Polarized Higher-Order Subtyping (Extended Abstract)

Martin Steffen
Institut für Informatik und Praktische Mathematik
Christian-Albrechts–Universität zu Kiel
Preußerstraße 1–9, D-24105 Kiel, Deutschland
Tel: +49 (431) 5604-79
Fax: +49 (431) 5661-43
Email: ms@informatik.uni-kiel.de

9th December 1997

Abstract

The calculus of higher-order subtyping, known as $F^{\omega}_{\leq}$, has been used as a prototypical core calculus for typed object-oriented languages. The versions considered in the literature usually support pointwise subtyping of type operators, only, where two applications $S\ U$ and $T\ U$ are in subtype relation, if the type operators S and T are. A natural extension going beyond pointwise subtyping distinguishes between monotone, antimonotone , and invariant operators. Thus, $T\ U_1$ is a subtype of $T\ U_2$, if $U_1 \leq U_2$ and provided T is a monotone operator.

The paper extends $F^{\omega}_{\leq}$ by polarized application and proves decidability of the subtyping relation. To take monotonicity information into account, the kinding system does not only characterize domain and range of a type operator of kind $K_1 \to K_2$, but uses, for instance, the kind $K_1 \to^{+} K_2$ for monotone type operators from K_1 to K_2. The corresponding set of rules leads to an interdependence of the subtyping and the kinding system. This contrasts with pure $F^{\omega}_{\leq}$ with point-wise subtyping only, where the kinding rules do not depend on subtyping. To retain decidability of the system, the equal-bounds subtyping rule for All-types is rephrased in the polarized setting as a mutual-subtype requirement of the upper bounds. To avoid the complexity of this mutual dependence in the proof of decidability, we reformulate the system, yielding a *stratified* version, where subtyping still depends on kinding as the more primitive statements but not vice versa.

Keywords: type systems, subtyping, polymorphic λ-calculi for object-oriented languages.

1 Introduction

The study of typed functional calculi, i.e., typed λ-calculi, flexible and expressive enough to provide a theoretical framework for object-oriented languages has received considerable attention in the literature over the past ten years. The object of study in this paper is $F^{\omega}_{\leq}$ (called "F-omega-sub"), one prominent representative of λ-calculi for object-oriented programming. It combines the ω-order polymorphic λ-calculus F^{ω} [14] and the polymorphic λ-calculus with subtyping, known as $F_{\leq}$. The meta-theoretical properties of $F_{\leq}$ have been extensively studied in the literature (cf. [6] [13] [11] [3] [20] et. al.). The integration of both subtyping and type operators, i.e., functions from types to types, leads to $F^{\omega}_{\leq}$ [5] [17] [4], a formal system sufficiently expressive to model class inheritance with late binding, encapsulation, and subtype polymorphism in a uniform framework, and thus well-known as one core calculus for class-based typed object-oriented languages [19] [16].

This paper presents an extension of $F^{\omega}_{\leq}$ to include information about the *monotonicity of type operators* allowing a more general form of subtyping, going beyond pointwise subtyping. We investigate its proof-theory, establishing decidability of "polarized $F^{\omega}_{\leq}$". What makes the proof-theory principally more challenging than for pure $F^{\omega}_{\leq}$(cf. eg. [18]) is that the generalization leads to a mutual dependence of kinding and subtyping not present in pure $F^{\omega}_{\leq}$.

Similar variants of $F^{\omega}_{\leq}$ with monotonicity information have been proposed in the literature, most notably in Cardelli's widely cited note [5] or in a restricted form in Hofmann and Pierce [16], but so far the meta-theory of such calculi has not been tackled. We mention some related work, before presenting the main parts of the calculus in Section 2 — the subtyping and the kinding system. We highlight the difficulties to obtain an algorithm for subtyping and sketch the proof of the main result: decidability of subtyping. The complete system and the proofs can be found in the full version of the paper and in [24].

Comparison with related work

In [16], Hofmann and Pierce present a uniform, type-theoretic account of object-oriented concepts of class-based languages, including encapsulation, subtyping, and message passing. They show that $F^{\omega}_{\leq}$'s object model fits in their uniform framework in case one restricts the object's interfaces to *positive* signatures, i.e., type operators of kind $\star \to^{+} \star$ to use the notation of this paper. To be able to characterize such monotone signatures, Hofmann and Pierce extend $F^{\omega}_{\leq}$ (or rather a fragment of it) with a predicate *pos* on type operators of kind $\star \to \star$, which corresponds to our $+$-polarity. The system we present here generalizes the definitions in [16]. First of all, *pos* only handles type variables of the base kind $\star$, and polarized operators of kind $\star \to \star$, which corresponds to a fragment of F^{3} with subtyping. Another difference is that Hofmann and Pierce do not consider the additional polarities $\pm$ and $\circ$. Similarly, Fisher [12] investigates the *variance* of type operators, there called row functions. She also restricts herself to operators of kind $\star \to \star$, so her variance analysis is simpler than our system.

$F^{\omega}_{\leq}$ is an impredicative type theory, where terms depend on terms and types, and types take types as parameters (in which case they are called type operators). Systems where types can depend on terms, are said to have *dependent types*. Recently, interest in calculi with dependent types extended by subtyping has started. Aspinall and Compagnoni [2] investigate $\lambda P_{\leq}$, a first-order calculus with dependent types (known as λP or $\lambda\Pi$) extended by subtyping,

establishing decidability of the system. Chen [7] studies a different formulation of this system, which he calls $\lambda\Pi_{\leq}$. The inclusion of dependent types leads to a cyclic dependence of the typing, the subtyping, and the kinding statements, posing similar proof-theoretic problems as in the study pursued here. The results have recently been extended to the calculus of constructions [8]. None of these calculi includes bounded universal quantification, which significantly simplifies the meta-theory.

The formulation of $F^{\omega}_{\leq}$ investigated here contains an asymmetry concerning the bindings of type variables. Whilst the type variable in bounded universal quantifier types $All(A{\leq}T_1{:}K_1)T_2$ ranges over all subtypes of its upper bound, no such restriction is expressible for the type operators. It is relative straightforward to formulate a system including operators with non-trivial upper bound. Indeed, it was in this more general form, that $F^{\omega}_{\leq}$ appeared in Cardelli's note [5]. The kind of a type operator $Fun(A{\leq}T_1{:}K_1)T_2$ with bounded abstract type parameter in this case has to capture the fact that no longer all types of kind K_1 are legitimate arguments for the operator, as expressed by the kind $K_1 \to K_2$, but only those less or equal type T_1, which calls for generalizing arrow kinds to Pi-kinds of the form $\Pi A{\leq}T_1{:}K_1.K_2$. The corresponding elimination rule then reads:

$$\frac{\Gamma \vdash S \in \Pi A{\leq}T_1{:}K_1.K_2 \qquad \Gamma \vdash T \leq T_1 \in K_1}{\Gamma \vdash S\ T \in [T/A]K_2} \qquad \text{(K-Pi-E)}$$

The second premise renders kinding dependent upon subtyping, leading to a mutual dependence of kinding and subtyping as in our system (cf. the rules for application in Section 2), only that the interdependence is due to bounded operator abstraction, not to polarized application. Compagnoni and Goguen [9] study such a system, proving decidability of $F^{\omega}_{\leq}$ with bounded operator abstraction (without subkinding, without polarities, and, to preserve decidability, using the equal-bounds subtyping rule for All-types). While on an abstract level the initial situation — mutual dependence of subtyping and kinding — seems similar, the proofs in [9] and presented here differ in important respects. The approach pursued in this work, breaking the cycle by rendering kinding independent of subtyping by some sort of variance analysis, is out of question for bounded operator abstraction, as the very nature of bounded operator abstraction requires this dependence in rule K-Pi-E. Instead, in an elegant proof Compagnoni and Goguen handle kinding and subtyping at the same time. They introduce a *typed operational semantics* to obtain cut-elimination, subject reduction, and decidability, where the core of the argument is a strong normalization proof à la Tait and Martin-Löf. The typed operational semantics can be understood as a derivation system combining subtyping statements, kinding statements, and normal-form reduction. It is a rough analogue to the algorithmic formulations of subtyping in [10] or [18] or in this paper.

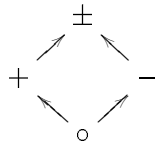
Monotonicity information for type variables or type operators is relevant also for *type inference*. By reconstructing (or trying to reconstruct) omitted types from the context, type inference can free the programmer from explicitly feeding a type as parameter to a polymorphic function, or from annotating the type of the parameter of a function. Recently, in a couple of papers, Pierce and Turner [23] [22] [21] investigate partial, local type-inference for a Kernel-Fun variant of the second order $F_{\leq}$ including a bottom type as the dual to the more common maximal type *Top*. Considering the case of inferring an omitted type argument of a polymorphic function, the types to be inferred are represented by type variables and at a certain stage during the inference process, the algorithm has to calculate a substitution of these variables, such that the result after the substitution is minimal and where the type

replacing the variable is drawn from an interval in the subtyping hierarchy. Obviously, the choice of the replacement is determined by the way the variable it replaces occurs inside the type; for example, if the type variable occurs monotonely, choosing the minimal substitution satisfying the constraint yields a minimal result. Their type inference algorithm can also handle type operator constants in case they are *monotone*, for instance, the *List* type constructor of kind $\star \to^+ \star$. Lacking is support for user-definable type-operators as in the higher-order $F^\omega_\leq$. A system of polarized higher-order subtyping as presented here should be helpful in the generalization.

2 Extending $F^\omega_\leq$ with monotonicity information

In this section we sketch the extension of $F^\omega_\leq$ by subtyping rules for a more general rule of application, taking monotonicity information into account. Besides pointwise subtyping, the system allows to derive for example $\Gamma \vdash T\ U_1 \leq T\ U_2$, if $\Gamma \vdash U_1 \leq U_2$ and provided T is monotone. In effect, $F^\omega_\leq$ appeared in a form including monotonicity information in Cardelli's note [5]. The syntax of polarized $F^\omega_\leq$ extends the one for the pure calculus (cf. for example [18]) by annotating arrow kinds with one of four different polarities. Thus a kind K is either a base kind $\star$ for the proper types, or an arrow kind $K_1 \to^? K_2$ for type operators from K_1 to K_2 with ? as polarity.

The polarities come in four sorts: $+$ and $-$ characterize monotone and antimonotone operators, $\circ$ stands for constant ones, and finally $\pm$ for absence of information. Under this interpretation, the polarities are ordered as given in the diagram on the right. The corresponding reflexive and transitive relation on polarities is written $? \leq ?'$. In accordance with usual conventions, the smaller the polarity, the better.



The *types* of the calculus — basically the same as from pure $F^\omega_\leq$ — are inductively built from type variables A, type operators $Fun(A{:}K)T$, type applications $T_1\ T_2$, maximal types $Top(K)$ of kind K, function types $T_1 \to T_2$, and universally quantified types $All(A{\leq}T_1{:}K)T_2$. The *terms* finally are formed of term variables x, term abstraction $fun(x{:}T)t$ and application $t_1\ t_2$, and type abstraction $fun(A{\leq}T{:}K)t$ and application $t\ T$. The type system for polarized $F^\omega_\leq$ contains the following five sets of statements:

$K \leq K'$	K is a subkind of K'
$\vdash \Gamma\ ok$	Γ is a well-formed context
$\Gamma \vdash T \in K$	type T has kind K in context Γ
$\Gamma \vdash S \leq T \in K$	S is a subtype of T in Γ
$\Gamma \vdash t \in T$	term t has type T in Γ

The derivability for each class of statements is given inductively by a set of inference rules. The ones for *context formation* $\vdash \Gamma\ ok$ are standard, assuring that each variable declaration — either for term variables $\Gamma, x{:}T$ or for type variables $\Gamma, A{\leq}T{:}K$ — adds only fresh variables to the context. Standard is also the typing system for statements $\Gamma \vdash t \in T$, and we elide both here. For *subkinding*, the order on polarities from above is lifted to the level of kinds:

$$\frac{K_1' \leq K_1 \qquad K_2 \leq K_2' \qquad ? \leq ?'}{K_1 \to^? K_2 \leq K_1' \to^{?'} K_2'} \qquad \text{(K-Sub)}$$

The two subsystems we present in detail are the ones for kinding and for subtyping.

Subtyping The difference between the $F^{\omega}_{\leq}$ and the polarized version of this section is most visible in the application rules. The pointwise application rule S-App lifts the subtype relation of two type operators to type applications, stipulating $S\ U$ and $T\ U$ in subtype relation, provided S and T are. To extend the system beyond pointwise subtyping we have to characterize the polarity of operators — this will be the the task of the kinding system in the following section — and to include new subtyping rules exploiting the additional knowledge about the polarity of the operators. The basic intuition about a monotone type operator T is that, applied to two arguments in subtype ordering, the application is ordered likewise. This is captured by the rule S-App+ below. The premise $\Gamma \vdash T +$ expresses monotonicity of the type operator T, abbreviating a kinding statement of the form $\Gamma \vdash T \in K_1 \to^{+} K_2$ for some kinds K_1 and K_2. Apart from monotone operators, we need to consider antimonotone ones, and we include a dual rule S-App−.

Furthermore, we would like to be able to denote that an operator does not depend upon its arguments at all, i.e., that it is *constant* in its arguments; this is formalized by rule S-App∘. Note that for a type operator being constant does not mean it throws away its arguments. In other words, $\Gamma \vdash T \circ$ does not imply β-equivalence of $T\ U_1$ and $T\ U_2$. The reason is that a type variable can be declared as constant operator, but it cannot swallow its arguments.

To make the system symmetric, we add a fourth polarized application rule, more generous than pointwise application, in that it does not insist on its arguments being identical, but requiring only that each argument is a subtype of the other. Writing $\Gamma \vdash U_1 \geqslant U_2 \in K$ for such pairs of subtyping statements $\Gamma \vdash U_1 \leq U_2 \in K$ and $\Gamma \vdash U_2 \leq U_1 \in K$, we can write this last application rule S-App±.

Having type variables act as constant type operators means that not only constant operators differ from the ones ignoring their arguments, but also that the notion of β-equivalence of two types S and T and the fact that the two types are in mutual subtype relationship are to be distinguished. This distinction is not present in pure $F^{\omega}_{\leq}$ and neither in the "polarized" calculi of Fisher [12] and Hofmann and Pierce [16]. But this fourth rule neither appears in the higher-order setting of Cardelli [5].

The remaining rules resemble the ones for pure $F^{\omega}_{\leq}$. The conversion rule S-Conv and transitivity S-Trans define $\leq$ an order relation on types, respecting $\beta\top$-equivalence.[1] A type variable is smaller than its upper bound in the context where additionally its kind has to conform to the kinding constraint of the subtyping statement. Each kind contains a maximal type by rule S-Top. As usual, the rule S-Arrow for function types behaves contravariantly on the left-hand side of the arrow, and covariantly on the right. The rule for type operators compares the bodies of the two operators and use the kinding system to determine their polarity. Finally the rule for universally quantified types. To retain decidability of the subtyping system, we stick close to the decidable variant of $F^{\omega}_{\leq}$, which insists on equal upper bounds for the All-types. By the same arguments that had us introduce S-App±, we are led to relax the conditions upon the two bounds of universally quantified types, requiring that they must be mutually smaller than each other.

$$\frac{S =_{\beta\top} T \qquad \Gamma \vdash S, T \in K}{\Gamma \vdash S \leq T \in K} \qquad \text{(S-Conv)}$$

[1] $\beta\top$-reduction is a minor variant of β-reduction, including $Top(K_1 \to^{?} K_2)\ T \longrightarrow_{\top} Top(K_2)$ as reduction step.

$$\frac{\Gamma \vdash S \leq U \in K \quad \Gamma \vdash U \leq T \in K}{\Gamma \vdash S \leq T \in K} \qquad \text{(S-Trans)}$$

$$\frac{\Gamma \vdash A \in K}{\Gamma \vdash A \leq \Gamma(A) \in K} \qquad \text{(S-TVar)}$$

$$\frac{\Gamma \vdash S \in K \qquad \Gamma \vdash \mathit{Top}(K') \in K}{\Gamma \vdash S \leq \mathit{Top}(K') \in K} \qquad \text{(S-Top)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathit{Fun}(A{:}K_1)S, \mathit{Fun}(A{:}K_1)T \in K_1 \to^? K_2 \\ \Gamma, A{:}K_1 \vdash S \leq T \in K_2\end{array}}{\Gamma \vdash \mathit{Fun}(A{:}K_1)S \leq \mathit{Fun}(A{:}K_1)T \in K_1 \to^? K_2} \qquad \text{(S-Abs)}$$

$$\frac{\Gamma \vdash S \leq T \in K_1 \to^{\pm} K_2 \qquad \Gamma \vdash U \in K_1}{\Gamma \vdash S\ U \leq T\ U \in K_2} \qquad \text{(S-App)}$$

$$\frac{\Gamma \vdash T \in K_1 \to^{\circ} K_2 \qquad \Gamma \vdash U_1, U_2 \in K_1}{\Gamma \vdash T\ U_1 \leq T\ U_2 \in K_2} \qquad \text{(S-App}\circ\text{)}$$

$$\frac{\Gamma \vdash T \in K_1 \to^{+} K_2 \qquad \Gamma \vdash U_1 \leq U_2 \in K_1}{\Gamma \vdash T\ U_1 \leq T\ U_2 \in K_2} \qquad \text{(S-App}+\text{)}$$

$$\frac{\Gamma \vdash T \in K_1 \to^{-} K_2 \qquad \Gamma \vdash U_2 \leq U_1 \in K_1}{\Gamma \vdash T\ U_1 \leq T\ U_2 \in K_2} \qquad \text{(S-App}-\text{)}$$

$$\frac{\Gamma \vdash T \in K_1 \to^{\pm} K_2 \qquad \Gamma \vdash U_1 \gtrless U_2 \in K_1}{\Gamma \vdash T\ U_1 \leq T\ U_2 \in K_2} \qquad \text{(S-App}\pm\text{)}$$

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

$$\frac{\Gamma \vdash S_1 \gtrless T_1 \in K_1 \qquad \Gamma, A{\leq}S_1{:}K_1 \vdash S_2 \leq T_2 \in \star}{\Gamma \vdash \mathit{All}(A{\leq}S_1{:}K_1)S_2 \leq \mathit{All}(A{\leq}T_1{:}K_1)T_2 \in \star} \qquad \text{(S-All)}$$

Kinding Next the kinding rules, mutually dependent with the rules for subtyping of above. As mentioned, arrow kinds come decorated with polarities to express monotonicity information. For instance, the type operator $\mathit{Fun}(A{:}\star)\,\mathit{Top}(\star) \to A$ will carry the kind $\star \to^+ \star$ indicating that it operates monotonely on its arguments of kind $\star$. A type operator $\mathit{Fun}(A{:}K)T$ is monotone, if for all types U_1 and U_2 kinded appropriately, $U_1 \leq U_2$ implies $[U_1/A]T \leq [U_2/A]T$ (cf. rule K-Arrow-I+ below). With the ordering on kinds generated by K-Sub, we get subsumption on the level of kinds. The kind of a type variable A is determined by its declaration $\mathit{kind}_\Gamma A$ in the context Γ. The type $\mathit{Top}(K)$ is maximal for the respective kind K. Since, regardless of type S, we think of the application $\mathit{Top}(K_1 \to^? K_2)\ S$ as bigger than all types of the appropriate kind K_2, the operator $\mathit{Top}(K_1 \to^? K_2)$ behaves constantly on its arguments. This also justifies the abbreviation $\Gamma_1, A{:}K, \Gamma_2$ for $\Gamma_1, A{\leq}\mathit{Top}(K){:}K, \Gamma_2$, as the upper bound $\mathit{Top}(K)$ carries no further formation. The four rules for arrow-introduction are responsible for the dependence of subtyping on kinding: the polarity of a type operator is determined by

a subtyping derivation, with the appropriate assumption about the formal parameters in the context. As in pure $F^{\omega}_{\leq}$, arrow- and All-types finally carry the unique kind $\star$.

$$\frac{K' \leq K \qquad \Gamma \vdash T \in K'}{\Gamma \vdash T \in K} \qquad \text{(K-Subsumption)}$$

$$\frac{\vdash \Gamma \; ok}{\Gamma \vdash A \in kind_{\Gamma} A} \qquad \text{(K-TVar)}$$

$$\frac{\vdash \Gamma \; ok}{\Gamma \vdash Top(\star) \in \star} \qquad \text{(K-Top}\star\text{)}$$

$$\frac{\Gamma \vdash Top(K_2) \in K_2'}{\Gamma \vdash Top(K_1 \to^{?} K_2) \in K_1 \to^{\circ} K_2'} \qquad \text{(K-Top)}$$

$$\frac{\Gamma, A{:}K_1 \vdash T \in K_2}{\Gamma \vdash Fun(A{:}K_1)T \in K_1 \to^{\pm} K_2} \qquad \text{(K-Arrow-I}\pm\text{)}$$

$$\frac{\Gamma, A{:}K_1 \vdash T \in K_2 \quad \Gamma, A_2{:}K_1, A_1{\leq}A_2{:}K_1 \vdash [A_1/A]T \leq [A_2/A]T \in K_2}{\Gamma \vdash Fun(A{:}K_1)T \in K_1 \to^{+} K_2} \qquad \text{(K-Arrow-I}+\text{)}$$

$$\frac{\Gamma, A{:}K_1 \vdash T \in K_2 \quad \Gamma, A_2{:}K_1, A_1{\leq}A_2{:}K_1 \vdash [A_2/A]T \leq [A_1/A]T \in K_2}{\Gamma \vdash Fun(A{:}K_1)T \in K_1 \to^{-} K_2} \qquad \text{(K-Arrow-I}-\text{)}$$

$$\frac{\Gamma, A{:}K_1 \vdash T \in K_2 \quad \Gamma, A_1{:}K_1, A_2{:}K_1 \vdash [A_1/A]T \leq [A_2/A]T \in K_2}{\Gamma \vdash Fun(A{:}K_1)T \in K_1 \to^{\circ} K_2} \qquad \text{(K-Arrow-I}\circ\text{)}$$

$$\frac{\Gamma \vdash S \in K_1 \to^{?} K_2 \qquad \Gamma \vdash T \in K_1}{\Gamma \vdash S\; T \in K_2} \qquad \text{(K-Arrow-E)}$$

$$\frac{\Gamma \vdash T_1 \in \star \qquad \Gamma \vdash T_2 \in \star}{\Gamma \vdash T_1 \to T_2 \in \star} \qquad \text{(K-Arrow)}$$

$$\frac{\Gamma, A{\leq}T_1{:}K_1 \vdash T_2 \in \star}{\Gamma \vdash All(A{\leq}T_1{:}K_1)T_2 \in \star} \qquad \text{(K-All)}$$

2.1 Proof sketch

This section sketches the proof of decidability of polarized subtyping. We start with a discussion of the main difficulties to obtain an algorithm.

The principal complication compared to pure $F^{\omega}_{\leq}$ is the *mutual dependence* of subtyping and kinding statements via the rules for application and abstraction on type level. Lacking the subformula property, the rule for *transitivity* introduces a non-determinism into the system, not tolerable for an algorithm; this problem is common to all subtyping systems. Specific to subtyping systems with a notion of conversion on type level, the second source of non-determinism is the *conversion* rule. We next hint at how to deal with these three obstacles.

Stratification of the subtyping system Instead of a direct approach, treating subtyping and kinding at the same time, we break the interdependence of both, such that, as in pure $F^{\omega}_{\leq}$, subtyping still depends upon kinding as the more primitive statement, but not vice versa. This amounts to finding an independent characterization of the polarity of type operators not relying on subtyping derivations: instead of comparing $[A_1/A]T \leq [A_2/A]T$ under appropriate subtyping assumptions for the type variables A_1 and A_2 to characterize the polarized kind of a type operator $Fun(A{:}K_1)T$, we directly determine its polarity by looking at the positions in which its formal parameter A occurs inside the operator's body. This will lead to a new set of statements $\Gamma \vdash T\ ?_A$, judging the occurrence of variables in types. With these statements for variable occurrence we can write the rule for arrow introduction as follows:

$$\frac{\Gamma, A{:}K_1 \vdash T\ ?_A \qquad \Gamma, A{:}K_1 \vdash T \in K_2}{\Gamma \vdash Fun(A{:}K_1)T \in K_1 \to^{?} K_2} \qquad \text{(K-Arrow-I?)}$$

The definition of the rules for $\Gamma \vdash T\ ?_A$ is straightforward. With the polarities ordered, we introduce subsumption for the corresponding polarity judgments. A type variable A occurs positively in A itself; if a variable A does not occur freely at all, its polarity is constant. For arrow-types we have to take into account that the subtype relation behaves contravariantly on the left-hand side of the arrow and covariantly on its right-hand side. Thus the polarity of a variable in an arrow-type $T_1 \to T_2$ cannot be better than either its polarity in T_2 or the negation of its polarity on the contravariant side T_1.[2] For a type variable occurring with non-trivial polarity, i.e. other than $\pm$, inside an All-type, we require it constant in the All-type's upper bound. The occurrence of a variable inside a type operator is determined by its occurrence inside the operator's body. Finally the statements for type applications: the polarity of A in an application $S\ T$ is calculated using the operator $\vee$ on the right. The relation $\Gamma \vdash T\ ?_A$ is then given inductively by the following set of rules:

$\times$	$\circ$	$+$	$-$	$\pm$
$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
$+$	$\circ$	$+$	$-$	$\pm$
$-$	$\circ$	$-$	$+$	$\pm$
$\pm$	$\circ$	$\pm$	$\pm$	$\pm$

$$\frac{?' \leq\ ? \qquad \Gamma \vdash T\ ?'_A}{\Gamma \vdash T\ ?_A} \qquad \frac{\vdash \Gamma\ ok}{\Gamma \vdash A +_A} \qquad \frac{A \notin fv(T) \qquad \vdash \Gamma\ ok}{\Gamma \vdash T \circ_A}$$

$$\frac{\Gamma \vdash T_1\ ?^1{}_A \qquad \Gamma \vdash T_2\ ?^2{}_A \qquad ?\ = \neg\, ?^1 \vee ?^2}{\Gamma \vdash T_1 \to T_2\ ?_A}$$

$$\frac{\Gamma \vdash T_1 \circ_A \qquad \Gamma, A'{\leq}T_1{:}K \vdash T_2\ ?_A \qquad A' \neq A}{\Gamma \vdash All(A'{\leq}T_1{:}K)T_2\ ?_A}$$

$$\frac{\Gamma, A'{:}K \vdash T\ ?_A \qquad A' \neq A}{\Gamma \vdash Fun(A'{:}K)T\ ?_A}$$

[2] In the rule for arrow-types, $\vee$ denotes the least upper bound in the small lattice of polarities, and negation toggles $+$ and $-$ and acts as the identity on the other two polarities.

$$\frac{\Gamma \vdash S\ ?^1{}_A \qquad \Gamma \vdash S\ ?^2 \qquad \Gamma \vdash T\ ?^3{}_A \qquad ? = ?^1 \vee (?^2 \times ?^3)}{\Gamma \vdash S\ T\ ?_A}$$

Fisher [12] defines a similar set of operations to determine the variance (or polarity) of a type variable in a "row", a restricted form of type operator. The calculation of polarity for applications is simpler in [12], because only operators of kind $\star \to \star$ (called row functions) are treated. Hence there is no need to distinguish between constant appearance and the fact, that a variable does not occur at all.

Directed version of the subtyping system The conversion rule S-Conv allows to convert a type to $\beta\top$-equivalent ones. For an algorithm we cannot tolerate such liberty, that in order to derive the subtype relationship between two types, we have to check all (i.e. infinitely many) $\beta\top$-equivalent pairs. The usual and obvious approach [1] [10] [2] [8] [18] is to prove that it suffices to check the subtype relation for the unique *normal forms* of types, only. As a first step towards a system operating exclusively on normal forms we use a directed version of the system by distributing the effect of the undirected conversion rule over the rest of the subtype rules. For example, instead of S-Arrow we will use the following rule:

$$\frac{S \longrightarrow^*_{\beta\top} S_1 \to S_2 \qquad T \longrightarrow^*_{\beta\top} T_1 \to T_2 \qquad \Gamma \vdash T_1 \leq S_1 \in \star \qquad \Gamma \vdash S_2 \leq T_2 \in \star}{\Gamma \vdash S \leq T \in \star}$$

To turn this system into an algorithm, we have to prove that the non-deterministic reduction relation in the premises can be replaced by *normalizing reduction*.

Elimination of transitivity The standard problem to obtain an algorithm for a subtyping system is to get rid of the rule of transitivity. It is well-known that subtyping systems like the one presented here *do not possess* a transitivity elimination property: in the presence of a variable rule such as S-TVar it is easy to think of subtyping statements whose derivability depends on the rule of transitivity. To compensate for the effect of transitivity in these cases we use a rule replacing a type variable occurring in head position, here A, in an application by its upper bound $\Gamma(A)$:

$$\frac{S \longrightarrow^!_{\beta\top} A\ S_1 \ldots S_n \quad \Gamma \vdash A\ S_1 \ldots S_n \in K \qquad \Gamma \vdash \Gamma(A)\ S_1 \ldots S_n \leq T \in K}{\Gamma \vdash S \leq T \in K} \qquad \text{(R-Promote)}$$

With this rule added, admissibility of the transitivity rule is provable by an inductive cut elimination argument.

2.2 Main results

We just mention the key steps to achieve main result: decidability of polarized subtyping.

1. *Decidability of kinding*: after stratification, decidability of kinding is relative straightforward. Due to subkinding, the system does not enjoy a unique kinding property, and the kinding algorithm proceeds by minimal kinding synthesis.

2. *Subject reduction for subtyping*: Preservation of subtyping under substitution and reduction is needed in the (stratified) subtyping system, as step towards the subtyping algorithm which works on types in normal forms, only.

3. *Termination of the subtyping algorithm*: The proof is a generalization of the one for pure $F^{\omega}_{\leq}$ in [18]. The rule that complicates termination is R-PROMOTE: in replacing a type variable by its upper bound, the subgoal may contain new $\beta\top$-redices not present in the conclusion of the rule. Termination of the algorithm follows from strong normalization of a reduction relation combining $\beta\top$-reduction and replacement of a type variable by its upper bound in the context. The *asymmetric* nature of the subtyping rule S-ALL further complicates the argument compared to pure $F^{\omega}_{\leq}$. Since this rule does not insist on the upper bounds to be identical, but only in mutual subtype relationship, a termination argument is needed for a relation, where a type variable may be replaced by a type, which is smaller and greater than its upper bound in the context at the same time.

4. *Mutual subtype relationship implies absence of promotion:* The fact that for $\Gamma \vdash S \gtrless T \in K$ the derivation does not even use the transitivity implicit in R-PROMOTE is needed for proving the stratified version equivalent to the original formulation.

5. *Cut elimination*: by a standard cut-elimination argument using induction on the length of derivations.

The items 1–5 are proven in the stratified version of the system. As a result one arrives at an algorithm, basically the stratified formulation of the subtyping system without the rule of transitivity and with normalizing reduction. Thus decidability of the original, non-stratified system follows from the proof of equivalence between the two presentations, shown by a big induction over derivations.

Proposition 2.1 (Decidability of subtyping) The subtyping relation $\Gamma \vdash S \leq T \in K$ for polarized higher-order subtyping is decidable.

References

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer, 1996.

[2] David Aspinall and Adriana Compagnoni. Subtyping dependent types. In *Eleventh Annual Symposium on Logic in Computer Science (LICS)*. IEEE, Computer Society Press, July 1996.

[3] Val Breazu-Tannen, Thierry Coquand, Carl Gunter, and André Ščedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991. Also in the collection [15].

[4] Kim Bruce and John Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Nineteenth Annual Symposium on Principles of Programming Languages (POPL) (Albuquerque, NM)*, pages 316–327. ACM, January 1992.

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

[6] Luca Cardelli, Simone Martini, John Mitchell, and André Ščedrov. An extension of system F with subtyping. *Information and Computation*, 109(1–2):4–56, 1994. A preliminary version appeared in TACS '91 (Sendai, Japan, pp. 750–770, LNCS 526).

[7] Gang Chen. Dependent type systems with subtyping; type level transitivity elimination. Technical report, Laboratoire d'Informatique ENS. and Université de Paris 7, July 1997. To appear in the Proceedings of the KIT 97 Summer School and Workshop, Beijing.

[8] Gang Chen. Subtyping calculus of constructions (extended abstract). In *Proceedings of Mathematical Foundations of Computer Science (MFCS '97)*, 1997. A longer version is available through `http://www.dmi.ens.fr/~gang`.

[9] Adriana Compagnoni and Healfdene Goguen. Typed operational semantics for higher order subtyping. Technical Report ECS-LFCS-97-361, Department of Computer Science, University of Edinburgh, 1997. Submitted for publication in *Information and Computation*.

[10] Adriana B. Compagnoni. *Higher-Order Subtyping with Intersection Types*. PhD thesis, Catholic University, Nijmegen, January 1995.

[11] Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption: Minimum typing and type-checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2:55–91, 1992. A preliminary version appeard as LIENS Report Nr. 90-10, 1990. Also in the collection [15].

[12] Kathleen Fisher. *Type Systems for Object-Oriented Languages*. PhD thesis, Stanford University, August 1996.

[13] Giorgio Ghelli. Modelling features of object-oriented languages in second order functional languages with subtypes. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages (REX Workshop)*, volume 489 of *Lecture Notes in Computer Science*, pages 311–340. Springer, 1991.

[14] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupure dans l'arithmetique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

[15] Carl A. Gunter and John C. Mitchell. *Theoretical Aspects of Object-Oriented Programming, Types, Semantics, and Language Design*. Foundations of Computing Series. MIT Press, 1994.

[16] Martin Hofmann and Benjamin Pierce. A unifying type-theoretic framework for objects. *Journal of Functional Programming*, 5(4):593–635, October 1995. Previous versions appeared in the Symposium on Theoretical Aspects of Computer Science, 1994, (pages 251–262) and, under the title "An Abstract View of Objects and Subtyping (Preliminary Report)," as University of Edinburgh, LFCS technical report ECS-LFCS-92-226, 1992.

[17] John C. Mitchell. Toward a typed foundation for method specialization and inheritance. In *Seventeenth Annual Symposium on Principles of Programming Languages (POPL) (San Fancisco, CA)*, pages 109–124. ACM, January 1990. Also in the collection [15].

[18] Benjamin Pierce and Martin Steffen. Higher-order subtyping. *Theoretical Computer Science*, 176(1,2):235–282, 1997. A shorter version appeared in the Proceedings IFIP Working Conference on Programming Concepts, Methods and Calculi (p. 511–530), 1994. Also LFCS technical report ECS-LFCS-94-280 and Interner Bericht IMMD7-01/94, Universität Erlangen.

[19] Benjamin Pierce and David Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 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".

[20] Benjamin C. Pierce. Bounded quantification is undecidable. *Information and Computation*, 112(1):131–165, July 1994. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994). A preliminary version appeared in POPL '92.

[21] Benjamin C. Pierce. Bounded quantification with bottom. Technical Report CSCI TR #492, Indiana University, November 1997.

[22] Benjamin C. Pierce and David N. Turner. Local type argument synthesis with bounded quantification. Technical Report CSCI TR #495, Indiana University, November 1997.

[23] Benjamin C. Pierce and David N. Turner. Local type inference. In *Proceedings of POPL '98*. ACM, 1998. Also as Indiana University Technical Report CSCI TR #493.

[24] Martin Steffen. *Polarized Higher-Order Subtyping*. Dissertation, Universität Erlangen, 1997. To appear.