Higher-rank Polymorphism: Type Inference and Extensions

by

Ningning Xie
(谢宁宁)

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
at The University of Hong Kong

February 2021

Abstract of thesis entitled
"Higher-rank Polymorphism: Type Inference and Extensions"

Submitted by
Ningning Xie

for the degree of Doctor of Philosophy
at The University of Hong Kong
in February 2021

Declaration

I declare that this thesis represents my own work, except where due acknowledgment is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualifications.

. . . .
Ningning Xie
February 2021

Acknowledgments

Contents

List of Figures

List of Tables

Part I

Prologue

1 Introduction

mention that in this thesis when we say “higher-rank polymorphism” we mean “predicative implicit higher-rank polymorphism”.

1.1 Contributions

In summary the contributions of this thesis are:

Part II

- Chapter 3 proposes a new design for type inference of higher-rank polymorphism.
  - We design a variant of bi-directional type checking where the inference mode is combined with a new, so-called, application mode. The application mode naturally propagates type information from arguments to the functions.
  - With the application mode, we give a new design for type inference of higher-rank polymorphism, which generalizes the HM type system, supports a polymorphic let as syntactic sugar, and infers higher rank types. We present a syntax-directed specification, an elaboration semantics to System F, and an algorithmic type system with completeness and soundness proofs.
- Chapter 4 presents a new approach for implementing unification.
  - We propose a process named promotion, which, given a unification variable and a type, promotes the type so that all unification variables in the type are well-typed with regard to the unification variable.
  - We apply promotion in a new implementation of the unification procedure in higher-rank polymorphism, and show that the new implementation is sound and complete.

Part III

- Chapter 5 extends higher-rank polymorphism with gradual types.
  - We define a framework for consistent subtyping with

* a new definition of consistent subtyping that subsumes and generalizes that of Siek and Taha [2007] and can deal with polymorphism and top types;

* and a syntax-directed version of consistent subtyping that is sound and complete with respect to our definition of consistent subtyping, but still guesses instantiations.

- Based on consistent subtyping, we present he calculus GPC. We prove that our calculus satisfies the static aspects of the refined criteria for gradual typing [Siek et al. 2015], and is type-safe by a type-directed translation to λB [Ahmed et al. 2009].

- We present a sound and complete bidirectional algorithm for implementing the declarative system based on the design principle of Garcia and Cimini [2015].

- Chapter 6 further explores the design of promotion in the context of kind inference for datatypes.

  - We formalize Haskell98's datatype declarations, providing both a declarative specification and syntax-driven algorithm for kind inference. We prove that the algorithm is sound and observe how Haskell98's technique of defaulting unconstrained kinds to $\star$ leads to incompleteness. We believe that ours is the first formalization of this aspect of Haskell98.

  - We then present a type and kind language that is unified and dependently typed, modeling the challenging features for kind inference in modern Haskell. We include both a declarative specification and a syntax-driven algorithm. The algorithm is proved sound, and we observe where and why completeness fails. In the design of our algorithm, we must choose between completeness and termination; we favor termination but conjecture that an alternative design would regain completeness. Unlike other dependently typed languages, we retain the ability to infer top-level kinds instead of relying on compulsory annotations.

Many metatheory in the paper comes with Coq proofs, including type safety, coherence, etc.[1]

[1] For convenience, whenever possible, definitions, lemmas and theorems have hyperlinks (click ☞) to their Coq counterparts.

1.2 Organization

This thesis is largely based on the publications by the author [Xie et al. 2018, 2019a,b; Xie and Oliveira 2017, 2018], as indicated below.

Chapter 3: Ningning Xie and Bruno C. d. S. Oliveira. 2018. "Let Arguments Go First". In European Symposium on Programming (ESOP).

Chapter 4: Ningning Xie and Bruno C. d. S. Oliveira. 2017. "Towards Unification for Dependent Types" (Extended abstract), In Draft Proceedings of Trends in Functional Programming (TFP).

Chapter 5: Ningning Xie, Xuan Bi, and Bruno C. d. S. Oliveira. 2018. "Consistent Subtyping for All". In European Symposium on Programming (ESOP).

Ningning Xie, Xuan Bi, Bruno C. d. S. Oliveira, and Tom Schrijvers. 2019. "Consistent Subtyping for All". In ACM Transactions on Programming Languages and Systems (TOPLAS).

Chapter 6: Ningning Xie, Richard Eisenberg and Bruno C. d. S. Oliveira. 2020. "Kind Inference for Datatypes". In Symposium on Principles of Programming Languages (POPL).

2 Background

2.1 The Odersky-Läufer Type System

2.2 Algorithmic Bidirectional Type System

Types	A, B	$::=$	$\mathsf{Int} \mid a \mid A \to B \mid \forall a.\, A$
Monotypes	τ, σ	$::=$	$\mathsf{Int} \mid a \mid \tau \to \sigma$
Terms	e	$::=$	$x \mid n \mid \lambda x : A.\, e \mid \lambda x.\, e \mid e_1\, e_2 \mid \mathsf{let}\, x = e_1\, \mathsf{in}\, e_2$
Contexts	Ψ	$::=$	$\bullet \mid \Psi, x : A \mid \Psi, a$

$\boxed{\Psi \vdash^{OL} e : A}$ (Typing)

$$\frac{(x : A) \in \Psi}{\Psi \vdash^{OL} x : A}\ \textsc{u-var} \qquad \frac{}{\Psi \vdash^{OL} n : \mathsf{Int}}\ \textsc{u-int} \qquad \frac{\Psi, x : A \vdash^{OL} e : B}{\Psi \vdash^{OL} \lambda x : A.\, e : A \to B}\ \textsc{u-lamann} \qquad \frac{\Psi, x : \tau \vdash^{OL} e : B}{\Psi \vdash^{OL} \lambda x.\, e : \tau \to B}\ \textsc{u-lam}$$

$$\frac{\Psi \vdash^{OL} e_1 : A_1 \to A_2 \qquad \Psi \vdash^{OL} e_2 : A_1}{\Psi \vdash^{OL} e_1\, e_2 : A_2}\ \textsc{u-app} \qquad \frac{\Psi \vdash^{OL} e : A_1 \qquad \Psi \vdash A_1 <: A_2}{\Psi \vdash^{OL} e : A_2}\ \textsc{u-sub}$$

$$\frac{\Psi, a \vdash^{OL} e : A}{\Psi \vdash^{OL} e : \forall a.\, A}\ \textsc{u-gen} \qquad \frac{\Psi \vdash^{OL} e_1 : A \qquad \Psi, x : A \vdash^{OL} e_2 : B}{\Psi \vdash^{OL} \mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 : B}\ \textsc{u-let}$$

$\boxed{\Psi \vdash A <: B}$ (Subtyping)

$$\frac{a \in \Psi}{\Psi \vdash a <: a}\ \textsc{s-tvar} \qquad \frac{}{\Psi \vdash \mathsf{Int} <: \mathsf{Int}}\ \textsc{s-int} \qquad \frac{\Psi \vdash B_1 <: A_1 \qquad \Psi \vdash A_2 <: B_2}{\Psi \vdash A_1 \to A_2 <: B_1 \to B_2}\ \textsc{s-arrow}$$

$$\frac{\Psi \vdash \tau \qquad \Psi \vdash A[a \mapsto \tau] <: B}{\Psi \vdash \forall a.\, A <: B}\ \textsc{s-forallL} \qquad \frac{\Psi, a \vdash A <: B}{\Psi \vdash A <: \forall a.\, B}\ \textsc{s-forallR}$$

Figure 2.1: Syntax and static semantics of the Odersky-Läufer type system.

Part II

Type Inference

3 Type Inference With The Application Mode

4 Unification with Promotion

Part III

Extensions

5 Higher Rank Gradual Types

6 Dependent Types

Part IV

Related and Future Work

7 Related Work

8 Future Work

Part V

Epilogue

9 Conclusion

Bibliography

[Citing pages are listed after each reference.]

Amal Ahmed, Robert Bruce Findler, Jacob Matthews, and Philip Wadler. 2009. Blame for All. In Proceedings for the 1st Workshop on Script to Program Evolution (STOP '09). Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/1570506.1570507 [cited on page 4]

Ronald Garcia and Matteo Cimini. 2015. Principal Type Schemes for Gradual Programs. In Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15). Association for Computing Machinery, New York, NY, USA, 303–315. https://doi.org/10.1145/2676726.2676992 [cited on page 4]

Jeremy Siek and Walid Taha. 2007. Gradual Typing for Objects. In Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP'07). Springer-Verlag, Berlin, Heidelberg, 2–27. [cited on page 4]

Jeremy G Siek, Michael M Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined criteria for gradual typing. In 1st Summit on Advances in Programming Languages (SNAPL 2015). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. [cited on page 4]

Ningning Xie, Xuan Bi, and Bruno C d S Oliveira. 2018. Consistent Subtyping for All. In European Symposium on Programming. Springer, 3–30. [cited on page 5]

Ningning Xie, Xuan Bi, Bruno C. D. S. Oliveira, and Tom Schrijvers. 2019a. Consistent Subtyping for All. ACM Transactions on Programming Languages and Systems 42, 1, Article 2 (Nov. 2019), 79 pages. https://doi.org/10.1145/3310339 [cited on page 5]

Ningning Xie, Richard A. Eisenberg, and Bruno C. d. S. Oliveira. 2019b. Kind Inference for Datatypes. Proc. ACM Program. Lang. 4, POPL, Article 53 (Dec. 2019), 28 pages. https://doi.org/10.1145/3371121 [cited on page 5]

Ningning Xie and Bruno C d S Oliveira. 2017. Towards Unification for Dependent Types. In Draft Proceedings of the 18th Symposium on Trends in Functional Programming (TFP '18). Extended abstract. [cited on page 5]

Ningning Xie and Bruno C d S Oliveira. 2018. Let Arguments Go First. In European Symposium on Programming. Springer, 272–299. [cited on page 5]

Part VI

Technical Appendix