Bounded Existentials and Minimal Typing

Giorgio Ghelli* **Benjamin Pierce†**

July 9, 1996

Abstract

We study an extension of the second-order calculus of bounded quantification, System $F_{\leq}$, with bounded existential types. Surprisingly, the most natural formulation of this extension lacks the important *minimal typing* property of $F_{\leq}$, which ensures that the set of types possessed by a typeable term can be characterized by a single least element. We consider alternative formulations and give an algorithm computing minimal types for the slightly weaker Kernel Fun variant of $F_{\leq}$.

1 Introduction

$F_{\leq}$ is a typed lambda-calculus combining subtyping and second-order bounded quantification [4, 5, 7, 3]. Besides its utility as a vehicle for theoretical investigations, it has come to be seen as a good basis for the design of programming languages incorporating subtyping and polymorphism.

The extension of $F_{\leq}$ with bounded existential quantifiers to support programming with abstract data types is commonly regarded as a straightforward task; indeed, in a sense, pure $F_{\leq}$ already contains bounded existentials, since they can be encoded in terms of bounded universals. However, the system obtained by extending $F_{\leq}$ with existential types is rather different from $F_{\leq}$ itself. We show here that terms in this extended system fail to possess a unique minimal type; it follows that the standard type synthesis algorithm, which is complete for pure $F_{\leq}$, cannot be extended to a complete algorithm for the richer system.

The loss of the minimal type property is a consequence of the fact that the derived elimination form for encoded existentials contains more type information than the corresponding form for primitive existentials. As a result, while primitive existantials fail to satisfy the minimal type property, encoded existentials are pragmatically unsatisfactory since their elimination form is too verbose.

On the other hand, in the case of the analogous extension of the "equal-bounds fragment" of $F_{\leq}$ (often called Kernel Fun) we can give a procedure for performing the crucial operation of promoting a given type to its minimal supertype in which a given variable does not occur free. This leads to a straightforward proof of the minimum type property and of the completeness of a type synthesis algorithm for Kernel Fun extended with primitive bounded existentials.

*Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125, Pisa, Italy. Electronic mail: ghelli@di.unipi.it.

†Computer Laboratory, University of Cambridge. Present address: Computer Science Department, Indiana University, Lindley Hall 215, Bloomington, Indiana 47405, USA. Electronic mail: pierce@cs.indiana.edu.

2 Bounded Existentials in System $F_{\leq}$

We begin with an essentially negative result, showing that the natural extension of System $F_{\leq}$ with bounded existentials lacks the minimal typing property. In Section 3, we weaken $F_{\leq}$ and obtain a sound and complete algorithm for calculating minimal types.

2.1 System $F_{\leq}$

The types of pure $F_{\leq}$ include type variables, function types, universal quantifiers, and a maximal type *Top*:

Types	T	$::=$	$A \mid T{\rightarrow}T \mid \mathit{All}(A{\leq}T)\,T \mid \mathit{Top}$
Terms	e	$::=$	$x \mid \mathit{fun}(x{:}T)\,e \mid e_1\ e_2 \mid \mathit{fun}(T)\,e \mid e\ T$
Environments	Γ	$::=$	$() \mid \Gamma, x{:}T \mid \Gamma, A{\leq}T$
Judgements	j	$::=$	$\Gamma \vdash e : T \mid \Gamma \vdash T \leq T$

Inference rules defining the subtyping and typing relations are given in Appendix B. We regard variables as names for their DeBruijn indexes, hence judgement provability is defined modulo α-conversion and substitution is always capture free. This convention allows us to ignore the type and environment formation rules that would otherwise be needed to assure that every type variable in a judgement is bound and that no variable appears twice in the environment. See [5, 3] for more detailed presentations of the system.

The usual presentation of $F_{\leq}$ is not syntax-directed: given a typing or subtyping statement, there may be different proofs of its validity, and given a term and an environment, there can be proofs of different typings. However, an important property of $F_{\leq}$ is that it is possible to identify a set of derivations in "normal form" such that whenever a subtyping or typing statement can be proved using any derivation whatsoever, it can be proved by a unique derivation in normal form. There is a syntax-directed algorithm (summarized by the rules in Appendix C) that discovers the minimal type for any typeable $F_{\leq}$ term by constructing its normal-form derivation [5]. Although it fails to terminate on some pathological ill-typed terms [8, 11], the algorithm is easy to implement and has been found to behave well in practice.

2.2 Encoded and Primitive Existentials in $F_{\leq}$

The simplest way to add existential quantifiers to $F_{\leq}$ is to regard them as syntactic sugar for combinations of universal quantifiers, using the standard encoding:

$$\begin{aligned}\mathit{Some}(A{\leq}S_1)\,S_2 \quad = \quad & \mathit{All}(B)\ \ (\mathit{All}(A{\leq}S_1)\,S_2{\rightarrow}B) \rightarrow B \\ & \text{where } B \text{ is fresh}\end{aligned}$$

Syntactic sugar for introducing and eliminating existentials is provided by the following abbreviations (where again B is fresh):

$$\begin{aligned}& \mathit{pack}\ e\ \mathit{as}\ \mathit{Some}(A{\leq}S_1)\,S_2\ \mathit{hiding}\ T \\ & \quad = \quad \mathit{fun}(B)\,\mathit{fun}(f{:}\mathit{All}(A{\leq}S_1)\,S_2{\rightarrow}B)\ \ f\ T\ e \\ & \mathit{open}\ e{:}\ \mathit{Some}(A{\leq}S_1)\,S_2\ \mathit{as}\ [A,x]\ \mathit{in}\ b{:}T \\ & \quad = \quad e\ T\ \ (\mathit{fun}(A{\leq}S_1)\,\mathit{fun}(x{:}S_2)\,b)\end{aligned}$$

These encodings give rise to a derived subtyping rule for existential types

$$\frac{\Gamma \vdash S_1 \leq T_1 \qquad \Gamma, A{\leq}S_1 \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{Some}\,(A{\leq}S_1)\; S_2 \leq \mathit{Some}\,(A{\leq}T_1)\; T_2} \qquad \text{(Der-Some-Sub)}$$

and derived typing rules for *pack* and *open*:

$$\frac{\Gamma \vdash T \leq S_1 \qquad \Gamma \vdash e : [T/A]S_2}{\Gamma \vdash (\mathit{pack}\; e\; \mathit{as}\; \mathit{Some}\,(A{\leq}S_1)\; S_2\; \mathit{hiding}\; T) : \mathit{Some}\,(A{\leq}S_1)\; S_2} \qquad \text{(Der-Some-I)}$$

$$\frac{\Gamma \vdash e : \mathit{Some}\,(A{\leq}S_1)\; S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b : T \qquad A \text{ not in } FV(T)}{\Gamma \vdash (\mathit{open}\; e{:}\mathit{Some}\,(A{\leq}S_1)\; S_2\; \mathit{as}\; [A,x]\; \mathit{in}\; b{:}T) : T} \qquad \text{(Der-Some-E)}$$

Note that the subtyping rule Der-Some-Sub is nearly identical to the primitive subtyping rule

$$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma, A{\leq}T_1 \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{All}(A{\leq}S_1)\; S_2 \leq \mathit{All}(A{\leq}T_1)\; T_2} \qquad \text{(All-Sub)}$$

for bounded universals — the difference being that the bounds are compared covariantly rather than contravariantly — and that Der-Some-I and Der-Some-E straightforwardly generalize the usual typing rules for unbounded existentials [9, 10].

Alternatively, bounded existential types can be provided as a primitive syntactic form [4] by extending the grammar of types and terms with the *Some*, *pack*, and *open* constructs, and adding typing and subtyping rules for existentials to those in Appendix B.

The natural subtyping rule for primitive existentials is the same as the derived rule:

$$\frac{\Gamma \vdash S_1 \leq T_1 \qquad \Gamma, A{\leq}S_1 \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{Some}\,(A{\leq}S_1)\; S_2 \leq \mathit{Some}\,(A{\leq}T_1)\; T_2} \qquad \text{(Some-Sub)}$$

There is some freedom in the formulation of the introduction and elimination rules. We consider the rules in turn.

In Some-I, the annotation "*as* $\mathit{Some}\,(A{\leq}S_1)\; S_2$" is clearly required, since, in general, a given "concrete body" can validly be packaged as an instance of many different (and incomparable) existential types. For example, the value $\mathit{fun}\,(x{:}\mathit{Int})\; x$ can be packaged as:

$\mathit{Some}\,(A{\leq}\mathit{Int})\; A{\rightarrow}A$
$\mathit{Some}\,(A{\leq}\mathit{Int})\; A{\rightarrow}\mathit{Int}$
$\mathit{Some}\,(A{\leq}\mathit{Int})\; \mathit{Int}{\rightarrow}\mathit{Int}$
$\mathit{Some}\,(A{\leq}\mathit{Top})\; \mathit{Int}{\rightarrow}\mathit{Int}$
$\mathit{Some}\,(A{\leq}\mathit{Top})\; \mathit{Int}{\rightarrow}A$
etc.

The explicit annotation "*as* $\mathit{Some}\,(A{\leq}S_1)\; S_2$" functions as an aid both to the compiler (which would have no chance of guessing correctly which existential type was intended) and to the programmer, since it documents the signature of the abstract type implemented by the package.

The annotation "*hiding* T" also seems desirable on pragmatic grounds, since it allows the compiler to generate much more comprehensible error messages. If the annotation were inferred, then when the programmer gave a wrong implementation of some operation he would get an error message saying "can't infer T" offering very little help in guessing which operation has been implemented incorrectly; on the other hand, if T is given, the type checker can point directly to the wrong operation saying, "I was expecting $Int \times Int$ but I found $Real \times Real$."[1]

The natural introduction form for primitive bounded existentials (proposed by Cardelli and Wegner) is thus identical to the derived form:

$$\frac{\Gamma \vdash T \leq S_1 \qquad \Gamma \vdash e : [T/A]S_2}{\Gamma \vdash (\mathit{pack}\ e\ \mathit{as}\ \mathit{Some}\,(A{\leq}S_1)\ S_2\ \mathit{hiding}\ T) : \mathit{Some}\,(A{\leq}S_1)\ S_2} \tag{Some-I}$$

What about the existential elimination rule?

At first glance, it appears that none of the type annotations should be needed: it cannot hurt to derive a *better* existential typing for the expression e, since this will result in smaller types being used in place of the given S_1 and S_2 in the second premise, necessarily yielding a better result. Likewise, it cannot hurt to derive a minimal type for the body b instead of the given (and perhaps non-minimal) type T. This might lead us to conjecture that the "pure" elimination rule proposed by Cardelli and Wegner, besides being the more natural formulation, is also free of problems from the algorithmic point of view:

$$\frac{\Gamma \vdash e : \mathit{Some}\,(A{\leq}S_1)\ S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b : T \qquad A \text{ not in } FV(T)}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) : T} \tag{Some-E}$$

However, although the first part of this line of reasoning is valid (we recapitulate it more formally below), the second part is not: choosing the minimal type for b is *not* always better than choosing a particular supertype of the minimal type. The source of the difficulty is the side condition "A not in $FV(T)$," which blocks the formulation of a complete type synthesis algorithm.

Suppose we *did* have a type-synthesis algorithm for $F_{\leq}$ with existentials. The idea of such algorithms is usually to give a system of syntax-directed rules defining a *minimal type synthesis* relation $\Gamma \vdash e \Rightarrow T$, pronounced "under assumptions Γ, the smallest type of e is T". For example, the usual function application rule

$$\frac{\Gamma \vdash f : T_1 {\to} T_2 \qquad \Gamma \vdash a : T_1}{\Gamma \vdash f a : T_2}$$

is replaced by a variant that explicitly takes into account any possible uses of the rule of subsumption that may be needed in order to give f an arrow type and to make the type of the argument match the domain of the arrow type

$$\frac{\Gamma \vdash f \Rightarrow T \qquad \Gamma \vdash T \ll T_1 {\to} T_2 \qquad \Gamma \vdash a \Rightarrow T_1' \qquad \Gamma \vdash T_1' \leq T_1}{\Gamma \vdash f a \Rightarrow T_2}$$

where the auxiliary function $\ll$ computes the minimal non-variable supertype of a given type (see Appendix C).

In such an algorithm, the syntax-directed reformulation of Some-E would be

[1] If this annotation *were* omitted, then the type synthesis algorithm would have the task of determining, in each case, whether there exists any T that can validly be supplied as the witness type of the newly created existential, i.e. it should be able to determine, for each triple of types S_1, S_2, U, whether there exists a type T such that $T \leq S_1$ and $U \leq [T/A]S_2$ (where U is the minimum type of the packed expression). The problem is undecidable, given the undecidability of subtyping for $F_{\leq}$ [11]: when S_2 is A, a T such that $T \leq S_1$ and $U \leq [T/A]S_2$ esists if and only if $U \leq S_1$. It remains an open question whether, if we assume an oracle for the subtyping relation or choose a decidable fragment of the subtyping relation, the "*hiding* T" annotation can be inferred.

$$\frac{\begin{array}{c}\Gamma \vdash e \Rightarrow U \qquad \Gamma \vdash U \ll \mathit{Some}\,(A{\leq}S_1)\,S_2 \\ \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow T \qquad A \text{ not in } FV(T)\end{array}}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) \Rightarrow T} \qquad \text{(SYNTH-SOME-E)}$$

where, in this case, U must have the shape of an existential. But this rule is clearly incomplete, since it reports a typechecking failure whenever the representation type variable A appears free in the minimal type T of the body, whereas in the original system the rule of subsumption can always be used to promote such a T to some A-free supertype, such as (trivially) *Top*.

To improve this rule, we would need to find a way of promoting the minimal type of the body to its minimal A-free supertype, which would then be the minimal type of the whole open expression. Unfortunately, this is not always possible: we may encounter a type T with a free variable A such that T has two *incomparable* minimal A-free supertypes — that is, there may be two different A-free types $U1$ and $U2$ such that both are supertypes of T and there is no A-free supertype of T that is strictly smaller than either $U1$ or $U2$. The remainder of this section is devoted to exhibiting such a T.

2.2.1 Fact: Let Z be any closed type (say, *Top*), and let T be the type:

$$T \;=\; \neg(\mathit{All}(B{\leq}A)\,\mathit{All}(B'{\leq}A)\,A),$$

where we abbreviate

$$\neg S = S \rightarrow \mathit{Top}.$$

Then the set of A-free supertypes of T under the context $A{\leq}Z$ has no smallest member.

The proof of this fact requires a couple of technical lemmas.

2.2.2 Lemma: In system $F_{\leq}$ with existential types, the following properties hold:

1. If $\Gamma \vdash T_1{\rightarrow}T_2 \leq W \leq V_1{\rightarrow}V_2$ then W has the shape $W_1{\rightarrow}W_2$ with $\Gamma \vdash V_1 \leq W_1 \leq T_1$ and $\Gamma \vdash T_2 \leq W_2 \leq V_2$.
2. If $\Gamma \vdash \mathit{All}(B{\leq}T_1)\,T_2 \leq W \leq \mathit{All}(B{\leq}V_1)\,V_2$, then W has the shape $\mathit{All}(B{\leq}W_1)\,W_2$ with $\Gamma \vdash V_1 \leq W_1 \leq T_1$ and $\Gamma, B{\leq}W_1 \vdash T_2 \leq W_2$ and $\Gamma, B{\leq}V_1 \vdash W_2 \leq V_2$.
3. If $\Gamma \vdash \mathit{Some}\,(B{\leq}T_1)\,T_2 \leq W \leq \mathit{Some}\,(B{\leq}V_1)\,V_2$, then W has the shape $\mathit{Some}\,(B{\leq}W_1)\,W_2$ with $\Gamma \vdash T_1 \leq W_1 \leq V_1$ and $\Gamma, B{\leq}T_1 \vdash T_2 \leq W_2$ and $\Gamma, B{\leq}W_1 \vdash W_2 \leq V_2$.
4. If $\Gamma \vdash W \leq A$ (where A is a type variable) then W is a type variable.
5. If $\Gamma \vdash A \leq W$ (where A is a type variable) and $W \neq A$, then $\Gamma \vdash \Gamma(A) \leq W$.
6. If $\Gamma \vdash S \leq T \leq S$, then $S = T$.

Proof: The completeness of the syntax-directed subtyping rules for pure $F_{\leq}$ is proved in [5]. The extension of this result to the system with existential subtyping is straightforward, since the only new rule, SOME-SUB, is syntax-directed on both sides of the $\leq$.

The lemma then follows from the fact that the syntax-directed rules are complete for $F_{\leq}$ subtyping. For example, in part (1) we see from SUB-ARROW that $\Gamma \vdash T{\rightarrow}T' \leq W$ implies that either $W = \mathit{Top}$ or $W = U{\rightarrow}U'$, with $\Gamma \vdash U \leq T$ and $\Gamma \vdash T' \leq U'$, while $\Gamma \vdash W \leq V{\rightarrow}V'$ implies that either W is a variable or $W = U{\rightarrow}U'$, with $\Gamma \vdash V \leq U$ and $\Gamma \vdash U' \leq V'$. □

2.2.3 Lemma: If $\Gamma, A{\leq}U \vdash S \leq T$ and $\Gamma \vdash V \leq U$, then $\Gamma, A{\leq}V \vdash S \leq T$.

Proof: Routine induction on derivations. $\square$

Proof of Fact 2.2.1: Consider the types

$$\begin{array}{lcl} U_1 & = & \neg(All(B{\leq}Z)\,All(B'{\leq}Z)\,B) \\ U_2 & = & \neg(All(B{\leq}Z)\,All(B'{\leq}Z)\,B'). \end{array}$$

It can easily be verified that U_1 and U_2 are both supertypes of the type $T = \neg(All(B{\leq}A)\,All(B'{\leq}A)\,A)$ under the context $A{\leq}Z$. We now show that they are both minimal in the set of A-free supertypes of T, by arguing that if some A-free type U lies between T and U_1 then it is identical to U_1, and similarly for U_2.

Assume that V is an A-free type that lies between T and U_1, i.e. that

$$A{\leq}Z \vdash \neg(All(B{\leq}A)\,All(B'{\leq}A)\,A) \leq V \leq \neg(All(B{\leq}Z)\,All(B'{\leq}Z)\,B).$$

Then, by Lemma 2.2.2 (1) and (2), we have

$$\begin{array}{ll} A{\leq}Z \vdash All(B{\leq}Z)\,All(B'{\leq}Z)\,B \leq All(B{\leq}V_1)\,V' & , \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \leq All(B{\leq}A)\,All(B'{\leq}A)\,A & \end{array}$$

where

$$\begin{array}{l} V = \neg(All(B{\leq}V_1)\,V') \\ A{\leq}Z \vdash A \leq V_1 \leq Z \\ A{\leq}Z, B{\leq}V_1 \vdash (All(B'{\leq}Z)\,B) \leq V' \\ A{\leq}Z, B{\leq}A \vdash V' \leq (All(B'{\leq}A)\,A). \end{array}$$

By Lemma 2.2.3,

$$A{\leq}Z, B{\leq}A \vdash (All(B'{\leq}Z)\,B) \leq V' \leq (All(B'{\leq}A)\,A).$$

By Lemma 2.2.2(2), $V' = All(B'{\leq}V_2)\,V_3$, where

$$\begin{array}{l} A{\leq}Z, B{\leq}A \vdash A \leq V_2 \leq Z \\ A{\leq}Z, B{\leq}A, B'{\leq}V_2 \vdash B \leq V_3 \\ A{\leq}Z, B{\leq}A, B'{\leq}A \vdash V_3 \leq A. \end{array}$$

Now since V is A-free, Lemma 2.2.2 (5) gives

$$\begin{array}{l} A{\leq}Z \vdash Z \leq V_1 \leq Z \\ A{\leq}Z, B{\leq}A \vdash Z \leq V_2 \leq Z, \end{array}$$

hence Lemma 2.2.2 (6) gives $V_1 = V_2 = Z$, i.e.

$$\begin{array}{ll} (1) & A{\leq}Z, B{\leq}A, B'{\leq}Z \vdash B \leq V_3 \\ (2) & A{\leq}Z, B{\leq}A, B'{\leq}A \vdash V_3 \leq A. \end{array}$$

By (2) and Lemma 2.2.2 (4), V_3 must be a variable; in particular, since it cannot be A, it must be B or B'. By (1) and Lemma 2.2.2 (5), if V_3 were B', then $A{\leq}Z, B{\leq}A, B'{\leq}Z \vdash \Gamma(B) \leq B'$ would hold, which is not true. Hence, $V_3 = B$ and so $V = \neg(All(B{\leq}V_1)\,All(B'{\leq}V_2)\,V_3) = \neg(All(B{\leq}Z)\,All(B'{\leq}Z)\,B) = U_1$.

The case for U_2 is similar. $\square$

Before exhibiting an expression with two different minimal types we need some other easy lemmas.

2.2.4 Lemma [Strengthening]:

1. If $\Gamma, x{:}U, \Gamma' \vdash S \leq T$, then $\Gamma, \Gamma' \vdash S \leq T$.
2. If $\Gamma, A{\leq}U, \Gamma' \vdash S \leq T$ and A is not in $FV(\Gamma')$, $FV(S)$, or $FV(T)$, then $\Gamma, \Gamma' \vdash S \leq T$.

Proof: By induction on normal-form proofs of $\Gamma, x{:}U, \Gamma' \vdash S \leq T$ and $\Gamma, A{\leq}U, \Gamma' \vdash S \leq T$. $\square$

2.2.5 Lemma [Weakening]:

1. If $\Gamma, \Gamma' \vdash S \leq T$ and x and A do not appear in the judgement, then $\Gamma, x{:}U, \Gamma' \vdash S \leq T$ and $\Gamma, A{\leq}U, \Gamma' \vdash S \leq T$.
2. If $\Gamma, \Gamma' \vdash e : T$ and x and A do not appear in the judgement, then $\Gamma, x{:}U, \Gamma' \vdash e : T$ and $\Gamma, A{\leq}U, \Gamma' \vdash e : T$.

Proof: By induction on the proofs of $\Gamma, \Gamma' \vdash S \leq T$ and $\Gamma, \Gamma' \vdash e : T$. $\square$

2.2.6 Lemma: If $\Gamma, x{:}T, \Delta \vdash x : U$ then $\Gamma, x{:}T, \Delta \vdash T \leq U$.

Proof: Since SUBSUMPTION is the only non-syntax-directed typing rule, a proof of $\Gamma, x{:}T, \Delta \vdash x : U$ must end with an instance of the VAR rule followed by a finite number of instances of SUBSUMPTION. The result then follows from the reflexivity and transitivity of subtyping. $\square$

Now it is very easy to build an open expression with two incomparable minimal types. Let Z, T, U_1, and U_2 be as above; let Δ be the context

$$\Delta = y{:}\ \mathit{Some}\,(A{\leq}Z)\,T,$$

and consider the expression

$$e = \mathit{open}\ y\ \mathit{as}\ [A, x]\ \mathit{in}\ x.$$

It is easy to see that any A-free supertype of T is a type for e under Δ. We now argue that if $\Delta \vdash e : U$, then U is an A-free supertype of T under $A{\leq}Z$.

Assume we are given a proof of $\Delta \vdash e : U$, and note that this proof must end with an instance of SOME-E followed by some number of instances of SUBSUMPTION, since SUBSUMPTION is the only non-syntax-directed rule in the definition of the $F_{\leq}$ typing relation. That is, we must have a subderivation ending in

$$\cfrac{\cfrac{\cfrac{\Delta \vdash y : \mathit{Some}\,(A{\leq}Z')\,T' \qquad \Delta, A{\leq}Z', x{:}T' \vdash x : V \qquad A \text{ not in } FV(V)}{\Delta \vdash e : V \qquad \Delta \vdash V \leq V_1}}{\Delta \vdash e : V_1 \qquad \Delta \vdash V \leq V_2}}{\cfrac{\cdots}{\Delta \vdash e : U}}$$

and (by SUB-TRANS)

$$\Delta \vdash V \leq V_1 \leq V_2 \leq \cdots \leq U$$

where U and V are A-free (V is A-free by assumption, while U is well-defined in an environment Δ where A is not defined).

Now, by Lemmas 2.2.4 and 2.2.5, we have $A{\leq}Z \vdash V \leq U$.

By Lemma 2.2.6 and the subderivation $\Delta \vdash y : \mathit{Some}\,(A{\leq}Z')\,T'$, we have

$$\Delta \vdash \mathit{Some}\,(A{\leq}Z)\,T \leq \mathit{Some}\,(A{\leq}Z')\,T'.$$

Hence,

$\Delta \vdash Z \leq Z'$, $\quad \Delta, A{\leq}Z \vdash T \leq T'$	by syntax-directedness;
$\Delta, A{\leq}Z, x{:}T \vdash x : V$	by Lemma 2.2.3
	from $\Delta, A{\leq}Z', x{:}T' \vdash x : V$;
$\Delta, A{\leq}Z, x{:}T \vdash T \leq V$	by Lemma 2.2.6;
$A{\leq}Z \vdash T \leq V$	by Lemma 2.2.4;
$A{\leq}Z \vdash T \leq U$	by SUB-TRANS.

Thus we have proved that the types of e under Δ are all and only the A-free supertypes of T, and we have already shown that this set of types does not have a unique minimum.

2.3 Discussion

Although the common assertion that "existential types can be encoded in $F_{\leq}$" remains true from a semantic perspective, we have seen that it is problematic from the point of view of type checking.

One could take the results developed in the previous section as evidence that the best elimination rule for existential types in the pure presentation of $F_{\leq}$ is the tersest one that still admits a complete synthesis algorithm:

$$\frac{\Gamma \vdash e : \mathit{Some}\,(A{\leq}S_1)\,S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b : T \qquad A \text{ not in } FV(T)}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b{:}T) : T} \tag{COMPLETE-SOME-E}$$

But this rule is unsatisfactory in a practical sense: it achieves completeness at the expense of convenience. The programmer's intended type for an expression will almost always coincide with its minimal type, so the occurrence of A in the minimal type of b should probably be regarded as a programming mistake. We therefore lean toward the other possible conclusion: that it is unreasonable to expect typecheckers for languages based on $F_{\leq}$ to be complete with respect to the pure system. The best rule for existential elimination is Cardelli and Wegner's original one.

An intruiging question now arises: What sort of correspondence should exist between the "canonical" presentation of a type system and its implementation as a type synthesis algorithm, so that programmers can be presented with a clean, simple model of the types that will be assigned to their programs? The best possible correspondance is completeness: programmers can reason in terms of the pure presentation in full confidence that the type synthesis algorithm will arrive at identical conclusions. The semi-completeness of $F_{\leq}$ is almost as good: the type synthesis algorithm may diverge under pathological conditions, but normally it derives exactly the same judgements as the pure presentation.

$F_{\leq}$ with Cardelli and Wegner's formulation of existentials takes another step away from completeness: the type synthesis algorithm may incorrectly reject some programs that are well typed under the pure presentation, but this can happen only in situations where the programmer has probably made a mistake. This is similar to the treatment of *if-then-else* in languages with subsumption and with a *Top* type, such as the Galileo language [1]. According to the abstract presentation of the language, the expression *if true then 3 else false* belongs to the *Top* type, since both *3* and *false* are in *Top* by subsumption. However, the type synthesis algorithm requires that some relationship exists between the types of the two branches, and would reject the expression above, giving some completeness away in the hope of capturing more programming errors.

Yet another step towards synthesis algorithms that attempt only to be "complete enough" is provided by Cardelli's implementation of $F_{\leq}$ with partial type reconstruction based on mixed-prefix unification [2]. Ultimately, this process can go too far, resulting in type systems whose best definition is the behavior of an incomprehensible algorithm. But a large centerground remains to be explored.

Another question we have left open here is whether every term of $F_{\leq}$ with bounded existentials has a finite set of minimal types, and (if so) whether there is a reasonably efficient algorithm for synthesizing these sets.

3 Bounded Existentials in Kernel Fun

Cardelli and Wegner's original formulation of bounded quantification [4], actually used a simpler rule for deriving subtyping judgements involving bounded quantifiers:

$$\frac{\Gamma, A{\leq}S \vdash T \leq U}{\Gamma \vdash \mathit{All}\,(A{\leq}S)\,T \leq \mathit{All}\,(A{\leq}S)\,U} \qquad (\textsc{All-Sub-KFun})$$

Although this "equal-bounds rule" seems semantically less natural than the one used in subsequent presentations of $F_{\leq}$, it yields a system with much simpler syntactic properties, including decidable subtyping [8].

The addition of bounded existential quantifiers to this fragment — often called Kernel Fun — is accomplished by adjoining the same introduction and elimination rules as before, but restricting the subtyping rule by analogy with the one for the universal quantifier:

$$\frac{\Gamma, A{\leq}S \vdash T \leq U}{\Gamma \vdash \mathit{Some}\,(A{\leq}S)\,T \leq \mathit{Some}\,(A{\leq}S)\,U} \qquad (\textsc{Some-Sub-KFun})$$

(The combination of Kernel Fun with the more general rule for existentials given above, in addition to being rather unnatural, is already undecidable; see Appendix A.)

We can show that the converse of the fact proved in the first section holds for this extended Kernel Fun: we define a function $\mathit{raise}_{A,\Gamma}$ such that $\mathit{raise}_{A,\Gamma}(T)$ is the minimal A-free supertype of T in the context Γ. The idea is to replace all positive occurrences of A in T by its upper bound from Γ; negative and non-positive occurrences of A are eliminated by finding the smallest enclosing positive subphrase and promoting it to *Top*. The intuition behind this algorithm was suggested to us by Mariangiola Dezani.

The existence of this procedure allows us to prove that the usual typechecking algorithm for Kernel Fun can be extended by the following algorithmic formulation of Some-E to yield a complete algorithm for the combined system:

$$\frac{\Gamma \vdash e \Rightarrow U \qquad \Gamma \vdash U \ll \mathit{Some}\,(A{\leq}S_1)\,S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow V}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A, x]\ \mathit{in}\ b) \Rightarrow \mathit{raise}_{A,\Gamma}(V)} \qquad (\textsc{Alg-Some-E})$$

3.1 Typing Algorithm

3.1.1 Definition: The functions $raise_{A,\Gamma}$ and $lower_{A,\Gamma}$, mapping types to their least A-free supertype and their largest A-free subtype, respectively, are defined as follows. (To avoid clutter, we normally elide the subscripts A and Γ.) The two definitions have different side-conditions, since, whenever *lower* appears, we have to check whether it is defined, while *raise* is always defined, thanks to the presence of the *Top* type.

$$
\begin{array}{lcl}
raise(All\,(B{\leq}S)\,T) & = & \begin{cases} All\,(B{\leq}S)\ raise(T) \\ \quad \text{if } A \text{ does not occur in } S \\ Top \\ \quad \text{if } A \text{ occurs in } S \end{cases} \\
raise(Some\,(B{\leq}S)\,T) & = & \begin{cases} Some\,(B{\leq}S)\ raise(T) \\ \quad \text{if } A \text{ does not occur in } S \\ Top \\ \quad \text{if } A \text{ occurs in } S \end{cases} \\
raise(S{\rightarrow}T) & = & \begin{cases} lower(S) \rightarrow raise(T) \\ \quad \text{if } lower(S) \text{ is defined} \\ Top \\ \quad \text{if } lower(S) \text{ is undefined} \end{cases} \\
raise(A) & = & \Gamma(A) \\
raise(B) B \neq A & = & B \\
raise(Top) & = & Top
\end{array}
$$

$$
\begin{array}{lcl}
lower(All\,(B{\leq}S)\,T) & = & \begin{cases} All\,(B{\leq}S)\ lower(T) \\ \quad \text{if } lower(T) \text{ defined and } A \text{ not in } S \\ \text{undefined otherwise} \end{cases} \\
lower(Some\,(B{\leq}S)\,T) & = & \begin{cases} Some\,(B{\leq}S)\ lower(T) \\ \quad \text{if } lower(T) \text{ defined and } A \text{ not in } S \\ \text{undefined otherwise} \end{cases} \\
lower(S{\rightarrow}T) & = & \begin{cases} raise(S) \rightarrow lower(T) \\ \quad \text{if } lower(T) \text{ is defined} \\ \text{undefined} \\ \quad \text{if } lower(T) \text{ is undefined} \end{cases} \\
lower(A) & = & \text{undefined} \\
lower(B) B \neq A & = & B \\
lower(Top) & = & Top
\end{array}
$$

Note that, if A does not occur free in T, then $raise(T) = lower(T) = T$.

3.1.2 Remark:

1. If $lower(T_1{\rightarrow}T_2)$ is defined, then $lower(T_2)$ is defined.
2. If $lower(All\,(B{\leq}T_1)\,T_2)$ is defined, then $lower(T_2)$ is defined.
3. If $lower(Some\,(B{\leq}T_1)\,T_2)$ is defined, then $lower(T_2)$ is defined.

3.1.3 Proposition:

1. $\Gamma \vdash T \leq raise(T)$ for any Γ and T.

2. If $lower(T)$ is defined, then $\Gamma \vdash lower(T) \leq T$.

Proof sketch: Routine induction on T. $\square$

3.1.4 Lemma [Completeness]: The syntax-directed subtyping rules given in Appendix C are complete for Kernel Fun subtyping.

Proof sketch: Either by adapting the proof given for $F_{\leq}$ in [5], or by induction on a Kernel Fun proof of $\Gamma \vdash T \leq U$, the interesting case being when the last applied rule is transitivity. $\square$

3.1.5 Remark: Since the syntax-directed rules are complete for subtyping, we use the notation $\Gamma \vdash A \leq B$ for both syntax-directed and ordinary subtyping judgements.

3.1.6 Lemma:

1. If $\Gamma \vdash T \leq All(B{\leq}T_1)\,T_2$ then $\Gamma \vdash T \ll All(B{\leq}T_1)\,T_2'$ and $\Gamma, B{\leq}T_1 \vdash T_2' \leq T_2$.
2. If $\Gamma \vdash T \leq Some(B{\leq}T_1)\,T_2$ then $\Gamma \vdash T \ll Some(B{\leq}T_1)\,T_2'$ and $\Gamma, B{\leq}T_1 \vdash T_2' \leq T_2$.
3. If $\Gamma \vdash T \leq T_1{\rightarrow}T_2$ then $\Gamma \vdash T \ll T_1'{\rightarrow}T_2'$ and $\Gamma \vdash T_1 \leq T_1', \Gamma \vdash T_2' \leq T_2$.
4. $\Gamma \vdash V \leq A$ implies that V is a variable.

Proof: Like Lemma 2.2.2. $\square$

3.1.7 Lemma: Let Γ be a context and A a variable in $dom(\Gamma)$ such that A does not appear free in Γ. If $lower_{A,\Gamma}(T)$ is undefined, then T has no A-free subtypes — that is, $A \notin FV(V)$ implies $\Gamma \nvdash V \leq T$.

Proof: By induction on the definition of *lower*.

Case $T = All(B{\leq}T_1)\,T_2$ where either $lower(T_2)$ is undefined or A occurs in T_1 (1)
Assume, for a contradiction, that $\Gamma \vdash V \leq T$, where A does not appear free in either Γ or V. By Lemma 3.1.6, $\Gamma \vdash V \ll All(B{\leq}T_1)\,V_2$ with $\Gamma, B{\leq}T_1 \vdash V_2 \leq T_2$. But $\Gamma \vdash V \ll All(B{\leq}T_1)\,V_2$ means that either $V = All(B{\leq}T_1)\,V_2$ or that $Z \leq All(B{\leq}T_1)\,V_2$ is in Γ, for some Z; in both cases, A cannot occurr free in T_1; hence, by (1), $lower(T_2)$ is undefined. But now $\Gamma, B{\leq}T_1 \vdash V_2 \leq T_2$ contradicts the induction hypothesis.

Case $T = Some(B{\leq}T_1)\,T_2$ where either $lower(T_2)$ is undefined or A occurs in T_1
Similar.

Case $T = T_1{\rightarrow}T_2$ where $lower(T_2)$ is undefined
Assume, for a contradiction, that $\Gamma \vdash V \leq T$. By Lemma 3.1.6, $\Gamma \vdash V \ll V_1{\rightarrow}V_2$ with $\Gamma \vdash V_2 \leq T_2$. But $\Gamma \vdash V \ll V_1{\rightarrow}V_2$ means that either $V = V_1{\rightarrow}V_2$ or that $Z \leq V_1{\rightarrow}V_2$ is in Γ; in both cases, A cannot occurr free in V_2. But now $\Gamma \vdash V_2 \leq T_2$ contradicts the induction hypothesis.

Case $T = A$
Assume, for a contradiction, that $\Gamma \vdash V \leq T$. By Lemma 3.1.6, V must be a variable. But no variable different from A can be less than A, since A is not in $FV(\Gamma)$.

The other two cases in the definition of *lower* are irrelevant, since we assumed that $lower(T)$ is undefined. $\square$

3.1.8 Proposition: Assume that A is not in $FV(U)$ or $FV(\Gamma)$. Then

1. If $\Gamma \vdash T \leq U$, then $\Gamma \vdash \mathit{raise}(T) \leq U$.
2. If $\Gamma \vdash U \leq T$ then $\mathit{lower}(T)$ is defined and $\Gamma \vdash U \leq \mathit{lower}(T)$.

That is, $\mathit{raise}(T)$ is the minimal A-free supertype of T and, when it is defined, $\mathit{lower}(T)$ is the maximal A-free subtype of T.

Proof: Simultaneously, by induction on the size of normal-form subtyping derivations. In every case, we assume that A appears free in T, since otherwise the result is immediate; recall that the implication

$$\Gamma \vdash U \leq T \quad \Longrightarrow \quad \mathit{lower}(T) \text{ is defined}$$

has already been established (Lemma 3.1.7).

Case ARROW-SUB (1): $\Gamma \vdash T_1 \rightarrow T_2 \leq U_1 \rightarrow U_2$ with $\Gamma \vdash U_1 \leq T_1$ and $\Gamma \vdash T_2 \leq U_2$
By Lemma 3.1.7 and the first premise, $\mathit{lower}(T_1)$ is defined. So $\mathit{raise}(T) = \mathit{lower}(T_1) \rightarrow \mathit{raise}(T_2)$, and, by the induction hypothesis,

$$\Gamma \vdash U_1 \leq \mathit{lower}(T_1)$$
$$\Gamma \vdash \mathit{raise}(T_2) \leq U_2.$$

By ARROW-SUB,

$$\Gamma \vdash \mathit{lower}(T_1) \rightarrow \mathit{raise}(T_2) \leq U_1 \rightarrow U_2.$$

Case ARROW-SUB (2): $\Gamma \vdash U_1 \rightarrow U_2 \leq T_1 \rightarrow T_2$ with $\Gamma \vdash T_1 \leq U_1$ and $\Gamma \vdash U_2 \leq T_2$
Since $\mathit{lower}(T)$ is defined, $\mathit{lower}(T_2)$ is defined, $\mathit{lower}(T) = \mathit{raise}(T_1) \rightarrow \mathit{lower}(T_2)$, and, by the induction hypothesis, $\Gamma \vdash \mathit{raise}(T_1) \leq U_1$ and $\Gamma \vdash U_2 \leq \mathit{lower}(T_2)$. By ARROW-SUB, $\Gamma \vdash U_1 \rightarrow U_2 \leq \mathit{raise}(T_1) \rightarrow \mathit{lower}(T_2)$.

Case ALL-SUB (1): $\Gamma \vdash \mathit{All}(B{\leq}U_1)\, T_2 \leq \mathit{All}(B{\leq}U_1)\, U_2$ with $\Gamma, B{\leq}U_1 \vdash T_2 \leq U_2$
Since A does not occur in U_1, we have $\mathit{raise}(T) = \mathit{All}(B{\leq}U_1)\, \mathit{raise}(T_2)$, and, by the induction hypothesis, $\Gamma, B{\leq}U_1 \vdash \mathit{raise}(T_2) \leq U_2$. By ALL-SUB, $\Gamma \vdash \mathit{All}(B{\leq}U_1)\, \mathit{raise}(T_2) \leq \mathit{All}(B{\leq}U_1)\, U_2$.

Case ALL-SUB (2): $\Gamma \vdash \mathit{All}(B{\leq}U_1)\, U_2 \leq \mathit{All}(B{\leq}U_1)\, T_2$ with $\Gamma, B{\leq}U_1 \vdash U_2 \leq T_2$
Since $\mathit{lower}(T)$ is defined, $\mathit{lower}(T_2)$ is defined, $\mathit{lower}(T) = \mathit{All}(B{\leq}U_1)\, \mathit{lower}(T_2)$, and, by the induction hypothesis, $\Gamma, B{\leq}U_1 \vdash U_2 \leq \mathit{lower}(T_2)$. By ALL-SUB, $\Gamma \vdash \mathit{All}(B{\leq}U_1)\, U_2 \leq \mathit{All}(B{\leq}U_1)\, \mathit{lower}(T_2)$.

The two cases for *Some* are analogous.

Case ALL-VAR-SUB (1): $\Gamma \vdash A \leq U$ with $\Gamma \vdash \Gamma(A) \leq U$
Immediate, since $\mathit{raise}(A) = \Gamma(A)$.

Case ALL-VAR-SUB (2): $\Gamma \vdash B \leq T$ with $\Gamma \vdash \Gamma(B) \leq T$
By the induction hypothesis, $\Gamma \vdash \Gamma(B) \leq \mathit{lower}(T)$, hence, by VAR-SUB, $\Gamma \vdash B \leq \mathit{lower}(T)$.

Case TOP-SUB (1,2): $\Gamma \vdash W \leq \mathit{Top}$
Both $\Gamma \vdash \mathit{raise}(W) \leq \mathit{Top}$ and $\Gamma \vdash W \leq \mathit{lower}(\mathit{Top})$ are immediate.

Case ALG-REFL (1,2): $\Gamma \vdash B \leq B$
Both $\Gamma \vdash \mathit{raise}(B) \leq B$ and $\Gamma \vdash B \leq \mathit{lower}(B)$ are trivial, since B (being A-free) cannot be A. □

We can now show that Kernel Fun has both the minimal typing property and a straightforward algorithm for calculating minimal types.

Let Alg be the type system defined by the syntax-directed Kernel Fun rules, including the following rules for existentials (the other syntax-directed rules are given in Appendix C):

$$\frac{\Gamma \vdash T \leq S_1 \qquad \Gamma \vdash e \Rightarrow U \qquad \Gamma \vdash U \leq [T/A]S_2}{\Gamma \vdash (\mathit{pack}\ e\ \mathit{as}\ \mathit{Some}\,(A{\leq}S_1)\, S_2\ \mathit{hiding}\ T) \Rightarrow \mathit{Some}\,(A{\leq}S_1)\, S_2} \qquad (\text{Alg-Some-I})$$

$$\frac{\Gamma \vdash e \Rightarrow S \qquad \Gamma \vdash S \ll \mathit{Some}\,(A{\leq}S_1)\, S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow T}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) \Rightarrow \mathit{raise}_{A,(\Gamma,A\leq S_1)}(T)} \qquad (\text{Alg-Some-E})$$

It can easily be checked that the syntax-directed rules are sound — i.e. that if $\Gamma \vdash a \Rightarrow T$ then $\Gamma \vdash a : T$. In the remainder of this section, we prove that they are complete, i.e., that if $\Gamma \vdash a : T$ then there exists M such that $\Gamma \vdash a \Rightarrow M$ and $\Gamma \vdash M \leq T$.

To begin, we need some auxiliary definitions and a few technical lemmas about the subtype relation.

3.1.9 Definition: A context Γ' *refines* a context Γ (written $\Gamma' \leq \Gamma$) iff

1. $\Gamma' = \Gamma = ()$, or
2. $\Gamma' = \Delta', x{:}T'$, $\Gamma = \Delta, x{:}T$, $\Delta' \leq \Delta$, and $\Gamma \vdash T' \leq T$, or
3. $\Gamma' = \Delta', A{\leq}T$, $\Gamma = \Delta, A{\leq}T$, and $\Delta' \leq \Delta$.

3.1.10 Lemma: If $\Gamma' \leq \Gamma$ then $\Gamma' \vdash T \leq U$ iff $\Gamma \vdash T \leq U$.

Proof: Γ' and Γ differ only on term variables. □

3.1.11 Lemma [Weakening, Strengthening]: If A and x do not appear free in $\Gamma, \Gamma', e, U, U'$, then:

$$\begin{array}{lcll}
\Gamma, \Gamma' \vdash P & \Longrightarrow & \Gamma, x{:}T, \Gamma' \vdash P, \quad \Gamma, A{\leq}T, \Gamma' \vdash P & \textit{(weakening)} \\
\Gamma, x{:}T, \Gamma' \vdash P & \Longrightarrow & \Gamma, \Gamma' \vdash P & \textit{(strengthening)} \\
\Gamma, A{\leq}T, \Gamma' \vdash P & \Longrightarrow & \Gamma, \Gamma' \vdash P & \textit{(strengthening)}
\end{array}$$

where P is either $e : U$ or $U \leq U'$.

Proof: Standard induction. □

3.1.12 Lemma: If $\Gamma \vdash T \leq S_1$ and $\Gamma, A{\leq}S_1 \vdash S_2 \leq T_2$, then $\Gamma \vdash [T/A]S_2 \leq [T/A]T_2$.

Proof: We prove a stronger statement: $\Gamma \vdash T \leq S_1$ and $\Gamma, A{\leq}S_1, \Delta \vdash S_2 \leq T_2$ together imply $\Gamma, [T/A]\Delta \vdash [T/A]S_2 \leq [T/A]T_2$, where applying a substitution to a context is defined as follows:

$$\begin{array}{lcl}
[T/A]\emptyset & = & \emptyset \\
[T/A](\Gamma, x{:}U) & = & ([T/A]\Gamma, x{:}[T/A]U) \\
[T/A](\Gamma, B{\leq}U) & = & ([T/A]\Gamma, B{\leq}[T/A]U)
\end{array}$$

The proof proceeds by induction on the shape of a normal-form derivation of $\Gamma, A{\leq}S_1, \Delta \vdash S_2 \leq T_2$ and by cases on the last rule applied.

Case Alg-Var-Sub (A): $\Gamma, A{\leq}S_1, \Delta \vdash A \leq T_2$ with $\Gamma, A{\leq}S_1, \Delta \vdash S_1 \leq T_2$
By induction, $\Gamma, [T/A]\Delta \vdash S_1 \leq [T/A]T_2$; by hypothesis $\Gamma \vdash T \leq S_1$; by weakening (3.1.11) $\Gamma, [T/A]\Delta \vdash T \leq S_1$; finally, by transitivity, $\Gamma, [T/A]\Delta \vdash T (= [T/A]A) \leq [T/A]T_2$.

Case ALG-VAR-SUB ($B \neq A$): $\Gamma, A{\le}S_1, \Delta \vdash B \le T_2$ with $\Gamma, A{\le}S_1, \Delta \vdash (\Gamma,\Delta)(B) \le T_2$ (where $(\Gamma,\Delta)(B)$ means "the bound of B in the environment (Γ,Δ)"). By induction, $\Gamma, [T/A]\Delta \vdash [T/A]((\Gamma,\Delta)(B)) \le [T/A]T_2$; since A does not occur in Γ, we have $\Gamma, [T/A]\Delta \vdash (\Gamma, [T/A]\Delta)(B) \le [T/A]T_2$. Finally, by ALG-VAR-SUB, $\Gamma, [T/A]\Delta \vdash B (= [T/A]B) \le [T/A]T_2$.

All the other cases are proved by straightforward use of the induction hypothesis. □

The completeness of the type synthesis algorithm can now be established as follows:

3.1.13 Proposition: If $\Gamma \vdash e : T$, then there exists T' such that $\Gamma \vdash e \Rightarrow T'$ and $\Gamma \vdash T' \le T$.

Proof: We prove the following stronger property

If $\Gamma \vdash e : T$ and $\Gamma' \le \Gamma$, then there exists T' such that $\Gamma' \vdash e \Rightarrow T'$ and $\Gamma \vdash T' \le T$.

by induction on the size of the given typing derivation, with a case analysis on the last rule applied. (The most interesting cases are SOME-E and ARROW-E.)

Case SOME-E:

$$\frac{\Gamma \vdash e : \mathit{Some}\,(A{\le}S_1)\,S_2 \qquad \Gamma, A{\le}S_1, x{:}S_2 \vdash b : T \qquad A \text{ not in } FV(T)}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) : T}$$

By the induction hypothesis and the first premise of SOME-E, $\Gamma' \vdash e \Rightarrow U$ with $\Gamma' \vdash U \le \mathit{Some}\,(A{\le}S_1)\,S_2$. By Lemma 3.1.6(2), $\Gamma' \vdash U \ll \mathit{Some}\,(A{\le}S_1)\,S_2'$ with $\Gamma', A{\le}S_1 \vdash S_2' \le S_2$. Applying the induction hypothesis to the second premise of SOME-E yields $\Gamma', A{\le}S_1, x{:}S_2' \vdash b \Rightarrow T'$ with $\Gamma, A{\le}S_1, x{:}S_2 \vdash T' \le T$, i.e. (by strengthening), $\Gamma, A{\le}S_1 \vdash T' \le T$. By ALG-SOME-E, $\Gamma' \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) \Rightarrow \mathit{raise}_{A,(\Gamma',A\le S_1)}(T')$. Since Γ differs from Γ' on term variables only, $\mathit{raise}_{A,(\Gamma',A\le S_1)}(T') = \mathit{raise}_{A,(\Gamma,A\le S_1)}(T')$. Now, since T is an A-free supertype of T' in the context $(\Gamma, A{\le}S_1)$, Proposition 3.1.8 gives $\Gamma, A{\le}S_1 \vdash \mathit{raise}_{A,(\Gamma,A\le S_1)}(T') \le T$. By strengthening (Lemma 3.1.11(2), $\Gamma \vdash \mathit{raise}_{A,(\Gamma',A\le S_1)}(T') \le T$.

Case ALL-E:

$$\frac{\Gamma \vdash e : \mathit{All}\,(A{\le}S_1)\,S_2 \qquad \Gamma \vdash T \le S_1}{\Gamma \vdash e\;T : [T/A]S_2}$$

By the induction hypothesis, $\Gamma' \vdash e \Rightarrow U$ with $\Gamma \vdash U \le \mathit{All}(A{\le}S_1)\,S_2$. By Lemma 3.1.6(1), $\Gamma \vdash U \ll \mathit{All}(A{\le}S_1)\,S_2'$ with $\Gamma, A{\le}S_1 \vdash S_2' \le S_2$. Applying ALG-ALL-E to the induction hypothesis, we obtain $\Gamma' \vdash e[T] \Rightarrow [T/A]S_2'$. By Lemma 3.1.12, since $\Gamma, A{\le}S_1 \vdash S_2' \le S_2$ and $\Gamma \vdash T \le S_1$, we have $\Gamma \vdash [T/A]S_2' \le [T/A]S_2$.

Case ARROW-E:

$$\frac{\Gamma \vdash f : T_1 {\to} T_2 \qquad \Gamma \vdash a : T_1}{\Gamma \vdash f\,a : T_2}$$

By the induction hypothesis, $\begin{array}{l}\Gamma' \vdash f \Rightarrow T' \\ \Gamma' \vdash a \Rightarrow T_1'\end{array}$ with $\begin{array}{l}\Gamma \vdash T' \le T_1 {\to} T_2 \\ \Gamma \vdash T_1' \le T_1.\end{array}$ By Lemma 3.1.6(3), $\Gamma \vdash T' \ll T_1'' {\to} T_2''$ with $\begin{array}{l}\Gamma \vdash T_1 \le T_1'' \\ \Gamma \vdash T_2'' \le T_2.\end{array}$ By TRANS, $\Gamma \vdash T_1' \le T_1''$, i.e., by Lemma 3.1.10, $\Gamma' \vdash T_1' \le T_1''$. By ALG-ARROW-E, $\Gamma' \vdash f\,a \Rightarrow T_2''$.

Case Some-I:

$$\frac{\Gamma \vdash T \leq S_1 \qquad \Gamma \vdash e : [T/A]S_2}{\Gamma \vdash (\textit{pack } e \textit{ as Some}\,(A{\leq}S_1)\; S_2 \textit{ hiding } T) : \textit{Some}\,(A{\leq}S_1)\; S_2}$$

By the induction hypothesis, $\Gamma' \vdash e \Rightarrow U$ with $\Gamma \vdash U \leq [T/A]S_2$. The result follows by Lemma 3.1.10 and Alg-Some-I.

Case All-I:

$$\frac{\Gamma, A{\leq}S_1 \vdash e : S_2}{\Gamma \vdash \textit{fun}\,(A{\leq}S_1)\; e : \textit{All}\,(A{\leq}S_1)\; S_2}$$

By the induction hypothesis, $\Gamma', A{\leq}S_1 \vdash e \Rightarrow S_2'$ with $\Gamma, A{\leq}S_1 \vdash S_2' \leq S_2$. By Alg-All-I, $\Gamma' \vdash \textit{fun}\,(A{\leq}S_1)\, e \Rightarrow \textit{All}\,(A{\leq}S_1)\, S_2'$, and by Lemma 3.1.10 and All-Sub, $\Gamma \vdash \textit{All}\,(A{\leq}S_1)\, S_2' \leq \textit{All}\,(A{\leq}S_1)\, S_2$.

Case Arrow-I:

$$\frac{\Gamma, x{:}S_1 \vdash e : S_2}{\Gamma \vdash \textit{fun}\,(x{:}S_1)\; e : S_1{\rightarrow}S_2}$$

By the induction hypothesis, $\Gamma', x{:}S_1 \vdash e \Rightarrow S_2'$ with $\Gamma, x{:}S_1 \vdash S_2' \leq S_2$, i.e., by Lemma 2.2.4(1), $\Gamma \vdash S_2' \leq S_2$. By Refl and Arrow-Sub, $\Gamma \vdash S_1{\rightarrow}S_2' \leq S_1{\rightarrow}S_2$. By Alg-All-I, $\Gamma' \vdash \textit{fun}\,(x{:}S_1)\; e \Rightarrow S_1{\rightarrow}S_2'$.

Case Var:

$$\Gamma \vdash x : \Gamma(x)$$

By Alg-Var, $\Gamma' \vdash x \Rightarrow \Gamma'(x)$. Let $\Gamma = \Delta, x{:}\Gamma(x), I$ and let $\Gamma' = \Delta', x{:}\Gamma'(x), I'$. By the definition of the refinement relation, $\Delta, x{:}\Gamma(x) \vdash \Gamma'(x) \leq \Gamma(x)$. The thesis $\Gamma \vdash \Gamma'(x) \leq \Gamma(x)$ follows by weakening.

Case Subsumption:

$$\frac{\Gamma \vdash e : U \qquad \Gamma \vdash U \leq T}{\Gamma \vdash e : T}$$

By the induction hypothesis, $\Gamma' \vdash e \Rightarrow U'$, where $\Gamma \vdash U' \leq U$. By Trans, $\Gamma \vdash U' \leq T$. □

3.2 Discussion

This proof of completeness is much easier than the one given in [5] to show existence of a minimum type and of a set of syntax-directed rules for $F_{\leq}$. The proof in [5] was based on the definition of a confluent and normalizing rewriting system of $F_{\leq}$ subtyping and typing proofs. We were able to adopt a simpler approach here for two reasons:

1. In [5] the term rewriting approach allows solving the coherence problem by proving that a semantic interpretation is coherent iff it satisfies the equations underlying the rewrite rule; here we are not attempting to address this problem.

2. In [5] the main technical problem to be solved was completeness of the *subtyping* rules. Since the addition of bounded existentials does not complicate the subtype relation in any essential way, we are able here to appeal to [5] for the necessary results about subtyping and concentrate on the much easier problem of analyzing the typing relation.

We have shown here that, in $F_{\leq}$, it is not possible to infer a minimum result type for an *open* expression when we move from the encoded *open* operator to the primitive one, while it is possible in Kernel Fun. The same is true, for simpler reasons, with the *if then else* operator. If we encode the boolean type as $All(A{\leq}Top)\, A{\rightarrow}A{\rightarrow}A$, then we can encode *if* b *then* e_1 *else* $e_2 : T$ as $b\ T\ e_1\ e_2$. However, if the result type T is omitted, we cannot in general infer it in $F_{\leq}$. More precisely, given $\Gamma,\ b,\ e_1,\ e_2$, there does not exist, in general, a minimum result type, i.e. a minimum type U such that $b\ U\ e_1\ e_2$ is well typed in Γ, and this is a direct consequence of the fact that not every pair of types with a common supertype has a *minimum* common supertype $F_{\leq}$ (see [7]). By contrast, in Kernel Fun every pair of types with a common supertype has a minimum common supertype, and this property implies that, given a typable *if* b *then* e_1 *else* e_2 expression, it is possible to infer its minimum type.

This result — that Kernel Fun enriched with the "terse formulation" of existential types does not lose the minimum type property — further extends the set of properties enjoyed by Kernel Fun but not by $F_{\leq}$. (The most important among these is the decidability of subtyping and typing, but see also [6].)

Acknowledgements

We are grateful for discussions with Mariangiola Dezani, Bob Harper, and the Edinburgh LFCS ML Club. The careful reading of two anonymous referees improved the paper in many places.

Work by the first author has been partially supported by E.E.C., Esprit Basic Research Action 6309 FIDE2, and by "Ministero dell'Università e della Ricerca Scientifica e Tecnologica." The second author has been supported by the Lab for Foundations of Computer Science, University of Edinburgh, by Projet Formel, INRIA, and by the Computer Laboratory, University of Cambridge under EPSRC grant GR/K 38403.

A Undecidability of Kernel Fun with Mixed-bound Existentials

Since the known proof of undecidability for $F_{\leq}$ uses the subtyping rule for universal quantifiers in an essential way, one might wonder whether decidability is obtained by adopting the equal-bound subtyping rule for univeral types, but taking existential types as primitive and allowing them to be compared using the mixed bound rule. In this appendix we remark that this is not the case, since Kernel Fun extended with mixed bound is already undecidable. This can be proved by encoding the $F_{\leq}$ subtyping problem as an extended Kernel Fun subtyping problem.

A.1 Definition: The encoding $[\![-]\!]$ from pure $F_{\leq}$ to Kernel Fun extended with mixed-bound existential types is defined as follows:

$$
\begin{array}{lcl}
[\![A]\!] & = & A \\
[\![\mathit{Top}]\!] & = & \mathit{Top} \\
[\![\mathit{All}(A{\leq}T)\,U]\!] & = & \neg(\mathit{Some}\,(A{\leq}[\![T]\!])\,\neg[\![U]\!]) \\
[\![T{\to}U]\!] & = & [\![T]\!]{\to}[\![U]\!] \\
\\
[\![A_1 \leq T_1, ..., A_n \leq T_n]\!] & = & A_1 \leq [\![T_1]\!], \ldots, A_n \leq [\![T_n]\!] \\
\\
[\![\Gamma \vdash T \leq U]\!] & = & [\![\Gamma]\!] \vdash [\![T]\!] \leq [\![U]\!]
\end{array}
$$

A.2 Proposition: $\Gamma \vdash T \leq U$ is provable in $F_{\leq}$ iff $[\![\Gamma \vdash T \leq U]\!]$ is provable in Kernel Fun extended with mixed bound subtyping.

Proof sketch: By induction on the size of the normal-form proof of $\Gamma \vdash T \leq U$, in one direction, and of $[\![\Gamma \vdash T \leq U]\!]$, in the other one. The crucial case is when $T = \mathit{All}(A{\leq}T_1)\,T_2$ and $U = \mathit{All}(A{\leq}U_1)\,U_2$. We reason as follows:

$$
\begin{array}{ll}
\text{let} & T = \mathit{All}(A{\leq}T_1)\,T_2 \text{ and } U = \mathit{All}(A{\leq}U_1)\,U_2. \\
\Gamma \vdash T \leq U & \\
\text{iff (All)} & \Gamma \vdash U_1 \leq T_1 \text{ and } \Gamma, A{\leq}U_1 \vdash T_2 \leq U_2 \\
\text{iff (Ind)} & [\![\Gamma]\!] \vdash [\![U_1]\!] \leq [\![T_1]\!] \text{ and } [\![\Gamma]\!], A < [\![U_1]\!] \vdash [\![T_2]\!] \leq [\![U_2]\!] \\
\text{iff } (\neg) & [\![\Gamma]\!] \vdash [\![U_1]\!] \leq [\![T_1]\!] \text{ and } [\![\Gamma]\!], A < [\![U_1]\!] \vdash \neg[\![U_2]\!] \leq \neg[\![T_2]\!] \\
\text{iff (Some)} & [\![\Gamma]\!] \vdash \mathit{Some}\,(A{\leq}[\![U_1]\!])\,\neg[\![U_2]\!] \leq \mathit{Some}\,(A{\leq}[\![T_1]\!])\,\neg[\![T_2]\!] \\
\text{iff } (\neg) & [\![\Gamma]\!] \vdash \neg\mathit{Some}\,(A{\leq}[\![T_1]\!])\,\neg[\![T_2]\!] \leq \neg\mathit{Some}\,(A{\leq}[\![U_1]\!])\,\neg[\![U_2]\!] \\
\stackrel{\text{def}}{=} & [\![\Gamma]\!] \vdash [\![\mathit{All}(A{\leq}T_1)\,T_2]\!] \leq [\![\mathit{All}(A{\leq}U_1)\,U_2]\!]
\end{array}
$$

The other cases are simpler. □

B Summary of $F_{\leq}$

Syntax:

Types	T	$::=$	$A \mid T{\to}T \mid \mathit{All}\,(A{\leq}T)\,T \mid \mathit{Some}\,(A{\leq}T)\,T \mid \mathit{Top}$
Terms	e	$::=$	$x \mid \mathit{fun}\,(x{:}T)\,e \mid e_1\ e_2 \mid \mathit{fun}\,(T)\,e \mid e\ T$
			$\mid \mathit{pack}\ e\ \mathit{as}\ \mathit{Some}\,(A{\leq}T)\,T_1\ \mathit{hiding}\ T_2$
			$\mid \mathit{open}\ e_1\ \mathit{as}\ [A, x]\ \mathit{in}\ e_2$
Envs	Γ	$::=$	$() \mid \Gamma, x{:}T \mid \Gamma, A{\leq}T$
Judgements	j	$::=$	$\Gamma \vdash e : T \mid \Gamma \vdash T \leq T$

Subtyping Rules:

$$\Gamma \vdash T \leq T \qquad (\textsc{Refl})$$

$$\frac{\Gamma \vdash S \leq T \qquad \Gamma \vdash T \leq U}{\Gamma \vdash S \leq U} \qquad (\textsc{Trans})$$

$$\Gamma \vdash A \leq \Gamma(A) \qquad (\textsc{Var-Sub})$$

$$\Gamma \vdash T \leq \mathit{Top} \qquad (\textsc{Top-Sub})$$

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

$$\frac{\Gamma \vdash T_1 \leq S_1 \qquad \Gamma, A{\leq}T_1 \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{All}\,(A{\leq}S_1)\,S_2 \leq \mathit{All}\,(A{\leq}T_1)\,T_2} \qquad (\textsc{All-Sub})$$

$$\frac{\Gamma \vdash S_1 \leq T_1 \qquad \Gamma, A{\leq}S_1 \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{Some}\,(A{\leq}S_1)\,S_2 \leq \mathit{Some}\,(A{\leq}T_1)\,T_2} \qquad (\textsc{Some-Sub})$$

Typing Rules:

$$\Gamma \vdash x : \Gamma(x) \qquad (\textsc{Var})$$

$$\frac{\Gamma \vdash e : U \qquad \Gamma \vdash U \leq T}{\Gamma \vdash e : T} \qquad (\textsc{Subsumption})$$

$$\frac{\Gamma, x{:}S_1 \vdash e : S_2}{\Gamma \vdash \mathit{fun}\,(x{:}S_1)\,e : S_1{\to}S_2} \qquad (\textsc{Arrow-I})$$

$$\frac{\Gamma \vdash f : T_1{\to}T_2 \qquad \Gamma \vdash a : T_1}{\Gamma \vdash f\,a : T_2} \qquad (\textsc{Arrow-E})$$

$$\frac{\Gamma, A{\leq}S_1 \vdash e : S_2}{\Gamma \vdash \mathit{fun}\,(A{\leq}S_1)\,e : \mathit{All}\,(A{\leq}S_1)\,S_2} \qquad (\textsc{All-I})$$

$$\frac{\Gamma \vdash e : \mathit{All}\,(A{\leq}S_1)\; S_2 \qquad \Gamma \vdash T \leq S_1}{\Gamma \vdash e[T] : [T/A]S_2} \tag{All-E}$$

$$\frac{\Gamma \vdash T \leq S_1 \qquad \Gamma \vdash e : [T/A]S_2}{\Gamma \vdash (\mathit{pack}\ e\ \ \mathit{as}\ \mathit{Some}\,(A{\leq}S_1)\; S_2\ \ \mathit{hiding}\ T) : \mathit{Some}\,(A{\leq}S_1)\; S_2} \tag{Some-I}$$

$$\frac{\Gamma \vdash e : \mathit{Some}\,(A{\leq}S_1)\; S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b : T \qquad A \text{ not in } FV(T)}{\Gamma \vdash (\mathit{open}\ e\ \ \mathit{as}\ [A, x]\ \ \mathit{in}\ b) : T} \tag{Some-E}$$

Restricted Quantifier Subtyping Rules for Kernel Fun:

$$\frac{\Gamma, A{\leq}U \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{All}\,(A{\leq}U)\; S_2 \leq \mathit{All}\,(A{\leq}U)\; T_2} \tag{All-Sub-KFun}$$

$$\frac{\Gamma, A{\leq}U \vdash S_2 \leq T_2}{\Gamma \vdash \mathit{Some}\,(A{\leq}U)\; S_2 \leq \mathit{Some}\,(A{\leq}U)\; T_2} \tag{Some-Sub-KFun}$$

C Type Synthesis Algorithm for $F_{\leq}$ and Kernel Fun

Judgement Forms:

$\Gamma \vdash S \ll T$	T is the minimal non-variable supertype of S
$\Gamma \vdash e \Rightarrow T$	T is the minimal type of e
$\Gamma \vdash S \leq T$	S is a subtype of T

Promotion Rules:

$$\frac{\Gamma \vdash \Gamma(A) \ll T}{\Gamma \vdash A \ll T} \qquad \text{(PROMOTE-VAR)}$$

$$\frac{T \text{ not a type variable}}{\Gamma \vdash T \ll T} \qquad \text{(PROMOTE-NONVAR)}$$

Typing Rules ($F_{\leq}$ and Kernel Fun):

$$\Gamma \vdash x \Rightarrow \Gamma(x) \qquad \text{(ALG-VAR)}$$

$$\frac{\Gamma, x{:}S_1 \vdash e \Rightarrow S_2}{\Gamma \vdash \mathit{fun}\,(x{:}S_1)\ e \Rightarrow S_1 \rightarrow S_2} \qquad \text{(ALG-ARROW-I)}$$

$$\frac{\Gamma \vdash f \Rightarrow T \qquad \Gamma \vdash T \ll T_1 \rightarrow T_2 \qquad \Gamma \vdash a \Rightarrow T_1' \qquad \Gamma \vdash T_1' \leq T_1}{\Gamma \vdash f\,a \Rightarrow T_2} \qquad \text{(ALG-ARROW-E)}$$

$$\frac{\Gamma, A{\leq}S_1 \vdash e \Rightarrow S_2}{\Gamma \vdash \mathit{fun}\,(A{\leq}S_1)\ e \Rightarrow \mathit{All}\,(A{\leq}S_1)\ S_2} \qquad \text{(ALG-ALL-I)}$$

$$\frac{\Gamma \vdash e \Rightarrow S \qquad \Gamma \vdash S \ll \mathit{All}\,(A{\leq}S_1)\ S_2 \qquad \Gamma \vdash T \leq S_1}{\Gamma \vdash e[T] \Rightarrow [T/A]S_2} \qquad \text{(ALG-ALL-E)}$$

$$\frac{\Gamma \vdash T \leq S_1 \qquad \Gamma \vdash e \Rightarrow U \qquad \Gamma \vdash U \leq [T/A]S_2}{\Gamma \vdash (\mathit{pack}\ e\ \mathit{as}\ \mathit{Some}\,(A{\leq}S_1)\ S_2\ \mathit{hiding}\ T) \Rightarrow \mathit{Some}\,(A{\leq}S_1)\ S_2} \qquad \text{(ALG-SOME-I)}$$

Existential elimination rule for Kernel Fun:

$$\frac{\Gamma \vdash e \Rightarrow S \qquad \Gamma \vdash S \ll \mathit{Some}\,(A{\leq}S_1)\ S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow T}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) \Rightarrow \mathit{raise}_{A,(\Gamma,A\leq S_1)}(T)} \qquad \text{(ALG-SOME-E)}$$

Choices for the existential elimination rule for $F_{\leq}$:

$$\frac{\begin{array}{c}\Gamma \vdash e \Rightarrow S \qquad \Gamma \vdash S \ll \mathit{Some}\,(A{\leq}S_1)\ S_2 \\ \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow T \qquad A \text{ not in } FV(T)\end{array}}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) \Rightarrow T} \qquad \text{(SYNTH-SOME-E)}$$

$$\frac{\Gamma \vdash e \Rightarrow S \qquad \Gamma \vdash S \ll \mathit{Some}\,(A{\leq}S_1)\ S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow T}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b) \Rightarrow \mathit{raise\text{-}Fsub}_{A,(\Gamma,A\leq S_1)}(T)} \qquad \text{(RAISE-SOME-E)}$$

where $\mathit{raise\text{-}Fsub}_{A,\Gamma}(T)$ is an $F_{\leq}$ version of the *raise* operator, which returns just one of the minimal A-free supertypes of T.

$$\frac{\Gamma \vdash e \Rightarrow S \qquad \Gamma \vdash S \ll \mathit{Some}\,(A{\leq}S_1)\,S_2 \qquad \Gamma, A{\leq}S_1, x{:}S_2 \vdash b \Rightarrow U \qquad \Gamma \vdash U \leq T \qquad A \text{ not in } FV(T)}{\Gamma \vdash (\mathit{open}\ e\ \mathit{as}\ [A,x]\ \mathit{in}\ b{:}T) \Rightarrow T} \tag{Alg-Der-Some-E}$$

Subtyping rules (for both $F_{\leq}$ and Kernel Fun):

The rules for *Top*, arrow, and the two quantifiers are identical to those in the original presentation. The rules Refl, Var, and Trans become:

$$\Gamma \vdash A \leq A \tag{Alg-Refl}$$

$$\frac{\Gamma \vdash \Gamma(A) \leq T \qquad (T \neq \mathit{Top},\ T \neq A)}{\Gamma \vdash A \leq T} \tag{Alg-Var-Sub}$$

References

[1] A. Albano, L. Cardelli, and R. Orsini. Galileo: A strongly typed, interactive conceptual language. *ACM Transactions on Database Systems*, 10(2):230–260, 1985.

[2] Luca Cardelli. An implementation of $F_{<:}$. Research report 97, DEC Systems Research Center, February 1993.

[3] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. 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).

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

[5] 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. Also in Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

[6] G. Ghelli. Recursive types are not conservative over $F_{\leq}$. In M. Bezen and J.F. Groote, editors, *Proceedings of thethe International Conference on Typed Lambda Calculi and Applications (TLCA), Utrecht, The Netherlands*, number 664 in Lecture Notes in Computer Science, pages 146–162, Berlin, March 1993. Springer Verlag.

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

[8] Giorgio Ghelli. Divergence of $F_{\leq}$ type checking. *Theoretical Computer Science*, 139(1,2):131–162, 1995.

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

[10] John Mitchell and Gordon Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3), July 1988.

[11] 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.