Type Theory with Explicit Universe Polymorphism

Marc Bezem
University of Bergen, Norway

Thierry Coquand
University of Gothenburg, Sweden

Peter Dybjer
Chalmers University of Technology, Gothenburg, Sweden

Martín Escardó
University of Birmingham, UK

Abstract

The aim of this paper is to refine and extend proposals by Sozeau and Tabareau and by Voevodsky for universe polymorphism in type theory. In those systems judgments can depend on explicit constraints between universe levels. We here present a system where we also have products indexed by universe levels and by constraints. Our theory has judgments for internal universe levels, built up from level variables by a successor operation and a binary supremum operation, and also judgments for equality of universe levels.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases type theory, universes in type theory, universe polymorphism, level-indexed products, constraint-indexed products

Digital Object Identifier 10.4230/LIPIcs.TYPES.2022.13

Acknowledgements The authors are grateful to the anonymous referees for useful feedback, and to Matthieu Sozeau for an update on the current state of universe polymorphism in Coq. We acknowledge the support of the Centre for Advanced Study (CAS) at the Norwegian Academy of Science and Letters in Oslo, Norway, which funded and hosted the research project Homotopy Type Theory and Univalent Foundations during the academic year 2018/19.

1 Introduction

The system of simple type theory, as introduced by Church [9], is elegant and forms the basis of several proof assistants. However, it has some unnatural limitations: it is not possible in this system to talk about an arbitrary type or about an arbitrary structure. For example, it is not possible to form the collection of all groups as needed in category theory. In order to address these limitations, Martin-Löf [22, 21] introduced a system with a type V of all types. A function $A \to V$ in this system can then be seen as a family of types over a given type A. It is natural in such a system to refine the operations exponential and cartesian product in simple type theory to operations of dependent products and sums. After the discovery of Girard's paradox [16], Martin-Löf [23] introduced a distinction between *small* and *large* types, similar to the distinction introduced in category theory between large and small sets, and the type V became the (large) type of small types. The name "universe" for such a type was chosen in analogy with the notion of universe introduced by Grothendieck to represent category theory in set theory.

Later, Martin-Löf [24] introduced a countable sequence of universes

$$\mathsf{U}_0 : \mathsf{U}_1 : \mathsf{U}_2 : \cdots$$

We refer to the indices $0, 1, 2, \ldots$ as *universe levels*.

Before the advent of univalent foundations, most type theorists expected only the first few universe levels to be relevant in practical formalisations. One thought that it might be feasible for a user of type theory to explicitly assign universe levels to their types and then simply add updated versions of earlier definitions when they were needed at different levels. However, the number of copies of definitions does not only grow with the level, but also with the number of type arguments in the definition of a type former. (The latter growth can be exponential!)

To deal with this, Huet [20] introduced a specific form of universe polymorphism that allowed the use of $\mathsf{U} : \mathsf{U}$ on the condition that each occurrence of U can be disambiguated as U_i in a consistent way. This approach has been followed by Harper and Pollack [18] and in Coq [35]. These approaches to *implicit* universe polymorphism are, however, problematic with respect to modularity. As pointed out in [11, 28]: one can prove $A \to B$ in one file, and $B \to C$ in another file, while $A \to C$ is not valid.

Leaving universe levels implicit also causes practical problems, since universe level disambiguation can be a costly operation, slowing down type-checking significantly. Moreover, so-called universe inconsistencies can be hard to explain to the user.

In order to cope with these issues, Courant [11] introduced explicit universe levels, with a supremum operation (see also Herbelin [19]). Explicit universe levels are also present in Agda [32] and Lean [12, 7]. However, whereas Courant has universe level *judgments*, Agda has a *type* of universe levels, and hence supports the formation of level-indexed products.

With the advent of Voevodsky's univalent foundations, the need for universe polymorphism has only increased. One often wants to prove theorems uniformly for arbitrary universes. These theorems may depend on several universes and there may be constraints on the level of these universes. In response to this Voevodsky [39] and Sozeau and Tabareau [30] proposed type theories parameterized by (arbitrary but fixed) universe levels and constraints.

The *univalence axiom* states that for any two types X, Y the canonical map

$$\mathsf{idtoeq}_{X,Y} : (X = Y) \to (X \simeq Y)$$

is an equivalence. Formally, the univalence axiom is an axiom scheme which is added to Martin-Löf type theory. If we work in Martin-Löf type theory with a countable tower of universes, each type is a member of some universe U_n. Such a universe U_n is *univalent* provided for all $X, Y : \mathsf{U}_n$ the canonical map $\mathsf{idtoeq}_{X,Y}$ is an equivalence. Let UA_n be the type expressing the univalence of U_n, and let $\mathsf{ua}_n : \mathsf{UA}_n$ for $n = 0, 1, \ldots$ be a sequence of constants postulating the respective instances of the univalence axiom. We note that $X = Y : \mathsf{U}_{n+1}$ and $X \simeq Y : \mathsf{U}_n$ and hence $\mathsf{UA}_n : \mathsf{U}_{n+1}$. We can express the universe polymorphism of these judgments internally in all of the above-mentioned systems by quantifying over universe levels, irrespective of having universe level judgments or a type of universe levels.

To be explicit about universes can be important, as shown by Waterhouse [40, 8], who gives an example of a large presheaf with no associated sheaf. A second example is the fact that the embedding $\mathsf{Group}(\mathsf{U}_n) \to \mathsf{Group}(\mathsf{U}_{n+1})$ of the type of groups in a universe U_n into that of the next universe U_{n+1} is not an equivalence. That is, there are more groups in the next universe [5].

We remark that universes are even more important in a predicative framework than in an impredicative one, for uniform proofs and modularity. Consider for example the formalisation of real numbers as Dedekind cuts, or domain elements as filters of formal neighbourhoods. Both belong to U_1 since they are properties of elements in U_0. However, even in a system using an impredicative universe of propositions, such as the ones in [20, 12], there is a need for definitions parametric in universe levels.

Terminology

Following Cardelli [6], we distinguish between *implicit* and *explicit* polymorphism:

> Parametric polymorphism is explicit when parametrization is obtained by explicit type parameters in procedure headings, and corresponding explicit applications of type arguments when procedures are called ... Parametric polymorphism is called implicit when the above type parameters and type applications are not admitted, but types can contain type variables which are unknown, yet to be determined, types.

Motivation

Many substantial Agda developments make essential use of *explicit* universe polymorphism with successor and finite suprema. Examples include the Agda standard library [34], the cubical Agda library [36], 1Lab [33], the Agda-HoTT library [37], agda-unimath [27], TypeTopology [14], HoTT-UF-in-Agda [13] (Midlands Graduate School 2019 lecture notes).

The original motivation for this work was to formalise the type theory of Agda, including explicit universe polymorphism. In doing that, we found ourselves modifying Agda's treatment of universes as follows:

- We have universe level *judgments*, like Courant [11], instead of a *type* of universe levels, like Agda.
- We add the possibility of expressing *explicit* universe level constraints. This is not only more general but also arguably gives a more natural way of expressing types involving universes.
- We do not require a first universe level zero, so that every definition that involves universes is polymorphic.
- We include a Type judgment, which does not refer to universes, as in Martin-Löf [25].

Our resulting type theory is orthogonal to the presence or absence of cumulativity. In the body of the paper, we treat universes à la Tarski, but we also give an appendix with a version à la Russell.

We have checked that the lecture notes [13] on HoTT/UF, which include 9620 lines of Agda without comments, can be rewritten without universe level zero. We believe, based on what we learned from this experiment, that the above Agda developments could also be rewritten in this way. Experience with these Agda developments suggest that a *type* for levels in Agda could be replaced by level judgments in practice. The fact that levels form a type in Agda automatically allows for nested universal quantification over levels, which we instead add explicitly to our type theory.

Summary of main contributions

Like Courant, we present a type theory with universe levels and universe level equations as *judgments*. Moreover, we don't restrict the levels to be natural numbers. Instead we just assume that they form a sup-semilattice with an inflationary endomorphism. In this way all levels are built up from level variables by a successor operation and a binary supremum operation. Unlike most other systems, we do not have a level constant 0 for the first universe level. Thus all types involving universes depend on level variables; they are *universe polymorphic*.

Furthermore, we make the polymorphism fully explicit in the sense of Cardelli by adding *level-indexed products*. In this way we regain some of the expressivity Agda gets from having a *type* Level of universe levels. Finally, we present a type theory with constraints as *judgments* similar to the ones by Sozeau and Tabareau [30] and Voevodsky [39] but extended with *constraint-indexed products*.

Plan

In Section 2 we display rules for a basic version of dependent type theory with Π, Σ, N, and an identity type former Id.

In Section 3 we explain how to add an externally indexed sequence of universes $\mathsf{U}_n, \mathsf{T}_n$ ($n \in \mathbb{N}$) à la Tarski, without cumulativity rules. In Appendix A we present a system with cumulativity, and in Appendix B we present a system à la Russell.

In Section 4 we introduce a notion of universe level, and let judgments depend not only on a context of ordinary variables, but also on level variables $\alpha, \ldots, \beta$. This gives rise to a type theory with level polymorphism, which we call "ML-style" as long as we do not bind level variables. We then extend this theory with level-indexed products of types $[\alpha]A$ and corresponding abstractions $\langle\alpha\rangle A$ to give full level polymorphism.

In Section 5 we extend the type theory in Section 4 with constraints (lists of equations between level expressions). Constraints can now appear as assumptions in hypothetical judgments. Moreover, we add constraint-indexed products of types $[\psi]A$ and corresponding abstractions $\langle\psi\rangle A$. This goes beyond the systems of Sozeau and Tabareau [30] and Voevodsky [39]. In Section 6 we compare our type theory with Voevodsky's and Sozeau-Tabareau's and briefly discuss some other approaches. Finally, in Section 7 we outline future work.

2 Rules for a basic type theory

We begin by listing the rules for a basic type theory with Π, Σ, N, and Id. A point of departure is the system described by Abel et al. in [1], since a significant part of the metatheory of this system has been formalized in Agda. This system has Π-types, N and one universe. However, for better readability we use named variables instead of de Bruijn indices. We also add Σ and Id, and, in the next sections, a tower of universes.

The judgment $\Gamma \vdash$ expresses that Γ is a context. The judgment $\Gamma \vdash A$ expresses that A is a type in context Γ. The judgment $\Gamma \vdash a : A$ expresses that A is a type and a is a term of type A in context Γ. The rules are given in Figure 1.

$$\frac{}{() \vdash} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash}\ (x \text{ fresh}) \qquad \frac{\Gamma \vdash}{\Gamma \vdash x : A}\ (x\!:\!A \text{ in } \Gamma)$$

Figure 1 Rules for context formation and assumption.

We may also write $A\ \mathsf{type}\ (\Gamma)$ for $\Gamma \vdash A$, and may omit the global context Γ, or the part of the context that is the same for all hypotheses and for the conclusion of the rule. Hypotheses that could be obtained from other hypotheses through inversion lemmas are often left out, for example, the hypothesis $A\ \mathsf{type}$ in the first rule for Π and Σ in Figure 2.

$$\frac{B\ \mathsf{type}\ (x : A)}{\Pi_{x:A} B\ \mathsf{type}} \qquad \frac{b : B\ (x : A)}{\lambda_{x:A} b : \Pi_{x:A} B} \qquad \frac{c : \Pi_{x:A} B \qquad a : A}{c\, a : B(a/x)}$$

$$\frac{B\ \mathsf{type}\ (x : A)}{\Sigma_{x:A} B\ \mathsf{type}} \qquad \frac{a : A \qquad b : B(a/x)}{(a, b) : \Sigma_{x:A} B} \qquad \frac{c : \Sigma_{x:A} B}{c.1 : A} \qquad \frac{c : \Sigma_{x:A} B}{c.2 : B(c.1/x)}$$

Figure 2 Rules for Π and Σ.

We write $=$ for definitional equality (or conversion). The following rules express that conversion is an equivalence relation and that judgments are invariant under conversion. The rules are given in Figures 3 and 4.

$$\frac{a : A \qquad A \,=\, B}{a : B} \qquad \frac{a \,=\, a' : A \qquad A \,=\, B}{a \,=\, a' : B}$$

$$\frac{A \,=\, B \qquad A \,=\, C}{B \,=\, C} \qquad \frac{A \ \mathsf{type}}{A \,=\, A} \qquad \frac{a \,=\, b : A \qquad a \,=\, c : A}{b \,=\, c : A} \qquad \frac{a : A}{a \,=\, a : A}$$

Figure 3 General rules for conversion.

$$\frac{A \,=\, A' \qquad B \,=\, B' \ (x : A)}{\Pi_{x:A}B \,=\, \Pi_{x:A'}B'} \qquad \frac{c \,=\, c' : \Pi_{x:A}B \qquad a \,=\, a' : A}{c\ a \,=\, c'\ a' : B(a/x)}$$

$$\frac{b : B\ (x : A) \qquad a : A}{\lambda_{x:A}b\ a \,=\, b(a/x) : B(a/x)} \qquad \frac{f\ x = g\ x : B\ (x : A)}{f = g : \Pi_{x:A}B}$$

$$\frac{A \,=\, A' \qquad B \,=\, B' \ (x : A)}{\Sigma_{x:A}B \,=\, \Sigma_{x:A'}B'} \qquad \frac{c \,=\, c' : \Sigma_{x:A}B}{c.1 \,=\, c'.1 : A} \qquad \frac{c \,=\, c' : \Sigma_{x:A}B}{c.2 \,=\, c'.2 : B(c.1/x)}$$

$$\frac{a : A \qquad b : B(a/x)}{(a, b).1 \,=\, a : A} \qquad \frac{a : A \qquad b : B(a/x)}{(a, b).2 \,=\, b : B(a/x)} \qquad \frac{c.1 \,=\, c'.1 : A \qquad c.2 \,=\, c'.2 : B(c.1/x)}{c \,=\, c' : \Sigma_{x:A}B}$$

Figure 4 Conversion rules for Π and Σ.

By now we have introduced several parametrized syntactic constructs for types and terms, such as $\Pi_{x:A}B$, $\lambda_{x:A}b$, $c\,a$, $(a, b).2$. Conversion rules for Π and Σ were given in Figure 4. and those rules imply that $=$ is a congruence.(Some cases of congruence are subtle. Exercise: show congruence of $=$ for $\lambda_{x:A}b$ and (a, b).) In the sequel we will tacitly assume the inference rules ensuring that $=$ is a congruence for all syntactic constructs that are to follow.

We now introduce the type of natural numbers N with the usual constructors $\mathsf{0}, \mathsf{S}$ and eliminator R, as an example of an inductive data type. Rules with the same hypotheses are written as one rule with several conclusions. The rules are given in Figure 5.

We also add identity types $\mathsf{Id}(A, a, a')$ for all A type, $a : A$ and $a' : A$, with constructor $\mathsf{refl}(A, a)$ and (based) eliminator $\mathsf{J}(A, a, C, d, a', q)$. The rules are given in Figure 6.

In this basic type theory we can define, for example, $\mathsf{isContr}(A) := \Sigma_{a:A}\Pi_{x:A}\mathsf{Id}(A, a, x)$ for A type, expressing that A is contractible. If also B type, we can define $\mathsf{Equiv}(A, B) := \Sigma_{f:A\to B}\Pi_{b:B}\mathsf{isContr}(\Sigma_{x:A}\mathsf{Id}(B, b, f(x)))$, which is the type of equivalences from A to B. This example will also be used later on.

$$\frac{}{\mathsf{N}\ \mathsf{type} \qquad \mathsf{0} : \mathsf{N}} \qquad \frac{n : \mathsf{N}}{\mathsf{S}(n) : \mathsf{N}} \qquad \frac{P\ \mathsf{type}\ (x : \mathsf{N}) \qquad a : P(\mathsf{0}/x) \qquad g : \Pi_{x:\mathsf{N}}(P \to P(\mathsf{S}(x)/x))}{\mathsf{R}(P, a, g, \mathsf{0}) = a : P(\mathsf{0}/x)}$$

$$\frac{P\ \mathsf{type}\ (x : \mathsf{N}) \qquad a : P(\mathsf{0}/x) \qquad g : \Pi_{x:\mathsf{N}}(P \to P(\mathsf{S}(x)/x)) \qquad n : \mathsf{N}}{\mathsf{R}(P, a, g, n) : P(n/x) \qquad \mathsf{R}(P, a, g, \mathsf{S}(n)) = g\ n\ \mathsf{R}(P, a, g, n) : P(\mathsf{S}(n)/x)}$$

Figure 5 Rules and conversion rules for the datatype N.

$$\frac{A \ \mathsf{type} \qquad a : A \qquad a' : A}{\mathsf{Id}(A, a, a') \ \mathsf{type}} \qquad \frac{a : A}{\mathsf{refl}(A, a) : \mathsf{Id}(A, a, a)}$$

$$\frac{a : A \qquad C \ \mathsf{type}\ (x : A, p : \mathsf{Id}(A, a, x)) \qquad d : C(a/x, \mathsf{refl}(A, a)/p) \qquad a' : A \qquad q : \mathsf{Id}(A, a, a')}{\mathsf{J}(A, a, C, d, a', q) : C(a'/x, q/p) \qquad \mathsf{J}(A, a, C, d, a, \mathsf{refl}(A, a)) = d : C(a/x, \mathsf{refl}(A, a)/p)}$$

Figure 6 Rules and conversion rule for identity types.

3 Rules for an external sequence of universes

We present an external sequence of universes of codes of types, together with the decoding functions. (We do not include rules for cumulativity here, leaving them for Appendix A.) The rules are given in Figure 7.

$$\frac{}{\mathsf{U}_m \ \mathsf{type}} \qquad \frac{A : \mathsf{U}_m}{\mathsf{T}_m(A) \ \mathsf{type}} \qquad \frac{}{\mathsf{U}^n_m : \mathsf{U}_n \qquad \mathsf{T}_n(\mathsf{U}^n_m) = \mathsf{U}_m}\ (n > m)$$

Figure 7 Rules and conversion rules for all universes U_m and their codes U^n_m $(n > m)$.

Here and below m and n, as super- and subscripts of U and T, are *external* natural numbers, and $n \vee m$ is the maximum of n and m. This means, for example, that U_m type is a *schema*, yielding one rule for each m.

Next we define how Π, Σ, N, and Id are "relativized" to codes of types, and how they are decoded, in Figures 8 and 9.

$$\frac{A : \mathsf{U}_n \qquad B : \mathsf{T}_n(A) \to \mathsf{U}_m}{\Pi^{n,m} AB : \mathsf{U}_{n \vee m} \qquad \mathsf{T}_{n \vee m}\ (\Pi^{n,m} AB) = \Pi_{x:\mathsf{T}_n(A)} \mathsf{T}_m(B\ x)}$$

$$\frac{A : \mathsf{U}_n \qquad B : \mathsf{T}_n(A) \to \mathsf{U}_m}{\Sigma^{n,m} AB : \mathsf{U}_{n \vee m} \qquad \mathsf{T}_{n \vee m}\ (\Sigma^{n,m} AB) = \Sigma_{x:\mathsf{T}_n(A)} \mathsf{T}_m(B\ x)}$$

Figure 8 Rules and conversion rules for Π and Σ for codes of types.

$$\frac{}{\mathsf{N}^n : \mathsf{U}_n} \qquad \frac{}{\mathsf{T}_n(\mathsf{N}^n) = \mathsf{N}}$$

$$\frac{A : \mathsf{U}_n \qquad a_0 : \mathsf{T}_n(A) \qquad a_1 : \mathsf{T}_n(A)}{\mathsf{Id}^n(A, a_0, a_1) : \mathsf{U}_n \qquad \mathsf{T}_n(\mathsf{Id}^n(A, a_0, a_1)) = \mathsf{Id}(\mathsf{T}_n(A), a_0, a_1)}$$

Figure 9 Rules and conversion rules for codes of N and Id.

In the following section we present a type theory with *internal* universe level expressions. This theory has finitely many inference rules.

4 A type theory with universe levels and polymorphism

The problem with the type system with an external sequence of universes is that we have to *duplicate* definitions that follow the same pattern. For instance, we have the identity function

$$\mathsf{id}_n := \lambda_{X:\mathsf{U}_n}\lambda_{x:\mathsf{T}_n(X)}x : \Pi_{X:\mathsf{U}_n}\mathsf{T}_n(X) \to \mathsf{T}_n(X)$$

This is a schema that may have to be defined (and type-checked) for several n. We address this issue by introducing *universe level* expressions: we write $\alpha, \beta, \ldots$ for *level variables*, and $l, m, \ldots$ for *level expressions* which are built from level variables by suprema $l \vee m$ and the next level operation l^+. Level expressions form a sup-semilattice $l \vee m$ with a next level operation l^+ such that $l \vee l^+ = l^+$ and $(l \vee m)^+ = l^+ \vee m^+$. (We don't need a 0 element.) We write $l \leqslant m$ for $l \vee m = m$ and $l < m$ for $l^+ \leqslant m$. See [4] for more details.

We have a new context extension operation that adds a fresh level variable α to a context, a rule for assumption, and typing rules for level expressions, in Figure 10.

$$\frac{\Gamma \vdash}{\Gamma, \alpha\ \mathsf{level} \vdash}\ (\alpha \text{ fresh}) \qquad \frac{\Gamma \vdash}{\Gamma \vdash \alpha\ \mathsf{level}}(\alpha \text{ in } \Gamma) \qquad \frac{l\ \mathsf{level} \qquad m\ \mathsf{level}}{l \vee m\ \mathsf{level}} \qquad \frac{l\ \mathsf{level}}{l^+\ \mathsf{level}}$$

Figure 10 Rules for typing level expressions, extending Figure 1.

We also have level equality judgments $\Gamma \vdash l = m$ and want to enforce that judgments are invariant under level equality. To this end we add the rule that $\Gamma \vdash l = m$ when $\Gamma \vdash l$ level and $\Gamma \vdash m$ level and $l = m$ in the free sup-semilattice above with $\_^+$ and generators (level variables) in Γ.

In the next section we will also consider *hypothetical* level equality judgments, i.e., we may have constraints in Γ, quotienting the free sup-semilattice above.

We tacitly assume additional rules ensuring that level equality implies definitional equality of types and terms. It then follows from the rules of our basic type theory that judgments are invariant under level equality: if $l = m$ and $a(l/\alpha) : A(l/\alpha)$, then $a(m/\alpha) : A(m/\alpha)$.

We will now add rules for internally indexed universes in Figure 11. Note that $l < m$ is shorthand for the level equality judgment $m = l^+ \vee m$.

$$\frac{l\ \mathsf{level}}{\mathsf{U}_l\ \mathsf{type}} \qquad \frac{A : \mathsf{U}_l}{\mathsf{T}_l(A)\ \mathsf{type}} \qquad \frac{l < m}{\mathsf{U}_l^m : \mathsf{U}_m \qquad \mathsf{T}_m(\mathsf{U}_l^m) = \mathsf{U}_l}$$

Figure 11 Rules and conversion rule for universes U_l and their codes.

The remaining rules are completely analogous to the rules in Figure 8 and Figure 9 for externally indexed universes with external numbers replaced by internal levels. (To rules without assumptions, such as the first two in Fig. 9, we need to add assumptions like n level, for other rules these assumptions can be obtained from inversion lemmas.)

We expect that normalisation holds for this system. This would imply decidable type-checking. This would also imply that if $a : \mathsf{N}$ in a context with only level variables, then a is convertible to a numeral.

Interpreting the level-indexed system in the system with externally indexed universes

A judgment in the level-indexed system can be interpreted in the externally indexed system relative to an assignment ρ of external natural numbers to level variables. We simply replace each level expression in the judgment by the corresponding natural number obtained by letting $l^+ \rho = l\,\rho + 1$ and $(l \vee m)\,\rho = \max(l\,\rho, m\,\rho)$.

Rules for level-indexed products

In Agda Level is a type, and it is thus possible to form level-indexed products of types as Π-types. In our system this is not possible, since level is not a type. Nevertheless, it is useful for modularity to be able to form level-indexed products. Thus we extend the system with the rules in Figure 12.

$$\frac{A \ \mathsf{type}\ (\alpha\ \mathsf{level})}{[\alpha]A\ \mathsf{type}} \qquad \frac{t : [\alpha]A \qquad l\ \mathsf{level}}{t\ l : A(l/\alpha)} \qquad \frac{u : A\ (\alpha\ \mathsf{level})}{\langle\alpha\rangle u : [\alpha]A} \qquad \frac{t\ \alpha = u\ \alpha : A\ (\alpha\ \mathsf{level})}{t = u : [\alpha]A}$$

$$\frac{u : A\ (\alpha\ \mathsf{level}) \qquad l\ \mathsf{level}}{(\langle\alpha\rangle u)\ l = u(l/\alpha) : A(l/\alpha)}$$

Figure 12 Rules and conversion rule for level-indexed products.

In this type theory we can reflect, for example, $\mathsf{isContr}(A) := \Sigma_{a:A}\Pi_{x:A}\mathsf{Id}(A, a, x)$ for A type as follows. In the context $\alpha\,\mathsf{level}, A : \mathsf{U}_\alpha$, define

$$\mathsf{isContr}^\alpha(A) := \Sigma^{\alpha,\alpha}\, A\, (\lambda_{a:\mathsf{T}_\alpha(A)}(\Pi^{\alpha,\alpha}\, A\, (\lambda_{x:\mathsf{T}_\alpha(A)}\mathsf{Id}^\alpha(A, a, x)))).$$

Then $\mathsf{T}_\alpha(\mathsf{isContr}^\alpha(A)) = \mathsf{isContr}(\mathsf{T}_\alpha(A))$. We can further abstract to obtain the following typing:

$$\langle\alpha\rangle\lambda_{A:\mathsf{U}_\alpha}\mathsf{isContr}^\alpha(A) : [\alpha](\mathsf{U}_\alpha \to \mathsf{U}_\alpha).$$

In a similar way we can reflect $\mathsf{Equiv}(A, B)$ for $A, B\,\mathsf{type}$ by defining in context $\alpha\,\mathsf{level}, \beta\,\mathsf{level},\ A : \mathsf{U}_\alpha, B : \mathsf{U}_\beta$ a term $\mathsf{Eq}^{\alpha,\beta}(A, B) : \mathsf{U}_{\alpha\vee\beta}$ such that $\mathsf{T}_{\alpha\vee\beta}(\mathsf{Eq}^{\alpha,\beta}(A, B)) = \mathsf{Equiv}(\mathsf{T}_\alpha(A), \mathsf{T}_\beta(B))$.

An example that uses level-indexed products beyond the ML-style polymorphism (provided by Sozeau and Tabareau and by Voevodsky) is the following type which expresses the theorem that univalence for universes of arbitrary level implies function extensionality for functions between universes of arbitrary levels.

$$([\alpha]\mathsf{IsUnivalent}\,\mathsf{U}_\alpha) \to [\beta][\gamma]\mathsf{FunExt}\,\mathsf{U}_\beta\,\mathsf{U}_\gamma$$

In other words, *global* univalence implies *global* function extensionality.

Since an assumption of global function extensionality can replace many assumptions of local function extensionality (provided by ML-style polymorphism), this can also give rise to shorter code, see the example `Eq-Eq-cong'` in [13].

5 A type theory with level constraints

To motivate why it may be useful to introduce the notion of judgment relative to a list of constraints on universe levels, consider the following type in a system without cumulativity. (We use Russell style notation for readability, see Appendix B for the rules for the Russell style version of our system.)

$$\Pi_{A:\mathsf{U}_l\ B:\mathsf{U}_m\ C:\mathsf{U}_n}\ \ \mathsf{Id}\,\mathsf{U}_{l\vee m}\,(A\times B)\,(C\times A)\to \mathsf{Id}\,\mathsf{U}_{m\vee l}\,(B\times A)\,(C\times A)$$

This is well-formed provided $l\vee m=n\vee l$. There are several independent solutions:

$$l=\alpha, m=\beta, n=\alpha\vee\beta$$
$$l=\alpha, m=\gamma\vee\alpha, n=\gamma$$
$$l=\beta\vee\gamma, m=\beta, n=\gamma$$
$$l=\alpha, m=\beta, n=\beta$$

where α,β, and γ are level variables. It should be clear that there cannot be any *most general* solution, since this solution would have to assign variables to l,m,n.

In a system with level constraints, we could instead derive the (inhabited under UA) type

$$\Pi_{A:\mathsf{U}_\alpha\ B:\mathsf{U}_\beta\ C:\mathsf{U}_\gamma}\ \ \mathsf{Id}\,\mathsf{U}_{\alpha\vee\beta}\,(A\times B)\,(C\times A)\to \mathsf{Id}\,\mathsf{U}_{\alpha\vee\gamma}\,(B\times A)\,(C\times A)$$

which is valid under the constraint $\alpha\vee\beta=\alpha\vee\gamma$, which captures all solutions simultaneously.

Without being able to declare explicitely such constraints, one would instead need to write four separate definitions.

Surprisingly, if we add a least level 0 to the term levels (like in Agda) then there *is* a most general solution, namely $l=\alpha\vee\beta\vee\delta,\ m=\beta\vee\gamma,\ n=\alpha\vee\gamma$,[1] since it can be seen as an instance of a Associative Commutative Unit Idempotent unification problem [3].

It is however possible to find equation systems which do not have a most general unifier, even with a least level 0, using the next level operation. For instance, the system $l^+=m\vee n$ does not have a most general unifier, using a reasoning similar to the one in [15].

Rules for level constraints

A constraint is an equation $l=m$, where l and m are level expressions. Voevodsky [39] suggested to introduce universe levels with constraints. This corresponds to mathematical practice: for instance, at the beginning of the book [17], the author introduces two universes U and V with the constraint that U is a member of V. In our setting this will correspond to introducing two levels α and β with the constraint $\alpha<\beta$.

Note that $\alpha<\beta$ holds iff $\beta=\beta\vee\alpha^+$. We can thus avoid declaring this constraint if we instead systematically replace β by $\beta\vee\alpha^+$. This is what is currently done in the system Agda. However, this is a rather indirect way to express what is going on. Furthermore, the example at the beginning of this section shows that this can lead to an artificial duplication of definitions.

Recall that we have in Section 4, e.g., the rule that $\mathsf{U}_l^m:\mathsf{U}_m$ if $l<m$ valid, that is, if $l<m$ holds in the free semilattice. In the extended system in this section, this typing rule also applies when $l<m$ is implied by the constraints in the context Γ. For instance, we have $\alpha^+\leqslant\beta$ in a context with constraints $\alpha\leqslant\gamma$ and $\gamma^+\leqslant\beta$.

To this end we introduce a new context extension operation Γ,ψ extending a context Γ by a finite set of constraints ψ. The first condition for forming Γ,ψ is that all level variables occurring in ψ are declared in Γ. The second condition is that the finite set of constaints in the extended context Γ,ψ is loop-free. A finite set of constraints is *loop-free* if it does not create a *loop*, i.e., a level expression l such that $l<l$ modulo this set of constraints, see [4].

[1] We learnt this from Thiago Felicissimo, with a reference to the work [15].

We also have a new judgment form $\Gamma \vdash \psi$ valid that expresses that the constraints in ψ hold in Γ, that is, are implied by the constraints in Γ. If there are no constraints in Γ, the judgment $\Gamma \vdash \{l = m\}$ valid amounts to the same as $\Gamma \vdash l = m$ in Section 4. Otherwise it means that the constraints in ψ hold in the sup-semilattice with $\_^+$ presented by Γ.

As shown in [4], $\Gamma \vdash \psi$ valid as well as loop-checking, is decidable in polynomial time.

Voevodsky [39] did not describe a mechanism to *eliminate* universe levels and constraints. In Figure 12 we gave rules for eliminating universe levels and in Figure 13 below we give rules for eliminating universe level constraints.

Rules for constraint-indexed products

We introduce a "restriction" or "constraining" operation with the rules in Figure 13.

$$\frac{A \ \mathsf{type} \quad (\psi)}{[\psi]A \ \mathsf{type}} \qquad \frac{t : A \quad (\psi)}{\langle\psi\rangle t : [\psi]A} \qquad \frac{\psi \ \mathsf{valid}}{[\psi]A = A} \qquad \frac{\psi \ \mathsf{valid}}{\langle\psi\rangle t = t}$$

Figure 13 Rules for constraining.

Here is a simple example of the use of this system. In order to represent set theory in type theory, we can use a type V satisfying the following equality $\mathsf{Id}\ \mathsf{U}_\beta\ V\ (\Sigma_{X:\mathsf{U}_\alpha} X \to V)$. This equation is only well-typed modulo the constraint $\alpha < \beta$.

We can define in our system a constant

$$c \;=\; \langle\alpha\ \beta\rangle\langle\alpha < \beta\rangle\lambda_{Y:\mathsf{U}_\beta}\mathsf{Id}\ \mathsf{U}_\beta\ Y\ (\Sigma_{X:\mathsf{U}_\alpha} X \to Y) \quad : \quad [\alpha\ \beta][\alpha < \beta](\mathsf{U}_\beta \to \mathsf{U}_{\beta^+})$$

This is because $\Sigma_{X:\mathsf{U}_\alpha} X \to Y$ has type U_β in the context

$$\alpha : \mathsf{level},\ \ \beta : \mathsf{level},\ \ \alpha < \beta,\ \ Y : \mathsf{U}_\beta$$

We can further instantiate this constant c on two levels l and m, and this will be of type

$$[l < m](\mathsf{U}_m \to \mathsf{U}_{m^+})$$

and this can only be used further if $l < m$ holds in the current context[2].

In the current system of Agda, the constraint $\alpha < \beta$ is represented indirectly by writing β on the form $\gamma \vee \alpha^+$ and c is defined as

$$c \;=\; \langle\alpha\ \gamma\rangle\lambda_{Y:\mathsf{U}_{\alpha^+\vee\gamma}}\mathsf{Id}\ \mathsf{U}_{\alpha^+\vee\gamma}\ Y\ (\Sigma_{X:\mathsf{U}_\alpha} X \to Y) \quad : \quad [\alpha\ \gamma](\mathsf{U}_{\alpha^+\vee\gamma} \to \mathsf{U}_{\alpha^{++}\vee\gamma^+})$$

which arguably is less readable.

[2] It is interesting to replace $\mathsf{Id}\ \mathsf{U}_\beta$ in the definition of c above by Eq. We leave it to the reader to verify the following typing, for which no constraint is needed:

$$c' \;=\; \langle\alpha\ \beta\rangle\lambda_{Y:\mathsf{U}_\beta}\ \mathsf{Eq}\ Y\ (\Sigma_{X:\mathsf{U}_\alpha} X \to Y) \quad : \quad [\alpha\ \beta](\mathsf{U}_\beta \to \mathsf{U}_{\beta\vee\alpha^+})$$

In general, if we build a term t of type A in a context using labels $\alpha_1, \dots, \alpha_m$ and constraint ψ and variables $x_1 : A_1, \dots, x_n : A_n$ we can introduce a constant

$$c \;=\; \langle \alpha_1 \;\dots\; \alpha_m \rangle \langle \psi \rangle \lambda_{x_1 \;\dots\; x_n} t \;:\; [\alpha_1 \;\dots\; \alpha_m][\psi]\Pi_{x_1:A_1 \;\dots\; x_n:A_n} A$$

We can then instantiate this constant $c\; l_1 \;\dots\; l_m\; u_1 \;\dots\; u_n$, but only if the levels $l_1 \;\dots\; l_m$ satisfy the constraint ψ.

We remark that Voevodsky's system [39] has no constraint-indexed products and no associated application operation, and instantiation of levels is only a meta-level operation. Sozeau and Tabareau [30] do not have constraint-index products either. However, they do have a special operation for instantiating universe-polymorphic constants defined in the global environment.

▶ Remark 1. Let's discuss some special cases and variations.

First, it is possible not to use level variables at all, making the semilattice empty, in which case the type theory defaults to one without universes as presented in Section 2.

Second, one could have exactly one level variable in the context. Then any constraint would either be a loop or trivial. In the latter case, the finitely presented semilattice is isomorphic to the natural numbers with successor and max. Still, we get some more expressivity than the type theory in Section 3 since we can express universe polymorphism in one variable.

Third, with arbitrarily many level variables but not using constraints we get the type theory in Section 4.

Fourth, we could add a bottom element, or empty supremum, to the semilattice. Without level variables and constraints, the finitely presented semilattice is isomorphic to the natural numbers with successor and max and we would get the type theory in Section 3. We would also get a first universe. (Alternatively, one could have one designated level variable 0 and constraints $0 \leqslant \alpha$ for all level variables α.)

Fifth, we note in passing that the one-point semilattice with $\_^+$ has a loop.

6 Related work

We have already discussed both Coq's and Agda's treatment of universe polymorphism in the introduction, including the work of Huet, Harper and Pollack, Courant, Herbelin, and Sozeau and Tabareau, as well as of Voevodsky. In this section we further discuss the latter two, as well as some recent related work.

Lean

One can roughly describe the type system of Lean [12, 7] as our current type system where we only can declare constants of the form $c \;=\; \langle \alpha_1 \;\dots\; \alpha_n \rangle M \;\;:\;\; [\alpha_1 \;\dots\; \alpha_n] A$ where there are no new level variables introduced in M and A.

Voevodsky

One of our starting points was the 79 pp. draft [39] by Voevodsky, where type theories are parametrized by a fixed but arbitrary finite set of constraints over a given finite set Fu of *u-level variables*. A *u-level expression* [39, Def. 2.0.2] is either a numeral, or a variable in Fu, or an expression of the form $M + n$ with n a numeral and M a u-level expression, or of the form $\max(M_1, M_2)$ with M_1, M_2 u-level expression. A *constraint* is an equation between two u-level expressions. Given the finite set of constraints, $\mathcal{A}$ is the set of assigments of natural numbers to variables in Fu that satisfy all constraints.

The rules 7 and 10 in [39, Section 3.4] define how to use constraints: two types (and, similarly, two terms) become definitionally equal if, for all assignments in $\mathcal{A}$, the two types become *essentially* syntactically equal after substitution of all variables in *Fu* by their assigned natural number. For example, the constraint $\alpha < \beta$ makes U_β and $\mathsf{U}_{\max(1,\beta)}$ definitionally equal.

For decidability, Voevodsky refers in the proof of [39, Lemma 2.0.4, proof] to Presburger Arithmetic, in which his constraints can easily be expressed.[3]This indeed implies that definitional equality is decidable, even "in practice [...] expected to be very easily decidable i.e. to have low complexity of the decision procedure" [39, p. 5, l. -13]. The latter is confirmed by [4].

The remaining sections of [39] are devoted to extending the type theory with data types, W-types and identity types, and to its metatheory.

We summarize the main differences between our type theories and Voevodsky's as follows. In [39], u-levels are natural numbers, even though u-level expressions can also contain u-level variables, successor and maximum. Our levels are elements of an abstract sup-semilattice with a successor operation. In the abstract setting, for example, $\alpha \vee \beta = \alpha^+$ does not imply $\beta = \alpha^+$, whereas in [39] it does. In [39], constraints are introduced, once and for all, at the level of the theory. In our proposal they are introduced at the level of contexts. There are no level-indexed products and no constraint-indexed products in [39]. We also remark that Voevodsky's system is Tarski-style and has cumulativity (rules 29 and 30 in [39, Section 3.4]). Our system is also Tarski-style, but we present a Russell-style version in Appendix B. We present rules for cumulativity in Appendix A.

Sozeau and Tabareau

In Sozeau and Tabareau's [30] work on universe polymorphism in the Coq tradition, there are special rules for introducing universe-monomorphic and universe-polymorphic constants, as well as a rule for instantiating the latter. However, their system does not include the full explicit universe polymorphism provided by level- and constraint-indexed products. In our system, with explicit universe polymorphism, we can have a uniform treatment of definitions, all of the form

$$c : A = t$$

where A is a type and t a term of type A, and these definitions can be local as well.

The constraint languages differ: their constraints are equalities or (strict) inequalities between level variables, while ours are equalities between level expressions generated by the supremum and successor operations.

Furthermore, they consider cumulative universe hierarchies à la Russell, while our universes are à la Tarski and we consider both non-cumulative (like Agda) and cumulative versions.

One further important difference is that their system has been completely implemented and tested on significant examples, while our system is at this stage only a proposal. The idea would be that the users have to declare explicitly both universe levels *and* constraints. The Agda implementation shows that it works in practice to be explicit about universe levels, and we expect that to be explicit about constraints will actually simplify the use of the system, but this has yet to be tested in practice. Recently, Coq has been extended to support universes and constraint annotations from entirely implicit to explicit. Moreover, our level- and constraint-indexed products can to some extent be simulated by using Coq's module system [29].

[3] For this it seems necessary to also require that $\mathcal{A}$ is defined by a *finite* set of constraints.

Assaf and Thiré

Assaf [2] considers an alternative version of the calculus of constructions where subtyping is explicit. This new system avoids problems related to coercions and dependent types by using the Tarski style of universes and by introducing additional equations to reflect equality. In particular he adds an explicit cumulativity map $\mathsf{T}_1^0 : \mathsf{U}_0 \to \mathsf{U}_1$. He argues that "full reflection" is necessary to achieve the expressivity of Russell style. He introduces the explicit cumulative calculus of constructions (CC↑) which is closely related to our system of externally indexed Tarski style universes. This is analysed further in the PhD thesis of F. Thiré [38].

7 Conjectures and future work

Canonicity and normalization have been proved for a type theory with an external tower of universes [10]. We conjecture that these proofs can be modified to yield proofs of analogous properties (and their corollaries) for our type theories in Section 4 and 5. In particular, decidability of type checking should follow using [4].

References

1. Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proceedings of the ACM on Programming Languages*, 2(POPL):23:1–23:29, 2018. `doi:10.1145/3158111`.
2. Ali Assaf. A calculus of constructions with explicit subtyping. In *20th International Conference on Types for Proofs and Programs, TYPES*, pages 27–46, 2014.
3. Franz Baader and Jörg H. Siekmann. Unification theory. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2, Deduction Methodologies*, pages 41–126. Oxford University Press, 1994.
4. Marc Bezem and Thierry Coquand. Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism. *Theoretical Computer Science*, 913:1–7, 2022. `doi:10.1016/j.tcs.2022.01.017`.
5. Marc Bezem, Thierry Coquand, Peter Dybjer, and Martín Escardó. The Burali-Forti argument in HoTT/UF with applications to the type of groups in a universe. `https://www.cs.bham.ac.uk/~mhe/TypeTopology/Ordinals.BuraliForti.html`, 2022.
6. Luca Cardelli. Basic polymorphic typechecking. *Science of Computer Programming*, 8(2):147–172, 1987. `doi:10.1016/0167-6423(87)90019-0`.
7. Mario Carneiro. The type theory of Lean. Master Thesis, Carnegie-Mellon University, 2019.
8. Antoine Chambert-Loir. A presheaf that has no associated sheaf. `https://freedommathdance.blogspot.com/2013/03/a-presheaf-that-has-no-associated-sheaf.html`, 2013.
9. Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
10. Thierry Coquand. Canonicity and normalization for dependent type theory. *Theoretical Computer Science*, 777:184–191, 2019. `doi:10.1016/j.tcs.2019.01.015`.
11. Judicaël Courant. Explicit universes for the calculus of constructions. In *Theorem Proving in Higher Order Logics, TPHOLs*, volume 2410 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2002. `doi:10.1007/3-540-45685-6_9`.
12. Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388, 2015.
13. Martín Hötzel Escardó. Introduction to univalent foundations of mathematics with Agda. *CoRR*, 2019. `arXiv:1911.00580`.
14. Martín Hötzel Escardó et al. TypeTopology. `https://www.cs.bham.ac.uk/~mhe/TypeTopology/index.html`. Agda development.
15. Thiago Felicissimo, Frédéric Blanqui, and Ashish Kumar Barnawal. Translating proofs from an impredicative type system to a predicative one. In *Computer Science Logic (CSL)*, 2023.

16 Jean-Yves Girard. *Thèse d'État*. PhD thesis, Université Paris VII, 1971.
17 Jean Giraud. *Cohomologie non abélienne*. Springer, 1971. doi:10.1007/978-3-662-62103-5.
18 Robert Harper and Robert Pollack. Type checking with universes. *Theoretical Computer Science*, 89:107–136, 1991.
19 Hugo Herbelin. Type inference with algebraic universes in the Calculus of Inductive Constructions. http://pauillac.inria.fr/~herbelin/articles/univalgcci.pdf, 2005.
20 Gérard Huet. Extending the calculus of constructions with Type:Type. unpublished manuscript, April 1987.
21 Per Martin-Löf. On the strength of intuitionistic reasoning. Preprint, Stockholm University, 1971.
22 Per Martin-Löf. A theory of types. Preprint, Stockholm University, 1971.
23 Per Martin-Löf. An intuitionistic theory of types. Preprint, Stockholm University, 1972.
24 Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North Holland, 1975.
25 Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, pages 153–175. North-Holland, 1982.
26 Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
27 Egbert Rijke, Elisabeth Bonnevier, Jonathan Prieto-Cubides, Fredrik Bakke, and others. Univalent mathematics in Agda. https://github.com/UniMath/agda-unimath/.
28 Carlos Simpson. Computer theorem proving in mathematics. *Letters in Mathematical Physics*, 69(1-3):287–315, July 2004. doi:10.1007/s11005-004-0607-9.
29 Matthieu Sozeau. Explicit universes. https://coq.inria.fr/refman/addendum/universe-polymorphism.html#explicit-universes.
30 Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in Coq. In *Interactive Theorem Proving (ITP)*, 2014.
31 Thomas Streicher. *Semantics of Type Theory*. Birkhäuser, 1991.
32 Agda team. The Agda manual. URL: https://agda.readthedocs.io/en/v2.6.2.1/.
33 The 1Lab Development Team. The 1Lab. https://1lab.dev.
34 The Agda Community. Agda standard library. https://github.com/agda/agda-stdlib.
35 The Coq Community. Coq. https://coq.inria.fr.
36 The Cubical Agda Community. A standard library for Cubical Agda. https://github.com/agda/cubical.
37 The HoTT-Agda Community. HoTT-Agda. https://github.com/HoTT/HoTT-Agda.
38 François Thiré. *Interoperability between proof systems using the logical framework Dedukti. (Interopérabilité entre systèmes de preuve en utilisant le cadre logique Dedukti)*. PhD thesis, École normale supérieure Paris-Saclay, Cachan, France, 2020. URL: https://tel.archives-ouvertes.fr/tel-03224039.
39 Vladimir Voevodsky. Universe polymorphic type system. http://www.math.ias.edu/Voevodsky/voevodsky-publications_abstracts.html#UPTS, 2014.
40 William C. Waterhouse. Basically bounded functors and flat sheaves. *Pacific Math. J*, 57(2):597–610, 1975.

A Formulation with cumulativity

We introduce an operation $\mathsf{T}_l^m(A) : \mathsf{U}_m$ if $A : \mathsf{U}_l$ and $l \leqslant m$ (i.e., $m = l \vee m$).[4]

We require $\mathsf{T}_m(\mathsf{T}_l^m(A)) = \mathsf{T}_l(A)$. Note that this yields, e.g., $a : \mathsf{T}_m(\mathsf{T}_l^m(A))$ if $a : \mathsf{T}_l(A)$. We also require $\mathsf{T}_l^m(\mathsf{N}^l) = \mathsf{N}^m$ $(l \leqslant m)$, and $\mathsf{T}_l^m(\mathsf{U}_k^l) = \mathsf{U}_k^m$ $(k < l \leqslant m)$, as well as $\mathsf{T}_l^m(A) = A$ $(l = m)$ and $\mathsf{T}_m^n(\mathsf{T}_l^m(A)) = \mathsf{T}_l^n(A)$ $(l \leqslant m \leqslant n)$, for all $A : \mathsf{U}_l$.

[4] Recall that the equality of universe levels is the one of sup-semilattice with the $\_^+$ operation.

We can then simplify the product and sum rules to

$$\frac{A : \mathsf{U}_l \qquad B : \mathsf{T}_l(A) \to \mathsf{U}_l}{\Pi^l AB : \mathsf{U}_l} \qquad\qquad \frac{A : \mathsf{U}_l \qquad B : \mathsf{T}_l(A) \to \mathsf{U}_l}{\Sigma^l AB : \mathsf{U}_l}$$

with conversion rules

$$\mathsf{T}_l\ (\Pi^l AB) = \Pi_{x:\mathsf{T}_l(A)} \mathsf{T}_l(B\ x) \qquad \mathsf{T}_l\ (\Sigma^l AB) = \Sigma_{x:\mathsf{T}_l(A)} \mathsf{T}_l(B\ x)$$

and

$$\mathsf{T}_l^m\ (\Pi^l AB) = \Pi^m \mathsf{T}_l^m(A)(\lambda_{x:\mathsf{T}_l(A)} \mathsf{T}_l^m(B\ x)) \quad \mathsf{T}_l^m\ (\Sigma^l AB) = \Sigma^m \mathsf{T}_l^m(A)(\lambda_{x:\mathsf{T}_l(A)} \mathsf{T}_l^m(B\ x))$$

Recall the family $\mathsf{Id}^l(A,a,b) : \mathsf{U}_l$ for $A : \mathsf{U}_l$ and $a : \mathsf{T}_l(A)$ and $b : \mathsf{T}_l(B)$, with judgemental equality $\mathsf{T}_l(\mathsf{Id}^l(A,a,b)) = \mathsf{Id}(\mathsf{T}_l(A),a,b)$. We add the judgmental equalities $\mathsf{T}_l^m(\mathsf{Id}^l(A,a,b)) = \mathsf{Id}^m(\mathsf{T}_l^m(A),a,b)$; note that a and b are well-typed since $\mathsf{T}_m(\mathsf{T}_l^m(A)) = \mathsf{T}_l(A)$.

Example. Recall the type $\mathsf{Eq}^{l,l}(A,B) : \mathsf{U}_l$ for A and B in U_l, with judgmental equality $\mathsf{T}_l(\mathsf{Eq}^{l,l}(A,B)) = \mathsf{Equiv}(\mathsf{T}_l(A),\mathsf{T}_l(B))$. For $m > l$, a consequence of univalence for U_m and U_l is that we can build an element of the type

$$\mathsf{Id}(\mathsf{U}_m, \mathsf{Eq}^{m,m}(\mathsf{T}_l^m(A),\mathsf{T}_l^m(B)), \mathsf{Id}^m(\mathsf{U}_l^m, A, B)).$$

B Notions of model and formulation à la Russell

Generalised algebraic presentation

In a forthcoming paper, we plan to present some generalised algebraic theories of level-indexed categories with families with extra structure. The models of these theories provide suitable notions of model of our type theories with level judgments. Moreover, the theories presented in this paper are initial objects in categories of such models.

▶ Remark 2. As explained in [31], in order to see the theories in this paper as presenting *initial* models, it is enough to use a variation where application $c\ a : B(a/x)$ for $c : \Pi_{x:A}B$ and $a : A$ is annoted by the type family A, B (and similarly for the pairing operation). If the theories satisy the normal form property, it can then be shown that also the theories without annotated application are initial.

Russell formulation

Above, we presented type theories with universe level judgments *à la Tarski*. There are alternative formulations *à la Russell* (using the terminology introduced in [26] of universes). One expects these formulations to be equivalent to the Tarski-versions, and thus also initial models. For preliminary results in this direction see [2, 38].

With this formulation, the version without cumulativity becomes

$$\frac{A : \mathsf{U}_n}{A\ \mathsf{type}}$$

$$\frac{A : \mathsf{U}_n \qquad B : \mathsf{U}_m(x : A)}{\Pi_{x:A}B : \mathsf{U}_{n \vee m}} \qquad\qquad \frac{A : \mathsf{U}_n \qquad B : \mathsf{U}_m(x : A)}{\Sigma_{x:A}B : \mathsf{U}_{n \vee m}}$$

$$\frac{l\ \mathsf{level}}{\mathsf{N} : \mathsf{U}_l}$$

$$\frac{A : \mathsf{U}_n \qquad a_0 : A \qquad a_1 : A}{\mathsf{Id}(A, a_0, a_1) : \mathsf{U}_n}$$

$$\frac{l < n}{\mathsf{U}_l : \mathsf{U}_n}$$

For the version with cumulativity, we add the rules

$$\frac{A : \mathsf{U}_l \qquad l \leqslant n}{A : \mathsf{U}_n}$$

and the rules for products and sums can be simplified to

$$\frac{A : \mathsf{U}_n \qquad B : \mathsf{U}_n \ (x : A)}{\Pi_{x:A} B : \mathsf{U}_n} \qquad\qquad \frac{A : \mathsf{U}_n \qquad B : \mathsf{U}_n \ (x : A)}{\Sigma_{x:A} B : \mathsf{U}_n}$$

For $m > l$ the consequence of univalence for U_m and U_l mentioned in Appendix A can now be written simply as

$$\mathsf{Id}(\mathsf{U}_m, \mathsf{Equiv}(A, B), \mathsf{Id}(\mathsf{U}_l, A, B)).$$

▶ Remark 3. In the version à la Tarski, with or without cumulativity, terms have unique types, in the sense that if $t : A$ and $t : B$ then $A = B$, by induction on t. But for this to be valid, we need to annotate application as discussed in Remark 2. Even with annotated application, the following property is not elementary: if U_n and U_m are convertible then n is equal to m. This kind of property is needed for showing the equivalence between the Tarski and the Russell formulation.

▶ Remark 4. If, in a system without cumulativity, we extend our system of levels with a least level 0, then if we restrict N to be of type U_0, and U_n to be of type U_{n+1} then well formed terms have unique types.

▶ Remark 5. It should be the case that the above formulation à la Russell presents the initial CwF with extra extructure for the standard type formers and a hierarchy of universes, but the proof doesn't seem to be trivial, due to Remark 3.