A Higher-Order Simulation Relation for System F

Jo Erskine Hannay

LFCS, Division of Informatics, University of Edinburgh
joh@dcs.ed.ac.uk

Abstract. The notion of data type specification refinement is discussed in a setting of System F and the logic for parametric polymorphism of Plotkin and Abadi. At first order, one gets a notion of specification refinement up to observational equivalence in the logic simply by using Luo's formalism. This paper generalises this notion to abstract data types whose signatures contain higher-order and polymorphic functions. At higher order, the tight connection in the logic between the existence of a simulation relation and observational equivalence ostensibly breaks down. We show that an alternative notion of simulation relation is suitable. This also gives a simulation relation in the logic that composes at higher order, thus giving a syntactic logical counterpart to recent advances on the semantic level.

1 Introduction

The idea behind formal specification refinement is that a program is the end-product of a step-wise refinement process starting from an abstract high-level specification. At each refinement step some design decisions and implementation issues are resolved, and if each refinement step can be proven correct, the resulting program is guaranteed to satisfy the initial specification.

There are several frameworks in which to do this and several ideas of what it is for one specification to be a refinement of another. A prominent framework is that of algebraic specification; see [9] for a survey and comprehensive bibliography. But there has been substantial development in other fields as well, notably in type theory, where also ideas from algebraic specification have been expressed.

This paper investigates specification refinement in a setting consisting of System F and relational parametricity in Reynolds' sense [35, 23] as expressed in Plotkin and Abadi's logic for parametric polymorphism [31]. This setting allows an elegant formalisation of abstract data types as existential types [27]. Moreover, the relational parametricity axiom enables one to derive in the logic that two concrete data types, *i.e.* inhabitants of existential type, are equal if and only if there exists a simulation relation [16] between their implementation parts. Together with the fact that at first order, equality at existential type is derivably equivalent to a notion of observational equivalence, this formalises the semantic proof principle of Mitchell [25]. This lifts the type-theoretic formalism

of refinement due to Luo [22] to a notion in the logic of specification refinement up to observational equivalence; a key issue in program development.

In this paper, we discuss the above type-theoretic notion of specification refinement in more generality, *i.e.* we treat data types whose operations may be higher order and polymorphic. At higher order, the formal link between the existence of a simulation relation and observational equivalence breaks down. Our solution in the logic is to use an alternative notion of simulation relation based on a weaker arrow-type relation. This notion composes at higher-order, thus relating the syntactic level to recent and on-going work on the semantic level remedying the fact that logical relations traditionally used to describe refinement do not compose at higher order [17, 18, 21, 20, 32].

In [12] an account of algebraic specification refinement [38, 37] is mapped to the first-order type-theoretic refinement notion, and the two accounts of refinement are shown to coincide. Important issues in algebraic specification refinement, such as the choice of input sorts [36] and the stability of constructors [39, 37, 10], are automatically resolved in the type-theoretic setting. Other work linking algebraic specification and type theory includes [28, 34, 2, 41, 40]. Relevant work using System F and parametricity includes [29, 30] showing that the introduction of non-terminating recursion also breaks down the tight correspondence between the existence of a simulation relation and observational equivalence.

In [12] a proof method from algebraic specification for proving observational refinements [5, 4, 6] is imported into the type-theory logic by adding axioms postulating the existence of quotients and sub-objects. Work related to this is [33, 42]. The higher-order generalisation of this is to be found in [13].

Section 2 outlines the type theory. In Sect. 3 refinement is introduced in a first-order setting, and Sect. 4 generalises to higher-order and polymorphism.

2 System F and the Logic for Parametric Polymorphism

We briefly recall the parametric λ-calculus System F, and sketch the accompanying logic of [31, 24] for relational parametricity on System F. It is this accompanying logic that bears a relational extension rather than the λ-calculus. See [1] for a more internalised approach. System F has types and terms as follows:

$$T ::= X \mid T \to T \mid \forall X.T \qquad\qquad t ::= x \mid \lambda x{:}T.t \mid tt \mid \Lambda X.t \mid tT$$

where X and x range over type and term variables resp. However, formulae are now built using the usual connectives from equations *and* relation symbols:

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

where R ranges over relation symbols. We write $\alpha[R, X, x]$ to indicate *possible* and *all* occurrences of R, X and x in α, and may write $\alpha[\rho, A, t]$ for the result of substitution, following the appropriate rules concerning capture.

A second-order environment consists of a type environment Δ and a term-environment Γ depending on Δ as usual. For notational convenience we will amalgamate environments into a single environment Γ. Judgements for type and

term formation are as usual. However, formula formation now involves relation symbols, and we therefore employ relation environments, *viz.* a finite sequence Υ of relational typings $R \subset A \times B$ of relation variables, depending on Δ, and obeying standard conventions for environments. The formation rules for atomic formulae consists of the usual one for equations, and now also one for relations:

$$\frac{\Gamma \vdash t{:}A, \quad \Gamma \vdash u{:}B, \quad \Gamma \vdash \Upsilon, \quad \Upsilon \vdash R \subset A \times B}{\Gamma, \Upsilon \vdash R(t,u)\ \mathit{Prop} \qquad \mathit{(also\ written\ tRu)}}$$

The other formation rules for formulae are as one would expect. Relation environments will also be amalgamated into Γ. Relation definition is accommodated:

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

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

If $\rho \subset A \times B$, $\rho' \subset A' \times B'$ and $\rho''[R] \subset A[Y] \times B[Z]$, then complex relations are built by $\rho \to \rho' \subset (A \to A') \times (B \to B')$ where

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

and $\forall(Y, Z, R \subset Y \times Z)\rho''[R] \subset (\forall Y.A[Y]) \times (\forall Z.B[Z])$ where

$$\forall(Y, Z, R \subset Y \times Z)\rho'' \stackrel{def}{=} (y{:}\forall Y.A[Y], z{:}\forall Z.B[Z]).(\forall Y \forall Z \forall R \subset Y \times Z.((yY)\rho''[R](zZ)))$$

One can now acquire further definable relations by substituting definable relations for type variables in types. For $\boldsymbol{X} = X_1, \ldots, X_n$, $\boldsymbol{B} = B_1, \ldots, B_n$, $\boldsymbol{C} = C_1, \ldots, C_n$ and $\boldsymbol{\rho} = \rho_1, \ldots, \rho_n$, where $\rho_i \subset B_i \times C_i$, we get $T[\boldsymbol{\rho}] \subset T[\boldsymbol{B}] \times T[\boldsymbol{C}]$, 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 giving the consequence relation of the logic is natural deduction over formulae now involving relation symbols, and is hence augmented with inference rules for relation symbols, for example we have for Φ a finite set of formulae:

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

We will usually conveniently omit the sequent symbol $\vdash_\Gamma$ henceforth. One also has axioms for equational reasoning and $\beta\eta$ equalities. Finally, the following parametricity axiom schema is asserted:

$$\textsc{Param}: \ \forall Y_1, \ldots, \forall Y_n \forall u{:}(\forall X.T[X, Y_1, \ldots, Y_n])\ .\ u(\forall X.T[X, \mathsf{eq}_{Y_1}, \ldots, \mathsf{eq}_{Y_n}])u$$

To understand, it helps to ignore the parameters Y_i and expand the definition to get $\forall u{:}(\forall X.T[X])\ .\forall Y \forall Z \forall R \subset Y \times Z\ .\ u(Y)\ T[R]\ u(Z)$ *i.e.* if one instantiates a polymorphic inhabitant at two related types then the results are also related. This logic is sound w.r.t. to the parametric PER-model of [3] and the syntactic parametric models of [14]. Crucially, we have the following link to equality:

Fact 1 (Identity Extension Lemma [31]**).** *For any $T[\boldsymbol{Z}]$, the following sequent is derivable using* PARAM.

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

Encapsulation is provided by the following encoding of existential types and the following pack and unpack combinators.

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

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

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

We omit subscripts to pack and unpack as much as possible. Operationally, pack packages a data representation and an implementation of operators on that data representation. The resulting package is a polymorphic functional that given a client and its result domain, instantiates the client with the particular elements of the package. And unpack is the application operator for pack.

Fact 2 (Characterisation by Simulation Relation [31]**).** *The following sequent schema is derivable using* PARAM.

$$\begin{array}{l}\forall \boldsymbol{Z}.\forall u,v{:}\exists X.T[X,\boldsymbol{Z}]\ .\\ \qquad u =_{\exists X.T[X,\boldsymbol{Z}]} v \quad\Leftrightarrow\quad \exists A,B.\exists \mathfrak{a}{:}T[A,\boldsymbol{Z}],\mathfrak{b}{:}T[B,\boldsymbol{Z}].\exists R\subset A\times B\ .\\ \qquad\qquad u=(\mathsf{pack}A\mathfrak{a})\ \wedge\ v=(\mathsf{pack}B\mathfrak{b})\ \wedge\ \mathfrak{a}(T[R,\mathbf{eq}_{\boldsymbol{Z}}])\mathfrak{b}\end{array}$$

The sequent in Fact 2 states the equivalence of equality at existential type with the existence of a simulation relation in the sense of [25]. From this we also get

$$\forall \boldsymbol{Z}.\forall u{:}\exists X.T[X,\boldsymbol{Z}].\exists A.\exists \mathfrak{a}{:}T[A,\boldsymbol{Z}]\ .\ u=(\mathsf{pack}A\,\mathfrak{a})$$

Weak versions of standard constructs such as products, initial and final (co-)algebras are encodable in System F [7]. With PARAM, these constructs are provably universal constructions. We can *e.g.* freely use product types. Given $\rho\subset A\times B$ and $\rho'\subset A'\times B'$, $(\rho\times\rho)$ is defined as the action $(X\times X')[\rho,\rho']$. One derives $\forall u{:}A\times A', v{:}B\times B'\ .\ u(\rho\times\rho')v\ \Leftrightarrow\ (\mathsf{fst}(u)\ \rho\ \mathsf{fst}(v)\ \wedge\ \mathsf{snd}(u)\ \rho\ \mathsf{snd}(v))$. We use the abbreviations $\mathsf{bool} \stackrel{def}{=} \forall X.X\to X\to X$, $\mathsf{nat} \stackrel{def}{=} \forall X.X\to(X\to X)\to X$, and $\mathsf{list}(A) \stackrel{def}{=} \forall X.X\to(A\to X\to X)\to X$. These inductive types are provably initial constructs.

3 Data Type Specification and First-Order Results

Existential types provide a nice way of specifying abstract data types [27]. In System F and the accompanying logic of [31], this mode of specification leads to

specification up to observational equivalence, where the latter is defined w.r.t. some given finite set *Obs* of closed inductive types for which the Identity Extension Lemma (Fact 1) gives $x\,(C[\rho])\,y \;\Leftrightarrow\; x =_C y$. Examples are bool and nat. In the following we shall use record type notation as a notational convenience.

Definition 1 (Abstract Data Type Specification). *An* abstract data type specification *SP is a tuple*

$$\langle\langle Sig_{SP}, \Theta_{SP}\rangle, Obs\rangle$$

where $Sig_{SP} = \exists X.\mathfrak{T}_{SP}[X]$, *for* $\mathfrak{T}_{SP}[X] = Record(f_1{:}T_1,\ \ldots\ , f_k{:}T_k)$, *and where* $\Theta_{SP}(u) = \exists X.\exists \mathfrak{x}{:}\mathfrak{T}_{SP}[X]\ .\ u = (\mathsf{pack}X\mathfrak{x})\ \wedge\ \Phi_{SP}[X,\mathfrak{x}]$.

If $\Theta_{SP}(u)$ *is derivable, then u is said to be a* realisation of *SP.*

Example 1. For example $\mathsf{Stack} \stackrel{def}{=} \langle\langle Sig_{\mathsf{Stack}}, \Theta_{\mathsf{Stack}}\rangle, \{\mathsf{nat}\}\rangle$, where
$Sig_{\mathsf{Stack}} = \exists X.\mathfrak{T}_{\mathsf{Stack}}[X]$,
$\mathfrak{T}_{\mathsf{Stack}}[X] = Record(\mathsf{empty}{:}X, \mathsf{push}{:}\,\mathsf{nat}\times X \to X, \mathsf{pop}{:}X \to X, \mathsf{top}{:}X \to \mathsf{nat})$,
$\Theta_{\mathsf{Stack}}(u) = \exists X.\exists \mathfrak{x}{:}\mathfrak{T}_{\mathsf{Stack}}[X]\ .\ u = (\mathsf{pack}X\mathfrak{x})\ \wedge$
$\forall x{:}\mathsf{nat}, s{:}X\ .\ \mathfrak{x}.\mathsf{pop}(\mathfrak{x}.\mathsf{push}(x,s)) = s\ \wedge$
$\forall x{:}\mathsf{nat}, s{:}X\ .\ \mathfrak{x}.\mathsf{top}(\mathfrak{x}.\mathsf{push}(x,s)) = x$ ○

We reserve $\mathfrak{T}[X]$ for the function-profile part of abstract data types $\exists X.\mathfrak{T}[X]$. For brevity, in this paper we do not consider parameterised specifications and so assume X to be the only free type variable in $\mathfrak{T}[X]$.

The notion of specification of Def. 1 resembles that of [22]. However, as we are about to see, the important difference is that here equality of data-type inhabitants is inherently behavioural, and implementation is up to observational equivalence. In analogy to the meta-level notion in [25], we define observational equivalence in terms of observable computations in the logic as follows.

Definition 2 (Observational Equivalence (ObsEq)**).** *Define* observational equivalence ObsEq w.r.t. *Obs in the logic by*

$\mathsf{ObsEq} \stackrel{def}{=} (u{:}\exists X.\mathfrak{T}[X], v{:}\exists X.\mathfrak{T}[X]).$
$(\exists A, B.\exists \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B]\ .\ u = (\mathsf{pack}A\mathfrak{a})\ \wedge\ v = (\mathsf{pack}B\mathfrak{b})\ \wedge$
$\bigwedge_{C\in Obs} \forall f{:}\forall X.(\mathfrak{T}[X] \to C)\ .\ (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b}))$

The first result is essential to understanding the notion of specification in Def. 1. It is a syntactic counterpart to a semantic result in [25, 26].

Theorem 3 ([12]). *Suppose* $\langle\langle\exists X.\mathfrak{T}[X], \Theta\rangle, Obs\rangle$ *is an abstract data type specification such that* $\mathfrak{T}[X]$ *only contains first-order function profiles. Then, assuming* PARAM, *equality at existential type is derivably equivalent to observational equivalence, i.e. the following is derivable in the logic.*

$$\forall u, v{:}\exists X.\mathfrak{T}[X]\ .\ u =_{\exists X.\mathfrak{T}[X]} v \quad\Leftrightarrow\quad u\ \mathsf{ObsEq}\ v$$

Proof: This follows from Fact 2 and Theorem 4 below. □

Theorem 4 ([12]). *Let* $\exists X.\mathfrak{T}[X]$ *be as in Theorem 3. Then, assuming* PARAM, *the existence of a simulation relation is derivably equivalent to observational equivalence, i.e. the following is derivable.*

$$\forall A, B.\forall \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B]\ .\ \exists R \subset A\times B\ .\ \mathfrak{a}(\mathfrak{T}[R])\mathfrak{b} \quad\Leftrightarrow\quad \bigwedge_{C\in Obs} \forall f{:}\forall X.(\mathfrak{T}[X]\to C)\ .\ (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b})$$

Proof: $\Rightarrow$: This follows from PARAM.

$\Leftarrow$: We must exhibit an R such that $\mathfrak{a}(\mathfrak{T}[R])\mathfrak{b}$. Semantically, [25, 26, 39] relate elements iff they are denotable by some common term. We mimic this: For R give $\mathsf{Dfnbl} \stackrel{def}{=} (a{:}A, b{:}B).(\exists f{:}\forall X.(\mathfrak{T}[X]\to X).(f\,A\,\mathfrak{a}) = a \ \wedge\ (f\,B\,\mathfrak{b}) = b)$. □

Given Theorem 3, $\Theta_{SP}(u)$ of Def. 1 expresses "u is observationally equivalent to a package $(\mathsf{pack}X\mathfrak{x})$ that satisfies the axioms Φ_{SP}". Hence specification according to Def. 1 is up to observational equivalence.

Notice that there is nothing hindering having free variables in an observable computation $f{:}\forall X.(\mathfrak{T}[X]\to C)$. Importantly, though, these free variables can not be of the existentially bound type.

Example 2 ([15]). Consider specification $\mathsf{Set} \stackrel{def}{=} \langle\langle Sig_{\mathsf{Set}}, \Theta_{\mathsf{Set}}\rangle, \{\mathsf{bool}, \mathsf{nat}\}\rangle$, for $Sig_{\mathsf{Set}} = \exists X.\mathfrak{T}_{\mathsf{Set}}[X]$,

$\mathfrak{T}_{\mathsf{Set}}[X] = \mathit{Record}(\mathsf{empty}{:}X, \mathsf{add}{:}\mathsf{nat}{\times}X{\to}X, \mathsf{remove}{:}\mathsf{nat}{\times}X{\to}X, \mathsf{in}{:}\mathsf{nat}{\times}X{\to}\mathsf{bool})$,

$\Theta_{\mathsf{Set}}(u) = \exists X.\exists \mathfrak{x}{:}\mathfrak{T}_{\mathsf{Set}}[X]\ .\ u = (\mathsf{pack}X\mathfrak{x})\ \wedge$

$\forall x{:}\mathsf{nat}, s{:}X\ .\ \mathfrak{x}.\mathsf{add}(x, \mathfrak{x}.\mathsf{add}(x,s)) = \mathfrak{x}.\mathsf{add}(x,s)\ \wedge$

$\forall x, y{:}\mathsf{nat}, s{:}X\ .\ \mathfrak{x}.\mathsf{add}(x, \mathfrak{x}.\mathsf{add}(y,s)) = \mathfrak{x}.\mathsf{add}(y, \mathfrak{x}.\mathsf{add}(x,s))\ \wedge$

$\forall x{:}\mathsf{nat}\ .\ \mathfrak{x}.\mathsf{in}(x, \mathfrak{x}.\mathsf{empty}) = \mathsf{false}\ \wedge$

$\forall x, y{:}\mathsf{nat}, s{:}X\ .\ \mathfrak{x}.\mathsf{in}(x, \mathfrak{x}.\mathsf{add}(y,s)) = \mathsf{if}\ x =_{\mathsf{nat}} y\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathfrak{x}.\mathsf{in}(x,s)\ \wedge$

$\forall x{:}\mathsf{nat}, s{:}X\ .\ \mathfrak{x}.\mathsf{in}(x, \mathfrak{x}.\mathsf{remove}(x,s)) = \mathsf{false}$

Consider the data type $LI \stackrel{def}{=} (\mathsf{pack}\ \mathsf{list}(\mathsf{nat})\ \mathfrak{l}){:}\, Sig_{\mathsf{Set}}$, where $\mathfrak{l}.\mathsf{empty}$ gives the empty list, $\mathfrak{l}.\mathsf{add}$ adds a given element to the end of a list only if the element does not occur in the list, $\mathfrak{l}.\mathsf{in}$ is the occurrence function, and $\mathfrak{l}.\mathsf{remove}$ removes the first occurrence of a given element. Typing allows users of LI to only build lists using $\mathfrak{l}.\mathsf{empty}$ and $\mathfrak{l}.\mathsf{add}$, and on such lists the efficient $\mathfrak{l}.\mathsf{remove}$ gives the intended result. Crucially, any closed observation $f{:}\forall X.(\mathfrak{T}_{\mathsf{Set}}[X]\to C)$, $C \in Obs$ can only refer to lists built using $\mathfrak{l}.\mathsf{empty}$ and $\mathfrak{l}.\mathsf{add}$. For example, in the observable computation $\Lambda X.\lambda \mathfrak{x}{:}\mathfrak{T}_{\mathsf{Set}}[X]\ .\ \mathfrak{x}.\mathsf{in}(x, \mathfrak{x}.\mathsf{remove}(x,g))$, where g is a term of the bound type X, the typing rules insist that g can only be of the form $\mathfrak{x}.\mathsf{add}(\cdots\mathfrak{x}.\mathsf{add}(\mathfrak{x}.\mathsf{empty})\cdots)$ and not a free variable. This implies through Theorem 3 that LI is a realisation of Set according to Def. 1.

In the world of algebraic specification, there is no formal restriction on the set In of so-called input-sorts. Thus, if one chooses the set of input sorts to be $In = \{\mathsf{set}, \mathsf{bool}, \mathsf{nat}\}$, then $\mathsf{in}(x, \mathsf{remove}(x,s))$ where s is a variable, is an observable computation. This computation might give true, since s ranges over all lists. In algebraic specification one has to explicitly restrict input sorts to not include the abstract sort, in this case set, when defining observational equivalence [36], whereas the type-theoretic formalism deals with this automatically. ○

The idea of specification refinement up to observational equivalence can now be expressed straight-forwardly by simply using the notion of refinement in [22].

Definition 3 (Type Theory Specification Refinement). *A specification SP' is a* refinement *of specification SP,* via constructor $F\colon Sig_{SP'} \to Sig_{SP}$ *if*

$$\forall u\colon Sig_{SP'} \;.\; \Theta_{SP'}(u) \;\Rightarrow\; \Theta_{SP}(Fu)$$

is derivable. We write $SP \overset{\sim}{\to}_F SP'$ for this fact.

The notion of constructor $F\colon Sig_{SP'} \to Sig_{SP}$ in Def. 3 is based on the notion of parameterised program [10]. Given a program P that is a realisation of SP', the instantiation $F(P)$ is then a realisation of SP. Constructors correspond to refinement maps in [22]. It is evident that the refinement relation of Def. 3 is in a sense transitive, *i.e.* we have *vertical composability* [11]:

$$SP \overset{\sim}{\to}_F SP' \;\text{ and }\; SP' \overset{\sim}{\to}_{F'} SP'' \;\Rightarrow\; SP \overset{\sim}{\to}_{F \circ F'} SP''$$

where $F \circ F' \stackrel{def}{=} \lambda u\colon Sig_{SP''}.F(F'u)$. In terms of algebraic-specification, any constructor $F\colon Sig_{SP'} \to Sig_{SP}$ is by Theorem 3 inherently stable under parametricity: Congruence gives $\forall u, v\colon Sig_{SP'} \;.\; u =_{Sig_{SP'}} v \;\Rightarrow\; F(u) =_{Sig_{SP}} F(v)$. And equality at existential type is of course observational equivalence.

Relating data types by simulation relations is often called *data refinement.* There are thus two refinement dimensions; one concerning specifications, and within each stage of this refinement process, a second dimension concerning observational equivalence, *i.e.* simulation relations, *i.e.* data refinement. At first order, theorems 3 and 4 give the essential property that the existence of simulation relations is transitive, but we can actually give a more constructive result:

Theorem 5 (Composability of Simulation Relations). *Suppose $\mathfrak{T}[X]$ only contains first-order function profiles. Then we can derive*

$$\forall A, B, G, R \subset A \times B, S \subset B \times G, \mathfrak{a}\colon \mathfrak{T}[A], \mathfrak{b}\colon \mathfrak{T}[B], \mathfrak{g}\colon \mathfrak{T}[G].$$
$$\mathfrak{a}(\mathfrak{T}[R])\mathfrak{b} \;\wedge\; \mathfrak{b}(\mathfrak{T}[S])\mathfrak{g} \;\Rightarrow\; \mathfrak{a}(\mathfrak{T}[S \circ R])\mathfrak{g}$$

4 Higher Order

If $\mathfrak{T}[X]$ has higher-order function profiles, Theorem 4 fails due to Dfnbl not extending to a logical relation. Theorem 5 fails as well, and indeed we cannot even derive that the existence of simulation relations is transitive.

The solution we present here is based on an alternative notion of simulation relation, and is motivated as follows. Consider the higher-order signature $\exists X.Record(f : (X \to X) \to \mathsf{nat}, g : X \to X)$. One requirement for an $R \subset A \times B$ to be respected in the standard sense by two implementations $\mathfrak{a}$ and $\mathfrak{b}$, is that $\forall \delta\colon A \to A, \forall \gamma\colon B \to B \;.\; \delta(R \to R)\gamma \;\Rightarrow\; \mathfrak{a}.f(\delta) =_{\mathsf{nat}} \mathfrak{b}.f(\gamma)$. But since f is defined within a package, f should be specific to that package, and f's behaviour on elements outside the package should be irrelevant. Therefore the proof obligation should not have to consider the behaviour of

$\mathfrak{a}.f$ and $\mathfrak{b}.f$ on arbitrary operators $\delta\colon A \to A$ and $\gamma\colon B \to B$ as long as their behaviour satisfies the requirement for operators defined in terms of $\mathfrak{a}.g$ and $\mathfrak{b}.g$ and operators of globally accessible types. This view is partly what the type system promotes through existential types: Operationally, the only way two concrete data types ($\mathsf{pack}A\mathfrak{a}$) and ($\mathsf{pack}B\mathfrak{b}$) can be used is in clients of the form $\Lambda X.\lambda\mathfrak{x}\colon \mathit{Record}(f\colon (X \to X) \to \mathsf{nat}, g\colon X \to X)\ .\ t$. Such a client can incite the application of $\mathfrak{a}.f$ and $\mathfrak{b}.f$ to $\mathfrak{a}.g$ and $\mathfrak{b}.g$ resp., but not to arbitrary $\delta\colon A \to A$ and $\gamma\colon B \to B$. Existential types therefore provide an abstraction barrier to which the standard definition of type relations is in a certain sense oblivious, and we suggest altering the relational proof criteria accordingly.

As before $\mathfrak{T}[X]$ denotes the body of an abstract data type $\exists X.\mathfrak{T}[X]$, now possibly with higher-order and polymorphic profiles. We shall assume that

***adt*:** $\mathfrak{T}[X] = \mathit{Record}(f_1\colon T_1[X], \ldots, f_k\colon T_k[X])$, where each $f_i\colon T_i[X]$ is in uncurried form, *i.e.* $T_i[X]$ is of the form $T_{i1}[X] \times \cdots \times T_{n_i}[X] \to T_{c_i}[X]$, where $T_{c_i}[X]$ is not an arrow type. If $T_{c_i}[X]$ is a universal type, then $T_{c_i}[X] \in \mathit{Obs}$.

4.1 The Alternative Simulation Relation

For brevity we will abuse vector notation. For a k-ary vector $\boldsymbol{Y}$, we write *e.g.* $\forall\boldsymbol{Y}$ for the string $\forall Y_1.\forall Y_2.\ldots\forall Y_k$, and similarly for $\boldsymbol{\Lambda Y}$. If $k = 0$ then the above all denote the empty string. The first l components of $\boldsymbol{Y}$ are denoted by $\boldsymbol{Y}|_l$.

Definition 4 (Data Type Relation). *For $\mathfrak{T}[X]$, for k-ary $\boldsymbol{Y}$, l-ary, $l \geq k$, $\boldsymbol{E}$, $\boldsymbol{F}$, $\boldsymbol{\rho} \subset \boldsymbol{E} \times \boldsymbol{F}$, A, B, $R \subset A \times B$, $\mathfrak{a}\colon \mathfrak{T}[A]$, $\mathfrak{b}\colon \mathfrak{T}[B]$, we define the* data type relation $U[\boldsymbol{\rho}, R]^\star$ *inductively by*

$$
\begin{array}{ll}
U = X & : U[\boldsymbol{\rho}, R]^\star \stackrel{def}{=} R \\
U = Y_i & : U[\boldsymbol{\rho}, R]^\star \stackrel{def}{=} \rho_i \\
U = \forall X'.U'[\boldsymbol{Y}, X', X] & : U[\boldsymbol{\rho}, R]^\star \stackrel{def}{=} \\
 & \qquad \forall(E_{l+1}, F_{l+1}, \rho_{l+1} \subset E_{l+1} \times F_{l+1})(U'[\boldsymbol{\rho}, \rho_{l+1}, R]^\star) \\
U = U' \to U'' & : U[\boldsymbol{\rho}, R]^\star \stackrel{def}{=} \\
\multicolumn{2}{l}{(g\colon U'[\boldsymbol{E}, A] \to U''[\boldsymbol{E}, A],\ h\colon U'[\boldsymbol{F}, B] \to U''[\boldsymbol{F}, B])\ .\ (\forall x\colon U'[\boldsymbol{E}, A], \forall y\colon U'[\boldsymbol{F}, B]\ .} \\
\multicolumn{2}{l}{\qquad (x\ U'[\boldsymbol{\rho}, R]^\star\ y \ \wedge\ \mathsf{Dfnbl}^\star_{U'[\boldsymbol{Y},X]}(x, y)) \ \Rightarrow\ (gx)\ U''[\boldsymbol{\rho}, R]^\star\ (hy))}
\end{array}
$$

where

$$
\begin{array}{l}
\mathsf{Dfnbl}^\star_{U'[\boldsymbol{Y},X]}(x, y) \stackrel{def}{=} \\
\qquad \exists f\colon \forall\boldsymbol{Y}.\forall X.(\mathfrak{T}[X] \to U'[\boldsymbol{Y}, X])\ .\ (f\boldsymbol{E}|_k A\,\mathfrak{a}) = x \ \wedge\ (f\boldsymbol{F}|_k B\,\mathfrak{b}) = y
\end{array}
$$

We usually omit the type subscript to the $\mathsf{Dfnbl}^\star$ *clause.*

The essence of Def. 4 is that the arrow type relation is weakened with the $\mathsf{Dfnbl}^\star$ clause. This clause is an extension of the relation exhibited for the proof of Theorem 4. We have conveniently:

Lemma 6. *For $\mathfrak{T}[X]$ satisfying* ***adt****, we can derive*

$$\mathfrak{a}(\mathfrak{T}[R]^\star)\mathfrak{b} \ \Leftrightarrow\ \wedge_{1 \leq i \leq k}\ \mathfrak{a}.f_i\ (T_i[R]^\star)\ \mathfrak{b}.f_i$$

We also want the data type relation of Def. 4 to retain the property of being the equality over types in *Obs*. This is not derivable, but since *Obs* contains only inductive types, we get a semantic justification for this property.

Lemma 7. *With respect to the parametric PER-model of [3] it is sound to assert the following axiom schema for $C \in Obs$.*

$$\textsc{Ident}:\ \forall x, y{:}C\ .\ x =_C y\ \Leftrightarrow\ x(C[\rho]^\star)y$$

Now with the alternative notion of simulation relation $\mathfrak{T}[R]^\star$ obtained from Def. 4, we obtain variants of Theorem 4 valid also for higher-order function profiles (theorems 9 and 15). However, this comes at a price, since we here choose not to alter the parametricity axiom schema. Consequently, we loose proof power when considering the alternative simulation relation in universal type relations, and we can no longer rely directly on parametricity, as in Lemma 4, when deriving observational equivalence from the existence of a simulation relation.

4.2 Special Parametricity

Our solutions to this is to validate semantically special instances of alternative parametricity sufficient to reinstate the necessary proof power.

The special instances come in two variants, both based on the notion of closed observations. In shifting attention from general observable computations as proclaimed in Def. 2, to a notion of closed observations, we must now specify the collection *In* of input types in observations. (Compare this to the discussion around Example 2.) A sensible choice is to regard all types in *Obs* as input types, and henceforth *In* is assumed to contain this.

In the following we write for instance $(\forall X.\mathfrak{T}[X]^\star \to U[X]^\star)$, meaning the relation $\forall(A, B, R \subset A \times B)(\mathfrak{T}[R]^\star \to U[R]^\star)$.

Lemma 8. *For $\mathfrak{T}[X]$ adhering to* `adt`*, for $f{:}\forall X.(\mathfrak{T}[X] \to U[X])$, for any $U[X]$, and where free term variables of f are of types in In, we can derive*

$$f\ (\forall X.\mathfrak{T}[X]^\star \to U[X]^\star)\ f$$

By Lemma 8, the following axiom schema is sound w.r.t. any model whose interpretations of all $f{:}\forall X.(\mathfrak{T}[X] \to U[X])$ are denotable by terms whose only free variables are of types in *In*. For $\mathfrak{T}[X]$ adhering to `adt`, for any $U[X]$,

$$\textsc{spParam}:\ \forall f{:}\forall X.(\mathfrak{T}[X] \to U[X])\ .\ f\ (\forall X.\mathfrak{T}[X]^\star \to U[X]^\star)\ f$$

And then using spParam we get a general version of Theorem 4:

Theorem 9. *Given* spParam*, for $\mathfrak{T}[X]$ adhering to* `adt`*, the existence of a simulation relation coincides with observational equivalence, i.e. we can derive*

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

Proof: $\Rightarrow$: This follows from spParam and Ident.

$\Leftarrow$: We have to show that $\exists R \subset A \times B \;.\; \mathfrak{a}(\mathfrak{T}[R]^\star)\mathfrak{b}$ is derivable. We exhibit $R \stackrel{def}{=} (a:A, b:B).(\mathsf{Dfnbl}^\star(a,b))$. Due to the assumption $\mathtt{adt}$, it suffices by Lemma 6 to show for every component $g{:}U \to V$ in $\mathfrak{T}[X]$ the derivability of

$$\forall x{:}U[A], \forall y{:}U[B] \;.\; (x\ U[R]^\star\ y \ \wedge\ \mathsf{Dfnbl}^\star(x,y)) \ \Rightarrow\ (\mathfrak{a}.g\,x)\ V[R]^\star\ (\mathfrak{b}.g\,y)$$

where $V[X]$ is either some $C \in \mathit{Obs}$, whence we recall Ident, or the variable X. Now $\mathsf{Dfnbl}^\star(x,y)$ gives $\exists f_U{:}\forall X.(\mathfrak{T}[X] \to U[X]) \;.\; (f_U A\,\mathfrak{a}) = x \ \wedge\ (f_U B\,\mathfrak{b}) = y$. Let $f \stackrel{def}{=} \Lambda X.\lambda\mathfrak{x}{:}\mathfrak{T}[X] \;.\; (\mathfrak{x}.g(f_U X\mathfrak{x}))$.

$V[X] = C \in \mathit{Obs}$: We may show that $\mathfrak{a}.g\,x =_C \mathfrak{b}.g\,y$ is derivable. The assumption gives $(f A\,\mathfrak{a}) =_C (f B\,\mathfrak{b})$ which by β-reduction gives the desired result.

$V[X] = X$: We must derive $\exists f{:}\forall X.(\mathfrak{T}[X] \to V[X]) \;.\; (f A\,\mathfrak{a}) = (\mathfrak{a}.g\,x) \ \wedge\ (f B\,\mathfrak{b}) = (\mathfrak{b}.g\,y)$. For this we display f above. □

We also regain not only transitivity of the existence of simulation relations, but also composability of simulation relations. This relates the syntactic level to recent and on-going work on the semantic level, namely the *pre-logical relations* of [17, 18], the *lax logical relations* of [32, 21], and the *L-relations* of [20].

Theorem 10 (Composability of Simulation Relations). *Given* spParam, *for* $\mathfrak{T}[X]$ *adhering to* $\mathtt{adt}$, *we can derive*

$$\forall A, B, G, R{\subset}A{\times}B, S{\subset}B{\times}G, \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B], \mathfrak{g}{:}\mathfrak{T}[G].\\ \mathfrak{a}(\mathfrak{T}[R]^\star)\mathfrak{b} \ \wedge\ \mathfrak{b}(\mathfrak{T}[S]^\star)\mathfrak{g} \ \Rightarrow\ \mathfrak{a}(\mathfrak{T}[S \circ R]^\star)\mathfrak{g}$$

Proof: Assuming $\mathfrak{a}(\mathfrak{T}[R]^\star)\mathfrak{b} \ \wedge\ \mathfrak{b}(\mathfrak{T}[S]^\star)\mathfrak{g}$, the goal is to derive for every component $g{:}U \to V$ in $\mathfrak{T}[X]$

$$\forall x{:}U[A], \forall z{:}U[G] \;.\; (x\ U[S\circ R]^\star\ z \ \wedge\ \mathsf{Dfnbl}^\star(x,z)) \ \Rightarrow\ (\mathfrak{a}.g\,x)\ V[S\circ R]^\star\ (\mathfrak{g}.g\,z)$$

By $\mathsf{Dfnbl}^\star(x,z)$ we construct $f \stackrel{def}{=} \Lambda X.\lambda\mathfrak{x}{:}\mathfrak{T}[X] \;.\; (\mathfrak{x}.g(f_U X\mathfrak{x}))$.

$V[X] = C \in \mathit{Obs}$: By assumption and Theorem 9 $(f A\,\mathfrak{a}) = (f B\,\mathfrak{b}) = (f G\mathfrak{g})$, and $\mathfrak{a}.g\,x = (f A\,\mathfrak{a})$ and $(f G\mathfrak{g}) = \mathfrak{g}.g\,z$

$V[X] = X$: We must show $\exists b{:}U[B] \;.\; (\mathfrak{a}.g\,x)\ R\ b \ \wedge\ b\ S\ (\mathfrak{g}.g\,z)$. Exhibit $f B\,\mathfrak{b} = (\mathfrak{b}.g(f_U B\mathfrak{b}))$ for b. To show *e.g.* $(\mathfrak{a}.g\,x)\ R\ (\mathfrak{b}.g(f_U B\mathfrak{b}))$ it suffices by assumption to show $x\ U[R]^\star\ (f_U B\mathfrak{b}) \ \wedge\ \mathsf{Dfnbl}^\star(x,(f_U B\mathfrak{b}))$. But $x = (f_U A\mathfrak{a})$, so $\mathsf{Dfnbl}^\star(x,(f_U B\mathfrak{b}))$ is trivial and $(f_U A\mathfrak{a})\ U[R]^\star\ (f_U B\mathfrak{b})$ follows by spParam. □

As far as we know, it is not known whether or not the parametric PER-model of [3] satisfies spParam, even for $U[X] = C$, $C \in \mathit{Obs}$. We can however validate spParam in the polymorphic extensionally collapsed syntactic models of [8] or the parametric term models of [14].

4.3 Sticking to the Parametric PER-Model

However, in this paper our preference is to continue to seek validation under the non-syntactic parametric PER-model of [3]. Semantically, observational equivalence is usually defined w.r.t. contexts that when filled, are closed terms. Thus a

reasonable alternative definition in the logic of observational equivalence is the following.

Definition 5 (Closed Context Observational Equivalence (ObsEqC)**).** *Define* closed context observational equivalence ObsEqC w.r.t. *Obs in the logic by*

$$\begin{array}{l}\mathsf{ObsEqC} \stackrel{def}{=} (u{:}\exists X.\mathfrak{T}[X], v{:}\exists X.\mathfrak{T}[X]).\\ (\exists A,B.\exists \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B]\ .\ u = (\mathsf{pack}A\mathfrak{a})\ \wedge\ v = (\mathsf{pack}B\mathfrak{b})\ \wedge\\ \qquad \bigwedge_{C\in Obs} \forall f{:}\forall X.(\mathfrak{T}[X]\to C)\ .\ \mathsf{Closed}_{\Gamma^{In}}(f)\ \Rightarrow\ (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b}))\end{array}$$

where $\mathsf{Closed}_{\Gamma^{In}}(f)$ *is derivable iff* $\Gamma^{In} \vdash f$.

The idea is that closedness is qualified by a given context Γ^{In} so as to allow for variables of input types in observable computations. Note that this was automatically taken care of in the notion of observational computations of Def 2.

The task is now to determine what the predicate $\mathsf{Closed}_{\Gamma^{In}}(f)$ should be. This is intractable in the existing logic, but we can easily circumvent this problem by introducing $\mathsf{Closed}_{\Gamma^{In}}$ as a family of new basic predicates together with a predefined semantics as follows.

Definition 6. *The logical language is extended with families of basic predicates* $\mathsf{Closed}_{\hat\Gamma}(T)$ *ranging over types* T*, and* $\mathsf{Closed}_{\hat\Gamma}(t,T)$ *ranging over terms* $t{:}\,T$*, both relative to a given environment* $\hat\Gamma$*. This new syntax is given a predefined semantics as follows. For any type* $\Gamma \vdash T$*, term* $\Gamma \vdash t{:}T$*, and evaluation* $\gamma \in [\![\Gamma]\!]$*,*

$$\models_{\Gamma,\gamma} \mathsf{Closed}_{\hat\Gamma}(T) \stackrel{def}{\Leftrightarrow} \textit{exists some type } \hat\Gamma \vdash A\textit{, some } \hat\gamma \in [\![\hat\Gamma]\!] \textit{ s.t. } [\![\Gamma \vdash T]\!]_\gamma = [\![\hat\Gamma \vdash A]\!]_{\hat\gamma}$$

$$\models_{\Gamma,\gamma} \mathsf{Closed}_{\hat\Gamma}(t,T) \stackrel{def}{\Leftrightarrow} \textit{exists some type } \hat\Gamma \vdash A\textit{, term } \hat\Gamma \vdash a{:}A\textit{, some } \hat\gamma \in [\![\hat\Gamma]\!] \textit{ s.t. } [\![\Gamma \vdash T]\!]_\gamma = [\![\hat\Gamma \vdash A]\!]_{\hat\gamma} \textit{ and } [\![\Gamma \vdash t{:}T]\!]_\gamma = [\![\hat\Gamma \vdash a{:}A]\!]_{\hat\gamma}$$

Lemma 11. *It is easily seen that the following axiom schemata are sound.*

1. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma,X}(X)$
2. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma}(U)\ \wedge\ \mathsf{Closed}_{\hat\Gamma}(V)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma}(U\to V)$
3. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma,X}(U)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma}(\forall X.U)$
4. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma,x:U}(x,U)$
5. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma,x:U}(t,V)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma}(\lambda x{:}U.t, U\to V)$
6. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma}(g,U\to V)\ \wedge\ \mathsf{Closed}_{\hat\Gamma}(t,U)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma}(gt,V)$
7. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma,X}(t,U)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma}(\Lambda X.t, \forall X.U)$
8. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma}(f,\forall X.U[X])\ \wedge\ \mathsf{Closed}_{\hat\Gamma}(A)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma}(f\,A, U[A])$
9. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma}(T)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma'}(T),\ \ \hat\Gamma \subseteq \hat\Gamma'$
10. $\vdash_\Gamma \mathsf{Closed}_{\hat\Gamma}(t,T)\ \Rightarrow\ \mathsf{Closed}_{\hat\Gamma'}(t,T),\ \ \hat\Gamma \subseteq \hat\Gamma'$

We will usually omit the type argument in the term family of Closed. Intuitively, we should now be able to use Lemma 8 to show the necessary special parametricity instance. However, to make the induction spiral work, we have to strengthen lemmas 8 and 6, by incorporating Closed into the $\mathsf{Dfnbl}^\star$ clause.

Definition 7 (Data Type Relation by Closed Observers). *Define the* data type relation by closed observers $U[\boldsymbol{\rho}, R]^{\star}_{\mathsf{C}}$ *as the data type relation* $U[\boldsymbol{\rho}, R]^{\star}$ *of Def. 4, but where we use*

$$\mathsf{DfnblC}^{\star}_{U[\boldsymbol{Y},X]}(x,y) \stackrel{def}{=} \exists f{:}\forall \boldsymbol{Y}.\forall X.(\mathfrak{T}[X] \to U[\boldsymbol{Y},X])\ . \\ \mathsf{Closed}_{\Gamma^{In}}(f)\ \wedge\ (f\boldsymbol{E}|_k A\,\mathfrak{a}) = x\ \wedge\ (f\boldsymbol{F}|_k B\,\mathfrak{b}) = y$$

in place of $\mathsf{Dfnbl}^{\star}_{U[\boldsymbol{Y},X]}$, *for* $\Gamma^{In} = x_1{:}U_1, \ldots, x_m{:}U_m$, $U_i \in In$, $1 \leq i \leq m$.

Lemma 12. *For* $\mathfrak{T}[X]$ *satisfying* **adt**, *we have the derivability of*

$$\mathfrak{a}(\mathfrak{T}[R]^{\star}_{\mathsf{C}})\mathfrak{b}\ \Leftrightarrow\ \wedge_{1\leq i\leq k}\ \mathfrak{a}.f_i\ (T_i[R]^{\star}_{\mathsf{C}})\ \mathfrak{b}.f_i$$

Lemma 13. *With respect to the parametric PER-model of [3] it is sound to assert the following axiom schema for* $C \in Obs$.

$$\textsc{IdentC}:\ \forall x,y{:}C\ .\ x =_C y\ \Leftrightarrow\ x(C[\rho]^{\star}_{\mathsf{C}})y$$

Lemma 14. *For* $\mathfrak{T}[X]$ *adhering to* **adt**, *for* $f : \forall X.(\mathfrak{T}[X] \to U[X])$, *for any* $U[X]$, *and where free term variables of* f *are of types in In, we can derive*

$$f\ (\forall X.\mathfrak{T}[X]^{\star}_{\mathsf{C}} \to U[X]^{\star}_{\mathsf{C}})\ f$$

By Lemma 14 it is sound w.r.t. the parametric PER-model to postulate the following axiom schema. For $\mathfrak{T}[X]$ adhering to **adt**, for $\Gamma^{In} = x_1{:}U_1, \ldots, x_m{:}U_m$, $U_i \in In$, $1 \leq i \leq m$, for any $U[X]$,

$$\textsc{CspParam}:\ \forall f{:}\forall X.(\mathfrak{T}[X] \to U[X])\ .\ \mathsf{Closed}_{\Gamma^{In}}(f)\ \Rightarrow\ f\ (\forall X.\mathfrak{T}[X]^{\star}_{\mathsf{C}} \to U[X]^{\star}_{\mathsf{C}})\ f$$

We can now show the higher-order polymorphic generalisation of Theorem 4 now validated w.r.t. the parametric PER-model:

Theorem 15. *Extending the language with the predicates* Closed *of Def. 6, given* CspParam, *for* $\mathfrak{T}[X]$ *adhering to* **adt**, *for* $\Gamma^{In} = x_1{:}U_1, \ldots, x_m{:}U_m$, $U_i \in In$, $1 \leq i \leq m$, *the following is derivable.*

$$\forall A,B.\forall \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B]\ . \\ \exists R{\subset}A{\times}B\ .\ \mathfrak{a}(\mathfrak{T}[R]^{\star}_{\mathsf{C}})\mathfrak{b}\ \Leftrightarrow \\ \textstyle\bigwedge_{C\in Obs} \forall f{:}\forall X.(\mathfrak{T}[X] \to C)\ .\ \mathsf{Closed}_{\Gamma^{In}}(f)\ \Rightarrow\ (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b})$$

Proof: $\Rightarrow$: This follows from CspParam and IdentC.

$\Leftarrow$: Along the lines of the proof of Theorem 9, and using Lemma 11 to obtain $\mathsf{Closed}_{\Gamma^{In}}(f)$ from $\mathsf{Closed}_{\Gamma^{In}}(f_U)$, and using Lemma 12 and IdentC in place of Lemma 6 and Ident. □

We now get composability validated w.r.t. the parametric PER-model:

Theorem 16 (Composability of Simulation Relations). *Given* CspParam, *for* $\mathfrak{T}[X]$ *adhering to* **adt**, *we can derive*

$$\forall A,B,G,R{\subset}A{\times}B,S{\subset}B{\times}G,\mathfrak{a}{:}\mathfrak{T}[A],\mathfrak{b}{:}\mathfrak{T}[B],\mathfrak{g}{:}\mathfrak{T}[G]. \\ \mathfrak{a}(\mathfrak{T}[R]^{\star}_{\mathsf{C}})\mathfrak{b}\ \wedge\ \mathfrak{b}(\mathfrak{T}[S]^{\star}_{\mathsf{C}})\mathfrak{g}\ \Rightarrow\ \mathfrak{a}(\mathfrak{T}[S\circ R]^{\star}_{\mathsf{C}})\mathfrak{g}$$

Proof: As for Theorem 10, but using CSPPARAM instead of SPPARAM. □

Finally we retrieve the notions of specification refinement. We have established the coincidence of observational equivalence and the existence of a simulation relation at higher order, but in this paper we do not tie the link to equality at existential type. This is of minor importance because we can simply redefine our notions in terms of ObsEqC (or ObsEq) instead of equality: The realisation predicate of Def. 1 then reads $\Theta_{SP}(u) = \exists X.\exists \mathfrak{r}{:}\mathfrak{T}_{SP}[X]\ .\ u$ ObsEqC (pack$X\mathfrak{r}$) $\wedge$ $\Phi_{SP}[X,\mathfrak{r}]$. Note that we now have to show the stability of constructors explicitly.

5 Final Remarks and Discussion

This paper has addressed specification refinement up to observational equivalence with System F using Plotkin and Abadi's logic for parametric polymorphism. At first order, specification refinement up to observational equivalence can be defined in the logic using Luo's formalism, because equality at existential type coincides (Theorem 3) with observational equivalence ObsEq (Def 2).

At higher order, *i.e.* when the data type signature has higher-order function types, we ostensibly loose the correspondence in the logic between observational equivalence and the existence of a simulation relation. We argued that at higher-order the usual notion of simulation relation is too strict, since it for function types requires that one consider arbitrary arguments, which might be other than those actually accessible in computations.

Thus an alternative simulation relation $\mathfrak{T}[R]^\star$ was proposed based on the Dfnbl* clause and data type relation (Def. 4). Then a correspondence in the logic between observational equivalence and the existence of this alternative simulation relation is re-established in any model in which the axiom schema SPPARAM is valid (Theorem 9). For the parametric PER-model, we also achieve the correspondence (Theorem 15) by extending the logical language with basic predicates Closed$_{\hat{f}}$, defining a second alternative simulation relation $\mathfrak{T}[R]^\star_{\mathsf{C}}$, and validating the axiom schema CSPPARAM w.r.t the parametric PER-model. Finally, we achieve a simulation relation in the logic that composes at higher-order (theorems 10 and 16). This relates to on-going work on the semantic level.

The approach taken in this paper is conservative in that we in the outset do not want to alter either the type theory nor the parametricity axiom schema. This is motivated by the view that it is the relational proof criteria specifically for abstract data types that need amending, not the type theory itself. The parametricity axiom is left alone in order to relate to established models for relational parametricity. However, there seem to be other interesting approaches worth looking into. One alternative would be to alter the type system so as to isolate separate types for use in abstract data types, and then extend the parametricity axiom schema to deal with these types. A very promising approach to finding a non-syntactic model satisfying SPPARAM seems to be to work along the lines of Jung and Tiuryn [19], and define a non-standard Kripke-like model to validate the logic.

Acknowledgements Thanks to Martin Hofmann, Don Sannella, Furio Honsell, Gordon Plotkin and Martin Wehr for helpful discussions. Thanks to the referees for very helpful comments. This research has been supported by EPSRC grant GR/K63795, and NFR (Norwegian Research Council) grant 110904/41.

References

1. M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. *Theoretical Computer Science*, 121:9–58, 1993.
2. D. Aspinall. *Type Systems for Modular Programs and Specifications.* PhD thesis, University of Edinburgh, 1998.
3. E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.
4. M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165:3–55, 1996.
5. M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25:149–186, 1995.
6. M. Bidoit, R. Hennicker, and M. Wirsing. Proof systems for structured specifications with observability operators. *Theoretical Computer Sci.*, 173:393–443, 1997.
7. C. Böhm and A. Beraducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.
8. V. Breazu-Tannen and T. Coquand. Extensional models for polymorphism. *Theoretical Computer Science*, 59:85–114, 1988.
9. M. Cerioli, M. Gogolla, H. Kirchner, B. Krieg-Brückner, Z. Qian, and M. Wolf. *Algebraic System Specification and Development. Survey and Annotated Bibliography, 2nd Ed.*, volume 3 of *Monographs of the Bremen Institute of Safe Systems.* Shaker, 1997. 1st edition available in *LNCS* 501, Springer, 1991.
10. J.A. Goguen. Parameterized programming. *IEEE Transactions on Software Engineering*, SE-10(5):528–543, 1984.
11. J.A. Goguen and R. Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. Tech. Rep. CSL-118, SRI International, 1980.
12. J.E. Hannay. Specification refinement with System F. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 530–545, 1999.
13. J.E. Hannay. Specification refinement with System F, the higher-order case. Submitted for publication, 2000.
14. R. Hasegawa. Parametricity of extensionally collapsed term models of polymorphism and their categorical properties. In *Proc. TACS'91*, volume 526 of *LNCS*, pages 495–512, 1991.
15. R. Hennicker. Structured specifications with behavioural operators: Semantics, proof methods and applications. Habilitationsschrift, LMU, München, 1997.
16. C.A.R. Hoare. Proofs of correctness of data representations. *Acta Inform.*, 1:271–281, 1972.
17. F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement in typed lambda calculus. In *Proc. FOSSACS 2000, LNCS*, 2000.
18. F. Honsell and D. Sannella. Pre-logical relations. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 546–561, 1999.
19. A. Jung and J. Tiuryn. A new characterization of lambda definability. In *Proc. of TLCA 93*, volume 664 of *LNCS*, pages 245–257, 1993.

20. Y. Kinoshita, P.W. O'Hearn, A.J. Power, M. Takeyama, and R.D. Tennent. An axiomatic approach to binary logical relations with applications to data refinement. In *Proc. of TACS'97*, volume 1281 of *LNCS*, pages 191–212, 1997.
21. Y. Kinoshita and A.J. Power. Data refinement for call-by-value programming languages. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 562–576, 1999.
22. Z. Luo. Program specification and data type refinement in type theory. *Math. Struct. in Comp. Sci.*, 3:333–363, 1993.
23. Q. Ma and J.C. Reynolds. Types, abstraction and parametric polymorphism, part 2. In *Proc. 7th MFPS*, volume 598 of *LNCS*, pages 1–40, 1991.
24. H. Mairson. Outline of a proof theory of parametricity. In *ACM Symposium on Functional Programming and Computer Architecture*, volume 523 of *LNCS*, pages 313–327, 1991.
25. J.C. Mitchell. On the equivalence of data representations. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 305–330. Academic Press, 1991.
26. J.C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
27. J.C. Mitchell and G.D. Plotkin. Abstract types have existential type. *ACM Trans. on Programming Languages and Systems*, 10(3):470–502, 1988.
28. N. Mylonakis. Behavioural specifications in type theory. In *Recent Trends in Data Type Spec., 11th WADT*, volume 1130 of *LNCS*, pages 394–408, 1995.
29. A.M. Pitts. Parametric polymorphism and operational equivalence. In *Proc. 2nd Workshop on Higher Order Operational Techniques in Semantics*, volume 10 of *ENTCS*. Elsevier, 1997.
30. A.M. Pitts. Existential types: Logical relations and operational equivalence. In *Proc. ICALP'98*, volume 1443 of *LNCS*, pages 309–326, 1998.
31. G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA 93*, volume 664 of *LNCS*, pages 361–375, 1993.
32. G.D. Plotkin, A.J. Power, and D. Sannella. A compositional generalisation of logical relations. Submitted for publication, 2000.
33. E. Poll and J. Zwanenburg. A logic for abstract data types as existential types. In *Proc. TLCA'99*, volume 1581 of *LNCS*, pages 310–324, 1999.
34. B. Reus and T. Streicher. Verifying properties of module construction in type theory. In *Proc. MFCS'93*, volume 711 of *LNCS*, pages 660–670, 1993.
35. J.C. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
36. D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences*, 34:150–178, 1987.
37. D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Inform.*, 25(3):233–281, 1988.
38. D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.
39. O. Schoett. *Data Abstraction and the Correctness of Modular Programming*. PhD thesis, University of Edinburgh, 1986.
40. T. Streicher and M. Wirsing. Dependent types considered necessary for specification languages. In *Recent Trends in Data Type Spec.*, volume 534 of *LNCS*, pages 323–339, 1990.
41. J. Underwood. Typing abstract data types. In *Recent Trends in Data Type Spec., Proc. 10th WADT*, volume 906 of *LNCS*, pages 437–452, 1994.
42. J. Zwanenburg. *Object-Oriented Concepts and Proof Rules: Formalization in Type Theory and Implementation in Yarrow*. PhD thesis, Technische Universiteit Eindhoven, 1999.