Mind the gap!
Abstract versus concrete models of specifications

Donald Sannella* Andrzej Tarlecki†

Abstract

In the theory of algebraic specifications, many-sorted algebras are used to model programs: the representation of data is arbitrary and operations are modelled as ordinary functions. The theory that underlies the formal development of programs from specifications takes advantage of the many useful properties that these models enjoy.

The models that underlie the semantics of programming languages are different. For example, the semantics of Standard ML uses rather concrete models, where data values are represented as closed constructor terms and functions are represented as "closures". The properties of these models are quite different from those of many-sorted algebras.

This discrepancy brings into question the applicability of the theory of specification and formal program development in the context of a concrete programming language, as has been attempted in the Extended ML framework for the formal development of Standard ML programs. This paper is a preliminary study of the difference between abstract and concrete models of specifications, inspired by the kind of concrete models used in the semantics of Standard ML, in an attempt to determine the consequences of the discrepancy.

1 Introduction

The starting point for work on algebraic specification is the use of *many-sorted algebras* as models of programs. Thus the representation of data values is arbitrary and operations on data are modelled as ordinary set-theoretic functions. This is a natural choice since the primary aim of this work is to provide foundations for the development of programs that are *correct* with respect to a given specification of requirements. Correctness is a property of the input/output behaviour of a program, and in studying correctness it is convenient to abstract away from all other aspects of program behaviour. Models of programs more complicated than many-sorted algebras have

*Laboratory for Foundations of Computer Science, Edinburgh University, Edinburgh, Scotland; e-mail `dts@dcs.ed.ac.uk`. This research was supported by the EC-funded COMPASS Basic Research working group and MeDiCiS Scientific Cooperation Network, by EPSRC grants GR/J07303 and GR/H73103 and by an EPSRC Advanced Fellowship.

†Institute of Informatics, Warsaw University, and Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland; e-mail `tarlecki@mimuw.edu.pl`. This research was supported by the EC-funded COMPASS Basic Research working group and MeDiCiS Scientific Cooperation Network, and by EPSRC grant GR/J07303.

been used to deal with advanced features of programming languages (higher-order functions, infinite behaviour, etc.) but the idea is still to abstract away from details of code and algorithms insofar as this is possible.

There is a well-developed theory of algebraic specification which provides syntax for specifications, a formalization of concepts involved in going from specifications to programs by stepwise refinement, methods for proving correctness of refinement steps, etc. Some recent references are [Wir90], [BKLOS91], [LEW96], [ST95] and [ST9?]. This theory takes full advantage of the properties of many-sorted algebras and uses various standard constructions (quotient, free extension, etc.) for building new algebras by adding to and/or combining others. It is mature enough to be applied to the specification and formal development of programs written using "pure" fragments of programming languages.

Of course, programs that have been formally developed are entirely useless unless they are expressed in some implemented programming language. Since the models that underlie the semantics of programming languages are often rather different from many-sorted algebras, there is a potential problem in ensuring that arguments about correctness in the theory of formal development are valid for the semantics of the programming language.

A concrete instance of this problem arises in our work on specification and formal development of Standard ML (SML) programs [Pau91] using the Extended ML (EML) framework [ST89], [San91], [KST95]. SML has an operational semantics [MTH90] which uses rather concrete models of programs, with data represented as closed constructor terms and functions represented as *closures* (a closure is a λ-term together with an environment binding its free variables). Although it is not difficult to map these concrete models to many-sorted algebras, there is *a priori* no guarantee that this map would preserve and/or reflect important properties of algebras. There are concrete models that do not correspond to any SML program, as well as many algebras that do not correspond to any concrete model, so one question concerns closure of these classes under various algebraic constructions. It is by no means obvious that the concrete models enjoy the particular properties of algebras that are used to explain why formal development in EML yields correct programs.

The situation in SML/EML is an example of a possibly worrying discrepancy between the abstract models used to reason about correctness and the more or less concrete models used to give semantics to programming languages. It seems necessary to check that the process of abstraction has not led us to make false conclusions about the underlying reality. This paper attempts to study this issue, looking to SML and EML for inspiration. This is an awkward thing to study because of the complexity of real programming languages (and SML is no exception) so that some degree of abstraction is unavoidable in the study itself. We therefore proceed by analogy, studying what happens when computability restrictions inspired by those arising in SML are placed on algebras. Even when we refer to SML, we will be considering a "pure" fragment which excludes complications like real numbers, input/output, and polymorphic types. We argue that the analogy is strong enough so that the answers we obtain are valid for SML.

This issue is by no means unique to the specification and formal development of programs using algebraic specifications. The same question arises for any theory of specification and formal development, and more generally whenever an abstract view is taken of a real system, whether it is a set of differential equations describing the forces acting on a bridge or a mathematical model of program behaviour.

Much of the paper is devoted to the development of a satisfactory notion of "algebra coded by a program" and an investigation of its properties. We begin with *modest algebras*: partial algebras that come equipped with a computable implementation using natural number encodings of data (Sect. 2). A special case is the $\mathcal{A}0$-*extensions*, modest algebras in which the implementation of certain types is fixed (Sect. 3), which appear to capture SML-codable algebras closely enough for the purposes of this paper. The meaning of formulae in modest algebras is defined in Sect. 4, and then in Sect. 5 we explore which constructions on partial algebras carry over to modest algebras. In Sect. 7 we consider the question of which partial algebras can be given the structure of a modest algebra. We proceed to define a simple specification language (Sect. 8) which can be interpreted in various institutions of partial and modest algebras (cf. Sect. 6) and then finally examine the gap between the interpretation of specifications using partial algebras and using modest algebras or modest $\mathcal{A}0$-extensions (Sect. 9). Our conclusion is that the discrepancy has no alarming consequences, but the analysis reveals some interesting choices which influence the completeness and flexibility of the formal program development process (Sect. 10).

2 Modest algebras

This section is devoted to the search for a formalization of the concept of "algebra coded by a program". Since this is sensitive to (at least) what we mean by "program", we cannot give a definitive answer. The notion of *modest* $\mathcal{A}0$-*extension*, presented in the next section, is an approximation that seems to be adequate for our purposes.

The reader is assumed to be familiar with usual notions of many-sorted signature, signature morphism, many-sorted algebra, homomorphism, reduct of a Σ'-algebra A' along a signature morphism $\sigma : \Sigma \to \Sigma'$ to yield the Σ-algebra $A'|_\sigma$ (written $A'|_\Sigma$ when σ is an inclusion) and similarly for Σ'-homomorphisms, etc.; see e.g. [Wir90]. We use ordinary many-sorted signatures with first-order operation symbols, but restrict to signatures having a countable number of operation symbols. Call the category of such signatures **AlgSig**, and let $\Sigma = \langle S, \Omega \rangle$ be a signature in $|\mathbf{AlgSig}|$. We use partial algebras having countable carrier sets, and so-called *strong* homomorphisms between such algebras, which preserve *and reflect* definedness of operations. The category of such Σ-algebras and Σ-homomorphisms is called $\mathbf{PAlg}(\Sigma)$. We also use *strong* congruences on partial algebras, which are closed under application of operations and preserve and reflect their definedness. Throughout, we use the arrow $\to$ for total functions (and in types of operation symbols) and the arrow $\rightharpoonup$ for partial functions.

A modest algebra is a partial algebra equipped with a computable "implementation": its data values are encoded as natural numbers, and its operations are mirrored by partial recursive functions over these encodings.

Definition 2.1 *A* modest Σ-algebra $\mathcal{A}$ *is a partial* Σ*-algebra* A *together with:*

- *a recursive set* $\bar{s} \subseteq \mathbb{N}$ *of* codes *for each* $s \in S$*;*
- *a total surjective* decoding function $[\cdot]_s : \bar{s} \to |A|_s$ *for each* $s \in S$*; and*

- *a partial recursive* tracking function $\bar{f} : \bar{s}_1 \times \cdots \times \bar{s}_n \rightharpoonup \bar{s}$ *for each* $f : s_1 \times \cdots \times s_n \to s$ *in* Ω

such that for any such f *and* $m_1 \in \bar{s}_1, \ldots, m_n \in \bar{s}_n$, $\bar{f}(m_1, \ldots, m_n)$ *is defined iff* $f_A([m_1]_{s_1}, \ldots, [m_n]_{s_n})$ *is defined and then* $[\bar{f}(m_1, \ldots, m_n)]_s = f_A([m_1]_{s_1}, \ldots, [m_n]_{s_n})$. *We drop subscripts from decoding functions when they are determined by the context. We write* $|\mathcal{A}|$ *for* A *and say that* $\mathcal{A}$ *is* over A, *and we write* $\bar{s}_{\mathcal{A}}$, $[\cdot]_{\mathcal{A}}$ *and* $\bar{f}_{\mathcal{A}}$ *when* $\mathcal{A}$ *is not obvious.*

This definition is intended to reflect the way that programs in a language like Standard ML "code" algebras. Think of each sort s as an SML type, with the codes in $\bar{s}$ being Gödel-style encodings of constructor terms of this type. If s is a function type, the codes in $\bar{s}$ are encodings of λ-terms. The requirement that $\bar{s}$ be recursive stems from the fact that typechecking in SML is decidable: given an SML term, we can decide whether or not it is a constructor term (or λ-term) of a given type over a given set of constructors. The tracking functions need to be partial recursive to make them SML-implementable. For any built-in type or user-defined concrete data type s whose definition does not involve abstract types or function types (in SML parlance, an "eqtype"), $[\cdot]$ is a bijection and so equality in $|A|_s$ corresponds to identity of constructor terms of type s. SML then provides a function $= : s \times s \to \texttt{bool}$ that decides the equality. But for a function type $s \to t$, if the interpretation taken in $|A|_{s\to t}$ is the usual set-theoretic function space (restricted to denotable functions) then $[\cdot]_{s\to t} : \overline{s \to t} \to |A|_{s\to t}$ is in general no longer a bijection since it maps extensionally equal λ-terms to the same set-theoretic function. The *kernel* of $[\cdot]_{s\to t}$, $\equiv_{[\cdot]_{s\to t}} =_{\text{def}} \{\langle m, m' \rangle \in \overline{s \to t} \times \overline{s \to t} \mid [m] = [m']\}$, is typically not even semi-decidable.

The term "modest algebra" comes from the term "modest set" [Ros90] where however the requirement that the sets of codes be recursive is absent. The function space used there would not satisfy such an assumption even if its domain and range did, since it is not decidable whether or not a given partial recursive function on $\mathbb{N}$ is a function between two given recursive sets. As explained above, we think of (codes of) statically well-typed λ-terms as codes for values of the function space.

The restriction to countable algebras is forced by the definition of modest algebra: obviously, uncountable carrier sets cannot be encoded using natural numbers (although see [SHT95] for a theory of computable approximations of uncountable algebras). Some other simple properties of modest algebras follow directly from the definition. We start with an alternative formulation of the definition itself.

Proposition 2.2 *The families* $\bar{S} = \langle$*recursive* $\bar{s} \subseteq \mathbb{N}\rangle_{s\in S}$, $[\cdot] = \langle [\cdot]_s : \bar{s} \to |A|_s \rangle_{s\in S}$ *and* $\bar{\Omega} = \langle$*partial recursive* $\bar{f} : \bar{s}_1 \times \cdots \times \bar{s}_n \rightharpoonup \bar{s}\rangle_{f:s_1\times\cdots\times s_n\to s\in\Omega}$ *form a modest* Σ*-algebra* $\mathcal{A}$ *over a partial* Σ*-algebra* A *iff* $[\cdot] : \bar{\mathcal{A}} \to A$ *is a surjective* Σ*-homomorphism, where* $\bar{\mathcal{A}}$ *is the partial* Σ*-algebra with carriers* $\bar{S}$ *and operations* $\bar{\Omega}$. □

This means that we can view any modest algebra $\mathcal{A}$ as a surjective homomorphism from $\bar{\mathcal{A}}$ to $|\mathcal{A}|$, and vice versa. It also says that one may view $\bar{\mathcal{A}}$ as a concrete implementation of $|\mathcal{A}|$, with $[\cdot]$ as the so-called "abstraction function" [Hoa72].

Corollary 2.3 *Let* $\bar{\mathcal{A}}$ *together with* $[\cdot]$ *form a modest* Σ*-algebra over* A. *If* $j : A \to B$ *is a surjective* Σ*-homomorphism then* $\bar{\mathcal{A}}$ *together with* $j \circ [\cdot] : \bar{\mathcal{A}} \to B$ *forms a modest* Σ*-algebra over* B. □

Proposition 2.4 *Let A be a partial Σ-algebra and $\bar{A}$ be as above. There is a surjective Σ-homomorphism $[\cdot] : \bar{A} \to A$ (i.e. a modest Σ-algebra over A with codes $\bar{A}$ and decoding function $[\cdot]$) iff there is a Σ-congruence $\approx$ on $\bar{A}$ such that $A \cong \bar{A}/{\approx}$.* □

Proposition 2.5 *Let $\bar{A}$ and $[\cdot] : \bar{A} \to A$ be as in Prop. 2.2, but with $\bar{S} = \langle$r.e. $\bar{s} \subseteq \mathbb{N}\rangle_{s \in S}$. Then there is a modest Σ-algebra over A.*

Proof: *Since each $\bar{s}$ is r.e. there is a recursive bijection $e_s : \bar{s}' \to \bar{s}$ where $\bar{s}' \subseteq \mathbb{N}$ is recursive. Define the decoding functions $[\cdot]' = \langle [\cdot]'_s : \bar{s}' \to |A|_s \rangle_{s \in S}$ by $[\cdot]'_s = [\cdot]_s \circ e_s$, and the tracking functions $\bar{\Omega}' = \langle \bar{f}' \rangle_{f : s_1 \times \cdots \times s_n \to s \in \Omega}$ by $\bar{f}'(m'_1, \ldots, m'_n) = e_s^{-1}(\bar{f}(e_{s_1}(m'_1), \ldots, e_{s_n}(m'_n)))$ for $m'_1 \in \bar{s}'_1, \ldots, m'_n \in \bar{s}'_n$. It is easy to see that $\bar{S}' = \langle \bar{s}' \rangle_{s \in S}$, $\bar{\Omega}'$ and $[\cdot]'$ form a modest Σ-algebra over A.* □

Definition 2.6 *A* modest Σ-homomorphism $h : \mathcal{A} \to \mathcal{B}$ *is a Σ-homomorphism between the underlying partial algebras $|h| : |\mathcal{A}| \to |\mathcal{B}|$ together with a family of partial recursive* tracking functions *$\langle \bar{h}_s : \mathbb{N} \rightharpoonup \mathbb{N} \rangle_{s \in S}$ such that for each $s \in S$ and $m \in \bar{s}_{\mathcal{A}}$, $\bar{h}_s(m)$ is defined, $\bar{h}_s(m) \in \bar{s}_{\mathcal{B}}$, and $[\bar{h}_s(m)]_{\mathcal{B}} = |h|_s([m]_{\mathcal{A}})$. We say that h is* over *$|h|$. Modest Σ-algebras and modest Σ-homomorphisms (with the obvious composition) form a category called $\mathbf{MAlg}(\Sigma)$, and $|\cdot| : \mathbf{MAlg}(\Sigma) \to \mathbf{PAlg}(\Sigma)$ is a functor.*

Proposition 2.7 *A modest Σ-isomorphism $h : \mathcal{A} \to \mathcal{B}$ (i.e. an isomorphism in $\mathbf{MAlg}(\Sigma)$) is an isomorphism $|h| : |\mathcal{A}| \to |\mathcal{B}|$ in $\mathbf{PAlg}(\Sigma)$ together with a (total) recursive S-sorted bijection on $\mathbb{N}$ that tracks $|h|$.* □

The usual definitions of reduct of a partial Σ'-algebra and Σ'-homomorphism along a signature morphism $\sigma : \Sigma \to \Sigma'$ extend to modest Σ'-algebras and their homomorphisms, e.g. using the formulation of modest algebras given by Prop. 2.2: the σ-reduct of $[\cdot] : \bar{\mathcal{A}}' \to A'$ is $[\cdot]|_\sigma : \bar{\mathcal{A}}'|_\sigma \to A'|_\sigma$. We write $\mathcal{A}'|_\sigma$, or $\mathcal{A}'|_\Sigma$ when σ is an inclusion. This, together with the category $\mathbf{MAlg}(\Sigma)$ for each signature Σ, gives a contravariant functor $\mathbf{MAlg} : \mathbf{AlgSig}^{op} \to \mathbf{Cat}$.

There is a well-developed theory of computable algebra, beginning with the work of Mal'cev [Mal61] (cf. [Rab60]) and including many papers on the expressive power of algebraic specification methods by Bergstra and Tucker, see e.g. [BT87]. For a recent overview see [SHT95]. Although the *total* modest algebras are exactly the so-called "effectively numbered algebras" of [SHT95], there has been little attention paid to the counterpart for partial algebras. Furthermore, it is common to impose computability restrictions on the decoding functions (e.g. the "computably numbered algebras" of [SHT95] are total modest algebras such that the kernel $\equiv_{[\cdot]_s}$ is decidable for all sorts $s \in S$); these apply to *all* sorts while we have seen that in the SML context such restrictions are appropriate for some sorts (for example, those corresponding to eqtypes) but not for others (for example, those corresponding to function types). Another difference is that while [SHT95] concentrates on *modestizable* algebras, where the encoding structure remains implicit (see Sect. 7 below), we deal with *modest* algebras, which contain the encoding structure explicitly.

3 Modest $\mathcal{A}0$-extensions

Despite the comments above concerning the relationship between modest algebras and SML, modest algebras do not capture only SML-codable algebras.

Counterexample 3.1 *Let A be a two-sorted algebra with carriers $|A|_s = |A|_t = \mathbb{N}$ such that $f_A : |A|_s \to |A|_t$ is a total non-computable bijection and f is the only operation symbol. A modest algebra over A is given by taking $\bar{s} = \bar{t} = \mathbb{N}$, $[\cdot]_s = f^{-1}$ and $[\cdot]_t$ and the tracking function $\bar{f}$ to be the identity.* □

The question of which partial algebras can and cannot be given the structure of a modest algebra will be treated in Sect. 7.

A further reason for the mismatch is that SML insists on a particular interpretation of certain types, the so-called "pervasive" types like `string` and `bool` whose implementation is fixed by the system. Suppose that $\Sigma0 = \langle S0, \Omega0 \rangle$ is a signature containing the sorts and operation symbols whose interpretation we want to fix as the one given by a particular modest $\Sigma0$-algebra $\mathcal{A}0$. We assume that $\equiv_{[\cdot]_s}$ is decidable for all $s \in S0$; then there is no loss of generality in assuming that $[\cdot]_s$ is a bijection for all $s \in S0$ so we make this assumption. We assume that $\Sigma0$ contains at least the sorts *bool* (Booleans) and *nat* (natural numbers) with the usual operations ($\mathit{true}, \mathit{false} : \to \mathit{bool}$, $0 : \to \mathit{nat}$, $+ : \mathit{nat} \times \mathit{nat} \to \mathit{nat}$, etc.) and that the interpretations of these in $|\mathcal{A}0|$ are as usual. It follows that all values in $|\mathcal{A}0|$ are $\Sigma0$-reachable. We restrict attention to signatures Σ that extend $\Sigma0$. Let $\mathbf{AlgSig}_{\Sigma0}$ be the category of signatures extending $\Sigma0$, with signature morphisms that are the identity on $\Sigma0$.

Definition 3.2 *Let Σ be a signature extending $\Sigma0$. A modest Σ-algebra $\mathcal{A}$* extends $\mathcal{A}0$ *($\mathcal{A}$ is a* modest $\mathcal{A}0$-extension*) if there is a modest $\Sigma0$-isomorphism $h : \mathcal{A}|_{\Sigma0} \to \mathcal{A}0$ such that $\bar{h}$ is the $S0$-sorted identity function on $\mathbb{N}$.*

Note that this fixes the *concrete implementation* of $\Sigma0$ while fixing its interpretation on the *abstract* level only up to isomorphism: if $\mathcal{A}$ is an $\mathcal{A}0$-extension then $\overline{\mathcal{A}|_{\Sigma0}} = \overline{\mathcal{A}0}$ while $|\mathcal{A}|_{\Sigma0}| \cong |\mathcal{A}0|$.

If $\mathcal{A}0$ is the built-in implementation of the pervasive types of SML, then for the purposes of this paper we will identify SML-codable algebras with modest $\mathcal{A}0$-extensions. The correspondence is not exact — there are modest $\mathcal{A}0$-extensions that are not SML-codable, since we have placed no computability restrictions on the kernel $\equiv_{[\cdot]_s}$ for $s \notin S0$ — but because of the complexity of the semantics of SML [MTH90], capturing exactly the class of SML-codable algebras (and proving that we have done so) would be very difficult. However, the match is close enough that the ideas and results in the sequel should apply to SML. On one hand, the class of modest $\mathcal{A}0$-extensions is small enough to expose some problems when compared with the class of partial algebras (see Sects. 7 and 9 below); on the other hand it is large enough to cover the SML-codable algebras. The reader is encouraged to check that the class of SML programs is closed under the main constructions on modest $\mathcal{A}0$-extensions we consider: reduct, amalgamation and definitional extension, cf. Sect. 5.

Definition 3.3 *Let Σ be a signature extending $\Sigma0$. $\mathbf{MAlg}_{\mathcal{A}0}(\Sigma)$ is the subcategory of $\mathbf{MAlg}(\Sigma)$ with modest $\mathcal{A}0$-extensions as objects, and as morphisms modest Σ-homomorphisms h such that $h|_{\Sigma0}$ is tracked by the $S0$-sorted identity function. This extends to a contravariant functor $\mathbf{MAlg}_{\mathcal{A}0} : \mathbf{AlgSig}_{\Sigma0}^{op} \to \mathbf{Cat}$.*

At the abstract level of partial algebras we will similarly concentrate on extensions (up to isomorphism) of the abstract interpretation of built-in sorts and operations given by $A0 =_{\text{def}} |\mathcal{A}0|$.

Definition 3.4 *Let $\Sigma \in |\mathbf{AlgSig}_{\Sigma 0}|$ be a signature extending $\Sigma 0$. A partial Σ-algebra* extends $A0$ *if $A|_{\Sigma 0} \cong A0$. $\mathbf{PAlg}_{A0}(\Sigma)$ is the subcategory of $\mathbf{PAlg}(\Sigma)$ with partial $A0$-extensions as objects, and as morphisms all Σ-homomorphisms h such that $h|_{\Sigma 0}$ is a $\Sigma 0$-isomorphism. This extends to a contravariant functor $\mathbf{PAlg}_{A0} : \mathbf{AlgSig}^{op}_{\Sigma 0} \to \mathbf{Cat}$.*

Proposition 3.5 *For any signature $\Sigma \in |\mathbf{AlgSig}_{\Sigma 0}|$ extending $\Sigma 0$, $|\cdot| : \mathbf{MAlg}(\Sigma) \to \mathbf{PAlg}(\Sigma)$ from Def. 2.6 restricts to a functor $|\cdot| : \mathbf{MAlg}_{\mathcal{A}0}(\Sigma) \to \mathbf{PAlg}_{A0}(\Sigma)$.* $\square$

4 Terms and formulae in modest algebras

Let X be an S-sorted set of variables. The (total) algebra $T_\Sigma(X)$ of Σ-terms with variables in X and the value t^v_A of a term t in a partial Σ-algebra A under a (total) valuation $v : X \to |A|$ are as usual. With evaluation of Σ-terms in a modest Σ-algebra $\mathcal{A}$, we have a choice between evaluation in the underlying partial algebra or using the tracking functions. The latter corresponds to ordinary evaluation in the partial algebra $\bar{\mathcal{A}}$ given by Prop. 2.2.

Definition 4.1 *A* valuation *of variables X in $\mathcal{A}$ is an S-sorted (total) function $v = \langle v_s : X_s \to \bar{s} \rangle_{s \in S}$. The* (extensional) value *of t* under v in $\mathcal{A}$, *written $t^v_{\mathcal{A}}$, is $t^{[\cdot] \circ v}_{|\mathcal{A}|}$. If t is ground (does not contain any variables) then we can write $t_{\mathcal{A}}$.*

We use formulae of first-order logic where the atomic formulae are definedness formulae and (strong) equations. Satisfaction of Σ-formulae in a modest Σ-algebra $\mathcal{A}$ may be defined either on the level of the underlying partial algebra or on the level of the tracking functions; the latter corresponds to satisfaction in $\bar{\mathcal{A}}$.

Definition 4.2 *Let φ be a formula with free variables in X and let v be a valuation of X in $\mathcal{A}$. We define* (extensional) satisfaction of φ by $\mathcal{A}$ under v, *written $\mathcal{A} \models_v \varphi$, by induction on the structure of φ. Here are the cases for atomic formulae:*

$$\begin{array}{lll} \mathcal{A} \models_v D(t) & \textit{iff} & t^v_{\mathcal{A}} \textit{ is defined} \\ \mathcal{A} \models_v t = u & \textit{iff} & t^v_{\mathcal{A}}, u^v_{\mathcal{A}} \textit{ are both undefined, or are both defined and equal} \end{array}$$

Thus $\mathcal{A} \models_v \varphi$ iff $|\mathcal{A}| \models_{[\cdot] \circ v} \varphi$ in the usual sense. As usual, $\mathcal{A} \models \varphi$ iff $\mathcal{A} \models_v \varphi$ for all valuations v.

5 Constructions on modest algebras

In this section we will discuss some constructions on modest algebras (reduct, definitional extension, and amalgamation) that are used in the formal development process to be presented in Sect. 10. We will also have a look at some other standard constructions (quotient and subalgebra). Although these do not appear explicitly in our formalization of the development process, they are taken for granted in work on algebraic specification. Some of these standard constructions carry over easily to modest algebras; others carry over only under certain additional conditions.

We have already seen that the reduct of a partial Σ'-algebra and Σ'-homomorphism along a signature morphism $\sigma : \Sigma \to \Sigma'$ extend to modest Σ'-algebras and modest Σ'-homomorphisms. Likewise for modest $\mathcal{A}0$-extensions and their homomorphisms, provided that σ is the identity on $\Sigma 0$.

Definition 5.1 *Let $\approx$ be a congruence on $|\mathcal{A}|$, where $\mathcal{A}$ is given by the surjective homomorphism $[\cdot] : \bar{A} \to |\mathcal{A}|$. The* quotient *of $\mathcal{A}$ by $\approx$, written $\mathcal{A}/\approx$, is the modest Σ-algebra given by the surjective homomorphism $[\cdot]_\approx \circ [\cdot] : \bar{A} \to |\mathcal{A}|/\approx$.*

Proposition 5.2 *If $\mathcal{A}$ extends $\mathcal{A}0$ and $\approx_s$ is the identity for all $s \in S0$ then $\mathcal{A}/\approx$ extends $\mathcal{A}0$.* □

Definition 5.3 *$\mathcal{B}$ is a* modest subalgebra *of a modest Σ-algebra $\mathcal{A}$ if: $|\mathcal{B}|$ is a subalgebra of $|\mathcal{A}|$; $\bar{\mathcal{B}}$ is a subalgebra of $\bar{\mathcal{A}}$; and $([\cdot]_\mathcal{B})_s = ([\cdot]_\mathcal{A})_s \restriction \bar{s}_\mathcal{B}$ for all $s \in S$. A modest Σ-algebra $\mathcal{A}$ is* reachable *if for any $s \in S$ and $m \in \bar{s}$, there is a ground Σ-term t such that $t_{\bar{\mathcal{A}}} = m$.*

Proposition 5.4 *If $\mathcal{A}$ is reachable then $\mathcal{A}$ has no proper modest subalgebra.* □

For partial Σ-algebras, the converse of Prop. 5.4 holds as well, but this does not extend to modest Σ-algebras as the following counterexample shows.

Counterexample 5.5 *Let Σ have sorts s and s' and operation symbols $0 : \to s$, $succ : s \to s$, $f : s \to s'$ and $g : s' \times s \to s'$. For any two r.e. sets $X, Y \subseteq \mathbb{N}$, let $A_{X,Y}$ be the Σ-algebra such that $|A|_s = |A|_{s'} = \mathbb{N}$ with 0 and $succ$ interpreted as usual and*

$$f_{A_{X,Y}}(n) = \begin{cases} n & \text{if } n \in X \\ \text{undefined} & \text{otherwise} \end{cases} \qquad g_{A_{X,Y}}(n, m) = \begin{cases} m & \text{if } n \in Y \\ \text{undefined} & \text{otherwise} \end{cases}$$

Let $\mathcal{A}$ be the modest Σ-algebra over A with $\bar{s} = \bar{s}' = \mathbb{N}$, $[\cdot]_s$ and $[\cdot]_{s'}$ the identity, and the evident tracking functions for 0, $succ$, f and g ($f_{A_{X,Y}}$ and $g_{A_{X,Y}}$ are partial recursive since X and Y are r.e.). $\mathcal{A}_{X,Y}$ is not reachable if X is a proper subset of $\mathbb{N}$.

Now choose X and Y so that they are disjoint and not recursively separable, that is, for any recursive set $Q \subseteq \mathbb{N}$, if $X \subseteq Q$ then $Q \cap Y \neq \emptyset$ (such sets X and Y exist, see e.g. [Rog67], Chap. 7, Th. XII). Let $\mathcal{B}$ be a modest subalgebra of $\mathcal{A}_{X,Y}$. Then $X \subseteq |\mathcal{B}|_{s'}$, and so there exists $y \in Y$ such that $y \in |\mathcal{B}|_{s'}$ (since $|\mathcal{B}|_{s'} = \bar{s}'_\mathcal{B}$ is a recursive subset of $\mathbb{N}$ and X and Y are not recursively separable). But then, for each $n \in \mathbb{N}$, $g_{|\mathcal{B}|}(y, n) = n$, and so $n \in |\mathcal{B}|_{s'}$. Thus $|\mathcal{B}|_{s'} = \mathbb{N} = |\mathcal{A}_{X,Y}|_{s'}$, and so $\mathcal{B} = \mathcal{A}_{X,Y}$. Therefore $\mathcal{A}_{X,Y}$ has no proper modest subalgebra. □

Any partial algebra has a unique reachable subalgebra. Counterexample 5.5 shows that this is not the case for modest algebras. For any modest Σ-algebra $\mathcal{A}$, the sets $\langle\{t_{\bar{\mathcal{A}}} \mid t \in |T_\Sigma|_s\} \subseteq \bar{s}\rangle_{s \in S}$ are r.e., and so using Prop. 2.5 it is possible to construct a reachable modest algebra from $\mathcal{A}$ and this family of r.e. sets. But the construction involves a re-coding so the result will not be a modest subalgebra of $\mathcal{A}$ in general.

The amalgamated union construction, used to combine algebras over different signatures having a common reduct, generalizes to modest algebras.

Proposition 5.6 *Consider a pushout in the category* **AlgSig***:*

$$\begin{tikzcd} & \Sigma' & \\ \Sigma_1 \arrow[ur, "\sigma'_1"] & & \Sigma_2 \arrow[ul, "\sigma'_2"'] \\ & \Sigma \arrow[ul, "\sigma_1"] \arrow[ur, "\sigma_2"'] & \end{tikzcd}$$

Then, for any modest Σ_1*-algebra* $\mathcal{A}_1$ *and modest* Σ_2*-algebra* $\mathcal{A}_2$ *such that* $\mathcal{A}_1|_{\sigma_1} = \mathcal{A}_2|_{\sigma_2}$*, there exists a unique modest* Σ'*-algebra* $\mathcal{A}'$ *such that* $\mathcal{A}'|_{\sigma'_1} = \mathcal{A}_1$ *and* $\mathcal{A}'|_{\sigma'_2} = \mathcal{A}_2$*. If* $\mathcal{A}_1$*,* $\mathcal{A}_2$ *and* $\mathcal{A}_1|_{\sigma_1}$ *are all modest* $\mathcal{A}0$*-extensions then so is* $\mathcal{A}'$*.* □

In Sect. 4 we have introduced formulae, which can be used as axioms to *specify* required properties of modest algebras. The quotient construction and amalgamation as discussed above are examples of how modest algebras can be modified and combined. To build modest algebras "from scratch" we need some elementary ways to *define* new modest algebras and/or their components. As a simple example, let us now consider *definitions* of operations.

Given a signature Σ, a Σ-*definition* has the form

$$\mathsf{fun}\ f(x_1{:}s_1, \ldots, x_n{:}s_n) = t{:}s$$

where f does not occur in Σ, t is a term of sort s over the signature $\Sigma[f] =_{\mathrm{def}} \Sigma \cup \{f : s_1 \times \cdots \times s_n \to s\}$ (so t may refer to f) with variables $\{x_1 : s_1, \ldots, x_n : s_n\}$. Mutual recursion is not provided but adding it should not introduce any new problems.

Such a definition determines an extension of an ordinary partial algebra, which can be used to explain the effect of definitions on modest algebras.

Definition 5.7 *Given a partial* Σ*-algebra* A*, a* Σ*-definition*

$$\mathsf{fun}\ f(x_1{:}s_1, \ldots, x_n{:}s_n) = t{:}s$$

determines a sequence of partial $\Sigma[f]$*-algebras* $A_0, A_1, A_2, \ldots$ *extending* A*. For each* $j \geq 0$*,* $A_j|_\Sigma = A$*, and for every* $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$*,* $f_{A_0}(a_1, \ldots, a_n)$ *is undefined and for* $j \geq 0$*,* $f_{A_{j+1}}(a_1, \ldots, a_n) = t_{A_j}^{\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}}$*. The* extension of A by $\mathsf{fun}\ f(x_1{:}s_1, \ldots, x_n{:}s_n) = t{:}s$*, written (somewhat imprecisely)* $A[f{\leftarrow}t]$*, is the partial* $\Sigma[f]$*-algebra such that* $A[f{\leftarrow}t]|_\Sigma = A$ *and for every* $a_1 \in |A|_{s_1}, \ldots, a_n \in |A|_{s_n}$ *and* $a \in |A|_s$*,* $f_{A[f\leftarrow t]}(a_1, \ldots, a_n) = a$ *iff* $f_{A_j}(a_1, \ldots, a_n) = a$ *for some* $j \geq 0$*.*

(This is well-defined, since f_{A_j}, $j \geq 0$, form a chain w.r.t. the evident ordering between partial functions.)

Proposition 5.8 *Given a* Σ*-homomorphism* $h : A \to B$ *between partial* Σ*-algebras and a* Σ*-definition* $\mathsf{fun}\ f(x_1{:}s_1, \ldots, x_n{:}s_n) = t{:}s$*,* $h : A[f{\leftarrow}t] \to B[f{\leftarrow}t]$ *is a* $\Sigma[f]$*-homomorphism.* □

Notice that for any modest Σ-algebra $\mathcal{A}$, $f_{\bar{\mathcal{A}}[f\leftarrow t]}$ is a partial recursive function. Then, since by Prop. 2.2 any modest Σ-algebra $\mathcal{A}$ may be identified with the surjective homomorphism $[\cdot] : \bar{\mathcal{A}} \to |\mathcal{A}|$, Prop. 5.8 justifies the following definition:

Definition 5.9 *Given any modest* Σ*-algebra* $\mathcal{A}$ *and* Σ*-definition*

$$\mathsf{fun}\ f(x_1{:}s_1, \ldots, x_n{:}s_n) = t{:}s$$

the extension of $\mathcal{A}$ by $\mathsf{fun}\ f(x_1{:}s_1, \ldots, x_n{:}s_n) = t{:}s$*, written (somewhat imprecisely)* $\mathcal{A}[f{\leftarrow}t]$*, is the modest* $\Sigma[f]$*-algebra given by the surjective* $\Sigma[f]$*-homomorphism* $[\cdot] : \bar{\mathcal{A}}[f{\leftarrow}t] \to |\mathcal{A}|[f{\leftarrow}t]$*.*

Proposition 5.10 *If* $\mathcal{A}$ *is a modest* $\mathcal{A}0$*-extension then so is* $\mathcal{A}[f{\leftarrow}t]$*.* □

6 Institutions

In the previous sections we have introduced concepts which can be put together to form a number of logical systems we might want to use in the process of software specification and development. To systematize this a bit, let us first recall the notion of an *institution*, which formalizes the concept of a logical system based on a model-theoretic view of logic. This will also allow us to take advantage of "institution-independent" concepts and results in the literature.

Definition 6.1 ([GB92]) *An* institution $\mathbf{I}$ *consists of: a category* $\mathbf{Sign_I}$ *of* signatures*; functors* $\mathbf{Sen_I}\colon \mathbf{Sign_I} \to \mathbf{Set}$ *and* $\mathbf{Mod_I}\colon \mathbf{Sign_I}^{op} \to \mathbf{Cat}$*, giving for each signature* $\Sigma \in |\mathbf{Sign_I}|$ *a set* $\mathbf{Sen_I}(\Sigma)$ *of* Σ-sentences *and a category* $\mathbf{Mod_I}(\Sigma)$ *of* Σ-models *respectively; and a family* $\langle \models_{\mathbf{I},\Sigma} \subseteq |\mathbf{Mod_I}(\Sigma)| \times \mathbf{Sen_I}(\Sigma)\rangle_{\Sigma \in |\mathbf{Sign_I}|}$ *of* satisfaction relations *such that for any signature morphism* $\sigma\colon \Sigma \to \Sigma'$*,* Σ*-sentence* $\varphi \in \mathbf{Sen_I}(\Sigma)$ *and* Σ'*-model* $M' \in |\mathbf{Mod_I}(\Sigma')|$*,* $M' \models_{\mathbf{I},\Sigma'} \mathbf{Sen_I}(\sigma)(\varphi)$ *iff* $\mathbf{Mod_I}(\sigma)(M') \models_{\mathbf{I},\Sigma} \varphi$*.*

We will omit the subscript $\mathbf{I}$ when the institution is obvious, and then for any signature morphism $\sigma : \Sigma \to \Sigma'$ and Σ'-model $M' \in |\mathbf{Mod}(\Sigma')|$, we will write $M'|_\sigma$ for $\mathbf{Mod}(\sigma)(M')$. Then, for any Σ-model $M \in |\mathbf{Mod}(\Sigma)|$, Σ-sentence $\varphi \subseteq \mathbf{Sen}(\Sigma)$ and set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$, we will write $M \models \Phi$ and $\Phi \models \varphi$ with the usual meaning. The latter notation introduces the crucial concept of *semantic entailment*.

In the previous sections we have in effect defined four institutions:

$\mathbf{I_P}$ (partial algebras): The category of signatures is $\mathbf{AlgSig}$; sentences are closed first-order formulae with equality and definedness formulae; the model functor is $\mathbf{PAlg} : \mathbf{AlgSig}^{op} \to \mathbf{Cat}$; and the satisfaction relation is the usual satisfaction of first-order formulae in partial algebras.

$\mathbf{I_{A0}}$ (partial $A0$-extensions): Like $\mathbf{I_P}$, but the category of signatures is $\mathbf{AlgSig}_{\Sigma 0}$ and the model functor is $\mathbf{PAlg}_{A0} : \mathbf{AlgSig}_{\Sigma 0}^{op} \to \mathbf{Cat}$.

$\mathbf{I_M}$ (modest algebras): Like $\mathbf{I_P}$, but the model functor is $\mathbf{MAlg} : \mathbf{AlgSig}^{op} \to \mathbf{Cat}$ and the satisfaction relation is the extensional satisfaction of first-order formulae in modest algebras.

$\mathbf{I}_{\mathcal{A}0}$ (modest $\mathcal{A}0$-extensions): Like $\mathbf{I_M}$, but the category of signatures is $\mathbf{AlgSig}_{\Sigma 0}$ and the model functor is $\mathbf{MAlg}_{\mathcal{A}0} : \mathbf{AlgSig}_{\Sigma 0}^{op} \to \mathbf{Cat}$.

An important property of institutions is whether they admit amalgamation of models, used in the process of modular composition of programs, cf. [EM85], [ST88b].

Definition 6.2 *An institution* $\mathbf{I}$ admits amalgamation[1] *if its category of signatures* $\mathbf{Sign}$ *has pushouts and its model functor* $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ *maps pushouts in* $\mathbf{Sign}$ *to pullbacks in* $\mathbf{Cat}$*.*

The most important consequence of this property is that amalgamation of models (and of model morphisms), as spelled out in Prop. 5.6 for modest algebras, is then unambiguously defined.

Proposition 6.3 $\mathbf{I_P}$*,* $\mathbf{I_{A0}}$*,* $\mathbf{I_M}$ *and* $\mathbf{I}_{\mathcal{A}0}$ *admit amalgamation.* □

[1]Such institutions were called *semiexact* in [DGS93].

7 Modestizable algebras

In Sect. 2 we suggested that modest algebras (or, to be more precise, modest $\mathcal{A}0$-extensions) can be viewed as a representation of the structures that arise in a real programming language like SML. Another approach would be to use partial algebras, but restrict attention only to those that are codable in the real programming language.

Definition 7.1 *A partial Σ algebra $A \in |\mathbf{PAlg}(\Sigma)|$ is* modestizable *if there exists a modest Σ-algebra $\mathcal{A} \in |\mathbf{MAlg}(\Sigma)|$ so that $|\mathcal{A}| = A$. $\mathbf{PAlg}_{\exists\mathbf{M}}(\Sigma)$ denotes the category of modestizable partial Σ-algebras, and there is an obvious functor $\mathbf{PAlg}_{\exists\mathbf{M}} : \mathbf{AlgSig}^{op} \to \mathbf{Cat}$.*

The first question to investigate is how many partial algebras are modestizable. We begin with two easy positive statements:

Proposition 7.2 *Any finite partial algebra is modestizable.* $\square$

Proposition 7.3 *Any total algebra $A \in \mathbf{PAlg}(\Sigma)$ is modestizable.*

Proof: *Suppose $A \in |\mathbf{PAlg}(\Sigma)|$ is total. Fix an enumeration $N_s = \{a_1, a_2, \dots\}$ of $|A|_s$ for each $s \in S$; this is possible since $|A|_s$ is required to be countable. Let $N = \langle N_s \rangle_{s \in S}$. Define $\mathcal{A} \in |\mathbf{MAlg}(\Sigma)|$: $|\mathcal{A}| = A$; for any $s \in S$, let $\bar{s}$ be the set of Gödel-style encodings $\ulcorner t \urcorner$ of terms $t \in |T_\Sigma(N)|_s$ and $[\ulcorner t \urcorner]_s = t_A^{id}$; and for any $f : s_1 \times \cdots \times s_n \to s$ in Ω, $\bar{f}(\ulcorner t_1 \urcorner, \dots, \ulcorner t_n \urcorner) = \ulcorner f(t_1, \dots, t_n) \urcorner$.* $\square$

Of course, the carrier of any modestizable algebra is countable — this is why we have restricted attention to countable algebras only. But even under this assumption, not all partial algebras are modestizable:[2]

Counterexample 7.4 *Let Σ_{nonmod} have sort s and operation symbols $0 : \to s$ and $succ : s \to s$. Consider the partial Σ_{nonmod}-algebra A with $|A|_s = \mathbb{N}$, 0 and $succ$ interpreted as usual, and where $f_A : \mathbb{N} \rightharpoonup \mathbb{N}$ is a function with non-r.e. domain, e.g.*

$$f_A(n) = \begin{cases} 0 & \textit{if the nth TM doesn't halt} \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

Now suppose that A is modestizable. Then $n \in dom(f_A)$ iff $f_A(n)$ is defined iff $f_A(succ_A^n(0_A))$ is defined iff $\bar{f}(\overline{succ}^n(\bar{0}))$ is defined iff $n \in dom(\lambda x.\bar{f}(\overline{succ}^x(\bar{0})))$. But $\bar{f}$ and $\overline{succ}$ are partial recursive, so $dom(\lambda x.\bar{f}(\overline{succ}^x(\bar{0})))$ is r.e., and this contradicts the assumption that $dom(f_A)$ is non-r.e. $\square$

The essence of this counterexample is that in any modest algebra, the set of ground terms with defined values is always a recursively enumerable subset of all terms, and so partial algebras for which this property fails are not modestizable.

The situation is even more delicate when modest $\mathcal{A}0$-extensions are considered. One might expect that any modestizable partial $A0$-extension $A \in \mathbf{PAlg}_{A0}(\Sigma)$ is modestizable so as to extend $\mathcal{A}0$. Unfortunately, the following counterexample shows that this is not the case, even if $A|_{\Sigma 0} = A0$.

[2]This example is due to Martin Hofmann and Thomas Streicher.

Counterexample 7.5 *Consider an algebra A_{halt} with $A_{halt}|_{\Sigma 0} = A0$ which contains a total function $halt : nat \to bool$ that solves the halting problem. A_{halt} is modestizable by Prop. 7.3 (assuming that $A0$ is total), but there is no modest algebra extending $\mathcal{A}0$ over A_{halt}.* □

On an abstract level, what is happening when we try to work with modestizable algebras is that we start with an institution $\mathbf{I}$ (in this case, the institution $\mathbf{I_P}$) and then, for each signature $\Sigma \in |\mathbf{Sign}|$, identify a class of "admissible" Σ-models (in this case, the modestizable algebras) which then may be identified with a full subcategory of $\mathbf{Mod}(\Sigma)$. In general, this need not yield an institution: for some signature morphisms $\sigma : \Sigma \to \Sigma'$ and admissible Σ'-models $M' \in |\mathbf{Mod}(\Sigma')|$, the reduct $M'|_\sigma \in |\mathbf{Mod}(\Sigma)|$ may not be admissible. Fortunately, in our case this is not a problem since reducts of modest algebras are well-defined and "commute" with reducts of partial algebras. Consequently, we have an institution:

$\mathbf{I_{\exists M}}$ (modestizable partial algebras): Like $\mathbf{I_P}$, but the model functor is $\mathbf{PAlg_{\exists M}} : \mathbf{AlgSig}^{op} \to \mathbf{Cat}$.

However, the institution induced by choosing a class of admissible models of an institution may lose some of the properties enjoyed by the original institution:[3]

Proposition 7.6 $\mathbf{I_{\exists M}}$ *does not admit amalgamation.*

Proof: *The partial Σ_{nonmod}-algebra A of Counterexample 7.4 is not modestizable, but can easily be presented as the amalgamated union of two modestizable partial algebras: its reduct to the subsignature of Σ_{nonmod} with f removed and its reduct to the subsignature of Σ_{nonmod} with 0 and $succ$ removed.* □

Similarly, in general we might lose the existence of reachable subalgebras, free extensions, quotients, etc.

When we restrict the class of models of an institution, its logical properties may also change. In general, the logical entailments of the original institution remain valid in the institution of "admissible models".

Proposition 7.7 *For any signature Σ, set of Σ-sentences Φ and Σ-sentence φ, if $\Phi \models_{\mathbf{I_P}} \varphi$ then $\Phi \models_{\mathbf{I_{\exists M}}} \varphi$.* □

However, the opposite implication does not hold, since over a smaller class of models more entailments might become true:

Counterexample 7.8 *Consider any enumeration $\langle \mathcal{R}_n \rangle_{n \in \mathbb{N}}$ of all the recursively enumerable subsets of $\mathbb{N}$. Let Σ be a signature with one sort s, constant $0 : \to s$ and two unary operations $succ, f : s \to s$. Put:*

$$\begin{aligned}\Phi = {} & \{D(f(succ^n(0))) \mid n \in \mathbb{N} \wedge n \notin \mathcal{R}_n\} \cup \\ & \{\neg D(f(succ^n(0))) \mid n \in \mathbb{N} \wedge n \in \mathcal{R}_n\} \cup \\ & \{D(0), \forall x{:}s. D(succ(x))\}.\end{aligned}$$

Then, Φ is a set of sentences that is consistent in $\mathbf{I_P}$, and so e.g. $\Phi \not\models_{\mathbf{I_P}} false$.

[3]This point arose in a discussion with José Fiadeiro.

But Φ is inconsistent in $\mathbf{I}_{\exists\mathbf{M}}$, since no partial algebra satisfying Φ is modestizable, by an argument similar to that in Counterexample 7.4: given any modest Σ-algebra $\mathcal{A}$, $X = dom(\lambda n.\bar{f}(\overline{succ}^n(\bar{0})))$ is an r.e. subset of $\mathbb{N}$. Moreover, $n \in X$ iff $\mathcal{A} \models D(f(succ^n(0)))$, and if $\mathcal{A} \models \Phi$ then this holds iff $n \notin \mathcal{R}_n$. So, for $\mathcal{A} \models \Phi$, $n \in X$ iff $n \notin \mathcal{R}_n$ for all $n \in \mathbb{N}$. Therefore X cannot be r.e., and this contradiction proves that Φ has no modest model. Thus $\Phi \models_{\mathbf{I}_{\exists\mathbf{M}}} \varphi$ for any Σ-sentence φ. □

8 Specifications

In [ST88a] we presented a powerful specification framework based on ASL [SW83] which can be used for writing specifications in an arbitrary institution. For the purposes of this paper, let us concentrate on a fragment of this formalism:

Definition 8.1 *The syntax of* specifications *in an institution* $\mathbf{I}$ *is given by the following grammar:*

SP	$::=$	$\langle\Sigma, \Phi\rangle$	*basic specification*
	$\mid$	$SP \cup SP'$	*combination of specifications*
	$\mid$	translate SP by σ	*renaming and/or adding symbols*
	$\mid$	derive from SP by σ	*hiding and/or renaming symbols*

where Σ ranges over signatures, Φ ranges over sets of sentences, and σ ranges over signature morphisms, all from $\mathbf{I}$.

As usual, $SP \cup SP'$ combines the requirements that are expressed separately in SP and SP', and then in typical institutions like the ones defined above, translate SP by σ renames sorts and operation symbols in SP and/or adds new sorts and operation symbols without constraining them at all, and derive from SP by σ hides and/or renames sorts and operation symbols in SP.

Definition 8.2 *The* signature $Sig(SP)$ of a specification SP *is determined as follows:*

$Sig(\langle\Sigma, \Phi\rangle) = \Sigma$	*if* $\Phi \subseteq \mathbf{Sen}(\Sigma)$
$Sig(SP \cup SP') = Sig(SP)$	*if* $Sig(SP) = Sig(SP')$
$Sig($translate SP by $\sigma) = \Sigma'$	*if* $\sigma : Sig(SP) \to \Sigma'$
$Sig($derive from SP' by $\sigma) = \Sigma$	*if* $\sigma : \Sigma \to Sig(SP')$

A specification is well-formed *if it has a signature; otherwise it is* ill-formed. *We will implicitly require all specifications below to be well-formed.*

The semantics of specifications is defined by associating a class of $Sig(SP)$-models to every well-formed specification SP, as follows.

Definition 8.3 *The* model class of a specification SP *in an institution* $\mathbf{I}$ *is the class of $Sig(SP)$-models $Mod(SP) \subseteq |\mathbf{Mod}(Sig(SP))|$ determined as follows:*

$$\begin{aligned}
&Mod(\langle\Sigma, \Phi\rangle) = \{M \in |\mathbf{Mod}(\Sigma)| \mid M \models_\Sigma \Phi\}\\
&Mod(SP \cup SP') = Mod(SP) \cap Mod(SP')\\
&Mod(\text{translate } SP \text{ by } \sigma) = \{M' \mid M'|_\sigma \in Mod(SP)\}\\
&Mod(\text{derive from } SP' \text{ by } \sigma) = \{M'|_\sigma \mid M' \in Mod(SP')\}
\end{aligned}$$

This semantics is compositional: the class of models of a specification is determined from the class of models of its immediate constituents.

The above definitions of specifications in an arbitrary institution can now be instantiated to the framework of each of the institutions of interest here, as defined in Sect. 6. These institutions share a common "syntax" and so they share the class of specifications. The model classes of specifications differ though. For each specification SP in $\mathbf{I_P}$, we will write $Mod_{\mathbf{P}}(SP)$ for the class of models of SP in $\mathbf{I_P}$ and $Mod_{\mathbf{M}}(SP)$ for its class of models in $\mathbf{I_M}$. Moreover, if SP is a specification in $\mathbf{I_{A0}}$ then we will write $Mod_{\mathbf{A0}}(SP)$ for its class of models in $\mathbf{I_{A0}}$ and $Mod_{\mathcal{A}\mathbf{0}}(SP)$ for its class of models in $\mathbf{I}_{\mathcal{A}\mathbf{0}}$.

The model classes $Mod_{\mathbf{A0}}(SP)$ and $Mod_{\mathbf{P}}(SP)$, and similarly $Mod_{\mathcal{A}\mathbf{0}}(SP)$ and $Mod_{\mathbf{M}}(SP)$, are related in the obvious way:

Proposition 8.4 *For any specification SP in* $\mathbf{I_{A0}}$,

1. $Mod_{\mathbf{A0}}(SP) = Mod_{\mathbf{P}}(SP) \cap |\mathbf{PAlg}_{A0}(Sig(SP))|$, *and*
2. $Mod_{\mathcal{A}\mathbf{0}}(SP) = Mod_{\mathbf{M}}(SP) \cap |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP))|$. □

The distinction between $Mod_{\mathbf{P}}(SP)$ and $Mod_{\mathbf{M}}(SP)$ is more interesting, and will be studied in detail in Sect. 9. Here, let us only recall that for any signature $\Sigma \in |\mathbf{AlgSig}|$, we have introduced a functor $|\cdot|_\Sigma : \mathbf{MAlg}(\Sigma) \to \mathbf{PAlg}(\Sigma)$ (in Def. 2.6 written without the subscript Σ). This extends to classes of models in the obvious way, which allows us to compare $Mod_{\mathbf{P}}(SP)$ and $Mod_{\mathbf{M}}(SP)$.

Proposition 8.5 *For any specification SP in* $\mathbf{I_P}$, $|Mod_{\mathbf{M}}(SP)| \subseteq Mod_{\mathbf{P}}(SP)$.

PROOF: *By easy induction on the structure of SP. The key to the proof is that the family of functors* $|\cdot|_\Sigma : \mathbf{MAlg}(\Sigma) \to \mathbf{PAlg}(\Sigma)$, $\Sigma \in |\mathbf{AlgSig}|$, *is "smooth" w.r.t. change of signature, so that we have a natural transformation* $|\cdot| : \mathbf{MAlg} \to \mathbf{PAlg}$. *Moreover, for any signature* Σ, Σ*-sentence* φ *and modest* Σ*-algebra* $\mathcal{A}$, $\mathcal{A} \models_\Sigma \varphi$ *(in the institution* $\mathbf{I_M}$*) iff* $|\mathcal{A}|_\Sigma \models_\Sigma \varphi$ *(in the institution* $\mathbf{I_P}$*).* □

In fact, the remarks in the proof show that we have an "institution representation" $\rho : \mathbf{I_P} \to \mathbf{I_M}$ [Tar96], or "plain map of institutions" [Mes89] (which here is trivial on signatures and sentences, but non-trivial on models) and a similar fact may be proved for structured specifications translated by an arbitrary institution representation.

Corollary 8.6 *For any specification SP in* $\mathbf{I_{A0}}$, $|Mod_{\mathcal{A}\mathbf{0}}(SP)| \subseteq Mod_{\mathbf{A0}}(SP)$. □

9 Mind the gap!

The institution $\mathbf{I_P}$ of partial algebras provides a basic abstract framework for program specification and development, and has for this reason been extensively studied. But if we consider the fact that actual programs are written in real programming languages, and accordingly take issues of computability into account, then we are in fact working in the more restricted framework of the institution of modest (or at least modestizable) algebras. This raises the question of whether there is an appropriate correspondence between the world of programs and the abstract world of algebras studied in the literature. If there is a mismatch, then in the worst case there is the danger of a program being certified as "verified" even though it contains errors. We will try to answer these questions in this section by comparing the meanings of specifications in the institutions of partial and modest algebras.

First, for any specification SP we have $|Mod_{\mathbf{M}}(SP)| \subseteq Mod_{\mathbf{P}}(SP)$ (Prop. 8.5) and $|Mod_{\mathcal{A}\mathbf{0}}(SP)| \subseteq Mod_{\mathbf{A0}}(SP)$ (Cor. 8.6), and so any program represented as a modest algebra that correctly realizes a specification SP in the framework of the institution $\mathbf{I_M}$ (or $\mathbf{I}_{\mathcal{A}\mathbf{0}}$), correctly realizes SP in the more abstract framework of the institution $\mathbf{I_P}$ (or $\mathbf{I_{A0}}$) as well. This shows that it is *sound* to develop programs in the framework of modest algebras, and so that there is no *dangerous gap* between the two frameworks.

Definition 9.1 *A specification SP* witnesses a gap *between $\mathbf{I_P}$ and $\mathbf{I_M}$ if $Mod_{\mathbf{P}}(SP) \neq |Mod_{\mathbf{M}}(SP)|$. The gap is* worrying *if for some modestizable algebra A, $A \in Mod_{\mathbf{P}}(SP)$ but $A \notin |Mod_{\mathbf{M}}(SP)|$. SP witnesses a* consistency *gap if $Mod_{\mathbf{P}}(SP) \neq \emptyset$ while $Mod_{\mathbf{M}}(SP) = \emptyset$. Similar definitions apply to gaps between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$.*

Clearly, gaps between $\mathbf{I_P}$ and $\mathbf{I_M}$ exist and are witnessed by the trivial specification $\langle\Sigma, \emptyset\rangle$ for any signature $\Sigma \in |\mathbf{AlgSig}|$ that is sufficiently rich to ensure that the functor $|\cdot|_\Sigma : \mathbf{MAlg}(\Sigma) \to \mathbf{PAlg}(\Sigma)$ is not surjective (cf. Counterexample 7.4). However, such a gap is in itself not a cause for alarm, since it might be that in the process of constructing modest algebras it would remain invisible. This is in contrast with worrying gaps: if for some specification SP there is a modest algebra $\mathcal{A} \notin Mod_{\mathbf{M}}(SP)$ such that $|\mathcal{A}| \in Mod_{\mathbf{P}}(SP)$ then the gap exhibited by SP should worry us, as there is a danger that by interpreting the specification SP in the institution of modest algebras we exclude some perfectly acceptable realizations of SP.

Unfortunately, we have consistency gaps between $\mathbf{I_P}$ and $\mathbf{I_M}$ (Counterexample 7.8; translating this to a signature including $\Sigma0$ we can also exhibit a consistency gap between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$). The specification formalism described in Sect. 8 is rich enough to exploit consistency gaps in a worrying way:

Proposition 9.2 *If there is a consistency gap between $\mathbf{I_P}$ and $\mathbf{I_M}$ (resp. between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$) then there is also a worrying consistency gap between $\mathbf{I_P}$ and $\mathbf{I_M}$ (resp. between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$).*

PROOF: *Let SP witness a consistency gap between $\mathbf{I_P}$ and $\mathbf{I_M}$, and let $\iota : \Sigma_\emptyset \hookrightarrow Sig(SP)$ be the inclusion of the empty signature $\Sigma_\emptyset$ into $Sig(SP)$. Then the specification* derive from SP by ι *witnesses a worrying consistency gap between $\mathbf{I_P}$ and $\mathbf{I_M}$: $Mod_{\mathbf{M}}(\text{derive from } SP \text{ by } \iota) = \emptyset$ while $Mod_{\mathbf{P}}(\text{derive from } SP \text{ by } \iota) = |\mathbf{PAlg}(\Sigma_\emptyset)|$, and the latter contains the trivial empty algebra, which is modestizable.*

The construction of a worrying consistency gap out of a consistency gap between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$ is similar: just let $\iota : \Sigma0 \hookrightarrow Sig(SP)$ be the inclusion of $\Sigma0$ into the signature of the specification which witnesses the consistency gap. □

Corollary 9.3 *There are worrying consistency gaps between $\mathbf{I_P}$ and $\mathbf{I_M}$, as well as between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$.* □

The reader might feel that the use of the empty algebra over the empty signature in the proof of Prop. 9.2 is a little dubious, but the same pattern applies in the case of any other signature. More significantly, Counterexample 7.8 is not very convincing:

we doubt that a specification like this would ever be written in practice (especially since it is essentially infinite). Unfortunately, we do not know at present if there is a *finite* first-order specification witnessing a worrying gap between $\mathbf{I_P}$ and $\mathbf{I_M}$.

However, we can exhibit quite natural specifications that witness a worrying consistency gap between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$. For instance, looking at Counterexample 7.5, since the halting function is arithmetical and so is first-order axiomatizable over the standard model of the natural numbers, there is a sentence φ_{halt} (over the signature Σ_{halt} which extends $\Sigma0$ by the operation symbol $halt : nat \rightarrow bool$) such that A_{halt} is (up to isomorphism) the only partial $A0$-extension satisfying φ_{halt}. Thus, repeating the argument from Counterexample 7.5, $Mod_{\mathbf{A0}}(\langle\Sigma_{halt}, \{\varphi_{halt}\}\rangle) \neq \emptyset$ while $Mod_{\mathcal{A}\mathbf{0}}(\langle\Sigma_{halt}, \{\varphi_{halt}\}\rangle) = \emptyset$ and then by Prop. 9.2 a worrying consistency gap can be obtained. Similarly but perhaps more convincingly:

Example 9.4 *Let Σ_{equiv} extend $\Sigma0$ by $equiv : nat \times nat \rightarrow bool$, and let φ_{equiv} be a sentence such that for any $A \in Mod_{\mathbf{A0}}(\langle\Sigma_{equiv}, \{\varphi_{equiv}\}\rangle)$, $equiv_A(n, m) = true$ iff the nth and mth Turing machines are equivalent (such a sentence exists, since the equivalence of TMs is arithmetical).*

This can be used as the basis for a specification of functions that perform transformations on Turing machines. For example, let Σ_{opt} extend $\Sigma0$ by $opt : nat \rightarrow nat$, and let $\iota_1 : \Sigma_{equiv} \hookrightarrow (\Sigma_{equiv} \cup \Sigma_{opt})$ and $\iota_2 : \Sigma_{opt} \hookrightarrow (\Sigma_{equiv} \cup \Sigma_{opt})$ be signature inclusions. Then a specification SP_{opt} defined as:

```
derive from
    translate $\langle\Sigma_{equiv}, \{\varphi_{equiv}\}\rangle$ by $\iota_1$      $\cup$
    $\langle\Sigma_{equiv} \cup \Sigma_{opt}, \{\forall n{:}nat.equiv(opt(n), n) = true\}\rangle$
by $\iota_2$
```

specifies an optimizing function opt transforming TMs. (Axioms could be added to require that the output of opt is at least as efficient as its input.) This specification witnesses a worrying consistency gap between $\mathbf{I_{A0}}$ and $\mathbf{I}_{\mathcal{A}\mathbf{0}}$: $Mod_{\mathcal{A}\mathbf{0}}(SP_{opt}) = \emptyset$ since $Mod_{\mathcal{A}\mathbf{0}}(\langle\Sigma_{equiv}, \{\varphi_{equiv}\}\rangle) = \emptyset$, while $Mod_{\mathbf{A0}}(SP_{opt})$ is not empty and contains many modestizable algebras. □

In contrast to some of our other examples, the pattern in Example 9.4 actually arises in practice. The notation used in SP_{opt} hides the idea; in a higher-level specification language it might look like this:

```
local   val $equiv : nat \times nat \rightarrow bool$
        axiom $\varphi_{equiv}$
in      val $opt : nat \rightarrow nat$
        axiom $\forall n{:}nat.equiv(opt(n), n) = true$
end
```

The example is expressed in terms of Gödel encodings of Turing machines where its practical utility may not be immediately obvious, but exactly the same example could be phrased in terms of program fragments in a real programming language and the above specification could then appear as part of the specification of an optimizing compiler or program transformation system.

Examples of realistic specifications where a similar worrying consistency gap arises lead us to the conclusion that we do in fact want to interpret specifications at the abstract level of partial algebras, i.e. in the institution $\mathbf{I_P}$. However, the construction of programs satisfying requirements specifications happens at the more concrete level of programs, modelled here by modest algebras. In effect, the development process (see Sect. 10) will use yet another semantics of specifications:

Definition 9.5 *For any specification SP in* $\mathbf{I_P}$*, define*

$$Mod_{\mathbf{P}\to\mathbf{M}}(SP) = \{\mathcal{A} \in |\mathbf{MAlg}(Sig(SP))| \mid |\mathcal{A}| \in Mod_{\mathbf{P}}(SP)\}.$$

Similarly, for any specification SP in $\mathbf{I_{A0}}$*, define*

$$Mod_{\mathbf{A0}\to\mathcal{A}0}(SP) = \{\mathcal{A} \in |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP))| \mid |\mathcal{A}| \in Mod_{\mathbf{A0}}(SP)\}.$$

Neither of these two semantics is compositional, as is shown by the examples we have just been discussing. However, the validation of specifications (proving their logical consequences) and the verification of correctness of refinement steps may be carried out at the abstract level, where the semantics of specifications is compositional:

Proposition 9.6 *For any specification SP in the institution* $\mathbf{I_P}$ *and* $Sig(SP)$*-sentence* φ*,* $Mod_{\mathbf{P}\to\mathbf{M}}(SP) \models \varphi$ *whenever* $Mod_{\mathbf{P}}(SP) \models \varphi$*.*

For any two specifications SP and SP' *in* $\mathbf{I_P}$*,* $Mod_{\mathbf{P}\to\mathbf{M}}(SP') \subseteq Mod_{\mathbf{P}\to\mathbf{M}}(SP)$ *whenever* $Mod_{\mathbf{P}}(SP') \subseteq Mod_{\mathbf{P}}(SP)$*.* □

Similar facts hold for specifications in $\mathbf{I_{A0}}$ and their model classes given by $Mod_{\mathbf{A0}\to\mathcal{A}0}$ and $Mod_{\mathbf{A0}}$.

10 Formal software development

In [ST88b] (cf. [ST95]) we have proposed to view the process of software development as the production of a sequence of *constructor implementation* steps, starting from a specification of requirements and gradually adding design and implementation decisions expressed as constructions on algebras, and leading to a stage where nothing is left to be implemented. Then the sequence of constructions used in the development determines a program that correctly implements the original specification.

In this section we will recast these ideas in the framework of this paper, paying special attention to the effects of considering both abstract and concrete models of programs (partial and modest algebras, respectively). We will explicitly work with modest $\mathcal{A}0$-extensions only; but of course everything can be repeated for arbitrary modest algebras if needed.

In the framework discussed here, constructions involved in implementation steps work on modest $\mathcal{A}0$-extensions and so can be modelled as functions on modest $\mathcal{A}0$-extensions. But in view of the gaps exhibited in Sect. 9, correctness of constructor implementation steps should involve the more permissive semantics of specifications from the institution of partial algebras to ensure that some possible constructions are not excluded:

Definition 10.1 *Given specifications SP and* SP' *in* $\mathbf{I_{A0}}$ *and a construction* $\kappa : |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP'))| \to |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP))|$ *on modest* $\mathcal{A}0$*-extensions, we say that* SP' implements SP via κ*, written* $SP \underset{\kappa}{\leadsto} SP'$*, if for all modest* $\mathcal{A}0$*-extensions* $\mathcal{A} \in |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP'))|$*,* $|\kappa(\mathcal{A})| \in Mod_{\mathbf{A0}}(SP)$ *whenever* $|\mathcal{A}| \in Mod_{\mathbf{A0}}(SP')$*, that is:* $\kappa(Mod_{\mathbf{A0}\to\mathcal{A}0}(SP')) \subseteq Mod_{\mathbf{A0}\to\mathcal{A}0}(SP)$*.*

Developments viewed as sequences of such steps ensure that correctness of the final program may be inferred from the correctness of all the individual steps:

Theorem 10.2 *Given a sequence* $SP_0 \underset{\kappa_1}{\rightsquigarrow\!\!>} SP_1 \underset{\kappa_2}{\rightsquigarrow\!\!>} \cdots \underset{\kappa_n}{\rightsquigarrow\!\!>} SP_n = \langle \Sigma 0, \emptyset \rangle$, *we have* $|\kappa_1(\kappa_2(\ldots \kappa_n(\mathcal{A}0)\ldots))| \in Mod_{\mathbf{A0}}(SP_0)$. □

We have defined various constructions on modest $\mathcal{A}0$-extensions, including definitional extension, reduct along a signature morphism (available in any institution) and amalgamation. The latter typically arises when the task of implementing a specification is decomposed into a number of subtasks, each to implement some simpler specification, and the results must then be combined to build an implementation of the original specification. Of course, this is the essence of a modular approach to software development, and consequently institutions which do not admit amalgamation can only be of limited use as frameworks for such a development methodology. In fact, this is an important technical argument against the use of the institution $\mathbf{I}_{\exists\mathbf{M}}$ of modestizable algebras. However, in the framework we adopted above — the institution $\mathbf{I}_{\mathcal{A}\mathbf{0}}$ of modest $\mathcal{A}0$-extensions with the semantics of specifications based on their interpretation in the institution $\mathbf{I}_{\mathbf{A0}}$ of partial $A0$-extensions — all these constructions are well defined and everything works fine!

A further refinement of the development methodology, enhancing its practical flexibility in an essential way, involves taking a *behavioural* interpretation of specifications [ST88b]. This is based on the notion of behavioural equivalence between partial algebras, which can easily be generalized to modest algebras.

Definition 10.3 *Let OBS be a set of* observable sorts *in a signature* Σ.

Partial Σ*-algebras* A *and* B *are* behaviourally equivalent *(*with respect to *OBS), written* $A \equiv_{OBS} B$, *if there is an OBS-sorted set* X *of variables and valuations* v_A *in* A *and* v_B *in* B *that are surjective on sorts in OBS such that: for any term* $t \in T_\Sigma(X)$, $A \models_{v_A} D(t)$ *iff* $B \models_{v_B} D(t)$*; and for any terms* $t, u \in T_\Sigma(X)$ *of the same sort in OBS,* $A \models_{v_A} t = u$ *iff* $B \models_{v_B} t = u$.

Modest Σ*-algebras* $\mathcal{A}$ *and* $\mathcal{B}$ *are* (extensionally) behaviourally equivalent *(*with respect to *OBS), written* $\mathcal{A} \equiv_{OBS} \mathcal{B}$, *if there is an OBS-sorted set* X *of variables and valuations* $v_{\mathcal{A}}$ *in* $\mathcal{A}$ *and* $v_{\mathcal{B}}$ *in* $\mathcal{B}$ *that are surjective on sorts in OBS such that: for any term* $t \in T_\Sigma(X)$, $\mathcal{A} \models_{v_{\mathcal{A}}} D(t)$ *iff* $\mathcal{B} \models_{v_{\mathcal{B}}} D(t)$*; and for any terms* $t, u \in T_\Sigma(X)$ *of the same sort in OBS,* $\mathcal{A} \models_{v_{\mathcal{A}}} t = u$ *iff* $\mathcal{B} \models_{v_{\mathcal{B}}} t = u$.

We will omit the set of observable sorts if it is unimportant or obvious. For instance, when we work with $\mathcal{A}0$-extensions it is natural to choose the sorts $S0$ as observable.

Proposition 10.4 *For any modest* Σ*-algebras* $\mathcal{A}$ *and* $\mathcal{B}$, $\mathcal{A} \equiv \mathcal{B}$ *iff* $|\mathcal{A}| \equiv |\mathcal{B}|$. □

The behavioural interpretation of a specification is simply the closure of its usual class of models under behavioural equivalence:

Definition 10.5 *The* behavioural interpretation *of any specification SP in* $\mathbf{I}_{\mathbf{A0}}$ *is* $Beh_{\mathbf{A0}\to\mathcal{A}\mathbf{0}}(SP) = \{\mathcal{A} \in |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP))| \mid \mathcal{A} \equiv \mathcal{B}$ *for some* $\mathcal{B} \in Mod_{\mathbf{A0}\to\mathcal{A}\mathbf{0}}(SP)\}$.

Prop. 10.4 might suggest that it is not important whether the behavioural closure of specifications is taken at partial algebra or at modest algebra level. But this is not the case: given a modest algebra $\mathcal{A}$, a partial algebra that is behaviourally equivalent to $|\mathcal{A}|$ need not be modestizable. A consequence of this is that behavioural correctness of implementations must be based on behavioural equivalence of modest algebras.

Definition 10.6 *Given two specifications SP and SP' in $\mathbf{I_{A0}}$ and a construction $\kappa : |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP'))| \to |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP))|$ on modest $\mathcal{A}0$-extensions, we say that SP'* behaviourally implements *SP via κ, written $SP \underset{\kappa}{\overset{\equiv}{\rightsquigarrow}} SP'$, if for all modest $\mathcal{A}0$-extensions $\mathcal{A} \in |\mathbf{MAlg}_{\mathcal{A}0}(Sig(SP'))|$, $\kappa(\mathcal{A}) \in Beh_{\mathbf{A0}\to\mathcal{A}0}(SP)$ whenever $|\mathcal{A}| \in Mod_{\mathbf{A0}}(SP')$, that is $\kappa(Mod_{\mathbf{A0}\to\mathcal{A}0}(SP')) \subseteq Beh_{\mathbf{A0}\to\mathcal{A}0}(SP)$.*

Developments viewed as sequences of such steps should again ensure that correctness of the final program may be inferred from the correctness of all the individual steps. But this holds only if the constructions available are stable [Sch87], [ST89]:

Definition 10.7 *A construction $\kappa : |\mathbf{MAlg}(\Sigma')| \to |\mathbf{MAlg}(\Sigma)|$ on modest algebras is* stable *if $\kappa(\mathcal{A}) \equiv \kappa(\mathcal{B})$ for all modest Σ'-algebras $\mathcal{A}, \mathcal{B}$ such that $\mathcal{A} \equiv \mathcal{B}$.*

Theorem 10.8 *If $SP_0 \underset{\kappa_1}{\overset{\equiv}{\rightsquigarrow}} SP_1 \underset{\kappa_2}{\overset{\equiv}{\rightsquigarrow}} \cdots \underset{\kappa_n}{\overset{\equiv}{\rightsquigarrow}} SP_n = \langle \Sigma 0, \emptyset \rangle$ and all the constructions used are stable, then $\kappa_1(\kappa_2(\ldots \kappa_n(\mathcal{A}0)\ldots)) \in Beh_{\mathbf{A0}\to\mathcal{A}0}(SP_0)$.* □

In the above we have been rather sloppy in omitting the subscript *OBS* everywhere. The subtleties of stability and local and global correctness of constructions, as introduced in [Sch87] and discussed in [ST89], apply here without change.

11 Future work

We have studied the apparent conflict between the use of abstract models in theories of specification and formal program development (here, partial algebras) and the use of more or less concrete models of programs in semantics of real programming languages (here, approximated by modest algebras and modest $\mathcal{A}0$-extensions). We conclude that all is well, but note the need for certain adjustments to the theory of specification and formal development to take account of the difference between the world of programs and the world of abstract models.

The picture is not yet complete. For instance, we have shown that the constructions needed for formal program development using constructor implementations (definitional extension, reduct, amalgamation) extend from partial algebras to modest algebras. But we have not yet explored the consequences of the fact that some other familiar constructions on partial algebras (e.g. existence of reachable subalgebras) do not carry over. Further constructions should be studied, such as closure under arrow types. The difficulty here is that (as explained in Sect. 2) we would like the underlying values of arrow type to be extensional functions, restricted to those that can be expressed using well-typed λ-terms.

The framework of specification and formal development on which the presentation in Sects. 8 and 10 is based was generalized to specification and development of (higher-order) parameterized algebras in [SST92]. A similar generalization should go through here, but this would have to be based on the $Mod_{\mathbf{P}\to\mathbf{M}}$ (or $Mod_{\mathbf{A0}\to\mathcal{A}0}$) semantics of specifications. However, the computability restrictions in modest algebras do not seem to extend easily to computability restrictions on parameterised modest algebras. For the case of first-order parameterised algebras, this corresponds to the (unposed but interesting) question of the computability of constructions like those in Sect. 5.

In the case of EML, the picture we have been painting is complicated by the fact that the semantics of EML [KST94] uses models that are based on, but more expressive than, those used in the semantics of SML. The main difference between the two is that the closures used to represent functions in EML may contain "logical" constructs such as universal and existential quantifiers. Thus we need to consider three levels: that of SML programs and their models; that of EML specifications and their models; and that of partial algebras. It seems that EML models are captured by relaxing the requirement in modest algebras that the tracking functions be partial recursive, and require them to be merely arithmetical. Call these *immodest algebras*; then EML models are approximated by immodest $\mathcal{A}0$-extensions. Since closures in EML models may be formed using a recursion operator as well as quantifiers, it is not clear if this is expressive enough. But it is possible to show, using Gödel's Fixpoint Theorem, that if the satisfaction of closed EML formulae can be defined in EML, then EML is inconsistent.[4] Thus in any case there is a gap between the level of EML models and partial algebras. This means that we have two separate gaps. It seems likely that our analysis applies to both, but there may be interesting complications in dealing with both at once.

Finally, this paper treats just one kind of abstract model, namely partial algebras. The same study could be repeated for other kinds of abstract models. Although we expect that the results would be much the same, there may be interesting variations in the details in some cases.

Acknowledgements: Thanks to Stefan Kahrs, Martin Hofmann and Thomas Streicher for useful discussions, and to Martin Wirsing and Jan Bergstra for providing pointers to the literature. Thanks especially to Stefan for service as an oracle concerning the semantics of SML and EML and recursive function theory.

References

[BT87] J. Bergstra and J. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science* 50:137–181 (1987).

[BKLOS91] M. Bidoit, H.-J. Kreowski, P. Lescanne, F. Orejas and D. Sannella (eds.) *Algebraic System Specification and Development: A Survey and Annotated Bibliography.* Springer LNCS 501 (1991).

[DGS93] R. Diaconescu, J.A. Goguen and P. Stefaneas. Logical support for modularisation. In: *Logical frameworks* (G. Huet and G. Plotkin, eds.), 83–130. Cambridge Univ. Press (1993).

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* Springer (1985).

[GB92] J. Goguen and R. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM* 39:95–146 (1992).

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

[KST94] S. Kahrs, D. Sannella and A. Tarlecki. The definition of Extended ML. Report ECS-LFCS-94-300, Univ. of Edinburgh (1994).

[KST95] S. Kahrs, D. Sannella and A. Tarlecki. The definition of Extended ML: a gentle introduction. Report ECS-LFCS-95-322, Univ. of Edinburgh (1995). *Theoretical Computer Science*, to appear (1996).

[4]Thanks to Martin Hofmann for this observation, and to Stefan Kahrs for attempting (and, luckily, failing!) to specify this. His conclusion is that in EML such a predicate cannot be total (predicates in EML are functions into `bool`, and functions need not be total).

[LEW96] J. Loeckx, H.-D. Ehrich and M. Wolf. *Specifications of Abstract Data Types.* Wiley (1996).

[Mal61] A.I. Mal'cev. Constructive algebras I. *Russian Mathematical Surveys* 16:77–129 (1961). Also in: *The Metamathematics of Algebraic Systems. Collected papers 1936–1967* (B. Wells ed.), 148–212. North-Holland (1971).

[Mes89] J. Meseguer. General logic. *Logic Colloquium'87* (H.-D. Ebbinghaus et al., eds.), 279–329. North-Holland (1989).

[MTH90] R. Milner, M. Tofte and R. Harper. *The Definition of Standard ML.* MIT Press (1990).

[Pau91] L. Paulson. *ML for the Working Programmer.* Cambridge Univ. Press (1991).

[Rab60] M. Rabin. Computable algebra, general theory and theory of computable fields. *Trans. of the AMS* 95:341–360 (1960).

[Rog67] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill (1967).

[Ros90] G. Rosolini. About modest sets. *Intl. Journal of Foundations of Computer Science* 1:341–353 (1990).

[San91] D. Sannella. Formal program development in Extended ML for the working programmer. *Proc. 3rd BCS/FACS Workshop on Refinement*, Hursley Park. Springer Workshops in Computing, 99–130 (1991).

[SST92] D. Sannella, S. Sokołowski and A. Tarlecki. Toward formal development of programs from algebraic specifications: parameterisation revisited. *Acta Informatica* 29:689–736 (1992).

[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. *Proc. 3rd 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 formal program development: basic concepts and motivation. ICS PAS Report 791, Institute of Computer Science PAS, Warsaw (1995).

[ST9?] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development.* Cambridge Univ. Press, to appear (199?).

[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).

[Sch87] O. Schoett. Data Abstraction and the Correctness of Modular Programming. Ph.D. thesis; Report CST-42-87, Univ. of Edinburgh (1987).

[SHT95] V. Stoltenberg-Hansen and J. Tucker. Effective algebras. In: *Handbook of Logic in Computer Science, Vol. 4* (S. Abramsky, D. Gabbay and T. Maibaum, eds.), 357–526. Oxford Univ. Press (1995).

[Tar96] A. Tarlecki. Moving between logical systems. *Recent Trends in Data Type Specifications. 11th Workshop on Specification of Abstract Data Types* (M. Haveraaen et al., eds.). Springer LNCS, to appear (1996).

[Wir90] M. Wirsing. Algebraic specification. In: *Handbook of Theoretical Computer Science, Vol. B* (J. van Leeuwen, ed.), 675–788. North-Holland (1990).