Specification Refinement with System F, The Higher-Order Case

Jo Erskine Hannay

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

Abstract. A type-theoretic counterpart to the notion of algebraic specification refinement is discussed for abstract data types with higher-order signatures. The type-theoretic setting consists of System F and the logic for parametric polymorphism of Plotkin and Abadi. For first-order signatures, this setting immediately gives a natural notion of specification refinement up to observational equivalence via the notion of simulation relation. Moreover, a proof strategy for proving observational refinements formalised by Bidoit, Hennicker and Wirsing can be soundly imported into the type theory. In lifting these results to the higher-order case, we find it necessary firstly to develop an alternative simulation relation and secondly to extend the parametric PER-model interpretation, both in such a way as to observe data type abstraction barriers more closely.

1 Introduction

One framework in algebraic specification that has particular appeal and applicability is that of *stepwise specification refinement*. The idea 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 is proven correct, the resulting program is guaranteed to satisfy the initial specification. This formal methodology for software development, although in principle first-order, is supported *e.g.* by the wide-spectrum language EML for SML result programs [17].

The motivation for our present work is a wish to transfer this concept and its theoretical rigour to a wider spectrum of language principles, and to go beyond the first-order boundaries inherent in the universal algebra approach.

In this paper we look at Girard/Reynolds' polymorphic λ-calculus System F. The accompanying logic of Plotkin and Abadi [30] asserts *relational parametricity* in Reynolds' sense [34, 21]. This setting allows an elegant formalisation of abstract data types as existential types [25], and the relational parametricity axiom enables one to derive in the logic that two concrete data types are equal if and only if there exists a simulation relation between their operational parts. At first order, this in turn corresponds to a notion of observational equivalence. Thus, the type-theoretic formalism of refinement due to Luo [20] automatically gives a notion in the logic of specification refinement up to observational equivalence; a key issue in program development.

In [11] a formal connection is shown at first order between an account of algebraic specification refinement due to Sannella and Tarlecki [37, 36] and the type-theoretic refinement notion above. Issues in algebraic specification refinement, such as input-sort choice [35] and constructor stability [38, 36, 9], are automatically resolved in the type-theoretic setting. Also, a proof method for proving observational refinements due to Bidoit, Hennicker and Wirsing [4, 3, 5] is soundly imported into the type-theory logic. The soundness of the logic as a whole is with reference to the parametric PER-model of Bainbridge *et al.* [2].

This paper now generalises the concepts established at first order and treats data types whose operations may be higher-order and polymorphic. At higher order we run into two problems, both of which are solvable by simply observing more closely the abstraction barrier supplied by existential types. Firstly, at higher order, the formal link between the existence of a simulation relation and observational equivalence breaks down. By analysing the existential-type abstraction barrier, one can devise an alternative notion of simulation relation in the logic [12]. This notion composes at higher-order, thus relating the syntactic level to on-going work on the semantic level remedying the fact that logical relations traditionally used to describe refinement do not compose at higher order [15, 16, 19, 18, 31]. Now, using an alternative simulation relation means that parametricity is not applicable in the needed form. This is solved soundly w.r.t. the parametric PER-model by augmenting the logical language with a new basic predicate Closed with predefined semantics, so as to restrict observable computations to be closed, and then asserting the needed variant of parametricity [12].

The second problem arises when attempting to validate the proof method of Bidoit *et al.* at higher order w.r.t. the parametric PER-model. Again, observing abstraction barriers more acutely, this problem is solved by supplying an interpretation for encapsulated operations that exactly mirror their applicability.

The theoretical foundations in universal algebra are formidable, see [8]. But there has been substantial development for refinement in type theory as well, and other relevant work include [32, 28, 29, 27, 33, 1, 40, 39].

Section 2 outlines the type theory, the logic, and the standard PER semantics. In Sect. 3 specification refinement is introduced. Simulation relations and the proof method for proving observational refinements are introduced for first order. In Sect. 4 we present the alternative simulation relation in the logic to cope with higher-order and polymorphism, and in Sect. 5 we develop the semantics to validate the proof method also at higher order.

2 Parametric Polymorphism

2.1 Syntax

The polymorphic λ-calculus System F has types and terms given by grammars

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

where X and x range over type and term variables respectively. Judgements for type and term formation involve type contexts, and term-contexts depending on

type contexts, *e.g.* $X, x{:}X \rhd x{:}X$. The logic in [30, 22] for relational parametricity on System F has formulae built using the standard connectives, but now basic predicates are not only equations, but also relation membership statements:

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

where R ranges over relation symbols. We write $\alpha[R, X, x]$ to indicate possible occurrences of R, X and x in type, term or formula α, and may write $\alpha[\rho, A, t]$ for the result of substitution, following the appropriate rules concerning capture. We also write $t\ R\ u$ in place of $R(t,u)$. Judgements for formula formation now involve relation symbols, so contexts are augmented with relation variables, depending on the type context, and obeying standard conventions for contexts. The judgements are as expected. Relation definition is accommodated,

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

by syntax as indicated. 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 acquires further relations by substituting 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:

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

The proof system is natural deduction over formulae now involving relation symbols, and is augmented with inference rules for relation symbols, for example we have for Φ a finite set of formulae:

$$\frac{\Phi \vdash_{\Gamma, R \subset A \times B} \phi[R]}{\Phi \vdash_\Gamma \forall R \subset A \times B\ .\ \phi[R]}, \text{no } R \text{ in } \Phi \qquad \frac{\Phi \vdash_\Gamma \forall R \subset A \times B.\phi[R], \quad \Gamma \rhd \rho \subset A \times B}{\Phi \vdash_\Gamma \phi[\rho]}$$

One has axioms for equational reasoning and $\beta\eta$ equalities. And now finally, the following relational 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. the parametric PER-model of [2] and also w.r.t. the syntactic parametric models of [13]. The following link to equality is essential.

Theorem 1 (Identity Extension Lemma [30]**).** *For any $T[\boldsymbol{Z}]$, the following sequent is derivable using* PARAM.

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

For abstract data types (ADTs), encapsulation is provided in the style of [25] by the following encoding of existential types and pack and unpack combinators. Parameters in existential types are omitted from the discussion, since we will not be dealing with parameterised specifications.

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

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

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

We omit subscripts to pack and unpack as much as possible. Operationally, pack packages a data representation and an implementation of operations on that data representation to give an instance of the ADT given by 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 the application operator for pack.

Existential types together with the pack and unpack combinators embody a crucial abstraction barrier. Any client computation $f{:}\ \forall X.(T[X] \to Y)$ is η-equivalent to a term of the form $\Lambda X.\lambda x{:}\,T[X]\ .\ t[X,x]$. Notice then that a client computation cannot have free variables of types involving the bound (viz. existentially quantified) type variable X. The only way a client computation may compute over types involving X is by accessing virtual operations in the supplied collection x of operations. Notice also that the only way a package can be used is through client computations for which the above holds. The result of all this is the following crucial fact that will be instrumental for our results later:

***abs-bar*:** Operations from a in a package $(\mathsf{pack}Aa){:}\,\exists X.T[X]$ will only be applied to terms $t[\boldsymbol{E}/\boldsymbol{Y}, A/X, a/x]$ s.t. $\boldsymbol{Y}, X, x{:}\,T[X] \rhd t[\boldsymbol{Y}, X, x]\ :\ U[\boldsymbol{Y}, X]$ where the vector $\boldsymbol{Y}$ accounts for instances of polymorphic operations.

Theorem 2 (Characterisation by Simulation Relation [30]**).** *The following sequent schema is derivable using* PARAM.

$$\forall u, v{:}\exists X.T[X]\ .\quad u =_{\exists X.T[X]} v\quad \Leftrightarrow$$
$$\exists A, B.\exists a{:}\,T[A], b{:}\,T[B].\exists R \subset A \times B\ .\ u = (\mathsf{pack}Aa)\ \wedge\ v = (\mathsf{pack}Bb)\ \wedge\ a(T[R])b$$

Theorem 2 states the equivalence of equality at existential type with the existence of a simulation relation in the sense of [23]. From this we also get

Theorem 3. $\forall u{:}\exists X.T[X].\exists A.\exists a{:}\,T[A]\ .\ u = (\mathsf{pack}Aa)$

Weak versions of standard constructs such as products, sums, initial and final (co-)algebras are encodable in System F [6]. 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 $\mathsf{list}(A) \stackrel{def}{=} \forall X.X \to (A \to X \to X) \to X$.

2.2 Semantics

We very briefly overview the parametric PER-model of [2] for the logic.

Let **PER** denote the universe of all partial equivalence relations (PERs) over the natural numbers $\mathbb{N}$. Types are interpreted as PERs, but intuitively it helps to think of the associated quotient instead, whose elements are equivalence classes. Terms are thus interpreted as functions mapping equivalence classes from one PER to another. Relations between PERs relate equivalence classes of the PERs.

Formally this is expressed in elementary terms as follows. A PER $\mathcal{A}$ is a symmetric and transitive binary relation on $\mathbb{N}$. The domain $dom(\mathcal{A})$ of $\mathcal{A}$ contains those $a \in \mathbb{N}$ for which $a\ \mathcal{A}\ a$. For any $a \in dom(\mathcal{A})$ we can form the equivalence class $[a]_{\mathcal{A}}$. A morphism from $\mathcal{A}$ to $\mathcal{B}$ is given by $n \in \mathbb{N}$ if for any $a, a' \in \mathbb{N}$, $a \in dom(\mathcal{A}) \;\Rightarrow\; n(a)\downarrow$ and $a\ \mathcal{A}\ a' \;\Rightarrow\; n(a)\ \mathcal{B}\ n(a')$. Here, $n(a)$ denotes the result of evaluating the n^{th} partial recursive function on a, and $n(a)\downarrow$ denotes that this function is defined for a. We can form a PER $(\mathcal{A} \to \mathcal{B})$ by defining $n\ (\mathcal{A} \to \mathcal{B})\ n' \stackrel{def}{\Leftrightarrow}$ for any $a, a' \in \mathbb{N}$, $a \in dom(\mathcal{A}) \;\Rightarrow\; n(a)\downarrow$ and $n'(a)\downarrow$, and $a\ \mathcal{A}\ a' \;\Rightarrow\; n(a)\ \mathcal{B}\ n'(a')$. That is, the equivalence classes in $(\mathcal{A} \to \mathcal{B})$ contain functions that are extensionally equal w.r.t. $\mathcal{A}$ and $\mathcal{B}$. Each such equivalence class is then a morphism between $\mathcal{A}$ and $\mathcal{B}$, with application defined as $[n]_{\mathcal{A}\to\mathcal{B}}[a]_{\mathcal{A}} \stackrel{def}{=} [n(a)]_{\mathcal{B}}$. Products are given by $n\ (\mathcal{A} \times \mathcal{B})\ n' \stackrel{def}{\Leftrightarrow} n.1\ \mathcal{A}\ n'.1 \;\wedge\; n.2\ \mathcal{B}\ n'.2$, where $m.i$ decodes the pairing encoding of natural numbers. A relation between $\mathcal{A}$ and $\mathcal{B}$ is given by a relation S between $dom(\mathcal{A})$ and $dom(\mathcal{B})$ that is *saturated*, *i.e.* $(m\ \mathcal{A}\ n$ and $n\ S\ n'$ and $n'\ \mathcal{B}\ m') \;\Rightarrow\; m\ S\ m'$. Thus S preserves, and can be seen to relate equivalence classes. Any member n of an equivalence class q is called a *realiser* for q.

Type semantics are now defined denotationally w.r.t. to an environment γ.

$$\begin{array}{l}
[\![\Gamma, X \rhd X]\!]_\gamma \stackrel{def}{=} \gamma(X) \\
[\![\Gamma \rhd U \to V]\!]_\gamma \stackrel{def}{=} ([\![\Gamma \rhd U]\!]_\gamma \to [\![\Gamma \rhd V]\!]_\gamma) \\
[\![\Gamma \rhd \forall X.U[X]]\!]_\gamma \stackrel{def}{=} (\cap_{\mathcal{A} \in \mathbf{PER}} [\![\Gamma, X \rhd U[X]]\!]_{\gamma[X \mapsto \mathcal{A}]})^\flat
\end{array}$$

where $(\cap_{\mathcal{A} \in \mathbf{PER}} [\![\Gamma, X \rhd U[X]]\!]_{\gamma[X \mapsto \mathcal{A}]})^\flat$ is the indicated intersection but trimmed down to only those elements invariant over all saturated relations. This trimming is what makes the model relational parametric.

For brevity we omit the details of term interpretation, since these are not extensively needed in our discussion. The fact we will need, is that the term semantics yields realisers that encode partial recursive functions according to term structure. One is thus justified in viewing a realiser as being generated

freely over a set of realisers representing free variables, using term-formation rules. This can be done independently of typing information.

In the parametric PER-model initial constructs interpret to objects isomorphic to interpretations of corresponding inductive types, *e.g.* let $T[X] = 1 + X$. Then $[\![\triangleright \forall X.((T[X] \to X) \to X)]\!] \cong [\![\triangleright \forall X.(X \to (X \to X) \to X)]\!] \cong \mathbb{N}$.

3 Abstract Data Type Specification Refinement

We describe ADT specification refinement up to observational equivalence. The latter is defined w.r.t. a finite set *Obs* of observable types, *viz.* closed inductive types. Examples are bool and nat. Henceforth we reserve $\mathfrak{T}[X]$ for the *body* part of abstract data type $\exists X.\mathfrak{T}[X]$. Parameterised specifications are outside the scope of this paper, so we assume X to be the only free type variable in $\mathfrak{T}[X]$.

Reflecting notions from algebraic specification and [23], we define observational equivalence in terms of observable computations in the logic as follows.

Definition 1 (Observational Equivalence (ObsEq) [11, 12]**).** *Define* observational equivalence ObsEq w.r.t. observable types *Obs in the logic by*

$$\begin{array}{l}\mathsf{ObsEq}^{Obs} \stackrel{def}{=} (u{:}\exists X.\mathfrak{T}[X], v{:}\exists X.\mathfrak{T}[X]).\\ (\exists A, B.\exists \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B] \;.\; u = (\mathsf{pack} A\,\mathfrak{a}) \;\wedge\; v = (\mathsf{pack} B\,\mathfrak{b}) \;\wedge \\ \qquad\qquad \bigwedge_{D \in Obs} \forall f{:}\forall X.(\mathfrak{T}[X] \to D) \;.\; (f\,A\,\mathfrak{a}) = (f\,B\,\mathfrak{b}))\end{array}$$

Now we define ADT specification up to observational equivalence. ADT bodies may contain higher-order and polymorphic types. In our setting, there is always a current set *Obs* of observable types. We assume the following:

***adt*:** A product $T_1[\boldsymbol{Y}, X] \times \cdots \times T_m[\boldsymbol{Y}, X]$ is on *ADT-body form* if each $T_i[\boldsymbol{Y}, X]$ is in uncurried form, *i.e.* of the form $T_{i1}[\boldsymbol{Y}, X] \times \cdots \times T_{n_i}[\boldsymbol{Y}, X] \to T_{c_i}[\boldsymbol{Y}, X]$, where $T_{c_i}[\boldsymbol{Y}, X]$ is not an arrow type, and secondly, $T_{c_i}[\boldsymbol{Y}, X]$ is either X or some $D \in Obs$ or a universal type $\forall Y'.T[\boldsymbol{Y}, Y', X]$ where $T[\boldsymbol{Y}, Y', X]$ is on ADT-body form. If $n_i = 0$, then $T_i[\boldsymbol{Y}, X] = T_{c_i}[\boldsymbol{Y}, X]$. At top level, then, we assume for the body $\mathfrak{T}[X]$ of $\exists X.\mathfrak{T}[X]$ that $\mathfrak{T}[X] \stackrel{def}{=} T_1[X] \times \cdots \times T_k[X]$ is on ADT-body form. We will write $\mathfrak{T}[X]$ as $\mathit{Record}(f_1{:}T_1[X], \ldots, f_k{:}T_k[X])$.

The uncurried form and the record-type notation are merely notational conveniences aiding discourse. The restriction on universal types is however a proper restriction. We do not know exactly how severe this restriction is in practice.

Definition 2 (Abstract Data Type Specification). *An* abstract data type specification *SP is a tuple* $\langle\langle \mathit{Sig}_{SP}, \Theta_{SP}\rangle, \mathit{Obs}_{SP}\rangle$ *where*

$$\begin{array}{l}\mathit{Sig}_{SP} \stackrel{def}{=} \exists X.\mathfrak{T}_{SP}[X],\\ \Theta_{SP}(u) \stackrel{def}{=} \exists X.\exists \mathfrak{x}{:}\mathfrak{T}_{SP}[X] \;.\; u\ \mathsf{ObsEq}^{Obs_{SP}}\ (\mathsf{pack} X\,\mathfrak{x}) \;\wedge\; \Phi_{SP}[X, \mathfrak{x}],\end{array}$$

where $\Phi_{SP}[X, \mathfrak{x}]$ *is a finite set of formulae in the logic. If* $\Theta_{SP}(u)$ *is derivable, then u is said to be a* realisation of *SP.*

Consider for example the specification $\mathsf{Set} \stackrel{def}{=} \langle\langle Sig_{\mathsf{Set}}, \Theta_{\mathsf{Set}}\rangle, \{\mathsf{bool}, \mathsf{nat}\}\rangle$, where

$$
\begin{array}{l}
Sig_{\mathsf{Set}} = \exists X.\mathfrak{T}_{\mathsf{Set}}[X], \\
\mathfrak{T}_{\mathsf{Set}}[X] = \mathit{Record}(\mathsf{empty}\colon X,\ \mathsf{add}\colon \mathsf{nat}\times X\to X,\ \mathsf{remove}\colon \mathsf{nat}\times X\to X, \\
\qquad\qquad \cap\colon X\times X\to X,\ \mathsf{in}\colon \mathsf{nat}\times X\to\mathsf{bool},\ \mathsf{prsrv}\cap\colon (X\to X)\to\mathsf{bool}), \\
\Theta_{\mathsf{Set}}(u) = \exists X.\exists \mathfrak{x}\colon \mathfrak{T}_{\mathsf{Set}}[X]\ .\ u\ \mathsf{ObsEqC}^{\{\mathsf{bool},\mathsf{nat}\}}\ (\mathsf{pack}X\mathfrak{x})\ \wedge \\
\qquad \forall x\colon \mathsf{nat}, s\colon X\ .\ \mathfrak{x}.\mathsf{add}(x,\mathfrak{x}.\mathsf{add}(x,s)) = \mathfrak{x}.\mathsf{add}(x,s)\ \wedge \\
\qquad \forall x, y\colon \mathsf{nat}, s\colon X\ .\ \mathfrak{x}.\mathsf{add}(x,\mathfrak{x}.\mathsf{add}(y,s)) = \mathfrak{x}.\mathsf{add}(y,\mathfrak{x}.\mathsf{add}(x,s))\ \wedge \\
\qquad \forall x\colon \mathsf{nat}\ .\ \mathfrak{x}.\mathsf{in}(x,\mathfrak{x}.\mathsf{empty}) = \mathsf{false}\ \wedge \\
\qquad \forall x, y\colon \mathsf{nat}, s\colon X\ .\ \mathfrak{x}.\mathsf{in}(x,\mathfrak{x}.\mathsf{add}(y,s)) = \mathsf{if}\ x =_{\mathsf{nat}} y\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathfrak{x}.\mathsf{in}(x,s)\ \wedge \\
\qquad \forall x\colon \mathsf{nat}, s\colon X\ .\ \mathfrak{x}.\mathsf{in}(x,\mathfrak{x}.\mathsf{remove}(x,s)) = \mathsf{false}\ \wedge \\
\qquad \forall s, s\colon X, x\colon \mathsf{nat}\ .\ \mathfrak{x}.\mathsf{in}(x,s)\ \wedge\ \mathfrak{x}.\mathsf{in}(x,s')\ \Leftrightarrow\ \mathfrak{x}.\mathsf{in}(x,\mathfrak{x}.\cap(s,s'))\ \wedge \\
\qquad \forall f\colon X\to X, s, s'\colon X\ .\ \mathfrak{x}.\mathsf{prsrv}\cap(f) = \mathsf{true}\ \Leftrightarrow\ \mathfrak{x}.\cap(s,s') = \mathfrak{x}.\cap(fs, fs')
\end{array}
$$

This higher-order specification also illustrates the notion of *input types/sorts*. Consider the package $LI \stackrel{def}{=} (\mathsf{pack}\ \mathsf{list}(\mathsf{nat})\ \mathfrak{l}) : Sig_{\mathsf{Set}}$, 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, $\mathfrak{l}.\mathsf{remove}$ removes the first occurrence of a given element, and $\mathfrak{l}.\cap$ takes two lists and generates a non-repeating list of common elements. By `abs-bar`, typing allows users of LI to only build lists using operations of $\mathfrak{l}$, such as $\mathfrak{l}.\mathsf{empty}$ and $\mathfrak{l}.\mathsf{add}$, and on such lists the efficient $\mathfrak{l}.\mathsf{remove}$ gives the intended result. By the same token, any observable computation $f\colon \forall X.(\mathfrak{T}_{\mathsf{Set}}[X]\to D)$, $D\in\{\mathsf{bool},\mathsf{nat}\}$ can only refer to such lists, and not to arbitrary lists. This is the crucial point that admits LI as a realisation of Set according to Def. 2. 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. One has to explicitly restrict input sorts to not include the abstract sort, in this case set, when defining observational equivalence [35], whereas the type-theoretic formalism here deals with this automatically.

The realisation predicate $\Theta_{SP}(u)$ of Def. 2 expresses *u is observationally equivalent to a package* $(\mathsf{pack}X\mathfrak{x})$ *that satisfies the axioms* Φ_{SP}. Hence specification is up to observational equivalence. Specification refinement up to observational equivalence can now be expressed in the logic as follows.

Definition 3 (Specification Refinement). *A specification SP' is a* refinement *of a 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 $SP \rightsquigarrow_F SP'$ for this fact.

The notion of constructor in Def. 3 is based on the notion of parameterised program [9]. 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 [20] and derived signature morphisms in [15]. It is evident that the refinement

relation of Def. 3 is transitive, *i.e.* for $F \circ F' \stackrel{def}{=} \lambda u{:}\, Sig_{SP''}.F(F'u)$:

$$SP \overset{\sim}{\underset{F}{\rightarrow}} SP' \quad \text{and} \quad SP' \overset{\sim}{\underset{F'}{\rightarrow}} SP'' \;\Rightarrow\; SP \overset{\sim}{\underset{F \circ F'}{\rightarrow}} SP''$$

If $\mathfrak{T}[X]$ is first-order, we get a string of interesting results in the logic.

Theorem 4 ([11]). *Suppose $\langle\langle \exists X.\mathfrak{T}[X], \Theta\rangle, Obs\rangle$ is a specification where $\mathfrak{T}[X]$ only contains first-order function profiles. With* Param *we derive that the existence of a simulation relation is equivalent to observational equivalence, i.e.*

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

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

$\Leftarrow$: We must exhibit an R such that $\mathfrak{a}(\mathfrak{T}[R])\mathfrak{b}$. Semantically, [23, 24, 38] relate elements iff they are denotable by some common term. We mimic this: For R give $\mathsf{Dfnbl} \stackrel{def}{=} (a{:}\,A, b{:}\,B).(\exists f{:}\forall X.(\mathfrak{T}[X] \to X).(f\,A\,\mathfrak{a}) = a \ \wedge\ (f\,B\,\mathfrak{b}) = b)$. □

Together with Fact 2 this gives:

Theorem 5 ([11]). *Let $\exists X.\mathfrak{T}[X]$ be as in Theorem 5. With* Param *we derive*

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

By Theorem 5 we can substitute equality for $\mathsf{ObsEq}^{Obs_{SP}}$ in Def. 2, the definition of specification. This reduces our formalisms of specification and specification refinement to those of Luo's [20], with the important difference that parametricity lifts the formalisms to observational equivalence. Also any constructor F is now inherently stable, *i.e.* $u\ \mathsf{ObsEq}^{Obs_{SP'}}\ v \ \Rightarrow\ F(u)\ \mathsf{ObsEq}^{Obs_{SP}}\ F(v)$, simply by congruence for equality. Stability simplifies observational proofs significantly.

Theorem 4 means that we can explain observational equivalence, and thus also specification refinement up to observational equivalence, in terms of the existence of simulation relations. At first order, theorems 5 and 4 give the essential property that the existence of simulation relations is transitive, but we can actually give a more constructive result:

Theorem 6 (Composability of Simulation Relations[12]). *Suppose $\mathfrak{T}[X]$ only contains first-order function profiles. Then we can derive*

$$\begin{array}{l}\forall A, B, C, R \subset A \times B, S \subset B \times C, \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B], \mathfrak{c}{:}\mathfrak{T}[C].\\ \qquad\qquad \mathfrak{a}(\mathfrak{T}[R])\mathfrak{b} \ \wedge\ \mathfrak{b}(\mathfrak{T}[S])\mathfrak{c} \ \Rightarrow\ \mathfrak{a}(\mathfrak{T}[S \circ R])\mathfrak{c}\end{array}$$

Thus simulation relations explain stepwise refinement, but methodologically this is not enough. Given instances u and v and constructor F one can check that there is a simulation relation relating (Fu) and v. But this point-wise method of verifying a refinement step is impractical, since it involves choosing candidates v at best heuristically, and then specialised verification is employed for each pair u, v. One would prefer a general method for proving refinement.

Such a universal method exists in algebraic specification, using only abstract information. One proves observational refinements by considering quotients w.r.t. a possibly partial congruence [4], and then one uses an axiomatisation of this congruence to prove relativised versions of the axioms of the specification to be refined. If the congruence is partial, clauses restricting to the domain of the congruence must also be incorporated [5, 3].

As observed in [32, 11], this method is not expressible in the type theory or the logic of [30]. The simple solution of [32, 11] is to soundly add axioms in order to axiomatise partial congruences. We give the axiom schemata below. Rather than being fundamental, they are tailored to suit refinement-proof purposes.

Definition 4 (Existence of Sub-objects (SUB) [11]**).**

$$\begin{array}{l}\forall X\,.\,\forall \mathfrak{x}{:}\mathfrak{T}[X]\,.\,\forall R \subset X\times X\;.\quad (\mathfrak{x}\ \mathfrak{T}[R]\ \mathfrak{x})\ \Rightarrow\\ \exists S\,.\,\exists \mathfrak{s}{:}\mathfrak{T}[S]\,.\,\exists R' \subset S\times S\,.\,\exists \mathsf{mono}{:}S\to X\;.\qquad \forall s{:}S\;.\;s\ R'\ s \qquad\qquad \wedge\\ \qquad\qquad \forall s.s'{:}S\;.\;s\ R'\ s' \;\Leftrightarrow\; (\mathsf{mono}\,s)\ R\ (\mathsf{mono}\,s') \;\wedge\\ \qquad\qquad\quad \mathfrak{x}\ (\mathfrak{T}[(x{:}X, s{:}S).(x =_X (\mathsf{mono}\,s))])\ \mathfrak{s}\end{array}$$

Definition 5 (Existence of Quotients (QUOT) [32]**).**

$$\begin{array}{l}\forall X\,.\,\forall \mathfrak{x}{:}\mathfrak{T}[X]\,.\,\forall R \subset X\times X\;.\quad (\mathfrak{x}\ \mathfrak{T}[R]\ \mathfrak{x}\ \wedge\ \mathit{equiv}(R))\ \Rightarrow\\ \quad \exists Q\,.\,\exists \mathfrak{q}{:}\mathfrak{T}[Q]\,.\,\exists \mathsf{epi}{:}X\to Q\;.\;\forall x,y{:}X\;.\;xRy \;\Leftrightarrow\; (\mathsf{epi}\,x) =_Q (\mathsf{epi}\,y) \;\wedge\\ \qquad\qquad\qquad \forall q{:}Q.\exists x{:}X\;.\;q =_Q (\mathsf{epi}\,x) \qquad\wedge\\ \qquad\qquad\quad \mathfrak{x}\ (\mathfrak{T}[(x{:}X, q{:}Q).((\mathsf{epi}\,x) =_Q q)])\ \mathfrak{q}\end{array}$$

where equiv(R) *specifies* R *to be an equivalence relation.*

Theorem 7. *If* $\mathfrak{T}[X]$ *adheres to* `adt` *and contains only first-order function profiles, then* SUB *and* QUOT *are valid in the parametric PER-model of* [2].

We refer to [32, 41] for concrete examples using QUOT and SUB and to the vast amount of specification examples in the literature, *e.g.* [14] for other examples using this framework. In [11] the axioms are instrumental for the correspondence between refinement in type theory and refinement in algebraic specification.

4 The Simulation Relation at Higher Order

If $\mathfrak{T}[X]$ has higher-order function profiles, theorems 4 and 6 fail, and indeed we cannot even derive that the existence of simulation relations is transitive.

We here take the view that the current notion of simulation relation is unduly demanding, and fails to observe closely enough the abstraction barrier provided by existential types. Consider $\mathsf{prsrv}\cap{:}\,(X\to X)\to \mathsf{bool}$ from specification Set. For $R \subset A\times B$ to be respected by two implementations $\mathfrak{a}$ and $\mathfrak{b}$, one demands $\forall \alpha{:}\,A\to A, \forall \beta{:}\,B\to B\;.\;\alpha(R\to R)\beta \;\Rightarrow\; \mathfrak{a}.\mathsf{prsrv}\cap(\alpha) =_{\mathsf{bool}} \mathfrak{b}.\mathsf{prsrv}\cap(\beta)$. But according to `abs-bar`, $\mathfrak{a}.\mathsf{prsrv}\cap$ and $\mathfrak{b}.\mathsf{prsrv}\cap$ can only be applied to arguments expressible by the supplied operations in $\mathfrak{a}$ and $\mathfrak{b}$. One solution is therefore to alter the relational proof criteria accordingly. Depending on what type of model one is interested in, this can be done in several ways [12]. We here recapture a solution that works in the parametric PER-model. For this we must first refine our notion of observational equivalence.

4.1 Closed Computations

Semantically, observational equivalence is usually defined w.r.t. contexts that when filled, are closed terms. A reasonable alternative definition in the logic of observational equivalence is therefore the following.

Definition 6 (Closed Context Observational Equivalence (ObsEqC) [12]**).** *Define* closed context observational equivalence ObsEqC w.r.t. *Obs by*

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

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

Closedness is qualified by a given context Γ^{In} so as to allow for variables of input types In in observable computations. This is automatically taken care of in the notion of general observable computations of Def 1, but now we are compelled to explicitly specify In. We set $In = Obs$ as a sensible choice [35, 11].

The predicate $\mathsf{Closed}_{\Gamma^{In}}(f)$ 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 7 ([12])**.** *The logical language is extended with families of 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 syntax is given a predefined semantics as follows. For any type* $\Gamma\rhd T$*, term* $\Gamma\rhd t{:}T$*, and valuation* γ *on* $[\![\Gamma]\!]$*,*

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

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

We will usually omit the type argument in the term family of Closed.

4.2 The Alternative Simulation Relation

For a k-ary vector $\boldsymbol{Y}$, we write $\forall\boldsymbol{Y}$ for the string $\forall Y_1.\forall Y_2.\ldots\forall Y_k$. If $k=0$ this denotes the empty string. The first l components of $\boldsymbol{Y}$ are denoted by $\boldsymbol{Y}|_l$.

Definition 8 (Data Type Relation [12]**).** *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*,* $R\subset A\times B$*,* $\mathfrak{a}{:}\mathfrak{T}[A]$*,* $\mathfrak{b}{:}\mathfrak{T}[B]$*, we define the* data type relation $U[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}}$ *for the string* $\varsigma=\boldsymbol{E},\boldsymbol{F},A,B,\mathfrak{a},\mathfrak{b}$ *inductively on* $U[\boldsymbol{Y},X]$ *by*

$$\begin{array}{lll}U=X & : U[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}} & \stackrel{def}{=} R\\ U=Y_i & : U[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}} & \stackrel{def}{=} \rho_i\\ U=\forall X'.U'[\boldsymbol{Y},X',X] & : U[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}} & \stackrel{def}{=}\\ & & \forall(E_{l+1},F_{l+1},\rho_{l+1}\subset E_{l+1}\times F_{l+1})(U'[\boldsymbol{\rho},\rho_{l+1},R]^{E_{l+1},F_{l+1},\varsigma}_{\mathsf{C}})\\ U=U'\to U'' & : U[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}} & \stackrel{def}{=}\end{array}$$

$$(g{:}U'[\boldsymbol{E},A]\to U''[\boldsymbol{E},A],\ h{:}U'[\boldsymbol{F},B]\to U''[\boldsymbol{F},B])\ .\ (\forall x{:}U'[\boldsymbol{E},A],\forall y{:}U'[\boldsymbol{F},B]\ .\\ (x\ U'[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}}\ y\ \ \wedge\ \ \mathsf{DfnbIC}^{\varsigma}_{U'[\boldsymbol{Y},X]}(x,y))\ \ \Rightarrow\ \ (gx)\ U''[\boldsymbol{\rho},R]^{\varsigma}_{\mathsf{C}}\ (hy))$$

where

$$\mathsf{DfnbIC}^{\varsigma}_{U'[\boldsymbol{Y},X]}(x,y)\stackrel{def}{=}\exists f{:}\forall\boldsymbol{Y}.\forall X.(\mathfrak{T}[X]\to U'[\boldsymbol{Y},X])\ .\\ \mathsf{Closed}_{\Gamma^{In}}(f)\ \wedge\ (f\boldsymbol{E}|_k A\,\mathfrak{a})=x\ \wedge\ (f\boldsymbol{F}|_k B\,\mathfrak{b})=y$$

for $\Gamma^{In}=x_1{:}U_1,\ldots,x_m{:}U_m$, $U_i\in In$, $1\leq i\leq m$.

We usually omit the type subscript to the $\mathsf{DfnbIC}^{\varsigma}$ clause. The essence of Def. 8 is the weakened arrow-type relation via the $\mathsf{DfnbIC}^{\varsigma}$ clause; an extension of the relation exhibited for proving Theorem 4. We have conveniently:

Lemma 8 ([12]). *For* $\mathfrak{T}[X]$ *satisfying* $\mathtt{adt}$, *we have the derivability of*

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

Lemma 9 ([12]). *With respect to the parametric PER-model of* [2] *it is sound to assert the following axiom schema for* $D\in Obs$.

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

4.3 Special Parametricity

With the alternative simulation relation in place we should now be able to re-establish versions of theorems 4 and 6 valid for $\mathfrak{T}[X]$ of any order. However, since we do not alter the parametricity axiom schema, we can no longer rely directly on parametricity as in Lemma 4, when deriving observational equivalence from the existence of a simulation relation. However, the needed instance of alternative parametricity can be validated semantically.

We write $f\ (\forall X.\mathfrak{T}[X]^{\epsilon}_{\mathsf{C}}\to U[X]^{\epsilon}_{\mathsf{C}})\ f$, meaning $\forall A,B,R\subset A\times B.\forall\mathfrak{a}{:}\mathfrak{T}[A],\mathfrak{b}{:}\mathfrak{T}[B].$ $\mathfrak{a}(\mathfrak{T}[R]^{\varsigma}_{\mathsf{C}})\mathfrak{b}\ \Rightarrow\ (f A\,\mathfrak{a})(U[R]^{\varsigma}_{\mathsf{C}})(f B\,\mathfrak{b}))$, for $\varsigma=A,B,\mathfrak{a},\mathfrak{b}$.

Lemma 10 ([12]). *For* $\mathfrak{T}[X]$ *adhering to* $\mathtt{adt}$, *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*

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

By Lemma 10 the following schema is sound w.r.t. the parametric PER-model.

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

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

Theorem 11. *With* spParamC, *for* $\mathfrak{T}[X]$ *adhering to* $\mathtt{adt}$, *the following is derivable, for* $\Gamma^{In}=x_1{:}U_1,\ldots,x_m{:}U_m$, $U_i\in In$, $1\leq i\leq m$.

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

We regain not only transitivity of the existence of simulation relations, but also composability of simulation relations. This is akin to recent notions on the semantic level, *i.e. pre-logical relations* [15, 16], *lax logical relations* [31, 19], and *L-relations* [18].

Theorem 12 (Composability of Simulation Relations). *For* $\mathfrak{T}[X]$ *adhering to* $\mathtt{adt}$*, let* $\varsigma = A, B, \mathfrak{a}, \mathfrak{b}$*,* $\varsigma' = B, C, \mathfrak{b}, \mathfrak{c}$*,* $\varsigma'' = A, C, \mathfrak{a}, \mathfrak{c}$*. Given* spParamC*,*

$$\forall A, B, C, R \subset A \times B, S \subset B \times C, \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B], \mathfrak{c}{:}\mathfrak{T}[C].$$
$$\mathfrak{a}(\mathfrak{T}[R]^{\varsigma}_{\mathsf{C}})\mathfrak{b} \ \wedge \ \mathfrak{b}(\mathfrak{T}[S]^{\varsigma'}_{\mathsf{C}})\mathfrak{c} \ \Rightarrow \ \mathfrak{a}(\mathfrak{T}[S \circ R]^{\varsigma''}_{\mathsf{C}})\mathfrak{c}$$

If one is content with syntactic models, one may drop the Closed clause everywhere in the previous discussion. One thus obtains relations $U[\boldsymbol{\rho}, R]^{\varsigma}$ in place of $U[\boldsymbol{\rho}, R]^{\varsigma}_{\mathsf{C}}$, and axioms spParam in place of spParamC, and Ident in place of IdentC. These axiom schema are valid in the parametric term model and the parametric second-order minimum model of Hasegawa [13], based on the polymorphic extensionally collapsed syntactic models of [7] and the second-order maximum consistent theory of [26]. In fact, with spParam we can show that the existence of an alternative simulation relation coincides with the existence of a standard simulation relation. Let $(\exists X.\mathfrak{T}[X])^{\epsilon}$ be the relation defined by

$$(\exists X.\mathfrak{T}[X])^{\epsilon} \stackrel{def}{=} (u{:}\exists X.\mathfrak{T}[X], v{:}\exists X.\mathfrak{T}[X]) \ .$$
$$(\ \forall Y.\forall Z.\forall S \subset Y \times Z \ . \quad \forall f{:}\forall X.\mathfrak{T}[X] \to Y.\ \forall g{:}\forall X.\mathfrak{T}[X] \to Z \ .$$
$$f\ (\forall X'.\mathfrak{T}[X']^{\epsilon} \to S)\ g \ \Rightarrow \ (uYf) \ \ S \ \ (vZg)\ )$$

Theorem 13. *The following is derivable using* spParam.

$$\forall u, v{:}\exists X.\mathfrak{T}[X] \ . \ u =_{\exists X.\mathfrak{T}[X]} v \ \Leftrightarrow \ u\ (\exists X.\mathfrak{T}[X])^{\epsilon}\ v$$

Theorem 14 (Characterisation by Alternative Simulation Relation). *The following is derivable.*

$$\forall u, v{:}\exists X.\mathfrak{T}[X] \ . \quad u\ (\exists X.\mathfrak{T}[X])^{\epsilon}\ v \quad \Leftrightarrow$$
$$\exists A, B.\exists \mathfrak{a}{:}\mathfrak{T}[A], \mathfrak{b}{:}\mathfrak{T}[B].\exists R \subset A \times B \ . \ u = (\mathsf{pack} A \mathfrak{a}) \ \wedge \ v = (\mathsf{pack} B \mathfrak{b}) \ \wedge \ \mathfrak{a}(\mathfrak{T}[R]^{\varsigma})\mathfrak{b}$$

5 The Refinement Proof Strategy at Higher Order

We now have that composable simulation relations explain specification refinement via observational equivalence—for arbitrary order function profiles and limited polymorphism. But what now about the general proof method for proving observational refinement? We do not know whether or not Sub and Quot with higher-order $\mathfrak{T}[X]$ are valid in the parametric PER-model. It is conjectured in [41] but not proven, that Quot and an extension of Sub is valid. In this paper we are however in possession of some additional insight, and we are able to validate versions of Sub and Quot using the alternative simulation relation in an interpretation using the parametric PER-model that reflects `abs-bar`. Again, observing existing abstraction barriers more closely, offers a solution.

To motivate, suppose we attempt to validate QUOT as is, w.r.t. the parametric PER-model. Immediately there is a definitional problem. For any PER $\mathcal{X}$ and element $\boldsymbol{x} \in [\![X \rhd \mathfrak{T}[X]]\!]_{[X \mapsto \mathcal{X}]}$, we should display a quotienting PER $\mathcal{Q}$ and element $\boldsymbol{q} \in [\![X \rhd \mathfrak{T}[X]]\!]_{[X \mapsto \mathcal{Q}]}$ with the desired characteristics. Quotients over an algebra A are usually constructed directly from A with the new operations derived from the original operations. Thus, the natural approach to constructing the operations of $\boldsymbol{q}$ is to use the same realisers that give the operations of $\boldsymbol{x}$.

At first order this is straight-forward, but at higher order this is problematic. Suppose $\boldsymbol{x} = \langle [e_0]_{\mathcal{X}}, [e_1]_{\mathcal{X}\to\mathcal{X}}, [e_2]_{(\mathcal{X}\to\mathcal{X})\to\mathcal{X}}, \ldots\rangle$. We would now like to simply define $\boldsymbol{q}$ as $\langle [e_0]_{\mathcal{Q}}, [e_1]_{\mathcal{Q}\to\mathcal{Q}}, [e_2]_{(\mathcal{Q}\to\mathcal{Q})\to\mathcal{Q}}, \ldots\rangle$. But to be able to do this we must check that the indicated equivalence classes actually exist. Suppose we want to show $e_2\ ((\mathcal{Q} \to \mathcal{Q}) \to \mathcal{Q})\ e_2$. First we must show that if $n\ (\mathcal{Q} \to \mathcal{Q})\ n$ then $e_2(n)\downarrow$. However, this does not follow from the running assumption that $n\ (\mathcal{X} \to \mathcal{X})\ n$ implies $e_2(n)\downarrow$; indeed it is easy to find counter-examples.

The angle of approach we take to this problem is simply a natural continuation of tactics so far, namely we refine the interpretation of ADT-instance operations to reflect exactly their actual applicability as captured in `abs-bar`. For example, above we need clearly only consider arguments n expressible over supplied ADT-instance operations e_0, e_1, e_2, *etc.*

5.1 Observing the ADT-Abstraction Barrier in the Semantics

We now implement this idea. We keep the parametric PER-model as structure, but supply a modified interpretation for ADT operations.

Definition 9 (ADT Semantics). *For any PER $\mathcal{X}$ and $x, y \in \mathbb{N}$,*

$$x\ [\![X \rhd \mathfrak{T}[X]]\!]_{[X\mapsto\mathcal{X}]}\ y \overset{def}{\Leftrightarrow} \textit{for all components } g_i{:}T_i[X] \textit{ in } \mathfrak{T}[X]\quad x.i\ ([\![X \rhd T_i[X]]\!]^{x,y,X,\mathfrak{x}:\mathfrak{T}[X]}_{[X\mapsto\mathcal{X}]})\ y.i$$

where, for $\varrho = x, y, \Gamma$ for $\Gamma = \boldsymbol{Y}, X, \mathfrak{x}{:}\mathfrak{T}[X]$, k-ary $\boldsymbol{Y}$, γ a valuation on $\boldsymbol{Y}, X$,

$$\begin{array}{l}
[\![\boldsymbol{Y}, X \rhd X]\!]^{\varrho}_{\gamma} \overset{def}{=} \gamma(X)\\
[\![\boldsymbol{Y}, X \rhd Y_j]\!]^{\varrho}_{\gamma} \overset{def}{=} \gamma(Y_j)\\
[\![\boldsymbol{Y}, X \rhd \forall X'.U[X', X]]\!]^{\varrho}_{\gamma} \overset{def}{=} (\cap_{A\in\mathbf{PER}} [\![\boldsymbol{Y}, Y_{k+1}, X \rhd U[\boldsymbol{Y}, Y_{k+1}, X]]\!]^{\varrho, Y_{k+1}}_{\gamma[Y_{k+1}\mapsto A]})^{\flat}\\
[\![\boldsymbol{Y}, X \rhd U[\boldsymbol{Y}, X] \to V[\boldsymbol{Y}, X]]\!]^{\varrho}_{\gamma} \overset{def}{=}\\
\qquad ((([\![\boldsymbol{Y}, X \rhd U[\boldsymbol{Y}, X]]\!]^{\varrho}_{\gamma} \cap \mathcal{D}^{\varrho;\gamma}_{U[\boldsymbol{Y},X]}) \to [\![\boldsymbol{Y}, X \rhd V[\boldsymbol{Y}, X]]\!]^{\varrho}_{\gamma})
\end{array}$$

where

$$n\ \mathcal{D}^{\varrho;\gamma}_{U[\boldsymbol{Y},X]}\ n' \overset{def}{\Leftrightarrow} \textit{there exist terms } t, t' \textit{ s.t.}\quad \Gamma \rhd t{:}U[\boldsymbol{Y}, X] \quad \textit{and} \quad \Gamma \rhd t'{:}U[\boldsymbol{Y}, X], \quad \textit{and}\quad t[x] = n \quad \textit{and} \quad t'[y] = n'$$

where $u[z]$ denotes the realiser freely generated over z, according to term u.

The special semantics defined in Def. 9 simply reflects the actual use of ADT operations captured by *abs-bar*. The essence is the weakened condition for arrow-type semantics. All relevant instances of PARAM and spPARAMC hold under this semantics, and previous results hold.

It would be nice to have special types or annotated types for use in ADTs. However, this does not seem feasible without involving recursive (domain-)equations, the solutions of which are not obvious. In the current approach, when to apply ADT semantics is not given by the type system; one simply has to know when to do this. This applies to the logic as well. Moreover, issues of co-existence of ADT semantics and normal semantics have not been thoroughly explored. The presented ADT semantics is thus an interesting solution utilising *abs-bar*, but more work may be necessary in order that this approach be fitted properly.

Here then, are the new versions of SUB and QUOT. Besides referring to the alternative simulation relation, there are also other changes: We need to treat higher-order variables in propositions, so the statements are generalised accordingly. Unfortunately, we cannot yet deal with variables of open universal types in propositions. Below, $U \in \Phi$ ranges over all types U, except open universal types, involving the existentially bound type variable, that occur in the axioms Φ of the relevant specification.

Definition 10 (Existence of Sub-objects (SUBG)**).** *For* $\varsigma = X, S, \mathfrak{x}, \mathfrak{s}$,

$$\begin{array}{l}\forall X\,.\,\forall \mathfrak{x}{:}\mathfrak{T}[X]\,.\,\forall R \subset X \times X\ .\quad (\mathfrak{x}\ \mathfrak{T}[R]\ \mathfrak{x})\ \Rightarrow \\ \quad \exists S\,.\,\exists \mathfrak{s}{:}\mathfrak{T}[S]\,.\,\exists R' \subset S \times S\,.\,\exists_{U\in\Phi}\mathsf{mono}_U{:}U[S] \to U[X]\ . \\ \qquad \bigwedge_{U\in\Phi} \forall s{:}U[S]\ .\ s\ U[R']\ s \qquad\qquad\qquad\qquad\qquad\qquad \wedge \\ \qquad \bigwedge_{U\in\Phi} \forall s.s'{:}U[S]\ .\ s\ U[R']\ s'\ \Leftrightarrow\ (\mathsf{mono}_U\ s)\ U[R]\ (\mathsf{mono}_U\ s')\ \wedge \\ \qquad \mathfrak{x}\ (\mathfrak{T}[(x{:}X, s{:}S).(x =_X (\mathsf{mono}\, s))]^{\varsigma}_{C})\ \mathfrak{s}\end{array}$$

Definition 11 (Existence of Quotients (QUOTG)**).** *For* $\varsigma = X, Q, \mathfrak{x}, \mathfrak{q}$,

$$\begin{array}{l}\forall X\,.\,\forall \mathfrak{x}{:}\mathfrak{T}[X]\,.\,\forall R \subset X \times X\ .\quad (\mathfrak{x}\ \mathfrak{T}[R]\ \mathfrak{x}\ \wedge\ \mathit{equiv}(R))\ \Rightarrow \\ \quad \exists Q\,.\,\exists \mathfrak{q}{:}\mathfrak{T}[Q]\,.\,\exists_{U\in\Phi}\mathsf{epi}_U{:}U[X] \to U[Q]\ . \\ \qquad \bigwedge_{U\in\Phi} \forall x, y{:}U[X]\ .\ x\ U[R]\ y\ \Leftrightarrow\ (\mathsf{epi}_U\ x) =_{U[Q]} (\mathsf{epi}_U\ y)\ \wedge \\ \qquad \bigwedge_{U\in\Phi} \forall q{:}U[Q].\exists x{:}U[X]\ .\ q =_{U[Q]} (\mathsf{epi}_U\ x) \qquad\qquad \wedge \\ \qquad \mathfrak{x}\ (\mathfrak{T}[(x{:}X, q{:}Q).((\mathsf{epi}\, x) =_Q q)]^{\varsigma}_{C})\ \mathfrak{q}\end{array}$$

where $\mathit{equiv}(R)$ *specifies* R *to be an equivalence relation.*

Theorem 15. SUBG *and* QUOTG *are valid in the parametric PER-model of* [2]*, under the assumption of ADT semantics.*

In the following, we will write *e.g.* $[\![U[\mathcal{X}]]\!]$ in place for $[\![X \rhd U[X]]\!]_{[X\mapsto\mathcal{X}]}$. In the following $\Gamma \stackrel{def}{=} X, \mathfrak{x}{:}\mathfrak{T}[X]$.

5.2 Validating SUBG (proof of Theorem 15)

Definition 12 (Sub-object PER). *Let* $\mathcal{X}$ *be any PER, and* $\Re$ *any relation on* $\mathcal{X}$*. Define the* sub-object $R_{\Re}(\mathcal{X})$ restricted on $\Re$ *by*

$$n\ R_{\Re}(\mathcal{X})\ m \stackrel{def}{\Leftrightarrow} n\ \mathcal{X}\ m \quad \textit{and} \quad [n]_{\mathcal{X}}\ \Re\ [m]_{\mathcal{X}}$$

As expected we do not in general have $n\ R_{\Re}(\mathcal{X})\ m \Leftarrow [n]_{\mathcal{X}}\ \Re\ [m]_{\mathcal{X}}$. We do have by definition and symmetry $n\ R_{\Re}(\mathcal{X})\ m \Rightarrow [n]_{\mathcal{X}}\ \Re\ [m]_{\mathcal{X}}$, and also $n\ \mathcal{X}\ m \Leftarrow n\ \mathcal{S}\ m$, but not necessarily the converse implication.

To validate SubG, consider an arbitrary PER $\mathcal{X}$, $\mathsf{x} \in [\![\mathfrak{T}[\mathcal{X}]]\!]$, and relation $\Re$ on $\mathcal{X}$. We must exhibit a PER $\mathcal{S}$, a relation $\Re'$ on $\mathcal{S}$, an $\mathsf{s} \in [\![\mathfrak{T}[\mathcal{S}]]\!]$, and maps $mono_U : [\![U[\mathcal{S}]]\!]^{\varrho} \to [\![U[\mathcal{X}]]\!]^{\varrho'}$, where $\varrho = s, \Gamma$, for s a realiser of s, and $\varrho' = x, \Gamma$, for x a realiser of x, all satisfying the following properties,

Sub-1. For all $s \in [\![U[\mathcal{S}]]\!]^{\varrho}$, $s\ [\![U[\Re']]\!]^{\varrho}\ s$
Sub-2. For all $s, s' \in [\![U[\mathcal{S}]]\!]^{\varrho}$, $s\ [\![U[\Re']]\!]^{\varrho}\ s' \Leftrightarrow mono_U(s)\ [\![U[\Re]]\!]^{\varrho'}\ mono_U(s')$
Sub-3. $\mathsf{x}\ [\![\mathfrak{T}[(x{:}\mathcal{X}, s{:}\mathcal{S}).(x =_{\mathcal{X}} (mono\ s))]^{\varsigma}_{\mathsf{C}}]\!]\ \mathsf{s}$

We exhibit $\mathcal{S} \stackrel{def}{=} R_{\Re}(\mathcal{X})$, define $mono_U([n]_{[\![U[\mathcal{S}]]\!]^{\varrho}}) \stackrel{def}{=} [n]_{[\![U[\mathcal{X}]]\!]^{\varrho'}}$, and define $\Re'$ by $s\ \Re'\ s' \stackrel{def}{\Leftrightarrow} mono(s)\ \Re\ mono(s')$. Well-definedness and (Sub-1) and (Sub-2) follow by definition and from Lemma 18 below.

We now postulate that we can construct s as the k-tuple where the i^{th} component is $[e_i]_{[\![T_i[\mathcal{S}]]\!]^{\varrho}}$ derived from the i^{th} component $[e_i]_{[\![T_i[\mathcal{X}]]\!]^{\varrho'}}$ of x. For each component $g_i : (U[X] \to V[X])$ in $\mathfrak{T}[X]$ we must show for all n, n' s.t. there exist terms t, t' s.t. $\Gamma \rhd t{:}U[X]$, $\Gamma \rhd t'{:}U[X]$, and $t[s] = n$, $t'[s] = n'$, that

verSub-1. $n\ [\![U[\mathcal{S}]]\!]^{\varrho}\ n \Rightarrow (e_i(n)\!\downarrow\ \wedge\ e_i(n)\!\downarrow)$,
verSub-2. $n\ [\![U[\mathcal{S}]]\!]^{\varrho}\ n' \Rightarrow e_i(n)\ [\![V[\mathcal{S}]]\!]^{\varrho}\ e_i(n')$.

The crucial observation that now lets us show the well-definedness of s, is that the realiser s can be assumed to be a realiser x for the existing x. It therefore suffices to show (verSub-1) and (verSub-2) for n, n' s.t. $t[x] = n$ and $t'[x] = n'$.

Lemma 16. *For any PER $\mathcal{X}$, realiser x of x, and relation $\Re$ on $\mathcal{X}$ s.t. $\mathsf{x}\ [\![\mathfrak{T}[\Re]]\!]\ \mathsf{x}$, for any n s.t. there exists a term t s.t. $\Gamma \rhd t{:}X$, and $t[x] = n$,*

$$[n]_{\mathcal{X}}\ \Re\ [n]_{\mathcal{X}}$$

Proof: This follows from a variant of Lemma 10. □

Lemma 17. *For any n, n' s.t. there exist terms t, t' s.t. $\Gamma \rhd t{:}X$, $\Gamma \rhd t'{:}X$, and $t[x] = n$, $t'[x] = n'$, where x is a realiser of x,*

$$n\ \mathcal{X}\ n' \Leftrightarrow n\ \mathcal{S}\ n'$$

Proof: Suppose $n\ \mathcal{X}\ n'$. It suffices to show $[n]_{\mathcal{X}}\ \Re\ [n']_{\mathcal{X}}$. By Lemma 16 we get $[n']_{\mathcal{X}}\ \Re\ [n']_{\mathcal{X}}$, and $n\ \mathcal{X}\ n'$ gives $[n]_{\mathcal{X}} = [n']_{\mathcal{X}}$. Suppose $n\ \mathcal{S}\ n'$. By definition this gives $n\ \mathcal{X}\ n'$. □

Lemma 18. *For any n, n' s.t. there exist terms t, t' s.t. $\Gamma \rhd t{:}U[X]$, $\Gamma \rhd t'{:}U[X]$, and $t[x] = n$, $t'[x] = n'$, where x is a realiser of x,*

$$n\ [\![U[\mathcal{X}]]\!]^{\varrho}\ n' \Leftrightarrow n\ [\![U[\mathcal{S}]]\!]^{\varrho}\ n'$$

Proof: This follows from Lemma 17 by induction on type structure. □

So let n, n' be as proposed, and recall that we are assuming the existence of x, thus we have $e_i\ [\![(U[\mathcal{X}] \to V[\mathcal{X}])]\!]^{\varrho}\ e_i$. We may now use Lemma 18 directly and immediately get what we want since e_i satisfies its conditions. However, for illustrative purposes we go a level down. By assumption we have

assmpSub-1. $n\ [\![U[\mathcal{X}]]\!]^{\varrho}\ n\ \Rightarrow\ (e_i(n)\!\downarrow\ \wedge\ e_i(n)\!\downarrow)$,
assmpSub-2. $n\ [\![U[\mathcal{X}]]\!]^{\varrho}\ n'\ \Rightarrow\ e_i(n)\ [\![V[\mathcal{X}]]\!]^{\varrho}\ e_i(n')$.

Showing (verSub-1) is now easy. By assumption on n, we can use Lemma 18, and then (assmpSub-1) yields $(e_i(n)\!\downarrow\ \wedge e_i(n)\!\downarrow)$. For (verSub-2) assume $n\ [\![U[\mathcal{S}]]\!]^{\varrho}\ n'$. Lemma 18 gives $n\ [\![U[\mathcal{X}]]\!]^{\varrho}\ n'$, and (assmpSub-2) gives $e_i(n)\ [\![V[\mathcal{X}]]\!]^{\varrho}\ e_i(n')$, and then Lemma 18 gives $e_i(n)\ [\![V[\mathcal{S}]]\!]^{\varrho}\ e_i(n')$. This concludes the definition of $\mathfrak{s}$.

It is time to verify $\mathsf{x}\ [\![\mathfrak{T}[(x{:}\mathcal{X}, s{:}\mathcal{S}).(x =_{\mathcal{X}} (mono\ s))]]\!]\ \mathfrak{s}$. First we have

Lemma 19. *For any $D \in Obs$, $n\ (D)^{\varrho}\ m\ \Leftrightarrow\ n\ (D)\ m$*

Let $\rho \stackrel{def}{=} (x{:}\mathcal{X}, s{:}\mathcal{S}).(x =_{\mathcal{X}} (mono\ s))$. For any component $g_i{:}(U[X] \to V[X])$ in $\mathfrak{T}[X]$. we must show for all $[n] \in [\![U[\mathcal{X}]]\!]^{\varrho}$ and $[m] \in [\![U[\mathcal{S}]]\!]^{\varrho}$ that

$$[n]\ [\![U[\rho]]\!]^{\varrho}\ [m]\ \wedge\ [\![\mathsf{DfnblC}^{\mathsf{s}}([n],[m])]\!]\ \Rightarrow\ [e_i(n)]\ [\![V[\rho]]\!]^{\varrho}\ [e_i(m)]$$

Let e be the realiser of the polymorphic functional of which $\mathsf{DfnblC}^{\mathsf{s}}$ asserts the existence. This realiser is the same for all instances of this functional, and by construction the realiser for x and the realiser for $\mathfrak{s}$ are the same, say x. Hence the $\mathsf{DfnblC}^{\mathsf{s}}$ clause asserts that $e(x) = n$ and $e(x) = m$, thus $n = m$. If $V[X]$ is X we must show $[e_i(n)]_{\mathcal{X}} = mono([e_i(m)]_{\mathcal{S}})$, *i.e.* $[e_i(n)]_{\mathcal{X}} = [e_i(m)]_{\mathcal{X}}$, and if $V[X]$ is some $D \in Obs$ we must show $[e_i(n)]_D = [e_i(m)]_D$. Both cases follow since $n = m$. To deal with $V[X]$ a universal type according to $\mathtt{adt}$, generalise the proof to incorporate quantifier-introduced $\boldsymbol{Y}$, omitted here for clarity.

5.3 Validating QuotG (proof of Theorem 15)

Definition 13 (Quotient PER). *Let $\mathcal{X}$ be any PER, and $\Re$ any equivalence relation on $\mathcal{X}$. Define the* quotient $\mathcal{X}/\!\!\Re$ *of $\mathcal{X}$* w.r.t. $\sim$ *by*

$$n\ \mathcal{X}/\!\!\Re\ m\ \stackrel{def}{\Leftrightarrow}\ n\ \mathcal{X}\ n\ \ and\ \ m\ \mathcal{X}\ m\ \ and\ \ [n]_{\mathcal{X}}\ \Re\ [m]_{\mathcal{X}}$$

The following lemma follows immediately by definition.

Lemma 20. *For all n, m, we have, provided that $[n]_{\mathcal{X}}$ and $[m]_{\mathcal{X}}$ exist,*

$$n\ \mathcal{X}/\!\!\Re\ m\ \Leftrightarrow\ [n]_X\ \Re\ [m]_X$$

We also have by definition and reflexivity of $\Re$, $n\ \mathcal{X}\ m\ \Rightarrow\ n\ \mathcal{Q}\ m$, but not necessarily the converse implication.

To validate QuotG, consider any PER $\mathcal{X}$, $\mathsf{x} \in [\![\mathfrak{T}[\mathcal{X}]]\!]$, and equivalence relation $\Re$ on $\mathcal{X}$. We must exhibit a PER $\mathcal{Q}$, a $\mathsf{q} \in [\![\mathfrak{T}[\mathcal{Q}]]\!]$, and maps $epi_U{:}$ $[\![U[\mathcal{X}]]\!]^{\varrho} \to [\![U[\mathcal{Q}]]\!]^{\varrho}$, where $\varrho = q, \Gamma$, for q a realiser of q, and $\varrho' = x, \Gamma$, for x a realiser of x, all satisfying the following properties.

QUOT-1. For all $x, y \in [\![U[\mathcal{X}]]\!]^{\varrho'}$, $x\ [\![U[\Re]]\!]^{\varrho'}\ y \ \Leftrightarrow\ epi_U(x) =_{[\![U[\mathcal{Q}]]\!]^{\varrho}} epi_U(y)$
QUOT-2. For all $q \in [\![U[\mathcal{Q}]]\!]^{\varrho}$, there exists $x \in [\![U[\mathcal{X}]]\!]^{\varrho}$ s.t. $q =_{[\![U[\mathcal{Q}]]\!]} epi_U(x)$
QUOT-3. $\boldsymbol{x}\ [\![\mathfrak{T}[(x{:}\mathcal{X}, s{:}\mathcal{Q}).(epi(x) =_{\mathcal{X}} q)]^{\varsigma}_{\mathsf{C}}]\!]\ \boldsymbol{q}$

We exhibit $\mathcal{Q} \stackrel{def}{=} \mathcal{X}/\Re$, and define $epi_U([n]_{[\![U[\mathcal{X}]]\!]^{\varrho'}}) \stackrel{def}{=} [n]_{[\![U[\mathcal{Q}]]\!]^{\varrho}}$. Well-definedness, (QUOT-1) and (QUOT-2) follow by definition and Lemma 20.

Both the construction of $\boldsymbol{q}$ and the rest of the proof follow analogously to the case for SUBG. Things are a bit simpler, because we have lemma 20.

5.4 Using QUOTG and SUBG

We illustrate the general use of QUOTG and SUBG in proving a refinement $SP \rightsquigarrow SP'$. For clarity we omit constructors, and then $\mathfrak{T}_{SP}[X] = \mathfrak{T}_{SP'}[X]$. We denote both by $\mathfrak{T}[X]$. We also assume $Obs = Obs_{SP} = Obs_{SP'}$, and for brevity we assume equational axioms. As mentioned before, QUOTG and SUBG cannot as yet deal with variables of open universal types in the axioms of specifications.

The task is to derive $\forall u{:}\, Sig_{SP'}\ .\ \Theta_{SP'}(u)\ \Rightarrow\ \Theta_{SP}(u)$, in other words, for $u{:}\, Sig_{SP'}$, assuming $\exists A.\exists \mathfrak{a}{:}\mathfrak{T}[A]\ .\ (\mathsf{pack}A\mathfrak{a})\ \mathsf{ObsEqC}^{Obs}\ u\ \wedge\ \Phi_{SP'}[A, \mathfrak{a}]$ we must derive $\exists B.\exists \mathfrak{b}{:}\mathfrak{T}[B]\ .\ (\mathsf{pack}B\mathfrak{b})\ \mathsf{ObsEqC}^{Obs}\ u\ \wedge\ \Phi_{SP}[B, \mathfrak{b}]$.

Following the strategy of algebraic specification, we attempt to define a (partial) congruence $\sim$ on A and show $\Phi_{SP}[A, \mathfrak{a}]_{rel}$, where $\Phi_{SP}[A, \mathfrak{a}]_{rel}$ is obtained from $\Phi_{SP}[A, \mathfrak{a}]$ by replacing all occurrences of $=_{U[A]}$ by $U[\sim]$ (justified by extensionality), and in case $\sim$ is partial, also conditioning every formula ϕ whose free variables of types $U_i[A]$ are among $x_1, \ldots, x_n$, by $\wedge_i(x_i\ U_i[\sim]\ x_i)\ \Rightarrow\ \phi$.

Suppose this succeeds. Then since $\sim$ is an axiomatisation of a partial congruence, we have $\mathfrak{a}\ \mathfrak{T}[\sim]\ \mathfrak{a}$. We use SUBG to get S_A, $\mathfrak{s}_{\mathfrak{a}}$ and $\sim' \subset S_A \times S_A$, and maps $\mathsf{mono}_U{:}\, U[S_A] \to U[A]$ such that we can derive

$(s1)\ \bigwedge_{U \in \Phi_{SP}} \forall s{:}\, U[S_A]\ .\ s\ U[\sim']\ s$
$(s2)\ \bigwedge_{U \in \Phi_{SP}} \forall s.s'{:}\, U[S_A]\ .\ s\ U[\sim']\ s'\ \Leftrightarrow\ (\mathsf{mono}_U\ s)\ U[\sim]\ (\mathsf{mono}_U\ s')$
$(s3)\ \mathfrak{a}\ (\mathfrak{T}[(a{:}\, A, s{:}\, S_A).(a =_A (\mathsf{mono}\ s))]^{\varsigma}_{\mathsf{C}})\ \mathfrak{s}_{\mathfrak{a}}$

By $(s2)$ we get $\mathfrak{s}_{\mathfrak{a}}\ \mathfrak{T}[\sim']\ \mathfrak{s}_{\mathfrak{a}}$. We also get $equiv(\sim')$ by $(s1)$. We now use QUOTG to get Q and $\mathfrak{q}{:}\mathfrak{T}[Q]$ and maps $\mathsf{epi}_U{:}\, U[S_A] \to U[Q]$ s.t.

$(q1)\ \bigwedge_{U \in \Phi_{SP}} \forall s, s'{:}\, U[S_A]\ .\ s\ U[\sim']\ s'\ \Leftrightarrow\ (\mathsf{epi}_U\ s) =_{U[Q]} (\mathsf{epi}_U\ s')$
$(q2)\ \bigwedge_{U \in \Phi_{SP}} \forall q{:}\, U[Q].\exists s{:}\, U[S_A]\ .\ q =_{U[Q]} (\mathsf{epi}_U\ s)$
$(q3)\ \mathfrak{s}_{\mathfrak{a}}\ (\mathfrak{T}[(s{:}\, S_A, q{:}\, Q).((\mathsf{epi}\ s) =_Q q)]^{\varsigma'}_{\mathsf{C}})\ \mathfrak{q}$

Thus we should exhibit Q for B, and $\mathfrak{q}$ for $\mathfrak{b}$; and it then remains to derive 1. $(\mathsf{pack}Q\mathfrak{q})\ \mathsf{ObsEqC}^{Obs}(\mathsf{pack}A\mathfrak{a})$, and 2. $\Phi_{SP}[Q, \mathfrak{q}]$. To show the derivability of (1), it suffices to observe that, through Theorem 11, $(s3)$ and $(q3)$ give $(\mathsf{pack}A\mathfrak{a})\ \mathsf{ObsEqC}^{Obs}\ (\mathsf{pack}S_A\mathfrak{s}_{\mathfrak{a}})\ \mathsf{ObsEqC}^{Obs}\ (\mathsf{pack}Q\mathfrak{q})$. For (2) we must show the derivability of $\forall \boldsymbol{x}_Q.u[Q, \mathfrak{q}] =_{V[Q]} v[Q, \mathfrak{q}]$ for every $\forall \boldsymbol{x}_X.u =_{V[X]} v$ in $\Phi_{SP}[X, \mathfrak{x}]$.

We may by extensionality assume that V is X or some $D \in Obs$. For any variable $q{:}\, U[Q]$ in $u[Q, \mathfrak{q}]$ or $v[Q, \mathfrak{q}]$, we may by $(q2)$ assume an $s_q{:}\, U[S_A]$ s.t. $(\mathsf{epi}_U\ s_q) =_{U[Q]} q$. Since we succeeded in deriving $\Phi_{SP}[A, \mathfrak{a}]_{rel}$, we can derive

$((\mathsf{mono}_U\ s_q)\ U[\sim]\ (\mathsf{mono}_U\ s_q))\ \Rightarrow\ u[A,\mathfrak{a}][(\mathsf{mono}_U\ s_q)]\ V[\sim]\ v[A,\mathfrak{a}][(\mathsf{mono}_U\ s_q)]$ (for clarity only displaying one variable). By $(s2)$ and $(s3)$ this is equivalent to $s_q\ U[\sim']\ s_q\ \Rightarrow\ u[S_A,\mathfrak{s}_\mathfrak{a}][s_q]\ V[\sim']\ v[S_A,\mathfrak{s}_\mathfrak{a}][s_q]$, which by $(s1)$ is equivalent to $u[S_A,\mathfrak{s}_\mathfrak{a}][s_q]\ V[\sim']\ v[S_A,\mathfrak{s}_\mathfrak{a}][s_q]$. Then from $(q1)$, or if $V = D \in Obs$ we have this immediately, we can derive $(\mathsf{epi}_V\ u[S_A,\mathfrak{s}_\mathfrak{a}][s_q]) =_{V[Q]} (\mathsf{epi}_V\ v[S_A,\mathfrak{s}_\mathfrak{a}][s_q])$. By $(q3)$ we get $(\mathsf{epi}_V\ u[S_A,\mathfrak{s}_\mathfrak{a}][s_q]) = u[Q,\mathfrak{q}]$ and $(\mathsf{epi}_V\ v[S_A,\mathfrak{s}_\mathfrak{a}][s_q]) = v[Q,\mathfrak{q}]$.

This illustration is somewhat oversimplified. In most concrete examples where SubG would be necessary, one would at least find useful the hiding constructor

$$\lambda u{:}\,Sig_{SP'}.(\mathsf{unpack}(u)(Sig_{SP})\ (\Lambda X.\lambda\mathfrak{x}{:}\mathfrak{T}_{SP'}[X].(\mathsf{pack}\ X\ \mathit{record}(\texttt{those items of}\ \mathfrak{x}\ \texttt{matching}\ Sig_{SP})))$$

6 Final Remarks

This paper has described specification refinement up to observational equivalence for specifications involving operations of any order and a limited form of polymorphism. This was done in System F using extensions of Plotkin and Abadi's logic for parametric polymorphism, which are sound w.r.t. the parametric PER-model. We established in the logic a common general method for proving observational refinements, and we established in the logic a simulation relation that composes at any order.

The main stratagem of this paper was to observe more closely existing abstraction barriers provided by existential types. This is fruitful both in devising the alternative simulation relation, and in providing a semantics to validate the higher-order version of the proof method for showing observational refinement.

In future work, one should clarify how the ADT semantics outlined in this paper should be integrated into a wider semantical framework. One should also try to find a counterpart in the syntax for the ADT-semantics. An interesting alternative to all of this would perhaps be to impose suitable abstraction barriers in the logical deduction part of the system instead, extending ideas in [10].

Obvious links to ongoing work on the semantic level concerning data refinement and specification refinement should be clarified. This promises interesting results in both directions. For example in System F and the logic, semantic notions such as applicative structures and combinatory algebras can be internalised in syntax. The semantic notion of pre-logical relations could therefore also be internalised. The problem posed by an infinite family of relations might be solved by using definability w.r.t. the relevant ADT according to *abs-bar*.

We have added axioms to the logic, but we have not made any model-theoretical deliberations outside relating the discussion mainly to the parametric PER-model. It is for example highly relevant to investigate the general power of these axioms in restricting the class of models. This would especially comes into play if one were to consider consistency w.r.t. related type systems.

Acknowledgements Many thanks to John Longley for a crucial hint concerning the ADT semantics. Thanks also to Martin Hofmann, Furio Honsell, and Don Sannella for valuable discussions. Thanks to the referees for helpful comments.

References

1. D. Aspinall. *Type Systems for Modular Programs and Specifications*. PhD thesis, University of Edinburgh, 1998.
2. E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.
3. M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165:3–55, 1996.
4. M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25:149–186, 1995.
5. M. Bidoit, R. Hennicker, and M. Wirsing. Proof systems for structured specifications with observability operators. *Theoretical Computer Sci.*, 173:393–443, 1997.
6. C. Böhm and A. Beraducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.
7. V. Breazu-Tannen and T. Coquand. Extensional models for polymorphism. *Theoretical Computer Science*, 59:85–114, 1988.
8. 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.
9. J.A. Goguen. Parameterized programming. *IEEE Transactions on Software Engineering*, SE-10(5):528–543, 1984.
10. J.E. Hannay. Abstraction barriers in equational proof. In *Proc. of AMAST'98*, volume 1548 of *LNCS*, pages 196–213, 1998.
11. J.E. Hannay. Specification refinement with System F. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 530–545, 1999.
12. J.E. Hannay. A higher-order simulation relation for System F. In *Proc. FOSSACS 2000*, volume 1784 of *LNCS*, pages 130–145, 2000.
13. 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.
14. R. Hennicker. Structured specifications with behavioural operators: Semantics, proof methods and applications. Habilitationsschrift, LMU, München, 1997.
15. F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement in typed lambda calculus. In *Proc. FOSSACS 2000*, volume 1784 of *LNCS*, pages 161–176, 2000.
16. F. Honsell and D. Sannella. Pre-logical relations. In *Proc. CSL'99*, volume 1683 of *LNCS*, pages 546–561, 1999.
17. S. Kahrs, D. Sannella, and A. Tarlecki. The definition of Extended ML: a gentle introduction. *Theoretical Computer Science*, 173:445–484, 1997.
18. 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.
19. 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.
20. Z. Luo. Program specification and data type refinement in type theory. *Math. Struct. in Comp. Sci.*, 3:333–363, 1993.
21. 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.

22. 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.
23. 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.
24. J.C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
25. J.C. Mitchell and G.D. Plotkin. Abstract types have existential type. *ACM Trans. on Programming Languages and Systems*, 10(3):470–502, 1988.
26. E. Moggi and R. Statman. The maximum consistent theory of the second order lambda calculus. e-mail to Types list. Available at `ftp://ftp.disi.unige.it/person/MoggiE/papers/maxcons`, 1986.
27. N. Mylonakis. Behavioural specifications in type theory. In *Recent Trends in Data Type Spec., 11th WADT*, volume 1130 of *LNCS*, pages 394–408, 1995.
28. 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.
29. A.M. Pitts. Existential types: Logical relations and operational equivalence. In *Proc. ICALP'98*, volume 1443 of *LNCS*, pages 309–326, 1998.
30. G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA 93*, volume 664 of *LNCS*, pages 361–375, 1993.
31. G.D. Plotkin, A.J. Power, and D. Sannella. Lax logical relations. To appear in *Proc. ICALP 2000, LNCS*, 2000.
32. 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.
33. 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.
34. J.C. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
35. D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences*, 34:150–178, 1987.
36. D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Inform.*, 25(3):233–281, 1988.
37. D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.
38. O. Schoett. *Data Abstraction and the Correctness of Modular Programming*. PhD thesis, University of Edinburgh, 1986.
39. 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.
40. J. Underwood. Typing abstract data types. In *Recent Trends in Data Type Spec., Proc. 10th WADT*, volume 906 of *LNCS*, pages 437–452, 1994.
41. J. Zwanenburg. *Object-Oriented Concepts and Proof Rules: Formalization in Type Theory and Implementation in Yarrow*. PhD thesis, Technische Universiteit Eindhoven, 1999.