Syntactic Metatheory of Higher-Order Subtyping

Andreas Abel and Dulma Rodriguez

Department of Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
{andreas.abel|dulma.rodriguez}@ifi.lmu.de

Abstract. We present a new proof of decidability of higher-order subtyping in the presence of bounded quantification. The algorithm is formulated as a judgement which operates on beta-eta-normal forms. Transitivity and closure under application are proven directly and syntactically, without the need for a model construction or reasoning on longest beta-reduction sequences. The main technical tool is hereditary substitution, i.e., substitution of one normal form into another, resolving all freshly generated redexes on the fly. Hereditary substitutions are used to keep types in normal-form during execution of the subtyping algorithm. Termination of hereditary substitutions can be proven in an elementary way, by a lexicographic induction on the kind of the substituted variable and the size of the expression substituted into—this is what enables a purely syntactic metatheory.
Keywords: Higher-order subtyping, bounded quantification, algorithmic subtyping, hereditary substitution.

1 Introduction

Higher-order subtyping with bounded quantification has been used to model aspects of object-oriented programming languages [Pie02, Ch. 32]. Decidability is non-trivial and has been studied extensively in the past. Both Compagnoni [Com95] and Pierce and Steffen [PS97] have provided an algorithm for deciding subtyping for Kernel System $F^{\omega}_{<:}$ and proven its completeness by establishing a strong normalization theorem, while Compagnoni and Goguen [CG03,CG06] have studied the more general system $\mathcal{F}^{\omega}_{\leq}$ and proved completeness by constructing a Kripke model.

The cited works are impressive, but the complexity of the proofs is a bit overwhelming when it comes to formalizing them in a theorem prover like Coq, Isabelle, or Twelf. The reason is that strong normalization theorems or models are laborious to mechanize and little is known about automating the involved proofs. However, recently there have been successes in formalizing purely syntactical developments of metatheory of programming languages, most notably SML [LCH07]. Such formalizations use only first-order inductive judgements over syntactical objects and proofs by induction over these judgements or simple arithmetical measures.

A modern technique to treat the metatheory of systems which rely on normalization is *hereditary substitution* [WC$^+$03]. Hereditary substitution provides an algorithm for bottom-up normalization whose termination can be proven by a simple lexicographic induction—provided the proof-theoretical strength of the language does not exceed that of the simply-typed lambda-calculus. The technique of hereditary substitution simplifies the metatheory considerably. Hereditary substitution has been successfully used for the normalization of the Concurrent Logical Framework [WC$^+$03], the Edinburgh LF [HL07], and the type language of SML [LCH07].

In this article, we present a purely syntactic metatheory of $F^{\omega}_{<:}$ using the technique of hereditary substitution, affirming that it is flexible enough to account for bounded quantification, which is similar to lazy let-binding or singleton types. The result is a major simplification of the metatheory of $F^{\omega}_{<:}$ and a proof structure that is ready for formalization in a proof assistant even with low proof-theoretical complexity such as Twelf.

Detailed proofs for the theorems of this paper can be found in the diploma thesis of the second author [Rod07].

Contents. In Section 2, we recapitulate System $F^{\omega}_{<:}$ with kinding, equality and declarative subtyping and present a subtyping algorithm which works on η-long β-normal type constructors. In Section 3 we present algorithms for hereditary substitution and normalization and prove their correctness; this section contains the main technical work. Afterwards, in Section 4, we show soundness, completeness, and termination of the subtyping algorithm, which is in essence a consequence of the results of Section 3. Finally, we discuss related and further work in the conclusions.

Judgements. In this article, the following judgements will be defined inductively:

$\Gamma \vdash F : \kappa$	constructor F has kind κ in context Γ
$\Gamma \vdash F = F' : \kappa$	F and F' of kind κ are $\beta\eta$-equal
$\Gamma \vdash F \leq F' : \kappa$	F is a higher-order subtype of F'
$\Gamma \vdash V \Uparrow \kappa$	V is hereditarily normal of kind κ
$\Gamma \vdash_{\mathsf{a}} V \leq V'$	V is a subtype of V', algorithmically

When we write $\mathcal{D} :: J$, we mean that judgement J is established by derivation $\mathcal{D}$. Then, $|\mathcal{D}|$ denotes the height of this derivation. We consider derivations ordered by their height: $\mathcal{D}_1 \leq \mathcal{D}_2$ iff $|\mathcal{D}_1| \leq |\mathcal{D}_2|$.

2 System $F^{\omega}_{<:}$

This section introduces the syntax and the rules of kinding, equality, and subtyping of system $F^{\omega}_{<:}$, a typed λ-calculus of polymorphic functions, subtyping, and type operators.

2.1 Constructors and Kinds

System $F^{\omega}_{<:}$ consists of terms (programs), type constructors and kinds. This article is dedicated to the decidability of subtyping alone, thus, we ignore terms and typing completely. Note, however, that decidability of typing follows from decidability of subtyping (via the concept of *promotion* [Pie02, Ch. 28.1] [PS97]).

Constructors are classified by their kind, i.e., as types, functions on types, etc. Kinds are given by the grammar:

$$\kappa ::= * \mid \kappa_1 \to \kappa_2$$

Let $\boldsymbol{\kappa} \to \kappa'$ be an abbreviation for $\kappa_1 \to \kappa_2 \to \ldots \to \kappa_n \to \kappa'$ where $|\boldsymbol{\kappa}| = n$. Let $|\kappa| \in \mathbb{N}$ denote a measure on kinds with $|\kappa'| \leq |\kappa \to \kappa'|$ and $|\kappa| < |\kappa \to \kappa'|$. An example of such a measure is the *rank*, which is defined recursively as $\mathsf{rk}(\boldsymbol{\kappa} \to *) = \max\{1 + \mathsf{rk}(\kappa_i) \mid 1 \leq i \leq |\boldsymbol{\kappa}|\}$. In particular, $\mathsf{rk}(*) = 0$.

Constructors are given by the following Church-style type-level λ-calculus. The meta-variable X ranges over a countable infinite set of constructor variables.

$$A, B, F, G ::= X \mid \lambda X{:}\kappa.\, F \mid F\, G \mid A \to B \mid \forall X \leq G{:}\kappa.\, A \mid \top$$

As usual, $\lambda X : \kappa.\, F$ binds variable X in F. We identify constructors up to α-equivalence, i.e., up to renaming of bound variables. Sometimes, when we want to stress syntactic identity of constructors, we use $\equiv$ for α-equivalence. The letters U, V, W denote β-normal constructors and A, B constructors of kind $*$ (types). We define $\top_{\boldsymbol{\kappa} \to *} = \lambda \boldsymbol{Y} : \boldsymbol{\kappa}.\, \top$ (meaning $\lambda Y_1 : \kappa_1.\, \ldots \lambda Y_{|\boldsymbol{Y}|} : \kappa_{|\boldsymbol{\kappa}|}.\, \top$) where the lengths of $\boldsymbol{Y}$ and $\boldsymbol{\kappa}$ coincide. A vector notation is also used for application: $F\, \boldsymbol{G}$ means $F\, G_1 \ldots G_{|\boldsymbol{G}|}$, where application associates to the left as usual.

Contexts follow the grammar $\Gamma ::= \diamond \mid \Gamma, X \leq G : \kappa$. We refer to G as the *bound* of X. We assume all variables bound in Γ to be distinct and define the notation $\Gamma, X{:}\kappa$ as an abbreviation for $\Gamma, X \leq \top_{\kappa} : \kappa$.

2.2 Kinding and Well-formed Contexts

Well-formed contexts $\Gamma \vdash$, defined mutually with the kinding judgement $\Gamma \vdash F : \kappa$ in Figure 1, are constructed from the empty context by adding well-kinded type variable declarations. Kinding is decidable and unique, since constructor variables are annotated with their kinds.

The extra assumption $\Gamma \vdash G : \kappa$ in rule (K-Var) is due to Pierce and Steffen [PS97] and simplifies the proof of Theorem 1, which entails termination of the subtyping algorithm.

We maintain the invariant that kinding statements are only derivable in well-formed contexts (see Lemma 1.1). Bounds do not matter for kinding, i.e., if a constructor F has kind κ in a context Γ , and Γ' is the same context as Γ but with different, well-kinded bounds, then F has kind κ in context Γ' as well.

Lemma 1 (Admissible rules for kinding).

1. *Validity: If $\Gamma \vdash F : \kappa$ then $\Gamma \vdash$.*

$\boxed{\Gamma \vdash}$

$$\text{(C-Empty)}\ \frac{}{\diamond \vdash} \qquad \text{(C-Bound)}\ \frac{\Gamma \vdash \qquad \Gamma \vdash G : \kappa}{\Gamma, X \leq G : \kappa \vdash}$$

$\boxed{\Gamma \vdash F : \kappa}$

$$\text{(K-Var)}\ \frac{(X \leq G : \kappa) \in \Gamma \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash X : \kappa} \qquad \text{(K-Top)}\ \frac{\Gamma \vdash}{\Gamma \vdash \top : *}$$

$$\text{(K-Abs)}\ \frac{\Gamma, X : \kappa \vdash F : \kappa'}{\Gamma \vdash \lambda X : \kappa.\, F : \kappa \to \kappa'} \qquad \text{(K-App)}\ \frac{\Gamma \vdash F : \kappa \to \kappa' \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash F\, G : \kappa'}$$

$$\text{(K-Arr)}\ \frac{\Gamma \vdash A : * \qquad \Gamma \vdash B : *}{\Gamma \vdash A \to B : *} \qquad \text{(K-All)}\ \frac{\Gamma \vdash G : \kappa \qquad \Gamma, X \leq G : \kappa \vdash A : *}{\Gamma \vdash \forall X \leq G : \kappa.\, A : *}$$

$\boxed{\Gamma \vdash F = F' : \kappa}$

$$\text{(Eq-}\beta\text{)}\ \frac{\Gamma, X : \kappa \vdash F : \kappa' \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash (\lambda X : \kappa.\, F)\, G = [G/X]F : \kappa'} \qquad \text{(Eq-}\eta\text{)}\ \frac{\Gamma \vdash F : \kappa \to \kappa' \qquad X \notin \mathsf{FV}(F)}{\Gamma \vdash \lambda X : \kappa.\, FX = F : \kappa \to \kappa'}$$

$$\text{(Eq-Var)}\ \frac{(X \leq G : \kappa) \in \Gamma \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash X = X : \kappa} \quad \text{(Eq-Abs)}\ \frac{\Gamma, X : \kappa \vdash F = F' : \kappa'}{\Gamma \vdash \lambda X : \kappa.\, F = \lambda X : \kappa.\, F' : \kappa \to \kappa'}$$

$$\text{(Eq-App)}\ \frac{\Gamma \vdash F = F' : \kappa \to \kappa' \qquad \Gamma \vdash G = G' : \kappa}{\Gamma \vdash F\, G = F'\, G' : \kappa'}$$

$$\text{(Eq-Top)}\ \frac{\Gamma \vdash}{\Gamma \vdash \top = \top : *} \qquad \text{(Eq-Arr)}\ \frac{\Gamma \vdash A = A' : * \qquad \Gamma \vdash B = B' : *}{\Gamma \vdash A \to B = A' \to B' : *}$$

$$\text{(Eq-All)}\ \frac{\Gamma \vdash G = G' : \kappa \qquad \Gamma, X \leq G : \kappa \vdash A = A' : *}{\Gamma \vdash \forall X \leq G : \kappa . A = \forall X \leq G' : \kappa . A' : *}$$

$$\text{(Eq-Sym)}\ \frac{\Gamma \vdash F = F' : \kappa}{\Gamma \vdash F' = F : \kappa} \qquad \text{(Eq-Trans)}\ \frac{\Gamma \vdash F_1 = F_2 : \kappa \qquad \Gamma \vdash F_2 = F_3 : \kappa}{\Gamma \vdash F_1 = F_3 : \kappa}$$

$\boxed{\Gamma \vdash F \leq F' : \kappa}$

$$\text{(S-Var)}\ \frac{\Gamma \vdash G : \kappa \qquad X \leq G : \kappa \in \Gamma}{\Gamma \vdash X \leq G : \kappa} \qquad \text{(S-App)}\ \frac{\Gamma \vdash F \leq F' : \kappa \to \kappa' \qquad \Gamma \vdash H : \kappa}{\Gamma \vdash FH \leq F'H : \kappa'}$$

$$\text{(S-Top)}\ \frac{\Gamma \vdash A : *}{\Gamma \vdash A \leq \top : *} \qquad \text{(S-Abs)}\ \frac{\Gamma, X : \kappa \vdash F \leq F' : \kappa'}{\Gamma \vdash \lambda X : \kappa.\, F \leq \lambda X : \kappa.\, F' : \kappa \to \kappa'}$$

$$\text{(S-Arr)}\ \frac{\Gamma \vdash A' \leq A : * \qquad \Gamma \vdash B \leq B' : *}{\Gamma \vdash A \to B \leq A' \to B' : *}$$

$$\text{(S-All)}\ \frac{\Gamma \vdash G : \kappa \qquad \Gamma, X \leq G : \kappa \vdash A \leq A' : *}{\Gamma \vdash \forall X \leq G : \kappa . A \ \leq\ \forall X \leq G : \kappa . A' : *}$$

$$\text{(S-Eq)}\ \frac{\Gamma \vdash F = G : \kappa}{\Gamma \vdash F \leq G : \kappa} \qquad \text{(S-Trans)}\ \frac{\Gamma \vdash F \leq G : \kappa \qquad \Gamma \vdash G \leq H : \kappa}{\Gamma \vdash F \leq H : \kappa}$$

Fig. 1. Declarative presentation of $F^{\omega}_{<:}$.

2. *Weakening: If* $\Gamma, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, X \leq G : \kappa, \Gamma' \vdash F : \kappa'$.
3. *Substitution: If* $\Gamma, X \leq H : \kappa, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F : \kappa'$.

Since bounds do not matter for kinding, we do not require $\Gamma \vdash G \leq H : \kappa$ in the substitution property.

2.3 Equality

In contrast to previous presentations of System $F^{\omega}_{<:}$ [Pie02, Ch. 31], we consider constructors equivalent modulo β *and* η. Instead of an untyped equality we define an equality judgement $\Gamma \vdash F = F' : \kappa$, since this is more robust w. r. t. extensions, e. g., by polarities [Ste98,Abe06b]. The judgement is given by the axioms (Eq-β) and (Eq-η) plus congruence and equivalence rules (see Figure 1).

The equality judgement has the usual properties.

Lemma 2 (Admissible rules for equality).

1. *Reflexivity: If* $\Gamma \vdash F : \kappa$ *then* $\Gamma \vdash F = F : \kappa$.
2. *Validity: If* $\Gamma \vdash G = G' : \kappa$ *then* $\Gamma \vdash G : \kappa$ *and* $\Gamma \vdash G' : \kappa$.
3. *Weakening: If* $\Gamma, \Gamma' \vdash F = F' : \kappa$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, X \leq G : \kappa, \Gamma' \vdash F = F' : \kappa$.
4. *Substitution: If* $\Gamma, X \leq H : \kappa, \Gamma' \vdash F = F' : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F = [G/X]F' : \kappa'$.

Proof. Each by induction on the first derivation. In the proof of item 2, case (Eq-All), we use the fact that bounds do not matter for kinding. □

2.4 Subtyping

The $F^{\omega}_{<:}$ subtyping relation $\Gamma \vdash F \leq F' : \kappa$ (see Figure 1) extends the subtyping relation of system $F_{\leq}$ [CW85]. Subtyping for type operators of higher kind is defined pointwise, see (S-Abs) and (S-App).

Decidability requires using the Kernel-Fun rule (S-All). *Full* subtyping would allow a bound $G' \leq G$ on the right hand side, losing decidability [Pie92]. We do not treat antisymmetry here. Reflexivity is inherited from equality.

Lemma 3 (Admissible rules for subtyping).

1. *Validity: If* $\Gamma \vdash F \leq F' : \kappa$ *then* $\Gamma \vdash F : \kappa$ *and* $\Gamma \vdash F' : \kappa$.
2. *Weakening: If* $\Gamma, \Gamma' \vdash F \leq F' : \kappa$ *and* $\Gamma \vdash G : \kappa'$ *then* $\Gamma, X \leq G : \kappa', \Gamma' \vdash F \leq F' : \kappa$.
3. *Substitution: If* $\Gamma, X \leq H : \kappa, \Gamma' \vdash F \leq F' : \kappa'$ *and* $\Gamma \vdash G \leq H : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F \leq [G/X]F' : \kappa'$ *(not needed in the following).*

2.5 Algorithmic Subtyping

The declarative definition of subtyping contains two rules that correspond to a logical *cut*: on the level of constructors, the transitivity rule (S-Trans), if one views types as predicates and subtyping as predicate inclusion; and on the level of kinds, the application rule (S-App), if one views kinds as implicational propositions and constructors as their proofs.[1] In an algorithmic version of subtyping, both kinds of cuts have to be eliminated [Com95,PS97]. Application is eliminated by considering constructors in normal form V only, in our case η-long β-normal form. Transitivity is incorporated into the variable rule (SA-Bound), which is a fusion of (S-Var), (S-App), (S-Eq), and (S-Trans). It looks up the bound U of the head X of a *neutral* constructor $X\boldsymbol{V}$ and recursively checks subtyping for $U\,\boldsymbol{V}$. To keep everything in normal form, a *normalizing application* $U \mathbin{@} \boldsymbol{V}$ is employed which will be defined in Section 3.1.

The algorithm receives as input a context Γ and two η-long constructors V, V' such that $\Gamma \vdash V : \kappa$ and $\Gamma \vdash V' : \kappa$ and decides $\Gamma \vdash V \leq V' : \kappa$.

$\boxed{\Gamma \vdash_{\mathsf{a}} V \leq V'}$

$$\text{(SA-Top)}\ \frac{}{\Gamma \vdash_{\mathsf{a}} V \leq \top} \qquad \text{(SA-Refl)}\ \frac{}{\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq X\boldsymbol{V}}$$

$$\text{(SA-Bound)}\ \frac{(X \leq U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash_{\mathsf{a}} U \mathbin{@} \boldsymbol{V} \leq W}{\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq W}$$

$$\text{(SA-Arr)}\ \frac{\Gamma \vdash_{\mathsf{a}} W \leq V : * \qquad \Gamma \vdash_{\mathsf{a}} V' \leq W'}{\Gamma \vdash_{\mathsf{a}} V \to V' \leq W \to W'}$$

$$\text{(SA-All)}\ \frac{\Gamma, X \leq V : \kappa \vdash_{\mathsf{a}} W \leq W'}{\Gamma \vdash_{\mathsf{a}} \forall X \leq V : \kappa . W \leq \forall X \leq V : \kappa . W'}$$

$$\text{(SA-Abs)}\ \frac{\Gamma, X : \kappa \vdash_{\mathsf{a}} V \leq V'}{\Gamma \vdash_{\mathsf{a}} \lambda X : \kappa . V \leq \lambda X : \kappa . V'}$$

Observe that since we are dealing with η-long forms, the only constructors of higher kind are λ-abstractions (SA-Abs). The rules specify a deterministic algorithm if one checks applicability of rules earlier in the list prior to rules mentioned later. In particular, (SA-Top) is checked first, and (SA-Refl) before (SA-Bound).

Our algorithmic subtyping rules correspond to Compagnoni's [Com95] and Pierce and Steffen's [PS97]—except that they consider Church-style β-normal constructors, hence, their algorithm does not validate η-equality. Thus, one cannot claim our algorithm is new, but in the remainder of this article we will present a novel, very direct and concise correctness proof with purely syntactic methods, which has a realistic chance of being mechanized in a theorem prover such as Coq, Isabelle, or Twelf.

[1] More precisely, (S-App) is the rule of modus ponens, but in unrestricted form it allows non-normal proofs.

$\boxed{[G/X]^\kappa F}$

$$
\begin{array}{lll}
[G/X]^\kappa Y & := G^\kappa & \text{if } X = Y \\
 & \quad Y & \text{otherwise} \\
[G/X]^\kappa(\lambda Y\!:\!\kappa'.\, F) & := \lambda Y\!:\!\kappa'.\, \underline{[G/X]^\kappa F} & \text{where } Y \text{ fresh for } X, G \\
[G/X]^\kappa(F H) & := ([\hat{\underline{H}}/Y]^{\kappa_1} F')^{\kappa_2} & \text{if } \hat{F} = (\lambda Y\!:\!\kappa_1'.\, F')^{\kappa_1 \to \kappa_2} \\
 & \quad \underline{\hat{F}}\ \underline{\hat{H}} & \text{otherwise} \\
 & \quad \text{herein, } \hat{F} = [G/X]^\kappa F & \\
 & \qquad\quad \hat{H} = [G/X]^\kappa H & \\
[G/X]^\kappa(A \to B) & := \underline{[G/X]^\kappa A} \to \underline{[G/X]^\kappa B} & \\
[G/X]^\kappa(\forall Y \le H\!:\!\kappa.A) & := \forall Y \le \underline{[G/X]^\kappa H}\!:\!\kappa. & \\
 & \quad \underline{[G/X]^\kappa A} & \text{where } Y \text{ fresh for } X, G \\
[G/X]^\kappa \top & := \top &
\end{array}
$$

$\boxed{F \mathbin{@} G}$

$$
\begin{array}{lll}
(\lambda X\!:\!\kappa.\, F) \mathbin{@} G & := \underline{[G/X]^\kappa F} & \\
F \mathbin{@} G & := \underline{F\, G} & \text{if } F \neq \lambda X\!:\!\kappa.\, F'
\end{array}
$$

$\boxed{F \mathbin{@} \boldsymbol{G}}$

$$
F \mathbin{@} \boldsymbol{G} := ((F \mathbin{@} G_1) \mathbin{@} G_2 \ldots) \mathbin{@} G_{|\boldsymbol{G}|}
$$

Fig. 2. Hereditary substitution.

3 Normalization of Constructors

In last section we have described a decision procedure for subtyping which works on η-long β-normal forms. In this section, we will describe a normalization algorithm $\mathsf{nf}_{\_}(-)$ which is correct w. r. t. judgmental equality:

1. Sound: If $\Gamma \vdash F : \kappa$ then $\Gamma \vdash \mathsf{nf}_\Gamma(F) = F : \kappa$.
2. Complete: If $\Gamma \vdash F = F' : \kappa$ then $\mathsf{nf}_\Gamma(F) \equiv \mathsf{nf}_\Gamma(F')$.

3.1 Hereditary Substitution

There are abundant strategies to compute β-normal forms; we use *bottom-up normalization*, because its termination can be shown directly in a simply-typed setting—in our case it is a *simply-kinded* setting.

The bottom-up strategy $\mathsf{nf}(H)$ normalizes the immediate subterms of H and then puts them back together. For an application $H = F\,G$, normalization $\mathsf{nf}(F)$ of the function part may yield an abstraction $\lambda Y{:}\kappa.\,F'$. In this case, a β-normal form can be recovered by substituting $\mathsf{nf}(G)$ for Y in F'. New redexes may be created in turn which are resolved immediately by another substitution etc. The iteration of this process, which we call *hereditary substitution*, terminates since the kind of the substituted variable decreases with each iteration.

Hereditary substitutions are implicit in combinatorial normalization proofs, e. g., Prawitz [Pra65], Levy [Lév76], Girard, Lafont, and Taylor [GLT89, Ch. 4], and Amadio and Curien [AC97, Thm. 2.2.9]. They were first made explicit by Watkins et. al. [WC$^+$03] and Adams [Ada05] for dependent types. We follow the presentation of the first author [Abe06a].

Hereditary substitution (see Figure 2) is given as a 4-ary function $[G/X]^\kappa H$, whose result $\hat{F}$ is either just a constructor F or a constructor annotated with a kind, written F^κ. If G and H are β-normal (and well-kinded) then the result will also be β-normal (and well-kinded). Results can be coerced back to constructors via an erasure operation given by $\underline{F^\kappa} = F$ and $\underline{F} = F$.

It is easy to see that if $[G/X]^\kappa H = F^{\kappa_2}$ then $|\kappa_2| \leq |\kappa|$. This invariant ensures the termination of hereditary substitution. For correctness we need to show that, modulo β-equality, hereditary substitution is conventional substitution. In the following, we leave the coercion implicit.

Lemma 4 (Termination and soundness of hereditary substitutions). *If $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash F : \kappa'$ and $\Gamma \vdash G : \kappa$ then $\Gamma, [G/X]\Gamma' \vdash [G/X]F = [G/X]^\kappa F : \kappa'$.*

Proof. By lexicographical induction on $(|\kappa|, \mathcal{D})$. The proof is a straightforward extension of the soundness proof in [Abe06a]. □

Corollary 1. *If $\Gamma \vdash F : \kappa \to \kappa'$ and $\Gamma \vdash G : \kappa$ then $\Gamma \vdash F\,@\,G = F\,G : \kappa'$.*

Lemma 5 (Commutativity of hereditary substitutions). *Let $\Gamma \vdash U : \kappa$ and $\Gamma, X{:}\kappa, \Gamma' \vdash V : \kappa'$ with $\kappa' = \boldsymbol{\kappa} \to \kappa_0$.*

1. *If $\Gamma, X{:}\kappa, \Gamma' \vdash W_i : \kappa_i$ for $1 \leq i \leq |\boldsymbol{\kappa}|$ then*

$$[U/X]^\kappa(V\,@\,\boldsymbol{W}) \equiv [U/X]^\kappa V\,@\,[U/X]^\kappa \boldsymbol{W}$$

2. *If $\Gamma, X{:}\kappa, \Gamma', Y{:}\kappa', \Gamma'' \vdash W : \kappa''$ then:*

$$[U/X]^\kappa([V/Y]^{\kappa'} W) \equiv [[U/X]^\kappa V/Y]^{\kappa'}([U/X]^\kappa W)$$

Proof. Simultaneously by simultaneous induction[2] on $\{\kappa, \kappa'\}$, first 1 and then 2. The proof of 2 proceeds by a local induction on W. □

[2] Simultaneous order is defined by $\{X, Y\} < \{X', Y'\}$ if $X < X'$ and $Y \leq Y'$.

3.2 Computing the Long Normal Form

Well-kinded constructors are β-normalizing, and we can compute their η-long β-normal form. First, let us define the η-expansion $\eta_\kappa(N)$ of a neutral constructor N at kind κ by $\eta_*(N) = N$ and $\eta_{\kappa\to\kappa'}(N) = \lambda X{:}\kappa.\, \eta_{\kappa'}(N\, \eta_\kappa(X))$.

Lemma 6 (η-expansion is sound). *If $\Gamma \vdash N : \kappa$ then $\Gamma \vdash N = \eta_\kappa(N) : \kappa$.*

Normalization is given by a function $\mathsf{nf}_\Gamma(F)$, defined by recursion on F.

$$
\begin{array}{lll}
\mathsf{nf}_\Gamma(X) & := \eta_\kappa(X) & \text{if } (X \le \_ : \kappa) \in \Gamma \\
\mathsf{nf}_\Gamma(\top) & := \top & \\
\mathsf{nf}_\Gamma(\lambda X{:}\kappa.\, F) & := \lambda X{:}\kappa.\, \mathsf{nf}_{\Gamma, X:\kappa}(F) & \\
\mathsf{nf}_\Gamma(A \to B) & := \mathsf{nf}_\Gamma(A) \to \mathsf{nf}_\Gamma(B) & \\
\mathsf{nf}_\Gamma(\forall X \le G{:}\kappa.A) & := \forall X \le \mathsf{nf}_\Gamma(G){:}\kappa.\, \mathsf{nf}_{\Gamma, X\le G:\kappa}(A) & \\
\mathsf{nf}_\Gamma(F\, G) & := \mathsf{nf}_\Gamma(F)\, @\, \mathsf{nf}_\Gamma(G) &
\end{array}
$$

The algorithm η-expands the variables, this way producing η-long β-normal constructors. For well-kinded constructors $\mathsf{nf}()$ returns the normal form (see Theorem 1), but not for all ill-kinded constructors, e.g., $\mathsf{nf}(\Omega) = \Omega$ for $\Omega = (\lambda X{:}*.\, X\, X)\, (\lambda X{:}*.\, X\, X)$.

We omit the subscript Γ if clear from the context of discourse. Normalization can be extended to contexts in the obvious way: $\mathsf{nf}(\Gamma)$ computes a context where all bounds have been normalized.

Lemma 7 (Soundness and termination of normalization). *If $\Gamma \vdash F : \kappa$ then $\Gamma \vdash F = \mathsf{nf}_\Gamma(F) : \kappa$.*

Proof. By induction on the kinding derivation, using Lemma 6 in the variable case, and Corollary 1 in the application case. □

3.3 Characterization of Long Normal Forms

We will now define a judgement $\Gamma \vdash V \Uparrow \kappa$, read "*$V$ is hereditarily normal of kind κ in context Γ*", that classifies the β-normal constructor V as a possible input for the subtyping algorithm. In particular, V must be η-long, and new redexes which might be created by (SA-Bound) during the execution of the algorithm must be normalizable. We call such redexes *hidden*, an example is $\forall X \le (\lambda Y{:}*.\, V){:}* \to *.\, X\, W$ which contains the hidden redex $(\lambda Y{:}*.\, V)\, W$.

$\boxed{\Gamma \vdash V \Uparrow \kappa}$

$$
\text{(LN-Bound)}\ \frac{(X \le U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash V_i \Uparrow \kappa_i \qquad \Gamma \vdash U\, @\, \boldsymbol{V} \Uparrow * \ (\star)}{\Gamma \vdash X\, \boldsymbol{V} \Uparrow *}
$$

$$
\text{(LN-Abs)}\ \frac{\Gamma, X{:}\kappa \vdash V \Uparrow \kappa'}{\Gamma \vdash \lambda X{:}\kappa.\, V \Uparrow \kappa \to \kappa'} \qquad \text{(LN-Arr)}\ \frac{\Gamma \vdash V \Uparrow * \qquad \Gamma \vdash W \Uparrow *}{\Gamma \vdash V \to W \Uparrow *}
$$

$$
\text{(LN-All)}\ \frac{\Gamma, X \le U{:}\kappa \vdash V \Uparrow *}{\Gamma \vdash \forall X \le U{:}\kappa.\, V \Uparrow *} \qquad \text{(LN-Top)}\ \frac{\Gamma \Uparrow}{\Gamma \vdash \top \Uparrow *}
$$

Remark 1. The third hypothesis ($\star$) in (LN-Bound) ensures normalization of hidden redexes, hence $\Gamma \vdash V \Uparrow \kappa$ can be used as a termination measure for the algorithm. Even without ($\star$) the judgement characterizes the η-long normal forms.

A context is normal, $\Gamma \Uparrow$, if all bounds in Γ are hereditarily normal. In the following, we establish that normalization is the identity on long normal forms. First we prove it for variables.

Lemma 8.

1. *If* $\Gamma \vdash V \Uparrow \kappa$ *then* $[V/X]^\kappa \eta_\kappa(X) \equiv V$.
2. *If* $\mathcal{D} :: \Gamma, X{:}\kappa, \Gamma' \vdash W \Uparrow \kappa'$ *then* $[\eta_\kappa(X)/X]^\kappa W \equiv W$.

Proof. Simultaneously by induction on κ, and a local induction on $\mathcal{D}$ in 2. □

Lemma 9 (Idempotency of nf). *If* $\mathcal{D} :: \Gamma \vdash U \Uparrow \kappa$ *then* $\mathsf{nf}(U) \equiv U$.

Proof. By induction on $\mathcal{D}$, using Lemma 8 in the variable case. □

We now build up to a normalization theorem: we will show that nf produces a hereditarily normal form from each well-kinded constructor. The following lemma, which can be seen as the heart of our technical development, proves normalization of application.

Lemma 10 (Substitution and application for normal forms).
Let $\mathcal{E} :: \Gamma \vdash U \Uparrow \kappa$.

1. *Assume* $\mathcal{D} :: \Gamma, X{:}\kappa, \Gamma' \Uparrow$. *Then* $\Gamma, [U/X]^\kappa \Gamma' \Uparrow$.
2. *Assume* $\mathcal{D} :: \Gamma, X{:}\kappa, \Gamma' \vdash V \Uparrow \kappa'$. *Then* $\Gamma, [U/X]^\kappa \Gamma' \vdash [U/X]^\kappa V \Uparrow \kappa'$.
3. *Assume* $\mathcal{D} :: \Gamma \vdash V \Uparrow \kappa \to \kappa'$. *Then* $\Gamma \vdash V \mathbin{@} U \Uparrow \kappa'$.

Proof. Simultaneously by lexicographical induction on $(|\kappa|, \mathcal{D})$. The interesting case of item 2 is (LN-Bound) with $V \equiv X\boldsymbol{V}$, $\kappa = \boldsymbol{\kappa} \to *$.

$$\frac{\Gamma, X{:}\kappa, \Gamma' \vdash V_i \Uparrow \kappa_i \qquad \Gamma, X{:}\kappa, \Gamma' \vdash \top_\kappa \mathbin{@} \boldsymbol{V} \Uparrow *}{\Gamma, X{:}\kappa, \Gamma' \vdash X\boldsymbol{V} \Uparrow *}$$

By induction hypothesis (2), we have $\mathcal{D}_i :: \Gamma, [U/X]^\kappa \Gamma' \vdash [U/X]^\kappa V_i \Uparrow \kappa_i$ for $i = 1..|\boldsymbol{\kappa}|$. Moreover, we have by induction hypothesis (3):

$$\mathcal{E}_1 :: \Gamma, [U/X]^\kappa \Gamma' \vdash U \mathbin{@} [U/X]^\kappa V_1 \Uparrow \kappa_2 \to \ldots \to \kappa_n \to *$$

because $(\kappa_1, \mathcal{E}) < (\kappa, \mathcal{D})$. We again have by induction hypothesis (3):

$$\mathcal{E}_2 :: \Gamma, [U/X]^\kappa \Gamma' \vdash U \mathbin{@} ([U/X]^\kappa V_1) \mathbin{@} ([U/X]^\kappa V_2) \Uparrow \kappa_3 \to \ldots \to \kappa_n \to *$$

because $(\kappa_2, \mathcal{E}_1) < (\kappa, \mathcal{D})$. Continuing this schema, we get

$$\mathcal{E}_n :: \Gamma, [U/X]^\kappa \Gamma' \vdash U \mathbin{@} [U/X]^\kappa \boldsymbol{V} \Uparrow *$$

which is equivalent to $\Gamma, [U/X]^\kappa \Gamma' \vdash [U/X]^\kappa (X\,\boldsymbol{V}) \Uparrow *$ and we are finished.

In case $V \equiv Y\boldsymbol{V}$, with $Y \neq X$, we use Lemma 5. □

Theorem 1 (Normalization).

1. *If* $\mathcal{D} :: \Gamma \vdash$ *then* $\mathsf{nf}(\Gamma) \Uparrow$.
2. *If* $\mathcal{D} :: \Gamma \vdash F : \kappa$ *then* $\mathsf{nf}(\Gamma) \vdash \mathsf{nf}(F) \Uparrow \kappa$.

Proof. Simultaneously by induction on $\mathcal{D}$, using the previous lemma in case of application. □

3.4 Completeness of Normalization

In this section, we prove completeness of the normalization function, i.e., that the normal forms of judgmentally equal constructors are identical.

Lemma 11. *If* $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\mathsf{nf}([G/X]F) \equiv [\mathsf{nf}(G)/X]^{\kappa}\mathsf{nf}(F)$.

Proof. By induction on $\mathcal{D}$. In the application case we use Lemma 5. □

Theorem 2 (Completeness of the normalization). *If* $\mathcal{D} :: \Gamma \vdash F = F' : \kappa$ *then* $\mathsf{nf}(F) \equiv \mathsf{nf}(F')$.

Proof. The proof is by induction on $\mathcal{D}$, in case (Eq-η) we use the characterization of η-long β-normal forms (Theorem 1 and Lemma 9) and in case (Eq-β) we use the previous lemma. □

Corollary 2 (Uniqueness of normal forms). *If* $\Gamma \vdash V = V' : \kappa$ *and* $\Gamma \vdash V, V' \Uparrow \kappa$ *then* $V \equiv V'$.

Proof. Directly, using Theorem 2 and Lemma 9. □

4 Verification of Algorithmic Subtyping

In this section, we show that the subtyping algorithm is sound and complete for the declarative rules in Section 2. The difficult part, namely establishing the necessary properties of hereditary substitution and normalization and constructing a termination measure $\Gamma \vdash V \Uparrow \kappa$, has been completed in the last section. The actual properties of algorithmic subtyping are now easy to verify.

Soundness of the algorithm is straightforward because the algorithmic rules are less permissive than the declarative ones.

Lemma 12. *Let* $\Gamma \vdash V, V' : \kappa$. *If* $\mathcal{D} :: \Gamma \vdash_{\mathsf{a}} V \leq V'$, *then* $\Gamma \vdash V \leq V' : \kappa$.

Proof. By induction on $\mathcal{D}$. □

Theorem 3 (Soundness of algorithmic subtyping). *Let* $\Gamma \vdash F, F' : \kappa$. *If* $\mathcal{D} :: \mathsf{nf}(\Gamma) \vdash_{\mathsf{a}} \mathsf{nf}(F) \leq \mathsf{nf}(F')$, *then* $\Gamma \vdash F \leq F' : \kappa$.

Proof. Combining lemmata 12 and 7. To account for normalization of Γ, we establish that declarative equality and subtyping remain valid if we replace bounds in the context by judgmentally equal ones. □

Completeness of the algorithm means that any derivable statement in the declarative system is also derivable in the algorithmic system. This is more difficult to show than soundness, because we have eliminated declarative rules like (S-Trans) or (S-App). Thus, we need to show that these rules are admissible in the algorithmic system.

Lemma 13 (Reflexivity). *If V is β-normal then $\Gamma \vdash_{\mathsf{a}} V \leq V$.*

Proof. By induction on V. □

Lemma 14 (Transitivity). *$\mathcal{D}_1 :: \Gamma \vdash_{\mathsf{a}} V_1 \leq V_2$ and $\mathcal{D}_2 :: \Gamma \vdash_{\mathsf{a}} V_2 \leq V_3$ imply $\Gamma \vdash_{\mathsf{a}} V_1 \leq V_3$.*

Proof. By induction on $\mathcal{D}_1$. □

The following substitution lemma is the key step in showing that algorithmic subtyping is closed under application, i.e., complete for rule (S-App).

Lemma 15 (Substitution). *Let $\Gamma, X:\kappa, \Gamma' \vdash V, V' \Uparrow \kappa'$ and $\Gamma \vdash U \Uparrow \kappa$. If $\mathcal{D} :: \Gamma, X:\kappa, \Gamma' \vdash_{\mathsf{a}} V \leq V'$ then $\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} [U/X]^\kappa V \leq [U/X]^\kappa V'$.*

Proof. By induction on $\mathcal{D}$. The interesting case is (SA-Bound) for $V \equiv Y\boldsymbol{V}$ with Y bound later in the context than X.

$$\frac{(Y \leq U' : \boldsymbol{\kappa} \to *) \in \Gamma' \qquad \Gamma, X:\kappa, \Gamma' \vdash_{\mathsf{a}} U' \mathbin{@} \boldsymbol{V} \leq W}{\Gamma, X:\kappa, \Gamma' \vdash_{\mathsf{a}} Y\boldsymbol{V} \leq W}$$

By induction hypothesis we have

$$\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} [U/X]^\kappa (U' \mathbin{@} \boldsymbol{V}) \leq [U/X]^\kappa W$$

which by Lemma 5, part (1), is equivalent to

$$\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} [U/X]^\kappa U' \mathbin{@} [U/X]^\kappa \boldsymbol{V} \leq [U/X]^\kappa W.$$

The goal $\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} Y\, [U/X]^\kappa \boldsymbol{V} \leq [U/X]^\kappa W$ follows by (SA-Bound). □

Completeness can now be shown by induction on derivations, using the previous lemmas: reflexivity, transitivity and substitution.

Theorem 4 (Completeness).
If $\mathcal{D} :: \Gamma \vdash F \leq F' : \kappa$ then $\mathsf{nf}(\Gamma) \vdash_{\mathsf{a}} \mathsf{nf}(F) \leq \mathsf{nf}(F')$.

Proof. By induction on $\mathcal{D}$, using Lemma 15 in case (S-App). □

Termination of algorithmic subtyping is now proven via the inductive termination predicate $\Gamma \vdash V \Uparrow \kappa$ (which by Theorem 1 holds for all normal forms of well-kinded constructors). It is a considerable simplification of Compagnoni's [Com95] termination measure which features intersection types as well. It has to be investigated whether the simplicity of our measure can be kept in the presence of intersection types.

Theorem 5 (Termination of algorithmic subtyping). *If* $\mathcal{D}_1 :: \Gamma \vdash V \Uparrow \kappa$ *and* $\mathcal{D}_2 :: \Gamma \vdash V' \Uparrow \kappa$ *then the query* $\Gamma \vdash_{\mathsf{a}} V \leq V'$ *terminates.*

Proof. By simultaneous induction on $\{\mathcal{D}_1, \mathcal{D}_2\}$. We detail the case that $\mathcal{D}_1$ ends with (LN-Bound).

$$\frac{(X \leq U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash V_i \Uparrow \kappa_i \text{ for all } i \qquad \Gamma \vdash U \mathbin{@} \boldsymbol{V} \Uparrow *}{\Gamma \vdash X\boldsymbol{V} \Uparrow *}$$

In case $V' \neq X\boldsymbol{V}$ we apply the rule (SA-Bound)

$$\frac{(X \leq U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash_{\mathsf{a}} U \mathbin{@} \boldsymbol{V} \leq V'}{\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq V'}$$

By the induction hypothesis, the query $\Gamma \vdash_{\mathsf{a}} U \mathbin{@} \boldsymbol{V} \leq V'$ terminates, thus, the query $\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq V'$ terminates as well. □

Corollary 3 (Decidability). *If* $\Gamma \vdash F, F' : \kappa$ *then* $\Gamma \vdash F \leq F' : \kappa$ *is decidable.*

Proof. By Theorem 2 $\mathsf{nf}(\Gamma) \vdash \mathsf{nf}(F), \mathsf{nf}(F') \Uparrow \kappa$. Hence, the query $\mathsf{nf}(\Gamma) \vdash_{\mathsf{a}} \mathsf{nf}(F) \leq \mathsf{nf}(F')$ terminates by Theorem 5. If it succeeds then $\Gamma \vdash F \leq F' : \kappa$ by soundness (Theorem 3), if it fails then $\Gamma \not\vdash F \leq F' : \kappa$ by completeness (Theorem 4).

5 Conclusions and Related Work

In this article we have proven decidability of subtyping for $F^{\omega}_{<:}$ with $\beta\eta$-equality in a purely syntactical way, with only first-order inductive definitions and simple induction measures. The proofs have been organized with a formalization in mind and can be mechanized in proof assistants such as Coq, Isabelle, or Twelf.

Related work. Foundations of $F^{\omega}_{<:}$ and the type-theoretic investigation of object-oriented languages have been laid by Cardelli [Car88], Mitchell [Mit90], and Abadi and Cardelli [AC96].

Most similar to the present work is the one of Compagnoni [Com95] both in the design of the subtyping algorithm and the organization of the proofs of soundness, completeness, and termination. She also treats intersection types but not η-equality of constructors. The main differences in the proof are: She inserts the notion of *normal subtyping* between the declarative and the algorithmic one, which we do not require, she refers to strong normalization of reduction, which adds some complexity to the proof, and she has a complicated termination measure which involves the longest reduction sequences of constructors classified by a judgement similar to our hereditary normal forms. Our approach has allowed considerable simplifications in comparison with Compagnoni's.

Pierce and Steffen [PS97] justify algorithmic subtyping using rewriting theory. They consider an extension of β-reductions by Γ-reduction, which replaces

a head variable by its bound, and $\top$-reduction which witnesses that $\top$ is defined pointwise for higher kinds. The confluence and strong normalization theorems shed light on the combination of these notions of reduction, however, they are not the shortest path to the verification of the subtyping algorithm.

Compagnoni and Goguen [CG03] construct a Kripke model based on typed operational semantics to analyze the metatheory of $\mathcal{F}^{\omega}_{\leq}$, an extension of $F^{\omega}_{<:}$ by bounded abstraction $\lambda X \leq F.\, G$. They use this model to prove anti-symmetry [CG06] of subtyping which allows replacing judgmental equality by bi-inclusion. Recently they have investigated Church-style subtyping [CG07], where each variable is annotated by its bound, hence, the complicated Kripke-model is replaced by a non-Kripke logical relation. Their subtyping algorithm works with weak-head normal forms, only the decision whether the bound-lookup rule (SA-Bound) should fire demands full normalization in some cases. The structure of their proof, with logical relation and strong normalization theorem, seems very robust w. r. t. extensions of the type language; however, it cannot compete with us in terms of proof economics.

The first author [Abe06b] considers $\mathsf{F}^{\widehat{}}_{\omega}$, the higher-order polymorphic lambda-calculus with sized types and polarized subtyping, but without bounded quantification. His subtyping algorithm uses weak head normal forms and is proven complete purely syntactically as in this work. The lexicographic induction on kinds and constructors, which in this article justifies hereditary substitution, is used by the first author to prove a substitution property for algorithmic subtyping, which entails completeness. It remains open whether this argument scales to bounded quantification.

Future work. Steffen has extended his work to $F^{\omega}_{<:}$ with polarities, i. e., co-, contra-, and mixed-variant type constructors are distinguished. For this extension, only backtracking algorithms exist [Ste98, p. 102]; we would like to see whether our proof technique can simplify its metatheory as well. The same question arises for the extension $\mathcal{F}^{\omega}_{\leq}$ by bounded abstraction.

Acknowledgments. The research has been partially supported by the EU coordination action TYPES (510996).

References

[Abe06a] A. Abel. Implementing a normalizer using sized heterogeneous types. In C. McBride and T. Uustalu, eds., *Wksh. on Mathematically Structured Functional Programming, MSFP 2006*. 2006.

[Abe06b] A. Abel. Polarized subtyping for sized types. In D. Grigoriev, J. Harrison, and E. A. Hirsch, eds., *Proc. of the 1st Int. Computer Science Symposium in Russia, CSR 2006*, vol. 3967 of *Lect. Notes in Comput. Sci.*, pp. 381–392. Springer-Verlag, 2006.

[AC96] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[AC97] R. Amadio and P.-L. Curien. *Domains and Lambda Calculi*. Cambridge University Press, 1997.

[Ada05] R. Adams. *A Modular Hierarchy of Logical Frameworks*. Ph.D. thesis, University of Manchester, 2005.

[Car88] L. Cardelli. Structural subtyping and the notion of power type. In *Proc. of the 15th ACM Symp. on Principles of Programming Languages, POPL'88*, pp. 70–79. 1988.

[CG03] A. B. Compagnoni and H. Goguen. Typed operational semantics for higher-order subtyping. *Inf. Comput.*, 184(2):242–297, 2003.

[CG06] A. Compagnoni and H. Goguen. Anti-symmetry of higher-order subtyping and equality by subtyping. *Math. Struct. in Comput. Sci.*, 16:41–65, 2006.

[CG07] A. Compagnoni and H. Goguen. Subtyping à la Church. In E. Barendsen, V. Capretta, H. Geuvers, and M. Niqui, eds., *Reflections on Type Theory, λ-calculus, and the Mind. Essays dedicated to Henk Barendregt on the Occasion of his 60th Birthday*. Radboud University Nijmegen, 2007.

[Com95] A. B. Compagnoni. Decidability of higher-order subtyping with intersection types. In L. Pacholski and J. Tiuryn, eds., *Computer Science Logic, 8th Int. Wksh., CSL '94*, vol. 933 of *Lect. Notes in Comput. Sci.*, pp. 46–60. Springer-Verlag, 1995.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, vol. 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[HL07] R. Harper and D. Licata. Mechanizing metatheory in a logical framework. *J. Func. Program.*, 17(4–5):613–673, 2007.

[LCH07] D. K. Lee, K. Crary, and R. Harper. Towards a mechanized metatheory of Standard ML. In M. Hofmann and M. Felleisen, eds., *Proc. of the 34th ACM Symp. on Principles of Programming Languages, POPL 2007*, pp. 173–184. ACM Press, 2007.

[Lév76] J.-J. Lévy. An algebraic interpretation of the $\lambda\beta$K-calculus; and an application of a labelled λ-calculus. *Theor. Comput. Sci.*, 2(1):97–114, 1976.

[Mit90] J. C. Mitchell. Toward a typed foundation for method specialization and inheritance. In *Proc. of the 17th ACM Symp. on Principles of Programming Languages, POPL'90*, pp. 109–124. 1990.

[Pie92] B. C. Pierce. Bounded quantification is undecidable. In *POPL*, pp. 305–315. 1992.

[Pie02] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[Pra65] D. Prawitz. *Natural Deduction*. Almqvist & Wiksell, Stockholm, 1965. Republication by Dover Publications Inc., Mineola, New York, 2006.

[PS97] B. C. Pierce and M. Steffen. Higher order subtyping. *Theor. Comput. Sci.*, 176(1,2):235–282, 1997.

[Rod07] D. Rodriguez. *Algorithmic Subtyping for Higher Order Bounded Quantification*. Diploma thesis, LMU Munich, 2007.

[Ste98] M. Steffen. *Polarized Higher-Order Subtyping*. Ph.D. thesis, Technische Fakultät, Universität Erlangen, 1998.

[WC+03] K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A concurrent logical framework I: Judgements and properties. Tech. rep., School of Computer Science, Carnegie Mellon University, Pittsburgh, 2003.