An extension of system F with subtyping

Luca Cardelli[1] *Simone Martini*[2] *John C. Mitchell*[3] *Andre Scedrov*[4]

Abstract

System F is a well-known typed λ-calculus with polymorphic types, which provides a basis for polymorphic programming languages. We study an extension of F, called $F_{<:}$, that combines parametric polymorphism with subtyping.

The main focus of the paper is the equational theory of $F_{<:}$, which is related to PER models and the notion of parametricity. We study some categorical properties of the theory when restricted to closed terms, including interesting categorical isomorphisms. We also investigate proof-theoretical properties, such as the conservativity of typing judgments with respect to F.

We demonstrate by a set of examples how a range of constructs may be encoded in $F_{<:}$. These include record operations and subtyping hierarchies that are related to features of object-oriented languages.

1. Introduction

System F [16] [21] is a well-known typed λ-calculus with polymorphic types, which provides a basis for polymorphic programming languages. We study an extension of F that combines parametric polymorphism [24] with subtyping. We call this language $F_{<:}$, where $<:$ is our symbol for the subtype relation. $F_{<:}$ is closely related to the language $F_{\leq}$ identified by Curien, and used by Curien and Ghelli primarily as a test case for certain mathematical techniques [15] [10]. $F_{\leq}$ is, in turn, a fragment of the language *Fun* of [9]. In spite of $F_{<:}$'s apparent minimality, it has become apparent that a range of constructs may be encoded in it (or in $F_{\leq}$); these include many of the record operations and subtyping features of [5], [8], and related work, which are connected to operations used in object-oriented programming. We illustrate some of the power of $F_{<:}$ in Section 3; see also [6].

In addition to the connections with object-oriented programming, we have found that the study of $F_{<:}$ raises semantic questions of independent interest. A major concern in this paper is an equational theory for $F_{<:}$ terms. The equational axioms for most systems of typed λ-calculi arise

[1]*Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave, Palo Alto CA 94301.*
[2]*Dipartimento di Informatica, Università di Pisa, Corso Italia 40, I-56125 Pisa, Italy.* *This author is partially supported by the CNR-Stanford collaboration grant 89.00002.26.*
[3]*Computer Science Department, Stanford University, Stanford CA 94305.*
[4]*Department of Mathematics, University of Pennsylvania, 209 South 33rd Street, Philadelphia, PA 19104-6395.*
This author is partially supported by the ONR Contract N00014-88-K-0635 and by NSF Grant CCR-87-05596.

Simone Martini and Andre Scedrov would like to thank John C. Mitchell, the Computer Science Department, and the Center for the Study of Language and Information at Stanford University for their hospitality during those authors' extended stay in 1989-1990, when much of this research was done.

Luca Cardelli and Simone Martini would like to thank Pierre-Louis Curien, Giorgio Ghelli, and Giuseppe Longo for many stimulating discussions related to this work. In particular, Curien helped in the early proof of $Top \sim \exists(X)X$.

naturally as a consequence of characterizing type connectives by adjoint situations (for example). In addition, it is often the case that provable equality may be captured by a reduction system obtained by orienting the equational axioms in a straightforward way. However, both of these properties appear to fail for $F_{<:}$. A simple example illustrates some of the basic issues.

A straightforward polymorphic type is $\forall(A)A{\to}A{\to}A$, which is commonly referred to as *Bool* since in system F and related systems there are two definable elements of this type. These elements are written as the following normal forms:

$$\mathit{true} \triangleq \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}A)\,x$$
$$\mathit{false} \triangleq \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}A)\,y$$

In $F_{<:}$, however, there are two additional normal forms of type *Bool*. These arise because we have a maximal type *Top*, which has all other types as subtypes. The main idea behind the additional terms is that we can change the type of any argument not used in the body of a term to *Top*, and still have a term of the same type (by antimonotonicity of the left operand of $\to$ with respect to $<:$). This gives us the following two normal forms of type *Bool*.

$$\mathit{true}' \triangleq \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}\mathit{Top})\,x$$
$$\mathit{false}' \triangleq \lambda(A)\,\lambda(x{:}\mathit{Top})\,\lambda(y{:}A)\,y$$

However, *true* and *true'* are completely equivalent terms when considered at type *Bool*. Specifically, for any type A, the terms *true*(A) and *true'*(A) define extensionally equal functions of type $A{\to}A{\to}A$. Put proof-theoretically, if we take any term a containing *true* with the property that when reducing a to normal form we apply each occurrence of *true* to two arguments, then we may replace any or all occurrences of *true* by *true'* and obtain a provably equal term. For this reason, it seems natural to consider *true* = *true'*, and similarly *false* = *false'*, even though these terms have different normal forms. When we add these two equations to our theory, we restore the pleasing property that *Bool* contains precisely two equivalence classes of normal forms.

While our initial examination of the equational theory of $F_{<:}$ was motivated by a vague intuition about observable properties of normal forms, our primary guide is the PER semantics of polymorphic λ-calculus with subtyping [4] [7] [15] [23]. One relevant characteristic of PER models is the *parametric* behavior of polymorphic functions. Specifically, since polymorphic functions operate independently of their type parameter, they may be considered equivalent at all their type instances. In $F_{<:}$ we can state a consequence of this notion of parametricity, namely that whenever the two type instances have a common supertype, the terms will be equal when considered as elements of that supertype (see the rule *(Eq appl2)* in section 2.2). Hence the syntax of $F_{<:}$ can state, at least to some extent, the semantic notion of parametricity investigated in [22], [14], and [1]. A general principle we have followed is to adopt axioms that express parametricity properties satisfied by PER models, but not to try to explicitly capture the exact theory of PER models [18]. This leads us to a new angle on parametricity which may prove useful in further study, and also gives us a set of axioms that are sufficient to prove *true* = *true'*, and other expected equations, without appearing contrived to fit these particular examples.

While $F_{<:}$ differs from each of the λ-calculi mentioned above, several properties of $F_{<:}$ transfer easily from related work. For syntactic properties we have strong normalization [15]; canonical type derivations, coherence, minimum typing [10]; and confluence of the β-η-*TopCollapse* equational theory [11]. The PER semantics follows easily from the work in [4], [7], [15], and [23]. While an alternative semantics could perhaps be developed in the style of [3] and [14], we do not explore that possibility here.

The main results of this paper are an equational theory for $F_{<:}$, some proof-theoretic properties developed in section 2 including conservativity of $F_{<:}$ typing over F, a set of examples in section 3 demonstrating the expressiveness of $F_{<:}$ (some reported earlier in [7] and in [15] with attribution), and some categorical properties in section 4 of the theory when restricted to closed terms.

2. System $F_{<:}$

$F_{<:}$ is obtained by extending F [16] [21] with a notion of subtyping (<:). This extension allows us to remain within a pure calculus. That is, we introduce neither the basic types, nor the structured types, normally associated with subtyping in programming languages. Instead, we show that these programming types can be obtained via encodings within the pure calculus. In particular, we can encode record types with their subtyping relations [5].

2.1 Syntax

Subtyping is reflected in the syntax of types by a new type constant *Top* (the supertype of all types), and by a subtype bound on second-order quantifiers: $\forall(X<:A)A'$ (*bounded quantifiers* [9]). Ordinary second-order quantifiers are recovered by setting the quantifier bound to *Top*; we use $\forall(X)A$ for $\forall(X<:Top)A$. The syntax of values is extended by a constant *top* of type *Top*, and by a subtype bound on polymorphic functions: $\lambda(X<:A)a$; we use $\lambda(X)a$ for $\lambda(X<:Top)a$.

Syntax

$A,B ::=$	Types
X	type variables
Top	the supertype of all types
$A \to B$	function spaces
$\forall(X<:A)B$	bounded quantifications
$a,b ::=$	Values
x	value variables
top	the canonical value of type *Top*
$\lambda(x:A)b$	functions
$b(a)$	applications
$\lambda(X<:A)b$	bounded type functions
$b(A)$	type applications

The $\to$ operator associates to the right. The scoping of λ and $\forall$ extends to the right as far as possible. Types and terms can be parenthesized.

A subtyping judgment is added to F's judgments. Moreover, the equality judgment on values is made relative to a type; this is important since values in $F_{<:}$ can have many types, and two values may or may not be equivalent depending on the type that those values are considered as possessing.

Judgments

$\vdash E\ env$	E is a well-formed environment
$E \vdash A\ type$	A is a type
$E \vdash A <: B$	A is a subtype of B
$E \vdash a : A$	a has type A
$E \vdash a \leftrightarrow b : A$	a and b are equal members of type A

We use *dom(E)* for the set of variables defined by an environment E. As usual, we identify terms up to renaming of bound variables; that is, using $B\{X\leftarrow C\}$ for the substitution of C for X in B:

$$\forall(X<:A)B \equiv \forall(Y<:A)\, B\{X\leftarrow Y\}$$
$$\lambda(X<:A)b \equiv \lambda(Y<:A)\, b\{X\leftarrow Y\} \qquad \lambda(x:A)b \equiv \lambda(y:A)\, b\{x\leftarrow y\}$$

These identifications can be made directly on the syntax; that is, without knowing whether the terms involved are the product of formal derivations in the system. By adopting these identifications, we avoid the need of a type equivalence judgment for quantifier renaming.

Moreover, in formal derivations we restrict ourselves to terms where all bound variables are distinct, to environments where variables are defined at most once, and to judgments where all bound and environment-defined variables are distinct. A more formal approach would use de Bruijn indices for free and bound variables [12].

2.2 Rules

The inference rules of $F_{<:}$ are listed below; the only essential difference between these and the ones of $F_{\leq}$ [15] [10], is in the more general *(Eq appl2)*. We now comment on the most interesting aspects of the rules; see also the discussion about *(Eq appl2)* in section 2.4.

The subtyping judgment, $E \vdash A <: B$, is, for any E, a reflexive and transitive relation on types with a *subsumption* property: that is, a member of a type is also a member of any supertype of that type. Every type is a subtype of *Top*. The function space operator $\rightarrow$ is antimonotonic in its first argument and monotonic in its second. A bounded quantifier is antimonotonic in its bound and monotonic in its body under an assumption about the free variable.

The rules for the typing judgment, $E \vdash a : A$, are the same as the corresponding rules in F, except for the extension to bounded quantifiers. However, additional typing power is hidden in the subsumption rule, which allows a function to take an argument of a subtype of its input type.

Most of the equivalence rules, $E \vdash a \leftrightarrow b : A$, are unremarkable. They provide symmetry, transitivity, congruence on the syntax, and β and η equivalences. Two rules, however, stand out. The first, *(Eq collapse)* (also called the *Top-collapse* rule), states that any two terms are equivalent when "seen" at type *Top*; since no operations are available on members of *Top*, all values are indistinguishable at that type. The second, *(Eq appl2)*, is the congruence rule for polymorphic type application, giving general conditions under which two expressions $b'(A')$ and $b''(A'')$ are equivalent at a type C. This rule has many intriguing consequences, which will be amply explored in the sequel. (We occasionally write $E \vdash A,B<:C$ for $E \vdash A<:C \wedge E \vdash B<:C$, and so on.)

Environments

(Env ∅)
$$\frac{}{\vdash \emptyset\ env}$$

(Env x)
$$\frac{E \vdash A\ type \quad x \notin dom(E)}{\vdash E,x:A\ env}$$

(Env X)
$$\frac{E \vdash A\ type \quad X \notin dom(E)}{\vdash E,X<:A\ env}$$

Types

(Type X)
$$\frac{\vdash E,X<:A,E'\ env}{E,X<:A,E' \vdash X\ type}$$

(Type Top)
$$\frac{\vdash E\ env}{E \vdash Top\ type}$$

(Type →)
$$\frac{E \vdash A\ type \quad E \vdash B\ type}{E \vdash A\rightarrow B\ type}$$

(Type ∀)
$$\frac{E,X<:A \vdash B\ type}{E \vdash \forall(X<:A)B\ type}$$

Subtypes

(Sub refl)
$$\frac{E \vdash A\ type}{E \vdash A <: A}$$

(Sub trans)
$$\frac{E \vdash A<:B \quad E \vdash B<:C}{E \vdash A <: C}$$

(Sub X)
$$\frac{\vdash E,X<:A,E'\ env}{E,X<:A,E' \vdash X<:A}$$

(Sub Top)
$$\frac{E \vdash A\ type}{E \vdash A <: Top}$$

(Sub →)
$$\frac{E \vdash A'<:A \quad E \vdash B<:B'}{E \vdash A\to B <: A'\to B'}$$

(Sub ∀)
$$\frac{E \vdash A'<:A \quad E,X<:A' \vdash B<:B'}{E \vdash \forall(X<:A)B <: \forall(X<:A')B'}$$

Values

(Subsumption)
$$\frac{E \vdash a:A \quad E \vdash A<:B}{E \vdash a : B}$$

(Val x)
$$\frac{\vdash E,x:A,E'\ env}{E,x:A,E' \vdash x:A}$$

(Val top)
$$\frac{\vdash E\ env}{E \vdash top : Top}$$

(Val fun)
$$\frac{E,x:A \vdash b:B}{E \vdash \lambda(x:A)b : A\to B}$$

(Val appl)
$$\frac{E \vdash b : A\to B \quad E \vdash a:A}{E \vdash b(a) : B}$$

(Val fun2)
$$\frac{E,X<:A \vdash b:B}{E \vdash \lambda(X<:A)b : \forall(X<:A)B}$$

(Val appl2)
$$\frac{E \vdash b : \forall(X<:A)B \quad E \vdash A'<:A}{E \vdash b(A') : B\{X\leftarrow A'\}}$$

Equivalence

(Eq symm)
$$\frac{E \vdash a \leftrightarrow b : A}{E \vdash b \leftrightarrow a : A}$$

(Eq trans)
$$\frac{E \vdash a \leftrightarrow b : A \quad E \vdash b \leftrightarrow c : A}{E \vdash a \leftrightarrow c : A}$$

(Eq x)
$$\frac{E \vdash x:A}{E \vdash x \leftrightarrow x : A}$$

(Eq collapse)
$$\frac{E \vdash a : Top \quad E \vdash b : Top}{E \vdash a \leftrightarrow b : Top}$$

(Eq fun)
$$\frac{E,x:A \vdash b\leftrightarrow b' : B}{E \vdash \lambda(x:A)b \leftrightarrow \lambda(x:A)b' : A\to B}$$

(Eq appl)
$$\frac{E \vdash b\leftrightarrow b' : A\to B \quad E \vdash a\leftrightarrow a' : A}{E \vdash b(a) \leftrightarrow b'(a') : B}$$

(Eq fun2)
$$\frac{E,X<:A \vdash b\leftrightarrow b' : B}{E \vdash \lambda(X<:A)b \leftrightarrow \lambda(X<:A)b' : \forall(X<:A)B}$$

(Eq appl2)
$$\frac{E \vdash b'\leftrightarrow b'' : \forall(X<:A)B \quad E \vdash A',A''<:A \quad E \vdash B\{X\leftarrow A'\}, B\{X\leftarrow A''\} <: C}{E \vdash b'(A') \leftrightarrow b''(A'') : C}$$

(Eq eta)
$$\frac{E \vdash b \leftrightarrow b' : A\to B \quad y \notin dom(E)}{E \vdash \lambda(y:A)b(y) \leftrightarrow b' : A\to B}$$

(Eq eta2)
$$\frac{E \vdash b \leftrightarrow b' : \forall(X<:A)B \quad Y \notin dom(E)}{E \vdash \lambda(Y<:A)b(Y) \leftrightarrow b' : \forall(X<:A)B}$$

(Eq beta)
$$\frac{E,x:A \vdash b \leftrightarrow b' : B \quad E \vdash a \leftrightarrow a' : A}{E \vdash (\lambda(x:A)b)(a) \leftrightarrow b'\{x\leftarrow a'\} : B}$$

(Eq beta2)
$$\frac{E,X<:A \vdash b \leftrightarrow b':B \quad E \vdash A' <: A}{E \vdash (\lambda(X<:A)b)(A') \leftrightarrow b'\{X\leftarrow A'\} : B\{X\leftarrow A'\}}$$

2.3 Basic properties

We now state some basic lemmas about $F_{<:}$ derivations. Most of these are proven by (simultaneous) induction on the size of the derivations; the proofs are long, but straightforward if carried out in the order indicated. We conclude the section with an application of these lemmas, showing that typing is preserved under β-η-reductions.

Notation

Let ϑ stand for either C *type*, $C{<:}C'$, $c{:}C$, or $c{\leftrightarrow}c'{:}C$.

Lemma (Implied judgments)

(ϑ/env) $\vdash E,F\ env \Rightarrow \vdash E\ env$ and $E,F\vdash\vartheta \Rightarrow \vdash E\ env$

(env/type) $\vdash E,X{<:}D,E'\ env \Rightarrow E\vdash D\ type$ and $\vdash E,x{:}D,E'\ env \Rightarrow E\vdash D\ type$

Lemma (Bound change)

$\vdash E,X{<:}D',E'\ env,\ E\vdash D\ type \Rightarrow \vdash E,X{<:}D,E'\ env$

$E,X{<:}D',E'\vdash C\ type,\ E\vdash D\ type \Rightarrow E,X{<:}D,E'\vdash C\ type$

Lemma (Weakening)

Let β stand for either $X{<:}D$ or $x{:}D$. Assume $\vdash E,\beta\ env$, and $X,x\notin dom(E')$. Then,

$\vdash E,E'\ env \Rightarrow \vdash E,\beta,E'\ env$ and $E,E'\vdash\vartheta \Rightarrow E,\beta,E'\vdash\vartheta$

Assume $\vdash E,F\ env$ and $dom(F)\cap dom(E')=\emptyset$. Then,

$\vdash E,E'\ env \Rightarrow \vdash E,F,E'\ env$ and $E,E'\vdash\vartheta \Rightarrow E,F,E'\vdash\vartheta$

Lemma (Implied judgments, continued)

(sub/type) $E\vdash C{<:}C' \Rightarrow E\vdash C\ type,\ E\vdash C'\ type$

Lemma (Bound weakening)

Let $<\beta,\beta'>$ stand for either $<X{<:}D,X{<:}D'>$ or $<x{:}D,x{:}D'>$. Assume $E\vdash D'{<:}D$. Then,

$\vdash E,\beta,E'\ env \Rightarrow \vdash E,\beta',E'\ env$ and $E,\beta,E'\vdash\vartheta \Rightarrow E,\beta',E'\vdash\vartheta$

Lemma (Type substitution)

Assume $E\vdash D'{<:}D$. Then,

$\vdash E,X{<:}D,E'\ env \Rightarrow \vdash E,E'\{X{\leftarrow}D'\}\ env$ and $E,X{<:}D,E'\vdash\vartheta \Rightarrow E,E'\{X{\leftarrow}D'\}\vdash\vartheta\{X{\leftarrow}D'\}$

Lemma (Value substitution)

Assume either $E\vdash d{:}D$, or d is any term and $x\notin FV(\vartheta)$; then

$\vdash E,x{:}D,E'\ env \Rightarrow \vdash E,E'\ env$ and $E,x{:}D,E'\vdash\vartheta \Rightarrow E,E'\vdash\vartheta\{x{\leftarrow}d\}$

Lemma (Implied judgments, continued)

(val/type) $E\vdash c:C \Rightarrow E\vdash C\ type,$

(eq/val) $E\vdash c{\leftrightarrow}c':C \Rightarrow E\vdash c:C,\ E\vdash c':C,$

Lemma (Eq subsumption)

$E\vdash c{\leftrightarrow}c':C,\ E\vdash C{<:}D \Rightarrow E\vdash c{\leftrightarrow}c':D$

Lemma (Implied judgments, continued)

(val/eq) $E\vdash c:C \Rightarrow E\vdash c{\leftrightarrow}c:C$

Lemma (Congruence)

$E\vdash d{\leftrightarrow}d':D \wedge E,x{:}D,E'\vdash c{:}C \Rightarrow E,E'\vdash c\{x{\leftarrow}d\}{\leftrightarrow}c\{x{\leftarrow}d'\}:C$

Lemma (Renaming)

Assume $Y\notin dom(E,X{<:}D,E')$. Then,

$\vdash E,X{<:}D,E'\ env \Rightarrow \vdash E,Y{<:}D,E'\{X{\leftarrow}Y\}\ env$ and $E,X{<:}D,E'\vdash\vartheta \Rightarrow E,Y{<:}D,E'\{X{\leftarrow}Y\}\vdash\vartheta\{X{\leftarrow}Y\}$

Assume $y\notin dom(E,x{:}D,E')$. Then,

$\vdash E,x{:}D,E'\ env \Rightarrow \vdash E,y{:}D,E'\ env$ and $E,x{:}D,E'\vdash\vartheta \Rightarrow E,y{:}D,E'\vdash\vartheta\{x{\leftarrow}y\}$

Lemma (Exchange)

Let β stand for either $X{<:}D$ or $x{:}D$. Let β' stand for either $X'{<:}D'$ or $x'{:}D'$. Assume $\vdash E,\beta'\ env$.

$\vdash E,\beta,\beta',E'\ env \Rightarrow \vdash E,\beta',\beta,E'\ env$ and $E,\beta,\beta',E'\vdash\vartheta \Rightarrow E,\beta',\beta,E'\vdash\vartheta$

Lemma (Substitution exchange)

Let β stand for either $x':D'$ or $X'<:D'$. Then,

$\vdash E,X<:D,\beta,E'\ env \Rightarrow \vdash E,\beta\{X\leftarrow D\},X<:D,E'\ env$

$E,X<:D,\beta,E' \vdash C\ type \Rightarrow E,\beta\{X\leftarrow D\},X<:D,E' \vdash C\ type$

The following two lemmas draw conclusions about the shape of terms and derivations, from the fact that certain subtyping and typing judgments have been derived.

Lemma (Subtyping decomposition)

- If $E \vdash A<:X$, then $A\equiv Y_1$ for some type variable Y_1 and either $Y_1\equiv X$, or for some $n\geq 1$, $Y_1<:Y_2\in E$ .. $Y_n<:X\in E$
- If $E,X<:B,E' \vdash X<:A$, then either $A\equiv X$ or $E,X<:B,E' \vdash B<:A$.
- If $E \vdash Top<:A$, then $A\equiv Top$.
- If $E \vdash B'\rightarrow B''<:A$, then either $A\equiv Top$ or $A\equiv A'\rightarrow A''$, $E \vdash A'<:B'$ and $E \vdash B''<:A''$
- If $E \vdash A<:B'\rightarrow B''$, then either $A\equiv A'\rightarrow A''$ for some A',A'', with $E \vdash B'<:A'$ and $E \vdash A''<:B''$, or $A\equiv X_1$ and for some $A',A'',n\geq 1$: $X_1<:X_2 \in E$ .. $X_n<:A'\rightarrow A'' \in E$ with $E \vdash B'<:A'$ and $E \vdash A''<:B''$
- If $E \vdash \forall(X<:B')B''<:A$, then either $A\equiv Top$ or $A\equiv\forall(X<:A')A''$, $E \vdash A'<:B'$ and $E,X<:A' \vdash B''<:A''$
- If $E \vdash A<:\forall(X<:B')B''$, then either $A\equiv\forall(X<:A')A''$ for some A',A'', with $E \vdash B'<:A'$ and $E,X<:B' \vdash A''<:B''$, or $A\equiv X_1$ and for some $A',A'',n\geq 1$: $X_1<:X_2 \in E$ .. $X_n<:\forall(X<:A')A'' \in E$ with $E \vdash B'<:A'$ and $E,X<:B' \vdash A''<:B''$

Lemma (Typing decomposition)

- If $E,x:D,E' \vdash x:C$, then $E \vdash D<:C$
- If $E \vdash top:A$, then $A\equiv Top$
- If $E \vdash \lambda(x:B')b : A$, then either $A\equiv Top$, or for some A',A'',B'', $A\equiv A'\rightarrow A''$ with $E \vdash A'<:B'$, $E \vdash B''<:A''$, and $E,x:B' \vdash b : B''$.
- If $E \vdash b(c) : B''$ then for some B', $E \vdash b : B'\rightarrow B''$ and $E \vdash c : B'$
- If $E \vdash \lambda(X<:B')b : A$, then either $A\equiv Top$, or for some A',A'',B'', $A\equiv\forall(X<:A')A''$ with $E \vdash A'<:B'$, $E,X<:A' \vdash B''<:A''$, and $E,X<:B' \vdash b : B''$.
- If $E \vdash b(C) : D$ then for some B',B'',X, $E \vdash C<:B'$, $E \vdash B''\{X\leftarrow C\} <: D$, and $E \vdash b : \forall(X<:B')B''$.

We conclude with a proposition about the preservation of typing under β and η reduction. The second-order η case is by far the hardest, and it requires the following lemma about the elimination of unused free variables (FV).

Lemma (Non-occurring type variable)

If $X\notin FV(c,E')$ and $E,X<:D,E' \vdash c : C$ then for some C_0 with $X\notin FV(C_0)$

$E,X<:D,E' \vdash c : C_0$ and $E,X<:D,E' \vdash C_0<:C$

Proposition (Preservation of typing under β-η-reductions)

(β1) $E \vdash (\lambda(x:B)b)(c) : A \Rightarrow E \vdash b\{x\leftarrow c\} : A$ **(η1)** $E \vdash \lambda(x:B)c(x) : A, x\notin FV(c) \Rightarrow E \vdash c : A$

(β2) $E \vdash (\lambda(X<:B)b)(C) : A \Rightarrow E \vdash b\{X\leftarrow C\} : A$ **(η2)** $E \vdash \lambda(X<:B)c(X) : A, X\notin FV(c) \Rightarrow E \vdash c : A$

Note that this proposition is non-trivial; for example, the (β1) case does not follow simply from the *(Eq beta)* rule and the eq/val lemma. Moreover, the derivation of $E \vdash b\{x\leftarrow c\} : A$ will have in general quite a different shape than the derivation of $E \vdash (\lambda(x:B)b)(c) : A$.

2.4 Derived rules

Most of the lemmas in the previous section can be written down as derived inference rules. Here we discuss some derived rules of special significance.

First, the eq-subsumption lemma in the previous section gives us a very interesting rule that lifts subsumption to the equality judgment; we remark that this is proven via the *(Eq beta)* rule.

(Eq subsumption)

$$\frac{E \vdash a \leftrightarrow a' : A \quad E \vdash A <: B}{E \vdash a \leftrightarrow a' : B}$$

Note that, in general, it is not true that $E \vdash a \leftrightarrow a' : B$ and $E \vdash A <: B$ imply $E \vdash a \leftrightarrow a' : A$.

The following two lemmas concern the equivalence of functions modulo domain restriction; the first one will find a useful application in section 3.1.

Lemma (Domain restriction)

If f: $A \to B$, then f is equivalent to its restriction $f|_{A'}$ to a smaller domain $A' <: A$, when they are both seen at type $A' \to B$. That is:

(Eq fun')

$$\frac{E \vdash A' <: A \quad E \vdash B <: B' \quad E, x:A \vdash b \leftrightarrow b' : B}{E \vdash \lambda(x:A)b \leftrightarrow \lambda(x:A')b' : A' \to B'}$$

Lemma (Bound restriction)

If f: $\forall(X <: A)B$, then f is equivalent to its restriction $f|_{A'}$ to a smaller bound $A' <: A$, when they are both seen at type $\forall(X <: A')B$. That is:

(Eq fun2')

$$\frac{E \vdash A' <: A \quad E, X <: A' \vdash B <: B' \quad E, X <: A \vdash b \leftrightarrow b' : B}{E \vdash \lambda(X <: A)b \leftrightarrow \lambda(X <: A')b' : \forall(X <: A')B'}$$

We now turn to the *(Eq appl2)* rule. This rule asserts that if a polymorphic function $b : \forall(X <: A)B$ is instantiated at two types $A' <: A$ and $A'' <: A$, then both instantiations evaluate to the same value with respect to any result type that is an upper bound of $B\{X \leftarrow A'\}$ and $B\{X \leftarrow A''\}$.

(Eq appl2)

$$\frac{E \vdash b' \leftrightarrow b'' : \forall(X <: A)B \quad E \vdash A' <: A \quad E \vdash A'' <: A \quad E \vdash B\{X \leftarrow A'\} <: C \quad E \vdash B\{X \leftarrow A''\} <: C}{E \vdash b'(A') \leftrightarrow b''(A'') : C}$$

Note that this rule asserts that the result of $b(A)$ is independent of A, in the proper result type.

A simpler derived rule (used in $F_{\leq}$ [10]) is obtained by setting $A' = A''$:

(Eq appl2 A'=A'')

$$\frac{E \vdash b' \leftrightarrow b'' : \forall(X <: A)B \quad E \vdash A' <: A}{E \vdash b'(A') \leftrightarrow b''(A') : B\{X \leftarrow A'\}}$$

However, the *(Eq appl2)* rule is most useful when $A' \neq A''$ and we can find an interesting upper bound to $B\{X \leftarrow A'\}$ and $B\{X \leftarrow A''\}$. This motivates the following derived rule, which is often used in practice.

Denote by $B\{X^- \leftarrow C, X^+ \leftarrow D\}$ the substitution of C for the negative occurrences of X in B, and of D for the positive ones. Take $A' <: A''$ ($<: A$), then we have (see [15, Sec. 14.3] for a proof):

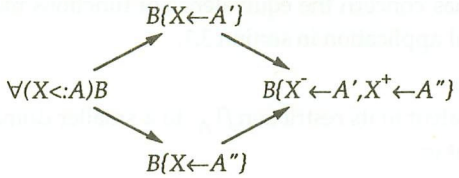
$$B\{X\leftarrow A'\} \equiv B\{X^-\leftarrow A', X^+\leftarrow A'\} <: B\{X^-\leftarrow A', X^+\leftarrow A''\}$$
$$B\{X\leftarrow A''\} \equiv B\{X^-\leftarrow A'', X^+\leftarrow A''\} <: B\{X^-\leftarrow A', X^+\leftarrow A''\}$$

Hence, for $A'<:A''<:A$ we have a (non trivial) common supertype for $B\{X\leftarrow A'\}$ and $B\{X\leftarrow A''\}$. This fact then justifies the rule:

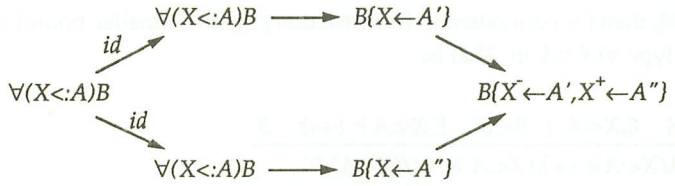
(Eq appl2^{-+})

$$\frac{E\vdash b'\leftrightarrow b'' : \forall(X<:A)B \quad E\vdash A'<:A''<:A}{E\vdash b'(A')\leftrightarrow b''(A'') : B\{X^-\leftarrow A', X^+\leftarrow A''\}}$$

This rule is in fact a special case of *dinaturality* of type application [3], where the dinaturality is required only with respect to coercions $A'<:A''$, for all A', A'' subtypes of A. We have the diagram:



The two arrows on the left are the A' and A'' instances of generic type application $x(X)$, where x is a variable of type $\forall(X<:A)B$, and B might have the type variable X free. The two arrows on the right are coercions induced by $A'<:A''$. Here $\forall(X<:A)B$ is constant in X, so the coercion $A'<:A''$ has no effect on this type. Hence the diagram above is just a brief version of:



where now the two horizontal arrows are the A' and A'' instances of $x(X)$. In the terminology of [3, p.42], the family given by $\{x(X) \mid X<:A\}$ is dinatural in the coercions.

We conclude this section with an application of *(Eq appl2)*, which is used insections 3.3 and 4.

Proposition (Eq-substitution)

Assume $E, X<:A, x: S \vdash b:B$ and X positive in S and B.

If $E\vdash A_1, A_2 <: A$, $E\vdash s_1:S\{X\leftarrow A_1\}$, $E\vdash s_2:S\{X\leftarrow A_2\}$, $E\vdash s_1\leftrightarrow s_2:S\{X\leftarrow A\}$

then $E\vdash b\{X\leftarrow A_1, x\leftarrow s_1\}\leftrightarrow b\{X\leftarrow A_2, x\leftarrow s_2\}: B\{X\leftarrow A\}$

Proof Let $M \triangleq \lambda(X<:A)\lambda(x:S)b$; then $E\vdash M : \forall(X<:A)S\rightarrow B$. Now prove:

(1) $E\vdash M(A_1)(s_1)\leftrightarrow M(A)(s_1) : B\{X\leftarrow A\}$ by *(Eq appl2)* and *(Eq appl)*, since X is positive in S and B

(2) $E\vdash M(A_2)(s_2)\leftrightarrow M(A)(s_2) : B\{X\leftarrow A\}$ similarly to (1)

(3) $E\vdash M(A)(s_1)\leftrightarrow M(A)(s_2) : B\{X\leftarrow A\}$ by *(Eq appl2)* and *(Eq appl)*, since $E\vdash s_1\leftrightarrow s_2:S\{X\leftarrow A\}$.

Conclude by *(Eq trans)*, *(Beta2)*, and *(Beta)*.

□

The proposition can be easily generalized to the case where there are several variables $x_1: S_1, \ldots, x_n: S_n$ (X positive in all of them) and terms $E\vdash s_1:S\{X\leftarrow A_1\},\ldots, E\vdash s_n:S\{X\leftarrow A_n\}$, with $E\vdash A_1,\ldots,A_n <: A$ and $E\vdash s_1\leftrightarrow\ldots\leftrightarrow s_n:S\{X\leftarrow A\}$.

2.5 PER semantics

For the PER semantics, the reader can consult [4], [7], [15], and [23]. The interpretation of $F_{<:}$ in PER is explained in those papers, except that the *(Eq appl2)* must be shown sound. The soundness for this rule is relatively straightforward, and omitted.

2.6 Conservativity of typing

Besides the presence of subtypes, the main new feature of $F_{<:}$ with respect to F lays in its equational theory, which extends the standard β-η equality in two directions, adding a terminal type *Top* and introducing the rule *(Eq appl2)*.

First of all, the equational theory ($\leftrightarrow$) of $F_{<:}$ is not conservative over F, the reason being the rule *(Eq appl2)*. Consider, for example:

Proposition

$E \vdash B\ type,\ \ E \vdash c : \forall(X)X{\to}B,\ \ E \vdash a : A \ \Rightarrow\ E \vdash c(Top)(top) \leftrightarrow c(A)(a) : B$

Proof

$E \vdash c(Top)(top) \leftrightarrow c(Top)(a) : B$	val/eq lemma *(Eq appl2) (Eq collapse) (Eq appl)*
$E \vdash c(Top)(a) \leftrightarrow c(A)(a) : B$	val/eq lemma *(Eq appl2) (Eq appl)*
$E \vdash c(Top)(top) \leftrightarrow c(A)(a) : B$	*(Eq trans)*

□

By applying this fact twice via *(Eq trans)* we can show:

$$y : \forall(X)X{\to}Bool \vdash y(Bool)(true) \leftrightarrow y(Bool)(false) : Bool$$

which is an F-judgment equating two different β-η-normal forms. It is well- known that no such judgment is derivable in F. A further application of *(Eq fun)* produces two closed terms with the same property.

As for the *typing* theory, however, $F_{<:}$'s rules are designed in such a way as to maintain and carefully generalize those of F. Writing $\vdash_F$ for derivations in F, and $\vdash_{<:}$ for derivations in $F_{<:}$, we can prove the following result.

Theorem

If $E \vdash_{<:} a : A$, where E, a, and A are in the language of F, then $E \vdash_F a : A$.

The proof of this statement (inspired by some results in [15]) requires a detour on *normal form proofs* in $F_{<:}$, a subject studied in [10] for a slightly different system, but sharing with $F_{<:}$ the same typing judgments. The reason for the detour is that trivial proofs by induction on the derivation of $E \vdash_{<:} a : A$ do not work, since $F_{<:}$ has "cut rules" (e.g. *(Subsumption)* or *(Val appl)*) that may introduce non-F types.

2.6.1 Normal and minimal proofs in $F_{<:}$

Subtype proofs

A *normal form proof* of $E \vdash_{<:} A{<:}B$ is a proof using either a single application of *(Sub Refl)* if $A{\equiv}B$, or it is a proof using only the rules *(Sub Top)*, *(Sub →)*, *(Sub ∀)*, or one of the following two rules:

(Sub X-Iter)

$$\frac{\vdash_{<:} E,X<:A,E' \; env}{E,X<:A,E' \vdash_{<:} X <: E^k(X)} \quad k\geq 1$$

(Sub X-Trans)

$$\frac{\vdash_{<:} E,X<:A,E' \; env \quad E,X<:A,E' \vdash_{<:} E^*(X)<:B}{E,X<:A,E' \vdash_{<:} X <: B}$$

where $E^1(X) = E(X)$, and $E^{k+1}(X) = E(X)$ if $E(X)$ is not a variable, or $E^{k+1}(X) = E^k(E(X))$ otherwise. Moreover, let k be the least k for which $E^k(X)$ is not a variable; then define $E^*(X) = E^k(X)$.

Type proofs

Normal form proofs and minimal normal form proofs of $E \vdash_{<:} a : A$ are simultaneously defined as follows.

A *normal form proof* of $E \vdash_{<:} a : A$ is either a minimal normal form proof or has the following shape:

$$\frac{E \vdash_{<:} a : A' \quad E \vdash_{<:} A' <: A}{E \vdash_{<:} a : A}$$

where $A' \neq A$, $E \vdash_{<:} a : A'$ is given by a minimal normal form proof, and $E \vdash_{<:} A' <: A$ is given by a normal form proof.

A *minimal normal form proof* of $E \vdash_{<:} a : A$ is a proof using only the rules: *(Val x)*; *(Val top)*; *(Val fun)* with the restriction that the premise is given by a minimal normal form proof; *(Val fun2)* with the restriction that the premise is given by a minimal normal form proof; or one of the following two rules *(Val appl-min)* and *(Val appl2-min)*.

(Val appl-min)

$$\frac{E \vdash_{<:} b : A\to B \quad E \vdash_{<:} a : A}{E \vdash_{<:} b(a) : B}$$

where $E \vdash_{<:} a : A$ is given by a normal form proof and $E \vdash_{<:} b : A\to B$ is given by either a minimal normal form proof or by a proof of the shape:

$$\frac{E \vdash_{<:} b : X \quad E \vdash_{<:} X <: E^*(X)}{E \vdash_{<:} b : E^*(X)}$$

where $E \vdash_{<:} b : X$ is given by a minimal normal form proof, X is a variable, $E^*(X) \equiv A\to B$, and $E \vdash_{<:} X <: E^*(X)$ is given by a single application of *(Sub X-Iter)*.

(Val appl2-min)

$$\frac{E \vdash_{<:} b : \forall(X<:A)B \quad E \vdash_{<:} A' <: A}{E \vdash_{<:} b(A') : B\{X\leftarrow A'\}}$$

where $E \vdash_{<:} A' <: A$ is given by a normal proof and $E \vdash_{<:} b : \forall(X<:A)B$ is given by either a minimal normal form proof or by a proof of the shape:

$$\frac{E \vdash_{<:} b : X \quad E \vdash_{<:} X <: E^*(X)}{E \vdash_{<:} b : E^*(X)}$$

where $E \vdash_{<:} b : X$ is given by a minimal normal form proof, X is a variable, $E^*(X) \equiv \forall(X<:A)B$, and $E \vdash_{<:} X <: E^*(X)$ is given by a single application of *(Sub X-Iter)*.

Proposition [10]

For any provable judgment $E \vdash_{<:} a : A$, there exists a (unique) normal form proof.

2.6.2 $F_{<:}$ typing is conservative over F typing

It is not difficult to see F as a subsystem of $F_{<:}$. As for the language, just define a translation function β so that:

$$\beta(\forall X.A) \equiv \forall(X<:Top)\,\beta(A) \qquad \beta(\Lambda X.M) \equiv \lambda(X<:Top)\,\beta(M)$$

and which is the identity on all the other constructs. A well formed environments E in F consist of a collection $E1 \equiv X_1,\ldots,X_h$ of type variables and a list $E2 \equiv x_1\colon S_1, \ldots, x_h\colon S_h$ of type assumptions, where at most the type variables in $E1$ can appear free. Then:

$$\beta(E) \equiv X_1<:Top, \ldots, X_h<:Top, x_1:\beta(S_1), \ldots, x_h:\beta(S_h).$$

By this, it is almost obvious that to any F-derivation $E \vdash_F a:A$ corresponds an $F_{<:}$-derivation $\beta(E) \vdash \beta(a):\beta(A)$ that never uses *(Subsumption)* (and thus subtyping rules) or *Top* rules and where *(Eq appl2)* is always applied in its special case when $A' \equiv A$" and $C \equiv B\{X \leftarrow A'\}$. In the following, we will argue directly in the language of $F_{<:}$ (thus dispensing from β).

Lemma

Let E be an F-environment, and let A and B be F-types.

$E \vdash_{<:} A<:B$ iff $A \equiv B$.

Lemma

Let E be an F-environment, a be an F-term, and let $E \vdash_{<:} a : A$ be a minimal normal form proof. Then A is an F-type and $E \vdash_F a : A$.

Proof By induction on the minimal normal form proof $E \vdash_{<:} a : A$. □

Theorem (Conservativity of typing over F)

Let E be an F-environment, a be an F-term and A be an F-type.

$$E \vdash_{<:} a : A \;\Rightarrow\; E \vdash_F a : A$$

Proof Consider a normal form proof of $E \vdash_{<:} a : A$. Note that it is necessarily a minimal normal form proof. Then the thesis reduces to that of the lemmas. □

3. Expressiveness

Since $F_{<:}$ is an extension of F, one can already carry out all the standard encodings of algebraic data types that are possible in F [2]. However, it is not clear that anything of further interest can be obtained from the subtyping rules of $F_{<:}$, which only involve an apparently useless type *Top* and the simple rules for $\rightarrow$ and $\forall$. In this section we begin to show that we can in fact construct rich subtyping relations on familiar data structures.

3.1 Booleans

In the sequel we concentrate on inclusion of structured types, but for this to make sense we need to show that there are some non-trivial inclusions already at the level of basic types. We investigate here the type of booleans, which also illustrates some consequences of the $F_{<:}$ rules.

Starting from the encoding of Church's booleans in F, we can define three subtypes of *Bool* as follows (cf. [13]):

$$\begin{array}{llll} Bool & \triangleq \forall(A)\,A\rightarrow A\rightarrow A & True & \triangleq \forall(A)\,A\rightarrow Top\rightarrow A \\ None & \triangleq \forall(A)\,Top\rightarrow Top\rightarrow A & False & \triangleq \forall(A)\,Top\rightarrow A\rightarrow A \end{array}$$

where:

$None <: True,\ None <: False,\ True <: Bool,\ False <: Bool$

Looking at all the closed normal forms (that is, the *elements*) of these types, we have:

$$true_{Bool} : Bool \triangleq \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}A)\,x \qquad true_{True} : True \triangleq \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}Top)\,x$$
$$false_{Bool} : Bool \triangleq \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}A)\,y \qquad false_{False} : False \triangleq \lambda(A)\,\lambda(x{:}Top)\,\lambda(y{:}A)\,y$$

We obtain four elements of type *Bool*; in addition to the usual two, $true_{Bool}$ and $false_{Bool}$, the extra $true_{True}$ and $false_{False}$ have type *Bool* by subsumption. However, we can show that $true_{Bool}$ and $true_{True}$ are provably equivalent at type *Bool*, by using the domain restriction lemma *(Eq fun')* from section 2.4.

$$\dfrac{\dfrac{\dfrac{\dfrac{E,A{<:}Top,x{:}A,y{:}Top \vdash x \leftrightarrow x : A \quad E \vdash A{<:}Top}{E,A{<:}Top,x{:}A \vdash \lambda(y{:}Top)\,x \leftrightarrow \lambda(y{:}A)\,x : A\to A}\ (Eq\ fun')}{E,A{<:}Top \vdash \lambda(x{:}A)\,\lambda(y{:}Top)\,x \leftrightarrow \lambda(x{:}A)\,\lambda(y{:}A)\,x : A\to A\to A}}{E \vdash \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}Top)\,x \leftrightarrow \lambda(A)\,\lambda(x{:}A)\,\lambda(y{:}A)\,x : \forall(A)\,A\to A\to A}}{E \vdash true_{True} \leftrightarrow true_{Bool} : Bool}$$

Similarly, we can show that $E \vdash false_{False} \leftrightarrow false_{Bool} : Bool$. Hence, there really are only two different values in *Bool*, one value each in *True* and *False*, and none in *None*.

3.2. Simple records

We restrict ourselves to the encoding of *simple records* (the ones with a fixed number of components [7]); *extensible records* are treated in [6].

Cartesian products are encoded, as usual, as $A\times B \triangleq \forall(C)(A\to B\to C)\to C$; note that by this definitions $\times$ is monotonic in both its arguments.

A *tuple type* is an iterated cartesian product; we consider only the ones ending with *Top*:

$$Tuple(A_1,..,A_n,Top) \triangleq A_1\times(..\times(A_n\times Top)..) \qquad n\geq 0$$

These types have the property that a tuple type with more components is a subtype of a corresponding tuple type with fewer components. For example:

$Tuple(A, B, Top) \equiv A\times B\times Top <: A\times Top \equiv Tuple(A, Top)$
because $A{<:}A$, $B\times Top{<:}Top$, and $\times$ is monotonic.

Tuple values are similarly encoded by iterated pairing, ending with *top*. The basic operations on tuple values are: $a\lfloor i$, dropping the first i components of tuple a; and $a.i$, selecting the i-th component of a. These are defined by iterating the product projections.

Let L be a countable set of *labels*, enumerated by a bijection $\iota\in L\to Nat$. We indicate by l^i, with a superscript, the i-th label in this enumeration. Often we need to refer to a list of n distinct labels out of this enumeration; we then use subscripts, as in $l_1..l_n$. So we may have, for example, $l_1,l_2,l_3 = l^5,l^1,l^{17}$. More precisely, $l_1..l_n$ stands for $l^{\sigma(1)},..,l^{\sigma(n)}$ for some injective $\sigma\in 1..n\to Nat$.

A record type has the form $Rcd(l_1{:}A_1,..,l_n{:}A_n,C)$; in this presentation C will always be *Top*. Once the enumeration of labels is fixed, a record type is encoded as a tuple type where the record components are allocated to tuple slots as determined by the index of their labels; the component of label l^i into the i-th tuple slot. The remaining slots are filled with *Top* "padding". For example:

$$Rcd(l^2{:}C, l^0{:}A, Top) \triangleq Tuple(A, Top, C, Top)$$

Since record type components are canonically sorted under the encoding, two record types

that differ only in the order of their components will be equal under the encoding. Hence we can consider record components as unordered.

From the encoding, we derive the familiar rule for simple records [5]:

$$\frac{E \vdash A_1 <: B_1 \;..\; E \vdash A_n <: B_n \quad E \vdash A_{n+1}\, type \;..\; E \vdash A_m\, type}{E \vdash Rcd(l_1{:}A_1,..,l_n{:}A_n,..,l_m{:}A_m,Top) <: Rcd(l_1{:}B_1,..,l_n{:}B_n,Top)}$$

This holds because any additional field $l_k{:}A_k$ ($n<k\le m$) on the left is absorbed either by the *Top* padding on the right, if $\iota(l_k)<max(\iota(l_1)..\iota(l_n))$, or by the final *Top*, otherwise.

Record values are similarly encoded; for example: $rcd(l^2{=}c,\ l^0{=}a,\ top) \triangleq tuple(a,\ top,\ c,\ top)$. Record selection is reduced to tuple selection by setting $r.l_i \triangleq r.\iota(l_i)$.

From these encodings we obtain all the usual typing rules for records. Moreover, the derived equational theory exhibits a form of observational equivalence:

$$\frac{\begin{array}{c}E \vdash a_1 \leftrightarrow b_1 : A_1 \;..\; E \vdash a_n \leftrightarrow b_n : A_n \\ E \vdash a_{n+1} : B_{n+1} \;..\; E \vdash a_p : B_p \quad E \vdash b_{n+1} : C_{n+1} \;..\; E \vdash b_q : C_q\end{array}}{E \vdash rcd(l_1{=}a_1,..,l_n{=}a_n,..,l_p{=}a_p,top) \leftrightarrow rcd(l_1{=}b_1,..,l_n{=}b_n,..,l_q{=}b_q,top) : Rcd(l_1{:}A_1,..,l_n{:}A_n,Top)}$$

That is, two records are equivalent if they coincide on the components that are observable at a given type. This holds ultimately because any two values are equivalent at type *Top*.

3.3. Lists

Following [2] we can define the algebra of parametric lists. *List[A]* stands for the homogeneous lists of type *A*.

$$List[A] \triangleq \forall(L)\, L \to (A \to L \to L) \to L$$

We have:

$$A <: B \;\Rightarrow\; List[A] <: List[B]$$

$$\begin{array}{ll}
nil{:}\ \forall(A)\ List[A] \triangleq & cons{:}\ \forall(A)\ A \to List[A] \to List[A] \triangleq \\
\quad \lambda(A)\,\lambda(L)\,\lambda(n{:}L)\,\lambda(c{:}A \to L \to L)\ n & \quad \lambda(A)\,\lambda(hd{:}A)\,\lambda(tl{:}List[A]) \\
 & \qquad \lambda(L)\,\lambda(n{:}L)\,\lambda(c{:}A \to L \to L) \\
length{:}\ \forall(A)\ List[A] \to Nat \triangleq & \qquad c(hd)(tl(L)(n)(c)) \\
\quad \lambda(A)\,\lambda(l{:}List[A]) & \\
\qquad l(Nat)(zero)(\lambda(a{:}A)\lambda(n{:}Nat)succ(n)) &
\end{array}$$

As an application of *(Eq appl2)* we can now show some interesting facts. Namely, any two empty lists are equal in *List[Top]*, and have the same length in *Nat*. Similarly for two singleton lists, and so on. In the proof, we will use the Eq-substitution proposition of Section 2.4.

Take *b:B* and *c:C*, then:

$$\vdash nil(B) \leftrightarrow nil(C) : List[Top] \qquad \textit{(Eq appl2)}$$

$$\vdash length(Top)(nil(B)) \leftrightarrow length(Top)(nil(C)) : Nat \qquad \textit{(Eq appl2, Eq appl)}$$

$$\vdash cons(B)(b)(nil(B)) \leftrightarrow cons(C)(c)(nil(C)) : List[Top]$$

by Eq-substitution, starting from $X{<:}Top,\ x{:}X,l{:}List[X] \vdash cons(X)(x)(l) : List[X]$

$$\vdash length(B)(cons(B)(b)(nil(B))) \leftrightarrow length(C)(cons(C)(c)(nil(C))) : Nat$$

by Eq-substitution, starting from $X{<:}Top,\ l{:}List[X] \vdash length\ (X)(l) : Nat$

Note that we have proven an interesting property of the behavior of *length* uniquely from its type; any function f: $\forall(A)\ List[A]\to Nat$ has such a property. This fact is related to the theorems proved by Wadler in [25] using only the types of terms. A difference is that our proof is carried out within $F_{<:}$, whereas Wadler uses parametricity properties beyond the proof system of F.

4. The category of closed terms

It is well known that the usual second-order encodings for products and coproducts, while logically sound, do not define, under β-η-equality, true categorical constructions. One can easily prove the existence of a term making a certain diagram commute, but its uniqueness does not follow from the standard equational rules.

As an example of the expressive power of *(Eq appl2)*, we show that those encodings are really categorical constructions when the underlining equational theory is the one of $F_{<:}$. In the same vein, motivated by the semantic isomorphisms obtained in [3] and [14] as consequences of parametricity, we investigate some provable isomorphisms in a suitable setting. The framework for our discussion is a category whose objects are the sets of closed terms of a closed type.

4.1 Definitions and basic properties

Recall first (see [LS 86] or [MS 89]) that given a typed λ-calculus language and a λ-theory **T**, a category $Cl(\mathbf{T})$ is determined by taking as objects of $Cl(\mathbf{T})$ the (closed) types of **T**. As for morphisms, choose first one variable for each type and define the morphisms from A to B to be equivalence classes of typing judgments $x{:}A \vdash t{:}B$, where x is the chosen variable of type A, and the equivalence relation is given by the equality judgments $x{:}A \vdash t \leftrightarrow t'{:}B$ of **T**. We will write $[x{:}A \vdash t{:}B]$ for the morphism given by the judgment $x{:}A \vdash t{:}B$. Identity is given by $[x{:}A \vdash x{:}A]$ and composition is defined by substitution:

$$[y{:}B \vdash s{:}C] \circ [x{:}A \vdash t{:}B] = [x{:}A \vdash s\{y \leftarrow t\}{:}C]$$

The category $Cl(F_{<:})$, obtained by applying this construction to $F_{<:}$, has a terminal object, given by *Top*. For any object A, the canonical morphism from A to *Top* is $[x{:}A \vdash top{:}Top]$; uniqueness is guaranteed by *(Eq collapse)*.

Now, given an arbitrary (small) category **C** with a terminal object *1*, consider the canonical functor $\ulcorner\_\urcorner : \mathbf{C} \to \mathbf{Sets}$ given by:

For any object A:
  $\ulcorner A\urcorner = \mathbf{C}(1,A)$ (the set of all morphisms $1\to A$)
For any morphism $f \in \mathbf{C}(A,B)$:
  $\ulcorner f\urcorner$ is the mapping from $\ulcorner A\urcorner$ to $\ulcorner B\urcorner$ given by composing with f
    (that is $\ulcorner f\urcorner(p) = f\circ p$ for $p \in \mathbf{C}(1,A)$)

Note that $\ulcorner\_\urcorner$ is not faithful if **C** is not well-pointed. Given $f,g \in \mathbf{C}(A,B)$, in fact, $\ulcorner f\urcorner$ and $\ulcorner g\urcorner$ are set-theoretical mappings and, therefore, in order to have $\ulcorner f\urcorner = \ulcorner g\urcorner$ it is sufficient that $f\circ p = g\circ p$ for any $p \in \mathbf{C}(1,A)$. The values of the functor $\ulcorner\_\urcorner : \mathbf{C} \to \mathbf{Sets}$ over all the objects and morphisms of **C** give a subcategory of **Sets** that can be denoted with $\ulcorner\mathbf{C}\urcorner$.

The category we are interested in is $\ulcorner Cl(F_{<:})\urcorner$. We will prove, as consequences of *(Eq appl2)*, that it has finite products and coproducts. For this, however, it is convenient to introduce the category **CL**, equivalent to $\ulcorner Cl(F_{<:})\urcorner$, for which we can give a more explicit description.

Remark

$\vdash A\ type$ reads "A is a closed type"

$\vdash a{:}A$ reads "a is a closed term of closed type A"

Definition (cl-equality)

We say $\vdash f \leftrightarrow^{cl} f' : A\to B$ iff

for all a, $\vdash f,f'{:}A\to B, \vdash a{:}A \Rightarrow \vdash f(a) \leftrightarrow f'(a) : B$

The *objects* of $\ulcorner Cl(F_{<:})\urcorner$ are, for any $\vdash A\ type$, the sets of morphisms $[z{:}Top \vdash t{:}A]$. By *(Eq collapse)* and congruence, $[z{:}Top \vdash t{:}A] = [z{:}Top \vdash t\{z\leftarrow top\}{:}A]$. The term $t\{z\leftarrow top\}$ is closed and $z{:}Top \vdash t\{z\leftarrow top\}{:}A$ iff $\vdash t\{z\leftarrow top\}{:}A$. Any object of $\ulcorner Cl(F_{<:})\urcorner$ is therefore isomorphic to the the set of equivalence classes $[\vdash a{:}A]$ of closed terms of a closed type; the equivalence relation is given by the equality judgments $\vdash a\leftrightarrow a'{:}A$ (write $\vdash A\ type$ for such a set). These sets are the objects of the category CL.

The *morphisms* of $\ulcorner Cl(F_{<:})\urcorner$ are, for any morphism $f=[x{:}A \vdash t{:}B]$ of $Cl(F_{<:})$, mappings from $\ulcorner A\urcorner$ to $\ulcorner B\urcorner$ given by, for any $[z{:}Top \vdash a{:}A]$, $\ulcorner f\urcorner([z{:}Top \vdash a{:}A]) = [z{:}Top \vdash t\{x\leftarrow a\}{:}B]$. By β- and η-conversion one obtains a category equivalent to $\ulcorner Cl(F_{<:})\urcorner$ by stipulating that a morphism of CL from $\vdash A\ type$ to $\vdash B\ type$ is an equivalence class of derivable term judgments:

$$\vdash f{:}A\to B$$

where the morphism equivalence is

$$(\vdash f{:}A\to B) = (\vdash f'{:}A\to B) \quad \text{iff} \quad \vdash f \leftrightarrow^{cl} f'{:}A\to B.$$

The *identity* and the *composition* judgment judgments are, for any $\vdash h{:}A\to B$ and $\vdash g{:}B\to C$:

$$id_A \triangleq \vdash \lambda(x{:}A)x : A\to A \qquad g\circ h \triangleq \vdash \lambda(x{:}A)g(h(x)) : A\to C$$

(We also ambiguously use $g\circ h \triangleq \lambda(x{:}A)g(h(x))$.)

We remark that morphism equivalence is *not* provable equality. For two morphisms $\vdash f{:}A\to B$ and $\vdash f'{:}A\to B$ to be equal it is sufficient that f and f' agree on the *closed* terms of type A. Similarly, the following two definitions correspond to isomorphism and uniqueness (for morphisms) in CL.

Definition (cl-isomorphism)

We say $\vdash A \sim^{cl} B$ iff there exist $\vdash f{:}A\to B, \vdash g{:}B\to A$ such that

$\vdash g\circ f \leftrightarrow^{cl} id_A : A\to A \qquad \vdash f\circ g \leftrightarrow^{cl} id_B : B\to B$

Definition (cl-uniqueness)

We say $\vdash f{:}A\to B$ is the cl-unique f satisfying $P(f)$ iff

for any other $\vdash f'{:}A\to B$ satisfying $P(f')$ we have $\vdash f \leftrightarrow^{cl} f' : A\to B$.

In order to prove that CL has finite products and coproducts, we need some more lemmas in $F_{<:}$, and especially the crucial consequence of *(Eq appl2)* expressed in the eq-var-substitution lemma, below.

Lemma (Type monotonicity)

Let $E,X{<:}B \vdash C <: D <: B$ and $E,X{<:}B,E' \vdash S\ type$. Then

(i) X positive in $S \Rightarrow E,X{<:}B,E' \vdash S\{X\leftarrow C\} <: S\{X\leftarrow D\}$

(ii) X negative in $S \Rightarrow E,X{<:}B,E' \vdash S\{X\leftarrow D\} <: S\{X\leftarrow C\}$

Definition (Pointed on X)

Given a type variable X, a type S is *pointed on* X iff X is positive in S and

$S\equiv\forall(Y_1{<:}B_1)\ldots\forall(Y_k{<:}B_k)T_1\to(\ldots\to(T_h\to X)\ldots)$ for $k\geq 0$, $h\geq 0$.

Lemma (Generalized collapse)

Let $E,X<:Top \vdash S\ type$, with S pointed on X.

$E \vdash D\ type$ and $E \vdash s : S\{X \leftarrow D\} \quad \Rightarrow \quad E,X<:Top,x:S \vdash x \leftrightarrow s : S\{X \leftarrow Top\}$

By generalised collapse and the eq-substitution proposition (Sect. 2.4) we obtain the following lemma, which expresses a parametricity property: A (possibly open) term a of a closed type A is provably equal to any term obtained by substituting specific types and terms for its free variables.

Lemma (Eq-var-substitution)

Assume, for $i=1..n$, $E',X<:Top \vdash S_i\ type$ and S_i pointed on X. Let $E \equiv E',X<:Top,x_1: S_1,\ldots, x_n:S_n$.

If $\vdash A\ type$, $E \vdash a:A$, $E' \vdash D\ type$ and $E' \vdash t_i: S_i\{X \leftarrow D\}$ for $i=1..n$,

then $E \vdash a \leftrightarrow a\{X \leftarrow D, x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\} : A$.

4.2 CL finite products and coproducts; well-pointedness

4.2.1 Terminal objects

Proposition

For any object $\vdash C\ type$, there is a unique morphism $\vdash 1_C : C \rightarrow Top$.

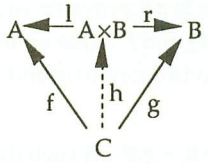
4.2.2 Binary products

Definition

$A \times B \triangleq \forall(C)\,(A \rightarrow B \rightarrow C) \rightarrow C$

Proposition

For any pair of objects $\vdash A\ type$, $\vdash B\ type$, the object $\vdash A \times B$ type is their categorical product.



That is, there exist $\vdash l:A \times B \rightarrow A$, $\vdash r:A \times B \rightarrow B$ such that for any $\vdash C\ type$, and for any $\vdash f:C \rightarrow A$, $\vdash g:C \rightarrow B$, there exists a unique (i.e. cl-unique) $\vdash h:C \rightarrow A \times B$ such that $\vdash l \circ h \leftrightarrow^{cl} f : C \rightarrow A$ and $\vdash r \circ h \leftrightarrow^{cl} g : C \rightarrow B$.

Corollary $\vdash A \sim^{cl} A', \quad \vdash B \sim^{cl} B' \quad \Rightarrow \quad \vdash A \times B \sim^{cl} A' \times B'$

4.2.3 Initial objects

Definition

$Bot \triangleq \forall(X)X$

Proposition

For any object $\vdash C\ type$, there is a unique morphism $\vdash 0_C : Bot \rightarrow C$.

Remark

$Bool \rightarrow Bot$ is also an initial object since there are no terms of type $Bool \rightarrow Bot$. The unique map is

the equivalence class of $\lambda(x: Bool\to Bot)\ x(true)(C)$, which includes $\lambda(x: Bool\to Bot)\ x(false)(C)$. More generally, any empty type V for which there exists a term $\vdash f:V\to Bot$ is initial. The canonical morphism is the equivalence class of $\lambda(x:V)\ f(x)(C)$, which is cl-unique since there are no closed terms $\vdash c:V$.

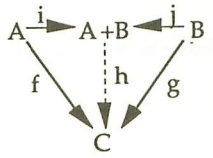
4.2.4 Binary coproducts

Definition

$A + B \triangleq \forall(C)\ (A\to C)\to(B\to C)\to C$

Proposition

For any pair of objects $\vdash A\ type$, $\vdash B\ type$, the object $\vdash A+B\ type$ is their categorical coproduct.



That is, there exist $\vdash i:A\to A+B$, $\vdash j:B\to A+B$ such that for any $\vdash C\ type$, and for any $\vdash f:A\to C$, $\vdash g:B\to C$, there exists a unique (i.e. cl-unique) $\vdash h:A+B\to C$ such that $\vdash h\circ i \leftrightarrow^{cl} f : A\to C$ and $\vdash h\circ j \leftrightarrow^{cl} g : B\to C$.

Proof Define: $i \triangleq \lambda(x:A)\lambda(C)\lambda(f:A\to C)\lambda(g:B\to C)f(x)$ then $\vdash i : A \to A+B$

$j \triangleq \lambda(y:B)\lambda(C)\lambda(f:A\to C)\lambda(g:B\to C)g(y)$ then $\vdash j : B \to A+B$

$case \triangleq \lambda(C)\lambda(f:A\to C)\lambda(g:B\to C)\lambda(c:A+B)c(C)(f)(g)$

then $\vdash case : \forall(C)\ (A\to C)\to(B\to C)\to(A+B)\to C$

We will only show that, for any $\vdash c:A+B$, $\vdash C\ type$, $\vdash D\ type$, $\vdash f:A\to C$, $\vdash g:B\to C$ and $\vdash k:C\to D$, $\vdash case(D)(k\circ f)(k\circ g)(c) \leftrightarrow (k\circ case(C)(f)(g))(c) : D$.

The normal form of c must have one of the shapes:

$c \equiv \lambda(C')\lambda(f':H)\lambda(g':G)f'(a)$

for some $C'<:Top \vdash A\to C <:H$, $C'<:Top \vdash B\to C'<:G$, and $C'<:Top, f':H, g':G \vdash a:A$

$c \equiv \lambda(C')\lambda(f':H)\lambda(g':G)g'(b)$

for some $C'<:Top \vdash A\to C <:H$, $C'<:Top \vdash B\to C'<:G$, and $C'<:Top, f':H, g':G \vdash b:B$

By bound weakening lemma,

$C'<:Top, f':A\to C', g':B\to C' \vdash a:A$ and $C'<:Top, f':A\to C', g':B\to C' \vdash b:B$.

By *(Eq fun')*, either $\vdash c \leftrightarrow \lambda(C')\lambda(f':A\to C')\lambda(g':B\to C')f'(a) : A+B$

or $\vdash c \leftrightarrow \lambda(C')\lambda(f':A\to C')\lambda(g':B\to C')g'(b) : A+B$

In the first case we have:

$\vdash case(D)(k\circ f)(k\circ g)(c) \leftrightarrow c(D)(k\circ f)(k\circ g) \leftrightarrow k(f(a\{C'\leftarrow D, f'\leftarrow k\circ f, g'\leftarrow k\circ g\})) : D$

$\vdash (k\circ case(C)(f)(g))(c) \leftrightarrow k(f(a\{C'\leftarrow C, f'\leftarrow f, g'\leftarrow g\})) : D$

From eq-var-substitution lemma:

$C'<:Top, f':A\to C', g':B\to C' \vdash a \leftrightarrow a\{C'\leftarrow D, f'\leftarrow k\circ f, g'\leftarrow k\circ g\} : A$

$C'<:Top, f':A\to C', g':B\to C' \vdash a \leftrightarrow a\{C'\leftarrow C, f'\leftarrow f, g'\leftarrow g\} : A$

Conclude by transitivity and *(Eq appl)*.

The second case is similar.

□

Corollary $\vdash A \sim^{cl} A',\ \vdash B \sim^{cl} B' \ \Rightarrow\ \vdash A+B \sim^{cl} A'+B'$

4.2.5 Well-pointedness

Recall that a category **C** with a terminal object *1* is well-pointed iff for any pair of objects *A* and *B* and any $f,g \in \mathbf{C}(A,B)$ we have:

$f=g$ iff for any $h \in \mathbf{C}(1,A)$, $f \circ h = g \circ h$.

Proposition

CL is well-pointed. That is, for any $\vdash A\ type$, $\vdash B\ type$, and any $\vdash f,g : A \to B$, we have:

$\vdash f \leftrightarrow^{cl} g : A \to B \ \Leftrightarrow$ for any $\vdash h : Top \to A$, $\vdash f \circ h \leftrightarrow^{cl} g \circ h : Top \to B$

4.3 CL Isomorphisms

For the following isomorphisms we have been inspired by [3] and [14].

4.3.1 Double negation

We prove that, for any $\vdash A\ type$ we have $A \sim \forall(C)(A \to C) \to C$. This is an isomorphism holding in the models studied in [3], but having no known proof in *F* (see the remark below).

Proposition

$\vdash A\ type \ \Rightarrow\ \vdash A \sim^{cl} \forall(C)(A \to C) \to C$

Proof

Define: $f \triangleq \lambda(x{:}\forall(C)(A \to C) \to C)\ x(A)(id(A))$ $\qquad$ $g \triangleq \lambda(y{:}A)\ \lambda(C)\ \lambda(z{:}A \to C)\ z(y)$

Then: $\vdash f{:}\ (\forall(C)(A \to C) \to C) \to A$ and $\vdash g{:}\ A \to (\forall(C)(A \to C) \to C)$.

Take *a* such that $\vdash a{:}A$. Then, by β-conversion:

$\vdash f(g(a)) \leftrightarrow a : A$

Take closed *b* such that $\vdash b : \forall(C)(A \to C) \to C$. Then *b* has a normal form of the shape

$b = \lambda(C)\ \lambda(z{:}D)\ z(a1)$

for some $C{<:}Top \vdash A \to C{<:}D$ and $C{<:}Top, z{:}D \vdash a1{:}A$. By bound weakening lemma,

$C{<:}Top, z{:}A \to C \vdash a1{:}A$

and hence

$\vdash b \leftrightarrow \lambda(C)\ \lambda(z{:}A \to C)\ z(a1)$.

Then

$\vdash g(f(b)) \leftrightarrow \lambda(C)\ \lambda(z{:}A \to C)\ z(a1\{C \leftarrow A,\ z \leftarrow id(A)\}) : \forall(C)(A \to C) \to C$

By eq-var-substitution lemma,

$C{<:}Top,\ z{:}A \to C \vdash a1 \leftrightarrow a1\{C \leftarrow A,\ z \leftarrow id(A)\} : A$

Hence,

$C{<:}Top,\ z{:}A \to C \vdash z(a1) \leftrightarrow z(a1\{C \leftarrow A,\ z \leftarrow id(A)\}) : C$

That is:

$\vdash \lambda(C)\ \lambda(z{:}A \to C)\ z(a1) \leftrightarrow \lambda(C)\ \lambda(z{:}A \to C)\ z(a1\{C \leftarrow A,\ z \leftarrow id(A)\}) : \forall(C)(A \to C) \to C$

Combining the two equations above:

$\vdash g(f(b)) \ \leftrightarrow\ \lambda(C)\ \lambda(z{:}A \to C)\ z(a1) \ \leftrightarrow\ b\ :\ \forall(C)(A \to C) \to C$

□

Remark

Christine Paulin-Mohring has shown that, even for A closed, $A \sim \forall(C)(A\to C)\to C$ is not provable in F via the isomorphism we have used in the proof above. (It is not known whether some other isomorphism would work.) To see this, let T be $\forall(R)R\to R$; the term:

$$\lambda(P)\,\lambda(x{:}(T\to T)\to P)$$
$$\quad x\,(\lambda(y{:}T)\,y\,(P\to T)\,(\lambda(u{:}P)y)\,(x(\lambda(v{:}T)v)))$$
$$: \forall(P)((T\to T)\to P)\to P$$

is not convertible to any term of the form $\lambda(P)\,\lambda(x{:}(T\to T)\to P)\,x(c)$ where $c{:}T\to T$ is a closed term. Roberto di Cosmo has shown that A is not isomorphic, in the usual sense, to $\forall(C)(A\to C)\to C$ in F.

4.3.2 Other isomorphisms

Existentials

Define $\exists(X{<:}A)B \triangleq \forall(V)(\forall(X{<:}A)B\to V)\to V$ and $\exists(X)B \triangleq \exists(X{<:}Top)B$. Then:
$\exists(X{<:}A)X \sim A$ (corollary: $\exists(X)X \sim Top$)

Domain restriction

$C \sim \forall(X)\,X\to C$
$A\to C \sim \forall(X{<:}A)\,X\to C$

Categorical

$(A\times B)\times C \sim A\times(B\times C)$ $\qquad$ $(A+B)+C \sim A+(B+C)$
$A\times Top \sim Top\times A \sim A$ $\qquad$ $A+Bot \sim Bot+A \sim A$

Various

$Top\to A \sim A$	(simple top collapse)
$A\to Top \sim Top$	(simple top collapse)
$Top \sim \forall(C)\,C\to C$	(analyzing the normal forms)
$Bot\to A \sim Top$	(analyzing the normal forms)
$A\to Bot \sim Bot$ for A nonempty	(vacuous $f\circ g \leftrightarrow^{cl} id$ conditions: both types are empty)
$\forall(X)\,(A\to X) \sim A\to\forall(X)\,X$	(β-η suffices)

References

[1] S.Abramsky, J.C.Mitchell, A.Scedrov, P.Wadler: ***Relators***, to appear.

[2] C.Böhm, A.Berarducci: ***Automatic synthesis of typed λ-programs on term algebras,*** Theoretical Computer Science, 39, pp. 135-154, 1985.

[3] E.S.Bainbridge, P.J.Freyd, A.Scedrov, P.J.Scott: ***Functorial polymorphism***, Theoretical Computer Science, vol.70, no.1, pp 35-64, 1990.

[4] K.B.Bruce, G.Longo: ***A modest model of records, inheritance and bounded quantification,*** Information and Computation, 87(1/2):196-240, 1990.

[5] L.Cardelli: ***A semantics of multiple inheritance***, in Information and Computation 76, pp 138-164, 1988.

[6] L.Cardelli: ***Extensible records in a pure calculus of subtyping***, to appear.

[7] L.Cardelli, G.Longo: ***A semantic basis for Quest,*** Proceedings of the 6th ACM LISP and Functional Programming Conference, ACM Press, 1990.

[8] L.Cardelli, J.C.Mitchell: ***Operations on records***, Proc. of the Fifth Conference on Mathematical Foundations of Programming Language Semantics, New Orleans, 1989. To appear in Mathematical Structures in Computer Science, 1991.

[9] L.Cardelli, P.Wegner: ***On understanding types, data abstraction and polymorphism***, Computing Surveys, Vol 17 n. 4, pp 471-522, December 1985.

[10] P.-L.Curien, G.Ghelli: ***Coherence of subsumption***, Mathematical Structures in Computer Science, to appear.

[11] P.-L.Curien, G.Ghelli: ***Subtyping + extensionality: confluence of $\beta\eta$-reductions in $F_{\leq}$***, to appear.

[12] N.G.de Bruijn: ***Lambda-calculus notation with nameless dummies***, in Indag. Math. 34(5), pp. 381-392, 1972.

[13] J.Fairbairn: ***Some types with inclusion properties in $\forall, \rightarrow, \mu$***, Technical report No 171, University of Cambridge, Computer Laboratory.

[14] P.J.Freyd: ***Structural polymorphism***, to appear in TCS.

[15] G.Ghelli: ***Proof theoretic studies about a mininal type system integrating inclusion and parametric polymorphism***, Ph.D. Thesis TD-6/90, Università di Pisa, Dipartimento di Informatica, 1990.

[16] J-Y.Girard: ***Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types***, Proceedings of the second Scandinavian logic symposium, J.E.Fenstad Ed. pp. 63-92, North-Holland, 1971.

[17] J.Lambek, P.J.Scott: ***Introduction to higher order categorical logic***, Cambridge University Press, 1986.

[18] J.C.Mitchell: ***A type inference approach to reduction properties and semantics of polymorphic expressions***, Logical Foundations of Functional Programming, ed. G. Huet, Addison-Wesley, 1990.

[19] J.C.Mitchell, P.J.Scott: ***Typed λ-models and cartesian closed categories***, in Categories in Computer Science and Logic, J.W.Gray and A.Scedrov Eds. Contemporary Math. vol. 92, Amer. Math. Soc., pp 301-316, 1989.

[20] A.M.Pitts: ***Polymorphism is set-theoretic, constructively***, in Category Theory and Computer Science, Proceedings Edinburgh 1987, D.H.Pitt, A.Poigne, and D.E.Rydeheard Eds. Springer Lecture Notes in Computer Science, vol. 283, pp 12-39, 1987.

[21] J.C.Reynolds: ***Towards a theory of type structure***, in Colloquium sur la programmation pp. 408-423, Springer-Verlag Lecture Notes in Computer Science, n.19, 1974.

[22] J.C.Reynolds: ***Types, abstraction, and parametric polymorphism***, in Information Processing '83, pp 513-523, R.E.A.Mason ed., North Holland, Amsterdam, 1983.

[23] A.Scedrov: ***A guide to polymorphic types***, in Logic and Computer Science, pp 387-420, P.Odifreddi ed., Academic Press, 1990.

[24] C.Strachey: ***Fundamental concepts in programming languages***, lecture notes for the International Summer School in Computer Programming, Copenhagen, August 1967.

[25] P.Wadler: ***Theorems for free!***, Proc. of the Fourth International Conference on Fuctional Programming and Computer Architecture, ACM Press, 1989.