On the Metatheory of Subtype Universes

Felix Bradley Zhaohui Luo

Department of Computer Science
Royal Holloway, University of London

14th June 2023

Presentation Overview

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

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: $\Sigma(x : A).P(x) \leq A$

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: $\{\text{name} : \text{String}, \text{age} : \text{Nat}\} \leq \{\text{name} : \text{String}\}$

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 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: $f : \text{Float} \to \text{Float}, x : \text{Int16}, \text{Int16} \leq_c \text{Float} \vdash f(x) = f(c(x)) : \text{Float}$

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:
  $f : \text{Float} \rightarrow \text{Float}, x : \text{Int16}, \text{Int16} \leq_c \text{Float} \vdash f(x) = f(c(x)) : \text{Float}$

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].

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

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

What is a power type?

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

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

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

- 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

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

Metatheory of power types

- 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 power types

- 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.

- 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].

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

What is subtyping?

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

What is subtyping?

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

What is subtyping?

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:

- More complex compared to subsumptive subtyping
- Need two-step reduction - first insert coercions, then perform standard reduction
- Checking the coherency of subtyping rules is non-trivial

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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

- 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 ($\mathrm{UTT}[C]$) 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 $\mathrm{UTT}[C]$ [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$[\mathcal{C}]$.

Subtype Universes

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

Subtype Universes

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.

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.

Subtype Universes

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.

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.

Does this matter?

Subtype Universes

Example

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

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

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.

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]).

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. 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 contains a subtype universe $\mathcal{U}(B)$ on the LHS, and so we can't use this relation in Maclean and Luo's system.

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.

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.

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

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

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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Example (Subtype Universes on the RHS)

We define a type family of ordinals

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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Example (Subtype Universes on the RHS)

We define a type family of ordinals

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

and add in the subtyping rule allowing us to take any object of one of these types as a natural number.

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

Generalising Subtype Universes

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Example (Subtype Universes on the RHS)

We define a type family of ordinals

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

and add in the subtyping rule allowing us to take any object of one of these types as a natural number.

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

However, we may also want to interpret a natural number as one of these ordinal types. So we also introduce the subtyping relation

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

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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Definition (Level of a Type)

For a given context Γ, define $\mathcal{L}_\Gamma$ such that

- $\mathcal{L}_\Gamma(\mathbf{1}), \mathcal{L}_\Gamma(\text{Prop}), \mathcal{L}_\Gamma(\mathbb{N}), ... = 0$
- $\mathcal{L}_\Gamma(\Pi(x : A).B), \mathcal{L}_\Gamma(\Sigma(x : A).B), ... = \max_{x:A}\{\mathcal{L}_\Gamma(A), \mathcal{L}_\Gamma(B[x])\}$
- $\mathcal{L}_\Gamma(\mathcal{U}(B)) = \mathcal{L}_\Gamma(B) + 1$

Monotonic Subtyping

Definition (Level of a Type)

For a given context Γ, define $\mathcal{L}_\Gamma$ such that

- $\mathcal{L}_\Gamma(\mathbf{1}), \mathcal{L}_\Gamma(\mathrm{Prop}), \mathcal{L}_\Gamma(\mathbb{N}), \ldots = 0$
- $\mathcal{L}_\Gamma(\Pi(x : A).B), \mathcal{L}_\Gamma(\Sigma(x : A).B), \ldots = \max_{x:A}\{\mathcal{L}_\Gamma(A), \mathcal{L}_\Gamma(B[x])\}$
- $\mathcal{L}_\Gamma(\mathcal{U}(B)) = \mathcal{L}_\Gamma(B) + 1$

Definition (Monotonicity)

A subtyping judgement $A \leq B$ is *monotonic* if $\mathcal{L}_\Gamma(A) \leq \mathcal{L}_\Gamma(B)$.

Monotonic Subtyping

Definition (Level of a Type)

For a given context Γ, define $\mathcal{L}_\Gamma$ such that

- $\mathcal{L}_\Gamma(\mathbf{1}), \mathcal{L}_\Gamma(\text{Prop}), \mathcal{L}_\Gamma(\mathbb{N}), \ldots = 0$
- $\mathcal{L}_\Gamma(\Pi(x : A).B), \mathcal{L}_\Gamma(\Sigma(x : A).B), \ldots = \max_{x:A}\{\mathcal{L}_\Gamma(A), \mathcal{L}_\Gamma(B[x])\}$
- $\mathcal{L}_\Gamma(\mathcal{U}(B)) = \mathcal{L}_\Gamma(B) + 1$

Definition (Monotonicity)

A subtyping judgement $A \leq B$ is *monotonic* if $\mathcal{L}_\Gamma(A) \leq \mathcal{L}_\Gamma(B)$.

Metatheory for monotonic subtyping

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

Metatheory for monotonic subtyping

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

We construct an embedding $\delta : \tau \rightarrow \mathrm{UTT}[C]$ 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.

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

$$\delta(\langle A, c \rangle) \stackrel{\text{def}}{=} (\mathbf{n}_{\mathcal{L}_{\Gamma}(B)}(\delta(A)), \delta(c))$$

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:*

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

Lemma

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

Lemma

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

- $\delta(\Gamma) \vdash \delta(A)$: **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 typable if and only if $A \leq B$ is derivable, e.g. $\lambda(f : B \to \mathbb{N}).\lambda(a : A).f(a)$

Metatheory for nonmonotonic subtyping

How should we go about nonmonotonic subtyping?

How should we go about nonmonotonic subtyping?

Our first idea:

Metatheory for nonmonotonic subtyping

How should we go about nonmonotonic subtyping?

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

Metatheory for nonmonotonic subtyping

How should we go about nonmonotonic subtyping?

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

In systems without subtype universes, extending with new subtyping rules is conservative.

Metatheory for nonmonotonic subtyping

How should we go about nonmonotonic subtyping?

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

In systems without subtype universes, extending with new subtyping rules is conservative.

Traditional proofs for coercive subtyping formalise the two-step reduction - insert coercions, then reduce as normal.

Metatheory for nonmonotonic subtyping

How should we go about nonmonotonic subtyping?

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

In systems without subtype universes, extending with new subtyping rules is conservative.

Traditional proofs for coercive subtyping formalise the two-step reduction - insert coercions, then reduce as normal.

Every subtype universe $\mathcal{U}(B)$ contains the term $\langle B, \mathrm{id} \rangle$.

Every subtype universe $\mathcal{U}(B)$ contains the term $\langle B, \mathrm{id} \rangle$.

Solution idea: Modify the reduction method to also send each $\langle A, c \rangle : \mathcal{U}(B)$ to $\langle B, \mathrm{id}_B \rangle : \mathcal{U}(B)$.

Every subtype universe $\mathcal{U}(B)$ contains the term $\langle B, \mathrm{id} \rangle$.

Solution idea: Modify the reduction method to also send each $\langle A, c \rangle : \mathcal{U}(B)$ to $\langle B, \mathrm{id}_B \rangle : \mathcal{U}(B)$.

- Terms depending on types, e.g. $f : \Pi(x : \mathcal{U}(B)).M$

Metatheory for nonmonotonic subtyping

Every subtype universe $\mathcal{U}(B)$ contains the term $\langle B, \mathrm{id} \rangle$.

Solution idea: Modify the reduction method to also send each $\langle A, c \rangle : \mathcal{U}(B)$ to $\langle B, \mathrm{id}_B \rangle : \mathcal{U}(B)$.

- Terms depending on types, e.g. $f : \Pi(x : \mathcal{U}(B)).M$

✓ OK because $f(\langle B, \mathrm{id}_B \rangle)$ is defined

Metatheory for nonmonotonic subtyping

Every subtype universe $\mathcal{U}(B)$ contains the term $\langle B, \mathrm{id}\rangle$.

Solution idea: Modify the reduction method to also send each $\langle A, c\rangle : \mathcal{U}(B)$ to $\langle B, \mathrm{id}_B\rangle : \mathcal{U}(B)$.

- Terms depending on types, e.g. $f : \Pi(x : \mathcal{U}(B)).M$

✓ OK because $f(\langle B, \mathrm{id}_B\rangle)$ is defined

- Types depending on terms, e.g. $\Sigma(x : \mathcal{U}).(x = \langle A, c\rangle)$

Metatheory for nonmonotonic subtyping

Every subtype universe $\mathcal{U}(B)$ contains the term $\langle B, \mathrm{id}\rangle$.

Solution idea: Modify the reduction method to also send each $\langle A, c\rangle : \mathcal{U}(B)$ to $\langle B, \mathrm{id}_B\rangle : \mathcal{U}(B)$.

- Terms depending on types, e.g. $f : \Pi(x : \mathcal{U}(B)).M$
- ✓ OK because $f(\langle B, \mathrm{id}_B\rangle)$ is defined
- Types depending on terms, e.g. $\Sigma(x : \mathcal{U}).(x = \langle A, c\rangle)$
  Uhh...

What if we only look at subtyping relations where there is a bound on how 'problematic' it is?

Metatheory for k-monotonic subtyping

ROYAL HOLLOWAY UNIVERSITY OF LONDON

What if we only look at subtyping relations where there is a bound on how 'problematic' it is?

Definition (k-Monotonicity)

A subtyping rule $\Gamma \vdash A \leq B$ is *k-monotonic* if $\forall \Gamma, \mathcal{L}_\Gamma(B) - \mathcal{L}_\Gamma(A) \geq -k$.

Moreover, a set of subtyping rules $\mathcal{C}$ is k-monotonic if every rule $R \in \mathcal{C}$ is k-monotonic.

Metatheory for k-monotonic subtyping

What if we only look at subtyping relations where there is a bound on how 'problematic' it is?

Definition (k-Monotonicity)

A subtyping rule $\Gamma \vdash A \leq B$ is *k-monotonic* if $\forall \Gamma, \mathcal{L}_\Gamma(B) - \mathcal{L}_\Gamma(A) \geq -k$.

Moreover, a set of subtyping rules $\mathcal{C}$ is k-monotonic if every rule $R \in \mathcal{C}$ is k-monotonic.

Remark

A subtyping rule is monotonic iff it is 0*-monotonic.*

Metatheory for k-monotonic subtyping

Example

For a given type B, recall the type of *pointed subtypes of B*

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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Example

For a given type B, recall the type of *pointed subtypes of B*

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

Note that $\forall\Gamma$,

$$\mathcal{L}_\Gamma(B) - \mathcal{L}_\Gamma(\Sigma(x : \mathcal{U}(B)).\sigma_1(x)) = \min(-1, -\mathcal{L}_\Gamma(\sigma_1(x)))$$

Example

For a given type B, recall the type of *pointed subtypes of B*

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

Note that $\forall\Gamma$,

$$\mathcal{L}_\Gamma(B) - \mathcal{L}_\Gamma(\Sigma(x : \mathcal{U}(B)).\sigma_1(x)) = \min(-1, -\mathcal{L}_\Gamma(\sigma_1(x)))$$

and so as long as we choose the subtype relations involving B on the RHS sensibly, we can have some k such that this subtyping relation is k-monotonic.

Metatheory for *k*-monotonic subtyping

We modify our embedding $\delta : \tau \longrightarrow \mathrm{UTT}[\mathcal{C}]$ to be correct when $\mathcal{C}$ is *k*-monotonic.

We modify our embedding $\delta : \tau \longrightarrow \mathrm{UTT}[\mathcal{C}]$ to be correct when $\mathcal{C}$ is k-monotonic.

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

$$\delta(\langle A, c\rangle) \stackrel{\text{def}}{=} (\mathbf{n}_{k+\mathcal{L}_\Gamma(B)}(\delta(A)), \delta(c))$$

Theorem (Logical Consistency for k-monotonic Subtyping)

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

Metatheory for k-monotonic subtyping

We modify our embedding $\delta : \tau \to \mathrm{UTT}[\mathcal{C}]$ to be correct when $\mathcal{C}$ is k-monotonic.

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

$$\delta(\langle A, c\rangle) \stackrel{\text{def}}{=} (\mathbf{n}_{k+\mathcal{L}_\Gamma(B)}(\delta(A)), \delta(c))$$

Theorem (Logical Consistency for k-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 k-monotonic Subtyping)

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

Conclusion

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 k-monotonic subtyping

Summary:

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

Open questions/future work:

- Results for non-monotonic subtyping?
- What does subtyping mean between propositions?

Conclusion

Summary:

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

Open questions/future work:

- Results for non-monotonic subtyping?
- 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 k-monotonic subtyping
- Decidability of type-checking, minimal types, and the subtype relation

Open questions/future work:

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

Thank you for listening!

References I

[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

ROYAL HOLLOWAY UNIVERSITY OF LONDON

[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.

[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.