On the Metatheory of Subtype Universes

Felix Bradley Zhaohui Luo

Department of Computer Science
Royal Holloway, University of London

24th April 2023

Presentation Overview

A few definitions

Definition (Type Universe)

A type U is a *universe* if its objects are types or can be interpreted as types.

A few definitions

Definition (Type Universe)

A type U is a *universe* if its objects are types or can be interpreted as types.

- U is *impredicative* if it contains itself, i.e. $U : U$, and *predicative* otherwise.

A few definitions

Definition (Type Universe)

A type U is a *universe* if its objects are types or can be interpreted as types.

- U is *impredicative* if it contains itself, i.e. $U : U$, and *predicative* otherwise.
- U is a *Russell-style universe* if the objects of U are types, i.e. $A : U \vdash A$ type.

A few definitions

Definition (Type Universe)

A type U is a *universe* if its objects are types or can be interpreted as types.

- U is *impredicative* if it contains itself, i.e. $U : U$, and *predicative* otherwise.
- U is a *Russell-style universe* if the objects of U are types, i.e. $A : U \vdash A$ type.
- U is a *Tarski-style universe* if it is instead equipped with an operator $\mathbb{T}$ that interprets the objects of U as types, i.e. $a : U \vdash \mathbb{T}(a)$ type

Examples

- Coq uses predicative Tarski-style universes $\mathsf{Type}_0, \mathsf{Type}_1, \mathsf{Type}_2 \ldots$ and an impredicative universe of propositions Prop [ST14]

Type universes

Examples

- Coq uses predicative Tarski-style universes $\mathsf{Type}_0, \mathsf{Type}_1, \mathsf{Type}_2 \ldots$ and an impredicative universe of propositions Prop [ST14]
- Agda uses predicative Russell-style universes for sets $\mathsf{Set}_0 : \mathsf{Set}_1 : \mathsf{Set}_2 : \ldots : \mathsf{Set}_\omega$ and propositions $\mathsf{Prop}_0 : \mathsf{Prop}_1 : \mathsf{Prop}_2 : \ldots$ [Nor+05]

Type universes

Examples

- Coq uses predicative Tarski-style universes $\mathrm{Type}_0, \mathrm{Type}_1, \mathrm{Type}_2 \ldots$ and an impredicative universe of propositions Prop [ST14]
- Agda uses predicative Russell-style universes for sets $\mathrm{Set}_0 : \mathrm{Set}_1 : \mathrm{Set}_2 : \ldots : \mathrm{Set}_\omega$ and propositions $\mathrm{Prop}_0 : \mathrm{Prop}_1 : \mathrm{Prop}_2 : \ldots$ [Nor+05]
- Book HoTT uses predicative Russell-style universes $U_0 : U_1 : U_2 : \ldots$ [Uni13]

A few definitions

What is a subtype? For types A and B, what does it mean for A to be a subtype of B?

A few definitions

What is a subtype? For types A and B, what does it mean for A to be a subtype of B?

Definition (Subsumptive Subtyping)

If A is a subtype of B, then any object of type A is also an object of type B.

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A \leq B}{\Gamma \vdash a : B}$$

Advantages:

- Extremely simple
- Closely linked to set-theoretic intuition
- Good for programming languages

Advantages:

- Extremely simple
- Closely linked to set-theoretic intuition
- Good for programming languages

Disadvantages:

- Canonicity of objects fails [Luo12]
- Unclear if extending the type theory with new subtyping rules is conservative
- Questions of decidable subtyping, type checking, minimal types

A few definitions

Definition (Coercive Subtyping)

Intuition: If A is a subtype of B, then whenever we require an object of type B, it is sufficient to provide an object of type A.

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

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

Coercive subtyping

Advantages:

- Well-behaved metatheory; adding any coherent subtyping rule is a conservative extension [LSX13]
- Canonicity of objects is preserved
- Ideal for logical type theories and proof assistants

Coercive subtyping

Advantages:

Well-behaved metatheory; adding any coherent subtyping rule is a conservative extension [LSX13]

Canonicity of objects is preserved

Ideal for logical type theories and proof assistants

Disadvantages:

Need two-step reduction - first insert coercions, then perform standard reduction

Checking the coherency of subtyping rules is non-trivial

Bounded quantification

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification is a desirable concept in the design of type theories and programming languages.

Bounded quantification

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification is a desirable concept in the design of type theories and programming languages.

- Structural subtyping: $\Sigma(x : A).P(x) \leq A$

Bounded quantification

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification is a desirable concept in the design of type theories and programming languages.

- Structural subtyping: $\Sigma(x : A).P(x) \leq A$
- Record types: $\{\text{name} : \text{String}, \text{age} : \text{Nat}\} \leq \{\text{name} : \text{String}\}$

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification is a desirable concept in the design of type theories and programming languages.

- Structural subtyping: $\Sigma(x : A).P(x) \leq A$
- Record types: $\{\text{name} : \text{String}, \text{age} : \text{Nat}\} \leq \{\text{name} : \text{String}\}$
- Type Conversions:
  $f : \text{Float} \to \text{Float}, x : \text{Int16}, \text{Int16} \leq_c \text{Float} \vdash f(x) = f(c(x)) : \text{Float}$

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification is a desirable concept in the design of type theories and programming languages.

- Structural subtyping: $\Sigma(x : A).P(x) \leq A$
- Record types: $\{\text{name} : \text{String}, \text{age} : \text{Nat}\} \leq \{\text{name} : \text{String}\}$
- Type Conversions:
  $f : \text{Float} \to \text{Float}, x : \text{Int16}, \text{Int16} \leq_c \text{Float} \vdash f(x) = f(c(x)) : \text{Float}$

Bounded quantification

Definition (Bounded Quantification)

A form of (universal or existential) quantifier which quantifies over subtypes of a given type.

$$\lambda(X \leq B).M$$

Bounded quantification is a desirable concept in the design of type theories and programming languages.

- Structural subtyping: $\Sigma(x : A).P(x) \leq A$
- Record types: $\{\text{name} : \text{String}, \text{age} : \text{Nat}\} \leq \{\text{name} : \text{String}\}$
- Type Conversions:
  $f : \text{Float} \to \text{Float}, x : \text{Int16}, \text{Int16} \leq_c \text{Float} \vdash f(x) = f(c(x)) : \text{Float}$

But certain choices of subtyping rules can cause problems.

- Benjamin Pierce showed that $F_{\leq}$ had undecidable subtyping and undecidable type checking [Pie92; CP94]

What is a power type?

Cardelli introduced the notion of power types for subsumptive subtyping - the universe of subtypes of a given type [Car88].

What is a power type?

Cardelli introduced the notion of power types for subsumptive subtyping - the universe of subtypes of a given type [Car88].

$$A \leq B \text{ as shorthand for } A : \mathrm{Power}(B)$$

What is a power type?

Cardelli introduced the notion of power types for subsumptive subtyping - the universe of subtypes of a given type [Car88].

$$A \leq B \text{ as shorthand for } A : \mathrm{Power}(B)$$

$$\lambda(X \leq A).M \stackrel{\text{def}}{=} \lambda(X : \mathrm{Power}(A)).M$$

What is a power type?

Cardelli introduced the notion of power types for subsumptive subtyping - the universe of subtypes of a given type [Car88].

$$A \leq B \text{ as shorthand for } A : \mathrm{Power}(B)$$

$$\lambda(X \leq A).M \stackrel{\text{def}}{=} \lambda(X : \mathrm{Power}(A)).M$$

- Cardelli was interested in programming language design

What is a power type?

Cardelli introduced the notion of power types for subsumptive subtyping - the universe of subtypes of a given type [Car88].

$$A \leq B \text{ as shorthand for } A : \mathrm{Power}(B)$$

$$\lambda(X \leq A).M \stackrel{\text{def}}{=} \lambda(X : \mathrm{Power}(A)).M$$

- Cardelli was interested in programming language design
- His proposed type theory included an impredicative universe of all types

What is a power type?

Cardelli introduced the notion of power types for subsumptive subtyping - the universe of subtypes of a given type [Car88].

$$A \leq B \text{ as shorthand for } A : \mathrm{Power}(B)$$

$$\lambda(X \leq A).M \stackrel{\text{def}}{=} \lambda(X : \mathrm{Power}(A)).M$$

- Cardelli was interested in programming language design
- His proposed type theory included an impredicative universe of all types
- Great for programming, problematic for nice metatheory

Metatheory of bounded quantification

- Aspinall reformulated Cardelli's ideas on power types into a predicative typed lambda calculus λ_{Power} [Asp00].

Metatheory of bounded quantification

- Aspinall reformulated Cardelli's ideas on power types into a predicative typed lambda calculus λ_{Power} [Asp00].
- Aspinall developed a 'rough type'-checking algorithm, and proved the system was strongly normalising...

Metatheory of bounded quantification

- Aspinall reformulated Cardelli's ideas on power types into a predicative typed lambda calculus λ_{Power} [Asp00].
- Aspinall developed a 'rough type'-checking algorithm, and proved the system was strongly normalising...
- ...but was unable to prove a inversion lemma/generation principle.

Metatheory of bounded quantification

- Aspinall reformulated Cardelli's ideas on power types into a predicative typed lambda calculus λ_{Power} [Asp00].
- Aspinall developed a 'rough type'-checking algorithm, and proved the system was strongly normalising...
- ...but was unable to prove a inversion lemma/generation principle.
- The metatheory of subtyping has an inherent difficulty: transitivity [AC96; Com04; Hut09].

$$\frac{\Gamma \vdash A \leq B \quad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C}$$

Unifying Theory of dependent Types

- Zhaohui Luo's UTT is a well-studied type theory with nice metatheoretical properties [Luo94; Gog94]

Unifying Theory of dependent Types

- Zhaohui Luo's UTT is a well-studied type theory with nice metatheoretical properties [Luo94; Gog94]
- UTT's extension with coercive subtyping ($\mathrm{UTT}[C]$) has been proven to be conservative [Luo97; LSX13].

Unifying Theory of dependent Types

- Zhaohui Luo's UTT is a well-studied type theory with nice metatheoretical properties [Luo94; Gog94]
- UTT's extension with coercive subtyping (UTT$[C]$) has been proven to be conservative [Luo97; LSX13].

Harry Maclean and Zhaohui Luo introduced subtype universes - predicative universes of subtypes for coercive subtyping - formulated as an extension of UTT$[C]$ [ML21].

Unifying Theory of dependent Types

- Zhaohui Luo's UTT is a well-studied type theory with nice metatheoretical properties [Luo94; Gog94]
- UTT's extension with coercive subtyping (UTT$[\mathcal{C}]$) has been proven to be conservative [Luo97; LSX13].

Harry Maclean and Zhaohui Luo introduced subtype universes - predicative universes of subtypes for coercive subtyping - formulated as an extension of UTT$[\mathcal{C}]$ [ML21].

$$A \leq_c B \vdash a : \mathcal{U}(B) \text{ where } \mathbb{T}(a) = A$$

Unifying Theory of dependent Types

- Zhaohui Luo's UTT is a well-studied type theory with nice metatheoretical properties [Luo94; Gog94]
- UTT's extension with coercive subtyping (UTT$[\mathcal{C}]$) has been proven to be conservative [Luo97; LSX13].

Harry Maclean and Zhaohui Luo introduced subtype universes - predicative universes of subtypes for coercive subtyping - formulated as an extension of UTT$[\mathcal{C}]$ [ML21].

$$A \leq_c B \vdash a : \mathcal{U}(B) \text{ where } \mathbb{T}(a) = A$$

This extended type theory UTT$[\mathcal{C}]_{\mathcal{U}}$ embeds back into UTT$[\mathcal{C}]$.

Monotonic Subtyping

However, they only proved that $\mathrm{UTT}[\mathcal{C}]_{\mathcal{U}}$ were strongly normalising when the set of subtyping relations were ‘nice’.

Monotonic Subtyping

However, they only proved that $\mathrm{UTT}[\mathcal{C}]_{\mathcal{U}}$ were strongly normalising when the set of subtyping relations were 'nice'.

1) Subtyping relations can't use subtype universes.

Monotonic Subtyping

ROYAL HOLLOWAY UNIVERSITY OF LONDON

However, they only proved that $\mathrm{UTT}[\mathcal{C}]_{\mathcal{U}}$ were strongly normalising when the set of subtyping relations were 'nice'.

1) Subtyping relations can't use subtype universes.
2) Subtyping relations can't use propositions.

Monotonic Subtyping

However, they only proved that $\mathrm{UTT}[\mathcal{C}]_{\mathcal{U}}$ were strongly normalising when the set of subtyping relations were 'nice'.

1) Subtyping relations can't use subtype universes.
2) Subtyping relations can't use propositions.
3) A type cannot inhabit a universe smaller than the universes its subtypes inhabit.

Monotonic Subtyping

However, they only proved that $\mathrm{UTT}[\mathcal{C}]_{\mathcal{U}}$ were strongly normalising when the set of subtyping relations were 'nice'.

1) Subtyping relations can't use subtype universes.
2) Subtyping relations can't use propositions.
3) A type cannot inhabit a universe smaller than the universes its subtypes inhabit.

Definition (Monotonicity of Subtyping)

A subtyping judgement $A \leq B$ is *monotonic* if the smallest type universe B inhabits is larger than the smallest type universe A inhabits.

$$\forall j \text{ s.t. } B : \mathrm{Type}_j, \exists i \text{ s.t. } A : \mathrm{Type}_i \text{ and } i \leq j$$

Monotonic Subtyping

ROYAL HOLLOWAY UNIVERSITY OF LONDON

However, they only proved that $\mathrm{UTT}[\mathcal{C}]_{\mathcal{U}}$ were strongly normalising when the set of subtyping relations were 'nice'.

1) Subtyping relations can't use subtype universes.
2) Subtyping relations can't use propositions.
3) A type cannot inhabit a universe smaller than the universes its subtypes inhabit.

Definition (Monotonicity of Subtyping)

A subtyping judgement $A \leq B$ is *monotonic* if the smallest type universe B inhabits is larger than the smallest type universe A inhabits.

$$\forall j \text{ s.t. } B : \mathrm{Type}_j, \exists i \text{ s.t. } A : \mathrm{Type}_i \text{ and } i \leq j$$

Are any of these conditions necessary?

Generalising Subtype Universes

We consider a base type theory τ with an impredicative universe of propositions, inductive type constructors, Π types and Σ types, and coercive subtyping given by a collection of coherent subtyping rules C.

Generalising Subtype Universes

We consider a base type theory τ with an impredicative universe of propositions, inductive type constructors, Π types and Σ types, and coercive subtyping given by a collection of coherent subtyping rules C.

Key idea:

Generalising Subtype Universes

We consider a base type theory τ with an impredicative universe of propositions, inductive type constructors, Π types and Σ types, and coercive subtyping given by a collection of coherent subtyping rules C.

Key idea: Objects of a subtype universe should also hold information about the coercion.

Generalising Subtype Universes

We consider a base type theory τ with an impredicative universe of propositions, inductive type constructors, Π types and Σ types, and coercive subtyping given by a collection of coherent subtyping rules C.

Key idea: Objects of a subtype universe should also hold information about the coercion.

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

$$\frac{\Gamma \vdash B \text{ type} \quad \Gamma \vdash t : \mathcal{U}(B)}{\Gamma \vdash \sigma_1(t) \text{ type}} \qquad \frac{\Gamma \vdash B \text{ type} \quad \Gamma \vdash \langle A, c \rangle : \mathcal{U}(B)}{\Gamma \vdash \sigma_1(\langle A, c \rangle) = A}$$

$$\frac{\Gamma \vdash B \text{ type} \quad \Gamma \vdash t : \mathcal{U}(B)}{\Gamma \vdash \sigma_2(t) : \sigma_1(t) \to B} \qquad \frac{\Gamma \vdash B \text{ type} \quad \Gamma \vdash \langle A, c \rangle : \mathcal{U}(B)}{\Gamma \vdash \sigma_2(\langle A, c \rangle) = c : A \to B}$$

Generalising Subtype Universes

Example

For a given type B, consider the type of *pointed subtypes of B* given by

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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Example

For a given type B, consider the type of *pointed subtypes of B* given by

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

Intuitively, this should be a subtype of B.

Generalising Subtype Universes

Example

For a given type B, consider the type of *pointed subtypes of B* given by

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

Intuitively, this should be a subtype of B. Define

$$f \stackrel{\text{def}}{=} \lambda(p : \Sigma(x : \mathcal{U}(B)).\sigma_1(x)).\sigma_2(\pi_1(p))(\pi_2(p))$$

Example

For a given type B, consider the type of *pointed subtypes of B* given by

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

Intuitively, this should be a subtype of B. Define

$$f \stackrel{\text{def}}{=} \lambda(p : \Sigma(x : \mathcal{U}(B)).\sigma_1(x)).\sigma_2(\pi_1(p))(\pi_2(p))$$

Then

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

is a coherent subtyping relation (in the sense of [LSX13]).

Generalising Subtype Universes

Example

For a given type B, consider the type of *pointed subtypes of B* given by

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

Intuitively, this should be a subtype of B. Define

$$f \stackrel{\text{def}}{=} \lambda(p : \Sigma(x : \mathcal{U}(B)).\sigma_1(x)).\sigma_2(\pi_1(p))(\pi_2(p))$$

Then

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

is a coherent subtyping relation (in the sense of [LSX13]).

However, this subtyping relation is nonmonotonic, as the LHS of the subtyping relation contains a larger universe than the RHS

We start by analysing the case where the collection of subtyping rules C is entirely monotonic.

We start by analysing the case where the collection of subtyping rules C is entirely monotonic.

We construct an embedding $\delta : \tau \to \mathrm{UTT}[C]$ defined inductively over terms of τ.

Metatheory for monotonic subtyping

We start by analysing the case where the collection of subtyping rules C is entirely monotonic.
We construct an embedding $\delta : \tau \to \mathrm{UTT}[C]$ defined inductively over terms of τ.

$$\delta(\mathcal{U}(B)) \stackrel{\text{def}}{=} \Sigma(X : \mathrm{Type}_{\mathcal{L}_\Gamma(B)}).(X \to \delta(B))$$

$$\delta(\langle A, c \rangle) \stackrel{\text{def}}{=} (\mathbf{n}(\delta(A)), \delta(c))$$

Metatheory for monotonic subtyping

Lemma

If $\Gamma \vdash A$ type, *then there exists some term n in* $\mathrm{UTT}[\delta(\mathcal{C})]$ *such that the following hold:*

- $\delta(\Gamma) \vdash \delta(A) : \mathbf{Type}$
- $\delta(\Gamma) \vdash n : \mathrm{Type}_{\mathcal{L}_\Gamma(A)}$
- $\mathbb{T}_{\mathcal{L}_\Gamma(A)}(n) = \delta(A)$

Metatheory for monotonic subtyping

Lemma

If $\Gamma \vdash A$ type*, then there exists some term n in* $\mathrm{UTT}[\delta(\mathcal{C})]$ *such that the following hold:*

- $\delta(\Gamma) \vdash \delta(A) : \textbf{Type}$
- $\delta(\Gamma) \vdash n : \mathrm{Type}_{\mathcal{L}_\Gamma(A)}$
- $\mathbb{T}_{\mathcal{L}_\Gamma(A)}(n) = \delta(A)$

Lemma

The rules of τ *under translation via* δ *are admissible in* $\mathrm{UTT}[\delta(\mathcal{C})]$.

Metatheory for monotonic subtyping

Lemma

If $\Gamma \vdash A$ type*, then there exists some term* n *in* $\mathrm{UTT}[\delta(\mathcal{C})]$ *such that the following hold:*

- $\delta(\Gamma) \vdash \delta(A) : \textbf{Type}$
- $\delta(\Gamma) \vdash n : \mathrm{Type}_{\mathcal{L}_\Gamma(A)}$
- $\mathbb{T}_{\mathcal{L}_\Gamma(A)}(n) = \delta(A)$

Lemma

The rules of τ *under translation via* δ *are admissible in* $\mathrm{UTT}[\delta(\mathcal{C})]$.

Proof.

Arduous. □

Theorem (Logical Consistency for Monotonic Subtyping)

τ is logically consistent, i.e. there is no Γ such that $\Gamma \vdash p : \forall(P : \mathrm{Prop}).P$

Metatheory for monotonic subtyping

Theorem (Logical Consistency for Monotonic Subtyping)

τ *is logically consistent, i.e. there is no* Γ *such that* $\Gamma \vdash p : \forall(P : \mathrm{Prop}).P$

Theorem (Strong Normalisation for Monotonic Subtyping)

τ *is strongly normalising, i.e. if* $\Gamma \vdash M : A$ *then every possible sequence of reductions of* M *is finite.*

Metatheory for monotonic subtyping

Theorem (Logical Consistency for Monotonic Subtyping)

τ is logically consistent, i.e. there is no Γ such that $\Gamma \vdash p : \forall(P : \mathrm{Prop}).P$

Theorem (Strong Normalisation for Monotonic Subtyping)

τ is strongly normalising, i.e. if $\Gamma \vdash M : A$ then every possible sequence of reductions of M is finite.

Remark (Decidability of Typing and Subtyping)

As δ is injective, we can type-check any given term M in τ by type-checking $\delta(M)$ in $\mathrm{UTT}[\delta(\mathcal{C})]$.
Likewise, we can decide if $\Gamma \vdash A \leq B$ by looking at a term t which is typed if and only if $A \leq B$ is derivable, e.g. $\lambda(f : B \to \mathbb{N}).\lambda(a : A).f(a)$

Key idea:

Metatheory for nonmonotonic subtyping

Key idea: We should be able to insert coercions to ‘reduce’ a system with a nonmonotonic subtyping relation to one without that relation.

Key idea: We should be able to insert coercions to 'reduce' a system with a nonmonotonic subtyping relation to one without that relation.

Definition

Let T be a type system, and T' be an extension of T. T' is a *weakly conservative extension* of T if the following hold:

- For every A in T', every derivation of the form $\vdash_{T'} A \text{ type}$ is also derivable in T
- For every a in T' and every derivation of the form $\vdash a : A$ in T', there exists some term b in T such that $\vdash b : A$

Metatheory for nonmonotonic subtyping

Key idea: We should be able to insert coercions to 'reduce' a system with a nonmonotonic subtyping relation to one without that relation.

Definition

Let T be a type system, and T' be an extension of T. T' is a *weakly conservative extension* of T if the following hold:

- For every A in T', every derivation of the form $\vdash_{T'} A\ \mathsf{type}$ is also derivable in T
- For every a in T' and every derivation of the form $\vdash a : A$ in T', there exists some term b in T such that $\vdash b : A$

In other words, T' is a weakly conservative extension of T if it does not add any new types, and only adds terms to already inhabited types.

Metatheory for nonmonotonic subtyping

Key idea: We should be able to insert coercions to ‘reduce’ a system with a nonmonotonic subtyping relation to one without that relation.

Definition

Let T be a type system, and T' be an extension of T. T' is a *weakly conservative extension* of T if the following hold:

- For every A in T', every derivation of the form $\vdash_{T'} A\ \mathsf{type}$ is also derivable in T
- For every a in T' and every derivation of the form $\vdash a : A$ in T', there exists some term b in T such that $\vdash b : A$

In other words, T' is a weakly conservative extension of T if it does not add any new types, and only adds terms to already inhabited types.

Corollary

Weakly conservative extensions preserve strong normalisation and logical consistency.

Metatheory for nonmonotonic subtyping

By adding new subtyping relations to τ, the conditions under which terms using subtype universes exists become stricter.

Metatheory for nonmonotonic subtyping

By adding new subtyping relations to τ, the conditions under which terms using subtype universes exists become stricter.

- No new functions.

Metatheory for nonmonotonic subtyping

By adding new subtyping relations to τ, the conditions under which terms using subtype universes exists become stricter.

- No new functions.
- Possibly some new pairs?

Metatheory for nonmonotonic subtyping

By adding new subtyping relations to τ, the conditions under which terms using subtype universes exists become stricter.

- No new functions.
- Possibly some new pairs?

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

Metatheory for nonmonotonic subtyping

By adding new subtyping relations to τ, the conditions under which terms using subtype universes exists become stricter.

- No new functions.
- Possibly some new pairs?

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

$$(\langle A, c\rangle, a) : \Sigma(x : \mathcal{U}(B)).F(x)$$

Metatheory for nonmonotonic subtyping

By adding new subtyping relations to τ, the conditions under which terms using subtype universes exists become stricter.

- No new functions.
- Possibly some new pairs?

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

$$(\langle A, c \rangle, a) : \Sigma(x : \mathcal{U}(B)).F(x)$$

$$(\langle B, \mathrm{id} \rangle, c(a)) : \Sigma(x : \mathcal{U}(B)).F(x)$$

Summary:

- Generalised subtype universes to be able to express new subtyping relations

Conclusion

Summary:

- Generalised subtype universes to be able to express new subtyping relations
- Strong normalisation and logical consistency for monotonic and non-monotonic subtyping

Summary:

- Generalised subtype universes to be able to express new subtyping relations
- Strong normalisation and logical consistency for monotonic and non-monotonic subtyping
- Decidability of type-checking, minimal types, and the subtype relation

Conclusion

Summary:

- Generalised subtype universes to be able to express new subtyping relations
- Strong normalisation and logical consistency for monotonic and non-monotonic subtyping
- Decidability of type-checking, minimal types, and the subtype relation

Open questions/future work:

- What does subtyping mean between propositions?

Summary:

- Generalised subtype universes to be able to express new subtyping relations
- Strong normalisation and logical consistency for monotonic and non-monotonic subtyping
- Decidability of type-checking, minimal types, and the subtype relation

Open questions/future work:

- What does subtyping mean between propositions?
- Formalisation of subtype universes in a proof assistant

Conclusion

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Summary:

- Generalised subtype universes to be able to express new subtyping relations
- Strong normalisation and logical consistency for monotonic and non-monotonic subtyping
- Decidability of type-checking, minimal types, and the subtype relation

Open questions/future work:

- What does subtyping mean between propositions?
- Formalisation of subtype universes in a proof assistant
- A pure subtype system for proofs [Hut09]

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Summary:

- Generalised subtype universes to be able to express new subtyping relations
- Strong normalisation and logical consistency for monotonic and non-monotonic subtyping
- Decidability of type-checking, minimal types, and the subtype relation

Open questions/future work:

- What does subtyping mean between propositions?
- Formalisation of subtype universes in a proof assistant
- A pure subtype system for proofs [Hut09]

Thank you for listening!

[AC96] David Aspinall and Adriana Compagnoni. “Subtyping dependent types”. In: *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*. 1996, pp. 86–97. DOI: 10.1109/LICS.1996.561307.

[Asp00] David Aspinall. “Subtyping with Power Types”. In: *Computer Science Logic*. Ed. by Peter G. Clote and Helmut Schwichtenberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 156–171. ISBN: 978-3-540-44622-4.

[Car88] 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. San Diego, California, USA: Association for Computing Machinery, 1988, pp. 70–79. ISBN: 0897912527. DOI: 10.1145/73560.73566. URL: https://doi.org/10.1145/73560.73566.

References II

[Com04] Adriana Compagnoni. “Higher-order subtyping and its decidability”. In: *Information and Computation* 191.1 (2004), pp. 41–103. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2004.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0890540104000094.

[CP94] 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. Portland, Oregon, USA: Association for Computing Machinery, 1994, pp. 151–162. ISBN: 0897916360. DOI: 10.1145/174675.177844. URL: https://doi.org/10.1145/174675.177844.

[Gog94] Healfdene Goguen. “A Typed Operational Semantics for Type Theory”. University of Edinburgh, 1994.

[Hut09] DeLesley S. Hutchins. “Pure Subtype Systems: A Type Theory for Extensible Software”. University of Edinburgh, 2009.

References III

[LSX13] Zhaohui Luo, Sergey Soloviev, and Tao Xue. “Coercive subtyping: Theory and implementation”. In: *Information and Computation* 223 (2013), pp. 18–42. ISSN: 0890-5401. DOI: 10.1016/j.ic.2012.10.020. URL: https://www.sciencedirect.com/science/article/pii/S0890540112001757.

[Luo12] Zhaohui Luo. *Notes on Universes in Type Theory*. 2012. URL: https://www.cs.rhul.ac.uk/home/zhaohui/universes.pdf.

[Luo94] Zhaohui Luo. *Computation and Reasoning: A Type Theory for Computer Science*. London: Oxford University Press, Mar. 1994. ISBN: 9780198538356.

[Luo97] Zhaohui Luo. “Coercive subtyping in type theory”. In: *Computer Science Logic*. Ed. by Dirk van Dalen and Marc Bezem. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 275–296. ISBN: 978-3-540-69201-0.

References IV

[ML21] Harry Maclean and Zhaohui Luo. “Subtype Universes”. In: *26th International Conference on Types for Proofs and Programs (TYPES 2020)*. Ed. by Ugo de’Liguoro, Stefano Berardi, and Thorsten Altenkirch. Vol. 188. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 9:1–9:16. ISBN: 978-3-95977-182-5. DOI: 10.4230/LIPIcs.TYPES.2020.9. URL: https://drops.dagstuhl.de/opus/volltexte/2021/13888.

[Nor+05] Norell et al. *Agda Language Reference Documentation*. 2005. URL: https://agda.readthedocs.io.

[Pie92] Benjamin C. Pierce. “Bounded Quantification is Undecidable”. In: *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’92. Albuquerque, New Mexico, USA: Association for Computing Machinery, 1992, pp. 305–315. ISBN: 0897914538. DOI: 10.1145/143165.143228. URL: https://doi.org/10.1145/143165.143228.

References V

[ST14] Matthieu Sozeau and Nicolas Tabareau. “Universe Polymorphism in Coq”. In: *Interactive Theorem Proving*. Ed. by Gerwin Klein and Ruben Gamboa. Cham: Springer International Publishing, 2014, pp. 499–514. ISBN: 978-3-319-08970-6.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.