Abstraction Barrier-Observing Relational Parametricity

Jo Hannay

Department of Software Engineering,
Simula Research Laboratory, Pb. 134, NO-1325 Lysaker, Norway
jo@simula.no

Abstract. A concept of relational parametricity is developed taking into account the encapsulation mechanism inherent in universal types. This is then applied to data types and refinement, naturally giving rise to a notion of simulation relations that compose for data types with higher-order operations, and whose existence coincides with observational equivalence. The ideas are developed syntactically in lambda calculus with a relational logic. The new notion of relational parametricity is asserted axiomatically, and a corresponding parametric *per*-semantics is devised.

1 Introduction

Information hiding is essential in software development. It allows on the one hand, clients of a module to disregard irrelevant detail of the module; this is the *abstraction* aspect of information hiding, and on the other hand it protects the module from unauthorised access to inner workings; this is the *encapsulation* aspect. Thus, information hiding is a prerequisite for the plugability and interchangeability of software components. Information hiding is achieved through interfaces by raising appropriate *abstraction barriers*, and the fundamental theoretical device for this is the concept of *data type*, consisting of an encapsulated data representation and operations on that data representation.

A measure for interchangeability is *observational equivalence*; two data types are interchangeable if they exhibit equivalent observable behaviours. Observational equivalence is generally hard to show, and a simpler strategy is to show the existence of a *simulation relation*, *i.e.,* a relation between the data representations of the data types, which the respective data-type operations preserve. The existence of a simulation relation must coincide with observational equivalence for this to be a valid method, and for the sake of constructive stepwise refinement, simulation relations should compose. For data types with first-order operations, this is the case, but when higher-order operations are involved, both these properties break down in general. Several remedies to this have been found; on the semantic level we have *pre-logical relations* [14, 13], *lax logical relations* [26, 16], and *L-relations* [15]. On the logical level we have *abstraction barrier-observing* simulation relations [9, 10]. The latter, developed in the setting of System F and a relational logic [25] with the assertion of *relational parametricity* [28], are directly motivated by the information-hiding mechanism in data

types, in that they reflect the way data-type operations are used by clients. Ultimately, these techniques apply in stepwise refinement processes producing certified components, *e.g.,* [30, 13, 1, 23, 10, 7, 8].

Abstraction barrier-observing simulation relations were designed specifically to solve the above issues regarding observational equivalence, and to remedy the lack of composability. However, the main ingredient in this new notion of simulation relation, namely respecting an abstraction barrier; and in fact the abstraction barrier itself, both originate from considerations on universal types.

This paper therefore goes back to basic principles and develops abstraction barrier-observing relations fundamentally at universal types. The first step is to define the appropriate family of relations, mirroring how polymorphic functionals may be used according to the abstraction barrier inherent in universal types. We argue that this is the correct relational concept for universal types. Then we go on and use this concept in asserting *abstraction barrier-observing relational parametricity*. If we insist on using abstraction barrier-observing relations for universal types, we must also assert the corresponding notion of relational parametricity, if we are to keep the parametricity principal or have any hopes of retaining proof power. The third step is to give a model for the logic and the abstraction barrier-observing relational parametricity axiom schema.

With this in place, abstraction barrier-observing simulation relations are a direct application of the abstraction barrier-observing relational concept for universal types, mirroring data-type abstraction barriers.

Besides the aesthetical benefit gained from developing the new notion of simulation relations as a natural consequence of basic considerations, we get a substantial simplification of formalism compared to [9, 10], where one only asserts specialised instances of abstraction barrier-observing relational parametricity. In [9, 10], the logical language and the calculus is augmented in order to present a non-syntactic model for these instances. With abstraction barrier-observing relational parametricity this is not necessary.

2 Type Theory and Logic

We review relevant formal aspects. For full accounts, see [3, 21, 6, 25, 2].

The second-order lambda-calculus F_2, or System F, has abstract syntax

$$\begin{array}{ll}\text{(types)} & T ::= X \mid (T \to T) \mid (\forall X.T) \\ \text{(terms)} & t \ ::= x \mid (\lambda x{:}T.t) \mid (tt) \mid (\Lambda X.t) \mid (tT)\end{array}$$

where X and x range over type and term variables respectively. For simplicity, we obey *Barendregt's variable convention*; bound variables are chosen to differ in name from free variables in any type or term. The main syntactic feature of System F is polymorphic functionals. For example, a general function-composition operator is achieved by $comp \stackrel{def}{=} \Lambda X, Y, Z.\lambda f : X \to Y.\lambda g : Y \to Z.\lambda x : X.g(fx)$. One can also define inductive types [4], *e.g.,* $\mathsf{Nat} \stackrel{def}{=} \forall X.X \to (X \to X) \to X$. The inductive inhabitants are the closed terms. It is easy to program constructors, as well as destructors and conditionals, since iteration is built in. We shall use

inductive types Nat, Bool, and List_U, and various self-explanatory terms such as $\mathsf{0}$, succ, true, false, cons, $\mathsf{cond}:\forall Y.\mathsf{Bool}\to Y\to Y\to Y$, isnil, *etc.* Binary products are encoded in System F as inductive types by $U\times V \stackrel{def}{=} \forall X.((U\to V\to X)\to X)$, where U and V have no free X, with constructor $\mathsf{pair}_{U,V}:U\to V\to U\times V$, and also destructors $\mathsf{proj}_{1\,U,V}: U\times V\to U$ and $\mathsf{proj}_{2\,U,V}: U\times V\to V$, defined by $\mathsf{pair}_{U,V}uv \stackrel{def}{=} \Lambda X.\lambda f: U\to V\to X.fuv$, $\mathsf{proj}_{1\,U,V}(x) \stackrel{def}{=} xU(\lambda x:U.\lambda y:V.x)$, and $\mathsf{proj}_{2\,U,V}(x) \stackrel{def}{=} xV(\lambda x{:}U.\lambda y{:}V.y)$. This generalises to n-ary products.

Consider a computation $f=\Lambda X.\lambda x{:}U[X].t[X,x]$. We refer to X as a *virtual data representation*, x as a collection of *virtual operations*, and the whole computation as a *virtual computation*. Any instance $(f\,A\,a)$ of the computation with an *actual data representation* A, and *actual operations* a then gives an *actual computation*. A crucial observation which will determine future development, is now embodied in the following obvious statement.

Abs-Bar1: A virtual client computation $\Lambda X.\lambda x:T[X].t[X,x]$ cannot have free variables of types involving the virtual data representation X.

For data types, encapsulation is provided in the style of [22] by the following encoding of existential (abstract) types and pack and unpack combinators.

$$\exists X.T[X] \stackrel{def}{=} \forall Y.(\forall X.(T[X]\to Y)\to Y), \qquad\qquad Y \text{ not free in } T$$

$$\mathsf{pack}_{T[X]}:\forall X.(T[X]\to\exists X.T[X])$$
$$\mathsf{pack}_{T[X]}(A)(opns) \stackrel{def}{=} \Lambda Y.\lambda f:\forall X.(T[X]\to Y).f(A)(opns)$$

$$\mathsf{unpack}_{T[X]}:(\exists X.T[X])\to\forall Y.(\forall X.(T[X]\to Y)\to Y)$$
$$\mathsf{unpack}_{T[X]}(package)(B)(client) \stackrel{def}{=} package(B)(client)$$

Operationally, pack packages a data representation and an implementation of operations on that data representation to give a data type of the existential type. The resulting package is a polymorphic functional that given a client computation and its result domain, instantiates the client with the particular elements of the package. The unpack combinator is merely the application operator for pack. An abstract type for stacks of natural numbers could be $\exists X.(X\times(\mathsf{Nat}\to X\to X)\times(X\to X)\times(X\to\mathsf{Nat}))$. A data type of this type is, *e.g.,* $(\mathsf{pack}\,\mathsf{List}_{\mathsf{Nat}}\,\mathfrak{l})$, where $(\mathsf{proj}_1\,\mathfrak{l})=\mathsf{nil}$, $(\mathsf{proj}_2\,\mathfrak{l})=\mathsf{cons}$, $(\mathsf{proj}_3\,\mathfrak{l})=\lambda l:\mathsf{List}_{\mathsf{Nat}}.(\mathsf{cond}\,\mathsf{List}_{\mathsf{Nat}}\,(\mathsf{isnil}\,l)\,\mathsf{nil}\,(\mathsf{cdr}\,l))$, and $(\mathsf{proj}_4\,\mathfrak{l})=\lambda l:\mathsf{List}_{\mathsf{Nat}}.(\mathsf{cond}\,\mathsf{Nat}\,(\mathsf{isnil}\,l)\,\mathsf{0}(\mathsf{car}\,l))$. For convenience we use a labelled product notation; $\exists X.T_{\mathsf{STACK}_{\mathsf{Nat}}}[X]$, where $T_{\mathsf{STACK}_{\mathsf{Nat}}}[X] \stackrel{def}{=} (\mathsf{empty}:X,\ \mathsf{push}:\mathsf{Nat}\to X\to X,\ \mathsf{pop}:X\to X,\ \mathsf{top}:X\to\mathsf{Nat})$. We call each $f_i:T_i[X]$ a *profile* of the existential type.

Existential types together with the pack and unpack combinators embody an abstraction barrier composed of three components.

First of all, ***Abs-Bar1*** says that a client computation $\Lambda X.\lambda x:T[X].t[X,x]$ cannot have free variables of types involving the virtual data representation X, *e.g.,* by Barendregt's variable convention, $y:X \ \rhd\ \Lambda X.\lambda\mathfrak{x}:T_{\mathsf{STACK}_{\mathsf{Nat}}}.(\mathfrak{x}.\mathsf{push}\,n\,y)$ is ill-formed. A client computation may compute over types containing the virtual data representation X only by accessing virtual operations in the supplied collection $\mathfrak{x}$, as in *e.g.,* $\Lambda X.\lambda\mathfrak{x}:T_{\mathsf{STACK}_{\mathsf{Nat}}}\ .\ \mathfrak{x}.\mathsf{top}(\mathfrak{x}.\mathsf{push}\,n\,\mathfrak{x}.\mathsf{empty})$.

In large, up to $\beta\eta$-equivalence, package operations may only be applied to arguments definable by package operations. This aspect is essential in restricting access to the underlying data representation, which may contain invalid values. Note that a supplied package operation might itself make calls involving arguments of types over the actual data representation, which are not expressed in terms of package operations. In this case, it is not true that package operations will only be applied to definable arguments, but crucially, this is at the discretion of the package implementor, and not due to direct user application.

There are two other aspects of the abstraction barrier inherent in the encoding of existential types. The next one, a prerequisite for plugability, is the already stated condition in the definition of the encoding of existential types.

***Abs-Bar2*:** The type Y does not occur in T in the encoding

$$\exists X.T[X] \stackrel{def}{=} \forall Y.(\forall X.(T[X] \to Y) \to Y).$$

This ensures that data types do not depend on their environment of usage, other than via explicitly given parameters if $\exists X.T[X]$ has free types.

The third aspect arises from the fact that when using a data type, *i.e.,* a package of existential type, the user must provide the result type of the client computation. This entails the following.

***Abs-Bar3*:** Client computations $f : \forall X.(T[X] \to Y)$ cannot have a result type containing the virtual data representation, *i.e.,* the bound type variable X.

For example, $f \stackrel{def}{=} \Lambda X.\lambda \mathfrak{x} : T_{\mathsf{STACK}_{\mathsf{Nat}}}.(\mathfrak{x}.\mathsf{push}\, n\, \mathfrak{x}.\mathsf{empty}) : \forall X.(T_{\mathsf{STACK}_{\mathsf{Nat}}} \to X)$ is not a possible client; in order to use a package $(\mathsf{pack}Aa)$ in f, we must do $(\mathsf{pack}Aa)(C)(f)$, where C is the result type of f, which in this case is the inaccessible virtual data representation X. Due to ***Abs-Bar3***, the only way a package can be used is via client computations adhering to the above. This then fixes how packages may be used in actual computations, since all actual computations are uniform in that they inherit the form of the virtual computation from which they stem. We will refer to ***Abs-Bar1***, ***Abs-Bar2***, and ***Abs-Bar3*** jointly as ***Abs-Bar***.

The logic we shall use as a starting point is the logic for parametric polymorphism due to [25]. This logic is essentially a second-order logic augmented with relation symbols and a syntax for relation definition, and the assertion of relational parametricity as an axiom schema. See also [18, 33].

The logic in [25] has formulae built using the standard connectives, but now basic predicates are not only equations, but also relation membership statements,

$$\phi ::= (t =_A u) \mid t\ R\ u \mid \cdots \mid \forall R \subset A \times B.\phi \mid \exists R \subset A \times B.\phi$$

where R ranges over relation variables. We write $\alpha[\xi]$ to indicate possible occurrences of variable ξ in type, term or formula α, and write $\alpha[\beta]$ for the substitution $\alpha[\beta/\xi]$, following the appropriate rules regarding capture.

Judgements for formula formation involve relation symbols, so contexts are augmented with relation variables, depending on type context, and obeying standard conventions. Relation definition is accommodated by the following syntax,

$$\frac{\Gamma, x{:}A, y{:}B \rhd \phi}{\Gamma \rhd (x{:}A, y{:}B)\ .\ \phi \quad \subset A \times B}$$

where ϕ is a formula. For example $\mathsf{eq}_A \stackrel{def}{=} (x{:}A, y{:}A).(x =_A y)$.

We now build complex relations using type formers. We get the *arrow-type relation* $\rho{\to}\rho' \subset (A{\to}A'){\times}(B{\to}B')$ from $\rho{\subset}A{\times}B$ and $\rho'{\subset}A'{\times}B'$ by

$$(\rho{\to}\rho') \stackrel{def}{=} (f{:}A{\to}A', g{:}B{\to}B') \;.\; (\forall x{:}A.\forall y{:}B \;.\; (x\rho y \;\Rightarrow\; (fx)\rho'(gy)))$$

i.e., f and g are related if they map ρ-related arguments to ρ'-related values. The *universal-type relation* $\forall(Y, Z, R{\subset}Y{\times}Z)\rho[R] \;\subset\; (\forall Y.A[Y]){\times}(\forall Z.B[Z])$ is defined from $\rho[R]{\subset}A[Y]{\times}B[Z]$, where Y, Z and $R{\subset}Y{\times}Z$ are free, by

$$\begin{array}{l}\forall(Y, Z, R{\subset}Y{\times}Z)\rho[R] \stackrel{def}{=}\\ \qquad (y{:}\forall Y.A[Y], z{:}\forall Z.B[Z]) \;.\; (\forall Y.\forall Z.\forall R{\subset}Y{\times}Z \;.\; ((yY)\rho[R](zZ)))\end{array}$$

i.e., two y and z are related at universal type if all instances yY and zZ are related in $\rho[R]$, whenever R relates Y and Z.

One defines the *action* of types on relations, by substituting relations for type variables in types. For $\boldsymbol{X} = X_1, \ldots, X_n$, $\boldsymbol{A} = A_1, \ldots, A_n$, $\boldsymbol{B} = B_1, \ldots, B_n$ and $\boldsymbol{\rho} = \rho_1, \ldots, \rho_n$, where $\rho_i{\subset}A_i{\times}B_i$, we get $T[\boldsymbol{\rho}]{\subset}T[\boldsymbol{A}]{\times}T[\boldsymbol{B}]$, the action of $T[\boldsymbol{X}]$ on $\boldsymbol{\rho}$, defined by cases on $T[\boldsymbol{X}]$ as follows:

$$\begin{array}{lll}T[\boldsymbol{X}] = X_i : & & T[\boldsymbol{\rho}] = \rho_i\\ T[\boldsymbol{X}] = T'[\boldsymbol{X}]{\to}T''[\boldsymbol{X}] : & & T[\boldsymbol{\rho}] = T'[\boldsymbol{\rho}]{\to}T''[\boldsymbol{\rho}]\\ T[\boldsymbol{X}] = \forall X'.T'[\boldsymbol{X}, X'] : & & T[\boldsymbol{\rho}] = \forall(Y, Z, R{\subset}Y{\times}Z)T'[\boldsymbol{\rho}, R]\end{array}$$

The proof system is natural deduction, intuitionistic style, over formulae now involving relation symbols, and is augmented with inference rules for relation symbols in the obvious way. There are standard axioms for equational reasoning and $\beta\eta$-equalities. These imply extensionality for arrow and universal types.

Parametric polymorphism promotes all instances of a polymorphic functional to exhibit a uniform behaviour. Of various notions [32, 2]; we adopt *relational parametricity* [28, 17], where uniformity is defined by saying that if a polymorphic functional is instantiated at two related domains, the resulting instances should be related as well. In [25] this is directly asserted by the axiom schema,

$$\textsc{Param} : \;\forall\boldsymbol{Z}.\forall u{:}(\forall X.U[X, \boldsymbol{Z}]) \;.\; u \;(\forall X.U[X, \mathbf{eq}_{\boldsymbol{Z}}])\; u$$

The logic with Param is sound w.r.t. the parametric *per*-model of [2] and also w.r.t. the syntactic parametric models of [11].

The assumption of Param yields interesting results. The following *Identity Extension Lemma* is fundamental and follows from Param and extensionality.

$$\forall\boldsymbol{Z}.\forall u, v{:}U[\boldsymbol{Z}] \;.\; (u \;U[\mathbf{eq}_{\boldsymbol{Z}}]\; v \;\Leftrightarrow\; (u =_{U[\boldsymbol{Z}]} v))$$

Weak versions of constructs such as products, sums, initial and final (co-)algebras are encodable in System F [4]. With Param, these constructs become provably universal constructions. For example, we can derive with Param,

$$\forall U, V.\forall z{:}U{\times}V \;.\; \mathsf{pair}(\mathsf{proj}_1 z)(\mathsf{proj}_2 z) = z$$

$$\begin{array}{l}\forall u{:}A{\times}A', v{:}B{\times}B', \rho{\subset}A{\times}B, \rho'{\subset}A'{\times}B' \;.\\ \qquad u(\rho{\times}\rho')v \;\Leftrightarrow\; (\mathsf{fst}(u)\;\rho\;\mathsf{fst}(v) \wedge \mathsf{snd}(u)\;\rho'\;\mathsf{snd}(v))\end{array}$$

where $\rho \times \rho'$ is obtained by the action $(X \times X')[\rho, \rho']$.

3 Abstraction Barrier-Observing Relations

Abs-Bar says that function arguments in computations are bounded by formal parameters, *e.g.,* in the closed computation $f \stackrel{def}{=} \Lambda X.\lambda x\!:\!X\,\lambda s\!:\!X \to X.t[x,s]$, s will only be applied to arguments built from formal parameters x and s. This transfers to instances fA and fB. Relationally for $R \subset A \times B$, we would like *e.g.,* $s_A\ (R \to R)\ s_B$ to reflect this, in that only those x, y are considered for the antecedent $x\ R\ y$, that are admissible in the computations; we want something like $\forall x\!:\!A, y\!:\!B\ .\ x\ R\ y \wedge \mathsf{Dfnbl}(x,y)\ \Rightarrow\ s_A x\ R\ s_B y$. Although the idea is based on observing closed terms, we neither can, nor wish to be that specific formalistically. Thus if A, B, a, b, s_A, s_B are actual parameters to f, we set $\mathsf{Dfnbl}(x,y)$ to stand for $\exists f_X\!:\!\forall X.X \to (X \to X) \to X\ .\ f_X A a s_A = x \wedge f_X B b s_B = y$. For closed computations $\Lambda X.t$ in particular, this mirrors precisely application in t.

The *abo*-relation we now define formalises this idea. The full definition is complicated by universal-type nesting, and by the need to keep track of which actual parameters have (not) been received. Above, a, b, s_A, s_B are all received at the stage when $\mathsf{Dfnbl}(x,y)$ is defined, but consider what needs to be done for $\Lambda X.\lambda s\!:\!(X \to X).\lambda x\!:\!X.t[s,x]$. The various rôles universal types may assume complicate matters further. The instrumental aspect is however simple argument definability. Example 1 below illustrates *abo*-relations. In Sect. 4 we use *abo*-relations to devise abstraction barrier-observing simulation relations.

Definition 1 (*abo*-Relation). *For any type $U[\boldsymbol{Z}]$, we define the* abstraction barrier-observing relation $U[\mathbf{eq}_{\boldsymbol{Z}}]^{\mathsf{abo}}$ *as follows.*

$$U[\mathbf{eq}_{\boldsymbol{Z}}]^{\mathsf{abo}} \stackrel{def}{=} U[\mathbf{eq}_{\boldsymbol{Z}}] \qquad \textit{if } U \textit{ is not a universal type.}$$

$$(\forall X.U[X, \mathbf{eq}_{\boldsymbol{Z}}])^{\mathsf{abo}} \stackrel{def}{=} (\forall X.U[X, \mathbf{eq}_{\boldsymbol{Z}}])^{\sigma_0 l_0 \mathsf{abo}}$$

where $(\forall X.U[X, \mathbf{eq}_{\boldsymbol{Z}}])^{\sigma_0 l_0 \mathsf{abo}}$ is given by the following. Regard $\forall X.U$ according to outermost arrow structure and universal quantifier nesting. For the m^{th} nesting, write $\forall X_m.U_m = \forall X_m.U_{m1} \to \cdots \to U_{mn_m} \to U_{mc}$, where U_{mc} is not an arrow type, but possibly a universal type $\forall X_{m+1}.U_{m+1}$. Setting $\forall X.U \stackrel{def}{=} \forall X_1.U_1$ as the first nesting, let N be the number of nestings in $\forall X.U$. Define

$$\begin{array}{l}(\forall X_m.U_m[\boldsymbol{R}, X_m, \mathbf{eq}_{\boldsymbol{Z}}])^{\sigma_{m-1} l_{m-1} \mathsf{abo}} \stackrel{def}{=} \\ \qquad (f\!:\!\forall X_m.U_m[\boldsymbol{A}, X_m],\ g\!:\!\forall X_m.U_m[\boldsymbol{B}, X_m])\ . \\ \quad (\forall A_m, B_m, R_m \subset A_m \times B_m\ .\ fA_m\ (U_m[\boldsymbol{R}, R_m, \mathbf{eq}_{\boldsymbol{Z}}]_{\top}^{\sigma_m l_m \langle A_m, B_m \rangle})\ gB_m)\end{array}$$

Define sequences σ_m and l_m as follows. Set $\sigma_0 = l_0 \stackrel{def}{=} \varepsilon$. If U_m has no free variables from σ_{m-1}, then $\sigma_m \stackrel{def}{=} \langle X_j, \langle U_{mi} \rangle_{i=1}^{n_m} \rangle_{j=m}^{q}$, $q \leq N$, where $\forall X_{q+1}.U_{q+1}$ is the first nesting within the m^{th} with no free occurrences of any X_j, $1 \leq j \leq q$. Set $l_m \stackrel{def}{=} \varepsilon$. If on the other hand, U_m has free variables from σ_{m-1} then $\sigma_m \stackrel{def}{=} \sigma_{m-1}$, and $l_m \stackrel{def}{=} l_{m-1}$. The sequence l_m will now be the basis for sequences l_{mk} at each U_{mk} with further items. These will have the format $\langle \langle A_j, B_j \rangle \langle a_{j_i}, b_{j_i} \rangle_{i=1}^{k_j} \rangle_{j=p}^{m}$, where p is the index of the first item in σ_m, and $1 \leq k_j \leq n_j$. We write $l\langle \alpha, \beta \rangle$ to indicate the implicit insertion of $\langle \alpha, \beta \rangle$ format-correctly in l. In general, we expand $U[\boldsymbol{R}, \mathbf{eq}_{\boldsymbol{Z}}]_{\top}^{\sigma l}$ through the following definitions.

$$(U'[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]\to U''[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l}_{\top} \stackrel{def}{=}$$
$$(f{:}U'[\boldsymbol{A}]\to U''[\boldsymbol{A}],\ g{:}U'[\boldsymbol{B}]\to U''[\boldsymbol{B}])\ .$$
$$(\forall a{:}U'[\boldsymbol{A}],\ b{:}U'[\boldsymbol{B}]\ .\ a\ (U'[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l\langle a,b\rangle}\ b$$
$$\Rightarrow\ (fa)\ (U''[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l\langle a,b\rangle}_{\top}\ (gb))$$

$U[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l}_{\top} \stackrel{def}{=} U[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l\mathsf{labo}}$, *if U is a universal type*

$U[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l}_{\top} \stackrel{def}{=} U[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l}$, *otherwise*

Then, $U[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l}$, is defined by

${R_i}^{\sigma l} \stackrel{def}{=} R_i$

${\mathsf{eq}_{Z_i}}^{\sigma l} \stackrel{def}{=} \mathsf{eq}_{Z_i}$

$$(U'[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]\to U''[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l} \stackrel{def}{=} (f{:}U'[\boldsymbol{A}]\to U''[\boldsymbol{A}],\ g{:}U'[\boldsymbol{B}]\to U''[\boldsymbol{B}])\ .$$
$$(\forall_{\sigma-l}.\forall x{:}U'[\boldsymbol{A}],\ y{:}U'[\boldsymbol{B}]\ .$$
$$(x\ U'[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l(\sigma-l)}\ y\ \wedge\ \mathsf{Dfnbl}^{\sigma l(\sigma-l)}_{U'}(x,y))$$
$$\Rightarrow\ (fx)\ U''[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l(\sigma-l)}\ (gy))$$

where $\forall_{\sigma-l} \stackrel{def}{=} \forall\{A_j,B_j,a_{j_i}{:}U_{j_i}[\boldsymbol{A}],b_{j_i}{:}U_{j_i}[\boldsymbol{B}]\ \mid$
$1\le j\le N,\ 1\le i\le n_j,\quad \langle A_j,B_j\rangle,a_{j_i},b_{j_i}\notin l\wedge X_j,U_{j_i}\in\sigma\}$

and $l(\sigma-l)$, is l appended format-correctly with the $\forall_{\sigma-l}$ quantified items, and

$$\mathsf{Dfnbl}^{\sigma l(\sigma-l)}_{U'}(x,y) \stackrel{def}{=}$$
$$\exists f_{U'}{:}\forall X_p.(U_{p_1}\to\cdots\to U_{p_{n_p}}\to\cdots\to(\forall X_q.U_{q_1}\to\cdots\to U_{q_{n_q}}\to U'))\ .$$
$$(f_{U'}A_pa_{p_1}\cdots a_{p_{n_p}}\cdots A_qa_{q_1}\cdots a_{q_{n_q}})=x\ \wedge$$
$$(f_{U'}B_pb_{p_1}\cdots b_{p_{n_p}}\cdots B_qb_{1_q}\cdots b_{1_{n_q}})=y$$

where p is the index of the first item in σ. The functional $f_{U'}$ receives as arguments in proper order, the types and terms in l, and then in place of missing actual arguments, the "hypothetical" arguments given by the quantification $\forall_{\sigma-l}$.

Finally, the relation $(\forall Y.U'[\boldsymbol{R},Y,\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l}$ at universal type within some U_{mk} reflects various rôles terms in this position may play. First we define, if $\forall Y.U'$ is internal, *i.e., is not equal to some $\forall X_r.U_r$, $m<r\le q$, nor closed, then*

$$(\forall Y.U'[\boldsymbol{R},Y,\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l} \stackrel{def}{=} (f'{:}\forall Y.U'[\boldsymbol{A}],g'{:}\forall Y.U'[\boldsymbol{B}])\ .$$
$$(\textstyle\bigwedge_{V[X]}\quad f'V[\boldsymbol{A}]\quad (U'[\boldsymbol{R},V[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}],\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l}\quad g'V[\boldsymbol{B}]$$

and if $\forall Y.U'$ is not internal, and occurs only negatively within some U_{mk},

$$(\forall Y.U'[\boldsymbol{R},Y,\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l} \stackrel{def}{=} (\forall Y.U'[\boldsymbol{R},Y,\mathbf{eq}_{\boldsymbol{Z}}]))^{\sigma l\mathsf{labo}}$$

and otherwise,

$$(\forall Y.U'[\boldsymbol{R},Y,\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l} \stackrel{def}{=} (f'{:}\forall Y.U'[\boldsymbol{A}],g'{:}\forall Y.U'[\boldsymbol{B}])\ .$$
$$(\textstyle\bigwedge_{V[X]}\quad f'V[\boldsymbol{A}]\quad (U'[\boldsymbol{R},V[\boldsymbol{R},\mathbf{eq}_{\boldsymbol{Z}}],\mathbf{eq}_{\boldsymbol{Z}}])^{\sigma l}\quad g'V[\boldsymbol{B}]\ \wedge$$
$$f'\ (\forall Y.U'[\boldsymbol{R},Y,\mathbf{eq}_{\boldsymbol{Z}}]))^{\sigma l\mathsf{labo}}\ g')$$

This concludes the definition.

Example 1. Consider the type $\mathsf{Nat} \stackrel{def}{=} \forall X.X \to (X \to X) \to X$. Observing the abstraction barrier in this universal type, we have for $n : \mathsf{Nat}$,

$$\begin{aligned} & n\ (\forall X.X \to (X \to X) \to X)^{\mathsf{abo}}\ n \\ & \qquad \stackrel{def}{\Leftrightarrow}\ \forall A, B, R \subset A \times B\ .\ nA\ (R \to (R \to R) \to R)^{\langle A,B \rangle}_{\top}\ nB \end{aligned}$$

omitting the superscript $\sigma = X, X, (X \to X)$. This expands to

$$\begin{aligned} & \forall A, B, R \subset A \times B\ . \\ & \quad \forall a : A, b : B\ .\ a\ R^{\langle A,B \rangle \langle a,b \rangle}\ b\ \Rightarrow \\ & \qquad \forall s : A \to A, s' : B \to B\ .\ s\ (R \to R)^{\langle A,B \rangle \langle a,b \rangle \langle s,s' \rangle}\ s'\ \Rightarrow\ nAas\ R\ nBbs' \end{aligned}$$

Here, $R^{\langle A,B \rangle \langle a,b \rangle}$ is simply R, and $s\ (R \to R)^{\langle A,B \rangle \langle a,b \rangle \langle s,s' \rangle}\ s'$ expands to

$$\begin{aligned} & \forall x : A, y : B\ .\ x\ R\ y\ \wedge \\ & \qquad (\exists f_X : \forall X.X \to (X \to X) \to X\ .\ f_X Aas = x \wedge f_X Bbs' = y) \\ & \qquad\qquad \Rightarrow\ sx\ R\ s'y \end{aligned}$$

The definability clause here reflects that s and s' can be applied only to x and y admissible in virtual computations. For instance, if n were a closed computation, this would mean x and y built from a, s, and b, s'. Moreover, x and y are uniform, since they arise from the same virtual computation n.

Now, consider the type $\mathsf{Nat}' \stackrel{def}{=} \forall X.(X \to X) \to X \to X$, where the successor argument is received prior to the zero argument. Now we have

$$\begin{aligned} & n\ (\forall X.(X \to X) \to X \to X)^{\mathsf{abo}}\ n \\ & \qquad \stackrel{def}{\Leftrightarrow}\ \forall A, B, R \subset A \times B\ .\ nA\ ((R \to R) \to R \to R)^{\langle A,B \rangle}_{\top}\ nB \end{aligned}$$

omitting the superscript $\sigma = X, (X \to X), X$. This expands to

$$\begin{aligned} & \forall A, B, R \subset A \times B\ . \\ & \quad \forall s : A \to A, s' : B \to B\ .\ s\ (R \to R)^{\langle A,B \rangle \langle s,s' \rangle}\ s'\ \Rightarrow \\ & \qquad\qquad \forall a : A, b : B\ .\ a\ R^{\langle A,B \rangle \langle s,s' \rangle \langle a,b \rangle}\ b\ \Rightarrow\ nAsa\ R\ nBs'b \end{aligned}$$

Here, $R^{\langle A,B \rangle \langle s,s' \rangle \langle a,b \rangle}$ is simply R, but $s\ (R \to R)^{\langle A,B \rangle \langle s,s' \rangle}\ s'$ expands to

$$\begin{aligned} & \forall a : A, b : B\ .\ \forall x : A, y : B\ . \\ & \quad x\ R\ y\ \wedge (\exists f_X : \forall X.(X \to X) \to X \to X\ .\ f_X Asa = x \wedge f_X Bs'b = y) \\ & \qquad\qquad \Rightarrow\ sx\ R\ s'y \end{aligned}$$

At the point when the successor arguments s and s' are received by n there is no knowledge of what the zero arguments will be. Therefore, the definability clause for the successor arguments is prepared for arbitrary zero arguments a, b.

To illustrate nesting, consider

$$f\ (\forall X.X \to (X \to X) \to (\forall Y.(Y \to X) \to (X \to Y) \to X))^{\mathsf{abo}}\ f$$

Omitting $\sigma = X, X, (X \to X), Y, (Y \to X), (X \to Y)$, this expands to

$$\begin{array}{l}\forall A,B,R\subset A\times B\ .\\ \quad \forall a:A,b:B\ .\ a\ R^{\langle A,B\rangle\langle a,b\rangle}\ b\ \Rightarrow\\ \quad \forall s:A\to A,s':B\to B\ .\ s\ (R\to R)^{\langle A,B\rangle\langle a,b\rangle\langle s,s'\rangle}\ s'\ \Rightarrow\\ \forall A',B',R'\subset A'\times B'\ .\\ \quad \forall r:A'\to A,r':B'\to B\ .\ r\ (R'\to R)^{\langle A,B\rangle\langle a,b\rangle\langle s,s'\rangle\langle A',B'\rangle\langle r,r'\rangle}\ r'\ \Rightarrow\\ \quad \forall q:A\to A',q':B\to B'\ .\ q\ (R\to R')^{\langle A,B\rangle\langle a,b\rangle\langle s,s'\rangle\langle A',B'\rangle\langle r,r'\rangle\langle q,q'\rangle}\ q'\\ \qquad\qquad \Rightarrow\ fAasA'rq\ R^{\langle A,B\rangle\langle a,b\rangle\langle s,s'\rangle\langle A',B'\rangle\langle r,r'\rangle\langle q,q'\rangle}\ fBbs'B'r'q'\end{array}$$

Here, $s\ (R\to R)^{\langle A,B\rangle\langle a,b\rangle\langle s,s'\rangle}\ s'$ expands to

$$\begin{array}{l}\forall A',B'.\forall r:A'\to A,r':B'\to B,q:A\to A',q':B\to B'\ .\ \forall x:A,y:B\ .\\ \quad x\ R\ y\ \wedge(\exists f_X:\forall X.X\to(X\to X)\to(\forall Y.(Y\to X)\to(X\to Y)\to X)\ .\\ \qquad\qquad f_XAasA'rq=x\wedge f_XBbs'B'r'q'=y)\\ \qquad\qquad \Rightarrow\ sx\ R\ s'y\end{array}$$

This reflects that arguments to s and s' may be defined also in terms of future r, q, r', and q', besides a, b, s, s'. For example if f were closed, we could have $f=\Lambda X.\lambda x:X,s:X\to X.\Lambda Y.\lambda r:Y\to X,q:X\to Y\ .\ s(r(qx))$.

To illustrate the situation for universal types within some U_{mk}, consider

$$f\ (\forall X.X\to(\forall Y.Y\to X)\to(X\to\mathsf{Bool}\to\mathsf{Nat})\to\mathsf{Nat})^{\mathsf{abo}}\ f$$

We get for $\sigma=X,X,(\forall Y.Y\to X),(X\to\mathsf{Bool}\to\mathsf{Nat})$,

$$\begin{array}{l}\forall A,B,R\subset A\times B\ .\ \forall a,b\ .\ a\ R\ b\ \Rightarrow\\ \quad \forall q,q'\ .\ q\ (\forall Y.Y\to R)^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle}\ q'\ \Rightarrow\\ \qquad \forall r,r'\ .\ r\ (R\to\mathsf{Bool}\to\mathsf{Nat})^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle\langle r,r'\rangle}\ r'\\ \qquad\qquad \Rightarrow\ fAaqr\ \mathsf{Nat}^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle\langle r,r'\rangle\mathsf{abo}}\ fBbq'r'\end{array}$$

Here, $\forall Y.Y\to X$ is inner, *i.e.,* does not occur as a nesting, and is not closed. So for every type V, $V\to X$ is just a local profile within the type of f. Thus, $q\ (\forall Y.Y\to R)^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle}\ q'$ says $\bigwedge_V qV[A]\ (V[R]\to R)^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle}\ q'V[B]$, reflecting that q,q' may only be used internally. In contrast, Nat is not inner, since it is indeed a nesting, and is also closed. In a computation, terms of this type may be used in two ways; internally, and also externally; the use of $fAaqr$ and $fBbq'r'$ of type Nat is no longer bound by arguments to f. Thus, we have

$$\begin{array}{l}x\ \ \mathsf{Nat}^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle\langle r,r'\rangle}\ \ y\overset{def}{\Leftrightarrow}\\ \bigwedge_{V[X]}\ xV[A]\ \ ((V[R]\to(V[R]\to V[R])\to V[R])^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle\langle r,r'\rangle}\ \ yV[B]\\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\wedge\ x\ \ \mathsf{Nat}^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle\langle r,r'\rangle\mathsf{abo}}\ \ y\end{array}$$

Since Nat has no free occurrences of X, $n\ \mathsf{Nat}^{\sigma\langle A,B\rangle\langle a,b\rangle\langle q,q'\rangle\langle r,r'\rangle\mathsf{abo}}\ m$ expands to the same as $\mathsf{Nat}^{\mathsf{abo}}$. Finally, Bool occurs only negatively in $X\to\mathsf{Bool}\to\mathsf{Nat}$ and the relation is here only $\mathsf{Bool}^{\mathsf{abo}}$, since any term in this position will not be used the internal way. All this reflects *Abs-Bar1*. Consider for example the closed computation $f=\Lambda X.\lambda x:X,q:\forall Y.Y\to X,r:X\to\mathsf{Bool}\to\mathsf{Nat}\ .\ r(qXx)\mathsf{true}$. ○

Example 1 illustrates *abo*-relations reflexively. If two computations, $n, m : \mathsf{Nat}$ for the example above, are involved, the uniformity aspect is not appropriate. The reason Def. 1 ignores this, is that it is geared toward *abo*-identity extension. Thus, if for example $n\ (\forall X.X \to (X \to X) \to X)^{\mathsf{abo}}\ m$, then we shall get $n = m$, so in this context, there is only one computation involved after-all.

The abstraction barrier-observing formulation of relational parametricity is now given by the following axiom schema.

Definition 2 (*abo*-Parametricity).

$$abo\text{-}\textsc{Param}: \ \forall \boldsymbol{Z}.\forall u{:}(\forall X.U[X, \boldsymbol{Z}])\ .\ u\ (\forall X.U[X, \mathbf{eq}_{\boldsymbol{Z}}])^{\mathsf{abo}}\ u$$

The *abo*-version of the identity extension lemma does not follow from *abo*-Param, because we can no longer use extensionality. Nevertheless, in the spirit of observing abstraction barriers, we argue that in virtual computations, it suffices to consider extensionality only w.r.t. function arguments that will actually occur. The simplest way to capture this is in fact by asserting identity extension.

Definition 3 (*abo*-Identity Extension for Universal Types).

$$abo\text{-}\textsc{Iel}: \ \forall \boldsymbol{Z}.\forall u{:}(\forall X.U[X, \boldsymbol{Z}])\ .\ u\ (\forall X.U[X, \mathbf{eq}_{\boldsymbol{Z}}])^{\mathsf{abo}}\ v \ \Leftrightarrow\ u = v$$

Both *abo*-Param and *abo*-Iel will be shown to hold in the *abo*-parametric *per*-model. Regular parametricity, Param, will not hold in this model; in fact any logic containing both of Param and *abo*-Param is inconsistent. Note that *abo*-Iel implies *abo*-Param. Nevertheless, we choose to display both. We get,

Theorem 1 (*abo*-Identity Extension). *With abo-*Iel*, we have*

$$\forall \boldsymbol{Z}.\forall u, v{:}U[\boldsymbol{Z}]\ .\ u\ U[\mathbf{eq}_{\boldsymbol{Z}}]^{\mathsf{abo}}\ v \ \Leftrightarrow\ (u =_{U[\boldsymbol{Z}]} v)$$

Proof: Easy induction. □

There is an inner aspect of *abo*-identity extension as well.

Theorem 2 (*abo*-Identity Extension (Inner Aspect)). *Let U be a type with no occurrences of X_j in σ. Then we derive with abo-*Iel*,*

$$\forall u, v{:}U[\boldsymbol{Z}]\ .\ u\ U[\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma l}\ v \ \Leftrightarrow\ u =_{U[\boldsymbol{Z}]} v$$

Proof: Induction on the structure of U. □

We also get

Theorem 3. *With abo-*Param *and abo-*Iel*, we derive*

$$\forall U, V.\forall z{:}U \times V\ .\ \mathsf{pair}(\mathsf{proj}_1 z)(\mathsf{proj}_2 z) = z$$

Proof: First show $\forall U, V, z{:}U \times V\ .\ z = z(U \times V)\mathsf{pair}$. Use Theorem 2. □

Theorem 4. *With abo-*PARAM *and abo-*IEL*, we derive*

$$\begin{array}{l}\forall u, v{:}T_1 \times T_2 \;.\; u\ (T_1[\mathbf{eq}_{\boldsymbol{Z}}]\times T_2[\mathbf{eq}_{\boldsymbol{Z}}])^{\mathsf{abo}}\ v \\ \quad \Leftrightarrow\ (\mathsf{proj}_1 u)\ T_1[\mathbf{eq}_{\boldsymbol{Z}}]^{\mathsf{abo}}\ (\mathsf{proj}_1 v)\ \wedge\ (\mathsf{proj}_2 u)\ T_2[\mathbf{eq}_{\boldsymbol{Z}}]^{\mathsf{abo}}\ (\mathsf{proj}_2 v)\end{array}$$

$$\begin{array}{l}\forall A, B, R{\subset} A \times B \;.\; \forall u{:}T_1[A] \times T_2[A],\ v{:}T_1[B] \times T_2[B] \;. \\ \quad u\ (T_1[R, \mathbf{eq}_{\boldsymbol{Z}}]\times T_2[R, \mathbf{eq}_{\boldsymbol{Z}}])^{T[X]\langle A,B\rangle\langle u,v\rangle}\ v \\ \qquad \Leftrightarrow\ (\mathsf{proj}_1 u)\ T_1[R, \mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle A,B\rangle\langle u,v\rangle}\ (\mathsf{proj}_1 v)\ \wedge \\ \qquad\qquad (\mathsf{proj}_2 u)\ T_2[R, \mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle A,B\rangle\langle u,v\rangle}\ (\mathsf{proj}_2 v)\end{array}$$

Proof: Use Theorem 3. □

4 Refinement

Consider now the issue of when two packages are interchangeable in a program. We view observational equivalence as the conceptual description of interchangeability, and simulation relations as part of a method for showing observational equivalence. We want the two notions to be equivalent. For data types with first-order operations, this equivalence is a fact under relational parametricity. At higher-order this is not the case. Also, the composability of simulation relations fails at higher order, compromising the constructive composition of refinement steps. We now show that by using *abo*-relations and *abo*-parametricity, we get the equivalence as well as composability, at any order.

To each refinement stage, a set *Obs* of *observable types* is associated, assumed to contain closed inductive types, such as Bool or Nat, and also any parameters. Two data types are interchangeable if their observable properties are indistinguishable, *i.e.,* packages should be observationally equivalent if it makes no difference which one is used in computations with observable result types. Thus,

Definition 4 (Observational Equivalence). *For A, B, $\mathfrak{a}{:}T[A], \mathfrak{b}{:}T[B]$, Obs,*

$$\bigwedge_{D \in Obs} \forall f{:}\forall X.(T[X] \to D) \;.\; (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b})$$

For example, an observable computation on natural-number stacks could be $\Lambda X.\lambda \mathfrak{x}{:}T_{\mathsf{STACK}_{\mathsf{Nat}}}[X] \;.\; \mathfrak{x}.\mathsf{top}(\mathfrak{x}.\mathsf{push}\ n\ \mathfrak{x}.\mathsf{empty})$.

Simulation relations arise from the concept of *data refinement* [12, 5] and the use of relations to show *representation independence* [19, 31, 28, 27], leading to *logical relations* for lambda calculus [20, 21, 34, 24]. In our logic one can use the action of types on relations to define a syntactic mirror of the above ideas. Two data types are related by a simulation relation if there exists a relation on their data representations that is preserved by their corresponding operations.

Definition 5 (Simulation Relation). *For A, B, $\mathfrak{a}{:}T[A], \mathfrak{b}{:}T[B]$,*

$$\exists R{\subset} A{\times}B \;.\; \mathfrak{a}(T[R, \mathbf{eq}_{\boldsymbol{Z}}])\mathfrak{b}$$

For specification refinement one wants to establish observational equivalence using simulation relations. The problem at higher order is that there might not exist a simulation relation, even in the presence of observational equivalence.

Example 2. Consider $Sig_{\mathsf{SetCE}} \stackrel{def}{=} \exists X.T_{\mathsf{SetCE}}[X]$, where

$$T_{\mathsf{SetCE}}[X] \stackrel{def}{=} (\mathsf{empty}:X,\ \mathsf{add}:\mathsf{Nat}\to X\to X,\ \mathsf{remove}:\mathsf{Nat}\to X\to X,\ \mathsf{in}:\mathsf{Nat}\to X\to\mathsf{Bool},\ \mathsf{crossover}:(\mathsf{Nat}\to X\to X)\to\mathsf{Nat}\to\mathsf{Bool})$$

and consider $(\mathsf{pack}\,\mathsf{List}_{\mathsf{Nat}}\ \mathfrak{a}):Sig_{\mathsf{SetCE}}$ and $(\mathsf{pack}\,\mathsf{List}_{\mathsf{Nat}}\ \mathfrak{b}):Sig_{\mathsf{SetCE}}$, where

$$\begin{aligned}\mathfrak{a} \stackrel{def}{=} (&\mathsf{empty} = \mathsf{nil},\\ &\mathsf{add} = \textit{cons-uniquesorted} \stackrel{def}{=} \lambda x:\mathsf{Nat}.\lambda l:\mathsf{List}_{\mathsf{Nat}}\ .\\ &\qquad \textit{return } l' \textit{ that is } l \textit{ with } x \textit{ uniquely inserted before first } y > x,\\ &\mathsf{remove} = \textit{del-first} \stackrel{def}{=} \lambda x:\mathsf{Nat}.\lambda l:\mathsf{List}_{\mathsf{Nat}}\ .\\ &\qquad\qquad \textit{return } l' \textit{ that is } l \textit{ with first occurrence of } x \textit{ removed},\\ &\mathsf{in} = \textit{in} \stackrel{def}{=} \lambda x:\mathsf{Nat}.\lambda l:\mathsf{List}_{\mathsf{Nat}}\ .\ \textit{return}\ \mathsf{true}\ \textit{if } x \textit{ occurs in } l,\ \mathsf{false}\ \textit{otherwise},\\ &\mathsf{crossover} \stackrel{def}{=} \lambda f:(\mathsf{Nat}\to\mathsf{List}_{\mathsf{Nat}}\to\mathsf{List}_{\mathsf{Nat}}).\lambda n:\mathsf{Nat}\ .\\ &\qquad\qquad \mathsf{in}(n)(f(n)(1::0::\mathsf{nil}))\ )\end{aligned}$$

Here we use the infix symbol :: to denote cons. Furthermore,

$$\begin{aligned}\mathfrak{b} \stackrel{def}{=} (&\mathsf{empty} = \mathsf{nil},\\ &\mathsf{add} = \textit{cons-uniquesorted},\\ &\mathsf{remove} = \textit{del-all} \stackrel{def}{=} \lambda x:\mathsf{Nat}.\lambda l:\mathsf{List}_{\mathsf{Nat}}\ .\\ &\qquad\qquad \textit{return } l' \textit{ that is } l \textit{ with all occurrences of } x \textit{ removed},\\ &\mathsf{in} = \textit{in},\\ &\mathsf{crossover} \stackrel{def}{=} \lambda f:(\mathsf{Nat}\to\mathsf{List}_{\mathsf{Nat}}\to\mathsf{List}_{\mathsf{Nat}}).\lambda n:\mathsf{Nat}\ .\\ &\qquad\qquad \mathsf{in}(n)(f(n)(1::1::0::\mathsf{nil}))\ )\end{aligned}$$

In the parametric minimal model of [11], all elements of the interpretation of $\mathsf{List}_{\mathsf{Nat}}$ and Bool are in correspondence with closed normal forms, and the ω-rule holds. Then observational equivalence holds between $\mathfrak{a}$ and $\mathfrak{b}$, but the existence of a simulation relation does not, because any simulation relation R demands

$$\mathfrak{a}.\mathsf{crossover}\ \ ((\mathsf{eq}_{\mathsf{Nat}}\to R\to R)\to\mathsf{eq}_{\mathsf{Nat}}\to\mathsf{eq}_{\mathsf{Bool}})\ \ \mathfrak{b}.\mathsf{crossover}$$

meaning that $\mathfrak{a}.\mathsf{crossover}(\textit{del-all})(1)$ and $\mathfrak{b}.\mathsf{crossover}(\textit{del-first})(1)$ are to be considered. Note that *del-all* really belongs to $\mathfrak{b}$ and *del-first* belongs to $\mathfrak{a}$.

For failure of composability, in addition to $(\mathsf{pack}\,\mathsf{List}_{\mathsf{Nat}}\ \mathfrak{a})$ and $(\mathsf{pack}\,\mathsf{List}_{\mathsf{Nat}}\ \mathfrak{b})$ above, consider $(\mathsf{pack}\,\mathsf{List}_{\mathsf{Nat}}\ \mathfrak{d}):Sig_{\mathsf{SetCE}}$, where

$$\begin{aligned}\mathfrak{d} \stackrel{def}{=} (&\mathsf{empty} = \mathsf{nil},\\ &\mathsf{add} = \textit{cons},\\ &\mathsf{remove} = \textit{del-all}\\ &\mathsf{in} = \textit{in},\\ &\mathsf{crossover} \stackrel{def}{=} \lambda f:(\mathsf{Nat}\to\mathsf{List}_{\mathsf{Nat}}\to\mathsf{List}_{\mathsf{Nat}}).\lambda n:\mathsf{Nat}\ .\\ &\qquad\qquad \mathsf{in}(n)(f(n)(1::1::0::\mathsf{nil}))\ )\end{aligned}$$

Then the existence of simulation relations holds between $\mathfrak{a}$ and $\mathfrak{d}$, and between $\mathfrak{d}$ and $\mathfrak{b}$, but as before, not between $\mathfrak{a}$ and $\mathfrak{b}$. ○

Thus the standard concept of simulation relation is not adequate. For us, the reason for this is rooted in the encapsulation issue for universal types. The interchangeability of data types in a program translates to interchangeability of packages in virtual computations. Hence, the relevant context is that of any given computation, and therefore simulation relations should account for the encapsulation inherent in such computations. We therefore define,

Definition 6 (*abo*-Simulation Relation). *For* A, B, $\mathfrak{a}{:}T[A], \mathfrak{b}{:}T[B]$,

$$\exists R \subset A \times B \;.\; \mathfrak{a}\ (T[R, \mathbf{eq_Z}])^{T[X]\langle A,B\rangle\langle \mathfrak{a},\mathfrak{b}\rangle}\ \mathfrak{b}$$

Example 2. (continued) We now have

$$\mathfrak{a}.\mathsf{crossover}\ ((\mathsf{eq}_{\mathsf{Nat}} \to R \to R) \to \mathsf{eq}_{\mathsf{Nat}} \to \mathsf{eq}_{\mathsf{Bool}})^{T_{\mathsf{SetCE}}\langle \mathsf{List}_{\mathsf{Nat}}, \mathsf{List}_{\mathsf{Nat}}\rangle\langle \mathfrak{a},\mathfrak{b}\rangle}\ \mathfrak{b}.\mathsf{crossover}$$

Only $\gamma, \delta{:}\,\mathsf{Nat} \to \mathsf{List}_{\mathsf{Nat}} \to \mathsf{List}_{\mathsf{Nat}}$ such that $\mathsf{Dfnbl}^{T_{\mathsf{SetCE}}\langle \mathsf{List}_{\mathsf{Nat}}, \mathsf{List}_{\mathsf{Nat}}\rangle\langle \mathfrak{a},\mathfrak{b}\rangle}_{\mathsf{Nat}\to X\to X}(\gamma, \delta)$, *i.e.,*

$$\exists f{:}\forall X.T_{\mathsf{SetCE}}[X] \to (\mathsf{Nat} \to X \to X) \;.\; (f\ \mathsf{List}_{\mathsf{Nat}}\ \mathfrak{a}) = \gamma \wedge (f\ \mathsf{List}_{\mathsf{Nat}}\ \mathfrak{b}) = \delta$$

are considered. This excludes *e.g.,* the cross-over pair (*del-all*, *del-first*). ○

In the following, using *abo*-simulation relations we establish the essential properties that do not hold for standard simulation relations. In the context of data types, we adhere to the following reasonable assumption.

HADT$_{Obs}$**:** Every profile $T_i[X] = T_{i1}[X] \to \cdots \to T_{n_i}[X] \to T_{c_i}[X]$ of an abstract type $\exists X.T[X]$ is such that $T_{c_i}[X]$ is either X or some $D \in Obs$.

Theorem 5. *Assuming* ***HADT***$_{Obs}$ *for* $T[X]$*, we get with abo-*Param *and abo-*Iel*,*

$$\begin{array}{l}\forall A, B.\forall \mathfrak{a}{:}T[A], \mathfrak{b}{:}T[B] \;. \\ \quad \exists R \subset A \times B \;.\; \mathfrak{a}\ T[R, \mathbf{eq_Z}]^{T[X]\langle A,B\rangle\langle \mathfrak{a},\mathfrak{b}\rangle}\ \mathfrak{b} \\ \qquad\qquad \Leftrightarrow\ \ \bigwedge_{D \in Obs} \forall f{:}\forall X.(T[X] \to D) \;.\; (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b})\end{array}$$

Proof: $\Rightarrow$: By *abo*-Param $f\ (\forall X.T[X, \mathbf{eq_Z}] \to D)^{\mathsf{abo}}\ f$ and Theorem 2.

$\Leftarrow$: Let $s \stackrel{def}{=} T[X]\langle A, B\rangle\langle \mathfrak{a}, \mathfrak{b}\rangle$. We must derive $\exists R \subset A \times B \;.\; \mathfrak{a}(T[R, \mathbf{eq_Z}]^s)\mathfrak{b}$. We exhibit $R \stackrel{def}{=} (a{:}A, b{:}B) \;.\; (\mathsf{Dfnbl}^s_X(a, b))$. By Theorem 4, we must for every component $g{:}U_1 \to \cdots \to U_l \to U_c$ in $T[X]$, show the derivability of

$$\begin{array}{l}\forall x_1{:}U_1[A], \ldots, x_l{:}U_l[A] \;.\; \forall y_1{:}U_1[B], \ldots, y_l{:}U_l[B] \;. \\ \qquad \bigwedge_{1 \le i \le l}(x_i\ U_i[R, \mathbf{eq_Z}]^s\ y_i\ \wedge\ \mathsf{Dfnbl}^s_{U_i}(x_i, y_i)) \\ \qquad\qquad \Rightarrow\ \ (\mathfrak{a}.g\,x_1 \cdots x_l)\ U_c[R, \mathbf{eq_Z}]^s\ (\mathfrak{b}.g\,y_1 \cdots y_l)\end{array}$$

We get $\exists f_{U_i} : \forall X.(T[X] \to U_i[X]) \;.\; (f_{U_i} A\,\mathfrak{a}) = x_i \wedge (f_{U_i} B\,\mathfrak{b}) = y_i$ from the antecedent $\mathsf{Dfnbl}^s_{U_i}(x_i, y_i)$. Let $f \stackrel{def}{=} \Lambda X.\lambda \mathfrak{x}{:}T[X] \;.\; (\mathfrak{x}.g(f_{U_1} X\mathfrak{x}) \cdots (f_{U_l} X\mathfrak{x}))$.

$U_c = D \in Obs$: By Theorem 2 it suffices to derive $\mathfrak{a}.g\,x_1 \cdots x_l =_D \mathfrak{b}.g\,y_1 \cdots y_l$. The assumption gives $(f\,A\,\mathfrak{a}) =_D (f\,B\,\mathfrak{b})$, which gives the result.

Suppose $U_c = X$: We must then derive

$$\exists f{:}\forall X.(T[X] \to U_c[X]) \;.\; (f\,A\,\mathfrak{a}) = (\mathfrak{a}.g\,x_1 \cdots x_l) \wedge (f\,B\,\mathfrak{b}) = (\mathfrak{b}.g\,y_1 \cdots y_l)$$

For this we display f above. □

Theorem 6. *Assuming* $\mathtt{HADT}_{Obs}$ *for* $T[X]$*, we get with abo-*PARAM *and abo-*IEL*,*

$$\begin{array}{l}\forall A,B,C,\ R\subset A\times B,\ S\subset B\times C,\ \mathfrak{a}{:}T[A],\mathfrak{b}{:}T[B],\mathfrak{c}{:}T[C].\\ \qquad \mathfrak{a}(T[R,\mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle A,B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle})\mathfrak{b}\ \wedge\ \mathfrak{b}(T[S,\mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle B,C\rangle\langle\mathfrak{b},\mathfrak{c}\rangle})\mathfrak{c}\\ \qquad\qquad \Rightarrow\ \mathfrak{a}(T[S\circ R,\mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle A,C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle})\mathfrak{c}\end{array}$$

Proof: The goal is to derive for every component $g{:}U_1\to\cdots\to U_l\to U_c$ in T,

$$\begin{array}{l}\forall x_1{:}U_1[A],\ldots,x_l{:}U_l[A]\ .\ \forall z_1{:}U_1[C],\ldots,z_l{:}U_l[C]\ .\\ \quad \bigwedge_{1\le i\le l}(x_i\ U_i[S\circ R,\mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle A,C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle}\ z_i\ \wedge\ \mathsf{Dfnbl}_{U_i}^{T[X]\langle A,C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle}(x_i,z_i))\\ \qquad\qquad \Rightarrow\ \ (\mathfrak{a}.g\,x_1\cdots x_l)\ U_c[S\circ R,\mathbf{eq}_{\boldsymbol{Z}}]^{T[X]\langle A,C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle}\ (\mathfrak{c}.g\,z_1\cdots z_l)\end{array}$$

$\mathsf{Dfnbl}_{U_i}^{T[X]\langle A,C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle}(x_i,z_i)$ gives an $f\stackrel{def}{=}\Lambda X.\lambda\mathfrak{x}{:}T[X]\ .\ (\mathfrak{x}.g(f_{U_1}X\mathfrak{x})\cdots(f_{U_l}X\mathfrak{x}))$.

$U_c=D\in Obs$: By assumption and Theorem 5, $(f\,A\,\mathfrak{a})=(f\,B\,\mathfrak{b})=(f\,C\mathfrak{c})$, and $\mathfrak{a}.g\,x_1\cdots x_l=(f\,A\,\mathfrak{a})$ and $(f\,C\mathfrak{c})=\mathfrak{c}.g\,z_1\cdots z_l$.

$U_c=X$: We must show $\exists b{:}U_c[B,\boldsymbol{Z}]\ .\ (\mathfrak{a}.g\,x_1\cdots x_l)\ R\ b\wedge b\ S\ (\mathfrak{c}.g\,z_1\cdots z_l)$. Exhibit $f\,B\,\mathfrak{b}=(\mathfrak{b}.g(f_{U_1}B\mathfrak{b})\cdots(f_{U_l}B\mathfrak{b}))$ for b. □

Conventional simulation relations and relational parametricity do not give corresponding results to Theorem 5; the coincidence of the existence of simulation relations with observational equivalence, and Theorem 6; the composability of simulation relations, except for data types with first-order operations.

5 *abo*-Semantics

We present a *per*-model for the relational logic with *abo*-PARAM and *abo*-IEL. Just as the parametric *per*-model [2] is defined directly according to PARAM, the *abo*-parametric *per*-model will be defined according to *abo*-PARAM.

We use an obvious shorthand notation and simply write things like $U[\mathcal{A},\boldsymbol{\mathcal{Z}}]$ instead of $[\![A,\boldsymbol{Z}\rhd U[A,\boldsymbol{Z}]]\!]_{[A\mapsto\mathcal{A},\boldsymbol{Z}\mapsto\boldsymbol{\mathcal{Z}}]}$, and also things like $U[\mathcal{R},\boldsymbol{\mathcal{Z}}]^{\sigma\langle\mathcal{A},\mathcal{B}\rangle}$ for $[\![A,B,R\subset A\times B,\boldsymbol{Z}\rhd U[R,\mathbf{eq}_{\boldsymbol{Z}}]^{\sigma\langle A,B\rangle}]\!]_{[A\mapsto\mathcal{A},B\mapsto\mathcal{B},\boldsymbol{Z}\mapsto\boldsymbol{\mathcal{Z}},R\mapsto\mathcal{R}]}$.

First, the semantics of a universal type $\forall X.U$ in the pure parametric *per*-model, *e.g.,* [29], is $(\cap_{\mathcal{A}\in\mathrm{PER}}[\![U[\mathcal{A}]]\!])$. This gives Strachey's parametricity. In the relational parametric *per*-model, the semantics $(\cap_{\mathcal{A}}U[\mathcal{A},\boldsymbol{\mathcal{Z}}])^\flat$ is obtained by including only those elements which satisfy relational parametricity, thus

$$\begin{array}{l}n\ (\cap_{\mathcal{A}}U[\mathcal{A},\boldsymbol{\mathcal{Z}}])^\flat\ m\ \stackrel{def}{\Leftrightarrow}\ \forall\mathcal{A},\mathcal{B}\in\mathrm{PER},\ \textit{saturated}\ \mathcal{R}\subset \mathit{Dom}(\mathcal{A})\times\mathit{Dom}(\mathcal{B})\ .\\ \qquad\qquad n\ U[\mathcal{A},\boldsymbol{\mathcal{Z}}]\ m\wedge n\ U[\mathcal{B},\boldsymbol{\mathcal{Z}}]\ m\ \wedge\\ \qquad\qquad n\ U[\mathcal{R},\boldsymbol{\mathcal{Z}}]\ n\wedge m\ U[\mathcal{R},\boldsymbol{\mathcal{Z}}]\ m\end{array}$$

Alternatively, we may give the following definition, since relational parametricity is equivalent to identity extension at universal type.

$$\begin{array}{l}n\ (\cap_{\mathcal{A}}U[\mathcal{A},\boldsymbol{\mathcal{Z}}])^\flat\ m\ \stackrel{def}{\Leftrightarrow}\ \forall\mathcal{A},\mathcal{B}\in\mathrm{PER},\ \textit{saturated}\ \mathcal{R}\subset \mathit{Dom}(\mathcal{A})\times\mathit{Dom}(\mathcal{B})\ .\\ \qquad\qquad n\ U[\mathcal{R},\boldsymbol{\mathcal{Z}}]\ m\end{array}$$

A model satisfying *abo*-PARAM and *abo*-IEL is then obtained by defining

$$n\ (\cap_{\mathcal{A}} U[\mathcal{A}, \boldsymbol{\mathcal{Z}}])^{\flat^{\mathsf{abo}}}\ m \overset{def}{\Leftrightarrow} \forall \mathcal{A}, \mathcal{B} \in \text{PER},\ \textit{saturated}\ \mathcal{R} \subset Dom(\mathcal{A}) \times Dom(\mathcal{B})\ .\ n\ U[\mathcal{R}, \boldsymbol{\mathcal{Z}}]_{\top}^{\sigma\langle \mathcal{A}, \mathcal{B}\rangle}\ m$$

where σ depends on U according to Def. 1. The rest of the semantics is standard *per*-semantics, see *e.g.,* [29, 2]. We must now show that this *abo*-parametric structure is a model for System F; every closed term has an interpretation. Intuitively, this should be so, since *abo*-relations are defined according to `Abs-Bar1`, which in particular captures what closed polymorphic functionals look like.

Theorem 7. *Every closed term of System F has an interpretation in the abo-parametric per-structure.*

Proof: The interesting part of this is the semantics for universal types. Thus, show for every closed $f{:}\forall X.U$, that $f\ (\forall X.U[X])^{\mathsf{abo}}\ f$ holds in the structure. □

6 Final Remarks

We have provided a notion of relational parametricity that takes into account the abstraction barrier-observing (*abo*) mechanism in universal types. We showed how this then gives the notion of *abo*-simulation relation from [9, 10], which resolves serious well-known problems for simulation relations in the framework of stepwise refinement. Furthermore, we introduced, asserted, and validated *abo*-relational parametricity. This massively simplifies both our task as formalists, as well as decreases the amount of reasoning to which a developer needs to commit, when using *abo*-simulation relations for refinement.

There is a link under *abo*-relational parametricity between *abo*-simulation relations and equality at existential type. This link exists for standard simulation relations under standard relational parametricity [25], but in that case there is a slightly annoying circularity issue. This issue is not present in the *abo* case. This is a consequence of Theorem 5. Also, relational parametricity gives induction, *e.g.,* on Nat, and so does *abo*-relational parametricity.

There are lots of further questions, *e.g.,* does the *abo*-parametric *per*-model have polymorphic functionals that are not closed-term denotable, and what else can be shown using *abo*-parametricity that cannot be analogously shown using regular parametricity; and *vice versa*, in particular w.r.t. universal constructions. Present research includes a comparison of *abo*-simulation relations to the prelogical relations of [14].

Acknowledgements Don Sannella, Martin Hofmann, Uday Reddy, and David Aspinall have given essential feed-back on early versions of ideas in this paper. The anonymous referees have given further detailed and valuable comments.

References

1. R.-J. Back and J. Wright. *Refinement Calculus, A Systematic Introduction*. Graduate Texts in Computer Science. Springer Verlag, 1998.
2. E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.
3. H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, *eds.*, *Handbook of Logic in Computer Science*, volume 2, pages 118–309. Oxford University Press, 1992.
4. C. Böhm and A. Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.
5. O.-J. Dahl. *Verifiable Programming, Revised version 1993*. Prentice Hall Int. Series in Computer Science; C.A.R. Hoare, Series Editor. Prentice-Hall, UK, 1992.
6. J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
7. J. Hannay. Specification refinement with System F. In *Computer Science Logic. Proc. of CSL'99*, vol. 1683 of *Lecture Notes in Comp. Sci.*, pages 530–545. Springer Verlag, 1999.
8. J. Hannay. Specification refinement with System F, the higher-order case. In *Recent Trends in Algebraic Development Techniques. Selected Papers from WADT'99*, volume 1827 of *Lecture Notes in Comp. Sci.*, pages 162–181. Springer Verlag, 1999.
9. J. Hannay. A higher-order simulation relation for System F. In *Foundations of Software Science and Computation Structures. Proc. of FOSSACS 2000*, vol. 1784 of *Lecture Notes in Comp. Sci.*, pages 130–145. Springer Verlag, 2000.
10. J. Hannay. *Abstraction Barriers and Refinement in the Polymorphic Lambda Calculus*. PhD thesis, Laboratory for Foundations of Computer Science (LFCS), University of Edinburgh, 2001.
11. R. Hasegawa. Parametricity of extensionally collapsed term models of polymorphism and their categorical properties. In *Theoretical Aspects of Computer Software. Proc. of TACS'91*, vol. 526 of *Lecture Notes in Comp. Sci.*, pages 495–512. Springer Verlag, 1991.
12. C.A.R. Hoare. Proofs of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
13. F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement in typed lambda calculus. In *Foundations of Software Science and Computation Structures. Proc. of FOSSACS 2000*, vol. 1784 of *Lecture Notes in Comp. Sci.*, pages 161–176. Springer Verlag, 2000.
14. F. Honsell and D. Sannella. Prelogical relations. *Information and Computation*, 178:23–43, 2002.
15. Y. Kinoshita, P.W. O'Hearn, J. Power, M. Takeyama, and R.D. Tennent. An axiomatic approach to binary logical relations with applications to data refinement. In *Theoretical Aspects of Computer Software. Proc. of TACS'97*, vol. 1281 of *Lecture Notes in Comp. Sci.*, pages 191–212. Springer Verlag, 1997.
16. Y. Kinoshita and J. Power. Data refinement for call-by-value programming languages. In *Computer Science Logic. Proc. of CSL'99*, vol. 1683 of *Lecture Notes in Comp. Sci.*, pages 562–576. Springer Verlag, 1999.
17. Q. Ma and J.C. Reynolds. Types, abstraction and parametric polymorphism, part 2. In *Mathematical Foundations of Programming Semantics, Proc. of MFPS*, vol. 598 of *Lecture Notes in Comp. Sci.*, pages 1–40. Springer Verlag, 1991.

18. H. Mairson. Outline of a proof theory of parametricity. In *Functional Programming and Computer Architecture. Proc. of the 5th acm Conf.*, vol. 523 of *Lecture Notes in Comp. Sci.*, pages 313–327. Springer Verlag, 1991.
19. R. Milner. An algebraic definition of simulation between programs. In *Joint Conferences on Artificial Intelligence, Proc. of JCAI*, pages 481–489. Morgan Kaufman Publishers, 1971.
20. J.C. Mitchell. On the equivalence of data representations. In V. Lifschitz, *ed.*, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 305–330. Academic Press, 1991.
21. J.C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. MIT Press, 1996.
22. J.C. Mitchell and G.D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, 1988.
23. C. Morgan. *Programming from Specifications, 2nd ed.* Prentice Hall International Series in Computer Science; C.A.R. Hoare, Series Editor. Prentice-Hall, UK, 1994.
24. P.W. O'Hearn and R.D. Tennent. Relational parametricity and local variables. In *20th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Proceedings*, pages 171–184. ACM Press, 1993.
25. G.D. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications. Proc. of TLCA'93*, vol. 664 of *Lecture Notes in Comp. Sci.*, pages 361–375. Springer Verlag, 1993.
26. G.D. Plotkin, J. Power, D. Sannella, and R.D. Tennent. Lax logical relations. In *Automata, Languages and Programming. Proc. of ICALP 2000*, vol. 1853 of *Lecture Notes in Comp. Sci.*, pages 85–102. Springer Verlag, 2000.
27. J.C. Reynolds. *The Craft of Programming*. Prentice-Hall International, 1981.
28. J.C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83, Proc. of the IFIP 9th World Computer Congress*, pages 513–523. Elsevier Science Publishers B.V. (North-Holland), 1983.
29. E. Robinson. Notes on the second-order lambda calculus. Report 4/92, University of Sussex, School of Cognitive and Computing Sciences, 1992.
30. D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.
31. O. Schoett. Behavioural correctness of data representations. *Science of Computer Programming*, 14:43–57, 1990.
32. C. Strachey. Fundamental concepts in programming languages. Lecture notes from the Int. Summer School in Programming Languages, Copenhagen, 1967.
33. I. Takeuti. An axiomatic system of parametricity. *Fundamenta Informaticae*, 20:1–29, 1998.
34. R.D. Tennent. Correctness of data representations in Algol-like languages. In A.W. Roscoe, *ed.*, *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice Hall International, 1997.