Subtyping and Parametricity

Gordon Plotkin* Martín Abadi† Luca Cardelli†

Abstract

In this paper we study the interaction of subtyping and parametricity. We describe a logic for a programming language with parametric polymorphism and subtyping. The logic supports the formal definition and use of relational parametricity. We give two models for it, and compare it with other formal systems for the same language. In particular, we examine the "Penn interpretation" of subtyping as implicit coercion.

Without subtyping, parametricity yields, for example, an encoding of abstract types and of initial algebras, with the corresponding proof principles of simulation and induction. With subtyping, we obtain partially abstract types and certain initial order-sorted algebras, and may derive proof principles for them.

1 Introduction

A function is polymorphic if it works on inputs of several types. We may distinguish various notions of polymorphism, particularly parametric polymorphism (e.g. [Rey83]) and subtype polymorphism (e.g. [CW85]). These may exist in isolation, as in ML [MTH90] or in Amber [Car86], but they can also interact, with useful results. For example, a theory of object-oriented programming has been based on a certain kind of bounded polymorphism (e.g. [CHC90, Bru93]).

In this paper we study the interaction of subtyping and parametricity. A polymorphic function may be said to be parametric in Strachey's sense [Str67, Rey83, PA93] if it can be given by a uniform algorithm or program, independently of the type of its arguments. A semantic definition of parametricity is due to Reynolds [Rey83], who requires instead that instances of the polymorphic function at related types be related. Reynolds' definition has been formalized in previous work [ACC93, PA93]. In this paper we extend the formalization of [PA93] to a programming language with subtyping.

A logic serves as the setting for this study. This logic can be viewed as an analogue of Scott's LCF, that is, as a fairly general system for proving properties of programs. Here the programs are those of System $F_{\leq}$, which is an extension of Girard's System F [Gir72] with subtyping, abstracted from work of Cardelli and Wegner [CW85] by Curien and Ghelli [CG92, CG94]. Our logic for $F_{\leq}$ is an extension of the logic for F presented in [PA93]. Beyond its possible use in program verification, the logic provides a language for stating parametricity assumptions and rules for deriving their consequences, formally and without reference to particular models.

While it remains to consider what might be the appropriate general form for parametric models of $F_{\leq}$ and of our logic, we do construct particular models—indeed two such. The first is a parametric per model combining the idea of Bruce and Longo [BL90] of treating subtypes as subpers with that of Bainbridge et al. [BFSS90] of forcing parametricity into per models of System F. The second is a closed-term model, following an idea of Moggi for System F [Mog86]. Having at least one non-trivial model, it follows that if two terms of the same type can be proved equal in our logic, then they are observationally equivalent.

A variant $F_{<:}$ of $F_{\leq}$ was given by Cardelli et al. [CMMS94]. A weakened version is derivable within our logic. Both this version and the full $F_{<:}$ yield some of the results associated with parametricity, frequently with a limitation to closed terms. Our logic gives these and other results in full generality, for terms with free variables. We conjecture that in fact $F_{<:}$ itself is derivable within our logic. Indeed we formulate a stronger theory, which may be said to embody Strachey's view of parametric polymorphism for $F_{\leq}$, and conjecture that it is derivable.

We also examine the "Penn interpretation" of $F_{\leq}$ [BCGS91], with its view of subtyping as implicit coercion. This interpretation is based on a translation from $F_{\leq}$ to F. We show that this translation can be extended to formulae; theorems of the logic

*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.

for $F_{\leq}$ are translated into theorems of the logic for F given in [PA93]. We consider full-abstraction issues and show that the translation is not conservative.

Parametricity conditions play an important role in the study of F and of similar languages (e.g. [Rey83, BFSS90, Wad89]). They appear in semantic constructions. They yield useful properties of types, for example that $Int = \forall X. ((X \to X) \to (X \to X))$, the type of Church integers, is isomorphic to the standard natural numbers. And they can be exploited in proving properties of polymorphic programs, for example that all functions of type $\forall X. (X \to Int)$ are constant. These results have interesting analogues for $F_{\leq}$. Just as the logic for F offers abstract types, initial algebras, and final co-algebras, the logic for $F_{\leq}$ offers partially abstract types, certain initial order-sorted algebras, and certain final order-sorted co-algebras, with corresponding proof principles. Further, we can apply the logic to prove theorems about programs from their types, or "theorems for free," as Wadler calls them. Some of these have an object-oriented flavor, in line with one of the intended applications of $F_{\leq}$.

There has been much related work for languages without subtyping. However, the combination of parametricity and subtyping has been little considered. As mentioned above, System $F_{<:}$ of Cardelli et al. incorporates a modest notion of parametricity (partly motivated by dinaturality considerations). Ma [Ma92] expresses parametricity for F via a translation into a language with subtyping and intersection types. Ma focuses on parametricity in F, not on parametricity in his target language with subtyping.

The next section introduces our logic and some fundamental results about it. Discussion of its semantics appears in section 3. Section 4 treats other theories for $F_{\leq}$, including one induced by the Penn interpretation. Section 5 provides encodings of extensible records, partially abstract types, and order-sorted algebras.

2 Basic logic

This section defines the logic. In this paper we sometimes reference or borrow from [PA93] for the fragment that corresponds to F. We emphasize the novelties, which concern subtyping.

2.1 Well-formed formulae

The type expressions and terms are those of $F_{\leq}$. The type expressions of $F_{\leq}$ are like those of F, with the addition of a largest type Top and a generalization from quantifiers to bounded quantifiers. The terms are extended similarly, with a constant (top) and bounded type abstractions. *Type expressions* and *terms* are given by the grammar:

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

Here X ranges over *type variables* and 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). Unbounded binders abbreviate the corresponding binders with bound Top; so for example $\forall X.A$ stands for $\forall X{\leq}\mathrm{Top}.A$. Throughout, expressions are understood up to α-equivalence.

We build *formulae* from equations and binary relations between terms.

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

Here R ranges over *relation variables*. The equality symbol is subscripted with a type expression, the type of the terms being equated. In F, this expression is unique, and so can be left implicit, but it proves necessary in treating subtyping, as we see below. The basic constructs are implication ($\supset$) and three sorts of universal quantification: 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. When writing formulae we often make use of evident abbreviations. While there are primitive notions of subtype and of bounded type quantification, there is no need for a corresponding primitive notion for relations.

A *second-order environment* E is a finite sequence of type variables with bounds $X \leq A$ or typings $x{:}A$ in which no variable is introduced twice. The typing judgment $E \vdash t{:}A$ and the subtyping judgment $E \vdash A{\leq}B$ are defined as in [CG92, CG94].

To specify the well-formed formulae, we also need *relation environments*, which are finite sequences of *relational typings* $R \subset A \times B$ with no relation variable repeated. We define a judgment $E \vdash G$ REnv to assert that G is a well-formed relation environment given E; the judgment holds if whenever $R \subset A \times B$ appears in G then A and B are well-formed type expressions given E. We define a judgment $E; G \vdash \phi$ Prop to assert that ϕ is a well-formed formula given E and 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:

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

2.2 Relational formulae

Next we introduce *relational formulae*. They are given by the grammar:

$$\text{Relational formulae:} \quad \rho ::= \quad (x\!:\!A, y\!:\!B).\, \phi[x,y]$$

We say that such a ρ is a relational formula between A and B, and write $\rho \subset A \times B$. We write $E;G \vdash \rho \subset A \times B$ for the judgment that ρ is a well-formed relational formula between A and B given E and G. There is one rule for this judgment:

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

For example, $eq_A = (x:A, y:A).\,(x =_A y)$ is a relational formula denoting the equality relation over A. With subtyping, useful, new relations become available, for example a variant of the equality relation is denoted by $(x\!:\!A, y\!:\!B).\,(x =_B y) \subset A \times B$; this is well formed in any environment where $A{\leq}B$.

We sometimes treat a relation variable $R \subset A \times B$ as the relational formula $(x:A, y:B).\; R(x,y)$. Also, when ρ is $(x:A, y:B).\; \phi[x,y]$, we sometimes use the abbreviations $\rho(t,u)$ or $t\rho u$ for $\phi[t,u]$. A relational formula ρ can be substituted for a relation variable R in a formula $\phi[R]$, yielding $\phi[\rho]$. In particular when ϕ is $R(t,u)$, the result of the substitution is $\rho(t,u)$.

2.3 Operations on relations

In order to give our axiomatization of parametricity, we need to be able to combine relations by exponentiation and bounded universal quantification.

For $\rho \subset A \times B$ and $\rho' \subset A' \times B'$, we define $(\rho \to \rho') \subset (A \to A') \times (B \to B')$ to be:

$$\begin{array}{l}(f\!:\!A \to A', g\!:\!B \to B').\\ \qquad \forall x\!:\!A\; \forall y\!:\!B.\, (x\rho y \supset f(x)\rho' g(y))\end{array}$$

If $E;G \vdash \rho \subset A \times B$ and $E;G \vdash \rho' \subset A' \times B'$ then $E;G \vdash (\rho \to \rho') \subset (A \to A') \times (B \to B')$.

Next, for $\rho \subset C \times D$ and $\rho' \subset A \times B$, we define $(\forall(Y{\leq}C, Z{\leq}D, R{\leq}\rho).\, \rho') \subset (\forall Y{\leq}C.\, A) \times (\forall Z{\leq}D.\, B)$ to be:

$$\begin{array}{l}(y\!:\!(\forall Y{\leq}C.\, A), z\!:\!(\forall Z{\leq}D.\, B)).\\ \qquad \forall Y{\leq}C \forall Z{\leq}D \forall R{\subset}Y{\times}Z.\, (R \leq \rho \supset (yY)\rho'(zZ))\end{array}$$

where $\rho_1 \leq \rho_2$ stands for $\forall x : C_1 \forall y : D_1.\, (\rho_1(x,y) \supset \rho_2(x,y))$, for $\rho_1 \subset C_1 \times D_1$ and $\rho_2 \subset C_2 \times D_2$. Suppose that $E;G \vdash \rho \subset C \times D$ and $E' \vdash \rho' \subset A \times B$, where E' is $E, Y \leq C, Z \leq D; G, R \subset Y \times Z$. Then $E;G \vdash (\forall(Y{\leq}C, Z{\leq}D, R{\leq}\rho).\, \rho') \subset (\forall Y{\leq}C.\, A) \times (\forall Z{\leq}D.\, B)$.

We can now abbreviate relational formulae by type expressions with a certain substitution of relational formulae for their free variables. If $\vec{X} = X_1, \ldots, X_n$, $\vec{B} = B_1, \ldots, B_n$, $\vec{C} = C_1, \ldots, C_n$, and $\vec{\rho} = \rho_1, \ldots, \rho_n$ with $\rho_i \subset B_i \times C_i$, then $A[\vec{\rho}] \subset A[\vec{B}] \times A[\vec{C}]$ is the result of substituting $\vec{\rho}$ for $\vec{X}$ in $A[\vec{X}]$. It is defined by cases:

- if A is X_i then $A[\vec{\rho}]$ is ρ_i;
- if A is $A'[\vec{X}] \to A''[\vec{X}]$ then $A[\vec{\rho}]$ is $A'[\vec{\rho}] \to A''[\vec{\rho}]$;
- if A is Top then $A[\vec{\rho}]$ is eq_{Top};
- lastly, if A is $\forall X'{\leq}D[\vec{X}].\, A'[\vec{X}, X']$ then $A[\vec{\rho}]$ is $\forall(Y{\leq}D[\vec{B}], Z{\leq}D[\vec{C}], R{\leq}D[\vec{\rho}]).\, A'[\vec{\rho}, R]$.

If $E;G \vdash \rho_i \subset B_i \times C_i$ then $E;G \vdash A[\vec{\rho}] \subset A[\vec{B}] \times A[\vec{C}]$.

For example if $A[X]$ is $\forall X'{\leq}X.\, X'$ then $A[eq_{Int}]$ is $\forall(Y{\leq}Int, Z{\leq}Int, R{\leq}eq_{Int}).\; R$. The definition applies when A is closed, when we write $A[]$ for the relational formula obtained. For example $(\forall X'{\leq}\mathrm{Top}.\, X')[]$ is $\forall(Y{\leq}\mathrm{Top}, Z{\leq}\mathrm{Top}, R{\leq}eq_{\mathrm{Top}}).\; R$.

2.4 Consequence

It remains to give axiom schemas 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 and G. The proof system has three parts: standard rules for the connectives and quantifiers; equational axioms (corresponding to the equational system of [CG94]); and a schema to express relational parametricity. We adopt the convention that if an axiom ϕ is written, what is meant is that the sequent $\Gamma \vdash_{E;G} \phi$ is asserted, provided ϕ and all formulae in Γ are well-formed given E and G.

The rules for the connectives and quantifiers are given as usual for natural deduction. Propositional logic is standard; intuitionistic rules suffice for our purposes, but classical rules are consistent as well. The rules for predicate logic consist of introduction and elimination rules for each of the quantifiers, such as:

$$\frac{\Gamma \vdash_{E,X\leq B;G} \phi[X]}{\Gamma \vdash_{E;G} \forall X{\leq}B.\, \phi[X]}$$

$$\frac{\Gamma \vdash_{E;G} \forall X{\leq}B.\, \phi[X] \quad E \vdash A \leq B}{\Gamma \vdash_{E;G} \phi[A]}$$

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

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

with the usual provisions about variable occurrences.

The axioms for equality include a reflexivity axiom, a substitution axiom, two congruence schemas, and some β-equalities and η-equalities:

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

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

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

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

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

$$\forall X{\leq}A.\, ((\Lambda X{\leq}A.\, t)X =_B t)$$

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

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

$$\forall x, y{:}\mathrm{Top}.\, (x =_{\mathrm{Top}} y)$$

Parametricity is embodied by an axiom schema:

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

where A has free type variables among $X, Y_1, \ldots, Y_n$ and B has free type variables among $Y_1, \ldots, Y_n$ and $eq_{\vec{Y}}$ is $eq_{Y_1}, \ldots, eq_{Y_n}$. To understand this, it is convenient to expand the definition, obtaining that if $X' \leq B[\vec{Y}]$, $X'' \leq B[\vec{Y}]$, $R \subset X' \times X''$, $R \leq B[eq_{\vec{Y}}]$, and $u : (\forall X{\leq}B.\, A[\vec{Y}])$, then $u(X')A[eq_{\vec{Y}}, R]u(X'')$. Thus, if one instantiates a polymorphic value u at two related types X' and X'' then the two values obtained $u(X')$ and $u(X'')$ are themselves related. This statement expresses Reynolds' idea of relational parametricity. It is adapted to a calculus with subtyping by constraining X' and X'' to be subtypes of $B[\vec{Y}]$ and the relation R between X' and X'' to be included in $B[eq_{\vec{Y}}]$ (which is provably the identity relation on B—see Lemma 1).

Note that $\forall X \forall x, y{:}X.\, (x =_{\mathrm{Top}} y)$ is provable, while $\forall X \forall x, y{:}X.\, (x =_X y)$ is false in any nontrivial model. This explains why the equality relation is indexed by a type.

2.5 Basic lemmas

The basic provable schemas within our logic are given by the Identity Extension Lemma, the Logical Relations Lemma, the Dinaturality Lemma, and the Graph Lemma, following the lines of [PA93].

Lemma 1 (Identity Extension Lemma)
Let $A[\vec{X}]$ have free variables in $\vec{X}$. It is provable that

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

The following simple version of the Logical Relations Lemma implies a more general one, which does not require B and t to be closed:

Lemma 2 (Logical Relations Lemma)
Suppose $t : B$ where B and t are closed. Then it is provable without using the parametricity schema that $t\, B[\,]\, t$.

Types and the functions between them form a category within our logic, as in [PA93]. In extending types $A[\vec{Y}, \vec{X}]$ to multivariant functors we impose not only that all occurrences of variables from $\vec{Y}$ are negative and all occurrences of variables from $\vec{X}$ positive, but also that none occur free in any bound; proceeding further than this presents a challenge. (In a type expression $\forall Z{\leq}C.\, B$ the bound C is considered anti-monotonic and the body B monotonic.)

Lemma 3 (Dinaturality Lemma)
It is provable that:

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

where in $A[Y, X]$, Y occurs only negatively, X only positively, and neither are free in any bound.

In order to state the Graph Lemma, we write $\langle t\rangle_{A,B}$ for $(x{:}A, y{:}B).\, tx =_B y$ (with x, y not free in t), write ρ^{op} for $(x : A, y : B).\, y\rho x$ (where $\rho \subset B \times A$ and x, y are not free in t) and consider two relations as equal if they coincide extensionally:

Lemma 4 (Graph Lemma)
Suppose $A[\vec{Y}, \vec{X}]$ has all its free variables in $\vec{Y}, \vec{X}$, the variables in $\vec{Y}$ occur only negatively, the variables in $\vec{X}$ only positively, and none are free in any bound. Then, for distinct $\vec{Y}, \vec{X}, \vec{Y'}, \vec{X'}$, it is provable that:

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

3 Semantics

The categorical semantics of parametric models of F has been investigated by Hasegawa and by Reynolds and Ma [Has94, MR92]. The categorical structures needed for models of $F_{\leq}$ have been investigated by Phoa [Pho92]. It remains to combine these investigations to provide a general categorical semantics of parametric models of $F_{\leq}$.

Fortunately, we do not need a general notion to consider particular models. We give ad hoc presentations of two models, one of partial equivalence relations and another based on closed terms.

For the first model, fix a partial combinatory algebra and take the *types* of the model to be the *partial equivalence relations (pers)*; these are the symmetric and transitive relations (over the algebra). A triple (R, P, Q) is a *($\neg\neg$-closed) relation* between pers P and Q iff R is a binary relation and $R = P;R;Q$. We set $dom(R, P, Q) = P$ and $cod(R, P, Q) = Q$; we say that (R, P, Q) is a *subrelation* of (R', P', Q') and write $(R, P, Q) \subseteq (R', P', Q')$ iff $R \subseteq R'$, $P \subseteq P'$, and $Q \subseteq Q'$; and we write $\widehat{P}$ for the *identity* relation (P, P, P) on a per P.

Type expressions receive a double interpretation. The first interpretation assigns to every type expression A a per $\mathcal{T}[\![A]\!]_\eta$ (for η a *type environment*, mapping type variables to pers). The second interpretation assigns a relation $\mathcal{R}[\![A]\!]_\theta$ between $\mathcal{T}[\![A]\!]_{dom\circ\theta}$ and $\mathcal{T}[\![A]\!]_{cod\circ\theta}$ (for θ a *(semantic) relation environment*, mapping relation variables to relations between pers). The type interpretation of universal quantification is: $a(\mathcal{T}[\![\forall X{\leq}B.\,A]\!]_\eta)a'$ iff (i) $a(\mathcal{T}[\![A]\!]_{\eta\{P/X\}})a'$ for all $P \subseteq \mathcal{T}[\![B]\!]_\eta$, and (ii) $a(\mathcal{R}[\![A]\!]_{\hat{\eta}\{(R,P,Q)/X\}})a$ and $a'(\mathcal{R}[\![A]\!]_{\hat{\eta}\{(R,P,Q)/X\}})a'$ for all $(R, P, Q) \subseteq \mathcal{R}[\![B]\!]_{\hat{\eta}}$, where $\hat{\eta}(Y) = \widehat{\eta(Y)}$. For relations, $\mathcal{R}[\![\forall X{\leq}B.\,A]\!]_\theta$ is the triple (R, P, Q) where P is $\mathcal{T}[\![\forall X{\leq}B.\,A]\!]_{dom\circ\theta}$, Q is $\mathcal{T}[\![\forall X{\leq}B.\,A]\!]_{cod\circ\theta}$, and aRa' iff (i) aPa and $a'Qa'$, and (ii) $a(\mathcal{R}[\![A]\!]_{\theta\{(R',P',Q')/X\}})a'$ for all $(R', P', Q') \subseteq \mathcal{R}[\![B]\!]_\theta$. Proposition 1 relates the two interpretations:

Proposition 1 *For all type expressions A and all type environments η, $\mathcal{R}[\![A]\!]_{\hat{\eta}} = \widehat{\mathcal{T}[\![A]\!]_\eta}$.*

An interpretation of the logic extends the interpretation of $F_{\leq}$. Formulae are interpreted classically, with type variables ranging over pers, ordinary variables over elements of the domain of the appropriate per, and relation variables over relations between the appropriate pers. With this, a relational formula $\rho \subset A \times B$ can be interpreted as a relation between the pers denoted by A and B. It is straightforward to validate all the axioms and rules of inference, except that the axiom of parametricity needs some work. For this Proposition 1 applies. One also needs a "semantic substitution lemma" to the effect that the relation defined by the substitution of relational formulae in a type expression is the same as the relational semantics of the type expression in the relation environment induced by the relational formulae being substituted.

For the second model, we follow the construction of a closed-term model of System F by Moggi [Mog86]. We work with a natural contextual (or observational) equivalence relation $\simeq_A$, indexed by closed type expressions A. Here, for any closed $F_{\leq}$ terms t and u of type A, $t \simeq_A u$ holds iff for every closed term c of type $(A \to Int)$, $\vdash ct = \underline{0} : Int$ holds in the equational system of [CG94] iff $\vdash cu = \underline{0} : Int$ does.

We take the *types* (of the model) to be the closed type expressions. For any such type A, set $Terms_A = \{t \mid \vdash t : A\}$ and say a *relation* between two types A and B is a relation between $Terms_A$ and $Terms_B$ that is closed under $\simeq$. Type and relation environments are defined as before; *(ordinary) environments* ν are taken to be maps from (ordinary) variables to closed terms. Type and ordinary environments are extended to type expressions and terms by substitution. For any E and G such that $E \vdash G$ REnv, we take the judgment $\eta, \nu, \theta \models E; G$ to hold iff $\vdash \nu(x) : \eta(A)$ for each $x : A$ in E, and $\vdash \eta(X) {\leq} \eta(A)$ for each $X {\leq} A$ in E, and $\theta(R)$ is a relation between $\eta(A)$ and $\eta(B)$ for each $R {\subset} A {\times} B$ in G.

Next, for any E, G, and ϕ such that $E; G \vdash \phi$ Prop and $\eta, \nu, \theta \models E; G$, we define a *satisfaction* judgment $E; G \models_{\eta,\nu,\theta} \phi$ by induction on the structure of ϕ. In particular we set $E; G \models_{\eta,\nu,\theta} t =_A u$ iff $\nu(t) \simeq_A \nu(u)$. It follows from the following lemma that all the equality axioms are *valid* (in the now evident sense):

Lemma 5 *1. Let B and C be closed type expressions and let t and u be closed terms of type $B \to C$. Then $t \simeq_{B\to C} u$ iff $tv \simeq_C uv$ for all closed terms v of type B.*

2. [Ghe90] Let t and u be closed terms of type Top. *Then $t \simeq_{\mathrm{Top}} u$.*

3. Let $\forall X{\leq}B.\,A[X]$ be a closed type expression and let t and u be closed terms of that type. Then $t \simeq_{\forall X\leq B.A[X]} u$ iff $tC \simeq_{A[C]} uC$ for all closed type expressions C with C a subtype of B.

As the rules of the logic are also valid, the Logical Relations Lemma holds in this interpretation. With that and a semantic substitution lemma, one can verify that the parametricity schema is valid (essentially because all elements of types are definable). Thus we have a second model of our logic.

4 On other systems for $\mathbf{F}_{\leq}$

In this section we relate our logic to other systems for $F_{\leq}$. These are: the equational system of [CG94]; the equational system $F_{\leq:}$ of [CMMS94]; an equational schema which expresses Strachey's view of parametric polymorphism (in the context of $F_{\leq}!$); and the system obtained by combining the Penn interpretation with parametricity assumptions.

4.1 Equations for $\mathbf{F}_{\leq}$

The equational system of [CG94] corresponds to the equational fragment of our logic, less parametricity. Writing $E \vdash t = u : A$ for provability in this system, we have:

Proposition 2 *$E \vdash t = u : A$ iff $\vdash_{E;} t =_A u$ is provable without using the parametricity schema.*

Using the parametricity schema, we can derive a weakening of the equational system $F_{\leq:}$. The difference between $F_{\leq:}$ and $F_{\leq}$ concerns the rule $(Eq\,appl2)$ of [CMMS94], which in the context of $F_{\leq}$ is equivalent to the equation:

$$\begin{array}{c}\lambda x:(\forall X{\leq}A.\ B[X]).\ xA' \\ =_{(\forall X\leq A.\ B[X])\to C} \\ \lambda x:(\forall X{\leq}A.\ B[X]).\ xA''\end{array}$$

assuming an environment E where $A', A'' \leq A$ and $B[A'], B[A''] \leq C$ hold. The weakening contains instead the rule $(Eq\,appl2^{-+})$ of [CMMS94], which in the context of $F_{\leq}$ is equivalent to the equation:

$$\begin{array}{c}\lambda x:(\forall X{\leq}A.\ B[X,X]).\ xA' \\ =_{(\forall X\leq A.\ B[X,X])\to B[A',A'']} \\ \lambda x:(\forall X{\leq}A.\ B[X,X]).\ xA''\end{array}$$

where $B[X^-, X^+]$ is a type expression in which X^- occurs only negatively and X^+ occurs only positively, and assuming an environment E where $A' \leq A'' \leq A$ holds. The variant system with the rule $(Eq\,appl2^{-+})$ suffices for the results of [CMMS94]. Derivability in this system is written as $E \vdash^{-+} t = u : A$.

Proposition 3 *If $E \vdash^{-+} t = u:A$ then $\vdash_{E;} t =_A u$.*

The only difficulty in the proof of this result is in the derivation of $(Eq\,appl2^{-+})$ in our logic. To show this, take $=_{A',A''}$ to be the relational formula $(x : A', y : A'').\ (x =_{A''} y)$. Then we have that $=_{A',A''} \leq eq_A$, and, by parametricity, for any x in $\forall X{\leq}A.\ B[X,X]$, $(xA')B[=_{A',A''}, =_{A',A''}](xA'')$. But $eq_{A'} \leq (=_{A',A''}) \leq eq_{A''}$; so we may use the facts that if X occurs only positively in a type $C[X]$ and $\rho \leq \rho'$ then $C[\rho] \leq C[\rho']$, and similarly in the negative case, to get $(xA')B[eq_{A'}, eq_{A''}](xA'')$. The result then follows by the Identity Extension Lemma.

Note that this proof is given in the usual informal mathematical style rather than presented formally within the logic; however, a formal version can easily be given. We proceed similarly with other arguments.

The equation for $(Eq\,appl2)$ is an instance of a more general schema which asserts all equations $t =_A u$ in which t and u have the same type erasures, assuming an environment E in which t and u have the same type A. (The *type erasure* of a term is the term of the untyped λ-calculus obtained from it by removing all type expressions in λ-abstractions, and all type abstractions and applications.) We conjecture that this schema is derivable in our logic. In [ACC93, PA93] similar conjectures were made for System F and the corresponding equational schema was argued to express Strachey's view of parametric polymorphism.

4.2 The Penn interpretation

In [BCGS91], Breazu-Tannen et al. describe a translation of an extension of $F_{\leq}$ to an extension of F (the Penn interpretation) and prove a coherence result. This work straightforwardly restricts to a translation of $F_{\leq}$ to F extended with a type Top. Replacing Top with $\forall X.\ (X \to X)$, we obtain a translation to F. Each type expression A is mapped to an F type expression $A^\star$, for example $(\forall X{\leq}B.\ A)^\star$ is $\forall X.\ ((X \to B^\star) \to A^\star)$, showing how subtyping is modeled by—arbitrary—coercion. Next, each environment E is mapped to an F environment $E^\star$, and for each proof Π of a typing $E \vdash t : A$ one obtains an F term t_Π and an F proof of $E^\star \vdash t_\Pi : A^\star$. According to the coherence result, t_Π is independent of Π up to provable equality, and we can write $t^\star$ rather than t_Π. We omit the definitions.

We now extend this translation, mapping our logic for $F_{\leq}$ to the logic for F of [PA93]. To translate relation environments, we replace each declaration $R \subset A \times B$ by $R \subset A^\star \times B^\star$. To each provable sequent $E; G \vdash \phi$ Prop we associate a sequent $E^\star; G^\star \vdash \phi^\star$ Prop by induction on the structure of ϕ. Here we just give two cases. If ϕ is $t =_A u$ then the translation is $E^\star; G^\star \vdash t^\star =_{A^\star} u^\star$ Prop where $E^\star \vdash t^\star : A^\star$ is the translation of $E \vdash t : A$ and similarly for u. If ϕ is $\forall X{\leq}B.\ \psi$ then the translation is $E^\star; G^\star \vdash \forall X \forall f : X \to B^\star.\ \psi^\star$ Prop where $E^\star, X, f : X \to B^\star; G^\star \vdash \psi^\star$ Prop is the translation of $E, X \leq B \vdash \psi$ Prop.

Theorem 1 (Translation Theorem)
Suppose that the translation of $E; G \vdash \phi$ Prop *is* $E^\star; G^\star \vdash \phi^\star$ Prop*. If* $\vdash_{E;G} \phi$ *is provable in the logic for* $F_\leq$ *then* $\vdash_{E^\star;G^\star} \phi^\star$ *is provable in the logic for F.*

Not everything translates well, however. The translation reflects but does not preserve the contextual equivalence relation $\simeq$ (defined for F as for $F_\leq$). It is simple to prove reflection: that $t^\star \simeq_{A^\star} u^\star$ implies $t \simeq_A u$. As to the failure of preservation, set B to be $\forall Y. ((\forall X{\leq}Y. X) \to (\forall X.(X \to Y) \to X))$. There are no closed terms of type B, but there is a closed term of type $B^\star$. Now let A be $B \to Int$, t be $\lambda x\!:\!B.\underline{0}$, and u be $\lambda x\!:\!B.\underline{1}$. Lemma 5 yields $t \simeq_A u$. On the other hand, it is easy to see that for some term c, $c(t^\star)$ is $\beta\eta$-equivalent to $\underline{0}$ and $c(u^\star)$ to $\underline{1}$ and so $t^\star \not\simeq_{A^\star} u^\star$. This can be viewed as a failure of full abstraction for the Penn interpretation.

In fact, $t^\star$ and $u^\star$ will have different denotations in any non-trivial model of F, so any model of $F_\leq$ defined by factoring through the Penn interpretation will not be fully abstract. (Answering a question of Breazu-Tannen, t and u also receive different interpretations in the parametric per model outlined in section 3 as there B has the same denotations as Top; so that model is not fully abstract either.)

Further, the translation of the logic is not conservative. Specifically, take ϕ to be $(t =_A u \supset \underline{0} =_{Int} \underline{1})$. Then $\phi^\star$ is provable in the logic for F (even without parametricity) but ϕ is not provable in the logic for $F_\leq$ as it is false in the closed-term model given above. We do not know whether the translation is conservative for equations.

It is not clear how seriously one should take these inadequacies of the Penn interpretation. After all, as we show, the logic for $F_\leq$ is powerful in that it supplies all the usual reasoning principles one might expect, and the Translation Theorem implies that the logic for F is powerful too. On the other hand, it is uncomfortable that via the translation one can prove false statements (in a certain sense) and it would be interesting to have a principled extension of the logic for $F_\leq$ that would refute statements like ϕ.

5 Datatypes

Finite products and sums, existentials, initial algebras, and final co-algebras can be treated without subtyping; see [PA93] for details. Now, in addition, extensible record and variant types, bounded existentials, and order-sorted algebras become available. That is, they can be represented as $F_\leq$ types, and the logic enables us to prove that these $F_\leq$ types have certain expected properties, for example that two extensible records of a type A are equal if they agree on the fields declared in A.

5.1 Extensible records

Extensible record types are treated as in [Car92]. One fixes an ordered, countably infinite list of names l_i ($i = 1, 2, \ldots$) and takes the record type $\Pi^*_{l \in L} A_l$ (where L is a finite set of names) to be $\Pi_{i \leq n+1} B_i$ where n is the greatest index of any element of L, $B_i = A_{l_i}$ (if $l_i \in L$) and $=$ Top (otherwise). This type is a finite product, categorically.

One can define *extensible sums* analogously. If one has available a least type Bot (necessarily the initial type) one can set $\Sigma^*_{l \in L} A_l = \Sigma_{i \leq n+1} B_i$ with n as above and $B_i = A_{l_i}$ (if $l_i \in L$) and $=$ Bot (otherwise). This yields a categorical sum. One can get the same effect without Bot by taking $B_i = \forall X. ((A_{l_i} \to X) \to X)$ (if $l_i \in L$; X not in A_{l_i}) and $= \forall X. (\mathrm{Top} \to X)$ (otherwise).

Records and bounded quantification have been used in combination to model some aspects of object-oriented programming. Parametricity is useful in understanding the issues involved in this approach to objects. The first example considered seems to have been a simple one concerning the type *Point* of extensible records with integer fields x and y. (Informally, we write *Point* as $\{x, y : Int\}$, and think of points as objects.) The type $\forall P{\leq}Point. (P \to P)$ was intended as the type of a program that modifies the x and y components of an element of an arbitrary subtype P of *Point*, that is, the type of a program that "moves a point" parametrically for any subtype of *Point*. However, Mitchell pointed out that in a per model this type contains only the identity function, hence no value of this type can "move" anything in a per model. This can be verified in general in our logic for $F_\leq$: if $f : \forall X{\leq}A. (X \to X)$ then $\forall X{\leq}A \forall x\!:\!X. (f(X)(x) =_X x)$. To show this, one considers a type X and an element x in X, and applies the parametricity scheme for $\forall X{\leq}A. (X \to X)$ to the relational formula $(y\!:\!X, z\!:\!X). y =_X z =_X x$.

What should then be the type of a parametric *move* function? One solution is to use a richer notion of extensible records. As in [Car92], we write $Z{\uparrow}\{x, y\}$ to mean that Z is a "record extension" that does not contain the labels x and y, so that $\{x, y\!:\!Int,\, Z\}$ is a well-formed record type. Then we take:

$$move : \forall Z{\uparrow}\{x, y\}. (\{x, y\!:\!Int,\, Z\} \to \{x, y\!:\!Int,\, Z\})$$

that is, the move function takes any extension of the record type $\{x, y : Int\}$ with Z and returns a similar extension, possibly modifying x and y. These extensible records can be encoded in $F_{\leq}$ as shown in [Car92], and so our logic applies to them as well. The encoding depends on an enumeration of labels. If we assume that x and y occur, say, first and third in the enumeration, then the type of the *move* function under the encoding is:

$$\forall X_2 \forall X_4.((Int \times X_2 \times Int \times X_4) \to (Int \times X_2 \times Int \times X_4))$$

Now, using parametricity, it is easy to show that $\forall X.\, (X \to X)$ is isomorphic to Top, and also that $\forall X.\, (X \to Y)$ is isomorphic to Y. From this we may deduce that $\forall X.\, ((X \times Y) \to (X \times Z))$ is isomorphic to $Y \to Z$. It follows that the type of *move* is isomorphic to $(Int \times Int) \to (Int \times Int)$, and hence can indeed contain a genuine move function.

5.2 Partially abstract types

Just as existential types model abstract types [MP85], bounded existential types model a corresponding programming construct: partially abstract types [CW85]. A partially abstract type is a type whose representation is left unspecified, but whose properties are partially known by virtue of it being a subtype of a known type. Partially abstract types are a significant feature of some object-oriented languages that support abstraction [Wir88, Nel91].

Formally, *bounded existentials* can be defined from bounded universals:

$$\exists X \leq B.\, A[X] = \forall Y.\, ((\forall X \leq B.\, (A[X] \to Y)) \to Y)$$

Combinators *pack* and *unpack* are available:

$$\begin{aligned} &pack : \forall X \leq B.\, (A[X] \to \exists X \leq B.\, A[X]) \\ &unpack : (\exists X \leq B.\, A[X]) \\ &\qquad\qquad \to \forall Y.\, ((\forall X \leq B.\, (A[X] \to Y)) \to Y) \end{aligned}$$

with $packXxYf =_Y fXx$ for any $X \leq B$, $x : A[X]$, and $f : \forall X \leq B.\, (A[X] \to Y)$, and *unpack* given by the identity. We have: $unpack(packXx)Yf =_Y fXx$. One has a categorical characterization: for any function $f : \forall X \leq B.\, (A[X] \to Y)$ there is a unique function $g : (\exists X \leq B.\, A[X]) \to Y$ such that for any $X \leq B$ and $x : A[X]$, $fXx =_Y g \circ (packXx)$. One can also show how the bounded existential operates on relations. A *bounded simulation principle* can then be derived. It is a rule for proving equalities between elements of partially abstract types: omitting parameters and types on equalities, for any $u, v : \exists X \leq B.\, A[X]$, $u = v$ holds if

$$\begin{aligned} &\exists X \leq B, Y \leq B\ \exists x : A[X], y : A[Y]\ \exists S \subset X \times Y. \\ &\quad S \leq eq_B \wedge u = packXx \wedge v = packYy \wedge xA[S]y \end{aligned}$$

This rule yields representation-independence theorems for partially abstract types. For example, the type $\exists X \leq Point.\, (X \times (Point \to X))$ the type of a package providing an element of an unknown subtype X of $Point$ and a function from $Point$ to X, is isomorphic to the much less intriguing type $\exists X \leq Point.\, (Point \times (Point \to Point))$, and in turn to $Point \times (Point \to Point)$, the type of a pair of a $Point$ and a function over $Point$.

More generally, one can replace occurrences of an existentially quantified variable with its bound in a package interface, provided all its occurrences in the interface are positive, and none occur freely in a bound. This is a consequence of the corresponding statement for types $\forall X \leq B.A[X]$ with universal quantifiers and negative occurrences. For such a type, $(\cdot)_B : (\forall X \leq B.\, A[X]) \to A[B]$ is an isomorphism with inverse $g = \lambda y : A[B] \Lambda X \leq B.\, A[\iota](y)$, where ι is the type inclusion $(\lambda x : X.\, x) : X \to B$. That $g \circ (\cdot)_B$ is the identity follows from $\forall z : (\forall X \leq B.\, A[X]).\, zX =_X A[\iota](zB)$ which is proved using parametricity with the relation $\langle \iota \rangle^{op}_{X,B}$ and then the Graph Lemma. That $(\cdot)_B \circ g$ is the identity is proved by equational reasoning.

5.3 Order-sorted algebras

Initial algebras and final co-algebras can be handled without subtyping, so for example the initial $A[X]$-algebra is $\forall X.\, ((A[X] \to X) \to X)$. One might also imagine bounded initial algebras, setting I to be $\mu X \leq B.\, A[X]$ when $A[B] \leq B$. One would like $I \leq B$ to hold; however the obvious attempt $\forall X \leq B.\, ((A[X] \to X) \to X)$ does not work.

One can construct a variety of initial order-sorted algebras [GM92] and final order-sorted co-algebras. Let L be a finite partial order over the set of names (recall section 5.1); for each $l \in L$ let $A_l[\vec{X}]$ be a type expression with $\vec{X}$ a vector of $|L|$ variables all occurring only positively in A_l and not in any bound. Then a (formal) *order-sorted* $\vec{A}$*-algebra* $((B_l)_{l \in L}, (g_l)_{l \in L})$ is a collection of types B_l $(l \in L)$ such that $B_m \leq B_l$ if $m \leq l$—the *carriers* of the algebra—and functions $g_l : A_l[\vec{B}] \to B_l$—the *operations* of the algebra. For example, one might have types P and N, with $P \leq N$, and operations $succ : N \to P$ and $0 : 1 \to N$. (Think of the natural numbers and the positive natural numbers.) Taking $L = \{0, 1\}$ with $0 \leq 1$, $A_0[X_0, X_1]$ is X_1 and $A_1[X_0, X_1]$ is 1.

An *order-sorted homomorphism* from an order-sorted $\vec{A}$-algebra $((B_l)_{l\in L}, (g_l)_{l\in L})$ to an order-sorted $\vec{A}$-algebra $((B'_l)_{l\in L}, (g'_l)_{l\in L})$ is a collection of functions $h_l : B_l \to B'_l$ which respects the operations in the sense that

$$h_l \circ g_l =_{(A_l[B_l]\to B'_l)} g'_l \circ A_l[\vec{h}]$$

(for l in L) and also respects the sorts, in the sense that

$$h_l \circ \iota_{m,l} =_{(B_m\to B'_l)} \iota'_{m,l} \circ h_m$$

(for $m \leq l$), where $\iota_{m,l} : B_m \to B_l$ and $\iota'_{m,l} : B'_m \to B'_l$ are the evident type inclusions.

One can show that an initial order-sorted algebra exists. The idea is to set $A^*_l = \Sigma^*_{m\leq l} A_l$ and let $((B_l)_{l\in L}, (g_l)_{l\in L})$ be an initial $\vec{A}^*$-algebra (for l in L), using the well-known extension of the single-sorted case in System F (see for example [PA93]). Set $C_l = A^*_l[\vec{B}]$ and take f_l to be the composite:

$$A_l[\vec{C}] \overset{A_l[\vec{g}]}{\to} A_l[\vec{B}] \overset{in_l}{\to} A^*_l[\vec{B}] = C_l$$

where $in_l : A_l[\vec{B}] \to A^*_l[\vec{B}]$ is the evident injection. Then $((C_l)_{l\in L}, (f_l)_{l\in L})$ is the initial order-sorted $\vec{A}$-algebra.

The final order-sorted $\vec{A}$-co-algebra can be similarly constructed from the final order-sorted $\vec{A}^*$-co-algebra.

One would really want to improve these results to allow coherent overloading (as exemplified by a + operation over both natural numbers and reals). This can perhaps be achieved by extending $F_\leq$ with intersection types, following Reynolds and Pierce [Pie91, Rey88]. It seems straightforward to extend our logic to handle these constructs.

Acknowledgments

We benefited from discussions with Val Breazu-Tannen, Pierre-Louis Curien, and John Mitchell.

References

[ACC93] Martín Abadi, Luca Cardelli, and Pierre-Louis Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 121(1–2):9–58, December 1993.

[BCGS91] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–222, July 1991.

[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.

[BL90] Kim Bruce and Giuseppe Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87(1/2):196–240, 1990.

[Bru93] Kim Bruce. Safe type checking in a statically-typed object-oriented programming language. In *Proceedings of the Twentieth Annual ACM Symposium on the Principles of Programming Languages*, pages 285–298, January 1993.

[Car86] Luca Cardelli. Amber. In Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet, editors, *Combinators and Functional Programming Languages*, pages 21–47. Springer-Verlag, 1986. Lecture Notes in Computer Science No. 242.

[Car92] Luca Cardelli. Extensible records in a pure calculus of subtyping. In Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects of Object-oriented Programming: Types, Semantics and Language Design*. MIT Press, to appear. A preliminary version appeared as SRC Research Report No. 81, 1992.

[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.

[CG94] Pierre-Louis Curien and Giorgio Ghelli. Decidability and confluence of $\beta\eta\mathrm{top}_\leq$ reduction in $F_\leq$. *Information and Computation*, 94(1–2):57–114, February/March 1994.

[CHC90] William R. Cook, Walter L. Hill, and Peter S. Canning. Inheritance is not subtyping. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 125–135. ACM, January 1990.

[CMMS94] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. *Information and Computation*, 94(1–2):4–56, February/March 1994.

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, December 1985.

[Ghe90] Giorgio Ghelli. *Proof-Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Università di Pisa, 1990. Report TD-6/90.

[Gir72] Jean-Yves Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur.* Thèse de doctorat d'état, Université Paris VII, June 1972.

[GM92] Joseph A. Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, November 1992.

[Has94] Ryu Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 4(1):71–110, March 1994.

[Ma92] QingMing Ma. Parametricity as subtyping. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pages 281–292. ACM, January 1992.

[Mog86] Eugenio Moggi. The maximum consistent theory of the second order $\beta\eta$-lambda calculus. Communication in the TYPES electronic forum (types@theory.lcs.mit.edu), July 1986.

[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.

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

[MTH90] Robin Milner, Mads Tofte, and Robert W. Harper. *The Definition of Standard ML.* MIT Press, Cambridge, Massachusetts, 1990.

[Nel91] Greg Nelson, editor. *Systems Programming in Modula-3.* Prentice Hall, 1991.

[PA93] Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer-Verlag, March 1993.

[Pho92] Wesley K. Phoa. Using fibrations to understand subtypes. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, volume 177 of *London Mathematical Lecture Note Series*, pages 239–254, 1992.

[Pie91] Benjamin C. Pierce. *Programming with intersection types and bounded polymorphism.* PhD thesis, Carnegie Mellon University, December 1991.

[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).

[Rey88] John C. Reynolds. Preliminary design of the programming language Forsythe. Technical Report CMU-CS-88-159, Carnegie Mellon University, June 1988.

[Str67] Christopher Strachey. Fundamental concepts in programming languages. Unpublished lecture notes of the International Summer School in Computer Programming, Copenhagen, August 1967.

[Wad89] Philip Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359. ACM, 1989.

[Wir88] Niklaus Wirth. The programming language Oberon. *Software—Practice and Experience*, 18:661–670, 1988.