The foundational legacy of ASL

Donald Sannella[1] and Andrzej Tarlecki[2]

[1] Laboratory for Foundations of Computer Science, University of Edinburgh
[2] Institute of Informatics, University of Warsaw

Abstract. We recall the kernel algebraic specification language ASL and outline its main features in the context of the state of research on algebraic specification at the time it was conceived in the early 1980s. We discuss the most significant new ideas in ASL and the influence they had on subsequent developments in the field and on our own work in particular.

1 Introduction

One of Martin Wirsing's most important contributions to the field of algebraic specification was his work on the ASL specification language. ASL is one of the milestones of the field and is one of Martin's most influential lines of work. It was also the highlight of our long-term collaboration and friendship with him — **many thanks, Martin!!!**

ASL is a simple algebraic specification language containing a small set of orthogonal constructs. Preliminary ideas were sketched in [Wir82], then modified and further developed in [SW83], with [Wir86] offering a complete, extended presentation. At the time, the first fully-fledged algebraic specification languages had recently been defined (Clear [BG80], CIP-L [BBB$^+$85] etc.). In contrast, ASL was conceived as a *kernel language*, with stress on expressive power, conceptual clarity, and simplicity, rather than on convenience of use. The idea was to penetrate to the essential concepts, suitable for foundational studies and supplying a basis that could be used to define other specification languages.

Among the key characteristics of ASL, as listed in [Wir86], are the following:

- "ASL is a language for describing *classes of algebras* rather than for building sets of axioms (theories)"; in particular, "an ASL specification may be *loose* (meaning that it may possess nonisomorphic models)": We will discuss the semantics of specifications in Sect. 3, including the relationship between model-class semantics and theory-level semantics.
- "The expressive power of ASL allows the choice of a *simple notion of implementation*" and "parameterization in ASL is λ-*abstraction*": We will discuss aspects of the software development process, as influenced by these two ideas, in Sect. 4.
- "ASL is oriented towards a *'behavioural' approach* ... ASL includes a very general *observability* operation which can be used to behaviourally abstract from a specification": We will discuss the technicalities of behavioural abstraction and its role in the software development process in Sect. 5.

This work has been partially supported by the National Science Centre, grant UMO-2013/11/B/ST6/01381 (AT).

Further characteristics, also listed in [Wir86], are:

- "*Infinite signatures* and *infinite sets of axioms* can be described by finite ASL expressions" and "Algebras in ASL are *generalized algebras* ... suitable for the description of strict and nonstrict operations": We are not going to dwell on these points as they are subsumed by the more general setup of [ST88a], where the semantics of ASL is given for an arbitrary logical system (institution); we will follow this approach in Sects. 2–4. The particular choices made in [Wir82], [SW83] and [Wir86] arise from particular institutions.
- "ASL can be seen as an applicative (programming) language where the basic modes are not only natural numbers, integers, or strings, but sorts, operation symbols, terms, formulas, signatures, and specifications": The novel feature here is that each of these modes was treated as a first-class citizen in ASL. As far as we know, this aspect of ASL has not been explicitly taken up in later work, at least not to the same extent.
- "ASL is a *universal specification language* allowing to write every computable transformation of specifications": The power of parameterization in ASL comes partly from the previous point. This universality property was an interesting technical point but in our view it evades the real question, concerning which *semantic* entities (model classes and transformations between them) can be captured.

As indicated above, this paper discusses some of the ideas and themes that were prominent in ASL and influenced further work. We comment on these from today's perspective, supported by pointers to the subsequent literature.

2 Preliminaries

We will rely here on the usual notions of many-sorted algebraic signatures $\Sigma = \langle S, \Omega \rangle$ and signature morphisms $\sigma \colon \Sigma \to \Sigma'$ mapping sorts in Σ to sorts in Σ' and operation names in Σ to operation names with corresponding arity and result sorts in Σ'. This yields the category $\mathbf{AlgSig}$. For each algebraic signature Σ, $\mathbf{Alg}(\Sigma)$ stands for the usual category of Σ-algebras and their homomorphisms. We restrict attention to algebras with non-empty carriers to avoid minor technical problems in the sequel, which are by now well-understood, see [Tar11]. As usual, each signature morphism $\sigma \colon \Sigma \to \Sigma'$ determines a *reduct* functor $\_|_\sigma \colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$. This yields a functor $\mathbf{Alg} \colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$. See [ST12] for a more detailed presentation.

Given a signature Σ, Σ-terms, Σ-equations, and first-order Σ-formulae with equality are defined as usual. The set of all Σ-terms with variables from X is denoted by $T_\Sigma(X)$, and for sets IN, OUT of sorts in Σ, $T_\Sigma(X_{IN})_{OUT}$ denotes the set of Σ-terms of sorts in OUT with variables of sorts in IN. Given a Σ-algebra A, a set of variables X and a valuation $v \colon X \to |A|$, the *value* $t_{A[v]}$ of a Σ-term t with variables X in A under v and the *satisfaction* $A[v] \models \varphi$ of a formula φ with variables X in A under v are defined as usual. The parameter v is omitted when X is empty.

A *derived signature morphism* $\delta \colon \Sigma \to \Sigma'$ maps sorts in Σ to sorts in Σ' and function symbols $f \colon s_1 \times \ldots \times s_n \to s$ in Σ to Σ'-terms of sort $\delta(s)$ with variables $\{x_1{:}\delta(s_1), \ldots, x_n{:}\delta(s_n)\}$. This generalises "ordinary" algebraic signature morphisms as recalled above. A derived signature morphism $\delta\colon \Sigma \to \Sigma'$ still determines a reduct functor $\_|_\delta\colon \mathbf{Alg}(\Sigma') \to \mathbf{Alg}(\Sigma)$.

Given a (derived) signature morphism $\delta\colon \Sigma \to \Sigma'$, the δ-translation of Σ-terms, Σ-equations, and first-order Σ-formulae to Σ'-terms, Σ'-equations, and first-order Σ'-formulae are as usual: we write $\delta(t)$ etc. For any term $t \in T_\Sigma(X)$, Σ'-algebra A', and valuation $v'\colon \delta(X) \to |A'|$, where $\delta(X)_{s'} = \biguplus_{\delta(s)=s'} X_s$, we have $t_{(A'|_\delta)[v]} = \delta(t)_{A'[v']}$, where $v\colon X \to |A'|_\delta|$ is the valuation of variables in X that corresponds to v' in the obvious way. Similarly for Σ-equations and Σ-formulae φ with free variables X: $(A'|_\delta)[v] \models \varphi$ iff $A'[v'] \models \delta(\varphi)$.

An *institution* [GB92] $\mathbf{INS}$ consists of:

- a category $\mathbf{Sign_{INS}}$ of *signatures*;
- a functor $\mathbf{Sen_{INS}}\colon \mathbf{Sign_{INS}} \to \mathbf{Set}$, giving a set $\mathbf{Sen_{INS}}(\Sigma)$ of *Σ-sentences* for each signature $\Sigma \in |\mathbf{Sign_{INS}}|$;
- a functor $\mathbf{Mod_{INS}}\colon \mathbf{Sign}^{op}_{\mathbf{INS}} \to \mathbf{Cat}$, giving a category $\mathbf{Mod_{INS}}(\Sigma)$ of *Σ-models* for each signature $\Sigma \in |\mathbf{Sign_{INS}}|$; and
- a family $\langle \models_{\mathbf{INS},\Sigma} \subseteq |\mathbf{Mod_{INS}}(\Sigma)| \times \mathbf{Sen_{INS}}(\Sigma)\rangle_{\Sigma\in|\mathbf{Sign_{INS}}|}$ of *satisfaction relations*

such that for any signature morphism $\sigma\colon \Sigma \to \Sigma'$ the translations $\mathbf{Mod_{INS}}(\sigma)$ of models and $\mathbf{Sen_{INS}}(\sigma)$ of sentences preserve the satisfaction relation, that is, for any $\varphi \in \mathbf{Sen_{INS}}(\Sigma)$ and $M' \in |\mathbf{Mod_{INS}}(\Sigma')|$ the following *satisfaction condition* holds:

$$M' \models_{\mathbf{INS},\Sigma'} \mathbf{Sen_{INS}}(\sigma)(\varphi) \quad \text{iff} \quad \mathbf{Mod_{INS}}(\sigma)(M') \models_{\mathbf{INS},\Sigma} \varphi$$

We will omit the subscripts $\mathbf{INS}$ and Σ whenever they are obvious from the context. For any signature morphism $\sigma\colon \Sigma \to \Sigma'$, the translation $\mathbf{Sen}(\sigma)\colon \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ will be denoted by $\sigma\colon \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$, and the reduct $\mathbf{Mod}(\sigma)\colon \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ by $\_|_\sigma\colon \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$. Thus, the satisfaction condition may be re-stated as: $M' \models \sigma(\varphi)$ iff $M'|_\sigma \models \varphi$. For any signature Σ, the satisfaction relation extends naturally to sets of Σ-sentences and classes of Σ-models.

Examples of institutions abound. The institution $\mathbf{EQ}$ of equational logic has the category $\mathbf{Sign_{EQ}} = \mathbf{AlgSig}$ of many-sorted algebraic signatures as its category of signatures; its models are algebras, so $\mathbf{Mod_{EQ}}$ is $\mathbf{Alg}\colon \mathbf{AlgSig}^{op} \to \mathbf{Cat}$; for any signature $\Sigma \in |\mathbf{AlgSig}|$, $\mathbf{Sen_{EQ}}(\Sigma)$ is the set of all (universally quantified) Σ-equations, with $\mathbf{Sen_{EQ}}(\sigma)\colon \mathbf{Sen_{EQ}}(\Sigma) \to \mathbf{Sen_{EQ}}(\Sigma')$ being the translation of Σ-equations to Σ'-equations for any signature morphism $\sigma\colon \Sigma \to \Sigma'$ in $\mathbf{AlgSig}$. Finally, the satisfaction relations $\models_{\mathbf{EQ},\Sigma} \subseteq |\mathbf{Alg}(\Sigma)| \times \mathbf{Sen_{EQ}}(\Sigma)$ are defined as usual: $A \models_{\mathbf{EQ},\Sigma} \forall X.t = t'$ iff $t_{A[v]} = t'_{A[v]}$ for all valuations $v\colon X \to |A|$. The institution $\mathbf{FOEQ}$ of first-order equational logic shares with $\mathbf{EQ}$ its category of signatures and its model functor, with its sets of sentences extended to include all closed formulae of first-order logic with equality, together with the standard satisfaction relations. Any institution having the same category of signatures and the same model functor as $\mathbf{EQ}$ (and $\mathbf{FOEQ}$) will be called *standard algebraic*. See [ST12] for detailed definitions of many other institutions.

For any signature Σ, a Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$ is a *semantic consequence* of a set of Σ-sentences $\Phi \subseteq \mathbf{Sen}(\Sigma)$, written $\Phi \models_\Sigma \varphi$ or simply $\Phi \models \varphi$, if for all Σ-models

$M \in |\mathbf{Mod}(\Sigma)|$, $M \models \varphi$ whenever $M \models \Phi$. A *Σ-theory* is a set of Σ-sentences that is closed under semantic consequence.

Semantic consequence is often approximated by an *entailment system*, that is, a family of relations $\langle \vdash_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|}$ where, for $\Sigma \in |\mathbf{Sign}|$, $\vdash_\Sigma$ is a relation between sets of Σ-sentences and individual Σ-sentences, subject to the usual requirements (reflexivity, transitivity, weakening). An entailment system (and its presentation via a system of proof rules) is sound for $\mathbf{INS}$ if for $\Sigma \in |\mathbf{Sign}|$, $\Phi \subseteq \mathbf{Sen}(\Sigma)$ and $\varphi \in \mathbf{Sen}(\Sigma)$, $\Phi \vdash_\Sigma \varphi$ implies $\Phi \models_\Sigma \varphi$, and it is complete if the opposite implication holds. Sound and complete proof systems for $\mathbf{EQ}$ and $\mathbf{FOEQ}$ are well known.

Institutional structure is rich enough to enable a number of key features of logical systems to be expressed. For instance, amalgamation and interpolation properties may be captured as follows.

Consider the following commuting diagram in $\mathbf{Sign}$:

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

This diagram *admits amalgamation* if for any two models $M_1 \in |\mathbf{Mod}(\Sigma_1)|$ and $M_2 \in |\mathbf{Mod}(\Sigma_2)|$ such that $M_1|_{\sigma_1} = M_2|_{\sigma_2}$, there exists a unique model $M' \in |\mathbf{Mod}(\Sigma')|$ such that $M'|_{\sigma'_2} = M_1$ and $M'|_{\sigma'_1} = M_2$ and similarly for model morphisms. An institution is *semi-exact* if pushouts of signature morphisms always exist and admit amalgamation. It is well-known that any standard algebraic institution (in particular, $\mathbf{EQ}$ and $\mathbf{FOEQ}$) is semi-exact.

The above diagram *admits parameterized interpolation* if for any $\Phi_1 \subseteq \mathbf{Sen}(\Sigma_1)$, $\Phi_2 \subseteq \mathbf{Sen}(\Sigma_2)$ and $\varphi_2 \in \mathbf{Sen}(\Sigma_2)$, whenever $\sigma'_2(\Phi_1) \cup \sigma'_1(\Phi_2) \models \sigma'_1(\varphi_2)$ then for some $\Phi \subseteq \mathbf{Sen}(\Sigma)$ such that $\Phi_1 \models \sigma_1(\Phi)$ we have $\Phi_2 \cup \sigma_2(\Phi) \models \varphi_2$. The diagram *admits Craig interpolation* if it admits parameterized interpolation with "parameter set" $\Phi_2 = \emptyset$. $\mathbf{INS}$ *admits parameterized* (resp. *Craig*) *interpolation* if all pushouts in the category of signatures admit parameterized (resp. Craig) interpolation.

The above reformulation of classical (first-order) Craig interpolation [CK90] has its source in [Tar86]. It is well-known that single-sorted first-order equational logic admits Craig as well as parameterized interpolation. But in the many-sorted case, interpolation requires additional assumptions on the signature morphisms involved: a pushout in $\mathbf{FOEQ}$ admits Craig and parameterized interpolation when at least one source morphism involved is injective on sorts, see [Bor05]. Interpolation properties for equational logic are even more delicate. $\mathbf{EQ}$ admits Craig interpolation for pushouts in which all morphisms are injective, but the restriction to non-empty carriers cannot be dropped [RG00], [Tar11]. Parameterized interpolation for $\mathbf{EQ}$ fails, unless injectivity and strong "encapsulation" properties are imposed on the morphisms in the pushouts considered, see [Dia08].

The interpolation requirement for an institution may be parameterized by classes of morphisms used in the pushouts considered. For simplicity of exposition we avoid this complication here; see [Bor05], [Dia08] for details.

3 Specifications and their semantics

Taking an institution as a starting point for talking about specifications and software development, each signature Σ captures static information about the interface of a software system with each Σ-model representing a possible realisation of such a system, and with Σ-sentences used to describe properties (axioms) that a realisation is required to satisfy. As a consequence, it is natural to regard the meaning of any specification SP built in an institution $\mathbf{INS} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ as given by its signature $Sig[SP] \in |\mathbf{Sign}|$ together with a class $Mod[SP]$ of $Sig[SP]$-models. Specifications SP with $Sig[SP] = \Sigma$ are referred to as *Σ-specifications*.

The stress here is not only on the use of model classes to capture the semantics of specifications, but also on the lack of restriction on the models in the class and on the class itself — so-called "loose semantics". This is in contrast to the approach of ADJ [GTW76] and its followers, see e.g. [EM85], in which the meaning of an (equational) specification SP was taken to be the isomorphism class of the *initial* models of SP. Similar restrictions appear in other early approaches: final models [Wan79], generated models [BBB$^+$85], etc. The clear benefit of the loose approach is that it avoids placing premature constraints on the semantics of specifications, leaving choices of implementation details open for later stages of the development process. Although based in earlier work — the notion of the class of models of a set of axioms is a backbone of mathematical logic — in the context of algebraic specification this loose view was first consequently adopted in the work on ASL.

Different formulations of ASL share a kernel where specifications are built from *basic specifications* using *union*, *translation* and *hiding*. We use a syntax that is close to that of CASL [BM04], rather than choosing one of the variants in the ASL literature.

***basic specifications*:** For any signature $\Sigma \in |\mathbf{Sign}|$ and set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ-sentences, the *basic specification* $\langle \Sigma, \Phi \rangle$ is a specification with:

$$Sig[\langle \Sigma, \Phi \rangle] = \Sigma$$
$$Mod[\langle \Sigma, \Phi \rangle] = \{M \in \mathbf{Mod}(\Sigma) \mid M \models \Phi\}$$

***union*:** For any signature $\Sigma \in |\mathbf{Sign}|$, given Σ-specifications SP_1 and SP_2, their *union* $SP_1 \cup SP_2$ is a specification with:

$$Sig[SP_1 \cup SP_2] = \Sigma$$
$$Mod[SP_1 \cup SP_2] = Mod[SP_1] \cap Mod[SP_2]$$

***translation*:** For any signature morphism $\sigma \colon \Sigma \to \Sigma'$ and Σ-specification SP, SP **with** σ is a specification with:

$$Sig[SP \textbf{ with } \sigma] = \Sigma'$$
$$Mod[SP \textbf{ with } \sigma] = \{M' \in \mathbf{Mod}(\Sigma') \mid M'|_\sigma \in Mod[SP]\}$$

***hiding*:** For any signature morphism $\sigma \colon \Sigma \to \Sigma'$ and Σ'-specification SP', SP' **hide via** σ is a specification with:

$$Sig[SP' \textbf{ hide via } \sigma] = \Sigma$$
$$Mod[SP' \textbf{ hide via } \sigma] = \{M'|_\sigma \mid M' \in Mod[SP']\}$$

Using this semantics as a basis, we can now study the expressible properties that a specification ensures.

A Σ-sentence $\varphi \in \mathbf{Sen}(\Sigma)$ is a *semantic consequence* of a specification SP with $Sig[SP] = \Sigma$ if $Mod[SP] \models \varphi$; we write this $SP \models \varphi$. The set of all semantic consequences of SP is called the *theory* of SP.

An alternative to the ASL model-class semantics for specifications is to assign a theory to a specification as its meaning. This goes back to Clear [BG80], and is the stance taken for instance in [DGS93] and [GR04]. See [ST14] for a careful analysis of this alternative.

One standard way of presenting such a semantics is by giving a proof system for deriving consequences of specifications. For the class of specifications described above, the following proof rules are standard [ST88a]:

$$\frac{SP \vdash \varphi \text{ for each } \varphi \in \Phi \qquad \Phi \vdash \psi}{SP \vdash \psi} \qquad \frac{}{\langle \Sigma, \Phi \rangle \vdash \varphi}\ \varphi \in \Phi$$

$$\frac{SP_1 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi} \qquad \frac{SP_2 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi}$$

$$\frac{SP \vdash \varphi}{SP \textbf{ with } \sigma \vdash \sigma(\varphi)} \qquad \frac{SP \vdash \sigma(\varphi)}{SP \textbf{ hide via } \sigma \vdash \varphi}$$

where $\Phi \vdash \psi$ calls upon a sound entailment system for the underlying institution $\mathbf{INS}$. This proof system is sound: $SP \vdash \varphi$ implies $SP \models \varphi$. Completeness ($SP \models \varphi$ implies $SP \vdash \varphi$) is more difficult.

Theorem 3.1 ([ST14]). *Given an institution* $\mathbf{INS}$ *that admits amalgamation, and an entailment system* $\langle \vdash_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|}$ *for* $\mathbf{INS}$ *that is sound and complete, the above proof system is sound and complete for consequences of specifications built from basic specifications using union, translation and hiding if and only if* $\mathbf{INS}$ *admits parameterized interpolation.*

The assumption that $\mathbf{INS}$ admits parameterized interpolation is a rather strong one, excluding important examples like $\mathbf{EQ}$ except under restrictive conditions (Sect. 2). But strengthening the proof system above in an attempt to make it complete even in the absence of this assumption has a high price. As explained in full technical detail in [ST14], the above proof system cannot be improved without breaking the well-known compositionality principle, whereby consequences of a specification are inferred from consequences of its immediate component subspecifications.

It follows that a compositional theory-level semantics for the above class of structured specifications — or any larger class — that would assign to any specification the theory of its model class can be given only under a rather strong assumption about the underlying logical system.

This negative conclusion shows that there is an unavoidable discrepancy between compositional theory-level and model-class semantics for specifications. As usual, proof theory gives an approximation to semantic truth, and where there is a difference the latter provides the definitive reference point.

That said, non-compositional sound and complete proof systems for consequences of specifications can be given by collapsing the structure of specifications via normalisation [Bor05], even if **INS** does not admit interpolation. Various ways of avoiding complete collapse of specification structure are possible, see [MAH06] and [MT14].

We may take the theory-level view a bit further and study consequence relative to a specification. Any Σ-specification SP determines a consequence relation $\models_{SP}$ where for any set Φ of Σ-sentences and any Σ-sentence φ, $\Phi \models_{SP} \varphi$ if φ holds in all models of SP that satisfy Φ. The corresponding semantics assigns to each specification an entailment relation, possibly given by a proof system as in [HWB97]. The standard proof rules for the above specifications are the following:

$$\frac{\Phi \vdash \psi}{\Phi \vdash_{SP} \psi} \qquad \frac{}{\vdash_{\langle \Sigma, \Phi \rangle} \varphi}\ \varphi \in \Phi$$

$$\frac{\Phi \vdash_{SP_1} \varphi}{\Phi \vdash_{SP_1 \cup SP_2} \varphi} \qquad \frac{\Phi \vdash_{SP_2} \varphi}{\Phi \vdash_{SP_1 \cup SP_2} \varphi}$$

$$\frac{\Phi \vdash_{SP} \varphi}{\sigma(\Phi) \vdash_{SP\ \textbf{with}\ \sigma} \sigma(\varphi)} \qquad \frac{\sigma(\Phi) \vdash_{SP} \sigma(\varphi)}{\Phi \vdash_{SP\ \textbf{hide via}\ \sigma} \varphi}$$

These rules are sound: $\Phi \vdash_{SP} \varphi$ implies $\Phi \models_{SP} \varphi$. Again, completeness ($\Phi \models_{SP} \varphi$ implies $\Phi \vdash_{SP} \varphi$) holds only under strong assumptions.

Theorem 3.2 ([Dia08]). *Given an institution* **INS** *that admits amalgamation, and an entailment system* $\langle \vdash_\Sigma \rangle_{\Sigma \in |\mathbf{Sign}|}$ *for* **INS** *that is sound and complete, the above proof system is sound and complete for consequence relative to specifications built from basic specifications using union, translation and hiding if and only if* **INS** *admits parameterized interpolation.*

The negative remarks above concerning compositional theory-level semantics carry over here as well.

3.1 An example

Without complicating the semantic foundations, we may add specification-building operations that are defined in terms of the ones above. For instance, in any algebraic institution, we may use the following operations:

***sum*:** For any Σ-specification SP and Σ'-specification SP', their *sum* is:

SP **and** $SP' = (SP$ **with** $\iota) \cup (SP'$ **with** $\iota')$

where $\iota \colon \Sigma \hookrightarrow \Sigma \cup \Sigma'$ and $\iota' \colon \Sigma' \hookrightarrow \Sigma \cup \Sigma'$ are the signature inclusions.

***enrichment*:** For any Σ-specification SP with $\Sigma = \langle S, \Omega \rangle$, set S' of sort names, set Ω' of operation names with arities and result sorts over $S \cup S'$, and set Φ' of sentences over the signature $\Sigma' = \langle S \cup S', \Omega \cup \Omega' \rangle$, we define:

SP **then sorts** S' **ops** Ω' $\bullet$ $\Phi' = (SP$ **with** $\iota) \cup \langle \Sigma', \Phi' \rangle$

where $\iota \colon \Sigma \hookrightarrow \Sigma'$ is the signature inclusion. Obvious notational variants (e.g. omitting "**sorts**" when $S' = \emptyset$) are used to enhance convenience.

***hiding*:** Hiding with respect to signature inclusion may be written by listing the hidden symbols. So, for any Σ-specification SP with $\Sigma = \langle S, \Omega \rangle$ and signature inclusion $\iota \colon \langle S', \Omega' \rangle \hookrightarrow \Sigma$, we define:

SP **hide sorts** $S \setminus S'$ **ops** $\Omega \setminus \Omega' = SP$ **hide via** ι

Assume given specifications Bool of Booleans and Nat of natural numbers. Then, working in **FOEQ**, we can build the following specifications:

spec NatBool =
Nat **and** Bool **then**
ops $\_\_ > \_\_ \colon \mathit{nat} \times \mathit{nat} \rightarrow \mathit{bool}$
$\forall n, m \colon \mathit{nat}$

- $0 > n = \mathit{false}$
- $\mathit{succ}(n) > 0 = \mathit{true}$
- $\mathit{succ}(n) > \mathit{succ}(m) = n > m$

spec Bag =
NatBool **then**
sorts bag
ops $\mathit{empty} \colon \mathit{bag}$
$\mathit{add} \colon \mathit{nat} \times \mathit{bag} \rightarrow \mathit{bag}$
$\mathit{count} \colon \mathit{nat} \times \mathit{bag} \rightarrow \mathit{nat}$
$\forall x, y \colon \mathit{nat}, B \colon \mathit{bag}$

- $\mathit{count}(x, \mathit{empty}) = 0$
- $\mathit{count}(x, \mathit{add}(x, B)) = \mathit{succ}(\mathit{count}(x, B))$
- $x \neq y \Rightarrow \mathit{count}(x, \mathit{add}(y, B)) = \mathit{count}(x, B)$

spec Container =
(Bag **then**
ops $\mathit{isin} \colon \mathit{nat} \times \mathit{bag} \rightarrow \mathit{bool}$
$\forall x \colon \mathit{nat}, B \colon \mathit{bag}$

- $\mathit{isin}(x, B) = \mathit{count}(x, B) > 0$)

hide ops count

It is now easy to check that, for instance,

Container $\models \forall x{:}\mathit{nat}, B{:}\mathit{bag}.\ \mathit{isin}(x, \mathit{add}(x, B)) = \mathit{true}$
Container $\models \forall x, y{:}\mathit{nat}, B{:}\mathit{bag}.\ \mathit{isin}(x, \mathit{add}(y, \mathit{add}(x, B))) = \mathit{true}$
Container $\models \forall x{:}\mathit{nat}.\ \mathit{isin}(x, \mathit{empty}) = \mathit{false}.$

Since we are working in **FOEQ**, which admits parameterized interpolation, by Theorem 3.1 these can be proved using the proof system given above. We encourage the reader to write out the details.

One may now attempt to upgrade Container to give a specification of sets, for example:

spec ExtContainer =
Container **then**
$\forall B, B' \colon \mathit{bag}$

- $(\forall x{:}\mathit{nat}.\ \mathit{isin}(x, B) = \mathit{isin}(x, B')) \Rightarrow B = B'$

However, this specification has no models. To see this, note that[1]

$$\textsc{Bag} \models \forall x{:}nat.\ add(x, add(x, empty)) \neq add(x, empty)$$

from which we encourage the reader to derive $\textsc{ExtContainer} \models \phi$ for all first-order formulae ϕ.

Instead, we may define

spec $\textsc{Set} =$
  $\textsc{NatBool}$ **then**
    sorts bag
    ops $empty\colon bag$
      $add\colon nat \times bag \to bag$
      $isin\colon nat \times bag \to bool$
    $\forall x, y\colon nat, B\colon bag$
      • $isin(x, empty) = false$
      • $isin(x, add(x, B)) = true$
      • $x \neq y \Rightarrow isin(x, add(y, B)) = isin(x, B)$

and then

spec $\textsc{ExtSet} =$
  $\textsc{Set}$ **then**
    $\forall B, B'\colon bag$
    • $(\forall x{:}nat.\ isin(x, B) = isin(x, B')) \Rightarrow B = B'$

We may now prove

$$\begin{array}{l}\textsc{ExtSet} \models \forall x, y{:}nat, B{:}bag.\ add(x, add(y, B)) = add(y, add(x, B)) \\ \textsc{ExtSet} \models \forall x{:}nat, B{:}bag.\ isin(x, B) = true \Rightarrow add(x, B) = B\end{array}$$

as well as

$$\begin{array}{l}\textsc{ExtSet} \models \forall x{:}nat, B{:}bag.\ isin(x, add(x, B)) = true \\ \textsc{ExtSet} \models \forall x, y{:}nat, B{:}bag.\ isin(x, add(y, add(x, B))) = true \\ \textsc{ExtSet} \models \forall x{:}nat.\ isin(x, empty) = false.\end{array}$$

We will refer to these specifications throughout the rest of the paper.

4 Implementations and parameterization

At the time when work on ASL began, one of the hottest research topics in algebraic specification was the search for the "right" definition of implementation of one specification by another. The goal was to achieve the expected composability properties [GB80] while capturing practical data representation and refinement techniques. Various approaches were proposed, of which the concept of implementation given in [EKMP82] was probably the most influential and well developed; see [SW82] for a

[1] This follows from $\textsc{Nat} \models \forall n{:}nat.\ succ(n) \neq n$.

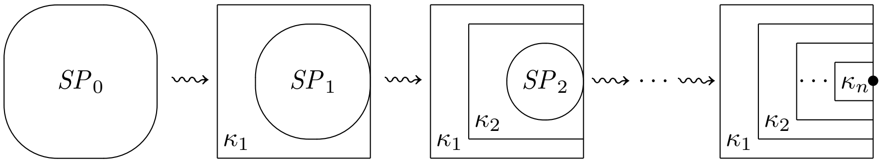
contribution from Martin. “Vertical composition” (transitivity) of two such implementations was the crucial goal, but this was not always possible except under additional assumptions about the specifications involved. In retrospect, this is no surprise, since the definitions proposed were all based on syntactic concepts and composition required some form of normalisation of the transition from implemented to implementing specifications. This corresponds to requiring programs to be written in a rather restricted programming language that provides no means of composing modules without syntactically merging their actual code. In addition to problems with vertical composition, these early definitions failed to cover certain naturally arising examples, and most disregarded loose specifications.

The breakthrough of ASL for implementations was to take seriously the idea that a loose specification has all of its legal realisations as models. Proceeding from an abstract specification of requirements to a more refined specification is then a matter of making a series of design decisions, each of which narrows the class of models under consideration. Thus, implementation corresponds simply to the inclusion of model classes of the specifications involved.

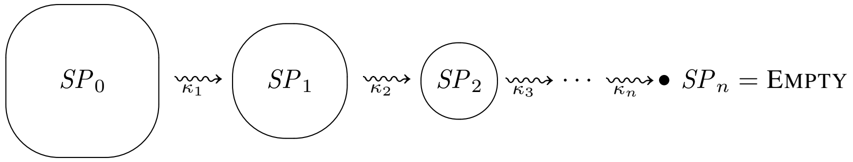
Given specifications SP and SP' with $Sig[SP] = Sig[SP']$, we say that SP' is a *simple implementation* of SP, written $SP \leadsto SP'$, if $Mod[SP] \supseteq Mod[SP']$. For instance, referring to Sect. 3.1, SET $\leadsto$ CONTAINER. (But CONTAINER $\not\leadsto$ SET.)

Vertical composition is now immediate: if $SP \leadsto SP'$ and $SP' \leadsto SP''$ then $SP \leadsto SP''$. Thus, given a chain $SP_0 \leadsto SP_1 \leadsto \cdots \leadsto SP_n$ of simple implementation steps and a model $M \in Mod[SP_n]$, we have $M \in Mod[SP_0]$. This ensures that the correctness of the final outcome of stepwise development may be inferred from the correctness of the individual implementation steps.

The definition of simple implementation requires the signatures of both specifications to be the same. Hiding may be used to adjust the signatures (for example, by hiding auxiliary functions in the implementing specification) if this is not the case. This is just one example of “wrapping” around specifications that may be needed to capture the relationship between implemented and implementing specifications when using simple implementations. In general, such wrapping may incorporate design decisions like definitions of types and operations in terms of other components that are yet to be implemented. These are expressible using the simple specification constructs defined in the last section, where definitions can be expressed using hiding via a derived signature morphism. Proceeding this way, successive specifications in the chain will incorporate more and more details arising from successive design decisions. Thereby, some parts become fully determined, and remain unchanged as a part of the specification until the final program is obtained. The following diagram is a visual representation of this situation, where $\kappa_1, \ldots, \kappa_n$ label the parts that become determined at consecutive steps:



It is more convenient to separate the finished parts from the specification, putting them aside, and proceeding with the development of the unresolved parts only:



where $\textsc{Empty}$ is a specification for which a standard model $\mathsf{empty} \in \mathit{Mod}[\textsc{Empty}]$ is available.

Semantically, the finished parts $\kappa_1, \ldots, \kappa_n$ are functions that map any realisation of the unresolved part to a realisation of what is being implemented. We call such functions *constructors* and capture the corresponding concept of implementation as follows [ST88b]: given specifications SP and SP', we say that SP' is a *constructor implementation of SP via κ*, written $SP \underset{\kappa}{\rightsquigarrow} SP'$, if κ is a constructor from SP' to SP, that is, a function $\kappa \colon |\mathbf{Mod}(\mathit{Sig}[SP'])| \to |\mathbf{Mod}(\mathit{Sig}[SP])|$ such that $\kappa(M') \in \mathit{Mod}[SP]$ for all $M' \in \mathit{Mod}[SP']$. Again, vertical composition follows immediately: if $SP \underset{\kappa}{\rightsquigarrow} SP'$ and $SP' \underset{\kappa'}{\rightsquigarrow} SP''$ then $SP \underset{\kappa';\kappa}{\rightsquigarrow} SP''$. Furthermore, given a chain of constructor implementation steps $SP_0 \underset{\kappa_1}{\rightsquigarrow} SP_1 \underset{\kappa_2}{\rightsquigarrow} \cdots \underset{\kappa_n}{\rightsquigarrow} SP_n = \textsc{Empty}$ we have $\kappa_1(\kappa_2(\ldots\kappa_n(\mathsf{empty})\ldots)) \in \mathit{Mod}[SP_0]$.

The general notion of a constructor above covers various constructs used in early notions of implementation. An important class of examples is reducts with respect to derived signature morphisms $\delta \colon \mathit{Sig}[SP] \to \mathit{Sig}[SP']$ which capture definitions of types and operations in SP terms of components in SP'.[2] By definition, this yields $SP \underset{-|_\delta}{\rightsquigarrow} SP'$ if $M'|_\delta \in \mathit{Mod}[SP]$ for all models $M' \in \mathit{Mod}[SP']$, which is a semantic statement of the correctness condition that needs to be verified.

For example, $\textsc{Container} \underset{-|_\delta}{\rightsquigarrow} \textsc{Bag}$ where $\delta \colon \mathit{Sig}[\textsc{Container}] \to \mathit{Sig}[\textsc{Bag}]$ maps isin to the term $\mathit{count}(x_1, x_2) > 0$.

This successfully deals with the definition of implementation and the issue of vertical composition. The other dimension, "horizontal composition" [GB80], captures the idea that combining implementations of components of a structured specification should yield an implementation of the whole original specification. This supposedly provides for modular decomposition of development tasks during the stepwise development process. Unfortunately, this does not allow for the very real possibility that there may be a mismatch between the structure of the original requirements specification and its realisation, see [FJ90]. For example, an implementation of CONTAINER would not need to be built on top of an implementation of BAG. The requirement of horizontal composition is missing a way of distinguishing between, on the one hand, the structure of the requirements specification used to facilitate its construction and understanding, and on the other hand, binding decisions made during the development process concerning the structure of the realisation. The latter fixes the design of the system architecture, and horizontal composition with respect to this structure is what really matters, see [ST06]. CASL [BM04] provides a way to capture designs of system architecture in the form of *architectural specifications* [BST02].

[2] This essentially gives a simple functional programming language if one generalises the notion of derived signature morphisms by allowing terms that involve constructs like conditionals, local (recursive) definitions, etc., see e.g. Example 4.1.25 of [ST12].

Even though this crucial distinction was never pointed out in the work on ASL, and it was not properly understood at the time, its technical roots are discernible in the ASL notion of parameterized specification.

Before ASL, the predominant style of parameterization in algebraic specification was in terms of pushouts in the category of specifications. These originated in Clear [BG80] and were then taken further in [TWW82], [EM85]. There, parameterized specifications were viewed both as specification-building operations and as specifications for the (free) functor mapping models of the parameter specification to models of the result specification, with compatibility between the two views being a cornerstone of this approach. This two-level view is another manifestation of the confusion between the structure of requirement specifications and the structure of realisations.

Parameterized specifications in ASL were quite different, formed by λ-abstracting specification expressions with respect to a specification variable. This obviously yields a function from specifications to specifications, but in general such a function will not correspond in any natural way to a function on the level of models, and in ASL there was never any intention that it would.

For instance, define

spec $\textsc{Ext} =$
  $\lambda\mathcal{X}{:}\mathit{Sig}[\textsc{Set}] \bullet$
    $\mathcal{X}$ **then**
      $\forall B, B' \colon \mathit{bag}$
      $\bullet\ (\forall x{:}\mathit{nat}.\ \mathit{isin}(x,B) = \mathit{isin}(x,B')) \Rightarrow B = B'$

Then $\textsc{ExtContainer} = \textsc{Ext}(\textsc{Container})$ and $\textsc{ExtSet} = \textsc{Ext}(\textsc{Set})$. Clearly, $\textsc{Ext}$ does not correspond to a function on the level of individual models: $\textsc{Container}$ has models but $\textsc{Ext}(\textsc{Container})$ does not.

An analysis of this situation suggests that what is missing is a distinction between parameterized specifications and specifications of parameterized models (*viz.* generic modules, constructors, ML-style functors). We studied this distinction in [SST92]: parameterized specifications denote functions that map model classes to model classes, while parameterized programs denote functions that map models to models and specifications of parameterized programs denote classes of such functions. The slogan is

parameterized (program specification) $\neq$ *(parameterized program) specification.*

Given this distinction, different specification constructs are appropriate for the two kinds of specifications. We used the notation $\mathit{\Pi} X{:}\mathit{SP} \bullet \mathit{SP}'[X]$ for the latter, following dependent type theory, and ASL-style λ-abstraction as above for the former. There is a Galois connection which links the two semantic domains, with closed elements corresponding to functions mapping models to non-empty classes of models [SST92]. A natural example is to generalise from $\textsc{Set}$ by taking the sort of elements as a parameter, in place of nat.

This can be taken further, to higher-order parameterization mechanisms in which objects of all kinds (parameterized specifications, parameterized programs, their specifications, etc.) are permitted as arguments and as results. This results in a complex hierarchy with some "types" of objects in this hierarchy being more useful than others

[Asp97] and is closely related to issues involved in the design of module systems, see e.g. [LB88] and [KBS91]. CASL architectural specifications, which feature parameterized units and their specifications, may be viewed as providing a simple module system, raising familiar issues of shared substructure [BST02].

5 Behavioural specifications

Probably the most novel feature of ASL, which first appeared in [SW83], was *behavioural abstraction*. If two algebras "behave the same", and one is a model of a specification, then it is natural to consider the specification in which the other is a model as well. Behavioural abstraction performs such a closure, with respect to an equivalence which is chosen to reflect the desired meaning of "behaves the same".

There had been some work on behavioural interpretation of specifications before ASL, notably [GGM76], [Rei81], [GM82] and [Gan83]. ASL introduced behavioural abstraction as an explicit construct, which facilitated understanding of the relationship between behaviourally abstracted specifications and "normal" specifications in a single language. It also proposed a general notion of behavioural equivalence, parameterized by an arbitrary set of terms W to be regarded as observable, which covered various notions of behavioural equivalence proposed in the literature and more.

In the previous sections, we have been working in the context of an arbitrary institution, but discussion of behavioural equivalence and behavioural abstraction is simplest in the context of ordinary algebraic signatures and algebras. Therefore, in this section we will restrict attention to algebraic institutions, which share signatures and models with $\mathbf{EQ}$ and $\mathbf{FOEQ}$. See [ST87] for a possible generalisation to the framework of an arbitrary institution, and Sect. 8.5.3 of [ST12] for some further remarks in this direction.

Given an algebraic signature $\Sigma = \langle S, \Omega \rangle$, an S-sorted set of variables X, and a set $W \subseteq T_\Sigma(X)$ of Σ-terms, two Σ-algebras A, B are *W-equivalent via X*, written $A \equiv^{ASL}_{W(X)} B$, if there are surjective valuations $v_A \colon X \to |A|$, $v_B \colon X \to |B|$ such that for all terms $t, t' \in W$ of the same sort, $t_{A[v_A]} = t'_{A[v_A]}$ iff $t_{B[v_B]} = t'_{B[v_B]}$.

The relation $\equiv^{ASL}_{W(X)}$ is clearly not reflexive on $\mathbf{Alg}(\Sigma)$: for algebras A with carrier of cardinality larger than that of X, $A \not\equiv^{ASL}_{W(X)} A$. This was not a problem for ASL, where only countable algebras were considered. However, a problem that has been overlooked so far is that, in general, $\equiv^{ASL}_{W(X)}$ is not transitive either.

Counterexample 5.1. Consider a signature Σ with sorts s, bool and operations $g \colon s \to \mathit{bool}$ and $\mathit{true}, \mathit{false} \colon \mathit{bool}$, with $X = \{x, y{:}s, t, f{:}\mathit{bool}\}$ and $W = \{g(x), \mathit{true}, \mathit{false}\}$; crucially, $g(y) \notin W$. Consider Σ-algebras A, B, C such that $A_s = B_s = C_s = \{a, b\}$ and $A_{\mathit{bool}} = B_{\mathit{bool}} = C_{\mathit{bool}} = \{\mathit{tt}, \mathit{ff}\}$, with $\mathit{true}_A = \mathit{true}_B = \mathit{true}_C = \mathit{tt}$, $\mathit{false}_A = \mathit{false}_B = \mathit{false}_C = \mathit{ff}$, and $g_A(a) = g_A(b) = \mathit{tt}$, $g_B(a) = \mathit{tt}$ but $g_B(b) = \mathit{ff}$, and $g_C(a) = g_C(b) = \mathit{ff}$. Then $A \equiv^{ASL}_{W(X)} B$ via valuations v_A, v_B with $v_A(x) = v_B(x) = a$, $v_A(y) = v_B(y) = b$, and $B \equiv^{ASL}_{W(X)} C$ via valuations w_B, w_C with $w_B(x) = w_C(x) = b$, $w_B(y) = w_C(y) = a$ (extended surjectively to bool). But $A \not\equiv^{ASL}_{W(X)} C$.

A consequence of this is that ASL's behavioural abstraction as a function on model classes is not a closure operation, contrary to some of the laws in [SW83], [Wir86].

The source of the problem indicated by the above counterexample is that when the set of terms considered is not closed under renaming of variables, two algebras A, B remain in the relation defined above if for each set of terms in W that share common variables we can identify subalgebras of A and B in which these terms have the same behaviour. Clearly, this is quite different from requiring these terms (and all terms in W) to have the same behaviour throughout A and B, and leads to the failure of transitivity.

To alleviate the above problems, we therefore need to allow the set of variables to be arbitrarily enlarged and the set of terms to be closed under renaming of variables.

Given a set $W \subseteq T_\Sigma(X)$ of Σ-terms and another set Y of variables, the *closure of W from X to Y* is $W[X{\mapsto}Y] = \{\theta(t) \mid \theta\colon X \to Y, t \in W\}$. W is *closed under renaming of variables* if $W = W[X{\mapsto}X]$.

Now, we define two Σ-algebras A, B to be *W-equivalent*, written $A \equiv_W B$, if there is a set Y of variables such that $A \equiv^{ASL}_{W[X\mapsto Y]} B$. Then we define

***abstraction*:** For any Σ-specification SP and set $W \subseteq T_\Sigma(X)$ of Σ-terms with variables in X, **abstract** SP **wrt** W is a specification with:

$$Sig[\textbf{abstract } SP \textbf{ wrt } W] = \Sigma$$
$$Mod[\textbf{abstract } SP \textbf{ wrt } W] =$$
$$\{A \in \mathbf{Mod}(\Sigma) \mid A \equiv_W B \text{ for some } B \in Mod[SP]\}$$

Proposition 5.2. *For any signature Σ and set $W \subseteq T_\Sigma(X)$ of Σ-terms with variables in X, W-equivalence is indeed an equivalence on $\mathbf{Alg}(\Sigma)$.*

Proof. Reflexivity and symmetry are obvious. For transitivity, suppose $A \equiv_W B$ as witnessed by a set Y of variables with valuations $v_A\colon Y \to |A|$ and $v_B\colon Y \to |B|$, and $B \equiv_W C$ as witnessed by a set Z of variables with valuations $w_B\colon Z \to |B|$ and $w_C\colon Z \to |C|$. Take YZ to be the set of variables given by a pullback $v'\colon YZ \to Y$, $w'\colon YZ \to Z$ of v_B and w_B. Then the equivalence $A \equiv_W C$ is witnessed by YZ with valuations $v';v_A\colon YZ \to |A|$ and $w';w_C\colon YZ \to |C|$. First, since v_B and w_B are surjective, so are v' and w', and hence also $v';v_A$ and $w';w_C$. Then, for any terms $t, t' \in W$ of the same sort, and $\theta\colon X \to YZ$, we have: $\theta(t)_{A[v';v_A]} = \theta(t')_{A[v';v_A]}$ iff $(\theta;v')(t)_{A[v_A]} = (\theta;v')(t')_{A[v_A]}$ iff (since Y, v_A, v_B witness $A \equiv_W B$) $(\theta;v')(t)_{B[v_B]} = (\theta;v')(t')_{B[v_B]}$ iff $\theta(t)_{B[v';v_B]} = \theta(t')_{B[v';v_B]}$ iff (since $v';v_B = w';w_B$) $\theta(t)_{B[w';w_B]} = \theta(t')_{B[w';w_B]}$ iff $(\theta;w')(t)_{B[w_B]} = (\theta;w')(t')_{B[w_B]}$ iff (since Z, w_B, w_C witness $B \equiv_W C$) $(\theta;w')(t)_{C[w_C]} = (\theta;w')(t')_{C[w_C]}$ iff $\theta(t)_{C[w';w_C]} = \theta(t')_{C[w';w_C]}$.

We do not need to assume here that the set W is closed under renaming of variables — the definition of W-equivalence invokes the closure now.

Furthermore, W-equivalence properly generalises the equivalence used in ASL:

Proposition 5.3. *$A \equiv_W B$ iff $A \equiv^{ASL}_{W(X)} B$ provided that W is closed under renaming of variables and $card(X) \geq card(|A|) + card(|B|)$.*

Proof. We take the easy direction first: if $A \equiv^{ASL}_{W(X)} B$ is witnessed by $v_A\colon X \to |A|$, $v_B\colon X \to |B|$ then, since W is closed under renaming of variables, $A \equiv_W B$ is witnessed by X with the same valuations.

For the opposite implication: suppose $A \equiv_W B$ is witnessed by Y with valuations $v_A : Y \to |A|$, $v_B : Y \to |B|$. Then, given the cardinality assumption to ensure that X is sufficiently large, there exists $\theta : X \to Y$ such that $\theta;v_A : X \to |A|$ and $\theta;v_B : X \to |B|$ are surjective. $A \equiv^{ASL}_{W(X)} B$ is witnessed by $\theta;v_A$ and $\theta;v_B$.

Completely arbitrary choices of W, as permitted in ASL, may yield odd equivalences. Even closing the sets of terms under variable renaming leaves an enormous wealth of possibilities. Only a few of these have ever been used, capturing different notions of behavioural equivalence. The most typical situation is where we want to indicate a set IN of sorts to be viewed as input data, and a set OUT of sorts to be viewed as observable outputs. Then $\equiv_{T_\Sigma(X_{IN})_{OUT}}$ identifies algebras that display the same input/output behaviour for observable computations (presented as Σ-terms) taking inputs from IN and yielding results in OUT. Often, one identifies a single set OBS of *observable sorts* and takes $IN = OUT = OBS$. An important twist is to select a subset of operations that are used to build observable terms, by considering $\equiv_{T_{\Sigma'}(X_{IN})_{OUT}}$ for a subsignature Σ' of Σ, see for instance [BH06].

The natural choice of observable sorts for the specifications SET, EXTSET and CONTAINER in Sect. 3.1 is $OBS = \{bool, nat\}$; in particular, $bag \notin OBS$. One may now check that, in this context, it is sufficient to consider as observable terms W^{SET} all variables of sorts nat and $bool$ as well as all terms of the form $isin(x, t_{bag})$ where x is a variable of sort nat and t_{bag} is a term of sort bag built using $empty$, add, and variables of sort nat.

A more general interesting case arises in the following situation. We consider an additional signature $\widehat{\Sigma}$ with sets IN and OUT of input and output sorts, together with a derived signature morphism $\delta : \widehat{\Sigma} \to \Sigma$. We may think of δ as defining $\widehat{\Sigma}$-operations in terms of Σ-operations. Suppose that we want to observe $\widehat{\Sigma}$-computations carried out in Σ-algebras according to the definitions given by δ. Then the relevant equivalence on Σ-algebras is given by the following set of terms: $W_{\delta(IN,OUT)} = \delta(T_{\widehat{\Sigma}}(\widehat{X}_{IN})_{OUT})$, where $X = \delta(\widehat{X}_{IN})$.

In ASL, the abstraction construct defined above was available for arbitrary use, freely intermixed with other specification constructs. This is in line with the idea that ASL is a kernel language which provides raw specification power, free from pragmatic or methodologically-motivated constraints.

In specification practice, the use of abstraction can be limited to specific contexts where it fits a methodological need. In particular, if SP is a requirements specification and W captures all of the computations that the user wishes to carry out in its realisation, then any implementation of **abstract** SP **wrt** W will be satisfactory. So this is the specification that should be used as the starting point in the development. That is, we want to have the liberty to implement SP up to $\equiv_W$. However, when using a realisation of another specification SP' to implement SP, we still want to be allowed to assume that it satisfies SP' “literally”. This is captured by the following definition.[3]

We say that SP' is a *behavioural implementation of SP via κ wrt W*, written $SP \;{}^{W}\!\underset{\kappa}{\rightsquigarrow}\; SP'$, if **abstract** SP **wrt** $W \underset{\kappa}{\rightsquigarrow} SP'$. Obviously, whenever $SP \underset{\kappa}{\rightsquigarrow} SP'$

[3] This is a special case of *abstractor implementations* as introduced in [ST88b]. We follow the terminology of Sect. 8.4 in [ST12] but generalise the concept from equivalence with respect to observable sorts to equivalence with respect to observable terms.

then also $SP \overset{W}{\underset{\kappa}{\rightsquigarrow}} SP'$. Hence, for instance, we have $\textsc{Container} \overset{W^{\textsc{Set}}}{\underset{\_|_\delta}{\rightsquigarrow}} \textsc{Bag}$ where $\delta\colon \mathit{Sig}[\textsc{Container}] \to \mathit{Sig}[\textsc{Bag}]$ maps isin to the term $\mathit{count}(x_1, x_2) > 0$ and $W^{\textsc{Set}}$ is as described above. However, we also have $\textsc{ExtSet} \overset{W^{\textsc{Set}}}{\underset{\_|_\delta}{\rightsquigarrow}} \textsc{Bag}$ even though $\textsc{ExtSet} \underset{\_|_\delta}{\not\rightsquigarrow} \textsc{Bag}$.

The alert reader will have sensed that we are about to run into a problem: vertical composability does not hold in general. $SP \overset{W}{\underset{\kappa}{\rightsquigarrow}} SP'$ and $SP' \overset{W'}{\underset{\kappa'}{\rightsquigarrow}} SP''$ does not imply $SP \overset{W}{\underset{\kappa';\kappa}{\rightsquigarrow}} SP''$. However, these behavioural implementations compose if the constructor κ is *stable* with respect to W' and W, that is, $\equiv_{W'} \subseteq \kappa^{-1}(\equiv_W)$. Or, spelling this out, we require that for any $A', B' \in \mathbf{Alg}(\mathit{Sig}[SP'])$, whenever $A' \equiv_{W'} B'$ then $\kappa(A') \equiv_W \kappa(B')$ [Sch90], [ST88b]. This technical notion captures a methodological point: κ must not differentiate between behaviourally equivalent realisations of SP'. This is exactly the encapsulation principle of data abstraction and hierarchical decomposition.

Now, given a chain of behavioural implementation steps using stable constructors

$$SP_0 \overset{W_0}{\underset{\kappa_1}{\rightsquigarrow}} SP_1 \overset{W_1}{\underset{\kappa_2}{\rightsquigarrow}} \cdots \overset{W_{n-1}}{\underset{\kappa_n}{\rightsquigarrow}} SP_n = \textsc{Empty}$$

we have $\kappa_1(\kappa_2(\ldots\kappa_n(\mathsf{empty})\ldots)) \equiv_{W_0} A_0$, for some $A_0 \in \mathit{Mod}[SP_0]$.

The crucial stability requirement on constructors may be approached in two different ways. On the one hand, following the ideas in [Sch90] and [BST08], we can fix the family of behavioural equivalences considered, referring to a fixed set of observable built-in sorts (booleans, etc.), and then limit constructors to those that preserve that equivalence. This is guaranteed by use of a programming language that appropriately enforces abstraction barriers. The other option is, at each development step, to determine the behavioural equivalence that is appropriate to the context of use. Technically, this means that given a behavioural implementation step $SP \overset{W}{\underset{\kappa}{\rightsquigarrow}} SP'$, we need a set W' of $\mathit{Sig}[SP']$-terms such that κ is stable with respect to W' and W. Picking W' to achieve $\equiv_{W'} = \kappa^{-1}(\equiv_W)$ gives maximal flexibility for further implementations of SP', since only the precise context of use in the implementation of SP by SP' via κ matters.

The latter option was proposed in [ST88b] but it does not seem to have been properly explored. The following simple fact shows how this might go.

Proposition 5.4. *Given a derived signature morphism $\delta\colon \Sigma \to \Sigma'$ and set $W \subseteq T_\Sigma(X)$ of Σ-terms closed under renaming of variables, let $W' = \delta(W) \subseteq T_{\Sigma'}(X')$ where $X' = \delta(X)$. Then for any Σ'-algebras A', B', $A' \equiv_{W'} B'$ iff $A'|_\delta \equiv_W B'|_\delta$. In particular the δ-reduct constructor is stable with respect to W' and W.*

Proof. *Suppose in Σ', $A' \equiv_{W'} B'$ is witnessed by Y' with valuations $v'_{A'}\colon Y' \to |A'|$ and $v'_{B'}\colon Y' \to |B'|$. Then in Σ, $A'|_\delta \equiv_W B'|_\delta$ is witnessed by $Y = Y'|_\delta$ (only the mapping on sorts matters here) with valuations $v'_{A'}|_\delta\colon Y \to |A'|_\delta|$ and $v'_{B'}|_\delta\colon Y \to |B'|_\delta|$.*

Let then in Σ, $A'|_\delta \equiv_W B'|_\delta$ be witnessed by Y with valuations $v_1\colon Y \to |A'|_\delta|$ and $v_2\colon Y \to |B'|_\delta|$. Let Y' be $\delta(Y)$ on sorts in the image of δ and $Y'_{s'} = |A'|_{s'} \uplus |B'|_{s'}$ for all sorts s' not in the image of δ. Let $v'_{A'}\colon Y' \to |A'|$ be given by v_1 on variables *in $\delta(Y)$, and be any surjective function on the sorts not in the image of δ; similarly, let $v'_{B'} \colon Y' \to |B'|$ be given by v_2 on variables in $\delta(Y)$, and be any surjective function on the sorts not in the image of δ. Then $A' \equiv_{W'} B'$ is witnessed by Y' with valuations $v'_{A'}$ and $v'_{B'}$.*

Stability of the reduct is just the former implication.

For instance, in the context of use of Bag taken as an implementation of ExtSet as indicated above, ExtSet $\overset{W^{\textsc{Set}}}{\underset{-|\delta}{\rightsquigarrow}}$ Bag, the relevant set of observable terms to determine equivalence up to which Bag is to be implemented is $\delta(W^{\textsc{Set}})$ which consists of all variables of sorts nat and $bool$ as well as all terms of the form $count(x, t_{bag}) > 0$ where x is a variable of sort nat and t_{bag} is a term of sort bag built using $empty$, add, and variables of sort nat. In particular, we do not care about keeping the exact count of the number of occurrences in a bag, as long as we can distinguish between the cases where it is 0 versus strictly positive.

6 Final remarks

In this essay we have presented what we see as the key characteristics of ASL and have outlined some of the developments that later emerged from this basis. We have focused on tracing the flow of ideas rather than on technical details or new results, although the technicalities in Sect. 5 regarding W-equivalence seem new.

Even though there has been a lot of work on these topics, some corners are worth further exploration.

Semantics: It seems to us that the relationship between model-class and theory-level semantics is completely resolved by Theorem 3.1 and its consequences, as discussed in Sect. 3, even if the choice between the two may remain controversial in some quarters. The class of specifications we consider is particularly well-understood with clear proof techniques etc. For some other specification constructs, including for instance behavioural abstraction, this is much less true.

Implementation: The semantic concept of implementation in Sect. 4 together with its refinement in Sect. 5 capture what is needed. We have not discussed issues arising from the need for proof techniques to establish the correctness of implementation steps — see Chap. 9 of [ST12] for our summary of the state of the art.

Parameterization: All of the syntactic and semantic concepts are established but in a raw form that is a little hard to use. We feel that this is still a somewhat open area where more ideas are needed to limit the scope of possibilities to what is really required and useful in practice.

Behavioural specifications: Following ASL, in Sect. 5 we sketched the "external" approach to behavioural interpretation of specifications, based on behavioural equivalence between algebras. A widely-studied alternative is to re-interpret the meaning of axioms, and hence of specifications, using the "internal" indistinguishability between values. The relationship between the two approaches is now well-understood, see [BHW95], but only for behavioural equivalence with respect to a set of observable sorts. It would be interesting to investigate the same relationship for W-equivalence. We also think that the methodological ideas on the use of context-tailored behavioural equivalence at the end of Sect. 5 are worth further exploration.

References

[Asp97] D. Aspinall. *Type Systems for Modular Programming and Specification*. Ph.D. thesis, University of Edinburgh, Department of Computer Science, 1997.

[BBB$^+$85] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner. *The Munich Project CIP: Vol. 1: The Wide Spectrum Language CIP-L*, *Lecture Notes in Computer Science*, vol. 183. Springer, 1985.

[BG80] R. M. Burstall and J. A. Goguen. The semantics of Clear, a specification language. In: D. Bjørner, ed., *Proceedings of the 1979 Copenhagen Winter School on Abstract Software Specification*, *Lecture Notes in Computer Science*, vol. 86, 292–332. Springer, 1980.

[BH06] M. Bidoit and R. Hennicker. Constructor-based observational logic. *Journal of Logic and Algebraic Programming*, 67(1–2):3–51, 2006.

[BHW95] M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25(2–3):149–186, 1995.

[BM04] M. Bidoit and P. D. Mosses, eds. CASL *User Manual*, *Lecture Notes in Computer Science*, vol. 2900. Springer, 2004. See also `http://www.informatik.uni-bremen.de/cofi/index.php/CASL`.

[Bor05] T. Borzyszkowski. Generalized interpolation in first order logic. *Fundamenta Informaticae*, 66(3):199–219, 2005.

[BST02] M. Bidoit, D. Sannella, and A. Tarlecki. Architectural specifications in CASL. *Formal Aspects of Computing*, 13:252–273, 2002.

[BST08] M. Bidoit, D. Sannella, and A. Tarlecki. Observational interpretation of CASL specifications. *Mathematical Structures in Computer Science*, 18:325–371, 2008.

[CK90] C.-C. Chang and H. J. Keisler. *Model Theory*. North-Holland, third edition, 1990.

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

[Dia08] R. Diaconescu. *Institution-Independent Model Theory*. Birkhäuser, 2008.

[EKMP82] H. Ehrig, H.-J. Kreowski, B. Mahr, and P. Padawitz. Algebraic implementation of abstract data types. *Theoretical Computer Science*, 20:209–263, 1982.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1*, *EATCS Monographs on Theoretical Computer Science*, vol. 6. Springer, 1985.

[FJ90] J. S. Fitzgerald and C. B. Jones. Modularizing the formal description of a database system. In: *Proceedings of the 3rd International Symposium of VDM Europe: VDM and Z, Formal Methods in Software Development*, Kiel, *Lecture Notes in Computer Science*, vol. 428, 189–210. Springer, 1990.

[Gan83] H. Ganzinger. Parameterized specifications: Parameter passing and implementation with respect to observability. *ACM Transactions on Programming Languages and Systems*, 5(3):318–354, 1983.

[GB80] J. A. Goguen and R. M. Burstall. CAT, a system for the structured elaboration of correct programs from structured specifications. Technical Report CSL-118, Computer Science Laboratory, SRI International, 1980.

[GB92] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[GGM76] V. Giarratana, F. Gimona, and U. Montanari. Observability concepts in abstract data type specifications. In: A. Mazurkiewicz, ed., *Proceedings of the 5th Symposium on*

Mathematical Foundations of Computer Science, Gdańsk, *Lecture Notes in Computer Science*, vol. 45, 567–578. Springer, 1976.

[GM82] J. A. Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In: M. Nielsen and E. M. Schmidt, eds., *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, Aarhus, *Lecture Notes in Computer Science*, vol. 140, 265–281. Springer, 1982.

[GR04] J. A. Goguen and G. Roşu. Composing hidden information modules over inclusive institutions. In: *From Object-Orientation to Formal Methods. Essays in Memory of Ole-Johan Dahl*, *Lecture Notes in Computer Science*, vol. 2635, 96–123. Springer, 2004.

[GTW76] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM Watson Research Center, Yorktown Heights NY, 1976. Also in: Raymond T. Yeh, ed., *Current Trends in Programming Methodology. Vol. IV (Data Structuring)*, 80–149. Prentice-Hall, 1978.

[HWB97] R. Hennicker, M. Wirsing, and M. Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2):393–443, 1997.

[KBS91] B. Krieg-Brückner and D. Sannella. Structuring specifications in-the-large and in-the-small: Higher-order functions, dependent types and inheritance in SPECTRAL. In: *Proc. Colloq. on Combining Paradigms for Software Development. Intl. Joint Conf. on Theory and Practice of Software Development (TAPSOFT'91)*, Brighton, *Lecture Notes in Computer Science*, vol. 494, 103–120. Springer, 1991.

[LB88] B. Lampson and R. M. Burstall. Pebble, a kernel language for modules and abstract data types. *Information and Computation*, 76(2–3):278–346, 1988.

[MAH06] T. Mossakowski, S. Autexier, and D. Hutter. Development graphs — proof management for structured specifications. *Journal of Logic and Algebraic Programming*, 67(1–2):114–145, 2006.

[MT14] T. Mossakowski and A. Tarlecki. A relatively complete calculus for structured heterogeneous specifications. In: A. Muscholl, ed., *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures. European Joint Conferences on Theory and Practice of Software (ETAPS 2014)*, *Lecture Notes in Computer Science*, vol. 8412, 441–456. Springer, 2014.

[Rei81] H. Reichel. Behavioural equivalence — a unifying concept for initial and final specification methods. In: *Proceedings of the 3rd Hungarian Computer Science Conference*, 27–39, 1981.

[RG00] G. Roşu and J. A. Goguen. On equational Craig interpolation. *Journal of Universal Computer Science*, 6(1):194–200, 2000.

[Sch90] O. Schoett. Behavioural correctness of data representations. *Science of Computer Programming*, 14(1):43–57, 1990.

[SST92] D. Sannella, S. Sokołowski, and A. Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29(8):689–736, 1992.

[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(2–3):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.

[ST06] D. Sannella and A. Tarlecki. Horizontal composability revisited. In: K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, eds., *Algebra, Meaning and Computation: Essays*

Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday, *Lecture Notes in Computer Science*, vol. 4060, 296–316. Springer, 2006.

[ST12] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012.

[ST14] D. Sannella and A. Tarlecki. Property-oriented semantics of structured specifications. *Mathematical Structures in Computer Science*, 24(2):e240205, 2014.

[SW82] D. Sannella and M. Wirsing. Implementation of parameterised specifications. In: M. Nielsen and E. M. Schmidt, eds., *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, Aarhus, *Lecture Notes in Computer Science*, vol. 140, 473–488. Springer, 1982.

[SW83] D. Sannella and M. Wirsing. A kernel language for algebraic specification and implementation. In: M. Karpinski, ed., *Proceedings of the 1983 International Conference on Foundations of Computation Theory*, Borgholm, *Lecture Notes in Computer Science*, vol. 158, 413–427. Springer, 1983.

[Tar86] A. Tarlecki. Bits and pieces of the theory of institutions. In: D. H. Pitt, S. Abramsky, A. Poigné, and D. E. Rydeheard, eds., *Proceedings of the Tutorial and Workshop on Category Theory and Computer Programming*, Guildford, *Lecture Notes in Computer Science*, vol. 240, 334–360. Springer, 1986.

[Tar11] A. Tarlecki. Some nuances of many-sorted universal algebra: A review. *Bulletin of the European Association for Theoretical Computer Science*, 104:89–111, 2011.

[TWW82] J. W. Thatcher, E. G. Wagner, and J. B. Wright. Data type specification: Parameterization and the power of specification techniques. *ACM Transactions on Programming Languages and Systems*, 4(4):711–732, 1982.

[Wan79] M. Wand. Final algebra semantics and data type extensions. *Journal of Computer and System Sciences*, 19:27–44, 1979.

[Wir82] M. Wirsing. Structured algebraic specifications. In: *Proceedings of the AFCET Symposium on Mathematics for Computer Science*, Paris, 93–107, 1982.

[Wir86] M. Wirsing. Structured algebraic specifications: A kernel language. *Theoretical Computer Science*, 42(2):123–249, 1986.