Extensible Records in a Pure Calculus of Subtyping

Luca Cardelli
Digital Equipment Corporation, Systems Research Center
130 Lytton Avenue, Palo Alto CA 94301

Abstract

Extensible records were introduced by Mitchell Wand while studying type inference in a polymorphic λ-calculus with record types. This paper describes a calculus with extensible records, $F_{<:}\rho$, that can be translated into a simpler calculus, $F_{<:}$, lacking any record primitives. Given independent axiomatizations of $F_{<:}\rho$ and $F_{<:}$ (the former being an extension of the latter) we show that the translation preserves typing, subtyping, and equality.

$F_{<:}\rho$ can then be used as an expressive calculus of extensible records, either directly or to give meaning to yet other languages. We show that $F_{<:}\rho$ can express many of the standard benchmark examples that appear in the literature.

Like other record calculi that have been proposed, $F_{<:}\rho$ has a rather complex set of rules but, unlike those other calculi, its rules are justified by a translation to a very simple calculus. We argue that thinking in terms of translations may help in simplifying and organizing the various record calculi that have been proposed, as well as in generating new ones.

SRC Research Report 81, January 3, 1992. Revised January 1, 1993.
© Digital Equipment Corporation 1992,1993.
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.

Contents

1. Introduction
2. System $F_{<:}$
    2.1 Syntax
    2.2 Rules
3. Basic encodings
    3.1 Booleans
    3.2 Products
    3.3 Enumerations
    3.4 Tuples
4. Records
    4.1 Simple records
    4.2 Extensible records
5. System $F_{<:}\rho$
    5.1 Syntax
    5.2 Rules
    5.3 Properties
    5.4 Some useful extensions
        5.4.1 Recursive types
        5.4.2 Label sets
        5.4.3 Definitions
    5.5 Examples
6. Translation of $F_{<:}\rho$ into $F_{<:}$
7. The translation preserves derivations
8. Conclusions
Acknowledgements
References

1. Introduction

Extensible records, and the associated notion of *row variables*, were introduced by Mitchell Wand while he was studying the problem of type inference in a polymorphic λ-calculus with record types [Wand 87]; a row variable is a type variable ranging over the possible field-extensions of a record type. Many calculi of row variables have been produced since then [Jategaonkar Mitchell 88] [Rémy 89] [Wand 89] [Harper Pierce 90] [Cardelli Mitchell 91], and many more can be imagined. As we try to increase the expressiveness of these calculi, the axiomatization techniques become more and more divergent and complex. To be able to compare and discuss these different calculi, we feel the need of some more fundamental framework. This paper suggests that a very simple calculus of subtyping can be used as a basis for studying much more complex calculi of extensible records.

In the search for a unifying framework, we can adopt the following working hypothesis: every reasonable calculus of row variables should be reducible to a calculus without row variables, via a well-behaved translation. The purpose of this hypothesis is not to eliminate row variables completely, since the translated programs would become too verbose to be useful; the purpose is to gain insights in the study of calculi with row variables. Even if our working hypothesis turns out to be false, which it may well be, we will have distinguished the easier features that can be translated from the more complex ones that cannot.

To carry out this plan, we need to fix a suitable target calculus for the translation. Since we are studying type variables, a likely choice would seem to be the second-order λ-calculus (*system F* [Girard 71] [Reynolds 74]). To express the idea that the translation is *well-behaved*, we require some basic soundness properties such as the preservation of typing, subtyping, and equality relations. But, in order to preserve subtyping relations, we need to translate to a target calculus that still has a notion of subtyping; otherwise we would gain little insight about the complex subtyping relations induced by extensible records. For a similar reason, we are not interested in untyped target calculi, for which translations are easily obtainable.

As target calculus we use therefore an extension of F with subtyping, called $F_{<:}$ (*F-sub*), which has been studied recently [Curien 90] [Curien Ghelli 91] [Cardelli Martini Mitchell Scedrov 91]. The fact that a translation of extensible records into $F_{<:}$ is at all possible also gives us new evidence about the expressiveness of $F_{<:}$, and reinforces our feeling that $F_{<:}$ can be regarded as a canonical calculus of subtyping.

Before the main discussion, we briefly review the motivations that led to the notions of row variables and extensible records.

In a calculus with records, a program may contain expressions like $r.l$ where r denotes a record value and the label l denotes a field of that record; then the *record selection* $r.l$ denotes the value of the field labeled l in record r.

Given the expression $r.l$ we can infer that r has a type of the form $Rcd(l:A)$, that is, a record type having a field labeled l of type A; the type A is to be determined later. Given another expression $r.l'$ in the same program, we can then infer that r has a type of the form $Rcd(l:A,l':A')$, and so on.

This form of typing, though, becomes insufficient when considering *record updates*. The expression $r.l \leftarrow a$ denotes a record similar to r, except that the value of its l component is updated to a. Consider now the program:

$$p \triangleq \lambda(r)\ r.l \leftarrow a$$

Assuming $a:A$, and for any type B, we can infer the typing:

$$p: Rcd(l:B) \rightarrow Rcd(l:A)$$

Given a record value $rcd(l=b,l'=b')$, having two fields labeled l and l' with respective values b and b', we consider legal the expression $p(rcd(l=b,l'=b'))$ because the argument has all the fields required by the type of p. This expression then receives the type $Rcd(l:A)$, because of the typing of p above. Unfortunately, by this typing we have forgotten that the argument of p, and hence its result, has another component labeled l'. This is unsatisfactory.

To capture the kind of polymorphism required by the record update operation, we introduce row variables. Record types are extended to the more general form $Rcd(l_1:A_1,..,l_n:A_n,X)$, where X is a *row variable* intended to represent "all the other fields" of a given record type; in this case all the fields except the ones labeled $l_1..l_n$. We can then assign to the program p the more informative type:

$$p: Rcd(l:B,X) \rightarrow Rcd(l:A,X)$$

Now, in $p(rcd(l=b,l'=b',l''=b''))$, where $b':B'$ and $b'':B''$, the row variable X is bound to $l':B',l'':B''$ (a *row type*), producing the expected result type $Rcd(l:A,l':B',l'':B'')$ by substitution of $l':B',l'':B''$ for X.

In this form of type inference we must keep track of constraints on the row variables, such as the fact that X in the example above must not come to contain l components (otherwise we would have a duplicate label). These constraints can be made manifest by adopting a type system featuring explicit polymorphism; then program p receives the typing:

$$p : \forall(Y)\ \forall(X \uparrow l)\ Rcd(l:Y,X) \rightarrow Rcd(l:A,X)$$

Here $X \uparrow l$ means that X is undefined at label l (that is, X can be bound only to row types that have no l components). Appropriate types and rows must then be explicitly supplied as arguments to p, as in:

$$p(B)(l':B',l'':B'')(rcd(l=b,l'=b',l''=b''))$$

This is finally a satisfactory typing of p, although for practical reasons we may require some type inference to avoid writing down the type arguments *(B)* and *(l':B',l'':B'')*. We do not discuss type inference here, which we consider as a pragmatic variation on the basic calculus.

In Wand's original view, and in further developments [Rémy 89] [Harper Pierce 90], row variables are type variables of a different *kind*. In contrast, in [Cardelli Mitchell 91] we studied an explicitly polymorphic type system where both row variables and type variables are instances of second-order type variables, therefore unifying the two concepts. In this paper we go back to the original view that row variables are separate, but we show that they can ultimately be expressed as ordinary type variables.

In outline, this paper shows how a calculus with row variables, $F_{<:}\rho$, can be represented in a simpler calculus without row variables, $F_{<:}$, via a translation. Given independent axiomatizations of $F_{<:}\rho$ and $F_{<:}$ (the former being an extension of the latter) we prove that the translation is well-behaved, in that it preserves typing, subtyping, and equality.

The paper is organized as follows. Sections 2 and 3 recall the definition of $F_{<:}$ and its expressive power (borrowing from [Cardelli Martini Mitchell Scedrov 91]). Section 4 gives the main intuitions of the encoding of extensible records in $F_{<:}$. Section 5 describes $F_{<:}\rho$. Section 6 gives the translation of $F_{<:}\rho$ into $F_{<:}$, and finally section 7 shows that the translation is sound.

Examples of the expressive power of $F_{<:}\rho$ and comparisons with other calculi are delayed until section 5.5. We show there that $F_{<:}\rho$ can express many of the standard benchmark examples that appear in the literature. We encourage readers to examine these examples whenever convenient.

Readers who wish to learn about $F_{<:}\rho$ as a language of records but who are not interested in the translation into $F_{<:}$, may confine themselves to sections 1, 2.0, 2.1, 2.2, 5.0, 5.1, 5.2, 5.4, 5.5, and 8.

2. System $F_{<:}$

In this section we describe the target calculus, $F_{<:}$, for the translation that will follow. $F_{<:}$ can be translated in turn into a trivial extension of F called F_1 [Breazu-Tannen Coquand Gunter Scedrov 89]. However, the known translations from $F_{<:}$ to F_1 do not preserve subtyping in $F_{<:}$ [Martini 90]; this reinforces the point that translating to $F_{<:}$ is more informative than translating directly to F.

$F_{<:}$ is obtained by extending F 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 [Cardelli 88].

2.1 Syntax

The syntax of $F_{<:}$ extends the syntax of F as follows. A new type constant *Top* denotes the supertype of all types. Second-order quantifiers acquire a subtype bound: $\forall(X<:A)A'$ (*bounded quantifiers* [Cardelli Wegner 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*, 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 variable
Top	the supertype of all types
$A \rightarrow B$	function space
$\forall(X<:A)B$	bounded quantification
a,b ::=	Values
x	value variable
top	canonical value of type *Top*
$\lambda(x:A)b$	function
b(a)	application
$\lambda(X<:A)b$	bounded type function
b(A)	type application

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 those values are considered as possessing.

Judgments

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

We use *dom(E)* for the set of variables *defined* by an environment *E*.

As usual, we identify terms up to renaming of bound variables; that is, using $C\{X \leftarrow D\}$ for the substitution of D for X in C :

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

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

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; we now comment on their most interesting aspects.

The subtyping judgment, $E \vdash A<:B$, defines, for any E, a reflexive and transitive relation on types with a *subsumption* property: 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.

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 for example allows a function to take an argument having a subtype of the function's input type.

Most of the equivalence rules, $E \vdash a \leftrightarrow b:A$, are unremarkable. They provide congruence over the syntax, and β and η equivalences. Two rules, however, stand out. The first, (Top collapse), 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; this fact will have many interesting consequences in the sequel. 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 also has many intriguing consequences, but these will not be explored here. They are described in [Cardelli Martini Mitchell Scedrov 91].

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)

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

This calculus was first extracted by Pierre-Louis Curien from the one in [Cardelli Wegner 85] and studied by him and Giorgio Ghelli [Curien Ghelli 91] under the name $F_{\leq:}$. The present $F_{<:}$ is a refinement of $F_{\leq:}$, achieved mostly by extending the *(Eq appl2)* rule. It is studied in [Cardelli Martini Mitchell Scedrov 91].

The following derived rules will be needed later. Their proofs follow from the lemmas listed in section 5.3 for $F_{<:}\rho$. (Those lemmas hold for $F_{<:}$ as well, when restricted to the syntax of $F_{<:}$.)

Lemma (subsumption equivalence)

The subsumption rule extends to the equality judgment:

(Eq subsumption)

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

Lemma (domain restriction)

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

(Eq fun')

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

Lemma (bound restriction)

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

(Eq fun2')

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

3. Basic encodings

Since $F_{<:}$ is an extension of F, it can express all the standard encodings of algebraic data types that are possible in F [Böhm Berarducci 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 encode rich subtyping relations on familiar data structures. In section 4 the encodings become more involved; this increase in complexity then motivates the switch to an independently axiomatized system ($F_{<:}\rho$) in section 5.

3.1 Booleans

In the sequel of section 3 we concentrate on inclusion of structured types, but for this to make sense we need to show that there are some non-trivial inclusions already at the level of basic types. We investigate here the type of booleans, and in the process we illustrate some interesting 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. [Fairbairn 89]):

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

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:

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

We obtain four elements of type *Bool*; in addition to the usual two, $true_{Bool}$ and $false_{Bool}$, the extra $true_{True}$ and $false_{False}$ have type *Bool* by subsumption. However, we can show that $true_{Bool}$ and $true_{True}$ are provably equivalent at type *Bool*, by using the domain restriction lemma (*(Eq fun')*, section 2.2).

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

Similarly, we can show that $E \vdash false_{False} \leftrightarrow false_{Bool} : Bool$. Hence, there really are only two different values in *Bool.*

3.2 Products

The standard encoding for pairs in *F* already exhibits useful subtyping properties:

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

Since both A and B occur in monotonic positions in $A \times B$ (being twice on the left of an arrow), we obtain the expected monotonic inclusion of products as a derived rule:

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

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

$$\begin{aligned}
&pair \ : \ \forall(A)\, \forall(B)\, A \to B \to A \times B \\
&\quad \triangleq \ \lambda(A)\, \lambda(B)\, \lambda(a:A)\, \lambda(b:B)\, \lambda(C)\, \lambda(f:A \to B \to C)\, f(a)(b) \\
&fst \ : \ \forall(A)\, \forall(B)\, A \times B \to A \\
&\quad \triangleq \ \lambda(A)\, \lambda(B)\, \lambda(c:A \times B)\, c(A)(\lambda(x:A)\lambda(y:B)x) \\
&snd \ : \ \forall(A)\, \forall(B)\, A \times B \to B \\
&\quad \triangleq \ \lambda(A)\, \lambda(B)\, \lambda(c:A \times B)\, c(B)(\lambda(x:A)\lambda(y:B)y)
\end{aligned}$$

We often use the following abbreviations, disambiguated by context:

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

3.3 Enumerations

Enumeration types (that is, finite sets) form another collection of base types with interesting inclusion relations. We describe them here because they show an interesting use of the *Top* type, and hint at the encoding of tuples in the next section.

The enumeration of *zero* elements can be defined as:

$$N_0 \quad \triangleq \quad \forall(A)\, Top \to A$$

This type has no closed normal forms, hence no “elements”.

The enumeration of *one* element is defined as:

$$N_1 \triangleq \forall(A)\, A \times Top \rightarrow A$$

This type has just one closed normal form:

$$one_1 : N_1 \triangleq \lambda(A)\ \lambda(x{:}A \times Top)\ fst(x)$$

Moreover, $N_0 <: N_1$ because $A \times Top <: Top$.

The enumeration of *two* elements is defined as:

$$N_2 \triangleq \forall(A)\, A \times A \times Top \rightarrow A$$

This type has the two closed normal forms:

$$one_2 : N_2 \triangleq \lambda(A)\ \lambda(x{:}A \times A \times Top)\ fst(x)$$
$$two_2 : N_2 \triangleq \lambda(A)\ \lambda(x{:}A \times A \times Top)\ fst(snd(x))$$

Moreover, $N_1 <: N_2$, and by subsumption:

$$one_1 : N_2$$

We find that N_2 has three elements. As for booleans, we can prove that two of these are equal in N_2:

$$\vdash one_1 \leftrightarrow one_2 : N_2$$

At this point the pattern of enumeration types should be clear:

$$N_n \triangleq \forall(A)\, \underbrace{A \times .. \times A}_{\text{n times}} \times Top \rightarrow A$$

with $N_n <: N_{n+1}$, where N_n has n distinct elements.

3.4 Tuples

A *tuple type* $Tuple(A_1,...,A_n,C)$ denotes an iterated product type. Its last slot, C, can be filled with any type. When C is a type variable, we have an *extensible tuple type*. When it is Top, we have a *simple tuple type*.

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

Hence we have:

$$Tuple(A_1,...,A_n,Tuple(B_1,...,B_m,C)) \equiv Tuple(A_1,...,A_n,B_1,...,B_m,C)$$

with derived rule:

$$\frac{E \vdash A_1 <: B_1 \ \ldots\ E \vdash A_n <: B_n \quad E \vdash C <: D}{E \vdash Tuple(A_1,...,A_n,C) <: Tuple(B_1,...,B_n,D)}$$

As a special case we obtain the rule for simple tuples:

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

For example:

$Tuple(A, B, Top) <: Tuple(A, Top)$
since $A <: A$, $B \times Top <: Top$, 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:

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

$tuple(a_1,...,a_n,tuple(b_1,...,b_m,c)) \;\equiv\; tuple(a_1,...,a_n,b_1,...,b_m,c)$

with derived rules:

$$\frac{E \vdash a_1 : A_1 \;\; ... \;\; E \vdash a_n : A_n \quad E \vdash a : A}{E \vdash tuple(a_1,...,a_n,a) : Tuple(A_1,...,A_n,A)}$$

$$\frac{E \vdash a_1 \leftrightarrow b_1 : A_1 \;\; ... \;\; E \vdash a_n \leftrightarrow b_n : A_n \quad E \vdash a \leftrightarrow b : A}{E \vdash tuple(a_1,...,a_n,a) \leftrightarrow tuple(b_1,...,b_n,b) : Tuple(A_1,...,A_n,A)}$$

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; we use the abbreviations:

$a\lfloor i \;\equiv\; a\lfloor_{A_i} i \;\equiv\; drop_i(A_i)(a) \;\equiv\; snd_i(a)$
$a.i \;\equiv\; a._{A_i} i \;\equiv\; sel_i(A_i)(a) \;\equiv\; fst(a\lfloor i)$

More precisely:

$drop_0 \;:\; \forall(A_0)\, A_0 \rightarrow A_0$
$\quad\triangleq\; \lambda(A_0)\, \lambda(t{:}A_0)\, t$
$sel_0 \;:\; \forall(A_0)\, A_0 \times Top \rightarrow A_0$
$\quad\triangleq\; \lambda(A_0)\, \lambda(t{:}A_0 \times Top)\, fst_{A_0 \times Top}(drop_0(A_0 \times Top)(t))$
$drop_1 \;:\; \forall(A_1)\, Top \times A_1 \rightarrow A_1$
$\quad\triangleq\; \lambda(A_1)\, \lambda(t{:}Top \times A_1)\, snd_{Top \times A_0}(drop_0(Top \times A_1)(t))$
$sel_1 \;:\; \forall(A_1)\, Top \times A_1 \times Top \rightarrow A_1$
$\quad\triangleq\; \lambda(A_1)\, \lambda(t{:}Top \times A_1 \times Top)\, fst_{A_1 \times Top}(drop_1(A_1 \times Top)(t))$
etc...

We obtain the derived rules:

$$\frac{E \vdash a : \ Tuple(A_0,..,A_{i-1},A)}{E \vdash a\lfloor i \ : A} \qquad \frac{E \vdash a : \ Tuple(A_0,..,A_i,A)}{E \vdash a.i \ : A_i}$$

$$\frac{E \vdash a_0 : A_0 \ \ldots \ E \vdash a_{i-1} : A_{i-1} \quad E \vdash a : A}{E \vdash tuple(a_0,...,a_{i-1},a)\lfloor i \leftrightarrow a : A} \qquad \frac{E \vdash a_0 : A_0 \ \ldots \ E \vdash a_i : A_i \quad E \vdash a : A}{E \vdash tuple(a_0,...,a_i,a).i \leftrightarrow a_i : A_i}$$

Example:

$$let\ f: \forall(X<:Tuple(B,Top))\ Tuple(A,X) \rightarrow Tuple(A,A,X) = \lambda(X<:Tuple(B,Top))\ \lambda(t:Tuple(A,X))\ tuple(t.0,\ t.0,\ t\lfloor 1)$$

$$f(Tuple(B,C,Top))(tuple(a,b,c,top)) \ \leftrightarrow \ tuple(a,a,b,c,top) \ : Tuple(A,A,B,C,Top)$$

We have now developed the necessary techniques for encoding record types; this is the subject of the next section.

4. Records

The general plan, carried out in later sections, is to axiomatize the rules for records independently, and then provide a translation (encoding) into a calculus without records. In this section we are a bit more informal, and we discuss the encoding of record types without first discussing their derived type rules. Some pathologies caused by this approach will disappear later.

4.1 Simple records

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 $Rcd(l_1{:}A_1, .., l_n{:}A_n, C)$, where the final type C will normally be either *Top* or a type variable. Once the enumeration of the set of labels $\mathcal{L}$ 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. That is, the component of label l_i is allocated to the i-th tuple slot; the remaining slots are filled with *Top* "padding". For example:

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

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.

As an artifact of the encoding, a missing record field of label l_i is equivalent to a field $l_i : Top$. However, the type rules for these two situations will differ, and in the former case the extraction of the label l_i will not be allowed.

A record type whose final component is *Top* is called a *simple record*; one whose final component is a type variable, is called an *extensible record*, or simply a *record*. Only these two situations will be allowed by the type rules for records; for example, notice that $Rcd(l^0{:}A, Rcd(l^1{:}B, C))$ is not very meaningful under the translation.

From the encoding, we can derive the familiar rule for simple records [Cardelli 88]:

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

The conclusion holds because any additional field $l_k{:}A_k$ $(n<k\leq m)$ on the left of $<:$ 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:

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

from which we obtain the rules for simple records:

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

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

Record selection is encoded as follows:

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

with the rule:

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

By subsumption, we have the following derived rules:

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

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

$$: Rcd(l_1{:}A_1,..,l_n{:}A_n,Top)$$

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

The second rule above is particularly interesting. It expresses a form of observational equivalence: two records are equivalent at a given type if they coincide with the components that are observable at that type. Ultimately, this is because any two values are equivalent at type *Top*.

An interesting question about simple records remains: what is the equivalent of the $\lfloor$ operator on tuples? To answer this, we must turn to extensible records.

4.2 Extensible records

In the next section we fully axiomatize a system with row variables, $F_{<:}\rho$. To understand that axiomatization better, it may be useful to have an idea of the translation into $F_{<:}$ that will follow. In this section we sketch the main ideas of that translation, but the reader can skip to section 5 at any point.

As we have done with tuples, we would like to place a type variable at the end of a record to capture all the "additional" components.

Tuple(A, B, C, X)	*X* represents all the other tuple components
$Rcd(l^0{:}A, l^2{:}C, X)$	*X* represents all the other record components

When translating these records into tuples, we see that, to achieve the desired effect, the final type variable must split into a set of type variables. (We use the symbol ≈ to mean, informally, "translates to".)

$$Rcd(l^0{:}A, l^2{:}C, X) \quad \approx \quad Tuple(A, X^1, C, X^3)$$

Here *X* cannot be bound to a single (record) type; it must be bound to a *labeled collection of types* that fills the slots X^1 and X^3 exactly. We call these collections *type rows*, and *X* a *row (type) variable*.

Consider, for example:

$$Rcd(l^0{:}A, l^2{:}C, l^4{:}E, X)$$

Here the row variable *X* can be instantiated only to a type row that does not contain components labeled l^0, l^2, or l^4, since these are already accounted for. For example, *X* can be instantiated to the type row $l^1{:}B, l^3{:}D, Top$.

We express this constraint on the instantiations of *X* by saying that *X* must have *kind* "$\uparrow l^0,l^2,l^4$", which reads "... is undefined (exactly) at l^0,l^2,l^4" or "... does not cover (exactly) l^0,l^2,l^4".

A constrained row variable $X\uparrow L$ is hence translated to a sequence of type variables with "gaps" at *L*; for example:

$X\uparrow() \approx X^0$

$X\uparrow l^0 \approx X^1$ $\qquad X\uparrow l^0,l^1 \approx X^2$

$X\uparrow l^1 \approx X^0, X^2$ $\qquad X\uparrow l^0,l^2 \approx X^1, X^3$

$X\uparrow l^2 \approx X^0, X^1, X^3$ $\qquad X\uparrow l^1,l^2 \approx X^0, X^3$

Therefore, the first step in extending $F_{<:}$ with row types is to allow constrained row variables in environments:

$$E', X\uparrow L, E'' \vdash ...$$

Then, if $X\uparrow L \approx X_1, ..., X_n$ we translate:

$$E',X\uparrow L \vdash Rcd(l_1{:}A_1, ..., l_m{:}A_m, X)\ type$$
$$\approx\ E',X_1, ..., X_n \vdash Tuple(B_1, ..., B_{n+m-1}, X_n)\ type$$

where the B_i are the $X_1...X_{n-1}$ and the $A_1...A_m$, in the proper order.

To manipulate type rows and row variables we introduce a new judgment form (described in detail in the next section):

$$E \vdash R\ \uparrow L$$

where R is a type row (including a row variable), and L is the set of labels that are not covered by the row R. In general we need to translate not just records, but rows, which may have missing components:

$(l^0{:}A, l^2{:}C, X)\uparrow l^1,l^4 \approx A, -, C, X^3, -, X^5$ (a row missing 1^{st} and 4^{th}).

Once row variables are allowed in environments, they give rise naturally to quantifiers $\forall(X\uparrow L)$, and binders $\lambda(X\uparrow L)$. These row quantifiers and row binders must decompose under translation into sequences of type quantifiers and type binders. For example, we have:

$$\forall(X\uparrow l^1)\ Rcd(l^1{:}A; X) \rightarrow B$$
$$\approx\ \forall(X^0)\ \forall(X^2)\ Tuple(X^0,A,X^2) \rightarrow B$$

We now come to the most important issue of the translation: matching the number of arguments of a row type function $\lambda(X\uparrow L)a$ to the number of parameters in a row type application $(\lambda(X\uparrow L)a)(R\uparrow L)$. The application form for a function b: $\forall(X\uparrow L)B$ will have the shape:

$$b(R\uparrow L) : B\{X\leftarrow R\} \qquad \text{for}\quad R\uparrow L$$

where $B\{X\leftarrow R\}$ is a *row substitution* such that (for ξ a row variable or *Top*):

$$Rcd(l_1{:}A_1,...,l_n{:}A_n, X)\ \{X\leftarrow(l'_1{:}B_1,...,l'_m{:}B_m, \xi)\} =$$
$$Rcd(l_1{:}A_1,...,l_n{:}A_n,l'_1{:}B_1,...,l'_m{:}B_m, \xi)$$

We have seen that the translations of $\forall(X\uparrow L)B$ and $\lambda(X\uparrow L)b$ convert the single parameter $X\uparrow L$ into a sequence of parameters whose length can depend only on L. We

call this length ∂L: the *dimension* of L. When translating an application $b(R \uparrow L)$ we must then produce a sequence of applications of size ∂L, irrespectively of the actual parameter R. This may require some regrouping of the components of an argument row R. For example:

$b(l^2{:}A^2, Y \uparrow l^1 l^3)$ (where $b{:}\ \forall(X \uparrow l^1 l^3)B$ and $Y \uparrow l^1 l^2 l^3 \approx Y^0, Y^4$
$\quad \approx \quad b(Y^0)(A^2)(Y^4)$ and $l^2{:}A^2, Y \uparrow l^1 l^3 \approx Y^0, A^2, Y^4$)

$b(l^3{:}A^3, Y \uparrow l^1 l^2)$ (where $b{:}\ \forall(X \uparrow l^1 l^2)B$ and $Y \uparrow l^1 l^2 l^3 \approx Y^0, Y^4$
$\quad \approx \quad b(Y^0)(Tuple(A^3, Y^4))$ and $l^3{:}A^3, Y \uparrow l^1 l^2 \approx Y^0, A^3, Y^4$)

In the second case, $b(Y^0)(A^3)(Y^4)$ would be wrong; we must group A^3 and Y^4 into $Tuple(A^3, Y^4)$, to match the two parameters (X^0 and X^3) expected by b. For uniformity in the translation, we always take the last parameter to be a tuple (since $Tuple(A) \equiv A$), so the first case above becomes:

$b(l^2{:}A^2, Y \uparrow l^1 l^3)$
$\quad \approx \quad b(Y^0)(A^2)(Tuple(Y^4))$

In conclusion, we can say informally that row variables translate to rows of variables, row types to rows of types, row quantifiers to rows of quantifiers, row applications to rows of applications, etc. The main difficulty in the translation is to ensure that all these rows match properly. For this, the precise relation between a row $R \uparrow L$ and its dimension ∂L, will be discussed in section 6.

We now turn to a formal system based on the intuitions about the translation of records into tuples developed in this section.

5. System $F_{<:}\rho$

We now extend $F_{<:}$ with records and row variables, as discussed in section 4; the resulting system is called $F_{<:}\rho$.

5.1 Syntax

Types in $F_{<:}$ are augmented by the following: record types $Rcd(R)$, where R is a *row type* that must be defined at all labels; row function types $R \uparrow L \rightarrow B$ from an input of row type $R \uparrow L$ to an output of type B; and row variable quantifications $\forall(X \uparrow L)B$, where L is a set of labels at which X is undefined.

A row type is either the constant Etc, standing for an "empty row" (more precisely, an unnamed extension of the current row type); a type variable X, standing for an extension of the current row type; or $l{:}A, R$, extending the row type R by a field of type A and label l.

Values are augmented by the following: records $rcd(r)$, where r is a *row value* defined at all labels; row functions $\lambda(x{:}R \uparrow L)b$ accepting a row value for x of row type $R \uparrow L$; and row type functions $\lambda(X \uparrow L)b$ accepting a row type for X that is undefined at L. Record

selection $a.l$ can be used on a record a that is defined at l. A row function b can be applied via $b(r \uparrow L)$ to a row value r undefined at L. A row type function b can be instantiated via $b(R \uparrow L)$ to a row type R undefined at L.

Finally, a row value is either the constant *etc*, standing for an "empty row" (or, an unnamed extension of the current row value); a row variable x; an extension $l=a,r$, extending row value r by a field of value a and label l; or a restriction $a\backslash L$, producing a row value undefined at L from a record a.

Syntax

$L ::= \quad l_1, .., l_n$	Label set
$A,B ::= ...$	Types as in $F_{<:}$, plus:
$Rcd(R)$	record type
$R \uparrow L \rightarrow B$	row function space
$\forall(X \uparrow L)B$	row quantification
$R,S ::=$	Row types
X	row type variable
Etc	empty row type
$l:A,R$	row type R plus field A labeled l
$a,b ::= ...$	Values as in $F_{<:}$, plus:
$rcd(r)$	record value
$a.l$	record selection
$\lambda(x{:}R \uparrow L)b$	row value function
$b(r \uparrow L)$	row value application
$\lambda(X \uparrow L)b$	row type function
$b(R \uparrow L)$	row type application
$r,s ::=$	Row values
x	row value variable
etc	empty row value
$l=a,r$	row value r plus field a labeled l
$a\backslash L$	row value of record a without fields in L

As discussed in section 2, we identify terms up to renaming of bound variables:

$$\forall(X \uparrow L)B \quad \equiv \quad \forall(Y \uparrow L)B\{X \leftarrow Y\}$$
$$\lambda(X \uparrow L)b \quad \equiv \quad \lambda(X \uparrow L)b\{X \leftarrow Y\}$$
$$\lambda(x{:}R \uparrow L)b \quad \equiv \quad \lambda(y{:}R \uparrow L)b\{x \leftarrow y\}$$

Moreover, we identify rows up to reordering of labeled components:

$$l:A,l':A',R \quad \equiv \quad l':A',l:A,R$$
$$l=a,l'=a',r \quad \equiv \quad l'=a',l=a,r$$

and we identify terms up to any permutation L' of a label set L:

$$\begin{array}{lll@{\qquad}lll} \forall(X\uparrow L)B & \equiv & \forall(X\uparrow L')B & R\uparrow L\to B & \equiv & R\uparrow L'\to B \\ \lambda(X\uparrow L)b & \equiv & \lambda(X\uparrow L')b & b(R\uparrow L) & \equiv & b(R\uparrow L') \\ \lambda(x{:}R\uparrow L)b & \equiv & \lambda(x{:}R\uparrow L')b & b(r\uparrow L) & \equiv & b(r\uparrow L') \\ a\backslash L & \equiv & a\backslash L' & & & \end{array}$$

Again, these identifications are legitimate because they depend only on the syntax of terms, and not on their derivations.

Given the identification of label sets above, we adopt the following notational convention used in the inference rules:

$$l.L \triangleq \{l\}\cup L \text{ where } l\notin L$$

We now add to $F_{<:}$ four judgments about rows, which all involve a set L at which the rows are undefined.

Judgments

...	Judgments as in $F_{<:}$, plus:
$E\vdash_\rho R\uparrow L$	R is a row type not covering L
$E\vdash_\rho r{:}R\uparrow L$	r has row type $R\uparrow L$
$E\vdash_\rho R<:S\uparrow L$	R is a subrow of S, both not covering L
$E\vdash_\rho r\leftrightarrow r'{:}R\uparrow L$	r is equal to r' at row type $R\uparrow L$

It is important to notice that the L information is preserved exactly in $F_{<:}\rho$ derivations, in the sense that $E\vdash_\rho \vartheta\uparrow L \Rightarrow E\vdash_\rho \vartheta\uparrow L'$ is never derivable for $L\neq L'$ for any of the four judgments. Hence, when we say that a row is undefined at L, we always mean undefined *exactly* at L.

5.2 Rules

We indicate by $\vdash_\rho$ the judgments in $F_{<:}\rho$, to distinguish them from the judgments $\vdash$ in $F_{<:}$. The rules of $F_{<:}\rho$ consist of a copy of the rules of $F_{<:}$ (with $\vdash$ replaced by $\vdash_\rho$) plus the ones listed below. We now briefly comment on the $F_{<:}\rho$ rules.

A row type is formed by starting with a row variable $X\uparrow L$, or with a row $Etc\uparrow L$, and then prefixing fields $l{:}A$ with $l\in L$, at each step discarding l from L. Note that Etc can be assumed to lack any set of labels to start with. Informally, we can imagine either that an element of $Etc\uparrow L$ is a collection of $n=\#L$ empty slots that are later "filled in", or that an element of $Etc\uparrow L$ is an infinite row with "gaps" corresponding to L, and with all the other components filled with an error value.

A record type can be formed only from a *complete* row $R\uparrow()$, one lacking no labels. (We call $R\uparrow()$ complete even though we have only finite information about the labels of R; for example, $Etc\uparrow()$ is complete but entirely unknown.) This completeness requirement is probably not essential, but gives us a simpler calculus where record types

carry only positive information, while row variables carry only negative information [Harper Pierce 90].

The subrow judgment, $E \vdash_\rho R <:\!\cdot S \uparrow L$, is mainly an auxiliary one used to define subtyping on records. According to this judgment, every row is a subrow of *Etc*; then we have componentwise subtyping on fields having the same label. Hence, a longer row ending in *Etc* is a subrow of a shorter row ending in *Etc* if their corresponding components are in subtype relation. Rows ending with the same type variables must have the same length (otherwise, assuming $X \uparrow l$, what could L be in $E \vdash_\rho l{:}A, X <:\!\cdot X \uparrow L$?). Rows ending in distinct type variables are unrelated, since we have no information about the labeled types that may be substituted for the variables.

Record values can be created only from complete rows, as discussed above. Given a record $a : Rcd(l{:}A,R)$ we can select its l component by $a.l : A$. Moreover, given a record $a : Rcd(l_1{:}A_1..l_n{:}A_n,R)$ we can extract a row $a\backslash L :\!\cdot R \uparrow L$ from it by removing all the components with labels in L.

In $F_{<:}$ any two values are equivalent in *Top*. Similarly, in $F_{<:}\rho$ any two row values are equivalent in *Etc*.

Environments

$$\frac{E \vdash_\rho R \uparrow L \quad x \notin dom(E)}{\vdash_\rho E, x :\!\cdot R \uparrow L \; env} \;(Env\ x \uparrow L) \qquad \frac{\vdash_\rho E \; env \quad X \notin dom(E)}{\vdash_\rho E, X \uparrow L \; env} \;(Env\ X \uparrow L)$$

Types

$$\frac{E \vdash_\rho R \uparrow ()}{E \vdash_\rho Rcd(R)\ type} \;(Type\ Rcd) \qquad \frac{E \vdash_\rho R \uparrow L \quad E \vdash_\rho B\ type}{E \vdash_\rho R \uparrow L \to B\ type} \;(Type \to \uparrow L) \qquad \frac{E, X \uparrow L \vdash_\rho B\ type}{E \vdash_\rho \forall (X \uparrow L) B\ type} \;(Type \forall \uparrow L)$$

Row types

$$\frac{\vdash_\rho E', X \uparrow L, E''\ env}{E', X \uparrow L, E'' \vdash_\rho X \uparrow L} \;(Type\ X) \qquad \frac{\vdash_\rho E\ env}{E \vdash_\rho Etc \uparrow L} \;(Type\ Etc) \qquad \frac{E \vdash_\rho R \uparrow l.L \quad E \vdash_\rho A\ type}{E \vdash_\rho l{:}A, R \uparrow L} \;(Type\ cons)$$

Subtypes

$$\frac{E \vdash_\rho R <:\!\cdot R' \uparrow ()}{E \vdash_\rho Rcd(R) <: Rcd(R')} \;(Sub\ Rcd) \qquad \frac{E \vdash_\rho R' <:\!\cdot R \uparrow L \quad E \vdash_\rho B <: B'}{E \vdash_\rho R \uparrow L \to B <: R' \uparrow L \to B'} \;(Sub \to \uparrow L) \qquad \frac{E, X \uparrow L \vdash_\rho B <: B'\ type}{E \vdash_\rho \forall (X \uparrow L) B <: \forall (X \uparrow L) B'\ type} \;(Sub \forall \uparrow L)$$

Subrows

(Sub Row refl)
$$\frac{E \vdash_\rho R \uparrow L}{E \vdash_\rho R <:\!\cdot R \uparrow L}$$

(Sub Row trans)
$$\frac{E \vdash_\rho R <:\!\cdot S \uparrow L \quad E \vdash_\rho S <:\!\cdot T \uparrow L}{E \vdash_\rho R <:\!\cdot T \uparrow L}$$

(Sub Etc)
$$\frac{E \vdash_\rho R \uparrow L}{E \vdash_\rho R <:\!\cdot Etc \uparrow L}$$

(Sub cons)
$$\frac{E \vdash_\rho A <: B \quad E \vdash_\rho R <:\!\cdot S \uparrow l.L}{E \vdash_\rho l{:}A,R <:\!\cdot l{:}B,S \uparrow L}$$

Values

(Val rcd)
$$\frac{E \vdash_\rho r :\!\cdot R \uparrow ()}{E \vdash_\rho rcd(r) : Rcd(R)}$$

(Val sel)
$$\frac{E \vdash_\rho a : Rcd(l{:}A,R)}{E \vdash_\rho a.l : A}$$

(Val fun $\uparrow L$)
$$\frac{E,x :\!\cdot R \uparrow L \vdash_\rho b : B}{E \vdash_\rho \lambda(x :\!\cdot R \uparrow L)\, b : R \uparrow L \rightarrow B}$$

(Val appl $\uparrow L$)
$$\frac{E \vdash_\rho b : R \uparrow L \rightarrow B \quad E \vdash_\rho r :\!\cdot R \uparrow L}{E \vdash_\rho b(r \uparrow L) : B}$$

(Val fun2 $\uparrow L$)
$$\frac{E,X \uparrow L \vdash_\rho b : B}{E \vdash_\rho \lambda(X \uparrow L)b : \forall(X \uparrow L)B}$$

(Val appl2 $\uparrow L$)
$$\frac{E \vdash_\rho b : \forall(X \uparrow L)B \quad E \vdash_\rho R \uparrow L}{E \vdash_\rho b(R \uparrow L) : B\{X \leftarrow R\}}$$

Row values

(Row Subsumption)
$$\frac{E \vdash_\rho r :\!\cdot R \uparrow L \quad E \vdash_\rho R <:\!\cdot S \uparrow L}{E \vdash_\rho r :\!\cdot S \uparrow L}$$

(Val x $\uparrow L$)
$$\frac{\vdash_\rho E', x :\!\cdot R \uparrow L, E'' \; env}{E', x :\!\cdot R \uparrow L, E'' \vdash_\rho x :\!\cdot R \uparrow L}$$

(Val etc)
$$\frac{\vdash_\rho E \; env}{E \vdash_\rho etc :\!\cdot Etc \uparrow L}$$

(Val cons)
$$\frac{E \vdash_\rho r :\!\cdot R \uparrow l.L \quad E \vdash_\rho a{:}A}{E \vdash_\rho l{=}a,r :\!\cdot l{:}A,R \uparrow L}$$

(Val restr)
$$\frac{E \vdash_\rho a : Rcd(l_1{:}A_1..l_n{:}A_n,R)}{E \vdash_\rho a \backslash l_1..l_n :\!\cdot R \uparrow l_1..l_n}$$

Value equivalence

(Eq rcd)
$$\frac{E \vdash_\rho r \leftrightarrow r' :\!\cdot R \uparrow ()}{E \vdash_\rho rcd(r) \leftrightarrow rcd(r') : Rcd(R)}$$

(Eq sel)
$$\frac{E \vdash_\rho a \leftrightarrow a' : Rcd(l{:}A,R)}{E \vdash_\rho a.l \leftrightarrow a'.l : A}$$

(Eq Eval sel)
$$\frac{E \vdash_\rho r :\!\cdot R \uparrow l \quad E \vdash_\rho a \leftrightarrow a' : A}{E \vdash_\rho rcd(l{=}a,r).l \leftrightarrow a' : A}$$

(Eq fun $\uparrow L$)
$$\frac{E,x :\!\cdot R \uparrow L \vdash_\rho b \leftrightarrow b' : B}{E \vdash_\rho \lambda(x :\!\cdot R \uparrow L)b \leftrightarrow \lambda(x :\!\cdot R \uparrow L)b' : R \uparrow L \rightarrow B}$$

(Eq appl $\uparrow L$)
$$\frac{E \vdash_\rho b \leftrightarrow b' : R \uparrow L \rightarrow B \quad E \vdash_\rho r \leftrightarrow r' :\!\cdot R \uparrow L}{E \vdash_\rho b(r \uparrow L) \leftrightarrow b'(r' \uparrow L) : B}$$

(Eq fun2 ↑L)

$$\frac{E,X\uparrow L \vdash_\rho b \leftrightarrow b' : B}{E \vdash_\rho \lambda(X\uparrow L)b \leftrightarrow \lambda(X\uparrow L)b' : \forall(X\uparrow L)B}$$

(Eq appl2 ↑L)

$$\frac{E \vdash_\rho b \leftrightarrow b' : \forall(X\uparrow L)B \quad E \vdash_\rho R \uparrow L}{E \vdash_\rho b(R\uparrow L) \leftrightarrow b'(R\uparrow L) : B\{X\leftarrow R\}}$$

(Eq Beta ↑L)

$$\frac{E,x{:}R\uparrow L \vdash_\rho b \leftrightarrow b' : B \quad E \vdash_\rho r \leftrightarrow r' :: R \uparrow L}{E \vdash_\rho (\lambda(x{:}R\uparrow L)b)(r) \leftrightarrow b'\{x\leftarrow r'\} : B}$$

(Eq Eta ↑L)

$$\frac{E \vdash_\rho b \leftrightarrow b' : R\uparrow L\rightarrow B \quad y\notin dom(E)}{E \vdash_\rho \lambda(y{:}R\uparrow L)b(y\uparrow L) \leftrightarrow b' : R\uparrow L\rightarrow B}$$

(Eq Beta2 ↑L)

$$\frac{E,X\uparrow L \vdash_\rho b \leftrightarrow b' : B \quad E \vdash_\rho R \uparrow L}{E \vdash_\rho (\lambda(X\uparrow L)b)(R\uparrow L) \leftrightarrow b'\{X\leftarrow R\} : B\{X\leftarrow R\}}$$

(Eq Eta2 ↑L)

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

Row value equivalence

(Eq Row symm)

$$\frac{E \vdash_\rho r \leftrightarrow s :: R \uparrow L}{E \vdash_\rho s \leftrightarrow r :: R \uparrow L}$$

(Eq Row trans)

$$\frac{E \vdash_\rho r \leftrightarrow s :: R \uparrow L \quad E \vdash_\rho s \leftrightarrow t :: R \uparrow L}{E \vdash_\rho r \leftrightarrow t :: R \uparrow L}$$

(Eq Row Subsumption)

$$\frac{E \vdash_\rho r \leftrightarrow r' :: R \uparrow L \quad E \vdash_\rho R <: S \uparrow L}{E \vdash_\rho r \leftrightarrow r' :: S \uparrow L}$$

(Eq Row collapse)

$$\frac{E \vdash_\rho r :: Etc \uparrow L \quad E \vdash_\rho s :: Etc \uparrow L}{E \vdash_\rho r \leftrightarrow s :: Etc \uparrow L}$$

(Eq x ↑L)

$$\frac{E \vdash_\rho x :: R \uparrow L}{E \vdash_\rho x \leftrightarrow x :: R \uparrow L}$$

(Eq etc)

$$\frac{\vdash_\rho E\ env}{E \vdash_\rho etc \leftrightarrow etc :: Etc \uparrow L}$$

(Eq cons)

$$\frac{E \vdash_\rho r \leftrightarrow r' :: R \uparrow l.L \quad E \vdash_\rho a \leftrightarrow a' : A}{E \vdash_\rho l{=}a,r \leftrightarrow l{=}a',r' :: l{:}A,R \uparrow L}$$

(Eq restr)

$$\frac{E \vdash_\rho a \leftrightarrow a' : Rcd(l_1{:}A_1..l_n{:}A_n,R)}{E \vdash_\rho a\backslash l_1..l_n \leftrightarrow a\backslash l_1..l_n :: R \uparrow l_1..l_n}$$

(Eq Eval restr)

$$\frac{E \vdash_\rho r \leftrightarrow r' :: R \uparrow l_1..l_n \quad E \vdash_\rho a_1 : A_1 \ldots E \vdash_\rho a_n : A_n}{E \vdash_\rho rcd(l_1{=}a_1..l_n{=}a_n,r)\backslash l_1..l_n \leftrightarrow r' :: R \uparrow l_1..l_n}$$

Example derivations

$$\frac{\dfrac{\dfrac{\dfrac{\vdash_\rho E\ env}{E \vdash_\rho Etc \uparrow l^3,l^5 \quad E \vdash_\rho A\ type}}{E \vdash_\rho l^3{:}A,Etc \uparrow l^5 \quad E \vdash_\rho B\ type}}{E \vdash_\rho l^5{:}B,l^3{:}A,Etc \uparrow ()}}{E \vdash_\rho Rcd(l^5{:}B,l^3{:}A,Etc)\ type}$$

$$\frac{\dfrac{\dfrac{\dfrac{\vdash_\rho E,X\uparrow l^3,l^5\ env}{E,X\uparrow l^3,l^5 \vdash_\rho X \uparrow l^3,l^5 \quad E \vdash_\rho A\ type}}{E,X\uparrow l^3,l^5 \vdash_\rho l^3{:}A,X \uparrow l^5 \quad E \vdash_\rho B\ type}}{E,X\uparrow l^3,l^5 \vdash_\rho l^5{:}B,l^3{:}A,X \uparrow ()}}{E,X\uparrow l^3,l^5 \vdash_\rho Rcd(l^5{:}B,l^3{:}A,X)\ type}$$

5.3 Properties

We now state some basic lemmas about the properties of $F_{<:\rho}$ derivations (and, implicitly, of $F_{<:}$ derivations). Unless otherwise noted, these are all proven by induction on the derivations; the proofs are long, but straightforward if done in the order indicated.

Notation

Let ϑ be any of

C *type*, $S \uparrow M$, $C <: C'$, $S <:\cdot S' \uparrow M$, $c:C$, $s \cdot S \uparrow M$, $c \leftrightarrow c':C$, $s \leftrightarrow s' \cdot S \uparrow M$

Lemma (renaming)

Let $\langle \xi, \xi', \beta, \beta' \rangle$ stand for either $\langle X, Y, X <: D, Y <: D \rangle$, $\langle X, Y, X \uparrow M, Y \uparrow M \rangle$, $\langle x, y, x:D, y:D \rangle$, or $\langle x, y, x \cdot T \uparrow M, y \cdot T \uparrow M \rangle$.

Assume $\xi' \notin dom(E, \beta, E')$.

$\vdash_\rho E, \beta, E'$ *env* $\Rightarrow \vdash_\rho E, \beta', E'\{\xi \leftarrow \xi'\}$ *env*

$E, \beta, E' \vdash_\rho \vartheta \Rightarrow E, \beta', E'\{\xi \leftarrow \xi'\} \vdash_\rho \vartheta\{\xi \leftarrow \xi'\}$

Lemma (implied judgments 1)

(ϑ/env) $\vdash_\rho E, F$ *env* $\Rightarrow \vdash_\rho E$ *env*

$E, F \vdash_\rho \vartheta \Rightarrow \vdash_\rho E$ *env*

(env/type) $\vdash_\rho E, X <: D, E'$ *env* $\Rightarrow E \vdash_\rho D$ *type*

$\vdash_\rho E, x:D, E'$ *env* $\Rightarrow E \vdash_\rho D$ *type*

(env/rowtype) $\vdash_\rho E, x \cdot R \uparrow L, E'$ *env* $\Rightarrow E \vdash_\rho R \uparrow L$

Lemma (bound change)

$\vdash_\rho E, X <: D', E'$ *env*, $E \vdash_\rho D$ *type* $\Rightarrow \vdash_\rho E, X <: D, E'$ *env*

$E, X <: D', E' \vdash_\rho C$ *type*, $E \vdash_\rho D$ *type* $\Rightarrow E, X <: D, E' \vdash_\rho C$ *type*

$E, X <: D', E' \vdash_\rho S \uparrow M$, $E \vdash_\rho D$ *type* $\Rightarrow E, X <: D, E' \vdash_\rho S \uparrow M$

Lemma (weakening)

Let β stand for either $X \uparrow L$, $X <: D$, $x:D$, or $x \cdot T \uparrow L$.

Assume $\vdash_\rho E, \beta$ *env*, and $X, x \notin dom(E')$; then

$\vdash_\rho E, E'$ *env* $\Rightarrow \vdash_\rho E, \beta, E'$ *env*

$E, E' \vdash_\rho \vartheta \Rightarrow E, \beta, E' \vdash_\rho \vartheta$

Assume $\vdash_\rho E, F$ *env* and $dom(F) \cap dom(E') = \emptyset$; then

$\vdash_\rho E, E'$ *env* $\Rightarrow \vdash_\rho E, F, E'$ *env*

$E, E' \vdash_\rho \vartheta \Rightarrow E, F, E' \vdash_\rho \vartheta$

Lemma (implied judgments 2)

(sub/type) $E \vdash_\rho C <: C' \Rightarrow E \vdash_\rho C$ *type*, $E \vdash_\rho C'$ *type*

(subrow/typerow) $E \vdash_\rho S <:\cdot S' \uparrow M \Rightarrow E \vdash_\rho S \uparrow M$, $E \vdash_\rho S' \uparrow M$

Lemma (bound weakening)

Let $\langle \beta, \beta' \rangle$ stand for either

$\langle X <: D, X <: D' \rangle$, $\langle x:D, x:D' \rangle$, or $\langle x \cdot R \uparrow L, x \cdot R' \uparrow L \rangle$.

Assume $E \vdash_\rho D' <: D$ and $E \vdash_\rho R' <:\cdot R \uparrow L$.

$\vdash_\rho E, \beta, E'$ *env* $\Rightarrow \vdash_\rho E, \beta', E'$ *env*

$E, \beta, E' \vdash_\rho \vartheta \Rightarrow E, \beta', E' \vdash_\rho \vartheta$

Lemma (type substitution)

Assume $E \vdash_\rho D' <: D$; then

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

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

Assume $E \vdash_\rho S \uparrow M$; then

$\vdash_\rho E, X \uparrow M, E'\ env \ \Rightarrow\ \vdash_\rho E, E'\{X \leftarrow S\}\ env$

$E, X \uparrow M, E' \vdash_\rho \vartheta \ \Rightarrow\ E, E'\{X \leftarrow S\} \vdash_\rho \vartheta\{X \leftarrow S\}$

Lemma (value substitution)

Assume $E \vdash_\rho d : D$; then

$\vdash_\rho E, x : D, E'\ env \ \Rightarrow\ \vdash_\rho E, E'\ env$

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

Assume $E \vdash_\rho t \therefore T \uparrow N$; then

$\vdash_\rho E, x \therefore T \uparrow N, E'\ env \ \Rightarrow\ \vdash_\rho E, E'\ env$

$E, x \therefore T \uparrow N, E' \vdash_\rho \vartheta \ \Rightarrow\ E, E' \vdash_\rho \vartheta\{x \leftarrow t\}$

Lemma (value strengthening)

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

$\vdash_\rho E, x : D, E'\ env \ \Rightarrow\ \vdash_\rho E, E'\ env$

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

Assume $x \notin FV(\vartheta)$; then, for $\vartheta \neq r \leftrightarrow r' \therefore R \uparrow L$

$\vdash_\rho E, x \therefore T \uparrow N, E'\ env \ \Rightarrow\ \vdash_\rho E, E'\ env$

$E, x \therefore T \uparrow N, E' \vdash_\rho \vartheta \ \Rightarrow\ E, E' \vdash_\rho \vartheta$

Lemma (implied judgments 3)

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

(rowval/rowtype) $E \vdash_\rho s \therefore S \uparrow M \ \Rightarrow\ E \vdash_\rho S \uparrow M,$

(eq/val) $E \vdash_\rho c \leftrightarrow c' : C \ \Rightarrow\ E \vdash_\rho c : C, \quad E \vdash_\rho c' : C,$

(roweq/rowval) $E \vdash_\rho s \leftrightarrow s' \therefore S \uparrow M \ \Rightarrow\ E \vdash_\rho s \therefore S \uparrow M, \quad E \vdash_\rho s' \therefore S \uparrow M,$

Lemma (subsumption equivalence)

$E \vdash_\rho c \leftrightarrow c' : C, \quad E \vdash_\rho C <: D \quad \Rightarrow \quad E \vdash_\rho c \leftrightarrow c' : D$

Proof By subsumption and beta; see [Cardelli Martini Mitchell Scedrov 91] □

Lemma (implied judgments 4)

(val/eq) $E \vdash_\rho c : C \quad \Rightarrow \quad E \vdash_\rho c \leftrightarrow c : C$

(rowval/roweq) $E \vdash_\rho s \therefore S \uparrow M \quad \Rightarrow \quad E \vdash_\rho s \leftrightarrow s \therefore S \uparrow M$

Lemma (exchange)

Let β stand for either $X <: D$, $Y \uparrow M$, $x : D$, or $x \therefore T \uparrow M$.

Let β' stand for either $X' <: D'$, $Y' \uparrow M'$, $x' : D'$, or $x' \therefore T' \uparrow M'$.

Assume $\vdash_\rho E, \beta'\ env$.

$\vdash_\rho E, \beta, \beta', E'\ env \ \Rightarrow\ \vdash_\rho E, \beta', \beta, E'\ env$

$E, \beta, \beta', E' \vdash_\rho \vartheta \ \Rightarrow\ E, \beta', \beta, E' \vdash_\rho \vartheta$

We can now show that an *observational equivalence* rule for records is derivable. This rule asserts that two record values are equal at a given type if all the equally-labeled fields that can be observed at that type are equal.

Proposition (observational equivalence for records)

$E \vdash_\rho a_1 \leftrightarrow b_1 : A_1 \wedge \ldots \wedge E \vdash_\rho a_n \leftrightarrow b_n : A_n \wedge E \vdash_\rho r :: R \uparrow l_1..l_n \wedge E \vdash_\rho s :: S \uparrow l_1..l_n$
$\Rightarrow E \vdash_\rho rcd(l_1=a_1,..,l_n=a_n,r) \leftrightarrow rcd(l_1=b_1,..,l_n=b_n,s) : Rcd(l_1:A_1,..,l_n:A_n,Etc)$

Proof

Let $L \equiv l_1..l_n$.

$E \vdash_\rho r :: R \uparrow L \Rightarrow \vdash_\rho E\ env$ *(implied judgment)*
$E \vdash_\rho r :: R \uparrow L \Rightarrow E \vdash_\rho R \uparrow L$ *(implied judgment)*

$E \vdash_\rho R \uparrow L \Rightarrow E \vdash_\rho R <:: Etc \uparrow L$ *(Sub Etc)*
$E \vdash_\rho r :: R \uparrow L \wedge E \vdash_\rho R <:: Etc \uparrow L \Rightarrow E \vdash_\rho r :: Etc \uparrow L$ *(subsumption equiv.)*

$\vdash_\rho E\ env \Rightarrow E \vdash_\rho etc \leftrightarrow etc :: Etc \uparrow L$ *(Eq etc)*
$E \vdash_\rho r :: Etc \uparrow L \wedge E \vdash_\rho etc :: Etc \uparrow L \Rightarrow E \vdash_\rho r \leftrightarrow etc :: Etc \uparrow L$ *(Eq Row collapse)*
$E \vdash_\rho s :: S \uparrow L \Rightarrow E \vdash_\rho etc \leftrightarrow s :: Etc \uparrow L$ *(similarly)*
$E \vdash_\rho r \leftrightarrow etc :: Etc \uparrow L \wedge E \vdash_\rho etc \leftrightarrow s :: Etc \uparrow L \Rightarrow E \vdash_\rho r \leftrightarrow s :: Etc \uparrow L$ *(Eq trans)*

$E \vdash_\rho r :: R \uparrow l_1..l_n \wedge E \vdash_\rho s :: S \uparrow l_1..l_n \Rightarrow E \vdash_\rho r \leftrightarrow s :: Etc \uparrow l_1..l_n$ *(above)*

$E \vdash_\rho a_1 \leftrightarrow b_1 : A_1 \wedge \ldots \wedge E \vdash_\rho a_n \leftrightarrow b_n : A_n \wedge E \vdash_\rho r \leftrightarrow s :: Etc \uparrow l_1..l_n$
$\Rightarrow E \vdash_\rho l_1=a_1,..,l_n=a_n,r \leftrightarrow l_1=b_1,..,l_n=b_n,s :: l_1:A_1,..,l_n:A_n,Etc \uparrow ()$ *(Eq Row cons)*
$\Rightarrow E \vdash_\rho rcd(l_1=a_1,..,l_n=a_n,r) \leftrightarrow rcd(l_1=b_1,..,l_n=b_n,s) : Rcd(l_1:A_1,..,l_n:A_n,Etc)$ *(Eq rcd)* □

5.4 Some useful extensions

In preparation for examples in the next section, we discuss some useful extensions of our system: recursive types, label-set variables, and definitions. These extensions are not treated in the formal part of the paper.

5.4.1 Recursive types

In order to introduce recursive types, we need to add type equivalence judgments to the system along with rules (omitted here) for making type equivalence into a congruence over the syntax:

$E \vdash_\rho A \leftrightarrow B\ type$	A and B are equivalent types
$E \vdash_\rho R \leftrightarrow S \uparrow L$	R and S are equivalent row types

A recursive type is, syntactically, a term $\mu(X)A$ where A is *contractive* in X (written $A \succ X$). This means that $A \neq X$, and if $A = \mu(Y)B$ then $B \succ X$. We immediately identify recursive types up to renaming of bound variables:

$\mu(X)A \equiv \mu(Y)A\{X \leftarrow Y\}$

Then, the rules for recursive types [Amadio Cardelli 91] are:

$$\frac{E,X<:Top \vdash_\rho A\ type \quad A>X}{E \vdash_\rho \mu(X)A\ type} \qquad \frac{E \vdash_\rho \mu(X)A\ type}{E \vdash_\rho \mu(X)A \leftrightarrow A\{X \leftarrow \mu(X)A\}\ type}\ (unfold)$$

$$\frac{E,X<:Top \vdash_\rho A \leftrightarrow B\ type \quad A>X \quad B>X}{E \vdash_\rho \mu(X)A \leftrightarrow \mu(X)B\ type}$$

$$\frac{E \vdash_\rho A \leftrightarrow C\{X \leftarrow A\}\ type \quad E \vdash_\rho B \leftrightarrow C\{X \leftarrow B\}\ type \quad C>X}{E \vdash_\rho A \leftrightarrow B\ type}\ (contract)$$

$$\frac{E \vdash_\rho \mu(X)A\ type \quad E \vdash_\rho \mu(Y)B\ type \quad E,Y<:Top,X<:Y \vdash_\rho A<:B}{E \vdash_\rho \mu(X)A <: \mu(Y)B}$$

A recursive value is, syntactically, a term $\mu(x:A)a$, with the identification:

$$\mu(x:A)a \equiv \mu(y:A)a\{x \leftarrow y\}$$

The standard rules for recursive values are:

$$\frac{E,x:A \vdash_\rho a : A}{E \vdash_\rho \mu(x:A)a : A} \qquad \frac{E,x:A \vdash_\rho a : A}{E \vdash_\rho \mu(x:A)a \leftrightarrow a\{x \leftarrow \mu(x:A)a\} : A} \qquad \frac{E,x:A \vdash_\rho a \leftrightarrow b : A}{E \vdash_\rho \mu(x:A)a \leftrightarrow \mu(x:A)b : A}$$

5.4.2 Label sets

The next extension involves variables W ranging over sets of labels. We allow these in environments, under an assumption $W\#L$ that W does not contain any of the labels in L.

$$\frac{E \vdash_\rho L \# M}{E \vdash_\rho M \# L} \qquad \frac{E \vdash_\rho L \# M}{E \vdash_\rho L \# \varnothing} \qquad \frac{E \vdash_\rho L \# l.M}{E \vdash_\rho L \# M}$$

$$\frac{\vdash_\rho E\ env}{E \vdash_\rho \varnothing \# l_1 .. l_n.\varnothing} \qquad \frac{E \vdash_\rho L \# l.M}{E \vdash_\rho l.L \# M}$$

$$\frac{E \vdash_\rho L \# \varnothing \quad W \notin dom(E)}{\vdash_\rho E,\ W \# L\ env} \qquad \frac{\vdash_\rho E,\ W \# L\ env}{E,\ W \# L,\ E' \vdash_\rho W \# L}$$

The rules of $F_{<:}\rho$ that involve label sets L, are extended to require $L\#\varnothing$, to make sure that L is well-formed. We do not define quantifiers or functions over label-set variables because we do not know how to translate them into $F_{<:}$; label-set variables will be used only in definitions.

5.4.3 Definitions

We now extend the system with various flavors of definitions. The simplest definitions are value and row value definitions (*let*'s):

$$\frac{E \vdash_\rho a : A \quad E,x{:}A \vdash_\rho b : B}{E \vdash_\rho \mathit{let}\ x : A = a\ \mathit{in}\ b : B} \qquad \frac{E \vdash_\rho a : A \quad E,x{:}A \vdash_\rho b : B}{E \vdash_\rho \mathit{let}\ x : A = a\ \mathit{in}\ b \leftrightarrow b\{x \leftarrow a\} : B}$$

$$\frac{E \vdash_\rho r \therefore R \uparrow L \quad E,x{\therefore}R \uparrow L \vdash_\rho b : B}{E \vdash_\rho \mathit{let}\ x \therefore R \uparrow L = r\ \mathit{in}\ b : B} \qquad \frac{E \vdash_\rho r \therefore R \uparrow L \quad E,x{\therefore}R \uparrow L \vdash_\rho b : B}{E \vdash_\rho \mathit{let}\ x \therefore R \uparrow L = r\ \mathit{in}\ b \leftrightarrow b\{x \leftarrow r\} : B}$$

There are several kinds of type-level definitions (*Let*'s); we may give a definition of either a type variable, a row type variable, or a label-set variable, in the scope of either a type, a row type, a value, a row value, or a label-set.

To compress several cases into one, we use the abbreviations:

$\mathcal{X},\mathcal{Y}$ are either type, row type, or label-set variables;
$\mathcal{A},\mathcal{B},\mathcal{C}$ are either types, row type, or label sets;
$\mathcal{A}a,\mathcal{B}b,\mathcal{C}c$ are either values, row values, types, row types, or label-sets;
pred is either $: A$, $\therefore R \uparrow L$, *type*, $\uparrow L$, or $\sharp L$.
$\in \mathcal{K}$ is either $\prec : A$, $\uparrow L$, or $\sharp L$ (we often omit $\prec : Top$);
$\mathcal{A}a\{X\}$ means X may occur in $\mathcal{A}a$; then $\mathcal{A}a\{\mathcal{B}\}$ stands for $\mathcal{A}a\{X \leftarrow \mathcal{B}\}$

For type, row type, and label-set definitions, in various scopes, we have the rules:

$$\mathit{Let}\ X = \mathcal{A}\ \mathit{in}\ \mathcal{B}b\{X\} \quad \equiv \quad \mathit{Let}\ X' = \mathcal{A}\ \mathit{in}\ \mathcal{B}b\{X'\}$$

$$\frac{E \vdash_\rho \mathcal{A} \in \mathcal{K} \quad E \vdash_\rho \mathcal{B}b\{\mathcal{A}\}\ \mathit{pred}}{E \vdash_\rho \mathit{Let}\ X \in \mathcal{K} = \mathcal{A}\ \mathit{in}\ \mathcal{B}b\{X\}\ \mathit{pred}} \qquad \frac{E \vdash_\rho \mathcal{A} \in \mathcal{K} \quad E \vdash_\rho \mathcal{B}b\{\mathcal{A}\}\ \mathit{pred}}{E \vdash_\rho \mathit{Let}\ X \in \mathcal{K} = \mathcal{A}\ \mathit{in}\ \mathcal{B}b\{X\} \leftrightarrow \mathcal{B}b\{\mathcal{A}\}\ \mathit{pred}}$$

Note that, unlike value definitions, we do not require $E, X \in \mathcal{K} \vdash_\rho \mathcal{B}b\{X\}\ \mathit{pred}$; this might not be typeable on its own.

We also introduce *parametric* type-level definitions, for example:

$$\mathit{Let}\ X[\mathcal{Y},\mathcal{Z}] = \mathcal{A}\{\mathcal{Y},\mathcal{Z}\}\ \mathit{in}\ \ldots\ X[\mathcal{B}_1,\mathcal{C}_1]\ \ldots\ X[\mathcal{B}_2,\mathcal{C}_2]\ \ldots$$
$$\leftrightarrow \ldots\ \mathcal{A}\{\mathcal{B}_1,\mathcal{C}_1\}\ \ldots\ \mathcal{A}\{\mathcal{B}_2,\mathcal{C}_2\}\ \ldots$$

for which we omit the obvious but technically complicated definitions.

Finally, we use *top level* declarations, in the following way:

let x : A = a
let y : B = b stands for *let x : A = a in let y : B = b in c*
c

and similarly for *Let*.

We now have enough useful features, and we can turn to examples.

5.5 Examples

Many examples in this section are adapted from [Canning Cook Hill Olthoff Mitchell 89] [Harper Pierce 90] and [Cardelli Mitchell 91].

We start with a list of standard test cases and compare them with other calculi.

- ***Extracting a field from a record that is known to possess it***.

$$let\ select_x : Rcd(x{:}Nat,Etc) \rightarrow Nat =$$
$$\lambda(a{:}Rcd(x{:}Nat,Etc))\ a.x$$

$$select_x(rcd(x=3,y=true,etc)) \leftrightarrow 3 : Nat$$

- ***Extracting a field from a record that is not known to possess it***.

This is a typing error in all the calculi that have been proposed.

- ***Removing a field from a record that is known to possess it.***

$$let\ restrict_x : \forall(X \uparrow x)\ Rcd(x{:}Nat,X) \rightarrow ...X... =$$
$$\lambda(X \uparrow x)\ \lambda(a{:}Rcd(x{:}Nat,X))\ ...\ a\backslash x\ ...$$ (in a row context)

$$restrict_x(y{:}Nat,Etc \uparrow x)(rcd(x=3,y=true,etc))$$

- ***Removing a field from a record that is not known to possess it.***

This is the crucial feature in [Cardelli Mitchell 91]. It is not possible here because the translation (section 6) requires exact knowledge of the missing fields.

- ***Adding a field to a record that is known not to possess it.***

Not applicable; all records are already "complete". However, we can add a field to a row that is known not to possess it:

$$\lambda(r{::}R \uparrow x.L)\ ...\ x=b,r\ ...$$ (in a row context)

- ***Adding a field to a record that is not known to possess it.***

Not applicable; all records are already "complete". Moreover, even for rows, "not knowing" is not a sufficient condition for adding a field. This operation is possible in [Wand 87], [Rémy 89], and [Cardelli Mitchell 91].

- ***Updating a field of a record that is known to possess it.***

Although adding a field under these conditions is not possible because all records are "complete", there is no problem with updating. Note that type information about additional input fields is preserved. This example motivated the work [Cardelli Mitchell 91].

$$let\ replace_x : \forall(X \uparrow x)\ \forall(A)\ Rcd(x{:}Top,X) \rightarrow A \rightarrow Rcd(x{:}A,X) =$$
$$\lambda(X \uparrow x)\ \lambda(A)\ \lambda(r{:}Rcd(x{:}Top,X))\ \lambda(a{:}A)\ rcd(x=a,r\backslash x)$$

$$replace_x(y{:}Bool,Etc \uparrow x)(String)(rcd(x=3,y=true,etc))("str")$$
$$\leftrightarrow\ rcd(x="str",y=true,etc) : Rcd(x{:}String,y{:}Bool,Etc)$$

A restricted version, called *consistent updating*, preserves the type of the field being updated.

$$let\ update_x : \forall(X \uparrow x)\ \forall(A)\ Rcd(x{:}A,X) \rightarrow A \rightarrow Rcd(x{:}A,X) =$$
$$\lambda(X \uparrow x)\ \lambda(A)\ \lambda(b{:}Rcd(x{:}A,X))\ \lambda(a{:}A)\ rcd(x=a,b\backslash x)$$

An interesting example of update occurs when "moving" the *x* field of a point. In this case we want to preserve the type of the *y* field (whatever subtype of *Int* that may be) and all the additional fields. If the input type of the *x* field is *0..9* (a proper subtype of *Int*), the corresponding output type must be *Int*, otherwise we could exceed the range *0..9* for *x*.

$$let\ move_x : \forall(Y<:Int)\ \forall(Z\uparrow x,y)\ Rcd(x:Int,y:Y,Z) \rightarrow Rcd(x:Int,y:Y,Z) =$$
$$\lambda(Y<:Int)\ \lambda(Z\uparrow x,y)\ \lambda(p:Rcd(x:Int,y:Y,Z))\ rcd(p.x+1,p\backslash x)$$

$$p:Rcd(x:0..9,y:0..9,c:Color,Etc)$$
$$move_x(0..9)(c:Color,Etc)(p) : Rcd(x:Int,y:0..9,c:Color,Etc)$$

A more challenging task is to update "deep" in a structure, while preserving all the type information of the input. Here it can be achieved as follows, for a second-level boolean update.

$$let\ deep\text{-}update_{xy} :$$
$$\forall(X\uparrow x)\ \forall(Y\uparrow y)\ Rcd(x:Rcd(y:Bool,Y),X)\rightarrow Rcd(x:Rcd(y:Bool,Y),X) =$$
$$\lambda(X\uparrow x)\ \lambda(Y\uparrow y)\ \lambda(a:Rcd(x:Rcd(y:Bool,Y),X))$$
$$rcd(x=rcd(y=not(a.x.y),a.x\backslash y),a\backslash x)$$

$$deepUpdate_{xy}(z:Nat,Etc\uparrow x)(w:Nat,Etc\uparrow y)(rcd(x=rcd(y=true,w=3,etc),z=4,etc))$$
$$\leftrightarrow\ rcd(x=rcd(y=false,w=3,etc),z=4,etc)$$
$$: Rcd(x:Rcd(y:Bool,w:Nat,Etc),z:Nat,Etc)$$

- ***Updating a field of a record that is not known to possess it.***
  Again, "not knowing" is not a sufficient condition here.

- ***Renaming.***
  Renaming is not possible in general. Consider, for example, $Rcd(x:A,X)\rightarrow Rcd(y:A,X)$; what would be the constraint on *X*?

We now pass to standard examples of "class hierarchies" and "methods". We use parametric type definitions, explained in section 5.4, to model record type extension, as in [Harper Pierce 90]. This technique compensates, up to a point, for the lack of the type operations of [Cardelli Mitchell 91].

- ***Points and color points***
  A point has components *x:Int, y:Int*, while a color point also has a component *c:Color*. The challenge is to define the *ColorPoint* type and values by reusing the *Point* type and values. Here we can reuse types in two steps by defining a parametric version of each type. (Similarly for values.) This is an instance of a powerful *generator* technique, widely employed in [Cook 89].

$$Let\ PointPlus[Z\uparrow x,y] =$$
$$Rcd(x:Int,\ y:Int,\ Z)$$

```
Let Point =
    PointPlus[Etc]                          ( ≡ Rcd(x,y:Int, Etc) )

Let ColorPointPlus[Z↑x,y,c] =
    PointPlus[c:Color, Z]                   ( ≡ Rcd(x,y:Int, c:Color, Z) )
Let ColorPoint =
    ColorPointPlus[Etc]                     ( ≡ Rcd(x,y:Int, c:Color, Etc) )

let originPlus: ∀(Z↑x,y) Z↑x,y→PointPlus[Z] =
    λ(Z↑x,y) λ(z:Z↑x,y) rcd(x=0, y=0, z)

let origin : Point =
    originPlus(Etc↑x,y)(etc↑x,y)

let whiteOriginPlus : ∀(Z↑x,y,c) Z↑x,y,c→ColorPointPlus[Z] =
    λ(Z↑x,y,c) λ(z:Z↑x,y,c) originPlus(c:Color, Z ↑x,y)(c=white, z ↑x,y)

let whiteOrigin : ColorPoint =
    whiteOriginPlus(Etc↑x,y,c)(etc↑x,y,c)
```

- ***Total orders***

Here we have a record type *TO* of total orders. The ordering is represented as a method *leq: TO→Bool*, that compares another element of *TO* to the *self* value. The type *TO* is then recursive in the input type of its only method.

The definition of *TO* is done in three steps; first we introduce a generator with open recursion (the *Self* type parameter), then a generator derived from it where the recursion is closed, and finally the actual type *TO*. In general, the last two steps are obtained uniformly from the first. This technique is a bit complex, but it should be seen as a standard way of translating a "class" written in some more amenable language.

```
Let TOGenPlus[Self, X↑leq] =
    Rcd(leq: Self→Bool,X)
Let TOPlus[X↑leq] =
    μ(Self) TOGenPlus[Self, X]          ( ≡ μ(Self) Rcd(leq: Self→Bool, X))
Let TO =
    TOPlus[Etc]                         ( ≡ μ(Self) Rcd(leq: Self→Bool, Etc))
```

Next we define the total order of Naturals (by reusing *TOGenPlus*), as:

```
Let NatTOGenPlus[Self, X↑leq,val,add] =
    TOGenPlus[Self, (val:Nat, add:Self→Self, X)]
        ( ≡ Rcd(leq:Self→Bool, val:Nat, add:Self→Self, X))
Let NatTOPlus[X↑leq,val,add] =
    μ(Self) NatTOGenPlus[Self, X]
        ( ≡ μ(Self) Rcd(leq:Self→Bool, val:Nat, add:Self→Self, X))
```

Let NatTO =
NatTOPlus[Etc]
(≡ μ(Self) Rcd(leq:Self→Bool, val:Nat, add:Self→Self, Etc))

let zero : NatTO =
rcd(val=0, add=λ(other:NatTO) other,
leq=λ(other:NatTO) 0≤other.val, etc)

(The methods of *zero* are too specialized to be inherited; this problem can be amended by defining a value generator with open recursion and, for example, *leq=λ (other: NatTO) self.val≤other.val*.)

We now discover that, although *NatTO* was obtained by adding components to *TO*, it is not a subtype of *TO* by the rules for recursive types. Hence we have the unpleasant situation that operations defined on *TO* may not apply to particular total orders.

The solution is to define those operations on *TOPlus* instead of *TO*. (As pointed out in [Harper Pierce 90] this can be done even without *F-bounded quantification* [Canning Cook Hill Olthoff Mitchell 89] in a calculus of "negative information", such as $F_{<:}\rho$.) We can say that *NatTOPlus* is a *subclass* of *TOPlus* [Cook 89].

let min : ∀(X↑ leq) TOPlus[X]→TOPlus[X]→TOPlus[X] =
λ(X↑ leq) λ(a: TOPlus[X]) λ(b: TOPlus[X])
if a.leq(b) then a else b

We can then specialize *min* to *NatTO*:

let minNat: NatTO→NatTO→NatTO =
min(val:Nat, add: NatTO→NatTO, Etc ↑ leq)

to see that this typechecks, compute:

TOPlus[val:Nat, add: NatTO→NatTO, Etc]
≡ μ(Self) TOGenPlus[Self, (val:Nat, add: NatTO→NatTO, Etc)]
≡ μ(Self) Rcd(leq: Self→Bool, val:Nat, add: NatTO→NatTO, Etc) *(𝒜)*
↔ NatTO *(ℬ)*

The step from formula $\mathcal{A}$ to formula $\mathcal{B}$ proceeds as follows, using the rules for recursive types given in section 5.4. By unfolding, we have:

$\mathcal{A} \leftrightarrow$ *Rcd(leq: 𝒜→Bool, val:Nat, add: ℬ→ℬ, Etc)*
$\mathcal{B} \leftrightarrow$ *Rcd(leq: ℬ→Bool, val:Nat, add: ℬ→ℬ, Etc)*

Consider the contractive context $\mathcal{C}[X]$:

$\mathcal{C}[X] \equiv$ *Rcd(leq: X→Bool, val:Nat, add: ℬ→ℬ, Etc)*

Then $\mathcal{A} \leftrightarrow \mathcal{C}[\mathcal{A}]$ and $\mathcal{B} \leftrightarrow \mathcal{C}[\mathcal{B}]$; hence $\mathcal{A} \leftrightarrow \mathcal{B}$ by the contract rule.

• ***Movables***

Following the three-step schema, we now give type definitions for "things that can be moved". For added flexibility, the first step defines a row type instead of a record type, using a label-set parameter (explained in section 5.4).

```
Let MovableGenPlus[Self, L #move, X↑move.L] ↑L =
    move: Int→Int→Self, X
Let MovablePlus[X↑move] =
    μ(Self) Rcd(MovableGenPlus[Self, ∅, X])
                                    ( ≡ μ(Self) Rcd(move: Int→Int→Self, X))
Let Movable =
    MovablePlus[Etc]                ( ≡ μ(Self) Rcd(move: Int→Int→Self, Etc))

let translate : ∀(X↑move) MovablePlus[X]→Int→Int→MovablePlus[X] =
    λ(X↑move) λ(m:MovablePlus[X]) λ(dx:Int) λ(dy:Int) m.move(dx)(dy)
```

We can see that in this case *Movable* is a rather useless type. The interesting definition is *MovablePlus*, which however must be instantiated before it can be used. Hence, we combine movables with points:

```
Let PointPlus[Z↑x,y] =
    Rcd(x:Int, y:Int, Z)
Let Point =
    PointPlus[Etc]     ( ≡ Rcd(x:Int, y:Int, Etc))

Let MPointGenPlus[Self, X↑x,y,move] =
    PointPlus[MovableGenPlus[Self, (x,y), X]]
                    ( ≡ Rcd(x:Int, y:Int, move: Int→Int→Self, X))
Let MPointPlus[X↑x,y,move] =
    μ(Self) MPointGenPlus[Self, X]
                    ( ≡ μ(Self) Rcd(x:Int, y:Int, move: Int→Int→Self, X))
Let MPoint =
    MPointPlus[Etc]  ( ≡ μ(Self) Rcd(x:Int, y:Int, move: Int→Int→Self, Etc))

let move : ∀(X↑x,y,move) MPointPlus[X]→Int→Int→MPointPlus[X] =
    λ(Z↑x,y,move) λ(self:MPointPlus[X] λ(dx:Int) λ(dy:Int)
        rcd(x=self.x+dx, y=self.y+dy, self\x,y)

let mOrigin : MPoint =
    μ(self:MPoint) rcd(x=0, y=0, move=move(Etc↑x,y,move)(self), etc)

translate(x:Int,y:Int,Etc ↑move)(mOrigin)(1)(1) : MPoint
```

Note that in *MPointGenPlus* we have successfully reused the definitions for both points and movables. Moreover, *move* can be inherited by subclasses (as opposed to subtypes) of *MPointPlus*, by defining appropriate generators.

• ***Concatenation***

Record concatenation can be handled by adapting a technique of Rémy [Rémy 91]. With an extra level of encoding, record concatenation can be modeled by function composition; in our system, this idea can be realized as follows.

We first define *segments*, as extensible records parameterized by their potential extensions:

$$Seg(l_1{:}A_1,..,l_n{:}A_n) \triangleq \forall(Z\uparrow l_1..l_n)\ Z\uparrow l_1..l_n \to Rcd(l_1{:}A_1,..,l_n{:}A_n,Z)$$
$$seg(l_1{=}a_1,..,l_n{=}a_n) \triangleq \lambda(Z\uparrow l_1..l_n)\ \lambda(z{::}Z\uparrow l_1..l_n)\ rcd(l_1{=}a_1,..,l_n{=}a_n,z)$$

A field of a segment can be extracted by *precipitating* the segment to a record:

$$s.l_i \triangleq s(Etc\uparrow l_1..l_n)(etc\uparrow l_1..l_n).l_i \qquad \text{where } s : Seg(l_1{:}A_1,..,l_n{:}A_n),\ i\in 1..n$$

Then, given two segments with distinct sets of labels:

$$s : Seg(l_1{:}A_1,..,l_n{:}A_n) \equiv \forall(Z\uparrow l_1..l_n)\ Z\uparrow l_1..l_n \to Rcd(l_1{:}A_1,..,l_n{:}A_n,Z)$$
$$t : Seg(k_1{:}B_1,..,k_m{:}B_m) \equiv \forall(Z\uparrow k_1..k_m)\ Z\uparrow k_1..k_m \to Rcd(k_1{:}B_1,..,k_m{:}B_m,Z)$$

we can define their concatenation ($\|$) as follows:

$$s \| t \triangleq$$
$$\lambda(Z\uparrow l_1..l_nk_1..k_m)\ \lambda(z{::}Z\uparrow l_1..l_nk_1..k_m)$$
$$s(k_1{:}B_1,..,k_m{:}B_m,Z\uparrow l_1..l_n)$$
$$(t(l_1{:}Top,..,l_n{:}Top,Z\uparrow k_1..k_m)(l_1{=}top,..,l_n{=}top,z\uparrow k_1..k_m)\backslash l_1..l_n)$$

so that we have:

$$s \| t : Seg(l_1{:}A_1,..,l_n{:}A_n,k_1{:}B_1,..,k_m{:}B_m)$$

It would now be possible to axiomatize an extension of $F_{<:}\rho$ with segments and concatenation, and define a translation of this extended calculus into $F_{<:}\rho$.

6. Translation of $F_{<:}\rho$ into $F_{<:}$

In this section we define the promised translation from a calculus with rows to one without rows. The basic idea is that row variables, row types, row values, and row judgments become rows or sequences of, respectively, variables, types, values, and judgments.

We start with some familiar notation from previous sections:

Notation

$\mathcal{L}$	the set of labels
$\iota : \mathcal{L}\to Nat$	(a bijection) a fixed enumeration of labels
$l^i \triangleq \iota^{-1}(i)$	the label whose index is i in the fixed enumeration
$L,M...$	finite sets of labels
$\#S$	size of a finite set

Next we define the set of indices of a set of labels, and its maximum index:

Definition (indices and maximum index of a set of labels)

$\iota L \triangleq \{\iota(l) | l \in L\} = \{i | l_i \in L\}$

$\Uparrow L \triangleq max(\iota L)$, *where* $\Uparrow\{\} \triangleq -1$

Finite sets of labels L are used mostly in contexts like $\uparrow L$, describing the labels a row *lacks*. If we need to talk about the labels a row *has*, we can consider the complement $\mathcal{L}\text{-}L$. This, though, is an infinite set, and the part beyond $\Uparrow L$ is uninteresting. Hence, it is natural to take its most interesting finite prefix, κL:

Definition (finite complement prefix of a finite set of labels)

$\kappa L \triangleq \{i | i < \Uparrow L \wedge l_i \notin L\}$

A central concept in the sequel is that of the *dimension* of (the tuple translation of) a row. Take any row that is undefined at L; that is, any row whose tuple translation sketched in section 4.2 has gaps at L. Then the labeled components to the right of the last gap ($\Uparrow L$) are contiguous, and they can be collected into a single tuple; we call the result a *normal* row. The dimension ∂L of any row that has gaps at L is then defined as the number of components of the corresponding normal row. We emphasize that for any row $r :\cdot R \uparrow L$ or $R \uparrow L$, its dimension depends only on L, and not on the structure of r or R. Hence ∂L can be defined very simply as:

Definition (dimension of a row undefined at L)

$\partial L \triangleq \#(\kappa L)+1$

When adding a new item to a row, the row dimension changes depending on whether the new item fills the last gap of the row or not. In the former case, a whole set of components may be compacted in the final tuple and the dimension decreases; in the latter case, the dimension increases by one. The following lemma is formulated in terms of adding or removing a gap.

Lemma (row dimension)

For $l_i \notin L$,

if $i < \Uparrow L$ then $\partial(l_i.L) = \partial L - 1$;

if $i > \Uparrow L$ then $\partial(l_i.L) = \partial L + (i - (\Uparrow L + 1))$.

We now need some notation for describing complex sequences and rows, and for this purpose we use a notation similar to set comprehension. For example, we use ${}^{i}\langle i | 2 \le i \le 4 \rangle$ to denote the sequence *2,3,4* in this order; the idea is that the superscript index i is increased monotonically to generate the elements of the sequence.

Notation (sequences)

$\#(S)$	length of a sequence
S,S'	sequence concatenation

$^{i}\langle \varphi(i) | \Phi(i) \rangle$ sequence comprehension; the sequence, generated by increasing i, whose elements are $\varphi(i)$ for $i \in Nat \wedge \Phi(i)$.

A row is a sequence of labeled elements, sorted by label index, of length greater than zero. The last element of a row is special; as discussed in section 4.2 this is the *rest* of the row. For bookkeeping purposes, we use the special label $\lfloor^{q}$ for this last element, where q is intuitively the index of the beginning of the uninteresting part of the row (as we can see from the row structure lemma below).

Notation (rows)

A type row R is a sequence of the form:

$l_1{:}A_1..l_n{:}A_n$ where $n \geq 1$ and $l_n \equiv \lfloor^{q}$ for some q.

A value row r is a sequence of the form:

$l_1{=}a_1..l_n{=}a_n$ where $n \geq 1$ and $l_n \equiv \lfloor^{q}$ for some q.

$\#(l_1{:}A_1..l_n{:}A_n) \triangleq n;$	$\#(l_1{=}a_1..l_n{=}a_n) \triangleq n$	size
$l_i{:}A_i \in l_1{:}A_1..l_n{:}A_n;$	$l_i{=}a_i \in l_1{=}a_1..l_n{=}a_n$	membership ($i \in 1..n$)
$l{:}A \rightsquigarrow R \triangleq$	$^{i}\langle (l^i{:}A^i) \mid (l^i{:}A^i) \equiv (l{:}A) \wedge (l^i{:}A^i) \in R \rangle$	sorting (if $l{:}B \notin R$ for any B)
$l{=}a \rightsquigarrow r \triangleq$	$^{i}\langle (l^i{=}a^i) \mid (l^i{=}a^i) \equiv (l{=}a) \wedge (l^i{=}a^i) \in r \rangle$	sorting (if $l{=}b \notin r$ for any b)

We can now define some basic sequences and rows that will be used in the translation. All these have dimension ∂L.

Definition (basic sequences and rows)

$VarSeq(X, \uparrow L) \triangleq {}^{i}\langle X^i \mid i \in \kappa L \rangle, X^{\Uparrow L+1}$

$VarRow(X, \uparrow L) \triangleq {}^{i}\langle (l^i{:}X^i) \mid i \in \kappa L \rangle, (\lfloor^{\Uparrow L+1}{:}X^{\Uparrow L+1})$

$TopRow(\uparrow L) \triangleq {}^{i}\langle (l^i{:}Top) \mid i \in \kappa L \rangle, (\lfloor^{\Uparrow L+1}{:}Top)$

$varSeq(x, \uparrow L) \triangleq {}^{i}\langle x^i \mid i \in \kappa L \rangle, x^{\Uparrow L+1}$

$varRow(x, \uparrow L) \triangleq {}^{i}\langle (l^i{=}x^i) \mid i \in \kappa L \rangle, (\lfloor^{\Uparrow L+1}{=}x^{\Uparrow L+1})$

$topRow(\uparrow L) \triangleq {}^{i}\langle (l^i{=}top) \mid i \in \kappa L \rangle, (\lfloor^{\Uparrow L+1}{=}top)$

$selRow(a, \uparrow L) \triangleq {}^{i}\langle (l^i{=}a.i) \mid i \in \kappa L \rangle, (\lfloor^{\Uparrow L+1}{=}a\lfloor^{\Uparrow L+1})$

Examples

$VarRow(X, \uparrow()) = \lfloor^{0}{:}X^0$	$TopRow(\uparrow()) = \lfloor^{0}{:}Top$
$VarRow(X, \uparrow l^0) = \lfloor^{1}{:}X^1$	$TopRow(\uparrow l^0) = \lfloor^{1}{:}Top$
$VarRow(X, \uparrow l^1) = l^0{:}X^0, \lfloor^{2}{:}X^2$	$TopRow(\uparrow l^1) = l^0{:}Top, \lfloor^{2}{:}Top$
$VarRow(X, \uparrow l^0, l^2) = l^1{:}X^1, \lfloor^{3}{:}X^3$	$TopRow(\uparrow l^0, l^2) = l^1{:}Top, \lfloor^{3}{:}Top$

In defining the full translation, $[\![-]\!]$, we need an auxiliary translation, $(\!| - \uparrow L |\!)$, for converting row types $R \uparrow L$, and row values $r \uparrow L$, into rows of types and values, respectively. The results of $(\!| - \uparrow L |\!)$ are unnormalized, in the sense that they may have a dimension greater than ∂L; that is, the final tupleable components of the results need not be grouped together into a tuple. This auxiliary translation refers back to the proper translation, $[\![-]\!]$, but for exposition purposes we present it first.

Definition (translation, part 1; auxiliary row translation)

$\langle\!| X \uparrow L |\!\rangle \triangleq VarRow(X, \uparrow L)$
$\langle\!| Etc \uparrow L |\!\rangle \triangleq TopRow(\uparrow L)$
$\langle\!| l{:}A,R \uparrow L |\!\rangle \triangleq l{:}[\![A]\!] \rightsquigarrow \langle\!| R \uparrow l.L |\!\rangle$

$\langle\!| x \uparrow L |\!\rangle \triangleq varRow(x, \uparrow L)$
$\langle\!| etc \uparrow L |\!\rangle \triangleq topRow(\uparrow L)$
$\langle\!| l{=}a,r \uparrow L |\!\rangle \triangleq l{=}[\![a]\!] \rightsquigarrow \langle\!| r \uparrow l.L |\!\rangle$
$\langle\!| a\backslash l_1..l_n \uparrow L |\!\rangle \triangleq selRow([\![a]\!], \uparrow L)$

Hence, for the base cases $\langle\!| X \uparrow L |\!\rangle$ and $\langle\!| Etc \uparrow L |\!\rangle$ of the row type translation, we produce rows of *X*'s or *Top*'s of size ∂L. For $\langle\!| l{:}A,R \uparrow L |\!\rangle$ we first compute $\langle\!| R \uparrow l.L |\!\rangle$, which has an additional gap for *l*, and we sort $l{:}[\![A]\!]$ into the result.

Similarly for the row value translation. In addition, $\langle\!| a\backslash l_1..l_n \uparrow L |\!\rangle$ produces a row of record selections; the idea here is that eliminating $l_1..l_n$ from *a* is the same as selecting and reassembling all the other components of *a*. (The type rules will ensure $(l_1..l_n) \equiv L$, if $a\backslash l_1..l_n \uparrow L$ is well-typed.)

Here is an example of the translation:

$\langle\!| X \uparrow l^0,l^1,l^3,l^6 |\!\rangle \equiv$
$\quad l^2{:}X^2,\ l^4{:}X^4,\ l^5{:}X^5,\ \lfloor^7{:}X^7$ (of size $\partial(l^0,l^1,l^3,l^6)$)

$\langle\!| (l^1{:}A^1,\ l^6{:}A^6,\ X) \uparrow l^0,l^3 |\!\rangle \equiv$
$\quad l^1{:}A^1,\ l^2{:}X^2,\ l^4{:}X^4,\ l^5{:}X^5,\ l^6{:}A^6,\ \lfloor^7{:}X^7$ (of size greater than $\partial(l^0,l^3)$)

Next, we provide a kind of normal form for row types $l_1{:}A_1..l_n{:}A_n,\xi$, based on the translations $\langle\!| R \uparrow L |\!\rangle$ (under typing assumptions). As we have seen, the translation returns rows whose length (which depends both on *L* and $l_1..l_n$) may exceed ∂L. The normal form reveals that the portion beyond ∂L-*1* has in fact no gaps and therefore can be collected into a tuple to form a single ∂L^{th} element. Similarly for value rows.

Lemma (row structure)

(1) Let $R \equiv l_1{:}A_1..l_n{:}A_n,\xi$ where $\xi{=}X$ or $\xi{=}Etc$.
Assume $E \vdash_\rho R \uparrow L$.
Then $\langle\!| R \uparrow L |\!\rangle$ has the following shape, for some *B*'s:
$\quad {}^i\langle (l^i{:}B^i) \mid i \in \kappa L \rangle\ ,\ {}^j\langle (l^j{:}B^j) \mid \Uparrow L < j < q \rangle,\ (\lfloor^q{:}B^q)$
$\quad$ with $q = (\Uparrow L+1) + (\partial(l_1..l_n.L)+n - \partial L) \quad (q > \Uparrow L)$

(2) Let $r \equiv l_1{=}a_1..l_n{=}a_n,\ \xi$ *where* $\xi{=}x$, $\xi{=}etc$, *or* $\xi{=}a\backslash M$.
Assume $E \vdash_\rho r :\cdot R \uparrow L$.
Then $\langle\!| r \uparrow L |\!\rangle$ has the following shape, for some *b*'s:
$\quad {}^i\langle (l^i{=}b^i) \mid i \in \kappa L \rangle\ ,\ {}^j\langle (l^j{=}b^j) \mid \Uparrow L < j < q \rangle,\ (\lfloor^q{=}b^q)$
$\quad$ with $q = (\Uparrow L+1) + (\partial(l_1..l_n.L)+n - \partial L) \quad (q > \Uparrow L)$

Considering the previous example:

$\langle\!|(l^1{:}A^1,\ l^6{:}A^6,\ X) \uparrow l^0,l^3|\!\rangle \equiv$

$(l^1{:}A^1,\ l^2{:}X^2),\ (l^4{:}X^4,\ l^5{:}X^5,\ l^6{:}A^6),\ \lfloor^7{:}X^7$

$\backslash_____/$ ---------- of size $\partial(l^0,l^3)$-1

$\backslash_____/$ ----- tupleable, $\partial(l^0,l^3)^{th}$ item

Now we are ready for the full translation. The translation of the $F_{<:}$ fragment of $F_{<:}\rho$ is uninteresting, but we list it for completeness.

Definition (translation, part 2; $F_{<:}$ fragment)

Environments

$[\![\vdash_\rho E\ env]\!] \triangleq \vdash [\![E]\!]\ env$

$[\![\emptyset]\!] \triangleq \emptyset$

$[\![E,x{:}A]\!] \triangleq [\![E]\!],x{:}[\![A]\!]$

$[\![E,X{<:}A]\!] \triangleq [\![E]\!],X{<:}[\![A]\!]$

Types

$[\![E \vdash_\rho A\ type]\!] \triangleq [\![E]\!] \vdash [\![A]\!]\ type$

$[\![X]\!] \triangleq X$

$[\![Top]\!] \triangleq Top$

$[\![A \to B]\!] \triangleq [\![A]\!] \to [\![B]\!]$

$[\![\forall(X{<:}A)B]\!] \triangleq \forall(X{<:}[\![A]\!])[\![B]\!]$

Subtypes

$[\![E \vdash_\rho A{<:}B]\!] \triangleq [\![E]\!] \vdash [\![A]\!] <: [\![B]\!]$

Values

$[\![E \vdash_\rho a : A]\!] \triangleq [\![E]\!] \vdash [\![a]\!] : [\![A]\!]$

$[\![x]\!] \triangleq x$

$[\![top]\!] \triangleq top$

$[\![\lambda(x{:}A)b]\!] \triangleq \lambda(x{:}[\![A]\!])[\![b]\!]$

$[\![b(a)]\!] \triangleq [\![b]\!]([\![a]\!])$

$[\![\lambda(X{<:}A)b]\!] \triangleq \lambda(X{<:}[\![A]\!])[\![b]\!]$

$[\![b(A)]\!] \triangleq [\![b]\!]([\![A]\!])$

Value equivalence

$[\![E \vdash_\rho a \leftrightarrow a' : A]\!] \triangleq [\![E]\!] \vdash [\![a]\!] \leftrightarrow [\![a']\!] : [\![A]\!]$

Finally, we can give the translation of the proper $F_{<:}\rho$ judgments and terms. An $F_{<:}\rho$ judgment $E \vdash_\rho \vartheta \uparrow L$ becomes a sequence of size ∂L of $F_{<:}$ judgments. A row variable $X \uparrow L$ in an environment becomes a sequence of ∂L type variables. The domain of row function space $R \uparrow L \to B$ becomes a sequence of ∂L domains; similarly for $\lambda(x{:}R \uparrow L)$, with $b(r \uparrow L)$ becoming a sequence of ∂L applications. A row quantifier $\forall(X \uparrow L)$ becomes a nesting of ∂L type quantifiers; similarly for an abstraction $\lambda(X \uparrow L)$, with $b(R \uparrow L)$ becoming a nesting of ∂L type applications. Record types and values are translated by applying $\langle\!| - \uparrow L|\!\rangle$ to the respective rows, and then normalizing the results to size ∂L.

Definition (translation, part 3; $F_{<:}\rho$ proper)

Environments (continued)

$[\![E,X \uparrow L]\!] \triangleq$ *let* $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ *in* $[\![E]\!],X_1,..,X_{\partial L}$

$[\![E,x{:}R \uparrow L]\!] =$

let $x_1..x_{\partial L} = varSeq(x, \uparrow L)$ *and* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *in* $[\![E]\!],x_1{:}A_1..x_{\partial L}{:}A_{\partial L}$

Types (continued)

$[\![Rcd(R)]\!] \triangleq [\![R \uparrow ()]\!]$

$[\![R \uparrow L \to B]\!] =$ *let* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *in* $A_1 \to .. \to A_{\partial L} \to [\![B]\!]$

$[\![\forall(X \uparrow L)B]\!] \triangleq$ *let* $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ *in* $\forall(X_1)..\forall(X_{\partial L})[\![B]\!]$

Type rows

$[\![E \vdash_\rho R \uparrow L]\!] \triangleq$ *let* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *in* $[\![E]\!] \vdash A_1$ *type ...* $[\![E]\!] \vdash A_{\partial L}$ *type*

$[\![R \uparrow L]\!] \triangleq$ *let* $(l_1{:}A_1..l_{\partial L}{:}A_{\partial L}..l_n{:}A_n) = (\!|R \uparrow L|\!)$ *in* $A_1..A_{\partial L-1}, Tuple(A_{\partial L}..A_n)$

Subrows

$[\![E \vdash_\rho R <:\!: S \uparrow L]\!] \triangleq$

let $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *and* $B_1..B_{\partial L} = [\![S \uparrow L]\!]$

in $[\![E]\!] \vdash A_1 <: B_1 ... [\![E]\!] \vdash A_{\partial L} <: B_{\partial L}$

Values (continued)

$[\![rcd(r)]\!] \triangleq [\![r \uparrow ()]\!]$

$[\![a.l]\!] \triangleq [\![a]\!].\iota(l)$

$[\![\lambda(x{:}R \uparrow L)\ b]\!] =$

let $x_1..x_{\partial L} = varSeq(x, \uparrow L)$ *and* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *in* $\lambda(x_1{:}A_1)..\lambda(x_{\partial L}{:}A_{\partial L})\ [\![b]\!]$

$[\![b(r \uparrow L)]\!] =$ *let* $a_1..a_{\partial L} = [\![r \uparrow L]\!]$ *in* $[\![b]\!](a_1)..(a_{\partial L})$

$[\![\lambda(X \uparrow L)b]\!] \triangleq$ *let* $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ *in* $\lambda(X_1)..\lambda(X_{\partial L})[\![b]\!]$

$[\![b(R \uparrow L)]\!] \triangleq$ *let* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *in* $[\![b]\!](A_1)..(A_{\partial L})$

Value rows

$[\![E \vdash_\rho r{:}\!:R \uparrow L]\!] \triangleq$

let $a_1..a_{\partial L} = [\![r \uparrow L]\!]$ *and* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$ *in* $[\![E]\!] \vdash a_1{:}A_1 ... [\![E]\!] \vdash a_{\partial L}{:}A_{\partial L}$

$[\![r \uparrow L]\!] \triangleq$ *let* $(l_1{=}a_1..l_{\partial L}{=}a_{\partial L}..l_n{=}a_n) = (\!|r \uparrow L|\!)$ *in* $a_1..a_{\partial L-1}, tuple(a_{\partial L}..a_n)$

Value row equivalence

$[\![E \vdash_\rho r \leftrightarrow r' .\!: R \uparrow L]\!] \triangleq$

let $a_1..a_{\partial L} = [\![r \uparrow L]\!]$ *and* $a'_1..a'_{\partial L} = [\![r' \uparrow L]\!]$ *and* $A_1..A_{\partial L} = [\![R \uparrow L]\!]$

in $[\![E]\!] \vdash a_1 \leftrightarrow a'_1{:}A_1 ... [\![E]\!] \vdash a_{\partial L} \leftrightarrow a'_{\partial L}{:}A_{\partial L}$

Examples

$[\![\lambda(x{:}(l^0{:}A, Etc) \uparrow l^1)\ rcd(l^1{=}b, x)]\!] =$

$\lambda(x^0{:}[\![A]\!])\ \lambda(x^2{:}Top)\ tuple(x^0, [\![b]\!], x^2)$

$[\![\lambda(x{:}(l^2{:}A, Etc) \uparrow l^1)\ rcd(l^1{=}b, x)]\!] =$

$\lambda(x^0{:}\ Top)\ \lambda(x^2{:}Tuple([\![A]\!], Top))\ tuple(x^0, [\![b]\!], x^2)$

$[\![\lambda(X \uparrow l^0, l^1)\ \lambda(x{:}(l^0{:}A, X) \uparrow l^1)\ rcd(l^1{=}b, x)]\!] =$

$\lambda(X^2)\ \lambda(x^0{:}[\![A]\!])\ \lambda(x^2{:}X^2)\ tuple(x^0, [\![b]\!], x^2)$

$[\![\lambda(X \uparrow l^1, l^2)\ \lambda(x{:}(l^2{:}A, X) \uparrow l^1)\ rcd(l^1{=}b, x)]\!] =$

$\lambda(X^0)\ \lambda(X^3)\ \lambda(x^0{:}\ X^0)\ \lambda(x^2{:}Tuple([\![A]\!], X^3))\ tuple(x^0, [\![b]\!], x^2)$

Using the row structure lemma, we can now show that the translation is well-defined, provided that the translated terms are well-typed.

Lemma (translation dimensions)

$\#(VarSeq(X, \uparrow L)) = \#(VarRow(X, \uparrow L)) = \#(TopRow(\uparrow L))$

$= \#(varSeq(x, \uparrow L)) = \#(topRow(\uparrow L)) = \#(varRow(x, \uparrow L)) = \#(selRow(a, \uparrow L))$

$= \partial L.$

If $E \vdash_\rho R \uparrow L$ then $\#(\lfloor R \uparrow L \rfloor) \geq \partial L$.
If $E \vdash_\rho R \uparrow L$ then $\#([\![R \uparrow L]\!]) = \partial L$.
If $E \vdash_\rho r :: R \uparrow L$ then $\#(\lfloor r \uparrow L \rfloor) \geq \partial L$.
If $E \vdash_\rho r :: R \uparrow L$ then $\#([\![r \uparrow L]\!]) = \partial L$.

Lemma (good translation)

If a judgment J is derivable, then the translation $[\![J]\!]$ is well-defined.
That is, all the assumptions made in the translation about sizes of rows, are justified.

The row structure lemma is also the key to the following row analysis lemma, which is then used in the proof of all the technical lemmas in the next section. The row analysis lemma describes in detail what happens when a single element is added to a row, or removed from it.

Lemma (row analysis)

(1) Assume $E \vdash_\rho l{:}A, R \uparrow L$.
Let $B_1..B_{\partial l.L} = [\![R \uparrow l.L]\!]$ and $C_1..C_{\partial L} = [\![l{:}A, R \uparrow L]\!]$
If $\iota(l) < \Uparrow L$ then $\partial(l.L) = \partial L\text{-}1$, and:
$C_1..C_{\partial L} = B_1..B_{k-1}, [\![A]\!], B_k..B_{\partial L-1}$
where $k = \#\{i \mid i \in \kappa L \wedge i < \iota(l)\} \leq \partial L\text{-}1$.
If $\iota(l) > \Uparrow L$ then $\partial(l.L) = \partial L + (\iota(l) - (\Uparrow L + 1))$, and:
$C_1..C_{\partial L-1} = B_1..B_{\partial L-1} \quad C_{\partial L} = Tuple(B_{\partial L}..B_{\partial l.L-1}, [\![A]\!], B_{\partial l.L})$
where $\partial(l.L) \geq \partial L$.

(2) Assume $E \vdash_\rho l{=}a, r :: R \uparrow L$.
Let $b_1..b_{\partial l.L} = [\![r \uparrow l.L]\!]$ and $c_1..c_{\partial L} = [\![l{=}a, r \uparrow L]\!]$
If $\iota(l) < \Uparrow L$ then $\partial(l.L) = \partial L\text{-}1$, and:
$c_1..c_{\partial L} = b_1..b_{k-1}, [\![a]\!], b_k..b_{\partial L-1}$
where $k = \#\{i \mid i \in \kappa L \wedge i < \iota(l)\} \leq \partial L\text{-}1$.
If $\iota(l) > \Uparrow L$ then $\partial(l.L) = \partial L + (\iota(l) - (\Uparrow L + 1))$, and:
$c_1..c_{\partial L-1} = b_1..b_{\partial L-1} \quad c_{\partial L} = tuple(b_{\partial L}..b_{\partial l.L-1}, [\![a]\!], b_{\partial l.L})$
where $\partial(l.L) \geq \partial L$.

(3) Assume $E \vdash_\rho a \backslash L :: R \uparrow L$.
Let $b_1..b_{\partial l.L} = [\![a \backslash l.L \uparrow l.L]\!]$ and $c_1..c_{\partial L} = [\![a \backslash L \uparrow L]\!]$
If $\iota(l) < \Uparrow L$ then $\partial(l.L) = \partial L\text{-}1$, and:
$c_1..c_{\partial L} = b_1..b_{k-1}, [\![a.l]\!], b_k..b_{\partial L-1}$
where $k = \#\{i \mid i \in \kappa L \wedge i < \iota(l)\} \leq \partial L\text{-}1$.
If $\iota(l) > \Uparrow L$ then $\partial(l.L) = \partial L + (\iota(l) - (\Uparrow L + 1))$, and:
$c_1..c_{\partial L-1} = b_1..b_{\partial L-1} \quad c_{\partial L} = tuple(b_{\partial L}..b_{\partial l.L-1}, [\![a.l]\!], b_{\partial l.L})$
where $\partial(l.L) \geq \partial L$.

7. The translation preserves derivations

In this section we show that the translation from $\vdash_\rho$ to $\vdash$ is sound. That is, if a judgment J is derivable in $\vdash_\rho$, then all the judgments in the sequence $[\![J]\!]$ are derivable in $\vdash$.

The following group of lemmas is used in the hardest cases of the soundness proof. These lemmas are complicated by the fact that the translations are well-defined only under typing assumptions. First we have lemmas regarding rows; they have the structure of some of the inference rules, but concern the translation of those rules.

Lemma (soundness of row inference rules)

(type row cons)

Assume $E \vdash_\rho R' \uparrow l.L$ and $E \vdash_\rho A\ type$.
If $[\![E \vdash_\rho R' \uparrow l.L]\!]$ and $[\![E \vdash_\rho A\ type]\!]$ then $[\![E \vdash_\rho l{:}A, R' \uparrow L]\!]$.

(sub row cons)

Assume $E \vdash_\rho A <: B$ and $E \vdash_\rho R' <:: S' \uparrow l.L$.
If $[\![E \vdash_\rho A <: B]\!]$ and $[\![E \vdash_\rho R' <:: S' \uparrow l.L]\!]$ then $[\![E \vdash_\rho l{:}A, R' <:: l{:}B, S' \uparrow L]\!]$.

(row cons)

Assume $E \vdash_\rho a{:}A$ and $E \vdash_\rho r :: R \uparrow l.L$.
If $[\![E \vdash_\rho a{:}A]\!]$ and $[\![E \vdash_\rho r :: R \uparrow l.L]\!]$ then $[\![E \vdash_\rho l{=}a, r :: l{:}A, R \uparrow L]\!]$.

(selection)

Assume $E \vdash_\rho a : Rcd(l{:}A, S)$.
If $[\![E \vdash_\rho a : Rcd(l{:}A, S)]\!]$ then $[\![E \vdash_\rho a.l : A]\!]$.

(restriction)

Assume $E \vdash_\rho a\backslash L :: l{:}A, S \uparrow L$.
If $[\![E \vdash_\rho a\backslash L :: l{:}A, S \uparrow L]\!]$ then $[\![E \vdash_\rho a\backslash l.L :: S \uparrow l.L]\!]$.

(eq-cons)

Assume $E \vdash_\rho r \leftrightarrow r' :: R \uparrow l.L$, and $E \vdash_\rho a \leftrightarrow a' : A$.
If $[\![E \vdash_\rho r \leftrightarrow r' :: R \uparrow l.L]\!]$ and $[\![E \vdash_\rho a \leftrightarrow a' : A]\!]$
then $[\![E \vdash_\rho l{=}a, r \leftrightarrow l{=}a', r' :: l{:}A, R \uparrow L]\!]$

(eq-selection)

Assume $E \vdash_\rho a \leftrightarrow a' : Rcd(l{:}A, S)$.
If $[\![E \vdash_\rho a \leftrightarrow a' : Rcd(l{:}A, S)]\!]$ then $[\![E \vdash_\rho a.l \leftrightarrow a'.l : A]\!]$.

(eval-selection)

Assume $E \vdash_\rho r :: R \uparrow l$ and $E \vdash_\rho a \leftrightarrow a' : A$.
If $[\![E \vdash_\rho r :: R \uparrow l]\!]$ and $[\![E \vdash_\rho a \leftrightarrow a' : A]\!]$ then $[\![E \vdash_\rho rcd(l{=}a, r).l \leftrightarrow a' : A]\!]$.

(eq-restriction)

Assume $E \vdash_\rho a\backslash L \leftrightarrow a'\backslash L :: l{:}A, S \uparrow L$.
If $[\![E \vdash_\rho a\backslash L \leftrightarrow a'\backslash L :: l{:}A, S \uparrow L]\!]$ then $[\![E \vdash_\rho a\backslash l.L \leftrightarrow a'\backslash l.L :: S \uparrow l.L]\!]$.

(eval-restriction)

(1) Assume $E \vdash_\rho r \leftrightarrow r' \therefore R \uparrow ()$.
If $[\![E \vdash_\rho r \leftrightarrow r' \therefore R \uparrow ()]\!]$ then $[\![E \vdash_\rho rcd(r)\backslash \leftrightarrow r' \therefore R \uparrow ()]\!]$.

(2) Assume $E \vdash_\rho rcd(l{=}a,r)\backslash L \leftrightarrow l{=}a,r' \therefore l{:}A,R \uparrow L$.
If $[\![E \vdash_\rho rcd(l{=}a,r)\backslash L \leftrightarrow l{=}a,r' \therefore l{:}A,R \uparrow L]\!]$
then $[\![E \vdash_\rho rcd(l{=}a,r)\backslash l.L \leftrightarrow r' \therefore R \uparrow l.L]\!]$.

Next we have substitution lemmas for all possible combinations of variables and terms.

Lemma (soundness of substitution)

(type in type)

Assume $E \vdash_\rho A' <: A$ and $E,X<:A,E' \vdash_\rho B\ type$.
Then $[\![B\{X \leftarrow A'\}]\!]$ is well-defined.
Then $[\![B]\!]\{X \leftarrow [\![A']\!]\} \equiv [\![B\{X \leftarrow A'\}]\!]$.

(type in row-type)

Assume $E \vdash_\rho A' <: A$ and $E,X<:A,E' \vdash_\rho S \uparrow M$.
Then $[\![S\{X \leftarrow A'\} \uparrow M]\!]$ is well-defined.
Let $B_1..B_{\partial M} = [\![S \uparrow M]\!]$ and $C_1..C_{\partial M} = [\![S\{X \leftarrow A'\} \uparrow M]\!]$.
Then $B_1\{X \leftarrow [\![A']\!]\} \equiv C_1$ ... $B_{\partial M}\{X \leftarrow [\![A']\!]\} \equiv C_{\partial M}$.

(row-type in type)

Assume $E \vdash_\rho R \uparrow L$ and $E,X \uparrow L,E' \vdash_\rho B\ type$.
Then $[\![B\{X \leftarrow R\}]\!]$ is well-defined.
Let $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ and $A_1..A_{\partial L} = [\![R \uparrow L]\!]$
Then $[\![B]\!]\{X_1 \leftarrow A_1\}..\{X_{\partial L} \leftarrow A_{\partial L}\} \equiv [\![B\{X \leftarrow R\}]\!]$.

(row-type in row-type)

Assume $E \vdash_\rho R \uparrow L$ and $E,X \uparrow L,E' \vdash_\rho S \uparrow M$.
Then $[\![S\{X \leftarrow R\} \uparrow M]\!]$ is well-defined.
Let $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ and $A_1..A_{\partial L} = [\![R \uparrow L]\!]$
Let $B_1..B_{\partial M} = [\![S \uparrow M]\!]$ and $C_1..C_{\partial M} = [\![S\{X \leftarrow R\} \uparrow M]\!]$
Then $B_i\{X_1 \leftarrow A_1\}..\{X_{\partial L} \leftarrow A_{\partial L}\} \equiv C_i$ for i *in* $1..\partial M$.

(type in value)

Assume $E \vdash_\rho A' <: A$ and $E,X<:A,E' \vdash_\rho b : B$
Then $[\![b\{X \leftarrow A'\}]\!]$ is well-defined.
Then $[\![b]\!]\{X \leftarrow [\![A']\!]\} \equiv [\![b\{X \leftarrow A'\}]\!]$.

(type in row-value)

Assume $E \vdash_\rho A' <: A$ and $E,X<:A,E' \vdash_\rho s \therefore S \uparrow \uparrow \uparrow M$
Then $[\![s\{X \leftarrow A'\} \uparrow M]\!]$ is well-defined.

Let $b_1..b_{\partial M} = [\![s \uparrow M]\!]$ and $c1..c_{\partial M} = [\![s\{X \leftarrow A'\} \uparrow M]\!]$.
Then $b_1\{X \leftarrow [\![A']\!]\} \equiv c_1$ ... $b_{\partial M}\{X \leftarrow [\![A']\!]\} \equiv c_{\partial M}$

(row-type in value)

Assume $E \vdash_\rho R \uparrow L$ and $E,X \uparrow L,E' \vdash_\rho c : C$.
Then $[\![c\{X \leftarrow R\}]\!]$ is well-defined.
Let $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ and $A_1..A_{\partial L} = [\![R \uparrow L]\!]$
Then $[\![c]\!]\{X_1 \leftarrow A_1\}..\{X_{\partial L} \leftarrow A_{\partial L}\} \equiv [\![c\{X \leftarrow R\}]\!]$.

(row-type in row-value)

Assume $E \vdash_\rho R \uparrow L$ and $E,X \uparrow L,E' \vdash_\rho s :: S \uparrow M$.
Then $[\![s\{X \leftarrow R\} \uparrow M]\!]$ is well-defined.
Let $X_1..X_{\partial L} = VarSeq(X, \uparrow L)$ and $A_1..A_{\partial L} = [\![R \uparrow L]\!]$
Let $b_1..b_{\partial M} = [\![s \uparrow M]\!]$ and $c_1..c_{\partial M} = [\![s\{X \leftarrow R\} \uparrow M]\!]$
Then $b_i\{X_1 \leftarrow A_1\}..\{X_{\partial L} \leftarrow A_{\partial L}\} \equiv c_i$ for i *in* $1..\partial M$.

(value in value)

Assume $E \vdash_\rho a : A$ and $E,x{:}A,E' \vdash_\rho b : B$
Then $[\![b\{x \leftarrow a\}]\!]$ is well-defined.
Then $[\![b]\!]\{x \leftarrow [\![a]\!]\} \equiv [\![b\{x \leftarrow a\}]\!]$.

(value in row-value)

Assume $E \vdash_\rho a : A$ and $E,x{:}A,E' \vdash_\rho s :: S \uparrow M$
Then $[\![s\{x \leftarrow a\} \uparrow M]\!]$ is well-defined.
Let $b_1..b_{\partial M} = [\![s \uparrow M]\!]$ and $c_1..c_{\partial M} = [\![s\{x \leftarrow a\} \uparrow M]\!]$
Then $b_1\{x \leftarrow [\![a]\!]\} \equiv c_1$ ... $b_{\partial M}\{x \leftarrow [\![a]\!]\} \equiv c_{\partial M}$.

(row-value in value)

Assume $E \vdash_\rho r :: R \uparrow L$ and $E,x{::}R \uparrow L,E' \vdash_\rho c : C$.
Then $[\![c\{x \leftarrow r\}]\!]$ is well-defined.
Let $x_1..x_{\partial L} = varSeq(x, \uparrow L)$ and $a_1..a_{\partial L} = [\![r \uparrow L]\!]$
Then $[\![c]\!]\{x_1 \leftarrow a_1\}..\{x_{\partial L} \leftarrow a_{\partial L}\} \equiv [\![c\{x \leftarrow r\}]\!]$.

(row-value in row-value)

Assume $E \vdash_\rho r :: R \uparrow L$ and $E,x{::}R \uparrow L,E' \vdash_\rho s :: S \uparrow M$.
Then $[\![s\{x \leftarrow r\} \uparrow M]\!]$ is well-defined.
Let $x_1..x_{\partial L} = varSeq(x, \uparrow L)$ and $a_1..a_{\partial L} = [\![r \uparrow L]\!]$
Let $b_1..b_{\partial M} = [\![s \uparrow M]\!]$ and $c_1..c_{\partial M} = [\![s\{x \leftarrow r\} \uparrow M]\!]$
Then $b_i\{x_1 \leftarrow a_1\}..\{x_{\partial L} \leftarrow a_{\partial L}\} \equiv c_i$ for i *in* $1..\partial M$.

Finally we have the soundness theorem, divided into mutual induction groups.

Theorem (soundness)

(1)	$\vdash_\rho E\ env$	$\Rightarrow$	$[\![\vdash_\rho E\ env]\!]$
	$E \vdash_\rho A\ type$	$\Rightarrow$	$[\![E \vdash_\rho A\ type]\!]$
	$E \vdash_\rho R \uparrow L$	$\Rightarrow$	$[\![E \vdash_\rho R \uparrow L]\!]$
(2)	$E \vdash_\rho A <: B$	$\Rightarrow$	$[\![E \vdash_\rho A <: B]\!]$
	$E \vdash_\rho R <\!\!: S \uparrow L$	$\Rightarrow$	$[\![E \vdash_\rho R <\!\!: S \uparrow L]\!]$
(3)	$E \vdash_\rho a : A$	$\Rightarrow$	$[\![E \vdash_\rho a : A]\!]$
	$E \vdash_\rho r :\!\!: R \uparrow L$	$\Rightarrow$	$[\![E \vdash_\rho r :\!\!: R \uparrow L]\!]$
(4)	$E \vdash_\rho a \leftrightarrow a' : A$	$\Rightarrow$	$[\![E \vdash_\rho a \leftrightarrow a' : A]\!]$
	$E \vdash_\rho r \leftrightarrow r' :\!\!: R \uparrow L$	$\Rightarrow$	$[\![E \vdash_\rho r \leftrightarrow r' :\!\!: R \uparrow L]\!]$

Proof

The proof is by simultaneous induction on the derivations, using the lemmas above in the hard cases. □

8. Conclusions

We have defined a calculus of row variables, $F_{<:}\rho$, and translated it into a simpler calculus with subtyping, $F_{<:}$. The constraints imposed by the translation have forced us into a restricted subset of the features that have been proposed for calculi of extensible records, but we can still express many benchmark examples.

The particular mixture of features chosen for $F_{<:}\rho$ is not uniquely determined. For example we might have attempted to incorporate bounds on row quantifiers ($\forall(X<\!\!:R\uparrow L)B$), row-valued functions ($A \rightarrow \uparrow_L R$), or record concatenation (sketched in section 5.5). The point is that many possible variations can be described and evaluated within a single basic framework. Underlying all these variations and bridging between them there is $F_{<:}$, often extended with recursion. This approach could provide us with a fundamental and unified framework in which to study complex features of object-oriented languages.

Acknowledgements

I would like to thank Martín Abadi for his careful reading of the draft.

References

[Amadio Cardelli 91] R.M.Amadio, L.Cardelli: ***Subtyping recursive types***, Proceedings of the ACM Conference on Principles of Programming Languages, ACM Press, 1991.

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

[Breazu-Tannen Coquand Gunter Scedrov 89] V.Breazu-Tannen, T.Coquand, C.Gunter, A.Scedrov: ***Inheritance and explicit coercion*** , Proc. of the Fourth IEEE Symposium on Logic in Computer Science, pp 112-129, 1989.

[Canning Cook Hill Olthoff Mitchell 89] P.Canning, W.Cook, W.Hill, W.Olthoff, J.C.Mitchell: ***F-bounded polymorphism for object-oriented programming***, Proc. ACM Conference on Functional Programming and Computer Architecture, ACM Press, 1989.

[Cardelli Martini Mitchell Scedrov 91] L.Cardelli, J.C.Mitchell, S.Martini, A.Scedrov: ***An extension of system F with subtyping***, to appear.

[Cardelli Mitchell 91] L.Cardelli, J.C.Mitchell: ***Operations on records***, Proc. of the Fifth Conference on Mathematical Foundations of Programming Language Semantics, New Orleans, 1989. To appear in Mathematical Structures in Computer Science, 1991.

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

[Cardelli 88] L.Cardelli: ***A semantics of multiple inheritance***, in Information and Computation 76, pp 138-164, 1988. (First appeared in Semantics of Data Types, G.Kahn, D.B.MacQueen and G.Plotkin Ed. Lecture Notes in Computer Science n.173, Springer-Verlag 1984.)

[Cook 89] W. Cook: ***A denotational semantics of inheritance***, Ph.D. thesis, Technical Report CS-89-33, Brown University, 1989.

[Curien Ghelli 91] P.-L.Curien, G.Ghelli: ***Coherence of subsumption***, Mathematical Structures in Computer Science, to appear.

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

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

[Girard 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.

[Harper Pierce 90] R.Harper, B.Pierce: ***A record calculus with symmetric concatenation***, Technical Report CMU-CS-90-157, CMU, 1990.

[Jategaonkar Mitchell 88] L.A.Jategaonkar, J.C.Mitchell: ***ML with extended pattern matching and subtypes***, Proc. of the ACM Conference on Lisp and Functional Programming, pp.198-211, 1988.

[Martini 90] S.Martini: personal communication.

[Rémy 89] D. Rémy: ***Typechecking records and variants in a natural extension of ML***, Proc. of the 16th ACM Symposium on Principles of Programming Languages, pp.77-88, 1989.

[Rémy 91] D. Rémy: ***Record concatenation for free***, to appear.

[Reynolds 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.

[Wand 87] M.Wand: ***Complete Type Inference for Simple Objects***, Proc. of the Second IEEE Symposium on Logic in Computer Science, pp 37-44, 1987.

[Wand 89] M.Wand: ***Type inference for record concatenation and multiple inheritance***, Proc. of the Fourth IEEE Symposium on Logic in Computer Science, pp. 92-97, 1989.