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 $F_{<:}$ (pronounced *ef-sub*) 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.

Appears in: International Conference on Theoretical Aspects of Computer Software, T.Ito, A.R.Meyer Eds., Lecture Notes in Computer Science n. 526, pp 750-770, Springer Verlag, 1991.
Also to appear in Information & Computation.

SRC Research Report 80, December 30, 1991. Revised January 1, 1993.
© Digital Equipment Corporation 1991,1993.
This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Systems Research Center of Digital Equipment Corporation in Palo Alto, California; an acknowledgment of the authors and individuals contributors to the work; and all applicable portions of the copyright notice. Copying, reproducing, or republishing for any other purpose shall require a license with payment of fee to the Systems Research Center. All rights reserved.

---

[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.

Contents

1. Introduction
2. System $F_{<:}$
    2.1 Syntax
    2.2 Rules
    2.3 Basic properties
    2.4 Derived rules
    2.5 PER semantics
    2.6 Conservativity of typing
        2.6.1 Normal and minimal proofs in $F_{<:}$
        2.6.2 $F_{<:}$ typing is conservative over F typing
        2.6.3 $F_{<:}$ typing is conservative "modulo an equality" over F_1 typing
3. Expressiveness
    3.1 Booleans
    3.2 Naturals
    3.3 Products
    3.4 Simple tuples
    3.5 Simple records
    3.6 Lists
4. The category of closed terms
    4.1 Definitions and basic properties
    4.2 CL finite products and coproducts; well-pointedness
        4.2.1 Terminal objects
        4.2.2 Binary products
        4.2.3 Initial objects
        4.2.4 Binary coproducts
        4.2.5 Well-pointedness
    4.3 CL isomorphisms
        4.3.1 Double negation
        4.3.2 Existentials
        4.3.3 Other cl-isomorphisms
Acknowledgments
References

1. Introduction

System F [Gir 71] [Rey 74] is a well-known typed λ-calculus with polymorphic types that provides a basis for polymorphic programming languages. We study an extension of F that combines parametric polymorphism [Str 67] 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 [Ghe 90] [CG 91]. $F_{\leq}$ is, in turn, a fragment of the language *Fun* [CW 85]. 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 [Car 88], [CM 91] and related work that are connected to operations used in object-oriented programming. We illustrate some of the power of $F_{<:}$ in Section 3; see also [Car 91].

We have also 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 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.

Consider the polymorphic type $\forall(A)A\to A\to A$. This type 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:

$$\begin{aligned} \mathit{true} &\triangleq \lambda(A)\ \lambda(x{:}A)\ \lambda(y{:}A)\ x \\ \mathit{false} &\triangleq \lambda(A)\ \lambda(x{:}A)\ \lambda(y{:}A)\ y \end{aligned}$$

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 its 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.*

$$\begin{aligned} \mathit{true}' &\triangleq \lambda(A)\ \lambda(x{:}A)\ \lambda(y{:}\mathit{Top})\ x \\ \mathit{false}' &\triangleq \lambda(A)\ \lambda(x{:}\mathit{Top})\ \lambda(y{:}A)\ y \end{aligned}$$

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 [BL 88] [CL 90] [Ghe 90] [Sce 90]. 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, they 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 [Rey 83], [Fre 91], and [MS 91]. A general principle we have followed is to adopt axioms that express parametricity properties satisfied by PER models, but not to capture explicitly the exact theory of PER models [Mit 90]. 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; in particular, $F_{<:}$ differs from $F_{\leq}$[CG 91] only in the equational theory. For syntactic properties we have strong normalization [Ghe 90]; canonical type derivations, coherence, minimum typing [CG 91]; and confluence of the β-η-*TopCollapse* equational theory [CG 91a]. The PER semantics follows easily from the work in [BL 88], [CL 90], [Ghe 90], and [Sce 90]. While an alternative semantics could perhaps be developed in the style of [BFSS 90] and [Fre 91], 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 [CL 90], and in [Ghe 90] with attribution), and in section 4 some categorical properties of the theory when restricted to closed terms.

2. System $F_{<:}$

$F_{<:}$ is obtained by extending F [Gir 71] [Rey 74] (see Appendix) 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 [Car 88].

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* [CW 85]). 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* (mostly a convenience), 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
$A \rightarrow B$	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 $\rightarrow$ 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 (see, for example, the rule *(Eq collapse)* in section 2.2).

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*, and *FV(-)* for sets of free variables:

$$\forall(X<:A)B \equiv \forall(Y<:A)\,B\{X\leftarrow Y\} \quad \text{where } Y \notin FV(B)$$
$$\lambda(x:A)b \equiv \lambda(y:A)\,b\{x\leftarrow y\} \quad \text{where } y \notin FV(b)$$
$$\lambda(X<:A)b \equiv \lambda(Y<:A)\,b\{X\leftarrow Y\} \quad \text{where } Y \notin FV(b)$$

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.

Environments, however, are not identified up to renaming of variables in their domains; environment variables are kept distinct by construction. A more formal approach would use de Bruijn indices for free and bound variables [deB 72].

2.2 Rules

The inference rules of $F_{<:}$ are listed below; the only essential difference between these and the ones of $F_{\leq}$ [Ghe 90] [CG 91] is in the more general *(Eq appl2)* rule. 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 throughout this work. (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 →)
$$\frac{E \vdash A\ type \quad E \vdash B\ type}{E \vdash A \to B\ type}$$

(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↔c':C.*

Lemma (Renaming)

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

$\vdash E,X{<:}D,E'\ env \Rightarrow \vdash E,Y{<:}D,E'\{X \leftarrow Y\}\ env$ (equal-size derivations)

$E,X{<:}D,E' \vdash \vartheta \Rightarrow E,Y{<:}D,E'\{X \leftarrow Y\} \vdash \vartheta\{X \leftarrow Y\}$ (equal-size derivations)

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

$\vdash E,x{:}D,E'\ env \Rightarrow \vdash E,y{:}D,E'\ env$ (equal-size derivations)

$E,x{:}D,E' \vdash \vartheta \Rightarrow E,y{:}D,E' \vdash \vartheta\{x \leftarrow y\}$ (equal-size derivations)

Lemma (Implied judgments)

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

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

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

$\vdash E,x:D,E'\ env \Rightarrow E \vdash D\ type$

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 $X,x \notin dom(E')$

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

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

Lemma (Multiple weakening)

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

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

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

Proof Induction on the length of F. □

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 $E \vdash D'<:D$.

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

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

Lemma (Type substitution)

Assume $E \vdash D'<:D$; then

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

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

Lemma (Value substitution)

Assume $E \vdash d:D$; then

$\vdash E,x:D,E'\ env \Rightarrow \vdash E,E'\ env$

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

Lemma (Value strengthening)

Assume $x \notin FV(\vartheta)$; then, for $\vartheta \neq c \leftrightarrow c':C$.

$\vdash E,x:D,E'\ env \Rightarrow \vdash E,E'\ env$

$E,x:D,E' \vdash \vartheta \Rightarrow E,E' \vdash \vartheta$

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)

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

Proof

By sub/type lemma, $E \vdash C\ type$. Take $x \notin dom(E)$.

Then $\vdash E,x:C\ env$ and $E,x:C \vdash x:C$.

By weakening lemma $E,x:C \vdash C <: D$

By *(Subsumption)* $E,x:C \vdash x:D$, and by *(Eq x)*, $E,x:C \vdash x \leftrightarrow x:D$.

By *(Eq fun)*, $E \vdash \lambda(x:C)x \leftrightarrow \lambda(x:C)x : C \rightarrow D$.

By hypothesis and *(Eq appl)*, $E \vdash (\lambda(x:C)x)(c) \leftrightarrow (\lambda(x:C)x)(c') : D$

By *(Eq beta)*, $E \vdash (\lambda(x:C)x)(c) \leftrightarrow c' : D$.

By *(Eq symm) (Eq beta)*, $E \vdash (\lambda(x:C)x)(c') \leftrightarrow c : D$.

Hence by *(Eq symm) (Eq trans)*, $E \vdash c \leftrightarrow c' : D$. □

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 (Exchange)

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

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

Assume $\vdash E,\beta'\ env$.

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

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

Lemma (Substitution exchange)

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

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

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

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

Proof (sketch)

All cases are proven by induction on the size of the derivations, in order to circumvent the *(Sub refl)* and *(Sub trans)* rules that do not follow the structure of terms. Otherwise the proofs are straightforward. □

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'', $A \equiv A' \to A''$
  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',
  $E \vdash b : B' \to B''$ 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 $E \vdash b : \forall (X <: B') B''$.

Proof (sketch)

All cases are proven by induction on the size of the derivations, in order to circumvent the *(Subsumption)* rule that does not follow the structure of terms.
Otherwise the proofs are straightforward. □

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

Proof

By induction on the derivation of $E,X<:D,E' \vdash c : C$. The interesting cases are *(Val appl)* and *(Val appl2)*, where we use the subtyping decomposition lemmas for $\rightarrow$ and $\forall$.

We show the *(Val appl2)* case, where we have:

$c \equiv b(A')$, $C \equiv B\{Y \leftarrow A'\}$ (for $Y \notin dom(E,X<:D,E')$)

$E,X<:D,E' \vdash b : \forall(Y<:A)B$, $\quad E,X<:D,E' \vdash A' <: A$.

Since $X \notin FV(b)$, by induction there is a type AB_0 with $X \notin FV(AB_0)$, and

$E,X<:D,E' \vdash b : AB_0$, $E,X<:D,E' \vdash AB_0 <: \forall(Y<:A)B$.

By the (subtyping decomposition lemma) $AB_0 \equiv \forall(Y<:A_0)B_0$ with:

either $AB_0 \equiv \forall(Y<:A_0)B_0$ for some A_0, B_0,

with $E,X<:D,E' \vdash A <: A_0$ and $E,X<:D,E',Y<:A_0 \vdash B_0 <: B$.

Hence, $X \notin FV(\forall(Y<:A_0)B_0)$, $E,X<:D,E' \vdash b : \forall(Y<:A_0)B_0$

or $AB_0 \equiv X_1$ and for some $A_0, B_0, n \geq 1$:

$X_1 <: X_2 \in E,X<:D,E' \quad ... \quad X_n <: \forall(Y<:A_0)B_0 \in E,X<:D,E'$

with $E,X<:D,E' \vdash A <: A_0$ and $E,X<:D,E',Y<:A_0 \vdash B_0 <: B$.

If $X_n <: \forall(Y<:A_0)B_0 \in E$; $X \notin FV(\forall(Y<:A_0)B_0)$ since X comes after E.

If $X_n <: \forall(Y<:A_0)B_0 \equiv X<:D$; $X \notin FV(D \equiv \forall(Y<:A_0)B_0)$.

If $X_n <: \forall(Y<:A_0)B_0 \in E'$; $X \notin FV(\forall(Y<:A_0)B_0)$ by the hyp. $X \notin FV(E')$.

By n uses of (Sub X) and (Subsumption), $E,X<:D,E' \vdash b : \forall(Y<:A_0)B_0$.

Hence, in both cases, by (Sub Trans), $E,X<:D,E' \vdash A' <: A_0$,

and $E,X<:D,E' \vdash b(A') : B_0\{Y \leftarrow A'\}$, with $X \notin FV(B_0\{Y \leftarrow A'\})$,

Moreover, from $E,X<:D,E',Y<:A_0 \vdash B_0 <: B$

by (bound weakening lemma) $E,X<:D,E',Y<:A' \vdash B_0 <: B$

and by (type substitution lemma) $E,X<:D,E' \vdash B_0\{Y \leftarrow A'\} <: B\{Y \leftarrow A'\}$.

Hence we can take $C_0 \equiv B_0\{Y \leftarrow A'\}$. $\square$

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

(β1) $\quad E \vdash (\lambda(x:B)b)(c) : A \implies E \vdash b\{x \leftarrow c\} : A$

(η1) $\quad E \vdash \lambda(x:B)c(x) : A,\ x \notin FV(c) \implies E \vdash c : A$

(β2) $\quad E \vdash (\lambda(X<:B)b)(C) : A \implies E \vdash b\{X \leftarrow C\} : A$

(η2) $\quad E \vdash \lambda(X<:B)c(X) : A,\ X \notin FV(c) \implies E \vdash c : A$

Proof

The first three cases are obtained easily by applying the appropriate decomposition lemmas, along with weakening, bound weakening, value and type substitution, and value strengthening.

The (η2) case goes as follows. From $E \vdash \lambda(X<:B)c(X) : A$ by the (typing decomposition lemma) for fun2 and appl2, we obtain (omitting the easy case of $A \equiv Top$), for some A',A",B",Y,C',C":

$A \equiv \forall(X<:A')A''$ with $E \vdash A' <: B'$, $E,X<:A' \vdash B'' <: A''$, and $E,X<:B' \vdash c(X) : B''$

$E,X<:B' \vdash c : \forall(Y<:C')C''$ with $E,X<:B' \vdash X <: C'$ and $E,X<:B' \vdash C''\{Y \leftarrow X\} <: B''$.

Since $X \notin FV(c)$, by the (non-occurring type variable) lemma there is a D with:

$X \notin FV(D)$ and $E,X<:B' \vdash c : D$, $E,X<:B' \vdash D <: \forall(Y<:C')C''$

Using the (subtyping decomposition lemma) on D we obtain two subcases that, for some D',D'', both lead to:

$E,X<:B' \vdash c : \forall(Y<:D')D''$, $X \notin FV(\forall(Y<:D')D'')$

with $E,X<:B' \vdash C' <: D'$ and $E,X<:B',Y<:C' \vdash D'' <: C''$

By the (type strengthening lemma) from $E,X<:B' \vdash c : \forall(Y<:D')D''$:

$E \vdash c : \forall(Y<:D')D''$ *i.e.* $E \vdash c : \forall(X<:D')D''\{Y \leftarrow X\}$

Now, to obtain the final goal $E \vdash c : \forall(X<:A')A''$ via subsumption, we need to show only that $E \vdash \forall(X<:D')D''\{Y \leftarrow X\} <: \forall(X<:A')A''$, i.e. that:

(1) $E \vdash A' <: D'$

(2) $E,X<:A' \vdash D''\{Y \leftarrow X\} <: A''$

For (1) we use the (type substitution lemma) to get:

$E \vdash B' <: C'\{X \leftarrow B'\}$ (from $E,X<:B' \vdash X <: C'$)

$E \vdash C'\{X \leftarrow B'\} <: D'\{X \leftarrow B'\} \equiv D'$ (from $E,X<:B' \vdash C' <: D'$)

Hence $E \vdash A' <: B' <: C'\{X \leftarrow B'\} <: D'$.

For (2) we use the (bound weakening lemma) twice to get:

$E,X<:A',Y<:X \vdash D'' <: C''$

(from $E,X<:B',Y<:C' \vdash D'' <: C''$, $E,X<:B' \vdash X <: C'$, $E \vdash A' <: B'$)

from this by the (type substitution lemma)

$E,X<:A' \vdash D''\{Y \leftarrow X\} <: C''\{Y \leftarrow X\}$

We also have, by the (bound weakening lemma):

$E,X<:A' \vdash C''\{Y \leftarrow X\} <: B''$ (from $E,X<:B' \vdash C''\{Y \leftarrow X\} <: B''$, $E \vdash A' <: B'$)

Finally: $E,X<:A' \vdash D''\{Y \leftarrow X\} <: C''\{Y \leftarrow X\} <: B'' <: A''$. □

Note that this proposition is nontrivial; 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')

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

Proof (sketch)

First derive $E \vdash \lambda(y:A')(\lambda(x:A)b)(y) \leftrightarrow \lambda(x:A')b' : A' \to B'$ via *(Eq-subsumption)* and *(Eq beta)*. Then pass from $E \vdash \lambda(x:A)b \leftrightarrow \lambda(x:A)b : A \to B$ to $E \vdash \lambda(x:A)b \leftrightarrow \lambda(x:A)b : A' \to B'$ by *(Eq subsumption)*, and to $E \vdash \lambda(y:A')(\lambda(x:A)b)(y) \leftrightarrow \lambda(x:A)b : A' \to B'$ by *(Eq eta)*.

Conclude by transitivity. □

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')

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

Proof

Similar to the previous lemma, using *(Eq beta2)* and *(Eq eta2)*. □

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 $F_{\leq}$ [CG 91]) 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 a nontrivial upper bound to $B\{X \leftarrow A'\}$ and $B\{X \leftarrow A''\}$. This fact 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:

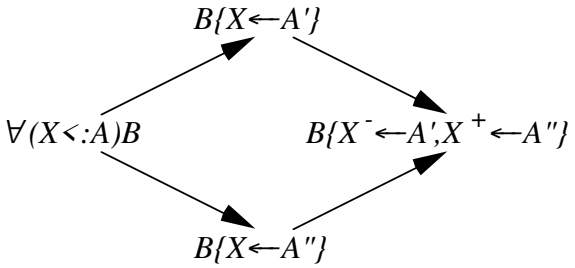
$$\begin{aligned} B\{X \leftarrow A'\} \quad &\equiv \quad B\{X^- \leftarrow A', X^+ \leftarrow A'\} \quad <: \quad B\{X^- \leftarrow A', X^+ \leftarrow A''\} \\ B\{X \leftarrow A''\} \quad &\equiv \quad B\{X^- \leftarrow A'', X^+ \leftarrow A''\} \quad <: \quad B\{X^- \leftarrow A', X^+ \leftarrow A''\} \end{aligned}$$

(A proof of this may be found in [Ghe 90], section 14.3.) Hence, for $A' <: A'' <: A$ we have a (nontrivial) common supertype for $B\{X \leftarrow A'\}$ and $B\{X \leftarrow A''\}$. This fact then justifies the rule:

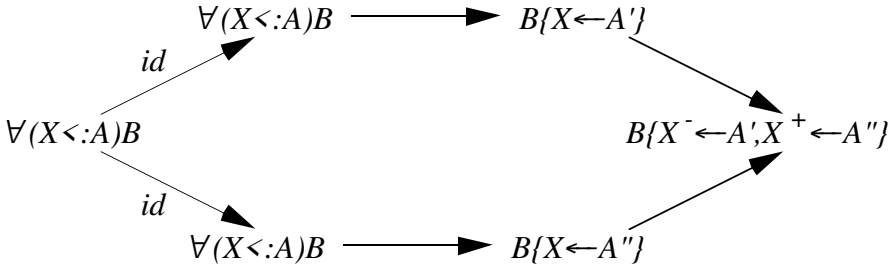
(Eq appl2 -+)

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

This rule is in fact a special case of *dinaturality* of type application [BFSS 90], 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 [BFSS 90, p.42], the family given by $\{x(X)|X<:A\}$ is dinatural in the coercions.

We conclude this section with an application of *(Eq appl2)*, which is used in sections 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 [BL 88], [CL 90], [Ghe 90], and [Sce 90]. The interpretation of $F_{<:}$ in PER is explained in those papers, except that the rule *(Eq appl2)* must be shown sound. The proof rests on the fact that, given types $\forall(X<:A)B$ and $A'<:A$ and denoting with $[\![_]\!]$ the interpretation function for types, we have $[\![\forall(X<:A)B]\!] \subseteq [\![B\{X\leftarrow A'\}]\!]$. From this, and the observation that the interpretation for terms is given by erasing the type information, the conclusion is straightforward.

2.6 Conservativity of typing

Besides the presence of subtypes, the main new feature of $F_{<:}$ with respect to F lies in its equational theory, which extends the standard β-η equality in two directions, by adding a terminal type *Top* and introducing the rule *(Eq appl2)*. Besides nonessential syntactic variants, the language of F is included in $F_{<:}$'s language and thus it makes sense to investigate whether $F_{<:}$ is conservative over F. We may, however, consider also an "intermediate" system between F and $F_{<:}$, with the property that the language inclusion of F into $F_{<:}$ "splits".

The system we are interested in is F_1 , obtained by adding to F the type constant *Top*, together with rule *(Eq collapse)* for making *Top* a terminal type. If we want to compare $F_{<:}$ with its underlying subtype-free systems, we need a system such as F_1 , and not F, since it is well known that the terminal type is not definable in F. Moreover, the conservativity result we will prove with respect to F holds because $F_{<:}$ proves only trivial subtype judgments between F types, while the situation for F_1 is more complex and its analysis sheds some more light on the structure of subtype proofs.

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

Proposition

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

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 to maintain and carefully generalize those of its subsystems. Writing $\vdash_F$ for derivations in F, $\vdash_1$ for derivations in F_1 , and $\vdash_{<:}$ for derivations in $F_{<:}$, we can prove the following result.

Theorem

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

(ii) If $E \vdash_{<:} a : A$, where *E, a,* and *A* are in the language of F_1 , then there exists an F_1-term, *a1*, such that $E \vdash_1 a1 : A$ and $E \vdash_{<:} a \leftrightarrow a1 : A$.

The proof of these statements (inspired by some results in [Ghe 90]) requires a detour on *normal form proofs* in $F_{<:}$. These normal forms are studied in [CG 91] for a slightly different system, but they share 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)*, *(Sub Trans)*, or *(Val appl)*) that may introduce non-F (or non-F_1) types.

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

In $F_{<:}$ a single typing judgment may have many proofs. The non-determinism of the proof search arises from the freedom in the order in which the rules *(Subsumption)* and *(Sub trans)* can be applied. However, as showed in [CG 91], this freedom does not provide additional proving power. In subtype proofs we can do without *(Sub trans)* except for the uses where the first (i.e., smallest) type is a variable appearing in the environment. In type proofs, we can restrict the use of *(Subsumption)* so as to derive only the *least* type for a given term, which may be then given a larger type with a single, last application of *(Subsumption)*. These ideas are the inspiration for the notions of normal and minimal normal proofs.

Subtype proofs

A *normal form proof* of $E \vdash_{<:} A<:B$ is a proof $E \vdash_{nf} A<:B$ obtained in the formal system $\vdash_{nf}$ consisting of the rules *(Sub Top)*, *(Sub →)*, *(Sub ∀)* (where $\vdash_{<:}$ is replaced by $\vdash_{nf}$), plus the following rules:

$$\frac{E \vdash_{nf} X \; type}{E \vdash_{nf} X <: X} \;\textit{(Sub Refl-X)} \qquad \frac{E', X<:B, E'' \vdash_{nf} B <: A}{E', X<:B, E'' \vdash_{nf} X <: A} \;\textit{(Sub Trans-X)} \qquad A \not\equiv Top$$

Type proofs

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

A *normal form proof* $E \vdash_{nf} a : A$ is either (1) a minimal normal form proof $E \vdash_{mnf} a : A$, or (2) a minimal normal form proof followed by a single nontrivial use of subsumption; in this case the final step has the form:

$$\frac{E \vdash_{mnf} a : A' \quad E \vdash_{nf} A' <: A}{E \vdash_{nf} a : A} \qquad \text{where } A' \not\equiv A.$$

A *minimal normal form proof* $E \vdash_{mnf} a : A$ is a proof using only the rules: *(Val x)*, *(Val top)*, *(Val fun)*, *(Val fun2)* (where $\vdash_{<:}$ is replaced by $\vdash_{mnf}$), or one of the two rules below, which use the following notation:

- $E(X) \equiv A$ if $E \equiv E1, X<:A, E2$.

- $E^*(C) \equiv C$ if C is not a variable;
  $E^*(X) \equiv E(X)$ if $E(X)$ is not a variable,
  $E^*(X) \equiv E1^*(E(X))$ if $E(X)$ is a variable and $E \equiv E1, X <: A, E2$.

(Val appl-min)

$$\frac{E \vdash_{mnf} b : C \quad E \vdash_{nf} a : A}{E \vdash_{mnf} b(a) : B} \qquad E^*(C) \equiv A \to B$$

(Val appl2-min)

$$\frac{E \vdash_{mnf} b : C \quad E \vdash_{nf} A' <: A}{E \vdash_{mnf} b(A') : B\{X \leftarrow A'\}} \qquad E^*(C) \equiv \forall (X <: A) B$$

Proposition

For any provable judgment $E \vdash_{<:} a : A$, there exists a *unique* derivation of $E \vdash_{nf} a : A$.

Proof [CG 91] □

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

It is not difficult to see F as a subsystem of $F_{<:}$. We can define a translation function τ over the language of F so that:

$$\tau(\forall X.A) \equiv \forall (X <: Top)\ \tau(A)$$
$$\tau(\Lambda X.M) \equiv \lambda (X <: Top)\ \tau(M)$$

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

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

From this, it is almost obvious that F-derivations $E \vdash_F a : A$ and $E \vdash_F a \leftrightarrow a' : A$ are mapped to $F_{<:}$-derivations $\tau(E) \vdash \tau(a) : \tau(A)$ and $\tau(E) \vdash \tau(a) \leftrightarrow \tau(a') : \tau(A)$ with the following properties. The resulting derivations never use *(Subsumption)* (and thus subtyping rules) or *Top* rules, and *(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 with τ).

Lemma

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

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

Proof

The "if" direction is a routine induction. For the other direction, take the normal form proof of $E \vdash_{<:} A <: B$. Then, *(Sub →)* and *(Sub ∀)* proceed by induction, and *(Sub Refl-X)* is

trivial. For *(Sub Trans-X)*, $E \vdash_{nf} X <: A$ must have been derived from $E', X <: Top, E'' \vdash_{nf} Top <: A$, but the latter implies $A \equiv Top$ by the subtyping decomposition lemma, which is absurd since A is an F-type. □

Lemma

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

Proof

By induction on the derivation $E \vdash_{mnf} a : A$.

(Val x) $E', x:A, E'' \vdash_{mnf} x : A$.

Then A is an F-type, since E is an F-environment.

(Val fun) The last rule is:

$$\frac{E, x:A \vdash_{mnf} b : B}{E \vdash_{mnf} \lambda(x:A)b : A \to B}$$

By hypothesis, $\lambda(x:A)b$ is an F-term and therefore A is an F-type.
By induction hypothesis, B is an F-type and $E, x:A \vdash_F b : B$.

(Val fun2) is analogous to *(Val fun)*.

(Val appl-min) The last rule is:

$$\frac{E \vdash_{mnf} b : C \quad E \vdash_{nf} a : A}{E \vdash_{mnf} b(a) : B} \qquad E^*(C) \equiv A \to B$$

Consider first the premise $E \vdash_{mnf} b : C$.
We show that C cannot be a variable. Indeed, if it were the case that $C \equiv X$, then $E^*(C) \equiv E(X) \equiv Top$, since E is an F-environment, contrary to the side-condition that $E^*(C)$ has to be a function type.
Therefore C is not a variable, and $E^*(C) \equiv C \equiv A \to B$.
By induction hypothesis, $A \to B$ is an F-type and $E \vdash_F b : A \to B$.
Consider now the proof $E \vdash_{nf} a : A$. We claim it is actually a minimal normal form proof. In fact, we already proved that $A \to B$ is an F-type; hence A is an F-type. If it were the case that the last step of the proof $E \vdash_{nf} a : A$ is

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

with $A' \not\equiv A$, then, by induction hypothesis, A' would be an F-type and $A' \equiv A$ by the previous lemma. Hence the proof $E \vdash_{nf} a : A$ is a minimal normal proof $E \vdash_{mnf} a : A$ and, by induction hypothesis, $E \vdash_F a : A$.

(Val appl2-min) The last rule is:

$$\frac{E \vdash_{mnf} b : C \quad E \vdash_{nf} A' <: A}{E \vdash_{mnf} b(A') : B\{X \leftarrow A'\}} \qquad E^*(C) \equiv \forall (X<:A)B$$

Note first that since $b(A')$ is an F-term, A' is an F-type. As in the previous case, C cannot be a variable, and $C \equiv \forall (X<:A)B$.
By induction hypothesis, $\forall (X<:A)B$ is an F-type (thus $A \equiv Top$, making trivial the other premise $E \vdash_{nf} A' <: Top$) and $E \vdash_F b : \forall (X<:Top)B$.
Then $E \vdash_F b(A') : B\{X \leftarrow A'\}$. □

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 the unique normal form proof $E \vdash_{nf} a : A$.
If its last step is:

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

with $A' \not\equiv A$, then, by the previous lemma, A' would be an F-type and $A' \equiv A$ by the other lemma. The proof $E \vdash_{nf} a : A$ is then a proof $E \vdash_{mnf} a : A$; the previous lemma allows us to obtain the conclusion. □

2.6.3 $F_{<:}$ typing is conservative "modulo an equality" over F_1 typing

As in the case of F, system F_1 can be easily viewed as a subsystem of $F_{<:}$. Consider the subsystem of $F_{<:}$ obtained by: restricting *(Env X)* to the case where $A \equiv Top$, dropping all the subtyping rules but *(Sub Top)*, removing *(Subsumption)*, and restricting *(Eq appl2)* to the case where $A' \equiv A$" and $C \equiv B\{X \leftarrow A'\}$. We will therefore identify F_1 with this subsystem and write $\vdash_1$ for F_1-derivations.

The reason why the typing theory of $F_{<:}$ is conservative over that of F (expressed in the first lemma of the previous subsection) is that only trivial subtype judgments $E \vdash_{<:} A<:B$ with $A \equiv B$ can be proved when A and B are F-types. The situation for F_1-types is more interesting, since, due to *(Sub Top),* nontrivial inclusions can be proved.

A first remark is that the typing of $F_{<:}$ is *not* conservative over that of F_1:

$$X<:Top, x:X \vdash_{<:} x:Top$$

but, of course,

$$\neg\, (X<:Top, x:X \vdash_1 x:Top)$$

This failure is, indeed, one of the pragmatic reasons (from the programming language design viewpoint) for introducing *(Subsumption)*, since this is the mechanism by which a program (method, function, …) can be inherited in other types.

We can look, however, for conservativity modulo an $F_{<:}$-equality. If $E \vdash_{<:} a : A$, where E, a, and A are in the language of F_1 , then there exists an F_1-term, $a1$ say, such that $E \vdash_1 a1 : A$ and $E \vdash_{<:} a \leftrightarrow a1 : A$. In the example above, it is obvious that $X<:Top,x:X \vdash_1 top:Top$ and $X<:Top,x:X \vdash_1 x \leftrightarrow top:Top$, by *(Eq Top)*.

We start with some preliminary lemmas. Let

$$id \equiv \lambda(X<:Top)\lambda(x:X)x$$

Lemma (Identity coercions)

Let E be an F_1-environment, A and B be F_1-types, and $E \vdash_{<:} A<:B$. Then there exists an F_1-term $k_{A,B}$ such that:

$E \vdash_1 k_{A,B}:A \to B$ and $E \vdash_{<:} k_{A,B} \leftrightarrow id(A) : A \to B$.

Proof

By induction on the normal form proof $E \vdash_{nf} A<:B$.

Note first that *(Sub Trans-X)* cannot be the last rule of such a proof, because its premise would be $E',X<:Top,E'' \vdash_{nf} Top <: A$ (since E is an F_1-environment), which would imply $A \equiv Top$ by subtyping decomposition lemma, which is impossible because of the side condition requiring $A \not\equiv Top$. In the other cases, we take $k_{A,B}$ as the (inductively defined) explicit coercion between A and B. Details are as follows.

(Sub Refl-X) is trivial.

(Sub Top) $E \vdash_{<:} A<:Top$. Take then $k_{A,Top} \equiv \lambda(x:A)top$.

Rules *(Eq collapse)* and *(Eq fun)* give $E \vdash_{<:} k_{A,Top} \leftrightarrow id(A) : A \to Top$.

(Sub →) Define $k_{A \to B, A' \to B'} \equiv \lambda(f:A \to B)\, k_{B,B'} \circ f \circ k_{A',A}$.

From $E \vdash_{nf} A \to B <: A' \to B'$, by induction hypothesis and an easy argument:

$E, f:A \to B \vdash_{<:} \lambda(x:A')k_{B,B'}(f(k_{A',A}(x))) \leftrightarrow \lambda(x:A')f(x) : A' \to B'$

by ***(Eq eta)*** and transitivity:

$E, f:A \to B \vdash_{<:} \lambda(x:A')k_{B,B}(f(k_{A',A}(x))) \leftrightarrow f : A' \to B'$

by *(Eq fun)*:

$E \vdash_{<:} \lambda(f:A \to B)\lambda(x:A')k_{B,B'}(f(k_{A',A}(x)))$

$\quad \leftrightarrow \lambda(f:A \to B)f : (A \to B) \to (A' \to B')$

(Sub ∀) $E \vdash_{nf} \forall(X<:A)B <: \forall(X<:A')B'$ where $A \equiv A' \equiv Top$ because both $\forall(X<:A)B$ and $\forall(X<:A')B'$ are F_1-types. Let:

$C \equiv \forall(X<:Top)B$ and $C' \equiv \forall(X<:Top)B'$

and define:

$k_{C,C'} \equiv \lambda(x:C)\lambda(X<:Top)k_{B,B'}(x(X))$

From $E \vdash_{nf} C <: C'$, by induction and an easy argument

$E,x:C \vdash_{<:} \lambda(X<:Top)k_{B,B'}(x(X)) \leftrightarrow \lambda(X<:Top)x(X) : C'$

by ***(Eq eta2)*** and transitivity

$E,x:C \vdash_{<:} \lambda(X<:Top)k_{B,B'}(x(X)) \leftrightarrow x : C'$

and hence the thesis, by *(Eq fun)*. □

Lemma

Let E be an F_1-environment, a an F_1-term and $E \vdash_{mnf} a : A$. Then:

(i) A is an F_1-type

(ii) there exists an F_1-term $a1$ such that $E \vdash_1 a1 : A$ and $E \vdash_{<:} a \leftrightarrow a1 : A$

Proof

By induction on $E \vdash_{mnf} a : A$.

(Val x) $E',x:A,E'' \vdash_{mnf} x : A$. Then A is an F_1-type, since E is an F_1-environment and $a1 \equiv x$; the conclusion (ii) follows by *(Eq x)*.

(Val top) $E \vdash_{mnf} top : Top$. Then also $E \vdash_1 top : Top$ and we can take $a1 \equiv top$.

(Val fun) The last rule is:

$$\frac{E,x:A \vdash_{mnf} b : B}{E \vdash_{mnf} \lambda(x:A)b : A \to B}$$

By hypothesis, $\lambda(x:A)b$ is an F_1-term and therefore A is an F_1-type.
By induction hypothesis, B is an F_1-type and there exists a term $b1$ such that $E,x:A \vdash_1 b1 : B$ and $E,x:A \vdash_{<:} b \leftrightarrow b1 : B$.
The thesis follows by *(Eq fun)*.

(Val fun2) is analogous to *(Val fun)*.

(Val appl-min) The last rule is:

$$\frac{E \vdash_{mnf} b : C \quad E \vdash_{nf} a : A}{E \vdash_{mnf} b(a) : B} \qquad E^*(C) \equiv A \to B$$

Consider first the left premise, $E \vdash_{mnf} b : C$.
We observe that C cannot be a variable X. If it were, since E is an F_1-environment, we would have $E^*(C) \equiv E(X) \equiv Top$, contradicting the assumption that $E^*(C) \equiv A \to B$.
Thus, $C \equiv A \to B$, induction applies, $A \to B$ is an F_1-type and we obtain an F_1-term $b1$ such that

$E \vdash_1 b1 : A \to B$ and $E \vdash_{<:} b \leftrightarrow b1 : A \to B$.

Consider now the other premise, $E \vdash_{nf} a : A$.
If it happens to be a minimal normal form proof $E \vdash_{mnf} a : A$ then by induction hypothesis we have a term $a1$ such that:

$E \vdash_1 a1 : A$ and $E \vdash_{<:} a \leftrightarrow a1 : A$.

Otherwise, the last step of $E \vdash_{nf} a : A$ is:

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

By induction hypothesis, A' is an F_1-type and we get an F_1-term a' such that $E \vdash_1 a' : A'$ and $E \vdash_{<:} a \leftrightarrow a' : A'$.
We already proved that $A \rightarrow B$ is an F_1-type; hence A is an F_1-type.
From $E \vdash_{nf} A' <: A$, the identity coercions lemma gives an F_1 term $k_{A',A}$ such that $E \vdash_1 k_{A',A} : A' \rightarrow A$ and $E \vdash_{<:} k_{A',A} \leftrightarrow id(A') : A' \rightarrow A$.
Take then $a1 \equiv k_{A',A}(a')$. Simple computations give:
$E \vdash_1 a1 : A$ and $E \vdash_{<:} a \leftrightarrow a1 : A$.
Finally, by *(Eq appl)*
$E \vdash_1 b1(a1) : B$ and $E \vdash_{<:} b1(a1) \leftrightarrow b(a) : B$.

(Val appl2-min) The last rule is

$$\frac{E \vdash_{mnf} b : C \quad E \vdash_{nf} A' <: A}{E \vdash_{mnf} b(A') : B\{X \leftarrow A'\}} \qquad E^*(C) \equiv \forall (X <: A)B$$

Note, first, that since $b(A')$ is an F_1-term, A' is an F_1-type.
As in the previous case, in $E \vdash_{mnf} b : C$, C cannot be a variable.
Therefore, the left premise is $E \vdash_{mnf} b : \forall (X <: A)B$.
By induction hypothesis, $\forall (X <: A)B$ is an F_1-type
(thus $A \equiv Top$ and the second premise is trivial)
and we have an F_1-term $b1$ such that
$E \vdash_1 b1 : \forall (X <: Top)B$ and $E \vdash_{<:} b \leftrightarrow b1 : \forall (X <: Top)B$.
Then $E \vdash_1 b1(A') : B\{X \leftarrow A'\}$ and $E \vdash_{<:} b(A') \leftrightarrow b1(A') : B\{X \leftarrow A'\}$. □

We can finally prove our conservativity result:

Theorem (Conservativity of typing over F_1)

If $E \vdash_{<:} a : A$, where E, a, and A are in the language of F_1 , then there exists an F_1-term, $a1$, such that $E \vdash_1 a1 : A$ and $E \vdash_{<:} a \leftrightarrow a1 : A$.

Proof

Take the normal form proof $E \vdash_{nf} a : A$. If it is a minimal normal form proof, then the thesis follows by the previous lemma. If, on the other hand, it consists of a minimal normal form proof $E \vdash_{mnf} a : A'$ followed by subsumption with premise $E \vdash_{nf} A' <: A$, then, by the previous lemma, A' is an F_1-type and we have an F_1-term, a', such that $E \vdash_1 a' : A'$ and $E \vdash_{<:} a \leftrightarrow a' : A'$. The thesis then follows by the identity coercions lemma and *(Eq appl)*. □

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 [BB 85]. However, it is not clear that anything of further interest can be obtained from the subtyping rules of $F_{<:}$, which involve only 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 rest of section 3 we concentrate on inclusion of structured types, but for this to make sense we need to show that there are some nontrivial inclusions already at the level of basic types. We investigate here the type of booleans, illustrating 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. [Fai 89]):

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

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:

$$\begin{array}{lcl} true_{Bool} : Bool & \triangleq & \lambda(A)\, \lambda(x:A)\, \lambda(y:A)\, x \\ false_{Bool} : Bool & \triangleq & \lambda(A)\, \lambda(x:A)\, \lambda(y:A)\, y \\ true_{True} : True & \triangleq & \lambda(A)\, \lambda(x:A)\, \lambda(y:Top)\, x \\ false_{False} : False & \triangleq & \lambda(A)\, \lambda(x:Top)\, \lambda(y:A)\, y \end{array}$$

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. This is somewhat surprising because computationally there are only two booleans. Intuitively, if two arguments of an arbitrary type are given, there are only two ways of providing a result of that type. This coincides with the fact that by removing all the type information in the terms above, we obtain only two distinct untyped terms. Fortunately, 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 Naturals

The encoding of booleans in the previous section does not seem to generalize to other algebraic types. A different style of encoding (which can also be applied to booleans) works better for naturals. In the following encoding, *Nat* stands for the type of naturals, Nat_z for the type of zero naturals (the singleton zero), and Nat_s for the type of non-zero naturals.

$$\begin{aligned} Nat &\triangleq \forall(N)\, \forall(N_z<:N)\, \forall(N_s<:N)\, N_z \to (N \to N_s) \to N \\ Nat_z &\triangleq \forall(N)\, \forall(N_z<:N)\, \forall(N_s<:N)\, N_z \to (N \to N_s) \to N_z \\ Nat_s &\triangleq \forall(N)\, \forall(N_z<:N)\, \forall(N_s<:N)\, N_z \to (N \to N_s) \to N_s \end{aligned}$$

The closed normal forms of minimal type for *Nat* are the usual Church numerals; for Nat_z we have only the zero natural, and for Nat_s the non-zero naturals. We obtain:

$$Nat_z <: Nat, \quad Nat_s <: Nat$$

$$\begin{aligned} &zero: Nat_z \triangleq \\ &\quad \lambda(N)\, \lambda(N_z<:N)\, \lambda(N_s<:N)\, \lambda(z:N_z)\, \lambda(s:N \to N_s)\, z \end{aligned}$$

$$\begin{aligned} &succ: Nat \to Nat_s \triangleq \\ &\quad \lambda(n:Nat) \\ &\qquad \lambda(N)\, \lambda(N_z<:N)\, \lambda(N_s<:N)\, \lambda(z:N_z)\, \lambda(s:N \to N_s) \\ &\qquad\quad s(n(N)(N_z)(N_s)(z)(s)) \end{aligned}$$

3.3 Products

The standard encoding for pairs in *F*, shown below, already exhibits useful subtyping properties.

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

Both *A* and *B* occur in monotonic positions in $A \times B$, being placed on the left of an $\to$ which is on the left of another $\to$. Hence we obtain the expected monotonic inclusion of products as a derived rule:

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

The operations on pairs are defined, as usual, as:

pair: $\forall(A)\ \forall(B)\ A \to B \to A \times B$
$\triangleq\ \lambda(A)\ \lambda(B)\ \lambda(a{:}A)\ \lambda(b{:}B)\ \lambda(C)\ \lambda(f{:}A \to B \to C)\ f(a)(b)$

fst: $\forall(A)\ \forall(B)\ A \times B \to A$
$\triangleq\ \lambda(A)\ \lambda(B)\ \lambda(c{:}A \times B)\ c(A)(\lambda(x{:}A)\lambda(y{:}B)x)$

snd: $\forall(A)\ \forall(B)\ A \times B \to B$
$\triangleq\ \lambda(A)\ \lambda(B)\ \lambda(c{:}A \times B)\ c(B)(\lambda(x{:}A)\lambda(y{:}B)y)$

We often use the following abbreviations, disambiguated by context:

$a,b \equiv a,_{A\times B} b \equiv pair(A)(B)(a)(b)$
$fst(c) \equiv fst_{A\times B}(c) \equiv fst(A)(B)(c)$
$snd(c) \equiv snd_{A\times B}(c) \equiv snd(A)(B)(c)$

3.4 Simple tuples

A *tuple type* is an iterated product type. When the last factor of this iterated product is a type variable, we have an *extensible tuple type*. When it is *Top*, we have a *simple tuple type*. In this paper we discuss only simple tuple types.

$Tuple(Top) \triangleq Top$
$Tuple(A_1,..,A_n,Top) \triangleq A_1 \times (.. \times (A_n \times Top)..) \qquad n \geq 1$

With derived rule:

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

For example:

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

We note here that the type *Top* assumes a very useful role, in allowing a longer tuple type to be a subtype of a shorter tuple type. The intuition is that a longer tuple value can always be regarded as a shorter tuple value, by "forgetting" the additional components, and this is possible since everything is forgotten in *Top*.

For tuple values we have:

$tuple(top) \triangleq top$
$tuple(a_1,..,a_n,top) \triangleq a_1,(..,(a_n, top)..) \qquad n \geq 1$

with derived rules:

$$\frac{E \vdash a_1 : A_1 \;..\; E \vdash a_n : A_n}{E \vdash tuple(a_1,..,a_n,top) : Tuple(A_1,..,A_n,Top)}$$

$$\frac{E \vdash a_1 \leftrightarrow b_1 : A_1 \;..\; E \vdash a_n \leftrightarrow b_n : A_n}{E \vdash tuple(a_1,..,a_n,top) \leftrightarrow tuple(b_1,..,b_n,top) : Tuple(A_1,..,A_n,Top)}$$

The basic tuple operations 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 product operations; again, we omit some typing information:

$$a\lfloor i \;\equiv\; snd^i(a)$$

$$a.i \;\equiv\; fst(a\lfloor i)$$

We obtain the derived rules:

$$\frac{E \vdash a : Tuple(A_0,..,A_n,Top)}{E \vdash a\lfloor i : Tuple(A_i,..,A_n,Top)} \quad n \geq 0,\; i \in 0..n+1$$

$$\frac{E \vdash a : Tuple(A_0,..,A_n,Top)}{E \vdash a.i : A_i} \quad n \geq 0,\; i \in 0..n$$

$$\frac{E \vdash a_0 : A_0 \;..\; E \vdash a_n : A_n}{E \vdash tuple(a_0,..,a_n,top)\lfloor i \leftrightarrow tuple(a_i,..,a_n,top) : Tuple(A_i,..,A_n,Top)} \quad \begin{array}{l} n \geq 0 \\ i \in 0..n+1 \end{array}$$

$$\frac{E \vdash a_0 : A_0 \;..\; E \vdash a_n : A_n}{E \vdash tuple(a_0,..,a_n,top).i \leftrightarrow a_i : A_i} \quad n \geq 0,\; i \in 0..n$$

3.5. Simple records

We restrict ourselves to the encoding of *simple records* (the ones with a fixed number of components [CL 90]); *extensible records* are treated in [Car 91].

Let $\mathcal{L}$ be a countable set of *labels*, enumerated by a bijection $\iota \in \mathcal{L} \rightarrow Nat$. 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 $\sigma \in 1..n \rightarrow Nat$.

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 is allocated 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 [Car 88]:

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

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. For example:

$$Rcd(l^0{:}A,\ l^1{:}B,\ l^2{:}C,\ Top) \equiv Tuple(A,\ B,\ C,\ Top)$$
$$<:\ Tuple(Top,\ B,\ Top) \equiv Rcd(l^1{:}B,\ Top)$$

Record values are similarly encoded, for example:

$$rcd(l^2{=}c,\ l^0{=}a,\ top) \ \triangleq\ tuple(a,\ top,\ c,\ top)$$

from which we obtain the rules:

$$\frac{E \vdash a_1 : A_1 \;..\; E \vdash a_n : A_n}{E \vdash rcd(l_1{=}a_1,..,l_n{=}a_n,top) : Rcd(l_1{:}A_1,..,l_n{:}A_n,Top)}$$

$$\frac{E \vdash a_1 \leftrightarrow a'_1 : A_1 \;..\; E \vdash a_n \leftrightarrow a'_n : A_n}{E \vdash rcd(l_1{=}a_1,..,l_n{=}a_n,top) \leftrightarrow rcd(l_1{=}a'_1,..,l_n{=}a'_n,top) : Rcd(l_1{:}A_1,..,l_n{:}A_n,Top)}$$

Record selection is encoded as follows:

$$r.l_i \ \triangleq\ r.\iota(l_i)$$

$$\frac{E \vdash r : Rcd(l{:}A,Top)}{E \vdash r.l : A}$$

Note that, by subsumption, we have the following as (further) derived rules:

$$\frac{E \vdash a_1 : A_1 \;..\; E \vdash a_n : A_n \;..\; E \vdash a_m : A_m}{E \vdash rcd(l_1{=}a_1,..,l_n{=}a_n,..,l_m{=}a_m,top) : Rcd(l_1{:}A_1,..,l_n{:}A_n,Top)}$$

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

$$\frac{E \vdash r : Rcd(l_1{:}A_1,..,l_n{:}A_n,Top)}{E \vdash r.l_i : A_i} \qquad i \in 1..n$$

The second rule above is particularly interesting. It expresses a form of observational equivalence: 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*.

3.6. Lists

Following the pattern used in the encoding of Naturals, we can define the algebra of parametric lists [BB 85]. *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:

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

$$nil: \forall(A)\, List[A] \triangleq \lambda(A)\, \lambda(L)\, \lambda(n:L)\, \lambda(c:A \to L \to L)\, n$$

$$cons: \forall(A)\, A \to List[A] \to List[A] \triangleq \lambda(A)\, \lambda(hd:A)\, \lambda(tl:List[A])\, \lambda(L)\, \lambda(n:L)\, \lambda(c:A \to L \to L)\, c(hd)(tl(L)(n)(c))$$

$$length: \forall(A)\, List[A] \to Nat \triangleq \lambda(A)\, \lambda(l:List[A])\, l(Nat)(zero)(\lambda(a:A)\lambda(n:Nat)succ(n))$$

As an application of *(Eq appl2)* we can now show some interesting facts. Namely, any two null 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:

$\vdash nil(B) \leftrightarrow nil(C) : List[Top]$ *(Eq appl2)*

$\vdash length(Top)(nil(B)) \leftrightarrow length(Top)(nil(C)) : Nat$ *(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 in [Wad 89] using only the types of terms. A difference is that our

proof is carried out within $F_{<:}$, whereas Wadler uses semantic 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 underlying equational theory is the one of $F_{<:}$. In the same vein, motivated by the semantic isomorphisms obtained in [BFSS 90] and [Fre 91] 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 that given a typed λ-calculus language and a λ-theory **T**, a category *Cl(***T***)* is determined by taking as objects of *Cl(***T***)* the (closed) types of **T** [LS 86] [MS 89]. 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 $\ulcorner _ \urcorner : \mathbf{C} \rightarrow \mathbf{Sets}$ given by:

For any object A:
$\ulcorner A \urcorner = \mathbf{C}(1,A)$ (the set of all morphisms $1 \rightarrow A$)
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 (as defined in 4.2.5). Given $f,g \in \mathbf{C}(A,B)$, $\ulcorner f \urcorner$ 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 $\ulcorner _ \urcorner : \mathbf{C} \rightarrow \mathbf{Sets}$

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)

For $\vdash f,f'{:}A\to B$, we say $\vdash f \leftrightarrow^{cl} f' : A\to B$ iff
for all a, $\vdash a{:}A \Rightarrow \vdash f(a) \leftrightarrow f'(a) : B$

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 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 $Cl(F_{<:})$, the mappings from $\ulcorner A\urcorner$ to $\ulcorner B\urcorner$ given by $\ulcorner f\urcorner([z{:}Top \vdash a{:}A]) = [z{:}Top \vdash t\{x\leftarrow a\}{:}B]$ for any $[z{:}Top \vdash a{:}A]$. 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* judgment is

$$id_A \triangleq \vdash \lambda(x{:}A)x : A\to A$$

and the c*omposition* judgment is, for any $\vdash h{:}A\to B$ and $\vdash g{:}B\to C$:

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

(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

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

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

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 $S \quad\Rightarrow\quad E,X<:B,E' \vdash S\{X\leftarrow C\} <: S\{X\leftarrow D\}$

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

Proof

By induction on the derivation $E,X<:B,E' \vdash S\ type$. The only less trivial case is *(Type ∀)*. Assume X positive in $\forall(Y<:S1)S2$. By induction hypothesis:

$E,X<:B,E' \vdash S1\{X\leftarrow D\} <: S1\{X\leftarrow C\}$

From $E,X<:B,E',Y<:S1 \vdash S2\ type$, by bound change lemma:

$E,X<:B,E',Y<:S1\{X\leftarrow D\} \vdash S2\ type$

Now conclude by induction and *(Sub ∀)*. □

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

Proof

Let $S\equiv\forall(Y_1<:B_1)\ldots\forall(Y_k<:B_k)T_1\to(\ldots\to(T_h\to X)\ldots)$.

By type monotonicity lemma,

$E,X<:Top \vdash S <: S\{X\leftarrow Top\}$ and $E,X<:Top \vdash S\{X\leftarrow D\} <: S\{X\leftarrow Top\}$.

Let $F\equiv Y_1<:B_1\{X\leftarrow Top\},\ldots,Y_k<:B_k\{X\leftarrow Top\},\ t_1:T_1\{X\leftarrow Top\},\ldots,t_h:T_h\{X\leftarrow Top\}$.

By *(Val x)*, weakening, and *(Subsumption)*,

$E,X<:Top,x:S,F \vdash x : S\{X\leftarrow Top\}$

by *(Eq appl2)* and *(Eq appl)*,

$E,X<:Top,x:S,F \vdash x(Y_1)\ldots(Y_k)(t_1)\ldots(t_h) : Top$

Analogously, from $E \vdash s : S\{X\leftarrow D\}$ we obtain:

$E,X<:Top,x:S,F \vdash s: S\{X\leftarrow Top\}$

and then:

$E,X<:Top,x:S,F \vdash s(Y_1)\ldots(Y_k)(t_1)\ldots(t_h) : Top$

By *(Eq collapse)*,

$E,X<:Top,x:S,F \vdash x(Y_1)\ldots(Y_k)(t_1)\ldots(t_h) \leftrightarrow s(Y_1)\ldots(Y_k)(t_1)\ldots(t_h) : Top$

By *(Eq fun)*, *(Eq fun2)*, *(Eq eta)* and *(Eq eta2)*,

$E,X<:Top,x:S \vdash x \leftrightarrow s : S\{X \leftarrow Top\}$. □

By generalized collapse and the eq-substitution property (section 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 $E \vdash a \leftrightarrow a\{X \leftarrow D, x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\} : A$.

Proof

By generalized collapse lemma, , for $i=1..n$:

$E',X<:Top,x_i: S_i \vdash x_i \leftrightarrow t_i : S_i\{X \leftarrow Top\}$.

The eq-substitution proposition (Sect. 2.4) allows us to conclude. □

4.2 CL finite products and coproducts; well-pointedness

In this section we show that the equational theory of $F_{<:}$ is strong enough to entail some basic categorical properties of **CL**..

4.2.1 Terminal objects

Proposition

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

Proof

Take $1_C \triangleq \lambda(x:C)\ top$.

Take any other morphism $\vdash f : C \rightarrow Top$.

$x:C \vdash f : C \rightarrow Top$ (weaken)

$x:C \vdash f(x) \leftrightarrow top : Top$ *(Eq collapse)*

$\vdash \lambda(x:C)\ f(x) \leftrightarrow \lambda(x:C)\ top : C \rightarrow Top$ *(Eq fun)*

$\vdash f \leftrightarrow 1_C : C \rightarrow Top$ *(Eq eta)*

A fortiori, $\vdash f \leftrightarrow^{cl} 1_C : C \rightarrow Top$. □

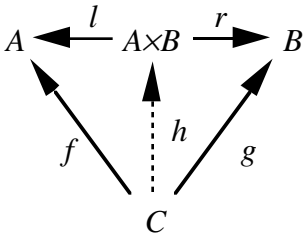
4.2.2 Binary products

Definition

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

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



Proof

Define:

$px \triangleq \lambda(x{:}A)\lambda(y{:}B)x$
$py \triangleq \lambda(x{:}A)\lambda(y{:}B)y$

$l \triangleq \lambda(p{:}A \times B)p(A)(px)$ then $\vdash l{:}A \times B \to A$
$r \triangleq \lambda(p{:}A \times B)p(B)(py)$ then $\vdash r{:}A \times B \to B$
$pair \triangleq \lambda(a{:}A)\lambda(b{:}B)\lambda(C)\lambda(q{:}A \to B \to C)q(a)(b)$
then $\vdash pair : A \to B \to A \times B$
$couple \triangleq \lambda(C)\lambda(f{:}C \to A)\lambda(g{:}C \to B)\lambda(c{:}C)pair(f(c))(g(c))$
then $\vdash couple : \forall(C)\,(C \to A) \to (C \to B) \to C \to (A \times B)$

Fix an object $\vdash C\ type$ and two morphisms $\vdash f{:}C \to A$ and $\vdash g{:}C \to B$.

1) Existence.
Take $h \triangleq couple(C)(f)(g) \leftrightarrow \lambda(c{:}C)pair(f(c))(g(c))$
$\vdash l \circ h \leftrightarrow \lambda(z{:}C)l(h(z)) \leftrightarrow \lambda(z{:}C)f(z) \leftrightarrow f : C \to A$
$\vdash r \circ h \leftrightarrow \lambda(z{:}C)r(h(z)) \leftrightarrow \lambda(z{:}C)g(z) \leftrightarrow g : C \to B$

2) The morphism above is well defined. Just show that:
$\vdash f' \leftrightarrow^{cl} f : C \to A, \vdash g' \leftrightarrow^{cl} g : C \to B$ implies
$\vdash couple(C)(f)(g) \leftrightarrow^{cl} couple(C)(f')(g') : C \to A \times B$

3) Uniqueness.

3.1) Show, for $\vdash c{:}A \times B$, that $\vdash couple(A \times B)(l)(r)(c) \leftrightarrow c : A \times B$
The normal form of c must have the shape:
$c \equiv \lambda(C)\lambda(q{:}D)q(a)(b)$
for some $C{<:}Top \vdash A \to B \to C{<:}D$, $C{<:}Top, q{:}D \vdash a{:}A$, and $C{<:}Top, q{:}D \vdash b{:}B$.
By the bound weakening lemma,
$C{<:}Top, q{:}A \to B \to C \vdash a{:}A$, and $C{<:}Top, q{:}A \to B \to C \vdash b{:}B$
and by *(Eq fun')*, for $c' \triangleq \lambda(C)\lambda(q{:}A \to B \to C)q(a)(b)$,

$\vdash c \leftrightarrow c' : A \times B$.

By β-conversion

$\vdash l(c) \leftrightarrow c(A)(px) \leftrightarrow a\{C \leftarrow A, q \leftarrow px\} : A$ Let $a1 \triangleq a\{C \leftarrow A, q \leftarrow px\}$

$\vdash r(c) \leftrightarrow c(B)(py) \leftrightarrow b\{C \leftarrow B, q \leftarrow py\} : B$ Let $b1 \triangleq a\{C \leftarrow B, q \leftarrow py\}$.

By the eq-var-substitution lemma,

$C <: Top, q : A \rightarrow B \rightarrow C \vdash a \leftrightarrow a1 : A$

$C <: Top, q : A \rightarrow B \rightarrow C \vdash b \leftrightarrow b1 : B$

$C <: Top, q : A \rightarrow B \rightarrow C \vdash q(a)(b) \leftrightarrow q(a1)(b1) : C$ *(Eq-appl)*

$\vdash \lambda(C)\lambda(q : A \rightarrow B \rightarrow C)q(a)(b) \leftrightarrow \lambda(C)\lambda(q : A \rightarrow B \rightarrow C)q(a1)(b1)$
$: A \times B$ *(Eq fun, Eq fun2)*

Hence:

$\vdash couple(A \times B)(l)(r)(c) \leftrightarrow pair(l(c))(r(c))$
$\leftrightarrow \lambda(C)\lambda(q : A \rightarrow B \rightarrow C)q(a1)(b1) \leftrightarrow \lambda(C)\lambda(q : A \rightarrow B \rightarrow C)q(a)(b)$
$\leftrightarrow c' \leftrightarrow c : A \times B$

3.2) Show, by β-conversion, that for any $\vdash D\ type$, $\vdash k : D \rightarrow C$, and $\vdash d : D$,

$\vdash couple(D)(f \circ k)(g \circ k)(d) \leftrightarrow (couple(C)(f)(g) \circ k)(d) : A \times B$

That h is cl-unique now follows by the usual argument. □

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

Proof

Standard diagram chasing, from the existence of products. □

4.2.3 Initial objects

Definition

$Bot \triangleq \forall(X)X$

Proposition

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

Proof

Take $0_C \triangleq \lambda(x : Bot)\, x(C)$.

Take any other morphism $\vdash f : Bot \rightarrow C$.

Since there are no terms c such that $\vdash c : Bot$, then it is vacuously true that for all $\vdash c : Bot$, $\vdash f(c) \leftrightarrow 0_C(c) : C$,

that is, that $\vdash f \leftrightarrow^{cl} 0_C : Bot \rightarrow C$. □

Remark

$Bool \rightarrow Bot$ is also an initial object, by the same argument, since there are no terms of type $Bool \rightarrow Bot$. The unique map is the equivalence class of $\lambda(x : Bool \rightarrow Bot)\, x(true)(C)$, which includes $\lambda(x : Bool \rightarrow Bot)\, x(false)(C)$. 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 $\vdash c:V$.

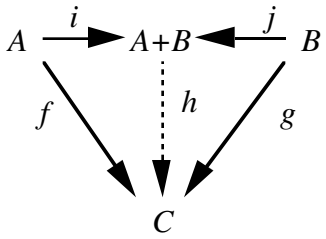
4.2.4 Binary coproducts

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

0) Let $\vdash c:A+B$; then the normal form of c must have one of the shapes:

$c \equiv \lambda(C')\lambda(f':D)\lambda(g':G)f'(a)$
for some $C'<:Top \vdash A\to C <:D$, $C'<:Top \vdash B\to C'<:G$, and
$C'<:Top, f':D, g':G \vdash a:A$

$c \equiv \lambda(C')\lambda(f':D)\lambda(g':G)g'(b)$
for some $C'<:Top \vdash A\to C <:D$, $C'<:Top \vdash B\to C'<:G$, and
$C'<:Top, f':D, g':G \vdash b:B$

By the bound weakening lemma,

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

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

and, by *(Eq fun')*,

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

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

Fix an object $\vdash C\ type$ and two morphisms $\vdash f:A\to C$ and $\vdash g:B\to C$.

1) Existence

Take $h \triangleq case(C)(f)(g)$.

$\vdash h\circ i \leftrightarrow \lambda(x:A)h(i(x)) \leftrightarrow \lambda(x:A)f(x) \leftrightarrow f : A\to C$

$\vdash h\circ j \leftrightarrow \lambda(x:A)h(j(x)) \leftrightarrow \lambda(x:A)g(x) \leftrightarrow g : B\to C$

2) The morphism above is well defined.
Show $\vdash f'' \leftrightarrow^{cl} f : A \to C, \vdash g'' \leftrightarrow^{cl} g : B \to C$ implies
$\vdash case(C)(f)(g) \leftrightarrow^{cl} case(C)(f'')(g'') : A+B \to C$
That is, for $\vdash c:A+B$,
$\vdash case(C)(f)(g)(c) \leftrightarrow case(C)(f'')(g'')(c) : C$
By (0) and β-conversion, either
$\vdash case(C)(f)(g)(c) \leftrightarrow f(a\{C' \leftarrow C, f' \leftarrow f, g' \leftarrow g\}) : C$ and
$\vdash case(C)(f'')(g'')(c) \leftrightarrow f''(a\{C' \leftarrow C, f' \leftarrow f'', g' \leftarrow g''\}) : C$
or
$\vdash case(C)(f)(g)(c) \leftrightarrow g(b\{C' \leftarrow C, f' \leftarrow f, g' \leftarrow g\}) : C$ and
$\vdash case(C)(f'')(g'')(c) \leftrightarrow g''(b\{C' \leftarrow C, f' \leftarrow f'', g' \leftarrow g''\}) : C$
In the first case (the other one is similar), the eq-var-substitution lemma gives:
$C' <: Top, f' : A \to C', g' : B \to C' \vdash a \leftrightarrow a\{C' \leftarrow C, f' \leftarrow f, g' \leftarrow g\} : A$ and also
$C' <: Top, f' : A \to C', g' : B \to C' \vdash a \leftrightarrow a\{C' \leftarrow C, f' \leftarrow f'', g' \leftarrow g''\} : A$
from which we infer:
$\vdash a\{C' \leftarrow C, f' \leftarrow f'', g' \leftarrow g''\} \leftrightarrow a\{C' \leftarrow C, f' \leftarrow f, g' \leftarrow g\} : A$
since both terms are closed. Now conclude by using $\vdash f'' \leftrightarrow^{cl} f : A \to C$.

3) Uniqueness.
3.1) Show, for $\vdash c:A+B$, that $\vdash case(A+B)(i)(j)(c) \leftrightarrow c : A+B$.
By cases on the normal form of c, according to (0).
In the first case,
$\vdash case(A+B)(i)(j)(c) \leftrightarrow c(A+B)(i)(j) \leftrightarrow i(a\{C' \leftarrow A+B, f' \leftarrow i, g' \leftarrow j\}) : A+B$
Let $a1 \triangleq a\{C' \leftarrow A+B, f' \leftarrow i, g' \leftarrow j\}$. By the eq-var-substitution lemma,
$C' <: Top, f' : A \to C', g' : B \to C' \vdash a1 \leftrightarrow a : A$
$C' <: Top, f' : A \to C', g' : B \to C' \vdash f'(a1) \leftrightarrow f'(a) : C'$ *(Eq appl)*
$\vdash \lambda(C')\lambda(f' : A \to C')\lambda(g' : B \to C')f'(a1)$
$\leftrightarrow \lambda(C')\lambda(f' : A \to C')\lambda(g' : B \to C')f'(a) : A+B$ *(Eq fun, Eq fun2)*
$\vdash i(a1) \leftrightarrow c : A+B$ (def)
$\vdash case(A+B)(i)(j)(c) \leftrightarrow c : A+B$ (equation above)

The second case is similar.

3.2) Show, for any $\vdash D\ type$, $\vdash k:C \to D$, and $\vdash c:A+B$,
$\vdash case(D)(k \circ f)(k \circ g)(c) \leftrightarrow (k \circ case(C)(f)(g))(c) : D$.

By cases on the normal form of c, according to (0).
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 the 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.

(4) Uniqueness can now be shown by the standard argument. □

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

Proof

Standard diagram chasing, from the existence of coproducts. □

4.2.5 Well-pointedness

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:

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

Proof

$\Rightarrow$)

$x:Top \vdash f(h(x)) \leftrightarrow f(h(top)) : B$	*(Eq collapse)* and *(Eq appl)*
$x:Top \vdash g(h(x)) \leftrightarrow g(h(top)) : B$	similarly
$x:Top \vdash f(h(top)) \leftrightarrow g(h(top)) : B$	hypothesis, weaken
$\vdash \lambda(x:Top)\, f(h(x)) \leftrightarrow \lambda(x:Top)\, g(h(x)) : Top\to B$	*(Eq trans)* and *(Eq fun)*

Hence $\vdash f\circ h \leftrightarrow g\circ h : Top\to B$.

$\Leftarrow$)

Take $\vdash a : A$, consider $h=\lambda(x:Top)a$.

$\vdash (f\circ h)(top) \leftrightarrow (g\circ h)(top) : B$	hypothesis
$\vdash f(a) \leftrightarrow g(a) : B$	*(Eq beta)*

Hence $\vdash f \leftrightarrow^{cl} g : A\to B$. □

4.3 CL isomorphisms

The following isomorphisms were inspired by [BFSS 90] and [Fre 91].

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 [BFSS 90], but which has no known proof in *F*. (See the remark at the end of this section.)

Proposition

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

Proof

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

$g \triangleq \lambda(y{:}A)\ \lambda(C)\ \lambda(z{:}A\to C)\ z(y)$

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:

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

$\leftrightarrow (\lambda(C)\ \lambda(z{:}A\to C)\ z(a))(A)(id(A))$

$\leftrightarrow id(A)(a) \leftrightarrow a : A$

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 the 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 the 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:

$\lambda(P)\ \lambda(x{:}(T\to T)\to P)$

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

$: \forall(P)((T\to T)\to P)\to P$

is not convertible to any term of the form:

$\lambda(P)\ \lambda(x{:}(T\to T)\to P)\ x(c)$

where c is a closed term of type $T\to T$.

Moreover, Roberto Di Cosmo [DiC 91] has shown that A is not isomorphic to $\forall(C)(A\to C)\to C$ in F in the usual sense of F-isomorphisms, as opposed to cl-isomorphisms.

4.3.2 Existentials

We prove in this section that the terminal type *Top* is isomorphic in **CL** to $\exists(X)X$. From the programming point of view this is consistent with the intuition that, although any value can be encapsulated as an object of type $\exists(X)X$, there is no way of using an object of this type. We will prove, more generally, that $\exists(X{<:}A)X \sim A$ (i.e. $\vdash \exists(X{<:}A)X \sim^{cl} A$)

Lemma 1

$E\vdash B\ type,\ E\vdash y : \forall(X{<:}A)X\to B,\quad E\vdash A' <: A,\quad E\vdash a' : A',\quad E\vdash a'\leftrightarrow a : A$

$\Rightarrow\quad E\vdash y(A)(a) \leftrightarrow y(A')(a') : B$

Proof

First,

$E\vdash y \leftrightarrow y : \forall(X{<:}A)X\to B$ hypothesis, *(Eq x)*

$E\vdash y(A) \leftrightarrow y(A) : A\to B$ *(Eq appl2)*, since $X\notin FV(B)$, by $E\vdash B\ type$

$E\vdash y(A)(a) \leftrightarrow y(A)(a') : B$ hypothesis, *(Eq appl)*

Then,

$E\vdash y \leftrightarrow y : \forall(X{<:}A)X\to B$ hypothesis, *(Eq x)*

$E\vdash y(A) \leftrightarrow y(A') : A'\to B$ *(Eq appl2)*

$E\vdash y(A)(a') \leftrightarrow y(A')(a') : B$ hypothesis, *(Eq appl)*

Finally,

$E\vdash y(A)(a) \leftrightarrow y(A')(a') : B.$ □

Definition

Let $id : \forall(A)\ \forall(W{<:}A)\ W\to W\ \triangleq\ \lambda(A)\ \lambda(W{<:}A)\ \lambda(w{:}W)\ w$

Definition

$\exists(W{<:}A)B\ \triangleq\ \forall(V)(\forall(W{<:}A)B\to V)\to V$

$some : \forall(A)\ \forall(X{<:}A)\ X\to\exists(W{<:}A)W$

$\triangleq\ \lambda(A)\ \lambda(X{<:}A)\ \lambda(x{:}X)$

$\lambda(V)\ \lambda(z{:}\forall(W{<:}A)W\to V)\ z(X)(x)$

Proposition

$\vdash A\ type\ \Rightarrow\ \vdash A\sim^{cl}\exists(X{<:}A)X$

Proof

Let $\vdash f : (\exists(W{<:}A)W)\to A$

where $f = \lambda(p{:}\exists(W{<:}A)W)p(A)(id(A))$

Let $\vdash g : A \to (\exists(W{<:}A)W)$

where $g = \lambda(x{:}A)some(A)(A)(x)$

Take a such that $\vdash a{:}A$. Then

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

$\leftrightarrow f(\lambda(V)(\lambda(z{:}\forall(W{<:}A)W{\to}V)z(A)(a))$

$\leftrightarrow (\lambda(V)\lambda(z{:}\forall(W{<:}A)W{\to}V)z(A)(a))\ (A)\ (id(A))$

$\leftrightarrow id(A)(A)(a)$

$\leftrightarrow a : A$

Take closed b such that $\vdash b : \exists(W{<:}A)W$.

Then b has a normal form of the shape:

$b = \lambda(V)\lambda(z{:}D)z(B1)(b1)$

for some $D, B1, b1$ such that:

$V{<:}Top \vdash \forall(W{<:}A)W{\to}V <: D$

$V{<:}Top, z{:}D \vdash b1 : B1 <: A$

By the bound weakening lemma, and *(Eq fun')*

$\vdash b \leftrightarrow \lambda(V)\lambda(z{:}\forall(W{<:}A)W{\to}V)z(B1)(b1)$

Then

$\vdash g(f(b)) \leftrightarrow g(b(A)(id(A)))$

$\leftrightarrow g(id(A)(B1\{V{\leftarrow}A\})(b1\{V{\leftarrow}A, z{\leftarrow}id(A)\})$

$\leftrightarrow g(b1\{V{\leftarrow}A, z{\leftarrow}id(A)\})$

$\leftrightarrow some(A)(A)(b1\{V{\leftarrow}A, z{\leftarrow}id(A)\})$

$\leftrightarrow \lambda(V)\lambda(z{:}\forall(W{<:}A)W{\to}V)\ z(A)(b1\{V{\leftarrow}A, z{\leftarrow}id(A)\})$

$: \exists(W{<:}A)W$

By the eq-var-substitution lemma, since

$\vdash id(A) : \forall(W{<:}A)W{\to}W <: \forall(W{<:}A)W{\to}A,$

$V{<:}Top, z{:}\forall(W{<:}A)W{\to}V \vdash b1 \leftrightarrow b1\{V{\leftarrow}A, z{\leftarrow}id(A)\} : A.$

Hence by Lemma 1,

$V{<:}Top, z{:}\forall(W{<:}A)W{\to}V \vdash z(A)(b1\{V{\leftarrow}A, z{\leftarrow}id(A)\}) \leftrightarrow z(B1)(b1) : V$

That is:

$\vdash \lambda(V)\lambda(z{:}\forall(W{<:}A)W{\to}V)\ z(A)(b1\{V{\leftarrow}A, z{\leftarrow}id(A)\})$

$\leftrightarrow \lambda(V)\lambda(z{:}\forall(W{<:}A)W{\to}V)\ z(B1)(b1)$

$: \exists(W{<:}A)W$

Combining the two equations above:

$\vdash g(f(b)) \leftrightarrow$

$\leftrightarrow \lambda(V)\lambda(z{:}\forall(W{<:}A)W{\to}V)\ z(B1)(b1)$

$\leftrightarrow b$

$: \exists(W{<:}A)W.$ □

Corollary

$\vdash Top \sim^{cl} \exists(X)X$

4.3.3 Other cl-isomorphisms

Many other isomorphisms can be derived with the techniques developed in the previous sections. Among them we have the following.

Domain restriction

$C \sim \forall(X)\, X \to C$

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

Categorical

$(A \times B) \times C \sim A \times (B \times C)$

$A \times Top \sim Top \times A \sim A$

$(A + B) + C \sim A + (B + C)$

$A + Bot \sim Bot + A \sim A$

Various

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

Conclusions

We study an extension of system F with subtyping and its equational theory. While the equational rules are not complete for PER models, the main inspirations for the most novel rules come from PER models and categorical notions of parametricity. Although our proof system is not a conservative extension of system *F*, we prove the conservativity of typing judgments with respect to *F*. We study some categorical properties of the theory when restricted to closed terms, including interesting categorical isomorphisms. These isomorphisms provide some confidence in the strength of the proof system. Additional evidence is given by a set of encodings; these include record operations and subtyping hierarchies that are related to features of object-oriented languages.

One important area we have not studied is an adequate computation system. Ideally we would like to have a notion of reduction such that any two provably equal terms reduce to a common term. If possible, we would like reductions to terminate as well. A standard approach is to orient each equational axiom in one direction. The two equational rules that lead to immediate problems are *(Eq collapse)* and *(Eq appl2)*; for these it is not obvious how to produce an oriented reduction rule. Furthermore, in order to capture equivalence, a set of oriented rules would have to be proved confluent. If we had a

computational characterization of equality, we would have decidability of the equational system; in its absence, decidability remains an open problem.

The final form of the *(Eq appl2)* rule is still under investigation. Some recent insights [ACC 93] seem to suggest that *(Eq appl2-+)* should be taken instead. Specifically, formal systems considered in [BFSS 90] and [ACC 93] have the latter as a consequence, but not the former. The *(Eq appl2)* rule was adopted here because it is valid in PER and has a simpler syntactic form.

Acknowledgments

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 $Top \sim \exists(X)X$.

Luca Cardelli also thanks Martín Abadi for his careful readings of the draft.

Appendix: System F

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)
$$\frac{\vdash E\ env \quad X \notin dom(E)}{\vdash E,X\ env}$$

Types

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

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

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

Values

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

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

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

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

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

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

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

References

[ACC 93] M.Abadi, L.Cardelli, P.-L.Curien: ***Formal parametric polymophism***, to appear.

[MS 91] J.C.Mitchell, A.Scedrov: ***Sconing, relators, and parametricity***, manuscript.

[BB 85] C.Böhm, A.Berarducci: ***Automatic synthesis of typed λ-programs on term algebras***, Theoretical Computer Science, 39, pp. 135-154, 1985.

[BFSS 90] E.S.Bainbridge, P.J.Freyd, A.Scedrov, P.J.Scott: ***Functorial polymorphism***, Theoretical Computer Science, vol.70, no.1, pp 35-64, 1990.

[BL 88] K.B.Bruce, G.Longo: ***A modest model of records, inheritance and bounded quantification***, Information and Computation, 87(1/2):196-240, 1990.

[Car 88] L.Cardelli: ***A semantics of multiple inheritance***, in Information and Computation 76, pp 138-164, 1988.

[Car 91] L.Cardelli: ***Extensible records in a pure calculus of subtyping***, to appear.

[CL 90] L.Cardelli, G.Longo: ***A semantic basis for Quest***, Proceedings of the 6th ACM LISP and Functional Programming Conference, ACM Press, 1990.

[CM 91] L.Cardelli, J.C.Mitchell: ***Operations on records***, Mathematical Structures in Computer Science, vol 1, pp.3-48, 1991.

[CW 85] L.Cardelli, P.Wegner: ***On understanding types, data abstraction and polymorphism***, Computing Surveys, Vol 17 n. 4, pp 471-522, December 1985.

[CG 91] P.-L.Curien, G.Ghelli: ***Coherence of subsumption***, Mathematical Structures in Computer Science, to appear. Short version in Proc. CAAP 90, Lecture Notes in Computer Science n.431.

[CG 91a] P.-L.Curien, G.Ghelli: ***Subtyping + extensionality: confluence of βη-top-reductions in $F_{\leq}$***, in T.Ito,A.R.Meyer Eds. Theoretical Aspects of Computer Software, Sendai, Japan, Lecture Notes in Computer Science n.526, pp. 731-749, Springer-Verlag, 1991

[deB 72] N.G.de Bruijn: ***Lambda-calculus notation with nameless dummies***, in Indag. Math. 34(5), pp. 381-392, 1972.

[DiC 91] R.Di Cosmo: ***Invertibility of terms and valid isomorphisms, a proof theoretic study on second order lambda-calculus with surjective pairing and terminal object***, Technical Report TR 10-91, LIENS Ecole Normale Supérieure, Paris, 1991.

[Fai 89] J.Fairbairn: ***Some types with inclusion properties in ∀, →, μ***, Technical report No 171, University of Cambridge, Computer Laboratory.

[Fre 91] P.J.Freyd: ***Structural polymorphism***, to appear in TCS.

[Ghe 90] 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.

[Gir 71] 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.

[LS 86] J.Lambek, P.J.Scott: ***Introduction to higher order categorical logic***, Cambridge University Press, 1986.

[Mit 90] 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.

[MS 89] 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.

[Pit 87] A.M.Pitts: ***Polymorphism is set-theoretic, constructively***, in Category Theory and Computer Science, Proceedings Edinburgh 1987, D.H.Pitt, A.Poigné, and D.E.Rydeheard Eds. Springer Lecture Notes in Computer Science, vol. 283, pp 12-39, 1987.

[Rey 74] 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.

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

[Sce 90] A.Scedrov: ***A guide to polymorphic types***, in Logic and Computer Science, pp 387-420, P.Odifreddi ed., Academic Press, 1990.

[Str 67] C.Strachey: ***Fundamental concepts in programming languages***, lecture notes for the International Summer School in Computer Programming, Copenhagen, August 1967.

[Wad 89] P.Wadler: ***Theorems for free!***, Proc. of the Fourth International Conference on Fuctional Programming and Computer Architecture, ACM Press, 1989.