On behavioural abstraction and behavioural satisfaction in higher-order logic ⋆

Martin Hofmann [1] and Donald Sannella [2]

Laboratory for Foundations of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom

Abstract

The behavioural semantics of specifications with higher-order logical formulae as axioms is analyzed. A characterization of behavioural abstraction via behavioural satisfaction of formulae in which the equality symbol is interpreted as indistinguishability, which is due to Reichel and was recently generalized to the case of first-order logic by Bidoit *et al*, is further generalized to this case. The fact that higher-order logic is powerful enough to express the indistinguishability relation is used to characterize behavioural satisfaction in terms of ordinary satisfaction, and to develop new methods for reasoning about specifications under behavioural semantics.

1 Introduction

An important ingredient in the use of algebraic specifications to describe data abstractions is the concept of *behavioural equivalence* between algebras, which seems to appropriately capture the "black box" character of data abstractions, see e.g. [GGM76], [GM82], [ST87] and [ST95]. Roughly speaking (since there are different choices of definition), two algebras A, B over a signature Σ are behaviourally equivalent with respect to a distinguished set OBS of *observable types* if all computations that can be expressed using the functions in Σ and

⋆ A condensed version of this paper appeared in: *Proc. 20th Colloq. on Trees in Algebra and Programming*, Intl. Joint Conf. on Theory and Practice of Software Development (TAPSOFT), Aarhus. Springer LNCS 915, 247–261 (1995).

[1] E-mail `mxh@dcs.ed.ac.uk`. Supported by a Human Capital and Mobility fellowship, contract number ERBCHBICT930420.

[2] E-mail `dts@dcs.ed.ac.uk`. Supported by an EPSRC Advanced Fellowship and EPSRC grants GR/H73103 and GR/J07303.

that yield a result of a type in *OBS* produce the same result in both A and B. (The set *OBS* is typically taken to include primitive types like Booleans and natural numbers.) A specification of a data abstraction should characterize a class of algebras that is closed under behavioural equivalence; otherwise it forbids some realizations that are indistinguishable from acceptable ones. Closure can be ensured by the specification framework (by making all specification-building operations deliver closed classes, see e.g. [NO88]) or by the specifier (by applying a specification-building operation, sometimes known as *behavioural abstraction*, to form the closure, see e.g. [SW83], [ST87]). The term "behavioural semantics" is sometimes used to characterize approaches that take the need for behavioural closure into account. Behavioural abstraction seems to be an implicit ingredient of model-oriented approaches to specification such as VDM and Z, where a specification spells out one or more concrete models but any program that delivers the same results is regarded as an acceptable realization.

An unfortunate problem with behavioural semantics in general and the behavioural abstraction operation in particular is that it complicates the task of reasoning about specifications. For example, if a specification *SP* satisfies a formula φ then the behavioural abstraction of *SP* need not satisfy φ. Reasoning methods that are appropriate in the context of behavioural semantics have been developed, but these are either insufficiently powerful (e.g. [ST87], cf. Section 5 of [Sch92]) or tend to be too complicated for convenient use in practice (e.g. [Hen91], [Far92]). One avenue of attack on this problem is to consider the relationship between the class of algebras produced by applying the behavioural abstraction operation to a specification $\langle \Sigma, \Phi \rangle$, and the class of algebras obtained by simply interpreting equality in the axioms Φ as *indistinguishability* rather than as identity. The latter approach, sometimes known as *behavioural satisfaction*, was pioneered by Reichel [Rei85] who showed that these two classes coincide when the axioms involved are conditional equations, provided that the conditions used are equations between terms of types in *OBS*. This yields a reasoning method for specifications involving behavioural abstraction: given a sound proof system for behavioural satisfaction, any consequence φ of a specification $\langle \Sigma, \Phi \rangle$ that can be proved in that system will hold in the behavioural abstraction of $\langle \Sigma, \Phi \rangle$, provided Φ and φ have the required form.

The usefulness of this reasoning method is limited by the fact that conditional equations are not powerful enough for convenient practical use in writing specifications (see e.g. [SW96]). But in a recent development, Bidoit *et al* have generalized Reichel's result to the case of specifications with infinitary first-order equational formulae as axioms, and to arbitrary relations of behavioural equivalence and indistinguishability. In [BHW95] they show that the coincidence of classes described above holds in this context as well, whenever the class of models of $\langle \Sigma, \Phi \rangle$ (under ordinary satisfaction) is closed under quotienting

with respect to indistinguishability of values, provided that indistinguishability is *weakly regular* and that behavioural equivalence is *factorizable* by indistinguishability. Subsequently, [BH95] and [BH96] use this characterization as the basis for reasoning methods.

In this paper we examine these issues for the case of (flat) specifications with higher-order logical formulae as axioms. Our first main contribution is a generalization [3] of the framework and results of [BHW95]. Although it is not made explicit there, the main results in [BHW95] including the characterization theorem do not strongly depend on the form of axioms. The same result holds for any logical system for which behavioural satisfaction of a formula φ in A coincides with ordinary satisfaction of φ in the quotient of A w.r.t. indistinguishability and for which isomorphisms preserve and reflect satisfaction. In Sections 2 and 3 we give syntax and semantics for higher-order formulae and show that these properties hold for such formulae (Theorem 3.35 and Corollary 3.14 respectively). In Section 4 we formulate definitions of behavioural equivalence and indistinguishability, and we show that the former is factorizable by the latter (Theorem 5.21) and that indistinguishability is regular (Proposition 4.7) and hence weakly regular (Proposition 4.9). This leads directly to a characterization result analogous to the one in [BHW95] (Theorem 6.7). Although the generalization to higher-order logic results in certain complications, it also yields a simplification: since equality may be expressed directly in higher-order logic, it need not be given specialized treatment, and the rôle of equality in the context of behavioural semantics is revealed as a special case of something more general.

Higher-order logic provides sufficient power to express the indistinguishability relation as a predicate (Theorem 5.4, cf. [Sch94]). A second main contribution is the application of this fact to develop methods for reasoning about specifications under behavioural semantics. In Section 5 we characterize behavioural satisfaction in terms of ordinary satisfaction, by giving a translation that takes any formula φ to a "relativized" formula $\ulcorner\varphi\urcorner$ such that the latter is satisfied exactly when the former is behaviourally satisfied (Corollary 5.10). This translation plays an important rôle in the comparison of various alternative definitions of behavioural equivalence, differing in the set of "experiments" used to test algebras, which leads to the conclusion that the three definitions considered yield the same relation (Corollary 5.22). These results, together with the characterization theorem of Section 6, lead directly to various proof methods that are summarized in Section 7.

[3] [BHW95] uses the infinitary logic $L_{\omega_1\omega}$, so strictly speaking our framework is not a generalization. The extension to infinitary logic is easy and raises no interesting issues, and it also seems gratuitous in the context of higher-order logic, so we omit it. Also, unlike [BHW95] we do not handle structured specifications, for reasons explained in Section 8.2.

2 The language of higher-order logic

The syntax of the typed variant of higher-order logic we will use is described below. The logic is higher-order because quantification over predicates (i.e. sets) is allowed in addition to the usual quantification over individuals. For the sake of simplicity, functions are not "first-class citizens", so there is no quantification over function types or use of functions as arguments to predicates or functions, but see Section 8.1 for comments on a possible extension.

Definition 2.1 *A* signature Σ *consists of a set* B *of* base types *and a set* C *of* constants *such that each* $c \in C$ *has an arity* $n \geq 0$*, an* n*-tuple of argument types* $b_1, \ldots, b_n \in B$ *and a result type* $b \in B$*, which we abbreviate* $c : b_1 \times \cdots \times b_n \to b$*.*

Let $\Sigma = \langle B, C \rangle$ be a signature.

Definition 2.2 *The* types *over* Σ *are given by the following grammar:*

$$\tau ::= b \mid [\tau_1, \ldots, \tau_n]$$

where $b \in B$ *and* $n \geq 0$*.* $\mathit{Types}(\Sigma)$ *denotes the set of all types over* Σ*.*

A type of the form $[\tau_1, \ldots, \tau_n]$ may be regarded as the type of n-ary predicates taking arguments of types $\tau_1, \ldots, \tau_n$. For example, $\mathit{is\text{-}even} : [\mathit{int}]$, $<: [\mathit{int}, \mathit{int}]$, $\mathit{has\text{-}property} : [\mathit{int}, [\mathit{int}]]$, where $\mathit{is\text{-}even}(3)$ does not hold but $\mathit{has\text{-}property}(2, \mathit{is\text{-}even})$ does. A more suggestive syntax for a type $[\tau_1, \ldots, \tau_n]$ might be $\tau_1 \times \cdots \times \tau_n \to \mathsf{Prop}$, where Prop is the type of propositions; in particular, the type $[\,]$ may be thought of as Prop. However, since λ-abstraction can be used to form predicates but not ordinary functions (see below), we need to distinguish between the arrow used to write the types of constants and the arrow in $\tau_1 \times \cdots \times \tau_n \to \mathsf{Prop}$.

Let X be a fixed infinite set of variables, ranged over by x.

Definition 2.3 *The* terms *over* Σ *are given by the following grammar:*

$$t ::= x \mid c(t_1, \ldots, t_n) \mid \lambda(x_1{:}\tau_1, \ldots, x_n{:}\tau_n).t \mid t(t_1, \ldots, t_n) \mid t \Rightarrow t' \mid \forall x{:}\tau.t$$

where $c \in C$ *and* $n \geq 0$*. Parentheses may be used for grouping as usual; in the absence of parentheses,* $\Rightarrow$ *associates to the right. As usual, we regard* α*-convertible terms as equal, where the binding constructs are* λ *and* $\forall$*. We write* c *as an abbreviation for* $c()$ *when* $c : \to b$*, and* $\forall x_1, \ldots, x_n{:}\tau.t$ *as an abbreviation for* $\forall x_1{:}\tau. \cdots . \forall x_n{:}\tau.t$*.*

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \quad (\textsc{var})$$

$$\frac{\Gamma, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash t : [\,]}{\Gamma \vdash \lambda(x_1{:}\tau_1, \ldots, x_n{:}\tau_n).t : [\tau_1, \ldots, \tau_n]} \quad (\lambda)$$

$$\frac{c : b_1 \times \cdots \times b_n \to b \qquad \Gamma \vdash t_1 : b_1 \quad \cdots \quad \Gamma \vdash t_n : b_n}{\Gamma \vdash c(t_1, \ldots, t_n) : b} \quad (\textsc{fun})$$

$$\frac{\Gamma \vdash t : [\tau_1, \ldots, \tau_n] \qquad \Gamma \vdash t_1 : \tau_1 \quad \cdots \quad \Gamma \vdash t_n : \tau_n}{\Gamma \vdash t(t_1, \ldots, t_n) : [\,]} \quad (\textsc{pred})$$

$$\frac{\Gamma \vdash t : [\,] \qquad \Gamma \vdash t' : [\,]}{\Gamma \vdash t \Rightarrow t' : [\,]} \quad (\Rightarrow)$$

$$\frac{\Gamma, x : \tau \vdash t : [\,]}{\Gamma \vdash \forall x{:}\tau.t : [\,]} \quad (\forall)$$

Fig. 1. Typing rules

Function application (written $c(t_1, \ldots, t_n)$) is distinguished from predicate application (written $t(t_1, \ldots, t_n)$) although both notations are similar. λ-abstraction is for forming predicates; implication ($\Rightarrow$) and universal quantification are for forming propositions. There is just one syntax class for terms: terms that denote individuals (e.g. $+(3, 2)$) are not distinguished syntactically from terms denoting predicates (e.g. $\lambda(x{:}int, y{:}int).prime(+(x, y))$) or propositions (e.g. $\forall P{:}[int].(\forall x{:}int.P(x)) \Rightarrow P(3)$). But in order for a term to denote anything at all, it has to be typable according to the following definitions.

Definition 2.4 *A* context Γ *is a sequence of the form* $x_1 : \tau_1, \ldots, x_n : \tau_n$ *where* $x_i \neq x_j$ *for all* $i \neq j$. *We write* $\Gamma(x_j)$ *for* τ_j *and* $Vars(\Gamma)$ *for* $\{x_1, \ldots, x_n\}$, *and we identify* Γ *with the* $Types(\Sigma)$*-sorted set of variables such that* $\Gamma_\tau = \{x \in Vars(\Gamma) \mid \Gamma(x) = \tau\}$ *for all* $\tau \in Types(\Sigma)$. *Concatenation of contexts, written* Γ, Γ', *is required to yield a context, i.e. it is required that* $Vars(\Gamma) \cap Vars(\Gamma') = \emptyset$. *Let* $T \subseteq Types(\Sigma)$ *be a subset of the set of types over* Σ; *then* Γ *is called a* T-context *if* $\Gamma(x) \in T$ *for all* $x \in Vars(\Gamma)$.

Definition 2.5 *We write* $\Gamma \vdash t : \tau$ *if this judgement is derivable using the rules in Figure 1, and then we call* t *a* term in context Γ. *A term* t *is* closed *if it is typable in the empty context, i.e. if* $\vdash t : \tau$. *A* predicate *(*in context Γ*) is a term* t *such that* $\Gamma \vdash t : [\tau_1, \ldots, \tau_n]$. *A* formula *(*in context Γ*) is a term* φ *such that* $\Gamma \vdash \varphi : [\,]$.

Proposition 2.6 *The following weakening and permutation rules are admissible in the system of rules given in Figure 1:*

$$\frac{\Gamma \vdash t : \tau}{\Gamma, x : \tau' \vdash t : \tau} \quad (\textsc{weak})$$

$$\frac{\Gamma, \Gamma' \vdash t : \tau}{\Gamma', \Gamma \vdash t : \tau} \quad (\textsc{perm})$$

PROOF: *Obvious.* □

There is no need to include equality as a built-in predicate, since it is expressible using higher-order quantification. That is, suppose $\Gamma \vdash t : \tau$ and $\Gamma \vdash t' : \tau$; then

$$t =_\tau t' \text{ abbreviates } \forall P{:}[\tau].P(t) \Rightarrow P(t')$$

where P is chosen arbitrarily such that $P \notin \mathit{Vars}(\Gamma)$.

Existential quantification and the missing connectives are expressible as usual in terms of $\forall$ and $\Rightarrow$:

$$\begin{aligned}
\mathsf{true} &\text{ abbreviates } \forall P{:}[\,].P \Rightarrow P \\
\mathsf{false} &\text{ abbreviates } \forall P{:}[\,].P \\
\neg\varphi &\text{ abbreviates } \varphi \Rightarrow \mathsf{false} \\
\varphi \vee \varphi' &\text{ abbreviates } (\neg\varphi) \Rightarrow \varphi' \\
\varphi \wedge \varphi' &\text{ abbreviates } \neg(\neg\varphi \vee \neg\varphi') \\
\exists x{:}\tau.\varphi &\text{ abbreviates } \neg\forall x{:}\tau.\neg\varphi
\end{aligned}$$

Alternatively, the higher-order encodings of $\vee$, $\wedge$ and $\exists$ could be used. In contrast to those above, these do not presuppose a classical setting, but otherwise there is no essential difference.

$$\begin{aligned}
\varphi \vee \varphi' &\text{ abbreviates } \forall P{:}[\,].(\varphi \Rightarrow P) \Rightarrow (\varphi' \Rightarrow P) \Rightarrow P \\
\varphi \wedge \varphi' &\text{ abbreviates } \forall P{:}[\,].(\varphi \Rightarrow \varphi' \Rightarrow P) \Rightarrow P \\
\exists x{:}\tau.\varphi &\text{ abbreviates } \forall P{:}[\,].(\forall x{:}\tau.\varphi \Rightarrow P) \Rightarrow P
\end{aligned}$$

Finally, there is no need to treat reachability constraints as a special case, since induction principles are expressible — see Example 2.7 below.

The following will be used as a running example to illustrate various definitions and results below. It is chosen for the sake of simplicity and because it was employed in [Sch92] to exhibit a weakness in existing methods for reasoning about specifications involving behavioural abstraction — see Example 7.6.

Example 2.7 We specify a counter which can be set to zero, incremented, decremented (stopping at zero), and tested to see if it is zero. The signature Σ_{ctr} has base types *bool* and *ctr* (counter), and constants $\mathit{zero} :\!\rightarrow \mathit{ctr}$, $\mathit{inc} : \mathit{ctr} \rightarrow \mathit{ctr}$, $\mathit{dec} : \mathit{ctr} \rightarrow \mathit{ctr}$, $\mathit{is\text{-}zero} : \mathit{ctr} \rightarrow \mathit{bool}$, $\mathit{true} :\!\rightarrow \mathit{bool}$ and

$$\neg(true =_{bool} false)$$

$$\forall b{:}bool.(b =_{bool} true \vee b =_{bool} false)$$

$$dec(zero) =_{ctr} zero$$

$$\forall c{:}ctr.dec(inc(c)) =_{ctr} c$$

$$is\text{-}zero(zero) =_{bool} true$$

$$\forall c{:}ctr.is\text{-}zero(inc(c)) =_{bool} false$$

$$\forall P{:}[ctr].(\ P(zero) \wedge \forall c{:}ctr.(P(c) \Rightarrow P(inc(c)))\ ) \Rightarrow \forall c{:}ctr.P(c) \qquad (GENCTR)$$

Fig. 2. The axioms Φ_{ctr}

$false :\rightarrow bool$ (not to be confused with the formulae $\mathsf{true} : [\,]$ and $\mathsf{false} : [\,]$!). These constants are required to satisfy the axioms Φ_{ctr} given in Figure 2. The last axiom (labelled *GENCTR*) expresses a reachability constraint for *ctr*, requiring that all values of type *ctr* are generated by *zero* and *inc*. The second axiom expresses reachability for *bool*.

The only explicit higher-order aspect of this example is in the formula *GENCTR* (there is implicit higher-order quantification in each of the equations) but this will suffice for our purposes. □

The following example gives a better demonstration of the expressive power of the language.

Example 2.8 Consider the signature with base types *sched* (schedule) and *proc* (process) and constants $start :\rightarrow sched$, $step : sched \rightarrow sched$ and $who : sched \rightarrow proc$. We would like to require that *start* is a fair schedule, i.e. that it schedules each process infinitely often. The following is essentially a translation of a formula in the modal mu-calculus [Sti92] into higher-order logic.

We begin with the least and greatest fixed point operators, which can be expressed directly as follows:

$$\mu =_{\text{def}} \lambda(\Phi{:}[[sched], sched]\ ,\ s{:}sched).$$
$$\forall P{:}[sched].(\forall s'{:}sched.\Phi(P, s') \Rightarrow P(s')) \Rightarrow P(s)$$

$$\nu =_{\text{def}} \lambda(\Phi{:}[[sched], sched]\ ,\ s{:}sched).$$
$$\exists P{:}[sched].(\forall s'{:}sched.P(s') \Rightarrow \Phi(P, s')) \wedge P(s)$$

The predicate *always* (a given predicate holds at every step in a given schedule) is expressed as a greatest fixed point, and the predicate *eventually* is expressed as a least fixed point:

$$\begin{aligned}
&\mathit{always} =_{\text{def}} \lambda(P{:}[\mathit{sched}], s{:}\mathit{sched}).\\
&\qquad \nu(\lambda(\mathit{always\text{-}P}{:}[\mathit{sched}], s'{:}\mathit{sched}).\\
&\qquad\qquad P(s') \wedge \mathit{always\text{-}P}(\mathit{step}(s'))\ ,\ s)\\
&\mathit{eventually} =_{\text{def}} \lambda(P{:}[\mathit{sched}], s{:}\mathit{sched}).\\
&\qquad \mu(\lambda(\mathit{eventually\text{-}P}{:}[\mathit{sched}], s'{:}\mathit{sched}).\\
&\qquad\qquad P(s') \vee \mathit{eventually\text{-}P}(\mathit{step}(s'))\ ,\ s)
\end{aligned}$$

These are used to code a predicate which checks that a given predicate holds infinitely often in a given schedule:

$$\begin{aligned}
&\mathit{infinitely\text{-}often} =_{\text{def}}\\
&\qquad \lambda(P{:}[\mathit{sched}], s{:}\mathit{sched}).\ \mathit{always}((\lambda(s'{:}\mathit{sched}).\mathit{eventually}(P, s'))\ ,\ s)
\end{aligned}$$

Then the required fairness property is $\mathit{fair}(\mathit{start})$, where *fair* is expressed in terms of *infinitely-often* as follows:

$$\mathit{fair} =_{\text{def}} \lambda(s{:}\mathit{sched}).\ \forall p{:}\mathit{proc}.\mathit{infinitely\text{-}often}((\lambda(s'{:}\mathit{sched}).\mathit{who}(s') = p)\ ,\ s)$$

Expanding $\mathit{fair}(\mathit{start})$ gives a single formula expressing the required property. □

The language defined above is a trimmed version of the "classical theory of simple types" as introduced by Henkin in [Hen50]. Henkin considers non-standard models for which a natural Gentzen-style proof system is sound and complete. A good reference is also Chapter 4 of Schütte's monograph [Sch77] where cut-elimination for this system is established.

3 Semantics of higher-order logic

Let $\Sigma = \langle B, C\rangle$ be a signature.

Terms over Σ are interpreted in the context of a Σ-algebra which gives meaning to the base types and the constants in Σ.

Definition 3.1 *A* Σ-algebra A *consists of a* carrier set $[\![b]\!]_A$ *for every* $b \in B$,

and interpretations of constants $[\![c]\!]_A \in ([\![b_1]\!]_A \times \cdots \times [\![b_n]\!]_A \to [\![b]\!]_A)$ *for every* $c : b_1 \times \cdots \times b_n \to b$ *in* C*. The class of all* Σ*-algebras is denoted* $Alg(\Sigma)$*.* Σ-homomorphisms *and* Σ-isomorphisms *are as usual; we write* $A \cong A'$ *if there is a* Σ*-isomorphism* $h : A \to A'$*.*

Let A be a Σ-algebra.

We define two interpretations for terms. The first is the obvious "standard" interpretation with respect to an environment mapping free variables to values. The second interpretation is modulo a partial congruence relation on A. In the latter interpretation, quantification (and λ-abstraction) is over only those elements of types that respect the congruence; as a result, equality in formulae refers to the congruence rather than to identity of values. The particular partial congruence of interest will be a relation of indistinguishability with respect to a given set of observable base types, to be defined in Section 4. Theorem 3.35 below demonstrates a relationship between the two interpretations that will be crucial in the sequel.

Our use of *partial* congruences in Section 3.2 below stems from the need to establish an appropriate relationship between indistinguishability and behavioural equivalence, see Theorem 5.21, in order to apply the characterization theorems in Section 6. If the indistinguishability relation were not defined as a partial congruence, the desired relationship with the behavioural equivalence relation would not hold.

3.1 Standard interpretation

Definition 3.2 *Types of the form* $[\tau_1, \ldots, \tau_n]$ *are interpreted as follows:*

$$[\![[\tau_1, \ldots, \tau_n]]\!]_A = Pow([\![\tau_1]\!]_A \times \cdots \times [\![\tau_n]\!]_A).$$

Thus, $[\![[\,]]\!]_A$ is $\{\{\}, \{*\}\}$ where $*$ is the empty tuple. Recalling that $[\,]$ means Prop, $\{\}$ may be thought of as denoting falsity and $\{*\}$ as denoting truth, so we will use the abbreviation *ff* for $\{\}$ and *tt* for $\{*\}$.

Let Γ be a context.

Definition 3.3 *A* Γ-environment *(*on *A) is a* $Types(\Sigma)$*-sorted function* $\rho = \langle \rho_\tau : \Gamma_\tau \to [\![\tau]\!]_A \rangle_{\tau \in Types(\Sigma)}$*. We write* $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ *to denote the evident environment, and the notation* $\rho[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ *denotes the environment* ρ *superseded at* $x_1, \ldots, x_n$ *by* $v_1, \ldots, v_n$ *respectively. When* $x \in Vars(\Gamma)$ *we write* $\rho(x)$ *for* $\rho_{\Gamma(x)}(x)$*. Let* $T \subseteq Types(\Sigma)$*; a* Γ*-environment* ρ *is* T-surjective *if* $\rho_\tau : \Gamma_\tau \to [\![\tau]\!]_A$ *is surjective for each* $\tau \in T$*.*

Definition 3.4 *Let ρ be a Γ-environment. The interpretation of constants is extended to terms in context Γ as follows:*

$$\begin{aligned}
[\![x]\!]_{\rho,A} &= \rho(x) \\
[\![c(t_1,\ldots,t_n)]\!]_{\rho,A} &= [\![c]\!]_A([\![t_1]\!]_{\rho,A},\ldots,[\![t_n]\!]_{\rho,A}) \\
[\![\lambda(x_1{:}\tau_1,\ldots,x_n{:}\tau_n).t]\!]_{\rho,A} &= \{(v_1,\ldots,v_n) \mid v_1 \in [\![\tau_1]\!]_A \textit{ and } \cdots \textit{ and } v_n \in [\![\tau_n]\!]_A \\
&\qquad\qquad \textit{and } [\![t]\!]_{\rho[x_1\mapsto v_1,\ldots,x_n\mapsto v_n],A} = tt\} \\
[\![t(t_1,\ldots,t_n)]\!]_{\rho,A} &= \textit{if } ([\![t_1]\!]_{\rho,A},\ldots,[\![t_n]\!]_{\rho,A}) \in [\![t]\!]_{\rho,A} \textit{ then } tt \textit{ else } ff \\
[\![t \Rightarrow t']\!]_{\rho,A} &= \textit{if } [\![t]\!]_{\rho,A} = tt \textit{ then } [\![t']\!]_{\rho,A} \textit{ else } tt \\
[\![\forall x{:}\tau.t]\!]_{\rho,A} &= \textit{if } [\![t]\!]_{\rho[x\mapsto v],A} = tt \textit{ for all } v \in [\![\tau]\!]_A \textit{ then } tt \textit{ else } ff
\end{aligned}$$

It is easy to see that the interpretation $[\![t]\!]_{\rho,A}$ of a term t does not depend on ρ when t is closed. We may therefore omit ρ and use the notation $[\![t]\!]_A$ in this case.

The following substitution property will be handy in proofs in later sections. As usual, $s[x := t]$ denotes the result of simultaneously replacing all occurrences of the variable x in the term s by the term t, with appropriate changes of bound variable names to avoid variable capture.

Proposition 3.5 *For any $\Gamma \vdash t : \tau$ and $\Gamma, x : \tau \vdash s : \tau'$ and any Γ-environment ρ, $[\![s]\!]_{\rho[x\mapsto[\![t]\!]_{\rho,A}],A} = [\![s[x := t]]\!]_{\rho,A}$.*

PROOF: *By induction on the structure of s.* □

The following shows that the above interpretation of terms and types is sound with respect to the typing relation.

Proposition 3.6 *If $\Gamma \vdash t : \tau$ and ρ is a Γ-environment then $[\![t]\!]_{\rho,A} \in [\![\tau]\!]_A$.*

PROOF: *By induction on the structure of the derivation of $\Gamma \vdash t : \tau$.* □

The following proposition demonstrates that $=_\tau$ really is equality (i.e. identity of values).

Proposition 3.7 *Suppose $v, v' \in [\![\tau]\!]_A$ for some type τ. Then for any environment ρ, $[\![x =_\tau y]\!]_{\rho[x\mapsto v,y\mapsto v'],A} = tt$ iff $v = v'$.*

PROOF:

$\Longleftarrow$: *Obvious.*

$\Longrightarrow$: *Suppose* $[\![\forall P{:}[\tau].P(x) \Rightarrow P(y)]\!]_{\rho[x\mapsto v,y\mapsto v'],A} = tt$ *and consider the predicate* $\{v\} \in [\![[\tau]]\!]_A$. *We have* $[\![P(x)]\!]_{\rho[x\mapsto v,P\mapsto\{v\}],A} = tt$ *and so* $[\![P(y)]\!]_{\rho[y\mapsto v',P\mapsto\{v\}],A} = tt$; *thus* $v' \in \{v\}$, *i.e.* $v = v'$. □

This entitles us to use $\varphi \Leftrightarrow \varphi'$ as an abbreviation for $\varphi =_{[]} \varphi'$ instead of the equivalent but longer $(\varphi \Rightarrow \varphi') \wedge (\varphi' \Rightarrow \varphi)$.

It is easy to see that the abbreviations defined for the connectives $\neg$, $\vee$, $\wedge$ and for $\exists$ and true have the expected meaning. The following shows that the abbreviation defined for false is also correct.

Proposition 3.8 *For any environment* ρ, $[\![\mathsf{false}]\!]_{\rho,A} = ff$.

PROOF: *Recall that* false *is* $\forall P{:}[\,].P$ *and take* $P = ff \in [\![[\,]]\!]_A$. □

Definition 3.9 *Let* φ *be a formula in context* Γ. *Suppose* ρ *is a* Γ*-environment; then we write* $A \models_\rho \varphi$ *if* $[\![\varphi]\!]_{\rho,A} = tt$. *We write* $A \models \varphi$ (*A* satisfies φ) *if* $A \models_\rho \varphi$ *for all* Γ*-environments* ρ. *If* φ' *is also a formula in context* Γ, *we write* $\varphi \models\!\!\models \varphi'$ (φ *is* equivalent *to* φ') *if for all* $A \in Alg(\Sigma)$ *and all* Γ*-environments* ρ, $A \models_\rho \varphi$ *iff* $A \models_\rho \varphi'$. *Finally, if* Φ *is a set of formulae in context* Γ *then we write* $A \models \Phi$ *if* $A \models \varphi$ *for all* $\varphi \in \Phi$.

Example 3.10 A Σ_{ctr}-algebra *Nat* is defined by taking $[\![ctr]\!]_{Nat} = \omega$ (the set of natural numbers) and $[\![bool]\!]_{Nat} = \{T, F\}$ (not to be confused with *tt* and *ff*!) with the evident interpretation of the constants. This satisfies all of the axioms in Φ_{ctr}; see below for comments concerning *GENCTR*. Another Σ_{ctr}-algebra is the term algebra *Term*, defined by taking $[\![ctr]\!]_{Term} = \{t \mid \vdash t : ctr\}$ and $[\![bool]\!]_{Term} = \{T, F\}$ with $[\![is\text{-}zero]\!]_{Term}(t) = [\![is\text{-}zero(t)]\!]_{Nat}$ for $t \in [\![ctr]\!]_{Term}$. The only axioms in Φ_{ctr} that *Term* satisfies are $is\text{-}zero(zero) =_{bool} true$, $\forall c{:}ctr.is\text{-}zero(inc(c)) =_{bool} false$, $\neg(true =_{bool} false)$ and $\forall b{:}bool.(b =_{bool} true \vee b =_{bool} false)$. □

Definition 3.11 *Let* $B' \subseteq B$ *be a subset of the set of base types in* Σ, *and let* $b \in B$. *A value* $v \in [\![b]\!]_A$ *is* B'-reachable *if there is a* B'*-context* Γ, *a term* t *with* $\Gamma \vdash t : b$, *and a* Γ*-environment* ρ, *such that* $[\![t]\!]_{\rho,A} = v$.

Intuitively, v is B'-reachable if v can be obtained by application of constants to values of types in B'.

Example 3.12 Let Σ'_{ctr} be Σ_{ctr} without the constant $dec : ctr \to ctr$. For any Σ_{ctr}-algebra A, $A \models GENCTR$ iff every value of type *ctr* in A is $\emptyset$-reachable with respect to the reduced signature Σ'_{ctr}. (The "if" direction is obvious; to see that the "only if" direction holds, simply instantiate *GENCTR* with the predicate $P = \{v \in [\![ctr]\!]_A \mid v \text{ is } \emptyset\text{-reachable in } \Sigma'_{ctr}\} \in [\![[ctr]]\!]_A$.) Then $Nat \models GENCTR$ and $Term \not\models GENCTR$ (since $dec(zero) \in [\![ctr]\!]_{Term}$ is not

$\emptyset$-reachable in Σ'_{ctr}). □

The following proposition is used to show that isomorphisms preserve and reflect satisfaction, as in almost any conceivable logical system.

Proposition 3.13 *Let $h : A \to A'$ be an isomorphism. Extend h to bracket types by taking $h_{[\tau_1,\ldots,\tau_n]}(p) = \{(h_{\tau_1}(v_1), \ldots, h_{\tau_n}(v_n)) \mid (v_1, \ldots, v_n) \in p\} \in [\![[\tau_1, \ldots, \tau_n]]\!]_{A'}$ for $p \in [\![[\tau_1, \ldots, \tau_n]]\!]_A$. Let t be a term in context Γ and let ρ be a Γ-environment; then $h([\![t]\!]_{\rho,A}) = [\![t]\!]_{h\circ\rho,A'}$.*

PROOF: *By induction on the structure of t.* □

Corollary 3.14 *If $A \cong A'$ then $A \models \varphi$ iff $A' \models \varphi$.*

PROOF: *By Proposition 3.13, since $h_{[]}$ is the identity and since h_τ is a bijection for every type τ.* □

3.2 Interpretation w.r.t. a partial congruence

Definition 3.15 *A* partial congruence $\approx$ *on A is a family of* partial equivalence relations *(i.e., symmetric and transitive relations)* $\langle \approx_b \subseteq [\![b]\!]_A \times [\![b]\!]_A \rangle_{b\in B}$ *such that for all $c : b_1 \times \cdots \times b_n \to b$ in C and all $v_1, v'_1 \in [\![b_1]\!]_A, \ldots, v_n, v'_n \in [\![b_n]\!]_A$, if $v_1 \approx_{b_1} v'_1$ and $\cdots$ and $v_n \approx_{b_n} v'_n$ then $[\![c]\!]_A(v_1, \ldots, v_n) \approx_b [\![c]\!]_A(v'_1, \ldots, v'_n)$. A (*total*) congruence is a reflexive partial congruence.*

Example 3.16 Equality is a total congruence on *Nat*. A total congruence on *Term* is given by $t \approx_{ctr} t' \Leftrightarrow [\![t]\!]_{Nat} = [\![t']\!]_{Nat}$ for all $t, t' \in [\![ctr]\!]_{Term}$ (so we have e.g. $dec(inc(zero)) \approx_{ctr} zero$) and $b \approx_{bool} b' \Leftrightarrow b = b'$. This is a partial congruence on the Σ_{ctr}-algebra $Term^\infty$ defined by taking $[\![ctr]\!]_{Term^\infty} = [\![ctr]\!]_{Term} \cup \{\infty\}$ and $[\![bool]\!]_{Term^\infty} = [\![bool]\!]_{Term}$, with interpretation of constants as in *Term* augmented by $[\![inc]\!]_{Term^\infty}(\infty) = [\![dec]\!]_{Term^\infty}(\infty) = \infty$ and $[\![is\text{-}zero]\!]_{Term^\infty}(\infty) = F$. Another total congruence on *Term*, which is again a partial congruence on $Term^\infty$, is obtained by taking $t \approx'_{ctr} t' \Leftrightarrow [\![t]\!]_{Nat} \equiv [\![t']\!]_{Nat} \pmod 2$ for all $t, t' \in [\![ctr]\!]_{Term}$ and $b \approx'_{bool} b'$ for all $b, b' \in \{T, F\}$ (this is forced by the fact that $inc(inc(zero)) \approx'_{ctr} zero$). □

Let $\approx$ be a partial congruence on A. As suggested at the beginning of Section 3, the partial congruence of interest will be a relation of indistinguishability to be defined later. The reader may find it helpful to keep this in mind in order to understand the motivation behind some of the definitions and results below. We do not restrict attention to this particular partial congruence at this point because much of the sequel does not depend on the special features of this relation, and because there are several different indistinguishability relations

of potential interest (although we will consider only one).

The idea behind the development which follows is to generalize the usual definition of satisfaction up to a partial congruence in first-order equational logic to higher-order logic. Whereas in the first-order case it is enough to interpret the primitive equality symbol as the partial congruence and to restrict all quantifiers to values lying in the domain of the partial congruence, the situation is more complicated here. We must make sure that the predicate variables only range over predicates which "respect" the partial congruence. What this means exactly is not entirely obvious for types with nested brackets. That the definition we give is indeed the right generalization of the first-order case is shown by Proposition 3.29 and Theorem 3.35. In the first-order case, Proposition 3.29 is obvious from the definition of satisfaction.

The following definition explains how to extend the partial congruence $\approx$, which relates values of base types only, to a so-called *logical relation* (see e.g. [Mit90]) over all types. The resulting relation will be used below to give an interpretation of bracket types.

Definition 3.17 *We extend $\approx$ to "bracket" types by taking $p \approx_{[\tau_1,\ldots,\tau_n]} p'$ for $p, p' \in [\![[\tau_1, \ldots, \tau_n]]\!]_A$ iff for all $v_1, v_1' \in [\![\tau_1]\!]_A, \ldots, v_n, v_n' \in [\![\tau_n]\!]_A$, if $v_1 \approx_{\tau_1} v_1'$ and $\cdots$ and $v_n \approx_{\tau_n} v_n'$ then $(v_1, \ldots, v_n) \in p$ iff $(v_1', \ldots, v_n') \in p'$. We say that $v \in [\![\tau]\!]_A$* respects $\approx$ *if $v \approx_\tau v$.*

A predicate $p \in [\![[\tau_1, \ldots, \tau_n]]\!]_A$ respects $\approx$ if it does not differentiate between values that are related by $\approx$. Note that trivially $\emptyset \approx_{[\tau_1,\ldots,\tau_n]} \emptyset$, and that $v \approx_{[]} v'$ iff $v = v'$.

Example 3.18 Consider $\approx$ on *Term* and $Term^\infty$ given in Example 3.16. Let $P = \{t \in [\![ctr]\!]_{Term} \mid t \text{ does not contain } dec\}$ and $P' = \{t \in [\![ctr]\!]_{Term} \mid [\![t]\!]_{Nat} \equiv 0 \pmod 2\}$, so $P, P' \in [\![[ctr]]\!]_{Term} \subset [\![[ctr]]\!]_{Term^\infty}$; then P does not respect $\approx$ (since $dec(inc(zero)) \approx_{ctr} zero$ and $zero \in P$ but $dec(inc(zero)) \notin P$) but P' does. Since $\approx$ is a partial congruence on $Term^\infty$, its extension to bracket types may identify distinct values. For instance, $\{\infty\} \approx_{[ctr]} \emptyset$ since $v \approx_{ctr} v$ implies that $v \neq \infty$. This in turn directly implies that $\{\{\infty\}\} \not\approx_{[[ctr]]} \{\{\infty\}\}$ (note that $\{\{\infty\}\} \in [\![[[ctr]]]\!]_{Term^\infty}$ is the predicate that holds only for the predicate $\{\infty\} \in [\![[ctr]]\!]_{Term^\infty}$). Finally, for an example of identification of distinct values at third-order types, consider $\{P\} \approx_{[[ctr]]} \emptyset$. □

Proposition 3.19 *$\approx_\tau$ is a partial equivalence relation for any type τ.*

PROOF: *Obvious.* □

Note that extending a (total) congruence to bracket types does not in general yield a (total) equivalence relation.

Corollary 3.20 *If $v \approx_\tau v'$ then v and v' respect $\approx$.*

PROOF: *Apply symmetry and transitivity.* □

The difference between the standard interpretation of terms and their interpretation with respect to a partial congruence stems from the following definition.

Definition 3.21 *Interpretation of types w.r.t. $\approx$ is defined as follows:*

$$[\![b]\!]^{\approx}_A = \{v \in [\![b]\!]_A \mid v \text{ respects } \approx\}$$
$$[\![[\tau_1, \ldots, \tau_n]]\!]^{\approx}_A = \{p \in Pow([\![\tau_1]\!]^{\approx}_A \times \cdots \times [\![\tau_n]\!]^{\approx}_A) \mid p \text{ respects } \approx\}$$

We have $[\![[\,]]\!]^{\approx}_A = [\![[\,]]\!]_A = \{ff, tt\}$. Note that if $\approx$ is a congruence, then $[\![b]\!]^{\approx}_A = [\![b]\!]_A$. The second clause of the above definition is well-formed because of the following proposition.

Proposition 3.22 $[\![\tau]\!]^{\approx}_A \subseteq [\![\tau]\!]_A$ *for any type τ.*

PROOF: *By induction on the structure of τ. (Thus the proof that $[\![[\tau_1, \ldots, \tau_n]]\!]^{\approx}_A$ is well-defined depends on $[\![\tau_1]\!]^{\approx}_A, \ldots, [\![\tau_n]\!]^{\approx}_A$, which have been shown to be well-defined at a previous stage.)* □

Example 3.23 Consider $\approx$ on $Term^\infty$ given in Example 3.16. We have $\infty \notin [\![ctr]\!]^{\approx}_{Term^\infty}$ but $v \in [\![ctr]\!]^{\approx}_{Term^\infty}$ for all other $v \in [\![ctr]\!]_{Term^\infty}$. From Example 3.18 we have $P \notin [\![[ctr]]\!]^{\approx}_{Term^\infty}$, $P' \in [\![[ctr]]\!]^{\approx}_{Term^\infty}$ and $\{\{\infty\}\} \notin [\![[[ctr]]]\!]^{\approx}_{Term^\infty}$. □

The following proposition shows that the extension of $\approx$ to bracket types, restricted to type interpretations $[\![[\tau_1, \ldots, \tau_n]]\!]^{\approx}_A$, is trivial in the sense that it does not identify distinct values.

Proposition 3.24 *For all $p, p' \in [\![[\tau_1, \ldots, \tau_n]]\!]^{\approx}_A$, if $p \approx_{[\tau_1, \ldots, \tau_n]} p'$ then $p = p'$.*

PROOF: *Let $v_1 \in [\![\tau_1]\!]^{\approx}_A, \ldots, v_n \in [\![\tau_n]\!]^{\approx}_A$; then $v_1 \approx_{\tau_1} v_1$ and $\cdots$ and $v_n \approx_{\tau_n} v_n$. If $p \approx_{[\tau_1, \ldots, \tau_n]} p'$ then $(v_1 \ldots, v_n) \in p$ iff $(v_1 \ldots, v_n) \in p'$, i.e. $p = p'$.* □

Returning to Example 3.18, we see that $\{\infty\} \notin [\![[ctr]]\!]^{\approx}_{Term^\infty}$ and $\{P\} \notin [\![[[ctr]]]\!]^{\approx}_{Term^\infty}$ and thus these do not provide counterexamples to Proposition 3.24.

Let Γ be a context.

Definition 3.25 *A* Γ-environment *(*w.r.t. $\approx$, on *A) is a Types(Σ)-sorted func-*

tion $\rho = \langle \rho_\tau : \Gamma_\tau \to [\![\tau]\!]^{\approx}_A \rangle_{\tau \in Types(\Sigma)}$. *We adopt the previously-explained notations for environments.*

Definition 3.26 *Let* ρ *be a* Γ*-environment w.r.t.* $\approx$*. The interpretation w.r.t.* $\approx$ *of terms that are typable in context* Γ *is defined as follows:*

$$
\begin{aligned}
[\![x]\!]^{\approx}_{\rho,A} &= \rho(x) \\
[\![c(t_1,\ldots,t_n)]\!]^{\approx}_{\rho,A} &= [\![c]\!]_A([\![t_1]\!]^{\approx}_{\rho,A},\ldots,[\![t_n]\!]^{\approx}_{\rho,A}) \\
[\![\lambda(x_1{:}\tau_1,\ldots,x_n{:}\tau_n).t]\!]^{\approx}_{\rho,A} &= \{(v_1,\ldots,v_n) \mid v_1 \in [\![\tau_1]\!]^{\approx}_A \text{ and } \cdots \text{ and } v_n \in [\![\tau_n]\!]^{\approx}_A \\
&\qquad \text{and } [\![t]\!]^{\approx}_{\rho[x_1\mapsto v_1,\ldots,x_n\mapsto v_n],A} = tt\} \\
[\![t(t_1,\ldots,t_n)]\!]^{\approx}_{\rho,A} &= \textit{if } ([\![t_1]\!]^{\approx}_{\rho,A},\ldots,[\![t_n]\!]^{\approx}_{\rho,A}) \in [\![t]\!]^{\approx}_{\rho,A} \textit{ then } tt \textit{ else } ff \\
[\![t \Rightarrow t']\!]^{\approx}_{\rho,A} &= \textit{if } [\![t]\!]^{\approx}_{\rho,A} = tt \textit{ then } [\![t']\!]^{\approx}_{\rho,A} \textit{ else } tt \\
[\![\forall x{:}\tau.t]\!]^{\approx}_{\rho,A} &= \textit{if } [\![t]\!]^{\approx}_{\rho[x\mapsto v],A} = tt \textit{ for all } v \in [\![\tau]\!]^{\approx}_A \textit{ then } tt \textit{ else } ff
\end{aligned}
$$

A comparison of the above definition with the corresponding definition for the standard interpretation (Definition 3.4) reveals that the only difference is the change to the meaning of λ-abstraction and universal quantification induced by the different interpretation of types.

The proof of soundness does not go through directly; a stronger induction hypothesis is required.

Proposition 3.27 *Suppose* $\Gamma \vdash t : \tau$ *and* ρ, ρ' *are* Γ*-environments w.r.t.* $\approx$ *such that* $\rho(x) \approx_{\Gamma(x)} \rho'(x)$ *for each* $x \in Vars(\Gamma)$*. Then:*

(i) If τ *is a base type* b*, then* $[\![t]\!]^{\approx}_{\rho,A}, [\![t]\!]^{\approx}_{\rho',A} \in [\![b]\!]_A$
(ii) If τ *is* $[\tau_1,\ldots,\tau_n]$*, then* $[\![t]\!]^{\approx}_{\rho,A}, [\![t]\!]^{\approx}_{\rho',A} \in Pow([\![\tau_1]\!]^{\approx}_A \times \cdots \times [\![\tau_n]\!]^{\approx}_A)$

and $[\![t]\!]^{\approx}_{\rho,A} \approx_\tau [\![t]\!]^{\approx}_{\rho',A}$.

PROOF: *By induction on the structure of the derivation of* $\Gamma \vdash t : \tau$. □

Corollary 3.28 *If* $\Gamma \vdash t : \tau$ *and* ρ *is a* Γ*-environment w.r.t.* $\approx$ *then* $[\![t]\!]^{\approx}_{\rho,A} \in [\![\tau]\!]^{\approx}_A$.

PROOF: *Apply Proposition 3.27 with* $\rho' = \rho$. □

The following proposition shows that $=_\tau$ refers to the partial congruence $\approx$ under interpretation of terms w.r.t. $\approx$. This is due to the fact that the quantifier in the formula $\forall P{:}[\tau].P(t) \Rightarrow P(t')$ (which $t =_\tau t'$ abbreviates) ranges over predicates that respect $\approx$.

Proposition 3.29 *Suppose $v, v' \in [\![\tau]\!]^{\approx}_A$ for some type τ. Then for any environment ρ w.r.t. $\approx$, $[\![x =_\tau y]\!]^{\approx}_{\rho[x\mapsto v, y\mapsto v'], A} = tt$ iff $v \approx_\tau v'$.*

PROOF:

$\Longleftarrow$*: Suppose $v \approx_\tau v'$. Any predicate $p \in [\![[\tau]]\!]^{\approx}_A$ respects $\approx$, i.e. $v \in p$ iff $v' \in p$.*

$\Longrightarrow$*: Suppose that $[\![\forall P{:}[\tau].P(x) \Rightarrow P(y)]\!]^{\approx}_{\rho[x\mapsto v, y\mapsto v'], A} = tt$ and consider the predicate $p = \{w \mid w \approx_\tau v\} \in [\![[\tau]]\!]^{\approx}_A$. We have $[\![P(x)]\!]^{\approx}_{\rho[x\mapsto v, P\mapsto p], A} = tt$ so $[\![P(y)]\!]^{\approx}_{\rho[y\mapsto v', P\mapsto p], A} = tt$; thus $v' \in p$, i.e. $v' \approx_\tau v$.* □

Definition 3.30 *Let φ be a formula in context Γ. Suppose ρ is a Γ-environment w.r.t. $\approx$; then we write $A \models^{\approx}_\rho \varphi$ if $[\![\varphi]\!]^{\approx}_{\rho, A} = tt$. We write $A \models^{\approx} \varphi$ (A* satisfies *φ* w.r.t. *$\approx$) if $A \models^{\approx}_\rho \varphi$ for all Γ-environments ρ w.r.t. $\approx$. If Φ is a set of formulae in context Γ then we write $A \models^{\approx} \Phi$ if $A \models^{\approx} \varphi$ for all $\varphi \in \Phi$.*

When $\approx$ is the indistinguishability relation (see Definition 4.1 below), $\models^{\approx}$ is known as *behavioural satisfaction*.

Example 3.31 The Σ_{ctr}-algebra *Nat* satisfies all of the axioms in Φ_{ctr} with respect to equality, since it satisfies them in the standard sense (Example 3.10). The algebras *Term* and $Term^\infty$ satisfy all of these axioms with respect to the congruence $\approx$ given in Example 3.16; see below for comments concerning *GENCTR*. They satisfy all of them with respect to $\approx'$ with the exception of the axiom $\neg(true =_{bool} false)$.

For any Σ_{ctr}-algebra A and any partial congruence $\approx$ on A, $A \models^{\approx} GENCTR$ iff every value in $[\![ctr]\!]^{\approx}_A$ is congruent to a value that is $\emptyset$-reachable with respect to the reduced signature Σ'_{ctr} from Example 3.12. (The "if" direction depends on the requirement that $P \in [\![[ctr]]\!]^{\approx}_A$ rather than $P \in [\![[ctr]]\!]_A$; to see that the "only if" direction holds, instantiate *GENCTR* with $P = \{v \in [\![ctr]\!]^{\approx}_A \mid \exists v' \in [\![ctr]\!]^{\approx}_A . v \approx_{ctr} v'$ and v' is $\emptyset$-reachable in $\Sigma'_{ctr}\} \in [\![[ctr]]\!]^{\approx}_A$. Note that there are A and $\approx$ for which $\{v \in [\![ctr]\!]^{\approx}_A \mid v$ is $\emptyset$-reachable in $\Sigma'_{ctr}\} \notin [\![[ctr]]\!]^{\approx}_A$.) This shows that $Term \models^{\approx} GENCTR$ and $Term^\infty \models^{\approx} GENCTR$ for $\approx$ as given in Example 3.16. □

3.3 Relating $\models$ and $\models^{\approx}$

Let $\approx$ be a partial congruence on A.

Definition 3.32 *Suppose $v \in [\![b]\!]_A$ for $b \in B$ such that $v \approx_b v$; then the* congruence class *of v w.r.t. $\approx$ is defined as $[v]_{\approx_b} = \{v' \in [\![b]\!]_A \mid v \approx_b v'\}$. The*

quotient *of A by $\approx$, written $A/\approx$, is then defined as follows:*

$$[\![b]\!]_{A/\approx} = \{[v]_{\approx_b} \mid v \in [\![b]\!]_A \text{ and } v \approx_b v\} \text{ for all } b \in B$$
$$[\![c]\!]_{A/\approx}([v_1]_{\approx_{b_1}}, \ldots, [v_n]_{\approx_{b_n}}) = [[\![c]\!]_A(v_1, \ldots, v_n)]_{\approx_b}$$
$$\text{for all } c : b_1 \times \cdots \times b_n \to b \text{ in } C.$$

Since $\approx$ is a partial congruence, the choice of representatives $v_1, \ldots, v_n$ in the definition of $[\![c]\!]_{A/\approx}$ doesn't matter. Note that if $\approx$ is a congruence, then $A/\approx$ is the usual quotient algebra, with $[\![b]\!]_{A/\approx} = [\![b]\!]_A/\approx_b$.

Proposition 3.33 *$A/\approx$ is a Σ-algebra, that is $[\![c]\!]_{A/\approx} \in ([\![b_1]\!]_{A/\approx} \times \cdots \times [\![b_n]\!]_{A/\approx} \to [\![b]\!]_{A/\approx})$ for every $c : b_1 \times \cdots \times b_n \to b$ in C.*

PROOF: *Easy, since if $v_1 \in [\![b_1]\!]_{A/\approx}, \ldots, v_n \in [\![b_n]\!]_{A/\approx}$ then $v_1 \approx_{b_1} v_1$ and $\cdots$ and $v_n \approx_{b_n} v_n$ so $[\![c]\!]_A(v_1, \ldots, v_n) \approx_b [\![c]\!]_A(v_1, \ldots, v_n)$.* □

Example 3.34 *Nat* $\cong$ *Term*$/\approx \cong$ *Term*$^\infty/\approx$ for $\approx$ as given in Example 3.16. □

The following theorem demonstrates a fundamental relationship between the two interpretations defined above. In the first-order case, it says that standard satisfaction of a formula φ in a quotient algebra $A/\approx$ is equivalent to satisfaction of φ, with the symbol $=$ interpreted as $\approx$, in A itself.

Theorem 3.35 *$A/\approx \models \varphi$ iff $A \models^\approx \varphi$.*

PROOF: *Define two families of functions $\langle \psi_\tau : [\![\tau]\!]^\approx_A \to [\![\tau]\!]_{A/\approx} \rangle_{\tau \in Types(\Sigma)}$ and $\langle \chi_\tau : [\![\tau]\!]_{A/\approx} \to [\![\tau]\!]^\approx_A \rangle_{\tau \in Types(\Sigma)}$ by induction on τ as follows:*

for all $v \in [\![b]\!]^\approx_A$,
$$\psi_b(v) = [v]_{\approx_b}$$
for all $v \in [\![b]\!]_{A/\approx}$,
$$\chi_b(v) = \text{some arbitrary element of } v$$
for all $p \in [\![[\tau_1, \ldots, \tau_n]]\!]^\approx_A$,
$$\psi_{[\tau_1,\ldots,\tau_n]}(p) = \{(v_1, \ldots, v_n) \in [\![\tau_1]\!]_{A/\approx} \times \cdots \times [\![\tau_n]\!]_{A/\approx} \mid (\chi_{\tau_1}(v_1), \ldots, \chi_{\tau_n}(v_n)) \in p\}$$
for all $p \in [\![[\tau_1, \ldots, \tau_n]]\!]_{A/\approx}$,
$$\chi_{[\tau_1,\ldots,\tau_n]}(p) = \{(v_1, \ldots, v_n) \in [\![\tau_1]\!]^\approx_A \times \cdots \times [\![\tau_n]\!]^\approx_A \mid (\psi_{\tau_1}(v_1), \ldots, \psi_{\tau_n}(v_n)) \in p\}$$

These functions are well-defined. First, ψ and $\chi_{[\tau_1,\ldots,\tau_n]}$ are not affected by the choice of χ_b. Second, for all $v \in [\![\tau]\!]_{A/\approx}$, $\chi_\tau(v)$ respects $\approx$ (for base types b, we have $v \approx_b v$ for $v \in [\![b]\!]_{A/\approx}$; for bracket types, it follows from the fact that $v \approx_\tau v'$ implies $\psi_\tau(v) = \psi_\tau(v')$). We have for all $v \in [\![\tau]\!]_{A/\approx}$, $\psi_\tau(\chi_\tau(v)) = v$ and for all $v \in [\![\tau]\!]^{\approx}_A$, $\chi_\tau(\psi_\tau(v)) \approx_\tau v$, both by induction on the structure of τ (the latter uses again the fact that $v \approx_\tau v'$ implies $\psi_\tau(v) = \psi_\tau(v')$). The former implies that ψ_τ is onto for all τ. Note that $\psi_{[]}$ is the identity.

The remainder of the proof relies on the following:

Lemma *For any $\Gamma \vdash t : \tau$ and Γ-environment ρ w.r.t. $\approx$, $[\![t]\!]_{\psi\circ\rho,A/\approx} = \psi_\tau([\![t]\!]^{\approx}_{\rho,A})$.*

PROOF: *By induction on the structure of the derivation of $\Gamma \vdash t : \tau$. For function application, we use the fact that for $c : b_1 \times \cdots \times b_n \to b$ in C we have $\psi_b([\![c]\!]_A(v_1,\ldots,v_n)) = [\![c]\!]_{A/\approx}(\psi_{b_1}(v_1),\ldots,\psi_{b_n}(v_n))$. For universal quantification, we use the fact that ψ is onto.* □ *(lemma)*

This gives $A/\approx \models \varphi \Rightarrow A \models^{\approx} \varphi$, as follows. Suppose that $A/\approx \models \varphi$, i.e. $\Gamma \vdash \varphi : [\,]$ and for all Γ-environments ρ on $A/\approx$, $A/\approx \models_\rho \varphi$. Then suppose ρ is a Γ-environment on A w.r.t. $\approx$. We need to show that $A \models^{\approx}_\rho \varphi$. But $\psi \circ \rho$ is a Γ-environment on $A/\approx$, and so $[\![\varphi]\!]^{\approx}_{\rho,A} = \psi_{[]}([\![\varphi]\!]^{\approx}_{\rho,A}) = [\![\varphi]\!]_{\psi\circ\rho,A/\approx} = tt$.

For $A/\approx \models \varphi \Leftarrow A \models^{\approx} \varphi$, suppose that $A \models^{\approx} \varphi$, i.e. $\Gamma \vdash \varphi : [\,]$ and for all Γ-environments ρ w.r.t. $\approx$, $A \models^{\approx}_\rho \varphi$. Then suppose ρ is a Γ-environment on $A/\approx$. We need to show that $A/\approx \models_\rho \varphi$. Since ψ is onto, ρ factors through ψ: $\rho = \psi\circ\rho'$ for some Γ-environment ρ' w.r.t. $\approx$. Thus $[\![\varphi]\!]_{\psi\circ\rho',A/\approx} = \psi_{[]}([\![\varphi]\!]^{\approx}_{\rho',A}) = \psi_{[]}(tt) = tt$, i.e. $A/\approx \models_\rho \varphi$. □

A trivial consequence of Theorem 3.35 is the fact that when $\approx$ is equality, $\models^{\approx}$ coincides with $\models$. Theorem 3.35 for the case of infinitary first-order equational logic is Theorem 3.11 of [BHW95], where the proof method is analogous.

Example 3.36 From Example 3.10 we know that $Nat \models \Phi_{ctr}$, and then Corollary 3.14 and the fact (Example 3.34) that $Nat \cong Term^\infty/\approx$ for $\approx$ as given in Example 3.16 gives $Term^\infty/\approx \models \Phi_{ctr}$. Then Theorem 3.35 tells us that $Term^\infty \models^{\approx} \Phi_{ctr}$, which is what we concluded in Example 3.31. (And vice versa.) □

We believe that the above development would go through, *mutatis mutandis*, for Henkin models [Hen50] as well as in a constructive framework like that of topos theory [Pho92], see Section 3.9 of [TvD88]. In the absence of the axiom of choice, e.g. in topos theory, one must replace the function χ in the proof of Theorem 3.35 by a relation which is functional up to $\approx$.

We now consider specific definitions of indistinguishability and behavioural equivalence. Let $\Sigma = \langle B, C\rangle$ be a signature, and let OBS, the *observable base types* of Σ, be a subset of B. The intention is that OBS includes just those base types that are directly visible to clients; typically this would include types like *bool* and *nat*. All other types, including all bracket types, are *hidden* in the sense that their values may only be inspected indirectly by performing experiments (i.e. evaluating terms) that yield a result of a type in OBS.

The following defines the indistinguishability relation used in [NO88]. Two values v, v' are indistinguishable if no experiment of observable type with additional observable inputs is able to distinguish between them.

Definition 4.1 *Let the family of partial congruences $\approx_{OBS} = \langle \approx_{OBS,A}\rangle_{A\in Alg(\Sigma)}$ be such that for any Σ-algebra A, base type $b \in B$ and $v, v' \in [\![b]\!]_A$, $v \approx_{OBS,A,b} v'$ (v and v' are* indistinguishable*) iff v and v' are OBS-reachable, and for any OBS-context Γ, variable $x \notin Vars(\Gamma)$, term t with $\Gamma, x : b \vdash t : b'$ for $b' \in OBS$, and Γ-environment ρ, $[\![t]\!]_{\rho[x\mapsto v],A} = [\![t]\!]_{\rho[x\mapsto v'],A}$.*

Example 4.2 The partial congruence $\approx_{\{bool\},Nat}$ is equality, while $\approx_{\{bool\},Term^\infty}$, which is the same as $\approx_{\{bool\},Term}$, is $\approx$ from Example 3.16. Making all base types observable, as in $\approx_{\{bool,ctr\},Term}$, yields equality. The partial congruence $\approx_{\{ctr\},Term}$ is equality on *ctr* but is the total relation on *bool* since there are no terms over Σ_{ctr} of type *ctr* containing a variable of type *bool*. Providing no observable types, as in $\approx_{\emptyset,Term}$, yields the total relation for similar reasons. □

Proposition 4.3 *For any Σ-algebra A, $\approx_{OBS,A}$ is a partial congruence on A.*

Proof: *Symmetry and transitivity are obvious. Suppose $c : b_1 \times \cdots \times b_n \to b$ in C and $v_1 \approx_{OBS,A,b_1} v_1'$ and $\cdots$ and $v_n \approx_{OBS,A,b_n} v_n'$; we have to show that $[\![c]\!]_A(v_1, \ldots, v_n) \approx_{OBS,A,b} [\![c]\!]_A(v_1', \ldots, v_n')$. Since v_i, v_i' are OBS-reachable for all $1 \le i \le n$, there is an OBS-context Γ, Γ-environment ρ and terms t_i, t_i' in context Γ such that $v_i = [\![t_i]\!]_{\rho,A}$ and $v_i' = [\![t_i']\!]_{\rho,A}$, for all $1 \le i \le n$. (There is no loss of generality in taking the same Γ and ρ for all of these terms.) Now, let Γ' be an OBS-context, suppose that $\Gamma', x : b \vdash s : b'$ for $b' \in OBS$ and $x \notin Vars(\Gamma')$, and let ρ' be a Γ'-environment. W.l.o.g. we may assume that $Vars(\Gamma)$ and $Vars(\Gamma')$ are disjoint and that $x \notin Vars(\Gamma)$. We then calculate*

as follows:

$$
\begin{aligned}
&[\![s]\!]_{\rho'[x\mapsto[\![c]\!]_A(v_1,\ldots,v_n)],A} \\
&\quad = [\![s]\!]_{(\rho'+\rho)[x\mapsto[\![c(t_1,\ldots,t_n)]\!]_{\rho'+\rho,A}],A} && \textit{(using disjointness)} \\
&\quad = [\![s[x := c(t_1,\ldots,t_n)]]\!]_{\rho'+\rho,A} && \textit{(by Proposition 3.5)} \\
&\quad = [\![s[x := c(y,t_2,\ldots,t_n)]]\!]_{(\rho'+\rho)[y\mapsto v_1],A} && \textit{(using Proposition 3.5)} \\
&\quad = [\![s[x := c(y,t_2,\ldots,t_n)]]\!]_{(\rho'+\rho)[y\mapsto v'_1],A} && \textit{(since } v_1 \approx_{OBS,A,b_1} v'_1\textit{)} \\
&\quad = [\![s[x := c(t'_1,t_2,\ldots,t_n)]]\!]_{\rho'+\rho,A} && \textit{(using Proposition 3.5)} \\
&\quad = [\![s[x := c(t'_1,\ldots,t'_n)]]\!]_{\rho'+\rho,A} && \textit{(iterating the above three steps)} \\
&\quad = [\![s]\!]_{\rho'[x\mapsto[\![c]\!]_A(v'_1,\ldots,v'_n)],A} && \textit{(reversing the first two steps).}
\end{aligned}
$$

Since $[\![c]\!]_A(v_1,\ldots,v_n)$ *and* $[\![c]\!]_A(v'_1,\ldots,v'_n)$ *are clearly OBS-reachable, the claim follows.* □

Note that the above proof makes essential use of the fact that *OBS*-reachability of v and v' is required for $v \approx_{OBS,A,b} v'$ in Definition 4.1. This is no accident, as the following counterexample shows.

Counterexample 4.4 *Let the signature* Σ *have base types* b *and* $bool$ *and constants* $f : b \to bool$ *and* $c : b \times b \to b$*, and take* $OBS = \{bool\}$*. Consider the* Σ*-algebra* A *such that* $[\![b]\!]_A = \{0,1,2\}$*,* $[\![bool]\!]_A = \{T,F\}$*,* $[\![f]\!]_A(0) = [\![f]\!]_A(1) = T$*,* $[\![f]\!]_A(2) = F$*, and*

$$[\![c]\!]_A(x,y) = \begin{cases} x & \textit{if } x = y \\ 2 & \textit{if } x \neq y \end{cases}$$

Now suppose that the requirement of OBS-reachability were removed from Definition 4.1. Then $0 \approx_{OBS,A,b} 0$ *and* $0 \approx_{OBS,A,b} 1$ *but* $[\![c]\!]_A(0,0) \not\approx_{OBS,A,b} [\![c]\!]_A(0,1)$ *since* $[\![f]\!]_A([\![c]\!]_A(0,0)) \neq [\![f]\!]_A([\![c]\!]_A(0,1))$*.* □

By analogy with the terminology of denotational semantics (see e.g. [Win93]), a Σ-algebra A is called *fully abstract* when the indistinguishability relation on A is simply equality. Such an A is called an *algebra of minimal redundancy* in [Rei85].

Definition 4.5 ([BHW95]) *Let* $\approx = \langle \approx_A \rangle_{A \in Alg(\Sigma)}$ *be a family such that each* $\approx_A$ *is a partial congruence on* A*. A* Σ*-algebra* A *is* $\approx$-fully abstract *when* $\approx_A$ *is the equality in* A*, that is, when for all* $b \in B$ *and* $v, v' \in [\![b]\!]_A$ *we have* $v \approx_A v'$ *iff* $v = v'$*. For any class* $\mathcal{A} \subseteq Alg(\Sigma)$ *of* Σ*-algebras,* $FA_{\approx}(\mathcal{A}) \subseteq \mathcal{A}$ *is*

the subclass of $\approx$-fully abstract algebras, that is:

$$FA_{\approx}(\mathcal{A}) = \{A \in \mathcal{A} \mid A \textit{ is } \approx\textit{-fully abstract}\}.$$

The family $\approx$ is regular *if $A/{\approx_A}$ is $\approx$-fully abstract for every $A \in Alg(\Sigma)$.*

Example 4.6 *Nat* is $\approx_{\{bool\}}$-fully abstract, while *Term* and $Term^{\infty}$ are not. □

Regularity ensures that the partial congruences which $\approx$ associates with different algebras are related in a natural way.

Proposition 4.7 *$\approx_{OBS}$ is regular.*

PROOF: *We have to show that for all $b \in B$ and for all $v, v' \in [\![b]\!]_{A/\approx_{OBS,A}}$, $v \approx_{OBS,A/\approx_{OBS,A}} v'$ iff $v = v'$.*

$\Longleftarrow$: We only need to show that $v(= v')$ is OBS-reachable. This follows from the lemma in the proof of Theorem 3.35 and the fact that v is a congruence class of OBS-reachable values in A.

$\Longrightarrow$: Suppose that $v \approx_{OBS,A/\approx_{OBS,A}} v'$. Then for any OBS-context Γ, $x \notin Vars(\Gamma)$, term t such that $\Gamma, x : b \vdash t : b'$ for $b' \in OBS$ and Γ-environment ρ on $A/\approx_{OBS,A}$, $[\![t]\!]_{\rho[x\mapsto v],A/\approx_{OBS,A}} = [\![t]\!]_{\rho[x\mapsto v'],A/\approx_{OBS,A}}$. Let $\widehat{v}, \widehat{v'} \in [\![b]\!]_A$ be such that $[\widehat{v}]_{\approx_{OBS,A}} = v$ and $[\widehat{v'}]_{\approx_{OBS,A}} = v'$; by the lemma in the proof of Theorem 3.35, it follows that $[[\![t]\!]_{\rho'[x\mapsto\widehat{v}],A}]_{\approx_{OBS,A}} = [[\![t]\!]_{\rho'[x\mapsto\widehat{v'}],A}]_{\approx_{OBS,A}}$ for any Γ-environment ρ' on A, because for any ρ there is a ρ' such that $\rho = [\cdot]_{\approx_{OBS,A}} \circ \rho'$, and vice versa since Γ is an OBS-context so $\rho'(x)$ is trivially OBS-reachable for any $x \in Vars(\Gamma)$. But $\approx_{OBS,A}$ on $b' \in OBS$ is equality so $\widehat{v} \approx_{OBS,A} \widehat{v'}$, i.e. $v = v'$. □

A slightly weaker property than regularity is required to state the main characterization theorem from [BHW95].

Definition 4.8 ([BHW95]) *Let $\approx = \langle \approx_A \rangle_{A \in Alg(\Sigma)}$ be a family such that each $\approx_A$ is a partial congruence on A. The family $\approx$ is* weakly regular *if $A/{\approx_A} \cong (A/{\approx_A})/{\approx_{(A/\approx_A)}}$ for every $A \in Alg(\Sigma)$.*

Proposition 4.9 ([BHW95]) *If $\approx$ is regular then it is weakly regular.*

PROOF: *From the definitions.* □

We will now define what it means for two Σ-algebras to be behaviourally equivalent. The definition resembles that of indistinguishability in the sense that it is based on the idea of performing experiments to probe for differences

between the two algebras. But in this case performing an experiment means testing satisfaction of a formula rather than evaluating a term of base type, and here we are comparing algebras rather than values within a single algebra. The formulae of importance are equations between terms of observable type, with variables of observable type.

Definition 4.10 *Let Γ be an OBS-context. An* observable equation *is a formula in context Γ of the form $t =_b t'$ where $b \in OBS$. Let $ObsEq_\Gamma(\Sigma)$ be the set of observable equations in context Γ.*

Definition 4.11 *Let $A, A' \in Alg(\Sigma)$. A is* behaviourally equivalent *to A' (*via equations*), written $A \equiv_{OBS} A'$, if there is an OBS-context Γ and Γ-environments ρ_A on A and $\rho_{A'}$ on A' that are OBS-surjective (see Definition 3.3) such that for any equation $\varphi \in ObsEq_\Gamma(\Sigma)$, $A \models_{\rho_A} \varphi$ iff $A' \models_{\rho_{A'}} \varphi$.*

Example 4.12 *$Nat \equiv_{\{bool\}} Term \equiv_{\{bool\}} Term^\infty$.* □

Proposition 4.13 *$\equiv_{OBS} \subseteq Alg(\Sigma) \times Alg(\Sigma)$ is an equivalence relation.*

PROOF: *Reflexivity and symmetry are obvious. Transitivity follows from the observation that the choice of variable names in the context Γ is arbitrary.* □

It might seem surprising that the definition of $\equiv_{OBS}$ does not make use of the higher-order features of the language, except as a result of the way that equality is expressed via quantification over predicates. So $\equiv_{OBS}$ is just the same as in e.g. [SW83], [MG85], [NO88]. The reason for this choice is that the natural modification of the definition of $\equiv_{OBS}$ to make use of higher-order formulae (Definition 5.19) gives exactly the same relation, see Corollary 5.22.

The following definition is the key to understanding the relationship between indistinguishability of values on the one hand and behavioural equivalence of algebras on the other. The idea is that a family of partial congruences naturally induces an equivalence on $Alg(\Sigma)$. If behavioural equivalence is the relation that is induced by indistinguishability (as will turn out to be the case, see Theorem 5.21) then it is possible to translate constructions phrased in terms of behavioural equivalence into constructions phrased in terms of indistinguishability, and vice versa. There is a close analogy with the case of finite state machines, where two machines M, M' are equivalent if quotienting M and M' by the so-called *Nerode equivalence* on states yields isomorphic machines.

Definition 4.14 ([BHW95]) *Let $\approx = \langle \approx_A \rangle_{A \in Alg(\Sigma)}$ be a family such that each $\approx_A$ is a partial congruence on A, and let $\equiv \subseteq Alg(\Sigma) \times Alg(\Sigma)$ be an equivalence relation. Then $\equiv$ is* factorizable *by $\approx$ if for any $A, A' \in Alg(\Sigma)$,*

$A \equiv A'$ iff $A/{\approx_A} \cong A'/{\approx_{A'}}$. (Factorizability can be decomposed as follows: $\equiv$ is left-factorizable *by $\approx$ if for any $A, A' \in Alg(\Sigma)$, $A \equiv A' \Leftarrow A/{\approx_A} \cong A'/{\approx_{A'}}$, and it is* right-factorizable *by $\approx$ if for any $A, A' \in Alg(\Sigma)$, $A \equiv A' \Rightarrow A/{\approx_A} \cong A'/{\approx_{A'}}$.)*

The following proposition gives right-factorizability of $\equiv_{OBS}$ by $\approx_{OBS}$. Left-factorizability can be proved directly, but we obtain it instead by applying a more general result, see Corollary 5.16 below.

Proposition 4.15 *For any $A, A' \in Alg(\Sigma)$, if $A \equiv_{OBS} A'$ then $A/{\approx_{OBS,A}} \cong A'/{\approx_{OBS,A'}}$.*

PROOF: *Suppose that $A \equiv_{OBS} A'$ via OBS-context Γ and Γ-environments ρ_A on A and $\rho_{A'}$ on A'. The proof uses the following:*

Lemma *Let t, t' be terms such that $\Gamma \vdash t : b$ and $\Gamma \vdash t' : b$. Then $[\![t]\!]_{\rho_A,A} \approx_{OBS,A,b} [\![t']\!]_{\rho_A,A}$ iff $[\![t]\!]_{\rho_{A'},A'} \approx_{OBS,A',b} [\![t']\!]_{\rho_{A'},A'}$.*

PROOF:

$\Longrightarrow$: Suppose that $[\![t]\!]_{\rho_A,A} \approx_{OBS,A,b} [\![t']\!]_{\rho_A,A}$. We know that $[\![t]\!]_{\rho_{A'},A'}$ and $[\![t']\!]_{\rho_{A'},A'}$ are OBS-reachable since Γ is an OBS-context. Let Γ' be an OBS-context, $x \notin Vars(\Gamma')$ be a variable, s be a term with $\Gamma', x : b \vdash s : b'$ for $b' \in OBS$, and ρ be a Γ-environment on A'. We need to show that $[\![s]\!]_{\rho[x \mapsto [\![t]\!]_{\rho_{A'},A']},A'} = [\![s]\!]_{\rho[x \mapsto [\![t']\!]_{\rho_{A'},A']},A'}$. W.l.o.g. (since $\rho_{A'}$ is OBS-surjective) we can restrict attention to the case where $\Gamma = \Gamma'$ and $\rho = \rho_{A'}$; then by Proposition 3.5 it suffices to show that $[\![s[x := t]]\!]_{\rho_{A'},A'} = [\![s[x := t']]\!]_{\rho_{A'},A'}$. Now consider the observable equation $\varphi =_{\text{def}} s[x := t] =_{b'} s[x := t']$. We have $A \models_{\rho_A} \varphi$ since $[\![t]\!]_{\rho_A,A} \approx_{OBS,A,b} [\![t']\!]_{\rho_A,A}$; then $A' \models_{\rho_{A'}} \varphi$ since $A \equiv_{OBS} A'$. Hence $[\![s[x := t]]\!]_{\rho_{A'},A'} = [\![s[x := t']]\!]_{\rho_{A'},A'}$ by Proposition 3.7.

$\Longleftarrow$: Similarly. $\square$ *(lemma)*

Now define a function $h : A/{\approx_{OBS,A}} \to A'/{\approx_{OBS,A'}}$ by taking $h([v]_{\approx_{OBS,A}}) = [[\![t]\!]_{\rho_{A'},A']}_{\approx_{OBS,A'}}$ for $b \in B$ and $v \in [\![b]\!]_A$, where t is a term in context Γ such that $[\![t]\!]_{\rho_A,A} = v$. We know that such a t exists because v is OBS-reachable by definition of $\approx_{OBS,A}$ and because ρ_A is OBS-surjective. To see that the choice of the term t and the representative v don't matter, suppose we have terms t, t' in context Γ such that $[\![t]\!]_{\rho_A,A} = v \approx_{OBS,A,b} v' = [\![t']\!]_{\rho_A,A}$; then $[\![t]\!]_{\rho_{A'},A'} \approx_{OBS,A',b} [\![t']\!]_{\rho_{A'},A'}$ by the lemma. Thus h is well-defined, and it is easy to see that h is a Σ-homomorphism. Since our signatures do not contain predicate symbols, h is a Σ-isomorphism if we can prove that it is bijective.

To see that h is surjective, consider any $b \in B$ and representative v of any congruence class in $[\![b]\!]_{A'/{\approx_{OBS,A'}}}$. Pick a term t in context Γ such that $[\![t]\!]_{\rho_{A'},A'} = v$; we know that such a t exists because v is OBS-reachable (since $v \approx_{OBS,A'} v$)

and because $\rho_{A'}$ is OBS-surjective. Then $[v]_{\approx_{OBS,A'}} = h([[\![t]\!]_{\rho_A,A}]_{\approx_{OBS,A}})$.

To see that h is injective, suppose there are $b \in B$ and $v, v' \in [\![b]\!]_A$ such that $h([v]_{\approx_{OBS,A}}) = h([v']_{\approx_{OBS,A}})$, where t, t' are terms in context Γ such that $[\![t]\!]_{\rho_A,A} = v$ and $[\![t']\!]_{\rho_A,A} = v'$. Then $[\![t]\!]_{\rho_{A'},A'} \approx_{OBS,A'} [\![t']\!]_{\rho_{A'},A'}$, so by the lemma we have $[\![t]\!]_{\rho_A,A} \approx_{OBS,A} [\![t']\!]_{\rho_A,A}$, i.e. $[v]_{\approx_{OBS,A}} = [v']_{\approx_{OBS,A}}$.

Thus we have shown that $h : A/\approx_{OBS,A} \to A'/\approx_{OBS,A'}$ is a Σ-isomorphism. Note that its inverse can be obtained from the proof of surjectivity. □

Example 4.16 Since $Nat \equiv_{\{bool\}} Term \equiv_{\{bool\}} Term^\infty$ (Example 4.12), by Proposition 4.15 $Nat/\approx_{\{bool\},Nat} \cong Term/\approx_{\{bool\},Term} \cong Term^\infty/\approx_{\{bool\},Term^\infty}$ (Example 3.34, see also Example 4.2). □

Proposition 4.15 essentially amounts to one direction of Example 5.4 of [BHW95], where the proof method is the same.

In this paper, we consider only the particular definitions of indistinguishability (Definition 4.1) and behavioural equivalence (Definition 4.11) given above. There are at least two other candidates for each of these definitions, as described in [BHW95]. The first variant, which has been studied by [Rei85], is obtained by allowing Γ to be an arbitrary B-context in both definitions, removing the requirement of *OBS*-reachability in Definition 4.1, and changing the requirement of *OBS*-surjectivity to B-surjectivity in Definition 4.11. The second variant is obtained by eliminating the context Γ and environments from both definitions, and changing the requirement of *OBS*-reachability to $\emptyset$-reachability in Definition 4.1; the resulting definition of behavioural equivalence has been studied in Section 2 of [ST87]. These alternatives are not studied here, although all of the proofs required should be similar to those given here. In our opinion, the first variant is simply wrong because the resulting behavioural equivalence relation fails to identify algebras that differ only in their behaviour on values of non-observable types that are not *OBS*-reachable: see [ONS91] for an example. The second variant seems to be unnecessarily restrictive in the presence of parameterised specifications, since (as discussed in [ST89]) *OBS* will normally include the parameter types and these types typically lack generators; this leads to a behavioural equivalence relation that is too coarse.

Schoett [Sch90] has shown that $A \equiv_{OBS} A'$ iff there exists an *OBS-correspondence* between A and A' (a family of relations $\langle \leftrightarrow_b \subseteq [\![b]\!]_A \times [\![b]\!]_{A'} \rangle_{b \in B}$ such that for all $c : b_1 \times \cdots \times b_n \to b$ in C and all $v_1 \in [\![b_1]\!]_A, \ldots, v_n \in [\![b_n]\!]_A$ and $v_1' \in [\![b_1]\!]_{A'}, \ldots, v_n' \in [\![b_n]\!]_{A'}$, if $v_1 \leftrightarrow_{b_1} v_1'$ and $\cdots$ and $v_n \leftrightarrow_{b_n} v_n'$ then $[\![c]\!]_A(v_1, \ldots, v_n) \leftrightarrow_b [\![c]\!]_{A'}(v_1', \ldots, v_n')$, and such that $\leftrightarrow_b$ is a bijection for $b \in OBS$).[4] This characterization is useful for proving that specific algebras are

[4] In fact, the definition of $A \equiv_{OBS} A'$ in [Sch90] requires $[\![b]\!]_A = [\![b]\!]_{A'}$ for all $b \in$

behaviourally equivalent. Very recently, [BT96] has generalized this result. First, they consider an arbitrary concrete category of models, rather than that of ordinary algebras, and study the concepts of behavioural satisfaction and behavioural equivalence in this context. They then generalize the characterization theorem of [Sch90] to the case of arbitrary $\approx$ and $\equiv$ (satisfying certain technical conditions) such that $\equiv$ is factorizable by $\approx$. They also generalize the characterization theorem of [BHW95] to this context.

5 Expressible congruences and relativization

The language of higher-order logic is powerful enough to express the indistinguishability relation $\approx_{OBS}$ by means of a family of predicates, i.e. terms in the language (cf. [Sch94]). We can use this fact to characterize behavioural satisfaction of a formula φ in terms of ordinary satisfaction of a "relativized" version of φ.

Let $\Sigma = \langle B, C\rangle$ be a signature for which B and C are finite and let $OBS \subseteq B$. The assumption of finiteness is required to obtain finite terms in Proposition 5.1 and Theorem 5.4 below.

Notations like $\bigwedge_{b\in B} x_b{:}b.t$ and $\lambda(\langle P_b{:}[b,b]\rangle_{b\in B}).t$ will be used below to abbreviate obvious (finite) terms. The latter assumes some fixed enumeration of the elements of B; this is not needed for the former since a sequence of universal quantifiers can be permuted without affecting meaning.

Proposition 5.1 *Let A be a Σ-algebra and suppose $\widehat{v} \in [\![\widehat{b}]\!]_A$ for $\widehat{b} \in B$. Then $A \models_{[x\mapsto\widehat{v}]} REACH_{\widehat{b}}(x)$ iff $\widehat{v}$ is OBS-reachable, where $REACH_{\widehat{b}}$ is defined as follows:*

If $\widehat{b} \in OBS$ then $REACH_{\widehat{b}} =_{\mathrm{def}} \lambda(x{:}\widehat{b}).\mathsf{true}$.

OBS, and so there $\leftrightarrow_b$ is required to be equality for $b \in OBS$. Moreover, Schoett uses *partial* algebras where the definition of $\equiv_{OBS}$ is a little more complicated and correspondences are required to satisfy additional conditions.

If $\widehat{b} \notin OBS$ then

$$REACH_{\widehat{b}} =_{\text{def}}$$
$$\lambda(x{:}\widehat{b}).\bigwedge_{b \notin OBS} P_b : [b].$$
$$\left(\bigwedge_{\substack{c:b_1\times\cdots\times b_n\to b' \text{ in } C \\ b'\notin OBS}} \bigwedge_{1\le i\le n} x_i : b_i.\left(\bigwedge_{\substack{1\le j\le n \\ b_j\notin OBS}} P_{b_j}(x_j)\right) \Rightarrow P_{b'}(c(x_1,\ldots,x_n))\right)$$
$$\Rightarrow P_{\widehat{b}}(x)$$

PROOF: *If $\widehat{b} \in OBS$ then the proof is trivial. So suppose that $\widehat{b} \notin OBS$.*

$\Longrightarrow$: Instantiate $REACH_{\widehat{b}}(x)$ with $P_b = \{v \in [\![b]\!]_A \mid v \text{ is } OBS\text{-reachable}\} \in [\![[b]]\!]_A$ for all $b \notin OBS$. It then suffices to show that the closure property on the left-hand side of the main implication is satisfied. This is easy: for each $c : b_1 \times \cdots \times b_n \to b'$ in C, the required term is simply the application of c to the terms that witness the OBS-reachability of $x_1, \ldots, x_n$.

$\Longleftarrow$: Suppose that $\widehat{v}$ is OBS-reachable; then there is an OBS-context Γ, term t with $\Gamma \vdash t : \widehat{b}$, and Γ-environment ρ such that $[\![t]\!]_{\rho,A} = \widehat{v}$. We need to show that $A \models_{[x\mapsto\widehat{v}]} REACH_{\widehat{b}}(x)$; by Proposition 3.5 it suffices to show that $A \models_\rho REACH_{\widehat{b}}(t)$. This follows by induction on the structure of t. □

Example 5.2 Consider the signature Σ_{ctr} and let $OBS = \{bool\}$; then $REACH_{bool}$ is $\lambda(x{:}bool).\mathsf{true}$ and $REACH_{ctr}$ is:

$$\lambda(x{:}ctr).\forall P{:}[ctr].$$
$$(P(zero) \wedge \forall y{:}ctr.P(y) \Rightarrow P(inc(y)) \wedge \forall z{:}ctr.P(z) \Rightarrow P(dec(z)))$$
$$\Rightarrow P(x)$$

□

Definition 5.3 *Let $\approx = \langle \approx_A \rangle_{A\in Alg(\Sigma)}$ be a family of partial congruences, and let $\sim = \langle \sim_b \rangle_{b\in B}$ be a family of closed predicates such that $\vdash \sim_b : [b, b]$ for every base type $b \in B$. Then $\approx$ is* expressible by *$\sim$ if $[\![\sim_b]\!]_A = \approx_{A,b}$ for every $b \in B$.*

The formulae in the following theorem do not make easy reading. $INDIST_{\widehat{b}}$ characterizes $\approx_{OBS}$ as the greatest $(\exists_{b\in B} P_b{:}[b, b].P_{\widehat{b}}(x, y) \wedge \cdots)$ partial congruence $(CONG(\langle P_b\rangle_{b\in B}))$ that is equality on OBS $(OBSEQ(\langle P_b\rangle_{b\in B}))$ and is defined only for OBS-reachable values $(REACH_{\widehat{b}}(x) \wedge REACH_{\widehat{b}}(y))$.

Theorem 5.4 *The indistinguishability relation $\approx_{OBS}$ is expressible by the*

family of predicates $\langle INDIST_{\widehat{b}}\rangle_{\widehat{b}\in B}$, *defined as follows:*

$$CONG =_{\text{def}}$$
$$\lambda(\langle P_b{:}[b,b]\rangle_{b\in B}).$$
$$\bigwedge_{c:b_1\times\cdots\times b_n\to b \text{ in } C} \ \forall_{1\le i\le n} x_i, x_i'{:}b_i.\Big(\bigwedge_{1\le j\le n} P_{b_j}(x_j,x_j')\Big)$$
$$\Rightarrow P_b(c(x_1,\ldots,x_n),c(x_1',\ldots,x_n'))$$

$$OBSEQ =_{\text{def}} \lambda(\langle P_b{:}[b,b]\rangle_{b\in B}).\bigwedge_{b\in OBS}\forall x,x'{:}b.\ P_b(x,x') \Leftrightarrow x =_b x'$$

$$INDIST_{\widehat{b}} =_{\text{def}}$$
$$\lambda(x{:}\widehat{b},y{:}\widehat{b}).\exists_{b\in B} P_b{:}[b,b].P_{\widehat{b}}(x,y) \wedge CONG(\langle P_b\rangle_{b\in B}) \wedge OBSEQ(\langle P_b\rangle_{b\in B})$$
$$\wedge REACH_{\widehat{b}}(x) \wedge REACH_{\widehat{b}}(y)$$

PROOF: *Let* A *be a* Σ*-algebra. We need to prove that if* $\widehat{v},\widehat{v'} \in [\![\widehat{b}]\!]_A$, *then* $A \models_{[x\mapsto\widehat{v},y\mapsto\widehat{v'}]} INDIST_{\widehat{b}}(x,y)$ *iff* $\widehat{v} \approx_{OBS,A,\widehat{b}} \widehat{v'}$.

$\Longleftarrow$: *Suppose that* $\widehat{v} \approx_{OBS,A,\widehat{b}} \widehat{v'}$. *We claim that* $A \models_{[x\mapsto\widehat{v},y\mapsto\widehat{v'}]} INDIST_{\widehat{b}}(x,y)$ *with the predicates* $P_b = \approx_{OBS,A,b} \in [\![[b,b]]\!]_A$ *for all* $b \in B$. *By the assumption we have that* $A \models_{[x\mapsto\widehat{v},y\mapsto\widehat{v'}]} P_{\widehat{b}}(x,y)$; *then* $A \models CONG(\langle P_b\rangle_{b\in B})$ *since* $\approx_{OBS,A}$ *is a partial congruence on* A *(Proposition 4.3),* $A \models OBSEQ(\langle P_b\rangle_{b\in B})$ *by the definition of* $\approx_{OBS,A}$, *and* $A \models_{[x\mapsto\widehat{v},y\mapsto\widehat{v'}]} REACH_{\widehat{b}}(x) \wedge REACH_{\widehat{b}}(y)$ *by Proposition 5.1.*

$\Longrightarrow$: *Suppose that* $A \models_{[x\mapsto\widehat{v},y\mapsto\widehat{v'}]} INDIST_{\widehat{b}}(x,y)$. *Then* $\widehat{v}$ *and* $\widehat{v'}$ *are OBS-reachable by Proposition 5.1. It remains to show that if* Γ *is an OBS-context,* $z \notin Vars(\Gamma)$, s *is a term such that* $\Gamma, z:\widehat{b} \vdash s : b'$ *for* $b' \in OBS$, *and* ρ *is a* Γ*-environment, then* $[\![s]\!]_{\rho[z\mapsto\widehat{v}],A} = [\![s]\!]_{\rho[z\mapsto\widehat{v'}],A}$. *This is a consequence of the following lemma:*

Lemma *For any OBS-context* Γ, *variable* $z \notin Vars(\Gamma)$, *term* s *such that* $\Gamma, z:\widehat{b} \vdash s : b'$, *and* Γ*-environment* ρ, $([\![s]\!]_{\rho[z\mapsto\widehat{v}],A}, [\![s]\!]_{\rho[z\mapsto\widehat{v'}],A}) \in P_{b'}$.

PROOF: *By induction on the structure of* s. *Suppose that* s *is a variable but not* z; *then* $b' \in OBS$ *since* Γ *is an OBS-context, and the required property follows from the fact that* $A \models OBSEQ(\langle P_b\rangle_{b\in B})$. *Suppose that* s *is* z; *then the required property follows directly from the fact that* $A \models_{[x\mapsto\widehat{v},y\mapsto\widehat{v'}]} P_{\widehat{b}}(x,y)$. *Suppose that* s *is a function application; then the required property follows from the inductive assumption and the fact that* $A \models CONG(\langle P_b\rangle_{b\in B})$. $\square$ *(lemma)*

From this, together with the fact that $A \models OBSEQ(\langle P_b \rangle_{b \in B})$ *and Proposition 3.7, it follows that* $[\![s]\!]_{\rho[z \mapsto \widehat{v}],A} = [\![s]\!]_{\rho[z \mapsto \widehat{v'}],A}$ *when* $b' \in OBS$. □

Example 5.5 Consider the signature Σ_{ctr} and let $OBS = \{bool\}$; then $INDIST_{bool}$ simplifies to $\lambda(x{:}bool, y{:}bool).x =_{bool} y$ and $INDIST_{ctr}$ simplifies to:

$$\begin{aligned}
&\lambda(x{:}ctr, y{:}ctr).\\
&\quad REACH_{ctr}(x) \wedge REACH_{ctr}(y)\\
&\quad \wedge \exists P{:}[ctr, ctr].P(zero, zero)\\
&\qquad\qquad \wedge \forall a, b{:}ctr.\, P(a,b) \Rightarrow\ P(inc(a), inc(b))\\
&\qquad\qquad\qquad\qquad \wedge P(dec(a), dec(b))\\
&\qquad\qquad\qquad\qquad \wedge \mathit{is\text{-}zero}(a) =_{bool} \mathit{is\text{-}zero}(b)\\
&\qquad\qquad \wedge P(x,y)
\end{aligned}$$

This can be used to show that $zero \approx_{\{bool\}, Term} dec(inc(zero))$ (and similarly for $\approx_{\{bool\}, Term^\infty}$) by taking

$$\begin{aligned}
P = \{(v, v') \mid\ & v = v' \vee \exists t.\ x : ctr \vdash t : ctr\\
& \wedge v = t[zero] \wedge v' = t[dec(inc(zero))]\}.
\end{aligned}$$

where $t[t']$ is shorthand for $[\![t]\!]_{[x \mapsto t'], Term}$. This choice of P works for showing that $t[zero] \approx_{\{bool\}, Term} t[dec(inc(zero))]$ for any "context" t, but a different choice of P is required for showing that $dec(inc(zero)) \approx_{\{bool\}, Term} zero$ or $zero \approx_{\{bool\}, Term} dec(dec(inc(inc(zero))))$. □

In [Sch94] an expressibility result analogous to Theorem 5.4 for the indistinguishability relation used in [Rei85] is given for a language of second-order logic. Detailed comparisons are rendered difficult by the fact that the logic used there is untyped. Very recently, [BT96] has shown that regularity (Proposition 4.7 above), among other properties of $\approx_{OBS}$, follows from the characterization of $\approx_{OBS}$ as the greatest partial congruence that is equality on OBS and is defined only for OBS-reachable values.

Let $\approx = \langle \approx_A \rangle_{A \in Alg(\Sigma)}$ be a family of partial congruences that is expressible by the family of predicates $\sim = \langle \sim_b \rangle_{b \in B}$.

Definition 3.17 showed how to extend a partial congruence to bracket types. We can express exactly the same thing for any expressible congruence.

Proposition 5.6 *For any type* τ *there is a closed predicate* $\sim_\tau$ *such that*

$\vdash \sim_\tau : [\tau, \tau]$ *and* $[\![\sim_\tau]\!]_A = \approx_{A,\tau}$*, given by the following definition:*

If $\tau = b \in B$ *then* $\sim_\tau =_{\text{def}} \sim_b$*.*

If $\tau = [\tau_1, \ldots, \tau_n]$ *then*

$$\sim_\tau =_{\text{def}} \lambda(p{:}[\tau_1, \ldots, \tau_n], p'{:}[\tau_1, \ldots, \tau_n]). \bigwedge_{1 \le i \le n} x_i, x_i'{:}\tau_i. \left(\bigwedge_{1 \le j \le n} x_j \sim_{\tau_j} x_j' \right) \Rightarrow (p(x_1, \ldots, x_n) \Leftrightarrow p'(x_1', \ldots, x_n'))$$

(The definition of $\sim_{[\tau_1,\ldots,\tau_n]}$ *is recursive, but the result is a finite term for any type* $[\tau_1, \ldots, \tau_n]$*.)*

PROOF: *Immediate.* □

This leads directly to a family of predicates characterizing the values that are in the interpretation of types w.r.t. $\approx$.

Proposition 5.7 *For any type* τ *there is a closed predicate* DOM_τ *such that* $\vdash DOM_\tau : [\tau]$ *and* $[\![DOM_\tau]\!]_A = [\![\tau]\!]_A^{\approx_A}$*, given by the following definition:*

If $\tau = b \in B$ *then* $DOM_\tau =_{\text{def}} \lambda(x{:}b).x \sim_b x$*.*

If $\tau = [\tau_1, \ldots, \tau_n]$ *then*

$$DOM_\tau =_{\text{def}} \lambda(p{:}[\tau_1, \ldots, \tau_n]). p \sim_{[\tau_1,\ldots,\tau_n]} p \wedge \bigwedge_{1 \le i \le n} x_i{:}\tau_i. \left(p(x_1, \ldots, x_n) \Rightarrow \bigwedge_{1 \le j \le n} DOM_{\tau_j}(x_j) \right)$$

(Again, this is a recursive definition that gives a finite term for any type.)

PROOF: *By induction on the structure of* τ*, using Propositions 3.22 and 5.6.* □

We can use the predicates DOM_τ thus defined to transform any formula φ into a formula $\ulcorner \varphi \urcorner$ such that $\ulcorner \varphi \urcorner$ is satisfied exactly when φ is satisfied w.r.t. $\approx$. The idea is simply to "relativize" each bound variable by attaching a requirement that the value taken on by the variable is in the interpretation of its type w.r.t. $\approx$.

Definition 5.8 *Let* t *be a term in context* Γ*. The* $\sim$-relativization *of* t *is the*

term $\ulcorner t \urcorner$ *(in context* Γ*) defined as follows:*

$$\begin{aligned}
\ulcorner x \urcorner &= x \\
\ulcorner c(t_1, \ldots, t_n) \urcorner &= c(\ulcorner t_1 \urcorner, \ldots, \ulcorner t_n \urcorner) \\
\ulcorner \lambda(x_1{:}\tau_1, \ldots, x_n{:}\tau_n).t \urcorner &= \lambda(x_1{:}\tau_1, \ldots, x_n{:}\tau_n). \\
&\qquad DOM_{\tau_1}(x_1) \wedge \cdots \wedge DOM_{\tau_n}(x_n) \wedge \ulcorner t \urcorner \\
\ulcorner t(t_1, \ldots, t_n) \urcorner &= \ulcorner t \urcorner(\ulcorner t_1 \urcorner, \ldots, \ulcorner t_n \urcorner) \\
\ulcorner t \Rightarrow t' \urcorner &= \ulcorner t \urcorner \Rightarrow \ulcorner t' \urcorner \\
\ulcorner \forall x{:}\tau.t \urcorner &= \forall x{:}\tau.DOM_\tau(x) \Rightarrow \ulcorner t \urcorner
\end{aligned}$$

The following results relate satisfaction of a formula to satisfaction of its relativized version.

Theorem 5.9 *Let* A *be a* Σ*-algebra,* $\Gamma \vdash t : \tau$ *and let* ρ *be a* Γ*-environment w.r.t.* $\approx_A$ *(so* ρ *is also an ordinary* Γ*-environment by Proposition 3.22). Then* $[\![t]\!]^{\approx_A}_{\rho,A} \approx_{A,\tau} [\![\ulcorner t \urcorner]\!]_{\rho,A}$.

Proof: *By induction on the structure of the derivation of* $\Gamma \vdash t : \tau$*. For* λ*-abstraction, we use Propositions 3.27 and 5.7. For universal quantification, we use Proposition 5.7.* □

Corollary 5.10 *Let* A *be a* Σ*-algebra, let* φ *be a formula in context* Γ *and let* ρ *be a* Γ*-environment w.r.t.* $\approx_A$*. Then* $A \models^{\approx_A}_\rho \varphi$ *iff* $A \models_\rho \ulcorner \varphi \urcorner$.

Proof: *Immediate from Theorem 5.9.* □

Corollary 5.11 *Let* A *be a* Σ*-algebra,* $v, v' \in [\![\tau]\!]^{\approx}_A$ *for some* τ*, and* ρ *be an environment w.r.t.* $\approx_A$*. Then* $[\![\ulcorner x =_\tau y \urcorner]\!]_{\rho[x \mapsto v, y \mapsto v'], A} = [\![x \sim_\tau y]\!]_{\rho[x \mapsto v, y \mapsto v'], A}$.

Proof: $[\![\ulcorner x =_\tau y \urcorner]\!]_{\rho[x \mapsto v, y \mapsto v'], A} = [\![x =_\tau y]\!]^{\approx_A}_{\rho[x \mapsto v, y \mapsto v'], A}$ *(by Theorem 5.9, since* $\approx_{A,[]}$ *is equality)* $= (v \approx_{A,\tau} v')$ *(by Proposition 3.29)* $= [\![x \sim_\tau y]\!]_{\rho[x \mapsto v, y \mapsto v'], A}$ *(by Proposition 5.6).* □

Example 5.12 Let φ be the Σ_{ctr}-formula $\forall c{:}ctr.dec(inc(c)) =_{ctr} c$; then $\ulcorner \varphi \urcorner$ is $\forall c{:}ctr.DOM_{ctr}(c) \Rightarrow \forall P{:}[ctr].DOM_{[ctr]}(P) \Rightarrow P(dec(inc(c))) \Rightarrow P(c)$. This is equivalent to $\forall c{:}ctr.DOM_{ctr}(c) \Rightarrow INDIST_{ctr}(dec(inc(c)), c)$ by Corollary 5.11. Let $OBS = \{bool\}$. In $Term^\infty$, the effect of restricting to values c such that $DOM_{ctr}(c)$ is to ignore ∞, and the effect of restricting to predicates P such that $DOM_{[ctr]}(P)$ is to consider only predicates that respect $\approx_{\{bool\}, Term^\infty}$, disregarding e.g. P in Example 3.18. Since $Term^\infty \models^{\approx} \varphi$ (Example 3.31), Corollary 5.10 says that $Term^\infty \models \ulcorner \varphi \urcorner$. □

The definition of the $\sim$-relativization of a formula is closely related to the definition of "lifted" formula in [BH96], and Corollary 5.10 is a higher-order version of Theorem 4.2(i) in [BH96].

Corollary 5.13 *Let A, A' be Σ-algebras such that $A/{\approx_A} \cong A'/{\approx_{A'}}$, and let φ be a closed formula. Then $A \models \ulcorner\varphi\urcorner$ iff $A' \models \ulcorner\varphi\urcorner$.*

PROOF: *$A \models \ulcorner\varphi\urcorner$ iff $A \models^{\approx_A} \varphi$ (by Corollary 5.10, since φ is closed) iff $A/{\approx_A} \models \varphi$ (by Theorem 3.35) iff $A'/{\approx_{A'}} \models \varphi$ (by Corollary 3.14) iff $A' \models^{\approx_{A'}} \varphi$ iff $A' \models \ulcorner\varphi\urcorner$.* □

The relativization construction may be used to define another behavioural equivalence relation, in which two algebras are regarded as behaviourally equivalent provided they cannot be distinguished by relativized formulae. The motivation for this apparent departure from our earlier notion of behavioural equivalence is that it is a convenient technical device for proving left-factorizability of $\equiv_{OBS}$ by $\approx_{OBS}$ (Corollary 5.16), since this follows directly from left-factorizability of this new relation by $\approx_{OBS}$ (Theorem 5.15). In fact, it will turn out (Corollary 5.22) that this "new" relation coincides with $\equiv_{OBS}$.

Definition 5.14 *Let $A, A' \in Alg(\Sigma)$. A is* behaviourally equivalent *to A'* via relativized formulae*, written $A \equiv_{RelForm} A'$, if there is an OBS-context Γ and Γ-environments ρ_A on A and $\rho_{A'}$ on A' that are OBS-surjective such that for any formula φ in context Γ, $A \models_{\rho_A} \ulcorner\varphi\urcorner$ iff $A' \models_{\rho_{A'}} \ulcorner\varphi\urcorner$, where $\ulcorner\varphi\urcorner$ is the $\langle INDIST_b\rangle_{b\in B}$-relativization of φ.*

Theorem 5.15 *For any $A, A' \in Alg(\Sigma)$, if $A/{\approx_{OBS,A}} \cong A'/{\approx_{OBS,A'}}$ then $A \equiv_{RelForm} A'$.*

PROOF: *Let $h : A/{\approx_{OBS,A}} \to A'/{\approx_{OBS,A'}}$ be an isomorphism. Since $\approx_{OBS}$ is equality on $b \in OBS$, we have a bijection $\widehat{h}_b : [\![b]\!]_A \to [\![b]\!]_{A'}$ for $b \in OBS$ defined by $[\widehat{h}_b(v)]_{\approx_{OBS,A',b}} = h_b([v]_{\approx_{OBS,A,b}})$ for $v \in [\![b]\!]_A$.*

Let Γ be the OBS-context such that $\Gamma_b = [\![b]\!]_A$ for every $b \in OBS$ (w.l.o.g. we assume that $[\![b]\!]_A \subseteq X$ and that $[\![b]\!]_A$ and $[\![b']\!]_A$ are disjoint for $b \neq b'$). Define an OBS-surjective Γ-environment ρ_A on A by $\rho_A(x) = x$. Define an OBS-surjective Γ-environment $\rho_{A'}$ on A' by $\rho_{A'}(x) = \widehat{h}(x)$. Since $\approx_{OBS}$ is equality on $b \in OBS$, ρ_A (resp. $\rho_{A'}$) is also a Γ-environment w.r.t. $\approx_{OBS}$ on A (resp. A'). Let φ be a formula in context Γ. Then $A \models_{\rho_A} \ulcorner\varphi\urcorner$ iff $A \models_{\rho_A}^{\approx_{OBS,A}} \varphi$ (by Corollary 5.10) iff $A/{\approx_{OBS,A}} \models_{\psi\circ\rho_A} \varphi$ (by the lemma in the proof of Theorem 3.35, where ψ is the function from that proof, since $\psi_{[]}$ is the identity) iff $A'/{\approx_{OBS,A'}} \models_{h\circ\psi\circ\rho_A} \varphi$ (by Proposition 3.13, since $h_{[]}$ is the identity) iff $A'/{\approx_{OBS,A'}} \models_{\psi\circ\rho_{A'}} \varphi$ (since $h \circ \psi \circ \rho_A = \psi \circ \widehat{h} \circ \rho_A$) iff $A' \models_{\rho_{A'}}^{\approx_{OBS,A'}} \varphi$ iff $A' \models_{\rho_{A'}} \ulcorner\varphi\urcorner$. □

Corollary 5.16 *For any $A, A' \in Alg(\Sigma)$, if $A/{\approx_{OBS,A}} \cong A'/{\approx_{OBS,A'}}$ then $A \equiv_{OBS} A'$.*

PROOF: *Observe that for any equation $\varphi \in ObsEq_\Gamma(\Sigma)$, $\varphi \models\!\!\models \ulcorner\varphi\urcorner$ since the extra premise in $\ulcorner\varphi\urcorner$ always holds, by an easy argument involving Proposition 5.7. Then just apply Theorem 5.15.* □

Example 5.17 To get an example of Corollary 5.16, just turn Example 4.16 around: since $Nat/{\approx_{\{bool\},Nat}} \cong Term/{\approx_{\{bool\},Term}} \cong Term^\infty/{\approx_{\{bool\},Term^\infty}}$ (Example 3.34) we obtain $Nat \equiv_{\{bool\}} Term \equiv_{\{bool\}} Term^\infty$ (Example 4.12). □

Yet another definition of behavioural equivalence is obtained by extending the definition of $\equiv_{OBS}$ to take advantage of the availability of higher-order formulae to perform experiments.

Definition 5.18 *A type τ is* observable *if either:*

– *τ is a base type that is in OBS; or*
– *$\tau = [\tau_1, \ldots, \tau_n]$ and τ_i is observable for all $1 \leq i \leq n$.*

Let Γ be a context. A term t in context Γ is observation-restricted *if all types occurring in t (i.e. as types of bound variables in λ-abstractions and universal quantifications) are observable. If t is a formula and Γ is an OBS-context then t is called* observable. *Let $ObsForm_\Gamma(\Sigma)$ be the set of observable formulae in context Γ.*

Since predicates in formulae can only arise in two ways — via λ-abstraction and via quantification — the restrictions imposed on observable formulae ensure that predicates in such formulae always have observable type. Note that $ObsEq_\Gamma(\Sigma) \subset ObsForm_\Gamma(\Sigma)$.

Definition 5.19 *Let $A, A' \in Alg(\Sigma)$. A is* behaviourally equivalent *to A'* via formulae*, written $A \equiv_{OBSForm} A'$, if there is an OBS-context Γ and Γ-environments ρ_A on A and $\rho_{A'}$ on A' that are OBS-surjective such that for any formula $\varphi \in ObsForm_\Gamma(\Sigma)$, $A \models_{\rho_A} \varphi$ iff $A' \models_{\rho_{A'}} \varphi$.*

Left-factorizability of $\equiv_{OBSForm}$ by $\approx_{OBS}$ is another direct consequence of Theorem 5.15.

Corollary 5.20 *For any $A, A' \in Alg(\Sigma)$, if $A/{\approx_{OBS,A}} \cong A'/{\approx_{OBS,A'}}$ then $A \equiv_{OBSForm} A'$.*

PROOF: *For any observable type τ, $v \approx_{OBS,A,\tau} v$ for any $v \in [\![\tau]\!]_A$, by induction on the structure of τ using Proposition 5.6. From this it follows that for any $\varphi \in ObsForm_\Gamma(\Sigma)$, $\varphi \models\!\!\models \ulcorner\varphi\urcorner$. Then apply Theorem 5.15.* □

Theorem 5.21 *$\equiv_{RelForm}$, $\equiv_{OBS}$ and $\equiv_{OBSForm}$ are factorizable by $\approx_{OBS}$.*

PROOF:

$\equiv_{RelForm}$: By Proposition 4.15 (since $\equiv_{RelForm} \subseteq \equiv_{OBS}$ by an argument like the one in Corollary 5.16) and Theorem 5.15.
$\equiv_{OBS}$: By Proposition 4.15 and Corollary 5.16.
$\equiv_{OBSForm}$: By Proposition 4.15 (since $\equiv_{OBSForm} \subseteq \equiv_{OBS}$) and Corollary 5.20.
□

It is an easy consequence of the above theorem that all three of our behavioural equivalence relations coincide. This demonstrates that using formulae more complex than equations as experiments does not allow finer distinctions between algebras to be made. This is not necessarily what one would expect: in the case of non-deterministic algebras, the use of more complex formulae does yield a different relation, see [Nip88].

Corollary 5.22 *$\equiv_{RelForm} = \equiv_{OBS} = \equiv_{OBSForm}$.*

PROOF: *Immediate from Theorem 5.21 and the definition of factorizability.*
□

6 Relating abstractor specifications and behavioural specifications

As discussed in the introduction, ordinary specifications consisting of a signature together with a set of axioms are not sufficiently abstract in that they sometimes describe classes of algebras that are not closed under behavioural equivalence. Two approaches to resolving this problem have been proposed. The first, due to [SW83], is to simply close the class of models of a specification under behavioural equivalence using an operation called *behavioural abstraction*. The second, due to [Rei85], is to take as models of a specification all those algebras that behaviourally satisfy the axioms. We provide syntax for all three kinds of specifications here in order to study how they are related.

Definition 6.1 *A (*flat*)* specification *consists of a signature Σ and a set Φ of closed Σ-formulae, called* axioms*. The* models *of a specification $\langle \Sigma, \Phi \rangle$ are all the algebras in the class*

$$Mod(\langle \Sigma, \Phi \rangle) = \{A \in Alg(\Sigma) \mid A \models \Phi\}.$$

Let $\langle \Sigma, \Phi \rangle$ be a specification. Let $\approx = \langle \approx_A \rangle_{A \in Alg(\Sigma)}$ be a family such that each $\approx_A$ is a partial congruence on A, and let $\equiv \subseteq Alg(\Sigma) \times Alg(\Sigma)$ be an

equivalence relation.

Definition 6.2 *For any class $\mathcal{A} \subseteq Alg(\Sigma)$, the* closure *of $\mathcal{A}$ under $\equiv$ is the class*

$$Abs_{\equiv}(\mathcal{A}) = \{A \in Alg(\Sigma) \mid A \equiv A' \text{ for some } A' \in \mathcal{A}\}.$$

When $\equiv$ is the relation $\equiv_{OBS}$ for some set OBS of base types, the operator $Abs_{\equiv}$ is known as behavioural abstraction.

*A (*flat*)* abstractor specification $\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv$ *has as models all those Σ-algebras that are equivalent to models of $\langle \Sigma, \Phi \rangle$:*

$$Mod(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv) = Abs_{\equiv}(Mod(\langle \Sigma, \Phi \rangle)).$$

Definition 6.3 *A (*flat*)* behavioural specification $\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx$ *has as models all those Σ-algebras that satisfy the axioms Φ w.r.t. $\approx$:*

$$Mod(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx) = \{A \in Alg(\Sigma) \mid A \models^{\approx_A} \Phi\}.$$

The notation used for behavioural specifications should not be confused with similar notation used in [SW83] and [ST87] for a particular special case of abstractor specifications.

Example 6.4 $\langle \Sigma_{ctr}, \Phi_{ctr} \rangle$ is a specification having as models the class of all Σ_{ctr}-algebras that are isomorphic to Nat. $\mathsf{abstract}\ \langle \Sigma_{ctr}, \Phi_{ctr} \rangle\ \mathsf{w.r.t.}\ \equiv_{\{bool\}}$ is an abstractor specification having as models Nat, $Term$, $Term^{\infty}$, all Σ_{ctr}-algebras that are isomorphic to one of these, and many other Σ_{ctr}-algebras besides. $\mathsf{behaviour}\ \langle \Sigma_{ctr}, \Phi_{ctr} \rangle\ \mathsf{w.r.t.}\ \approx_{\{bool\}}$ is a behavioural specification having the same class of models. □

We have now built up enough machinery to redo the development in [BHW95] in the framework of higher-order logic. Although it is not made explicit there, their results are independent of the logic used in axioms, provided properties corresponding to Corollary 3.14 and Theorem 3.35 hold for the logic of interest. In the remainder of this section we merely state the most important theorems and indicate dependencies; for proofs, more results, and discussion, see [BHW95].

The theorems below are stated for arbitrary choices of $\approx$ and $\equiv$ such that $\approx$ is regular or weakly regular and $\equiv$ is factorizable by $\approx$. The particular case of interest is where $\approx$ and $\equiv$ are $\approx_{OBS}$ and $\equiv_{OBS}$ respectively, for an arbitrary choice OBS of observable base types, which satisfy the requirements by Propositions 4.7 and 4.9, and Theorem 5.21.

Theorem 6.5 ([BHW95]) *If $\approx$ is regular and $\equiv$ is factorizable by $\approx$, then* $Mod(\textsf{behaviour } \langle \Sigma, \Phi \rangle \textsf{ w.r.t. } \approx) = Abs_{\equiv}(FA_{\approx}(Mod(\langle \Sigma, \Phi \rangle)))$.

PROOF: *See [BHW95]. The proof depends on Theorem 3.35 and Corollary 3.14, and the fact that regularity and factorizability implies $A \equiv A/{\approx_A}$ for any A.* □

Theorem 6.6 ([BHW95]) *If $\approx$ is weakly regular and $\equiv$ is factorizable by $\approx$, then* $Mod(\textsf{behaviour } \langle \Sigma, \Phi \rangle \textsf{ w.r.t. } \approx) \subseteq Mod(\textsf{abstract } \langle \Sigma, \Phi \rangle \textsf{ w.r.t. } \equiv)$.

PROOF: *See [BHW95]. The proof depends on Theorem 3.35.* □

Note that if the requirement of weak regularity of $\approx$ is slightly strengthened to regularity, then Theorem 6.6 is an immediate corollary of Theorem 6.5.

The main characterization theorem is the following:

Theorem 6.7 ([BHW95]) *Suppose $\approx$ is weakly regular and $\equiv$ is factorizable by $\approx$. The following conditions are equivalent:*

(i) $Mod(\textsf{behaviour } \langle \Sigma, \Phi \rangle \textsf{ w.r.t. } \approx) = Mod(\textsf{abstract } \langle \Sigma, \Phi \rangle \textsf{ w.r.t. } \equiv)$
(ii) $Mod(\langle \Sigma, \Phi \rangle) \subseteq Mod(\textsf{behaviour } \langle \Sigma, \Phi \rangle \textsf{ w.r.t. } \approx)$
(iii) For all $A \in Mod(\langle \Sigma, \Phi \rangle)$, $A/{\approx_A} \in Mod(\langle \Sigma, \Phi \rangle)$

PROOF: *See [BHW95]. The proof depends on Theorems 3.35 and 6.6 and Corollary 3.14.* □

Example 6.8 According to Theorem 6.7, the fact that the model class $Mod(\textsf{abstract } \langle \Sigma_{ctr}, \Phi_{ctr} \rangle \textsf{ w.r.t. } \equiv_{\{bool\}})$ coincides with the model class $Mod(\textsf{behaviour } \langle \Sigma_{ctr}, \Phi_{ctr} \rangle \textsf{ w.r.t. } \approx_{\{bool\}})$ (Example 6.4) follows from the fact that for all $A \in Mod(\langle \Sigma_{ctr}, \Phi_{ctr} \rangle)$, $A/{\approx_A} \in Mod(\langle \Sigma_{ctr}, \Phi_{ctr} \rangle)$. The latter holds because the only model of $\langle \Sigma_{ctr}, \Phi_{ctr} \rangle$ is (up to isomorphism) Nat, which is $\approx_{\{bool\}}$-fully abstract (Example 4.6) and is hence isomorphic to its quotient by $\approx_{\{bool\},Nat}$.

Now consider the signature Σ'_{ctr} obtained by adding a constant $zero' : ctr$ to Σ_{ctr}, and the set of axioms Φ'_{ctr} obtained by adding the formula $\neg(zero =_{ctr} zero')$ to Φ_{ctr} and removing the formula *GENCTR*. Let Nat' be the Σ'_{ctr}-algebra consisting of "two copies of Nat", defined by taking $[\![ctr]\!]_{Nat'} = \omega \uplus \omega$ and the obvious interpretation of the constants such that $[\![zero]\!]_{Nat'} \neq [\![zero']\!]_{Nat'}$ and $[\![is\text{-}zero(zero)]\!]_{Nat'} = [\![is\text{-}zero(zero')]\!]_{Nat'} = T$. Then $Nat' \in Mod(\langle \Sigma'_{ctr}, \Phi'_{ctr} \rangle)$ and so $Nat' \in Mod(\textsf{abstract } \langle \Sigma'_{ctr}, \Phi'_{ctr} \rangle \textsf{ w.r.t. } \equiv_{\{bool\}})$. On the other hand, $Nat' \notin Mod(\textsf{behaviour } \langle \Sigma'_{ctr}, \Phi'_{ctr} \rangle \textsf{ w.r.t. } \approx_{\{bool\}})$ since $Nat' \not\models^{\approx_{Nat'}} \neg(zero =_{ctr} zero')$, and this is because there is no context which allows us to distinguish $[\![zero]\!]_{Nat'}$ and $[\![zero']\!]_{Nat'}$. Thus we see that conditions (i) and (ii) of The-

orem 6.7 are not satisfied. Condition (iii) is also not satisfied since $Nat'/{\approx_{Nat'}}$ is isomorphic to Nat augmented by taking $[\![zero']\!]_{Nat'} = [\![zero]\!]_{Nat'}$, which does not satisfy $\neg(zero =_{ctr} zero')$. □

7 Reasoning about specifications

The results presented above serve to clarify our understanding of behavioural satisfaction and behavioural equivalence and the relationship between these in the context of higher-order logic. A concrete benefit of this is a number of methods for reasoning about specifications, as will be summarized below. Some of these appear in a different form in [BH96] or elsewhere, while others (Proof Methods 7.5, 7.8 and 7.10) are new.

We begin by introducing some (mostly standard) concepts and notation. Let $\Sigma = \langle B, C\rangle$ be a signature. In this section we restrict attention to closed formulae.

Definition 7.1 *A closed formula φ is a* consequence *of a set Φ of closed formulae, written $\Phi \models \varphi$, if for any Σ-algebra A, $A \models \Phi$ implies $A \models \varphi$.*

When reasoning about a specification SP, our goal is to discover whether or not a given formula φ is satisfied by all models of SP. Let $\approx = \langle \approx_A \rangle_{A \in Alg(\Sigma)}$ be a family of partial congruences. A related goal is that of discovering whether or not φ is satisfied w.r.t. $\approx$ by all models of SP. These questions amount to determining whether or not φ is in the *theory* (resp. *theory w.r.t.* $\approx$) of SP.

Definition 7.2 *Let $\mathcal{A} \subseteq Alg(\Sigma)$ be a class of Σ-algebras. The* theory w.r.t. $\approx$ *of $\mathcal{A}$ is the set $Th_{\approx}(\mathcal{A}) = \{\varphi \mid A \models^{\approx_A} \varphi$ for every $A \in \mathcal{A}\}$. The* (ordinary) theory *of $\mathcal{A}$ is the set $Th(\mathcal{A}) = \{\varphi \mid A \models \varphi$ for every $A \in \mathcal{A}\}$; note that $Th(\mathcal{A}) = Th_{=}(\mathcal{A})$. If SP is a specification, we write $Th(SP)$ for $Th(Mod(SP))$ and $Th_{\approx}(SP)$ for $Th_{\approx}(Mod(SP))$.*

The essence of reasoning about specifications is to find a way of reducing the problems of determining $\varphi \in Th(SP)$ and $\varphi \in Th_{\approx}(SP)$ to that of consequence ($\Phi \models \psi$ for appropriate Φ and ψ); then any proof system that is sound for $\models$ may be used to finish the job. For the ordinary theory of a flat specification, the reduction is trivial: $\varphi \in Th(\langle \Sigma, \Phi\rangle)$ iff $\Phi \models \varphi$. For the theory w.r.t. $\approx$ and for behavioural specifications and abstractor specifications, the problem is much more difficult. We consider each case below, giving proof methods that provide such reductions. Some cases have particular importance, as indicated below, while others are treated only for the sake of completeness and/or because the important cases are reducible to these under certain conditions.

Let $\approx$ be expressible by the family of predicates $\sim = \langle \sim_b \rangle_{b \in B}$, and let $\equiv \subseteq Alg(\Sigma) \times Alg(\Sigma)$ be an equivalence relation.

7.1 $\varphi \in Th(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$

This problem can be reduced to ordinary consequence by applying the following easy consequence of Corollary 5.10:

Proposition 7.3 $Mod(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx) = Mod(\langle \Sigma, \ulcorner \Phi \urcorner \rangle)$ *where* $\ulcorner \Phi \urcorner = \{\ulcorner \varphi \urcorner \mid \varphi \in \Phi\}$. □

This leads to the following proof method:

Proof Method 7.4 $\varphi \in Th(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$ *iff* $\ulcorner \Phi \urcorner \models \varphi$. □

If $\approx$ is weakly regular and $\equiv$ is factorizable by $\approx$, then any behavioural specification is at least as restrictive as the corresponding abstractor specification by Theorem 6.6. Thus, under these conditions the proof methods in Section 7.2 below (i.e. Proof Methods 7.5–7.10) may be soundly applied to this problem.

7.2 $\varphi \in Th(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv)$

This is the problem that is of importance for reasoning about specifications in a language like ASL [SW83] that includes a specification-building operation corresponding to abstract; cf. [Far92].

If Theorem 6.7 applies, this problem can be reduced to the problem treated in Section 7.1 above. Then Proof Method 7.4 is applicable.

Alternatively, if the formula to be proved is a relativized formula or is logically equivalent to such a formula, Corollary 5.13 yields the following reduction.

Proof Method 7.5 *Suppose that* $\equiv$ *is factorizable by* $\approx$ *and* φ *is equivalent to a relativized formula, i.e.* $\varphi \models\!\models \ulcorner \psi \urcorner$ *for some (closed) formula* ψ. *Then* $\Phi \models \varphi$ *implies* $\varphi \in Th(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv)$.

Proof: *Suppose* $A \in Mod(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv)$, *i.e. there is some algebra* A' *such that* $A' \in Mod(\langle \Sigma, \Phi \rangle)$ *and* $A \equiv A'$, *and* $\Phi \models \varphi$. *But then* $A' \models \varphi$, *so* $A' \models \ulcorner \psi \urcorner$, *and then* $A \models \ulcorner \psi \urcorner$ *(by factorizability and Corollary 5.13) so* $A \models \varphi$. □

This is a direct extension of the method for reasoning about abstractor specifications presented in Section 4 of [ST87], which applies only to formu-

lae built in certain ways from observable equations. By analogy with an observation there, Proof Method 7.5 is not confined to inferring formulae that are equivalent to relativized formulae. In order to validly conclude that $\varphi \in Th(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv)$, it is enough to have a proof of $\Phi \models \varphi$ for which there is a "horizontal cut" containing only formulae that are equivalent to relativized formulae. Similar remarks apply to the proof methods presented below. In applying Proof Method 7.5, we normally (as in the example below) take ψ to be φ, but this is not required. A formula that is equivalent to its own relativization is called a "$\approx$-invariant" formula in [BH96], but this concept is not used as the basis of a reasoning method there.

Example 7.6 Suppose that $\langle \Sigma_{nat}, \Phi_{nat} \rangle$ specifies the natural numbers with the usual constants (at least 0, 1 and +) and the usual axioms. Now consider the specification:

$$Counter =_{\text{def}} \mathsf{abstract}\ \langle \Sigma_{ctr} \cup \Sigma_{nat}, \Phi_{ctr} \cup \Phi_{nat} \rangle\ \mathsf{w.r.t.}\ \equiv_{\{bool,nat\}}$$

Our aim is to prove the property

$$\forall n, m{:}nat.n < m \Rightarrow \mathit{is\text{-}zero}(\mathit{mdec}(n, \mathit{minc}(m, \mathit{zero}))) =_{bool} \mathit{false}$$

where the functions *minc* and *mdec* (multiple *inc*/*dec*) are defined recursively "on top of" the specification *Counter* by the following ML-like code:

$$\begin{array}{l} \texttt{fun}\ \mathit{minc}(0, c) = c \\ \quad \texttt{|}\ \mathit{minc}(n+1, c) = \mathit{minc}(n, \mathit{inc}(c)) \\ \texttt{fun}\ \mathit{mdec}(0, c) = c \\ \quad \texttt{|}\ \mathit{mdec}(n+1, c) = \mathit{mdec}(n, \mathit{dec}(c)) \end{array}$$

and $<$ (less than) is as usual.

Our framework admits neither structured specifications nor quantification over functional types (see Section 8 for comments on both of these omissions), which rules out the obvious ways of expressing the above problem. But we can encode it using predicates as follows:

$$\begin{array}{l} \varphi =_{\text{def}} \forall n, m{:}nat.n < m \Rightarrow \\ \qquad \forall c, c'{:}ctr.\mathit{Minc}(m, \mathit{zero}, c) \wedge \mathit{Mdec}(n, c, c') \Rightarrow \mathit{is\text{-}zero}(c') =_{bool} \mathit{false} \end{array}$$

where $<$, *Minc*, and *Mdec* are predicates defined as follows:

$$< =_{\text{def}} \lambda(n{:}nat, m{:}nat).\forall P{:}[nat, nat].$$
$$(\forall x{:}nat.P(0, x+1)) \wedge (\forall x, y{:}nat.P(x, y) \Rightarrow P(x+1, y+1)) \Rightarrow P(n, m)$$

$$Minc =_{\text{def}} \lambda(n{:}nat, c{:}ctr, c'{:}ctr).\forall P{:}[nat, ctr, ctr].$$
$$(\forall t{:}ctr.P(0, t, t)) \wedge (\forall x{:}nat.\forall t, v{:}ctr.P(x, inc(t), v) \Rightarrow P(x+1, t, v)) \Rightarrow P(n, c, c')$$

$$Mdec =_{\text{def}} \lambda(n{:}nat, c{:}ctr, c'{:}ctr).\forall P{:}[nat, ctr, ctr].$$
$$(\forall t{:}ctr.P(0, t, t)) \wedge (\forall x{:}nat.\forall t, v{:}ctr.P(x, dec(t), v) \Rightarrow P(x+1, t, v)) \Rightarrow P(n, c, c')$$

Our goal is to establish $\varphi \in Th(Counter)$. Proof Method 7.5 says that this follows if we can establish that $\Phi_{ctr} \cup \Phi_{nat} \models \varphi$ and $\varphi \models\!\!\models \ulcorner\psi\urcorner$ for some formula ψ. The former follows by induction on n. For the latter we first establish that *Minc* and *Mdec* are compatible with $\approx_{\{bool,nat\}}$ in the sense that whenever $Minc(n, c, c')$ and $c \approx_{\{bool,nat\}} d$ then we can find d' such that $Minc(n, d, d')$ and $c' \approx_{\{bool,nat\}} d'$, and similarly for *Mdec*. Using this property we can characterize $\ulcorner Minc\urcorner$ and $\ulcorner Mdec\urcorner$ as the $\approx_{\{bool,nat\}}$-closures of *Minc* and *Mdec* respectively. $\varphi \models \ulcorner\varphi\urcorner$ then follows from this and the compatibility property using the fact that the final conclusion of φ is observable and thus respects $\approx_{\{bool,nat\}}$. For the converse $\ulcorner\varphi\urcorner \models \varphi$ one uses the characterization of $\ulcorner Minc\urcorner$ and $\ulcorner Mdec\urcorner$ directly.

This problem is taken from Section 5 of [Sch92] where it is shown that an infinite number of applications of the proof method in [ST87] would be required in a proof of the property given above. □

Convenient use of Proof Method 7.5 requires a syntactic criterion which enables us to conclude $\varphi \models\!\!\models \ulcorner\varphi\urcorner$ directly from the form of φ rather than via a semantic argument as in the above example. It appears, however, that the above line of reasoning is independent of the particular definition of the functions *minc* and *mdec* and thus should generalize to other recursively defined local functions. Indeed, we believe that one has $\varphi \models\!\!\models \ulcorner\varphi\urcorner$ for any formula φ that arises from an encoding of a formula having a form like $\forall f{:}\tau.\langle \textit{recursive definition of } f\rangle \Rightarrow \psi$ where τ is a functional type and $\psi \models\!\!\models \ulcorner\psi\urcorner$, but we leave this question to future research — see Section 8.1. For the time being we content ourselves with a simpler but useful syntactic criterion that does not apply directly to the above example. This is obtained by

adding "respectful" abstraction λ^{r} and quantification $\forall^{\mathrm{r}}$ to the syntax, where:

$$\lambda^{\mathrm{r}}(x_1{:}\tau_1, \ldots, x_n{:}\tau_n).t \text{ abbreviates } \lambda(x_1{:}\tau_1, \ldots, x_n{:}\tau_n). \\ DOM_{\tau_1}(x_1) \wedge \cdots \wedge DOM_{\tau_n}(x_n) \wedge t$$

$$\forall^{\mathrm{r}} x{:}\tau.t \text{ abbreviates } \forall x{:}\tau. DOM_{\tau}(x) \Rightarrow t$$

Definition 7.7 *A* respectful formula *is a formula that may contain λ^{r} and/or $\forall^{\mathrm{r}}$ but does not contain λ or $\forall$.*

It is easy to see that $\varphi \models\!\models \ulcorner\varphi\urcorner$ for any respectful formula φ. This gives the following.

Proof Method 7.8 *Suppose that $\equiv$ is factorizable by $\approx$ and φ is a closed respectful formula. Then $\Phi \models \varphi$ implies $\varphi \in Th(\mathsf{abstract}\ \langle \Sigma, \Phi\rangle\ \mathsf{w.r.t.}\ \equiv)$.* □

In the case of behavioural abstraction, note that $\forall^{\mathrm{r}}$ on base types corresponds exactly to reachable quantification as in [Sch92].

Example 7.9 Consider the binary predicate on *ctr* that determines if one value can be obtained from another by application of *inc*:

$$\textit{comes-after}(x{:}\textit{ctr}, y{:}\textit{ctr}) = \text{“}\exists n \geq 0. \textit{inc}^n(y) =_{\textit{ctr}} x\text{”}$$

This can be expressed in our framework as follows:

$$\textit{comes-after} =_{\mathrm{def}} \lambda(x{:}\textit{ctr}, y{:}\textit{ctr}).\forall P{:}[\textit{ctr}, \textit{ctr}]. \\ (\forall a{:}\textit{ctr}.P(a,a)) \wedge (\forall a{:}\textit{ctr}.P(\textit{inc}(a), a)) \\ \wedge (\forall a, b, c{:}\textit{ctr}.P(a,b) \wedge P(b,c) \Rightarrow P(a,c)) \\ \Rightarrow P(x,y)$$

Now consider the formula

$$\varphi =_{\mathrm{def}} \forall x, y{:}\textit{ctr}.\textit{comes-after}(x,y) \vee \textit{comes-after}(y,x)$$

Then $\varphi \notin Th(\mathsf{abstract}\ \langle \Sigma_{\textit{ctr}}, \Phi_{\textit{ctr}}\rangle\ \mathsf{w.r.t.}\ \equiv_{\{\textit{bool}\}})$, as can be seen by considering $\textit{Term} \in \textit{Mod}(\mathsf{abstract}\ \langle \Sigma_{\textit{ctr}}, \Phi_{\textit{ctr}}\rangle\ \mathsf{w.r.t.}\ \equiv_{\{\textit{bool}\}})$ (Example 6.4) and instantiating x to $\textit{dec}(\textit{inc}(\textit{zero}))$ and y to $\textit{zero}$.

Next, consider a respectful version of φ:

$$\varphi^{\mathrm{r}} =_{\mathrm{def}} \forall^{\mathrm{r}} x, y{:}\textit{ctr}.\textit{comes-after}^{\mathrm{r}}(x,y) \vee \textit{comes-after}^{\mathrm{r}}(y,x)$$

where

$$\begin{aligned} \mathit{comes\text{-}after}^{\mathrm{r}} =_{\mathrm{def}} \lambda^{\mathrm{r}}(x{:}\mathit{ctr}, y{:}\mathit{ctr}).\forall^{\mathrm{r}} P{:}[\mathit{ctr}, \mathit{ctr}]. \\ (\forall^{\mathrm{r}} a{:}\mathit{ctr}.P(a,a)) \wedge (\forall^{\mathrm{r}} a{:}\mathit{ctr}.P(\mathit{inc}(a),a)) \\ \wedge (\forall^{\mathrm{r}} a,b,c{:}\mathit{ctr}.P(a,b) \wedge P(b,c) \Rightarrow P(a,c)) \\ \Rightarrow P(x,y) \end{aligned}$$

Informally, the effect of the respectful quantifiers is to give

$$\mathit{comes\text{-}after}^{\mathrm{r}}(x{:}\mathit{ctr}, y{:}\mathit{ctr}) = \text{“}\exists n \geq 0.\mathit{inc}^n(y) \approx_{\mathit{ctr}} x\text{”}$$

for x, y that respect $\approx$. Taking $\approx$ to be $\approx_{\{\mathit{bool}\}}$, the respectful quantifier over ctr in φ^{r} quantifies over the $\{\mathit{bool}\}$-reachable values of type ctr (so in Term^{∞}, the value ∞ is excluded). We can use Proof Method 7.8 to conclude $\varphi^{\mathrm{r}} \in \mathit{Th}(\mathsf{abstract}\ \langle \Sigma_{\mathit{ctr}}, \Phi_{\mathit{ctr}} \rangle\ \mathsf{w.r.t.}\ \equiv_{\{\mathit{bool}\}})$ from $\Phi_{\mathit{ctr}} \models \varphi^{\mathrm{r}}$, since φ^{r} is a respectful formula. As observed above, this conclusion is not valid for the non-respectful version φ. □

Since every observable formula amounts to a respectful formulae (since respectful abstraction and quantification over observable types is equivalent to ordinary abstraction and quantification), we have the following:

Proof Method 7.10 *Suppose that φ is a closed observable formula. Then $\Phi \models \varphi$ implies $\varphi \in \mathit{Th}(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv_{\mathit{OBS}})$.* □

7.3 $\varphi \in \mathit{Th}_{\approx}(\langle \Sigma, \Phi \rangle)$

This is the problem that is studied in [BH96], where it is argued that a solution to this problem provides the basis of a strategy for proving correctness of implementation steps in stepwise refinement of specifications (cf. [BH95] and “abstractor” implementations in [ST88b]).

The following proof method follows immediately from Corollary 5.10:

Proof Method 7.11 $\varphi \in \mathit{Th}_{\approx}(\langle \Sigma, \Phi \rangle)$ *iff* $\Phi \models \ulcorner \varphi \urcorner$. □

This is essentially the same as the solution proposed in [BH96], except that because the analogue of our Corollary 5.10 there involves infinitary formulae, more work is required to reduce the problem to one of consequence for finitary formulae.

If Theorem 6.7 applies then this problem is equivalent to the problem treated in Section 7.4 below according to the following result, taking $\equiv$ to be the equivalence induced by $\approx$, i.e. such that $A \equiv A'$ iff $A/{\approx_A} \cong A'/{\approx_{A'}}$. See Example 7.18.

Proposition 7.12 ([BHW95]) *If $\equiv$ is factorizable by $\approx$ then $Th_{\approx}(Abs_{\equiv}(\mathcal{A})) = Th_{\approx}(\mathcal{A})$.*

PROOF: *Straightforward, using Corollary 3.14 and two applications of Theorem 3.35.* □

(In [BHW95], this result has an additional assumption, not used in the proof, that $\approx$ is weakly regular.)

7.4 $\varphi \in Th_{\approx}(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$

Corollary 5.10 and Proposition 7.3 yield the following proof method:

Proof Method 7.13 $\varphi \in Th_{\approx}(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$ *iff* $\ulcorner \Phi \urcorner \models \ulcorner \varphi \urcorner$. □

Another approach, which appears to be more powerful, is obtained by appealing to the following results:

Proposition 7.14 ([BHW95]) *If $\equiv$ is factorizable by $\approx$ then*

$$Th_{\approx}(Abs_{\equiv}(FA_{\approx}(\mathcal{A}))) = Th(FA_{\approx}(\mathcal{A})).$$

PROOF: *By Proposition 7.12 and the definition of fully abstract algebra.* □

Proposition 7.15 ([BH96])

$$FA_{\approx}(Mod(\langle \Sigma, \Phi \rangle)) = Mod(\langle \Sigma, \Phi \cup \{\forall x, y{:}b.(x \sim_b y \Leftrightarrow x =_b y) \mid b \in B\} \rangle).$$

PROOF: *Directly from the definition of fully abstract algebra.* □

These together with Theorem 6.5 (taking $\equiv$ to be the equivalence induced by $\approx$) yield the following:

Proof Method 7.16 *Suppose that $\approx$ is regular. Then $\varphi \in Th_{\approx}(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$ iff $\Phi \cup \{\forall x, y{:}b.(x \sim_b y \Leftrightarrow x =_b y) \mid b \in B\} \models \varphi$.* □

This is essentially the same as the method proposed in [BH96], with the proviso concerning infinitary formulae mentioned earlier.

It is worth pointing out that a weaker but very simple and potentially useful consequence of this is the following:

Proof Method 7.17 *Suppose that* $\approx$ *is regular. Then* $\varphi \in Th_{\approx}(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$ *if* $\Phi \models \varphi$. □

Example 7.18 Consider the formula φ from Example 7.9. We know $\varphi \in Th_{\approx_{\{bool\}}}(\mathsf{behaviour}\ \langle \Sigma_{ctr}, \Phi_{ctr} \rangle\ \mathsf{w.r.t.}\ \approx_{\{bool\}})$ by Proof Method 7.17 since $\Phi_{ctr} \models \varphi$. Since Theorem 6.7 applies (see Example 6.8), by Proposition 7.12 we also obtain $\varphi \in Th_{\approx_{\{bool\}}}(\langle \Sigma_{ctr}, \Phi_{ctr} \rangle)$. □

Finally, a more direct approach to this problem is to reduce it trivially to consequence w.r.t. $\approx$:

Definition 7.19 *A closed formula* φ *is a* consequence *of a set* Φ *of closed formulae* w.r.t. $\approx$, *written* $\Phi \models^{\approx} \varphi$, *if for any* Σ*-algebra* A, $A \models^{\approx_A} \Phi$ *implies* $A \models^{\approx_A} \varphi$.

Proof Method 7.20 $\varphi \in Th_{\approx}(\mathsf{behaviour}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \approx)$ *iff* $\Phi \models^{\approx} \varphi$. □

Then what is required to finish the job is a proof system that is sound for $\models^{\approx}$. See [Rei85], where a proof system for conditional equational logic is given that is sound for an indistinguishability relation different from $\approx_{OBS}$, in the context of partial algebras; see also [HW95].

7.5 $\varphi \in Th_{\approx}(\mathsf{abstract}\ \langle \Sigma, \Phi \rangle\ \mathsf{w.r.t.}\ \equiv)$

If $\equiv$ is factorizable by $\approx$ then this problem is equivalent to the problem treated in Section 7.3 above according to Proposition 7.12. Thus Proof Method 7.11 is applicable. If Theorem 6.7 applies then Proof Methods 7.13–7.20 from Section 7.4 are applicable as well.

Example 7.21 Consider once again the formula φ from Example 7.9, and recall from Example 7.18 that $\varphi \in Th_{\approx_{\{bool\}}}(\mathsf{behaviour}\ \langle \Sigma_{ctr}, \Phi_{ctr} \rangle\ \mathsf{w.r.t.}\ \approx_{\{bool\}})$ using Proof Method 7.17. Since Theorem 6.7 applies, we also obtain $\varphi \in Th_{\approx_{\{bool\}}}(\mathsf{abstract}\ \langle \Sigma_{ctr}, \Phi_{ctr} \rangle\ \mathsf{w.r.t.}\ \equiv_{\{bool\}})$. □

8 Further work

We consider below the treatment of functional types and structured specifications, and the potential for application of the results presented above.

8.1 Functional types

The language defined in Section 2 is higher-order because it allows quantification over predicates in addition to the usual quantification over individuals. There is however no quantification over function types or use of functions as arguments to predicates or functions, cf. e.g. [Möl87], [Mei92]. There are two reasons for wanting to include functional types. First, higher-typed functional constants in signatures permit elegant specifications both of functional data structures such as streams and λ-models, and of higher-typed constants such as *map* and *iter*. Second, (higher-typed) functions used locally in formulae allow us to express some examples in a much clearer way — compare the informal presentation in Example 7.6 with the formula that encodes it.

Higher-typed functions can be added either by encoding them in terms of what is already present in the language, or by extending the basic framework with additional primitives. A specification having a signature involving higher-typed constants can be translated into a specification with a first-order signature by closing the set of base types under a formal function space constructor $\Rightarrow$, adding appropriately-typed S, K and application operators, and changing the types of constants to use $\Rightarrow$ in place of $\rightarrow$ (assuming w.l.o.g. that all constants are curried). The axioms of the specification have to be translated to make the use of application explicit and to use S and K instead of λ-abstraction, and finally augmented by the usual equations constraining the interpretation of these additional operators. In addition to S and K one could also add a family of fixpoint operators (Y) to allow for recursively-defined functions in axioms. The source language of this encoding would then resemble the framework of [Grü90], except that his logic is first-order.

Even when signatures contain only first-order constants, function types can be useful in formulae as shown by Example 7.6. Such types can be encoded using predicate types as shown there. We believe that the passage from the informal presentation to its encoding can be made more systematic, following ideas in [TvD88]; it also seems to be possible to encode uses of a description operator like Hilbert's ι which selects values that satisfy a predicate. The interpretation we give to predicate types means that this encoding yields the full set-theoretic function space. This is in contrast to the encoding above, where carriers of "functional" types like $int \Rightarrow int$ may exclude functions that

are not λ-definable. We can use such types for types of variables in formulae, but the result would be different from the result of using predicate types because of this different interpretation of the function space.

Reasoning about values of functional types by explicitly expanding formulae according to an encoding is cumbersome. A more attractive alternative is to develop proof methods that work directly on the "high-level" syntax. An example would be a proof method to enable us to conclude $\varphi \models\!\!\models \ulcorner\varphi\urcorner$ in Example 7.6, exploiting the fact that φ arises as the encoding of a specification involving recursively-defined functions.

Alternatively, we could extend the basic framework itself with function types. This involves giving an interpretation to such types, both in the standard case and w.r.t. a partial congruence. As we have seen above, there are different choices for the standard case. Using the full function space to interpret function types in signatures precludes examples like the specification of λ-models: postulating an injection from the full function space $D \to D$ to D admits only the trivial model. On the other hand, using the full function space to interpret functional types occurring locally in formulae seems unproblematic, and one can imagine situations in which an oracle deciding the halting problem, which would require more than just the λ-definable functions, would be useful. We might even want to provide two different function spaces, with different notations to distinguish between them. The interpretation of function types w.r.t. a partial congruence is possible but is not straightforward, since with the obvious choice, namely $[\![\tau \to \tau']\!]^{\approx}_A = \{f : [\![\tau]\!]^{\approx}_A \to [\![\tau']\!]^{\approx}_A \mid \forall u, v \in [\![\tau]\!]^{\approx}_A . u \approx_{A,\tau} v \Rightarrow f(u) \approx_{A,\tau'} f(v)\}$, Proposition 3.22 fails to hold. A solution will be presented in a future paper.

8.2 Structured specifications

We have restricted attention above to the study of flat specifications consisting of a signature together with a set of axioms. Large specifications are normally built in a structured fashion, using specification-building operations like enrich, + and derive. It is well-known that structured specifications cannot in general be reduced to equivalent flat specifications (see e.g. [ST95]), and the structure of specifications provides an interesting added dimension to the study of reasoning about specifications, implementation of specifications, etc.

An attempt to extend the characterization results to structured specifications in the context of first-order logic appears in [BHW95], where the extension of behaviour to structured specifications is a *post hoc* construction on the class of models of the underlying specification:

$$Mod(\textsf{behaviour } SP \textsf{ w.r.t. } \approx) = \{A \in Alg(\Sigma) \mid A/{\approx_A} \in Mod(SP)\}$$

where Σ is the signature of SP. An alternative is to interpret the specification-building operations in SP in the usual way but with axioms in SP satisfied according to $\models^{\approx}$ rather than $\models$, along the following lines:

$$\widehat{Mod}(\textsf{behaviour } SP \textsf{ w.r.t. } \approx) = Mod_{\approx}(SP)$$

$$Mod_{\approx}(\langle \Sigma, \Phi \rangle) = \{A \in Alg(\Sigma) \mid A \models^{\approx} \Phi\}$$

$$Mod_{\approx}(\textsf{derive from } SP \textsf{ by } \sigma) = \{A\big|_{\sigma} \mid A \in Mod_{\approx}(SP)\}$$

$$Mod_{\approx}(\textsf{enrich } SP \textsf{ by types } B \textsf{ constants } C \textsf{ axioms } \Phi) =$$
$$\{A \in Alg(Sig(SP) \cup \langle B, C \rangle) \mid A\big|_{Sig(SP)} \in Mod_{\approx}(SP) \wedge A \models^{\approx} \Phi\}$$

$$Mod_{\approx}(SP + SP') =$$
$$\{A \in Alg(Sig(SP) \cup Sig(SP')) \mid$$
$$A\big|_{Sig(SP)} \in Mod_{\approx}(SP) \wedge A\big|_{Sig(SP')} \in Mod_{\approx}(SP')\}$$

where $Sig(SP)$ denotes the signature of the specification SP, $\sigma : \Sigma \to \Sigma'$ is a signature morphism, $A'\big|_{\sigma}$ is the reduct of a Σ'-algebra A' to a Σ-algebra, and $A'\big|_{\Sigma}$ is the reduct along an inclusion $\Sigma \hookrightarrow \Sigma'$. If SP is a flat specification then $Mod(\textsf{behaviour } SP \textsf{ w.r.t. } \approx)$ and $\widehat{Mod}(\textsf{behaviour } SP \textsf{ w.r.t. } \approx)$ coincide by Theorem 3.35. But for structured specifications they do not coincide in general. Methods for reasoning about structured specifications — see e.g. the inference rules in [ST88a] — apply to the second interpretation but appear to be inapplicable to the first. Further research is required to clarify the relationship between abstractor specifications (which generalize easily to structured specifications) and this alternative interpretation of behavioural specifications. For now, it is perhaps worth mentioning that the characterization results above should straightforwardly extend to the case of structured specifications using the extended definition of behaviour given in [BHW95].

8.3 Application of results

One of our motivations for studying behavioural semantics of specifications with higher-order formulae as axioms was a desire to apply the results in the Extended ML framework for the formal development of ML programs from specifications [ST89], [KST95]. The characterization results and reasoning methods are of direct relevance in this context: the interpretation of Extended ML interfaces involves abstractor specifications, and the logical system used for writing axioms is (a form of) higher-order logic. However, it is difficult to apply the results as they stand to Extended ML because of the lack of functional types and treatment of structured specifications discussed above.

In particular, the most obvious pertinent examples of the use of behavioural semantics in the context of higher-order logic require functional types.

Once these extensions have been carried out, we will be in a position to apply the results and proof methods to examples in Extended ML and elsewhere, which should shed considerable light on their usefulness. Without having attempted many examples, we are not yet in a position to understand the tradeoffs between the various proof methods that may be applicable in a particular situation. But in view of the size and complexity of the predicates $INDIST_{\widehat{b}}$ in Theorem 5.4, it seems clear that proof methods that involve the direct manipulation of relativized formulae will not be convenient for use in practice when $\approx$ is the indistinguishability relation $\approx_{OBS}$. Here, a promising avenue is the search for more tractable predicates which correctly express $\approx_{OBS}$ under restrictions that are acceptable in practice (cf. [BH96]). Proof methods which make no use of the predicates $INDIST_{\widehat{b}}$ (e.g. Proof Methods 7.20 and 7.10) do not suffer from this problem.

Acknowledgements: Thanks to Michel Bidoit and Rolf Hennicker for many very useful comments, including an explanation of how their results relate to concepts and results in Sections 5 and 7. Proof Method 7.16 is due to them, and they pointed out that a previous version of Proof Methods 7.5 and 7.8 were unnecessarily restrictive. Thanks to Andrzej Tarlecki for many discussions on related topics and for drawing our attention to the idea behind the predicate $INDIST_{\widehat{b}}$ in Theorem 5.4, and to him and Michel Bidoit for comments concerning the results in [BT96]. Thanks to David Aspinall for helpful comments on a draft of this paper, and to Wolfgang Degen for providing useful pointers to the literature.

References

[BH95] M. Bidoit and R. Hennicker. Modular correctness proofs of behavioural implementations. Report 9513, Ludwig-Maximilians-Universität Mïchen (1995).

[BH96] M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. To appear in *Theoretical Computer Science* (1996).

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

[BT96] M. Bidoit and A. Tarlecki. Behavioural satisfaction and equivalence in concrete model categories. *Proc. 21st Colloq. on Trees in Algebra and Programming*, Linköping. Springer LNCS 1059, 241–256 (1996).

[Far92] J. Farrés-Casals. Verification in ASL and Related Specification Languages. Ph.D. thesis, Report CSR-92-92, Univ. of Edinburgh (1992).

[GGM76] V. Giarratana, F. Gimona and U. Montanari. Observability concepts in abstract data type specification. *Proc. 1976 Symp. on Mathematical Foundations of Computer Science*, Gdansk. Springer LNCS 45, 567–578 (1976).

[GM82] J. Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. *Proc. 9th Intl. Colloq. on Automata, Languages and Programming*, Aarhus. Springer LNCS 140, 265–281 (1982).

[Grü90] T. Grünler. Spezifikation höherer Ordnung. Ph.D. thesis, Report MIP-9012, Universität Passau (1990).

[Hen50] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic* 15:81–91 (1950).

[Hen91] R. Hennicker. Context induction: a proof principle for behavioural abstractions and algebraic implementations. *Formal Aspects of Computing* 4:326–345 (1991).

[HW95] R. Hennicker and M. Wirsing. Behavioural specifications. In: *Proof and Computation*. Marktoberdorf International Summer School, 1993. NATO ASI Series F, Vol. 139. Springer (1995).

[KST95] S. Kahrs, D. Sannella and A. Tarlecki. The definition of Extended ML: a gentle introduction. Report ECS-LFCS-95-322, Dept. of Computer Science, Univ. of Edinburgh (1995). To appear in *Theoretical Computer Science*.

[Mei92] K. Meinke. Universal algebra in higher types. *Theoretical Computer Science* 100:385–417 (1992).

[MG85] J. Meseguer and J. Goguen. Initiality, induction and computability. In: *Algebraic Methods in Semantics* (M. Nivat and J. Reynolds, eds.). Cambridge Univ. Press, 459–540 (1985).

[Mit90] J. Mitchell. Type systems for programming languages. In *Handbook of Theoretical Computer Science, Vol. B* (J. van Leeuwen, ed.), 365–458. North Holland (1990).

[Möl87] B. Möller. Higher-Order Algebraic Specifications. Habilitationsschrift, Technische Universität München (1987).

[Nip88] T. Nipkow. Observing nondeterministic data types. *Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane. Springer LNCS 332, 170–183 (1988).

[NO88] P. Nivela and F. Orejas. Initial behaviour semantics for algebraic specifications. *Selected Papers from the 5th Workshop on Specification of Abstract Data Types*, Gullane. Springer LNCS 332, 184–207 (1988).

[ONS91] F. Orejas, M. Navarro and A. Sánchez. Implementation and behavioural equivalence: a survey. *Selected Papers from the 8th Workshop on Specification of Abstract Data Types*, Dourdan. Springer LNCS 655, 93–125 (1991).

[Pho92] W. Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. Report ECS-LFCS-92-208, Univ. of Edinburgh (1992).

[Rei85] H. Reichel. Behavioural validity of conditional equations in abstract data types. *Proc. of the Vienna Conf. on Contributions to General Algebra*, 1984. Teubner-Verlag, 301–324 (1985).

[ST87] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *Journal of Computer and System Sciences* 34:150–178 (1987).

[ST88a] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation* 76:165–210 (1988).

[ST88b] D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: implementations revisited. *Acta Informatica* 25:233–281 (1988).

[ST89] D. Sannella and A. Tarlecki. Toward formal development of ML programs: foundations and methodology. *Joint Conf. on Theory and Practice of Software Development*, Barcelona. Springer LNCS 352, 375–389 (1989).

[ST95] D. Sannella and A. Tarlecki. Model-theoretic foundations for program development: basic concepts and motivation. Report 791, Institute of Computer Science, Polish Academy of Sciences (1995).

[SW83] D. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation. *Proc. 1983 Intl. Conf. on Foundations of Computation Theory*, Borgholm. Springer LNCS 158, 413–427 (1983).

[SW96] D. Sannella and M. Wirsing. Specification languages. Chapter 7 of *Algebraic Foundations of Systems Specification* (E. Astesiano, H.-J. Kreowski and B. Krieg-Brückner, eds.). Springer, to appear (1996).

[Sch94] P.-Y. Schobbens. Second-order proof systems for algebraic specification languages. *Selected Papers from the 9th Workshop on Specification of Abstract Data Types*, Caldes de Malavella. Springer LNCS 785, 321–336 (1994).

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

[Sch92] O. Schoett. Two impossibility theorems on behavioural specification of abstract data types. *Acta Informatica* 29:595–621 (1992).

[Sch77] K. Schütte. *Proof Theory*. Springer (1977).

[Sti92] C. Stirling. Modal and temporal logics for processes. In: *Logics for Concurrency: Structure versus Automata* (F. Moller and G. Birtwistle, eds.). Springer LNCS 1043, 149–237 (1996).

[TvD88] A. Troelstra and D. van Dalen. *Constructivism in Mathematics: An Introduction, Vol. 1.* North-Holland (1988).

[Win93] G. Winskel. *The Formal Semantics of Programming Languages.* MIT Press (1993).