Unifying Typing and Subtyping

Yanpeng Yang   Bruno C. d. S. Oliveira

Department of Computer Science,
The University of Hong Kong

OOPSLA 2017
October 25, 2017

Motivation

Dependent types are gaining popularity: Idris (2013), Agda (2007), Coq, etc.

- Expressiveness of type system
- Economy of concept: less syntax, rules, meta-theory $\Rightarrow$ **our focus**

Theoretical basis of OO is becoming complex:

- Generics in *Java 5* (2004): System $F_{\leqslant}/F_{\omega}^{\leqslant}$
- Path-dependent types in *Scala* (2004): Dependent Object Types (DOT, Amin et al., 2012)

> Bring the advantages of dependent types to object-oriented programming?

Challenges

One thing that makes the study of these systems difficult is that **with dependent types, the typing and subtyping relations become intimately tangled**, *which means that tested techniques of examining subtyping in isolation no longer apply.*

— Aspinall and Compagnoni (1996),
Subtyping Dependent Types

Challenges (cont.)

- Mutual dependency of typing and subtyping:

Subsumption rule

$$\frac{\Gamma \vdash e : A \qquad \Gamma \vdash A \leqslant B}{\Gamma \vdash e : B}$$

Subtyping of top type

$$\frac{\Gamma \vdash A : \star}{\Gamma \vdash A \leqslant \top}$$

- Duplication of concepts:

Abstractions in System $F_\omega^\leqslant$

Term abstraction	$\lambda x : A.\ e$
Type abstraction	$\lambda X \leqslant A.\ e$
Operator abstraction	$\lambda X \leqslant A.\ \boldsymbol{B}$

Previous Attempts

- Try to carefully **untangle** typing and subtyping
- More restricted and **lose expressiveness** w.r.t. System $F_{\leqslant}$:

Calculus	Dep. Types	Contravariance	Bounded Quan.	$\top$-type
$F_{\leqslant}$	×	✓	✓	✓
$PTS^{\leqslant}$ (1999)	✓	✓	✓	×
PSS (2010)	✓	×	✓	✓
$\lambda P_{\leqslant}$ (1996)	✓	✓	×	×
$\lambda \Pi_{\leqslant}$ (2001)	✓	✓	×	×

Contribution

Instead of trying to untangle, we embrace tangling!
Key idea: unify typing and subtyping into one relation

- Unified subtyping and $\lambda I_{\leqslant}$ calculus:
  - A single relation for both subtyping and typing
  - Dependent types with unified syntax
  - Fully subsume System $F_{\leqslant}$
- Mechanized metatheory in Coq
- Object encodings in $\lambda I_{\leqslant}$ calculus

Unified Subtyping

- Unified subtyping combines typing and subtyping:

$$\begin{tikzcd}[column sep=small]
\Gamma \vdash e : A \arrow[dr] & & \Gamma \vdash e_1 \leqslant e_2 \arrow[dl] \\
& \Gamma \vdash e_1 \leqslant e_2 : A &
\end{tikzcd}$$

Explanation: e_1 is a subtype of e_2 and both of them have type A.

- Typing and type well-formedness are now **syntactic sugar**:

$$\Gamma \vdash \ e : A \triangleq \Gamma \vdash \ e \leqslant e : A$$

$$\Gamma \vdash \ A : \star \triangleq \Gamma \vdash \ A \leqslant A : \star$$

Unified Subtyping (Cont.)

Simplify the combination of typing and subtyping:

- No mutual dependency, trivially:

Subsumption rule

$$\frac{\Gamma \vdash e \leqslant e : A \qquad \Gamma \vdash A \leqslant B : \star}{\Gamma \vdash e \leqslant e : B}$$

Subtyping of top type

$$\frac{\Gamma \vdash A \leqslant A : \star}{\Gamma \vdash A \leqslant \top : \star}$$

- Only one single form of abstraction $\lambda x \leqslant e_1 : A.\ e_2$:

$$\lambda x : A.\ e \triangleq \lambda x \boxed{\leqslant \top} : A.\ e \quad (*)$$

$$\lambda x \leqslant A.\ B \triangleq \lambda x \leqslant A \boxed{:\star}.\ B$$

**Note: we generalize $\top$ to be any type A.*

Dependent Types with Casts

- Traditional approach: mix up **conversion rule** and **subsumption rule**

Subsumption

$$\frac{\Gamma \vdash e : A \qquad \boxed{\Gamma \vdash A \leqslant B}}{\Gamma \vdash e : B}$$

$\xrightarrow{\text{subsumes}}$

Conversion

$$\frac{\Gamma \vdash e : A \qquad \boxed{A =_\beta B}}{\Gamma \vdash e : B}$$

  - *Consequence*: **strong normalization** is entangled with other proofs (transivitity, decidable type checking, etc.)

- Our approach: replace conversion rule with **explicit type casts**
  - Well-studied approach to *decouple* strong normalization (Sjöberg et al.; Stump et al.; Yang et al.; Casinghino et al.)
  - Only up to alpha equality, inconvenience for *heavy* type conversion
  - Still sufficient for object encodings

Summary

To summarize, we simplify the design by two means:

- **Unified subtyping**: avoid circularity
- **Explicit type casts**: decouple strong normalization

Is it oversimplified?

No, still expressive enough to encode objects that require **high-order subtyping**.

Object Encodings

- **Existential object encodings** by Pierce and Turner (1994):

$$\underset{}{Obj =} \underset{\textit{interface}}{\lambda I : \boxed{\star \to \star}.} \quad \underset{\textit{state type}}{\exists X.} \quad \underset{\textit{state}}{X} \times \underset{\textit{methods}}{(X \to I\,X)}$$

- Dependency: Church encoding of (weak) dependent sum types:

$$\Sigma x : A.\ B \triangleq \Pi z : \star.\ (\Pi x : A.\ B \to z) \to z$$

Useful to encode OO concepts, e.g. type member:

Abstract Set Interface (Scala)

```
trait Set {
  type T
  def empty(): T
  def member(x: Int, s: T): Boolean
  def insert(x: Int, s: T): T
}
```

Encoding in $\lambda I_{\leqslant}$

$$\begin{aligned} Set = \Sigma T : \star.\ \{ \\ & empty : T, \\ & member : Int \to T \to Bool, \\ & insert : Int \to T \to T\} \end{aligned}$$

Proof Strategy

High-level view of methodology:

- Two main targets: **transitivity** and **type safety**
- Figure out the correct **form** and **proof order** of lemmas (Tricky!)
- Use standard induction techniques to finish proofs

Circularity of Lemmas

- One normally tries to first prove the fundamental reflexivity lemma (also called regularity):

Lemma (Reflexivity)

If $\Gamma \vdash e_1 \leqslant e_2 : A$, *then both* $\Gamma \vdash e_1 : A$ *and* $\Gamma \vdash e_2 : A$ *hold.*

- However, it is mutually dependent on correctness of types:

Lemma (Correctness of Types)

If $\Gamma \vdash e_1 \leqslant e_2 : A$, *then* $\Gamma \vdash A : \star$.

$$
\begin{array}{ccccc}
\textbf{Reflexivity} & \longleftarrow & \text{Narrowing} & \longleftarrow & \text{Transitivity} \\
\updownarrow & & & & \uparrow \\
\text{Correctness of Types} & & \longrightarrow & & \text{Substitution}
\end{array}
$$

Break the Circularity

Observation: correctness of types only depends on **right** reflexivity. We divide reflexivity into two sub-lemmas:

- *Left reflexivity*: $\Gamma \vdash\ e_1 \leqslant e_2 : A \Rightarrow \Gamma \vdash\ e_1 : A$
- *Right reflexivity*: $\Gamma \vdash\ e_1 \leqslant e_2 : A \Rightarrow \Gamma \vdash\ e_2 : A$

$$\begin{array}{ccccc}
\text{Left Reflexivity} & \longleftarrow & \text{Narrowing} & \longleftarrow & \text{Transitivity} \\
\textbf{Right Reflexivity} & & & & \uparrow \\
\updownarrow & & & & \\
\text{Correctness of Types} & & \longrightarrow & & \text{Substitution}
\end{array}$$

Related Work

Subtyping w/ traditional dependent types:

- Summary of calculi:

Name	D.T.	β-eq.	Unified Syn.	Contravar.	B. Quan.	$\top$-type
$F_{\leqslant}$	×	×	×	✓	✓	✓
$\lambda I_{\leqslant}$	✓	×	✓	✓	✓	✓
$PTS^{\leqslant}$ (1999)	✓	✓	✓	✓	✓*	×
PSS (2010)	✓	✓	✓	×	✓	✓
$\lambda P_{\leqslant}$ (1996)	✓	✓	×	✓	×	×
$\lambda \Pi_{\leqslant}$ (2001)	✓	✓	×	✓	×	×

**No subtyping on bounded quantified abstractions.*

- Pure Subtype Systems (Hutchins, 2010):
  - Purely based on subtyping; no concept of typing
  - Function and function type are merged; no contravariance
  - Metatheory proof is complex and incomplete

Related Work (Cont.)

Subtyping w/ *restricted* dependent types: path-dependent types (Amin et al., 2014)

- DOT (Amin et al., 2012): theoretic foundation for Scala
- Both lower and upper bounded quantification; subsume System $F_{\leqslant}$
- Complex soundness proof (Rompf and Amin, 2016); no transitivity elimination
- Compared to $\lambda I_{\leqslant}$: non-unified **stratified** syntax; non-traditional dependent types based on **path selection**

More in the Paper...

- **Object encodings**: generic message passing; cell object encoding and evaluation
- **Formal presentation of $\lambda I_{\leqslant}$**: syntax, static and dynamic semantics
- **Proof details**: generalization of transitivity and type preservation; Coq scripts available online
- **Subsumption of $F_{\leqslant}$**: translation to $F_{\leqslant}$ and manual proofs
- **Discussion of variants**: bidirectional system, recursion, call-by-value reduction, full reduction, full contravariance, etc.

Conclusion and Future Work

- **Unified subtyping**: simplify combining dependent types and subtyping
- **The $\lambda I_{\leqslant}$ calculus**: application of unified subtyping with well-developed metatheory
- **Future work**: decidability; different variants; extended with lower bounds like DOT

Thanks for listening!

Any questions?

References (1)

Nada Amin, Adriaan Moors, and Martin Odersky. 2012. Dependent object types. In *FOOL '12*. ACM.

Nada Amin, Tiark Rompf, and Martin Odersky. 2014. Foundations of path-dependent types. In *OOPSLA '14*. ACM, 233–249.

David Aspinall and Adriana Compagnoni. 1996. Subtyping dependent types. In *LICS '96*. 86–97.

Edwin Brady. 2013. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming* 23, 05 (2013), 552–593.

Chris Casinghino, Vilhelm Sjöberg, and Stephanie Weirich. 2014. Combining Proofs and Programs in a Dependently Typed Language. In *POPL '14*. ACM, 33–45.

Giuseppe Castagna and Gang Chen. 2001. Dependent types with subtyping and late-bound overloading. *Information and Computation* 168, 1 (2001), 1–67.

References (2)

DeLesley S. Hutchins. 2010. Pure Subtype Systems. In *POPL '10*. ACM, 287–298.

Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory*. Ph.D. Dissertation. Chalmers University of Technology.

Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. 2004. *An Overview of the Scala Programming Language*. Technical Report IC/2004/64. EPFL Lausanne, Switzerland.

Benjamin C. Pierce and David N. Turner. 1994. Simple type-theoretic foundations for object-oriented programming. *Journal of functional programming* 4, 02 (1994), 207–247.

Tiark Rompf and Nada Amin. 2016. Type soundness for dependent object types (DOT). In *OOPSLA '16*. ACM, 624–641.

References (3)

Vilhelm Sjöberg, Chris Casinghino, Ki Yung Ahn, Nathan Collins, Harley D. Eades III, Peng Fu, Garrin Kimmell, Tim Sheard, Aaron Stump, and Stephanie Weirich. 2012. Irrelevance, Heterogenous Equality, and Call-by-value Dependent Type Systems. In *MSFP '12*. 112–162.

Aaron Stump, Morgan Deters, Adam Petcher, Todd Schiller, and Timothy Simpson. 2008. Verified Programming in Guru. In *PLPV '09*. 49–58.

Yanpeng Yang, Xuan Bi, and Bruno C. d. S. Oliveira. 2016. Unified Syntax with Iso-types. In *APLAS '16*. Springer, 251–270.

Jan Zwanenburg. 1999. Pure type systems with subtyping. In *TLCA '99*. 381–396.