A Logic for Parametric Polymorphism

Gordon Plotkin[*]   Martín Abadi[†]

Abstract

In this paper we introduce a logic for parametric polymorphism. Just as LCF is a logic for the simply-typed λ-calculus with recursion and arithmetic, our logic is a logic for System F. The logic permits the formal presentation and use of relational parametricity. Parametricity yields—for example—encodings of initial algebras, final co-algebras and abstract datatypes, with corresponding proof principles of induction, co-induction and simulation.

1 Introduction

In this paper we introduce a logic for parametric polymorphism, in the binary relational sense of Reynolds [Rey83]. Just as LCF is a first-order logic for the simply-typed λ-calculus, with recursion and arithmetic, so our logic is a second-order logic for System F. It is intended as a step towards a general logic of polymorphically typed programs. The terms are those of the second-order λ-calculus, and the formulae are built from equations and relations by propositional operators and quantifiers over elements of types, or over types, or over relations between types. The logic permits the formal presentation and use of relational parametricity, which is expressed by an axiom schema. Parametricity yields—for example—encodings of initial algebras, final co-algebras and abstract datatypes, with corresponding proof principles of induction, co-induction and simulation.

Our first goal is to provide a formal system for arguments that exploit relational parametricity. In all models of System F, standard constructions such as products and initial algebras are available in a weak sense (see *e.g.* [Böh85, Has90, RP90, Wra89]). If the models are relationally parametric, then these constructions become universal constructions in the usual sense of category theory. Bainbridge, Freyd, Scedrov, and Scott have given such results for the parametric Per model [BFSS90], and Hasegawa and Wadler for classes of models [Has90, Wad89]. Hasegawa [Has91] has shown that the second order minimal model—arising from the maximal consistent theory of Moggi and Statman—is parametric. By defining a formal logic, we hope to display the assumption of relational parametricity in a clear way, and to be able to obtain simple, general arguments for the results.

Our second goal is to pursue the idea of LCF. In LCF, a logic is given over a simply-typed λ-calculus with recursion and arithmetic. This λ-calculus is suitable for denotational semantics, and thus the corresponding logic acts as a rather powerful logic of programs. However, the simply-typed λ-calculus is inadequate for dealing with programming languages with abstract or polymorphic types, and an extension of the kind considered here is needed. The present work is but an intermediate step: it does not include recursion, at either

[*]Department of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, UK. Part of this work was completed while at Digital Equipment Corporation, Systems Research Center.

[†]Digital Equipment Corporation, Systems Research Center. 130 Lytton Avenue, Palo Alto, California 94301, USA.

the level of values or the level of types. Moreover, one may also wish to consider richer type systems (*e.g.*, that of F_ω or even Constructions), richer logics (*e.g.*, that of Topos Theory), or more general notions of computation (*e.g.*, as in Moggi's suggestions for the use of monads). In yet another direction, we may consider a formal theory of subtypes, based, say, on $F_\leq$ [CG91, CG92].

There has been some work along related lines. Abadi, Cardelli, and Curien gave a system with very elementary judgments and few relations, and with a syntax as close as possible to the basic typed λ-calculus [ACC93]. Mairson used second-order logic over the untyped λ-calculus [Mai91]. He interpreted the types of System F as relations; as it happens his interpretation of polymorphic types is not parametric, but it could easily be made so. The second-order theory of subtypes of Cardelli, Martini, Mitchell and Scedrov embodies some aspects of parametricity *via* an equational rule [CMMS91].

In Section 2 of this paper we present the logic. Apart from axioms giving the equational theory of System F, the only non-logical principle is a schema for parametricity; further, only intuitionistic reasoning is employed. Other axioms may well be possible: one would expect that intuitionistic reasoning is (unsurprisingly) consistent with various choice principles. The availability of these principles would be a miniature form of Pitts' result that polymorphism can be considered as set-theoretic as long as one works constructively [Pit87]. A model can be provided as in [BFSS90]; it remains to consider a more abstract notion and other examples. We also compare other notions of polymorphism: we present a tentative formalisation of Strachey's idea; we consider the dinatural approach of Bainbridge *et al*; and, finally, we make the evident generalisation of Reynold's binary relations to relations of other degrees. We conjecture that if two terms of System F are equal in all models parametric in Strachey's sense then they are also equal in all models parametric in Reynold's sense. We prove that a schema expressing dinaturality is a consequence of the schema expressing binary relational parametricity. The relationship betweeen the various notions of relational parametricity is unknown.

In Section 3 we consider a variety of constructions showing the availability of finite products and sums, second-order existential types, initial algebras and final co-algebras. The constructions are by now standard; the main point here is that we can derive their properties within our logic. We also consider the logical properties of the constructions. For example, we give a general induction principle for initial algebras and a general bisimulation (or co-induction) principle for final co-algebras [AM89, Smy91, Pit92]. For second-order existential types, parametricity becomes a "simulation principle", namely that if there is a relation between two types respected by the two corresponding "implementations" then the corresponding elements of the existential type (the abstract types) are equal. Thus we also obtain a proof rule for abstract types, along the lines envisaged by Mitchell [Mit91]. Presumably one would obtain similar principles relative to relational parametricity of other degrees, and, again, it would be interesting to know the relationships between these. In both Sections 2 and 3, we generally omit proofs.

2 Basic Logic

In this section we present the basic logic. The types and terms are those of System F and they are given by the grammar:

$$\begin{array}{lrl} \text{Types:} & A ::= & X \mid A \to B \mid \forall X.\, A \\ \text{Terms:} & t ::= & x \mid \lambda x : A.\, t \mid u(t) \mid \Lambda X.\, t \mid t(A) \end{array}$$

Here X ranges over type variables, x over ordinary variables; we use notations such as $A[X]$ to indicate possible occurrences of variables in expressions, and then may write for example $A[B]$ to represent the result of substituting B for X in A (avoiding capture of bound variables).

Using these terms we build up formulae from equations and binary relations.

$$\begin{array}{lll}\text{Formulae:} & \phi ::= & (t =_A u) \mid R(t,u) \mid \\ & & \phi \supset \psi \mid \forall x : A.\, \phi \mid \forall X.\, \phi \mid \forall R {\subset} A \times B.\, \phi \mid \\ & & \bot \mid \phi \wedge \psi \mid \phi \vee \psi \mid \exists x : A.\, \phi \mid \exists X.\, \phi \mid \exists R {\subset} A \times B.\, \phi\end{array}$$

Here R ranges over relation variables. The equality symbol is subscripted with a type, the type of the terms being equated. In System F, this type is unique, in a given environment, and so we often leave it implicit. The basic constructs are implication ($\supset$) and three sorts of universal quantifiers: over values; over types; and over relations between types (where $R {\subset} A \times B$ is read as "R is a relation between A and B"). The other constructs are useful but not altogether necessary. We make use of standard abbreviations when writing formulae, terms and types.

Typing judgments are used to specify which terms have which types, and which formulae are well-formed. A second-order environment E is a finite sequence of type variables X or typings $x : A$ in which no variable is introduced twice. The judgments $E \vdash A$ Type and $E \vdash t : A$ are defined in the usual way. To specify the well-formed formulae, we also need relation environments, which are finite sequences of relational typings $R {\subset} A \times B$ in which no variable is introduced twice. We define a judgment $E \vdash G$ REnv to assert that G is a well-formed relation environment given E, with the obvious rule that if $R {\subset} A \times B$ appears in G then A and B should be well-formed types given E. We define a judgment $E; G \vdash \phi$ Prop to assert that ϕ is a well-formed formula given $E; G$. The rules for atomic formulae are:

$$\frac{E \vdash t : A \quad E \vdash u : A \quad E \vdash G \ \text{REnv}}{E; G \vdash t =_A u \ \text{Prop}}$$

$$\frac{E \vdash t : A \quad E \vdash u : B \quad E \vdash G \ \text{REnv} \quad R {\subset} A \times B \text{ in } G}{E; G \vdash R(t,u) \ \text{Prop}}$$

Among the other rules we have, for example:

$$\frac{E, X; G \vdash \phi \ \text{Prop}}{E; G \vdash \forall X.\, \phi \ \text{Prop}} \qquad \frac{E; G, R {\subset} A \times B \vdash \phi \ \text{Prop}}{E; G \vdash \forall R {\subset} A \times B.\, \phi \ \text{Prop}}$$

In order to define the consequence relation of the logic we need further logical apparatus. This concerns definable relations, their substitution for relation variables in formulae (obtaining formulae) and for type variables in types (obtaining definable relations). Substitution in formulae is needed for the rules for relational quantification; substitution in types is needed to state the parametricity schema. Definable relations are given by the grammar:

$$\text{Definable relations:} \qquad \rho ::= \ (x : A, y : B).\, \phi[x,y]$$

We say that such a ρ is a definable relation between A and B, writing this as $\rho \subset A \times B$. We define a judgment $E; G \vdash \rho \subset A \times B$ to assert that ρ is a well-formed definable relation between A and B, given $E; G$. There is one rule for this judgment:

$$\frac{E; G, x : A, y : B \vdash \phi \ \text{Prop}}{E; G \vdash (x : A, y : B).\, \phi \subset A \times B}$$

For example, the expression $(x : A, y : A).\, (x =_A y)$ defines the equality relation eq_A on A, and $\langle f \rangle = (x : A, y : B).\, (y =_B f(x))$ defines the graph of a function f; here if f has type $A \to B$ then $\langle f \rangle$ is a well-formed definable relation between A and B (given appropriate $E;G$). We sometimes treat a relation variable $R \subset A \times B$ as the definable relation $(y : A, z : B).\, R(y, z)$.

When ρ is such a definable relation between A and B, we sometimes use the abbreviations $t\rho u$ or $\rho(t, u)$ for the corresponding $\phi[t, u]$. A definable relation ρ can be substituted for a relation variable R in a formula $\psi[R]$, yielding $\psi[\rho]$. In particular when $\psi[R]$ is $R(t, u)$, the result of the substitution is $\rho(t, u)$.

To define the substitution of definable relations for type variables in types, we need first to be able to combine relations by exponentiation and universal quantification. If $\rho \subset A \times B$ and $\rho' \subset A' \times B'$ then

$$(\rho \to \rho') \subset (A \to A') \times (B \to B')$$

is defined by:

$$f(\rho \to \rho')g \equiv \forall x : A \forall x' : A'.\, (x\rho x' \supset f(x)\rho' g(x'))$$

If ρ and ρ' are well-formed given $E;G$ then so is $\rho \to \rho'$.

For universal quantification, if $\rho \subset A \times B$, then

$$\forall(Y, Z, R{\subset}Y \times Z)\rho \subset (\forall Y.\, A) \times (\forall Z.\, B)$$

is defined by:

$$y(\forall(Y, Z, R{\subset}Y \times Z).\, \rho)z \equiv \forall Y \forall Z \forall R{\subset}Y \times Z.\, ((yY)\rho(zZ))$$

If ρ is well-formed given $E, Y, Z; G, (R{\subset}Y \times Z)$ then $\forall(Y, Z, R{\subset}Y \times Z)\rho$ is well-formed given $E;G$.

We can now define the substitution of definable relations for type variables in types. If $\vec{X} = X_1, \ldots, X_n$, $\vec{B} = B_1, \ldots, B_n$, $\vec{C} = C_1, \ldots, C_n$, and $\vec{\rho} = \rho_1, \ldots, \rho_n$ where $\rho_i \subset B_i \times C_i$, then $A[\vec{\rho}] \subset A[\vec{B}] \times A[\vec{C}]$ is the result of substituting $\vec{\rho}$ for $\vec{X}$ in $A[\vec{X}]$. The definition of $A[\vec{\rho}]$ is by cases:

- if A is X_i then $A[\vec{\rho}]$ is ρ_i;
- if A is $A' \to A''$ then $A[\vec{\rho}] = A'[\vec{\rho}] \to A''[\vec{\rho}]$;
- if A is $\forall Y.\, A'[\vec{X}, Y]$ then $A[\vec{\rho}] = \forall(Y, Z, R{\subset}Y \times Z).\, A'[\vec{\rho}, R]$.

If each ρ_i is well-formed given $E;G$ then so is $A[\vec{\rho}]$.

It remains to give axiom schemes and rules in order to define the consequence relation of the logic. This relation is written as $\Gamma \vdash_{E;G} \phi$, where Γ is a finite set of formulae, and all formulae involved are well-formed given $E;G$. The proof system has three parts:

- standard rules for the connectives and quantifiers;
- axioms and rules for the equational part of System F;
- a schema to express relational parametricity.

Natural deduction rules for the connectives and quantifiers are given as usual for intuitionistic logic. For example the rules for implication and universal quantification over types and relations are:

$$\frac{\Gamma, \phi \vdash_{E;G} \psi}{\Gamma \vdash_{E;G} \phi \supset \psi} \qquad \frac{\Gamma \vdash_{E;G} \phi \supset \psi \qquad \Gamma \vdash_{E;G} \phi}{\Gamma \vdash_{E;G} \psi}$$

$$\frac{\Gamma \vdash_{E,X;G} \phi[X]}{\Gamma \vdash_{E;G} \forall X.\, \phi[X]} \qquad \frac{\Gamma \vdash_{E;G} \forall X.\, \phi[X] \qquad E \vdash A \ \text{Type}}{\Gamma \vdash_{E;G} \phi[A]}$$

$$\frac{\Gamma \vdash_{E;G,R\subset A\times B} \phi[R]}{\Gamma \vdash_{E;G} \forall R{\subset} A\times B.\, \phi[R]} \qquad \frac{\Gamma \vdash_{E;G} \forall R{\subset} A\times B.\, \phi[R] \qquad E;G \vdash \rho \subset A\times B}{\Gamma \vdash_{E;G} \phi[\rho]}$$

In the above rules the usual provisions are made that the variables being bound do not appear in assumptions.

We now give that part of the logic that deals with equality. This consists of certain axioms and axioma schemes. We adopt the convention that if an axiom ϕ is written, what is meant is that all sequents of the form $\Gamma \vdash_{E;G} \phi$ are asserted, where ϕ and all formulae in Γ are well-formed for some $E;G$. Reflexivity is the axiom

$$\forall X \forall x : X.\, (x =_X x)$$

Substitution is the axiom

$$\begin{aligned}&\forall X \forall Y \forall R{\subset} X\times Y \forall x : X \forall x' : X \forall y : Y \forall y' : Y.\\ &\qquad R(x,y) \wedge x =_X x' \wedge y =_Y y' \supset R(x',y')\end{aligned}$$

We also use the congruence schemas

$$(\forall x : A.\, t =_B u) \supset (\lambda x : A.\, t) =_{A\to B} (\lambda x : A.\, u)$$

$$(\forall X.\, t =_B u) \supset (\Lambda X.\, t) =_{\forall X.\, B} (\Lambda X.\, u)$$

Lastly there are β equalities:

$$\forall x : A.\, ((\lambda x : A.\, t)x =_B t)$$

$$\forall X.\, ((\Lambda X.\, t)X =_B t)$$

and η equalities:

$$\forall X \forall Y \forall f : X \to Y.\, ((\lambda x : X.\, fx) =_{X\to Y} f)$$

$$\forall f : \forall X.\, B.\, ((\Lambda X.\, fX) =_{\forall X.\, B} f)$$

We can now formulate the parametricity schema:

$$\forall Y_1 \ldots \forall Y_n \forall u : (\forall X.\, A[X, Y_1, \ldots, Y_n]).\, u(\forall X.\, A[X, eq_{Y_1}, \ldots, eq_{Y_n}])u$$

where A has free type variables $X, Y_1, \ldots, Y_n$.

To understand this, it is convenient to ignore the parameters, $Y_1, \ldots, Y_n$, and expand the definition to obtain

$$\forall u : (\forall X.\, A[X]).\, \forall Y \forall Z \forall R{\subset} Y \times Z.\, u(Y) A[R] u(Z)$$

This formula states that if one instantiates an element of a polymorphic type at two related types, then the two values obtained are themselves related; this statement expresses Reynolds' idea of binary relational parametricity.

One way to interpret this logic is to follow Bainbridge *et al.* [BFSS90], taking types to be pers over the natural numbers and with universal quantification being interpreted parametrically with respect to the double-negation-closed relations. Formulae are interpreted classically, with type variables ranging over pers, ordinary variables ranging over equivalence classes relative to the appropriate per, and relation variables ranging over the double-negation closed relations. There are other evident possibilities of interpretation; for example one could interpret the logic intutionistically *via* a realisability interpretation. It would be good to have a general framework for interpreting the logic. Both Reynolds and Ma [MR92] and Hasegawa [Has90, Has91] have considered frameworks for parametric interpretations of System F. Pitts has shown that every hyperdoctrine model of System F fully embeds in a topos model [Pit87]. In this way the types of the model appear as "sets" in the topos. Perhaps something similar can be done here, in such a way that the relations of the model appear as relations in the topos, in the usual categorical sense. One would hope for completeness results for the present logic. In this connection one can ask too what is the most convenient language combining that of the present logic and the higher-order logic afforded by a topos; this question is apposite even without any consideration of parametricity.

Lemma 1 (Identity Extension Lemma) *For any $A[\vec{X}]$ with free variables in $\vec{X}$ it is provable in the logic that*

$$\forall\vec{X}\forall u : A, v : A.\ (uA[eq_{\vec{X}}]v \ \equiv\ (u =_A v))$$

Lemma 2 (Logical Relations Lemma) *Let $A_1[\vec{X}]$, ..., $A_m[\vec{X}]$, $B[\vec{X}]$ have free type variables in $\vec{X}$. Suppose $t[x_1, \ldots, x_m] : B$ given the environment $\vec{X}, x_1 : A_1, \ldots, x_m : A_m$. Then the following is provable in the logic without using the parametricity schema:*

$$\begin{array}{l}\forall\vec{X}\forall\vec{Y}\forall\vec{R}\subset\vec{X}\times\vec{Y}\\ \quad\forall x_1 : A_1[\vec{X}], \ldots, x_m : A_m[\vec{X}]\\ \quad\forall y_1 : A_1[\vec{Y}], \ldots, y_m : A_m[\vec{Y}].\\ \qquad(x_1A_1[\vec{R}]y_1 \wedge \ldots \wedge x_mA_m[\vec{R}]y_m) \supset t[x_1, \ldots, x_m]\, A[\vec{R}]\, t[y_1, \ldots, y_m]\end{array}$$

2.1 Categorical Matters

The types form a category, in a formal sense, within our logic. The idea is to take the type $X \to X'$ as the "set" of morphisms from X to X'. Composition is given by the combinator $\mathrm{comp} : \forall X\forall X'\forall X''.(X' \to X'') \to (X \to X') \to (X \to X'')$, where:

$$\mathrm{comp} = \Lambda X\Lambda X'\Lambda X''\lambda g : X' \to X''\lambda f : X \to X'\lambda x : X.\, g(f(x))$$

and the identity, $\mathrm{id} : \forall X.\,(X \to X)$ by:

$$\mathrm{id} = \Lambda X\lambda x : X.\, x$$

We write $t; u$ for $\mathrm{comp}(A)(A')(A'')(u)(t)$ (where u and t have respective types $A' \to A''$ and $A \to A'$, given $E; G$); we write id_A for $\mathrm{id}(A)$. One can show that composition is associative, in that it is provable in the logic (without using the parametricity schema) that:

$$\forall X, X', X'', X'''.\forall f : X \to X', g : X' \to X'', h : X'' \to X'''.\,(f; g); h = f; (g; h)$$

and for the identity that:

$$\forall X, X'.\forall f : X \to X'.\,(id_X; f = f) \wedge (f; id_{X'} = f)$$

Now let $A[\vec{X}, \vec{Y}]$ be a type such that all type variables in $\vec{X}$ have only negative occurrences, and all type variables in $\vec{Y}$ have only positive occurrences. Then one may think of $A[\vec{X}, \vec{Y}]$ as acting on types $\vec{X}, \vec{Y}$ to produce a type. With an additional action on functions one obtains an associated multivariant formal functor. This action is given by a term:

$$M_A : \forall\vec{X}, \vec{Y}.\,\forall\vec{X'}, \vec{Y'}.\,(\vec{X'} \to \vec{X}) \to (\vec{Y} \to \vec{Y'}) \to (A[\vec{X}, \vec{Y}] \to A[\vec{X'}, \vec{Y'}])$$

The term is given inductively on the structure of A.

$$
\begin{aligned}
M_Z\vec{X}\vec{Y}\vec{X}'\vec{Y}'\vec{f}\vec{g} &= \begin{cases} f_i & \text{(if } Z \text{ is the } i\text{th entry in } \vec{X}) \\ g_j & \text{(if } Z \text{ is the } j\text{th entry in } \vec{Y}) \\ id_Z & \text{(otherwise)} \end{cases} \\
M_{(B[\vec{Y},\vec{X}]\to C[\vec{X},\vec{Y}])}\vec{X}\vec{Y}\vec{X}'\vec{Y}'\vec{f}\vec{g} &= \lambda h : B[\vec{Y},\vec{X}] \to C[\vec{X},\vec{Y}]. \\
&\quad (M_{B[\vec{Y},\vec{X}]}\vec{Y}'\vec{X}'\vec{Y}\vec{X}\vec{g}\vec{f}); h; (M_{C[\vec{X},\vec{Y}]}\vec{X}\vec{Y}\vec{X}'\vec{Y}'\vec{f}\vec{g}) \\
M_{\forall Z.\, B[\vec{X},\vec{Y}]}\vec{X}\vec{Y}\vec{X}'\vec{Y}'\vec{f}\vec{g} &= \lambda z : (\forall Z.\, B[\vec{X},\vec{Y}])\Lambda Z.\, (B[\vec{X},\vec{Y}]\vec{X}\vec{Y}\vec{X}'\vec{Y}'\vec{f}\vec{g}(zZ))
\end{aligned}
$$

Given $\vec{t}$,$\vec{u}$ of types $\vec{B}' \to \vec{B}$, $\vec{C} \to \vec{C}'$, we write $A[\vec{t},\vec{u}]$ for $M_A\vec{A}\vec{B}\vec{A}'\vec{B}'\vec{t}\vec{u}$. One can show that this yields a functor. That is, it is provable in the logic (without using the parametricity schema) that:

$$\forall \vec{X},\vec{Y}.\, (A[id_{\vec{X}}, id_{\vec{Y}}] = id_{A[\vec{X},\vec{Y}]})$$

and also:

$$\forall \vec{X},\vec{Y},\vec{X}',\vec{Y}',\vec{X}'',\vec{Y}''.\, \forall \vec{f}{:}\vec{X}' \to \vec{X}.\, \forall \vec{g}{:}\vec{Y} \to \vec{Y}'.\, \forall \vec{f}'{:}\vec{X}'' \to \vec{X}'.\, \forall \vec{g}'{:}\vec{Y}' \to \vec{Y}''.$$

$$(A[(\vec{f}';\vec{f}),(\vec{g};\vec{g}')] = A[\vec{f},\vec{g}];A[\vec{f}',\vec{g}'])$$

One can also show that the operations of applying functors and taking graphs commute. The *opposite* of a relation $\rho \subset A \times B$ is defined by:

$$\rho^{op} = (y : B, x : A).x\rho y$$

Lemma 3 (Graph Lemma) *Let $A[\vec{X},\vec{Y}]$ be a type all of whose free variables appear in the list $\vec{X},\vec{Y}$ and such that all type variables in $\vec{X}$ have only negative occurrences, and all type variables in $\vec{Y}$ have only positive occurrences. Then it can be proved in the logic that:*

$$\forall \vec{X},\vec{Y},\vec{X}',\vec{Y}'.\, \forall \vec{f}{:}\vec{X}' \to \vec{X}.\, \forall \vec{g}{:}\vec{Y} \to \vec{Y}'.\, (\langle A[\vec{f},\vec{g}]\rangle = A[\langle \vec{f}\rangle^{op}, \langle \vec{g}\rangle])$$

Here two relations are considered as equal if they coincide extensionally.

2.2 Other Notions of Parametricity

As well as Reynold's relational notion of parametricity, there are two others: an informal one due to Strachey [Str67] and a categorical one due to Bainbridge *et al.* [BFSS90]. Strachey's original idea was that a parametric polymorphic function behaves the same way for all types. Let us essay a formalisation of this idea. In Per models, universal types can be modelled as intersections of all their instances; so at any type a polymorphic function has the same realisor as at any other. It seems reasonable, therefore, to claim that Strachey could accept any Per model as being parametric. Now Mitchell has shown that two terms of the same type are equal in a wide class of such models iff their erasures are $\beta\eta$-equivalent [Mit90]. Let us (informally) define two terms of System F to be *Strachey equivalent* iff they are equal in all models of parametric polymorphism in Strachey's sense. Accepting Per models as parametric, we are lead to assert that Strachey equivalence implies being of the same type and having the same erasure. Conversely if two terms have the same type and erasure it seems not unreasonable that they would denote the same element of any model parametric in Strachey's sense. We therefore tentatively identify Strachey equivalence with having the same type and the same erasure (up to $\beta\eta$-equivalence).

Along similar lines, one would informally define two terms of System F to be *Reynold's equivalent* iff they are equal in all models of binary relational polymorphism in Reynold's sense. Anticipating a completeness theorem for our logic relative to models of System F which are polymorphic in the binary relational sense, we can identify Reynold's equivalence with provable equality in our logic. Since there are many examples of terms provably equal in the above theory which are not Strachey equivalent, Reynolds equivalence does not imply Strachey equivalence. We conjecture the converse does hold, (A similar conjecture appears in [ACC93].) Many positive instances encourage this conjecture. For example, $\lambda x : (\forall X.\, X).\, x$ and $\lambda x : (\forall X.\, X).\, x(\forall X.\, X)$ have the same erasure and the same type $(\forall X. X) \to (\forall X. X)$, and they are provably equal; if X is not free in A then $\lambda x : (\forall X. A). xB$ and $\lambda x : (\forall X.\, A).\, xC$ have the same erasure and the same type $(\forall X.\, A) \to A$, and they too are provably equal.

According to Bainbridge *et al.* parametricity is provided by the notion of a dinatural transformation: in certain models, all terms denote such transformations between types [BFSS90]. One can express dinaturality by a schema. Let $A[Y, X]$ be a type in which Y occurs only negatively and X occurs only positively. Then:

$$\begin{array}{l} \forall X, Y \forall t : (\forall X. A[X, X] \to A[Y, Y)]) \forall f : X \to Y. \\ \quad A[f, id_X]; t(X); B[id_X, f] = A[id_Y, f]; t(Y); B[f, id_Y] \end{array}$$

We show now that this schema follows from that for (relational) parametricity. (This seems to answer a question raised in [BFSS90].)

First, the schema can be recast in a simpler way, as:

$$\forall X, Y \forall f : X \to Y.\, (\cdot)_X; A[id_X, f] = (\cdot)_Y; A[f, id_Y]$$

where (again) $A[Y, X]$ is a type in which Y occurs only negatively and X occurs only positively and where, *e.g.*, $(\cdot)_X$ is $\lambda u : (\forall X.\, A[X, X]).\, u(X)$. For the sake of notational simplicity let us only consider the case where there are no parameters (*i.e.* free type variables in A other than X or Y). So let X and Y be types and suppose that $f : X \to Y$. Choose u in $(\forall X.\, A[X, X])$. Appying the parametricity schema to $\langle f \rangle$ we obtain:

$$u(X) A[\langle f \rangle, \langle f \rangle] u(Y)$$

We have that $A[h, g] : A[V, U] \to A[V', U']$ given the $U, V, U', V', g : U \to U', h : V' \to V$. We may therefore apply the Logical Relations Lemma, obtaining:

$$A[id_X, f](A[\langle f \rangle, \langle f \rangle] \to A[eq_X, eq_Y]) A[f, id_Y]$$

as $f(\langle f \rangle \to eq_Y) id_Y$ and $id_X(eq_X \to \langle f \rangle) f$. But now it follows that:

$$A[id_X, f](u(X)) A[eq_X, eq_Y] A[f, id_Y](u(Y))$$

and by the Identity Extension Lemma this is just:

$$A[id_X, f](u(X)) =_{A[X,Y]} A[f, id_Y](u(Y))$$

as required.

There is an evident generalisation of relational parametricity to n-ary relational parametricity. One introduces n-ary definable relations ($n \geq 0$) by:

$$\rho = (x_1 : A_1, \ldots, x_n : A_n).\phi[x_1, \ldots, x_n]$$

We write $\rho \subset (A_1 \times \ldots \times A_n)$, and say ρ is well-formed given $E; G$ provided that $\phi[x_1, \ldots, x_n]$ is well-formed given $E, x_1 : A_1, \ldots, x_n : A_n; G$. We also write $\rho(t_1, \ldots, t_n)$ for $\phi[t_1, \ldots, t_n]$. The place of the equality relation is taken by the diagonal Δ^n_A where:

$$\Delta^n_A = (x_1 : A, \ldots, x_n : A).\, (x_1 = x_2) \wedge \ldots \wedge (x_{n-1} = x_n)$$

Exponentiation and universal quantification of n-ary relations is defined analogously to the case of binary relations, and we may substitute definable n-ary relations in types to obtain definable n-ary relations. The schema of n-ary relational parametricity is:

$$\forall Y_1 \ldots \forall Y_m \forall u : (\forall X.\, A[X, Y_1, \ldots, Y_m]).\, (\forall X.\, A[X, \Delta^n_{Y_1}, \ldots, \Delta^n_{Y_m}])(u, \ldots, u)$$

where A has free type variables $X, Y_1, \ldots, Y_n$. It is unknown what the relations between these schemas are. We do not even know if the schema for binary relational parametricity implies that for unary parametricity.

3 Parametricity and Datatypes

We now treat datatype constructors, comprising: finite products and sums, second-order existential quantification; and initial algebras and final co-algebras. As well as categorical properties, we derive logical properties of the constructs.

3.1 Products

For the empty product, let us define:

$$\mathbf{1} = \forall \mathbf{X}.\, \mathbf{X} \rightarrow \mathbf{X}$$

The term $\star = \Lambda X \lambda x : X.\, x$ inhabits $\mathbf{1}$. It is provable, using dinaturality, that:

Theorem 1 $\forall u : \mathbf{1}.\, (\mathbf{u} =_{\mathbf{1}} \star)$

Note that we have adopted an informal style, asserting formulae. What we intend is to assert that the formulae are provable in the logic. The categorical statement

$$\forall X \exists ! f : X \rightarrow \mathbf{1}.\, \top$$

of terminality follows at once from the theorem.

Turning to binary products, for any types A and B, set

$$A \times B = \forall Z.\, ((A \rightarrow B \rightarrow Z) \rightarrow Z)$$

where Z does not appear in A or B. This yields a weak product in System F, with combinators $\mathit{fst}_{A,B} : A \times B \rightarrow A$, $\mathit{snd}_{A,B} : A \times B \rightarrow B$, and $\mathit{pair}_{A,B} : A \rightarrow B \rightarrow A \times B$, given by

$$\mathit{fst}_{A,B}(z) = zAK_{A,B}$$

$$\mathit{snd}_{A,B}(z) = zBK'_{A,B}$$

(where $K_{A,B} = \lambda x : A \lambda y : B.\, x$ and $K'_{A,B} = \lambda x : A \lambda y : B.\, y$),and:

$$\mathit{pair}_{A,B}(a)(b) = \Lambda Z \lambda f : A \rightarrow B \rightarrow Z.\, fab$$

Below we drop the subscripts A, B and also write $\langle t, u \rangle$ for $\mathit{pair}(t)(u)$.

The formulae

$$\forall x : A \forall y : B.\, (\mathit{fst}(\langle x, y \rangle) = x)$$

and

$$\forall x : A \forall y : B.\, (\mathit{snd}(\langle x, y \rangle) = y)$$

are provable without using parametricity. Using dinaturality we obtain:

Theorem 2 $\forall w : A \times B. \, (\langle \mathit{fst}(w), \mathit{snd}(w) \rangle =_{A\times B} w)$

The theorem yields the usual categorical characterization of binary products, namely:

$$\forall Z \forall f : Z \to A \forall g : Z \to B. \, \exists! h : Z \to A \times B. \, (f = h; \mathit{fst} \wedge g = h; \mathit{snd})$$

Lastly we consider the action of binary products on relations. Given $\rho \subset A \times A'$ and $\sigma \subset B \times B'$ we take $(\rho \times \sigma) \subset (A \times B) \times (A' \times B')$ to be $\forall Z. \, (\rho \to (\sigma \to Z)) \to Z$. This is obtained by substituting ρ and σ for X and Y in $X \times Y$, which is $\forall Z. \, (X \to (Y \to Z)) \to Z$.

Theorem 3

$$\forall u : A \times A' \forall v : B \times B'. \, u(\rho \times \sigma)v \equiv (\mathit{fst}(u) \rho \, \mathit{fst}(v) \wedge \mathit{snd}(u) \sigma \, \mathit{snd}(v))$$

3.2 Sums

For the empty sum, define:

$$\mathbf{0} = \forall \mathbf{X}. \, \mathbf{X}$$

Theorem 4 $\forall u : \mathbf{0}. \; \bot$

It follows that $\mathbf{0}$ is initial in the categorical sense.

Turning to binary sums, for types A and B we take

$$A + B = \forall Z. \, ((A \to Z) \to (B \to Z) \to Z)$$

where Z does not occur in A or B. The combinators $\mathit{inl}_{A,B} : A \to A{+}B$, $\mathit{inr}_{A,B} : B \to A{+}B$, and $\mathit{cases}_{A,B} : \forall Z. \, ((A \to Z) \to (B \to Z) \to (A + B) \to Z)$ are defined by:

$$\mathit{inl}_{A,B}(a) = \Lambda Z \lambda f : A \to Z \lambda g : B \to Z. \, f(a)$$

$$\mathit{inr}_{A,B}(b) = \Lambda Z \lambda f : A \to Z \lambda g : B \to Z. \, g(b)$$

and

$$\mathit{cases}_{A,B} C f g u = u C f g$$

We may drop the subscripts and may also write $[t, u]_C$ for $\mathit{cases}(t, u)$.

It is provable without using parametricity that

$$\forall x : A \forall Z \forall f : A \to Z \forall g : B \to Z. \, ([f, g]_Z(\mathit{inl}(x)) = f(x))$$

and

$$\forall y : B \forall Z \forall f : A \to Z \forall g : B \to Z. \, ([f, g]_Z(\mathit{inr}(y)) = g(y))$$

With dinaturality we obtain:

Theorem 5 $\forall Z \forall h : A + B \to Z. \, h = [\mathit{inl}; h, \mathit{inr}; h]_Z$

from which the usual categorical characterisation of binary sums follows.

Next we consider the action of binary sums on relations. Given $\rho \subset A \times A'$ and $\sigma \subset B \times B'$ we take $(\rho + \sigma) \subset (A + B) \times (A' + B')$ to be $\forall Z. \, ((\rho \to Z) \to (\sigma \to Z) \to Z)$, following the same pattern as for binary products.

Theorem 6

$$\begin{array}{l}\forall u : A + B \forall u' : A' + B'.\\ \quad u(\rho + \sigma)u'\\ \quad\quad \equiv\\ \quad (\exists x : A \exists x' : A'.\, (u = \mathit{inl}(x) \wedge u' = \mathit{inl}(x') \wedge x\rho x')\\ \quad\quad\quad \vee\\ \quad \exists y : B \exists y' : B'.\, (u = \mathit{inr}(y) \wedge u' = \mathit{inr}(y') \wedge y\sigma y'))\end{array}$$

By the Identity Extension Lemma, $\mathit{eq}_A + \mathit{eq}_B$ and eq_{A+B} are equivalent. With this and the above theorem we get that:

$$\forall u : A + B.\, (\exists x : A.\, u = \mathit{inl}(x)) \vee (\exists y : B.\, u = \mathit{inr}(y))$$

3.3 Existential Quantification

The existential quantification $\exists X.\, A[X]$ is taken to abbreviate

$$\forall Y.\, (\forall X.\, (A[X] \to Y) \to Y)$$

where Y does not appear in A and is different from X.

One defines the combinator $\mathit{pack} : \forall X.\, (A[X] \to \exists X.\, A[X])$ and the combinator unpack : $(\exists X.\, A[X]) \to \forall Y.\, (\forall X.\, (A[X] \to Y) \to Y)$ by:

$$\mathit{pack}(X)(x) = \Lambda Y \lambda f : \forall X.\, (A[X] \to Y).\, f(X)(x)$$

and

$$\mathit{unpack}(u)(Y)(f) = u(Y)(f)$$

and it is provable without using parametricity that

$$\forall X \forall x : A[X] \forall Y \forall f : (\forall X.\, (A[X] \to Y).\, (\mathit{unpack}(\mathit{pack} X x) Y f =_Y f X x)$$

Let us see how existential quantification operates on relations. Let $\vec{Z}$ be the type variables free in $A[X, \vec{Z}]$, other than X, and let $\vec{\rho} \subset \vec{B} \times \vec{C}$ be a vector of relations of the same length. Then $\exists X.\, A[X, \vec{\rho}]$ is taken to be $\forall Y.\, (\forall X.\, (A[X, \vec{\rho}] \to Y) \to Y)$, following the usual pattern. Using the parametricity schema, one can show:

Theorem 7

$$\begin{array}{l}\forall u : (\exists X.\, A[X, \vec{B}]), v : (\exists X.\, A[X, \vec{C}]).\\ \quad u(\exists X.\, A[X, \vec{\rho}])v\\ \quad\quad \equiv\\ \quad (\exists X, Y \exists x : A[X, \vec{B}], y : A[Y, \vec{C}] \exists S \subset X \times Y.\\ \quad\quad u = (\mathit{pack} X x) \wedge v = (\mathit{pack} Y y) \wedge x(A[S, \vec{\rho}])y)\end{array}$$

Applying the Identity Extension Lemma, we get:

$$\begin{array}{l}\forall \vec{Z} \forall u, v : (\exists X.\, A[X, \vec{Z}]).\\ \quad u = v\\ \quad\quad \equiv\\ \quad (\exists X, Y \exists x : A[X, \vec{Z}], y : A[Y, \vec{Z}] \exists S \subset X \times Y.\\ \quad\quad u = (\mathit{pack} X x) \wedge v = (\mathit{pack} Y y) \wedge x(A[S, \mathit{eq}_{\vec{Z}}])y)\end{array}$$

The implication from right to left can be viewed as stating that existentials are parametric (in a sense dual to that axiomatized for universals); it is a principle that enables one to show that two abstract datatypes are equal, if one can demonstrate a simulation relation between them. We view this as the proof principle requested by Mitchell in [Mit91].

Working from left to right, one obtains:

$$\forall \vec{Z} \forall u : (\exists X.\, A[X, \vec{Z}]) \exists X \exists x : A[X, \vec{Z}].(u = (\mathit{pack} X x))$$

and one can then obtain a categorical characterization of existential quantification in the form:

$$\forall Y \forall f : (\forall X.\, (A[X] \to Y)) \exists ! g : (\exists X.\, A[X] \to Y) \forall X.\, (f X =_Y (\mathit{pack} X); g)$$

3.4 Initial Algebras

Let $A[Z]$ be a type where the variable Z occurs only positively. The initial $A[Z]$ algebra is given by

$$\mu Z.\, A[Z] = \forall Z.\, ((A[Z] \to Z) \to Z)$$

and the combinators are

$$\mathit{fold} : \forall Z.\, ((A[Z] \to Z) \to ((\mu Z.\, A[Z]) \to Z))$$

and

$$\mathit{in} : A[\mu Z.\, A[Z]] \to \mu Z.\, A[Z]$$

where:

$$\mathit{fold}(Z)(f) = \lambda z : \mu Z.\, A[Z].\, z(Z)(f)$$

and

$$\mathit{in}(z) = \Lambda Z \lambda f : A[Z] \to Z.\, f(A[\mathit{fold} Z f] z)$$

.

The structure $\langle \mu Z.\, A[Z], \mathit{in} \rangle$ is a weakly initial $A[Z]$ algebra:

$$\forall Z \forall f : A[Z] \to Z.\, (\mathit{in}; (\mathit{fold} Z f) = A[\mathit{fold} Z f]; f)$$

This can be shown without using parametricity (or any η-equalities). With parametricity one has the following expression of the initiality of $\langle \mu Z.\, A[Z], \mathit{in} \rangle$:

Theorem 8 $\forall Z \forall f : A[Z] \to Z \exists ! h : \mu Z.\, A[Z] \to Z.\, (\mathit{in}; h = A[h]; f)$

Turning to logical relations let $\vec{X}$ be a list of the type variables free in $A[Z, \vec{X}]$ apart from Z and let $\vec{\rho} \subset \vec{B} \times \vec{C}$ be a list of relations of the same length. Then the relation $\mu Z.\, A[Z, \vec{\rho}] \subset \mu Z.\, A[Z, \vec{B}] \times \mu Z.\, A[Z, \vec{C}]$ is taken to be $\forall Z.\, ((A[Z, \vec{\rho}] \to Z) \to Z)$, following the usual pattern. Using parametricity one has:

Theorem 9

$$\begin{array}{l}
\forall x : (\mu Z.\, A[Z, \vec{B}]) \forall y : (\mu Z.\, A[Z, \vec{C}]). \\
\quad x(\mu Z.\, A[Z, \vec{\rho}])y \\
\quad \equiv \\
\quad \forall R \subset (\mu Z.\, A[Z, \vec{B}]) \times (\mu Z.\, A[Z, \vec{C}]). \\
\qquad (\forall u : A[(\mu Z.\, A[Z, \vec{B}]), \vec{B}] \forall v : A[(\mu Z.\, A[Z, \vec{C}]), \vec{C}]. \\
\qquad\quad u A[R, \vec{\rho}] v \supset \mathit{in}(u) R\, \mathit{in}(v)) \\
\qquad \supset \\
\qquad xRy
\end{array}$$

Put more informally, this theorem states that $\mu Z.\, A[Z, \vec{\rho}]$ is the *least* relation R such that $in(A[R, \vec{\rho}]) \subset R$.

Applying the Identity Extension Lemma to this theorem we obtain the following *binary* induction principle for a relation $\rho \subset \mu Z.\, A[Z, \vec{X}] \times \mu Z.\, A[Z, \vec{X}]$:

$$\begin{array}{l} \forall u, v : A[\mu Z.\, A[Z, eq_{\vec{X}}], eq_{\vec{X}}]. \\ \quad (u A[\rho, eq_{\vec{X}}] v \supset in(u) \rho\, in(v)) \\ \quad \supset \\ \quad \forall x, y : \mu Z.\, A[Z, \vec{X}].\, x \rho y \end{array}$$

This amounts to saying that, if $in(A[\rho, eq_{\vec{X}}]) \subset \rho$ then $x\rho y$ holds for all x,y in $\mu Z.\, A[Z, \vec{X}]$. It is somewhat surprising that we do not obtain the expected *unary* induction principle, that, for any property (*i.e.* unary relation) π, if $in(A[\pi, \Delta^1_{\vec{X}}]) \subset \pi$ then $\pi(x)$ holds for all x in $\mu Z.\, A[Z, \vec{X}]$. It would be interesting to know the relation between the two, and also with the corresponding principles of other degrees.

3.5 Final Co-algebras

Let $A[Z]$ be a type where the variable Z occurs only positively. The final $A[Z]$ co-algebra is given by

$$\nu Z.\, A[Z] = \exists Z.\, ((Z \to A[Z]) \times Z)$$

and the combinators are

$$\mathit{unfold} : \forall Z.\, ((Z \to A[Z]) \to (Z \to (\nu Z.\, A[Z])))$$

and

$$\mathit{out} : \nu Z.\, A[Z] \to A[\nu Z.\, A[Z]]$$

where

$$\mathit{unfold}(Z)(f)(z) = \mathit{pack}(Z)(\langle f, z \rangle)$$

and

$$\begin{array}{rcl} \mathit{out}(u) & = & \mathit{unpack}(u)(A[\nu Z.\, A[Z]]) \\ & & (\Lambda Z \lambda w : ((Z \to A[Z]) \times Z).\, A[\mathit{unfold}\, Z(\mathit{fst}\, w)]((\mathit{fst}\, w)(\mathit{snd}\, w))) \end{array}$$

The structure $\langle \nu Z.\, A[Z], \mathit{out} \rangle$ is a weakly final co-algebra:

$$\forall Z \forall f : Z \to A[Z].\, (\mathit{unfold}(Z)(f); \mathit{out} = f; A[\mathit{unfold}(Z)(f)])$$

This can be shown without using parametricity (or any η-equalities). With parametricity one obtains a characterization of $\langle \nu Z.\, A[Z], \mathit{out} \rangle$ as the final $A[Z]$ co-algebra:

Theorem 10 $\forall Z \forall f : Z \to A[Z] \exists! h : Z \to (\nu Z.\, A[Z]).\, (h; \mathit{out} = f; A[h])$

Turning to logical relations, let $\vec{X}$ be a list of the type variables free in $A[Z, \vec{X}]$ apart from Z and let $\vec{\rho} \subset \vec{B} \times \vec{C}$ be a list of relations of the same length. Then the relation $\nu Z.\, A[Z, \vec{\rho}] \subset \nu Z.\, A[Z, \vec{B}] \times \nu Z.\, A[Z, \vec{C}]$ is taken to be $\exists Z.\, ((Z \to A[Z, \vec{\rho}]) \times Z)$, following the usual pattern.

Theorem 11

$$\begin{array}{l} \forall x : (\nu Z.\, A[Z, \vec{B}]) \forall y : (\nu Z.\, A[Z, \vec{C}]). \\ \quad x (\nu Z.\, A[Z, \vec{\rho}]) y \\ \quad \equiv \\ \quad \exists R \subset (\nu Z.\, A[Z, \vec{B}]) \times (\nu Z.\, A[Z, \vec{C}]). \\ \qquad x R y \wedge \forall x' : (\nu Z.\, A[Z, \vec{B}]) \forall y' : (\nu Z.\, A[Z, \vec{C}]). \\ \qquad\qquad x' R y' \supset \mathit{out}(x') A[R, \vec{\rho}]\, \mathit{out}(y') \end{array}$$

Put more informally,what this theorem states is that $\nu Z.\ A[Z, \vec{\rho}]$ is the *greatest* relation R such that $out(R) \subset A[R, \vec{\rho}]$.

From this theorem and the Identity Extension Lemma one obtains the following principle of co-induction (or bisimulation) [AM89, Smy91, Pit92]:

$$\begin{array}{l} \forall x, y : (\nu Z.\ A[Z]). \\ \quad (\exists R {\subset} (\nu Z.\ A[Z]) \times (\nu Z.\ A[Z]). \\ \qquad xRy \wedge \forall x', y' : (\nu Z.\ A[Z]).\ x'Ry' \supset out(x')A[R, eq_{\vec{X}}]out(y')) \\ \quad \supset \\ \quad x = y \end{array}$$

To put this in perhaps more familiar terms, say that $R \subset (\nu Z.\ A[Z]) \times (\nu Z.\ A[Z])$ is a *bisimulation* relation if $out(R) \subset A[R, eq_{\vec{X}}]$. Then the principle of co-induction states that equality is the maximal bisimulation relation. From previous work a binary principle is expected here. There are also evident principles of other degrees. The unary principle is logically trivial. Ternary and higher degree principles seem pointless; however we do not know if they are equivalent to the usual binary one.

Acknowledgments

We benefited from many conversations with Luca Cardelli, Pierre-Louis Curien and John Mitchell.

References

[ACC93] Martin Abadi, Luca Cardelli and Pierre-Louis Curien. Formal parametric polymorphism. To Appear in proceedings of *Principles of Prgramming Languages '93*.

[AM89] Peter Aczel and Nax Mendler. A final co-algebra theorem. In D. H. Pitt *et al.*, editors, *Category Theory and Computer Science* Lecture Notes in Computer Science, 389:357–365 Berlin, 1989. Springer-Verlag.

[BFSS90] E. S. Bainbridge, Peter J. Freyd, Andre Scedrov, and Philip J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70(1):35–64, January 15 1990. Corrigendum in (3) 71, 10 April 1990, p. 431.

[Böh85] Corrado Böhm and A. Berarducci. Automatic synthesis of typed Λ-programs on term algebras. *Theoretical Computer Science*, 39:85–114, 1985.

[CMMS91] Luca Cardelli, Simone Martini, John Mitchell, and Andre Scedrov. An extension of system F with subtyping. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pages 750–770, Berlin, 1991. Springer-Verlag.

[CG91] Pierre-Louis Curien and Giorgio Ghelli. Subtyping + extensionality: Confluence of $\beta\eta$top reduction in $F_{\leq}$. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pages 731–749, Berlin, 1991. Springer-Verlag.

[CG92] Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption, minimum typing and type-checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2(1):55–92, March 1992.

[Has90] Ryu Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 1990. To appear.

[Has91] Ryu Hasegawa. Parametricity of extensionally collapsed models of polymorphism and their categorical properties. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pages 495–512, Berlin, 1991. Springer-Verlag.

[MR92] QingMing Ma and John C. Reynolds. Types, abstraction, and parametric polymorphism, part 2. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. A. Schmidt, editors, *Proceedings of the 1991 Mathematical Foundations of Programming Semantics Conference*, Lecture Notes in Computer Science, Berlin, 1992. Springer-Verlag. To appear.

[Mai91] Harry Mairson. Outline of a proof theory of parametricity. In *Proc. 5th International Symp. on Functional Programming Languages and Computer Architecture*, Springer-Verlag, 1991.

[MP85] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 37–51, 1985.

[Mit90] John C. Mitchell. A type inference approach to reduction properties and semantics of polymorphic expressions (summary). In G. Huet, editor, *Logical Foundations of Functional Programming*, pages 195–212, Reading, 1990. Addison-Wesley.

[Mit91] John C. Mitchell. On the equivalence of data representations. In V. Lifshitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, Academic Press, pages 305–330, 1991.

[Pit87] Andrew M. Pitts. Polymorphism is set-theoretic, constructively. In D. H. Pitt *et al.*, editors, *Category Theory and Computer Science* Lecture Notes in Computer Science, 283:12–39 Berlin, 1987. Springer-Verlag.

[Pit92] Andrew M. Pitts. A co-induction principle for recursively defined domains. To appear in *Theoretical Computer Science.*

[Rey83] John C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Amsterdam, 1983. Elsevier Science Publishers B. V. (North-Holland).

[RP90] John C. Reynolds and Gordon D. Plotkin. On functors expressible in the polymorphic typed lambda calculus. In G. Huet, editor, *Logical Foundations of Functional Programming*, pages 127–152, Reading, 1990. Addison-Wesley.

[Smy91] Michael B. Smyth. I-categories and duality. In M. P. Fourman, P. T. Johnstone and A. M. Pitts, *Applications of Categories in Computer Science* London Mathematical Society Lecture Note Series, 177:270–287, Cambridge, 1991 Cambridge University Press.

[Str67] Christopher Strachey. Fundamental concepts in programming languages. Lecture Notes, International Summer School in Programming Languages, Copenhagen, Unpublished, August 1967.

[Wad89] Philip Wadler. Recursive types in polymorphic second-order lambda-calculus Draft, University of Glasgow 1990.

[Wra89] Gavin C. Wraith. A note on categorical datatypes. In A. M. Pitts and A. Poigné, editors, *Category Theory and Computer Science* Lecture Notes in Computer Science, 39:118–127, Berlin, 1989, Springer-Verlag.