}(\Delta)$. Other cases can be easily shown by using induction hypotheses.

$\square$

Theorem 4.4 (Soundness)**.** *Our type system is sound with respect to the set-theoretic interpretation defined in Definition 4.2 in the following sense:*

(1) *If* $\mathcal{WF}(\Gamma)$, *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* $\Gamma \vdash M \prec N$, *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 $(weak\text{-}eq)$ are obvious.

(Π) Suppose
$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.B : s_3} .$$
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
$$[\![\Gamma \vdash \Pi x : A.B]\!]_\gamma \in [\![\Gamma \vdash s_3]\!]_\gamma .$$

If $s_2 = s_3 = \mathtt{Prop}$, then Lemma 3.3 implies the claim. Assume $s_1 = \mathtt{Type}_i, s_2 = \mathtt{Type}_j, s_3 = \mathtt{Type}_k$, and $i, j \leq k$. Then, $[\![\Gamma \vdash s_3]\!] = \mathcal{V}_{\kappa_k}$, where κ_k is the kth inaccessible cardinal; hence, $\mathcal{V}_{\kappa_k}$ is closed under the power set operation.

The cases (Π-eq), (λ), and (λ-eq) are obvious.

(app) Suppose
$$\frac{\Gamma \vdash M : \Pi x : A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x \backslash N]} \; .$$
By induction hypothesis, it holds that $[\![\Gamma \vdash N]\!]_\gamma \in [\![\Gamma \vdash A]\!]_\gamma$ and $[\![\Gamma \vdash M]\!]_\gamma = \mathsf{lam}(f)$ for some function f with $\mathrm{dom}(f) = [\![\Gamma \vdash A]\!]$ and $f(\alpha) \in [\![\Gamma, x : A \vdash B]\!]_{\gamma,\alpha}$ for any $\alpha \in [\![\Gamma \vdash A]\!]_\gamma$. Thus, we have
$$[\![\Gamma \vdash MN]\!]_\gamma = \mathsf{app}([\![\Gamma \vdash M]\!]_\gamma, [\![\Gamma \vdash N]\!]_\gamma) = f([\![\Gamma \vdash N]\!]_\gamma)$$
$$\in [\![\Gamma, x : A \vdash B]\!]_{\gamma, [\![\Gamma \vdash N]\!]_\gamma} = [\![\Gamma \vdash B[x \backslash N]]\!]_\gamma \; .$$
The cases (app-eq), (let), and (let-eq) are similar.

The soundness of (ind-wf), (ind-$type$), and (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 (fix), (fix-eq), ($case$), and ($case$-eq).

The cases (ref), (sym), ($trans$), ($conv$), and ($conv$-eq) are obvious.

(β) Suppose
$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x : A.M)N = M[x \backslash N] : B[x \backslash N]} \; .$$
It remains to show that $[\![\Gamma \vdash (\lambda x : A : M)N]\!]_\gamma = [\![M[x \backslash N]]\!]_\gamma$:
$$[\![\Gamma \vdash (\lambda x : A : M)N]\!]_\gamma = \mathsf{app}([\![\Gamma \vdash \lambda x : A.M]\!]_\gamma, [\![\Gamma \vdash N]\!]_\gamma)$$
$$= \mathsf{app}(\mathsf{lam}(\alpha \in [\![\Gamma \vdash A]\!]_\gamma \mapsto [\![\Gamma, x : A \vdash M]\!]_{\gamma,\alpha}), [\![\Gamma \vdash N]\!]_\gamma)$$
$$= [\![\Gamma, x : A \vdash M]\!]_{\gamma, [\![\Gamma \vdash N]\!]_\gamma}$$
$$= [\![M[x \backslash N]]\!]_\gamma$$
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 (δ) is obvious, and the case (ζ) follows from Lemma 4.3 and induction hypothesis. The case (ι) 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 $\vdash t : \Pi x : *.x$.*

Proof. Note that $[\![\vdash \Pi x : *.x]\!]_{\mathtt{nil}} = \varnothing$. $\square$

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 β-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

References

[Aczel(1977)] Peter Aczel. An introduction to inductive definition. In Jon Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, 1977.

[Aczel(1998)] Peter Aczel. On relating type theories and set theories. In *Proceedings of Types '98*, volume 1657 of *Lecture Notes in Comput. Sci.*, pages 1–18. Springer, 1998.

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

[Bertot and Castéran(2004)] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer Verlag, 2004.

[Coquand(1990)] Thierry Coquand. Metamathematical Investigation on a Calculus of Constructions. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of APIC series, pages 91–122. Academic Press, 1990.

[Coquand(1991)] Thierry Coquand. An algorithm for testing conversion in type theory. In *Logical frameworks*, pages 255–279. Cambridge University Press, 1991.

[Drake(1974)] Frank R. Drake. *Set Theory: An Introduction to Large Cardinals*, volume 76 of *Studies in logic and the foundations of mathematics*. North Holland, 1974.

[Dybjer(1991)] Peter Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In *Logical frameworks (Sophia-Antipolis, 1990)*, pages 280–306. Cambridge Univ. Press, 1991.

[Dybjer(2000)] Peter Dybjer. A General Formulation of Simultaneous Inductive-Recursive Definitions in Type Theory. *J. Symb. Log.*, 65(2):525–549, 2000.

[Giménez(1995)] Eduardo Giménez. Codifying guarded definitions with recursive schemes. In *Types for proofs and programs (Båstad, 1994)*, volume 996 of *Lecture Notes in Comput. Sci.*, pages 39–59. Springer, 1995.

[Girard et al.(1989)Girard, Taylor, and Lafont] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Goguen(1994)] Healfdene Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, 1994.

[Goguen(1999)] Healfdene Goguen. Soundness of the Logical Framework for Its Typed Operational Semantics. In *Typed Lambda Calculi and Applications, TLCA'99*, pages 177–197, 1999.

[Letouzey(2004)] Pierre Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, July 2004.

[Luo(1989)] Zhaohui Luo. Ecc, an extended calculus of constructions. In *LICS*, pages 386–395. IEEE Computer Society, 1989.

[Luo(1990)] Zhaohui Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, November 1990.

[Luo(1994)] Zhaohui Luo. *Computation and reasoning: a type theory for computer science*, volume 11 of *International Series of Monographs on Computer Science*. Oxford University Press, Inc., 1994.

[Martin-Löf(1984)] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory. Lecture Notes*. Bibliopolis, 1984.

[Miquel and Werner(2003)] Alexandre Miquel and Benjamin Werner. The not so simple proof-irrelevant model of CC. In *Types for proofs and programs*, volume 2646 of *Lecture Notes in Comput. Sci.*, pages 240–258. Springer, 2003.

[Moschovakis(1974)] Yiannis N. Moschovakis. *Elementary induction on abstract structures*, volume 77 of *Studies in logic and the foundations of mathematics*. North Holland, 1974.

[Moschovakis(1980)] Yiannis N. Moschovakis. *Descriptive set theory*, volume 100 of *Studies in logic and the foundations of mathematics*. North Holland, 1980.

[Nordström et al.(1990) Nordström, Petersson, and Smith] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's type theory: An introduction*. Oxford University Press, 1990.

[Paulin-Mohring(1996)] Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation á diriger les recherches. Université Claude Bernard Lyon I, 1996.

[Reynolds(1984)] John C. Reynolds. Polymorphism is not set-theoretic. In *Semantics of Data Types*, volume 173 of *Lecture Notes in Comput. Sci.*, pages 145–156. Springer, 1984.

[Siles and Herbelin(2010)] Vincent Siles and Hugo Herbelin. Equality is typable in semi-full pure type systems. In *LICS*, pages 21–30, 2010.

[Werner(1997)] Benjamin Werner. Sets in Types, Types in Sets. In *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Comput. Sci.*, pages 530–546. Springer, 1997.

[Werner(2008)] Benjamin Werner. On the strength of proof-irrelevant type theories. *Logical Methods in Computer Science*, 4(3), 2008.

Appendix A. Definition of free variables and substitution

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

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

$$s[x\backslash u] := s,$$

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

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

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

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

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

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

$$(\text{fix } y_i\, \overrightarrow{\{y/k : A := t\}})[x\backslash u] := \begin{cases} \text{fix } y_i\, \overrightarrow{\{y/k : A := t\}} \\ \quad \text{if } x \in \{\vec{y}\} \text{ or } FV(u) \cap \{\vec{y}\} \neq \varnothing, \quad (\dagger) \\ \text{fix } y_i\, \overrightarrow{\{y/k : (A[x\backslash u]) := t[x\backslash u]\}} \text{ otherwise.} \end{cases}$$

$(*)$ By using α-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 Π and λ.

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 Δ_I, Δ_C, and they have three forms: the *empty constraint* ϵ, the constraint $\lessdot z$, which describes the structural smallness with respect to z, and the constraint $=z$, which describes the equivalence to z. The constraints will be added to any occurrence of a variable in a term. Let $\mathbf{c}, \mathbf{d}, \ldots$ vary over constraints. The judgments of constrained typing have the form

$$\frac{\begin{array}{c} \mathsf{Ind}_n\{\Delta_I := \Delta_C\} \in \Gamma \quad (d_i : \Pi\vec{p} : \vec{P}.\, A) \in \Delta_I \quad lh(\vec{p}) = n \\ \Gamma \vdash Q :^\epsilon B \quad \mathcal{C}(d_i\,\vec{p} : A; B) \quad \Gamma \vdash e :^{\mathbf{c}} d_i\,\vec{p}\,\vec{u} \\ \Gamma \vdash h_k :^{\mathbf{d}} \Pi(\vec{v} : \vec{V}_k)^{<\mathbf{c}}.\, Q\,\vec{w}_k\,(c_k\,\vec{p}\,\vec{v}) \text{ for all } (c_k : \Pi\vec{p} : \vec{P}.\,\Pi\vec{v} : \vec{V}_k.\, d_i\,\vec{p}\,\vec{w}_k) \in \Delta_C \end{array}}{\Gamma \vdash \mathsf{case}(e, Q, (h_k)_k) :^{\mathbf{d}} Q\,\vec{u}\,e}$$

$$\frac{\Gamma \vdash M : s}{\mathcal{WF}(\Gamma, x :^{\mathbf{c}} M)} \qquad \frac{\mathcal{WF}(\Gamma) \quad x :^{\mathbf{c}} A \in \Gamma}{\Gamma \vdash x :^{\mathbf{c}} A} \qquad \frac{\Gamma \vdash t :^{\mathbf{c}} A}{\Gamma \vdash t :^\epsilon A}$$

$$\frac{\Gamma \vdash A :^\epsilon s_1 \quad \Gamma, x :^{\mathbf{c}} A \vdash B :^\epsilon s_2 \quad \mathcal{P}(s_1, s_2, s_3)}{\Gamma \vdash \Pi x :^{\mathbf{c}} A.B :^\epsilon s_3}$$

$$\frac{\Gamma \vdash A = A' :^\epsilon s_1 \quad \Gamma, x :^{\mathbf{c}} A \vdash B = B' :^\epsilon s_2 \quad \mathcal{P}(s_1, s_2, s_3)}{\Gamma \vdash \Pi x :^{\mathbf{c}} A.B = \Pi x :^{\mathbf{c}} A'.B' :^\epsilon s_3}$$

$$\frac{\begin{array}{c} \Gamma \vdash A :^\epsilon s_1 \quad \Gamma, x :^{\mathbf{c}} A \vdash B :^\epsilon s_2 \\ \Gamma, x :^{\mathbf{c}} A \vdash M :^{\mathbf{d}} B \end{array}}{\Gamma \vdash \lambda x :^{\mathbf{c}} A.M :^{\mathbf{d}} \Pi x :^{\mathbf{c}} A.B}$$

$$\frac{\begin{array}{c} \Gamma \vdash A = A' :^\epsilon s_1 \quad \Gamma, x :^{\mathbf{c}} A \vdash B :^\epsilon s_2 \\ \Gamma, x :^{\mathbf{c}} A \vdash M = M' :^{\mathbf{d}} B \end{array}}{\Gamma \vdash \lambda x :^{\mathbf{c}} A.M = \lambda x :^{\mathbf{c}} A'.M' :^{\mathbf{d}} \Pi x :^{\mathbf{c}} A.B}$$

$$\frac{\Gamma \vdash M :^{\mathbf{d}} \Pi x :^{\mathbf{c}} A.B \quad \Gamma \vdash N :^{\mathbf{c}} A}{\Gamma \vdash MN :^{\mathbf{d}} B[x\backslash N]}$$

$$\frac{\Gamma \vdash M = M' :^{\mathbf{d}} \Pi x :^{\mathbf{c}} A.B \quad \Gamma \vdash N = N :^{\mathbf{c}} A}{\Gamma \vdash MN = M'N' :^{\mathbf{d}} B[x\backslash N]}$$

Figure 4: Constrained typing

$\Gamma \vdash M :^{\mathbf{c}} N$, where the constraints are added to all the variables from $\mathrm{dom}(\Gamma)$. M^ϵ and Γ^ϵ denote the term M and the term sequence Γ, respectively, where only the constraint ϵ is added.

Given a constraint $\mathbf{c}$, the constraint $<\mathbf{c}$ is defined as follows:

$$< \epsilon := \epsilon, \qquad <\!\Rightarrow\!z := \,<\!z, \qquad <\!<\!z := \,<\!z.$$

The following defines the restriction of a recursive call of inductive type when defining a mutual recursion. Given a declaration Δ, $\Delta^{<z}$ is defined as follows:

$$([])^{<z} := [],$$
$$(\Delta, x : A)^{<z} := \Delta^{<z}, x :^\epsilon A \quad \text{if } FV(A) \cap \mathrm{dom}(\Delta_I, \Delta_C) = \varnothing,$$
$$(\Delta, x : A)^{<z} := \Delta^{<z}, x :^{<z} A \quad \text{if } FV(A) \cap \mathrm{dom}(\Delta_I, \Delta_C) \neq \varnothing.$$

$\Gamma^{<z}$ 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.

Appendix C. Interpretation of Inductive types

Suppose $\Gamma \vdash \mathcal{D}$, where $\mathcal{D} = \mathsf{Ind}_n\{\Delta_I := \Delta_C\}$, and $\gamma \in [\![\Gamma]\!]$. As mentioned before, we suppress Γ and γ for better readability. Suppose that

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

$$A_i := \Pi\vec{p} : \vec{P}. \Pi\vec{b}_i : \vec{B}_i. s_i \,, \ \ T_k := \Pi\vec{p} : \vec{P}. \Pi\vec{z}_k : \vec{Z}_k. d_{i_k} \, \vec{p} \, \vec{t}_k \,, \ \ Z_{k,j} := \Pi\vec{u}_{k,j} : \vec{H}_{k,j}. d_{i_{k,j}} \, \vec{p} \, \vec{w}_{k,j} \,,$$

where $lh(\vec{p}) = n$, $lh(\vec{B}_i) = \ell_i$, $lh(\vec{t}_k) = \ell_{i_k}$, $j \in \nu(k) := \{j \mid FV(Z_{k,j}) \cap \mathrm{dom}(\Delta_I) \neq \varnothing\}$, and $i_k, i_{k,j} \leq \ell$. Furthermore, $\vec{Z}_k'$ is defined as

$$Z_{k,j}' := \begin{cases} Z_{k,j} & \text{if } j \notin \nu_k, \\ \Pi\vec{u}_{k,j} : \vec{H}_{k,j}. d_{i_{k,j}}' & \text{if } j \in \nu_k, \end{cases}$$

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

Then, we set $[\![\mathcal{D}]\!] := \mathcal{I}(\Phi)$, where $\Phi :=$

$$\bigcup_{i \leq \ell} \bigcup_{k \in \mu_i} \left\{ \frac{\bigcup_{j \in \nu_k} \{\langle i_{k,j}, \vec{p}, [\![\vec{w}_{k,j}]\!]_{\vec{p}, \vec{z}_k, \vec{u}}, \mathsf{app}(z_{k,j}, \vec{u})\rangle \mid \vec{u} \in [\![\vec{H}_{k,j}]\!]_{\vec{p}, \vec{z}_k}\}}{\langle i, \vec{p}, [\![\vec{t}_k]\!]_{\vec{p}, \vec{z}_k}, \langle k, \vec{z}_k\rangle\rangle} \ \middle| \ \vec{p}, \vec{z}_k \in [\![\vec{P}]\!], [\![\vec{Z}_k']\!]_{\rho_k} \right\}.$$

Here, $\mu_i := \{k \mid d_{i_k} = d_i\}$, and ρ_k associates $\mathcal{V}_{\kappa_{r(k,j)}}$ with $d_{i_{k,j}}'$, where $r(k,j) := rank(d_{i_{k,j}})$.

We also set $[\![\mathcal{D} \cdot d_i]\!] := \mathsf{lam}(\vec{f_i})$, where $f_i(\vec{p}, \vec{b}_i) = \mathcal{IF}(\Phi)(i, \vec{p}, \vec{b}_i)$ for $\vec{p}, \vec{b}_i : [\![\vec{P}, \vec{B}_i]\!]$, and $[\![\mathcal{D} \cdot c_k]\!] := \mathsf{lam}(\vec{g_k})$, where $g_k(\vec{p}, \vec{z}_k) = \langle k, \vec{z}_k\rangle$ 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 $\Gamma \vdash \mathsf{fix} \ f_\ell \, \{R\} : A_j$, where

$$R := \overrightarrow{f/k : A := t}, \quad A_i \equiv \Pi\vec{x}_i : \vec{B}_i. A_i', \quad lh(\vec{B}_i) = k_i + 1, \quad \ell \leq n,$$

$$(\Gamma \vdash A_i : s_i)_{\forall i \leq n}, \quad (\Gamma, \vec{f} : \vec{A} \vdash t_i : A_i)_{\forall i \leq n}, \quad \mathcal{F}(\vec{f}, \vec{A}, \vec{k}, \vec{t})\,.$$

Let $\gamma \in [\![\Gamma]\!]$ be given. We suppress Γ and γ for better readability. Then, $[\![\mathsf{fix} \ f_\ell \, \{R\}]\!]$ will depend on the ι-reduction.

Suppose $\Gamma \vdash \vec{a}, a_{k_\ell + 1} : \vec{B}_\ell$, where $a_{k_\ell + 1} = T_k \, \vec{p} \, \vec{u}_k$, and $B_{\ell, k_\ell + 1} = x_{i_\ell} \, \vec{p} \, \vec{t}_k$, i.e., $k \in \mu_{i_\ell}$, 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, $(t_\ell[f_i \backslash (\mathsf{fix} \ f_i \, \{R\})]) \, \vec{a} \, (T_k \, \vec{p} \, \vec{u}_k)$ β-, ι-reduces to the term $M_{\ell, k}$ which is obtained from the node term of the branch which $T_k \, \vec{p} \, \vec{u}_k$ represents.

Suppose that for some $g_{\ell, k}, h_{\ell, k} \in \mathbb{N}$, $u_1', ..., u_{g_{\ell, k}}', u_{g_{\ell, k} + 1}', ..., u_{h_{\ell, k}}'$ list all the subterms of $M_{\ell, k}$ that are structurally smaller than $\vec{u}_k$. Each u_q' with $q \leq g_{\ell, k}$ occurs as the $(k_{n_q} + 1)$th argument of $(\mathsf{fix} \ f_{n_q} \, \{R\})$. Thus,

$$((\mathsf{fix} \ f_{n_q} \, \{R\}) \, \vec{b}_{n_q} \, u_q')$$

is a subterm of $M_{j,k}$ for some $\vec{b}_{n_q} : B_{n_q,1}, ..., B_{n_q,k_{n_q}}$. Note that each u'_q is an argument of some constructor $T_{m_q} : \Pi\vec{p} : \vec{P}.\Pi\vec{z}_{m_q} : \vec{Z}_{m_q}.x_{i_{m_q}} \vec{p}\vec{t}_{m_q}$ such that the head of u'_q is of some type $Z_{m_q,j_q} = \Pi\vec{u}_{m_q,j_q} : \vec{H}_{m_q,j_q}.x_{i_{m_q,j_q}} \vec{p}\,\vec{w}_{m_q,j_q}$ and that $x_{i_{m_q,j_q}} = x_{i_{n_q}}$ if $q \leq g_{\ell,k}$. Furthermore, we suppose that $u'_{q_1}, ..., u'_{q_h}$ are all terms among $u'_1, ..., u'_{h_{\ell,k}}$, which are headed by some variables.

Thus, $u'_{q_r} = a'_{q_r} \vec{u}'_{m_{q_r},j_{q_r}}$ for a variable a'_{q_r} of type

$$Z_{m_{q_r},j_{q_r}} = \Pi\vec{u}_{m_{q_r},j_{q_r}} : \vec{H}_{m_{q_r},j_{q_r}}.x_{i_{m_{q_r},j_{q_r}}} \vec{p}\,\vec{w}_{m_{q_r},j_{q_r}}$$

and for some terms $\vec{u}'_{m_{q_r},j_{q_r}}$. Note that $a'_{q_1}, ..., a'_{q_h}$ are exactly the free variables occurring in $\vec{u}_k$. Suppose that $u'_{q_{r_1}}, ..., u'_{q_{r_q}}$ are all such terms structurally smaller than u'_q. Then,

$$u'_q = (\lambda\vec{u}_{m_{q_{r_1}},j_{q_{r_1}}} : \vec{H}_{m_{q_{r_1}},j_{q_{r_1}}}. ... .\lambda\vec{u}_{m_{q_{r_q}},j_{q_{r_q}}} : \vec{H}_{m_{q_{r_q}},j_{q_{r_q}}}.\tilde{u}'_q)\, \vec{u}'_{m_{q_{r_1}},j_{q_{r_1}}} \cdots \vec{u}'_{m_{q_{r_q}},j_{q_{r_q}}}$$

for some $\tilde{u}'_q$. Similarly, $M_{\ell,k}$ can be written as follows:

$$(\lambda\vec{u}_{m_{q_{r_1}},j_{q_{r_1}}} : \vec{H}_{m_{q_{r_1}},j_{q_{r_1}}}. ... .\lambda\vec{u}_{m_{q_{r_q}},j_{q_{r_q}}} : \vec{H}_{m_{q_{r_q}},j_{q_{r_q}}}.(\mathsf{fix}\ f_{n_q}\,\{R\})\,\vec{b}_{n_q}\,\bar{u}'_q)\, \vec{u}'_{m_{q_{r_1}},j_{q_{r_1}}} \cdots \vec{u}'_{m_{q_{r_q}},j_{q_{r_q}}}$$

where

$$\bar{u}'_q = \lambda\vec{u}_{m_{q_{r_1}},j_{q_{r_1}}} : \vec{H}_{m_{q_{r_1}},j_{q_{r_1}}}. ... .\lambda\vec{u}_{m_{q_{r_q}},j_{q_{r_q}}} : \vec{H}_{m_{q_{r_q}},j_{q_{r_q}}}.\tilde{u}'_q.$$

Further, set

$$\bar{M}_{\ell,k} = (\lambda\vec{u}_{m_{q_{r_c}},j_{q_{r_c}}} : \vec{H}_{m_{q_{r_c}},j_{q_{r_c}}})_c.(\mathsf{fix}\ f_{n_q}\,\{R\})\,\vec{b}_{n_q}\,\bar{u}'_q$$

of type $(\Pi\vec{u}_{m_{q_{r_c}},j_{q_{r_c}}} : \vec{H}_{m_{q_{r_c}},j_{q_{r_c}}})_c.A'_{n_q}\{\vec{x}_{n_q} : \vec{b}_{n_q}, u'_q\}$, where c ranges over $1, ..., q$. Lastly, let $M'_{\ell,k}$ be obtained from $M_{\ell,k}$ by replacing $\bar{M}_{\ell,k}$ with a fresh variable X_{n_q}.

Then, $\bigcup_\ell [\![\mathsf{fix}\ f_\ell\,\{R\}]\!]$ will correspond to the fixpoint of the following rule set:

$$\Psi := \bigcup_{\ell \leq n} \bigcup_{k \in \mu_{i_\ell}} \left\{ \frac{\bigcup_{q \in \{1,...,g_{\ell,k}\}}\{\langle[\![\vec{b}_{n_q}]\!]_\rho, [\![u'_q]\!]_\rho, \mathsf{a\vec{p}p}(v'_q, \vec{u}_{m_{q_{r_1}},j_{q_{r_1}}}, ..., \vec{u}_{m_{q_{r_q}},j_{q_{r_q}}})\rangle \,|\,\mathcal{C}\}}{\langle\alpha_1, ..., \alpha_{k_\ell}, \langle k, [\![\vec{u}_k]\!]_\rho\rangle, [\![M'_{\ell,k}]\!]_\eta\rangle} \right.$$

$$\mathcal{C} \equiv \vec{u}_{m_{q_{r_1}},j_{q_{r_1}}} \in [\![\vec{H}_{m_{q_{r_1}},j_{q_{r_1}}}]\!], ..., \vec{u}_{m_{q_{r_q}},j_{q_{r_q}}} \in [\![\vec{H}_{m_{q_{r_q}},j_{q_{r_q}}}]\!],$$

$$\vec{\alpha} \in [\![B_{\ell,1}]\!], ..., [\![B_{\ell,k_\ell}]\!],$$

$$v_{q_r} \in [\![\Pi\vec{u}_{m_{q_r},j_{q_r}} : \vec{H}_{m_{q_r},j_{q_r}}.x_{i_{m_{q_r},j_{q_r}}} \vec{p}\,\vec{w}_{m_{q_r},j_{q_r}}]\!],$$

$$\rho \text{ associates } \vec{\alpha} \text{ to } \vec{a},\ v_{q_r} \text{ to } a'_{q_r},\ r \in \{1, ..., h\},$$

$$v'_q \in [\![(\Pi\vec{u}_{m_{q_{r_c}},j_{q_{r_c}}} : \vec{H}_{m_{q_{r_c}},j_{q_{r_c}}})_c.A'_{n_q}\{\vec{x}_{n_q} : \vec{b}_{n_q}, u'_q\}]\!]_\rho,$$

$$\left. \eta \text{ associates } \vec{\alpha} \text{ to } \vec{a},\ v_{q_r} \text{ to } a'_{q_r},\ r \in \{1, ..., h\}, \text{ and } v'_q \text{ to } X_{n_q}. \right\}$$

We set $[\![\mathsf{fix}\ f_\ell\,\{R\}]\!] := \vec{\mathsf{lam}}(h)$, where h is a function such that

$$h(a_1, ..., a_{k_\ell}, \langle k, \vec{z}_k\rangle) = \mathcal{IF}(\Psi)(a_1, ..., a_{k_\ell}, \langle k, \vec{z}_k\rangle)$$

where $\vec{a}, \langle k, \vec{z}_k\rangle \in [\![\vec{B}_\ell]\!]$.