A Metatheoretic Analysis of Subtype Universes

Felix Bradley
Royal Holloway, University of London, UK

Zhaohui Luo
Royal Holloway, University of London, UK

Abstract

Subtype universes were initially introduced as an expressive mechanisation of bounded quantification extending a modern type theory. In this paper, we consider a dependent type theory equipped with coercive subtyping and a generalisation of subtype universes. We prove results regarding the metatheoretic properties of subtype universes, such as consistency and strong normalisation. We analyse the causes of undecidability in bounded quantification, and discuss how coherency impacts the metatheoretic properties of theories implementing bounded quantification. We describe the effects of certain choices of subtyping inference rules on the expressiveness of a type theory, and examine various applications in natural language semantics, programming languages, and mathematics formalisation.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Type theory, coercive subtyping, subtype universes

Digital Object Identifier 10.4230/LIPIcs.TYPES.2022.9

1 Introduction

Power types were initially introduced by Cardelli as a way of integrating subtyping into a type theory to model bounded quantification [2]. Power(A) represents the collection of subtypes of A, and a given subtyping relation $A \leq B$ can be considered as shorthand for $A : \text{Power}(B)$. Cardelli's system was designed with language design in mind, focusing on behavioural subtyping defined by shared properties of objects. In particular, Cardelli's power types could be used to model a notion of parametric polymorphism called bounded quantification, where one can quantify over the subtypes of a given type. By writing $\lambda(X \leq A).M$ as shorthand for $\lambda(X : \text{Power}(A)).M$.

Cardelli's initial system for power types prioritised expressivity over well-behaved metatheory and included a **Type** : **Type** judgement, which was chosen to express non-terminating computations. Power types have since been revisited by other authors such as Aspinall, who reformulated power types into a predicative system [1]. However, these system have often had issues within the metatheory closely linked to subtyping and bounded quantification. The particular choice of subtyping rules is a common issue, where certain combinations of rules can cause undecidability in the subtyping relation [17, 4].

Maclean and Luo later introduced subtype universes, [16] an analogue of power types designed specifically for extending UTT equipped with coercive subtyping [15, 13]. They showed that this extension preserved metatheoretic properties such as logical consistency and strong normalisation. As subtype universes were initially formulated as an extension of UTT, they are built to work in conjunction with the particular structure of UTT's type universes in mind, which makes for complex proofs. UTT is also restricted in the kind of subtyping rules the system can use, in that subtypes must be present in the same type universe as supertypes, which prevents the use of otherwise useful subtyping rules.

In this paper, we generalise Maclean and Luo's results by formulating rules for a more expressive notion of subtype universes, designed to extend a more basic dependent type theory. We continue to use coercive subtyping, a method of subtyping best suited for preserving canonicity of terms. Our subtype universes are described by the pseudo-rules

$$\frac{\Gamma \vdash A\,\mathrm{type}}{\Gamma \vdash \mathcal{U}(A)\,\mathrm{type}} \qquad \frac{\Gamma \vdash A \leq_c B}{\Gamma \vdash (A, c) : \mathcal{U}(B)}$$

combined with operators which allow us to retrieve the first object and the second object of the pair. The addition of being able to retrieve the coercion from a subtyping relation allows for subtyping relations using subtypes or bounded quantification. In particular, if the type theory being extended lacks traditional type universes, being able to retrieve coercions allows the type theory to express more complex subtyping relations.

Section 2 describes the implementation of coercive subtyping, outlines the rules used to formulate our generalised notion of subtype universes, discusses what it means for a type theory to lack type universes and why this matters. Section 3 discusses the metatheoretic properties of subtype universes, dependent on the choice of subtyping judgements and rules the underlying type theory is equipped with. In particular, this section analyses an important property of a subset of subtyping judgements: wherein the choice of subtyping relations allows one to "reflect" subtype universes on to the more traditional type universes; and the other case where this is not possible. Section 4 looks at particular choices of subtyping rules and the implications that this work has for the use and application of them. In particular, it focuses on the use of subtyping rules regarding dependent function, universal supertypes, and subtype universes. Finally, section 5 discusses various applications of subtype universes, and showcases several examples of how subtype universes may be used in programming, natural language semantics, and the formalisation of mathematics.

2 Expressive Subtype Universes

In order to be able to introduce subtype universes, we first need to discuss the notion of subtyping and analyse the particular design choice to use coercive subtyping over subsumptive subtyping. From there, we briefly cover Cardelli's power types – designed with programming languages in mind – and Maclean and Luo's prior work on subtype universes – designed for dependent type theories with logic and proofs in mind – and some of the advantages and restrictions of these approaches, before moving on to introducing subtype universes.

2.1 Coercive Subtyping

Introducing subtyping is a very natural extension of any type system, especially when working from a set-theoretic notion or understanding. Subtyping intuitively corresponds to the subset relation, and many properties of subtyping extend from this intuition; for example, we should be able to process any natural number as a rational number, or be able to say that the rational numbers *include* the natural numbers.

When it comes to attempts to implement subtyping, most approaches introduce some form of a new judgement $\Gamma \vdash A \leq B$, read as "the type A is a subtype of the type B", from which we can derive that any term of type A is also a term of type B. This notion as-is without alteration is subsumptive subtyping – any supertype subsumes its subtypes.

This approach runs into issues quickly, however. Subsumptive subtyping as presented breaks the canonicity of a type system: we expect that any object of an inductive type to be computationally equivalent to some canonical object described by the type's rules. With

subsumptive subtyping, one can no longer comprehend objects given the computation and elimination rules for the object's type, as that object may actually be of a subtype. As all natural numbers are also rational number, but we can no longer use the rules of rational numbers to process the rational numbers.

One proposed solution to this issue is coercive subtyping [13]. The core concept behind coercive subtyping is that subtyping describes implied coercions that allow us to interpret objects of a subtype as a given canonical form in the supertype. These coercions are functions described by the underlying type theory, allowing us to preserve a lot of the underlying metatheory of the type system. Using the same example as discussed for subsumptive subtyping, we can interpret a natural number as a ration number through the explicit coercion which sends $n \mapsto n/1$.

We can reduce our system with subtyping to a system without subtyping simply by inserting coercions where necessary, and so adding coercive subtyping to a theory tends to be a conservative extension. Of particular use to us is UTT, a modern type theory written in Martin-Löf's Logical Framework, where extending the system with coercive subtyping has been proven to be conservative [15].

▶ Remark 1. We use $\tau[\mathcal{C}]$ to denote both a type theory τ implementing coercive subtyping extended by some set of subtyping judgements (arbitrary or dependent on some other choice), as well as the type theory τ implementing coercive subtyping extended by the specific collection of subtyping judgements $\mathcal{C}$. For example, we later describe a syntactic transformation from τ to UTT$[\mathcal{C}]$: as these systems can't use the same set of subtyping judgements, it can be inferred that the $\mathcal{C}$ in UTT$[\mathcal{C}]$ is dependent on the choice of $\mathcal{C}$ in τ.

The key rules for coercive subtyping are as follows:

$$\frac{\Gamma \vdash f : \Pi(x : B).C \qquad \Gamma \vdash A <_c B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : [c(a)/x]C} \qquad \frac{\Gamma \vdash f : \Pi(x : A).C \qquad \Gamma \vdash A <_c B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) = f(c(a)) : [c(a)/x]C}$$

These rules (Sub-Intro and Sub-Comp respectively) are used in conjunction with other rules, such as those for congruence, transitivity, and others[1].

Of particular note is the computation rule – in Luo, Soloviev and Xue's own analysis of the metatheory and implementation of coercive subtyping, this is not formally a reduction rule added to the system [15]. Instead, the type theory they describe has a two-step reduction process: the first step is c-reduction, or the insertion of coercions. The second step is the more typical reduction process involving the application of β-reduction. This process is a necessary step due to the need to correctly mark the places in terms where coercions need to be inserted in order for a term to be well-typed before standard reduction can occur.

In this work, we opt to use the same notion of reduction for the type theory we describe – this allows us to treat c-reduction as part of the normal reduction process. In particular, to avoid the complicated metatheory that Luo et al. worked through, we first show that our type theory can be transformed into UTT$[\mathcal{C}]$ for particular choices of subtyping judgements. As UTT$[\mathcal{C}]$ handles the actual two-step reduction process, this allows us to informally treat c-reduction as on the same level as β-reduction.

When one implements subtyping, one also needs to decide which types are subtypes of which types. This could be both single cases, or families of subtypes (for example, one may wish to say that all finite types are subtypes of $\mathbb{N}$). We consider the most general case possible where the type theory τ is extended by a set of subtyping rules $\mathcal{C}$.

[1] The full set of rules for the implementation of coercive subtyping can be found in [15]. Whilst we do not use the same judgements in this work, the rules are fundamentally the same.

To ensure that τ is sound for a given choice of $\mathcal{C}$, we need a notion of coherence – "that every possible derivation of a statement $\Gamma \vdash a : A$ has the same meaning" [17].We use a similar definition of coherence as Luo et al. as follows:

▶ **Definition 2** (Coherence). *A set of subtyping judgements and inference rules $\mathcal{C}$ is coherent if the following hold:*

- *If* $\Gamma \vdash A <_c B$, *then* $\Gamma \vdash A\,\mathrm{type}$, $\Gamma \vdash B\,\mathrm{type}$, *and* $\Gamma \vdash c : A \to B$
- $\Gamma \not\vdash A <_c A$ *for every* Γ, A, *and* c
- *If* $\Gamma \vdash A <_c B$ *and* $\Gamma \vdash A <_{c'} B$, *then* $\Gamma \vdash c = c'$

The coherency of the subtyping judgements and rules used coercive subtyping is critical – without the guarantee of coherency, τ loses any hope of consistency. In practice, reasoning about coherency can be tedious at best. However, other authors have found difficulty within the metatheory of bounded quantification when using subsumptive subtyping [1, 11]. Even for simpler systems, prior authors have provided proofs which were later found to contain errors [17]. Only as recently as 2004 did Compagnoni provide the first proof of the decidability of subtyping for a higher order lambda calculus [6].

2.2 Subtype Universes

Cardelli initially introduced power types as a means of explicitly mechanising bounded quantification – the type $\mathrm{Power}(A)$ as the type of subtypes of A [2]. In his original formulation, the judgement $A \leq B$ was in shorthand for $A : \mathrm{Power}(B)$, thus typing had completely subsumed subtyping in his system.

Cardelli describes a very expressive type system made possible by these power types, but made compromises in the underlying metatheory of the system in favour of expressiveness. For example, Cardelli's system had a type of all types – while quantification over types is useful, this simple statement can be used to express non-terminating computations, but is also the source of Girard's paradox, which causes logical inconsistency [9, 7, 10]

Maclean and Luo later introduced subtype universes as an extension of UTT[$\mathcal{C}$], UTT equipped with coercive subtyping and a set of subtyping judgements $\mathcal{C}$ [16]. In this implementation, subtyping wasn't completely subsumed by subtype universes; the notions of subtyping and typing were kept disjoint, and subtype universes presented a way for typing to interface with subtyping.

There were some restrictions with Maclean and Luo's presentation, however; they associated subtype universes with the underlying predicative type universes that allowed one to internally quantify over types. This required annotating subtype universes to ensure that types had names in the correct universes, and it also restricted the choice of subtyping relations that could be introduced into the system. In particular, their proof required that for every $A \leq_c B$, A inhabited the same type universe as B.

One of the ways we sought to improve on this design was to expand upon it and remove these restrictions. We use the following rules[2]:

$\mathcal{U}$-Form
$$\frac{\Gamma \vdash B\,\mathrm{type}}{\Gamma \vdash \mathcal{U}(B)\,\mathrm{type}}$$

[2] The rules described here only cover types, and do not touch on kinds or subkinding – for the purposes of this work, the rules covering types are sufficient. Whenever we use describe a relation $A \leq_c B$, it is always the case that A and B are types.

$\mathcal{U}$-Intro $$\frac{\Gamma \vdash A \leq_c B}{\Gamma \vdash \langle A, c\rangle : \mathcal{U}(B)}$$

$\mathcal{U}$-σ_1-Elim $$\frac{\Gamma \vdash B\,\mathrm{type} \quad \Gamma \vdash t : \mathcal{U}(B)}{\Gamma \vdash \sigma_1(t)\,\mathrm{type}}$$

$\mathcal{U}$-σ_2-Elim $$\frac{\Gamma \vdash B\,\mathrm{type} \quad \Gamma \vdash t : \mathcal{U}(B)}{\Gamma \vdash \sigma_2(t) : \sigma_1(t) \to B}$$

$\mathcal{U}$-σ_1-Comp $$\frac{\Gamma \vdash B\,\mathrm{type} \quad \Gamma \vdash \langle A, c\rangle : \mathcal{U}(B)}{\Gamma \vdash \sigma_1(\langle A, c\rangle) = B}$$

$\mathcal{U}$-σ_2-Comp $$\frac{\Gamma \vdash B\,\mathrm{type} \quad \Gamma \vdash \langle A, c\rangle : \mathcal{U}(B)}{\Gamma \vdash \sigma_2(\langle A, c\rangle) = c : A \to B}$$

For a given type A, $\mathcal{U}(A)$ is the type of subtypes of A (intuitively, this corresponds to the power set operator). Terms of a subtype universe behave in a similar fashion to pairs, from which we can obtain both the subtype (via the operator σ_1) and the coercion through which we may obtain the corresponding object of the supertype (via the operator σ_2). This design more closely resembles Cardelli's original intent where subtyping is subsumed by typing, as we can now describe any subtyping relation by declaring an object of a subtype universe.

▶ Remark 3. For coherent $\mathcal{C}$, the type of a given $\langle A, c\rangle$ can be calculated by type-checking the term $\sigma_2(\langle A, c\rangle) = c$. One could extend the subtype universes we use here to also explicitly carry information about the supertype, either as part of their data or via annotations. However, assuming that $\mathcal{C}$ is coherent, one is also able to derive the type of any given $\langle A, c\rangle$ by examining the codomain of c. For simplicity, this work does not include these annotations as the metatheory does not fundamentally change.

2.3 Flat Type Theories

This notion that subtyping implies a partial ordering on types in a system is a property we call monotonicity. Under any set-theoretic notion, this seems obvious; the partial ordering would be inclusion. However, if the system has multiple type universes, then monotonicity presents quite a restriction on the choice of subtyping relations one can introduce; there's some natural subtyping relations we may want to use in a system. Consider the example of the type of *pointed subtypes*:

$$\Sigma(x : \mathcal{U}(B)).\sigma_1(x)$$

Intuitively, a pointed subtype of B should also a subtype of B, and so we may want to use the subtyping relation

$$\Sigma(x : \mathcal{U}(B)).\sigma_1(x) \leq_q B$$

where $q \stackrel{\text{def}}{=} \lambda(y : \Sigma(x : \mathcal{U}(B)).\sigma_1(x)).(\sigma_2(\pi_1(y)))(\pi_2(y))$. However, if we were to follow Maclean and Luo's method for translating $\mathcal{U}(A)$ into an object of UTT$[\mathcal{C}]$, we would quickly find that cannot; their method for mapping subtype universes on to type universes simply does not work here. There is a sense in which the left-hand-side of the subtyping relation is more complicated or of a higher order than the right-hand-side – that the LHS should inhabit a higher type universe than the RHS – due to the presence of $\mathcal{U}(B)$.

This leads to one of the key motivations for this work: when can subtype universes be mapped onto type universes? Are there particular choices of subtyping judgements and rules such that the resultant system can't be described by a system with the standard hierarchy of type universes $\text{Type}_1, \text{Type}_2, ...$? Does the choice of subtyping relations affect the metatheory of the system, and if it does, when and how?

In order to better understand how a choice of subtyping relations affects the system, we need to look at a system with coercive subtyping and subtype universes but with minimal structure on its type universes. As we also want to look at the logical consistency of type theories implementing subtype universes, we allow for an impredicative type of propositions Prop.

Whilst the proofs we describe in this work theoretically apply to any "flat" type theory that has no type universes or at most a universe of propositions, we opt to use a subtheory of UTT[$\mathcal{C}$] to make several of the proofs in this work more convenient[3] – for example, we use the fact that UTT[$\mathcal{C}$] is logically consistent and strongly normalising. In particular, this subtheory of UTT[$\mathcal{C}$] contains dependent function types, dependent pair types, an impredicative type universe of propositions, and the atomic types $\mathbf{0}$, $\mathbf{1}$, and $\mathbb{N}$[4].

We write τ to denote the chosen subtheory of UTT[$\mathcal{C}$], extended with subtype universes and a (sometimes arbitrary or variable) set of subtyping judgements $\mathcal{C}$. When it is not necessarily clear what specific set of subtyping judgements τ is equipped with, we write $\tau[\mathcal{C}]$ to denote τ equipped with the specific set of subtyping judgements $\mathcal{C}$. We also write $\tau[\mathcal{C}; \text{R}]$ to denote the theory $\tau[\mathcal{C}]$ which has been extended by a specific subtyping judgement or rule R.

3 Metatheory

In our analysis of τ, we will first examine the metatheory of τ equipped with a set of only monotonic subtyping relations. In particular, we will describe an embedding of $\tau[\mathcal{C}]$ in UTT[$\mathcal{C}$]. In order to describe this embedding, we first need to develop a notion of the level of a type – a measure of it's complexity or "order" under the Curry-Howard interpretation of types as propositions. Afterwards, we will examine the metatheory of the general case where τ is equipped with a set of subtyping relations wherein some are non-monotonic. In both cases, we will prove logical consistency and strong normalisation of τ.

3.1 Type Level

To properly analyse the metatheory of subtype universes, we need to understand under what conditions can subtype universes be reflected on to type universes. In order to do this, we need a notion of the "level" of a type; a description of where a given type fits in the type universe hierarchy. If this notion is well-formed, we will be able to transform terms of a type theory with "well-behaved" subtyping judgements into a type theory where the metatheoretic properties we care about have already been proven.

---

[3] We use a different set of subtyping judgements to UTT[$\mathcal{C}$], such as using $\vdash A$ type rather than $\vdash A : \mathbf{Type}$, but this is primarily for brevity when writing judgements

[4] The atomic types, type constructors, and type of Propositions are all defined using UTT's inductive schemata [12]. While τ can easily be expanded to include inductive data types and inductive propositions, we have elected to skip these inclusions for simplicity and brevity of argument.

In particular, we use UTT[$\mathcal{C}$], Luo's *Unifying Theory of dependent Types* extended with coercive subtyping, as our target theory for this syntactic transformation. This is due to many of the metatheoretic properties we are interested in having been proven for this theory[15, 8]. As such, our own notion of type level is similar in practice to that which Luo uses, but a different approach is necessary; Luo's approach uses type isomorphism and type universes to define level, and we do not have the luxury of the latter [12].

Instead, our notion of type level is defined recursively; basic types (e.g. propositions, Prop, $\mathbb{N} \to \mathbb{N}$, etc.) should be of type level 0, and subtype universes should correspond to increasing the type level by 1.

▶ **Definition 4.** *For a given type A within a context Γ considered in τ, we define its* type level $\mathcal{L}_\Gamma(A)$ *by recursion as follows:*

- *If $\Gamma \vdash A = P : \mathrm{Prop}$, then $\mathcal{L}_\Gamma(A) = 0$;*
- *If $\Gamma \vdash A = \mathrm{Prop}$, $\boldsymbol{0}$, $\boldsymbol{1}$, or $\mathbb{N}$, then $\mathcal{L}_\Gamma(A) = 0$;*
- *If $\exists B, C$ such that $\Gamma \vdash A = \Pi(x : B).C$, then $\mathcal{L}_\Gamma(A) = \max_{x:B}\{\mathcal{L}_\Gamma(B), \mathcal{L}_\Gamma(C[x])\}$;*
- *If $\exists B, C$ such that $\Gamma \vdash A = \Sigma(x : B).C$, then $\mathcal{L}_\Gamma(A) = \max_{x:B}\{\mathcal{L}_\Gamma(B), \mathcal{L}_\Gamma(C[x])\}$;*
- *If $\exists B$ such that $\Gamma \vdash A = \mathcal{U}(B)$, then $\mathcal{L}_\Gamma(A) = \mathcal{L}_\Gamma(B) + 1$.*
- *If $\exists B, c, s$ such that $\Gamma \vdash A = \sigma_1(s)$ and $\Gamma \vdash s = \langle B, c\rangle$, then $\mathcal{L}_\Gamma(A) = \mathcal{L}_\Gamma(B)$;*
- *If $\exists N$ such that $\Gamma, s : \mathcal{U}(N) \vdash A = \sigma_1(s)$ then $\mathcal{L}_\Gamma(A) = \max_{M \leq N}\{\mathcal{L}_\Gamma(M)\}$;*
- *Otherwise, $\mathcal{L}_\Gamma(A)$ is undefined.*

We need to ensure that our notion of type level is well-formed, i.e. every type has a type level, and only types have a type level.

▶ **Lemma 5.** *If $\Gamma \vdash A$ type, then precisely one of the following hold:*

- $\Gamma \vdash A = P : \mathrm{Prop}$
- $\Gamma \vdash A = \mathrm{Prop}$
- $\Gamma \vdash A = \boldsymbol{0}$
- $\Gamma \vdash A = \boldsymbol{1}$
- $\Gamma \vdash A = \mathbb{N}$
- $\exists B, C$ *such that* $\Gamma \vdash A = \Pi(x : B).C$
- $\exists B, C$ *such that* $\Gamma \vdash A = \Sigma(x : B).C$
- $\Gamma \vdash A = \mathcal{U}(B)$
- $\exists s$ *such that* $\Gamma \vdash A = \sigma_1(s)$

Proof. By induction on derivations of the form $\Gamma \vdash A$ type. ◀

▶ **Corollary 6.** *$\mathcal{L}_\Gamma(A)$ is defined if and only if $\Gamma \vdash A$ type.*

One of the key metatheoretic features we are interested in is strong normalisation, and so it is also important to check that our notion of type level is invariant under reduction. As discussed in section 2.1, we inherit the same two-step reduction process as that described in Luo, Soloviev and Xue's work on the implementation of coercive subtyping [15]. The first step is c-reduction, wherein coercions are inserted where appropriate to ensure that terms are well-formed and well-typed. The second step is the more typical reduction process via the application of β-reduction. As our goal in this section is to describe an embedding of τ into UTT[$\mathcal{C}$], we can informally treat this reduction process as if it were a single step, putting c-reduction at the same level as β-reduction.

▶ **Definition 7.** *Let $M \rhd N$ denote that applying a single step of reduction to M yields N. Likewise, let $\rhd^*$ denote the reflexive and transitive closure of $\rhd$, i.e. $M \rhd^* N$ denotes that applying 0 or more steps of reduction to M yields N.*

▶ **Theorem 8.** *If $A \rhd B$ then $\mathcal{L}_\Gamma(A) = \mathcal{L}_\Gamma(B)$.*

Proof. To show that this is true in general, we simply need to show that this holds true for reduction via the computation rule $\mathcal{U}$-σ_1-Comp. For any object s of type $\mathcal{U}(B)$, we obtain that it is of the form $\langle B, c\rangle$ for some B and c by induction on derivations. As $\mathcal{L}_\Gamma(\sigma_1(s))$ is defined by its reduction with respect to $\mathcal{U}$-σ_1-Comp, this holds. ◀

▶ **Corollary 9.** *If $A \rhd^* B$ then $\mathcal{L}_\Gamma(A) = \mathcal{L}_\Gamma(B)$.*

▶ Remark 10. Luo's notion of type level had a couple of valuable properties: for example, if two types are type-isomorphic, then they had the same level. For our own notion of type level, this doesn't necessarily hold: if we consider the case of τ with empty $\mathcal{C}$, then every subtype universe is type-isomorphic to the unit type.

There are also other cases which should serve as counterexamples at a glance, but end up being much more interesting under a closer look. Unfortunately, this is outside the scope of this paper.[5]

3.2 A Syntactic Transformation

As previously mentioned, subtype universes can be thought of as a collection of pairs of objects; the first object in the pair is the subtype of the supertype, and the second object of the pair is the coercion through which it is a subtype. We can make this intuition explicit with a syntactic transformation by turning subtype universes in τ into the type of dependent pairs in UTT[$\mathcal{C}$], where the first object is the name of the subtype and the second object is the coercion.

This intuition only works if the subtype *has a name in the corresponding type universe.* This leads us to a formal definition of monotonicity:

▶ **Definition 11.** *A coercive subtyping relation $A \leq_c B$ is* monotonic *if $\mathcal{L}_\Gamma(A) \leq \mathcal{L}_\Gamma(B)$. A set of coercive subtyping judgements and rules $\mathcal{C}$ is* monotonic *if every coercive subtyping relation derived from $\mathcal{C}$ is monotonic.*

However, we've already seen that there are some non-monotonic subtyping judgements which may be desirable, so we will revisit the case of non-monotonic subtyping later; for now, we focus on the case of monotonic subtyping. For monotonic $\mathcal{C}$, we define a syntactic transformation $\delta : \tau \to \text{UTT}[\mathcal{C}]$ by recursion as described in figure 1.

In this section, we describe both judgements derived in τ and judgements derived in UTT[$\mathcal{C}$]. While one can view the underlying type theory T as a subtheory of UTT, we use a different set of judgements. As such, we distinguish between them where necessary; any judgement derived in τ will be denoted with $\vdash_\tau$, and any judgement derived in UTT[$\mathcal{C}$] will be denoted with $\vdash_{\text{UTT}}$. Moreover, any context in τ will be written as Γ, and any context in UTT[$\delta(\mathcal{C})$] will be written as $\delta(\Gamma)$

To ensure this is a useful transformation, we need to check whether or not types have names in the expected type universes; whether we can derive the judgements we expect regarding translated terms; and whether this transformation preserves metatheoretic properties we're interested in, such as logical consistency and strong normalisation.

Prior to this, however, we need to discuss the translation of subtyping and how δ can preserve any notion of subtyping. For some derivation $\Gamma \vdash_\tau A \leq_c B$, we expect to be able to take any $\Gamma \vdash a : A$ and derive that $\Gamma \vdash a : B$; if δ is to preserve subtyping, then we also need to ensure that not only can we derive $\delta(\Gamma) \vdash_{\text{UTT}} \delta(a) : \delta(A)$, but also that $\delta(\Gamma) \vdash_{\text{UTT}} \delta(a) : \delta(B)$.

[5] We encourage the curious reader to think about the following example: for $\Gamma, a : A \vdash P : \text{Prop}$, consider whether or not the types $\Pi(a : A).P$ and $\Pi(X : \mathcal{U}(A)).\Pi(x : \sigma_1(X)).P[c(x)/a]$ are type-isomorphic.

$$\delta(\mathcal{U}(B)) = \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)}).(X \to \delta(B)) \qquad \delta(\sigma_2) = \pi_2$$

$$\delta(\langle A, c\rangle) = (\mathbf{t}^{\mathcal{L}_\Gamma(B)}_{\mathcal{L}_\Gamma(A)} \circ \mathbf{n}(\delta(A)), \delta(c)) \qquad \delta(\sigma_1) = \mathbf{T} \circ \pi_1$$

$$\delta(\Pi_{x:A}B) = \Pi_{x:\delta(A)}\delta(B) \qquad \delta(\lambda(a:A).B) = \lambda(a:\delta(A)).\delta(B)$$

$$\delta(\Sigma_{x:A}B) = \Sigma_{x:\delta(A)}\delta(B) \qquad \delta((a,b)) = (\delta(a), \delta(b))$$

$$\delta(\pi_1) = \pi_1 \qquad \delta(\pi_2) = \pi_2 \qquad \delta(f(x)) = \delta(f)(\delta(x))$$

$$\delta(\mathbf{0}) = \mathbf{0} \qquad \delta(\mathbf{1}) = \mathbf{1} \qquad \delta(*) = *$$

$$\delta(\mathbb{N}) = \mathbb{N} \qquad \delta(0) = 0 \qquad \delta(\mathrm{S}) = \mathrm{S}$$

$$\delta(\mathrm{Prop}) = \mathrm{Prop} \qquad \delta(\forall(x:A).P) = \mathbf{Prf}(\forall(x:\delta(A)).\delta(P))$$

$$\delta(\Lambda(a:A).P) = \Lambda(a:\delta(A)).\delta(P)$$

Figure 1 The transformation of terms in $\tau[\mathcal{C}]$ into terms in $\mathrm{UTT}[\delta(\mathcal{C})]$ under δ.

As we have only defined the domain of our transformation δ as being $\mathrm{UTT}[\mathcal{C}]$ for *some* $\mathcal{C}$, we can exactly specify our target theory by choosing which subtyping judgements and rules it uses, depending on our initial choice of $\mathcal{C}$ for τ. As such, we extend our definition of δ to include the subtyping judgements of τ, sending any $A \leq_c B \in \mathcal{C}$ to $\delta(A) \leq_{\delta(c)} \delta(B)$; we write the collection of the latter as $\delta(\mathcal{C})$. Moreover, we can precisely say that, for $\mathcal{C}$ a fixed set of subtyping judgements and rules, we can define a syntactic transformation $\delta : \tau \to \mathrm{UTT}[\delta(\mathcal{C})]$ per the above.

▶ **Theorem 12.** *If* $\Gamma \vdash A\,\mathrm{type}$*, then* $\delta(\Gamma) \vdash \delta(A) : \mathbf{Type}$ *and there exists some* $i \in \omega$ *and some term* n *in* $\mathrm{UTT}[\delta(\mathcal{C})]$ *such* $\delta(\Gamma) \vdash n : \mathrm{Type}_i$ *and* $\mathbf{T}_i(n) = \delta(A)$*.*

Proof. Proof by induction on derivations of $\Gamma \vdash A\,\mathrm{type}$ and cross-referencing with lemma 5. We consider the following cases:

Case 1. $\exists B, P$ such that A is of the form $\forall(b:B).P$. As in UTT the predicate $\forall(x:A).P$ exists for any type A and any predicate P over A and has a name in Prop, we may take $i = 0$ and n to be the name of $\mathbf{Prf}(\forall(b:\delta(B)).\delta(P))$ in Type_0.

Case 2. A of the form Prop. Trivially, we may take $i = 0$ and $n = \mathit{prop} : \mathrm{Type}_0$.

Case 3. A of the form $\mathbf{0}$, $\mathbf{1}$, or $\mathbb{N}$. We take advantage of UTT's rules which introduce the names of inductive data types to establish that, as all of these constructors do not have any types in their generating sequence of inductive schemata, $i = 0$ and $\delta(A)$ must have names in Type_0 [12].

Case 4. $\exists B, C$ such that A is of the form $\Pi(x:B).C$, or $\Sigma(x:B).C$. Similarly, these types have a name in Type_i if and only if both $\delta(B)$ has a name in Type_j and $\delta(C)$ has a name in Type_k. Assuming $\delta(B)$ has a name in Type_j and $\delta(C)$ has a name in Type_k, we may take $i = \max\{j,k\}$ and thus $\delta(A)$ has a name in Type_i, which is as desired.

Case 5. $\exists B$ such that A is of the form $\mathcal{U}(B)$. Using the same reasoning as per Π types: as $\delta(A) = \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)}).(X \to \delta(B))$, we note that $\mathrm{Type}_{\mathcal{L}_\Gamma(B)}$ has a name in $\mathrm{Type}_{\mathcal{L}_\Gamma(B)+1}$ and that $X \to \delta(B)$ has a name in $\mathrm{Type}_{\mathcal{L}_\Gamma(B)}$, and so we may take n as the name for $\mathrm{Type}_{\mathcal{L}_\Gamma(B)}$.

Case 6. $\exists s$ such that A is of the form $\sigma_1(s)$. By induction, we have some B and c such that $\Gamma \vdash s = \langle B, c\rangle$ and thus $A = B$ by $\mathcal{U}$-σ_1-Comp. We may take $i = \mathcal{L}_\Gamma(A)$ and n to be the name of $\delta(B)$ in $\mathrm{Type}_{\mathcal{L}_\Gamma(A)}$.

Case 7. $\exists B$ such that A is of the form $\sigma_1(s)$, where $s : \mathcal{U}(B)$ is variable. As $\mathcal{C}$ is monotonic, we know that $\mathcal{L}_\Gamma(A)$ is at most $\mathcal{L}_\Gamma(B)$ and thus we may take $i = \mathcal{L}_\Gamma(B)$ and n to be the name of $\delta(A)$ in $\mathrm{Type}_{\mathcal{L}_\Gamma(B)}$. ◀

▶ **Theorem 13.** *For coherent and monotonic $\mathcal{C}$, the rules of $\tau[\mathcal{C}]$ are admissible in* $\mathrm{UTT}[\delta(\mathcal{C})]$ *under transformation by δ.*

Proof. This is a special case of theorem 23 taking $k = 0$. ◀

▶ **Lemma 14.** *Let R be a coherent subtyping judgement or rule. Then $\delta(R)$ is coherent.*

Proof. Assume that $R \vdash A \leq_c B$. By our definition of δ, we immediately have that $\delta(\Gamma) \vdash \delta(c) : \delta(A) \to \delta(B)$. Injectivity of δ implies both: that $\delta(\Gamma) \not\vdash \delta(A) \leq_{\delta(c)} \delta(A)$ for every Γ, A, and c; and that if $\delta(\Gamma) \vdash \delta(A) \leq_{\delta(c)} \delta(B)$ and $\delta(\Gamma) \vdash \delta(A) \leq_{\delta(c')} \delta(B)$, then $\delta(\Gamma) \vdash c = c'$. Thus $\delta(A) \leq_{\delta(c} \delta(B)$ is coherent for every derivation of $A \leq_c B$ from R. ◀

3.3 On Monotonic Subtyping

▶ **Theorem 15** (Logical consistency)**.** *For monotonic $\mathcal{C}$, τ is logically consistent, i.e. there does not exist some Γ and p such that $\Gamma \vdash p : \forall(P : \mathrm{Prop}).P$.*

Proof. Proof by contradiction. Assume that there does exist some Γ and p such that $\Gamma \vdash p : \forall(P : \mathrm{Prop}).P$. Under syntactic transformation by δ, we obtain $\delta(\Gamma) \vdash \delta(p) : \mathbf{Prf}(\forall(P : \mathrm{Prop}).P)$, which contradicts the logical consistency of $\mathrm{UTT}[\mathcal{C}]$. ◀

▶ **Theorem 16** (Preservation of one-step reduction)**.** *For monotonic $\mathcal{C}$, if $M \rhd N$ then $\delta(M) \rhd \delta(N)$.*

Proof. Proof by induction on the terms of τ. For every reduction $M \rhd^R N$ in τ via a computation rule R, we show that there exists a computation rule S such that $\delta(M) \rhd^S \delta(N)$ in $\mathrm{UTT}[\mathcal{C}]$.

As before, there are several trivial cases which have been omitted for brevity, most of which are special cases of the computation rule μ for UTT's inductive types[6]. We focus on the non-trivial cases regarding subtyping and subtype universes.

Case 1. $f(a) \rhd^{\text{Sub-Comp}} f(c(a)) \Rightarrow$
$\delta(f(a)) \overset{\text{def}}{=} \delta(f)(\delta(a)) \rhd^{\mathrm{CA}_2} \delta(f)(\delta(c)(\delta(a))) \overset{\text{def}}{=} \delta(f)(\delta(c(a))) \overset{\text{def}}{=} \delta(f(c(a)))$

Case 2. $\sigma_1(\langle A, c\rangle) \rhd^{\mathcal{U}\text{-}\sigma_1\text{-Comp}} A \Rightarrow$
$\delta(\sigma_1(\langle A, c\rangle)) \overset{\text{def}}{=} \delta(\sigma_1)(\delta(\langle A, c\rangle)) \overset{\text{def}}{=} \mathbf{T}_{\mathcal{L}_\Gamma(B)} \circ \pi_1(\mathbf{t}^{\mathcal{L}_\Gamma(B)}_{\mathcal{L}_\Gamma(A)} \circ \mathbf{n}(\delta(A)), \delta(c))$
$\rhd^{\Sigma_1} \mathbf{T}_{\mathcal{L}_\Gamma(B)}(\mathbf{t}^{\mathcal{L}_\Gamma(B)}_{\mathcal{L}_\Gamma(A)} \circ \mathbf{n}(\delta(A))) \overset{\text{def}}{=} \delta(A)$

Case 3. $\sigma_2(\langle A, c\rangle) \rhd^{\mathcal{U}\text{-}\sigma_2\text{-Comp}} c \Rightarrow$
$\delta(\sigma_2(\langle A, c\rangle)) \overset{\text{def}}{=} \delta(\sigma_2)(\delta(\langle A, c\rangle)) \overset{\text{def}}{=} \pi_2((\mathbf{t}^{\mathcal{L}_\Gamma(B)}_{\mathcal{L}_\Gamma(A)} \circ \mathbf{n}(\delta(A)), \delta(c))) \rhd^{\Sigma_2} \delta(c)$ ◀

▶ **Corollary 17** (Preservation of multi-step reduction)**.** *For monotonic $\mathcal{C}$, if $M \rhd^* N$ then $\delta(M) \rhd^* \delta(N)$.*

▶ **Theorem 18** (Strong normalisation)**.** *For monotonic $\mathcal{C}$, if $\Gamma \vdash M : A$, then M is strongly normalisable, i.e. every possible sequence of reductions of M is finite.*

[6] These include Π types, Σ types, $\mathbb{N}$, $\mathbf{1}$, $\mathbf{0}$.

Proof. Proof by contradiction. Assume that there does exist some Γ and M such that $\Gamma \vdash M : A$ where M has an infinite reduction sequence. Under transformation by δ we obtain $\delta(M)$. As δ preserves multi-step reduction, we obtain an infinite reduction sequence of $\delta(M)$, which contradicts the strong normalisation of UTT$[\mathcal{C}]$. ◀

▶ Remark 19. For monotonic $\mathcal{C}$, we can take advantage of our transformation δ and the decidability of type-checking in UTT$[\mathcal{C}]$ to show that in τ both type checking and the subtyping relation is decidable. For any given term M in τ, we can consider the type of $\delta(M)$. As UTT$[\mathcal{C}]$ is a conservative extension[7] of UTT, we can type-check $\delta(M)$. As δ is injective, we are also able to know the form of the type of $\delta(M)$ in UTT$[\mathcal{C}]$ and thus also in τ.

To show that subtyping is also decidable, for any given pair of types A, B, we can consider the construction of a term t which is well-typed if and only if the subtyping relation $A \leq B$ is derivable (such as $\lambda(f : B \to \mathbb{N}).\lambda(a : A).f(a).$) [17]. By checking if $\delta(t)$ is well-typed in UTT$[\mathcal{C}]$, we can decide whether or not $A \leq B$.

3.4 On Non-Mononotic Subtyping

When analysing the more general case of the metatheory of $\tau[\mathcal{C}]$, where the set of subtyping judgements $\mathcal{C}$ contains some non-monotonic subtyping relations, one will often run into immediate difficulty. Our first approach to this problem was to try and use the notion that the use of coercive subtyping is a kind of shorthand for the insertion of exact coercions. The extension of a type theory with coercive subtyping should be a conservative extension, and likewise extending a type theory with additional subtyping rules should not affect the underlying theory [15, 13].

One could interpret the extension of a type theory with additional coercive subtyping rules as a kind of weakly conservative extension – it should not "add" new types to the theory, and one should not suddenly be able to obtain terms in types that were previously uninhabited. Furthermore, if you have a term that depends on the existence of a subtyping judgement, then it should be possible to construct another term of the same type that *doesn't* rely on that subtyping judgement through the insertion of coercions – this can form the basis of a type-checking algorithm, assuming you have a type-checking algorithm for the case where $\mathcal{C}$ is empty.

When applying this idea to τ, the existence of subtype universes causes immediate problems. If you have a term that depends on an object of a subtype universe, say $\langle A, c\rangle : \mathcal{U}(B)$, then you can attempt to construct another object of the same type by both inserting coercions and by replacing instances of $\langle A, c\rangle$ with $\langle B, \mathrm{id}_B\rangle$. At the term-level, this idea requires effort, but is sound. Unfortunately, the idea does not work in general due to the presence of new types.

Consider a subtyping rule from which we may derive $A \leq_c B$, and extending $\tau[\mathcal{C}]$ with R. Observe that

$$\Sigma(x : \mathcal{U}(B)).\,\mathrm{Eq}_{\mathcal{U}(B)}(x, \langle A, c\rangle)$$

where $\mathrm{Eq}_A \stackrel{\text{def}}{=} \forall(x : A).\forall(y : A).\forall(P : \mathrm{Prop}).(P(x) \leftrightarrow P(y))$ is the type of propositional Leibniz equality on a given type A. Whilst this type can be derived in $\tau[\mathcal{C}; \mathrm{R}]$, it cannot be derived in $\tau[\mathcal{C}]$, and there's no obvious process from which we may try to extend the type-checking algorithm for $\tau[\mathcal{C}]$.

[7] More accurately, UTT$[\mathcal{C}]$ is equivalent to a type theory which is a conservative extension of UTT – the exact meaning of "conservativity" of τ with respect to T is not always clear with the rules for coercive subtyping we have used.

David Aspinall's work on the predicative typed lambda calculus λ_{Power} lead to him introducing a notion of "rough types" [1]. Where as one may intuit our first approach described above as an attempt to blur terms together to extend a type-checking algorithm, Aspinall's approach instead blurs types together, organises them into rough types, and develops a rough-type-checking algorithm. Aspinall shows that, as long as there is a method to calculate the rough type of a given term, this is sufficient to be able to prove strong normalisation for the calculus.

We believe Aspinall's approach would also work for τ. However, it also possible to further generalise the definitions and proofs given in sections 3.2 and 3.3 to cover practically most non-monotonic subtyping relations. The key insight is that most non-monotonic subtyping relations are still relatively well-behaved: by measuring how far a subtyping relation is from being monotonic, we're able to adjust the embedding of τ into $\text{UTT}[\mathcal{C}]$ in response. We first introduce this measurement:

▶ **Definition 20.** *A coercive subtyping relation $A \leq_c B$ is* k-monotonic *if $\mathcal{L}_\Gamma(B) - \mathcal{L}_\Gamma(B) + k \geq 0$. A set of coercive subtyping judgements and rules $\mathcal{C}$ is* k-monotonic *if every coercive subtyping relation derived from $\mathcal{C}$ is k-monotonic.*

This is a generalisation of monotonicity of subtyping – in particular, any given monotonic subtyping relation is 0-monotonic. If we consider the example of pointed subtypes introduced earlier in section 2.3

$$\Sigma(x : \mathcal{U}(B)).\sigma_1(x) \leq_q B,$$

we can calculate that the difference between the level of the supertype and the level of the subtype is independent of the choice of the type B.

$$\begin{aligned} \mathcal{L}_\Gamma(B) - \mathcal{L}_\Gamma(\Sigma(x : \mathcal{U}(B)).\sigma_1(x)) &= \mathcal{L}_\Gamma(B) - \max(\mathcal{L}_\Gamma(B) + 1, \mathcal{L}_\Gamma(\sigma_1(x))) && (1) \\ &= \min(-1, -\mathcal{L}_\Gamma(\sigma_1(x))) && (2) \end{aligned}$$

If you wanted to extend $\tau[]$ with the rule $B\,\text{type} \vdash \Sigma(x : \mathcal{U}(B)).\sigma_1(x) \leq_q B$, then this subtyping rule would be 1-monotonic. At worst, because a type can only be constructed with a finite number of $\mathcal{U}$s, we know that there must always exist some k such that this subtyping rule is k-monotonic.

▶ **Corollary 21.** *Let R be an i-monotonic coherent coercive subtyping rule, and let $\mathcal{C}$ be a j-monotonic set of coherent coercive subtyping judgements. Then $[\mathcal{C}; R]$ is (at worst) $(i + j)$-monotonic.*

From here, one can modify the embedding δ described in section 3.2: if you have a bounded measure k of the difference in level between a supertype and a subtype, then this says that if you were attempting to embed $\tau[\mathcal{C}]$ into $\text{UTT}[\mathcal{C}]$ by mapping $\mathcal{U}(B)$ to $\Sigma(X : \text{Type}_{\mathcal{L}_\Gamma(B)}).(X \to \delta(B))$, then the particular choice of $\text{Type}_{\mathcal{L}_\Gamma(B)}$ is k levels under where it needs to be for $\delta(A)$ to have a name.

▶ **Definition 22.** *For a given set of k-monotonic subtyping judgements $\mathcal{C}$, define a syntactic transformation $\delta_k : \tau[\mathcal{C}] \to \text{UTT}[\delta_k(\mathcal{C})]$ defined identically to δ except*

$$\delta_k(\mathcal{U}(B)) \stackrel{def}{=} \Sigma(X : \text{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B)), \quad \delta_k(\langle A, c\rangle) = (\mathbf{t}^{\mathcal{L}_\Gamma(B)+k}_{\mathcal{L}_\Gamma(A)} \circ \mathbf{n}(\delta_k(A)), \delta_k(c))$$

As there is only a minor difference between δ_k and δ, it's easy to see that a lot of the proofs needed to show that δ_k is a well-behaved embedding that preserves term reduction are almost identical to the proofs for δ, except for the extra terms of k, as seen in the proof of theorem 23. As a result, the proofs of theorem 12 and lemma 14 are functionally identical, as are the proofs regarding logical consistency, term reduction and strong normalisation.

▶ **Theorem 23.** *For coherent and k-monotonic $\mathcal{C}$, the rules of $\tau[\mathcal{C}]$ are admissible in* $\mathrm{UTT}[\delta(\mathcal{C})]$ *under transformation by δ_k.*

Proof. As τ is a subtheory of $\mathrm{UTT}[\mathcal{C}]$ as discussed in section 2.3, the majority of the rules of τ are effectively derivable in $\mathrm{UTT}[\mathcal{C}]$ by default. We omit the trivial cases (such as rules for the unit type, dependent function types, etc.) and instead focus on the non-trivial cases regarding coercive subtyping and subtype universes.

δ_k-Sub-Intro
$$\frac{\delta_k(\Gamma) \vdash \delta_k(f) : \Pi(x : \delta_k(B)).\delta_k(C) \quad \delta_k(\Gamma) \vdash \delta_k(A) <_{\delta_k(c)} \delta_k(B) \quad \delta_k(\Gamma) \vdash \delta_k(a) : \delta_k(A)}{\delta_k(\Gamma) \vdash \delta_k(f)(\delta_k(a)) : [\delta_k(c)(\delta_k(a))/x]\delta(C)} \quad \textit{derivable}$$

δ_k-Sub-Comp
$$\frac{\delta_k(\Gamma) \vdash \delta_k(f) : \Pi_{x:\delta_k(A)}\delta_k(C) \quad \delta_k(\Gamma) \vdash \delta_k(A) <_{\delta_k(c)} \delta_k(B) \quad \delta_k(\Gamma) \vdash \delta_k(a) : \delta_k(A)}{\Gamma \vdash \delta_k(f)(\delta_k(a)) = \delta_k(f)(\delta_k(c(a))) : [\delta_k(c)(\delta_k(a))/x]\delta_k(C)} \quad \textit{derivable}$$

δ_k-$\mathcal{U}$-Form
$$\frac{\delta_k(\Gamma) \vdash \delta_k(B)\,\mathrm{type}}{\delta_k(\Gamma) \vdash \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B)) : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k+1}} \quad \textit{derivable}$$

δ_k-$\mathcal{U}$-Intro
$$\frac{\delta_k(\Gamma) \vdash \delta_k(A) <_{\delta_k(c)} \delta_k(B)}{\delta_k(\Gamma) \vdash \delta_k(\langle A, c\rangle) : \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B))} \quad \textit{derivable}$$

δ_k-$\mathcal{U}$-σ_1-Elim
$$\frac{\delta_k(\Gamma) \vdash \delta_k(B) : \mathrm{Type}_{\mathcal{L}_\Gamma(B)} \quad \Gamma \vdash \delta_k(t) : \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B))}{\Gamma \vdash \mathbf{T}_{\mathcal{L}_\Gamma(B)+k} \circ \pi_1(\delta_k(t)) : \mathbf{Type}} \quad \textit{derivable}$$

δ_k-$\mathcal{U}$-σ_2-Elim
$$\frac{\delta_k(\Gamma) \vdash \delta_k(B) : \mathrm{Type}_{\mathcal{L}_\Gamma(B)} \quad \delta_k(\Gamma) \vdash \delta_k(t) : \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B))}{\delta_k(\Gamma) \vdash \pi_2(\delta_k(t)) : \pi_1(\delta_k(t)) \to \delta_k(B)} \quad \textit{derivable}$$

δ_k-$\mathcal{U}$-σ_1-Comp
$$\frac{\delta_k(\Gamma) \vdash \delta_k(B) : \mathrm{Type}_{\mathcal{L}_\Gamma(B)} \quad \delta_k(\Gamma) \vdash (\delta_k(A), \delta_k(c)) : \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B))}{\delta_k(\Gamma) \vdash \mathbf{T}_{\mathcal{L}_\Gamma(B)+k} \circ \pi_1(\delta_k(\langle A, c\rangle)) = \delta_k(A) : \mathbf{Type}} \quad \textit{derivable}$$

δ_k-$\mathcal{U}$-σ_2-Comp
$$\frac{\delta_k(\Gamma) \vdash \delta_k(B) : \mathrm{Type}_{\mathcal{L}_\Gamma(B)} \quad \delta_k(\Gamma) \vdash \delta_k(A), \delta_k(c)) : \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)+k}).(X \to \delta_k(B))}{\delta_k(\Gamma) \vdash \pi_1(\delta_k(\langle A, c\rangle)) = \delta_k(c) : \delta_k(A) \to \delta_k(B)} \quad \textit{derivable}$$

◀

▶ **Theorem 24** (Logical consistency)**.** *For k-monotonic $\mathcal{C}$, τ is logically consistent, i.e. there does not exist some Γ and p such that $\Gamma \vdash p : \forall(P : \mathrm{Prop}).P$.*

▶ **Theorem 25** (Preservation of one-step reduction)**.** *For k-monotonic $\mathcal{C}$, if $M \rhd N$ then $\delta(M) \rhd \delta(N)$.*

▶ **Corollary 26** (Preservation of multi-step reduction)**.** *For k-monotonic $\mathcal{C}$, if $M \rhd^* N$ then $\delta(M) \rhd^* \delta(N)$.*

▶ **Theorem 27** (Strong normalisation). *For k-monotonic $\mathcal{C}$, if $\Gamma \vdash M : A$, then M is strongly normalisable, i.e. every possible sequence of reductions of M is finite.*

▶ Remark 28. As in the case with monotonic subtyping, as the embedding δ_k is injective, we can type-check any given term M of $\tau[\mathcal{C}]$ with k-monotonic $\mathcal{C}$ by type-checking the term $\delta_k(M)$ in $\mathrm{UTT}[\delta(\mathcal{C})]$.

4 On Subtyping and Bounded Quantification

Works on subtyping and on specific type systems or programming languages with an implementation of subtyping or bounded quantification often have a variety of basic subtyping rules or judgements used to enrich the type system. Some of these are particularly popular amongst authors due to their power, or their ability in making for an expressive type system. When it comes to the metatheory of subtyping, particular instances or combinations of subtyping rules can also often cause problems with regards to normalisation and logical consistency, but also in often desirable properties, such as the decidability of subtyping.

4.1 With Dependent Functions

Bounded quantification was first introduced by Cardelli and Wegner in the language *Fun*, with a handful of subsumptive subtyping rules to introduce non-trivial subtypes into the system [3]. *Fun* has been a core for study and analysis, and several variations, simplifications and extensions have come about. In a paper analysing the subtyping of one of these variations called *minimal bounded Fun*, Pierce proves that the subtyping relation is undecidable by encoding the halting problem as a subtyping problem [17].

In particular, the interaction between two subtyping rules causes this undecidability; the existence of a universal supertype **Top**, and a dependent function subtyping rule.

$$\frac{\Gamma \vdash A\,\mathrm{type}}{\Gamma \vdash A \leq \mathbf{Top}} \qquad \frac{\Gamma \vdash B_1 \leq A_1 \quad \Gamma, \alpha \leq B_1 \vdash A_2 \leq B_2}{\Gamma \vdash \forall(\alpha \leq A_1).A_2 \leq \forall(\alpha \leq B_1).B_2}$$

Top alone can cause a plethora of issues (as discussed later in this section), but the function subtyping rule used is of particular interest. Castagna and Pierce have spoken about the issues this rule presents, and have discussed several variations [4]. However, the version presented above can also be presented in coercive subtyping.

▶ **Lemma 29** (Coherency of Π-Infer). *The subtyping rule*

$$\Pi\text{-}\mathit{Infer} \qquad \frac{\Gamma \vdash A_1 \leq_c B_1 \quad \Gamma, a : A_1 \vdash A_2\,\mathrm{type} \quad \Gamma, b : B_1 \vdash B_2\,\mathrm{type} \quad \Gamma, a : A_1 \vdash [c(a)/b]B_2 \leq_d A_2}{\Gamma \vdash \Pi(b : B_1).B_2 \leq_q \Pi(a : A_1).A_2}$$

is coherent, where

$$q = \lambda(g : \Pi(b : B_1).B_2).\lambda(a : A_1).d(a, g(c(a)))$$

Proof. Per the definition of coherency, we have three conditions we need to check. First, we check that q has the expected type.

$$q : (\Pi(x : B_1).B_2) \to (\Pi(x : A_1).A_2)$$

Secondly, we check the case where $A_1 = B_1$ and $B_2 = A_2$, and thus c is the identity function id_{A_1} and d is the constant function $\lambda(x : A_1).\,\mathrm{id}_{A_2}$. We obtain, through various computation rules,

$$\begin{aligned} q &= \lambda(g : \Pi(x : A_1).A_2).\lambda(x : A_1).d(x, g(c(x))) && (3)\\ &= \lambda(g : \Pi(x : A_1).A_2).\lambda(x : A_1).(\lambda(x : A_1).\,\mathrm{id})(x, g(\mathrm{id}(x))) && (4)\\ &= \lambda(g : \Pi(x : A_1).A_2).\lambda(x : A_1).\,\mathrm{id}(g(\mathrm{id}(x))) && (5)\\ &= \lambda(g : \Pi(x : A_1).A_2).\lambda(x : A_1).g(x) && (6)\\ &= \mathrm{id}_{\Pi(x:A_1).A_2)} && (7) \end{aligned}$$

as desired. Similarly, for the third case, it's easy to see that coherency of the hypothesis implies equality for multiple different derivations of q. ◀

One of the primary hurdles with this subtyping rule is that how the context is filled with regards to the codomains is uncertain, which may be evidence of an issue. Splitting this rule into two, with one for contravariance in the domain and the other for covariance in the codomain, is also possible [13] and arguably easier: Cardelli and Wegner's original formulation of the language *Fun* only uses a subtyping rule for the codomain, which can be shown to be decidable [17].

4.2 With Universal Supertypes

In coercive subtyping, the implementation of a universal supertype is often impossible for the sheer reason that it cannot be implemented coherently – **Top** cannot contain a single object[8], and so needs to be able to describe every possible object of the system. For extremely simple type theories, such as a theory containing only finite types and no type constructors, this is relatively trivial; but for even marginally more complex theories, the complexity and size of **Top** grows rapidly.

Even ignoring coherency issues for a moment, the same approach we have taken with respect to the metatheory of reflecting subtype universes on to type universes can't be taken. Adding a universal supertype **Top** to τ always results in non-monotonic subtyping; if we choose any $n \in \omega$ such that $\mathcal{L}_\Gamma(\mathbf{Top}) \stackrel{\text{def}}{=} n$, we can always find a subtype of strictly greater level (such as $\mathcal{U}^{n+1}(\mathbf{1})$.)

Introducing bounded quantification in conjunction with **Top** into a system also has an immediate concern in the semantics of the type $\forall(X \leq \mathbf{Top}).X$. Equivalently, in a system where we have mechanised bounded quantification via subtype universes or power types, we can consider the type $\mathcal{U}(\mathbf{Top})$ – for all intents and purposes, this should be a type of all types. By Girard's paradox, these systems should be non-normalising and thus inconsistent[9] [7].

Under the set-theoretic containment semantics, any universal supertype has to be transfinite in nature in the same way that a type of all types is transfinite in nature. It may be possible to "solve" the metatheoretic issues that universal supertypes present by taking a similar approach to type universes: by replacing **Top** with a series of partial supertypes $\mathbf{Top}_1, \mathbf{Top}_2, \ldots$ equipped with subtyping relations $\mathbf{Top}_1 \leq \mathbf{Top}_2 \leq \ldots$.

[8] Assuming the system has at least two distinct terms!

[9] There are several interesting routes through which one may attempt to obtain a proof of this inconsistency; the traditional approach here is to obtain what is essentially a bijection between a type and its power type, which is a contradiction by the diagonal lemma [10, 19]. Another possible route may be that, through subtype universes, a type theory may be capable of modelling itself – how large this model may be is unclear, however.

In fact, with subtype universes, such a set of partial supertypes can completely replace the typical use of type universes by replacing quantification over Type_i with bounded quantification over $\mathbf{Top}_i$. Furthermore, i doesn't necessarily need to be indexed by ω; one can take any partially ordered set I and, for $i, j \in I$, let $\mathbf{Top}_i \leq \mathbf{Top}_j$ whenever $i \leq j$.

4.3 With Subtype Universes

A subtyping inference rule for power types introduced by both Cardelli and Aspinall is as follows:

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

We can form an equivalent rule for τ as follows:

$$\frac{\Gamma \vdash A \leq_c B}{\Gamma \vdash \mathcal{U}(A) \leq_{\lambda(X:\mathcal{U}(A)).\langle \sigma_1(X), c \circ \sigma_2(X) \rangle} \mathcal{U}(B)}$$

which is well-typed and thus coherent by transitivity of subtyping. Under the set-theoretic notion of subtypes as subsets, this is also an extremely useful rule; we can reason about collections of subsets. On the other hand, this also greatly impacts any higher structure on subtype universes; we may wish to reason about the subtypes of $\mathcal{U}(B)$ without taking into account the subtypes of B itself.

There is also an issue of whether subtype universes and subtyping should be allowed to interact in the first place; subtype universes are an extension of a system with subtyping, and one may consider that system to have already had a set of subtyping relations judgements and rules implemented. Even then, being able to reason about subtyping relations with subtype universes can still be useful. We consider the following example:

$$O(n) \stackrel{\text{def}}{=} \Sigma(x : \mathbb{N}).(x < n)$$

$$O(n) \leq_{\pi_1} \mathbb{N}$$

$$\mathbb{N} \leq_{\lambda(n:\mathbb{N}).\langle O(n), \pi_1 \rangle} \mathcal{U}(\mathbb{N})$$

This example of ordinals-as-types was derived from looking at the logical consistency of certain subtyping relations and attempting to recreate Girard's paradox [7]. However, the two above subtyping judgements have an interesting property; the coherency of the second subtyping judgement now depends on the former. By allowing subtyping judgements to quantify over subtype universes, the coherency of any one subtyping judgement becomes dependent on the other judgements in the system.

4.4 Decidability of Typing and Subtyping

While we have sketched a proof that type-checking is decidable for monotonic subtyping and a subset of non-monotonic subtyping (i.e. those which are k-monotonic), there is still the open question of whether or not non-monotonic subtyping is decidable in general (i.e. for $\mathcal{C}$ that are non-monotonic but where there does not exist a k such that $\mathcal{C}$ is k-monotonic). Furthermore, our results rely primarily on the advantages that coercive subtyping brings: whether or not these ideas apply to systems that use subsumptive subtyping is left unanswered, especially for non-monotonicity.

There do exist examples of non-monotonic subtyping being decidable, such as Compagnoni's proof for System $F^{\omega}_{\wedge}$ [6]. $F^{\omega}_{\wedge}$ uses subsumptive subtyping and the dependent function subtyping rule, but drops the universal supertype **Top** in favour of empty intersection types quantifying over a kind. Additionally, Aspinall's work on power types lead to the development of rough-typing [1] – a kind of approximate type-checking that's powerful enough to still prove results such as strong normalisation. The algorithm Aspinall outlines provides enough information that one could likely refine it into a full type-checking algorithm.

In both Aspinall's work on power types and in Hutchins' work on pure subtype systems, [11] the authors point out that bounded quantification can be used to subsume a notion of kinds. Aspinall emulates the Edinburgh Logical Framework in λ_{Power}, and his rough type-checking can be seen as a notion of kinding . Likewise, Hutchins describes a process through which functions using kinds can be equivalently described through bounded quantification over types without any loss of generality.

This does open up several questions, such as whether you can "retrofit" kinds into a pure type system with bounded quantification. The difficulty of a generation or inversion lemma when working without coercive subtyping lends to one fear in regards to the combined use of both **Top** and power types. $\mathbf{Top} \leq \mathbf{Top}$ allows one to express non-terminating computations with power types; in a system where the distinction between terms and types are blurred, is it possible to form non-normalising types, just as one can express non-normalising terms?

5 Applications

5.1 Bounded Quantification

The mechanisation of bounded quantification was one of the key motivations for introducing power types in Cardelli's original paper [2]. Cardelli described a focus on the expressiveness of his system at the cost of non-terminating type-checking, but his formulation considered the case where subtyping was entire subsumed by typing (i.e. $A \leq B$ as shorthand for $A : \text{Power}(B)$). Maclean and Luo's subtype universes showed that the mechanisation of bounded quantification could preserve metatheoretic properties, but also kept typing and subtyping disjoint enough that subtype universes could lead to a more expressive system [16].

Subtype universes as described in this paper are capable of modelling bounded quantification; one should consider $\lambda(A \leq B).M$ as shorthand for $\lambda(x : \mathcal{U}(B)).[\sigma_1(x)/A]M$. This is particularly useful when it comes to record types. For example, consider a function

$$\text{darken} : \{\text{luminosity} : \text{Float32}\} \to \text{Float32} \to \{\text{luminosity} : \text{Float32}\}$$

This function is clearly sufficient in the case where we're handling objects that only carry luminosity data, but if we were to use subtyping to parse an object which also carried hue and saturation data, then we would receive an object with only luminosity data back. To fix this issue, we can use bounded quantification, and instead use the function

$$\text{darken} : \Pi(X : \mathcal{U}(\{\text{luminosity} : \text{Float32}\})).\sigma_1(X) \to \text{Float32} \to \sigma_1(X)$$

which allows us to preserve any excess information parsed in.

Subtyping can also be taken into consideration and used when designing languages and software to prevent errors. Often a collection of types designed to model information or objects will have some higher notion of structure on them; for example, the type $\mathbb{Q}$ equipped with addition, subtraction, multiplication and division forms a field. When designing the data types used to model these objects, we may wish for these operations to be as close to

independent of which type we're considering them in. For example, take Int16 $\leq_c$ Float32. For any two x, y : Int16, we would expect $c(x + y) = c(x) + c(y)$, and we may wish to choose a c or change our definitions of $+$ accordingly.

These approaches make for future-safe design and development of software. Often during the development of software one may wish to refactor code to improve its maintainability, reduce complexity, or prepare for adding new features; by taking these safe-guarding measures in the design-process, errors can be prevented and type-safety can be ensured. We also retain one of the key advantages of subtype universes and power types in that these objects can be interpreted as types; we can consider functions that range over types, which is not possible with just bounded quantification. Our system is also capable of modelling new kinds of subtyping relations through this process.

For example, for a given type B, consider the type of pointed subtypes $\Pi(x : \mathcal{U}(B)).\sigma_1(x)$. Intuitively, a pointed subtype of B is also a subtype of B, but Maclean and Luo's subtype universes had no way of describing this subtyping relation coherently. However, with the introduction of σ_2 in our system, we can obtain the exact coercion through which one type is a subtype of another type, and so we can use the coherent subtype relation

$$\Sigma(x : \mathcal{U}(B)).\sigma_1(x) \leq_q B \text{ where } q \stackrel{\text{def}}{=} \lambda(y : \Pi(x : \mathcal{U}(B)).\sigma_1(x)).(\sigma_2(\pi_1(y)))(\pi_2(y)).$$

With Maclean's subtype universes, we could implement behavioural subtyping with relative ease, but one could not describe subtyping relations which used bounded quantification. Being able to combine the two makes for a more expressive system.

5.2 Natural Langauge Semantics

Subtyping has a variety of applications in natural language semantics in describing the relationships between different categories and groups. When we start formalising these relationships, there quickly becomes a desire for some notion of bounded quantification.

Montague grammar, introduced in Richard Montague's seminal work, solves this problem by interpreting categorisation as propositions [18]. By semantically typing different language constructs, we can interpret a fully constructed sentence as a type. For example, we could interpret the sentence "all grass is green" as a term of type $\forall x.\text{isGrass}(x) \to \text{isGreen}(x)$. As lemongrass is a type of grass and thus $\text{isLemongrass}(x) \leq \text{isGrass(x)}$, we would also obtain the sentence "all lemongrass is green"[10].

However, we quickly run into an issue with Montague grammar in that we can form nonsense sentences: we can semantically type the sentence "all purple is trains" or "the month of December plays football", but these sentences don't make sense and are likely undesirable. We can instead model natural language semantics in a modern type theory, where every category of objects has its own type and subtyping is used to describe the relationships between categories of objects [14, 5]. For example, we can consider the type of Woman as a subtype of Human, or Chair as a subtype of Furniture. This allows us to use subtype universes to model categorisation of objects. Using $\mathcal{U}$-Infer as an example, one may infer from Fish $\leq$ Animals that $\mathcal{U}(\text{Fish}) \leq \mathcal{U}(\text{Animals})$ – i.e. that a type of species of fish is also type of species of animal.

We can also use subtype universes to model subsective adjectives. For example, how should one interpret the adjective "skillful" versus the adjective "small"? Let CN be the universe of common nouns. For any common noun, the interpretation of small : $\Pi(A$:

[10] While one may understand types as propositions via the Curry-Howard correspondence, the subtleties of subtyping with propositions in a type theory where propositions are treated distinctly from types is still an unexplored topic. Further analysis and discussion on this is outside the scope of this work, however.

CN).$A \to$ Prop is both sound and meaningful. However, using the same idea to obtain skillful : $\Pi(A : \mathrm{CN}).A \to$ Prop presents some issues. Whilst skillful(Doctor) makes sense, an example such as skillful(Chair) is obviously not intended. If we wish to exclude unintended combinations from our modelling of language, we can instead consider the semantic typing skillful : $\Pi(A : \mathcal{U}(\mathrm{Human})).\sigma_1(A) \to$ Prop. Of course, as Doctor $\leq_c$ Human, we have that skillful($\langle \mathrm{Doctor}, c \rangle$) is a well-typed expression. However, this now excludes unintended cases – skillful($\langle \mathrm{Chair}, c' \rangle$) is ill-typed because Chair $\not\leq$ Human.

5.3 Point-Set Topology

Subtyping has some interesting relationships with topology. For example, one could choose a set of subtyping judgements and rules such that a type and its subtypes model a space and its open sets. Under this application, the subtype universe of a space corresponds to its topology – the set of open sets.

This is a relatively easy process if the space we want to look at has a given metric, as the topology derived from a metric space is given by the union of open balls around points. As an example, we consider the rational numbers with the Euclidean metric (denoted d). We first model the rational numbers $\mathbb{Q}$ as $\mathbb{N} \times \mathbb{N}/0$ with addition, multiplication, metric, and ordering defined in the typical ways, e.g. we define the Euclidean metric $d : \mathbb{Q} \to \mathbb{Q} \to \mathbb{Q}$ such that $d(x, y) = |x - y|$.

We then consider the following three coherent subtyping rules:

$$\frac{\Gamma \vdash z : \mathbb{Q}, \quad \Gamma \vdash r : \mathbb{Q}}{\Gamma \vdash \Sigma(x : \mathbb{Q}).(d(z, x) < r) \leq_{\pi_1} \mathbb{Q}} \qquad \frac{\Gamma \vdash I \,\mathrm{type} \quad \Gamma, x : I \vdash A \leq_c \mathbb{Q}}{\Gamma \vdash \Sigma(i : I)A \leq_{\lambda(p:\Sigma(i:I)A).c(\pi_1(p),\pi_2(p))} \mathbb{Q}}$$

$$\frac{\Gamma \vdash A <_c \mathbb{Q} \quad \Gamma \vdash B <_{c'} \mathbb{Q}}{\Gamma \vdash \Sigma(a : A).\Sigma(b : B).(c(a) = c'(b)) <_{c \circ \pi_1} \mathbb{Q}}$$

These three rules are sufficient for $\mathcal{U}(\mathbb{Q})$ to be a topology of $\mathbb{Q}$. It's rather simple to check that there exists an empty subtype; that the arbitrary union of subtypes is also a subtype; and that the intersection of two subtypes is also a subtype.

We do, however, have a multitude of technical and semantic issues to work our way through. Is this the correct notion of union and intersection, for example? Have we chosen our basis correctly, or is there a different basis for the topology which is more convenient to work with (for example, slicing the real line)?

Under the above rules, there exists multiple empty subtypes – whilst the rules we've introduced could be further refined to prevent these issues, we may also wish to reason about $\mathcal{U}(\mathbb{Q})$ as a setoid. Similarly, we may also want to reason about $\mathbb{Q}$ as a setoid as there exists multiple different ways of expressing the same rational number: for example, $1/2$ is represented by the pairs $(1, 2)$, $(2, 4)$, $(3, 6)$, and so on. There is an obvious notion of propositional equality $\mathrm{Eq}_{\mathbb{Q}}$ we can equip to $\mathbb{Q}$ by defining

$$\mathrm{Eq}_{\mathbb{Q}}(p, q) \stackrel{\mathrm{def}}{=} (\pi_1(p - q) = 0) : \mathrm{Prop}\,.$$

However, with a notion of point-set topology formalised, we can also equip $\mathbb{Q}$ with a notion of equality based on open sets.

$$\mathrm{Eq}_{\mathbb{Q}}{}'(p, q) \stackrel{\mathrm{def}}{=} \forall(x : \mathcal{U}(\mathbb{Q})).((\exists(r : \sigma_1(x)).(p = r)) \wedge (\exists(s : \sigma_1(x)).(q = s))).$$

Whilst $\mathrm{Eq}_{\mathbb{Q}}$ is certainly more reasonable for reasoning about arithmetic or number theory, it's plausible that $\mathrm{Eq}_{\mathbb{Q}}{}'$ and similar notions may be more useful for reasoning about continuity, limits, or Cauchy sequences. Exploring this further is outside the scope of this paper, however.

When using subtype universes to model topologies, the coercions used in the subtyping judgements can be understood as mapping open sets to open sets. We leave open the question of whether these coercions can be interpreted as a continuous embedding of one space into another and what this means for the semantics of a type theory. Nonetheless, understanding subtyping as continuous embedding could provide some new intuition for the problems regarding universal supertypes: **Top** is not only a universal supertype, but also a space in which every space in a type theory can be embedded into. If we want to be able to use a universal super type with coercive subtyping, then we need some way to describe every object of our type theory.

We can immediately ask questions and draw conclusions about what such a space looks like: for example, if we take a type theory consisting only of finite types with no type constructors, then **Top** is described by $\mathbb{N}$. For any type theory modelling anything more complicated, **Top** is unlikely to look like a slice of $\mathbb{R}^\infty$, as any space embedding into $\mathbb{R}^\infty$ must be both seperable and metrizable[11].

6 Conclusion

This work generalised and extended Maclean and Luo's prior notion of subtype universes in order to provide support for a much wider range of coercive subtyping relations. By examining a type system lacking the traditional type universe hierarchy of $\mathrm{Type}_0, \mathrm{Type}_1, \mathrm{Type}_2, ...$, we have allowed for subtypes more complicated than their supertype; and by allowing one to obtain the coercion through the σ_2 operator, we are able to express coherent subtyping judgements and rules that use subtype universes.

In doing so, we have found that the metatheory remains relatively well-behaved regardless of the choice in subtyping relations; both monotonic and k-monotonic subtyping result in logical consistency and strong normalisation of terms. Additionally, we have sketched a proof of the decidability of type-checking and subtyping in both cases by embedding terms into $\mathrm{UTT}[\mathcal{C}]$ and type-checking there instead. However, whether or not similar results can be proven for non-monotonic subtyping in general (i.e. where there does not exist a k such that subtyping is k-monotonic) is left open.

Throughout the course of this paper, we hope to have shed some light regarding particular uses of subtyping, such as the difficulties of using a universal supertype **Top**. However, we have left several questions open, such as whether or not τ is confluent, the subtleties of subtyping between propositions, and how closely linked subtyping and subtype universes are to a notion of continuous embeddings.

In particular, we want to more closely examine Hutchins' work on pure subtype systems [11]: systems akin to pure type systems wherein the typing relation is entirely subsumed by the subtyping relation, a notion "almost completely dual to [...] the approach taken by Cardelli" . Simple questions such as the decidability of subtyping or how one may implement a notion of propositional logic into a pure subtype system are left open, and we look forward to working on this in the future.

References

1 David Aspinall. Subtyping with power types. In Peter G. Clote and Helmut Schwichtenberg, editors, *Computer Science Logic*, pages 156–171, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[11] More formally, every seperable and metrizable space is homeomorphic to a subset of the Hilbert cube $[0,1]^\infty$, which is a subspace of $\mathbb{R}^\infty$. This is established in the proof of Urysohn's metrization theorem.

2 Luca Cardelli. Structural subtyping and the notion of power type. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, pages 70–79, New York, NY, USA, 1988. Association for Computing Machinery. doi:10.1145/73560.73566.

3 Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, 17(4):471–523, December 1985. doi:10.1145/6041.6042.

4 Giuseppe Castagna and Benjamin C. Pierce. Decidable bounded quantification. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, pages 151–162, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/174675.177844.

5 Stergios Chatzikyriakidis and Zhaohui Luo. On the interpretation of common nouns: Types versus predicates. In Stergios Chatzikyriakidis and Zhaohui Luo, editors, *Modern Perspectives in Type-Theoretical Semantics*, pages 43–70. Springer Cham, 2017. doi:10.1007/978-3-319-50422-3.

6 Adriana Compagnoni. Higher-order subtyping and its decidability. *Information and Computation*, 191(1):41–103, 2004. doi:10.1016/j.ic.2004.01.001.

7 Thierry Coquand. An analysis of Girard's paradox. Technical Report RR-0531, INRIA, May 1986. URL: https://hal.inria.fr/inria-00076023.

8 Healfdene Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, 1994.

9 Douglas J. Howe. The computational behaviour of girard's paradox. Technical report, Cornell University, Ithaca, New York, USA, March 1987.

10 Antonius J. C. Hurkens. A simplification of Girard's paradox. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, pages 266–278, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

11 DeLesley S. Hutchins. Pure subtype systems. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, pages 287–298, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1706299.1706334.

12 Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Oxford University Press, London, March 1994.

13 Zhaohui Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, February 1999. doi:10.1093/logcom/9.1.105.

14 Zhaohui Luo. Common nouns as types. In D. Béchet and A. Dikovsky, editors, *Proceedings of the 7th International Conference on Logical Aspects of Computational Linguistics (LACL'12)*, pages 173–185, Berlin, Heidelberg, 2012. Springer-Verlag.

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

16 Harry Maclean and Zhaohui Luo. Subtype Universes. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs (TYPES 2020)*, volume 188 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TYPES.2020.9.

17 Benjamin C. Pierce. Bounded quantification is undecidable. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '92, pages 305–315, New York, NY, USA, 1992. Association for Computing Machinery. doi:10.1145/143165.143228.

18 Richmond H. Thomason, editor. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974.

19 Kevin Watkins. Hurkens' simplification of Girard's paradox, July 2004. URL: https://www.cs.cmu.edu/~kw/research/hurkens95tlca.elf.