Decidable Bounded Quantification

Giuseppe Castagna[*]
LIENS(CNRS)-DMI
castagna@dmi.ens.fr

Benjamin C. Pierce[†]
LFCS, Univ. of Edinburgh
bcp@dcs.ed.ac.uk

February 27, 1994

Abstract

The standard formulation of bounded quantification, system $F_{\leq}$, is difficult to work with and lacks important syntactic properties, such as decidability. More tractable variants have been studied, but those studied so far either exclude significant classes of useful programs or lack a compelling semantics.

We propose here a simple variant of $F_{\leq}$ that ameliorates these difficulties. It has a natural semantic interpretation, enjoys a number of important properties that fail in $F_{\leq}$, and includes all of the programming examples for which $F_{\leq}$ has been used in practice.

Since publication of this paper in POPL '94, we have become aware of a serious shortcoming in the system studied here. Although the subtyping relation is well behaved, the typing *relation, which we did not consider explicitly, fails to possess the crucial property of minimal typing. More details can be found in two messages from the* Types *forum, reproduced in Appendix B.*

1 Introduction

Since its introduction by Cardelli and Wegner [CW85], the idea of *bounded quantification* has become a fundamental tool in the study of statically typed programming languages combining subtyping and polymorphism.

Systems with bounded quantification are formed by enriching a typed λ-calculus, such as System F [Gir72, Rey74] or F^{ω} [Gir72], with a *subtype relation* on type expressions, $\Gamma \vdash$

An extended abstract of this paper has been presented in the *21st Annual Symposium on Principles of Programming Languages.* Portland, January 1994.

[*]Supported by grant no. 203.01.56 of the Consiglio Nazionale delle Ricerche, Comitato Nazionale delle Scienze Matematiche, Italy

[†]Supported by funding from the U.K. Science and Engineering Research Council and the ESPRIT Basic Research Actions TYPES and CONFER. This work was begun while the author was a visitor at INRIA-Roquencourt.

$S \leq T$, and extending the introduction and elimination rules for polymorphic types so that each quantifier mentions a *bound* for the variable it introduces. The standard formulation of second-order bounded quantification, often called $F_{\leq}$ ("F-sub"), was given by Curien and Ghelli [CG92]. Its subtype relation includes the following rule for comparing polymorphic types:

$$(\forall\text{-}\mathsf{orig}) \qquad \frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \cup \{X \leq T_1\} \vdash S_2 \leq T_2}{\Gamma \vdash \forall(X{\leq}S_1)S_2 \leq \forall(X{\leq}T_1)T_2} \qquad X \notin dom(\Gamma)$$

Intuitively, this rule may be read as follows. A type T of the form $\forall(X{\leq}T_1)T_2$ describes a collection of polymorphic values (functions from types to values) each mapping subtypes of T_1 to instances of T_2. If T_1 is a subtype of S_1, then the domain of T is smaller than that of $\forall(X{\leq}S_1)T_2$, so this type has a weaker constraint and therefore it describes a larger collection of polymorphic values. Moreover, if given the type $\forall(X{\leq}S_1)S_2$ for each type U that is an acceptable argument to the functions in both collections (i.e., one that satisfies the more stringent requirement $U{\leq}T_1$), the U-instance of S_2 is a subtype of the U-instance of T_2, then T is a "pointwise weaker" constraint and again describes a larger collection of polymorphic values. In other terms the two functions are pointwise compared only on their common domain. (For brevity, we elide the standard well-formedness condition that the bound variable X should not already occur in the context Γ.)

Though semantically quite natural, this rule is responsible for the loss of numerous desirable syntactic properties in $F_{\leq}$. To begin with, the subtype relation is undecidable [Pie93, Ghe93b], which implies the undecidability of the typechecking problem even for explicitly typed terms. Even more seriously, $F_{\leq}$ lacks greatest lower bounds for compatible (i.e. lower-bounded) sets of types [Ghe90], which blocks certain useful forms of argument by induction on the subtype relation. The most natural extension of $F_{\leq}$ with bounded existential types [GP93] even fails to possess a minimal type for every typeable term! In all these cases, it is the quantifier subtyping rule that appears as the principal culprit.

The crux of the problem is that the upper bound of the bound variable X in S_2 changes from S_1 in the rule's conclusion to T_1 in the right-hand premise. This "re-bounding" of variables is syntactically rather bizarre; in particular, it invalidates a whole class of arguments based on structural induction on types, where the case for a type variable normally requires applying the induction hypothesis to its upper bound.

The weaker "equal-bounds subtyping rule" from Cardelli and Wegner's original Fun calculus [CW85]

$$(\forall\text{-}\mathsf{Fun}) \qquad \frac{\Gamma \cup \{X{\leq}U\} \vdash S_2 \leq T_2}{\Gamma \vdash \forall(X{\leq}U)S_2 \leq \forall(X{\leq}U)T_2} \qquad X \notin dom(\Gamma)$$

avoids this confusion and yields a system for which the subtyping problem can easily be shown to be decidable, at the cost of introducing an ugly syntactic restriction — that the bounds of the two types must be *identical* — with no natural semantic motivation.

Other variants of ($\forall$-orig) have been proposed in the literature. Katiyar and Shankar [KS92] describe a restriction in which the bounds on polymorphic types may not contain Top. In this way they obtain a decidable subtype relation, but at significant expense in expressiveness. Another obvious variation uses the bound S_1 of the smaller type in place of T_1 in the right-hand premise:

$$(\forall\text{-local}) \qquad \frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \cup \{X{\leq}S_1\} \vdash S_2 \leq T_2}{\Gamma \vdash \forall(X{\leq}S_1)S_2 \leq \forall(X{\leq}T_1)T_2} \qquad X \notin dom(\Gamma)$$

But Giorgio Ghelli has pointed out [Ghe93c] that this variant is algorithmically impractical, although the problem of its decidability remains open.

We propose a simple and appealing alternative: a quantifier subtyping rule

$$(\forall\text{-top}) \qquad \frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \cup \{X{\leq}\mathsf{Top}\} \vdash S_2 \leq T_2}{\Gamma \vdash \forall(X{\leq}S_1)S_2 \leq \forall(X{\leq}T_1)T_2} \qquad X \notin dom(\Gamma)$$

in which the right-hand premise requires that the bodies be (covariantly) related under *no* assumption about the bound variable. This essentially amounts to considering the subtyping relation relative to an unchanging context, since the type variables added to the context always have trivial bounds; the only type variables with interesting bounds will be those already present in the environment at the point where a subtyping check is required. (These are introduced, as usual, by the quantifier introduction rule.)

Clearly, the proposed rule is strictly weaker than (∀-orig). The system with (∀-top) in place of (∀-orig) — we call it $F_{\leq}^{\top}$ and give its full definition in in Section 2 — cannot be used to prove $F_{\leq}$-inequalities like

$$\vdash \forall(X{\leq}S_1)X \leq \forall(X{\leq}S_2)S_2.$$

However, as we we show in Section 3, this difference in power does not matter in any of the situations where bounded quantification has been applied. Moreover, the rule (∀-top) is arguably more natural than the other variants we have mentioned, since it embodies a notion of pointwise subtyping familiar from the treatment of other type constructors [Mit90, BM92, Bru93, PT93]: it simply says that $\forall F \leq \forall G$ (with $F, G : \mathbf{Type}{\rightarrow}\mathbf{Type}$) iff $dom(G) \subseteq dom(F)$ and F is pointwise smaller than G.

$F_{\leq}^{\top}$ enjoys many syntactic properties missing from $F_{\leq}$. First and foremost, its subtyping problem is decidable, as we show in Section 4; moreover, the system has least upper bounds for all pairs of types and, for all lower-bounded pairs, greatest lower bounds.

The easier syntactic formulation also leads to a significant simplification of the proofs for this system. For example, Curien and Ghelli prove the admissibility of the rule of transitivity [CG92] using a rewiting technique similar to Gentzen's cut-elimination theorem, whereas here a simple induction on proofs suffices. In the same way, we obtain a simpler proof of *coherence* of the semantic framework for bounded quantification proposed by Breazu-Tannen, Coquand, Gunter, and Scedrov [BTCGS91], which we sketch in Section 5; indeed, the existence of meets and joins in our subtype relation permits us to prove the coherence of a stronger system including variant types. ([BTCGS91] established a similar result for the variant of $F_{\leq}$ with variant types and the (∀-Fun) rule.)

One of the principal benefits of the simpler formulation of bounded quantification is the ease with which its basic properties may be extended to richer systems. We illustrate this in

Section 6, which shows how a simple kind of recursive types form a conservative extension of $F_{\leq}^{\top}$, another property that fails in $F_{\leq}$ [Ghe93b].

2 Syntax

In what follows, we work modulo α-conversion for type variables, with the convention that bound variables are silently α-converted as necessary so that contexts and types appearing in instances of the rules are well formed. (Equivalently, we follow deBruijn [dB72] in regarding the connection between occurrences of variables and their binders as part of the syntax of the calculus, considering "poorly scoped" terms as not even parseable.) It easy to check that, for example, the rules below preserve the well-formedness of judgments, i.e. the judgments in the premises are well formed if and only if the one in the conclusion is.

$\emptyset$ is a type context with $dom(\emptyset)=\emptyset$. If Γ is a context, $X \notin dom(\Gamma)$, and for every $Y \in FV(T)$, $Y \in dom(\Gamma)$, then $\Gamma \cup \{X \leq T\}$ is a type context and $dom(\Gamma \cup \{X \leq T\}) = dom(\Gamma) \cup \{X\}$. We write $\Gamma(X)$ for the upper bound assigned to X by Γ. (In every situation where we use this notation, it will be clear that Γ includes a binding for X.)

The types of $F_{\leq}^{\top}$ are:

$$T ::= X \mid \mathsf{Top} \mid T \to T \mid \forall(X \leq T)T$$

The syntax of $F_{\leq}$ terms is summarized in Appendix A. Since the change in our definition of the subtype relation does not affect the treatment of terms, we will focus only on types and subtyping in the body of the paper.

We use $\equiv$ to denote syntactic identity.

There are two kinds of judgment: the subtyping relation $\Gamma \vdash T \leq T'$ and the typing relation $\Gamma \vdash a{:}T$. A judgment $\Gamma \vdash \Delta$ is said to be *well-formed* iff, for every type T appearing in Δ, $FV(T) \subseteq dom(\Gamma)$. In the rest of the paper we consider only well-formed judgments.

The $F_{\leq}^{\top}$ subtype relation is the least three-place relation closed under the following rules:

(refl) $$\Gamma \vdash T \leq T$$

(trans) $$\frac{\Gamma \vdash T_1 \leq T_2 \qquad \Gamma \vdash T_2 \leq T_3}{\Gamma \vdash T_1 \leq T_3}$$

(taut) $$\Gamma \vdash X \leq \Gamma(X)$$

(Top) $$\Gamma \vdash T \leq \mathsf{Top}$$

($\to$) $$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \vdash S_2 \leq T_2}{\Gamma \vdash S_1 \to S_2 \leq T_1 \to T_2}$$

$$(\forall) \qquad \frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \cup \{X{\leq}\mathsf{Top}\} \vdash S_2 \leq T_2}{\Gamma \vdash \forall(X{\leq}S_1)S_2 \leq \forall(X{\leq}T_1)T_2} \qquad X \notin dom(\Gamma)$$

(When no ambiguity can arise, we denote the rule ($\forall$-top) simply by ($\forall$).)

3 Expressiveness

There are some values of S and T for which $\Gamma \vdash S \leq T$ is provable in $F_{\leq}$ but not in $F_{\leq}^{\top}$. For instance, the construction used to show the undecidability of $F_{\leq}$ [Pie93] depends crucially on the form of $F_{\leq}$'s original quantifier subtyping rule; similar examples can be built using techniques studied by Ghelli [Ghe93a] in his analysis of the possibility of nontermination in the standard semi-decision procedure for the $F_{\leq}$ subtype relation. But these examples were designed specifically to illustrate pathologies in $F_{\leq}$. Are there any *useful* examples that lie between $F_{\leq}$ and $F_{\leq}^{\top}$? We believe not.

For example, one of the areas where bounded quantification has been most intensively applied is in static type systems for object-oriented languages. One recent study by Pierce and Turner [PT93] presents a statically typed model of the core of Smalltalk using an extension of $F_{\leq}$ with the higher-order polymorphism of Girard's System F^{ω} [Gir72]. All of the terms in this model have exactly the same types in $F_{\leq}^{\top}$ as in $F_{\leq}$. Similarly, all the example programs in Cardelli's implementation of $F_{\leq}$ [Car93] also typecheck in $F_{\leq}^{\top}$, apart from the intentionally pathological cases cited above [Luca Cardelli, personal communication, 1993]

Another fairly complex application of bounded quantification is Cardelli's translation of a calculus of *extensible record* operations into pure $F_{\leq}$ [Car92]. Again, since each of the *row variables* in the high-level calculus is translated into a row of type variables bounded by Top, $F_{\leq}^{\top}$ appears to work just as well.

The difference in expressive power between $F_{\leq}$ and $F_{\leq}^{\top}$ can be understood by observing the behavior of their "standard" algorithms for checking the subtype relation. (For $F_{\leq}$, the standard algorithm [CG92] is a semi-decision procedure; the algorithm for $F_{\leq}^{\top}$ presented in Section 4.1 below is a decision procedure.) The two algorithms are identical at all points, except for the cases for comparing two quantified types, which are just the rules ($\forall$-orig) and ($\forall$-top), respectively. In both cases the left-hand premise is the same; the only difference lies in the recursive call corresponding to the right-hand premise, which extends the environment with a nontrivial bound for the new variable in the case of $F_{\leq}$ and, for $F_{\leq}^{\top}$, with a trivial bound. If the $F_{\leq}$ algorithm later encounters this variable on the left-hand side of the $\leq$, it has two choices: either it can check that the expression on the right of the $\leq$ is the same variable and immediately succeed, or it can *promote* the variable on the left to its upper bound from the context and try to show that this bound is less than the right hand side. The $F_{\leq}^{\top}$ algorithm has only one choice: if the expression on the right of the $\leq$ is not the same variable (or Top), it must fail.

$F_{\leq}^{\top}$ types strictly fewer terms than $F_{\leq}$. On the other hand, $F_{\leq}^{\top}$ and Cardelli and Wegner's original Fun are not strictly comparable. The assertion

$$\vdash \forall(X{\leq}S)X \leq \forall(X{\leq}S)S$$

is true in Fun and false in $F_{\leq}^{\top}$, while

$$\vdash \forall(X{\leq}\,\mathsf{Top})X \leq \forall(X{\leq}S)X$$

is true in $F_{\leq}^{\top}$ and false in Fun. But in practical situations, this difference does not appear to be very important. It is principally the more natural semantic intuition of [$\forall$-top] that leads us to prefer it over [$\forall$-Fun]. There is one intriguing difference, however. Consider the standard enconding of existential quantifier:

$$\exists(X{\leq}S)T = \forall Y.(\forall(X{\leq}S)\ T \rightarrow Y) \rightarrow Y$$

This encoding in $F_{\leq}^{\top}$ give rise to the following derived subtyping rule for existential types:

$$(\exists\text{-top}) \qquad \frac{\Gamma \vdash S_1 \leq T_1 \qquad \Gamma \cup \{X{\leq}\,\mathsf{Top}\} \vdash S_2 \leq T_2}{\Gamma \vdash \exists(X{\leq}S_1)S_2 \leq \exists(X{\leq}T_1)T_2}$$

In Fun, on the other hand, this encoding leads to a derived rule in which S_1 and T_1 are required to be the same.

$$(\exists\text{-}\mathsf{Fun}) \qquad \frac{\Gamma \cup \{X{\leq}U\} \vdash S_2 \leq T_2}{\Gamma \vdash \exists(X{\leq}U)S_2 \leq \exists(X{\leq}U)T_2}$$

It is not too difficult to imagine an application where the former rule would be preferable: intuitively, it corresponds to the observation that one package implementing an abstract data type may be more refined than another either when the first provides more (or more refined) operations than the second, or when the first publically reveals more of the structure of its hidden witness type. This may also have an impact on the use of $F_{\leq}$'s polymorphism as the basis for a language of modules, where the types exported by a module are modeled as existentially quantified type variables.

4 Basic Properties

In this section we explore the fundamental properties of $F_{\leq}^{\top}$. We show the decidability of the subtyping relation, the existence of a least upper bound (lub) for every finite set of types and a greatest lower bound (glb) for every finite and downward-bounded set of types.

4.1 Subtyping algorithm

It is easy to adapt Curien and Ghelli's algorithm [CG92] to $F_{\leq}^{\top}$. Only the rule (Alg$\forall$) is different: here it is identical to ($\forall$-top), whereas in the algorithm for $F_{\leq}$ it coincides with ($\forall$-orig). We use $\vdash_{\mathcal{A}}$ to denote the algorithmic system.

$$(\text{AlgRefl}) \qquad \Gamma \vdash_{\mathcal{A}} X \leq X$$

(AlgTrans) $$\frac{\Gamma \vdash_{\!A} \Gamma(X) \leq T}{\Gamma \vdash_{\!A} X \leq T}$$

(AlgTop) $$\Gamma \vdash_{\!A} T \leq \mathsf{Top}$$

(Alg$\rightarrow$) $$\frac{\Gamma \vdash_{\!A} T_1 \leq S_1 \qquad \Gamma \vdash_{\!A} S_2 \leq T_2}{\Gamma \vdash_{\!A} S_1 \rightarrow S_2 \leq T_1 \rightarrow T_2}$$

(Alg$\forall$) $$\frac{\Gamma \vdash_{\!A} T_1 \leq S_1 \qquad \Gamma \cup \{X{\leq}\mathsf{Top}\} \vdash_{\!A} S_2 \leq T_2}{\Gamma \vdash_{\!A} \forall(X{\leq}S_1)S_2 \leq \forall(X{\leq}T_1)T_2} \qquad X \notin dom(\Gamma)$$

Note that the algorithm defined by these rules is deterministic, since the form of the input unequivocally determines the rule that must be used — not true in the original system because of (refl) and (trans) — and all the parameters of any recursive calls — not true in the original system because of (trans).

4.1.1 Soundness and completeness

We now show that every judgment provable in the algorithmic system $\vdash_{\!A}$ is also provable in the original system $\vdash$ (soundness of the algorithm) and that every relation provable by $\vdash$ is provable also by $\vdash_{\!A}$ (completeness of the algorithm). Finally, we show that the algorithm actually halts on every input and hence constitutes a decision procedure.

Lemma 4.1 *Let $\Gamma \vdash_{\!A} S \leq T$. Then:*

1. *If $S = \mathsf{Top}$, then $T = \mathsf{Top}$.*
2. *If S is a variable X, then either $T = X$ or $T = \mathsf{Top}$, or else $\Gamma \vdash_{\!A} \Gamma(X) \leq T$.*
3. *If $S = S_1 \rightarrow S_2$, then either $T = \mathsf{Top}$ or else $T = T_1 \rightarrow T_2$ with $\Gamma \vdash_{\!A} T_1 \leq S_1$ and $\Gamma \vdash_{\!A} S_2 \leq T_2$.*
4. *If $S = \forall(X{\leq}S_1)S_2$, then either $T = \mathsf{Top}$ or else $T = \forall(X{\leq}T_1)T_2$ with $\Gamma \vdash_{\!A} T_1 \leq S_1$ and $\Gamma \cup \{X{\leq}\mathsf{Top}\} \vdash_{\!A} S_2 \leq T_2$.*

Moreover, the implied subtyping judgements are proved by subderivations of the originals.

Proof. By inspection. *End of proof.*

To prove the completeness of the algorithm we need two simple lemmas, which show that the relation defined by $\vdash_{\!A}$ is reflexive (lemma 4.2) and transitive (lemma 4.3); in other words the two rules that belong to the definition of $\vdash$ and not $\vdash_{\!A}$ are derived rules in $\vdash_{\!A}$.

Lemma 4.2 *For each context Γ and type T (well-formed in Γ), the judgment $\Gamma \vdash_{\!A} T \leq T$ is provable.*

Proof. By induction on the structure of T.

$T{\equiv}X$ immediate from (AlgRefl).

$T \equiv$ **Top** immediate from (AlgTop).

$T \equiv S_1 \to S_2$ then by the induction hypothesis $\Gamma \vdash_{\mathcal{A}} S_1 \leq S_1$ and $\Gamma \vdash_{\mathcal{A}} S_2 \leq S_2$. The result follows by (Alg$\to$).

$T \equiv \forall(X \leq S_1)S_2$ then by the induction hypothesis $\Gamma \vdash_{\mathcal{A}} S_1 \leq S_1$ and $\Gamma \cup \{X \leq \mathsf{Top}\} \vdash_{\mathcal{A}} S_2 \leq S_2$. The result follows by (Alg$\forall$).

End of proof.

Lemma 4.3 *For each context Γ and well-formed types S_1, S_2, and S_3, if $\Gamma \vdash_{\mathcal{A}} S_1 \leq S_2$ and $\Gamma \vdash_{\mathcal{A}} S_2 \leq S_3$, then $\Gamma \vdash_{\mathcal{A}} S_1 \leq S_3$.*

Proof. If $S_3 \equiv \mathsf{Top}$, then the result follows from (AlgTop). Thus we can restrict our attention to those cases in which $S_3 \not\equiv \mathsf{Top}$; note that by lemma 4.1 S_2 must also be different from Top.

Let d_i denote the depth of the proof of $\Gamma \vdash_{\mathcal{A}} S_i \leq S_{i+1}$ for $i = 1, 2$. We proceed by induction on $(d_1 + d_2)$, with a case analysis on the last rule of the proof of $\Gamma \vdash_{\mathcal{A}} S_1 \leq S_2$

(AlgRefl) $\Gamma \vdash_{\mathcal{A}} S_1 \equiv S_2 \leq S_3$ is satisfied by hypothesis.

(AlgTrans) $S_1 \equiv X$. By applying the induction hypothesis to the first main premise's second premise and the second main premise, we obtain that $\Gamma \vdash_{\mathcal{A}} \Gamma(X) \leq S_3$; the result follows by (AlgTrans).

(AlgTop) Cannot occur, since we assumed $S_2 \not\equiv \mathsf{Top}$.

(Alg$\to$) $S_1 \equiv S_1 \to S_1'$ and $S_2 \equiv S_2 \to S_2'$, and, since $S_3 \not\equiv \mathsf{Top}$, by lemma 4.1 $S_3 \equiv S_3 \to S_3'$. By the induction hypothesis, $\Gamma \vdash_{\mathcal{A}} S_3 \leq S_1$ and $\Gamma \vdash_{\mathcal{A}} S_1' \leq S_3'$; by (Alg$\to$) we obtain the result.

(Alg$\forall$) Similarly, we have by assumption that $S_1 \equiv \forall(X \leq S_1)S_1'$ and $S_2 \equiv \forall(X \leq S_2)S_2'$, and, since $S_3 \not\equiv \mathsf{Top}$ by lemma 4.1 $S_3 \equiv \forall(X \leq S_3)S_3'$. By the induction hypothesis, $\Gamma \vdash_{\mathcal{A}} S_3 \leq S_1$ and $\Gamma \cup \{X \leq \mathsf{Top}\} \vdash_{\mathcal{A}} S_1' \leq S_3'$; the result follows by (Alg$\forall$).

Note that this case causes significant difficulty in the proofs of Curien and Ghelli [CG92] and Breazu-Tannen et al. [BTCGS91].

End of proof.

Theorem 4.4 (Soundness and completeness) *For each context Γ and well-formed types S_1 and S_2,*

$$\Gamma \vdash_{\mathcal{A}} S_1 \leq S_2 \iff \Gamma \vdash S_1 \leq S_2.$$

Proof. We start by proving soundness ($\Rightarrow$) by induction on the depth of the proof of $\Gamma \vdash_{\mathcal{A}} S_1 \leq S_2$. At each stage of the induction, we proceed by cases on the final rule used in the proof. The cases for (AlgRefl) and (AlgTop) are immediate; (Alg$\to$) and (Alg$\forall$) follow by straightforward use of the induction hypothesis. For (AlgTrans), we have $S_1 \equiv X$. By the induction hypothesis, $\Gamma \vdash \Gamma(X) \leq S_2$; by (tvar) $\Gamma \vdash X \leq \Gamma(X)$; thus by (trans) we obtain the result.

Completeness ($\Leftarrow$) follows from lemmas 4.2 and 4.3. *End of proof.*

This establishes that the systems $\vdash$ and $\vdash_{\mathcal{A}}$ are equivalent and we can eliminate the $\mathcal{A}$ indexing the turnstyle symbol. In particular this implies that the lemma 4.1 is also valid for $\vdash$.

Note that in a proof of subtyping we are now allowed to mix rules of the two systems since, by induction, for every subproof there exists an equivalent proof using only rules of the same system. Thus in the proofs of the theorems that follow we will freely use rules from both systems. In particular we will refer to algorithmic or full transitivity, according to our needs.

4.1.2 Termination

We first define the *weight* of a type T with respect to a type context Γ (with T well formed in Γ):

$$\begin{aligned}
\mathcal{T}(\mathsf{Top})_\Gamma &= 1 \\
\mathcal{T}(X)_\Gamma &= \mathcal{T}(\Gamma(X))_\Gamma + 1 \\
\mathcal{T}(S_1 \to S_2)_\Gamma &= \mathcal{T}(S_1)_\Gamma + \mathcal{T}(S_2)_\Gamma \\
\mathcal{T}(\forall(X{\leq}S_1)S_2)_\Gamma &= \mathcal{T}(S_1)_\Gamma + \mathcal{T}(S_2)_{\Gamma\cup\{X\leq S_1\}}
\end{aligned}$$

In the final case, the variable X is added to the context with the bound S_1 instead of Top so that it will have the same bound on the left as on the right. This is used in the following proof to define an ordering that depends on the bounds of a variable. Katiyar and Shankar [KS92] give a similar proof of termination for their variant of $F_{\leq}$.

Lemma 4.5 *For each type T well formed in a context Γ, the weight $\mathcal{T}(T)_\Gamma$ is finite and positive.*

Proof. First, it is obvious that the weight $\mathcal{T}(T)_\Gamma$ is always positive. Now, we give a well founded rank for $\mathcal{T}(T)_\Gamma$ (i.e. we define a rank for the definition of the weight itself) and we show that it decreases at each stage in the definition of $\mathcal{T}$. To define the rank of $\mathcal{T}(T)_\Gamma$ consider all the variables that appear in T and Γ (no matter whether they appear free or bound, only in a quantifier or in a bound). Since T is well formed in Γ, every variable is associated to a unique bound (either in Γ or in T); furthermore it is also possible to totally order these variables in a way that if X_i is defined in the bound of X_j then X_i precedes X_j (since T is well formed in Γ, loops are not possible). If there is more than one order satisfying this condition then choose one arbitrarily. Define the *depth* of each variable as the number of variables that precede it in this order. Then the rank of $\mathcal{T}(T)_\Gamma$ is the lexicographical size of the pair (D, L), where D is the maximum depth of any of the variables that appear in T, and L is the textual length of T. This rank is well founded (the least element is $(0,1)$). It easy to see that for the subproblem on the right-hand side of $\mathcal{T}(S_1 \to S_2)_\Gamma$ and $\mathcal{T}(\forall(X{\leq}S_1)S_2)_\Gamma$, the component D either is the same or it decreases, while the L component always strictly decreases; for the case $\mathcal{T}(X)_\Gamma$, the component D strictly decreases. *End of proof.*

The weight function is extended to type judgments in the obvious way:

$$\mathcal{J}(\Gamma \vdash S_1 \leq S_2) = \mathcal{T}(S_1)_\Gamma + \mathcal{T}(S_2)_\Gamma.$$

Now we can show the termination of the algorithm.

Lemma 4.6 *For all types S and T well formed in Γ, $\mathcal{T}(T)_{\Gamma\cup\{X\leq \mathit{Top}\}} \leq \mathcal{T}(T)_{\Gamma\cup\{X\leq S\}}$*

Proof. By the previous lemma we are now allowed to prove this lemma by induction on $\mathcal{T}(T)_{\Gamma\cup\{X\leq \mathit{Top}\}} + \mathcal{T}(T)_{\Gamma\cup\{X\leq S\}}$. The proof is straightforward. *End of proof.*

Theorem 4.7 *At every step of the subtyping algorithm, the weight of each of the premises is strictly smaller than the weight of the conclusion.*

Proof. The verification is easy in most cases. The only non-trivial case is (Alg$\forall$), which is proved by the following inequalities:

$$\begin{array}{rcl}
\mathcal{J}(C\cup\{X\leq \mathit{Top}\}\vdash S_2\leq T_2) & = & \mathcal{T}(S_2)_{C\cup\{X\leq \mathit{Top}\}} + \mathcal{T}(T_2)_{C\cup\{X\leq \mathit{Top}\}} \\
& \leq & \mathcal{T}(S_2)_{C\cup\{X\leq S_1\}} + \mathcal{T}(T_2)_{C\cup\{X\leq T_1\}} \qquad \text{by lemma 4.6} \\
& < & \mathcal{T}(S_1)_C + \mathcal{T}(T_1)_C + \mathcal{T}(S_2)_{C\cup\{X\leq S_1\}} + \mathcal{T}(T_2)_{C\cup\{X\leq T_1\}} \\
& = & \mathcal{T}(\forall(X\leq S_1)S_2)_C + \mathcal{T}(\forall(X\leq T_1)T_2)_C \\
& = & \mathcal{J}(C\vdash \forall(X\leq S_1)S_2 \leq \forall(X\leq T_1)T_2).
\end{array}$$

End of proof.

Corollary 4.8 *The algorithm terminates.*

4.2 Meets and joins

Decidability of subtyping is clearly a desirable property, but in practice the undecidability of $F_\leq$ has been much less problematic than the nonexistence of least upper bounds and greatest lower bounds in the subtype relation. Ghelli [Ghe90] observed that, in $F_\leq$, the types

$$S \equiv \forall(Z\leq X\to Y)\ X\to Y \qquad\qquad T\equiv \forall(Z\leq X'\to Y')\ X'\to Y'$$

in the context

$$\Gamma \equiv \{X\leq \mathsf{Top},\ Y\leq \mathsf{Top},\ X'\leq X,\ Y'\leq Y\}$$

have two lower bounds — namely $U\equiv\forall(Z\leq X'\to Y)\ X\to Y'$ and $V\equiv\forall(Z\leq X'\to Y)\ Z$ — but that there is no greatest lower bound of S and T, since U and V have no common supertype that is also a subtype of S and T. From this, it is easy to show that $F_\leq$ also lacks lubs. For example, $S\to\mathsf{Top}$ and $T\to\mathsf{Top}$ have no lub.

Here, by contrast, a simple induction on the weight defined in the provious section shows that every pair of types in $F_\leq^\top$ *does* possess a least upper bound, and that every pair of types with any lower bound at all possesses a greatest lower bound.

Theorem 4.9 *Let S and T be well-formed types in a context Γ.*

(a) There is some U such that $\Gamma\vdash S,T\leq U$ and such that $\Gamma\vdash U\leq U'$ whenever $\Gamma\vdash S,T\leq U'$.

(b) If there is any V such that $\Gamma\vdash V\leq S,T$, then there is some L such that $\Gamma\vdash L\leq S,T$ and such that $\Gamma\vdash L'\leq L$ whenever $\Gamma\vdash L'\leq S,T$.

Proof. In this proof we consider derivations in $\vdash_{\mathcal{A}}$. We prove both cases simultaneously by induction on $\mathcal{T}(S)_\Gamma + \mathcal{T}(T)_\Gamma$.

First, note that the theorem is immediate when one of the two types is Top: the lub is Top and the glb is the other type. Also trivial is the case when $S\equiv T$. Thus, we may assume that T and S are two distinct types both different from Top, and consider the remaining cases:

1. Both S and T are variables. For part (a) we have two cases: either the two variables are related, and in that case the greater of them is the lub; or the two variables are unrelated and we can apply the induction hypothesis to $\Gamma(S)$ and $\Gamma(T)$, obtaining that there exists a type U which is their lub. Now, every upper bound of $\Gamma(S)$ is an upper bound of S too, and every *proper* upper bound of S is an upper bound for $\Gamma(S)$. Thus the set of proper upper bounds of S coincides with the set of upper bounds for $\Gamma(S)$; the same holds for T. Since the two variables are unrelated the set of their common upper bounds coincides with the set of their common proper upper bounds, which is equal to the set of common upper bounds of $\Gamma(S)$ and $\Gamma(T)$.

   For part (b), simply note that by lemma 4.1 if S and T have a common lower bound Z then it is a variable. If a variable is provably smaller than another variable then the proof consists only of rules (AlgTrans) (this can be obtained by a trivial induction on the depth of the derivation). Consider the proofs of $\Gamma \vdash Z \leq S, T$. Since they both start from Z and always use the same rule (AlgTrans), which is deterministic, then the shorther proof must appear as a subproof of the longer one; thus using full transitivity either $\Gamma \vdash S \leq T$ or $\Gamma \vdash T \leq S$ holds. The lesser is the glb.

2. Both S and T are arrow types: $S \equiv S_1 \to S_2$ and $T \equiv T_1 \to T_2$.

   For part (a), if S_1 and T_1 have a common lower bound then by the induction hypothesis there exists $S_1 \sqcap T_1$, the glb of S_1 and T_1 with respect to Γ. Then it is trivial to show that the lub of S, T is $S_1 \sqcap T_1 \to S_2 \sqcup T_2$ where $S_2 \sqcup T_2$ is the lub of S_2 and T_2 with respect to Γ (which exists by the induction hypothesis). If S_1, T_1 have no common lower bound then it is clear that the lub of T and S is Top.

   For part (b), suppose that there exists V satisfying the hypothesis of the lemma. Then by lemma 4.1, either $V \equiv V_1 \to V_2$ or V is a variable. If $V \equiv V_1 \to V_2$ then $\Gamma \vdash V_2 \leq S_2, T_2$; thus by the induction hypothesis there exists $S_2 \sqcap T_2$, the glb of S_2 and T_2 with respect to Γ. As in part (a) it is then easy to show that $S_1 \sqcup T_1 \to S_2 \sqcap T_2$ is the glb of T, S where $S_1 \sqcup T_1$ is the lub of S_1 and T_1 with respect to Γ, which exists by the induction hypothesis.

   If V is a variable then consider the following definition of $\mathcal{B}(T)_\Gamma$:
   1. $\mathcal{B}(T)_\Gamma = T$ if T is not a type variable.
   2. $\mathcal{B}(X)_\Gamma = \mathcal{B}(\Gamma(X))_\Gamma$.

   Clearly, $\Gamma \vdash V \leq \mathcal{B}(V)_\Gamma$, and, by induction on the size of this derivation, $\Gamma \vdash \mathcal{B}(V)_\Gamma \leq S, T$. By lemma 4.1 $\mathcal{B}(V)_\Gamma \equiv V_1 \to V_2$. Thus it suffices to apply the induction hypothesis as above, using $\mathcal{B}(V)_\Gamma$ in place of V, to obtain the result.

3. Both S and T are universally quantified types. This case is similar to the previous case, but since this is where the induction hypothesis fails if we try to use the same argument for $F_{\leq}$, let us examine the critical passage in more detail. Consider part (b) of the statement when V is not a variable; let $S \equiv \forall(X \leq S_1)S_2$, $T \equiv \forall(X \leq S_1)S_2$, and $V \equiv \forall(X \leq V_1)V_2$. Then by rule ($\forall$) we have

$$\begin{array}{l} \Gamma \vdash S_1, T_1 \leq V_1 \\ \Gamma \cup \{X \leq \mathsf{Top}\} \vdash V_2 \leq S_2, T_2. \end{array}$$

By the definition of $\mathcal{T}$ we can apply the induction hypothesis on both judgments, obtaining L_1 and L_2 such that

$$\begin{array}{l} \Gamma \vdash S_1, T_1 \leq L_1 \\ \Gamma \cup \{X \leq \mathsf{Top}\} \vdash L_2 \leq S_2, T_2, \end{array}$$

and such that $\Gamma \vdash W_1 \leq L_1$ and $\Gamma \cup \{X \leq \mathsf{Top}\} \vdash W_2 \leq L_2$ whenever

$$\begin{array}{l} \Gamma \vdash S_1, S_1 \leq W_1 \\ \Gamma \cup \{X \leq \mathsf{Top}\} \vdash W_2 \leq S_2, T_2. \end{array}$$

It is then clear that $\forall(X \leq L_1)L_2$ is the glb.

4. One of the two types, say S is a type variable and the other not. Then for part (a) we can apply the induction hypothesis to $\Gamma(S)$ and T. For part (b) we must have $\Gamma \vdash S \leq T$ (by the same observation as in case 1).

End of proof.

5 Semantics

Up to now we have dealt only with types and subtyping. In this section we introduce terms and three new type constructors. We provide a sound semantics for the obtained language and a coherence result for the corresponding proof system.

This section is largely based on [BTCGS91], where the reader can find a more detailed presentation of the technique we use here. In [BTCGS91], a semantics for $F_{\leq}$ plus record and recursive types is given by translating the system into an extension of System F for which sound semantic interpretations are already defined. The soundness of the method is given by a coherence result. This proof fails if variant types are also added to the system (this is essentially due to the absence of meets and joins in $F_{\leq}$). Our system behaves better since the technique of [BTCGS91] works for our system even if we extend it by variant types, as we show in this section.

5.1 Extended $F_{\leq}^{\top}$

We enrich our set of types by record types ($\langle\!\langle \ell_1 : T_1, \ldots, \ell_p : T_p \rangle\!\rangle$), variant types ($[\ell_1 : T_1, ..., \ell_p : T_p]$) and recursive types ($\mu X.T$). We extend the subtyping relation by adding just the following rules.

(recd)
$$\frac{\Gamma \vdash S_1 \leq T_1 \ldots \Gamma \vdash S_p \leq T_p}{\Gamma \vdash \langle\!\langle \ell_1 \colon S_1, ..., \ell_p \colon S_p, ..., \ell_q \colon S_q \rangle\!\rangle \leq \langle\!\langle \ell_1 \colon T_1, \ldots, \ell_p \colon T_p \rangle\!\rangle}$$

(vart)
$$\frac{\Gamma \vdash S_1 \leq T_1 \ldots \Gamma \vdash S_p \leq T_p}{\Gamma \vdash [\ell_1 \colon S_1, \ldots, \ell_p \colon S_p] \leq [\ell_1 \colon T_1, \ldots, \ell_p \colon T_p, \ldots \ell_q \colon T_q]}$$

Of course, this yields a rather naive treatment of recursive types since only reflexivity can be used to compare them; for that reason all the results of the previous section can easily be extended to this system. We consider a more realistic formulation of recursion in the next section.

The raw terms of the language are:

$$\begin{array}{rcl} a & ::= & x \mid (\lambda x^T.a) \mid a(a) \\ & \mid & \lambda X{\leq}T.a \mid a(T) \mid \mathsf{top} \\ & \mid & \langle \ell_1 = a_1, \ldots, \ell_n = a_n \rangle \mid a.\ell \\ & \mid & [\ell_1 \colon T_1, \ldots, \ell_i = a, \ldots, \ell_n \colon T_n] \\ & \mid & \textbf{case } a \textbf{ of } \ell_1 \Rightarrow a_1, \ldots, \ell_n \Rightarrow a_n \\ & \mid & \textbf{intro}_{\mu X.T}(a) \mid \textbf{elim}(a) \end{array}$$

The typing rules for this language are exactly the same as those of $F_{\leq}$ (see Appendix A.1). plus the rules for the new terms:

[$\langle\!\langle\rangle\!\rangle$INTRO]
$$\frac{\Gamma \vdash a_1 \colon T_1 \ldots \Gamma \vdash a_p \colon T_p}{\Gamma \vdash \langle \ell_1 = a_1, \ldots, \ell_p = a_p \rangle \colon \langle\!\langle \ell_1 \colon T_1, \ldots, \ell_p \colon T_p \rangle\!\rangle}$$

[$\langle\!\langle\rangle\!\rangle$ELIM]
$$\frac{\Gamma \vdash a \colon \langle\!\langle \ell_1 \colon T_1, \ldots, \ell_p \colon T_p \rangle\!\rangle}{\Gamma \vdash a.\ell_i \colon T_i}$$

[[]INTRO]
$$\frac{\Gamma \vdash a \colon T_i}{\Gamma \vdash [\ell_1 \colon T_1, \ldots, \ell_i = a_i, \ldots \ell_p \colon T_p] \colon [\ell_1 \colon T_1, \ldots, \ell_i \colon T_i, \ldots \ell_p \colon T_p]}$$

[[]ELIM]
$$\frac{\Gamma \vdash b \colon [\ell_1 \colon T_1, \ldots, \ell_p \colon T_p] \qquad \Gamma \vdash a_1 \colon T_1 \to T \quad \ldots \quad \Gamma \vdash a_p \colon T_p \to T}{\Gamma \vdash \textbf{case } b \textbf{ of } \ell_1 \Rightarrow a_1, \ldots, \ell_p \Rightarrow a_p \colon T}$$

[μINTRO]
$$\frac{\Gamma \vdash a \colon T[X := \mu X.T]}{\Gamma \vdash \textbf{intro}_{\mu X.T}(a) \colon \mu X.T}$$

[μELIM]
$$\frac{\Gamma \vdash a \colon \mu X.T}{\Gamma \vdash \textbf{elim}(a) \colon [X := \mu X.T]}$$

To give a semantics to this language we consider a language **TARGET**, for which many semantic interpretations already exist. We translate the derivations of judgments of our language into derivations of judgments of **TARGET**. Then we give the semantics via this translation: the intepretation of a typing judgment in our source language is the interpretation of its translation. But this leads to a problem of coherence since in the source language there may be may different derivations of a single typing judgment, each with a different translation. Thus we end the section with a theorem proving that the typing judgments obtained as translations of different derivations for a same judgment are provably equal in **TARGET** and thus they have the same semantic interpretation. In this way we obtain the coherence of our semantic interpretation.

5.2 The language TARGET

The target language is an extension of Girard's System F [Gir72] with recursive types, variants, records, and coercion spaces. Coercion spaces are used to interpret subtyping judgments (for the reader acquaitened with the models of System F, coercion spaces are interpreted as strict maps in the model of dI-domains and as linear maps in coherent spaces). The types of **TARGET** are:

$$
\begin{array}{lcll}
C & ::= & T \multimap T & \text{coercion spaces} \\
\\
T & ::= & X & \\
 & | & T \to T & \\
 & | & C \to C & \\
 & | & C \to T & \\
 & | & \forall X.T & \\
 & | & \langle\!\langle \ell_1 : T_1, \ldots, \ell_n : T_n \rangle\!\rangle & \\
 & | & [\ell_1 : T_1, \ldots, \ell_n : T_n] & \\
 & | & \mu X.T &
\end{array}
$$

The key idea of the translation is to consider the rule of subsumption as the use of an implicit coercion; the translation makes this coercion explicit. The translation of a term $\lambda X \leq T.a$ takes as parameters both a type X and a coercion function from X to T; this coercion will be explicitly used in the translation of a to replace the implicit use of subsumption in a. More precisely, we give a translation $(\text{—})^*$ that translates the type $\forall(X \leq S)T$ into $\forall X.(X \multimap S^*) \to T^*$.

The terms of **TARGET** are:

$$
\begin{array}{lcl}
a & ::= & x^T \mid (\lambda x^T.a) \mid a(a) \\
 & | & \lambda X.a \mid a(T) \\
 & | & \langle \ell_1 = a_1, \ldots, \ell_n = a_n \rangle \mid a.\ell \\
 & | & [\ell_1 : T_1, \ldots, \ell_i = a, \ldots, \ell_n : T_n] \\
 & | & \textbf{case } a \textbf{ of } \ell_1 \Rightarrow a_1, \ldots, \ell_n \Rightarrow a_n \\
 & | & \textbf{intro}_{\mu X.T}(a) \mid \textbf{elim}(a)
\end{array}
$$

A subtyping judgment $S \leq T$ is translated as a coercion in $S^* \multimap T^*$. We introduce a set of constants $\iota_{S,T}$ that transform a coercion into a function (in order to apply it), and several *coercion combinators*, which we use to translate subtyping judgments; we use 1 to denote the empty record type $\langle\!\langle\,\rangle\!\rangle$:

$$\iota_{S,T} \colon (S \hookrightarrow T) \to (S \to T)$$

$$\begin{aligned}
&\mathsf{top}[T] \colon T \hookrightarrow 1\\
&\mathsf{refl}[T] \colon T \hookrightarrow T\\
&\mathsf{trans}[S,T,U] \colon (S \hookrightarrow T) \to (T \hookrightarrow U) \to (S \hookrightarrow U)\\
&\mathsf{arrow}[S,T,U,V] \colon (S \hookrightarrow T) \to (U \hookrightarrow V) \to ((T \to U) \hookrightarrow (S \to V))\\
&\mathsf{forall}[S,T,U,V] \colon (S \hookrightarrow T) \to (\forall X((X \hookrightarrow 1) \to (U \hookrightarrow V))\\
&\qquad\qquad \to \forall X((X \hookrightarrow T) \to U) \hookrightarrow \forall X((X \hookrightarrow S) \to V)\\
&\mathsf{recd}[S_1,\ldots,S_q,T_1,\ldots T_p] \colon (S_1 \hookrightarrow T_1) \to \ldots \to (S_p \hookrightarrow T_p)\\
&\qquad\qquad \to (\langle\!\langle \ell_1 \colon S_1,\ldots,\ell_p \colon S_p,\ldots,\ell_q \colon S_q\rangle\!\rangle \hookrightarrow \langle\!\langle \ell_1 \colon T_1,\ldots;\ell_p \colon T_p\rangle\!\rangle)\\
&\mathsf{vart}[S_1,\ldots,S_p,T_1,\ldots T_q] \colon (S_1 \hookrightarrow T_1) \to \ldots \to (S_p \hookrightarrow T_p)\\
&\qquad\qquad \to ([\ell_1 \colon S_1,\ldots,\ell_p \colon S_p] \hookrightarrow [\ell_1 \colon T_1,\ldots,\ell_p \colon T_p,\ldots \ell_q \colon T_q])
\end{aligned}$$

For example, $\mathsf{arrow}[S,T,U,V]$ is the coercion combinator that takes coercions from S to T and from U to V and returns the corresponding coercion from $T \to U$ to $S \to V$. The precise behavior of each combinator is determined by ι, i.e. by its transformation into a function (we omit types annotations when they are clear from the context):

$$\begin{aligned}
\iota(\mathsf{top}) &= \lambda x^T.\langle\,\rangle\\
\iota(\mathsf{refl}) &= \lambda x^T.x\\
\iota(\mathsf{trans}(a)(b)) &= \iota(a);\iota(b) && \text{where } a \colon R \hookrightarrow S, b \colon S \hookrightarrow T\\
\iota(\mathsf{arrow}(a)(b)) &= \lambda z^{T \to U}.\iota(a);z;\iota(b) && \text{where } a \colon S \hookrightarrow T, b \colon U \hookrightarrow V\\
\iota(\mathsf{forall}(a)(b)) &= \lambda z^{\forall X(X \hookrightarrow T) \to U}.\lambda X.\lambda x^{X \hookrightarrow S}.\iota(b(X)(\mathsf{top}[X]))(z(X)(\mathsf{trans}(x)(a)))\\
& && \text{where } a \colon S \hookrightarrow T,\ b \colon \forall X.(X \hookrightarrow 1) \to (U \hookrightarrow V)\\
\iota(\mathsf{recd}(a_1)\cdots(a_p)) &= \lambda z^{\langle\!\langle \ell_1:S_1,\ldots;\ell_p:S_p,\ldots;\ell_q:S_q\rangle\!\rangle}.\langle\!\langle \ell_1 = \iota(a_1)(z.\ell_1),\ldots;\ell_p = \iota(a_p)(z.\ell_p)\rangle\!\rangle\\
& && \text{where } a_1 \colon S_1 \hookrightarrow T_1,\ldots,a_p \colon S_p \hookrightarrow T_p\\
\iota(\mathsf{vart}(a_1)\cdots(a_p)) &= \lambda z^{[\ell_1:S_1,\ldots;\ell_p:S_p]}.\mathbf{case}\ z\ \mathbf{of}\ \ell_1 \Rightarrow \iota(a_1);\mathbf{inj}_{\ell_1},\ldots;\ell_p = \iota(a_p);\mathbf{inj}_{\ell_p}\\
& && \text{where } a_1 \colon S_1 \hookrightarrow T_1,\ldots,a_p \colon S_p \hookrightarrow T_p \text{ and}\\
& && \mathbf{inj}_{\ell_i} = \lambda x^{T_i}[\ell_1 \colon T_1,\ldots,\ell_i = a,\ldots,\ell_q \colon T_q]
\end{aligned}$$

Note that, unlike the translation in [BTCGS91], the term interpreting forall is linear in the parameter x.

The interpretation via ι of coercions as functions must completely determine their identity as coercions. This is expressed by the following equality rule in **TARGET**:

$$\langle \iota\text{-INJ}\rangle \qquad \frac{\iota(a) = \iota(b)}{a = b}$$

The complete equational theory of **TARGET** includes this rule plus the classical of rules of F of records, variant and recursive types ($\langle\beta\rangle$, $\langle\beta_2\rangle$, $\langle\eta\rangle$, $\langle\eta_2\rangle$, $\langle\xi\rangle$, $\langle\xi_2\rangle$, $\langle\beta_{recd}\rangle$, $\langle\eta_{recd}\rangle$, $\langle\beta_\mu\rangle$, $\langle\eta_\mu\rangle$), plus a special rule for variant types to handle coercion terms:

$$\begin{aligned}
\langle\mathrm{CRN}\rangle \qquad & \iota(P)(\mathbf{case}\ a\ \mathbf{of}\ \ell_1 \Rightarrow a_1,\ldots,\ell_n \Rightarrow a_n)\\
&= (\mathbf{case}\ a\ \mathbf{of}\ \ell_1 \Rightarrow a_1;\iota(P),\ldots,\ell_n \Rightarrow a_n;\iota(P))
\end{aligned}$$

The precise definitions of all the rules can be found in the Appendix B of [BTCGS91], which also discusses the motivation for the rule $\langle\mathrm{CRN}\rangle$.

5.3 Translation

We are now ready to give the formal translation of derivations of our source language. We start with the translation of the types and the contexts:

$$
\begin{array}{lcl}
X^* & = & X \\
\mathsf{Top}^* & = & 1 \\
(S \to T)^* & = & S^* \to T^* \\
(\forall(X{\leq}S)T)^* & = & \forall X.((X \multimap S^*) \to T^*) \\
\langle\!\langle \ell_1{:}\,T_1, \ldots \ell_n{:}\,T_n \rangle\!\rangle^* & = & \langle\!\langle \ell_1{:}\,T_1^*, \ldots \ell_n{:}\,T_n^* \rangle\!\rangle \\
([\ell_1{:}\,T_1, \ldots \ell_n{:}\,T_n])^* & = & [\ell_1{:}\,T_1^*, \ldots \ell_n{:}\,T_n^*] \\
(\mu X.T)^* & = & \mu X.T^* \\
 & & \\
\emptyset^* & = & \emptyset \\
(\Gamma \cup \{X{\leq}T\})^* & = & \Gamma^* \cup X \cup \{x{:}\,X \multimap T^*\} \\
(\Gamma \cup \{x{:}\,T\})^* & = & \Gamma^* \cup \{x{:}\,T^*\}
\end{array}
$$

The translation of the subtyping rules is:

$$(\text{refl})^* \qquad \Gamma^* \vdash \mathsf{refl}{:}\,T^* \multimap T^*$$

$$(\text{trans})^* \qquad \frac{\Gamma^* \vdash a : T_1^* \multimap T_2^* \qquad \Gamma^* \vdash b : T_2^* \multimap T_3^*}{\Gamma^* \vdash \mathsf{trans}(a)(b) : T_1^* \multimap T_3^*}$$

$$(\text{tvar})^* \qquad \Gamma^* \cup \{X\} \cup \{x{:}\,X \multimap \Gamma(X)^*\} \vdash x{:}\,X \multimap \Gamma(X)^*$$

$$(\text{Top}) \qquad \Gamma \vdash \mathsf{top}[T^*]{:}\,T^* \multimap 1$$

$$(\to)^* \qquad \frac{\Gamma^* \vdash a{:}\,T_1^* \multimap S_1^* \qquad \Gamma^* \vdash b{:}\,S_2^* \multimap T_2^*}{\Gamma^* \vdash \mathsf{arrow}(a)(b){:}\,(S_1^* \to S_2^*) \multimap (T_1^* \to T_2^*)}$$

$$(\forall)^* \qquad \frac{\Gamma^* \vdash a{:}\,T_1^* \multimap S_1^* \qquad \Gamma^* \cup \{X\} \cup \{x{:}\,X \multimap 1\} \vdash b{:}\,S_2^* \multimap T_2^*}{\Gamma^* \vdash \mathsf{forall}(a)(\lambda X.\lambda x^{X \multimap 1}.b){:}\,\forall X((X \multimap S_1^*) \to S_2^*) \multimap \forall X((X \multimap T_1^*) \to T_2^*)}$$

$$(\text{recd})^* \qquad \frac{\Gamma^* \vdash a_1{:}\,S_1^* \multimap T_1^* \ldots \Gamma^* \vdash a_p{:}\,S_p^* \multimap T_p^*}{\Gamma^* \vdash \mathsf{recd}(a_1)\cdots(a_p){:}\,\langle\!\langle \ell_1{:}\,S_1^*, \ldots; \ell_p{:}\,S_p^*, \ldots; \ell_q{:}\,S_q^* \rangle\!\rangle \multimap \langle\!\langle \ell_1{:}\,T_1^*, \ldots; \ell_p{:}\,T_p^* \rangle\!\rangle}$$

$$(\text{vart})^* \qquad \frac{\Gamma^* \vdash a_1{:}\,S_1^* \multimap T_1^* \ldots \Gamma^* \vdash p{:}\,S_p^* \multimap T_p^*}{\Gamma^* \vdash \mathsf{vart}(a_1)\cdots(a_p){:}\,[\ell_1{:}\,S_1^*, \ldots; \ell_p{:}\,S_p^*] \multimap [\ell_1{:}\,T_1^*, \ldots; \ell_p{:}\,T_p^*, \ldots \ell_q{:}\,T_q^*]}$$

Most of the typing rules are translated in a trivial way. We show just three of them to give an example. The complete translation can be found in Appendix A.2.

$[\forall\text{INTRO}]^*$
$$\frac{\Gamma^* \cup X \cup \{x{:}\, X \multimap T^*\} \vdash a{:}\, T'^*}{\Gamma^* \vdash \lambda X \lambda x^{X \multimap T^*}.a{:}\, \forall X(X \multimap T^*) \to T'^*}$$

$[\forall\text{ELIM}]^*$
$$\frac{\Gamma^* \vdash a{:}\, \forall X(X \multimap S^*) \to T^* \qquad \Gamma^* \vdash b{:}\, S'^* \multimap S^*}{\Gamma^* \vdash a(S'^*)(b){:}\, T^*[X := S'^*]}$$

$[\text{SUBSUMPTION}]^*$
$$\frac{\Gamma^* \vdash a{:}\, T'^* \qquad \Gamma^* \vdash b{:}\, T'^* \multimap T^*}{\Gamma^* \vdash \iota(b)(a){:}\, T^*}$$

5.4 Coherence

Theorem 5.1 *If Δ_1, Δ_2 are two $F_{\leq}^{\top}$ derivations of the same typing judgment, then their translations yield provably equal terms in* ***TARGET***.

Proof. The proof is much the same as the one in [BTCGS91], although it becomes a little bit simpler. It is not surprising that the main differences lie in the proof of the coherence of the translation of the subtyping derivation, since this is where the systems differ. Thus the reader can just follow the proof in section 5 of [BTCGS91]; the differences are as follows:

- In LEMMA 6 of [BTCGS91], use the equality

$$\begin{aligned} & \mathsf{forall}(a)(\lambda X.\lambda x^{X \multimap 1}.b) \odot \mathsf{forall}(c)(\lambda X.\lambda y^{X \multimap 1}.d) \\ = \; & \mathsf{forall}(c \odot a)(\lambda X.\lambda y^{X \multimap 1}.(b[x := y] \odot d) \end{aligned}$$

  to prove the equality of the translations when the derivation ends by a ($\forall$) rule.

- Erase LEMMA 7.

- Use the same proof for LEMMA 8 (of course you no longer need to use LEMMA 7).

- Extend the proof of lemma 4.9 of the previous section to take into account the new type constructors (it essentially amounts to add two cases that can be solved by a straightforward use of the induction hypothesis induction: the weight of a record/variant being the sum of the weights of its components): this corresponds to LEMMA 11 of [BTCGS91] (and it is where that proof failed for $F_{\leq}$).

- The proof of LEMMA 12 is nearly the same: just replace in the case [$\forall$ELIM] (in the paper [B-SPEC]) every occurrence of $a{\leq}\, s_i$ and $a{\leq}\, r$ in the type contexts by $a{\leq}\,\mathsf{Top}$ and use the (FORALL) version of the lemma; the sublemma requires a slight modification but on the other hand the proof is simpler since you no longer need to use LEMMAS 6 and 7 in the proof of this case.

- THEOREM 13 is unchanged

End of proof.

For a review of the models of **TARGET** see section 6 of [BTCGS91].

6 Conservativity of Simple Recursive Types

In Section 5, the only rule for comparing recursive types was reflexivity. Here we study a more interesting formulation of recursive types.[1] We add to the subtyping rules of section 2 the following rules

(Unfold-L) $$\frac{\Gamma \vdash S[X := \mu X.S] \leq T}{\Gamma \vdash \mu X.S \leq T}$$

(Unfold-R) $$\frac{\Gamma \vdash T \leq S[X := \mu X.S]}{\Gamma \vdash T \leq \mu X.S}$$

Let us denote the new system by $F_{\leq}^{\top\mu}$ and write its judgments with $\vdash_{\mu}$. In [Ghe93b] it is shown that adding these two rules to $F_{\leq}$ leads to a non-conservative extension — i.e. that there is a subtyping judgment in $F_{\leq}$ which is not provable in the original system but which is provable with the extended system. In that same paper a characterization is given for the pairs of type for which the conservativity does not hold. Here we can show that $F_{\leq}^{\top\mu}$ is a conservative extension of $F_{\leq}^{\top}$.

Unfortunately we cannot use Ghelli's characterization of the non-conservative judgments to prove this, since the set of non-provable judgments in our system is strictly larger than in $F_{\leq}$; thus there might be a judgment that makes conservativity fail in $F_{\leq}^{\top}$, but that is provable in $F_{\leq}$ and hence hence wouldn't be taken into account by this method. Therefore we prove conservativity directly, by showing that if a judgment not containing recursive types is provable in $F_{\leq}^{\top\mu}$ then it is provable in $F_{\leq}^{\top}$. More exactly, we prove that if $\Gamma \vdash_{\mu} S \leq T$ is provable and Γ, S, and T do not contain recursive types then $\Gamma \vdash S \leq T$ is provable, too. This can be obtained very straightforwardly by replacing in $\vdash_{\mu}$ the full transitivity rule by transitivity only on variables.

(AlgTrans) $$\frac{\Gamma \vdash \Gamma(X) \leq T}{\Gamma \vdash X \leq T}$$

We denote the corresponding proof system by $\vdash_{\mu\mathcal{A}}$.

Let us see more precisely the types we take into account; they are those of $F_{\leq}^{\top}$ plus the recursive types:

$$T ::= X \mid T \to T \mid \forall(X{\leq}S)T \mid \mu X.T$$

As in [Ghe93b], we forbid recursive types whose body is either Y, Top, or $\mu X.T$; this will simplify the proofs without changing the power of the system: the first two can be forbidden since $\mu X.Y$ ($Y \not\equiv X$), and $\mu X.\mathsf{Top}$ denote just Y and Top; $\mu X.X$ is meaningless (this is the only real restriction); and $\mu X.\mu Y.T$ can be forbidden since it has the same unfolding as $\mu X.T[Y := X]$. From a formal point of view we have to distinguish syntactically between recursion variables and universally quantified variables, since only the latter have upper bounds, but this is not essential for the purposes of this section.

[1] It is known that even this formulation is insufficiently flexible for some situations in which recursive types can be applied. For a deeper treatment of the combination of recursion and subtyping see [Ama91]. It is not known whether the proof of conservativity given here can be extended to a second-order variant of that system.

Lemma 6.1 *If there is a proof of* $\Gamma \vdash_{\mu\mathcal{A}} S \leq T$ *and no recursive type appears in the judgment then no unfold rule appears in the proof.*

Proof. A trivial induction on the depth of the proof of $\Gamma \vdash_{\mu\mathcal{A}} S \leq T$. *End of proof.*

In order to shorten the statement of the lemma that follows, we introduce the notion "having the same shape."

Definition 6.2 Two types *have the same shape* if they are both type variables, or both Top, or both arrow types, or both parametric types quantified over the same type variable, or both recursive types quantified over the same type variable.

Now we can to prove a lemma which roughly corresponds to Lemma 4.1:

Lemma 6.3 *Let* $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T_2$. *Then*

1. *If* T_1 *is not a variable then one of these cases holds:*
   - T_2 *is* Top
   - T_2 *has the same shape as* T_1
   - $T_1 \equiv \mu X.T$ *and* T_2 *has the same shape as* T
   - $T_2 \equiv \mu X.T$ *and* T_1 *has the same shape as* T
2. *If* T_2 *is not* Top *then one of these cases holds:*
   - T_1 *is a variable*
   - T_1 *has the same shape as* T_2
   - $T_2 \equiv \mu X.T$ *and* T_1 *has the same shape as* T
   - $T_1 \equiv \mu X.T$ *and* T_2 *has the same shape as* T

Proof. By induction on the depth of the proof of $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T_2$. We perform a case analysis on the last rule of the proof: when the last rule is (refl), ($\rightarrow$) or ($\forall$) then the result follows since T_1 and T_2 have the same shape. When the last rule is (Top) or (VarTrans) both the points of the lemma are satisfied, too. Suppose that the last rule is (unfold-L): then $T_1 \equiv \mu X.T$ and $\Gamma \vdash_{\mu\mathcal{A}} T[X := T_1] \leq T_2$. By assumption, T can be neither a variable nor a recursive type. Thus by part 1 of the induction hypothesis, there remain three subcases:

a) $T_2 \equiv \mathsf{Top}$ and the result is immediate.
b) $T[X := T_1]$ has the same shape as T_2. But since T is not a variable this implies that also T has the same shape as T_2 which gives the result.
c) T_2 is a recursive type, but then it has the same shape as T_1.

The case for (unfold-R) is similar: use the hypothesis that T cannot be Top. *End of proof.*

Finally we have transitivity elimination:

Lemma 6.4 *For each context* Γ *and types* T_1, T_2, *and* T_3 *well-formed in* Γ, *if* $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T_2$ *and* $\Gamma \vdash_{\mu\mathcal{A}} T_2 \leq T_3$ *then* $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T_3$.

Proof. The proof is very similar to the one of lemma 4.3. Again we prove the lemma by induction on the sum of the depths d_i of the proofs of $\Gamma \vdash_{\mu\mathcal{A}} T_i \leq T_{i+1}$ $(i = 1, 2)$, considering only the cases in which T_2 and T_3 are different from Top. Moreover, we can suppose that the last rule of $\Gamma \vdash_{\mu\mathcal{A}} T_2 \leq T_3$ is different from (refl), since in that case the result trivially holds. Proceed by case analysis on the last rule of the proof of $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T_2$:

(refl) the result then gets $\Gamma \vdash_{\mu\mathcal{A}} T_1 \equiv T_2 \leq T_3$ satisfied by hypothesis.

(AlgTrans) $T_1 \equiv X$. By the induction hypothesis we obtain $\Gamma \vdash_{\mu\mathcal{A}} \Gamma(X) \leq T_3$; by applying (AlgTrans) we obtain the result.

(Top) Not possible by assumption.

($\rightarrow$) $T_1 \equiv T_1' \rightarrow T_1''$ and $T_2 \equiv T_2' \rightarrow T_2''$. There are two subcases and both of them are proved by applying the induction hypothesis on $\Gamma \vdash_{\mu\mathcal{A}} T_3' \leq T_1'$ and $\Gamma \vdash_{\mu\mathcal{A}} T_1'' \leq T_3''$. Since $T_3 \not\equiv \mathsf{Top}$, by lemma 6.3 either:
1. $T_3 \equiv T_3' \rightarrow T_3''$. By the form of T_1, T_2, and T_3 and the definition of $\vdash_{\mu\mathcal{A}}$, there are smaller derivations of $\Gamma \vdash_{\mu\mathcal{A}} T_3' \leq T_2' \leq T_1'$ and $\Gamma \vdash_{\mu\mathcal{A}} T_1'' \leq T_2'' \leq T_3''$. The induction hypothesis then yields $\Gamma \vdash_{\mu\mathcal{A}} T_3' \leq T_1'$ and $\Gamma \vdash_{\mu\mathcal{A}} T_1'' \leq T_3''$, from which the result follows by ($\rightarrow$).
2. $T_3 \equiv \mu X.T' \rightarrow T''$. Similar, except that the right-hand subderivation must end with (unfold-R) preceded by ($\rightarrow$), yielding subderivations of $\Gamma \vdash_{\mu\mathcal{A}} T_3'[X := T_3] \leq T_2' \leq T_1'$ and $\Gamma \vdash_{\mu\mathcal{A}} T_1'' \leq T_2'' \leq T_3''[X := T_3]$. The result then follows from the induction hypothesis, ($\rightarrow$), and (unfold-R).

($\forall$) Similar.

(unfold-l) By the induction hypothesis and (unfold-L).

(unfold-r) $T_2 \equiv \mu X.T$. Then there are two possible subcases:

1. The last rule of $\Gamma \vdash_{\mu\mathcal{A}} T_2 \leq T_3$ is (unfold-L). Then we apply the induction hypothesis on $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T[X := T_2]$ and $\Gamma \vdash_{\mu\mathcal{A}} T[X := T_2] \leq T_3$.
2. The last rule of $\Gamma \vdash_{\mu\mathcal{A}} T_2 \leq T_3$ is (unfold-R), so $T_3 \equiv \mu Y.T'$. Thus we can apply the induction hypothesis to $\Gamma \vdash_{\mu\mathcal{A}} T_1 \leq T_2$ and $\Gamma \vdash_{\mu\mathcal{A}} T_2 \leq T'[Y := T_3]$. By applying (unfold-R) again, we obtain the result.

End of proof.

Corollary 6.5 *If* $\Gamma \vdash_{\mu} S \leq T$ *is provable then* $\Gamma \vdash_{\mu\mathcal{A}} S \leq T$ *is also provable.*

Proof. By induction on the depth of the proof of $\Gamma \vdash_{\mu} S \leq T$. Apply lemma 6.4 when the last rule is (trans). *End of proof.*

7 Conclusions

We have presented a new formulation of bounded quantification, enjoying many properties that the usual formulation, $F_{\leq}$, lacks. This system also has a high tolerance for extensions, as we showed in the case of recursive typesand in the study of the semantics. This leads us to believe that $F_{\leq}^{\top}$ should also retain the minimal typing property when extended by bounded existential types. (This property has been established in [GP93] for $F_{\leq}$ with ($\forall$-Fun) instead of ($\forall$-orig) by showing the completeness of a type synthesis algorithm suggested by Dezani, which can perhaps be adapted to ($\forall$-top).) The investigation of extensions of $F_{\leq}^{\top}$ with higher-order polymorphism is also underway.

Acnowledgments

Special thanks go to Luca Cardelli, without whose encouragement this paper would not have been written. Giorgio Ghelli joined us in enlightening conversations about the relation of $F_{\leq}^{\top}$ to its neighbors. The observation that the Penn-style semantics is linear is due to Martín Abadi.

References

[Ama91] R. Amadio. *Recursion and Subtyping in Lambda Calculi.* PhD thesis, Università degli Studi di Pisa, 1991.

[BM92] Kim Bruce and John 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.

[Bru93] K. B. Bruce. Safe type checking in a statically typed object-oriented programming language. In *20th Ann. ACM Symp. on Principles of Programming Languages*. ACM Press, 1993.

[BTCGS91] V. Breazu-Tannen, T. Coquand, C. Gunter, and A. Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, July 1991.

[Car92] Luca Cardelli. Extensible records in a pure calculus of subtyping. Research report 81, DEC Systems Research Center, January 1992. To appear in [GM93].

[Car93] L. Cardelli. An implementation of $F_{<:}$. Technical Report 97, Digital Equipment Corporation, February 1993.

[CG92] P. L. Curien and G. Ghelli. Coherence of subsumption, minimum typing and the type checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2(1), 1992.

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

[dB72] Nicolas 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.

[Ghe90] G. Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1990. Tech. Rep. TD-6/90.

[Ghe93a] G. Ghelli. Divergence of $F_{\leq}$ type-checking. Technical Report 5/93, Dipartimento d'Informatica, Università degli Studi di Pisa, 1993.

[Ghe93b] G. Ghelli. Recursive types are not conservative over $F_{\leq}$. In M. Bezem and J.F. Groote, editors, *International Conference on Typed Lambda calculi and Applications*, number 664 in LNCS, pages 146–162, Utrecht, The Netherlands, March 1993. Springer-Verlag. TLCA'93.

[Ghe93c] G. Ghelli. S-All-Loc is not transitive. Mail to the TYPES mailing list, February 1993.

[Gir72] J-Y. Girard. Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Thèse de doctorat d'état, 1972. Université Paris VII.

[GM93] Carl A. Gunter and John C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design.* The MIT Press, 1993. To appear.

[GP93] G. Ghelli and B. Pierce. Bounded existentials and minimal typing. Draft report, Dipartimento d'Informatica Università di Pisa, 1993. Unpublished.

[KS92] D. Katiyar and S. Sankar. Completely bounded quantification is decidable. In *Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*, pages 68–77, San Francisco, June 1992.

[Mit90] John 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. To appear in [GM93].

[Pie93] Benjamin C. Pierce. Bounded quantification is undecidable. *Information and Computation*, 1993. To appear; also to appear in [GM93]. Preliminary version in proceedings of POPL '92.

[PT93] B.C. Pierce and D.N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 1993. To appear; 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".

[Rey74] J.C. Reynolds. Towards a theory of type structures. *LNCS*, 19:408–425, 1974.

A Summary of Typing Rules

A.1 Original $F_{\leq}$ typing and subtyping rules

(refl) $$\Gamma \vdash T \leq T$$

(trans) $$\frac{\Gamma \vdash T_1 \leq T_2 \qquad \Gamma \vdash T_2 \leq T_3}{\Gamma \vdash T_1 \leq T_3}$$

(tvar) $$\Gamma \vdash X \leq \Gamma(X)$$

(Top) $$\Gamma \vdash T \leq \mathsf{Top}$$

($\rightarrow$) $$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \vdash S_2 \leq T_2}{\Gamma \vdash S_1 \rightarrow S_2 \leq T_1 \rightarrow T_2}$$

($\forall$) $$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma \cup \{X \leq T_1\} \vdash S_2 \leq T_2}{\Gamma \vdash \forall(X{\leq}S_1)S_2 \leq \forall(X{\leq}T_1)T_2}$$

[Vars] $$\Gamma \vdash x\colon \Gamma(x)$$

[$\rightarrow$Intro] $$\frac{\Gamma \cup \{x\colon T\} \vdash a\colon T'}{\Gamma \vdash (\lambda x^T.a)\colon T \rightarrow T'}$$

[$\rightarrow$Elim] $$\frac{\Gamma \vdash a\colon S \rightarrow T \qquad \Gamma \vdash b\colon S}{\Gamma \vdash a(b)\colon T}$$

[Top] $$\Gamma \vdash \mathsf{top}\colon \mathsf{Top}$$

[$\forall$Intro] $$\frac{\Gamma \cup \{X{\leq}T\} \vdash a\colon T'}{\Gamma \vdash \Lambda X{\leq}T.a\colon \forall(X{\leq}T)T'}$$

[$\forall$Elim] $$\frac{\Gamma \vdash a\colon \forall(X{\leq}S)T \qquad \Gamma \vdash S' \leq S}{\Gamma \vdash a(S')\colon T[X := S']}$$

[Subsumption] $$\frac{\Gamma \vdash a\colon T' \qquad \Gamma \vdash T' \leq T}{\Gamma \vdash a\colon T}$$

A.2 Translation of $F_{\leq}^{\top}$ into explicit coercions

$$
\begin{array}{lcl}
X^* & = & X \\
\mathsf{Top}^* & = & 1 \\
(S \to T)^* & = & S^* \to T^* \\
(\forall(X{\leq}S)T)^* & = & \forall X.((X \multimap S^*) \to T^*) \\
\langle\!\langle \ell_1 : T_1, \ldots, \ell_n : T_n \rangle\!\rangle^* & = & \langle\!\langle \ell_1 : T_1^*, \ldots, \ell_n : T_n^* \rangle\!\rangle \\
([\ell_1 : T_1, \ldots, \ell_n : T_n])^* & = & [\ell_1 : T_1^*, \ldots, \ell_n : T_n^*] \\
(\mu X.T)^* & = & \mu X.T^* \\
\\
\emptyset^* & = & \emptyset \\
(\Gamma \cup \{X{\leq}T\})^* & = & \Gamma^* \cup X \cup \{x : X \multimap T^*\} \\
(\Gamma \cup \{x : T\})^* & = & \Gamma^* \cup \{x : T^*\}
\end{array}
$$

(refl)*
$$\Gamma^* \vdash \mathsf{refl} \colon T^* \multimap T^*$$

(trans)*
$$\frac{\Gamma^* \vdash a : T_1^* \multimap T_2^* \qquad \Gamma^* \vdash b : T_2^* \multimap T_3^*}{\Gamma^* \vdash \mathsf{trans}(a)(b) : T_1^* \multimap T_3^*}$$

(tvar)*
$$\Gamma^* \cup X \cup \{x \colon X \multimap \Gamma(X)^*\} \vdash x \colon X \multimap \Gamma(X)^*$$

(Top)
$$\Gamma \vdash \mathsf{top}[T^*] \colon T^* \multimap 1$$

$(\to)^*$
$$\frac{\Gamma^* \vdash a \colon T_1^* \multimap S_1^* \qquad \Gamma^* \vdash b \colon S_2^* \multimap T_2^*}{\Gamma^* \vdash \mathsf{arrow}(a)(b) \colon (S_1^* \to S_2^*) \multimap (T_1^* \to T_2^*)}$$

$(\forall)^*$
$$\frac{\Gamma^* \vdash a \colon T_1^* \multimap S_1^* \qquad \Gamma^* \cup X \cup \{x \colon X \multimap 1\} \vdash b \colon S_2^* \multimap T_2^*}{\Gamma^* \vdash \mathsf{forall}(a)(\lambda X.\lambda x^{X \multimap 1}.b) \colon \forall X((X \multimap S_1^*) \to S_2^*) \multimap \forall X((X \multimap T_1^*) \to T_2^*)}$$

(recd)*
$$\frac{\Gamma^* \vdash a_1 \colon S_1^* \multimap T_1^* \ldots \Gamma^* \vdash a_p \colon S_p^* \multimap T_p^*}{\Gamma^* \vdash \mathsf{recd}(a_1) \cdots (a_p) \colon \langle\!\langle \ell_1 \colon S_1^*, \ldots, \ell_p \colon S_p^*, \ldots, \ell_q \colon S_q^* \rangle\!\rangle \multimap \langle\!\langle \ell_1 \colon T_1^*, \ldots, \ell_p \colon T_p^* \rangle\!\rangle}$$

(vart)*
$$\frac{\Gamma^* \vdash a_1 \colon S_1^* \multimap T_1^* \ldots \Gamma^* \vdash p \colon S_p^* \multimap T_p^*}{\Gamma^* \vdash \mathsf{vart}(a_1) \cdots (a_p) \colon [\ell_1 \colon S_1^*, \ldots, \ell_p \colon S_p^*] \multimap [\ell_1 \colon T_1^*, \ldots, \ell_p \colon T_p^*, \ldots, \ell_q \colon T_q^*]}$$

[Vars]*
$$\Gamma^* \vdash x \colon \Gamma(x)^*$$

[$\to$INTRO]*
$$\frac{\Gamma^* \cup \{x \colon T^*\} \vdash a \colon T'^*}{\Gamma^* \vdash (\lambda x^{T^*}.a) \colon T^* \to T'^*}$$

$$[\to\text{ELIM}]^* \qquad \frac{\Gamma^* \vdash a{:}\, S^* \to T^* \qquad \Gamma^* \vdash b{:}\, S^*}{\Gamma^* \vdash a(b){:}\, T^*}$$

$$[\text{TOP}]^* \qquad \Gamma^* \vdash \langle\rangle{:}\, 1$$

$$[\forall\text{INTRO}]^* \qquad \frac{\Gamma^* \cup X \cup \{x{:}\, X \circ\!\!\to T^*\} \vdash a{:}\, T'^*}{\Gamma^* \vdash \lambda X \lambda x^{X \circ\!\to T^*}.a{:}\, \forall X(X \circ\!\!\to T^*) \to T'^*}$$

$$[\forall\text{ELIM}]^* \qquad \frac{\Gamma^* \vdash a{:}\, \forall X(X \circ\!\!\to S^*) \to T^* \Gamma^* \vdash b{:}\, S'^* \circ\!\!\to S^*}{\Gamma^* \vdash a(S'^*)(b){:}\, T^*[X := S'^*]}$$

$$[\langle\!\langle\rangle\!\rangle\text{INTRO}]^* \qquad \frac{\Gamma^* \vdash a_1{:}\, T_1^* \ldots \Gamma^* \vdash a_p{:}\, T_p^*}{\Gamma^* \vdash \langle \ell_1 = a_1, \ldots, \ell_p = a_p\rangle{:}\, \langle\!\langle \ell_1{:}\, T_1^*, \ldots, \ell_p{:}\, T_p^* \rangle\!\rangle}$$

$$[\langle\!\langle\rangle\!\rangle\text{ELIM}]^* \qquad \frac{\Gamma^* \vdash a{:}\, \langle\!\langle \ell_1{:}\, T_1^*, \ldots, \ell_p{:}\, T_p^* \rangle\!\rangle}{\Gamma^* \vdash a.\ell_i{:}\, T_i^*}$$

$$[\,[]\text{INTRO}]^* \qquad \frac{\Gamma^* \vdash a{:}\, T_i^*}{\Gamma^* \vdash [\ell_1{:}\, T_1^*, \ldots, \ell_i = a_i, \ldots, \ell_p{:}\, T_p^*]{:}\, [\ell_1{:}\, T_1^*, \ldots, \ell_i{:}\, T_i^*, \ldots, \ell_p{:}\, T_p^*]}$$

$$[\,[]\text{ELIM}]^* \qquad \frac{\Gamma^* \vdash b{:}\, [\ell_1{:}\, T_1^*, \ldots, \ldots, \ell_p{:}\, T_p^*] \qquad \Gamma^* \vdash a_1{:}\, T_1^* \to T^* \ldots \Gamma^* \vdash a_p{:}\, T_p^* \to T^*}{\Gamma^* \vdash \textbf{case}\ b\ \textbf{of}\ \ell_1 \Rightarrow a_1, \ldots, \ell_p \Rightarrow a_p{:}\, T^*}$$

$$[\mu\text{INTRO}]^* \qquad \frac{\Gamma^* \vdash a{:}\, T^*[X := \mu X.T^*]}{\Gamma^* \vdash \textbf{intro}_{\mu X.T^*}(a){:}\, \mu X.T^*}$$

$$[\mu\text{ELIM}]^* \qquad \frac{\Gamma^* \vdash a{:}\, \mu X.T^*}{\Gamma^* \vdash \textbf{elim}(a){:}\, [X := \mu X.T^*]}$$

$$[\text{SUBSUMPTION}]^* \qquad \frac{\Gamma^* \vdash a{:}\, T'^* \qquad \Gamma^* \vdash b{:}\, T'^* \circ\!\!\to T^*}{\Gamma^* \vdash \iota(b)(a){:}\, T^*}$$

B Correspondence about $F_{\leq}^{\top}$ from the Types list

From: Giorgio Ghelli <ghelli@di.unipi.it>
Subject: Fsubtop does not enjoy the minimum type property
To: types@dcs.gla.ac.uk
Date: Fri, 28 Jan 1994 13:07:51 +0100 (MET)

Castagna and Pierce recently introduced a new decidable variation "FSubTop" of system Fsub, the standard type system which extends Girard's F with subtyping. Fsubtop, although less natural then Fsub, looked more tractable and more robust with respect to extensions ([Castagna and Pierce, POPL 94]).

I am sorry to notice that this is not true, and that Fsubtop lacks one of the fundamental properties of Fsub, the existence of a minimum type for any term; consequently, no type checking algorithm which returns just one type for any term can be complete. A counterexample is below.

The problem in the paper presenting Fsubtop is that the authors focus their efforts on the subtyping relation, and their completeness result for the subtype checking algorithm is indeed correct. Then they assume that the typing rules, which are the same as for Fsub, will not create any problem. The authors, like most of us who know their system, did not realize that the Fsub rules smoothly interact with Fsub subtyping, but have a poor interaction with Fsubtop subtyping.

The counterexample I give shows that the problem stems from the fact that Fsubtop is not an extension of kernel-Fun (I recall here Fsubtop and kernel-Fun universal subtyping rule; the other rules are as for Fsub:

```
 G |- T <: T'   G, X<:Top |- U' <: U
 ----------------------------------  (Fsubtop-All<:)
  G |- All(X<:T').U' <: All(X<:T).U

 G, X<:T |- U' <: U
 -------------------------------  (k-Fun-All<:)
 G |- All(X<:T).U' <: All(X<:T).U                  )
```

However, the following fact holds:

FACT: System Fsubtop enriched with rule k-Fun-All is Fsub.

This reminds me to an older discussion on this forum, where it was proved that another subsystem of Fsub (here called F-sub-loc minus transitivity) does not enjoy transitivity, and that its minimum transitive extension is exactly Fsub. It seems difficult to find well-behaved subsystems of Fsub different from kernel-Fun.

****

Fsubtop does not enjoy the minimum type property.
===

The term Y<:Top |- /\X<:Y.\x:X.x is a counterexample; it enjoys both types:

```
Y<:Top |- /\X<:Y.\x:X.x : All(X<:Y).X->X
Y<:Top |- /\X<:Y.\x:X.x : All(X<:Y).X->Y
```

In Fsub the first type is its minimum type, and is a subtype of the second one (by the way, the same is true in system kernel-Fun, which enjoys minimum typing).
In Fsubtop, the first type is not a subtype of the second one (as can be proved by applying Castagna and Pierce's complete subtype-checking algorithm), and they are both minimal in the set of the types of the above term.
We give just the final steps of one proof of the second judgement:

```
                 .                               .
                 .                               .
                 .                               .
   Y<:Top, Y<:X |- \x:X.x : X->X    Y<:Top, Y<:X |-  X->X <: X->Y
   ---------------------------------------------------------------(Subsumption)
                   Y<:Top, Y<:X |-  \x:X.x : X->Y
                  ---------------------------------------(All Intro)
                  Y<:Top |- /\X<:Y.\x:X.x : All(X<:Y).X->Y
```

```
*****************************************************************************
```

System Fsubtop enriched with rule k-Fun-All is Fsub.
===

PROOF: Any Fsub subtyping proof becomes an Fsubtop+kernelFun proof if you substitute any application of the All rule:

```
 G |- T' <: T         G, X<:T' |- U' <: U
 ---------------------------------------  (Fsub-All)
     G |- All(X<:T).U' <: All(X<:T').U
```

With:

```
                              (1)
G |- T' <: T        G, X<:Top |- U' <: U'         G, X<:T' |- U' <: U
-----------------------------------------(2) ------------------------------ (3)
   G |- All(X<:T).U' <: All(X<:T').U'       G |- All(X<:T').U' <: All(X<:T').U
   ---------------------------------------------------------------------------
                       G |- All(X<:T).U' <: <: All(X<:T').U
```

(1): G, X<:Top is strong enough to prove U' <: U'
(2): Fsubtop-All (3): K-fun-All.

Prof. Giorgio Ghelli,   Universita' di Pisa, Dipartimento di Informatica,
Corso Italia 40, I-56125, Pisa, ITALY
E-mail: ghelli@di.unipi.it; Phone: +39-50-887258;   Fax: +39-50-887226

From: Benjamin Pierce <bcp@dcs.ed.ac.uk>
To: types@dcs.gla.ac.uk
Subject: Re: Fsubtop and minimal typing
Date: Fri, 28 Jan 1994 12:42:35 +0000

Fsub's score of dead and wounded is still rising... Giorgio's observation is, alas, quite correct. Although Fsubtop's subtype relation behaves better than Fsub's, there are serious difficulties lurking in the typing relation.

At this point, there seem to be several ways to proceed:

1) We could return to the complex, intractable world of full Fsub, with its undecidable subtyping problem, missing order-theoretic properties, and difficult proof theory. Maybe life is just complicated.

2) Or we could step even further back, to kernel Fun, with its "unnatural looking" syntactic restriction that bounds of related quantifiers must be the same (which may not be so unnatural in the end!) and its simpler proof theory and good properties of both typing and subtyping relations.

   Although I've argued that the subtyping rule of kernel Fun looks less natural than Fsubtop's version, the pragmatic difference does not actually seem to be large. For example, Martin Steffen and I have checked that our analysis of the omega-order extension of Fsubtop can be modified to deal with the kernel Fun variant. (The new analysis includes a proof of the completeness of a subtyping algorithm, as before, as well as a proof of the completeness of a typechecking algorithm, which we had claimed *incorrectly* for FomegaSub with the top-rule.) A corrected version of our paper will be announced shortly.

3) We could keep Fsubtop, for the sake of its nice subtype relation, and alter the typechecking relation to fit better with subtyping. This might be accomplished in a couple of ways:

   - Refining the type synthesis procedure so that it returns *sets* of minimal types. But after a few days of thought [Giorgio was kind enough to communicate his observation to a few people privately before announcing it], I cannot see any reasonable basis for an acceptably efficient procedure.

     My pessimism is based on the following depressing example. If

     ```
     G  =  ... D < C->C, B < C, A < B
     G' =  ... D < C->C, B < C, A < Top,
     ```

     then the statement

     ```
     G  |- D < A->C
     ```

     is provable, but

     ```
     G' |- D < A->C
     ```

     is not! So if you try to compute a complete set of minimal types for a term like

     ```
     fun(A<B) fun(x:D) x
     ```

     by "breaking the link" between A and its upper bound B, you will have to consider promoting D to C->C and then further promoting the result, even though D appears quite a distance from A in the context and (worse yet) even though D is bound *before* A. This observation suggests that the "sets of minimal types" we are imagining computing will be huge even if they are finite.

- Disallowing subsumption except in the premises of the elimination rules, effectively replacing the usual definition of the typing relation by the usual type synthesis algorithm.

  But if we adopt this solution, we have to give up another important property: subject reduction. For example, consider the term:

  ```
  fun(X<Y) fun(x:X)  ((fun(z:Y)z) x)
  ```

  The unique type assigned to this term is All(X<Y) X->Y. But after reducing its internal beta-redex to yield

  ```
  fun(X<Y) fun(x:X) x,
  ```

  it has another unique type, All(X<Y)X->X, which is not a subtype of the original.

  It remains unclear whether some weaker form of subject reduction could still hold.

- Forcing the programmer to declare the intended type of the body of each abstraction on a type variable. The quantifier introduction rule would become something like this:

  ```
  G, X<S |- e in T'    G, X<Top |- T' < T
  ---------------------------------------
     G |- fun(X<S) e@T  in  All(X<Y) T
  ```

  Of course, it is unlikely that programmers will be happy having to insert these "@T" annotations in all their polymorphic function definitions, so the almost inevitable next step is allowing them to be omitted: an incomplete -- but perhaps perfectly useable -- notion of partial type reconstruction.

  A number of languages these days seem to be choosing powerful type systems with simple-but-incomplete type inference procedures, so this solution might not be as bad as it first sounds. Moreover, it can be shown that even Fsub itself exhibits a similar failure of minimal types when extended in a natural way with existential types. (Intriguingly, kernel Fun retains minimal typing even under this extension.)

4) We can search for much simpler notions of subtyping that capture our pragmatic intentions (characterizing the behavior of objects and message passing, for example) more directly. This amounts to taking seriously Giorgio's oft-repeated exhortation that "we need *semantic* justifications for our type systems."

Personally, I tend to favor (2) and (4) -- (2) in the short term, as the most reasonable of the known formulations of bounded quantification, and (4) as a longer-range research goal.

-- Benjamin Pierce