Formal Parametric Polymorphism

Martín Abadi — DEC Systems Research Center
Luca Cardelli — DEC Systems Research Center
Pierre-Louis Curien — CNRS Ecole Normale Supérieure

Abstract

A polymorphic function is parametric if its behavior does not depend on the type at which it is instantiated. Starting with Reynolds' work, the study of parametricity is typically semantic. In this paper, we develop a syntactic approach to parametricity, and a formal system that embodies this approach: system $\mathcal{R}$. Girard's system F deals with terms and types; $\mathcal{R}$ is an extension of F that deals also with relations between types.

In $\mathcal{R}$, it is possible to derive theorems about functions from their types, or "theorems for free", as Wadler calls them. An easy "theorem for free" asserts that the type $\forall(X)X\rightarrow$Bool contains only constant functions; this is not provable in F. There are many harder and more substantial examples. Various metatheorems can also be obtained, such as a syntactic version of Reynolds' abstraction theorem.

1. Explicit relations

A polymorphic function is parametric if its behavior does not depend on the type at which it is instantiated [Strachey 1967]. A function that reverses lists, for example, is parametric because it does not look at the types of the elements of the lists given as inputs. There are important non-parametric polymorphic functions, such as a print function that maps values of any type to text representations. With this caveat, it can be argued that "truly" polymorphic functions are parametric, and in any case it is the parametric polymorphic functions that form the core of languages such as ML [Milner, Tofte, Harper 1989].

Reynolds' work provides a precise counterpart to the informal definition of parametricity just given [Reynolds 1983]. Reynolds' abstraction theorem concerns a language similar to Girard's system F [Girard, Lafont, Taylor 1989], and says that the instances of a polymorphic function at different types behave in "related" ways. For example, let f be an expressible function of type $\forall(X)X\rightarrow X$ (the type of the identity function), and let f(A) and f(B) be its instantiations at types A and B, respectively. In this case, the theorem says that, for any relation S between A and B, if $\langle a,b\rangle\in S$ then $\langle f(A)(a),f(B)(b)\rangle\in S$. A bit of calculation reveals that the identity function is the only function with this property, so f must be the identity function. This is what Wadler would call a "theorem for free" [Wadler 1989]: a result about a function that is obtained by examining its type only, and not its code. Reynolds' results about his system suggest that, more generally, one should view a function as parametric if and only if its instances at related types behave in related ways.

In the preceding discussion, functions, types, and relations are all semantic objects. Reynolds' results concern the models of polymorphic languages, such as F, and only indirectly their syntax. Similarly, Wadler's free theorems concern semantic objects in these models, and do not immediately refer to the world of syntax, where they might serve in proving properties of programs.

In this paper we develop a syntactic approach to parametricity. This approach is embodied in an extension of F, called $\mathcal{R}$, where relations between types are constructed and treated formally. In $\mathcal{R}$, the free theorems can be stated and proved in a logical framework and without reference to particular classes of models. Several of these free theorems come from Wadler's work, and we hope that our detailed, formal treatment illuminates their proofs; others seem to be new and intriguing. Various metatheorems about $\mathcal{R}$ can also be obtained, for example a syntactic version of the abstraction theorem. In all cases our results are not limited to closed terms.

The study of $\mathcal{R}$ seems to help in clarifying the notions of parametricity and the properties of parametric models. Semantic explorations steer a difficult course between heavyweight categorical constructions and lightweight fuzzy explanations; in contrast, we use a precise, elementary syntax. With this syntax, it is possible to formulate results and conjectures that relate the intuitive definition of

ACM-20th PoPL-1/93-S.C., USA

parametricity ("types are not needed at run time") with Reynolds' mathematical one.

The remainder of this introduction contains an informal technical introduction and a comparison with a few recent related works. Sections 2 and 3 introduce $\mathcal{R}$, its theory, and then some of the free theorems. In the conclusions we discuss further work. An appendix contains the complete set of rules of the system.

1.1 Parametricity

As an introduction to parametricity and to $\mathcal{R}$, we give an example: we prove that all parametric functions of type $\forall(X)X\to Bool$ are constant. (Here Bool is the type of booleans as encoded in F.) We start with an informal discussion of the functions of this type, then make the reasoning a little more precise, and later introduce the judgments and some of the rules of $\mathcal{R}$, which enable us to formalize the reasoning for this and other free theorems.

At the very least, a function f in $\forall(X)X\to Bool$ maps values of any type to booleans. More precisely:

(i) If A is a type and b has type A,
then f(A) maps b to a boolean.

(Throughout we focus on total functions. The complications in dealing with partial functions are well known.)

The primary examples of functions that satisfy (i) are the constant functions whose instances map any input to either true or false. But, in some models, there are other functions that satisfy (i) and that may be considered as belonging to $\forall(X)X\to Bool$, such as a function zero-p with instances that always map 0 to true and any other input to false. It is hard to code these additional functions in such a way that a typechecker would accept them, and the resulting code requires the use of types at run time. Hence, none of these functions can be considered parametric. Only the constant functions remain.

The sort of discussion of parametric functions that we just went through, to exclude for example zero-p, is vague and not entirely satisfactory; it depends on the use of particular models and on implementation intuitions. Reynolds' more satisfactory approach is based on relations between types. But before we discuss relations in general, it is convenient to introduce the per model [Longo, Moggi 1991], which is based on special relations.

In per semantics, types are interpreted as pers, that is, as partial-equivalence relations (symmetric and transitive relations on the universe of values). Intuitively, b and c are related by the type A if they are equal elements of A, and in particular b is related to itself if it is an element of A. For example, A may be the type of all records with a field n of type Nat, and b and c may be two records that have a field n with the value 3, but differ on other fields; in this case b and c are related by A. We write b[A]c for $\langle b,c\rangle\in A$.

Given two pers A and B, the set of all functions from A to B is also represented as a per:

f[A→B]g iff for all x, y, if x[A]y then f(x)[B]g(y)

That is, two functions are equal in A→B if they map inputs equal in A to results equal in B. Universal quantification is interpreted as intersection, with bound variables ranging over pers.

For example, in the language of pers, the condition for f to be in the type $\forall(X)X\to Bool$ is that f[$\forall(X)X\to Bool$]f. It follows that f(A)[A→Bool]f(A), for all A, and then:

(ii) If b and c are equal as elements of A,
then f(A) maps b and c to the same boolean.

In the per model, the only functions of type $\forall(X)X\to Bool$ are the two obvious constant functions (but this does not follow from (ii) alone). In case A is a record type, for instance, requirement (ii) implies that f(A)(b) cannot depend on fields in b not shown in the definition of A.

Reynolds' work does not assume a per semantics, but his notion of parametricity can be seen as a strengthening of requirement (ii); in this example, it says:

(iii) If $\mathcal{S}$ is a relation between types A and B,
with a in A, b in B, and $\mathcal{S}$ relating a and b,
then f(A)(a) and f(B)(b) are equal booleans.

Requirement (ii) corresponds to the special case $\mathcal{S} = A = B$.

Intuitively, as Reynolds suggests, we may think of A and B as two different representations of the same type, and of a and b as two different representations of the same value; then requirement (iii) means that the function f respects representation abstractions, returning results independently of the representation of its input.

In order to state the general form of (iii), we extend the operations $\to$ and $\forall$. They are defined on arbitrary relations just as they were on pers, except that the variables bound by $\forall$ (now written $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$, $\mathcal{X}$, ...) range over all relations, not just over pers. With this notation, there is a natural relation A* associated with each type expression A: the relation denoted by the type expression, where all quantified variables are interpreted as ranging over arbitrary relations rather than over pers. For example, the relation $(\forall(X)X\to Bool)^*$ is $\forall(\mathcal{W})\mathcal{W}\to Bool$, and $(\forall(X)X\to Y)^*$ is $\forall(\mathcal{W})\mathcal{W}\to Y$.

The general form of (iii) can now be stated:

An element of type A is related to itself by the associated relation A*.

Essentially, Reynolds' abstraction theorem says that all the functions expressible in F satisfy this property. Thus, according to the abstraction theorem, if f is expressible with type $\forall(X)X\to Bool$, then f must be related to itself by $\forall(\mathcal{W})\mathcal{W}\to Bool$. It follows that if A and B are two types and $\mathcal{S}$ a relation between them, then f(A) and f(B) are related in $\mathcal{S}\to Bool$, and so if $\mathcal{S}$ relates a and b it follows that Bool relates f(A)(a) and f(B)(b), as stated in (iii).

With (iii), it is simple to prove that constant functions are the only elements of the type considered: Let f be a function of this type, let A be a type, and let $\mathcal{S}$ be the relation between A and Bool that associates every element of A with true. Then f(A) and f(Bool) are related by $\mathcal{S}\to$Bool, and if a is an element of A then f(A)(a) and f(Bool)(true) are related by Bool, that is, f(A)(a) is equal to the fixed boolean f(Bool)(true), independently of A and a. By extensionality, f is one of the two constant functions. (The use of Bool and true is arbitrary; they can be replaced with any other closed type and closed term of that type.)

1.2 Formal parametricity

The relational approach to parametricity lends itself to a syntactic treatment. System $\mathcal{R}$ provides such a treatment, based on judgments and rules in the style of those of F.

Three judgments generalize those of F:

$\vdash E$	E is a legal environment
$E \vdash \begin{matrix} A \\ \mathcal{R} \\ B \end{matrix}$	$\mathcal{R}$ is relation between A and B in E
$E \vdash \begin{matrix} a : A \\ \mathcal{R} \\ b : B \end{matrix}$	$\mathcal{R}$ relates a of type A and b of type B in E

In addition, an auxiliary judgment concerns relation equalities:

$E \vdash \begin{matrix} A \\ \mathcal{R} = \mathcal{S} \\ B \end{matrix}$	$\mathcal{R}$, $\mathcal{S}$ are equal relations between A and B

An equality judgment at the level of values is not needed. Instead of writing that b and c are equal in A, we can promote the type A to a relation (between A and A; intuitively, the identity relation) and write that A relates b and c. As a consequence we write:

$E \vdash \begin{matrix} b : A \\ A \\ c : A \end{matrix}$	corresponding to the F judgment $E \vdash b = c : A$

The environments of $\mathcal{R}$ extend those of F. They contain two sorts of assumptions from F and two new ones:

X	X is a type variable
x:A	x is a variable of type A
$\begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix}$	$\mathcal{W}$ is a relation variable between type variables X (domain) and Y (codomain)
$\begin{matrix} x : A \\ \mathcal{R} \\ y : B \end{matrix}$	the variables x and y have types A and B, respectively, and are related by $\mathcal{R}$

Using these judgments, we now review some of the central rules of $\mathcal{R}$. We start with rules that imitate those of F for $\to$ and $\forall$.

The introduction and elimination rules for $\to$ are, respectively:

$$\frac{E, \begin{matrix} x : A \\ \mathcal{R} \\ x' : A' \end{matrix} \vdash \begin{matrix} b : B \\ \mathcal{S} \\ b' : B' \end{matrix} \qquad E \vdash \begin{matrix} B \\ \mathcal{S} \\ B' \end{matrix} \qquad \begin{matrix} x \notin b' \\ x' \notin b \end{matrix}}{E \vdash \begin{matrix} \lambda(x:A)b : A \to B \\ \mathcal{R} \to \mathcal{S} \\ \lambda(x':A')b' : A' \to B' \end{matrix}}$$

$$\frac{E \vdash \begin{matrix} b : A \to B \\ \mathcal{R} \to \mathcal{S} \\ b' : A' \to B' \end{matrix} \qquad E \vdash \begin{matrix} a : A \\ \mathcal{R} \\ a' : A' \end{matrix}}{E \vdash \begin{matrix} b(a) : B \\ \mathcal{S} \\ b'(a') : B' \end{matrix}}$$

These rules follow the same pattern as the F rules:

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

$$\frac{E \vdash b : A \to B \qquad E \vdash a : A}{E \vdash b(a) : B}$$

The introduction rule says: Assume that if $\mathcal{R}$ relates x of type A and x' of type A', then $\mathcal{S}$ relates b of type B and b' of type B'. Then $\mathcal{R}\to\mathcal{S}$, a relation between A$\to$B and A'$\to$B', relates the functions λ(x:A)b of type A$\to$B and λ(x':A')b' of type A'$\to$B'. An extra hypothesis that $\mathcal{S}$ relates B and B' is added to simplify our technical lemmas. The elimination rule works in the opposite direction, applying related functions to related arguments and obtaining related results.

The introduction and elimination rules for $\forall$ are:

$$\frac{E, \begin{matrix} X \\ \mathcal{W} \\ X' \end{matrix} \vdash \begin{matrix} b : B \\ \mathcal{S} \\ b' : B' \end{matrix} \qquad \begin{matrix} X \notin b', B', \mathcal{S} \\ X' \notin b, B, \mathcal{S} \end{matrix}}{E \vdash \begin{matrix} \lambda(X)b : \forall(X)B \\ \forall(\mathcal{W})\mathcal{S} \\ \lambda(X')b' : \forall(X')B' \end{matrix}}$$

$$\frac{E \vdash \begin{matrix} b : \forall(X)B \\ \forall(\mathcal{W})\mathcal{S} \\ b' : \forall(X')B' \end{matrix} \qquad E \vdash \begin{matrix} C \\ \mathcal{T} \\ C' \end{matrix}}{E \vdash \begin{matrix} b(C) : B\{X \leftarrow C\} \\ \mathcal{S}\{\mathcal{W} \leftarrow \mathcal{T}\} \\ b'(C') : B'\{X' \leftarrow C'\} \end{matrix}}$$

These rules are generalizations of the F rules:

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

The introduction rule says: Assume that if $\mathcal{W}$ is a relation between types X and X', then $\mathcal{S}$ relates b of type B and b' of type B'. Then $\forall(\mathcal{W})\mathcal{S}$, a relation between $\forall$(X)B and

$\forall(X')B'$, relates the polymorphic terms $\lambda(X)b$ of type $\forall(X)B$ and $\lambda(X')b'$ of type $\forall(X')B'$. Again, the elimination rule works in the opposite direction: it applies two related polymorphic terms to related types, obtaining related instances.

Relations can be formed by quantification over relation variables, as in the rules above, or by quantification over type variables (see the appendix). Two kinds of quantifiers are needed because of the two kinds of environment assumptions for relation variables and type variables. The following rule connects the two quantifiers, and is a formal counterpart to Reynolds' identity extension property discussed in section 2.

$$\frac{\begin{array}{lll} X & B & X \notin B', \mathcal{S}, \mathcal{S}' \\ E, \mathcal{W} \vdash \mathcal{S} = \mathcal{S}' & & X' \notin B, \mathcal{S}, \mathcal{S}' \\ X' & B' & Z \notin \mathrm{dom}(E) \end{array}}{\begin{array}{c} \forall(X)B \\ E \vdash \forall(Z)\mathcal{S}\{\mathcal{W} \leftarrow Z\} = \forall(\mathcal{W})\mathcal{S}' \\ \forall(X)B' \end{array}}$$

This rule is very powerful, as it equates the two quantifiers in arbitrary relation expressions; unfortunately it is also a source of semantic difficulties (see section 4). We have considered several restrictions of this rule, but they lead to syntactic difficulties, and we are not certain they solve the semantic ones. Hence, we prefer to keep the general rule above and to leave the complete semantics of $\mathcal{R}$ for further work.

For our example of section 1.1, this rule yields $\forall(X)X \to Bool = \forall(\mathcal{W})\mathcal{W} \to Bool$, and hence that if f has type $\forall(X)X \to Bool$ then it is related to itself by $\forall(\mathcal{W})\mathcal{W} \to Bool$. From here we can apply the elimination rules for $\forall$ and $\to$, and obtain requirement (iii). This kind of reasoning is common in our examples of section 3.

Until now, the relational constructions have followed closely the ordinary type constructions. In addition we allow relations defined from functions:

$$\frac{E \vdash b : A \to B \qquad E \vdash a : A}{\begin{array}{c} a : A \\ E \vdash \langle b \rangle \\ b(a) : B \end{array}}$$

$$\frac{\begin{array}{ll} a : A & \\ E \vdash \langle b \rangle & E \vdash b : A \to B \\ c : B & \end{array}}{\begin{array}{c} b(a) : B \\ E \vdash B \\ c : B \end{array}}$$

With these rules, terms can be turned into relations: any function b from A to B can be seen as a relation $\langle b \rangle$ between A and B, intuitively the graph of the function. The rules for functional relations have no analogue in F.

Our formalism yields the results typically associated with parametricity only when we include rules for constructing functional relations. Functional relations are often useful for obtaining free theorems; for the example of section 1.1, the relevant functional relation is a constant one, obtained from the function from A to Bool that maps any a in A to true.

One can easily imagine mechanisms for defining relations beyond taking the graphs of functions. We have not yet found examples where these mechanisms are needed.

1.3 Related work

By now there are many papers on semantic aspects of parametricity ([Bainbridge, *et al.* 1990; Hasegawa 1991; Ma, Reynolds 1991; Hasegawa 1992; Mitchell, Scedrov 1992], and others). On the other hand, the syntactic study of parametricity is about one-year old. Some recent work is related to ours.

Mairson advocated and developed a syntactic approach to parametricity in order to provide careful formal versions of some of Wadler's theorems [Mairson 1991]. Mairson's approach consists in translating a polymorphic language into a second-order logic. Because the second-order logic used is fairly weak, induction arguments become necessary in some of the proofs; our proofs, like Wadler's, do not rely on induction. Mairson treated a system with implicit typing; this stands in contrast with our approach where types and relations are treated explicitly. The resulting formalisms have very different properties.

Cardelli et al. have defined $F_{<:}$, an extension of F with subtyping [Cardelli, *et al.* 1991]. Curiously, the rules for $F_{<:}$ capture some aspects of parametricity; see sections 3.4 and 4 below.

Ma suggested another syntactic approach to parametricity [Ma 1992]; it is based on encoding relations using subtyping. The power of Ma's system seems to be less understood; there is also some difficulty in finding a model for all the desired subtyping rules.

Longo, Milstead, and Soloviev investigated parametricity in a system like F with just one new rule (a special case of one of the rules of $F_{<:}$) [Longo, Milstead, Soloviev 1992]. The system is clearly weaker than $\mathcal{R}$, and leads to different sorts of results.

Finally, a forthcoming paper [Plotkin, Abadi, Cardelli 1992] explores an alternative formalization of parametricity closer in spirit to Mairson's. That paper describes a second-order logic with an axiom of parametricity; this logic is not an extension of system F, like $\mathcal{R}$, but rather a logic about system F terms.

2. Formal parametricity

In this section we describe our formalization of parametricity. We aim at a hypothetical system called $\mathcal{R}$ that would be sufficient to prove all the desired parametricity

properties of polymorphic programs. Our current approximations to $\mathcal{R}$ are called $\mathcal{R}^0$ and $\mathcal{R}^1$.

The system $\mathcal{R}^0$ is a rather weak system of pure relations with relational constructions induced by the type constructions of F. A number of technical lemmas can be proved for $\mathcal{R}^0$, and these lead to several interesting metatheorems. For example, a suitable encoding of F in $\mathcal{R}^0$ yields all F typings and F equalities. In addition, $\mathcal{R}^0$ is a conservative extension of F for typing and equality derivations. The abstraction theorem for F holds in $\mathcal{R}^0$ but it is not very useful (as the conservativity result indicates) without some additional means for constructing relations. Hence, we extend $\mathcal{R}^0$ with functional relations, obtaining $\mathcal{R}^1$. Relation expressions become dependent on value expressions, and the syntactic properties of the system become considerably more complex. Fortunately, most $\mathcal{R}^0$ metatheorems extend easily to $\mathcal{R}^1$, simply because $\mathcal{R}^0$ derivations are also $\mathcal{R}^1$ derivations. As a typing system, $\mathcal{R}^1$ is still conservative over F, but new equations are provable.

The appendix contains the full set of rules of $\mathcal{R}^1$, and it indicates which rules must be subtracted to obtain $\mathcal{R}^0$. In this section, results are stated for $\mathcal{R}^1$ unless otherwise indicated.

Preliminaries

- We use $\vdash^F$ for derivations in F, and $\vdash^{\mathcal{R}}$ (or simply $\vdash$) for derivations in either $\mathcal{R}^0$ or $\mathcal{R}^1$, as appropriate.

- We use the abbreviations:

$$E \vdash^{\mathcal{R}} A \triangleq E \vdash^{\mathcal{R}} \begin{matrix} A \\ A \\ A \end{matrix}$$

$$E \vdash^{\mathcal{R}} a : A \triangleq E \vdash^{\mathcal{R}} \begin{matrix} a : A \\ A \\ a : A \end{matrix}$$

- We often use the derived rule:

(Rel Val Appl2 X$\mathcal{R}$)

$$\frac{E \vdash \begin{matrix} b : \forall(X)B \\ \forall(X)S \\ b' : \forall(X')B' \end{matrix} \qquad E \vdash^{\mathcal{R}} \begin{matrix} A \\ \mathcal{R} \\ A' \end{matrix}}{E \vdash \begin{matrix} b(A) : B\{X \leftarrow A\} \\ S\{X \leftarrow \mathcal{R}\} \\ b'(A') : B'\{X \leftarrow A'\} \end{matrix}}$$

Our first result, the conservativity over F for typing, requires a definition for flattening an $\mathcal{R}$environment E into an F environment $(E)_F$. The relation part of E is forgotten in $(E)_F$:

Definition (Environment flattening)

- $(\emptyset)_F = \emptyset$
- $(E,X)_F = (E)_F, X$
- $(E,x:A)_F = (E)_F, x:A$
- $\begin{pmatrix} X \\ E,\mathcal{W} \\ Y \end{pmatrix}_F = (E)_F, X, Y$
- $\begin{pmatrix} x : A \\ E, \mathcal{R} \\ y : B \end{pmatrix}_F = (E)_F, x:A, y:B$

Theorem (Conservativity over F for typings)

- $\vdash E \Rightarrow \vdash^F (E)_F$
- $E \vdash \begin{matrix} A \\ \mathcal{R} = S \\ B \end{matrix} \Rightarrow (E)_F \vdash^F A \wedge (E)_F \vdash^F B$
- $E \vdash \begin{matrix} A \\ \mathcal{R} \\ B \end{matrix} \Rightarrow (E)_F \vdash^F A \wedge (E)_F \vdash^F B$
- $E \vdash \begin{matrix} a : A \\ \mathcal{R} \\ b : B \end{matrix} \Rightarrow (E)_F \vdash^F a : A \wedge (E)_F \vdash^F b : B$

A simple encoding of F in $\mathcal{R}$is implicit in the following soundness theorem. Using the abbreviations introduced above, we have:

Theorem (Soundness of F in $\mathcal{R}$)

- $\vdash^F E \Rightarrow \vdash^{\mathcal{R}} E$
- $E \vdash^F a : A \Rightarrow E \vdash^{\mathcal{R}} a : A$
- $E \vdash^F A \Rightarrow E \vdash^{\mathcal{R}} A$
- $E \vdash^F a = b : A \Rightarrow E \vdash^{\mathcal{R}} \begin{matrix} a : A \\ A \\ b : A \end{matrix}$

There is also a more interesting way of embedding F in $\mathcal{R}$, by splitting an F judgment into two related F judgments. This relational interpretation of F turns out to be the fundamental theorem about $\mathcal{R}^0$, and it yields the abstraction theorem as a corollary. The following definitions are needed:

Definition

- The translation $[-]_s$ decorates each variable occurring in an F or $\mathcal{R}$ term or judgment with the subscript s. Subscripts can be numbers or other symbols. (We assume that this translation does not introduce variable clashes.) For example:

$$[E, X, x : \forall(Y)X \to Y \vdash^F X]_1 = [E]_1, X_1, x_1 : \forall(Y_1)X_1 \to Y_1 \vdash^F X_1$$

- The translation $[A]_{\mathcal{R}}$ transforms an F-type into an $\mathcal{R}$-relation. In particular, it transforms type quantifiers into relation quantifiers, and free type variables into free relation variables.
  - $[X]_{\mathcal{R}} \triangleq \mathcal{X}$
  - $[A \to B]_{\mathcal{R}} \triangleq [A]_{\mathcal{R}} \to [B]_{\mathcal{R}}$
  - $[\forall(X)B]_{\mathcal{R}} \triangleq \forall(\mathcal{X})[B]_{\mathcal{R}}$

• The translation $[-]^{\frac{1}{2}}_{\mathcal{R}}$ transforms an F environment into an $\mathcal{R}$environment.

- $[\emptyset]^{\frac{1}{2}}_{\mathcal{R}} \triangleq \emptyset$
- $[E, X]^{\frac{1}{2}}_{\mathcal{R}} \triangleq [E]^{\frac{1}{2}}_{\mathcal{R}}, \begin{matrix} X_1 \\ \mathcal{X} \\ X_2 \end{matrix}$
- $[E, x : A]^{\frac{1}{2}}_{\mathcal{R}} \triangleq [E]^{\frac{1}{2}}_{\mathcal{R}}, \begin{matrix} x_1 : [A]_1 \\ [A]_{\mathcal{R}} \\ x_2 : [A]_2 \end{matrix}$

Theorem (Relational interpretation of F in $\mathcal{R}$)

- $\vdash^F E \Rightarrow \vdash^{\mathcal{R}} [E]^{\frac{1}{2}}_{\mathcal{R}}$
- $E \vdash^F a : A \Rightarrow [E]^{\frac{1}{2}}_{\mathcal{R}} \vdash^{\mathcal{R}} \begin{matrix} [a]_1 : [A]_1 \\ [A]_{\mathcal{R}} \\ [a]_2 : [A]_2 \end{matrix}$
- $E \vdash^F A \Rightarrow [E]^{\frac{1}{2}}_{\mathcal{R}} \vdash^{\mathcal{R}} \begin{matrix} [A]_1 \\ [A]_{\mathcal{R}} \\ [A]_2 \end{matrix}$
- $E \vdash^F a = b : A$

$$\Rightarrow [E]^{\frac{1}{2}}_{\mathcal{R}} \vdash^{\mathcal{R}} \begin{matrix} [a]_1 : [A]_1 \\ [A]_{\mathcal{R}} \\ [b]_2 : [A]_2 \end{matrix} \wedge [E]^{\frac{1}{2}}_{\mathcal{R}} \vdash^{\mathcal{R}} \begin{matrix} [b]_1 : [A]_1 \\ [A]_{\mathcal{R}} \\ [a]_2 : [A]_2 \end{matrix}$$

Corollary (Reynolds' abstraction theorem)
Assume $E \vdash^F a : A$ and let $\vec{X} = X_1 ... X_n$ be the collection of all type variables in dom(E).

Assume $E_0 \vdash \begin{matrix} B_1 \\ \mathcal{R}_1 \\ C_1 \end{matrix} \ldots E_0 \vdash \begin{matrix} B_n \\ \mathcal{R}_n \\ C_n \end{matrix}$, $\mathrm{dom}(E_0) \cap \mathrm{dom}(E) = \emptyset$.

Let $\hat{\emptyset} \triangleq E_0$; $(F,X)\hat{} \triangleq \hat{F}$; $(F,x:D)\hat{} \triangleq \hat{F}, \begin{matrix} x_1 : D\{\vec{X} \leftarrow \vec{B}\} \\ \vec{\mathcal{R}} \\ x_2 : D\{\vec{X} \leftarrow \vec{C}\} \end{matrix}$.

Then: $\hat{E} \vdash \begin{matrix} [a:A]_1\{[\vec{X}]_1 \leftarrow \vec{B}\} \\ [A]_{\mathcal{R}}\{[\vec{X}]_{\mathcal{R}} \leftarrow \vec{\mathcal{R}}\} \\ [a:A]_2\{[\vec{X}]_2 \leftarrow \vec{C}\} \end{matrix}$.

The interpretations of F in $\mathcal{R}$and the abstraction theorem do not depend on the rule (Rel Eq Forall X$\mathcal{W}$). Using (Rel Eq Forall X$\mathcal{W}$) we obtain a further result: a syntactic version of Reynolds' identity extension lemma. The relation A* was discussed in the introduction.

Definition

- $X^* = X$
- $(A \to B)^* = A^* \to B^*$
- $(\forall(X)B)^* = \forall(\mathcal{X})(B^*\{X \leftarrow \mathcal{X}\})$

Theorem (Identity extension lemma)

$$E \vdash^F A \Rightarrow E \vdash^{\mathcal{R}} \begin{matrix} A \\ A = A^* \\ A \end{matrix}$$

A weak form of this result, for closed terms, is provable without using the rule (Rel Eq Forall X$\mathcal{W}$):

$$\vdash^{\mathcal{R}} \begin{matrix} a : A \\ A \\ b : A \end{matrix} \Rightarrow \vdash^{\mathcal{R}} \begin{matrix} a : A \\ A^* \\ b : A \end{matrix}$$

This explains why, in reasoning about closed terms, it is generally possible to avoid uses of (Rel Eq Forall X$\mathcal{W}$).

Finally, we show that $\mathcal{R}^0$ equality is no stronger than F equality.

Definition
Let $\{E\downarrow \leftarrow E\uparrow\}$ be the substitution that:

for each $\begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix}$ in E replaces Y with X,

for each $\begin{matrix} x : A \\ \mathcal{R} \\ y : B \end{matrix}$ in E replaces y with x.

(The X and x:A components of E are ignored.)

Let $\{E\uparrow \leftarrow E\downarrow\}$ be defined symmetrically.

Theorem (Conservativity of $\mathcal{R}^0$ over F for equalities)

$$E \vdash^{\mathcal{R}^0} \begin{matrix} a : A \\ A \\ b : A \end{matrix} \Rightarrow (E)_F \vdash^F (a = b : A)\{E\downarrow \leftarrow E\uparrow\} \wedge (E)_F \vdash^F (a = b : A)\{E\uparrow \leftarrow E\downarrow\}$$

This result does not extend to $\mathcal{R}^1$, as the theorems for free of the next section show.

3. Theorems for free, syntactically

In this section we illustrate the power of $\mathcal{R}^1$ by carrying out formal proofs. The results given below apply to all terms, and not just to closed terms. In some cases, even the results for closed terms are somewhat difficult; Wadler's work includes a few interesting semantic results that can be read as results about closed terms. In order to deal with open terms we do not use structural induction (like Mairson), but rather the rule (Rel Eq Forall X$\mathcal{W}$) and the identity extension lemma.

3.1 $\forall(X)X \to X$ contains only the identity

As a first example we show that all terms of type $\forall(X)X \to X$ equal the polymorphic identity function id = $\lambda(X)\lambda(x:X)x$, and hence that this type is terminal. For closed terms this result follows easily from strong normalization, but a strong-normalization argument does not extend to open terms.

Proposition

$$E \vdash f : \forall(X)X \to X \quad \Rightarrow \quad E \vdash \begin{matrix} f : \forall(X)X \to X \\ \forall(X)X \to X \\ id : \forall(X)X \to X \end{matrix}$$

Proof
By (Rel Val Fun A), (Rel Val Fun2 X), (Rel Val Eta), and (Rel Val Eta2), it suffices to prove:

$$E, X, x : X \vdash \begin{matrix} f(X)(x) : X \\ X \\ x : X \end{matrix}$$

Using (Rel FRel) we first obtain:

$$E, X, x : X \vdash < \lambda(g : \forall(Y)Y \to Y)x > \begin{matrix} \forall(Y)Y \to Y \\ X \end{matrix}$$

Hence we derive, using the identity extension lemma and (Rel Val Appl2 $\mathcal{W}$):

$$E, X, x : X \vdash \begin{matrix} f(\forall(Y)Y \to Y) : (\forall(Y)Y \to Y) \to (\forall(Y)Y \to Y) \\ < \lambda(g : \forall(Y)Y \to Y)x > \to < \lambda(g : \forall(Y)Y \to Y)x > \\ f(X) : X \to X \end{matrix}$$

By (Rel Val FRel Intro), we have:

$$E, X, x : X \vdash < \lambda(g : \forall(Y)Y \to Y)x > \begin{matrix} f : \forall(Y)Y \to Y \\ x : X \end{matrix}$$

and by (Rel Val Appl):

$$E, X, x : X \vdash \begin{matrix} f(\forall(Y)Y \to Y)(f) : (\forall(Y)Y \to Y) \\ < \lambda(g : \forall(Y)Y \to Y)x > \\ f(X)(x) : X \end{matrix}$$

Finally by (Rel Val FRel Elim) we obtain:

$$E, X, x : X \vdash \begin{matrix} x : X \\ X \\ f(X)(x) : X \end{matrix}$$

and the claim follows by (Rel Val Symm).
□

3.2 Properties of map

The naturality conditions that arise from parametricity have mundane applications, for example in proving properties about the type of the map function:

$$\forall(X)\ \forall(Y)\ (X \to Y) \to (List\{X\} \to List\{Y\})$$

where List{B} denotes the type of B-lists, encoded as usual as $\forall(X)X \to (B \to X \to X) \to X$.

Here we include only one theorem about this type; Wadler and Mairson have treated this theorem too. Wadler's map theorem asserts that any term m of this type is the composition (in either order) of map and a rearrangement function, like reverse. The rearrangement function is retrieved from m by instantiating X and Y to a same type, say X, and then by applying m(X)(X) to the identity on X; the resulting term has type List{X}→List{X}. The usual composition operation $\lambda(x)g(f(x))$ is written (f;g).

Proposition
Let E = X, Y, m:∀(X)∀(Y)(X→Y)→(List{X}→List{Y}), f:X→Y. Then:

$$E \vdash \begin{matrix} m(X)(Y)(f) : List\{X\} \to List\{Y\} \\ List\{X\} \to List\{Y\} \\ m(X)(X)(id(X)) ; map(X)(Y)(f) : List\{X\} \to List\{Y\} \end{matrix}$$

$$E \vdash \begin{matrix} m(X)(Y)(f) : List\{X\} \to List\{Y\} \\ List\{X\} \to List\{Y\} \\ map(X)(Y)(f) ; m(Y)(Y)(id(Y)) : List\{X\} \to List\{Y\} \end{matrix}$$

For the (omitted) proof of this proposition, as well as for many others, we need a commutation lemma described next.

3.3 A commutation lemma

We say that a type A is covariant in X when X occurs only positively in A. For example, $(X \to Y) \to X$ and List{X} are covariant in X. A type A depending on X (the other free variables being considered as fixed parameters) may be viewed as a map $B \mapsto A\{X \leftarrow B\}$ from types to types. When A is covariant in X, it determines a (covariant) functor, which associates with any h:B→B' a term A{X←h} of type A{X←B}→A{X←B'} as follows:

If $E \vdash a : A' \to A$, $E \vdash b : B \to B'$, then
$a \to b \triangleq$

$$\lambda(x : A \to B)\lambda(y : A')b(x(a(y))) : (A \to B) \to (A' \to B')$$

If $E, Y \vdash a : A \to A'$, then
$\forall(Y)a \triangleq$

$$\lambda(x : \forall(Y)A)\ \lambda(Y)\ a(x(Y)) : (\forall(Y)A) \to (\forall(Y)A')$$

Then A{X←h} is defined inductively:

- $X\{X \leftarrow h\} \triangleq h$
- $(A \to B)\{X \leftarrow h\} \triangleq A\{X \leftarrow h\} \to B\{X \leftarrow h\}$
- $Y\{X \leftarrow h\} \triangleq id(Y)$ for $Y \neq X$
- $(\forall(Y)A)\{X \leftarrow h\} \triangleq \forall(Y)A\{X \leftarrow h\}$ for $Y \neq X$

For example, for each h:B→B', map(B)(B')(h) is simply List{Y←h}.

Typically, in our proofs, we get relations of the form A{X←<h>} from an application of the identity extension lemma and (Rel Val Appl2 $\mathcal{W}$), while <A{X←h}> may be needed. The following lemma says that covariant functors commute with functional relations, so A{X←<h>} can be transformed into <A{X←h}>.

Lemma
Assume $E,X \vdash A$, where A is covariant in X, and $E \vdash h : B \to B'$. Then:

$$E \vdash A\{X \leftarrow <h>\} = <\begin{matrix} A\{X \leftarrow B\} \\ A\{X \leftarrow B'\} \end{matrix}>$$

The proof of this lemma relies on the rule (Rel Eq Extension) given in the appendix; this rule says that any two relations with the same graph are equal. Without this rule, an "extensional" version of the lemma could still be obtained, and it would suffice for our purposes.

3.4 Type isomorphisms and initial algebras

Given a type A covariant in X, an A-algebra is a pair of a type B and a morphism $t:A\{X \leftarrow B\} \to B$. An A-algebra morphism from (B,t) to (B',t') is a term $h:B \to B'$ such that $t;h = A\{X \leftarrow h\};t'$. An initial A-algebra is an A-algebra (T,*in*) such that for any other A-algebra (B,t) there exists exactly one A-algebra morphism from (T,*in*) to (B,t). The goal of this subsection is to show that, given A covariant in X, the type $T = \forall(X)(A \to X) \to X$ can be turned into an initial A-algebra. This means that the initial algebras useful in programming (for example, that of natural numbers) can be defined properly as polymorphic types; see [Böhm, Berarducci 1985] for background and [Wadler 1991] for a semantic proof. We define:

$$\begin{aligned} &fold: \forall(X)(A \to X) \to (T \to X) = \\ &\quad \lambda(X)\, \lambda(k : A \to X)\, \lambda(x : T)\, x(X)(k) \end{aligned}$$

$$\begin{aligned} &in : A\{X \leftarrow T\} \to T = \\ &\quad \lambda(y : A\{X \leftarrow T\})\, \lambda(X)\, \lambda(k : A \to X) \\ &\qquad k(A\{X \leftarrow fold(X)(k)\}(y)) \end{aligned}$$

$$\begin{aligned} &out : T \to A\{X \leftarrow T\} = \\ &\quad fold(A\{X \leftarrow T\})(A\{X \leftarrow in\}) \end{aligned}$$

Here *fold*(X)(k) takes an algebra (X,k) to an algebra morphism $\lambda(x:T)x(X)(k)$ from (T,*in*) to (X,k). Initiality of (T,*in*) means that if a is a morphism from (T,*in*) to (X,k), then a must equal *fold*(X)(k):

Theorem
(T, *in*) is initial. That is:

$$E,X,\, k : A \to X \vdash \begin{matrix} in : A\{X \leftarrow T\} \to T \\ <A\{X \leftarrow a\}> \to <a> \\ k : A \to X \end{matrix}$$

$$\Rightarrow \quad E,X,\, k : A \to X \vdash \begin{matrix} a : T \to X \\ T \to X \\ fold(X)(k) : T \to X \end{matrix}$$

A consequence of initiality is that *in* is actually an isomorphism from $A\{X \leftarrow T\}$ to T. Hence, the initial A-algebra is a solution for the fixpoint equation $X = A\{X\}$; and *in* and *out* are the two halves of the isomorphism between T and $A\{X \leftarrow T\}$. Polymorphic types thus suffice to encode covariant recursive types. In particular, if X does not occur in A, then A and $\forall(X)(A \to X) \to X$ are isomorphic.

Other universality properties are expressible and provable in $\mathcal{R}$, for example:

- $\forall(X)X \to X$ is a terminal object 1 (as already proved).
- $\forall(X)(B \to B' \to X) \to X$ is a product B×B' of B and B'.
- $\forall(X)X$ is an initial object 0.
- $\forall(X)(B \to X) \to (B' \to X) \to X$ is a coproduct B+B' of B and B'.
- The type Bool = $\forall(X)X \to X \to X$ is isomorphic to 1+1.
- The type Nat = $\forall(X)X \to (X \to X) \to X$ of Church integers is the initial A-algebra for A=1+X. Hence Nat and 1+Nat are provably isomorphic in $\mathcal{R}$.
- The type List{B} is the initial A-algebra for A=1+(B×X). Hence List{B} and 1+(B×List{B}) are provably isomorphic.

Weaker forms of these results, for a category of closed terms, were already available in $F_{<:}$.

We put the universality properties to use in the proof of a property of Bool. In F the two only closed normal forms of type Bool are:

$$\begin{aligned} &true = \lambda(Z)\lambda(x : Z)\lambda(y : Z)x \\ &false = \lambda(Z)\lambda(x : Z)\lambda(y : Z)y \end{aligned}$$

We prove that any two functions from Bool that coincide on true and false are equal. For example the terms $\lambda(x:Bool)\ 3$ and $\lambda(x:Bool)$ if x then 3 else 3 are provably equal.

Proposition
Let $E \vdash A$, $E \vdash b : Bool \to A$, and $E \vdash b' : Bool \to A$. Then:

$$E \vdash \begin{matrix} b(true) : A \\ A \\ b'(true) : A \end{matrix}, \quad E \vdash \begin{matrix} b(false) : A \\ A \\ b'(false) : A \end{matrix}$$

$$\Rightarrow \quad E \vdash \begin{matrix} b : Bool \to A \\ Bool \to A \\ b' : Bool \to A \end{matrix}$$

Proof
We exploit the following isomorphisms: Bool is isomorphic to 1+1, (C+C')→A is isomorphic to (C→A)×(C'→A) for any C and C', and 1→A is isomorphic to A. Hence Bool→A is isomorphic to A×A. The two halves of the isomorphism are:

$$\begin{aligned} i &= \lambda(f : Bool \to A)\lambda(Y)\lambda(g : A \to A \to Y) \\ &\quad g(f(true))(f(false)) \\ j &= \lambda(h : \forall(Y)(A \to A \to Y) \to Y)\lambda(x : Bool) \\ &\quad h(A)(x(A)) \end{aligned}$$

The conclusion follows by transitivity and replacement of equals for equals:

- it is enough to prove the equality of i(b) and i(b') (since b is equal to j(i(b)), and similarly for b');
- i(b) and i(b') are equal since the argument f occurs only in the contexts f(true) and f(false) in i.

□

3.5 On erasures

We end the section with a collection of examples of a somewhat different flavor. They are all examples of a general "erasure conjecture". Roughly, the conjecture states that two F terms having the same type in the same environment and having the same erasure are provably equal in $\mathcal{R}$.

The erasure of an F term is the untyped term obtained by erasing all its type information. Formally:

erase(x) = x
erase(a(b)) = erase(a)(erase(b))
erase(λ(x:A)a) = λ(x) erase(a)
erase(a(A)) = erase(a)
erase(λ(X)a) = erase(a)

The precise formulation of the conjecture is:

Conjecture
If $E \vdash^F a : A$, $E \vdash^F b : A$, and erase(a) = erase(b), then:

$$E \vdash \begin{matrix} a : A \\ A \\ b : A \end{matrix}$$

If the conjecture holds, it gives precise evidence that Reynolds' notion of parametricity, which our formal system captures in syntax, reflects the intuition that types do not matter in computations of polymorphic programs.

Here we neither prove nor disprove the conjecture, but simply verify some instances:

Instance 1
Let $E \vdash^F a : \forall(X)A$, where $X \notin A$, and let $E \vdash^F B$ and $E \vdash^F C$. Then:

$$E \vdash \begin{matrix} a(B) : A \\ A \\ a(C) : A \end{matrix}$$

Proof
We show how to prove:

$$E \vdash \begin{matrix} a(\forall(X)X) : A \\ A \\ a(B) : A \end{matrix} \qquad E \vdash \begin{matrix} a(\forall(X)X) : A \\ A \\ a(C) : A \end{matrix}$$

The desired result follows from (Rel Val Symm) and (Rel Val Saturation Lft). We derive the first judgment ; the other derivation is similar. From the hypotheses and the soundness of F in R, we have $E \vdash^{\mathcal{R}} a : \forall(X)A$. Moreover, (Rel FRel) yields:

$$E \vdash \begin{matrix} \forall(X)X \\ \langle \lambda(x : \forall(X)X)x(B) \rangle \\ B \end{matrix}$$

We conclude by using (Rel Val Appl2 X$\mathcal{R}$) (see section 2).
□

This first instance is the $\mathcal{R}$ analogue of Axiom C, the rule that Longo, Milstead, and Soloviev add to system F.

The second instance concerns the terminal type:

Instance 2

$$x : \forall(X)X \vdash \begin{matrix} x(\forall(X)X) : \forall(X)X \\ \forall(X)X \\ x : \forall(X)X \end{matrix}$$

Proof
We start by constructing a functional relation:

$$X \vdash < \lambda(x : \forall(Y)Y)x(X) > \begin{matrix} \forall(Y)Y \\ X \end{matrix}$$

By applying (Rel Val Appl2 X$\mathcal{R}$) we get:

$$x : \forall(Y)Y,\ X \vdash < \lambda(x : \forall(Y)Y)x(X) > \begin{matrix} x(\forall(Y)Y) : \forall(Y)Y \\ x(X) : X \end{matrix}$$

and (Rel Val FRel Elim) leads to:

$$x : \forall(Y)Y,\ X \vdash \begin{matrix} x(\forall(Y)Y)(X) : X \\ X \\ x(X) : X \end{matrix}$$

The result follows by (Rel Val Eta2).
□

A simple variant of this proof yields:

Instance 3
Assume $E \vdash a : A$ and $X \notin A$.

$$E,\ x : \forall(X)A \to X \vdash \begin{matrix} x(\forall(X)X)(a) : \forall(X)X \\ \forall(X)X \\ \lambda(X)\, x(X)(a) : \forall(X)X \end{matrix}$$

The final instance is based on two different ways of assigning the type $(\forall(X)X \to X) \to (\forall(X)X \to X)$ to the untyped term $\lambda(x)\, x(x)$:

Instance 4

$$x : \forall(X)X \to X \vdash \begin{matrix} x(\forall(X)X \to X)(x) : \forall(X)X \to X \\ \forall(X)X \to X \\ \lambda(X)\, x(X \to X)(x(X)) : \forall(X)X \to X \end{matrix}$$

Proof
We start by constructing a functional relation:

$$x: \forall(X)X \to X,\ Y \vdash \begin{array}{c} \forall(X)X \to X \\ < \lambda(x: \forall(X)X \to X)x(Y) > \\ Y \to Y \end{array}$$

By applying (Rel Val Appl2 X$\mathcal{R}$) we get:

$$x: \forall(X)X \to X,\ Y \vdash$$

$$\begin{array}{c} x(\forall(X)X \to X): (\forall(X)X \to X) \to (\forall(X)X \to X) \\ < \lambda(x: \forall(X)X \to X)x(Y) > \to < \lambda(x: \forall(X)X \to X)x(Y) > \\ x(Y \to Y): (Y \to Y) \to (Y \to Y) \end{array}$$

By (Rel Val FRel Intro) we have:

$$x: \forall(X)X \to X,\ Y \vdash \begin{array}{c} x: \forall(X)X \to X \\ < \lambda(x: \forall(X)X \to X)x(Y) > \\ x(Y): Y \to Y \end{array}$$

Hence, by (Rel Val Appl) we obtain:

$$x: \forall(X)X \to X,\ Y \vdash \begin{array}{c} x(\forall(X)X \to X)(x): (\forall(X)X \to X) \\ < \lambda(x: \forall(X)X \to X)x(Y) > \\ x(Y \to Y)(x(Y)): (Y \to Y) \end{array}$$

and after (Rel Val FRel Elim):

$$x: \forall(X)X \to X,\ Y \vdash \begin{array}{c} x(\forall(X)X \to X)(x)(Y): (Y \to Y) \\ Y \to Y \\ x(Y \to Y)(x(Y)): (Y \to Y) \end{array}$$

The result follows using (Rel Val Eta2).
□

Of course $\mathcal{R}$ yields far more equations than the ones arising from the conjecture. For example, we can show that $\forall(X)X \to$ Bool contains only constant functions, and hence that f(A)(a) and f(B)(b) are equal for any f: $\forall(X)X \to$ Bool. Here a and b can be any terms, of types A and B, respectively. In particular the terms f(A)(a) and f(B)(b) need not have the same erasure.

4. Conclusions

After working with $\mathcal{R}$ for some time, we feel that it is a useful system, with good syntactic properties. In particular we are able to prove theorems and metatheorems in full generality for open terms. However, the power of $\mathcal{R}$, both in syntactic and in semantic terms, seems to deserve further exploration.

In the realm of syntax, we are particularly interested in the conjecture discussed in section 3.5 that if two F terms have the same erasure and the same type, then they are provably equal in $\mathcal{R}$.

As for semantics, we have constructed a model for our rules except (Rel Eq Forall X$\mathcal{W}$). The model is an extension of a model of F based on pers. The works of Hasegawa [Hasegawa 1991] and Hyland, Robinson, and Rosolini [Hyland, Robinson, Rosolini 1990] suggest that the usual per model, or closely related ones, may validate (Rel Eq Forall X$\mathcal{W}$).

We have investigated other relational constructions, such as relational inverse and composition. These create further syntactic and semantic complications. In particular the natural generalization of (Rel Eq Forall X$\mathcal{W}$) seems unsound, since for example $\forall(X)\,(X \circ X) \to X$ contains the polymorphic identity function while $\forall(\mathcal{W})(\mathcal{W} \circ \mathcal{W}) \to \mathcal{W}$ is empty.

As mentioned in the introduction, system $F_{<:}$ [Cardelli, *et al.* 1991] captures some aspects of parametricity. An extension of $\mathcal{R}$ with subtyping may yield an encoding of $F_{<:}$ and provide a basis for studying parametricity in languages with subtyping. An analogous extension of a logic for parametric polymorphism is carried out in [Plotkin, Abadi, Cardelli 1992].

Acknowledgments

We would like to thank Ryu Hasegawa and Phil Wadler for helpful discussions.

References

[Bainbridge, *et al.* 1990] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott, **Functorial polymorphism**. Theoretical Computer Science **70**, 35-64.

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

[Cardelli, *et al.* 1991] L. Cardelli, J.C. Mitchell, S. Martini, and A. Scedrov. **An extension of system F with subtyping**. *Proc. Theoretical Aspects of Computer Software*. Sendai, Japan. Lecture Notes in Computer Science 526. Springer-Verlag.

[Girard, Lafont, Taylor 1989] J.-Y. Girard, Y. Lafont, and P. Taylor, **Proofs and types**. Cambridge University Press.

[Hasegawa 1991] R. Hasegawa. **Parametricity of extensionally collapsed term models of polymorphism and their categorical properties.** *Proc. Theoretical Aspectes of Computer Software*. Lecture Notes in Computer Science 526. Springer-Verlag.

[Hasegawa 1992] R. Hasegawa, **Categorical data types in parametric polymorphism**. Manuscript.

[Hyland, Robinson, Rosolini 1990] J.M.E. Hyland, E.P. Robinson, and G. Rosolini. **Algebraic types in PER models.** *Proc. Mathematical Foundations of Programming Semantics*. Lecture Notes in Computer Science 442. Springer-Verlag.

[Longo, Milstead, Soloviev 1992] G. Longo, C. Milstead, and S. Soloviev, **A syntactic understanding of parametricity in polymorphic languages.** Manuscript.

[Longo, Moggi 1991] G. Longo and E. Moggi, **Constructive natural deduction and its 'ω-set' interpretation.** Mathematical Structures in Computer Science 1(2).

[Ma 1992] Q.M. Ma. **Parametricity as subtyping.** *Proc. 19th Annual ACM Symposium on Principles of Programming Languages.*

[Ma, Reynolds 1991] Q.M. Ma and J. Reynolds. **Types, abstraction, and parametric polymorphism, part 2.** *Proc. Mathematical Foundations of Programming Semantics.* Springer-Verlag.

[Mairson 1991] H. Mairson. **Outline of a proof theory of parametricity.** *Proc. 5th International Symposium on Functional Programming Languages and Computer Architecture.* Springer-Verlag.

[Milner, Tofte, Harper 1989] R. Milner, M. Tofte, and R. Harper, **The definition of Standard ML.** MIT Press.

[Mitchell, Scedrov 1992] J.C. Mitchell and A. Scedrov, **Sconing, relators, and parametricity.** Manuscript.

[Plotkin, Abadi, Cardelli 1992] G.D. Plotkin, M. Abadi, and L. Cardelli, **A logic for parametric polymorphism.** Manuscript.

[Reynolds 1983] J.C. Reynolds, **Types, abstraction, and parametric polymorphism,** in *Information Processing,* R.E.A. Mason, Editor. North Holland: p. 513-523.

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

[Wadler 1989] P. Wadler. **Theorems for free!** *Proc. 4th International Symposium on Functional Programming Languages and Computer Architecture.* Springer-Verlag.

[Wadler 1991] P. Wadler, **Recursive types for free!** Manuscript.

Appendix: System $\mathcal{R}^1$

Notation

- We use the following metavariables: x,y,z range over value variables; X,Y,Z range over type variables; $\mathcal{W}$ ranges over relation variables; a,b,c,d range over value terms; A,B,C,D range over type terms; $\mathcal{R},\mathcal{S},\mathcal{T},\mathcal{U}$ range over relation terms; E ranges over environments.
- We use the abbreviations: $E \vdash \begin{matrix} A \\ A \\ A \end{matrix} \triangleq E \vdash \begin{matrix} A \\ A \\ A \end{matrix}$ $\qquad E \vdash a : A \triangleq E \vdash \begin{matrix} a : A \\ A \\ a : A \end{matrix}$

Environments

(Env $\emptyset$)

$$\frac{}{\vdash \emptyset}$$

(Env X)

$$\frac{\vdash E \qquad X \notin \mathrm{dom}(E)}{\vdash E,\ X}$$

(Env x)

$$\frac{E \vdash A \qquad x \notin \mathrm{dom}(E)}{\vdash E,\ x : A}$$

(Env X$\mathcal{W}$Y)

$$\frac{\vdash E \qquad \begin{matrix} X,\mathcal{W},Y \notin \mathrm{dom}(E) \\ X,\mathcal{W},Y \text{ distinct} \end{matrix}}{\vdash E,\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix}}$$

(Env x$\mathcal{R}$y)

$$\frac{E \vdash \begin{matrix} A \\ \mathcal{R} \\ B \end{matrix} \qquad \begin{matrix} x,y \notin \mathrm{dom}(E) \\ x,y \text{ distinct} \end{matrix}}{\vdash E,\ \begin{matrix} x : A \\ \mathcal{R} \\ y : B \end{matrix}}$$

Related types

(Rel $\mathcal{W}$)

$$\frac{\vdash E',\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix},\ E''}{E',\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix},\ E'' \vdash \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix}}$$

(Rel X)

$$\frac{\vdash E',\ X,\ E''}{E',\ X,\ E'' \vdash X}$$

(Rel $\mathcal{W}$X)

$$\frac{\vdash E',\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix},\ E''}{E',\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix},\ E'' \vdash X}$$

(Rel $\mathcal{W}$Y)

$$\frac{\vdash E',\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix},\ E''}{E',\ \begin{matrix} X \\ \mathcal{W} \\ Y \end{matrix},\ E'' \vdash Y}$$

(Rel Arrow)

$$\frac{E \vdash \begin{matrix} A \\ \mathcal{R} \\ A' \end{matrix} \qquad E \vdash \begin{matrix} B \\ \mathcal{S} \\ B' \end{matrix}}{E \vdash \begin{matrix} A \to B \\ \mathcal{R} \to \mathcal{S} \\ A' \to B' \end{matrix}}$$

(Rel Forall $\mathcal{W}$)

$$\frac{E,\ \begin{matrix} X \\ \mathcal{W} \\ X' \end{matrix} \vdash \begin{matrix} B \\ \mathcal{S} \\ B' \end{matrix} \qquad \begin{matrix} X \notin B',\mathcal{S} \\ X' \notin B,\mathcal{S} \end{matrix}}{E \vdash \begin{matrix} \forall(X)B \\ \forall(\mathcal{W})\mathcal{S} \\ \forall(X')B' \end{matrix}}$$

(Rel Forall X)

$$\frac{E,\ X \vdash \begin{matrix} B \\ \mathcal{S} \\ B' \end{matrix}}{E \vdash \begin{matrix} \forall(X)B \\ \forall(X)\mathcal{S} \\ \forall(X)B' \end{matrix}}$$

(Rel Forall X$\mathcal{W}$)

$$\frac{E,\ \begin{matrix} X \\ \mathcal{W} \\ X' \end{matrix} \vdash \begin{matrix} B \\ \mathcal{S} \\ B' \end{matrix} \qquad \begin{matrix} X \notin B',\mathcal{S} \\ X' \notin B,\mathcal{S} \\ Z \notin \mathrm{dom}(E) \end{matrix}}{E \vdash \begin{matrix} \forall(X)B \\ \forall(Z)\mathcal{S}\{\mathcal{W} \leftarrow Z\} \\ \forall(X')B' \end{matrix}}$$

(Rel FRel) [$\mathcal{R}^1$ only]

$$\frac{E \vdash A \to B \qquad E \vdash b : A \to B}{E \vdash \overset{A}{(b)} \overset{}{} \overset{}{B}}$$

Relation equality

(Rel Eq Symm)

$$\frac{E \vdash \overset{A}{\mathcal{R}} = \overset{}{\mathcal{S}} \overset{}{} \overset{B}{}}{E \vdash \overset{A}{\mathcal{S}} = \overset{}{\mathcal{R}} \overset{}{} \overset{B}{}}$$

(Rel Eq Trans)

$$\frac{E \vdash \overset{A}{\mathcal{R}} = \overset{}{\mathcal{S}} \overset{B}{} \qquad E \vdash \overset{A}{\mathcal{S}} = \overset{}{\mathcal{T}} \overset{B}{}}{E \vdash \overset{A}{\mathcal{R}} = \overset{}{\mathcal{T}} \overset{B}{}}$$

(Rel Eq $\mathcal{W}$)

$$\frac{\vdash E', \overset{X}{\mathcal{W}}, E'' \overset{Y}{}}{E', \overset{X}{\mathcal{W}}, E'' \vdash \overset{X}{\mathcal{W}} = \overset{X}{\mathcal{W}} \overset{}{} \overset{Y}{\qquad Y}}$$

(Rel Eq X)

$$\frac{\vdash E', X, E''}{E', X, E'' \vdash \overset{X}{X} = \overset{X}{X}}$$

(Rel Eq $\mathcal{W}$X)

$$\frac{\vdash E', \overset{X}{\mathcal{W}}, E'' \overset{Y}{}}{E', \overset{X}{\mathcal{W}}, E'' \vdash \overset{X}{X} = \overset{X}{X} \overset{}{} \overset{Y}{\qquad X}}$$

(Rel Eq $\mathcal{W}$Y)

$$\frac{\vdash E', \overset{X}{\mathcal{W}}, E'' \overset{Y}{}}{E', \overset{X}{\mathcal{W}}, E'' \vdash \overset{X}{Y} = \overset{Y}{Y}}$$

(Rel Eq Arrow)

$$\frac{E \vdash \overset{A}{\mathcal{R}} = \overset{A'}{\mathcal{R}'} \qquad E \vdash \overset{B}{\mathcal{S}} = \overset{B'}{\mathcal{S}'}}{E \vdash \mathcal{R} \to \mathcal{S} = \mathcal{R}' \to \mathcal{S}' \overset{A \to B}{\qquad A' \to B'}}$$

(Rel Eq Forall $\mathcal{W}$)

$$\frac{E, \overset{X}{\mathcal{W}} \vdash \overset{B}{\mathcal{S}} = \overset{B'}{\mathcal{S}'} \overset{X}{} \qquad X \notin B', \mathcal{S}, \mathcal{S}' \quad X' \notin B, \mathcal{S}, \mathcal{S}'}{E \vdash \forall(\mathcal{W})\mathcal{S} = \forall(\mathcal{W})\mathcal{S}' \overset{\forall(X)B}{\qquad \forall(X')B'}}$$

(Rel Eq Forall X)

$$\frac{E, X \vdash \overset{B}{\mathcal{S}} = \overset{B'}{\mathcal{S}'}}{E \vdash \forall(X)\mathcal{S} = \forall(X)\mathcal{S}' \overset{\forall(X)B}{\qquad \forall(X)B'}}$$

(Rel Eq Forall X$\mathcal{W}$)

$$\frac{E, \overset{X}{\mathcal{W}} \vdash \overset{B}{\mathcal{S}} = \overset{B'}{\mathcal{S}'} \overset{X}{} \qquad X \notin B', \mathcal{S}, \mathcal{S}' \quad X' \notin B, \mathcal{S}, \mathcal{S}' \quad Z \notin \mathrm{dom}(E)}{E \vdash \forall(Z)\mathcal{S}\{\mathcal{W} \leftarrow Z\} = \forall(\mathcal{W})\mathcal{S}' \overset{\forall(X)B}{\qquad \forall(X)B'}}$$

(Rel Eq Extension) [$\mathcal{R}^1$ only]

$$\frac{E, \overset{x : A}{\mathcal{R}} \vdash \overset{x : A}{\mathcal{S}} \overset{y : B}{} \qquad E, \overset{x : A}{\mathcal{S}} \vdash \overset{x : A}{\mathcal{R}} \overset{y : B}{}}{E \vdash \mathcal{R} = \mathcal{S} \overset{A}{\qquad B}}$$

Related values

(Rel Val Symm)

$$\frac{E \vdash \overset{a : A}{} \overset{}{A} \overset{b : A}{}}{E \vdash \overset{b : A}{} \overset{}{A} \overset{}{a : A}}$$

(Rel Val Saturation Lft)

$$\frac{E \vdash \overset{a : A}{} \overset{}{A} \overset{b : A}{} \qquad E \vdash \overset{b : A}{} \overset{}{\mathcal{R}} \overset{}{c : B}}{E \vdash \overset{a : A}{} \overset{}{\mathcal{R}} \overset{}{c : B}}$$

(Rel Val Saturation Rht)

$$\frac{E \vdash \overset{b : A}{} \overset{}{\mathcal{R}} \overset{c : B}{} \qquad E \vdash \overset{c : B}{} \overset{}{B} \overset{}{d : B}}{E \vdash \overset{b : A}{} \overset{}{\mathcal{R}} \overset{}{d : B}}$$

(Rel Val Rel Eq)

$$\frac{E \vdash \overset{a : A}{} \overset{}{\mathcal{R}} \overset{b : B}{} \qquad E \vdash \overset{A}{\mathcal{R}} = \overset{}{\mathcal{S}} \overset{B}{}}{E \vdash \overset{a : A}{} \overset{}{\mathcal{S}} \overset{}{b : B}}$$

(Rel Val x$\mathcal{R}$y)

$$\frac{\vdash E',\ \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix},\ E''}{E',\ \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix},\ E'' \vdash \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix}}$$

(Rel Val x)

$$\frac{\vdash E',\ x : A,\ E''}{E',\ x : A,\ E'' \vdash x : A}$$

(Rel Val $\mathcal{R}$x)

$$\frac{\vdash E',\ \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix},\ E''}{E',\ \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix},\ E'' \vdash x : A}$$

(Rel Val $\mathcal{R}$y)

$$\frac{\vdash E',\ \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix},\ E''}{E',\ \begin{matrix}x : A\\ \mathcal{R}\\ y : B\end{matrix},\ E'' \vdash y : B}$$

(Rel Val Fun $\mathcal{R}$)

$$\frac{E,\ \begin{matrix}x : A\\ \mathcal{R}\\ x' : A'\end{matrix} \vdash \begin{matrix}b : B\\ \mathcal{S}\\ b' : B'\end{matrix} \qquad E \vdash \begin{matrix}B\\ \mathcal{S}\\ B'\end{matrix} \quad \begin{matrix}x \notin b'\\ x' \notin b\end{matrix}}{\begin{matrix}\lambda(x : A)b : A \to B\\ E \vdash \mathcal{R} \to \mathcal{S}\\ \lambda(x' : A')b' : A' \to B'\end{matrix}}$$

(Rel Val Fun A)

$$\frac{E,\ x : A \vdash \begin{matrix}b : B\\ \mathcal{S}\\ b' : B'\end{matrix} \qquad E \vdash \begin{matrix}B\\ \mathcal{S}\\ B'\end{matrix}}{\begin{matrix}\lambda(x : A)b : A \to B\\ E \vdash A \to \mathcal{S}\\ \lambda(x : A)b' : A \to B'\end{matrix}}$$

(Rel Val Appl)

$$\frac{E \vdash \begin{matrix}b : A \to B\\ \mathcal{R} \to \mathcal{S}\\ b' : A' \to B'\end{matrix} \qquad E \vdash \begin{matrix}a : A\\ \mathcal{R}\\ a' : A'\end{matrix}}{\begin{matrix}b(a) : B\\ E \vdash \mathcal{S}\\ b'(a') : B'\end{matrix}}$$

(Rel Val Fun2 $\mathcal{W}$)

$$\frac{E,\ \begin{matrix}X\\ \mathcal{W}\\ X'\end{matrix} \vdash \begin{matrix}b : B\\ \mathcal{S}\\ b' : B'\end{matrix} \quad \begin{matrix}X \notin b', B', \mathcal{S}\\ X' \notin b, B, \mathcal{S}\end{matrix}}{\begin{matrix}\lambda(X)b : \forall(X)B\\ E \vdash \forall(\mathcal{W})\mathcal{S}\\ \lambda(X')b' : \forall(X')B'\end{matrix}}$$

(Rel Val Fun2 X)

$$\frac{E,\ X \vdash \begin{matrix}b : B\\ \mathcal{S}\\ b' : B'\end{matrix}}{\begin{matrix}\lambda(X)b : \forall(X)B\\ E \vdash \forall(X)\mathcal{S}\\ \lambda(X)b' : \forall(X)B'\end{matrix}}$$

(Rel Val Appl2 $\mathcal{W}$)

$$\frac{E \vdash \begin{matrix}b : \forall(X)B\\ \forall(\mathcal{W})\mathcal{S}\\ b' : \forall(X')B'\end{matrix} \qquad E \vdash \begin{matrix}C\\ \mathcal{T}\\ C'\end{matrix}}{\begin{matrix}b(C) : B\{X \leftarrow C\}\\ E \vdash \mathcal{S}\{\mathcal{W} \leftarrow \mathcal{T}\}\\ b'(C') : B'\{X' \leftarrow C'\}\end{matrix}}$$

(Rel Val Appl2 X) [derivable in $\mathcal{R}^0$]

$$\frac{E \vdash \begin{matrix}b : \forall(X)B\\ \forall(X)\mathcal{S}\\ b' : \forall(X)B'\end{matrix} \qquad E \vdash C}{\begin{matrix}b(C) : B\{X \leftarrow C\}\\ E \vdash \mathcal{S}\{X \leftarrow C\}\\ b'(C) : B'\{X \leftarrow C\}\end{matrix}}$$

(Rel Val FRel Intro) [$\mathcal{R}^1$ only]

$$\frac{E \vdash b : A \to B \qquad E \vdash a : A}{\begin{matrix}a : A\\ E \vdash \langle b \rangle\\ b(a) : B\end{matrix}}$$

(Rel Val FRel Elim) [$\mathcal{R}^1$ only]

$$\frac{E \vdash \begin{matrix}a : A\\ \langle b \rangle\\ c : B\end{matrix} \qquad E \vdash b : A \to B}{\begin{matrix}b(a) : B\\ E \vdash B\\ c : B\end{matrix}}$$

(Rel Val Beta)

$$\frac{E,\ x : A \vdash b : B \qquad E \vdash a : A}{\begin{matrix}(\lambda(x : A)b)(a) : B\\ E \vdash B\\ b\{x \leftarrow a\} : B\end{matrix}}$$

(Rel Val Beta2)

$$\frac{E,\ X \vdash b : B \qquad E \vdash A}{\begin{matrix}(\lambda(X)b)(A) : B\{X \leftarrow A\}\\ E \vdash B\{X \leftarrow A\}\\ b\{X \leftarrow A\} : B\{X \leftarrow A\}\end{matrix}}$$

(Rel Val Eta)

$$\frac{E \vdash b : A \to B \qquad x \notin dom(E)}{\begin{matrix}\lambda(x : A)b(x) : A \to B\\ E \vdash A \to B\\ b : A \to B\end{matrix}}$$

(Rel Val Eta2)

$$\frac{E \vdash b : \forall(X)B \qquad X \notin dom(E)}{\begin{matrix}\lambda(X)b(X) : \forall(X)B\\ E \vdash \forall(X)B\\ b : \forall(X)B\end{matrix}}$$