A Calculus of Constructions with Explicit Subtyping

Ali Assaf[1,2]

1   **INRIA Paris-Rocquencourt, Paris, France**
2   **École Polytechnique, Palaiseau, France**

Abstract

The calculus of constructions can be extended with an infinite hierarchy of universes and cumulative subtyping. Subtyping is usually left implicit in the typing rules. We present an alternative version of the calculus of constructions where subtyping is explicit. We avoid problems related to coercions and dependent types by using the Tarski style of universes and by adding equations to reflect equality.

1   Introduction

The predicative calculus of inductive constructions (PCIC), the theory behind the Coq proof system [20], contains an infinite hierarchy of predicative universes $\mathsf{Type}_0 : \mathsf{Type}_1 : \mathsf{Type}_2 : \ldots$ and an impredicative universe $\mathsf{Prop} : \mathsf{Type}_1$ for propositions, together with a cumulativity relation:

$$\mathsf{Prop} \subseteq \mathsf{Type}_0 \subseteq \mathsf{Type}_1 \subseteq \mathsf{Type}_2 \subseteq \ldots.$$

Cumulativity gives rise to an asymmetric subtyping relation $\leq$ which is used in the subsumption rule:

$$\frac{\Gamma \vdash M : A \qquad A \leq B}{\Gamma \vdash M : B}.$$

Subtyping in Coq is implicit and is handled by the kernel. Type uniqueness does not hold, as a term can have many non-equivalent types, but a notion of *minimal type* can be defined. While subject reduction does hold, the minimal type of a term is not preserved during reduction.

The goal of this paper is to investigate whether it is possible to make subtyping explicit, by inserting explicit coercions such as

$$\uparrow_0 : \mathsf{Type}_0 \to \mathsf{Type}_1$$

and rely on a kernel that uses only the classic conversion rule:

$$\frac{\Gamma \vdash M : A \qquad A \equiv B}{\Gamma \vdash M : B}.$$

In this setting, a well-typed term would have a unique type *up to equivalence* and the type would be preserved during reduction.

Coercions and dependent types

In the presence of dependent types, coercions can interfere with type checking because $\uparrow_0 (A) \not\equiv A$. As a result, terms that were well-typed in a system with implicit subtyping become ill-typed after introducing explicit coercions.

▶ **Example 1.1.** In the context

$$\Gamma = (a : \mathsf{Type}_0, b : \mathsf{Type}_0, f : a \to b, g : \Pi\, c\!:\! \mathsf{Type}_1.\, c)\,,$$

the term $f\,(g\,a)$ is well-typed and has type b:

$$\Gamma \vdash f\,(g\,a) : b\,.$$

In a system with explicit subtyping, before inserting coercions, this term is not well-typed because a has type Type_0 while g has type $\Pi\, c\!:\! \mathsf{Type}_1.\, c$ and $\mathsf{Type}_0 \not\equiv \mathsf{Type}_1$. With an explicit coercion $\uparrow_0\colon \mathsf{Type}_0 \to \mathsf{Type}_1$, the term $g\,(\uparrow_0\,(a))$ has type $\uparrow_0\,(a)$ but $f\,(g\,(\uparrow_0\,(a)))$ is not well-typed because f has type $a \to b$ and $\uparrow_0\,(a) \not\equiv a$.

The easiest way to circumvent this problem is to add a new equation

$$\uparrow_0\,(A) \equiv A.$$

In other words, we erase the coercions to check if two terms are equivalent. While this solution is straightforward, it unfortunately means that we use ill-typed terms. In a system where equivalence is defined by reduction rules, this solution amounts to adding a new reduction rule $\uparrow_0\,(A) \longrightarrow A$, which would completely break subject reduction. A calculus of constructions with explicit subtyping should therefore avoid these rules. The solution to this problem is to use universes *à la Tarski*.

Russell vs. Tarski

There are two ways of presenting universes: the *Russell style* and the *Tarski style*. The first is implicit and is used in the calculus of constructions and in pure type systems [1]. The second is explicit and is mainly used in Martin-Löf's intuitionistic type theory [15]. While the Tarski style is usually regarded as the more fundamental of the two, the Russell style is often used as a practical informal version of the other.

In the Tarski style, we make the distinction between terms and types. Every sort Type_i has a corresponding universe symbol U_i and a decoding function T_i. If A is a term of type U_i, it is not itself a type but a code that represents a type, and $\mathsf{T}_i\,(A)$ is its corresponding type. Types do not have a type and there is a separate judgment for well-formed types. For example, $\pi_i\, x : A.B$ is the term of type U_i that represent the product type $\mathsf{T}_i\,(\pi_i\, x : A.B) \equiv \Pi\, x\!:\! \mathsf{T}_i\,(A).\ \mathsf{T}_i\,(B)$. In this setting, the context Γ of example 1.1 becomes

$$\Gamma = (a : \mathsf{U}_0, b : \mathsf{U}_0, f : \mathsf{T}_0\,(a) \to \mathsf{T}_0\,(b)\,, g : \Pi\, a\!:\! \mathsf{U}_1.\ \mathsf{T}_1\,(a))$$

and with the coercion $\uparrow_0\colon \mathsf{U}_0 \to \mathsf{U}_1$, the term $g\uparrow_0\,(a)$ has type $\mathsf{T}_1\,(\uparrow_0\,(a))$:

$$\Gamma \vdash g\,(\uparrow_0\,(a)) : \mathsf{T}_1\,(\uparrow_0\,(a))\,.$$

By introducing the following equation at the level of types:

$$\mathsf{T}_1\,(\uparrow_0\,(a)) \equiv \mathsf{T}_0\,(a)\,,$$

we get

$$\Gamma \vdash g \left(\uparrow_0 (a) \right) : \mathsf{T}_0 (a)$$

and therefore

$$\Gamma \vdash f \left(g \left(\uparrow_0 (a) \right) \right) : \mathsf{T}_0 (b) .$$

Notice that the equation is well-formed because both members are types, not terms, so they only need to both be well-formed.

Using Tarski-style universes allows for a finer and cleaner distinction between terms and types. Aside from solving the problem above, it is better suited for studying the metatheory, e.g. for building models, justifying proof irrelevance and extraction, etc. where the implicit cumulativity would be a pain. It is often considered as the "fundamental" formalization of universes that should be taken as reference, while the Russell style is more convenient to use in practice, e.g. in proof assistants like Coq and Agda.

However, are the two styles equivalent? This question comes up frequently, for example in recent work homotopy type theory [17]. While the question has already been partially answered for intuitionistic type theory, it has never been studied for the calculus of constructions before. The answer turns out to be *no*, the two styles are not always equivalent. Luo [14] already showed that there is some discrepancy between them. Example 1.1 confirms this idea and suggests that explicit subtyping in the Russell style is not possible. More importantly, depending on the system, the Russell style can sometimes be strictly more expressive than the Tarski style because the equality of types is not reflected at the level of terms.

This discrepancy can be adressed in one of 3 ways:

- either discard the Tarski style and justify taking the Russell style as reference,
- or keep things as they are and argue that the difference is acceptable,
- or find a formulation where the two styles are equivalent.

In this paper, we go for the last option by presenting a Tarski version of the cumulative calculus of constructions that is equivalent to the Russell version. The key is to add enough equations to reflect the equality of types at the level of terms.

Reflecting equalities

Within the Tarski style, there are two main ways of introducing universes known as *universes as full reflections* and *universes as uniform constructions* [16]. The first method requires reflecting equalities, meaning that codes corresponding to equivalent types are equivalent: if $\mathsf{T}_i (A) \equiv \mathsf{T}_i (B)$ then $A \equiv B$. In order to achieve that, additional equations must be introduced such as

$$\uparrow_0 (\pi_0 \, x : A.B) \equiv \pi_1 \, x : (\uparrow_0 (A)) . \uparrow_0 (B) . \tag{1}$$

The second method drops that principle. Instead, $\uparrow_0$ is used as a constructor to inject types from U_0 into U_1. In practice, the usefulness of reflection has not been shown until now and uniform constructions have been preferred [13, 14, 16].

While reflecting equality can be hard to achieve, we argue here that, on the contrary, it is *essential* to be equivalent to the Russell style. First, we note that a term can have multiple translations with the following example.

▶ **Example 1.2.** With Russell-style universes, in the context $\Gamma = (a : \mathsf{Type}_0, b : \mathsf{Type}_0)$, the term $M = \Pi \, x{:}a.\, b$ has type Type_1:

$$\Gamma \vdash_\subset \Pi \, x{:}a.\, b : \mathsf{Type}_1 .$$

With Tarski-style universes, this term can be translated in two different ways as $M_1 = \uparrow_0 (\pi_0 x : a.b)$ and $M_2 = \pi_1 x : \uparrow_0 (a) . \uparrow_0 (b)$:

$$\Gamma \vdash_\uparrow \uparrow_0 (\pi_0 x : a.b) : \mathsf{U}_1, \qquad \Gamma \vdash_\uparrow \pi_1 x : \uparrow_0 (a) . \uparrow_0 (b) : \mathsf{U}_1.$$

When M_1 and M_2 are used as types, this is not a problem because $\mathsf{T}_1 (M_1) \equiv \mathsf{T}_1 (M_2) \equiv \Pi x : \mathsf{T}_0 (a) . \mathsf{T}_0 (b)$. The problem appears when proving higher-order statements about such terms: if p is an abstract predicate of type $\mathsf{Type}_1 \to \mathsf{Type}_1$ then $\mathsf{T}_1 (p M_1) \not\equiv \mathsf{T}_1 (p M_2)$. As a result, we lose some of the expressivity of the Russell-style universes with implicit subtyping.

▶ **Example 1.3** (Necessity of reflecting equalities). In the context

$$\begin{aligned}
\Gamma \quad = \quad & p : \mathsf{Type}_1 \to \mathsf{Type}_1, \\
& q : \mathsf{Type}_1 \to \mathsf{Type}_1, \\
& f : \Pi c : \mathsf{Type}_0.\ p\,c \to q\,c, \\
& g : \Pi a : \mathsf{Type}_1.\ \Pi b : \mathsf{Type}_1.\ p\,(\Pi x : a.\,b) \\
& a : \mathsf{Type}_0, \\
& b : \mathsf{Type}_0,
\end{aligned}$$

the term $f\,(\Pi x : a.\,b)\,(g\,a\,b)$ has type $q\,(\Pi x : a.\,b)$:

$$\Gamma \vdash f\,(\Pi x : a.\,b)\,(g\,a\,b) : q\,(\Pi x : a.\,b)$$

but the corresponding Tarski-style term

$$f\,(\pi_0 x : a.b)\,(g \uparrow_0 (a) \uparrow_0 (b))$$

is ill-typed because $\mathsf{T}_1 (p\,(\pi_1 x : \uparrow_0 (a) . \uparrow_0 (b))) \not\equiv \mathsf{T}_1 (p\,(\uparrow_0 (\pi_0 x : a.b)))$. The type corresponding to $q\,(\Pi x : a.\,b)$ is not provable in the Tarski style without further equations!

Reflecting equality with Equation 1 solves this problem by ensuring that any type has a single term representation up to equivalence. While the equations needed for the predicative universes Type_i have been known for some time [13, 16], the equations for the impredicative universe Prop are less obvious and have not been studied before.

Related work

Geuvers and Wiedijk [6] presented a dependently typed system with explicit conversions. In that system, every conversion is annotated inside the term and there is no implicit conversion rule. Terms have a unique type instead of a unique type *up to equivalence*. To solve the issue of dependent types mentioned above, they rely on an erasure equation similar to $\uparrow_0 (A) \equiv A$. They also present a variant of the system which does not go through ill-typed terms, but that uses typed heterogeneous equality judgments instead.

In Martin-Löf's intuitionistic type theory, Palmgren [16] and Luo [13] formalized systems with a cumulative hierarchy of predicative universes U_i and an impredicative universe Prop. They both use the Tarski style of universes, which distinguishes between a term A of type U_i and the type $\mathsf{T}_i (A)$ that it represents, and which allows them to introduce well-typed equations such as Equation 1. However, they only show how to reflect equality for the predicative universes. As a result, these systems lose some of the expressivity of Russell-style universes with implicit subtyping and are therefore incomplete. Similarly, Herbelin and Spiwack [9] presented a variant of the calculus of constructions with one Type universe and explicit coercions from Prop to Type but they do not reflect equality.

Cousineau and Dowek [5] showed how to embed functional pure type systems in the $\lambda\Pi$-calculus modulo, a logical framework that can be seen as a subset of Martin-Löf's framework where equations are expressed as rewrite rules. When the rewrite system is confluent and strongly normalizing, the $\lambda\Pi$-calculus modulo becomes a decidable version of Martin-Löf's framework. Burel and Boespflug [4] used this embedding to formalize and translate Coq proofs to Dedukti [18], a type-checker based on the $\lambda\Pi$-calculus modulo rewriting, but they handle neither the universe hierarchy nor cumulativity.

Contribution

We present a formulation of the cumulative calculus of constructions where subtyping is explicit. By using Tarski-style universes, we are able to solve the problems related to coercions and dependent types. We show that reflecting equality is necessary for the Tarski style to be equivalent to the Russell style, thus settling an old question for good. Our system fully reflects equality: by introducing additional equations between terms, we ensure that every well-typed term in the original system has a unique representation up to equivalence in the new system.

To our knowledge, this is the first time such work has been done for the cumulative calculus of constructions, which includes both a cumulative hierarchy of predicative universes and an impredicative universe. We also show how to orient the equations into rewrite rules so that equivalence can be defined as a congruence of reduction steps. In summary, this paper answers the question:

What is the system that corresponds to the question mark in Figure 1?
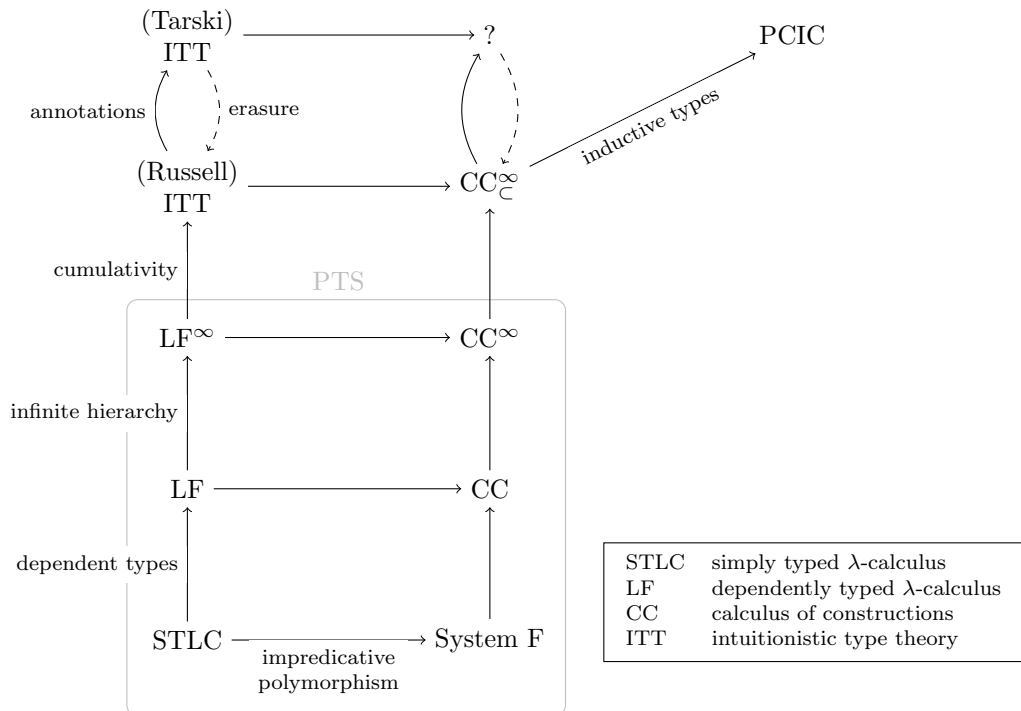
Figure 1 Type theory zoo.

Outline

In Section 2, we present a subset of the original PCIC that we call the *cumulative calculus of constructions* ($CC_{\subseteq}^{\infty}$). It will serve as the reference to which systems with explicit subtyping will be compared. In Section 3, we present our system with explicit subtyping called the *explicit cumulative calculus of constructions* ($CC_{\uparrow}^{\infty}$). We show exactly which equations are needed to reflect equality. In Section 4, we show that it is complete with respect to $CC_{\subseteq}^{\infty}$ by defining a translation and proving that it preserves typing. Finally, in Section 5, we show how to transform the equations into rewrite rules so that the system can be implemented in practice.

2 The cumulative calculus of constructions

We consider a subset of PCIC that does not contain inductive types so that we can focus entirely on universes and cumulativity. This system was introduced by Luo [11] under the name $CC_{\subseteq}^{\infty}$, although with a slightly different presentation. We will also refer to it as the *cumulative calculus of constructions.* It is an extension of the calculus of constructions (CC) with universes and subtyping. It is related to the *extended calculus of constructions* (ECC) [12] but it does not contain sum types. It is also related to the *generalized calculus of constructions* [7, 8] but that one is not fully cumulative as it lacks the $\mathsf{Prop} \subseteq \mathsf{Type}_0$ inclusion[1].

Our presentation differs slightly from Luo's original presentation. The main differences are that $\mathsf{Prop} : \mathsf{Type}_1$ instead of $\mathsf{Prop} : \mathsf{Type}_0$ and that all the rules ($\mathsf{Type}_i, \mathsf{Type}_j, \mathsf{Type}_{\max(i,j)}$) are allowed, as done in Coq. The reason for having Prop in Type_1 instead of Type_0 is mainly historical and not of importance for the purposes of this paper. All of Luo's results still hold for our presentation.

Syntax

The syntax is defined as usual for type theories based on pure type systems. For further background, we refer the reader to [1, 20].

▶ **Definition 2.1** (Syntax).

$$
\begin{array}{llll}
\text{variables} & x, y, \alpha, \beta & \in & \mathcal{V} \\
\text{sorts} & s & \in & \mathcal{S} = \{\mathsf{Prop}\} \cup \{\mathsf{Type}_i \mid i \in \mathbb{N}\} \\
\text{terms} & M, N, A, B & \in & \mathcal{T} ::= x \mid s \mid \Pi\, x{:}A.\, B \mid \lambda\, x{:}A.\, M \mid M\, N \\
\text{contexts} & \Gamma, \Delta & \in & \mathcal{C} ::= .\mid \Gamma, x : A
\end{array}
$$

Typing

Since the pure type system at the core of $CC_{\subseteq}^{\infty}$ is functional and complete, we can define its axiom relation $(s_1 : s_2) \in \mathcal{A}$ as a function $\mathcal{A}(s_1)$ and its product rule relation $(s_1, s_2, s_3) \in \mathcal{R}$ as a function $\mathcal{R}(s_1, s_2)$. The cumulativity relation $\subseteq$ can be defined as the reflexive transitive closure of $\mathsf{Prop} \subseteq \mathsf{Type}_0$ and $\mathsf{Type}_i \subseteq \mathsf{Type}_{i+1}$. In order to give a uniform presentation, we define the following operations on sorts.

---

[1] The $\mathsf{Prop} \subseteq \mathsf{Type}_0$ barrier is often the main difficulty in the metatheory of systems with cumulativity.

▶ **Definition 2.2** (Sort operations). The unary operations $\mathcal{A}$ and $\mathcal{N}$ and the binary operation $\mathcal{R}$ are defined as follows.

$$
\begin{array}{llll}
\mathcal{A}(\mathsf{Prop}) & = & \mathsf{Type}_1 & \mathcal{R}(\mathsf{Prop}, \mathsf{Prop}) & = & \mathsf{Prop} \\
\mathcal{A}(\mathsf{Type}_i) & = & \mathsf{Type}_{i+1} & \mathcal{R}(\mathsf{Type}_i, \mathsf{Prop}) & = & \mathsf{Prop} \\
\mathcal{N}(\mathsf{Prop}) & = & \mathsf{Type}_0 & \mathcal{R}(\mathsf{Prop}, \mathsf{Type}_j) & = & \mathsf{Type}_j \\
\mathcal{N}(\mathsf{Type}_i) & = & \mathsf{Type}_{i+1} & \mathcal{R}(\mathsf{Type}_i, \mathsf{Type}_j) & = & \mathsf{Type}_{\max(i,j)}
\end{array}
$$

The cumulativity relation $\subseteq$ is the reflexive transitive closure of $\mathcal{N}$.

To simplify the presentation, we also decouple the context well-formedness judgment from the typing judgment to break the mutual dependency between the two. This formulation is equivalent to the original one but is better suited for proofs by induction. For more information on this common technique, we refer the reader to [21].

▶ **Definition 2.3** (Typing). A term M *has type* A in the context Γ when the judgment $\Gamma \vdash_\subset M : A$ can be derived from the following rules.

$$
\frac{(x : A) \in \Gamma}{\Gamma \vdash_\subset x : A} \; variable
$$

$$
\frac{}{\Gamma \vdash_\subset s : \mathcal{A}(s)} \; sort
$$

$$
\frac{\Gamma \vdash_\subset A : s}{\Gamma \vdash_\subset A : \mathcal{N}(s)} \; cumulativity
$$

$$
\frac{\Gamma \vdash_\subset A : s_1 \qquad x \notin \Gamma \qquad \Gamma, x : A \vdash_\subset B : s_2}{\Gamma \vdash_\subset \Pi\, x{:}A.\, B : \mathcal{R}(s_1, s_2)} \; product
$$

$$
\frac{\Gamma \vdash_\subset A : s \qquad x \notin \Gamma \qquad \Gamma, x : A \vdash_\subset M : B}{\Gamma \vdash_\subset \lambda\, x{:}A.\, M : \Pi\, x{:}A.\, B} \; abstraction
$$

$$
\frac{\Gamma \vdash_\subset M : \Pi\, x{:}A.\, B \qquad \Gamma \vdash_\subset N : A}{\Gamma \vdash_\subset M\, N : \{N/x\}B} \; application
$$

$$
\frac{\Gamma \vdash_\subset M : A \qquad \Gamma \vdash_\subset B : s \qquad A \equiv B}{\Gamma \vdash_\subset M : B} \; conversion
$$

A context Γ *is well-formed* when the judgment $\mathsf{WF}_\subset(\Gamma)$ can be derived from the following rules.

$$
\frac{}{\mathsf{WF}_\subset(.)} \; empty
$$

$$
\frac{\mathsf{WF}_\subset(\Gamma) \qquad x \notin \Gamma \qquad \Gamma \vdash_\subset A : s}{\mathsf{WF}_\subset(\Gamma, x : A)} \; declaration
$$

We write $\Gamma \vdash M : A$ and $\mathsf{WF}(\Gamma)$ instead of $\Gamma \vdash_\subset M : A$ and $\mathsf{WF}_\subset(\Gamma)$ when there is no ambiguity.

▶ Remark. The system $\mathrm{CC}^\infty_\mathbb{C}$ is *not* equivalent to a non-functional traditional PTS [1]. Indeed, even if the PTS had $\mathsf{Prop} : \mathsf{Type}_i$ for all $i \in \mathbb{N}$ as axioms, the term $\lambda\,a\!:\!\mathsf{Prop}.\,(\lambda\,a\!:\!\mathsf{Type}_0.\,a)\,a$ would not be well-typed, while it has type $\mathsf{Prop} \to \mathsf{Type}_0$ with cumulativity in $\mathrm{CC}^\infty_\mathbb{C}$.

Unlike in Coq, cumulativity in $\mathrm{CC}^\infty_\mathbb{C}$ is not presented as a subtyping rule. This makes it very slightly weaker because product covariance does not hold: if $M : \Pi\,x\!:\!A.\,\mathsf{Type}_i$ then it does *not* follow that $M : \Pi\,x\!:\!A.\,\mathsf{Type}_i$. Luo [11] covers this difference in great detail. Since our main focus is the difficulties that arise from universe cumulativity, this is not an issue for us. The notion of subtyping is still useful however. We will define it and use it to caracterize minimal typing.

Minimal typing

The system $\mathrm{CC}^\infty_\mathbb{C}$ does not satisfy the uniqueness of types because a term can have multiple non-equivalent types. However, it does have a notion of *minimal type*.

▶ **Definition 2.4** (Subtyping). A term A is a subtype of B when the relation $A \leq B$ can be derived from the following rules where $\equiv$ is the usual β-equivalence relation.

$$\frac{A \equiv B}{A \leq B}\ \textit{reflexivity} \qquad \frac{A \leq B \qquad B \leq C}{A \leq C}\ \textit{transitivity}$$

$$\frac{s_1 \subseteq s_2}{s_1 \leq s_2}\ \textit{cumulativity} \qquad \frac{B \leq C}{\Pi\,x\!:\!A.\,B \leq \Pi\,x\!:\!A.\,C}\ \textit{covariance}$$

▶ **Definition 2.5.** A term M *has minimal type* A in the context Γ when $\Gamma \vdash M : A$ and for all B, $\Gamma \vdash M : B$ implies $A \leq B$. We write $\Gamma \vdash_{\mathsf{m}} M : A$.

▶ **Theorem 2.6** (Existence of minimal types). *If $\Gamma \vdash_\mathbb{C} M : B$ then there is an A such that $\Gamma \vdash_{\mathsf{m}} M : A$.*

Proof. The details can be found in Luo's paper, pages 16–17, [11]. ◀

This notion will be useful when we define our translation. However, note that while minimal types always exist, they are *not* preserved by substitution and β-reduction, as shown in the following example.

▶ **Example 2.7.** In the context $\Gamma = (a : \mathsf{Type}_0, x : \mathsf{Type}_1)$, the term x has minimal type Type_1:

$$a : \mathsf{Type}_0, x : \mathsf{Type}_1 \vdash_{\mathsf{m}} x : \mathsf{Type}_1$$

but the term $\{a/x\}x = a$ has minimal type $\mathsf{Type}_0 \not\equiv \{a/x\}\mathsf{Type}_1$:

$$a : \mathsf{Type}_0 \vdash_{\mathsf{m}} a : \mathsf{Type}_0.$$

3 Explicit subtyping

In this section, we define the *explicit cumulative calculus of constructions* ($\mathrm{CC}^\infty_\uparrow$) where subtyping is explicit. The syntax is extended to include coercions and to make the distinction between terms and types. We introduce additional equations in the equivalence relation $\equiv$ and give the typing rules based on the rules of $\mathrm{CC}^\infty_\mathbb{C}$.

Syntax

For each sort s, we introduce the universe symbol U_s. A term A of type U_s is a code that represents a type in that universe. The decoding function $\mathsf{T}_s(A)$ gives the corresponding type. We extend the syntax with the codes u_s and $\pi_{s_1,s_2} x : A.B$ that represent the type U_s in the universe $\mathsf{U}_{\mathcal{A}(s)}$ and the product type in the universe $\mathsf{U}_{\mathcal{R}(s_1,s_2)}$ respectively. The universe hierarchy being cumulative, each universe contains codes for all the types of the previous universe using the coercion[2] $\uparrow_s(A)$.

▶ **Definition 3.1** (Syntax).

$$
\begin{array}{rclcl}
\text{terms} & M, N, A, B & \in & \mathcal{T} & ::= \quad x \mid \lambda\, x{:}A.\, M \mid M\, N \\
& & & & \quad\;\; \mid \mathsf{u}_s \mid \uparrow_s(A) \mid \pi_{s_1,s_2}\, x : A.B \\
& & & & \quad\;\; \mid \mathsf{U}_s \mid \mathsf{T}_s(A) \mid \Pi\, x{:}A.\, B \\
\text{contexts} & \Gamma, \Delta & \in & \mathcal{C} & ::= \quad .\mid \Gamma, x : A
\end{array}
$$

We write U_i, $\mathsf{T}_i(A)$, u_i, $\uparrow_i(A)$, and $\pi_{i,j} x : A.B$ instead of $\mathsf{U}_{\mathsf{Type}_i}$, $\mathsf{T}_{\mathsf{Type}_i}(A)$, $\mathsf{u}_{\mathsf{Type}_i}$, $\uparrow_{\mathsf{Type}_i}(A)$, and $\pi_{\mathsf{Type}_i,\mathsf{Type}_j} x : A.B$ respectively. When $s_1 \subseteq s_2$ and $s_2 = \mathcal{N}^i(s_1)$, we write $\uparrow_{s_1}^{s_2}(A)$ for the term

$$
\uparrow_{\mathcal{N}^{i-1}(s)} \left(\cdots \uparrow_{\mathcal{N}(s)} (\uparrow_s(A)) \right).
$$

For example, $\uparrow_1^3(A) = \uparrow_2(\uparrow_1(A))$ and $\uparrow_{\mathsf{Prop}}^1(A) = \uparrow_0\left(\uparrow_{\mathsf{Prop}}(A)\right)$.

▶ **Remark**. It is possible to completely split the syntax into distinct categories for types (U_s, $\mathsf{T}_s(A)$, $\Pi\, x{:}A.\,B$) and terms. This is one of the advantages of using the Tarski style and it could help simplify the theoretical studies of the calculus of constructions, where the lack of syntactic stratification between terms and types is cumbersome. However, it is not necessary for our purposes so we will not do that here.

Equivalence

Because we are using Tarski-style universes, we need to consider additional equations besides β-equivalence. For now, we just state the equations that are needed and assume a congruence relation $\equiv$ that satisfies those equations. We do not worry about the algorithmic aspect. Later in Section 5, we show how to define $\equiv$ as the usual congruence induced by a set of reduction rules.

In addition to β-equivalence:

$$
(\lambda\, x{:}A.\, M)\, N \equiv \{N/x\}M,
$$

we need equations to describe the behaviour of the decoding function $\mathsf{T}_s(A)$. These are the same as in intuitionistic type theory:

$$
\begin{array}{rcl}
\mathsf{T}_{\mathcal{A}(s)}(\mathsf{u}_s) & \equiv & \mathsf{U}_s \\
\mathsf{T}_{\mathcal{N}(s)}(\uparrow_s(A)) & \equiv & \mathsf{T}_s(A) \\
\mathsf{T}_{\mathcal{R}(s_1,s_2)}(\pi_{s_1,s_2}\, x : A.B) & \equiv & \Pi\, x{:}\mathsf{T}_{s_1}(A).\, \mathsf{T}_{s_2}(B).
\end{array}
$$

Finally, we also need equations that reflect equality to ensure that each term of a given type has a unique representation.

---

[2] One can also view $\uparrow_s(A)$ as the code representing $\mathsf{T}_s(A)$ in the universe $\mathsf{U}_{\mathcal{N}(s)}$.

▮ **Table 1** Different typing derivations for same terms.

$\mathrm{CC}_{\subseteq}^{\infty}$ typing derivation	$\mathrm{CC}_{\uparrow}^{\infty}$ term representation
$$\dfrac{\dfrac{A : \mathsf{Type}_i \qquad x : A \vdash B : \mathsf{Type}_i}{\Pi\,x{:}A.\,B : \mathsf{Type}_i}}{\Pi\,x{:}A.\,B : \mathsf{Type}_{i+1}}$$	$\uparrow_i (\pi_{i,i}\,x : A.B)$
$$\dfrac{\dfrac{A : \mathsf{Type}_i}{A : \mathsf{Type}_{i+1}} \qquad \dfrac{x : A \vdash B : \mathsf{Type}_i}{x : A \vdash B : \mathsf{Type}_{i+1}}}{\Pi\,x{:}A.\,B : \mathsf{Type}_{i+1}}$$	$\pi_{i+1,i+1}\,x : \uparrow_i (A).\uparrow_i (B)$
$$\dfrac{A : \mathsf{Type}_i \qquad x : A \vdash B : \mathsf{Prop}}{\Pi\,x{:}A.\,B : \mathsf{Prop}}$$	$\pi_{i,\mathsf{Prop}}\,x : A.B$
$$\dfrac{\dfrac{A : \mathsf{Type}_i}{A : \mathsf{Type}_{i+1}} \qquad x : A \vdash B : \mathsf{Prop}}{\Pi\,x{:}A.\,B : \mathsf{Prop}}$$	$\pi_{i+1,\mathsf{Prop}}\,x : \uparrow_i (A).B$
$$\dfrac{A : \mathsf{Type}_i \qquad \dfrac{x : A \vdash B : \mathsf{Prop}}{x : A \vdash B : \mathsf{Type}_0}}{\Pi\,x{:}A.\,B : \mathsf{Type}_i}$$	$\pi_{i,0}\,x : A.\uparrow_{\mathsf{Prop}} (B)$
$$\dfrac{\dfrac{\dfrac{\dfrac{A : \mathsf{Type}_i \qquad x : A \vdash B : \mathsf{Prop}}{\Pi\,x{:}A.\,B : \mathsf{Prop}}}{\Pi\,x{:}A.\,B : \mathsf{Type}_0}}{\vdots}}{\Pi\,x{:}A.\,B : \mathsf{Type}_i}$$	$\uparrow_{\mathsf{Prop}}^{i} (\pi_{i,\mathsf{Prop}}\,x : A.B)$

Which equations are needed to reflect equality? The answer lies in the multiplicity of typing derivations in $\mathrm{CC}_{\subseteq}^{\infty}$. For example, the product $\Pi\,x{:}A.\,B$ of minimal type Type_0 can be typed at the level Type_1 in two different ways, each giving a different term in $\mathrm{CC}_{\uparrow}^{\infty}$:

$$\dfrac{\dfrac{A : \mathsf{Type}_0 \qquad x : A \vdash B : \mathsf{Type}_0}{\Pi\,x{:}A.\,B : \mathsf{Type}_0}}{\Pi\,x{:}A.\,B : \mathsf{Type}_1} \qquad \uparrow_1 (\pi_{0,0}\,x : A.B)$$

$$\dfrac{\dfrac{A : \mathsf{Type}_0}{A : \mathsf{Type}_1} \qquad \dfrac{x : A \vdash B : \mathsf{Type}_0}{x : A \vdash B : \mathsf{Type}_1}}{\Pi\,x{:}A.\,B : \mathsf{Type}_1} \qquad \pi_{1,1}\,x : \uparrow_0 (A).\uparrow_0 (B)$$

The equivalence relation must therefore take this multiplicity into account. Table 1 lists the different typing derivations that can occur for product types. A careful analysis yields the

following equations:

$$
\begin{aligned}
\pi_{\mathcal{N}(s),\mathsf{Prop}}\, x : \uparrow_s (A)\, .B &\equiv \pi_{s,\mathsf{Prop}}\, x : A.B \\
\pi_{\mathsf{Prop},\mathcal{N}(s)}\, x : A.\uparrow_s (B) &\equiv \uparrow_s (\pi_{\mathsf{Prop},s}\, x : A.B) \\
\pi_{0,j}\, x : \uparrow_{\mathsf{Prop}} (A)\, .B &\equiv \pi_{\mathsf{Prop},j}\, x : A.B \\
\pi_{i,0}\, x : A.\uparrow_{\mathsf{Prop}} (B) &\equiv \uparrow_{\mathsf{Prop}}^i (\pi_{i,\mathsf{Prop}}\, x : A.B) \\
\pi_{i+1,j+1}\, x : \uparrow_i (A)\, .B &\equiv \pi_{i,j+1}\, x : A.B && \text{when } i \le j \\
\pi_{i+1,j+1}\, x : \uparrow_i (A)\, .B &\equiv \uparrow_i (\pi_{i,j+1}\, x : A.B) && \text{when } i > j \\
\pi_{i+1,j+1}\, x : A.\uparrow_j (B) &\equiv \pi_{i+1,j}\, x : A.B && \text{when } i \ge j \\
\pi_{i+1,j+1}\, x : A.\uparrow_j (B) &\equiv \uparrow_j (\pi_{i+1,j}\, x : A.B) && \text{when } i < j.
\end{aligned}
$$

It turns out we can express these concisely using the $\uparrow_{s_1}^{s_2} (A)$ notation:

$$
\begin{aligned}
\pi_{\mathcal{N}(s_1),s_2}\, x : \uparrow_{s_1} (A)\, .B &\equiv \uparrow_{\mathcal{R}(s_1,s_2)}^{\mathcal{R}(\mathcal{N}(s_1),s_2)} (\pi_{s_1,s_2}\, x : A.B) \\
\pi_{s_1,\mathcal{N}(s_2)}\, x : A.\uparrow_{s_2} (B) &\equiv \uparrow_{\mathcal{R}(s_1,s_2)}^{\mathcal{R}(s_1,\mathcal{N}(s_2))} (\pi_{s_1,s_2}\, x : A.B) .
\end{aligned}
$$

▶ **Definition 3.2** (Equivalence). The equivalence relation $\equiv$ is the smallest congruence relation that satisfies the following equations:

$$
\begin{aligned}
(\lambda x : A.\, M)\, N &\equiv \{N/x\}M \\
\mathsf{T}_{\mathcal{A}(s)} (\mathsf{u}_s) &\equiv \mathsf{U}_s \\
\mathsf{T}_{\mathcal{N}(s)} (\uparrow_s (A)) &\equiv \mathsf{T}_s (A) \\
\mathsf{T}_{\mathcal{R}(s_1,s_2)} (\pi_{s_1,s_2}\, x : A.B) &\equiv \Pi\, x : \mathsf{T}_{s_1} (A)\, .\, \mathsf{T}_{s_2} (B) \\
\pi_{\mathcal{N}(s_1),s_2}\, x : \uparrow_{s_1} (A)\, .B &\equiv \uparrow_{\mathcal{R}(s_1,s_2)}^{\mathcal{R}(\mathcal{N}(s_1),s_2)} (\pi_{s_1,s_2}\, x : A.B) \\
\pi_{s_1,\mathcal{N}(s_2)}\, x : A.\uparrow_{s_2} (B) &\equiv \uparrow_{\mathcal{R}(s_1,s_2)}^{\mathcal{R}(s_1,\mathcal{N}(s_2))} (\pi_{s_1,s_2}\, x : A.B) .
\end{aligned}
$$

Typing

To make the distinction between types and terms, we introduce an additional judgment $\Gamma \vdash_\uparrow \mathsf{type}\,(A)$ to capture the property that a type is well-formed. The derivation rules mirror the rules of $\mathrm{CC}_\subseteq^\infty$.

▶ **Definition 3.3** (Typing). A term M *has type* A in the context Γ when the judgment $\Gamma \vdash_\uparrow M : A$ can be derived from the following rules, and a term A *is a type* in the context Γ when the judgment $\Gamma \vdash_\uparrow \mathsf{type}\,(A)$ can be derived from the following rules:

$$
\frac{(x : A) \in \Gamma}{\Gamma \vdash_\uparrow x : A}\ variable
$$

$$
\frac{}{\Gamma \vdash_\uparrow \mathsf{type}\,(\mathsf{U}_s)}\ sort\text{-}type
\qquad
\frac{\Gamma \vdash_\uparrow A : \mathsf{U}_s}{\Gamma \vdash_\uparrow \mathsf{type}\,(\mathsf{T}_s (A))}\ decode\text{-}type
$$

$$
\frac{\Gamma \vdash_\uparrow \mathsf{type}\,(A) \qquad x \notin \Gamma \qquad \Gamma, x : A \vdash_\uparrow \mathsf{type}\,(B)}{\Gamma \vdash_\uparrow \mathsf{type}\,(\Pi\, x : A.\, B)}\ product\text{-}type
$$

$$
\frac{}{\Gamma \vdash_\uparrow \mathsf{u}_s : \mathsf{U}_{\mathcal{A}(s)}}\ sort
\qquad
\frac{\Gamma \vdash_\uparrow A : \mathsf{U}_s}{\Gamma \vdash_\uparrow \uparrow_s (A) : \mathsf{U}_{\mathcal{N}(s)}}\ cumulativity
$$

$$
\frac{\Gamma \vdash_\uparrow A : \mathsf{U}_{s_1} \qquad x \notin \Gamma \qquad \Gamma, x : \mathsf{T}_{s_1} (A) \vdash_\uparrow B : \mathsf{U}_{s_2}}{\Gamma \vdash_\uparrow \pi_{s_1,s_2}\, x : A.B : \mathsf{U}_{\mathcal{R}(s_1,s_2)}}\ product
$$

$$\frac{\Gamma \vdash_\uparrow \text{type}\,(A) \qquad x \notin \Gamma \qquad \Gamma, x : A \vdash_\uparrow M : B}{\Gamma \vdash_\uparrow \lambda\,x : A.\,M : \Pi\,x : A.\,B}\; abstraction$$

$$\frac{\Gamma \vdash_\uparrow M : \Pi\,x : A.\,B \qquad \Gamma \vdash_\uparrow N : A}{\Gamma \vdash_\uparrow M\,N : \{N/x\}B}\; application$$

$$\frac{\Gamma \vdash_\uparrow M : A \qquad \Gamma \vdash_\uparrow \text{type}\,(B) \qquad A \equiv B}{\Gamma \vdash_\uparrow M : B}\; conversion$$

A context Γ *is well-formed* when the judgment $\mathsf{WF}_\uparrow(\Gamma)$ can be derived from the following rules:

$$\frac{}{\mathsf{WF}_\uparrow(.)}\; empty \qquad \frac{\mathsf{WF}_\uparrow(\Gamma) \qquad x \notin \Gamma \qquad \Gamma \vdash_\uparrow \text{type}\,(A)}{\mathsf{WF}_\uparrow(\Gamma, x : A)}\; declaration$$

We write $\Gamma \vdash M : A$, $\Gamma \vdash \text{type}\,(A)$, and $\mathsf{WF}\,(\Gamma)$ instead of $\Gamma \vdash_\uparrow M : A$, $\Gamma \vdash_\uparrow \text{type}\,(A)$, and $\mathsf{WF}_\uparrow(\Gamma)$ when there is no ambiguity.

▶ Remark. The equations of Definition 3.2 are well-formed because the left and right side of each equation are either both types or both terms of the same type. In particular, the last two are well-typed because $\mathcal{R}(s_1, s_2) \subseteq \mathcal{R}(\mathcal{N}(s_1), s_2)$ and $\mathcal{R}(s_1, s_2) \subseteq \mathcal{R}(s_1, \mathcal{N}(s_2))$ for all $s_1, s_2 \in \mathcal{S}$.

▶ **Theorem 3.4** (Type uniqueness). *If $\Gamma \vdash_\uparrow M : A$ and $\Gamma \vdash_\uparrow M : B$ then $A \equiv B$.*

Proof. By induction over the derivations of $\Gamma \vdash_\uparrow M : A$ and $\Gamma \vdash_\uparrow M : B$. We can eliminate conversion rules until we hit a non-conversion rule, in which case we remove the rule from both derivations at the same time. ◀

Erasure

Systems with Tarski-style universes are related to systems with Russell-style universes in a precise sense: we can define an erasure function $|M|$ such that the erasure of a well-typed term in the Tarski style is well-typed in the Russell style. In our setting, this function shows that $\mathrm{CC}_\uparrow^\infty$ is sound with respect to $\mathrm{CC}_\subseteq^\infty$.

▶ **Definition 3.5** (Erasure). The *term erasure* $|M|$, the *type erasure* $\|A\|$, and the *context erasure* $\|\Gamma\|$ are defined as follows.

$$
\begin{aligned}
|x| &= x \\
|\mathsf{u}_s| &= s \\
|{\uparrow_s}\,(A)| &= |A| \\
|\pi_{s_1, s_2}\,x : A.B| &= \Pi\,x : |A|\,.\,|B| \\
|\lambda\,x : A.\,M| &= \lambda\,x : \|A\|\,.\,|M| \\
|M\,N| &= |M|\,|N| \\
\\
\|\mathsf{U}_s\| &= s \\
\|\mathsf{T}_s\,(A)\| &= |A| \\
\|\Pi\,x : A.\,B\| &= \Pi\,x : \|A\|\,.\,\|B\| \\
\\
\|.\| &= . \\
\|\Gamma, x : A\| &= \|\Gamma\|, x : \|A\|
\end{aligned}
$$

▶ **Lemma 3.6.** *For all B, x, N, $|\{N/x\}B| = \{|N|/x\}|B|$.*

Proof. By induction on B.   ◀

▶ **Theorem 3.7** (Soundness). *If $\Gamma \vdash_\uparrow M : A$ then $\|\Gamma\| \vdash_\subset |M| : \|A\|$. If $\Gamma \vdash_\uparrow \mathsf{type}\,(A)$ then $\|\Gamma\| \vdash_\subset \|A\| : s$ for some sort s. If $\mathsf{WF}_\uparrow(\Gamma)$ then $\mathsf{WF}_\subset(\|\Gamma\|)$.*

Proof. By induction on the derivations in $\mathrm{CC}_\uparrow^\infty$, using Lemma 3.6 for the application rule.   ◀

4   Completeness

In this section, we show that the new system is complete with respect to the original system, meaning that it can express all well-typed terms. We define a function that translates any well-typed term of $\mathrm{CC}_\subset^\infty$ into a term of $\mathrm{CC}_\uparrow^\infty$ and we prove that this translation preserves typing.

Translation

When translating a term, we want to choose the representation that has the minimal type. However, we sometimes need to lift some subterms, such as the argument of applications, in order to get a well-typed term. We therefore define two translations: $[M]_\Gamma$ which translates M according to its minimal type and $[M]_{\Gamma \vdash A}$ which translates M as a term of type A. Finally, since we distinguish between terms and types, we also define $[\![A]\!]_\Gamma$, the translation of A as a type.

▶ **Definition 4.1** (Translation). Let Γ be a well-formed context, A and B be well-formed types in Γ, and M be a well-typed term in Γ such that $\Gamma \vdash_\mathsf{m} M : A$ and $\Gamma \vdash_\subset M : B$. The *term translation* $[M]_\Gamma$, the *cast translation* $[M]_{\Gamma \vdash B}$, and the *type translation* $[\![A]\!]_\Gamma$ are mutually defined as follows.

Term translation

$$
\begin{aligned}
[s]_\Gamma &= \mathsf{u}_s \\
[x]_\Gamma &= x \\
[\Pi\, x{:}A'.\, B']_\Gamma &= \pi_{s_1,s_2}\, x : [A']_\Gamma\,.\,[B']_{\Gamma,x:A'} \\
&\quad \text{where } \Gamma \vdash_\mathsf{m} A' : s_1 \\
&\quad \text{and } \Gamma, x : A' \vdash_\mathsf{m} B' : s_2 \\
[\lambda\, x{:}A'.\, M']_\Gamma &= \lambda\, x{:}[\![A']\!]_\Gamma\,.\,[M']_{\Gamma,x:A'} \\
[M'\, N']_\Gamma &= [M']_\Gamma\,[N']_{\Gamma \vdash A'} \\
&\quad \text{where } \Gamma \vdash_\mathsf{m} M' : \Pi\, x{:}A'.\, B'
\end{aligned}
$$

Cast translation

$$
\begin{aligned}
[M]_{\Gamma \vdash B} &= [M]_\Gamma \\
&\quad \text{when } A \equiv B \\
[M]_{\Gamma \vdash B} &= \uparrow_{s_1}^{s_2}\,([M]_\Gamma) \\
&\quad \text{when } A \equiv s_1 \subseteq s_2 \equiv B
\end{aligned}
$$

Type translation

$$
\begin{aligned}
[\![A]\!]_\Gamma &= \mathsf{T}_s\,([A]_\Gamma) \\
&\quad \text{where } \Gamma \vdash_\mathsf{m} A : s
\end{aligned}
$$

The context translation $[\![\Gamma]\!]$ where $\mathsf{WF}_{\sqsubset}(\Gamma)$ is defined as follows.

Context translation

$$
\begin{aligned}
[\![.]\!] &= \ . \\
[\![\Gamma, x : A]\!] &= [\![\Gamma]\!], x : [\![A]\!]_\Gamma
\end{aligned}
$$

We will write $[M]$, $[M]_{\vdash C}$, and $[\![A]\!]$ instead of $[M]_\Gamma$, $[M]_{\Gamma \vdash C}$, and $[\![A]\!]_\Gamma$ when unambiguous.

Substitution preservation

A key property for proving the preservation of typing by the translation is that it preserves substitution. If $\Gamma, x : A \vdash_{\sqsubset} M : B$ and $\Gamma \vdash_{\sqsubset} N : A$ then the translation of the substitution is the same as the substitution of the translation. However, the naive statement $\{[N]/x\}[M] \equiv [\{N/x\}M]$ is not true. First, x has type A while N has some minimal type $C \leq A$ so we need to use the cast translation $[N]_{\vdash A}$. Second, the minimal typing is not preserved by substitution, as we showed in Example 2.7. Therefore we also need to fix the type of $\{N/x\}M$ using the cast translation $[\{N/x\}M]_{\vdash \{N/x\}B}$.

▶ **Lemma 4.2** (Translation distributivity). *The translation satisfies the following properties:*
- *For all $s \in \mathcal{S}$, $[\![s]\!] \equiv \mathsf{U}_s$.*
- *If $\Gamma \vdash_{\sqsubset} A : s_1$ and $\Gamma, x : A \vdash_{\sqsubset} B : s_2$ then $[\![\Pi\, x{:}A.\, B]\!] \equiv \Pi\, x{:}[\![A]\!] .\, [\![B]\!]$.*
- *If $\Gamma \vdash_{\sqsubset} A : s_1$ and $\Gamma, x : A \vdash_{\sqsubset} B : s_2$ then*

$$
[\Pi\, x{:}A.\, B]_{\vdash \mathcal{R}(s_1, s_2)} \equiv \pi_{s_1, s_2}\, x : [A]_{\vdash s_1} .\, [B]_{\vdash s_2} .
$$

- *If $\Gamma \vdash_{\sqsubset} A : s_1$ and $\Gamma, x : A \vdash_{\sqsubset} M : B$ then*

$$
[\lambda\, x{:}A.\, M]_{\vdash \Pi\, y{:}A.\, B} \equiv \lambda\, x{:}[\![A]\!] .\, [M]_{\vdash B} .
$$

- *If $\Gamma \vdash_{\sqsubset} M : \Pi\, x{:}A.\, B$ and $\Gamma \vdash_{\sqsubset} N : A$ then*

$$
[M\, N]_{\vdash \{N/x\}B} \equiv [M]_{\vdash \Pi\, x{:}A.\, B}\, [N]_{\vdash A} .
$$

Proof. Follows from the definition of the equivalence relation $\equiv$ and of the translations $[\![A]\!]$ and $[M]_{\Gamma \vdash A}$. Note that this proposition would not be true if $\equiv$ did not reflect equality. ◀

▶ **Lemma 4.3** (Substitution preservation). *If $\Gamma, x : A, \Gamma' \vdash_{\mathsf{m}} M : B$ and $\Gamma \vdash_{\sqsubset} N : A$ then*

$$
\{[N]_{\Gamma \vdash A}/x\}[M]_{\Gamma, x:A, \Gamma'} \equiv [\{N/x\}M]_{\Gamma, \{N/x\}\Gamma' \vdash \{N/x\}B} .
$$

If $\Gamma, x : A, \Gamma' \vdash_{\subseteq} M : B$ and $\Gamma \vdash_{\sqsubset} N : A$ then

$$
\{[N]_{\Gamma \vdash A}/x\}[M]_{\Gamma, x:A, \Gamma' \vdash B} \equiv [\{N/x\}M]_{\Gamma, \{N/x\}\Gamma' \vdash \{N/x\}B} .
$$

If $\Gamma, x : A, \Gamma' \vdash_{\sqsubset} M : s$ and $\Gamma \vdash_{\sqsubset} N : A$ then

$$
\{[N]_{\Gamma \vdash A}/x\}[\![B]\!]_{\Gamma, x:A, \Gamma'} \equiv [\![\{N/x\}B]\!]_{\Gamma, \{N/x\}\Gamma'} .
$$

Proof. The second and third statements derive from the first. We prove the first by induction on M, using Lemma 4.2.
- Case x. Then we must have $A \equiv B \equiv \{N/x\}B$. Therefore

$$
\begin{aligned}
\{[N]_{\vdash A}/x\}[x] &\equiv [N]_{\vdash A} \\
&\equiv [N]_{\vdash \{N/x\}B} \\
&\equiv [\{N/x\}x]_{\vdash \{N/x\}B} .
\end{aligned}
$$

- Case $y \neq x$. Then
$$\begin{aligned} \{[N]_{\vdash A}/x\}[y] &\equiv y \\ &\equiv [\{N/x\}y]_{\vdash\{N/x\}B} . \end{aligned}$$

- Case s. Then
$$\begin{aligned} \{[N]_{\vdash A}/x\}[s] &\equiv s \\ &\equiv [\{N/x\}s]_{\vdash\{N/x\}B} . \end{aligned}$$

- Case $\Pi\, y{:}C.\, D$. Then $B \equiv s_3$ where $\Gamma, x : A, \Gamma' \vdash_{\mathsf{m}} C : s_1$ and $\Gamma, x : A, \Gamma', y : C \vdash_{\mathsf{m}} D : s_2$ and $s_3 = \mathcal{R}(s_1, s_2)$. Therefore
$$\begin{aligned} \{[N]_{\vdash A}/x\}[\Pi\, y{:}C.\, D] &\equiv \pi_{s_1, s_2}\, x : \{[N]_{\vdash A}/x\}[C] . \{[N]_{\vdash A}/x\}[D] \\ &\equiv \pi_{s_1, s_2}\, x : [\{N/x\}C]_{\vdash s_1} . [\{N/x\}D]_{\vdash s_2} \\ &\equiv [\{N/x\}(\Pi\, y{:}C.\, D)]_{\vdash s_3} \end{aligned}$$

- Case $\lambda\, y{:}C.\, M'$. Then $B \equiv \Pi\, x{:}C.\, D$ where $\Gamma, x : A, \Gamma' \vdash_{\mathsf{m}} C : s_1$ and $\Gamma, x : A, \Gamma', y : C \vdash_{\mathsf{m}} M' : D$. Therefore
$$\begin{aligned} \{[N]_{\vdash A}/x\}[\lambda\, y{:}C.\, M'] &\equiv \lambda\, y{:}\{[N]_{\vdash A}/x\}[\![C]\!] . \{[N]_{\vdash A}/x\}[M'] \\ &\equiv \lambda\, y{:}[\![\{N/x\}C]\!] . [\{N/x\}M']_{\vdash\{N/x\}D} \\ &\equiv [\{N/x\}(\lambda\, y{:}C.\, M')]_{\vdash\{N/x\}(\Pi\, y C.\, D)} \end{aligned}$$

- Case $M'\, N'$. Then $B \equiv \{N'/y\}D$ where $\Gamma, x : A, \Gamma' \vdash_{\mathsf{m}} M' : \Pi\, y{:}C.\, D$ and $\Gamma, x : A, \Gamma' \vdash_{\mathsf{m}} N' : C$. Therefore
$$\begin{aligned} \{[N]_{\vdash A}/x\}[M\, N] &\equiv \{[N]_{\vdash A}/x\}[M'] \{[N]_{\vdash A}/x\}[N']_{\vdash C} \\ &\equiv [\{N/x\}M']_{\vdash\{N/x\}(\Pi\, y C.\, D)} [\{N/x\}N']_{\vdash\{N/x\}C} \\ &\equiv [\{N/x\}\, M'\, N']_{\vdash\{N/x\}\{N'/y\}D} \end{aligned}$$

◀

Equivalence preservation

Having proved substitution preservation, we prove that the translation preserves equivalence: if two well-typed terms are equivalent in $\mathrm{CC}^\infty_\subseteq$ then their translations are equivalent in $\mathrm{CC}^\infty_\uparrow$.

▶ **Lemma 4.4** (Equivalence preservation). *If $\Gamma \vdash_\subseteq M : B$ and $\Gamma \vdash_\subseteq N : B$ and $M \equiv N$ then $[M]_{\Gamma \vdash B} \equiv [N]_{\Gamma \vdash B}$. If $\Gamma \vdash_\subseteq A : s$ and $\Gamma \vdash_\subseteq B : s$ and $A \equiv B$ then $[\![A]\!] \equiv [\![B]\!]$.*

Proof. By induction on the derivation of $M \equiv N$. The second statement derives from the first. We show the base case $(\lambda\, x{:}C.\, M')\, N' \equiv \{N'/x\}M'$. Then $B \equiv \{N'/y\}D$ where $\Gamma \vdash_\subseteq \lambda\, x{:}C.\, M' : \Pi\, x{:}C.\, D$ and $\Gamma \vdash_\subseteq N' : C$. Therefore

$$\begin{aligned} [(\lambda\, x{:}C.\, M')\, N']_{\vdash\{N'/x\}D} &\equiv \big(\lambda\, x{:}[C]_{\vdash s_1} . [M']_{\vdash D}\big)\, [N']_{\vdash C} \\ &\quad \text{using Proposition 4.2} \\ &\equiv \{[N']_{\vdash C}/x\}[M']_{\vdash D} \\ &\quad \text{by } \beta\text{-equivalence} \\ &\equiv [\{N'/x\}M']_{\vdash\{N'/x\}D} \\ &\quad \text{using Lemma 4.3} \end{aligned}$$

◀

Typing preservation

With substitution preservation and equivalence preservation at hand, we can finally prove the main theorem, namely that the translation preserves typing.

▶ **Lemma 4.5.** *The translation satisfies the following properties:*
- *For all $s \in \mathcal{S}$, $\Gamma \vdash_{\uparrow} \mathsf{u}_s : [\![\mathcal{A}(s)]\!]$.*
- *If $\Gamma \vdash_{\uparrow} [A]_{\vdash s} : [\![s]\!]$ then*

$$\Gamma \vdash_{\uparrow} [A]_{\vdash \mathcal{N}(s)} : [\![\mathcal{N}(s)]\!] \,.$$

- *If $\Gamma \vdash_{\uparrow} [A]_{\vdash s_1} : [\![s_1]\!]$ and $\Gamma, x : [\![A]\!] \vdash_{\uparrow} [B]_{\vdash s_2} : [\![s_2]\!]$ then*

$$\Gamma \vdash_{\uparrow} [\Pi\, x{:}A.\, B]_{\vdash \mathcal{R}(s_1, s_2)} : [\![\mathcal{R}(s_1, s_2)]\!] \,.$$

- *If $\Gamma \vdash_{\uparrow} \mathsf{type}\,([\![A]\!])$ and $\Gamma, x : [\![A]\!] \vdash_{\uparrow} [M]_{\vdash B} : [\![B]\!]$ then*

$$\Gamma \vdash_{\uparrow} [\lambda\, x{:}A.\, M]_{\vdash \Pi\, xA.\, B} : [\![\Pi\, x{:}A.\, B]\!] \,.$$

- *If $\Gamma \vdash_{\uparrow} [M]_{\vdash \Pi\, xA.\, B} : \Pi\, x{:}[\![A]\!]\,.\,[\![B]\!]$ and $\Gamma \vdash_{\uparrow} [N]_{\vdash A} : [\![A]\!]$ then*

$$\Gamma \vdash_{\uparrow} [M\, N]_{\vdash \{N/x\}B} : [\![\{N/x\}B]\!] \,.$$

- *If $\Gamma \vdash_{\uparrow} [M]_{\vdash A} : [\![A]\!]$ and $\Gamma \vdash_{\uparrow} \mathsf{type}\,(B)$ and $[\![A]\!] \equiv [\![B]\!]$ then $[\![\Gamma]\!] \vdash_{\uparrow} [M]_{\vdash B} : [\![B]\!]$.*

Proof. Using Lemmas 4.2, 4.3, and 4.4. ◀

▶ **Theorem 4.6** (Typing preservation). *If $\Gamma \vdash_{\subset} M : A$ then $[\![\Gamma]\!] \vdash_{\uparrow} [M]_{\Gamma \vdash A} : [\![A]\!]_{\Gamma}$. If $\Gamma \vdash_{\subset} A : s$ then $[\![\Gamma]\!] \vdash_{\uparrow} \mathsf{type}\,([\![A]\!])$.*

Proof. By induction on the derivation of $\Gamma \vdash_{\subset} M : A$, using Lemma 4.5. The second statement derives from the first.
- Case *variable*. Then $(x : [\![A]\!]) \in [\![\Gamma]\!]$ so $[\![\Gamma]\!] \vdash_{\uparrow} x : [\![A]\!]$.
- Case *sort*. By Lemma 4.5, $[\![\Gamma]\!] \vdash_{\uparrow} \mathsf{u}_s : [\![\mathcal{A}(s)]\!]$.
- Case *cumulativity*. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\uparrow} [A]_{\vdash s} : [\![s]\!]$. By Lemma 4.5, $[\![\Gamma]\!] \vdash_{\uparrow} [A]_{\vdash \mathcal{N}(s)} : [\![\mathcal{N}(s)]\!]$.
- Case *product*. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\uparrow} [A]_{\vdash s_1} : [\![s_1]\!]$ and $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\uparrow} [B]_{\vdash s_2} : [\![s_2]\!]$. By Lemma 4.5,

$$[\![\Gamma]\!] \vdash_{\uparrow} [\Pi\, x{:}A.\, B]_{\vdash \mathcal{R}(s_1, s_2)} : [\![\mathcal{R}(s_1, s_2)]\!] \,.$$

- Case *abstraction*. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\uparrow} \mathsf{type}\,([\![A]\!])$ and $[\![\Gamma]\!], x : [\![A]\!] \vdash_{\uparrow} [M]_{\vdash B} : [\![B]\!]$. By Lemma 4.5,

$$\Gamma \vdash_{\uparrow} [\lambda\, x{:}A.\, M]_{\vdash \Pi\, xA.\, B} : [\![\Pi\, x{:}A.\, B]\!] \,.$$

- Case *application*. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\uparrow} [M]_{\vdash \Pi\, xA.\, B} : \Pi\, x : [\![A]\!]\,.\,[\![B]\!]$ and $[\![\Gamma]\!] \vdash_{\uparrow} [N]_{\vdash A} : [\![A]\!]$. By Lemma 4.5,

$$[\![\Gamma]\!] \vdash_{\uparrow} [M\, N]_{\vdash \{N/x\}B} : [\![\{N/x\}B]\!] \,.$$

- Case *conversion*. By induction hypothesis, $[\![\Gamma]\!] \vdash_{\uparrow} [M]_{\vdash A} : [\![A]\!]$ and $[\![\Gamma]\!] \vdash_{\uparrow} \mathsf{type}\,([\![B]\!])$. By Lemma 4.5, $[\![\Gamma]\!] \vdash_{\uparrow} [M]_{\vdash B} : [\![B]\!]$.

◀

▶ **Corollary 4.7.** *If $\mathsf{WF}_{\subset}\,(\Gamma)$ then $\mathsf{WF}_{\uparrow}\,([\![\Gamma]\!])$.*

Proof. By induction on Γ. ◀

5   Operational semantics

We presented $\mathrm{CC}_\uparrow^\infty$ assuming the equivalence relation $\equiv$ satisfies the equations of Definition 3.2. In practice, such equivalence relations are defined as the congruence closure of a set of reduction rules. In the case of $\mathrm{CC}_\subseteq^\infty$, it is the closure of β-reduction $\longrightarrow_\beta$, which enjoys confluence, subject reduction, and strong normalization. We now do the same for $\mathrm{CC}_\uparrow^\infty$.

Rewrite rules

The equations for the decoding function $\mathsf{T}_s(A)$ are easily oriented into rewrite rules:

$$\mathsf{T}_{\mathcal{A}(s)}(\mathsf{u}_s) \quad \longrightarrow \quad \mathsf{U}_s$$
$$\mathsf{T}_{\mathcal{N}(s)}(\uparrow_s (A)) \quad \longrightarrow \quad \mathsf{T}_s(A)$$
$$\mathsf{T}_{\mathcal{R}(s_1,s_2)}(\pi_{s_1,s_2} x : A.B) \quad \longrightarrow \quad \Pi\, x : \mathsf{T}_{s_1}(A).\, \mathsf{T}_{s_2}(B).$$

With these rules, we can view $\mathsf{T}_s(A)$ as a recursively defined function that decodes terms of type U_s into types by traversing their structure.

Orienting the equations for $\uparrow_s$ is more delicate. In Martin-Löf's intuitionistic type theory, a single equation is needed to reflect equality:

$$\uparrow_i (\pi_{i,i} x : A.B) \equiv \pi_{i+1,i+1} x : \uparrow_i (A).\uparrow_i (B).$$

In that case, it seems natural to orient the equation from left to right and see $\uparrow_i$ as a function that recursively transforms codes in U_i into equivalent codes in U_{i+1}:

$$\uparrow_i (\pi_{i,i} x : A.B) \longrightarrow \pi_{i+1,i+1} x : \uparrow_i (A).\uparrow_i (B).$$

While elegant, that solution does not behave well with the impredicative universe Prop. The equation

$$\pi_{i+1,\mathsf{Prop}} x : \uparrow_i (A).B \equiv \pi_{i,\mathsf{Prop}} x : A.B$$

requires the rewrite rule

$$\pi_{i+1,\mathsf{Prop}} x : \uparrow_i (A).B \longrightarrow \pi_{i,\mathsf{Prop}} x : A.B$$

which would break confluence with the previous rule because of the critical pair

$$\pi_{i+1,\mathsf{Prop}} x : \uparrow_i (\pi_{i,i} y : A.B).C.$$

Fortunately, we can still orient the equations in the other direction and obtain a well-behaved system. Again, we can express this concisely using the $\uparrow_{s_1}^{s_2}(A)$ notation.

▶ **Definition 5.1.** The equivalence relation $\equiv$ in $\mathrm{CC}_\uparrow^\infty$ is defined as the congruence induced by the following set of rewrite rules:

$$(\lambda\, x : A.\, M)\, N \quad \longrightarrow_\beta \quad \{N/x\}M$$
$$\mathsf{T}_{\mathcal{A}(s)}(\mathsf{u}_s) \quad \longrightarrow_\tau \quad \mathsf{U}_s$$
$$\mathsf{T}_{\mathcal{N}(s)}(\uparrow_s (A)) \quad \longrightarrow_\tau \quad \mathsf{T}_s(A)$$
$$\mathsf{T}_{\mathcal{R}(s_1,s_2)}(\pi_{s_1,s_2} x : A.B) \quad \longrightarrow_\tau \quad \Pi\, x : \mathsf{T}_{s_1}(A).\, \mathsf{T}_{s_2}(B)$$
$$\pi_{\mathcal{N}(s_1),s_2} x : \uparrow_{s_1}(A).B \quad \longrightarrow_\sigma \quad \uparrow_{\mathcal{R}(s_1,s_2)}^{\mathcal{R}(\mathcal{N}(s_1),s_2)}(\pi_{s_1,s_2} x : A.B)$$
$$\pi_{s_1,\mathcal{N}(s_2)} x : A.\uparrow_{s_2}(B) \quad \longrightarrow_\sigma \quad \uparrow_{\mathcal{R}(s_1,s_2)}^{\mathcal{R}(s_1,\mathcal{N}(s_2))}(\pi_{s_1,s_2} x : A.B).$$

In this formulation, the coercions $\uparrow_s$ propagate upwards towards the root of the term. This behavior matches the idea that, when computing minimal types, the cumulativity rule should be delayed as much as possible.

Properties

We show that the rewrite system $\longrightarrow_{\beta\tau\sigma}$ enjoys the usual properties of confluence, subject-reduction, and strong normalization. The last one follows from the strong normalization of $CC_\subseteq^\infty$.

▶ **Theorem 5.2** (Normalization of $\longrightarrow_{\tau\sigma}$). *The rewrite system $\longrightarrow_{\tau\sigma}$ is terminating.*

Proof. The relation $\longrightarrow_\tau$ strictly decreases the total height of T_s symbols and the relation $\longrightarrow_\sigma$ strictly decreases the total depth of $\uparrow_s$ symbols (while leaving the height of T_s unchanged), therefore $\longrightarrow_{\tau\sigma}$ is terminating. ◀

▶ **Theorem 5.3** (Confluence). *The rewrite system $\longrightarrow_{\beta\tau\sigma}$ is locally confluent.*

Proof. The rewrite rules of $\longrightarrow_{\tau\sigma}$ are left-linear and the critical pairs are convergent, therefore $\longrightarrow_{\tau\sigma}$ is locally confluent. By Proposition 5.2, it is terminating and hence confluent. Therefore its union with $\longrightarrow_\beta$ is confluent [23]. ◀

▶ **Theorem 5.4** (Subject reduction). *If $\Gamma \vdash_\uparrow M : A$ and $M \longrightarrow_{\beta\tau\sigma} M'$ then $\Gamma \vdash_\uparrow M' : A$.*

Proof. By induction on M. ◀

▶ **Theorem 5.5** (Strong normalization). *The rewrite system $\longrightarrow_{\beta\tau\sigma}$ is strongly normalizing for well-typed terms.*

Proof. By Theorem 5.2, $\longrightarrow_{\tau\sigma}$ is terminating, so any infinite sequence of reductions must have an infinite number of $\longrightarrow_\beta$ steps. If $M \longrightarrow_{\tau\sigma} M'$ then $|M| = |M'|$. If $M \longrightarrow_\beta M'$ then $|M| \longrightarrow_\beta |M'|$. An infinite reduction sequence in $CC_\uparrow^\infty$ would therefore lead to an infinite reduction sequence in $CC_\subseteq^\infty$. Moreover, according to Theorem 3.7 and Proposition 5.4, the sequence would be well-typed. Since $CC_\subseteq^\infty$ is strongly normalizing [11], this is impossible. ◀

6   Conclusion

We presented a formulation of the cumulative calculus of constructions with explicit subtyping. We used the Tarski style of universes to solve the issues related to dependent types and coercions. We showed that, by reflecting equality, we were able to preserve the expressiveness of Russell-style universes.

A thorough and definitive study of the two styles remains to be done. Are the two styles always equivalent? Can we always define an equivalence relation that reflects equality? Can it always be oriented into well-behaved rewrite rules? Finally, how does this solution interact with product covariance or other extensions of the theory, such as inductive types or universe polymorphism? Our guess is that inductive types should not pose a problem. Product covariance could be handled either by pre-expanding the terms to η-long form or by using a more general form of coercions $\uparrow_A^B$ where $A \leq B$. The interaction with universe polymorphism is still unclear.

Our results connect work done in pure type systems to work done in Martin-Löf's intuitionistic type theory. While the two theories have a clearly related core (namely the λ-calculus with dependent types), it is less obvious if they can still be unified or if they have definitively diverged. Pure type systems allow for a wide variety of specifications while intuitionistic type theory has a clear and intuitive interpretation for cumulativity. We feel that this problem deserves to be studied as the two theories form the basis for many logical

frameworks and proof assistants. The work of Herbelin and Siles [19], and van Doorn et al [22] already showed some progress in this direction.

A requirement for the aforementioned program is the development of a notion of cumulativity in pure type systems. We can imagine extending PTS specifications with a cumulativity relation as done by Barras, Grégoire, and Lasson for example [2, 3, 10]. However, it is unclear if such an extension is meaningful on its own, or if it only makes sense in $CC_\subseteq^\infty$ (which is both a functional *and* complete PTS). In particular, the equations of $CC_\uparrow^\infty$ rely on the fact that lifting inside a product cannot decrease the type of the product: $\mathcal{R}(s_1, s_2) \subseteq \mathcal{R}(\mathcal{N}(s_1), s_2)$ and $\mathcal{R}(s_1, s_2) \subseteq \mathcal{R}(s_1, \mathcal{N}(s_2))$. Whether this condition is essential or whether it can be avoided is unclear. The possibility of using universes à la Tarski remains to be studied.

Finally, while our system allowed us to get rid of the implicit subsumption rule, it did so at the expense of some complexity in the conversion rule. Whether this trade-off is beneficial in practical applications remains to be discussed. How does the Tarski style simplify the theoretical studies of the calculus of constructions? Can the current implementation of the calculus of constructions like Coq or Matita benefit from it? Nevertheless, this presentation is better suited for logical frameworks such as Dedukti, which usually do not support subtyping as a built-in. Our work opens the way for exporting Coq proofs to such frameworks.

References

1 Henk Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and Thomas S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.

2 Bruno Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, Université Paris 7, 1999.

3 Bruno Barras and Benjamin Grégoire. On the role of type decorations in the calculus of inductive constructions. In Luke Ong, editor, *Computer Science Logic*, number 3634 in Lecture Notes in Computer Science, pages 151–166. Springer Berlin Heidelberg, 2005.

4 Mathieu Boespflug and Guillaume Burel. CoqInE: Translating the calculus of inductive constructions into the λΠ-calculus modulo. In *Proof Exchange for Theorem Proving–Second International Workshop, PxTP*, page 44, 2012.

5 Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications*, number 4583 in Lecture Notes in Computer Science, pages 102–117. Springer Berlin Heidelberg, 2007.

6 Herman Geuvers and Freek Wiedijk. A logical framework with explicit conversions. *Electronic Notes in Theoretical Computer Science*, 199:33–47, February 2008.

7 Robert Harper and Robert Pollack. Type checking, universe polymorphism, and typical ambiguity in the calculus of constructions draft. In J. Díaz and F. Orejas, editors, *TAPSOFT'89*, number 352 in Lecture Notes in Computer Science, pages 241–256. Springer Berlin Heidelberg, 1989.

8 Robert Harper and Robert Pollack. Type checking with universes. *Theoretical Computer Science*, 89(1):107–136, October 1991.

9 Hugo Herbelin and Arnaud Spiwack. The Rooster and the Syntactic Bracket. In Ralph Matthes and Aleksy Schubert, editors, *19th International Conference on Types for Proofs*

    and Programs (TYPES 2013), volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 169–187, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

10   Marc Lasson. *Réalisabilité et paramétricité dans les systèmes de types purs.* PhD thesis, Ecole normale supérieure de Lyon, 2012.

11   Zhaohui Luo. CC_C^∞ and its meta theory. *Laboratory for Foundations of Computer Science Report ECS-LFCS-88-58*, 1988.

12   Zhaohui Luo. ECC, an extended calculus of constructions. In *Fourth Annual Symposium on Logic in Computer Science, 1989. LICS'89, Proceedings*, pages 386–395, June 1989.

13   Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press, Inc., New York, NY, USA, 1994.

14   Zhaohui Luo. Notes on universes in type theory. Lecture notes for a talk at Institute for Advanced Study, Princeton (`http://www.cs.rhul.ac.uk/home/zhaohui/universes.pdf`), 2012.

15   Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*, volume 17. Bibliopolis Naples, 1984.

16   Erik Palmgren. On universes in type theory. In *Twenty-five years of constructive type theory*, pages 191–204. Oxford University Press, October 1998.

17   The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

18   Ronan Saillard. Dedukti: a universal proof checker. In *Foundation of Mathematics for Computer-Aided Formalization Workshop*, 2013.

19   Vincent Siles and Hugo Herbelin. Pure type system conversion is always typable. *Journal of Functional Programming*, 22(02):153–180, 2012.

20   The Coq Development Team. *The Coq Reference Manual, version 8.4*, August 2012. Available electronically at `http://coq.inria.fr/doc`.

21   L. S. van Benthem Jutting, J. McKinna, and R. Pollack. Checking algorithms for pure type systems. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs*, number 806 in Lecture Notes in Computer Science, pages 19–61. Springer Berlin Heidelberg, 1994.

22   Floris van Doorn, Herman Geuvers, and Freek Wiedijk. Explicit convertibility proofs in pure type systems. In *Proceedings of the Eighth ACM SIGPLAN International Workshop on Logical Frameworks & Meta-languages: Theory & Practice*, LFMTP'13, pages 25–36, New York, NY, USA, 2013. ACM.

23   Vincent van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit, Amsterdam, 1994.