Decidable Bounded Quantification

Detailed Project Plan

YANG Mingtian

1 October 2024

Contents

1. Background

In the field of programming language theory, a type system is a formal system that categorises terms into specific types. A robust type system plays a crucial role in preventing certain undesirable program behaviours, thus enabling early detection of programming errors.

Subtyping and parametric polymorphism are key features in modern programming languages. By incorporating bounded quantification, which combines both features, programming languages can achieve greater expressive capabilities.

Nevertheless, the inclusion of bounded quantification can lead to increased metatheoretical complexity within the type system [10]. For instance, the standard calculus for bounded quantification, System $\mathrm{F}_{<:}$ [2], [5], pronounced 'F-sub', has been proved to have an undecidable subtype relation. That is, it has no subtyping algorithm that terminates on all inputs [6].

$$\frac{}{\Gamma \vdash \sigma <: \top}\,\textsc{Top} \qquad \frac{}{\Gamma \vdash \alpha <: \alpha}\,\textsc{Refl} \qquad \frac{\Gamma \vdash \Gamma(\alpha) <: \tau}{\Gamma \vdash \alpha <: \tau}\,\textsc{Var}$$

$$\frac{\Gamma \vdash \tau_1 <: \sigma_1 \quad \Gamma \vdash \sigma_2 <: \tau_2}{\Gamma \vdash \sigma_1 \to \sigma_2 <: \tau_1 \to \tau_2}\,\textsc{Arrow} \qquad \frac{\Gamma \vdash \tau_1 <: \sigma_1 \quad \Gamma, \alpha <: \tau_1 \vdash \sigma_2 <: \tau_2}{\Gamma \vdash \forall \alpha <: \sigma_1 . \sigma_2 <: \forall \alpha <: \tau_1 . \tau_2}\,\textsc{All}$$

The root of the problem lies in the ALL rule, in which the bound on instances of α occuring in σ_2 changes from σ_1 to τ_1.

Various proposals have been made. One of them is Kernel $\mathrm{F}_{<:}$ [2], in which the subtype order is restricted to those which have equal bounds:

$$\frac{\Gamma, \alpha <: \sigma_1 \vdash \sigma_2 <: \tau_2}{\Gamma \vdash \forall \alpha <: \sigma_1 . \sigma_2 <: \forall \alpha <: \sigma_1 . \tau_2}\,\textsc{All-K}.$$

Although this revision guarantees algorithmic decidability, it severely compromises the expressiveness of the language, and thus lacks practicality.

Another decidable variant of System $\mathrm{F}_{<:}$ is System $\mathrm{F}^{\top}_{<:}$ [3], in which the type information of bounds is ignored when inferring on the bodies:

$$\frac{\Gamma \vdash \tau_1 <: \sigma_1 \quad \Gamma, \alpha <: \top \vdash \sigma_2 <: \tau_2}{\Gamma \vdash \forall \alpha <: \sigma_1 . \sigma_2 <: \forall \alpha <: \tau_1 . \tau_2}\,\textsc{All-T}.$$

Although this yields an expressive subtyping relation, it does not interact nicely with the typing rules [8], and thus is still not an ideal type system.

2. Objective

The main focus of this project is to investigate the design of a mechanism for bounded quantification that ensures its decidability. This involves proposing new subtyping rules and using a interactive theorem prover to verify their desirable properties, such as reflexivity, transitivity, and decidability. Subsequently, a toy programming language incorporating the revised type system will be implemented. The ultimate goal is to improve the usability of programming languages and increase the productivity of programmers.

3. Methodology

We will first examine an existing type system, completely bounded quantification [7], which is a decidable variant of $F_{<:}$ such that it prohibits $\top$ from appearing in the bounds. Although it is even more restrictive than Kernel $F_{<:}$ and loses more expressiveness, it is a suitable starting point due to its simplicity.

On the basis of completely bounded quantification, we will propose a new variant of the type system, formalise it, prove it properties, and finally implement a toy programming language.

3.1. Formalisation and Proof

Coq [4] will be used as the interactive theorem prover in which we will formalise the type system and prove its properties. Specifically, we will use the Metalib [1], a Coq library, in which a novel formalisation style that combines locally nameless representation of terms and cofinite quantification of free variable names in inductive definitions of relations on terms is proposed.

The theorems we are going to prove can be formalised in Coq as follows:

- Reflexivity

```
Theorem sub_reflexivity :
  forall E T,
  wf_env E ->
  wf_typ E T ->
  sub E T T.
```

- Transitivity

```
Theorem sub_transitivity :
  forall E, wf_env E ->
  forall Q, wf_typ E Q ->
  forall S, wf_typ E S -> sub E S Q ->
  forall T, wf_typ E T -> sub E Q T ->
  sub E S T.
```

- Decidability

```
Theorem sub_decidability :
  forall E S T,
  wf_env E ->
  wf_typ E S ->
  wf_typ E T ->
  sub E S T \/ ~ sub E S T.
```

The proofs for reflexivity and transitivity are fairly standard and straightforward. To prove decidability, we will first define a complexity metric which is finite and positive for each subtyping problem,

and show that the application of inference rules causes the complexity of the new subtyping problems to decrease. For completely bounded quantification, we will follow the original definition in [7]:

$$
\begin{aligned}
\mathit{complexity}(\Gamma \vdash \sigma <: \tau) &\stackrel{\text{def}}{=} \mathit{size}(\sigma)_{\Gamma} + \mathit{size}(\tau)_{\Gamma} \\
\mathit{size}(\top)_{\Gamma} &= 1 \\
\mathit{size}(\alpha)_{\Gamma} &= \begin{cases} \mathit{size}(\Gamma(\alpha))_{\Gamma} & \text{if } \Gamma(\alpha) \text{ is defined} \\ 1 & \text{otherwise} \end{cases} \\
\mathit{size}(\tau_1 \to \tau_2)_{\Gamma} &= \mathit{size}(\tau_1)_{\Gamma} + \mathit{size}(\tau_2)_{\Gamma} \\
\mathit{size}(\forall \alpha <: \tau_1 . \tau_2)_{\Gamma} &= \mathit{size}(\tau_1)_{\Gamma} + \mathit{size}(\tau_2)_{\Gamma, \alpha <: \tau_1}
\end{aligned}
$$

Afterwards, we can prove the decidability by induction on complexity.

3.2. Revision of the Type System

The approach to revise the type system is inspired by the recent work of [11], where they extended Kernel $F_{<:}$ with iso-recursive types and subtyping using the nominal unfolding rules.

The rules are one of the proposed variants of rules for subtyping recursive types introduced by [12]. In nominal unfolding rules, an extra label with the recursive variable name is added, enabling us to track the name of the recursive variables during double unfolding to avoid accidental subtyping. The rules have been proven to be type sound and have the same level of expressiveness as the iso-recursive Amber rules, and they are easier to work with formally and can be seamlessly integrated into an existing calculus in a modular manner.

Based on these factors, we believe that the nominal unfolding rules are a powerful and appropriate tool for addressing our research topic. Therefore, we will incorporate them as part of our solution.

3.3. Lanugage Implementation

We will use OCaml [9] to implement the toy programming language. OCaml is an industrial-strength functional programming language with an emphasis on expressiveness and safety. It is widely used in the implementation of programming languages, such as Coq, ReScript, and MoonBit.

Since the main purpose of this project is the design and implementation of a type system, we will implement an interpreter instead of a compiler. In other words, we will not generate any low-level code.

On the other hand, since our project focuses on the semantics (type system) rather than the syntax, we will not spend too much energy on the frontend. Therefore, we will use OCamllex and Menhir to generate the lexer and parser.

4. Schedule and Milestones

- **Research Internship** (Jun – Aug 2024)
  - Literature review
  - Formalisation of completely bounded quantification
  - Proof of the key properties of its subtype relation:
    - Reflexivity
    - Transitivity
    - Decidability
- **Phase 1** (Sept 2024)
  - Detailed project plan
  - Project web page
- **Phase 2** (Oct 2024 – Jan 2025)

- Investigation of the nominal unfolding rules
- Formalisation of the revised type system
- Proof of the above-mentioned key properties
- Interim report

- **Phase 3** (Feb – Apr 2025)
  - Interpreter
  - Final report

Bibliography

[1] Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. 2008. Engineering formal metatheory. *SIGPLAN Not.* 43, 1 (January 2008), 3–15. https://doi.org/10.1145/1328897.1328443

[2] Luca Cardelli and Peter Wegner. 1985. On understanding types, data abstraction, and polymorphism. *Computing Surveys* 17, 4 (1985).

[3] Giuseppe Castagna and Benjamin C. Pierce. 1994. Decidable bounded quantification. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '94)*, 1994. Association for Computing Machinery, Portland, Oregon, USA, 151–162. https://doi.org/10.1145/174675.177844

[4] The Coq Development Team. 2024. The Coq Reference Manual – Release 8.19.0.

[5] Pierre-Louis Curien and Giorgio Ghelli. 1992. Coherence of subsumption, minimum typing and type-checking in $F_{\leq}$. *Mathematical Structures in Computer Science* 2, 1 (1992), 55–91.

[6] Pierre-Louis Curien and Giorgio Ghelli. 1992. Bounded quantification is undecidable. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '92)*, 1992.

[7] Dinesh Katiyar and Sriram Sankar. 1992. Completely Bounded Quantification is Decidable. In *Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*, 1992. 68–77.

[8] James Laird. 2023. Revisiting Decidable Bounded Quantification, via Dinaturality. *Electronic Notes in Theoretical Informatics and Computer Science* (February 2023). https://doi.org/10.46298/entics.10474

[9] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The OCaml system: Documentation and user's manual. *INRIA* 3, 42–43.

[10] Benjamin C. Pierce. 2002. *Types and Programming Languages*. The MIT Press.

[11] Litao Zhou, Yaoda Zhou, and Bruno C. d. S. Oliveira. 2023. Recursive Subtyping for All. *Proc. ACM Program. Lang.* 7, POPL (January 2023). https://doi.org/10.1145/3571241

[12] Yaoda Zhou, Jinxu Zhao, and Bruno C. d. S. Oliveira. 2022. Revisiting Iso-Recursive Subtyping. *ACM Trans. Program. Lang. Syst.* 44, 4 (September 2022). https://doi.org/10.1145/3549537