Semantic and Syntactic Approaches to Simulation Relations*

Jo Hannay[1], Shin-ya Katsumata[2], and Donald Sannella[2]

[1] Department of Software Engineering, Simula Research Laboratory
[2] Laboratory for Foundations of Computer Science, University of Edinburgh

Abstract. Simulation relations are tools for establishing the correctness of data refinement steps. In the simply-typed lambda calculus, logical relations are the standard choice for simulation relations, but they suffer from certain shortcomings; these are resolved by use of the weaker notion of pre-logical relations instead. Developed from a syntactic setting, abstraction barrier-observing simulation relations serve the same purpose, and also handle polymorphic operations. Meanwhile, second-order pre-logical relations directly generalise pre-logical relations to polymorphic lambda calculus (System F). We compile the main refinement-pertinent results of these various notions of simulation relation, and try to raise some issues for aiding their comparison and reconciliation.

1 Introduction

One of the central activities involved in stepwise development of programs is the transformation of "abstract programs" involving types of data that are not normally available as primitive in programming languages (graphs, sets, etc.) into "concrete programs" in which a representation of these in terms of simpler types of data (integers, arrays, etc.) is provided. Apart from the change to data representation, such *data refinement* should have no effect on the results computed by the program: the concrete program should be equivalent to the abstract program in the sense that all computational observations should return the same results in both cases.

The usual way of establishing this property, known as *observational equivalence*, is by exhibiting a *simulation relation* that gives a correspondence between the data values involved in the two programs that is respected by the functions they implement. The details depend on the nature of the language in which the programs are written. In the simple case of a language with only first-order functions, it is usually enough to use an invariant on the domain of concrete values together with a function mapping concrete values (that satisfy the invariant) to abstract values [Hoa72], but a strictly more general method is to use a homomorphic relation [Mil71], [Sch90], [ST97]. If non-determinism is present in the language then some kind of bisimulation relation is required.

* This research was partly supported by the MRG project (IST-2001-33149) which is funded by the EC under the FET proactive initiative on Global Computing. SK was supported by an LFCS studentship.

When the language in question is the simply-typed lambda calculus, the standard choice of simulation relation — which originates with Reynolds in [Rey81,Rey83] but is described most clearly in [Ten94], cf. [Mit96] — is to use a *logical relation*, a type-indexed family of relations that respects not just function application (like homomorphisms) but also lambda abstraction. Logical relations are used extensively in the study of typed lambda calculus and have applications outside lambda calculus. A problem with the use of logical relations, in connection with data refinement and other applications, is the fact that they lack some convenient algebraic properties; in particular, the composition of two logical relations is not in general a logical relation. This calls into question their application to data refinement at least, where one might expect composition to account for the correctness of *stepwise* refinement.

An alternative is to use instead a *pre-logical relation* [HS02], a weaker form of logical relations that nevertheless has many of the features that make logical relations so useful as well as being composable. This yields a proof method for establishing observational equivalence that is not just sound, as with logical relations, but is also complete. The use of pre-logical relations in data refinement is studied in [HLST00].

The situation is more complicated when we consider polymorphically typed lambda calculi such as System F [Gir71,Rey74]. Pre-logical relations can be extended to this context, see [Lei01], but then they do not compose in general although they remain sound and complete for observational equivalence. At the same time, the power of System F opens the possibility of taking a syntactic approach, placing the concept of simulation relation in a logical setting and using existential type quantification for data abstraction [MP88]. This line of development has been investigated in a string of papers on *abstraction barrier-observing simulation relations* by Hannay [Han99,Han00,Han01,Han03] based on a logic for parametric polymorphism due to Plotkin and Abadi [PA93]. A clear advantage of such an approach is that it is amenable to computer-aided reasoning but there are certain compromises forced by the syntactic nature of the framework.

We present this background in Sects. 2–4 and then make a number of remarks aiming at some kind of reconciliation in Sect. 5. There are more questions than answers but some possible lines of enquiry are suggested.

2 Pre-Logical Relations

Our journey begins with $\lambda^{\rightarrow}$, the simply-typed lambda calculus having $\rightarrow$ as its only type constructor.

Definition 2.1. *The set of* types *over a set B of* base types *(or* type constants*) is given by the grammar $\sigma ::= b \mid \sigma \rightarrow \sigma$ where b ranges over B. A* signature Σ *consists of a set B of type constants and a collection C of typed* term constants $c : \sigma$. *$Types^{\rightarrow}(\Sigma)$ denotes the set of types over B.*

Σ-terms are given by the grammar $M ::= x \mid c \mid \lambda x{:}\sigma.M \mid M\ M$ where x ranges over variables and c over term constants. The usual typing rules associate

each well-formed term M in a Σ-*context* $\Gamma = x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ with a type $\sigma \in \mathit{Types}^{\rightarrow}(\Sigma)$, written $\Gamma \rhd M : \sigma$. If Γ is empty then we write simply $M : \sigma$.

Definition 2.2. *A* Σ-combinatory algebra $\mathcal{A}$ *consists of:*

- *a* carrier *set* $[\![\sigma]\!]^{\mathcal{A}}$ *for each* $\sigma \in \mathit{Types}^{\rightarrow}(\Sigma)$*;*
- *a function* $\mathit{App}_{\mathcal{A}}^{\sigma,\tau} : [\![\sigma \to \tau]\!]^{\mathcal{A}} \to [\![\sigma]\!]^{\mathcal{A}} \to [\![\tau]\!]^{\mathcal{A}}$ *for each* $\sigma, \tau \in \mathit{Types}^{\rightarrow}(\Sigma)$*;*
- *an element* $[\![c]\!]^{\mathcal{A}} \in [\![\sigma]\!]^{\mathcal{A}}$ *for each term constant* $c : \sigma$ *in* Σ*; and*
- combinators $K_{\mathcal{A}}^{\sigma,\tau} \in [\![\sigma \to (\tau \to \sigma)]\!]^{\mathcal{A}}$ *and* $S_{\mathcal{A}}^{\rho,\sigma,\tau} \in [\![(\rho \to \sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau]\!]^{\mathcal{A}}$ *for each* $\rho, \sigma, \tau \in \mathit{Types}^{\rightarrow}(\Sigma)$

such that $K_{\mathcal{A}}^{\sigma,\tau}\, x\, y = x$ *(i.e.* $\mathit{App}_{\mathcal{A}}^{\tau,\sigma}\,(\mathit{App}_{\mathcal{A}}^{\sigma,\tau\to\sigma}\, K_{\mathcal{A}}^{\sigma,\tau}\, x)\, y = x$*) and* $S_{\mathcal{A}}^{\rho,\sigma,\tau}\, x\, y\, z = (x\, z)(y\, z)$ *(ditto).*

A Γ-*environment* η on a combinatory algebra $\mathcal{A}$ assigns elements of $\mathcal{A}$ to variables, with $\eta(x) \in [\![\sigma]\!]^{\mathcal{A}}$ for $x : \sigma$ in Γ. A Σ-term $\Gamma \rhd M : \sigma$ is interpreted in $\mathcal{A}$ under a Γ-environment η in the usual way with λ-abstraction interpreted via translation to combinators, written $[\![\Gamma \rhd M : \sigma]\!]_{\eta}^{\mathcal{A}}$, and this is an element of $[\![\sigma]\!]^{\mathcal{A}}$. If M is closed then we write simply $[\![M : \sigma]\!]^{\mathcal{A}}$.

A signature Σ models the interface (type names and function names) of a functional program, with Σ-combinatory algebras modelling programs that match interface Σ. Observational equivalence of two Σ-combinatory algebras is then fundamental to the notion of data refinement.

Definition 2.3. *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* Σ*-combinatory algebras and let* OBS*, the* observable types*, be a subset of* $\mathit{Types}^{\rightarrow}(\Sigma)$*. Then* $\mathcal{A}$ *is* observationally equivalent *to* $\mathcal{B}$ *with respect to* OBS*, written* $\mathcal{A} \equiv_{\mathit{OBS}} \mathcal{B}$*, if for any two closed* Σ*-terms* $M, N : \sigma$ *for* $\sigma \in \mathit{OBS}$*,* $[\![M : \sigma]\!]^{\mathcal{A}} = [\![N : \sigma]\!]^{\mathcal{A}}$ *iff* $[\![M : \sigma]\!]^{\mathcal{B}} = [\![N : \sigma]\!]^{\mathcal{B}}$*.*

It is usual to take OBS to be the "built-in" types for which equality is decidable, for instance *bool* and/or *nat*. Then $\mathcal{A}$ and $\mathcal{B}$ are observationally equivalent iff it is not possible to distinguish between them by performing computational experiments. Note that $\mathit{OBS} \subseteq \mathit{OBS}'$ implies $\equiv_{\mathit{OBS}} \supseteq \equiv'_{\mathit{OBS}}$.

Logical relations are structure-preserving relations on combinatory algebras.

Definition 2.4. *A* logical relation $\mathcal{R}$ *over* Σ*-combinatory algebras* $\mathcal{A}$ *and* $\mathcal{B}$ *is a family of relations* $\{R^{\sigma} \subseteq [\![\sigma]\!]^{\mathcal{A}} \times [\![\sigma]\!]^{\mathcal{B}}\}_{\sigma \in \mathit{Types}^{\rightarrow}(\Sigma)}$ *such that:*

- $R^{\sigma\to\tau}(f, g)$ *iff* $\forall a \in [\![\sigma]\!]^{\mathcal{A}}.\forall b \in [\![\sigma]\!]^{\mathcal{B}}.R^{\sigma}(a, b) \Rightarrow R^{\tau}(\mathit{App}_{\mathcal{A}}\, f\, a, \mathit{App}_{\mathcal{B}}\, g\, b)$*.*
- $R^{\sigma}([\![c]\!]^{\mathcal{A}}, [\![c]\!]^{\mathcal{B}})$ *for every term constant* $c : \sigma$ *in* Σ*.*

For $\mathit{OBS} = \{\mathit{nat}\}$, the connection between logical refinement and observational equivalence is given by Mitchell's representation independence theorem.

Theorem 2.5 (Representation Independence [Mit96]). *Let* Σ *be a signature that includes a type constant nat, and let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* Σ*-combinatory algebras*[3] *with* $[\![\mathit{nat}]\!]^{\mathcal{A}} = [\![\mathit{nat}]\!]^{\mathcal{B}} = \mathbb{N}$*. If there is a logical relation* $\mathcal{R}$ *over* $\mathcal{A}$ *and*

[3] Actually Henkin models, which are extensional combinatory algebras; however extensionality is not a necessary condition for this theorem.

$\mathcal{B}$ with R^{nat} the identity relation on natural numbers, then $\mathcal{A} \equiv_{\{nat\}} \mathcal{B}$. Conversely, if $\mathcal{A} \equiv_{\{nat\}} \mathcal{B}$, Σ provides a closed term for each element of $\mathbb{N}$, and Σ contains only first-order term constants, then there is a logical relation $\mathcal{R}$ over $\mathcal{A}$ and $\mathcal{B}$ with R^{nat} the identity relation. □

This theorem corresponds directly to the following method for establishing the correctness of data refinement steps.

Proof method ([Ten94]). *Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-combinatory algebras and let $OBS \subseteq Types^{\rightarrow}(\Sigma)$. To show that $\mathcal{B}$ is a refinement of $\mathcal{A}$, find a logical relation $\mathcal{R}$ over $\mathcal{A}$ and $\mathcal{B}$ such that R^{σ} is the identity relation for each $\sigma \in OBS$. We then say that $\mathcal{B}$ is a* logical refinement *of $\mathcal{A}$ and write $\mathcal{A} \rightsquigarrow \mathcal{B}$, or $\mathcal{A} \overset{\mathcal{R}}{\rightsquigarrow} \mathcal{B}$ when we want to make $\mathcal{R}$ explicit.*

A well-known problem with logical relations is the fact that they are not closed under composition. It follows that, given logical refinements $\mathcal{A} \overset{\mathcal{R}}{\rightsquigarrow} \mathcal{B}$ and $\mathcal{B} \overset{\mathcal{S}}{\rightsquigarrow} \mathcal{C}$, the composition $\mathcal{S} \circ \mathcal{R}$ cannot in general be used as a witness for the composed refinement $\mathcal{A} \rightsquigarrow \mathcal{C}$. (In fact, the problem is more serious than it appears at first: sometimes there is no witness for $\mathcal{A} \rightsquigarrow \mathcal{C}$ at all.) This is at odds with the *stepwise* nature of refinement, and the transitivity of the underlying notion of observational equivalence. It is one source of examples demonstrating the incompleteness of the above proof method; there are other examples that do not involve composition of refinement steps, see [HLST00].

The restriction to signatures with first-order term constants in the second part of Theorem 2.5 is necessary, and this is the key to the incompleteness of logical refinements as a proof method and the problem with composability of logical refinements. If $\mathcal{A} \rightsquigarrow \mathcal{B} \rightsquigarrow \mathcal{C}$ then $\mathcal{A} \equiv_{OBS} \mathcal{B} \equiv_{OBS} \mathcal{C}$, and so $\mathcal{A} \equiv_{OBS} \mathcal{C}$ since $\equiv_{OBS}$ is an equivalence relation. But then it follows that $\mathcal{A} \rightsquigarrow \mathcal{C}$ only for signatures without higher-order term constants.

In [HS02], a weakening of the notion of logical relations called *pre-logical relations* was studied; see [PPST00] for a categorical formulation.

Definition 2.6 ([HS02]). *An* algebraic relation *$\mathcal{R}$ over Σ-combinatory algebras $\mathcal{A}, \mathcal{B}$ is a family of relations $\{R^{\sigma} \subseteq [\![\sigma]\!]^{\mathcal{A}} \times [\![\sigma]\!]^{\mathcal{B}}\}_{\sigma \in Types^{\rightarrow}(\Sigma)}$ such that:*

- *If $R^{\sigma \rightarrow \tau}(f, g)$ then $\forall a \in [\![\sigma]\!]^{\mathcal{A}}. \forall b \in [\![\sigma]\!]^{\mathcal{B}}. R^{\sigma}(a, b) \Rightarrow R^{\tau}(App_{\mathcal{A}}\, f\, a, App_{\mathcal{B}}\, g\, b)$.*
- *$R^{\sigma}([\![c]\!]^{\mathcal{A}}, [\![c]\!]^{\mathcal{B}})$ for every term constant $c : \sigma$ in Σ.*

A pre-logical relation *$\mathcal{R}$ is an algebraic relation such that:*

- *$R(S^{\rho,\sigma,\tau}_{\mathcal{A}}, S^{\rho,\sigma,\tau}_{\mathcal{B}})$ and $R(K^{\sigma,\tau}_{\mathcal{A}}, K^{\sigma,\tau}_{\mathcal{B}})$ for all $\rho, \sigma, \tau \in Types^{\rightarrow}(\Sigma)$.*

The idea of this definition is to replace the reverse implication in the definition of logical relations with a requirement that the relation contains the S and K combinators. Since these suffice to express all lambda terms, this amounts to requiring the reverse implication to hold only for pairs of functions that are expressible by the same lambda term. It is easy to see that any logical relation is a pre-logical relation.

Example 2.7. A Σ-*homomorphism* $h : \mathcal{A} \to \mathcal{B}$ is a type-indexed family of functions $\{h^\sigma : [\![\sigma]\!]^{\mathcal{A}} \to [\![\sigma]\!]^{\mathcal{B}}\}_{\sigma \in Types^{\to}(\Sigma)}$ such that for any term constant $c : \sigma$ in Σ, $h^\sigma([\![c]\!]^{\mathcal{A}}) = [\![c]\!]^{\mathcal{B}}$, $h^\tau(App_{\mathcal{A}}^{\sigma,\tau}\ f\ a) = App_{\mathcal{B}}^{\sigma,\tau}\ h^{\sigma\to\tau}(f)\ h^\sigma(a)$ and $h^{\sigma\to\tau}([\![\Gamma \rhd \lambda x{:}\sigma.M : \sigma \to \tau]\!]^{\mathcal{A}}_{\eta_{\mathcal{A}}}) = [\![\Gamma \rhd \lambda x{:}\sigma.M : \sigma \to \tau]\!]^{\mathcal{B}}_{h\circ\eta_{\mathcal{A}}}$. Any Σ-homomorphism is a pre-logical relation but is not in general a logical relation.

The binary case of pre-logical relations over $\mathcal{A}$ and $\mathcal{B}$ is derived from the unary case of *pre-logical predicates* for the product structure $\mathcal{A} \times \mathcal{B}$. Similarly for n-ary relations for $n > 2$.

Definition 2.8 ([HS02]). *A* pre-logical predicate $\mathcal{P}$ *over a* Σ*-combinatory algebra* $\mathcal{A}$ *is a family of predicates* $\{P^\sigma \subseteq [\![\sigma]\!]^{\mathcal{A}}\}_{\sigma \in Types^{\to}(\Sigma)}$ *such that:*

- *If* $P^{\sigma\to\tau}(f)$ *then* $\forall a \in [\![\sigma]\!]^{\mathcal{A}}.P^\sigma(a) \Rightarrow P^\tau(App_{\mathcal{A}}\ f\ a)$.
- $P^\sigma([\![c]\!]^{\mathcal{A}})$ *for every term constant* $c : \sigma$ *in* Σ.
- $P(S_{\mathcal{A}}^{\rho,\sigma,\tau})$ *and* $R(K_{\mathcal{A}}^{\sigma,\tau})$ *for all* $\rho, \sigma, \tau \in Types^{\to}(\Sigma)$.

Example 2.9. For any signature Σ and combinatory algebra $\mathcal{A}$, the family

$$P^\sigma(v) \Leftrightarrow v \text{ is the value of a closed } \Sigma\text{-term } M : \sigma$$

is a pre-logical predicate over $\mathcal{A}$. (In fact, $\mathcal{P}$ is the *least* such — see Prop. 2.17 below.) Now, consider the signature Σ containing the type constant *nat* and term constants 0 : *nat* and *succ* : *nat* $\to$ *nat* and let $\mathcal{A}$ be the combinatory algebra over $\mathbb{N}$ where 0 and *succ* have their usual interpretations and $[\![\sigma \to \tau]\!]^{\mathcal{A}} = [\![\sigma]\!]^{\mathcal{A}} \to [\![\tau]\!]^{\mathcal{A}}$ for every $\sigma, \tau \in Types^{\to}(\Sigma)$ with $App_{\mathcal{A}}^{\sigma,\tau}\ f\ x = f(x)$. Then $\mathcal{P}$ is not a logical predicate over $\mathcal{A}$: any function $f \in [\![nat \to nat]\!]^{\mathcal{A}}$, including functions that are not lambda definable, takes values in P to values in P and so must itself be in P.

An improved version of Theorem 2.5, without the restriction to first-order signatures, holds if pre-logical relations are used in place of logical relations.

Theorem 2.10 (Representation Independence for Pre-Logical Relations [HS02]). *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* Σ*-combinatory algebras and let* $OBS \subseteq Types^{\to}(\Sigma)$. *Then* $\mathcal{A} \equiv_{OBS} \mathcal{B}$ *iff there exists a pre-logical relation over* $\mathcal{A}$ *and* $\mathcal{B}$ *which is a partial injection on* OBS. □

This suggests the following. (We switch to a notation that makes the set of observable types explicit.)

Definition 2.11 ([HLST00]). *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* Σ*-combinatory algebras and* $OBS \subseteq Types^{\to}(\Sigma)$. *Then* $\mathcal{B}$ *is a* pre-logical refinement *of* $\mathcal{A}$, *written* $\mathcal{A} \overset{OBS}{\rightsquigarrow} \mathcal{B}$, *if there is a pre-logical relation* $\mathcal{R}$ *over* $\mathcal{A}$ *and* $\mathcal{B}$ *such that* R^σ *is a partial injection for each* $\sigma \in OBS$.

We phrase this as a definition, rather than as a proof method for the underlying notion of data refinement, in contrast to logical refinements. As a proof method it is sound and complete, and therefore equivalent to this underlying notion.

Pre-logical relations compose — in fact, for extensional models they are the minimal weakening of logical relations with this property (see [HS02] for details).

Proposition 2.12 ([HS02]). *The composition $\mathcal{S} \circ \mathcal{R}$ of pre-logical relations $\mathcal{R}$ over $\mathcal{A}, \mathcal{B}$ and $\mathcal{S}$ over $\mathcal{B}, \mathcal{C}$ is a pre-logical relation over $\mathcal{A}, \mathcal{C}$.* □

So pre-logical refinements compose, and this explains why stepwise refinement is sound. Another explanation goes via Theorem 2.10: $\mathcal{A} \overset{OBS}{\rightsquigarrow} \mathcal{B} \overset{OBS}{\rightsquigarrow} \mathcal{C} \Rightarrow \mathcal{A} \equiv_{OBS} \mathcal{B} \equiv_{OBS} \mathcal{C} \Rightarrow \mathcal{A} \equiv_{OBS} \mathcal{C} \Rightarrow \mathcal{A} \overset{OBS}{\rightsquigarrow} \mathcal{C}$. The set of observable types need not be the same in both steps, as the following result spells out.

Proposition 2.13. *If $\mathcal{A} \overset{OBS}{\rightsquigarrow} \mathcal{B} \overset{OBS'}{\rightsquigarrow} \mathcal{C}$ and $OBS \subseteq OBS'$ then $\mathcal{A} \overset{OBS}{\rightsquigarrow} \mathcal{C}$.* □

The key to many of the applications of logical relations, including Theorem 2.5, is the Basic Lemma, which says that any logical relation over $\mathcal{A}$ and $\mathcal{B}$ relates the interpretation of each lambda term in $\mathcal{A}$ to its interpretation in $\mathcal{B}$.

Lemma 2.14 (Basic Lemma for Logical Relations). *Let $\mathcal{R}$ be a logical relation over $\mathcal{A}$ and $\mathcal{B}$. Then for all Γ-environments $\eta_{\mathcal{A}}, \eta_{\mathcal{B}}$ such that $R^{\Gamma}(\eta_{\mathcal{A}}, \eta_{\mathcal{B}})$ and every term $\Gamma \rhd M : \sigma$, $R^{\sigma}([\![\Gamma \rhd M : \sigma]\!]^{\mathcal{A}}_{\eta_{\mathcal{A}}}, [\![\Gamma \rhd M : \sigma]\!]^{\mathcal{B}}_{\eta_{\mathcal{B}}})$.* □

(Here, $R^{\Gamma}(\eta_{\mathcal{A}}, \eta_{\mathcal{B}})$ refers to the obvious extension of $\mathcal{R}$ to environments.) For pre-logical relations, we get a two-way implication. This says that pre-logical relations are the most liberal weakening of logical relations that give the Basic Lemma. (The reverse implication fails for logical relations.)

Lemma 2.15 (Basic Lemma for Pre-Logical Relations [HS02]). *Let $\mathcal{R} = \{R^{\sigma} \subseteq [\![\sigma]\!]^{\mathcal{A}} \times [\![\sigma]\!]^{\mathcal{B}}\}_{\sigma \in Types^{\rightarrow}(\Sigma)}$ be a family of relations over $\mathcal{A}$ and $\mathcal{B}$. Then $\mathcal{R}$ is a pre-logical relation iff for all Γ-environments $\eta_{\mathcal{A}}, \eta_{\mathcal{B}}$ such that $R^{\Gamma}(\eta_{\mathcal{A}}, \eta_{\mathcal{B}})$ and every Σ-term $\Gamma \rhd M : \sigma$, $R^{\sigma}([\![\Gamma \rhd M : \sigma]\!]^{\mathcal{A}}_{\eta_{\mathcal{A}}}, [\![\Gamma \rhd M : \sigma]\!]^{\mathcal{B}}_{\eta_{\mathcal{B}}})$.* □

Composability of pre-logical relations (Prop. 2.12) is an easy consequence of this.

Pre-logical relations enjoy a number of useful algebraic properties apart from closure under composition. For instance:

Proposition 2.16 ([HS02]). *Pre-logical relations are closed under intersection, product, projection, permutation and $\forall$. Logical relations are closed under product, permutation and $\forall$ but not under intersection or projection.* □

A consequence of closure under intersection is that given a property P of relations that is preserved under intersection, there is always a *least* pre-logical relation satisfying P. We then have the following lambda-definability result (recall Example 2.9 above):

Proposition 2.17 ([HS02]). *The least pre-logical predicate over a given combinatory algebra contains exactly those elements that are the values of closed Σ-terms.* □

In a signature with no term constants, a logical relation may be constructed by defining a relation R on base types and using the definition to *"lift"* R inductively to higher types. The situation is different for pre-logical relations: there are in general many pre-logical liftings of a given R, one being of course

its lifting to a logical relation. But since the property of lifting a given R is preserved under intersection, the least pre-logical lifting of R is also a well-defined relation. Similarly, the least pre-logical *extension* of a given family of relations is well-defined for any signature. Lifting $\mathcal{R}$ to a logical relation is not possible in general for signatures containing higher-order term constants. Extension is also problematic: the cartesian product $\mathcal{A} \times \mathcal{A}$ is a logical relation that trivially extends any binary relation on $\mathcal{A}$, but this is uninteresting.

3 Pre-Logical Relations for System F

The simply-typed lambda calculus $\lambda^{\rightarrow}$ considered in the last section is a very simple language. Extending it with other type constructors, for example sum and product types, is unproblematic, see [HS02]. Much more challenging is the addition of parametric polymorphism as found in functional programming languages, which yields *System F* [Gir71,Rey74]. A hint of the power this adds, apart from the obvious ability to define functions that work uniformly over a family of types, is the fact that it is possible to encode inductive types, including the natural numbers, booleans, lists and products, in pure System F [BB85].

Our interest is in data refinement over System F viewed as a programming language, as a means of applying the ideas in the previous section to languages like Standard ML. The key concept underlying data refinement, as we have seen, is that of observational equivalence, and thus understanding the notion of observational equivalence between models of System F is the main theme. Towards this goal, we extend the semantic approach described in Sect. 2 to System F. This involves extending pre-logical relations to System F, and characterising observational equivalence by pre-logical relations. Leiß [Lei01] has developed a formulation of pre-logical relations and studied their properties in F_ω, the extension of System F by type constructors. In the context of this paper we restrict attention to plain System F, even though the further extension to F_ω presents no additional difficulties.

Definition 3.1. *The set of* types *for System F over a set B of base types is given by the grammar $\sigma ::= b \mid \alpha \mid \sigma \to \sigma \mid \forall\alpha\ .\ \sigma$, where α ranges over* type variables*. A* signature *Σ consists of a set B of type constants and a collection C of typed* term constants *$c : \sigma$ where σ is closed. $Types_{\to\forall}(\Sigma)$ denotes the set of types over B.*

*The set of Σ-*terms *for System F is given by the grammar $M ::= x \mid c \mid \lambda x : \sigma\ .\ M \mid MM \mid \Lambda\alpha\ .\ M \mid M\sigma$.*

For simplicity, we obey *Barendregt's variable convention*: bound variables are chosen to differ in name from free variables in any type or term. A *Σ-type context* (ranged over by Δ) is a list of distinct type variables. A *Σ-context* (ranged over by Γ) is a list of pairs of variables and types in $Types_{\to\forall}(\Sigma)$, where variables are distinct from each other. We often omit Σ if it is clear from the context. For the type system and representation of data types, see e.g. [GTL90]. By $\Delta \rhd \tau, \Delta \rhd \Gamma$

and $\Delta \mid \Gamma \rhd M : \tau$ we mean to declare a type, a context and a term which are well-formed.

First we introduce the underlying model theory of System F in the style of Bruce, Mitchell and Meyer [BMM90].

Definition 3.2. *A* Σ-BMM interpretation *(abbreviated* BMMI*)* $\mathcal{A}$ *consists of*

- *a set* $T_{\mathcal{A}}$ *and a family* $[T_{\mathcal{A}}^n \to T_{\mathcal{A}}] \subseteq T_{\mathcal{A}}^n \to T_{\mathcal{A}}$ *for each* $n \in \mathbf{N}$ *satisfying certain conditions,*[4]
- *an element* $[\![b]\!]^{\mathcal{A}} \in T_{\mathcal{A}}$ *for each* $b \in B$ *of* Σ*,*
- *functions* $\Rightarrow_{\mathcal{A}} : T_{\mathcal{A}} \times T_{\mathcal{A}} \to T_{\mathcal{A}}$ *and* $\forall_{\mathcal{A}} : [T_{\mathcal{A}} \to T_{\mathcal{A}}] \to T_{\mathcal{A}}$*,*
- *a* $T_{\mathcal{A}}$*-indexed family of sets* A*,*
- *a function* $App_{\mathcal{A}}^{t \Rightarrow u} : A^{t \Rightarrow_{\mathcal{A}} u} \to (A^t \to A^u)$ *for each* $t, u \in T_{\mathcal{A}}$*,*
- *a function* $App_{\mathcal{A}}^{\forall f} : A^{\forall_{\mathcal{A}} f} \to \prod_{t \in A} A^{f(t)}$ *for each* $f \in [T_{\mathcal{A}} \to T_{\mathcal{A}}]$*.*

Here we introduce two terminologies. A Δ-environment *is a mapping from type variables in* Δ *to* $T_{\mathcal{A}}$*. We write* $T_{\mathcal{A}}^{\Delta}$ *for the set of* Δ*-environments. For a context* $\Delta \rhd \Gamma$*, a* Γ-environment *is a mapping which maps a variable in* Γ *to an element in* $A^{[\![\Gamma(x)]\!]_\chi^{\mathcal{A}}}$*, where* χ *is a* Δ*-environment. We write* $A^{[\![\Gamma]\!]_\chi^{\mathcal{A}}}$ *for* Γ*-environments. We continue the definition:*

- *a meaning function for types* $[\![-]\!]^{\mathcal{A}}$*, which maps a type* $\Delta \rhd \sigma$ *and* $\chi \in T_{\mathcal{A}}^{\Delta}$ *to* $[\![\sigma]\!]_\chi^{\mathcal{A}} \in T_{\mathcal{A}}$ *(for details, see [Has91]),*
- *an element* $[\![c]\!]^{\mathcal{A}} \in A^{[\![\sigma]\!]^{\mathcal{A}}}$ *for each* $c : \sigma$ *in* Σ*,*
- *a meaning function for terms* $[\![-]\!]^{\mathcal{A}}$ *(we use the same symbol), which maps a term* $\Delta \mid \Gamma \vdash M : \sigma$ *and environments* $\chi \in T_{\mathcal{A}}^{\Delta}, \eta \in R^{[\![\Gamma]\!]_\chi^{\mathcal{A}}}$ *to a value* $[\![M]\!]_{\chi;\eta}^{\mathcal{A}} \in R^{[\![\sigma]\!]_\chi^{\mathcal{A}}}$ *(for details, see [Has91]).*

Given Σ*-BMMIs* $\mathcal{A}$ *and* $\mathcal{B}$*, we can define the* product Σ-BMMI $\mathcal{A} \times \mathcal{B}$ *in the obvious componentwise fashion.*

Definition 3.3. *Let* $\mathcal{A}$ *be a* Σ*-BMMI. A* predicate $\mathcal{R}$ *over* A *(written* $\mathcal{R} \subseteq \mathcal{A}$*) consists of a subset* $T_{\mathcal{R}} \subseteq T_{\mathcal{A}}$ *and* $T_{\mathcal{R}}$*-indexed family of subsets* $R^t \subseteq A^t$*. For* $t, u \in T_{\mathcal{A}}$ *and* $f \in [T_{\mathcal{A}} \to T_{\mathcal{A}}]$ *such that* $f(t) \in T_{\mathcal{R}}$ *for any* $t \in T_{\mathcal{R}}$*, we define*

$$R^t \to R^u = \{x \in A^{t \Rightarrow_{\mathcal{A}} u} \mid \forall y \in R^t \;.\; App_{\mathcal{A}}^{t \Rightarrow u}(x)(y) \in R^u\}$$
$$\forall x \in T_{\mathcal{R}} \;.\; R^f = \{x \in A^{\forall_{\mathcal{A}} f} \mid \forall t \in T_{\mathcal{R}} \;.\; \pi_t(App_{\mathcal{A}}^{\forall f}(x)) \in R^{f(t)}\}$$

Binary relations for Σ*-BMMIs are just predicates over product interpretations.*

Now a predicate $\mathcal{R} \subseteq \mathcal{A}$ *is*

- pre-logical for types *if it satisfies the following:*
  - $[\![b]\!]^{\mathcal{A}} \in T_{\mathcal{R}}$*,*
  - $t, u \in T_{\mathcal{R}}$ *implies* $t \Rightarrow_{\mathcal{A}} u \in T_{\mathcal{R}}$*,*

[4] $[T_{\mathcal{A}}^n \to T_{\mathcal{A}}]$ includes projections and is closed under composition, $\Rightarrow_{\mathcal{A}}$ and $\forall_{\mathcal{A}}$. See [Has91] for a detailed account.

- *for all types* $\Delta, \alpha \rhd \sigma$ *with* $\chi \in T^{\Delta}_{\mathcal{R}}$, *if* $[\![\sigma]\!]^{\mathcal{A}}_{\chi\{\alpha\mapsto t\}} \in T_{\mathcal{R}}$ *holds for all* $t \in T_{\mathcal{R}}$, *then* $[\![\forall\alpha\ .\ \sigma]\!]^{\mathcal{A}}_{\chi} \in T_{\mathcal{R}}$,

– algebraic *if it is pre-logical for types and*
- $[\![c]\!]^{\mathcal{A}} \in R^{[\![\sigma]\!]^{\mathcal{A}}}$ *for all* $c : \sigma$ *in* Σ,
- *for all* $t, u \in T_{\mathcal{R}}$, $R^{t \Rightarrow_{\mathcal{A}} u} \subseteq R^t \to R^u$,
- *for all types* $\Delta, \alpha \rhd \sigma$ *with* $\chi \in T^{\Delta}_{\mathcal{R}}$, $R^{\forall_{\mathcal{A}} f} \subseteq \forall x \in T_{\mathcal{R}}\ .\ R^f$ *holds, where* $f(t) = [\![\sigma]\!]^{\mathcal{A}}_{\chi\{\alpha\mapsto t\}}$,[5]

– pre-logical *if it is algebraic and*
- *for all terms* $\Delta \mid \Gamma, x : \sigma \rhd M : \sigma'$ *with* $\chi \in T^{\Delta}_{\mathcal{R}}$ *and* $\eta \in R^{[\![\Gamma]\!]^{\mathcal{A}}_{\chi}}$, *if* $[\![M]\!]^{\mathcal{A}}_{\chi;\eta\{x\mapsto v\}} \in R^{[\![\sigma']\!]^{\mathcal{A}}_{\chi}}$ *holds for all* $v \in R^{[\![\sigma]\!]^{\mathcal{A}}_{\chi}}$, *then* $[\![\lambda x : \sigma\ .\ M]\!]^{\mathcal{A}}_{\chi;\eta} \in R^{[\![\sigma]\!]^{\mathcal{A}}_{\chi} \Rightarrow_{\mathcal{A}} [\![\sigma']\!]^{\mathcal{A}}_{\chi}}$,
- *for all terms* $\Delta, \alpha \mid \Gamma \rhd M : \sigma$ *with* $\chi \in T^{\Delta}_{\mathcal{R}}$ *and* $\eta \in R^{[\![\Gamma]\!]^{\mathcal{A}}_{\chi}}$, *if* $[\![M]\!]^{\mathcal{A}}_{\chi\{\alpha\mapsto t\};\eta} \in R^{ft}$ *holds for all* $t \in T_{\mathcal{R}}$, *then* $[\![\Lambda\alpha\ .\ M]\!]^{\mathcal{A}}_{\chi;\eta} \in R^{\forall_{\mathcal{A}} f}$, *where* $f(t) = [\![\sigma]\!]^{\mathcal{A}}_{\chi\{\alpha\mapsto t\}}$.

– logical *if it is pre-logical for types and conditions of algebraic relations hold by equality.*

Alternatively, we can extend the definition of pre-logical relations (Definition 2.6) in terms of algebraic relations relating additional combinators for System F [BMM90].

We have the following main theorem of pre-logical predicates:

Theorem 3.4 (Basic Lemma for Pre-Logical Predicates [Lei01]). *Let* $\mathcal{A}$ *be a* Σ*-BMMI and* $\mathcal{R} \subseteq \mathcal{A}$ *be a predicate.*

1. $\mathcal{R}$ *is pre-logical for types iff for all types* $\Delta \rhd \sigma$ *with* $\chi \in T^{\Delta}_{\mathcal{R}}$, $[\![\sigma]\!]^{\mathcal{A}}_{\chi} \in T_{\mathcal{R}}$ *holds.*
2. $\mathcal{R}$ *is pre-logical iff for all terms* $\Delta \mid \Gamma \rhd M : \sigma$ *with* $\chi \in T^{\Delta}_{\mathcal{R}}$ *and* $\eta \in R^{[\![\Gamma]\!]^{\mathcal{A}}_{\chi}}$, $[\![M]\!]^{\mathcal{A}}_{\chi;\eta} \in R^{[\![\sigma]\!]^{\mathcal{A}}_{\chi}}$ *holds.* □

Corollary 3.5 ([Lei01]). *Logical predicates over* Σ*-BMMIs are pre-logical.* □

Proposition 3.6 ([Lei01]). *Let* $\mathcal{A}$ *be a* Σ*-BMMI. We define the* definability predicate $\mathcal{D}$ *by* $T_{\mathcal{D}} = \{[\![\emptyset \rhd \sigma]\!]^{\mathcal{A}}\}$ *and* $D^{[\![\emptyset \rhd \sigma]\!]^{\mathcal{A}}} = \{[\![\emptyset \mid \emptyset \rhd M : \sigma]\!]^{\mathcal{A}}\}$. *Then* $\mathcal{D}$ *is the least pre-logical predicate over* $\mathcal{A}$. □

It is easy to see that pre-logical predicates for System F are closed under product, permutation and arbitrary intersection. On the other hand, they are not closed under composition (nor under projection). This is pointed out by Leiß in the setting of $F\omega$ [Lei01]. The composition $\mathcal{R} \circ \mathcal{S}$ of two relations $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{B}, \mathcal{S} \subseteq \mathcal{B} \times \mathcal{C}$ is given as follows:

$$T_{\mathcal{R}\circ\mathcal{S}} = T_{\mathcal{R}} \circ T_S, \quad R \circ S^{t,u} = \{R^{t,r} \circ S^{r,u} \mid \exists r.(t,r) \in T_{\mathcal{R}}, (r,s) \in T_S\}$$

[5] At this point we know that $f(t) \in T_{\mathcal{R}}$ for any $t \in T_{\mathcal{R}}$ by the first part of Theorem 3.4. Thus $\forall x \in T_R\ .\ R^f$ is defined.

Proposition 3.7. *Binary pre-logical relations between Σ-BMMIs do not compose in general.*

Proof. Let $\Sigma_c = (\{b\}, \{c : b\})$ be a signature and $\mathcal{A}$ be a Σ_c-BMMI where $[\![b]\!]^{\mathcal{A}}$ contains at least two elements, namely $\{\top, \bot\}$. Any non-trivial BMMI for the empty signature can be used for this purpose. We interpret the constant by $[\![c]\!]^{\mathcal{A}} = \top$. We use $\lambda x\ .\ \bot$ as shorthand for $[\![\lambda x : b\ .\ y]\!]^{\mathcal{A}}_{\emptyset;\{y \mapsto \bot\}} \in A^{[\![b]\!]^{\mathcal{A}} \Rightarrow_{\mathcal{A}} [\![b]\!]^{\mathcal{A}}}$.

Let $\theta = [b/\alpha]$ and $\theta' = [b \Rightarrow b/\alpha]$ be type substitutions. We define relation $T_{\mathcal{R}}$ by $T_{\mathcal{R}} = \{([\![\sigma\theta]\!]^{\mathcal{A}}, [\![\sigma\theta']\!]^{\mathcal{B}}) \mid \alpha \vdash \sigma\}$. It is easy to show that this is pre-logical for types. Next we define $R^{[\![\sigma\theta]\!]^{\mathcal{A}}_{\chi}, [\![\sigma\theta']\!]^{\mathcal{B}}_{\chi}}$ for all $\Delta, \alpha \rhd \sigma$ and $\chi \in T^{\Delta}_{\mathcal{R}}$ by induction:

$$
\begin{aligned}
&R^{[\![b\theta]\!]^{\mathcal{A}}_{\chi}, [\![b\theta']\!]^{\mathcal{B}}_{\chi}} = \{(\top, \top)\} \\
&R^{[\![\alpha\theta]\!]^{\mathcal{A}}_{\chi}, [\![\alpha\theta']\!]^{\mathcal{B}}_{\chi}} = \{(\bot, \lambda x\ .\ \bot)\} \\
&R^{[\![\beta\theta]\!]^{\mathcal{A}}_{\chi}, [\![\beta\theta']\!]^{\mathcal{B}}_{\chi}} = R^{\chi(\beta)} \\
&R^{[\![(\sigma \Rightarrow \sigma')\theta]\!]^{\mathcal{A}}_{\chi}, [\![(\sigma \Rightarrow \sigma')\theta']\!]^{\mathcal{B}}_{\chi}} = R^{[\![\sigma\theta]\!]^{\mathcal{A}}_{\chi}, [\![\sigma\theta']\!]^{\mathcal{B}}_{\chi}} \to R^{[\![\sigma'\theta]\!]^{\mathcal{A}}_{\chi}, [\![\sigma'\theta']\!]^{\mathcal{B}}_{\chi}} \\
&R^{[\![(\forall\beta.\sigma)\theta]\!]^{\mathcal{A}}_{\chi}, [\![(\forall\beta.\sigma)\theta']\!]^{\mathcal{B}}_{\chi}} = \forall x \in T_{\mathcal{R}}\ .\ R^{f} \quad (f(t,u) = ([\![\sigma\theta]\!]^{\mathcal{A}}_{\chi\{\beta \mapsto t\}}, [\![\sigma\theta']\!]^{\mathcal{B}}_{\chi\{\beta \mapsto u\}}))
\end{aligned}
$$

We can show that $\mathcal{R} = (T_{\mathcal{R}}, R)$ is pre-logical. However the relation $\mathcal{R}^{-1} \circ \mathcal{R}$ relates $(\lambda x\ .\ \bot, \lambda x\ .\ \bot)$ but not $(\bot, \bot)$. This contradicts algebraicity. □

One natural question is when the composition of two pre-logical relations is pre-logical, and Leiß showed a sufficient condition [Lei01].

We give a characterisation of observational equivalence by pre-logical relations, as in Theorem 2.10. We can reuse Definition 2.3 for observational equivalence between Σ-BMMIs with respect to *closed* observable types.

Theorem 3.8 ([Lei01]). *Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-BMMIs and let OBS be a set of closed types of System F. Then $\mathcal{A} \equiv_{OBS} \mathcal{B}$ iff there exists a pre-logical relation over $\mathcal{A}$ and $\mathcal{B}$ which is a partial injection on OBS.*

4 Expressing Simulation Relations Syntactically

Our journey now moves into the syntactic realm, placing the concepts of simulation relation and representation independence in a logical setting. The main incentive is that computer-aided reasoning requires syntactic expressibility.

One reasonable choice for syntactic formalism is the polymorphic lambda calculus, together with a second-order logic, with the lambda calculus being the object programming language. The decision in this section to make use of polymorphism is motivated primarily by expressibility, since semantic notions may then be internalised in syntax. At the outset, we use this expressive power to express the simply-typed notions of Sect. 2. However, in Sect. 4.4, polymorphism in data types is also handled. Nevertheless, in Sect. 4.6 we suggest that the appropriate setting for describing polymorphic data types is actually F_3.

Our choice here of formalism influences the way we regard the failure of standard simulation relations, *i.e.,* logical relations, at higher order. It turns

out that a natural trail of development gives a solution that is conceptually different from that of pre-logical relations, although it should be evident that the concepts are strongly related. The syntactic approach here is developed directly from syntactic abstraction barriers inherent in polymorphic types. This gives a notion of *abstraction barrier-observing* (*abo*) simulation relation.

4.1 Internalisation into Syntax

To start, we consider System F, cf. Sect. 3. Here we wish to be formalistic, so we use pure System F without constants. Self-iterating inductive types can be encoded [BB85], *e.g.,* $nat \stackrel{def}{=} \forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha$, $bool \stackrel{def}{=} \forall\alpha.\alpha \to \alpha \to \alpha$, and $list_\sigma \stackrel{def}{=} \forall\alpha.\alpha \to (\sigma \to \alpha \to \alpha) \to \alpha$, with programmable constructors, destructors and conditionals. Products encode as $\sigma \times \tau \stackrel{def}{=} \forall\alpha.(\sigma \to \tau \to \alpha) \to \alpha$ with constructor $pair_{\sigma,\tau}$ and destructors $proj_{1\sigma,\tau}$ and $proj_{2\sigma,\tau}$.

We use the logic for parametric polymorphism due to [PA93], a second-order logic over System F augmented with relation symbols, relation definition, and the axiomatic assertion of relational parametricity. See also [Mai91,Tak98]. Formulae now include relational statements as basic predicates and quantifiables,

$$\phi ::= (M =_\sigma N) \mid M\ \xi\ N \mid \cdots \mid \forall\xi \subset \sigma\times\tau\ .\ \phi \mid \exists\xi \subset \sigma\times\tau\ .\ \phi$$

where ξ ranges over relation variables. Relation definition is given by the syntax

$$\Gamma \rhd (x{:}\sigma, y{:}\tau)\ .\ \phi \quad \subset \sigma\times\tau$$

where ϕ is a formula. For example $eq_\sigma \stackrel{def}{=} (x{:}\sigma, y{:}\sigma)\ .\ (x =_\sigma y)$.

We write $U[X]$ to indicate possible occurrences of variable X in type, term or formula U, and write $U[A]$ for the capture-correct substitution $U[A/X]$.

Complex relations may be built from simpler ones. We get the *arrow-type relation* $R{\to}R' \subset (\sigma{\to}\sigma')\times(\tau{\to}\tau')$ from $R \subset \sigma\times\tau$ and $R' \subset \sigma'\times\tau'$ by

$$(R{\to}R') \stackrel{def}{=} (f{:}\sigma{\to}\sigma', g{:}\tau{\to}\tau')\ .\ (\forall x{:}\sigma.\forall y{:}\tau\ .\ (x\ R\ y\ \Rightarrow\ (fx)\ R'\ (gy)))$$

The *universal-type relation* $\forall(\alpha,\beta,\xi \subset \alpha\times\beta)R[\xi] \subset (\forall\alpha.\sigma[\alpha])\times(\forall\beta.\tau[\beta])$ is defined from $R[\xi] \subset \sigma[\alpha]\times\tau[\beta]$, where α, β and $\xi \subset \alpha\times\beta$ are free, by

$$\forall(\alpha,\beta,\xi \subset \alpha\times\beta)R[\xi] \stackrel{def}{=} (y{:}\forall\alpha.\sigma[\alpha], z{:}\forall\beta.\tau[\beta])\ .\ (\forall\alpha.\forall\beta.\forall\xi\ .\ (y\alpha)R[\xi](z\beta))$$

For n-ary $\boldsymbol{\alpha}$, $\boldsymbol{\sigma}$, $\boldsymbol{\tau}$, $\boldsymbol{R}$, where $R_i \subset \sigma_i\times\tau_i$, we get $\rho[\boldsymbol{R}] \subset \rho[\boldsymbol{\sigma}]\times\rho[\boldsymbol{\tau}]$, the *action of type* $\rho[\boldsymbol{\alpha}]$ *on* $\boldsymbol{R}$, by substituting relations for type variables:

$$\begin{array}{ll}
\rho[\boldsymbol{\alpha}] = \alpha_i : & \rho[\boldsymbol{R}] = R_i \\
\rho[\boldsymbol{\alpha}] = \rho'[\boldsymbol{\alpha}]{\to}\rho''[\boldsymbol{\alpha}] : & \rho[\boldsymbol{R}] = \rho'[\boldsymbol{R}]{\to}\rho''[\boldsymbol{R}] \\
\rho[\boldsymbol{\alpha}] = \forall\alpha'.\rho'[\boldsymbol{\alpha},\alpha'] : & \rho[\boldsymbol{R}] = \forall(\beta,\gamma,\xi \subset \beta\times\gamma)\rho'[\boldsymbol{R},\xi]
\end{array}$$

Here, $\boldsymbol{R}$ may be seen as *base relations* from which one uniquely defines relations according to type construction. This is *logical lifting* and gives the mechanism for logical relations in our syntactic setting.

The proof system is intuitionistic natural deduction, augmented with inference rules for relation symbols in the obvious way. There are standard axioms for equational reasoning implying extensionality for arrow and universal types.

Parametric polymorphism requires all instances of a polymorphic functional to exhibit a uniform behaviour [Str67,BFSS90,Rey83]. We adopt *relational parametricity* [Rey83,MR91]: A polymorphic functional instantiated at two related domains should give related instances. This is asserted by the schema

$$\textsc{Param}: \ \forall \boldsymbol{\gamma}.\forall f{:}(\forall\alpha.\sigma[\alpha,\boldsymbol{\gamma}]) \ . \ f \ (\forall\alpha.\sigma[\alpha,\boldsymbol{eq}_{\boldsymbol{\gamma}}]) \ f$$

The logic with PARAM is sound; we have, *e.g.,* the parametric *per*-model of [BFSS90] and the syntactic models of [Has91]. In order to prove the existence of a model, one has to show that PARAM holds for all closed f. If one then expands the statement, one obtains a syntactic analogue of the Basic Lemma for logical relations, but here involving universal types.

Lemma 4.1 (Basic Lemma PARAM [PA93]**).** *For all closed* $f : \forall\alpha.\sigma[\alpha]$*, we derive without* PARAM*,* $f \ (\forall\alpha.\sigma[\alpha]) \ f$*.* □

Constructs such as products, sums, initial and final (co-)algebras are encodable in System F. With PARAM, these become provably universal constructions. Relational parametricity also yields the fundamental

Lemma 4.2 (Identity Extension PARAM [PA93]**).** *With* PARAM*, we derive*

$$\forall\boldsymbol{\gamma}.\forall u,v{:}\sigma[\boldsymbol{\gamma}] \ . \ (u \ \sigma[\boldsymbol{eq}_{\boldsymbol{\gamma}}] \ v \ \ \Leftrightarrow \ \ (u =_{\sigma[\boldsymbol{\gamma}]} v))$$ □

For data types, we use the following notation: A data type over a signature T consists of a data representation A and an implementation of a set of operations $\mathfrak{a}{:}T[A]$. Encapsulation is provided in the style of [MP88] by the following encoding of existential (abstract) types and *pack* and *unpack* combinators:

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

$$pack_T(A)(\mathfrak{a}) \stackrel{def}{=} \Lambda\beta.\lambda f{:}\forall\alpha.(T[\alpha]\to\beta).f(A)(\mathfrak{a})$$
$$unpack_T(package)(\tau)(f) \stackrel{def}{=} package(\tau)(f)$$

Operationally, *pack* packages a data representation and an implementation of operations on that data representation to give a data type of the existential type. The resulting package is a polymorphic functional, that given a client computation and its result domain, instantiates the client with the particular elements of the package. The *unpack* combinator is merely the application operator for *pack*. An abstract type for stacks of natural numbers could be

$$\exists\alpha.(\alpha\times(nat\to\alpha\to\alpha)\times(\alpha\to\alpha)\times(\alpha\to nat))$$

A data type of this type is, *e.g.,* $(pack\ list_{nat}\ \mathfrak{l})$, where

$$(proj_1\ \mathfrak{l}) = nil,\ (proj_2\ \mathfrak{l}) = cons,$$
$$(proj_3\ \mathfrak{l}) = \lambda l{:}list_{nat} \ . \ (cond\ list_{nat}\ (isnil\ l)\ nil\ (cdr\ l)),$$
$$(proj_4\ \mathfrak{l}) = \lambda l{:}list_{nat} \ . \ (cond\ nat\ (isnil\ l)\ 0\ (car\ l)).$$

For convenience we use a labelled product notation,

$$\exists\alpha.T_{Stack_{nat}}[\alpha]$$

where $T_{Stack_{nat}}[\alpha] \stackrel{def}{=} (\mathit{empty}:\alpha,\ \mathit{push}:\mathit{nat}\to\alpha\to\alpha,\ \mathit{pop}:\alpha\to\alpha,\ \mathit{top}:\alpha\to\mathit{nat})$. Each $f_i : T_i[\alpha]$ is a *profile* in $T[\alpha]$. The analogy to Sect. 2 is that f_i is a term constant in the signature T, and models are internalised as packages $(\mathit{pack}A\mathfrak{a})$.

Consider now the issue of when two packages are interchangeable in a program. To each refinement stage, a set *OBS* of *observable types* is associated, assumed to contain closed inductive types, such as *bool* or *nat*, and also any parameters. Two data types are interchangeable if their observable properties are the same, *i.e.,* packages should be observationally equivalent if it makes no difference which one is used in computations with observable result types. Thus:

Definition 4.3 (Observational Equivalence). Observational equivalence *of* $(\mathit{pack}A\mathfrak{a})$, $(\mathit{pack}B\mathfrak{b})$ *with respect to OBS is expressed by*

$$\bigwedge_{\iota\in OBS} \forall f{:}\forall\alpha.(T[\alpha]\to\iota).(f\,A\,\mathfrak{a}) =_\iota (f\,B\,\mathfrak{b})$$

For example, an observable computation on natural-number stacks could be $\Lambda\alpha.\lambda\mathfrak{x}{:}T_{Stack_{nat}}[\alpha]\ .\ \mathfrak{x}.\mathit{top}(\mathfrak{x}.\mathit{push}\,n\ \mathfrak{x}.\mathit{empty})$.

Observational equivalence is the conceptual description of interchangeability, and simulation relations is a means for showing observational equivalence. In the logic one uses the action of types on relations to define logical relations. Two data types are related by a simulation relation if there exists a relation on their data representations that is preserved by their corresponding operations.

Definition 4.4 (Simulation Relation). *The existence of a simulation relation between* $(\mathit{pack}A\mathfrak{a})$ *and* $(\mathit{pack}B\mathfrak{b})$ *is expressed by* $\exists\xi\subset A\times B\ .\ \mathfrak{a}(T[\xi, \boldsymbol{eq_\gamma}])\mathfrak{b}$.

We want the two notions to be equivalent. For data types with first-order operations, this equivalence is a fact under relational parametricity. At higher-order this is not the case. Also, the composability of simulation relations fails at higher order, compromising the constructive composition of refinement steps.

Consider the assumption that $T[\alpha]$ has only first-order function profiles:

FDT^T_{OBS}**:** Every profile $T_i[\alpha] = T_{i1}[\alpha]\to\cdots\to T_{n_i}[\alpha]\to T_{c_i}[\alpha]$ of $T[\alpha]$ is first order, and such that $T_{c_i}[\alpha]$ is either α or some $\iota\in OBS$.

Theorem 4.5 (Composability [Han99]**).** *Assuming* FDT^T_{OBS}*, with* PARAM *we get*

$$\forall A,B,C,\ \xi\subset A\times B,\ \zeta\subset B\times C,\ \mathfrak{a}{:}T[A],\mathfrak{b}{:}T[B],\mathfrak{c}{:}T[C].$$
$$\mathfrak{a}\ T[\xi, \boldsymbol{eq_\gamma}]\ \mathfrak{b}\wedge\ \mathfrak{b}\ T[\zeta, \boldsymbol{eq_\gamma}]\ \mathfrak{c}\ \Rightarrow\ \mathfrak{a}\ T[\xi\circ\zeta, \boldsymbol{eq_\gamma}]\ \mathfrak{c}$$ □

Theorem 4.6 (Representation Independence [Han99]**).** *Assuming* FDT^T_{OBS}*, we get with* PARAM*, for* A,B, $\mathfrak{a}{:}T[A],\mathfrak{b}{:}T[B]$ *and OBS,*

$$\exists\xi\subset A\times B\ .\ \mathfrak{a}\ T[\xi, \boldsymbol{eq_\gamma}]\ \mathfrak{b}\quad\Leftrightarrow\quad\bigwedge_{\iota\in OBS}\forall f{:}\forall\alpha.(T[\alpha]\to\iota)\ .\ (f\,A\,\mathfrak{a}) =_\iota (f\,B\,\mathfrak{b})$$

□

For Theorem 4.6, consider how to derive $\Rightarrow$. In Sect. 2, we would use the Basic Lemma in this situation. Here, we apply PARAM; $f\ (\forall\alpha.T[\alpha, \boldsymbol{eq_\gamma}] \to \iota)\ f$. As mentioned before, PARAM for closed f is essentially the Basic Lemma for logical relations. In the semantic setting one can talk about closed terms. This is not immediately possible syntactically, and note that the definition of observational equivalence here says nothing about f being closed. To compensate for this, we use a extended 'Basic Lemma', namely relational parametricity.

For the opposite direction, one must construct a relation ξ. Analogous to the case in the semantic setting, we use definability, but since closedness is intangible for us, we can only exhibit $\xi \stackrel{def}{=} (a{:}A, b{:}B)\ .\ (\mathit{Dfnbl}(a,b))$, where

$$\mathit{Dfnbl} \stackrel{def}{=} (x{:}A, y{:}B)\ .\ (\exists f_\alpha{:}\forall\alpha.T[\alpha]\to\alpha\ .\ f_\alpha A\mathfrak{a} = x \wedge f_\alpha B\mathfrak{b} = y)$$

This works since observational equivalence is defined 'open' as well. For example, if there is a profile $g{:}\alpha\to\alpha$ in T, then we show $\mathfrak{a}.g\ (\mathit{Dfnbl}\to\mathit{Dfnbl})\ \mathfrak{b}.g$ which follows easily by giving $f \stackrel{def}{=} \Lambda\alpha.\lambda\mathfrak{x}:\alpha.\mathfrak{x}.g(f_\alpha\alpha\mathfrak{x})$, where f_α is postulated by the antecedentary *Dfnbl*. If $g:\alpha\to\iota$, then $f:\forall\alpha.T[\alpha]\to\iota$, and by observational equivalence we have $fA\mathfrak{a} =_\iota fB\mathfrak{b}$, which gives $fA\mathfrak{a}\ \iota\ fB\mathfrak{b}$ by PARAM.

This particular proof fails if $T[\alpha]$ has higher-order profiles. Consider

$$T[\alpha] \stackrel{def}{=} (p{:}(\alpha\to\alpha)\to nat, s{:}\alpha\to\alpha)$$

We must derive $\forall x:A\to A, y:B\to B\ .\ x(\mathit{Dfnbl}\to\mathit{Dfnbl})y\ \Rightarrow\ \mathfrak{a}.px =_{nat} \mathfrak{b}.py$. However, $x(\mathit{Dfnbl}\to\mathit{Dfnbl})y$ does not give us an $f_{\alpha\to\alpha}{:}\forall\alpha.T[\alpha]\to(\alpha\to\alpha)$ such that $f_{\alpha\to\alpha}A\mathfrak{a} = x \wedge f_{\alpha\to\alpha}B\mathfrak{b} = y$, so we cannot construct our $f{:}\forall\alpha.T[\alpha]\to nat$ to complete the proof.

This negative result involving *Dfnbl* generalises. At higher order, there might not exist any simulation relation in the presence of observational equivalence [Han01]. To exemplify with $T[\alpha]$ above, any candidate $R\subset A\times B$ has to satisfy $\forall x:A\to A, y:B\to B\ .\ x\ (R\to R)\ y\ \Rightarrow\ \mathfrak{a}.px =_{nat} \mathfrak{b}.py$, and this includes x and y that do not belong to, or are not expressible by, operations in the respective data types. This, one might argue, is unreasonable.

In fact, it is. Consider a computation $f = \Lambda\alpha.\lambda\mathfrak{x}:T[\alpha].M[\alpha,\mathfrak{x}]$. A crucial observation is now embodied in the following obvious statement.

`Abs-Bar`: A computation $\Lambda\alpha.\lambda\mathfrak{x}{:}T[\alpha].M[\alpha,\mathfrak{x}]$ cannot have free variables of types involving the virtual data representation α.

This has a direct bearing on how data type operations may be used. For example, $x{:}A\to A$ and $y{:}B\to B$ above cannot be arbitrary, but must be expressible by respective data type operations; in this case, the only possible candidate for x is $\mathfrak{a}.g$, and $\mathfrak{b}.g$ for y.

4.2 *abo*-Simulation Relations with Special Parametricity

An obvious solution is now to define a notion of simulation relation where arrow-type relations are weakened by definability clauses for arguments [Han00,Han01]. For example, write $\mathfrak{a}.s\ (R\to R)^{\beth}\ \mathfrak{b}.s$, for $\beth \stackrel{def}{=} \langle A, B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle$, meaning

$$\forall x{:}A, y{:}B\ .\ x\ R\ y\ \wedge\ \mathit{Dfnbl}^{\beth}_{\alpha}(x,y)\ \Rightarrow\ \mathfrak{a}.sx\ R\ \mathfrak{b}.sy$$

where $Dfnbl^{\beth}_{\alpha}(x,y) \stackrel{def}{=} (x{:}A, y{:}B)\ .\ (\exists f_\alpha{:}\forall\alpha.T[\alpha]\to\alpha\ .\ f_\alpha A\mathfrak{a} = x \wedge f_\alpha B\mathfrak{b} = y)$. In general, definability clauses are inserted recursively in arrow types, bottoming out at base relations, *i.e.,* $R^{\beth} \stackrel{def}{=} R$. The full definition is in [Han01], and includes the formulation at universal type as well. With a slight abuse of notation,

Definition 4.7 (*abo*-Simulation Relation). *For any A, B and $R \subset A \times B$,*

$$T[R, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth} \stackrel{def}{=} (\mathfrak{a}{:}T[A,\boldsymbol{\gamma}], \mathfrak{b}{:}T[B,\boldsymbol{\gamma}])\ .\ (\wedge_{1\le i\le k}\ \mathfrak{a}.g_i\ (T_i[R, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth})\ \mathfrak{b}.g_i)$$

Observable types such as *nat* are universal types and appear in $\beth$-variants inside $T[\xi]$. Therefore, it is important that $nat^{\beth}$ is eq_{nat}. This holds for closed inductive types using PARAM. However, we do not get desired relational properties at product type, hence the formulation in Definition 4.7.

We are still working under the assumption of relational parametricity. However, notice that we cannot apply PARAM, *e.g.,* when using $T[\xi, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth}$. One can recover the needed proof power by asserting the missing piece of relational parametricity. Write $f\ (\forall\alpha.T[\alpha, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\varepsilon}\to\sigma[\alpha, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\varepsilon})\ f$, meaning

$$\forall A, B, \xi\subset A\times B.\forall\mathfrak{a}{:}T[A,\boldsymbol{\gamma}], \mathfrak{b}{:}T[B,\boldsymbol{\gamma}]\ .\\ \mathfrak{a}(T[\xi, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth})\mathfrak{b}\ \Rightarrow\ (f A\,\mathfrak{a})(\sigma[\xi, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth})(f B\,\mathfrak{b})$$

where $\beth = A, B, \mathfrak{a}, \mathfrak{b}$. We assume the following:

HDT^{T}_{OBS}**:** Every profile $T_i[\alpha] = T_{i1}[\alpha]\to\cdots\to T_{n_i}[\alpha]\to T_{c_i}[\alpha]$ of $T[\alpha]$ is such that $T_{ij}[\alpha]$ has no occurrences of universal types other than those in OBS, and $T_{c_i}[\alpha]$ is either α or some $\iota \in OBS$.

Definition 4.8 (Special *abo*-Parametricity (SPPARAM)**).** *For HDT^{T}_{OBS}, for $\sigma[\alpha,\boldsymbol{\gamma}]$ having no occurrences of universal types other than those in OBS, and whose only free variables are among α and $\boldsymbol{\gamma}$,*

$$\text{SPPARAM:}\ \ \forall f{:}\forall\alpha.(T[\alpha,\boldsymbol{\gamma}]\to\sigma[\alpha,\boldsymbol{\gamma}])\ .\ f\ (\forall\alpha.T[\alpha, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\varepsilon}\to\sigma[\alpha, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\varepsilon})\ f$$

Lemma 4.9 (Basic Lemma SPPARAM [Han01]**).** *For HDT^{T}_{OBS}, for $\sigma[\alpha]$ having no occurrences of universal types other than those in OBS, and for closed $f{:}\forall\alpha.(T[\alpha]\to\sigma[\alpha])$, we derive $f\ (\forall\alpha.T[\alpha]^{\varepsilon}\to\sigma[\alpha]^{\varepsilon})\ f$.* □

Lemma 4.9 entails soundness for the logic with PARAM and SPPARAM with respect to the closed type and term model and the parametric minimal model due to [Has91].

Theorem 4.10 (Composability [Han00]**).** *Assuming HDT^{T}_{OBS}, we get using* SPPARAM*, for $\beth = \langle A, B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle$, $\beth' = \langle B, C\rangle\langle\mathfrak{b},\mathfrak{c}\rangle$, and $\beth'' = \langle A, C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle$,*

$$\forall A, B, C,\ \xi\subset A\times B,\ \zeta\subset B\times C,\ \mathfrak{a}{:}T[A], \mathfrak{b}{:}T[B], \mathfrak{c}{:}T[C].\\ \mathfrak{a}\ T[\xi, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth}\ \mathfrak{b} \wedge\ \mathfrak{b}\ T[\zeta, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth'}\ \mathfrak{c}\ \Rightarrow\ \mathfrak{a}\ T[\zeta\circ\xi, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth''}\ \mathfrak{c}$$ □

Theorem 4.11 (Representation Independence [Han00]**).** *With the assumption HDT^{T}_{OBS}, we get with* SPPARAM*, for A, B, $\mathfrak{a} : T[A], \mathfrak{b} : T[B]$, OBS, and $\beth = \langle A, B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle$,*

$$\exists\xi\subset A\times B\ .\ \mathfrak{a}\ T[\xi, \boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth}\ \mathfrak{b}\quad\Leftrightarrow\quad\bigwedge_{\iota\in OBS}\ \forall f{:}\forall\alpha.(T[\alpha]\to\iota)\ .\ (f A\,\mathfrak{a}) =_{\iota} (f B\,\mathfrak{b})$$

□

4.3 *abo*-Simulation Relations with Closed Special Parametricity

If we can express closedness in the logic, then we can relate to non-syntactic models as well. Closedness is inherently intractable, but we can approximate to a certain degree. We add a basic predicate *Closed* to the syntax together with a pre-defined semantics. The effect of this is, for example, that the interpretation in any model of $\forall f{:}\forall\alpha.(T[\alpha,\boldsymbol{\gamma}]\to\iota)\ .\ \mathit{Closed}_{OBS}(f)\ \Rightarrow\ \phi(f)$ restricts attention in ϕ to those interpretations of all $f{:}\forall\alpha.(T[\alpha]\to\iota)$ that are denotable by terms whose only free variables are of types in OBS.

The semantics for the predicate *Closed* is not stable under term formation, so we cannot make axioms for *Closed* in order to derive the closedness of a term from its subterms. We can however add a second symbol *ClosedS* with a pre-defined semantics that does allow the derivation of closedness, but *ClosedS* will then not satisfy substitutivity. This is resolved by giving a separate non-substitutive calculus for deriving closedness, together with rules for importing the needed results into the main logic. Details are in [Han01]. We now get:

Definition 4.12 (Observational Equivalence by Closed Computation). Observational equivalence by closed computation *of* $(\mathit{pack}A\mathfrak{a})$ *and* $(\mathit{pack}B\mathfrak{b})$ *with respect to OBS is expressed as*

$$\bigwedge_{\iota\in OBS}\ \forall f{:}\forall\alpha.(T[\alpha]\to\iota)\ .\ \mathit{Closed}_{OBS}(f)\ \Rightarrow\ (f\,A\,\mathfrak{a})=_{\iota}(f\,B\,\mathfrak{b})$$

Also we write for example, $\mathfrak{a}.s\ (R\to R)^{\beth}_{C}\ \mathfrak{b}.s$, for $\beth\stackrel{def}{=}\langle A,B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle$, meaning

$$\forall x{:}A,y{:}B\ .\ x\ R\ y\ \wedge\ \mathit{Dfnbl}\,C^{\beth}_{\alpha}(x,y)\ \Rightarrow\ \mathfrak{a}.sx\ R\ \mathfrak{b}.sy$$

where $\mathit{Dfnbl}\,C^{\beth}_{\alpha}(x,y)\stackrel{def}{=}(x{:}A,y{:}B)\ .$
$(\exists f_{\alpha}{:}\forall\alpha.T[\alpha]\to\alpha\ .\ \mathit{Closed}_{OBS}(f)\ \wedge\ f_{\alpha}A\mathfrak{a}=x\wedge f_{\alpha}B\mathfrak{b}=y)$

Again, with a slight abuse of notation:

Definition 4.13 (*abo*-Simulation Relation by Closed Computation). *For any A, B and $R\subset A\times B$,*

$$T[R,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth}_{C}\stackrel{def}{=}(\mathfrak{a}{:}T[A,\boldsymbol{\gamma}],\mathfrak{b}{:}T[B,\boldsymbol{\gamma}])\ .\ (\wedge_{1\le i\le k}\ \mathfrak{a}.g_i\ (T_i[R,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\beth}_{C})\ \mathfrak{b}.g_i)$$

Definition 4.14 (Special Closed *abo*-Parametricity (spParamC)**).** *For HDT^{T}_{OBS}, for $\sigma[\alpha,\boldsymbol{\gamma}]$ having no occurrences of universal types other than those in OBS, and whose only free variables are among α and $\boldsymbol{\gamma}$,*

$$\text{spParamC}:\ \forall f{:}\forall\alpha.(T[\alpha,\boldsymbol{\gamma}]\to\sigma[\alpha,\boldsymbol{\gamma}])\ .\ \mathit{Closed}_{OBS}(f)\ \Rightarrow\ f\ (\forall\alpha.T[\alpha,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\varepsilon}_{C}\to\sigma[\alpha,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\varepsilon}_{C})\ f$$

Using this, we get analogous results to the previous section. The corresponding Basic Lemma entails the soundness of the logic with Param and spParamC with respect to any relational parametric model.

4.4 *abo*-Relational Parametricity

The previous two subsections augmented relational parametricity with special instances of what one could call *abo*-relational parametricity. Now we replace relational parametricity altogether with full-fledged *abo*-relational parametricity. This gives a much simpler treatment than the previous approaches, but at a price: we now need infinite conjunctions in the logic. These are however well-behaved in the sense that proofs only need pointwise treatment. Moreover, refinement proofs need not be concerned with infinite conjunctions.

`Abs-Bar` says that function arguments in computations are bounded by formal parameters, *e.g.,* in the computation $f \stackrel{def}{=} \Lambda\alpha.\lambda x\!:\!\alpha.\lambda s\!:\!\alpha\to\alpha.t[x,s]$, s will only be applied to arguments built from formal parameters x and s. This transfers to instances $f\sigma$ and $f\tau$. So even at the basic level of universal types, one could for $R\subset\sigma\times\tau$ say that *e.g.,* $s_\sigma\ (R\to R)\ s_\tau$ should reflect this, in that only those x,y are considered for the antecedent $x\ R\ y$ that are admissible in the computations. Thence, $f\ (\forall\alpha.\alpha\to(\alpha\to\alpha)\to\alpha)^{\mathsf{abo}}\ g$ is the relation given by

$$\forall\gamma,\delta,\xi\subset\gamma\times\delta\ .\ \forall a\!:\!\gamma,b\!:\!\delta,s\!:\!\gamma\to\gamma,s'\!:\!\delta\to\delta\ .\\ a\ \xi\ b\ \Rightarrow\ s\ (\xi\to\xi)^{\urcorner}\ s'\ \Rightarrow\ f\gamma as\ \xi\ g\delta bs'$$

where $s\ (\xi\to\xi)^{\urcorner}\ s'$ for $\urcorner=\langle\gamma,\delta\rangle\langle a,b\rangle\langle s,s'\rangle$ is

$$\forall x\!:\!\gamma,y\!:\!\delta\ .\ x\ \xi\ y\wedge \mathit{Dfnbl}^{\urcorner}_{\gamma}(x,y)\ \Rightarrow\ sx\ \xi\ s'y$$

where $\mathit{Dfnbl}^{\urcorner}_{\gamma}(x,y)\stackrel{def}{=}(x\!:\!\gamma,y\!:\!\delta)\ .\ (\exists f_\alpha\!:\!\forall\alpha.T[\alpha]\to\alpha\ .\ f_\alpha\gamma as=x\wedge f_\alpha\gamma bs'=y)$. In general, *Dfnbl* clauses are inserted recursively in arrow types, bottoming out at base relations. The notion of *abo*-relation in [Han03] formalises the idea. Universal types play two rôles. Consider $(\forall\alpha.\alpha\to(\alpha\to\alpha)\to(\forall\beta.(\alpha\to\beta)\to\beta)\to\alpha)^{\mathsf{abo}}$, and a term of this type, *e.g.,* $\Lambda\alpha.\lambda x\!:\!\alpha,s\!:\!\alpha\to\alpha,p\!:\!\forall\beta.(\alpha\to\beta)\to\beta\ .\ s(p\alpha s)$. The *abo*-relation treats the outer universal type as the type of a computation, and sets up the *Dfnbl* clauses according to formal parameters x,s,p. Then, the inner universal type must be treated as a polymorphic parameter, and it is necessary to capture that instances $p\sigma$ may only vary in α. This is where infinite conjunctions enters the scene, but this discernability for universal types is what enable *abo*-simulation relations to handle polymorphism in data types, see below.

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

Definition 4.15 (*abo*-Parametricity).

$$\textit{abo}\text{-}\textsc{Param}:\ \forall\boldsymbol{\gamma}.\forall f\!:\!(\forall\alpha.\sigma[\alpha,\boldsymbol{\gamma}])\ .\ f\ (\forall\alpha.\sigma[\alpha,\boldsymbol{eq}_{\boldsymbol{\gamma}}])^{\mathsf{abo}}\ f$$

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

Definition 4.16 (*abo*-Identity Extension for Universal Types).

$$\textit{abo}\text{-IEL}: \ \forall\boldsymbol{\gamma}.\forall u,v{:}(\forall\alpha.\sigma[\alpha,\boldsymbol{\gamma}]) \ . \ u \ (\forall\alpha.\sigma[\alpha,\boldsymbol{eq}_{\boldsymbol{\gamma}}])^{\mathsf{abo}} \ v \ \Leftrightarrow \ u = v$$

Both *abo*-PARAM and *abo*-IEL hold in the *abo*-parametric *per*-model [Han03]. We can also formulate a basic lemma for *abo*-PARAM, if we allow infinite derivations. Regular parametricity, PARAM, will not hold in this model; in fact any logic containing both PARAM and *abo*-PARAM is inconsistent. Note that *abo*-IEL implies *abo*-PARAM. Nevertheless, we choose to display both. With *abo*-PARAM and *abo*-IEL, we regain universal properties, for example for products:

$$\forall\sigma,\tau.\forall z{:}\sigma\times\tau \ . \ \mathit{pair}(\mathit{proj}_1 z)(\mathit{proj}_2 z) = z$$

$$\begin{aligned}\forall u,v{:}\sigma\times\tau \ . \ u \ (\sigma[\boldsymbol{eq}_{\boldsymbol{\gamma}}]\times\tau[\boldsymbol{eq}_{\boldsymbol{\gamma}}])^{\mathsf{abo}} \ v \\ \Leftrightarrow \ (\mathit{proj}_1 u) \ \sigma[\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\mathsf{abo}} \ (\mathit{proj}_1 v) \ \wedge \ (\mathit{proj}_2 u) \ \tau[\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\mathsf{abo}} \ (\mathit{proj}_2 v)\end{aligned}$$

Theorem 4.17 (Representation Independence [Han03]**).** *Under the assumption* $\mathtt{HDT}^T_{OBS}$*, we get with abo-*PARAM *and abo-*IEL*,*

$$\begin{aligned}&\forall A,B.\forall\mathfrak{a}{:}T[A],\mathfrak{b}{:}T[B] \ . \\ &\quad \exists\xi\subset A\times B \ . \ \mathfrak{a} \ T[\xi,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\langle A,B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle} \ \mathfrak{b} \\ &\qquad\qquad \Leftrightarrow \ \textstyle\bigwedge_{\iota\in OBS}\forall f{:}\forall\alpha.(T[\alpha]\to\iota) \ . \ (f A\,\mathfrak{a}) =_\iota (f B\,\mathfrak{b})\end{aligned} \qquad \square$$

Theorem 4.18 (Composability [Han03]**).** *Assuming* $\mathtt{HDT}^T_{OBS}$*, we get with abo-*PARAM *and abo-*IEL*,*

$$\begin{aligned}&\forall A,B,C, \ \xi\subset A\times B, \ \zeta\subset B\times C, \ \mathfrak{a}{:}T[A],\mathfrak{b}{:}T[B],\mathfrak{c}{:}T[C]. \\ &\qquad \mathfrak{a}(T[\xi,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\langle A,B\rangle\langle\mathfrak{a},\mathfrak{b}\rangle})\mathfrak{b} \ \wedge \ \mathfrak{b}(T[\zeta,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\langle B,C\rangle\langle\mathfrak{b},\mathfrak{c}\rangle})\mathfrak{c} \\ &\qquad\qquad \Rightarrow \ \mathfrak{a}(T[\zeta\circ\xi,\boldsymbol{eq}_{\boldsymbol{\gamma}}]^{\langle A,C\rangle\langle\mathfrak{a},\mathfrak{c}\rangle})\mathfrak{c}\end{aligned} \qquad \square$$

If we allow infinite derivations, we get representation independence and composability for data types with polymorphic operations, under one requirement: In the sense of Sects. 2 and 3, all type constants must either be observable or hidden, if they are the result type of any operation. Here, type constants correspond to closed types, and since we have polymorphism, type instantiation requires added caution. For example, if $OBS = \{\mathit{bool}\}$, then $T[\alpha]$ may not have a profile $g : \alpha \to (\forall\beta.\beta\to\beta)$, since $\forall\beta.\beta\to\beta$ is not observable, nor any profile $g : \alpha \to (\forall\beta.\alpha\to\beta)$, since gx can then be instantiated by a non-observable closed type yielding a derived profile $gx(\forall\beta.\beta\to\beta) : \alpha\to(\forall\beta.\beta\to\beta)$. Thus, the requirement takes the form

$\mathtt{DT}^T_{OBS}$**:** Every profile $T_i[\alpha] = T_{i1}[\alpha]\to\cdots\to T_{ni}[\alpha]\to T_{ci}[\alpha]$ of $T[\alpha]$ is such that if $T_{ci}[\alpha]$ has a deepest rightmost universal type $\forall\beta.V$, then this subtype is not closed, nor is the deepest rightmost subtype of $\forall\beta.V$ the quantified β.

Then, Theorem 4.17 and Theorem 4.18 hold under $\mathtt{DT}^T_{OBS}$.

In closing, we mention that for this section, $\mathtt{HDT}^T_{OBS}$ can in any case be relaxed by dropping the restriction on T_{ij}.

4.5 *pl*-Relational Parametricity

It is possible to define algebraic relations in the logic. We do this from basic principles, just as we do for *abo*-relations. Consider again for example the universal type $\forall\alpha.\alpha\to(\alpha\to\alpha)\to\alpha$. In a sense, universal types determine signatures with function profiles. This inductive type has a profile for 'zero', and a profile for 'successor'. Relative to the 'signature' consisting of these profiles, one can then define algebraic relations in a finite way. Here this can be done for any σ and τ, by giving a relation $R_\alpha \subset \sigma\times\tau$, taking the rôle of a base type relation, and then giving a relation $R_{\alpha\to\alpha} \subset (\sigma\to\sigma)\times(\tau\to\tau)$ that we insist satisfies algebraicity: $R_{\alpha\to\alpha}(s,s') \;\Rightarrow\; s\ (R_\alpha\to R_\alpha)\ s'$. In this manner, the universal type induces a family of relations, namely R_α and $R_{\alpha\to\alpha}$, over the 'signature' of the universal type. Thus, we write *e.g.,* $f\ (\forall\alpha.\alpha\to(\alpha\to\alpha)\to\alpha)^{pl}\ g$ for the relation

$$\begin{array}{l}\forall\gamma,\delta,\xi_\alpha\subset\gamma\times\delta,\xi_{\alpha\to\alpha}\subset\gamma\to\gamma\times\delta\to\delta\ .\ pl_{\alpha\to\alpha}(\xi_{\alpha\to\alpha};\xi_\alpha)\ \Rightarrow\\ \quad\forall a{:}\gamma,b{:}\delta,s{:}\gamma\to\gamma,s'{:}\delta\to\delta\ .\ R_\alpha(a,b)\wedge R_{\alpha\to\alpha}(s,s')\ \Rightarrow\ R_\alpha(f\gamma as,g\delta bs')\end{array}$$

where $pl_{\alpha\to\alpha}(\xi_{\alpha\to\alpha};\xi_\alpha)$ asserts algebraicity of $\xi_{\alpha\to\alpha}$ relative to ξ_α. In general, one completes the finite family of relations with all so-called free subtypes, in order to ensure well-definedness of the algebraicity conditions. Also, the full definition of algebraic relations in this manner must reflect the two levels of polymorphism mentioned in the previous section.

To get pre-logical relations (*pl*-relations), one must additionally ensure closure over abstraction. This spoils finiteness, since for a combinatorial approach, we must assert relatedness of an infinite set of combinators. Again this infinite conjunction is well-behaved, since it ranges over all types only varying over the data representations.

We may soundly assert *pl*-relational parametricity and using this, we get similar results to those in the previous section.

It might be possible to get away with a finite number of combinators. The rationale behind this is that one may proceed with only a finite family of algebraic relations. If relations of higher order than those in the family are needed, then these can be constructed by logical lifting. This is relevant for polymorphic instantiation. Based on this, it may suffice to have an upper bound on the type complexity of combinators needed. This is under investigation.

4.6 Polymorphic Data Types in F_3

Polymorphism within data types is dealt with in a somewhat general manner in the two previous sections. However, it is hard to find natural examples of data types with polymorphic operations that are expressible in System F. Instead, F_3 is appropriate, and then one could give *e.g.,* polymorphic stacks as follows.

$$\begin{array}{l}\exists\mathfrak{X}{:}*\to*.T_{polyStack}[\mathfrak{X}],\\ T_{polyStack}[\mathfrak{X}] = (\mathit{empty}{:}\forall\gamma\,.\,\mathfrak{X}\gamma,\ \mathit{push}{:}\forall\gamma\,.\,\gamma\to\mathfrak{X}\gamma\to\mathfrak{X}\gamma,\\ \qquad\qquad \mathit{pop}{:}\forall\gamma\,.\,\mathfrak{X}\gamma\to\mathfrak{X}\gamma,\ \mathit{top}{:}\forall\gamma\,.\,\mathfrak{X}\gamma\to\gamma\to\gamma,\\ \qquad\qquad \mathit{map}{:}\forall\gamma,\gamma'\,.\,(\gamma\to\gamma')\to\mathfrak{X}\gamma\to\mathfrak{X}\gamma'\ )\end{array}$$

This provides polymorphic stack operations. The data representation is a type constructor $\mathfrak{X}$ to be instantiated by the relevant stack element type.

One can treat this kind of shallow polymorphism in a pointwise fashion in F_2 [Han01], so that one essentially reduces the problem to non-polymorphic signatures. Then it is not necessary that F_2 technology deals with polymorphic signatures, neither in one way or another. Alternatively, one could devise appropriate notions of relational parametricity for F_3.

5 Reconciliation

In comparison with the neat and tidy story of pre-logical relations in the simply-typed lambda calculus told in Sect. 2, both the semantic account of pre-logical relations in System F in Sect. 3 and the syntactic approach of abstraction barrier-observing simulation relations in Sect. 4 exhibit certain shortcomings. Our present feeling is that true enlightenment on this subject will require some bridge between the two. The following subsections suggest some possible lines of enquiry that seem promising to us.

5.1 Internalisation of Semantic Notions into Syntax

Sects. 2 and 3 deal with semantic simulation relations between models for lambda calculi. Sect. 4, on the other hand, internalises models and simulation relations into syntax. Models (data types) then become terms of an existential type of the form $\exists\alpha.T[\alpha] \stackrel{def}{=} \forall\beta.(\forall\beta.T[\alpha] \to \beta) \to \beta$, for some 'signature' $T[\alpha]$, and computations or programs using data types are $f : \forall\alpha.T[\alpha] \to \sigma$. Thus, polymorphism is used to internalise semantic notions. This use of polymorphism is at a level external to data types (models); as in the outermost universal type in computations $f : \forall\alpha.T[\alpha] \to \sigma$, in contrast to polymorphism within models arising from any polymorphic profiles in $T[\alpha]$.

Relationally, this two-leveled aspect gives rise to certain difficulties. For logical relations it suffices to give a uniform relational definition for universal types, but for *abo*-relations which use definability relative to data type signatures, it is necessary to reflect the two levels in the relational definitions. This gives a non-uniform relational treatment at universal type. Note that the semantic approach in Sect. 3 does not work on internalised structures in the syntax, and the notion of relational parametricity is there cleaner.

Internalising models as, *e.g.,* inhabitants of existential type, gives syntactic control, especially in the context of refining abstract specifications to executable programs. However, we think that the benefits of this to mechanised reasoning should be weighed against a possible scenario without internalisation, but perhaps with sound derived proofs rules for data refinement. This would be a more domain-specific calculus, but might provide a simpler formalism, perhaps more in style with semantic reasoning.

5.2 Equivalence of Models Versus Equivalence of Values in a Model

As a consequence of the internalisation of semantic notions discussed above, the term "observational equivalence" has been applied at two different levels to achieve similar aims. In Sects. 2 and 3, observational equivalence is a relation between two models over the same signature, representing programs. In Sect. 4, it is a relation on encapsulated data types *within* a single model; such a relation is sometimes referred to as *indistinguishability*, written $\approx$. In both of the latter two sections, the power of System F would allow the opposite approach to be taken. Then the question of the relationship between the resulting definitions arises. This question has been investigated in a number of simpler frameworks in [BHW95,HS96,Kat03], where the connection is given by a *factorisability* result of the form $\mathcal{A} \equiv \mathcal{B}$ iff $\mathcal{A}/\approx \cong \mathcal{B}/\approx$. It is likely that the same applies in the context of System F, and this might in turn help to shed light on the relationship between the semantic and syntactic worlds.

5.3 Finiteness

Pre-logical relations are defined in terms of definable elements. In logic it is hard to deal with definability in a term-specific way. In Sect. 4.3 we approximated by introducing a new predicate *Closed*, and in Sect. 4.2 we explicitly related to syntactic models. In both Sect. 4.4 and Sect. 4.5, we basically end up with infinitary logic, albeit in a tractable manner. It may also be feasible to combine elements, for example to use the *Closed* predicate together with *pl*-relations.

From a purist point of view, all these approaches are slightly unsatisfactory, although for practical purposes they provide methods for proving refinement, since the infinitary issues are basic and of no concern when doing refinement proofs. In fact this is true *a fortiori* for *abo*-simulation relations, since these are in fact finite, unlike pre-logical relations.

5.4 Pre-Logical Relations and *abo*-Relations

Both pre-logical relations (*pl*-relations in the logic) and *abo*-relations solve the same problems for refinement. The question is then what else they have in common. To make a comparison easier, one can do two things. First, one can transpose the syntactic idea of *abo*-relation into the semantic setting around *e.g.,* combinatory algebras. It is then probably natural to interpret the *Dfnbl* clauses as term-definability. In that case when considering data types, it is evident that *abo*-simulation relations specialise to a finitary version of the minimal pre-logical relation, which is not surprising. The general relationship is however unclear.

Conversely, one might transpose the idea of pre-logical relation into syntax with internalised data types. This is mentioned in Sect. 4.5. Then, the connection is not so clear, since the *Dfnbl* clause says nothing about term-definability, unless we use the *Closed* predicate of Sect. 4.3. Any comparison would probably depend on the model of choice for the logic.

Furthermore, at a more fundamental level, one gets various concepts of relational parametricity. We have the ones in the syntactic setting where data types are internalised, but we also have the external semantic concept in connection to the scenario in Sect. 3. Characterising these in terms of one another is left as an interesting challenge, the start of which is described in the next section.

5.5 Connection with Pre-Logical Relations and Relational Interpretation of System F

We can regard a binary relation over a BMM interpretation as an interpretation of types by relations. The origin of this viewpoint, the *relational interpretation of System F*, goes back to Reynolds [Rey83] in his attempt to obtain a set-theoretic model of System F. This relational viewpoint enables him to capture the nature of polymorphism in terms of relational parametricity.

A semantic account of this viewpoint is given in [MR91,Has91,RR94,BAC95]. Roughly speaking, a relational interpretation of System F consists of two components: a *reflexive graph* (a graph with an *identity edge* at each node), which gives a skeleton of binary relations; and an underlying interpretation of System F together with binary relations over it. The interpretation ties nodes and edges of the reflexive graph to the carrier sets of the underlying interpretation and binary relations. This mapping respects identity, i.e. identity edges are mapped to identity relations.

We can find a correspondence between pre-logical relations and the relational interpretation of System F. Reflexive graphs are a generalisation of reflexive relations. Thus a pair consisting of a BMM interpretation $\mathcal{A}$ and a relation $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ such that $T_{\mathcal{R}}$ is reflexive and $R^{t,t} = id_{A^t}$ form a relational interpretation of System F. Conversely, any relational interpretation whose reflexive graph is just a reflexive relation can be regarded as a relation over its underlying interpretation of System F. Moreover we often assume that the mapping from edges to relations respects the interpretation of types. This situation is called *natural* in [Has91], and under the above correspondence, this means that the corresponding relation is *logical*.

This correspondence suggests that we can bring back our notion of pre-logical relations to consider a class of relational interpretations of System F. We expect that this new class includes interpretations which satisfy Reynolds' *abstraction theorem*. The question is how do we understand the notion of relational parametricity in this new class. Parametricity states that relations at universal types include identity relations, but the identity relation itself is a logical notion (in extensional models). One negative consequence of this mismatch is that the Identity Extension Lemma does not hold. On the other hand, modifying parametricity is a good idea for achieving a finer characterisation of observational equivalence. This is exactly achieved on the syntactic side in Sect. 4. We expect that this modification and relevant results developed in the syntactic approach will provide interesting feedback to the relational interpretation of System F.

References

[BFSS90] E. Bainbridge, P. Freyd, A. Scedrov, and P. Scott. Functorial polymorphism. *Theoretical Computer Science* 70:35–64 (1990).

[BAC95] R. Bellucci, M. Abadi, and P.-L. Curien. A model for formal parametric polymorphism: a PER interpretation for system R. *Proc. 2nd Intl. Conf. on Typed Lambda Calculi and Applications, TLCA'95*, Edinburgh. Springer LNCS 902, 32–46 (1995).

[BHW95] M. Bidoit, R. Hennicker and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer and Programming*, 25:149–186 (1995).

[BB85] C. Böhm and A. Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science* 39:135–154 (1985).

[BMM90] K. Bruce, A. Meyer, and J. Mitchell. The semantics of the second-order lambda calculus. *Information and Computation* 85(1):76–134 (1990).

[Gir71] J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. *Proc. 2nd Scandinavian Logic Symp.*, Oslo. *Studies in Logic and the Foundations of Mathematics*, Vol. 63, 63–92. North-Holland (1971).

[GTL90] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press (1990).

[Han99] J. Hannay. Specification refinement with System F. *Proc. 13th Intl. Workshop on Computer Science Logic, CSL'99*, Madrid. Springer LNCS 1683, 530–545 (1999).

[Han00] J. Hannay. A higher-order simulation relation for System F. *Proc. 3rd Intl. Conf. on Foundations of Software Science and Computation Structures*. ETAPS 2000, Berlin. Springer LNCS 1784, 130–145 (2000).

[Han01] J. Hannay. *Abstraction Barriers and Refinement in the Polymorphic Lambda Calculus*. PhD thesis, Laboratory for Foundations of Computer Science (LFCS), University of Edinburgh (2001).

[Han03] J. Hannay. Abstraction barrier-observing relational parametricity. *Proc. 6th Intl. Conf. on Typed Lambda Calculi and Applications, TLCA 2003*, Valencia. Springer LNCS 2701 (2003).

[Has91] R. Hasegawa. Parametricity of extensionally collapsed term models of polymorphism and their categorical properties. *Proc. Intl. Conf. on Theoretical Aspects of Computer Software, TACS'91*, Sendai. Springer LNCS 526, 495–512 (1991).

[Hoa72] C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica* 1:271–281 (1972).

[HS96] M. Hofmann and D. Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Computer Science* 167:3–45 (1996).

[HLST00] F. Honsell, J. Longley, D. Sannella and A. Tarlecki. Constructive data refinement in typed lambda calculus. *Proc. 3rd Intl. Conf. on Foundations of Software Science and Computation Structures*. ETAPS 2000, Berlin. Springer LNCS 1784, 161–176 (2000).

[HS02] F. Honsell and D. Sannella. Prelogical relations. *Information and Computation* 178:23–43 (2002). Short version in *Proc. Computer Science Logic, CSL'99*, Madrid. Springer LNCS 1683, 546–561 (1999).

[Kat03] S. Katsumata. Behavioural equivalence and indistinguishability in higher-order typed languages. *Selected papers from the 16th Intl. Workshop on*

Algebraic Development Techniques, Frauenchiemsee. Springer LNCS, to appear (2003).

[Lei01] H. Leiß. Second-order pre-logical relations and representation independence. *Proc. 5th Intl. Conf. on Typed Lambda Calculi and Applications, TLCA'01*, Cracow. Springer LNCS 2044, 298–314 (2001).

[MR91] Q. Ma and J. Reynolds. Types, abstraction and parametric polymorphism, part 2. *Proc. 7th Intl. Conf. on Mathematical Foundations of Programming Semantics, MFPS*, Pittsburgh. Springer LNCS 598, 1–40 (1991).

[Mai91] H. Mairson. Outline of a proof theory of parametricity. *Proc. 5th ACM Conf. on Functional Programming and Computer Architecture*, Cambridge, MA. Springer LNCS 523, 313–327 (1991).

[Mil71] R. Milner. An algebraic definition of simulation between programs. *Proc. 2nd Intl. Joint Conf. on Artificial Intelligence*. British Computer Society, 481–489 (1971).

[Mit96] J. Mitchell. *Foundations for Programming Languages*. MIT Press (1996).

[MP88] J. Mitchell and G. Plotkin. Abstract types have existential type. *ACM Trans. on Programming Languages and Systems* 10(3):470–502 (1988).

[PA93] G. Plotkin and M. Abadi. A logic for parametric polymorphism. *Proc. Intl. Conf. Typed Lambda Calculi and Applications, TLCA'93*, Utrecht. Springer LNCS 664, 361–375 (1993).

[PPST00] G. Plotkin, J. Power, D. Sannella and R. Tennent. Lax logical relations. *Proc. 27th Int. Colloq. on Automata, Languages and Programming*, Geneva. Springer LNCS 1853, 85–102 (2000).

[Rey74] J. Reynolds. Towards a theory of type structures. *Programming Symposium (Colloque sur la Programmation)*, Paris, Springer LNCS 19, 408–425 (1974).

[Rey81] J. Reynolds. *The Craft of Programming*. Prentice Hall (1981).

[Rey83] J. Reynolds. Types, abstraction and parametric polymorphism. *Proc. 9th IFIP World Computer Congress*, Paris. North Holland, 513–523 (1983).

[RR94] E. Robinson and G. Rosolini. Reflexive graphs and parametric polymorphism. *Proc., Ninth Annual IEEE Symposium on Logic in Computer Science*, Paris, 364–371. IEEE Computer Society Press (1994).

[ST97] D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing* 9:229–269 (1997).

[Sch90] O. Schoett. Behavioural correctness of data representations. *Science of Computer Programming* 14:43–57 (1990).

[Str67] C. Strachey. Fundamental concepts in programming languages. Lecture notes from the Intl. Summer School in Programming Languages, Copenhagen (1967).

[Ten94] R. Tennent. Correctness of data representations in Algol-like languages. In: *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice Hall (1994).

[Tak98] I. Takeuti. An axiomatic system of parametricity. *Fundamenta Informaticae*, 33(4):397–432 (1998).