Subtype Universes

Harry Maclean
Royal Holloway, University of London, UK

Zhaohui Luo
Royal Holloway, University of London, UK

Abstract

We introduce a new concept called a subtype universe, which is a collection of subtypes of a particular type. Amongst other things, subtype universes can model bounded quantification without undecidability. Subtype universes have applications in programming, formalisation and natural language semantics. Our construction builds on coercive subtyping, a system of subtyping that preserves canonicity. We prove Strong Normalisation, Subject Reduction and Logical Consistency for our system via transfer from its parent system UTT[$\mathbb{C}$]. We discuss the interaction between subtype universes and other sorts of universe and compare our construction to previous work on Power types.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type theory, coercive subtyping, subtype universe

Digital Object Identifier 10.4230/LIPIcs.TYPES.2020.9

1 Introduction

In this paper we define a new sort of universe, which we call a subtype universe. The key idea is embodied in the following two pseudo-rules:

$$\text{UNIVERSE-FORM}\ \frac{\Gamma \vdash A : \mathbf{Type}}{\Gamma \vdash U(A) : \mathbf{Type}} \qquad \text{UNIVERSE-INTRO}\ \frac{\Gamma \vdash B \leq A}{\Gamma \vdash B : U(A)}$$

The first rule states that for any type A, there is a type $U(A)$ which we call the subtype universe of A. The second rule states that any subtype of A is an object of $U(A)$. $U(A)$ is therefore a type representing a collection of all subtypes of A. It is similar to universes such as $Type_0$ in that its objects are types (technically, names for types), but whilst $Type_0$ contains all types (at the time of formation, at least), the membership of a type in $U(A)$ is based on the presence of a subtyping judgement between the type in question and A.

Subtype universes provide a simple model for *bounded quantification*, a concept first introduced by Cardelli and Wegner for the language Fun[5]. Bounded quantification extends the notion of parametric polymorphism with support for subtypes. In a system with support for subtyping, the bounded quantifier $\Pi A \leq B.T$ binds a type A in the body T under the constraint that A is a subtype of B. In essence, bounded quantification allows a function to be defined over all subtypes of a particular type.

A typical use of bounded quantification is in writing operations on records. Consider a system with record types similar to [13], although for simplicity without dependence of record fields on each other. We write record types as $R := \langle\rangle \mid \langle R, l : A\rangle$ and records as $r := \langle\rangle \mid \langle r, l = a : A\rangle$. Record types each have a corresponding kind $RType[L]$, where L is the set of labels occurring in the record type. We can define the following function, which translates a one-dimensional point by a given amount to the right.

$$translateX : Nat \to \langle x : Nat\rangle \to \langle x : Nat\rangle$$
$$translateX(n, r) = set(r, x, r.x + n)$$

$\langle x : Nat \rangle$ is a record type containing a single field x of type Nat. $set(r, l, a)$ is a primitive operation on records which updates the record r, setting the value of the field labelled l to a. $r.l$ selects the value in the record r corresponding to the label l.

We may wish to apply this function to coordinates in higher dimensions, such as objects of type $\langle x : Nat, y : Nat \rangle$. This prompts us to consider a subtyping relation for records, and there is a natural one arising from the record restriction operator $[r]$, which removes the outermost field of a record: $\langle R, l : A \rangle \leq_{[\_]} R$. In other words, any extension of a record type R by additional fields produces a subtype of R. Thus we have $\langle x : Nat, y : Nat \rangle \leq \langle x : Nat \rangle$. We can now apply $translateX$ to objects of $\langle x : Nat, y : Nat \rangle$, but this will implicitly downcast them, as the result type is $\langle x : Nat \rangle$.

Bounded quantification solves this problem by introducing quantification over subtypes. In this way the original subtype is named and can be given in the result type. With bounded quantification we can write the type of $translateX$ as

$$translateX : \Pi R \leq \langle x : Nat \rangle . Nat \to R \to R$$

which describes a function that takes an argument of some arbitrary record type R constrained to be a subtype of $\langle x : Nat \rangle$. When a function of this type is applied to an argument of type $\langle x : Nat, y : Nat \rangle$, R is instantiated to $\langle x : Nat, y : Nat \rangle$ and the result is an object of the same type.

System $F_{\leq}$[4] is System F[9, 22] extended with subtyping and bounded quantification, and is a foundation for much of the research on subtyping in functional programming languages. However there is one drawback: typechecking in $F_{\leq}$ is undecidable [20]. The crux of the problem is the rule for subtyping between bounded quantifiers:

$$\frac{\Gamma \vdash A_1 \leq B_1 \qquad \Gamma, x \leq A_1 \vdash B_2 \leq A_2}{\Gamma \vdash \Pi(x \leq B_1).\ B_2 \ \leq \ \Pi(x \leq A_1).\ A_2}$$

When combined with a Top type, of which every type is a subtype, this rule causes the subtyping relation to become undecidable, which in turn causes typechecking to become undecidable [2]. Various modifications have been proposed to get around this problem[6, 23]. For example, disallowing Top in the bounds of quantifiers or requiring $A_1 = B_1$ in the rule above. Each has its own trade-offs in terms of expressiveness and algorithmic practicality.

Because of this undecidability result and the difficulties in extending System F with bounded quantification, many researchers have thought that extending dependent type theories with bounded quantification would also be problematic, or at least, it would not be an easy task. This has turned out to be mistaken. We take up this challenge in this paper and show that bounded quantification can be modelled by subtype universes in a way that maintains nice meta-theoretic properties. Moreover, our system is a full dependent type theory, providing richer types than $F_{\leq}$.

With subtype universes, bounded quantification can be modelled using normal Π types: the type $\Pi X : U(A).B$ is equivalent to $\Pi X \leq A.B$. However it is important to note that Π types cannot model all uses of subtype universes. A subtype universe $U(A)$ is a type of types, whereas Π is a type of functions. Using subtype universes we can construct types such as $A \to U(A)$. The right hand side of this function type is a type whose objects are subtypes of A, and there is no equivalent to this using Π types.

We use coercive subtyping, which is a subtyping system well suited to type theories due to its preservation of canonicity. The system UTT[$\mathbb{C}$] is an extension of UTT [11] with coercive subtyping. We further extend this system with support for subtype universes, forming the system UTT[$\mathbb{C}$]$_U$. The extension consists of a handful of new syntactic forms and six new

typing rules, which are described in Section 2. Working in a dependent type theory rather than a weaker language allows us to apply the concept to a wide range of fields. Section 3 describes examples applicable to programming, formalisation and natural language semantics. Section 4 proves several important meta-theoretic properties, including logical consistency and strong normalisation. Section 5 discusses the design decisions we have taken and some interesting alternatives.

2 Subtype Universes

In Martin-Löf's intuitionistic type theory [17] the concept of a universe is introduced to represent a collection of types which is closed under specific type-forming operations. Typically one starts by defining a group of base type-forming operations and then one defines a universe $Type_0$ which contains the closure of these operations. $Type_0$ is itself a type, and can be used in combination with other type-forming operations to form new types. For example, we can construct the polymorphic identity function $\Pi T : Type_0.\ \Pi x : T.\ T$. A function with this type can be applied to any type in $Type_0$, but not to $Type_0$ itself. We can construct a more powerful type for the identity function by repeating the process: we define a new universe $Type_1$ which contains the closure of all type-forming operations *including* $Type_0$. We can then construct the type $\Pi(T : Type_1).\ \Pi(x : T).\ T$, objects of which can be applied to $Type_0$. This process can be iterated indefinitely, forming a hierarchy of predicative universes, each one stronger than the previous one. This hierarchy allows us to quantify over arbitrarily large collections of types, providing great proof-theoretic strength. At the same time, the absence of a "type of all types" means that it neatly avoids Girard's Paradox[8]. Intuitively, this construction of universes is an application of the reflection principle well known to set theory, and there are analogous constructions in other fields (such as the Grothendieck universes of category theory). Universes are typically expressed in either *Tarski* style or *Russell* style. Tarski style is more explicit, and to avoid ambiguity it is the style we use here.

2.1 Tarski style universes

The Tarski formulation introduces a new type for each universe, objects of which are names for other types. Alongside, we introduce a family of operators to map names to their corresponding types. We will briefly walk through this construction, as our system builds on some of the concepts. Firstly, to represent each universe we form a type $Type_i$, where i is a positive integer indicating the level of the universe.

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash Type_i : \mathbf{Type}}$$

Objects of the universe $Type_i$ are names of other types. For each universe we introduce a reflection operator $\mathbf{T}_i$ which maps names in $Type_i$ to their corresponding types.

$$\frac{\Gamma \vdash a : Type_i}{\Gamma \vdash \mathbf{T}_i(a) : \mathbf{Type}}$$

Now whenever we introduce a new type to the system, we also introduce its name in each universe. For example, if we introduce a type Nat of natural numbers then we would also add the following axiom, which states that each universe contains a name nat for Nat.

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash nat_i : Type_i}$$

We also add rules defining how $\mathbf{T}_i$ behaves on these new names:

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash \mathbf{T}_i(nat_i) = Nat : \mathbf{Type}}$$

Finally, for each universe we introduce a lifting operator $\mathbf{t}_i$ which "lifts" names from a universe $Type_i$ into the successive universe $Type_{i+1}$.

$$\frac{\Gamma \vdash a : Type_i}{\Gamma \vdash \mathbf{t}_i(a) : Type_{i+1}}$$

$$\frac{\Gamma \vdash a : Type_i}{\Gamma \vdash \mathbf{T}(\mathbf{t}_i(a)) = \mathbf{T}(a) : \mathbf{Type}}$$

Thus (informally) for the type Nat we have a name nat_0 in $Type_0$, and $\mathbf{T}_0(nat_0)$ yields Nat. We can apply the lifting operator $\mathbf{t}_1$ to this name, giving $\mathbf{t}_1(nat_0) = nat_1$, and of course $\mathbf{T}_1(nat_1) = Nat$. We can lift again, giving $\mathbf{t}_2(nat_1) = nat_2$, and so on.

2.2 Coercive subtyping

Coercive subtyping [12, 15] is a model of subtyping in type theories which expresses the subtyping relationship via a specific *coercion*, which is a function from the subtype to the supertype. It is a powerful form of subtyping which is particularly well suited to type theories with canonical objects, as it preserves canonicity [15]. A type theory T can be extended with coercive subtyping by adding two new judgement forms, for subtyping and subkinding. We will focus on the former; for a full description of this extension we refer the reader to [15]. The subtyping judgement is written $\Gamma \vdash A \leq_c B : \mathbf{Type}$. It declares that A is a proper subtype of B via a coercion $c : (A)B$.[1] There are associated rules that define subtyping to be congruent and transitive, among other things. A particularly important addition is the coercive definition rule:

$$\frac{\Gamma \vdash f : (x : B)C \qquad \Gamma \vdash a : A \qquad \Gamma \vdash A \leq_c B}{\Gamma \vdash f(a) = f(c(a)) : [c(a)/x]C}$$

This rule states that if $A \leq_c B$, then a function f, despite having domain B, may be applied to an object a of A. When this happens, it is equal to $f(c(a))$, which is the application of f to the coercion of a to an object of B, using the specific coercion c. This is the primary mechanism by which coercive subtyping relations are put to work.

[1] The systems described in this paper are defined in the meta-level framework LF, which is a typed version of Martin-Löf's Logical Framework. Where appropriate we will use LF syntax. In brief, $(x : A)B$ is the type of a meta-level function from A to B, where $x : A$ is bound in B. $[x : A]b$ is a meta-level function from an object of A to an expression b, where $x : A$ is bound in b. See [11] for a full description of the language.

Finally, we add a set $\mathbb{C}$ of axiomatic subtyping judgements. The only restriction on $\mathbb{C}$ is that it must be *coherent*. This means that all coercions between any two types A and B must be the same, i.e. $A \leq_c B$ and $A \leq_{c'} B$ implies $c = c'$. We can then write $T[\mathbb{C}]$ for the system formed by extending T with coercive subtyping and a coherent set $\mathbb{C}$ of axiomatic subtyping judgements. A key property of this process is that $T[\mathbb{C}]$ is a conservative extension of T: any T-judgement is derivable in $T[\mathbb{C}]$ if and only if it is derivable in T.

Coercive subtyping is a conservative extension of type theory, as stated in the following result, where T is either the type theory UTT [11] or Martin-Löf's type theory [19]. For example, a corollary of the following theorem is that $T[\mathbb{C}]$ is logically consistent if T is.

▶ **Theorem 1** (Conservativity [15]). *For any coherent set $\mathbb{C}$ of coercion judgements, $T[\mathbb{C}]$ is equivalent to a system that is a conservative extension of the type theory T.*

Extending the type theory UTT [11] with coercive subtyping yields the system UTT[$\mathbb{C}$] ("replacing" T by UTT), and it is this system that we build on in this paper. Specifically, we extend UTT[$\mathbb{C}$] with additional syntax and rules concerning subtype universes, yielding a system we call UTT[$\mathbb{C}$]$_U$.

2.3 The system UTT[$\mathbb{C}$]$_U$

Our system is an extension of UTT [11], although any type theory with a predicative universe hierarchy is suitable. The extension consists of some new syntactic forms and six new typing rules. The syntactic forms are $U_i(A)$, $\mathbf{T}^A_{U_i}$, $n(A)$ and $u_i(A)$. $U_i(A)$ is a subtype universe parameterised by the type A. $\mathbf{T}^A_{U_i}$ is an operator parameterised by the subtype universe $U_i(A)$. $n(A)$ is a meta-level operation which gives the name for the type A. Similarly, $u_i(A)$ is a meta-level operation which gives the name for the subtype universe $U_i(A)$. These forms are given meaning via six new typing rules. The rules are given in Figure 1. They can be divided into four groups, which we call formation (U-FORM), introduction (U-INTRO), reflection (U-REFL1, U-REFL2) and predicativity (U-PRED1, U-PRED2). Note that these are convenient labels rather than precise categorisations.

The formation rule (U-FORM) introduces a subtype universe $U_i(A)$ for every type A which has a name in a traditional universe $Type_i$. We apply a single restriction in the form of the side condition $\mathcal{L}_\Gamma(A) = i$, which requires that the type level of A is equal to i.

Type levels are explained in Definition 2, but informally the level of a type is the index of the smallest traditional universe in which the type has a name. For example, $Bool$ has type level 0 whilst $Type_0$ has type level 1. Type levels are important because they allow us to determine "size" of a type. By annotating every subtype universe with a type level, we syntactically expose a lower bound for the type when placing it in the traditional universe hierarchy. For example, we cannot allow the subtype universe $U_2(Type_1)$ to have a name in $Type_0$, as $Type_1$ is a strictly larger type. We prevent this by ensuring that any subtype universe $U_i(A)$ has a name only in the traditional universe $Type_{i+1}$ (see rules (U-PRED1) and (U-PRED2)).

▶ **Definition 2** (Type Level). *For any type A in a context Γ, in UTT[$\mathbb{C}$] or UTT[$\mathbb{C}$]$_U$, its type level $\mathcal{L}_\Gamma(A)$ is defined as follows:*

- *If $\exists P.\ \Gamma \vdash Prf(P) = A : Type$, $\mathcal{L}_\Gamma(A) =_{df} -1$*
- *Otherwise, $\mathcal{L}_\Gamma(A)$ is the least m such that $\exists a.\ \Gamma \vdash \mathbf{T}_m(a) = A : \mathbf{Type}$*

Since every type in UTT[$\mathbb{C}$] and UTT[$\mathbb{C}$]$_U$ has a name in some universe, Definition 2 is well defined for all types.

$$\text{U-FORM}\quad \frac{\Gamma \vdash \mathbf{T}_i(a) = A : \mathbf{Type}}{\Gamma \vdash U_i(A) : \mathbf{Type}}\ (\mathcal{L}_\Gamma(A) = i)$$

$$\text{U-INTRO}\quad \frac{\Gamma \vdash \mathbf{T}_i(a) = A : \mathbf{Type} \qquad \Gamma \vdash B \leq_c A : \mathbf{Type}}{\Gamma \vdash n(B) : U_i(A)}\ (\mathcal{L}_\Gamma(B) \leq \mathcal{L}_\Gamma(A))$$

$$\text{U-REFL1}\quad \frac{\Gamma \vdash n(B) : U_i(A)}{\Gamma \vdash \mathbf{T}^A_{U_i}(n(B)) : \mathbf{Type}}$$

$$\text{U-REFL2}\quad \frac{\Gamma \vdash \mathbf{T}_i(a) = A : \mathbf{Type} \qquad \Gamma \vdash B \leq_c A : \mathbf{Type}}{\Gamma \vdash \mathbf{T}^A_{U_i}(n(B)) = B : \mathbf{Type}}\ (\mathcal{L}_\Gamma(B) \leq \mathcal{L}_\Gamma(A))$$

$$\text{U-PRED1}\quad \frac{\Gamma \vdash U_i(A) : \mathbf{Type}}{\Gamma \vdash u_i(A) : Type_{i+1}} \qquad \text{U-PRED2}\quad \frac{\Gamma \vdash U_i(A) : \mathbf{Type}}{\Gamma \vdash \mathbf{T}_{i+1}(u_i(A)) = U_i(A) : \mathbf{Type}}$$

Figure 1 The typing rules for subtype universes. The extension of UTT[$\mathbb{C}$] by these rules forms the system UTT[$\mathbb{C}$]$_U$.

It is important to note that because proof types (types of the form $\mathbf{Prf}(P)$ for some proposition P) have a defined type level of -1, we cannot form subtype universes of them. This is because the premiss of (U-FORM) is $\Gamma \vdash \mathbf{T}_i(a) = A : \mathbf{Type}$, where $i = \mathcal{L}_\Gamma(A)$. There is no operator $\mathbf{T}_{-1}$ in UTT[$\mathbb{C}$], and therefore we cannot derive this judgement for proof types. Proof types can have names in other subtype universes, if there exists a corresponding subtyping relation, but the inverse is not possible. Intuitively, proof types are not data types and one usually does not consider subtyping relationships between them. We therefore do not consider subtype universes of a proof type. This decision is a point in the design space and there are alternative options. We discuss some of these in Section 5.

The introduction rule (U-INTRO) defines the membership of subtype universes. If a type B is a subtype of A, then its name, given by $n(B)$, is an object of the subtype universe of A. As we shall see, we will be able to convert from $n(B)$ to B. In this way we represent the concept that B is a "member" of $U_i(A)$. Again, there is an additional restriction on this rule: the type level of B must not be greater than the type level of A. This restriction ensures that we can translate derivations in our system into derivations in UTT[$\mathbb{C}$], and is critical in proving some meta-theoretic properties, as we will describe shortly.

We now have a connection between subtypes of A and their corresponding names in $U_i(A)$. The reflection rules (U-REFL1) and (U-REFL2) complete the circle by relating the names back to their subtypes. (U-REFL1) introduces an operator $\mathbf{T}^A_{U_i}$, which is parameterised by a type A and its type level i. For any object $n(B)$ in $U_i(A)$, $\mathbf{T}^A_{U_i}(n(B))$ is a type. (U-REFL2) then tells us what type: $\mathbf{T}^A_{U_i}(n(B))$ is equal to the type B. This rule has the

same side condition as (U-INTRO), which has no effect on the semantics but simplifies the metatheory. Together, these four rules are effectively a translation of (UNIVERSE-FORM) and (UNIVERSE-INTRO) into the Tarski universe formulation.

Finally, we relate subtype universes to the traditional universe hierarchy. Rules (U-PRED1) and (U-PRED2) state that a subtype universe $U_i(A)$ has a name in the universe $Type_{i+1}$. Placing subtype universes into the traditional universes is a design choice, rather than a necessary construction. We discuss this approach and alternatives in Section 5.

3 Applications

Subtype universes have a clear use as a way to model bounded quantification, and this section describes some examples in programming, formalisation and natural language semantics. As subtype universes are first-class types in the system they are inherently more flexible than bounded quantifiers and we expect there are other applications to be discovered.

3.1 Bounded Quantification

With subtype universes we can straightforwardly construct an equivalent to the bounded quantifier $\Pi A \leq B$. Continuing our running example, the function $translateX$ can be given the following type:

$$translateX : \Pi r : U_i(\langle x : Nat \rangle).Nat \to \mathbf{T}_{U_i}^{\langle x:Nat \rangle}(r) \to \mathbf{T}_{U_i}^{\langle x:Nat \rangle}(r)$$

Applying $translateX$ to $\langle x = 1, y = 2 \rangle$ gives a result of type $\mathbf{T}_{U_i}^{\langle x:Nat \rangle}(n(\langle x : Nat, y : Nat \rangle))$, which reduces to $\langle x : Nat, y : Nat \rangle$. Thus we retain the information that the result is an two-dimensional coordinate.

This kind of extensibility has many applications in programming, where it is useful to be able to deal with *partially specified* data. As a software system is evolved, data is often embellished with new fields. Functions like $translateX$ will continue to work as new fields are added to the records it is applied to, allowing for easy and type-safe system extension.

Being a type itself, a subtype universe provides more flexibility than bounded quantification. For example, a subtype universe can appear in both the domain and codomain position of a function type, whereas bounded quantification is only valid in the domain. For example, we can construct types such as $A \to U_0(A)$, which are functions from objects of type A to subtypes of A. Another example is the type

$$U_0(A) \to U_0(B) \to U_0(\Sigma(A, [x : A]B))$$

Given a subtype of A and a subtype of B, a function of this type will return a subtype of their sum, $\Sigma(A, [x : A]B)$.

3.2 Extending predicates to subtypes

With coercive subtyping it is straightforward for a predicate $P : (x : A)Prop$ on some type A to be extended to all subtypes of A, since we can always convert objects of a subtype to objects of the supertype. For example, given $B \leq_c A$ and $b : B$ then $P(b)$ becomes $P(c(b))$ after coercion insertion, which is well typed.

However this fact is not expressed in the type. We rely on meta-level reasoning to know that the domain of P is implicitly extended to all subtypes of A. If we rewrite P to use subtype universes we can better express this property:

$$P : (t : U_i(A))(x : \mathbf{T}_{U_i}^{A}(t))Prop$$

Here P is now a predicate on subtypes of A. Its first argument is the name of a type in the subtype universe for A, and its second argument is an object of that type, as before. We can apply P both to A and its subtypes: $P(n(A), a)$ and $P(n(B), b)$ are both well typed, assuming $a : A$ and $b : B$.

3.3 Natural language semantics

As well as applications in programming, subtype universes have proved to be useful in formalising the semantics of natural language. In order to describe this application we will first introduce some basic concepts in natural language semantics. Then we will describe how subtype universes can model gradable adjectives.

3.3.1 Montague Grammar

The seminal treatment of natural language semantics is the system developed by Montague in the 1970s [18]. Known as Montague Grammar, this system uses an embedding of higher-order logic in the simply typed lambda calculus to model sentences of natural language. Language constructs are divided into categories: sentences, verb phrases, noun phrases, and common nouns, amongst others. Each category is assigned a type with respect to the two atomic types e and t, representing objects and propositions respectively. A complete sentence (e.g. "Socrates is a man") is regarded as a proposition, and thus has type t. Verb phrases such as "is a man" form complete propositions when supplied with an object, and therefore have type $e \to t$. Common nouns are interpreted as predicates. For example, the common noun "man" is represented by the function $\lambda x.man(x)$. Common nouns therefore have the type $e \to t$. We can also make use of logical operators such as implication ($\Rightarrow$), universal quantification ($\forall$) and existential quantification ($\exists$): the sentence "all men are mortal" can be expressed as $\forall x.\ man(x) \Rightarrow mortal(x)$.

There are two notable downsides to this approach. Firstly, the use of a single type e for all objects means there is no distinction between different classes of objects, and we can therefore form nonsensical sentences such as "the colour green plays football". Secondly, the interpretation of common nouns (and indeed, noun phrases) is not intuitive. One would naturally expect common nouns to be interpreted simply as objects.

3.3.2 MTT Semantics

An alternative model of natural language based on Modern Type Theories (MTTs) [21][14] provides a solution to these problems. In MTT semantics, common nouns are interpreted as types in a type theory such as UTT. The interpretation of "man" is as the type Man, and the sentence "Socrates is a man" is interpreted as $Socrates : Man$. We can construct as many types as necessary to precisely describe the context, for example a type Man representing men and a type $Human$ representing humans. This naturally leads to problems when a particular object can be seen to inhabit multiple types. For example, both of the judgements $Socrates : Man$ and $Socrates : Human$ seem reasonable. To solve this we can apply coercive subtyping. We might define $Man \leq_c Human$ via some coercion c, and then we have $Socrates : Man$ and $c(Socrates) : Human$. A full comparison of Montague grammar with MTT semantics is not within the scope of this paper; we refer the reader to [7] for details.

In the context of MTT semantics, subtype universes turn out to be useful in modelling gradable adjectives [16]. Gradable adjectives (words such as "tall") can be interpreted as predicates which involve comparison of a measurement on the entity with some threshold

$$
\begin{aligned}
\delta : \mathrm{UTT}[\mathbb{C}]_U &\to \mathrm{UTT}[\mathbb{C}] \\
\delta(U_i(A)) &= Type_i \\
\delta(u_i(A)) &= type_i \\
\delta(\mathbf{T}^A_{U_i}) &= \mathbf{T}_i \\
\delta(\mathbf{Type}) &= \mathbf{Type} \\
\delta(\mathrm{El}(A)) &= \mathrm{El}(\delta(A)) \\
\delta([x : A]B) &= [x : \delta(A)]\delta(B) \\
\delta((x : A)B) &= (x : \delta(A))\delta(B) \\
\delta(f(x)) &= \delta(f)(\delta(x)) \\
\delta(c) &= c
\end{aligned}
$$

Figure 2 The transformation δ converts terms in $\mathrm{UTT}[\mathbb{C}]_U$ to terms in $\mathrm{UTT}[\mathbb{C}]$.

value. In the case of "tall", the measurement is the height of the argument. Furthermore, the threshold often varies based on the type of the argument. The threshold height that is considered tall for a human is very different from the height considered tall for a building. We will describe how "tall" can be interpreted in $\mathrm{UTT}[\mathbb{C}]_U$, thereby motivating the use of subype universes in formal semantics.

We first collect together into a new universe T all the common nouns for which it makes sense to measure a height. T may contain, amongst others, (names for) the types $Human$ and $Building$. Each of these may have subtypes such as $Man \leq Human$. We will define a function $height$ which measures the height of an object of some type in the universe T, and a function ξ which calculates the threshold height for a particular type to be considered tall:

$$
\begin{aligned}
height :&\ \Pi A : T.\Pi B : U_0(A).B \to Nat \\
\xi :&\ \Pi A : T.U_0(A) \to Nat
\end{aligned}
$$

Note that $height$ and ξ are defined over all subtypes of all types in T. For simplicity, we assume that all types in T have names in the universe $Type_0$. We can then define $tall$ as follows:

$$
\begin{aligned}
&tall : \Pi A : T.\ \Pi B : U_0(A).B \to Prop \\
&tall(A, B, x) = height(A, B, x) \geq \xi(A, B)
\end{aligned}
$$

Compared to other approaches subtype universes provide a simpler semantic construction for gradable adjectives, and may be useful in modelling other linguistic features.

4 Metatheory

The system $\mathrm{UTT}[\mathbb{C}]_U$ retains many of the nice meta-theoretic properties of its base system $\mathrm{UTT}[\mathbb{C}]$: logical consistency, strong normalisation and subject reduction. In particular, logical consistency and strong normalisation results can be transferred from $\mathrm{UTT}[\mathbb{C}]$ because our new rules are admissible in $\mathrm{UTT}[\mathbb{C}]$ after applying a simple syntactic transformation. This transformation is named δ, and it converts terms of $\mathrm{UTT}[\mathbb{C}]_U$ to terms of $\mathrm{UTT}[\mathbb{C}]$, as shown in Figure 2.

For any type A, δ converts the universe $U_i(A)$ to $Type_i$, which is a valid type in UTT[$\mathbb{C}$]. For any object A, δ converts $u_i(A)$ to $type_i$. For any object a, δ converts an application of the subtype universe lifting operator $\mathbf{T}^A_{U_i}(a)$ to an application of the traditional universe lifting operator $\mathbf{T}_i(a)$. For any constant c, δ leaves c unchanged. The translation is extended to other syntax forms by recursion on their structure, ensuring that the result contains none of the syntax introduced by our extension. Using δ we can transform the typing rules in Figure 1, producing rules in the syntax of UTT[$\mathbb{C}$]. These are shown in Figure 3.

In this section we will need to refer to derivations in both UTT[$\mathbb{C}$]$_U$ and UTT[$\mathbb{C}$]. To avoid confusion we will use $\vdash_U$ for judgements in UTT[$\mathbb{C}$]$_U$ and $\vdash$ for judgements in UTT[$\mathbb{C}$]. Contexts in UTT[$\mathbb{C}$]$_U$ will be written Γ whereas contexts in UTT[$\mathbb{C}$] will typically be of the form $\delta(\Gamma)$.

We first note that the side condition $\mathcal{L}_\Gamma(B) \leq \mathcal{L}_\Gamma(A)$ in (U-INTRO) and (U-REFL2) provides information about the traditional universes in which A and B have names.

▶ **Lemma 3.** *If* $\Gamma \vdash \mathbf{T}_i(a) = A : \mathbf{Type}$, $\Gamma \vdash B : \mathbf{Type}$ *and* $\mathcal{L}_\Gamma(B) \leq \mathcal{L}_\Gamma(A)$ *then there exists a name* $b : Type_j$ *such that* $\Gamma \vdash \mathbf{T}_j(b) = B : \mathbf{Type}$ *for some* $j \leq i$.

Proof. There are six cases to consider:
1. $\mathcal{L}_\Gamma(B) = \mathcal{L}_\Gamma(A) = -1$
2. $\mathcal{L}_\Gamma(B) = -1$, $\mathcal{L}_\Gamma(A) \geq 0$
3. $\mathcal{L}_\Gamma(B) \geq 0$, $\mathcal{L}_\Gamma(A) = -1$
4. $\mathcal{L}_\Gamma(B) = \mathcal{L}_\Gamma(A) \geq 0$
5. $\mathcal{L}_\Gamma(B) > \mathcal{L}_\Gamma(A) \geq 0$
6. $\mathcal{L}_\Gamma(B) < \mathcal{L}_\Gamma(A) \geq 0$

Cases 3 and 5 violate the premiss $\mathcal{L}_\Gamma(B) \leq \mathcal{L}_\Gamma(A)$ and can be discounted.

For cases 1 and 2 we follow the same reasoning: via the definition of type level (Definition 2) there exists some P_B such that $\Gamma \vdash \mathbf{Prf}(P_B) = B : \mathbf{Type}$. Thus we have $\Gamma \vdash \mathbf{T}_0(\mathbf{t}_0(P_B)) = B : \mathbf{Type}$. We also have $i \geq 0$ (otherwise $\mathbf{T}_i$ would not be defined) and therefore we can choose $j = 0$ and $b = \mathbf{t}_0(P_B)$ to satisfy the conclusion.

Cases 4 and 6 also follow the same reasoning: we use the definition of type level to get $\exists b'.\ \Gamma \vdash \mathbf{T}_{\mathcal{L}_\Gamma(B)}(b') = B : \mathbf{Type}$. Choosing $j = \mathcal{L}_\Gamma(B)$ and $b = b'$ satisfies the conclusion. ◀

▶ **Lemma 4.** *The rules in Figure 3 are admissible in UTT[*$\mathbb{C}$*].*

Proof. The conclusions of (δ-U-FORM), (δ-U-PRED1) and (δ-U-PRED2) are all axioms in UTT, so these rules are trivially admissible. (δ-U-REFL1) follows from the definition of $\mathbf{T}_i$, which has type $(Type_i)\mathbf{Type}$. This leaves (δ-U-INTRO) and (δ-U-REFL2).

For (δ-U-REFL2), we consider the premiss $\delta(\Gamma) \vdash \mathbf{T}_i(a) = \delta(A) : \mathbf{Type}$ and side condition $\mathcal{L}_{\delta(\Gamma)}(\delta(B)) \leq \mathcal{L}_{\delta(\Gamma)}(\delta(A))$. By Lemma 3 we can deduce $\exists j \leq i.\ \delta(\Gamma) \vdash \mathbf{T}_j(b) = \delta(B) : \mathbf{Type}$, where b is the name for $\delta(B)$ in $Type_j$. Since the traditional universes form a cumulative hierarchy we can conclude $\delta(\Gamma) \vdash \mathbf{T}_i(b') = \delta(B) : \mathbf{Type}$, where b' is the name for $\delta(B)$ in $Type_i$.

For (δ-U-INTRO), we follow the same derivation. From $\delta(\Gamma) \vdash \mathbf{T}_i(b) = \delta(B) : \mathbf{Type}$ we can then conclude $\delta(\Gamma) \vdash b : Type_i$. ◀

▶ **Theorem 5** (Logical consistency)**.** *There is no proof* M *such that* $\Gamma \vdash_U M : \forall X : Prop.X$.

Proof. By contradiction. Assume there is an M such that $\vdash_U M : \forall X : Prop.X$. Then it follows that $\vdash \delta(M) : \delta(\forall X : Prop.X)$, and therefore $\vdash \delta(M) : \forall X : Prop.X$. This implies that UTT[$\mathbb{C}$] is logically inconsistent, which is false [15]. ◀

$$\delta\text{-U-FORM}\quad \frac{\delta(\Gamma) \vdash \mathbf{T}_i(a) = \delta(A) : \mathbf{Type}}{\delta(\Gamma) \vdash Type_i : \mathbf{Type}}\ (\mathcal{L}_{\delta(\Gamma)}(\delta(A)) = i)$$

$$\delta\text{-U-INTRO}\quad \frac{\delta(\Gamma) \vdash \mathbf{T}_i(a) = \delta(A) : \mathbf{Type} \quad \delta(\Gamma) \vdash \delta(B) \leq_{\delta(c)} \delta(A) : \mathbf{Type}}{\delta(\Gamma) \vdash n(B) : Type_i}\ (\mathcal{L}_{\delta(\Gamma)}(\delta(B)) \leq \mathcal{L}_{\delta(\Gamma)}(\delta(A)))$$

$$\delta\text{-U-REFL1}\quad \frac{\delta(\Gamma) \vdash n(B) : Type_i}{\delta(\Gamma) \vdash \mathbf{T}_i(n(B)) : \mathbf{Type}}$$

$$\delta\text{-U-REFL2}\quad \frac{\delta(\Gamma) \vdash \mathbf{T}_i(a) = \delta(A) : \mathbf{Type} \quad \delta(\Gamma) \vdash \delta(B) \leq_{\delta(c)} \delta(A) : \mathbf{Type}}{\delta(\Gamma) \vdash \mathbf{T}_i(n(B)) = \delta(B) : \mathbf{Type}}\ (\mathcal{L}_{\delta(\Gamma)}(\delta(B)) \leq \mathcal{L}_{\delta(\Gamma)}(\delta(A)))$$

$$\delta\text{-U-PRED1}\quad \frac{\delta(\Gamma) \vdash Type_i : \mathbf{Type}}{\delta(\Gamma) \vdash type_i : Type_{i+1}}$$

$$\delta\text{-U-PRED2}\quad \frac{\delta(\Gamma) \vdash Type_i : \mathbf{Type}}{\delta(\Gamma) \vdash \mathbf{T}_{i+1}(type_i) = Type_i : \mathbf{Type}}$$

Figure 3 The rules in Figure 1 after transformation by δ.

▶ **Definition 6** (Reduction). *We write $M \rightsquigarrow N$ to mean that applying a single step of reduction to the expression M yields the expression N. We write $M \rightsquigarrow^* N$ to mean that applying zero or more steps of reduction to the expression M yields the expression N.*

▶ **Lemma 7** (δ preserves one-step reduction). *For every term M in $UTT[\![\mathbb{C}]\!]_U$, if $M \rightsquigarrow N$ then in $UTT[\![\mathbb{C}]\!]$ we have $\delta(M) \rightsquigarrow \delta(N)$.*

Proof. See appendix A. ◀

▶ **Lemma 8** (δ preserves multi-step reduction). *For every term M in $UTT[\![\mathbb{C}]\!]_U$, if $M \rightsquigarrow^* N$ then in $UTT[\![\mathbb{C}]\!]$ we have $\delta(M) \rightsquigarrow^* \delta(N)$.*

Proof. There are two cases to consider: a reduction sequence of zero steps and a reduction sequence of one or more steps. For the former case we have $M = N$ and therefore it follows that $\delta(M) = \delta(N)$ and hence $\delta(M) \rightsquigarrow^* \delta(N)$. The latter case follows from Lemma 7 by induction. ◀

▶ **Theorem 9** (Strong Normalisation). *If $\Gamma \vdash_U M : A$ then M is strongly normalisable. In other words, every sequence of reductions starting from M is finite.*

Proof. Assume that Strong Normalisation does not hold for $\mathrm{UTT}[\mathbb{C}]_U$ - i.e. there exists a term M in $\mathrm{UTT}[\mathbb{C}]_U$ with an infinite reduction sequence. By Lemma 8 it follows that there is a corresponding infinite reduction sequence in $\mathrm{UTT}[\mathbb{C}]$ for $\delta(M)$. This is a contradiction because Strong Normalisation holds for $\mathrm{UTT}[\mathbb{C}]$, as it is a conservative extension of UTT [15]. ◀

▶ **Theorem 10** (Subject Reduction). *If* $\Gamma \vdash_U M : A$ *and* $M \rightsquigarrow N$*, then* $\Gamma \vdash_U N : A$.

Proof. UTT$[\mathbb{C}]_U$ is an extension of UTT$[\mathbb{C}]$, for which subject reduction holds. Therefore it is sufficient to show that the new syntax forms and rules we have introduced preserve this property. We proceed by induction on the structure of terms. Most cases follow straightforwardly by the induction hypothesis, except for the two base cases induced by the reduction rules $urefl$ and $upred$, derived from the typing rules (U-REFL2) and (U-PRED2). Given the premises of each rule we can derive identical types for either side of the equality in the conclusion. For $urefl$ we have $M \equiv \mathbf{T}^A_{U_i}(n(B))$, $N \equiv B$ and $M \rightsquigarrow^{urefl} N$. From the premises of (U-REFL2) we can then derive $\Gamma \vdash_U M : \mathbf{Type}$ and $\Gamma \vdash \_UN : \mathbf{Type}$:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash_U \mathbf{T}_i(a) = A : \mathbf{Type} \qquad \Gamma \vdash_U B \leq_c A : \mathbf{Type}}{\Gamma \vdash_U n(B) : U_i(A)}\,(\text{U-INTRO})}{\Gamma \vdash_U \mathbf{T}^A_{U_i}(n(B)) : \mathbf{Type}}\,(\text{U-REFL1})}{\Gamma \vdash_U M : \mathbf{Type}}\,(\text{definition of } M)$$

$$\dfrac{\dfrac{\Gamma \vdash_U A \leq_c B : \mathbf{Type}}{\Gamma \vdash_U B : \mathbf{Type}}}{\Gamma \vdash_U N : \mathbf{Type}}\,(\text{definition of } N)$$

For $upred$ we have $M \equiv \mathbf{T}_{i+1}(u_i(A))$, $N \equiv U_i(A)$ and $M \rightsquigarrow^{upred} N$. From the premiss of (U-PRED2) we already have $\Gamma \vdash_U N : \mathbf{Type}$. We can then derive $\Gamma \vdash_U M : \mathbf{Type}$ as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash_U U_i(A) : \mathbf{Type}}{\Gamma \vdash_U u_i(A) : Type_{i+1}}\,(\text{U-PRED1})}{\Gamma \vdash_U \mathbf{T}_{i+1}(u_i(A)) : \mathbf{Type}}\,(\text{definition of } \mathbf{T}_{i+1})}{\Gamma \vdash_U M : \mathbf{Type}}\,(\text{definition of } M)$$

The four new syntactic forms, $U_i(A)$, $u_i(A)$, $n(B)$ and $\mathbf{T}^A_{U_i}(n(B))$, are irreducible under the existing reduction rules of UTT$[\mathbb{C}]$ and so do not affect the subject reduction property of the original rules. Therefore subject reduction holds for UTT$[\mathbb{C}]_U$. ◀

5 Discussion on design choices

Whilst most aspects of our system follow directly from the pseudo rules given in Section 2 or from metatheoretic constraints (for example, the annotation of subtype universes with type levels), some parts reflect specific design choices that could be modified. These areas concern how subtype universes interact with other universes, such as the traditional predicative universes $Type_i$, the impredicative universe $Prop$ and even other subtype universes. We might ask if a particular subtype universe can have a name in the universe $Type_1$, or vice versa. Or even: can we construct subtype universes containing names for other subtype universes? Whilst these questions are formally interesting, we have not identified any clear applications of alternative designs.

Firstly, our rules permit universes such as $U_1(Type_0)$, meaning that judgements like $Type_0 : U_1(Type_0)$ are derivable. In the opposite direction, the rules (U-PRED1) and (U-PRED2) define that $U_i(A) : Type_{i+1}$. However, these two rules are optional. We include them so that the traditional universes continue to allow quantification over "all types", but the system remains standing if they are removed. There are three main options for this point in the design.

The first option is that subtype universes have names in the same traditional universe as the type they are parameterised by - i.e. $U_i(A) : Type_i$. It is straightforward to show that this option leads to inconsistency when translated to UTT[$\mathbb{C}$]. Applying δ to the judgement $\Gamma \vdash U_i(A) : Type_i$ yields $\delta(\Gamma) \vdash \delta(U_i(A)) : \delta(Type_i)$, which simplifies to $\delta(\Gamma) \vdash Type_i : Type_i$.

The second option is that subtype universes are contained in the traditional universe directly "above" the universe of the type they are parameterised by: $U_i(A) : Type_{i+1}$. So for some base type $T : Type_0$, $U_0(T) : Type_1$. When translated to UTT this judgement becomes (in the general case) $\Gamma \vdash Type_i : \mathrm{Type}_{i+1}$ which is a derivable judgement in UTT. This, therefore, seems the most natural option for the relation between subtype universes and traditional universes, not least because $Type_{i+1}$ is the smallest universe in which we can place $U_i(A)$ without encountering paradoxes.

The third option is not to include subtype universes as objects in the traditional universe hierarchy. This is equivalent to removing the rules (U-PRED1) and (U-PRED2). The resulting system is still admissible in UTT, and therefore retains the desired meta-theoretic properties. However this weakens the traditional universes, because there are now types that they cannot capture (the subtype universes).

Subtype universes can be formed from any type in our system, with the except of proof types of propositions, i.e. types of the form $\mathbf{Prf}(P)$. We have made this decision to simplify the typing rules, but there is also an intuitive argument that subtyping between proof types is not desirable. It is important to note that whilst we cannot construct subtype universes of proof types, these types can still have names in the subtype universes of other types. Subtype universes of proof types can be supported by a small modification to the system: removing the first clause from Definition 2. This has the auxiliary effect of making the system independent of *Prop* entirely. We have not fully explored the relationship between subtype universes and the impredicative universe *Prop* (and its related types); further work is needed in this area.

Subtyping as a relation has the property of transitivity:

$$\frac{\Gamma \vdash T \leq_c A : \mathbf{Type} \qquad \Gamma \vdash A \leq_{c'} B : \mathbf{Type}}{\Gamma \vdash T \leq_{c' \circ c} C : \mathbf{Type}}$$

By analogy with set theory, we might expect there to be a corresponding subtyping relation between the subtype universes of A and B. The (informal) reasoning for this is as follows: every type with a name in $U_i(A)$ is a subtype of A, and therefore by transitivity it is also a subtype of B, and must have a name in $U_j(B)$. To formalise this argument we must find a coercion between names of T in $U_i(A)$ and names of T in $U_j(B)$. This is not generally derivable in our system as-is, but we can support it by the addition of the following rule:

U-EQUIV

$$\frac{\Gamma \vdash A \leq_c B : \mathbf{Type} \qquad \Gamma \vdash \mathbf{T}^A_{U_i}(t) = T : \mathbf{Type} \qquad \Gamma \vdash \mathbf{T}^B_{U_j}(t') = T : \mathbf{Type}}{\Gamma \vdash \mathbf{t}^{A,B}_{i,j}(t) = t' : U_j(B)} \; (\mathcal{L}_\Gamma(A) \leq \mathcal{L}_\Gamma(B))$$

Here we introduce a lifting operator $\mathbf{t}^{A,B}_{i,j}$ which takes a name in the universe $U_i(A)$ to a name of the same type in $U_j(B)$. This operator can act as a coercion between universes, allowing us to state the following subtype relation:

$$\Gamma \vdash U_i(A) \leq_{\mathbf{t}^{A,B}_{i,j}} U_j(B)$$

By (U-PRED2) we have $\mathcal{L}_\Gamma(U_j(B)) = j + 1$, and therefore we can derive

$$\Gamma \vdash U_{j+1}(U_j(B)) : \mathbf{Type}$$
$$\Gamma \vdash \mathbf{T}^B_{U_j}(n(U_i(A))) = U_i(A) : \mathbf{Type}$$

6 Conclusion

Subtype universes are a novel and useful construct, providing amongst other things a decidable alternative to bounded quantification. We have seen that a coercive subtyping system can be extended to support subtype universes with the addition of six typing rules. Our implementation builds on UTT[$\mathbb{C}$], an existing system supporting coercive subtyping. We prove logical consistency, strong normalisation and subject reduction for our system.

We have extended UTT[$\mathbb{C}$] with subtype universes in a way that preserves its nice metatheoretic properties. In this process we rely on the existence of the predicative universes $Type_i$ in order to syntactically convert subtype universes into predicative universes. However it is important to note that this is done only to make the metatheoretic proofs straightforward. An early abstract of this paper [16] specified a simpler system where subtype universes were not annotated with a type level. This formulation was entirely independent of UTT's predicative universes, but proving the admissibility of the typing rules was difficult. As a result, it is possible that subtype universes can be formulated without predicative universes; this would be an interesting subject for further work.

Although we believe that the system UTT[$\mathbb{C}$]$_U$ has the Church-Rosser property, we have not succeeded in proving it. We also leave unanswered the question of exactly why a system with subtype universes enjoys decidable typechecking whilst traditional bounded quantification does not. Notably absent in our system is the maximal type Top of which every other type is a subtype. Indeed it is not clear how one could introduce such a type in a system with a predicative universe hierarchy, for the same reason that we cannot introduce a type of all types. This distinction is worthy of further investigation.

Related Work. Subtype universes bear similarities to Cardelli's power type [3] $Power(A)$, a type containing all subtypes of A. Power types are constructed in the context of a system of structural subtyping, where subtyping relations are determined by the structure of types rather than by arbitrary axioms. This is motivated by the desire for types to be "self-describing", simplifying typechecking and enabling features like the type-safe (de)serialisation. There is no distinct subtyping relation; the judgement $\vdash A : Power(B)$ reads "A is a subtype of B" and is abbreviated $\vdash A \leq B$. A notable typing rule is that of Power Subtyping, written:

$$\frac{\Gamma \vdash B \leq A}{\Gamma \vdash Power(B) \leq Power(A)}$$

This rule states that if B is a subtype of A then the power type of B is a subtype of the power type of A. The conclusion can also be written $\Gamma \vdash Power(B) : Power(Power(A))$. Written this way, it is clear that this has the same effect as the rule (U-EQUIV) described in Section 5.

Another important aspect of Cardelli's system is that it includes the axiom $Type : Type$, well known to be logically inconsistent. A full discussion of the problems with this property is beyond the scope of this paper; we merely note that, as an extension of UTT, our system does not share this property.

Aspinall's λ_{Power} [1] is a predicative and simplified alternative to Cardelli's system, but it has been difficult to prove some of its metatheoretic properties (such as subject reduction).

References

1 David Aspinall. Subtyping with power types. In *Computer Science Logic, 14th Annual Conference of the EACSL, Fischbachau, Germany, August 21-26, 2000 Proceedings*, volume 1862 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2000. doi:10.1007/3-540-44622-2_10.

2 David Aspinall and Adriana B. Compagnoni. Subtyping dependent types (summary). In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 86–97. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561307.

3 Luca Cardelli. Structural subtyping and the notion of power type. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, pages 70–79. ACM Press, 1988. doi:10.1145/73560.73566.

4 Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. *Inf. Comput.*, 109(1-2):4–56, 1994. doi:10.1006/inco.1994.1013.

5 Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, 17(4):471–522, 1985. doi:10.1145/6041.6042.

6 Giuseppe Castagna and Benjamin C. Pierce. Decidable bounded quantification. In *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 151–162. ACM Press, 1994. doi:10.1145/174675.177844.

7 Stergios Chatzikyriakidis and Zhaohui Luo. On the interpretation of common nouns: Types versus predicates. In *Modern perspectives in type-theoretical semantics*, pages 43–70. Springer, 2017.

8 Thierry Coquand. An analysis of girard's paradox. In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 227–236. IEEE Computer Society, 1986.

9 Jean-Yves Girard. *Interprétation functionelle et Élimination des coupures dans l'arithmétique d'order supérieure.* PhD thesis, Université Paris VII, 1972.

10 Healfdene Goguen. The metatheory of UTT. In *Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6-10, 1994, Selected Papers*, volume 996 of *Lecture Notes in Computer Science*, pages 60–82. Springer, 1994. doi:10.1007/3-540-60579-7_4.

11 Zhaohui Luo. *Computation and reasoning - a type theory for computer science*, volume 11 of *International series of monographs on computer science.* Oxford University Press, 1994.

12 Zhaohui Luo. Coercive subtyping. *J. Log. Comput.*, 9(1):105–130, 1999. doi:10.1093/logcom/9.1.105.

13 Zhaohui Luo. Dependent record types revisited. In *Proceedings of the 1st Workshop on Modules and Libraries for Proof Assistants*, MLPA '09, page 30–37, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1735813.1735819.

14 Zhaohui Luo. Formal semantics in modern type theories with coercive subtyping. *Linguistics and Philosophy*, 35(6):491–513, 2012.

15 Zhaohui Luo, Sergei Soloviev, and Tao Xue. Coercive subtyping: Theory and implementation. *Inf. Comput.*, 223:18–42, 2013. doi:10.1016/j.ic.2012.10.020.

16 Harry Maclean and Zhaohui Luo. Subtype universes (extended abstract). *26th International Conference on Types for Proofs and Programs (TYPES20)*, 2020.

17 Per Martin-Löf. *Intuitionistic type theory.* Bibliopolis, 1984.

18 Richard Montague. *Formal Philosophy: Selected Papers of Richard Montague.* Springer, 1974.

19 Bengt Nordström, Kent Petersson, and Jan M Smith. *Programming in Martin-Löf's type theory. An introduction.* Oxford University Press, 1990.

20 Benjamin C. Pierce. Bounded quantification is undecidable. *Inf. Comput.*, 112(1):131–165, 1994. doi:10.1006/inco.1994.1055.

21 Aarne Ranta. *Type-Theoretical Grammar.* Oxford University Press, 1994.

22 John C. Reynolds. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974. doi:10.1007/3-540-06859-7_148.

23 Sergei G. Vorobyov. Structural decidable extensions of bounded quantification. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, page 164–175, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/199448.199479.

A Proof of Lemma 7

▶ **Lemma** (δ preserves one-step reduction)**.** For every term M in UTT$[\mathbb{C}]_U$, if $M \rightsquigarrow N$ then in UTT$[\mathbb{C}]$ we have $\delta(M) \rightsquigarrow \delta(N)$.

Proof. By induction on the terms of UTT$[\mathbb{C}]_U$. For each reduction step $M \rightsquigarrow^R N$ in UTT$[\mathbb{C}]_U$ via a computation rule R we will show that there is a reduction $\delta(M) \rightsquigarrow^S \delta(N)$ in UTT$[\mathbb{C}]$ via a (possibly identical) rule S. In the special case where $\delta(M) = M$, we will show that $\delta(N) = N$. There are eight reduction (or computation) rules in UTT [10]:

1. $([x : K]k')k \rightsquigarrow^{\beta} [k/x]k'$
2. $E_{\forall}(A, P, R, f, \Lambda(A, P, g)) \rightsquigarrow^{E_{\forall}} f(g)$
3. $\mathbf{T}_{i+1}(type_i) \rightsquigarrow^{type_i} Type_i$
4. $\mathbf{T}_0(prop) \rightsquigarrow^{prop} Prop$
5. $\mathbf{T}_{i+1}(\mathbf{t}_{i+1}(a)) \rightsquigarrow^{\mathbf{t}_{i+1}} \mathbf{T}_i(a)$
6. $\mathbf{T}_0(\mathbf{t}_0(P)) \rightsquigarrow^{prf} \mathbf{Prf}(P)$
7. the computation rule for inductive types $\mathbf{E}[\bar{\Theta}]$
8. $\mathbf{T_i}(\mu_i[\bar{\Theta}]) \rightsquigarrow^{\mu} \mathcal{M}[\bar{\Theta}]$

Our extension adds two more:

1. $\mathbf{T}^A_{U_i}(n(B)) \rightsquigarrow^{urefl} B$
2. $\mathbf{T}_{i+1}(u_i(A)) \rightsquigarrow^{upred} U_i(A)$

We will consider the last two rules in detail. The others follow straightforwardly from the definition of δ. By induction we can therefore extend the result to all expressions in UTT$[\mathbb{C}]_U$.

▶ **Case 1** ($M \rightsquigarrow^{urefl} N$)**.** This rule eliminates $n(B)$, the name for a type B in the subtype universe $U_i(A)$.

$$\begin{aligned} \mathbf{T}^A_{U_i}(n(B)) &\rightsquigarrow^{urefl} B \\ \delta(\mathbf{T}^A_{U_i}(n(B))) &= \delta(\mathbf{T}^A_{U_i})(\delta((n(B)))) \\ &= \mathbf{T}_i(n(\delta(B))) \\ &\rightsquigarrow^{X} \delta(B) \end{aligned}$$

where X stands for the relevant reduction rule reflecting a name in a traditional universe to its type. For example, if B is an inductive type then X stands for the rule $\rightsquigarrow^{\mu}$.

▶ **Case 2** ($M \rightsquigarrow^{upred} N$)**.** This rule eliminates $u_i(A)$, the name for the subtype universe $U_i(A)$.

$$\begin{aligned} \mathbf{T}_{i+1}(u_i(A)) &\rightsquigarrow^{upred} U_i(A) \\ \delta(\mathbf{T}_{i+1}(u_i(A))) &= \delta(\mathbf{T}_{i+1})(\delta((u_i(A)))) \\ &= \mathbf{T}_{i+1}(type_i) \\ &\rightsquigarrow^{type_i} Type_i \\ &= \delta(U_i(A)) \end{aligned}$$

◀