Decidable Higher Order Subtyping

Adriana Compagnoni

abc@dcs.ed.ac.uk

Department of Computer Science, University of Edinburgh
The King's Buildings, Edinburgh, EH9 3JZ, United Kingdom
Tel: (+44) (131) 650-1000 Fax: (+44) (131) 667-7209

Abstract

This paper establishes the decidability of typechecking in $F^{\omega}_{\wedge}$, a typed lambda calculus combining higher-order polymorphism, subtyping, and intersection types. It contains the first proof of decidability of subtyping for a higher-order system.

1 Introduction

The system $F^{\omega}_{\wedge}$ (F-omega-meet) was first introduced in [16], where it was shown to be rich enough to provide a typed model of object oriented programming with multiple inheritance. The system $F^{\omega}_{\wedge}$ is an extension of F^{ω} [22] with bounded quantification and intersection types, which can be seen as a natural generalization of the type disciplines present in the current literature, for example in [19, 26, 27, 11]. Systems including either subtyping or intersection types or both have been widely studied for many years. What follows is not intended to be an exhaustive description, but a framework for the present work.

First-order type disciplines with intersection types have been investigated by the group in Torino [18, 2] and elsewhere (see [10] for background and further references). A second-order λ-calculus with intersection types was studied in [26]. Systems including subtyping were present in [9, 5]. Higher order generalizations of subtyping appear in [4, 17, 25, 3].

Lambda calculi syntax is often presented by several interdependent relations or judgements (such as context formation, kinding, and typing, for example), and we say that a given lambda calculus is decidable if its typing relation is, but because of the interdependence of judgements it actually means the decidability of all the interwoven relations.

The system $F_{\leq}$, a second-order λ-calculus with bounded quantification, was studied in [21]. In [26] it was proved that typing in $F_{\leq}$ was undecidable and that undecidability was caused by the subtyping relation, the rule for bounded quantification being responsible for the failure.

An alternative rule for subtyping quantified types was presented in [11] in an attempt to find a decidable system, and the decidability of subtyping was proved for an extension of system F with bounded polymorphism, but different from $F_{\leq}$. Unfortunately, the typing relation fails to satisfy the minimal type property. This failure, discovered by Ghelli, introduces serious problems in type checking and type inference. At the moment it is not clear how to solve them or, even more problematic, whether the typing relation is decidable. The solution we chose here to overcome this problem is to replace the subtyping rule between quantifiers by the corresponding rule of Cardelli and Wegner's kernel *Fun* [9].

Because $F^{\omega}_{\wedge}$ has reduction on types we introduce a conversion rule that includes inter-convertible types in the subtype relation. Therefore, our subtyping relation relates types of a more expressive type system than that presented in [11]. In fact, treating the interaction between interface refinement and encapsulation of objects, in object oriented programming, has required higher-order generalizations of subtyping: the F-bounded quantification of Canning, Cook, Hill, Olthoff and Mitchell [4] or system $F^{\omega}_{\leq}$ [6, 8, 7, 25, 3].

In this paper we give a positive answer to the decidability of typing in the presence of $\beta\wedge$-convertible types and subtyping. We prove that subtyping in $F^{\omega}_{\wedge}$ is decidable, which *a fortiori* gives the decidability of subtyping for the $F^{\omega}_{\leq}$ fragment because the former is a conservative extension of the latter – namely, each subtyping statement derivable in $F^{\omega}_{\wedge}$ containing no intersections other than the empty ones is also derivable in $F^{\omega}_{\leq}$. A major task in establishing the decidability result for our system is in proving that subtyping is decidable.

We use a definition of $F^{\omega}_{\wedge}$ from [15, 12] that differs from the one introduced in [16] in two ways. First, the ill-behaved Castagna and Pierce quantifier rule has been replaced by the Cardelli and Wegner rule. Secondly, we introduce a richer notion of reduction on types, and thereby the four distributivity rules become particular cases of the conversion rule. This alternative (and equivalent) presentation provides a different view of the system that is the key to proving the decidability of subtyping.

This new perspective suggests that to prove the decidability of subtyping it is enough to concentrate on types in normal form. Note that the solution cannot be as simple as to restrict the subtyping rules of $F^{\omega}_{\wedge}$ to handle only types in normal form and replace conversion by reflexivity. The following is a good example of the intricacies of the problem to be solved. Consider the context $\Gamma \equiv W{:}K,\ X \leq \Lambda Y{:}K.Y{:}K{\to}K,\ Z \leq X{:}K{\to}K$, where every type is in normal form; observe that X and Z are subtypes of the identity on K. Then $\Gamma \vdash X(Z\,W) \leq W$, which is also in normal form, is not derivable without using conversion, i.e. without performing any β-reduction steps.

The subtyping rules of $F^{\omega}_{\wedge}$ are not syntax directed, in the sense that the form of a derivable subtyping statement does not uniquely determine the last rule of its derivation. A deterministic version of the subtyping relation, $AlgF^{\omega}_{\wedge}$, is presented in [15, 12]. There it is proved to be equivalent to the original presentation on well-formed types. The system $AlgF^{\omega}_{\wedge}$ considers only types in normal form, and will be our starting point to prove the decidability of subtyping.

In this paper we establish the decidability of subtyping in $F^{\omega}_{\wedge}$ by proving that the algorithm described by $AlgF^{\omega}_{\wedge}$ terminates, which is equivalent to showing that the definition

of the $AlgF^{\omega}_{\wedge}$ is well-founded. We discuss this further in section 5.3.

In the algorithm, checking whether $\Gamma \vdash_{Alg} S\,T \leq A$ is reduced to checking if $\Gamma \vdash_{Alg} lub_{\Gamma}(S\,T)^{\mathrm{nf}} \leq A$, where $lub_{\Gamma}(S\,T)$ substitutes the leftmost innermost variable of $S\,T$ by its bound in Γ. Such replacements may produce a term that is not in normal form, in which case we normalize it afterwards. (In an alternative notation used in [1], $\Gamma \vdash_{Alg} X S_1 \cdots S_n \leq A$ is reduced to checking if $\Gamma \vdash_{Alg} (\Gamma(X) S_1 \cdots S_n)^{\mathrm{nf}} \leq A$.) The main problem here is that the size of the types to be examined in the recursive call may not decrease. This indicates that the proof of termination of the algorithm is not immediate. In particular, the proof of termination for the second order calculus presented in [11] cannot be modified to serve our purposes, because of the interaction between reduction and the substitution of type variables by their bounds in our system. See section 5.3 for more details.

Typing and subtyping are defined using context formation and kinding judgements, therefore in section 4 we show that the latter are decidable.

Finally we prove in section 8 that typing is decidable. To illustrate the problem of typing consider the following situation. Imagine trying to type $f\,a$ in different contexts. We know we can type an application if we can find an arrow type for f whose domain types a. In the context Γ, $X{\leq}S{\rightarrow}T{:}\star$, $f{:}X$, $a{:}S$, we need to use that the type X, with which f is declared, is a subtype of $S{\rightarrow}T$. Note that X, the minimal type of f, does not have enough structural information to type an application. Therefore, one needs to "climb" the subtyping hierarchy until a type with enough information is found, in this case an arrow type.

If we want to type $f\,a$ with respect to the context $\Gamma' \equiv \Gamma$, $Z{\leq}\Lambda Y{:}\star.S{\rightarrow}Y{:}\star{\rightarrow}\star$, $W{\leq}Z{:}\star{\rightarrow}\star$, $f{:}WT$, $a{:}S$, we have to use that WT is a subtype of $S{\rightarrow}T$, which involves replacing W by Z, then Z by $\Lambda Y{:}\star.S{\rightarrow}Y$ in WT, and finishing with a step of β-reduction. Since we need to continue making replacements and reduction steps until we find an arrow type, this procedure is slightly more complicated than finding the $lub_{\Gamma'}(WT)$, which is ZT. The situation is even more twisty because of the presence of intersection types, but we leave that for section 8. This suggests that proving the termination of typechecking is not immediate, and that we will need to use similar ideas to those needed to prove the termination of subtype checking.

This paper extends the decidability result of [13] for subtyping to the typing relation.

2 Syntax of $F^{\omega}_{\wedge}$

We now present the rules for kinding, subtyping, and typing in $F^{\omega}_{\wedge}$. They are organized as proof systems for four interdependent judgement forms:

$\Gamma \vdash \mathrm{ok}$	well-formed context
$\Gamma \vdash T : K$	well-kinded type
$\Gamma \vdash S \leq T$	subtype
$\Gamma \vdash e : T$	well-typed term.

We sometimes use the metavariable Σ to range over statements (right-hand sides of judgements) of any of these four forms.

2.1 Syntactic Categories

The *kinds* of $F^\omega_\wedge$ are those of F^ω: the kind $\star$ of proper types and the kinds $K_1 \to K_2$ of functions on types (sometimes called type operators).

$$\mathbb{K} ::= \star \mid \mathbb{K} \to \mathbb{K}$$

The language of *types* of $F^\omega_\wedge$ is a straightforward higher-order extension of $F_\leq$, Cardelli and Wegner's second-order calculus of bounded quantification. Like $F_\leq$, it includes type variables (written X), function types ($T \to T'$), and polymorphic types ($\forall X{\leq}T{:}K.T'$), in which the bound type variable X ranges over all subtypes of the upper bound T. Moreover, like F^ω, we allow types to be abstracted on types ($\Lambda X{:}K.T$) and applied to argument types ($T\,T'$); in effect, these forms introduce a simply typed λ-calculus at the level of types. Finally, we allow arbitrary finite intersections ($\bigwedge^K[T_1..T_n]$), where all the T_i's are members of the same kind K.

$$\mathbb{T} ::= \mid X \mid \mathbb{T} \to \mathbb{T} \mid \forall X{\leq}\mathbb{T}{:}\mathbb{K}.\mathbb{T} \mid \Lambda X{:}\mathbb{K}.\mathbb{T} \mid \mathbb{T}\,\mathbb{T} \mid \textstyle\bigwedge^{\mathbb{K}}[\mathbb{T}..\mathbb{T}]$$

We use the abbreviation $\top^K$ for nullary intersections and sometimes $X{:}K$ for $X \leq \top^K{:}K$.

$$\top^K \;\equiv\; \textstyle\bigwedge^K[\,] \qquad X{:}K \;\equiv\; X \leq \top^K{:}K$$

We drop the maximal type *Top* of $F_\leq$, since its role is played here by the empty intersection $\top^\star$. For technical convenience, we provide kind annotations on bound variables and intersections so that every type has an "obvious kind," which can be read off directly from its structure and the kind declarations in the context.

The language of terms includes the variables (x), applications ($e\,e$) , and functional abstractions ($\lambda x{:}T.e$) of the simply typed λ-calculus, plus the type abstraction ($\lambda X{\leq}T{:}K.e$) and application ($e\,T$) of F^ω. As in $F_\leq$, each type variable is given an upper bound at the point where it is introduced.

Intersection types are introduced by expressions of the form "for($X{\in}T_1..T_n$)e", which can be read as instructions to the type-checker to analyze the expression e separately under the assumptions $X \equiv T_1$, $X \equiv T_2$, $\ldots$, $X \equiv T_n$ and conjoin the results. For example, if $+ : \text{Int}\to\text{Int}\to\text{Int} \wedge \text{Real}\to\text{Real}\to\text{Real}$, then we can derive

$$\text{for}(X{\in}\text{Int, Real})\lambda x{:}X.x + x \quad : \quad \text{Int}\to\text{Int} \wedge \text{Real}\to\text{Real}.$$

$$e ::= x \mid \lambda x{:}\mathbb{T}.e \mid e\,e \mid \lambda X{\leq}\mathbb{T}{:}\mathbb{K}.e \mid e\,\mathbb{T} \mid \text{for}(X{\in}\mathbb{T}..\mathbb{T})e$$

The operational semantics of $F^\omega_\wedge$ is given by the following reduction rules on types and terms.

Definition 2.1.1 (*Reduction rules for types*)

1. $(\Lambda X{:}K.T_1)T_2 \rightarrow_{\beta\wedge} T_1[X{\leftarrow}T_2]$
2. $S \rightarrow \bigwedge^{\star}[T_1..T_n] \rightarrow_{\beta\wedge} \bigwedge^{\star}[S{\rightarrow}T_1 \,..\, S{\rightarrow}T_n]$
3. $\forall X{\leq}S{:}K.\bigwedge^{\star}[T_1..T_n] \rightarrow_{\beta\wedge} \bigwedge^{\star}[\forall X{\leq}S{:}K.T_1 \,..\, \forall X{\leq}S{:}K.T_n]$
4. $\Lambda X{:}K_1.\bigwedge^{K_2}[T_1..T_n] \rightarrow_{\beta\wedge} \bigwedge^{K_1\rightarrow K_2}[\Lambda X{:}K_1.T_1 \,..\, \Lambda X{:}K_1.T_n]$
5. $(\bigwedge^{K_1\rightarrow K_2}[T_1..T_n])\,U \rightarrow_{\beta\wedge} \bigwedge^{K_2}[T_1\,U \,..\, T_n\,U]$
6. $\bigwedge^{K}[T_1 \,..\, \bigwedge^{K}[S_1..S_n] \,..\, T_m] \rightarrow_{\beta\wedge} \bigwedge^{K}[T_1 \,..\, S_1..S_n \,..\, T_m]$

The first rule is the usual β-reduction rule for types. Rules 2 through 5 express the fact that intersections in positive positions distribute with respect to the other type constructors. Rule 6 states that intersection is an associative operator. The left-hand side of each reduction rule is a *redex* and the right-hand side its *reduct*. The relation $\rightarrow_{\beta\wedge}$ is extended so as to become a compatible relation with respect to type formation, $\twoheadrightarrow_{\beta\wedge}$ is the transitive and reflexive closure of $\rightarrow_{\beta\wedge}$, and $=_{\beta\wedge}$ is the least equivalence relation containing $\rightarrow_{\beta\wedge}$. The capture-avoiding substitution of S for X in T is written $T[X{\leftarrow}S]$. Substitution is written similarly for terms, and is extended point-wise to contexts. The $\beta\wedge$-normal form of a type S is written S^{nf}, and is extended point-wise to contexts.

Definition 2.1.2 (*Reduction rules for terms*)

1. $(\lambda x{:}T_1.e_1)e_2 \rightarrow_{\beta\mathrm{fors}} e_1[x{\leftarrow}e_2]$
2. $(\lambda X{\leq}T_1{:}K_1.e)T \rightarrow_{\beta\mathrm{fors}} e[X{\leftarrow}T]$
3. $(\mathrm{for}(X{\in}T_1..T_n)e_1)e_2 \rightarrow_{\beta\mathrm{fors}} \mathrm{for}(X{\in}T_1..T_n)(e_1\,e_2)$
4. $\mathrm{for}(X{\in}T_1..T_n)e \rightarrow_{\beta\mathrm{fors}} e$, if $X \notin \mathrm{FV}(e)$

Rules 1 and 2 are the β-reductions on terms. Rule 3 says that the for constructor can be pushed to the outermost level. We consider the reduction defined by rules 1 through 3 as $\rightarrow_{\beta\mathrm{for}}$ and the one defined by 4 as $\rightarrow_{\mathrm{s}}$ (s comes from simplification). The left-hand side of each reduction rule is a *redex* and the right-hand side its *reduct*. The relation $\rightarrow_{\beta\mathrm{fors}}$ is extended so as to become a compatible relation with respect to term formation, $\twoheadrightarrow_{\beta\mathrm{fors}}$ is the transitive reflexive closure of $\rightarrow_{\beta\mathrm{fors}}$, and $=_{\beta\mathrm{fors}}$ is the least equivalence relation containing $\rightarrow_{\beta\mathrm{fors}}$.

2.2 Contexts

A *context* Γ is a finite sequence of typing and subtyping assumptions for a set of term and type variables.

The empty context is written $\emptyset$. Term variable bindings have the form $x{:}T$; type variable bindings have the form $X{\leq}T{:}K$, where T is the upper bound of X and K is the kind of T.

$$\Gamma ::= \emptyset \,|\, \Gamma,\, x{:}T \,|\, \Gamma,\, X{\leq}T{:}K$$

When writing nonempty contexts, we omit the initial $\emptyset$. The domain of Γ is written $\mathrm{dom}(\Gamma)$. The functions $\mathrm{FV}(\text{—})$ and $\mathrm{FTV}(\text{—})$ give the sets of free term variables and free type variables of a term, type, or context. Since we are careful to ensure that no variable is bound more than once, we sometimes abuse notation and consider contexts as finite functions: $\Gamma(X)$ yields the bound of X in Γ, where X is implicitly asserted to be in $\mathrm{dom}(\Gamma)$.

Types, terms, contexts, statements, and derivations that differ only in the names of bound variables are considered identical. The underlying idea is that variables are de Bruijn indexes [20].

Definition 2.2.1 (*Closed*)

1. A term e is closed with respect to a context Γ if $\mathrm{FV}(e) \cup \mathrm{FTV}(e) \subseteq \mathrm{dom}(\Gamma)$.
2. A type T is closed with respect to a context Γ if $\mathrm{FTV}(T) \subseteq \mathrm{dom}(\Gamma)$.
3. A typing statement $\Gamma \vdash e : T$ is closed if e and T are closed with respect to Γ.
4. A kinding statement $\Gamma \vdash T : K$ is closed if T is closed with respect to Γ.
5. A subtyping statement $\Gamma \vdash S \leq T$ is closed if S and T are closed with respect to Γ.

We consider only closed typing statements. Observe that in the limit case of the rule T-Meet, when $n = 0$, not having the closure convention would allow nonsensical terms to be typed. On the other hand, the free variable lemma (lemma 3.2) guarantees that kinding statements are closed and the well-kindedness of subtyping (lemma 3.8) ensures that subtyping statements are closed as well.

2.3 Context Formation

The rules for well-formed contexts are the usual ones: a start rule for the empty context and rules allowing a given well-formed context to be extended with either a term variable binding or a type variable binding.

$$\emptyset \vdash \mathrm{ok} \qquad \text{(C-Empty)}$$

$$\frac{\Gamma \vdash T : \star \qquad x \notin \mathrm{dom}(\Gamma)}{\Gamma,\, x{:}T \vdash \mathrm{ok}} \qquad \text{(C-Var)}$$

$$\frac{\Gamma \vdash T : K \qquad X \notin \mathrm{dom}(\Gamma)}{\Gamma,\ X{\leq}T{:}K \vdash \mathrm{ok}} \tag{C-TVar}$$

2.4 Type Formation

For each type constructor, we give a rule specifying how it can be used to build well-formed type expressions. The critical rules are K-OAbs and K-OApp, which form type abstractions and type applications (essentially as in a simply typed λ-calculus).

The well-formedness premise $\Gamma \vdash \mathrm{ok}$ in K-Meet (and in T-Meet below) is required for the case where $n = 0$.

$$\frac{\Gamma_1,\ X{\leq}T{:}K,\ \Gamma_2 \vdash \mathrm{ok}}{\Gamma_1,\ X{\leq}T{:}K,\ \Gamma_2 \vdash X : K} \tag{K-TVar}$$

$$\frac{\Gamma \vdash T_1 : \star \qquad \Gamma \vdash T_2 : \star}{\Gamma \vdash T_1{\rightarrow}T_2 : \star} \tag{K-Arrow}$$

$$\frac{\Gamma,\ X{\leq}T_1{:}K_1 \vdash T_2 : \star}{\Gamma \vdash \forall X{\leq}T_1{:}K_1.T_2 : \star} \tag{K-All}$$

$$\frac{\Gamma,\ X{:}K_1 \vdash T_2 : K_2}{\Gamma \vdash \Lambda X{:}K_1.T_2 : K_1{\rightarrow}K_2} \tag{K-OAbs}$$

$$\frac{\Gamma \vdash S : K_1{\rightarrow}K_2 \qquad \Gamma \vdash T : K_1}{\Gamma \vdash S\,T : K_2} \tag{K-OApp}$$

$$\frac{\Gamma \vdash \mathrm{ok} \qquad \text{for each } i \in \{1..n\}\,,\ \Gamma \vdash T_i : K}{\Gamma \vdash \bigwedge^{K}[T_1..T_n] : K} \tag{K-Meet}$$

2.5 Subtyping

The rules defining the subtype relation are a natural extension of familiar calculi of bounded quantification. Aside from some extra well-formedness conditions, the rules S-Trans, S-TVar, and S-Arrow are the same as in the usual, second-order case. Rules S-OAbs and S-OApp extend the subtype relation point-wise to kinds other than $\star$. The rule of type conversion in F^{ω}, that is, if $\Gamma \vdash e : T$ and $T =_{\beta} T'$ then $\Gamma \vdash e : T'$, is captured here as the subtyping rule S-Conv, which also gives reflexivity as a special case. The rule S-All is the rule of Cardelli and Wegner's *Fun* language [9] in which the bounds of the quantifiers are equal. Rules S-Meet-G and S-Meet-LB specify that an intersection of a set of types is the set's order-theoretic greatest lower bound.

$$\frac{\Gamma \vdash S : K \qquad \Gamma \vdash T : K \qquad S =_{\beta\wedge} T}{\Gamma \vdash S \leq T} \tag{S-Conv}$$

$$\frac{\Gamma \vdash S \leq T \qquad \Gamma \vdash T \leq U}{\Gamma \vdash S \leq U} \tag{S-Trans}$$

$$\frac{\Gamma_1,\ X{\leq}T{:}K,\ \Gamma_2 \vdash \mathrm{ok}}{\Gamma_1,\ X{\leq}T{:}K,\ \Gamma_2 \vdash X \leq T} \tag{S-TVar}$$

$$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \vdash S_2 \leq T_2 \qquad \Gamma \vdash S_1 \rightarrow S_2 : \star}{\Gamma \vdash S_1 \rightarrow S_2 \leq T_1 \rightarrow T_2} \quad \text{(S-Arrow)}$$

$$\frac{\Gamma,\ X{\leq}U{:}K \vdash S \leq T \qquad \Gamma \vdash \forall X{\leq}U{:}K.S : \star}{\Gamma \vdash \forall X{\leq}U{:}K.S \leq \forall X{\leq}U{:}K.T} \quad \text{(S-All)}$$

$$\frac{\Gamma,\ X{:}K \vdash S \leq T}{\Gamma \vdash \Lambda X{:}K.S \leq \Lambda X{:}K.T} \quad \text{(S-OAbs)}$$

$$\frac{\Gamma \vdash S \leq T \qquad \Gamma \vdash S\,U : K}{\Gamma \vdash S\,U \leq T\,U} \quad \text{(S-OApp)}$$

$$\frac{\text{for each } i \in \{1..n\}\,,\ \Gamma \vdash S \leq T_i \qquad \Gamma \vdash S : K}{\Gamma \vdash S \leq \bigwedge^K [T_1..T_n]} \quad \text{(S-Meet-G)}$$

$$\frac{\Gamma \vdash \bigwedge^K [T_1..T_n] : K}{\Gamma \vdash \bigwedge^K [T_1..T_n] \leq T_i} \quad \text{(S-Meet-LB)}$$

2.6 Term Formation

Except for T-Meet and T-For, the term formation rules are precisely those of the second-order calculus of bounded quantification. T-For provides for type checking under any of a set of alternate assumptions. For each S_i, the type derived for the instance of the body e when X is replaced by S_i is a valid type of the for expression itself. The T-Meet rule can then be used to collect these separate typings into a single intersection. Type-theoretically, T-Meet is the introduction rule for the $\wedge$ constructor; the corresponding elimination rule need not be given explicitly, since it follows from T-Subsumption and S-Meet-LB.

$$\frac{\Gamma_1,\ x{:}T,\ \Gamma_2 \vdash \text{ok}}{\Gamma_1,\ x{:}T,\ \Gamma_2 \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma,\ x{:}T_1 \vdash e : T_2}{\Gamma \vdash \lambda x{:}T_1.e : T_1 \rightarrow T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash f : T_1 \rightarrow T_2 \qquad \Gamma \vdash a : T_1}{\Gamma \vdash f\,a : T_2} \quad \text{(T-App)}$$

$$\frac{\Gamma,\ X{\leq}T_1{:}K_1 \vdash e : T_2}{\Gamma \vdash \lambda X{\leq}T_1{:}K_1.e : \forall X{\leq}T_1{:}K_1.T_2} \quad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash f : \forall X{\leq}T_1{:}K_1.T_2 \qquad \Gamma \vdash S \leq T_1}{\Gamma \vdash f\,S : T_2[X{\leftarrow}S]} \quad \text{(T-TApp)}$$

$$\frac{\Gamma \vdash e[X{\leftarrow}S]{:}T \qquad S : \{S_1..S_n\}}{\Gamma \vdash \text{for}(X{\in}S_1..S_n)e : T} \quad \text{(T-For)}$$

$$\frac{\Gamma \vdash \text{ok} \qquad \text{for each } i \in \{1..n\}\,,\ \Gamma \vdash e : T_i}{\Gamma \vdash e : \bigwedge^{\star}[T_1..T_n]} \quad \text{(T-Meet)}$$

$$\frac{\Gamma \vdash e : S \qquad \Gamma \vdash S \leq T}{\Gamma \vdash e : T} \qquad \text{(T-Subsumption)}$$

Most of the rules include premises which have two rather different sorts: *structural premises*, which play an essential role in giving the rule its intended semantic force, and *well-formation premises*, which ensure that the entities named in the rule are of the expected sorts.

We sometimes omit well-formation premises that can be derived from others. For example, in the rule S-Arrow, we drop the premise $\Gamma \vdash T_1 \rightarrow T_2 : \star$, since it follows from $\Gamma \vdash S_1 \rightarrow S_2 : \star$ using structural properties.

3 Properties

In this section we list properties whose proofs can be found in [12].

Lemma 3.1 If $\Gamma \vdash \Sigma$ and Γ_1 is a prefix of Γ, then $\Gamma_1 \vdash \text{ok}$ as a subderivation. Moreover, except for the case $\Gamma_1 \equiv \Gamma$ and $\Sigma \equiv \text{ok}$, the subderivation is strictly shorter.

Lemma 3.2 (*Free variables*)

1. If $\Gamma \vdash T : K$, then $\text{FTV}(T) \subseteq \text{dom}(\Gamma)$.
2. If $\Gamma \vdash \text{ok}$, then each variable or type variable in $\text{dom}(\Gamma)$ is declared only once.

Lemma 3.3 (*Generation for context judgements*)

1. If $\Gamma_1, X{\leq}T{:}K, \Gamma_2 \vdash \text{ok}$, then $\Gamma_1 \vdash T : K$ by a proper subderivation.
2. If $\Gamma_1, x{:}T, \Gamma_2 \vdash \text{ok}$, then $\Gamma_1 \vdash T : \star$ by a proper subderivation.

Proposition 3.4 (*Generation for kinding*)

1. $\Gamma \vdash X : K$ implies $\Gamma \equiv \Gamma_1, X{\leq}T{:}K, \Gamma_2$ for some Γ_1, T, and Γ_2.
2. $\Gamma \vdash T_1 \rightarrow T_2 : K$ implies $K \equiv \star$ and $\Gamma \vdash T_1, T_2 : \star$.
3. $\Gamma \vdash \forall X{\leq}T_1{:}K_1.T_2 : K$ implies $K \equiv \star$ and $\Gamma, X{\leq}T_1{:}K_1 \vdash T_2 : \star$.
4. $\Gamma \vdash \Lambda(X{:}K_1)T_2 : K$ implies $K \equiv K_1 \rightarrow K_2$ and $\Gamma, X{\leq}\top^{K_1}{:}K_1 \vdash T_2 : K_2$, for some K_2.
5. $\Gamma \vdash S\,T : K$ implies $\Gamma \vdash S : K' \rightarrow K$ and $\Gamma \vdash T : K'$, for some K'.
6. $\Gamma \vdash \bigwedge^K [T_1..T_n] : K'$ implies $K \equiv K'$ and $\Gamma \vdash \text{ok}$ and $\Gamma \vdash T_i : K$ for each i.

If one tries to prove Weakening (Corollary 3.7) directly by induction on derivations the induction hypothesis is too weak in the cases for K-All and S-OAbs, for example. We adapt McKinna and Pollack's idea of *renaming* [24] to overcome this problem.

Definition 3.5 (*Parallel Substitution*) A *parallel substitution* γ for Γ is an assignment of types to type variables in $\mathrm{dom}(\Gamma)$ and terms to term variables in $\mathrm{dom}(\Gamma)$. A *renaming* for Γ in Δ is a parallel substitution γ from variables to variables such that

- for every $x{:}A$ in Γ, $\gamma(x){:}A[\gamma]$ is in Δ, and
- for every $X{\leq}T{:}K$ in Γ, $\gamma(X){\leq}A[\gamma]{:}K$ is in Δ.

We write $\Sigma[\gamma]$ for the result of performing the substitution γ in the judgement Σ. The renaming $\gamma\{x \mapsto y\}$ maps x to y and behaves like γ elsewhere, similarly for type variables.

Lemma 3.6 (*Renaming*) If $\Delta \vdash \mathrm{ok}$ and γ is a renaming for Γ in Δ then $\Gamma \vdash \Sigma$ implies $\Delta \vdash \Sigma[\gamma]$.

Corollary 3.7 (*Weakening/Permutation*) Let Γ and Γ' be contexts such that $\Gamma \subseteq \Gamma'$ and $\Gamma' \vdash \mathrm{ok}$. Then $\Gamma \vdash \Sigma$ implies $\Gamma' \vdash \Sigma$.

Proposition 3.8 (*Well-kindedness of subtyping*) If $\Gamma \vdash S \leq T$, then $\Gamma \vdash S : K$ and $\Gamma \vdash T : K$ for some K.

Proposition 3.9 (*Well-kindedness of typing*) If $\Gamma \vdash e : T$, then $\Gamma \vdash T : \star$.

4 Decidability of context formation and kinding

The decidability of kinding is used to prove the decidability of subtyping in section 5.3, and the decidability of ok judgements is used to prove the decidability of typechecking in section 8. We want to define a measure (size) on judgements to show that the complexity of the hypothesis in a kinding derivation rule is smaller than the complexity of the conclusion. Given that kinding rules may depend on ok judgements we need to define a measure for both kinding and well-formation judgements. Rules C-TVar and C-Var suggest that the size of ok should be bigger than 0 (let it be 1). The rule K-Meet in the empty case, suggests that the size of $\top^K$ should be bigger than the size of ok, while K-TVar indicates that the size of a variable should be bigger than the size of ok. With all these ideas in mind we define the following measure.

Definition 4.1 (*Size*)

1. The size of a type expression T, $\mathit{size}_t(T)$, is defined as follows.
   (a) $\mathit{size}_t(X) = 2$,
   (b) $\mathit{size}_t(S{\to}T) = \mathit{size}_t(\forall X{\leq}S{:}K.T) = \mathit{size}_t(S\,T) = \mathit{size}_t(S) + \mathit{size}_t(T) + 1$,
   (c) $\mathit{size}_t(\Lambda X{:}K.T) = \mathit{size}_t(T) + 3$,
   (d) $\mathit{size}_t(\bigwedge^K[T_1..T_n]) = 2 + \Sigma_{1\leq i\leq n}\mathit{size}_t(T_i)$.

2. The homomorphic extension to contexts, $size_c(\Gamma)$, is defined as follows.
   (a) $size_c(\emptyset) = 0$,
   (b) $size_c(\Gamma,\, X \leq T{:}K) = size_c(\Gamma,\, x{:}T) = size_c(\Gamma) + size_t(T)$.
3. The size of a subtyping, kinding, or ok judgement J, $size_j(J)$, is defined as follows.
   (a) $size_j(\Gamma \vdash \text{ok}) = size_c(\Gamma) + 1$,
   (b) $size_j(\Gamma \vdash T : K) = size_c(\Gamma) + size_t(T)$.
   (c) $size_j(\Gamma \vdash S \leq T) = size_c(\Gamma) + size_t(S) + size_t(T)$.

Lemma 4.2 (*Well-foundedness of context formation and kinding rules*)

1. For every kinding or ok judgement J, $size_j(\emptyset \vdash \text{ok}) \leq size_j(J)$.
2. If $\dfrac{J_1 \;..\; J_n}{J}$ is a kinding rule or a context formation rule, then $size_j(J_i) < size_j(J)$ for each $i \in \{1..n\}$.

Corollary 4.3 (*Decidability of context formation and kinding*)

1. For any context Γ it is decidable whether $\Gamma \vdash \text{ok}$.
2. For any context Γ, type expression T, and kind K, it is decidable whether $\Gamma \vdash T : K$.

Proof: Lemma 3.3 and proposition 3.4 imply that context formation rules and kinding rules determine an algorithm to check context judgements and kinding judgements, and lemma 4.2 implies that this algorithm terminates. □

5 Decidability of subtyping

In the solution for the second order lambda calculus presented in [26], the distributivity rules for intersection types are not considered as rewrite rules. For that reason, new syntactic categories have to be defined (composite and individual canonical types) and an auxiliary mapping (flattening) transforms a type into a canonical type. Our solution does not need either new syntactic categories or elaborate auxiliary mappings, since the role played there by canonical types is performed here by types in normal form.

After the work in [13] was completed, Steffen and Pierce proved a similar result for $F^{\omega}_{\leq}$ [28]. There are several differences between our work and the proof of decidability of subtyping in [28]. Our result is for a stronger system which also includes intersection types, and implies the result for $F^{\omega}_{\leq}$. Moreover, our proof of termination has the novel idea of using a choice operator to model the behavior of type variables during subtype checking.

An important property of derivation systems is the information that a derivable judgement contains about its proofs. This information is essential to produce results which not only state properties about the subproofs, but also help identify ill formed judgements. Consider the following example.

Example 5.1 In $F^{\omega}_{\wedge}$ we can prove: $W{:}K, X \leq \Lambda Y{:}K.Y{:}K{\to}K,\ Z \leq X{:}K{\to}K \vdash X(Z\,W) \leq W$. Note that X and Z are subtypes of the identity on K, therefore it makes sense for $X(Z\,W)$ to be a subtype of W. The derivation is as follows. Let $\Gamma \equiv W{:}K,\ X \leq \Lambda Y{:}K.Y{:}K{\to}K,\ Z \leq X{:}K{\to}K$. For the sake of readability we omit kinding judgements.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash X \leq \Lambda Y{:}K.Y}\ \text{S-TVar}}{\Gamma \vdash X(Z\,W) \leq (\Lambda Y{:}K.Y)Z\,W}\ \text{S-OApp} \quad \dfrac{(\Lambda Y{:}K.Y)Z\,W =_{\beta\wedge} Z\,W}{(\Lambda Y{:}K.Y)Z\,W \leq Z\,W}\ \text{S-Conv}}{\Gamma \vdash X(Z\,W) \leq Z\,W}\ \text{S-Trans}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash Z \leq X}\ \text{S-TVar} \quad \dfrac{\Gamma \vdash \mathrm{ok}}{\Gamma \vdash X \leq \Lambda Y{:}K.Y}\ \text{S-TVar}}{\Gamma \vdash Z \leq (\Lambda Y{:}K.Y)}\ \text{S-Trans}}{\Gamma \vdash Z\,W \leq (\Lambda Y{:}K.Y)W}\ \text{S-OApp} \quad \dfrac{(\Lambda Y{:}K.Y)W =_{\beta\wedge} W}{(\Lambda Y{:}K.Y)W \leq W}\ \text{S-Conv}}{\Gamma \vdash Z\,W \leq W}\ \text{S-Trans}$$

$$\dfrac{\Gamma \vdash X(Z\,W) \leq Z\,W \quad \Gamma \vdash Z\,W \leq W}{\Gamma \vdash X(Z\,W) \leq W}\ \text{S-Trans}$$

This simple example already shows that S-Trans erases information obtained by S-Conv that is not present in the conclusion any longer.

5.1 A subtype checking algorithm, $AlgF^{\omega}_{\wedge}$

Subtyping statements in $AlgF^{\omega}_{\wedge}$ are written $\Gamma \vdash_{Alg} S \leq T$, and S, T, and all types appearing in Γ are in $\beta\wedge$-normal form. A, B, and C range over types whose outermost constructor is not an intersection.

Remark 5.1.1 It is an immediate consequence of the $\beta\wedge$-reduction rules that, if T is in $\beta\wedge$-normal form, then T is either X, $S{\to}A$, $\forall X{\leq}S{:}K.A$, $\Lambda X{:}K.A$, $A\,S$ where A is not an abstraction, or $\bigwedge^{K}[A_1..A_n]$. We frequently use this notation as a reminder of the shape of types in normal form.

We now define $lub_{\Gamma}(S)$. In [12], it is proved that, when defined, it is the smallest type beyond S with respect to Γ.

Definition 5.1.2 (*Least strict Upper Bound*)

$$\begin{aligned} lub_{\Gamma}(X) &= \Gamma(X), \\ lub_{\Gamma}(T\,S) &= lub_{\Gamma}(T)\,S. \end{aligned}$$

Definition 5.1.3 (*$AlgF^{\omega}_{\wedge}$ subtyping rules*)

$$\frac{\Gamma \vdash X : K}{\Gamma \vdash_{Alg} X \leq X} \quad \text{(AlgS-TVarRefl)}$$

$$\frac{\Gamma \vdash T\,S : K}{\Gamma \vdash_{Alg} T\,S \leq T\,S} \quad \text{(AlgS-OAppRefl)}$$

$$\frac{\Gamma \vdash_{Alg} \Gamma(X) \leq A \qquad X \not\equiv A}{\Gamma \vdash_{Alg} X \leq A} \quad \text{(AlgS-TVar)}$$

$$\frac{\Gamma \vdash_{Alg} T \leq S \qquad \Gamma \vdash_{Alg} A \leq B \qquad \Gamma \vdash S{\to}A : \star}{\Gamma \vdash_{Alg} S{\to}A \leq T{\to}B} \quad \text{(AlgS-Arrow)}$$

$$\frac{\Gamma,\, X{\leq}S{:}K \vdash_{Alg} A \leq B \qquad \Gamma \vdash \forall X{\leq}S{:}K.A : \star}{\Gamma \vdash_{Alg} \forall X{\leq}S{:}K.A \leq \forall X{\leq}S{:}K.B} \quad \text{(AlgS-All)}$$

$$\frac{\Gamma,\, X{\leq}\top^K{:}K \vdash_{Alg} A \leq B}{\Gamma \vdash_{Alg} \Lambda X{:}K.A \leq \Lambda X{:}K.B} \quad \text{(AlgS-OAbs)}$$

$$\frac{\Gamma \vdash_{Alg} (lub_{\Gamma}(T\,S))^{\mathrm{nf}} \leq A \qquad \Gamma \vdash T\,S : K \qquad T\,S \not\equiv A}{\Gamma \vdash_{Alg} T\,S \leq A} \quad \text{(AlgS-OApp)}$$

$$\frac{\forall i \in \{1..m\}\, \Gamma \vdash_{Alg} A \leq A_i \qquad \Gamma \vdash A : K}{\Gamma \vdash_{Alg} A \leq \bigwedge^K [A_1..A_m]} \quad \text{(AlgS-}\forall\text{)}$$

$$\frac{\exists j \in \{1..n\}\, \Gamma \vdash_{Alg} A_j \leq A \qquad \forall k \in \{1..n\}\, \Gamma \vdash A_k : K}{\Gamma \vdash_{Alg} \bigwedge^K [A_1..A_n] \leq A} \quad \text{(AlgS-}\exists\text{)}$$

$$\frac{\forall i \in \{1..m\}\, \exists j \in \{1..n\}\, \Gamma \vdash_{Alg} A_j \leq B_i \qquad \forall k \in \{1..n\}\, \Gamma \vdash A_k : K}{\Gamma \vdash_{Alg} \bigwedge^K [A_1..A_n] \leq \bigwedge^K [B_1..B_m]} \quad \text{(AlgS-}\forall\exists\text{)}$$

In [15, 12] it was proved that this algorithm is equivalent to the original subtyping system.

Proposition 5.1.4 (*Equivalence of ordinary and algorithmic subtyping*)
Let $\Gamma \vdash S : K$ and $\Gamma \vdash T : K$. Then $\Gamma \vdash S \leq T$ if and only if $\Gamma^{\mathrm{nf}} \vdash_{Alg} S^{\mathrm{nf}} \leq T^{\mathrm{nf}}$.

Now first let's go back to our example.

5.2 Example

In this section, we give the derivation in $AlgF^{\omega}_{\wedge}$ of the example 5.1 (also mentioned in the introduction). Let $\Gamma \equiv W{:}K,\, X \leq \Lambda Y{:}K.Y{:}K{\to}K,\, Z \leq X{:}K{\to}K$. We present a proof in the normal system of $\Gamma^{nf} \vdash_n X(Z\,W)^{nf} \leq W^{nf}$, which is the translation of $\Gamma \vdash X(Z\,W) \leq W$.

Observe that $\Gamma^{nf} \equiv \Gamma$,
$(X(Z\,W))^{nf} \equiv X(Z\,W)$, and
$W^{nf} \equiv W$.

For the sake of readability we omit kinding judgements. The derivation in normal form in $AlgF^{\omega}_{\wedge}$ is substantially shorter than the one in $F^{\omega}_{\wedge}$ shown in section 5.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash W : K}{\Gamma \vdash_n ((\Lambda Y{:}K.Y)W)^{nf} \leq W}\ \text{AS-Refl}}{\Gamma \vdash_n X\,W \leq W}\ \text{AS-OApp}}{\Gamma \vdash_n ((\Lambda Y{:}K.Y)(Z\,W))^{nf} \leq W}\ \text{AS-OApp}}{\Gamma \vdash_n X(Z\,W) \leq W}\ \text{AS-OApp}$$

5.3 Termination of subtype checking

To establish the decidability of the subtyping relation of $F^{\omega}_{\wedge}$ we prove the termination or well-foundedness of the relation defined by the $AlgF^{\omega}_{\wedge}$ subtyping rules. We show this by reducing the well-foundedness of $AlgF^{\omega}_{\wedge}$ to the strong normalization property of the $\rightarrow_{\beta\wedge+}$ relation.

We begin by extending the language of types with the constructor + as follows.

$\mathbb{T}^+$	$::=$	X	type variable
	$\mid$	$\mathbb{T}^+ \rightarrow \mathbb{T}^+$	function type
	$\mid$	$\forall(X{\leq}\mathbb{T}^+{:}\mathbb{K})\mathbb{T}^+$	polymorphic type
	$\mid$	$\Lambda(X{:}\mathbb{K})\mathbb{T}^+$	operator abstraction
	$\mid$	$\mathbb{T}^+\,\mathbb{T}^+$	operator application
	$\mid$	$\bigwedge^{\mathbb{K}}[\mathbb{T}^+..\mathbb{T}^+]$	intersection at kind $\mathbb{K}$
	$\mid$	$\mathbb{T}^+ + \mathbb{T}^+$	choice

Since we have enriched the language of types with a new type constructor, we need to extend our kinding judgements (section 2.4) with the following kinding rule.

$$\dfrac{\Gamma \vdash_+ S : K \qquad \Gamma \vdash_+ T : K}{\Gamma \vdash_+ S + T : K} \qquad \text{(K-Plus)}$$

The reduction $\rightarrow_{\beta\wedge+}$ is obtained from $\rightarrow_{\beta\wedge}$ by adding the reductions associated with the choice operator +, $S+T \rightarrow_{\beta\wedge+} S$ and $S+T \rightarrow_{\beta\wedge+} T$. We also need the corresponding kinding rule saying that $\Gamma \vdash S+T : K$ whenever $\Gamma \vdash S, T : K$. As far as we are aware, choice operators have not been used before to analyze subtyping.

Notation 5.3.1 We write + modulo commutativity and associativity.

We now define a new reduction $\rightarrow_{\beta\wedge+}$.

Definition 5.3.2 ($\rightarrow_{\beta\wedge+}$) The reduction on types $\rightarrow_{\beta\wedge+}$ is obtained from $\rightarrow_{\beta\wedge}$ (definition 2.1.1) by adding the following two rules:

1. $S + T \rightarrow_{\beta\wedge+} S$, and
2. $S + T \rightarrow_{\beta\wedge+} T$.

We also write $\rightarrow_+$ to refer to these two new reduction rules.

As usual, $\rightarrow_{\beta\wedge+}$ is extended to become a compatible relation with respect to type formation, $\twoheadrightarrow_{\beta\wedge+}$ is the reflexive, transitive closure of $\rightarrow_{\beta\wedge+}$, and $=_{\beta\wedge+}$ is the reflexive, symmetric, and transitive closure of $\rightarrow_{\beta\wedge+}$.

Proposition 5.3.3 (*Strong normalization for* $\rightarrow_{\beta\wedge+}$)
If $\Gamma \vdash_+ T : K$, then every $\beta\wedge+$-reduction sequence starting from T is finite.

Proof: The result follows using the strategy used to prove that the reduction $\rightarrow_{\beta\wedge}$ is strongly normalizing on well kinded types (see [15, 12]). We only need to modify the definition of saturated sets by adding the following closure condition:
if T, U, $R_1..R_n \in SN^+$, then $TR_1..R_n \in S$ and $UR_1..R_n \in S$ imply $(T + U)R_1..R_n \in S$.
□

Next, we define a measure for subtyping statements such that, given a subtyping rule, the measure of each hypothesis is smaller than that of the conclusion. Most measures for showing the well-foundedness of a relation defined by a set of inference rules involve a clever assignment of weights to judgements, often involving the number of symbols. We need a more sophisticated measure, since in AlgS-OApp it is not necessarily the case that the size of the hypothesis is smaller than the size of the conclusion.

We introduce a new mapping from types to types in the extended language in order to define a new measure on subtyping statements. To motivate the definition of this new measure, we analyze the behavior of type variables during subtype checking. Assume that we want to check if $\Gamma \vdash_{Alg} S \leq T$, where S is a variable or a type application. It can be the case that the judgement is obtained with an application of AlgS-TVar or AlgS-OApp, in which case we have to consider a new statement $\Gamma \vdash_{Alg} S' \leq T$, where S' is obtained from S by replacing a variable by its bound (and eventually normalizing). However, we do not replace every variable by its bound, as this would constitute an unsound operation with respect to subtyping. Consider the following example.

Example 5.3.4 Two unrelated variables may have the same bound.
$X{\leq}\top^\star{:}\star,\ Y{\leq}\top^\star{:}\star \not\vdash X \leq Y$, but
$X{\leq}\top^\star{:}\star,\ Y{\leq}\top^\star{:}\star \vdash \top^\star \leq \top^\star$.

Our new mapping, *plus*, includes in each type expression this nondeterministic behavior of its type variables.

Definition 5.3.5 (*plus*)
The mapping $plus_\Gamma : \mathbb{T} \rightarrow \mathbb{T}^+$ is defined as follows.

1. $plus_{\Gamma_1, X \leq T:K, \Gamma_2}(X) = X + plus_{\Gamma_1}(T)$,
2. $plus_\Gamma(T \rightarrow S) = plus_\Gamma(T) \rightarrow plus_\Gamma(S)$,
3. $plus_\Gamma(\forall X \leq T{:}K.S) = \forall X \leq plus_\Gamma(T){:}K.plus_{\Gamma, X \leq T:K}(S)$,
4. $plus_\Gamma(\Lambda X{:}K.S) = \Lambda X{:}K.plus_{\Gamma, X:K}(S)$,
5. $plus_\Gamma(S\,T) = plus_\Gamma(S)\,plus_\Gamma(T)$,
6. $plus_\Gamma(\bigwedge^K[S_1..S_n]) = \bigwedge^K[plus_\Gamma(S_1)..plus_\Gamma(S_n)]$.

Example 5.3.6 $plus_{X \leq \top^\star:\star, Y \leq X:\star, Z \leq Y:\star}(Z) = Z + Y + X + \top^\star$.

We need to show that *plus* is well defined on well kinded arguments.

Lemma 5.3.7 (*Well-foundedness of plus*) If $\Gamma \vdash T : K$, then $plus_\Gamma(T)$ is defined.

Proof: Observe that the $size_j$ of the kinding judgements of the arguments strictly decreases in each recursive call. Consider

$$rank_\Gamma(S) = size_j(\Gamma \vdash S : kind(\Gamma, S)),$$

where $size_j(\Gamma \vdash S : K)$ is the size of the derivation of the kinding judgement (see definition 4.1). The function *kind* can be defined straightforwardly using proposition 3.4, such that $kind(\Gamma, S) = K$ if $\Gamma \vdash S : K$, and gives a constant *NoKind* otherwise. Moreover, lemma 4.2 implies that the function *kind* is total. Given that $\Gamma \vdash S : K$, by lemmas 3.3(1) and 3.4, the rank decreases in each recursive call and the least value is that of $size_j(\vdash \top^K : K)$. □

Lemma 5.3.8 If $\Gamma \vdash T : K$, then $\Gamma \vdash_+ plus_\Gamma(T) : K$.

Proof: By induction on the derivation of $\Gamma \vdash T : K$, observing that $\Gamma \vdash T : K$ implies $\Gamma \vdash_+ T : K$. It is straightforward to verify that $\vdash_+$ satisfies weakening using renamings (see corollary 3.7). We consider here the case for K-TVar, the rest follows by straightforward induction. We are given, $\Gamma_1, X \leq T{:}K, \Gamma_2 \vdash$ ok. By lemma 3.3, there is a proper subderivation of $\Gamma_1 \vdash T : K$. Finally, the result follows by the induction hypothesis, weakening, and K-Plus. □

Lemma 5.3.9 (*Strengthening for plus*)

1. Let $X \notin \mathrm{FTV}(\Gamma_2) \cup \mathrm{FTV}(S)$. Then
   $\Gamma_1, X \leq T_X{:}K_X, \Gamma_2 \vdash S : K$ implies $plus_{\Gamma_1, X \leq T_X:K_X, \Gamma_2}(S) = plus_{\Gamma_1, \Gamma_2}(S)$.
2. $\Gamma_1, x{:}T, \Gamma_2 \vdash S : K$ implies $plus_{\Gamma_1, x:T, \Gamma_2}(S) = plus_{\Gamma_1, \Gamma_2}(S)$.

Proof:

1. By lemma 5.3.7, $plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Gamma_2}(S)$ is defined, therefore we can reason by induction on the number of unfolding steps of *plus*. We proceed by case analysis on the form of S.

   $S \equiv Y$. We have to consider two cases.

   (a) $\Gamma_1 \equiv \Delta_1,\, Y{\leq}T_1{:}K_1,\, \Delta_2$. Then, by definition,
   $$plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Gamma_2}(Y) = Y + plus_{\Delta_1}(T_1).$$
   On the other hand, also by the definition of *plus*,
   $$plus_{\Gamma_1,\, \Gamma_2}(Y) = Y + plus_{\Delta_1}(T_1).$$

   (b) $\Gamma_2 \equiv \Delta_1,\, Y{\leq}T_1{:}K_1,\, \Delta_2$. By the definition of *plus*,
   $$plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Gamma_2}(Y) = Y + plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Delta_1}(T_1).$$
   By lemma 3.1,
   $$\Gamma_1,\, X{\leq}T_X{:}K_X,\, \Gamma_2 \vdash \mathrm{ok},$$
   and, by lemma 3.3(1),
   $$\Gamma_1,\, X{\leq}T_X{:}K_X,\, \Delta_1 \vdash T_1 : K_1.$$
   Moreover, since $X \notin \mathrm{FTV}(\Gamma_2)$, it follows that $X \notin \mathrm{FTV}(\Delta_1) \cup \mathrm{FTV}(T_1)$. Then, applying the induction hypothesis we obtain
   $$Y + plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Delta_1}(T_1) = Y + plus_{\Gamma_1,\, \Delta_1}(T_1),$$
   and the result follows by the definition of *plus*.

   $S \equiv \forall Y{\leq}T_1{:}K_1.T_2$. By the definition of *plus*,
   $$\begin{aligned}&plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Gamma_2}(\forall Y{\leq}T_1{:}K_1.T_2)\\ &= \forall Y{\leq}plus_{\Gamma_1,\, \leq T_X:K_X,\, \Gamma_2}(T_1){:}K_1.plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Gamma_2,\, Y\leq T_1:K_1}(T_2).\end{aligned}$$
   By generation for kinding (proposition 3.4),
   $$\Gamma_1,\, X{\leq}T_X{:}K_X,\, \Gamma_2,\, Y{\leq}T_1{:}K_1 \vdash T_2 : \star,$$
   and, since $X \notin \mathrm{FTV}(\Gamma_2,\, Y{\leq}T_1{:}K_1) \cup \mathrm{FTV}(T_2)$, by the induction hypothesis,
   $$\begin{aligned}&\forall Y{\leq}plus_{\Gamma_1,\, \leq T_X:K_X,\, \Gamma_2}(T_1){:}K_1.plus_{\Gamma_1,\, X\leq T_X:K_X,\, \Gamma_2,\, Y\leq T_1:K_1}(T_2)\\ &= \forall Y{\leq}plus_{\Gamma_1,\, \leq T_X:K_X,\, \Gamma_2}(T_1){:}K_1.plus_{\Gamma_1,\, \Gamma_2,\, Y\leq T_1:K_1}(T_2).\end{aligned}$$
   By lemma 3.1,
   $$\Gamma_1,\, X{\leq}T_X{:}K_X,\, \Gamma_2,\, Y{\leq}T_1{:}K_1 \vdash \mathrm{ok},$$
   by generation for context judgements (lemma 3.3(1)),
   $$\Gamma_1,\, X{\leq}T_X{:}K_X,\, \Gamma_2 \vdash T_1 : K_1.$$
   Since $X \notin \mathrm{FTV}(\Gamma_2) \cup \mathrm{FTV}(T_1)$, by the induction hypothesis,
   $$\begin{aligned}&\forall Y{\leq}plus_{\Gamma_1,\, \leq T_X:K_X,\, \Gamma_2}(T_1){:}K_1.plus_{\Gamma_1,\, \Gamma_2,\, Y\leq T_1:K_1}(T_2)\\ &= \forall Y{\leq}plus_{\Gamma_1,\, \Gamma_2}(T_1){:}K_1.plus_{\Gamma_1,\, \Gamma_2,\, Y\leq T_1:K_1}(T_2)\\ &= plus_{\Gamma_1,\, \Gamma_2}(\forall Y{\leq}T_1{:}K_1.T_2).\end{aligned}$$

For all the other cases, the result follows by straightforward application of the induction hypothesis, using generation for kinding (proposition 3.4).

2. The definition of *plus* does not depend on the assumptions of term variables. □

LEMMA 5.3.10 (*Weakening for plus*) If $\Gamma' \vdash \text{ok}$, $\Gamma \subseteq \Gamma'$, and $\Gamma \vdash S : K$, then $plus_\Gamma(S) = plus_{\Gamma'}(S)$.

PROOF: The assumptions ensure that $plus_\Gamma(S)$ is defined, so we can proceed by induction on the number of unfolding steps of the definition of *plus*. We proceed by case analysis on the form of S.

$S \equiv X$. By generation for kinding (proposition 3.4) and the fact that $\Gamma \subseteq \Gamma'$,

$$\Gamma \equiv \Gamma_1,\, X{\leq}T{:}K,\, \Gamma_2 \quad \text{and} \quad \Gamma' \equiv \Gamma'_1,\, X{\leq}T{:}K,\, \Gamma'_2.$$

There are two cases to consider.

1. If $\Gamma_1 \equiv \Gamma'_1$, then the result follows by the definition of *plus*.
2. If $\Gamma_1 \not\equiv \Gamma'_1$, then $\Gamma_1 \subseteq \Gamma'_1 \cup \Gamma'_2$.
   By the definition of *plus*,
   $$plus_\Gamma(X) = X + plus_{\Gamma_1}(T).$$
   By lemmas 3.1 and 3.3(1), it follows that $\Gamma_1 \vdash T : K$. Hence, by the induction hypothesis,
   $$X + plus_{\Gamma_1}(T) = X + plus_{\Gamma'}(T).$$
   Since $\Gamma' \vdash \text{ok}$, from lemma 3.3(1), it follows that $\Gamma'_1 \vdash T : K$. Consequently, $(\{X\} \cup \text{FTV}(\Gamma'_2)) \cap \text{FTV}(T) = \emptyset$ by the free variables lemma (lemma 3.2). Hence, starting from the last declaration in Γ'_2, we can iterate the strengthening lemma for *plus* (lemma 5.3.9 items 1 and 2) to obtain
   $$X + plus_{\Gamma'}(T) = X + plus_{\Gamma'_1}(T) = plus_{\Gamma'}(X).$$

$S \equiv \forall X{\leq}T_1{:}K_1.T_2$. We have that $\Gamma \vdash \forall X{\leq}T_1{:}K_1.T_2 : \star$, by generation for kinding (proposition 3.4).

Let Z be a new type variable, in particular $Z \notin \text{dom}(\Gamma')$. Then, by α-conversion, $S =_\alpha \forall Z{\leq}T_1{:}K_1.T_2[X{\leftarrow}Z]$.

By the definition of *plus*,

$$\begin{aligned} plus_\Gamma(S) &=_\alpha plus_\Gamma(\forall Z{\leq}T_1{:}K_1.T_2[X{\leftarrow}Z]) \\ &= \forall Z{\leq}plus_\Gamma(T_1){:}K_1.plus_{\Gamma,\, Z\leq T_1:K_1}(T_2[X{\leftarrow}Z]). \end{aligned}$$

By generation for kinding and lemmas 3.1 and 3.3(1), it follows that $\Gamma \vdash T_1 : K_1$. Then, by the induction hypothesis,

$$\forall Z{\leq}plus_{\Gamma}(T_1){:}K_1.plus_{\Gamma,\,Z\leq T_1:K_1}(T_2[X{\leftarrow}Z])$$
$$= \forall Z{\leq}plus_{\Gamma'}(T_1){:}K_1.plus_{\Gamma,\,Z\leq T_1:K_1}(T_2[X{\leftarrow}Z]).$$

By generation for kinding, $\Gamma,\, X{\leq}T_1{:}K_1 \vdash T_2 : \star$, and by renaming (lemma 3.6), we have that $\Gamma,\, Z{\leq}T_1{:}K_1 \vdash T_2[X{\leftarrow}Z] : \star$. (Taking the renaming that maps X to Z and is the identity elsewhere.)

Applying again the induction hypothesis, it follows that

$$\begin{aligned}
&\forall Z{\leq}plus_{\Gamma'}(T_1){:}K_1.plus_{\Gamma,\,Z\leq T_1:K_1}(T_2[X{\leftarrow}Z])\\
&= \forall Z{\leq}plus_{\Gamma'}(T_1){:}K_1.plus_{\Gamma',\,Z\leq T_1:K_1}(T_2[X{\leftarrow}Z])\\
&= plus_{\Gamma'}(\forall Z{\leq}T_1{:}K_1.T_2[X{\leftarrow}Z])\\
&=_{\alpha} plus_{\Gamma'}(S).
\end{aligned}$$

The case for $S \equiv \Lambda X{:}K.T$ is similar to the last case. In all other cases, the proof follows by straightforward application of the induction hypothesis. □

The operation *plus* does not have the usual properties under substitution; as following example shows, the equality

$$plus_{\Gamma_1,\,X\leq S:K_1,\,\Gamma_2}(T_2)[X{\leftarrow}plus_{\Gamma_1}(T_1)] = plus_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(T_2[X{\leftarrow}T_1])$$

does not hold in general.

Example 5.3.11 Consider the case where
$\Gamma_1 \equiv Y{\leq}\top^{\star}{:}\star$, $\Gamma_2 \equiv \emptyset$, $S \equiv Y$, $T_1 \equiv Y$, and $T_2 \equiv X$.
Then

$$\begin{aligned}
plus_{Y\leq\top^{\star}:\star,\,X\leq Y:\star}(X)[X{\leftarrow}plus_{Y\leq\top^{\star}:\star}(Y)] &= (X + Y + \top^{\star})[X{\leftarrow}(Y + \top^{\star})]\\
&= Y + \top^{\star} + Y + \top^{\star}.
\end{aligned}$$

On the other hand,

$$plus_{Y\leq\top^{\star}:\star}(X[X{\leftarrow}Y]) \;=\; plus_{Y\leq\top^{\star}:\star}(Y) \;=\; Y + \top^{\star}.$$

We therefore need a lemma which says that the well-formed types are well-behaved under substitution with respect to the *plus* operation.

Lemma 5.3.12 (*Substitution for plus*) If $\Gamma_1,\, X{\leq}S{:}K_1,\, \Gamma_2 \vdash T_2 : K_2$ and $\Gamma_1 \vdash T_1 : K_1$, then

$$plus_{\Gamma_1,\,X\leq S:K_1,\,\Gamma_2}(T_2)[X{\leftarrow}plus_{\Gamma_1}(T_1)] \twoheadrightarrow_{\beta\wedge+} plus_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(T_2[X{\leftarrow}T_1]).$$

Proof: By induction on the size of the derivation of $\Gamma_1,\, X{\leq}S{:}K_1,\, \Gamma_2 \vdash T_2 : K_2$. We proceed by case analysis on the form of T_2.

$T_2 \equiv Y$. By the free variables lemma (lemma 3.2), $Y \in \mathrm{dom}(\Gamma_1,\, X{\leq}S{:}K_1,\, \Gamma_2)$. Then there are three cases to consider.

$Y \in \mathrm{dom}(\Gamma_1)$. Let $\Gamma_1 \equiv \Delta_1,\ Y{\leq}U{:}K,\ \Delta_2$. Then

$$\begin{aligned}
&\mathit{plus}_{\Gamma_1,\,X\leq S:K_1,\,\Gamma_2}(Y)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)],\\
&\qquad \text{by the definitions of } \mathit{plus} \text{ and substitution,}\\
&= Y + (\mathit{plus}_{\Delta_1}(U)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)])\\
&\qquad \text{since } X \notin \mathrm{FTV}(U) \cup \mathrm{FTV}(\Delta_1), X \notin \mathrm{FTV}(\mathit{plus}_{\Delta_1}(U)).\\
&= Y + \mathit{plus}_{\Delta_1}(U),\\
&= \mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(Y[X{\leftarrow}T_1]).
\end{aligned}$$

$Y \equiv X$. Then

$$\begin{aligned}
&\mathit{plus}_{\Gamma_1,\,X\leq S:K_1,\,\Gamma_2}(X)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)],\\
&\qquad \text{by the definitions of } \mathit{plus} \text{ and substitution,}\\
&= \mathit{plus}_{\Gamma_1}(T_1) + (\mathit{plus}_{\Delta_1}(U)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)]),\\
&\rightarrow_{+} \mathit{plus}_{\Gamma_1}(T_1).
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(X[X{\leftarrow}T_1])\\
&= \mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(T_1),\\
&\qquad \text{since } \mathrm{FTV}(T_1) \cup \mathrm{dom}(\Gamma_1), \text{ by strengthening for } \mathit{plus}(5.3.9),\\
&= \mathit{plus}_{\Gamma_1}(T_1).
\end{aligned}$$

$Y \in \mathrm{dom}(\Gamma_2)$. Let $\Gamma_2 \equiv \Delta_1,\ Y{\leq}U{:}K,\ \Delta_2$. Then

$$\begin{aligned}
&\mathit{plus}_{\Gamma_1,\,X\leq S:K_1,\,\Gamma_2}(Y)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)],\\
&\qquad \text{by the definitions of } \mathit{plus} \text{ and substitution,}\\
&= Y + (\mathit{plus}_{\Gamma_1,\,X\leq S:K_1,\,\Delta_1}(U)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)]),\\
&\qquad \text{by generation (3.4) and the induction hypothesis,}\\
&\rightarrow_{\beta\wedge+} Y + \mathit{plus}_{\Gamma_1,\,\Delta_1[X\leftarrow T_1]}(U[X{\leftarrow}T_1]),\\
&= \mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(Y[X{\leftarrow}T_1]).
\end{aligned}$$

$T_2 \equiv \forall Y{\leq}S_1{:}K.S_2$. Let $\Gamma \equiv \Gamma_1,\ X{\leq}S{:}K_1,\ \Gamma_2$. Then

$$\begin{aligned}
&\mathit{plus}_{\Gamma}(\forall Y{\leq}S_1{:}K.S_2)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)],\\
&\qquad \text{by the definitions of } \mathit{plus} \text{ and substitution,}\\
&= \forall Y{\leq}\mathit{plus}_{\Gamma}(S_1)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)]{:}K.\mathit{plus}_{\Gamma,\,Y\leq S_1:K}(S_2)[X{\leftarrow}\mathit{plus}_{\Gamma_1}(T_1)],\\
&\qquad \text{by generation (proposition 3.4) and the induction hypothesis,}\\
&\rightarrow_{\beta\wedge+} \forall Y{\leq}\mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}(S_1[X{\leftarrow}T_1]){:}K.\\
&\qquad\qquad\qquad \mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1],\,Y\leq S_1[X\leftarrow T_1]:K}(S_2[X{\leftarrow}T_1])\\
&\qquad \text{by the definitions of } \mathit{plus} \text{ and substitution,}\\
&= \mathit{plus}_{\Gamma_1,\,\Gamma_2[X\leftarrow T_1]}((\forall Y{\leq}S_1{:}K.S_2)[X{\leftarrow}T_1]).
\end{aligned}$$

Other cases. All the other cases are similar to the case $T_2 \equiv \forall Y{\leq}S_1{:}K.S_2$. □

LEMMA 5.3.13 (*Monotonicity of plus with respect to* $\rightarrow_{\beta\wedge}$) If $\Gamma \vdash T : K$, then

1. $\Gamma \rightarrow_{\beta\wedge} \Gamma'$ implies $plus_{\Gamma}(T) \twoheadrightarrow_{\beta\wedge+} plus_{\Gamma'}(T)$.
2. $T \rightarrow_{\beta\wedge} T'$ implies $plus_{\Gamma}(T) \twoheadrightarrow_{\beta\wedge+}{}^{>0} plus_{\Gamma}(T')$.

PROOF: By simultaneous induction on the size of the derivation of $\Gamma \vdash T : K$. We proceed by case analysis on the form of T.

1. $\Gamma \rightarrow_{\beta\wedge} \Gamma'$.

   $T \equiv X$. Let $\Gamma \equiv \Gamma_1,\ X{\leq}S{:}K_1,\ \Gamma_2$. Then we have to consider three cases.

   (a) $\Gamma_1 \rightarrow_{\beta\wedge} \Gamma_1'$. Then
   $plus_{\Gamma}(X) = X + plus_{\Gamma_1}(S)$
   by lemma 3.3 and part (1) of the induction hypothesis,
   $\twoheadrightarrow_{\beta\wedge} X + plus_{\Gamma_1'}(S) = plus_{\Gamma_1'}(X)$.

   (b) $S \rightarrow_{\beta\wedge} S'$. By lemma 3.3 and part (2) of the induction hypothesis.

   (c) $\Gamma_2 \rightarrow_{\beta\wedge} \Gamma_2'$. By the definition of *plus*.

   $T \equiv \forall X{\leq}T_1{:}K_1.T_2$. By generation for kinds (proposition 3.4), there are proper subderivations of $\Gamma \vdash T_1 : K_1$ and $\Gamma,\ X{\leq}T_1{:}K_1 \vdash T_2 : \star$. Then, by part (1) of the induction hypothesis, it follows that

   $$\begin{array}{rcl} plus_{\Gamma}(T_1) & \twoheadrightarrow_{\beta\wedge} & plus_{\Gamma'}(T_1), \text{ and} \\ plus_{\Gamma,\, X\leq T_1:K_1}(T_2) & \twoheadrightarrow_{\beta\wedge} & plus_{\Gamma',\, X\leq T_1:K_1}(T_2). \end{array}$$

   The result follows by the definitions of *plus* and $\twoheadrightarrow_{\beta\wedge}$.

   Other cases. The rest of the cases are similar to the case $T \equiv \forall X{\leq}T_1{:}K_1.T_2$, using generation for kinding (proposition 3.4) and part 1 of the induction hypothesis.

2. $T \rightarrow_{\beta\wedge} T'$.

   $T \equiv \forall X{\leq}T_1{:}K_1.T_2$. We have to consider three cases.

   (a) $T_1 \rightarrow_{\beta\wedge} T_1'$. By generation for kinding (proposition 3.4), there are proper subderivations of $\Gamma \vdash T_1 : K_1$ and $\Gamma,\ X{\leq}T_1{:}K_1 \vdash T_2 : \star$. Then, by parts (2) and (1) of the induction hypothesis respectively, it follows that

   $$\begin{array}{rcl} plus_{\Gamma}(T_1) & \twoheadrightarrow_{\beta\wedge}{}^{>0} & plus_{\Gamma}(T_1'), \text{ and} \\ plus_{\Gamma,\, X\leq T_1:K_1}(T_2) & \twoheadrightarrow_{\beta\wedge} & plus_{\Gamma,\, X\leq T_1{}':K_1}(T_2). \end{array}$$

   The result follows by the definitions of *plus* and $\twoheadrightarrow_{\beta\wedge}$.

   (b) $T_2 \rightarrow_{\beta\wedge} T_2'$. By part (2) of the induction hypothesis.

   (c) $\forall X{\leq}T_1{:}K_1.\bigwedge^{\star}[S_1..S_n] \rightarrow_{\beta\wedge} \bigwedge^{\star}[\forall X{\leq}T_1{:}K_1.S_1..\forall X{\leq}T_1{:}K_1.S_n]$.

$$
\begin{aligned}
& plus_\Gamma(\forall X{\leq}T_1{:}K_1.\bigwedge^\star[S_1..S_n]) \\
& = \forall X{\leq}plus_\Gamma(T_1){:}K.\bigwedge^\star[plus_{\Gamma,\,X\leq T_1:K_1}(S_1)..plus_{\Gamma,\,X\leq T_1:K_1}(S_n)] \\
& \rightarrow_{\beta\wedge+} \bigwedge^\star[\forall X{\leq}plus_\Gamma(T_1){:}K.plus_{\Gamma,\,X\leq T_1:K_1}(S_1).. \\
& \qquad\qquad ..\forall X{\leq}plus_\Gamma(T_1){:}K.plus_{\Gamma,\,X\leq T_1:K_1}(S_n)] \\
& = plus_\Gamma(\bigwedge^\star[\forall X{\leq}T_1{:}K_1.S_1..\forall X{\leq}T_1{:}K_1.S_n])
\end{aligned}
$$

$T \equiv T_1\,T_2$. We have to consider four cases.

(a) $T_1 \rightarrow_{\beta\wedge} T_1'$,

(b) $T_2 \rightarrow_{\beta\wedge} T_2'$,

(c) $T \equiv \bigwedge^\star[S_1..S_n]$ and $\bigwedge^\star[S_1..S_n]T_2 \rightarrow_{\beta\wedge} \bigwedge^\star[S_1\,T_2..S_n\,T_2]$.

(d) $T \equiv \Lambda X{:}K.S_1$ and $(\Lambda X{:}K.S_1)\,T_2 \rightarrow_{\beta\wedge} S_1[X{\leftarrow}T_2]$

Cases 2a, 2b, and 2c follow using similar arguments to those used for the case $T \equiv \forall X{\leq}T_1{:}K_1.T_2$. Consider case 2d.

$$
\begin{aligned}
& plus_\Gamma((\Lambda X{:}K.S_1)\,T_2) \\
& = (\Lambda X{:}K.plus_{\Gamma,\,X\leq\top^K:K}(S_1))plus_\Gamma(T_2) \\
& \rightarrow_{\beta\wedge} plus_{\Gamma,\,X\leq\top^K:K}(S_1)[X{\leftarrow}plus_\Gamma(T_2)], \\
& \qquad \text{by lemma 5.3.12,} \\
& \twoheadrightarrow_{\beta\wedge+} plus_\Gamma(S_1[X{\leftarrow}T_2]).
\end{aligned}
$$

Other cases. The rest of the cases follows using a similar argument to the one used in the case $T \equiv \forall X{\leq}T_1{:}K_1.T_2$. □

Lemma 5.3.14 Let $lub_\Gamma(S)$ be defined and $\Gamma \vdash S : K$. Then $plus_\Gamma(S) \twoheadrightarrow_+{}^{>0} plus_\Gamma(lub_\Gamma(S))$.

Proof: By induction on the structure of S. Since $lub_\Gamma(S)$ is defined, it is enough to consider the following two cases.

$S \equiv X$. Let $\Gamma \equiv \Gamma_1,\, X \leq T{:}K,\, \Gamma_2$.

$$
\begin{aligned}
plus_\Gamma(X) &= X + plus_{\Gamma_1}(T) \\
&= X + plus_\Gamma(T) \qquad \text{by weakening (lemma 5.3.10),} \\
&\rightarrow_+ plus_\Gamma(T) \\
&= plus_\Gamma(lub_\Gamma(X))
\end{aligned}
$$

$S \equiv A\,T$. By the induction hypothesis. □

Our measure to show the well-foundedness of $AlgF^\omega_\wedge$ considers the $\beta\wedge+$-reduction paths of the *plus* versions of the types in the subtyping judgements. As we mentioned before, in AlgS-TVar and AlgS-OApp the types appearing in the hypothesis may be larger than those in their conclusions. Therefore, the well foundedness of the $AlgF^\omega_\wedge$ relation is not immediate. The next corollary gathers the previous results to serve our purposes.

Corollary 5.3.15

1. If $\Gamma \vdash X : K$, then $plus_{\Gamma}(X) \twoheadrightarrow_{\beta\wedge+}{}^{>0} plus_{\Gamma}(\Gamma(X))$.
2. If $\Gamma \vdash AT : K$ then $plus_{\Gamma}(AT) \twoheadrightarrow_{\beta\wedge+}{}^{>0} plus_{\Gamma}(lub_{\Gamma}(AT)^{\mathrm{nf}})$.

Proof: Item 1 is a particular case of the previous lemma (lemma 5.3.14), and item 2 is a consequence of lemma 5.3.14 and the monotonicity of *plus* with respect to $\rightarrow_{\beta\wedge+}$ (5.3.13(2)). □

Finally, we can define our measure.

Definition 5.3.16 (*Weight*)

1. $weight(\Gamma \vdash_{Alg} S \leq T) = <\text{max-red}(plus_{\Gamma}(S)) + \text{max-red}(plus_{\Gamma}(T)), size_j(\Gamma \vdash S \leq T)>$,
2. $weight(\Gamma \vdash T : K) = <0, 0>$,

where max-red(S) is the length of a maximal $\beta\wedge+$-reduction path starting from S, and $size_j$ is defined in definition 4.1.

Pairs are ordered lexicographically. Note that $<0, 0>$ is the least *weight*.

Proposition 5.3.17 (*Well-foundedness of $AlgF^{\omega}_{\wedge}$*)

If $\dfrac{J_1 \,..\, J_n}{J}$ is an $AlgF^{\omega}_{\wedge}$ rule, then $weight(J_i) < weight(J)$, for each $i \in \{1..n\}$.

Proof: By inspection of the rules of $AlgF^{\omega}_{\wedge}$. □

Finally, we can state the main result of this section.

Theorem 5.3.18 (*Decidability of subtyping in $F^{\omega}_{\wedge}$*)

For any context Γ and for any two types S and T, it is decidable whether $\Gamma \vdash S \leq T$.

6 Our decidability proof and full $F_{\leq}$

In the introduction we mentioned that subtyping in $F_{\leq}$, a second-order λ-calculus with bounded quantification defined by Curien and Ghelli in 1989, is undecidable. A question that comes to mind is: if we try to apply our proof of the decidability of subtyping in $F^{\omega}_{\wedge}$ to $F_{\leq}$, where will it fail?

If we consider the algorithm for the subtyping relation in [21], the place where our proof does not go through is when we try to prove that the algorithm terminates by calculating the maximal length of the *plus* versions of the types in the rule for subtyping quantified types. Remember that the subtyping rule for quantified types in full $F_{\leq}$ is:

$$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma,\, X{\leq}T_1 \vdash S_2 \leq T_2}{\Gamma \vdash \forall X{\leq}S_1.\ S_2 \leq \forall X{\leq}T_1.\ T_2} \qquad (F_{\leq}\text{- S-All})$$

Consider now the following case.

$$\begin{array}{lcl}
\Gamma & \equiv & Y_4{\leq}\top^\star,\ Y_3{\leq}Y_4,\ Y_2{\leq}Y_3,\ Y_1{\leq}Y_2, \\
T_1 & \equiv & Y_1, \\
S_1 & \equiv & \top^\star, \\
T_2 & \equiv & X{\rightarrow}X, \quad \text{and} \\
S_2 & \equiv & X{\rightarrow}X.
\end{array}$$

The *plus* versions of the types in the subtyping statements of this example are as follows.

$$\begin{array}{l}
\mathit{plus}_{\Gamma,\,X\leq Y_1}(S_2) \equiv (X+Y_1+Y_2+Y_3+Y_4+\top^\star){\rightarrow}(X+Y_1+Y_2+Y_3+Y_4+\top^\star) \\
\mathit{plus}_{\Gamma,\,X\leq Y_1}(T_2) \equiv (X+Y_1+Y_2+Y_3+Y_4+\top^\star){\rightarrow}(X+Y_1+Y_2+Y_3+Y_4+\top^\star) \\
\mathit{plus}_{\Gamma}(\forall X{\leq}S_1.\ S_2) \equiv \forall X{\leq}\top^\star.\ (X+\top^\star){\rightarrow}(X+\top^\star) \\
\mathit{plus}_{\Gamma}(\forall X{\leq}T_1.\ T_2) \equiv \forall X{\leq}Y_1+Y_2+Y_3+Y_4+\top^\star. \\
\qquad\qquad (X+Y_1+Y_2+Y_3+Y_4+\top^\star){\rightarrow}(X+Y_1+Y_2+Y_3+Y_4+\top^\star)
\end{array}$$

The length of a maximal +-reduction in each case is:

$$\begin{array}{lcl}
\text{max-red}(\mathit{plus}_{\Gamma,\,X\leq Y_1}(S_2)) & = & 10 \\
\text{max-red}(\mathit{plus}_{\Gamma,\,X\leq Y_1}(T_2)) & = & 10 \\
\text{max-red}(\mathit{plus}_{\Gamma}(\forall X{\leq}S_1.\ S_2)) & = & 2 \\
\text{max-red}(\mathit{plus}_{\Gamma}(\forall X{\leq}T_1.\ T_2)) & = & 14.
\end{array}$$

The *weight* of the conclusion $\Gamma \vdash \forall X{\leq}S_1.\ S_2 \leq \forall X{\leq}T_1.\ T_2$, as defined in definition 5.3.16, is smaller than the *weight* of the hypothesis $\Gamma,\ X{\leq}T_1 \vdash S_2 \leq T_2$, because the maximal length of a +-reduction starting from the *plus* version of the conclusion is shorter than the maximal length of a +-reduction starting from the *plus* version of that hypothesis. To be more precise,

$$\begin{array}{c}
\text{max-red}(\mathit{plus}_{\Gamma}(\forall X{\leq}S_1.\ S_2)) + \text{max-red}(\mathit{plus}_{\Gamma}(\forall X{\leq}T_1.\ T_2)) \\
< \\
\text{max-red}(\mathit{plus}_{\Gamma,\,X\leq Y_1}(S_2)) + \text{max-red}(\mathit{plus}_{\Gamma,\,X\leq Y_1}(T_2)).
\end{array}$$

7 Type checking and type inference

Motivating examples of the following definitions can be found in [15, 12].

The function *lub* (definition 5.1.2) is a partial function which is only defined for type variables and type applications. Here, we extend the definition of *lub* to intersection types in such a way that it is defined if the least upper bound is defined for at least one of the types in the intersection.

Definition 7.1 (*Homomorphic extension of lub to intersections, lub**)

$$\begin{array}{rcl} \mathit{lub}^*_\Gamma(X) & = & \Gamma(X), \\ \mathit{lub}^*_\Gamma(S\,T) & = & \mathit{lub}^*_\Gamma(S)\,T, \\ \mathit{lub}^*_\Gamma(\bigwedge^K[T_1..T_n]) & = & \bigwedge^K[T'_1..T'_n], \qquad \text{if } \exists i \in \{1..n\} \text{ such that } \mathit{lub}^*_\Gamma(T_i)\downarrow, \end{array}$$

where T'_i is $\mathit{lub}^*_\Gamma(T_i)$, if $\mathit{lub}^*_\Gamma(T_i)\downarrow$, and T_i otherwise.

We define the mapping *flub* which given a type T (and a context Γ) finds the smallest type larger than T (with respect to the subtype relation) having structural information to perform an application.

Definition 7.2 (*Functional Least Upper Bound*) The functional least upper bound of a type T, in a context Γ, $\mathit{flub}_\Gamma(T)$ is defined as follows.

$$\mathit{flub}_\Gamma(T) = \begin{cases} \mathit{flub}_\Gamma(\mathit{lub}^*_\Gamma(T^{\mathrm{nf}})), & \text{if } \mathit{lub}^*_\Gamma(T^{\mathrm{nf}})\downarrow; \\ T^{\mathrm{nf}}, & \text{otherwise.}^1 \end{cases}$$

The intuition behind the definition of the function *flub* is to find $S{\to}T$ starting form $W\,T$ as in the last example mentioned in the introduction. In other words, $\mathit{flub}_{\Gamma'}(W\,T) = S{\to}T$. For simplicity we assume $S{\to}T$ in normal form. Step by step,

$$\begin{array}{rcl} \mathit{flub}_{\Gamma'}(W\,T) & = & \mathit{flub}_{\Gamma'}(\mathit{lub}^*_{\Gamma'}(W\,T)) \\ & = & \mathit{flub}_{\Gamma'}(Z\,T) \\ & = & \mathit{flub}_{\Gamma'}(\mathit{lub}^*_{\Gamma'}(Z\,T)) \\ & = & \mathit{flub}_{\Gamma'}((\Lambda Y{:}\star.S{\to}Y)\,T) \\ & = & S{\to}T. \end{array}$$

More generally, *flub* climbs the subtyping hierarchy until it finds an arrow, a quantifier, or an intersection of these two.

Definition 7.3 (*arrows and alls*)

1. $$\begin{array}{lcl} \mathit{arrows}\,(T_1{\to}T_2) & = & \{T_1{\to}T_2\}, \\ \mathit{arrows}\,(\bigwedge^\star[T_1..T_n]) & = & \cup_{i\in\{1..n\}}\,\mathit{arrows}\,(T_i), \\ \mathit{arrows}\,(T) & = & \emptyset, \quad \text{if } T \not\equiv T_1{\to}T_2 \text{ and } T \not\equiv \bigwedge^\star[T_1..T_n]. \end{array}$$

2. $$\begin{array}{lcl} \mathit{alls}\,(\forall X{\leq}T_1{:}K.T_2) & = & \{\forall X{\leq}T_1{:}K.T_2\}, \\ \mathit{alls}\,(\bigwedge^\star[T_1..T_n]) & = & \cup_{i\in\{1..n\}}\,\mathit{alls}\,(T_i), \\ \mathit{alls}\,(T) & = & \emptyset, \quad \text{if } T \not\equiv \forall X{\leq}T_1{:}K.T_2 \text{ and } T \not\equiv \bigwedge^\star[T_1..T_n]. \end{array}$$

[1]This step can be optimised in an implementation of the type checking algorithm, allowing us to avoid the normalization of T when T is either an arrow type or a quantified type.

The situation here is significantly more complex than in [26] for $F_\wedge$, an extension of the second order λ-calculus. There it is enough to recursively search for arrows or polymorphic types in the context, because in $F_\wedge$ there is no reduction on types. The information to be searched for is explicit in the context, so the job done here by *flub* is simply an extra case in the definition of *arrows* and *alls*. Namely,

$$\begin{array}{rcl} \mathit{arrows}(X) & = & \mathit{arrows}(\Gamma(X)) \qquad \text{and} \\ \mathit{alls}(X) & = & \mathit{alls}(\Gamma(X)). \end{array}$$

Moreover, to prove that *flub* is well-founded is similar for us in complexity to proving termination of subtype checking. The similarity comes from the fact that computing *flub* involves replacing variables by their bounds in a given context and normalizing with respect to $\rightarrow_{\beta\wedge}$, as in lemma 7.8. In contrast, in [26] it is enough to observe that well-formed contexts cannot contain cycles of variable references.

NOTATION 7.4 We introduce a new notation for intersection types. We write $\bigwedge^K[T \mid \phi(T)]$, meaning the intersection of all types T such that $\phi(T)$ holds. Note that this is an alternative notation to $\bigwedge^K[T_1..T_n]$ such that $\phi(T_i)$ holds if and only if $i \in \{1..n\}$.

We can now define a type inference algorithm for $F^\omega_\wedge$.

DEFINITION 7.5 (*A type inference algorithm,* inf)

$$\frac{\Gamma_1,\ x{:}T,\ \Gamma_2 \vdash \text{ok}}{\Gamma_1,\ x{:}T,\ \Gamma_2 \vdash_{\text{inf}} x : T} \qquad \text{(AT-VAR)}$$

$$\frac{\Gamma,\ x{:}T_1 \vdash_{\text{inf}} e : T_2}{\Gamma \vdash_{\text{inf}} \lambda x{:}T_1.e : T_1{\rightarrow}T_2} \qquad \text{(AT-ABS)}$$

$$\frac{\Gamma \vdash_{\text{inf}} f : T \qquad \Gamma \vdash_{\text{inf}} a : S}{\Gamma \vdash_{\text{inf}} f\ a : \bigwedge^\star[T_i \mid S_i{\rightarrow}T_i \in \mathit{arrows}(\mathit{flub}_\Gamma(T)) \text{ and } \Gamma \vdash S \leq S_i]} \qquad \text{(AT-APP)}$$

$$\frac{\Gamma,\ X{\leq}T_1{:}K_1 \vdash_{\text{inf}} e : T_2}{\Gamma \vdash_{\text{inf}} \lambda X{\leq}T_1{:}K_1.e : \forall X{\leq}T_1{:}K_1.T_2} \qquad \text{(AT-TABS)}$$

$$\frac{\Gamma \vdash_{\text{inf}} f : T}{\Gamma \vdash_{\text{inf}} f\ S : \bigwedge^\star[T_i[X{\leftarrow}S] \mid \forall X{\leq}S_i{:}K.T_i \in \mathit{alls}(\mathit{flub}_\Gamma(T)) \text{ and } \Gamma \vdash S \leq S_i]} \qquad \text{(AT-TAPP)}$$

$$\frac{\text{for all } i \in \{1..n\} \quad \Gamma \vdash_{\text{inf}} e[X{\leftarrow}S_i] \in T_i}{\Gamma \vdash_{\text{inf}} \text{for}(X{\in}S_1..S_n)e : \bigwedge^\star[T_1..T_n]} \qquad \text{(AT-FOR)}$$

The algorithmic information of rule AT-APP is that in order to find a type for $f\,a$ in Γ, we need to infer a type S for a and a type T for f, and to take the intersection of all the $T_i's$ such that $T_i{\rightarrow}S_i \in \mathit{arrows}(\mathit{flub}_\Gamma(T))$ and $\Gamma \vdash S \leq S_i$.

To show that *flub* is well-defined we use a similar argument to that used in section 5.3 to show that the relation defined by $\mathit{Alg}F^\omega_\wedge$ is well-founded. We show in lemma 7.8 that a

maximal $\beta\wedge+$-reduction path of the *plus* version of the argument of *flub* is strictly longer than a maximal $\beta\wedge+$-reduction path of the *plus* version of the argument of its recursive call. But first we need to show the following auxiliary result.

LEMMA 7.6 If $lub^*_\Gamma(T)$ is defined, then $\Gamma \vdash T \leq lub^*_\Gamma(T)$.

LEMMA 7.7 Let $lub^*_\Gamma(T)$ be defined and $\Gamma \vdash T : K$. Then $plus_\Gamma(T) \twoheadrightarrow_{\beta\wedge+}{}^{>0} plus_\Gamma(lub^*_\Gamma(T))$.

PROOF: The proof follows by induction on the structure of T. If $T \equiv X$ or $T \equiv S\,T$, then the argument is the same as in lemma 5.3.14. The case remaining to be checked is when $T \equiv \bigwedge^K[T_1..T_n]$. Then

$$\begin{array}{rcl} plus_\Gamma(\bigwedge^K[T_1..T_n]) & = & \bigwedge^K[plus_\Gamma(T_1)..plus_\Gamma(T_n)] \\ plus_\Gamma(lub^*_\Gamma(\bigwedge^K[T_1..T_n])) & = & \bigwedge^K[plus_\Gamma(T'_1)..plus_\Gamma(T'_n)], \end{array}$$

where $T'_1 \equiv T_i$ or $T'_i = lub^*_\Gamma(T_i)$. Since $lub^*_\Gamma(T)$ is defined, there exists $j \in \{1..n\}$ such that $lub^*_\Gamma(T_j)$ is defined. Now, for every k such that $lub^*_\Gamma(T_k)$ is defined, by the induction hypothesis, we have that

$$plus_\Gamma(T_k) \twoheadrightarrow_{\beta\wedge+}{}^{>0} plus_\Gamma(lub^*_\Gamma(T_k)).$$

Hence,

$$plus_\Gamma(\bigwedge^K[T_1..T_n]) \twoheadrightarrow_{\beta\wedge+}{}^{>0} plus_\Gamma(lub^*_\Gamma(\bigwedge^K[T_1..T_n])).$$ □

LEMMA 7.8 (*Well-foundedness of flub*)
If $\Gamma \vdash T : K$, then $flub_\Gamma(T)$ is defined.

PROOF: If $lub^*_\Gamma(T^{\mathrm{nf}})$ is undefined, *flub* terminates because $\rightarrow_{\beta\wedge}$ is strongly normalizing on well kinded types. Otherwise, define

$$weight(flub_\Gamma(T)) = max\text{-}red(plus_\Gamma(T)),$$

where $max\text{-}red(S)$ is the length of a maximal $\beta\wedge+$-reduction path starting from S. Lemma 5.3.8 and the strong normalization property of $\rightarrow_{\beta\wedge+}$ imply that *weight* is well defined and always positive on well kinded types. Since $lub^*_\Gamma(T^{\mathrm{nf}})$ is defined,

$$\begin{array}{lll} plus_\Gamma(T) & \twoheadrightarrow_{\beta\wedge+} & plus_\Gamma(T^{\mathrm{nf}}), \qquad \text{by lemma 5.3.13(2)}, \\ & \twoheadrightarrow_{\beta\wedge+}{}^{>0} & plus_\Gamma(lub^*_\Gamma(T^{\mathrm{nf}})), \qquad \text{by lemma 7.7}. \end{array}$$

Then the *weight* of the arguments of *flub* reduces in each recursive call, which proves that *flub* is well-founded. □

LEMMA 7.9 Let $\Gamma \vdash S, T : \star$ and $S =_{\beta\wedge} T$. Then $flub_\Gamma(S) \equiv flub_\Gamma(T)$.

8 Decidability of type checking and type inference

We know from [15, 12], that the algorithm *inf* is sound and computes minimal types for the $F^{\omega}_{\wedge}$ typing system.

PROPOSITION 8.1 (*Soundness of inf*) If $\Gamma \vdash_{\text{inf}} e : T$, then $\Gamma \vdash e : T$.

PROPOSITION 8.2 (*inf computes minimal types*)
If $\Gamma \vdash e : T$ and $\Gamma \vdash_{\text{inf}} e : T'$, then $\Gamma \vdash T' \leq T$.

The next step is to prove that the algorithm *inf* always terminates. This result completes the proof of decidability of type checking and type inference in $F^{\omega}_{\wedge}$.

We first define a measure for terms such that the type information inside the terms is considered to have constant value. The intuition behind the definition is to find a measure on terms which is invariant under type substitution (see lemma 8.4).

DEFINITION 8.3 (*size* $\|-\|$)

$$\begin{array}{lcl}
\|x\| & = & 1, \\
\|\lambda x{:}T.e\| & = & 1 + \|e\|, \\
\|e_1\, e_2\| & = & \|e_1\| + \|e_2\|, \\
\|\lambda X{\leq}T{:}K.e\| & = & 1 + \|e\|, \\
\|e\, T\| & = & 1 + \|e\|, \\
\|\text{for}(X{\in}T_1..T_n)e\| & = & 1 + \|e\|.
\end{array}$$

LEMMA 8.4 $\|e\| = \|e[X{\leftarrow}T]\|$.

PROPOSITION 8.5 (*Well-foundedness of* inf)
The inference rules for *inf* define a terminating algorithm.

PROOF: In the case of AT-VAR, the termination follows from the decidability of ok judgements (see corollary 4.3(1)). Furthermore, for each rule R of *inf*, if $\Gamma \vdash e : T$ is a hypothesis and $\Gamma \vdash e' : T'$ is the conclusion of R, then $\|e\| < \|e'\|$. Moreover, in the cases for AT-APP and AT-TAPP, $\Gamma \vdash f : T$ by the soundness of *inf* (proposition 8.1), $\Gamma \vdash T : \star$ by well-kindedness of typing (proposition 3.9). Hence $\mathit{flub}_{\Gamma}(T)$ is defined by lemma 7.8. Furthermore, *arrows* and *alls* define finite sets, and, as we proved in section 5.3, subtyping is decidable. Hence, the algorithm *inf* always terminates. □

We can now state and prove that type checking in $F^{\omega}_{\wedge}$ is decidable.

THEOREM 8.6 (*Decidability of type checking in* $F^{\omega}_{\wedge}$)
For any context Γ, and for any term e and type T closed in Γ, it is decidable whether $\Gamma \vdash e : T$.

PROOF: Infer a minimal type T' for e in Γ using *inf*, which is decidable by proposition 8.5, and check whether $\Gamma \vdash T' \leq T$, which is also decidable by theorem 5.3.18. $\square$

Every term e closed in a context Γ has type $\top^\star$. We are interested in finding types other than $\top^\star$, namely non-trivial types. Since *inf* computes minimal types and $\top^\star$ is the largest type (modulo $=_{\beta\wedge}$) , if a term has a non trivial type in a given context, then the algorithm *inf* finds it.

THEOREM 8.7 (*Decidability of type inference in* $F^\omega_\wedge$)

For any context Γ and for any term e closed in Γ, it is decidable whether there exists a type T such that $\Gamma \vdash e : T$ and $T \neq_{\beta\wedge} \top^\star$.

PROOF: Infer a minimal type T for e in Γ using *inf*, which is decidable by proposition 8.5, and reduce T to normal form which is decidable because $\rightarrow_{\beta\wedge}$ is strongly normalising. Finally, check whether $T^{\mathrm{nf}} \equiv \top^\star$. $\square$

9 Conclusions

In this paper we proved that typing in $F^\omega_\wedge$ is decidable. A major part of the problem is proving that the subtyping relation is decidable. A novel aspect of our proof is the use of a choice operator to model the behaviour of variables during subtype checking.

This paper contains the first proof of decidability of subtyping for a higher order lambda calculus. (The proof presented in [28] was developed afterwards and with knowledge of our proof.) The decidability of subtyping is reduced to proving the strong normalization of the language of types enriched with a choice reduction. In section 6 we show where our proof of decidability breaks if applied to the undecidable second order system $F_\leq$. Because the decidability of subtyping is established for well-formed types, we also show that well-formation of types and contexts are decidable judgements. Finally, the proof of termination of typechecking uses the technology developed for the decidability of subtyping.

The work presented here has been extracted from the Ph.D thesis of the author [15]. A short version of the decidability of subtyping result has been published in CSL'94 [14], and the decidability of subtyping result was first published in [13]. The techniques developed here have been applied to study the combination of dependent types and subtyping in [1].

10 Acknowledgements

I want to express my gratitude to Mariangiola Dezani-Ciancaglini for her scientific and moral support which made the present work possible. I am also grateful for discussions with Henk Barendregt and Benjamin Pierce, and for Healfdene Goguen's comments on previous versions.

This research was supported by the Dutch organization for scientific research, NWO-SION project *Typed lambda calculus*, and by a European Community TMR grant.

References

[1] D. Aspinall and A. Compagnoni. Subtyping dependent types. In *Eleventh Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA.*, July 27-30 1996. Preliminary version July 1995.

[2] H. P. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[3] K. B. Bruce and J. Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Proceedings of the Nineteenth ACM Symposium on Principles of Programming Languages*, Albequerque, NM, January 1992.

[4] P. Canning, W. Cook, W. Hill, W. Olthoff, and J. Mitchell. F-bounded quantification for object-oriented programming. In *Fourth International Conference on Functional Programming Languages and Computer Architecture*, pages 273–280, September 1989.

[5] L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988. Preliminary version in *Semantics of Data Types*, Kahn, MacQueen, and Plotkin, eds., Springer-Verlag LNCS 173, 1984.

[6] L. Cardelli. Types for data-oriented languages. In *First Conference on Extending Database Technology*, volume 303 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1988.

[7] L. Cardelli. Notes about $F^{\omega}_{<:}$. Unpublished manuscript, October 1990.

[8] L. Cardelli. Typeful programming. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*. Springer-Verlag, 1991. An earlier version appeared as DEC Systems Research Center Research Report #45, February 1989.

[9] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4), December 1985.

[10] F. Cardone and M. Coppo. Two extensions of Curry's type inference system. In P. Odifreddi, editor, *Logic and Computer Science*, number 31 in APIC Studies in Data Processing, pages 19–76. Academic Press, 1990.

[11] G. Castagna and B. Pierce. Decidable bounded quantification. In *Proceedings of Twenty-First Annual ACM Symposium on Principles of Programming Languages, Portland, OR.* ACM, January 1994.

[12] A. Compagnoni. Subject reduction and minimal types for higher order subtyping. Technical Report ECS-LFCS-97-363, University of Edinburgh, LFCS, August 1997.

[13] A. B. Compagnoni. Subtyping in $F^{\omega}_{\wedge}$ is decidable. Technical Report ECS-LFCS-94-281, LFCS, University of Edinburgh, January 1994.

[14] A. B. Compagnoni. Decidability of higher-order subtyping with intersection types. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic,*

CSL'94, Kazimierz, Poland, number 933 in Lecture Notes in Computer Science. Springer-Verlag, June 1995. Preliminary version available as University of Edinburgh technical report ECS-LFCS-94-281, under the title "Subtyping in $F^{\omega}_{\wedge}$ is decidable", January 1994.

[15] A. B. Compagnoni. *Higher-Order Subtyping with Intersection Types.* PhD thesis, University of Nijmegen, The Netherlands, January 1995. ISBN 90-9007860-6.

[16] A. B. Compagnoni and B. C. Pierce. Higher-order intersection types and multiple inheritance. *Mathematical Structures in Computer Science*, 6:469–501, 1996. Preliminary version available under the title *Multiple Inheritance via Intersection Types* as University of Edinburgh technical report ECS-LFCS-93-275 and Catholic University Nijmegen computer science technical report 93-18, Aug. 1993.

[17] W. R. Cook, W. L. Hill, and P. S. Canning. Inheritance is not subtyping. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 125–135, San Francisco, CA, January 1990. Also in [23].

[18] M. Coppo and M. Dezani-Ciancaglini. A new type-assignment for λ-terms. *Archiv. Math. Logik*, 19:139–156, 1978.

[19] P.-L. Curien and G. Ghelli. Coherence of subsumption: Minimum typing and type-checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2:55–91, 1992.

[20] N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.

[21] G. Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Università di Pisa, March 1990. Technical report TD–6/90, Dipartimento di Informatica, Università di Pisa.

[22] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[23] C. A. Gunter and J. C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design.* The MIT Press, 1994.

[24] J. McKinna and R. Pollack. Pure type systems formalized. In M. Bezem and J. F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 289–305. Springer–Verlag, LNCS 664, Mar. 1993.

[25] J. C. Mitchell. Toward a typed foundation for method specialization and inheritance. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages*, pages 109–124, January 1990. Also in [23].

[26] B. C. Pierce. *Programming with Intersection Types and Bounded Polymorphism.* PhD thesis, Carnegie Mellon University, December 1991. Available as School of Computer Science technical report CMU-CS-91-205.

[27] B. C. Pierce and D. N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, Apr. 1994. A preliminary version appeared in Principles of Programming Languages, 1993, and as University of Edinburgh technical report ECS-LFCS-92-225, under the title "Object-Oriented Programming Without Recursive Types".

[28] M. Steffen and B. Pierce. Higher-order subtyping. In *IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, June 1994. An earlier version appeared as University of Edinburgh technical report ECS-LFCS-94-280 and Universität Erlangen-Nürnberg Interner Bericht IMMD7-01/94, February 1994.